1. Copyright.

2. *enumerate_T_alphabet* **grammar.**
Enumerate the Terminal symbols starting at zero and going positively outward. When the enumeration is finished, the global `START_OF_RULES_ENUM` becomes the starting enumerate of the grammar's rules. This is the cut off boundary for shift / reduce LR1 compatibilty. Why? As a state is composed of shift vectors using their enumerates, only terminals up to but not including the start rule's enumerate are needed. The exception to this is the "eosubrule" terminal that represents the reducing subrule situation is not included in the shift set. So it is easy to iterate thru the state's vectors in building up the shift set for the lr1 compatibility check: as the state's vector map is sorted by enumerate value, just stop when the current vector's enumerate is part of the rule's domain.

In this grammar, each rule is a logic sequencer fetching its phase's batch of symbols for the baptism. This grammar demonstrates zero token consumption.

How:
Each terminal phase contains its mapped symbols and create order list. Global Phase table:
O2_xxx are the individual phases.
Grammar Phases:

0 - O2_FSM_PHASE : T_fsm_phrase
1 - O2_PP_PHASE : T_parallel_parser_phrase
2 - O2_T_ENUM_PHASE : T_enum_phrase
3 - O2_ERROR_PHASE : T_error_symbols_phrase
4 - O2_RC_PHASE : T_rc_phrase
5 - O2_LRK_PHASE : T_lr1_k_phrase
6 - O2_T_PHASE : T_terminals_phrase
7 - O2_RULES_PHASE : T_rules_phrase

Within each rule is the symbol iteration.
The enumeration starts with the *lrk* symbols followed by raw characters, Terminals, and Errors. Why is zero the start point? Glad u asked as i use modulo 32 on the terminal enumerate to arrive at its set number and bit position within the set. See set discussion in O_2's library documentation as to the reasons.
Caveat:
As i used a keyword descent trigger parse phases, i must check here if all my phrases are present before the enumeration can take place. Why here? This grammar frontends the triggered rule phase parse due to its related grammars using symbol enumeration as edit checks. So dot the Ts and Is before the ruling!

3. Fsm Cenumerate_T_alphabet class.

4. Cenumerate_T_alphabet constructor directive.

⟨ Cenumerate_T_alphabet constructor directive 4 ⟩ ≡
```
START_OF_RULES_ENUM = 0;
enum_phrase_ = 0;
```

5. Cenumerate_T_alphabet op directive.

⟨ Cenumerate_T_alphabet op directive 5 ⟩ ≡

```
START_OF_RULES_ENUM = 0;
CAbs_lr1_sym * gps = O2_FSM_PHASE;
CAbs_lr1_sym * esym(0);
CAbs_lr1_sym * ph = O2_T_ENUM_PHASE;
if (ph ≡ 0) {
  esym = new ERR_no_T_enum_phrase;
  goto error_fnd;
}
gps = ph;
ph = O2_ERROR_PHASE;
if (ph ≡ 0) {
  esym = new ERR_no_errors_phrase;
  goto error_fnd;
}
gps = ph;
ph = O2_RC_PHASE;
if (ph ≡ 0) {
  esym = new ERR_no_rc_phrase;
  goto error_fnd;
}
gps = ph;
ph = O2_LRK_PHASE;
if (ph ≡ 0) {
  esym = new ERR_no_lrk_phrase;
  goto error_fnd;
}
gps = ph;
ph = O2_T_PHASE;
if (ph ≡ 0) {
  esym = new ERR_no_terminals_phrase;
  goto error_fnd;
}
all_phases_ok: enum_phrase_ = O2_T_ENUM_PHASE;
  return;
error_fnd: parser__→add_token_to_error_queue(*esym);
  if (gps ≠ 0)      /* anchor error against previously good phase */
    esym→set_rc(*gps, __FILE__, __LINE__);
  parser__→set_abort_parse(true);
  return;
```

6. Cenumerate_T_alphabet user-declaration directive.

⟨ Cenumerate_T_alphabet user-declaration directive 6 ⟩ ≡

```
public: NS_yacco2_terminals :: T_enum_phrase * enum_phrase_;
```

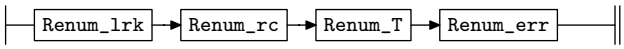
7. Cenumerate_T_alphabet user-prefix-declaration directive.

⟨ Cenumerate_T_alphabet user-prefix-declaration directive 7 ⟩ ≡

```
#include "o2_externs.h"
```

8. *Renumerate_T_alphabet* **rule.**

```
Renumerate_T_alphabet
```



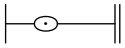
As START_OF_RULES_ENUM is the next enumerate value to use and the enumeration starts from 0, it is more efficient to use this value instead of suming all the totals per terminal type: lrk, rc, err, T now if i was paranoid, maybe the comparison of the calculated against the traversal value would be asserted

⟨ Renumerate_T_alphabet subrule 1 op directive 8 ⟩ ≡

```
Cenumerate_T_alphabet * fsm = ( Cenumerate_T_alphabet * ) rule_info__.parser__→fsm_tbl__;
fsm→enum_phrase_→total_enumerate(START_OF_RULES_ENUM);
lrclog ≪ "Total␣symbols:␣" ≪ START_OF_RULES_ENUM ≪ std::endl;
```

9. *Renum_lrk* **rule.**

```
Renum_lrk
```

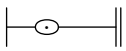


⟨ Renum_lrk subrule 1 op directive 9 ⟩ ≡

```
Cenumerate_T_alphabet * fsm = ( Cenumerate_T_alphabet * ) rule_info__.parser__→fsm_tbl__;
T_lr1_k_phrase * lr_ph = O2_LRK_PHASE;
std::vector < T_terminal_def *> *order = lr_ph→crt_order( );
std::vector < T_terminal_def *> ::iterator i = order→begin( );
std::vector < T_terminal_def *> ::iterator ie = order→end( );
for (fsm→enum_phrase_→start_lrk_enumerate(START_OF_RULES_ENUM); i ≠ ie; ++i,
        ++START_OF_RULES_ENUM) {
  T_terminal_def * tdef = *i;
  tdef→enum_id(START_OF_RULES_ENUM);
}
fsm→enum_phrase_→stop_lrk_enumerate(START_OF_RULES_ENUM − 1);
fsm→enum_phrase_→total_lrk_enumerate(fsm→enum_phrase_→stop_lrk_enumerate( ) −
    fsm→enum_phrase_→start_lrk_enumerate( ) + 1);
lrclog ≪ "Total␣lrk␣symbols:␣" ≪ fsm→enum_phrase_→total_lrk_enumerate( ) ≪ std::endl;
lrclog ≪ "Start␣lrk␣symbol:␣" ≪ fsm→enum_phrase_→start_lrk_enumerate( ) ≪
    "␣Stop␣lrk␣symbol:␣" ≪ fsm→enum_phrase_→stop_lrk_enumerate( ) ≪ std::endl;
```

10. *Renum_rc* **rule.**

Renum_rc

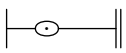


⟨ Renum_rc subrule 1 op directive 10 ⟩ ≡

```
Cenumerate_T_alphabet * fsm = ( Cenumerate_T_alphabet * ) rule_info__.parser__->fsm_tbl__;
T_rc_phrase * lr_ph = O2_RC_PHASE;
std::vector < T_terminal_def *> *order = lr_ph->crt_order( );
std::vector < T_terminal_def *> ::iterator i = order->begin( );
std::vector < T_terminal_def *> ::iterator ie = order->end( );
for (fsm->enum_phrase_->start_rc_enumerate(START_OF_RULES_ENUM); i ≠ ie; ++i,
        ++START_OF_RULES_ENUM) {
  T_terminal_def * tdef = *i;
  tdef->enum_id(START_OF_RULES_ENUM);
}
fsm->enum_phrase_->stop_rc_enumerate(START_OF_RULES_ENUM - 1);
fsm->enum_phrase_->total_rc_enumerate(fsm->enum_phrase_->stop_rc_enumerate( ) -
      fsm->enum_phrase_->start_rc_enumerate( ) + 1);
lrclog ≪ "Total␣rc␣symbols:␣" ≪ fsm->enum_phrase_->total_rc_enumerate( ) ≪ std::endl;
lrclog ≪ "Start␣rc␣symbol:␣" ≪ fsm->enum_phrase_->start_rc_enumerate( ) ≪ "␣Stop␣rc␣symbol:␣" ≪
      fsm->enum_phrase_->stop_rc_enumerate( ) ≪ std::endl;
```

11. *Renum_T* **rule.**

Renum_T

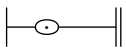


⟨ Renum_T subrule 1 op directive 11 ⟩ ≡

```
Cenumerate_T_alphabet * fsm = ( Cenumerate_T_alphabet * ) rule_info__.parser__->fsm_tbl__;
T_terminals_phrase * lr_ph = O2_T_PHASE;
std::vector < T_terminal_def *> *order = lr_ph->crt_order( );
std::vector < T_terminal_def *> ::iterator i = order->begin( );
std::vector < T_terminal_def *> ::iterator ie = order->end( );
for (fsm->enum_phrase_->start_T_enumerate(START_OF_RULES_ENUM); i ≠ ie; ++i,
        ++START_OF_RULES_ENUM) {
  T_terminal_def * tdef = *i;
  tdef->enum_id(START_OF_RULES_ENUM);
}
fsm->enum_phrase_->stop_T_enumerate(START_OF_RULES_ENUM - 1);
fsm->enum_phrase_->total_T_enumerate(fsm->enum_phrase_->stop_T_enumerate( ) -
      fsm->enum_phrase_->start_T_enumerate( ) + 1);
lrclog ≪ "Total␣T␣symbols:␣" ≪ fsm->enum_phrase_->total_T_enumerate( ) ≪ std::endl;
lrclog ≪ "Start␣T␣symbol:␣" ≪ fsm->enum_phrase_->start_T_enumerate( ) ≪ "␣Stop␣T␣symbol:␣" ≪
      fsm->enum_phrase_->stop_T_enumerate( ) ≪ std::endl;
```

12. *Renum_err* **rule.**

```
Renum_err
```



⟨ Renum_err subrule 1 op directive 12 ⟩ ≡

```
Cenumerate_T_alphabet * fsm = ( Cenumerate_T_alphabet * ) rule_info__.parser__->fsm_tbl__;
T_error_symbols_phrase * lr_ph = O2_ERROR_PHASE;
std::vector < T_terminal_def *> *order = lr_ph->crt_order( );
std::vector < T_terminal_def *> ::iterator i = order->begin( );
std::vector < T_terminal_def *> ::iterator ie = order->end( );
for (fsm->enum_phrase_->start_err_enumerate(START_OF_RULES_ENUM); i ≠ ie; ++i,
        ++START_OF_RULES_ENUM) {
  T_terminal_def * tdef = *i;
  tdef->enum_id(START_OF_RULES_ENUM);
}
fsm->enum_phrase_->stop_err_enumerate(START_OF_RULES_ENUM − 1);
fsm->enum_phrase_->total_err_enumerate(fsm->enum_phrase_->stop_err_enumerate( ) −
      fsm->enum_phrase_->start_err_enumerate( ) + 1);
lrclog ≪ "Total␣error␣symbols:␣" ≪ fsm->enum_phrase_->total_err_enumerate( ) ≪ std::endl;
lrclog ≪ "Start␣error␣symbol:␣" ≪ fsm->enum_phrase_->start_err_enumerate( ) ≪
      "␣Stop␣error␣symbol:␣" ≪ fsm->enum_phrase_->stop_err_enumerate( ) ≪ std::endl;
```

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

```
/*
 File: enumerate_T_alphabet.fsc
 Date and Time: Fri Jan  2 15:33:33 2015
*/
transitive   n
grammar-name "enumerate_T_alphabet"
name-space   "NS_enumerate_T_alphabet"
thread-name  "Cenumerate_T_alphabet"
monolithic   y
file-name    "enumerate_T_alphabet.fsc"
no-of-T      569
list-of-native-first-set-terminals 0
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
"Enumerate grammar's terminal symbols: \na 0 and a 1, ... the oracle for parsing
lookups."
```

14. Lr1 State Network.

⇒ State: 1 state type: $^{s/r}$

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	Renum_lrk		2	1	1	ϵ		1	0	1	1
c	Renumerate_T_alphabet		1	1	1	Renum_lrk $\underline{Renum\_rc^{\epsilon}}$ $\underline{Renum\_T^{\epsilon}}$ ...		1	2	5	

⇒$^{Renum\_lrk}$ State: 2 state type: $^{s/r}$

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	Renum_rc		3	1	1	ϵ		2	0	2	1
t	Renumerate_T_alphabet		1	1	2	Renum_rc $\underline{Renum\_T^{\epsilon}}$ $\underline{Renum\_err^{\epsilon}}$ ...		1	3	5	

⇒$^{Renum\_rc}$ State: 3 state type: $^{s/r}$

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	Renum_T		4	1	1	ϵ		3	0	3	1
t	Renumerate_T_alphabet		1	1	3	Renum_T $\underline{Renum\_err^{\epsilon}}$		1	4	5	

⇒$^{Renum\_T}$ State: 4 state type: $^{s/r}$

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	Renum_err		5	1	1	ϵ		4	0	4	1
t	Renumerate_T_alphabet		1	1	4	Renum_err		1	5	5	

⇒$^{Renum\_err}$ State: 5 state type: ^{r}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Renumerate_T_alphabet		1	1	5			1	0	5	1

15. Index.

ϵ : 9, 10, 11, 12.
__FILE__: 5.
__LINE__: 5.
add_token_to_error_queue: 5.
all_phases_ok: 5.
begin: 9, 10, 11, 12.
CAbs_lr1_sym: 5.
Cenumerate_T_alphabet: 8, 9, 10, 11, 12.
crt_order: 9, 10, 11, 12.
end: 9, 10, 11, 12.
endl: 8, 9, 10, 11, 12.
enum_id: 9, 10, 11, 12.
*enum_phrase_*: 4, 5, 6, 8, 9, 10, 11, 12.
enumerate_T_alphabet: 2.
ERR_no_errors_phrase: 5.
ERR_no_lrk_phrase: 5.
ERR_no_rc_phrase: 5.
ERR_no_T_enum_phrase: 5.
ERR_no_terminals_phrase: 5.
error_fnd: 5.
esym: 5.
fsm: 8, 9, 10, 11, 12.
fsm_tbl__: 8, 9, 10, 11, 12.
gps: 5.
ie: 9, 10, 11, 12.
iterator: 9, 10, 11, 12.
lr: 2.
lr_ph: 9, 10, 11, 12.
lrclog: 8, 9, 10, 11, 12.
NS_yacco2_terminals: 6.
order: 9, 10, 11, 12.
O2_ERROR_PHASE: 5, 12.
O2_FSM_PHASE: 5.
O2_LRK_PHASE: 5, 9.
O2_RC_PHASE: 5, 10.
O2_T_ENUM_PHASE: 5.
O2_T_PHASE: 5, 11.
O2_xxx: 2.
parser__: 5, 8, 9, 10, 11, 12.
ph: 5.
Renum_err: 8.
Renum_lrk: 8.
Renum_rc: 8.
Renum_T: 8.
Renum_err: 12.
Renum_lrk: 9.
Renum_rc: 10.
Renum_T: 11.
Renumerate_T_alphabet: 8.
rule_info__: 8, 9, 10, 11, 12.
set_abort_parse: 5.
set_rc: 5.
start_err_enumerate: 12.
start_lrk_enumerate: 9.
START_OF_RULES_ENUM: 2, 4, 5, 8, 9, 10, 11, 12.
start_rc_enumerate: 10.
start_T_enumerate: 11.
std: 8, 9, 10, 11, 12.
stop_err_enumerate: 12.
stop_lrk_enumerate: 9.
stop_rc_enumerate: 10.
stop_T_enumerate: 11.
T_enum_phrase: 6.
T_error_symbols_phrase: 12.
T_lr1_k_phrase: 9.
T_rc_phrase: 10.
T_terminal_def: 9, 10, 11, 12.
T_terminals_phrase: 11.
tdef: 9, 10, 11, 12.
total_enumerate: 8.
total_err_enumerate: 12.
total_lrk_enumerate: 9.
total_rc_enumerate: 10.
total_T_enumerate: 11.
true: 5.
vector: 9, 10, 11, 12.

⟨ Cenumerate_T_alphabet constructor directive 4 ⟩
⟨ Cenumerate_T_alphabet op directive 5 ⟩
⟨ Cenumerate_T_alphabet user-declaration directive 6 ⟩
⟨ Cenumerate_T_alphabet user-prefix-declaration directive 7 ⟩
⟨ Renum_T subrule 1 op directive 11 ⟩
⟨ Renum_err subrule 1 op directive 12 ⟩
⟨ Renum_lrk subrule 1 op directive 9 ⟩
⟨ Renum_rc subrule 1 op directive 10 ⟩
⟨ Renumerate_T_alphabet subrule 1 op directive 8 ⟩

enumerate_T_alphabet Grammar

Date: January 2, 2015 at 15:35

File: enumerate_T_alphabet.lex

Ns: NS_enumerate_T_alphabet

Version: 1.0 Debug: false

Grammar Comments: Type: Monolithic

Enumerate grammar's terminal symbols: a 0 and a 1, ... the oracle for parsing lookups.

	Section	Page
Copyright	1	1
enumerate_T_alphabet **grammar**	2	2
Fsm Cenumerate_T_alphabet class	3	2
Cenumerate_T_alphabet constructor directive	4	2
Cenumerate_T_alphabet op directive	5	3
Cenumerate_T_alphabet user-declaration directive	6	3
Cenumerate_T_alphabet user-prefix-declaration directive	7	3
Renumerate_T_alphabet rule	8	4
Renum_lrk rule	9	4
Renum_rc rule	10	5
Renum_T rule	11	5
Renum_err rule	12	6
First Set Language for O_2^{linker}	13	7
Lr1 State Network	14	8
Index	15	9