1. Copyright.

2. *eval_phrases* **Grammar.**
Make sure all the grammar phases are properly parsed. This is due to my not programming a grammar sequencer to explicitly define the syntax of a grammar. I wanted to parse each phrase separately by keyword triggering a descent procedure.

Question: What source GPS do u align the error against? i just place it to the beginning of the grammar file if there is no previous phase seen. A previous phase becomes the reference point to the source file. If i was energetic, i should get the last token of that phase to be more accurate but today i'm lazy. The message is adequate to correct the problem.

3. Fsm Ceval_phrases class.

4. Ceval_phrases op directive.

⟨ Ceval_phrases op directive 4 ⟩ ≡
```
gps_ = 0;
```

5. Ceval_phrases user-declaration directive.

⟨ Ceval_phrases user-declaration directive 5 ⟩ ≡
```
public: CAbs_lr1_sym * gps_;
  void post_error(CAbs_lr1_sym * Err);
  void post_gps(CAbs_lr1_sym * Sym);
```

6. Ceval_phrases user-implementation directive.

⟨ Ceval_phrases user-implementation directive 6 ⟩ ≡
```
void Ceval_phrases::post_error(CAbs_lr1_sym * Err)
{
  using namespace NS_yacco2_T_enum;
  Err->set_rc(*gps_, __FILE__, __LINE__);
  if (gps_->enumerated_id__ ≡ T_Enum::T_LR1_eog_) {
    Err->set_line_no_and_pos_in_line(1, 1);
  }
  parser__->add_token_to_error_queue(*Err);
  parser__->set_abort_parse(true);
}
```

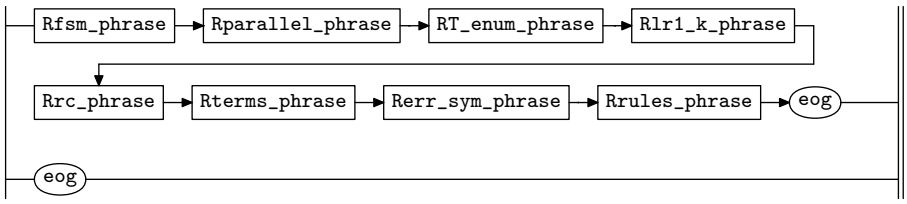
7. *post_gps*.

⟨ More code 7 ⟩ ≡
```
void Ceval_phrases::post_gps(CAbs_lr1_sym * Sym)
{
  gps_ = Sym;
}
```

8. *Reval_phrases* **rule.**

Use the property of Rule's sequencing within a production to determine whether the parsed phrases were missed or out of sequence. Report them as errors.

`Reval_phrases`

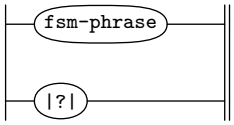


9. *Reval_phrases*'**s subrule 2.**

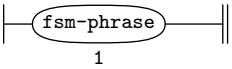
⟨ Reval_phrases subrule 2 op directive 9 ⟩ ≡
Ceval_phrases ∗ *fsm* = (*Ceval_phrases* ∗) *rule_info__.parser__→fsm_tbl__*;
fsm→post_error(**new** *Err_empty_file*);
fsm→post_gps(*sf→p1__*);

10. *Rfsm_phrase* **rule.**

`Rfsm_phrase`

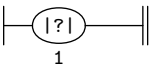


11. *Rfsm_phrase*'**s subrule 1.**



⟨ Rfsm_phrase subrule 1 op directive 11 ⟩ ≡
Ceval_phrases ∗ *fsm* = (*Ceval_phrases* ∗) *rule_info__.parser__→fsm_tbl__*;
fsm→post_gps(*sf→p1__*);

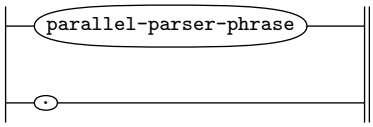
12. *Rfsm_phrase*'**s subrule 2.**



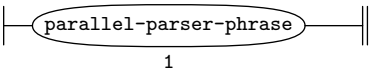
⟨ Rfsm_phrase subrule 2 op directive 12 ⟩ ≡
Ceval_phrases ∗ *fsm* = (*Ceval_phrases* ∗) *rule_info__.parser__→fsm_tbl__*;
fsm→post_gps(*sf→p1__*);
fsm→post_error(**new** *ERR_no_fsm_phrase*);

13. *Rparallel_phrase* **rule.**

Rparallel_phrase



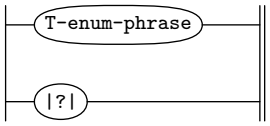
14. *Rparallel_phrase*'**s subrule 1.**



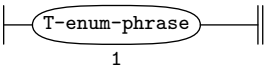
⟨ Rparallel_phrase subrule 1 op directive 14 ⟩ ≡
Ceval_phrases ∗ *fsm* = (*Ceval_phrases* ∗) *rule_info__.parser__→fsm_tbl__*;
fsm→post_gps(*sf→p1__*);

15. *RT_enum_phrase* **rule.**

RT_enum_phrase

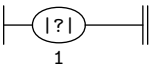


16. *RT_enum_phrase*'**s subrule 1.**



⟨ RT_enum_phrase subrule 1 op directive 16 ⟩ ≡
Ceval_phrases ∗ *fsm* = (*Ceval_phrases* ∗) *rule_info__.parser__→fsm_tbl__*;
fsm→post_gps(*sf→p1__*);

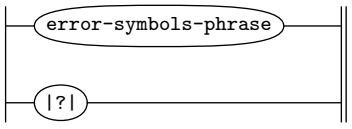
17. *RT_enum_phrase*'**s subrule 2.**



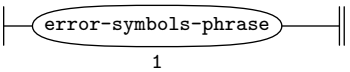
⟨ RT_enum_phrase subrule 2 op directive 17 ⟩ ≡
Ceval_phrases ∗ *fsm* = (*Ceval_phrases* ∗) *rule_info__.parser__→fsm_tbl__*;
fsm→post_gps(*sf→p1__*);
fsm→post_error(**new** *ERR_no_T_enum_phrase*);

18. *Rerr_sym_phrase* **rule.**

Rerr_sym_phrase

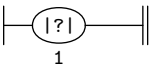


19. *Rerr_sym_phrase*'**s subrule 1.**



⟨ Rerr_sym_phrase subrule 1 op directive 19 ⟩ ≡
```
Ceval_phrases * fsm = ( Ceval_phrases * ) rule_info__.parser__→fsm_tbl__;
fsm→post_gps(sf→p1__);
```

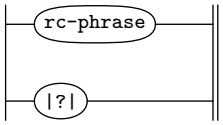
20. *Rerr_sym_phrase*'**s subrule 2.**



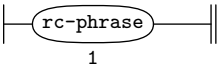
⟨ Rerr_sym_phrase subrule 2 op directive 20 ⟩ ≡
```
Ceval_phrases * fsm = ( Ceval_phrases * ) rule_info__.parser__→fsm_tbl__;
fsm→post_gps(sf→p1__);
fsm→post_error(new ERR_no_errors_phrase);
```

21. *Rrc_phrase* **rule.**

Rrc_phrase

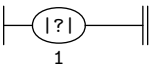


22. *Rrc_phrase*'**s subrule 1.**



⟨ Rrc_phrase subrule 1 op directive 22 ⟩ ≡
```
Ceval_phrases * fsm = ( Ceval_phrases * ) rule_info__.parser__→fsm_tbl__;
fsm→post_gps(sf→p1__);
```

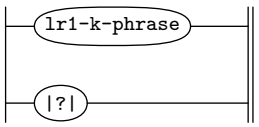
23. *Rrc_phrase*'**s subrule 2.**



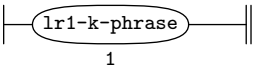
⟨ Rrc_phrase subrule 2 op directive 23 ⟩ ≡
```
Ceval_phrases * fsm = ( Ceval_phrases * ) rule_info__.parser__→fsm_tbl__;
fsm→post_gps(sf→p1__);
fsm→post_error(new ERR_no_rc_phrase);
```

24. *Rlr1_k_phrase* **rule.**

Rlr1_k_phrase

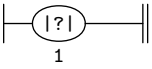


25. *Rlr1_k_phrase*'**s subrule 1.**



⟨ Rlr1_k_phrase subrule 1 op directive 25 ⟩ ≡
Ceval_phrases ∗ *fsm* = (*Ceval_phrases* ∗) *rule_info__.parser__→fsm_tbl__*;
fsm→post_gps(*sf→p1__*);

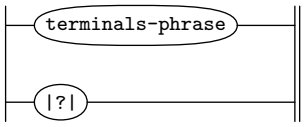
26. *Rlr1_k_phrase*'**s subrule 2.**



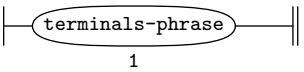
⟨ Rlr1_k_phrase subrule 2 op directive 26 ⟩ ≡
Ceval_phrases ∗ *fsm* = (*Ceval_phrases* ∗) *rule_info__.parser__→fsm_tbl__*;
fsm→post_gps(*sf→p1__*);
fsm→post_error(**new** *ERR_no_lrk_phrase*);

27. *Rterms_phrase* **rule.**

Rterms_phrase

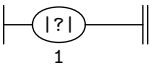


28. *Rterms_phrase*'**s subrule 1.**



⟨ Rterms_phrase subrule 1 op directive 28 ⟩ ≡
Ceval_phrases ∗ *fsm* = (*Ceval_phrases* ∗) *rule_info__.parser__→fsm_tbl__*;
fsm→post_gps(*sf→p1__*);

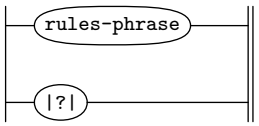
29. *Rterms_phrase*'**s subrule 2.**



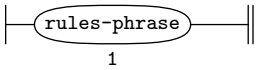
⟨ Rterms_phrase subrule 2 op directive 29 ⟩ ≡
Ceval_phrases ∗ *fsm* = (*Ceval_phrases* ∗) *rule_info__.parser__→fsm_tbl__*;
fsm→post_gps(*sf→p1__*);
fsm→post_error(**new** *ERR_no_terminals_phrase*);

30. *Rrules_phrase* **rule.**

Rrules_phrase

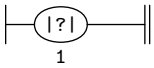


31. *Rrules_phrase*'**s subrule 1.**



⟨ Rrules_phrase subrule 1 op directive 31 ⟩ ≡
Ceval_phrases ∗ *fsm* = (*Ceval_phrases* ∗) *rule_info__.parser__→fsm_tbl__*;
fsm→post_gps(*sf→p1__*);

32. *Rrules_phrase*'**s subrule 2.**



⟨ Rrules_phrase subrule 2 op directive 32 ⟩ ≡
Ceval_phrases ∗ *fsm* = (*Ceval_phrases* ∗) *rule_info__.parser__→fsm_tbl__*;
fsm→post_gps(*sf→p1__*);
fsm→post_error(**new** *ERR_no_rules_phrase*);

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

```
/*
 File: eval_phrases.fsc
 Date and Time: Fri Jan  2 15:33:35 2015
*/
transitive   n
grammar-name "eval_phrases"
name-space   "NS_eval_phrases"
thread-name  "Ceval_phrases"
monolithic   y
file-name    "eval_phrases.fsc"
no-of-T      569
list-of-native-first-set-terminals 3
   LR1_questionable_shift_operator
   LR1_eog
   T_fsm_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
"Evaluate parse phrase sequencer: \n as i use a top / down approach to dispatching the
various phrases."
```

34. Lr1 State Network.

⇒ State: 1 state type: ^{s}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	Rfsm_phrase		2	2	1	|?|		1	2	2	
c	Reval_phrases		1	2	1	eog		1	3	3	
c	Rfsm_phrase		2	1	1	fsm-phrase		1	4	4	
c	Reval_phrases		1	1	1	Rfsm_phrase *Rparallel_phrase*$^{\epsilon}$ *RT_enum_phrase*		1	5	13	

⇒|?| State: 2 state type: ^{r}

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

⇒^{eog} State: 3 state type: ^{r}

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

⇒$^{fsm-phrase}$ State: 4 state type: ^{r}

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

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

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	Rparallel_phrase		3	2	1	ϵ		5	0	5	3
c	Rparallel_phrase		3	1	1	parallel-parser-phrase		5	14	14	
t	Reval_phrases		1	1	2	Rparallel_phrase *RT_enum_phrase*		1	6	13	

⇒$^{Rparallel\_phrase}$ State: 6 state type: ^{s}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	RT_enum_phrase		4	2	1	|?|		6	15	15	
c	RT_enum_phrase		4	1	1	T-enum-phrase		6	16	16	
t	Reval_phrases		1	1	3	RT_enum_phrase *Rlr1_k_phrase*		1	7	13	

⇒$^{RT\_enum\_phrase}$ State: 7 state type: ^{s}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	Rlr1_k_phrase		7	2	1	|?|		7	17	17	
c	Rlr1_k_phrase		7	1	1	lr1-k-phrase		7	18	18	
t	Reval_phrases		1	1	4	Rlr1_k_phrase *Rrc_phrase*		1	8	13	

⇒$^{Rlr1\_k\_phrase}$ State: 8 state type: ^{s}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	Rrc_phrase		6	2	1	|?|		8	19	19	
c	Rrc_phrase		6	1	1	rc-phrase		8	20	20	
t	Reval_phrases		1	1	5	Rrc_phrase *Rterms_phrase*		1	9	13	

⇒$^{Rrc\_phrase}$ State: 9 state type: ^{s}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	Rterms_phrase		8	2	1	|?|		9	21	21	
c	Rterms_phrase		8	1	1	terminals-phrase		9	22	22	
t	Reval_phrases		1	1	6	Rterms_phrase *Rerr_sym_phrase*		1	10	13	

⇒$^{Rterms\_phrase}$ State: 10 state type: ^{s}

←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
c	Rerr_sym_phrase		5	2	1	|?|			10	23	23	
c	Rerr_sym_phrase		5	1	1	error-symbols-phrase			10	24	24	
t	Reval_phrases		1	1	7	Rerr_sym_phrase	*Rrules_phrase*		1	11	13	

⇒*Rerr_sym_phrase* State: 11 state type: *s*

←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
c	Rrules_phrase		9	2	1	|?|			11	25	25	
c	Rrules_phrase		9	1	1	rules-phrase			11	26	26	
t	Reval_phrases		1	1	8	Rrules_phrase	*eog*		1	12	13	

⇒*Rrules_phrase* State: 12 state type: *s*

←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Reval_phrases		1	1	9	eog			1	13	13	

⇒*eog* State: 13 state type: *r*

←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Reval_phrases		1	1	10				1	0	13	2

⇒*parallel−parser−phrase* State: 14 state type: *r*

←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rparallel_phrase		3	1	2				5	0	14	3

⇒|?| State: 15 state type: *r*

←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	RT_enum_phrase		4	2	2				6	0	15	4

⇒*T−enum−phrase* State: 16 state type: *r*

←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	RT_enum_phrase		4	1	2				6	0	16	4

⇒|?| State: 17 state type: *r*

←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rlr1_k_phrase		7	2	2				7	0	17	5

⇒*lr1−k−phrase* State: 18 state type: *r*

←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rlr1_k_phrase		7	1	2				7	0	18	5

⇒|?| State: 19 state type: *r*

←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rrc_phrase		6	2	2				8	0	19	6

⇒*rc−phrase* State: 20 state type: *r*

←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rrc_phrase		6	1	2				8	0	20	6

⇒|?| State: 21 state type: *r*

←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rterms_phrase		8	2	2				9	0	21	7

⇒*terminals−phrase* State: 22 state type: *r*

←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rterms_phrase		8	1	2				9	0	22	7

⇒|?|

State: 23 state type: ^{r}

←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rerr_sym_phrase		5	2	2				10	0	23	8

$\Rightarrow^{error-symbols-phrase}$

State: 24 state type: ^{r}

←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rerr_sym_phrase		5	1	2				10	0	24	8

⇒|?|

State: 25 state type: ^{r}

←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rrules_phrase		9	2	2				11	0	25	9

$\Rightarrow^{rules-phrase}$

State: 26 state type: ^{r}

←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rrules_phrase		9	1	2				11	0	26	9

35. Index.

eval_phrases Grammar

Date: January 2, 2015 at 15:35

File: eval_phrases.lex

Ns: NS_eval_phrases

Version: 1.0 Debug: true

Grammar Comments: Type: Monolithic

Evaluate parse phrase sequencer: as i use a top / down approach to dispatching the various phrases.