1. Copyright.

2. *rules_use_cnt* grammar.

Determine each rule's use count for generating recycled rules tables. It answers the question: What is the maximum number of rules needed in each recycled rules table?. If the rule has recursion then indicate in rule's definition. The heart of the algorithm is to determine the maximum rule's use count against all rules rhs. The grammar tree is walked in top-down fashion: prefix using a filter for only rule-def, subrule-def, and referenced-rules.

Stage 1: Build per rule its rule use skeleton. This is done upon entry into the grammar using the "op directive" before the grammar parses. Removeal of all the T vocabulary from the rhs leave its bones. If the resultant rhs is empty then it does not partake in the analysis as there are no rule used. The rule's skeleton is deposited within its "rule-def" for the balance of the stages.

Stage 2: Determine rule's use count per skeleton.
The max use count is posted in the rule's definition to generate the space requirement for the rule's recycling table.

Remarks:
The walked skeleton checks the use count of indirect rule's in use. For example:

Rule		*subrule's symbols*
Rs	$\rightarrow$	Ra Rxx Ra Rxxx $\perp$
Ra	$\rightarrow$	a
	$\rightarrow$	Ra a
Rxx	$\rightarrow$	x Ra x Ra b
Rxxx	$\rightarrow$	x Ra x Ra x Ra

What is the maximum use count of Ra in the above grammar? It is 6 caused by the 1st rule. Why? Here is an example to support my statement.

Parse Stack	*input string to parse*	Comments
	a a x a x a b a x a x a x a a	
a	a x a x a b a x a x a x a a	
Ra	a x a x a b a x a x a x a a	
Ra a	x a x a b a x a x a x a a	recursion: 2 Ra — lhs and stacked rhs
Ra	x a x a b a x a x a x a a	Ra lhs, popped rhs Ra recycled
Ra x	a x a b a x a x a x a a	
Ra x a	x a b a x a x a x a a	
Ra x Ra	x a b a x a x a x a a	
Ra x Ra x	a b a x a x a x a a	
Ra x Ra x a	b a x a x a x a a	
Ra x Ra x Ra	b a x a x a x a a	
Ra x Ra x Ra b	a x a x a x a a	
Ra Rxx	a x a x a x a a	
Ra Rxx a	x a x a x a a	
Ra Rxx Ra	x a x a x a a	
Ra Rxx Ra x	a x a x a a	
Ra Rxx Ra x a	x a x a a	
Ra Rxx Ra x Ra	x a x a a	
Ra Rxx Ra x Ra x	a x a a	
Ra Rxx Ra x Ra x a	x a a	
Ra Rxx Ra x Ra x Ra	x a a	
Ra Rxx Ra x Ra x Ra x	a a	

Ra Rxx Ra x Ra x Ra x a	a
Ra Rxx Ra x Ra x Ra x Ra	a
Ra Rxx Ra x Ra x Ra x Ra a	note: extra Ra for lhs/rhs from recursion
Ra Rxx Ra x Ra x Ra x Ra	though 5 Ra on stack, above lhs/rhs requires 6
Ra Rxx Ra Rxxx	where Rxxx's rhs needs 4 before it reduces

There are 2 direct Ra used and 4 indirect Ra from Rxxx's rhs before it is reduced: 3 direct Ra used + 1 Ra for the recursion on its last recognized Ra before Rxxx's rhs is completely recognized. Recursion of Ra demands 2 Ra: lhs / rhs. The first "a" recognized creates a Ra that is pushed onto the stack. The following "a" get recognized by the Ra left recursion rule. The first Ra sits on the parse stack whilst the second Ra representing the lhs gets ready for the reduce to take place. With each recurse reduction the rhs's popped Ra is freed up for recycling. After the Rxxx reduction, its 3 Ra also get recycled for use. On the parse stack there still remains 2 active Ra of Rs's rhs that remains to be recognized. If Rxxx was not present in the grammar, the Ra maximum use count would be 5: 2 direct and 2+1 indirect counts from Rxx.

3. Fsm Crules_use_cnt class.

4. Crules_use_cnt op directive.

Build use skeletons per rule definition. As the container is of tree type, the first token is the "start" rule. I can get its tree node and breadth walk it for its brothers. The rules list for "use count" assessment is built: Rules vocabulary - "start" rule. The "start" rule is removed as it never appears within a rhs as per a grammar's definition.

⟨ Crules_use_cnt op directive 4 ⟩ ≡

```
  tok_can < AST *> *can = ( tok_can < AST *> * ) parser( )→token_supplier( );
  AST * start_rule_t = can→ast(0);
  AST * t = start_rule_t; for ( ; t ≠ 0; t = AST::get_younger_sibling(*t, 1)) { rule_def * rd = ( rule_def * )
      AST::content(*t);
  AST * use_skeleton_t = bld_rule_s_use_skeleton(t);
  if (use_skeleton_t ≠ 0) rd→rule_use_skeleton(use_skeleton_t);
  if (t ≠ start_rule_t) rules_list_for_use_cnt_.push_back(rd);
  } rule_def_ = 0;
  subrule_def_ = 0;
```

5. Crules_use_cnt user-declaration directive.

⟨ Crules_use_cnt user-declaration directive 5 ⟩ ≡

```
public: std::list < NS_yacco2_terminals::rule_def *> rules_list_for_use_cnt_;
  NS_yacco2_terminals::rule_def * rule_def_;
  NS_yacco2_terminals::T_subrule_def * subrule_def_;
  void mark_recursion_rule_use(NS_yacco2_terminals::refered_rule * Refered_rule);
  yacco2::AST * bld_rule_s_use_skeleton(yacco2::AST * Rule_t);
```

6. Crules_use_cnt user-implementation directive.

⟨ Crules_use_cnt user-implementation directive 6 ⟩ ≡

```
void Crules_use_cnt::mark_recursion_rule_use(NS_yacco2_terminals::refered_rule * Refered_rule)
{
  using namespace NS_yacco2_terminals;
  rule_in_stbl * rule_in_tbl = Refered_rule→Rule_in_stbl( );
  rule_def * rd = rule_in_tbl→r_def( );
  if (rd ≡ rule_def_) {
    rd→recursive(YES);
    rd→lhs_use_cnt(1);
  }
}
```

7. *bld_rule_s_use_skeleton*.

⟨ More code 7 ⟩ ≡

```
AST * Crules_use_cnt::bld_rule_s_use_skeleton(AST * Rule_t){ using namespace NS_yacco2_T_enum;
        set < int > rules_use_cnt_filter;
    rules_use_cnt_filter.insert(T_Enum::T_T_subrule_def_);
    rules_use_cnt_filter.insert(T_Enum::T_rule_def_);
    rules_use_cnt_filter.insert(T_Enum::T_refered_rule_);
    tok_can_ast_functor rules_use_walk_functr;
    ast_postfix rules_use_walk(*Rule_t, &rules_use_walk_functr, &rules_use_cnt_filter, ACCEPT_FILTER);
    tok_can < AST *> rules_use_can(rules_use_walk);
    int x(0);
    CAbs_lr1_sym * sym = rules_use_can[x];
    rule_def * rd(0);
    AST * lk_rr_t(0);
    AST * rr_t(0);
    AST * sr_t(0);
    AST * lk_sr_t(0);
    AST * r_t(0);
    for ( ; sym→enumerated_id() ≠ LR1_Eog; ++x, sym = rules_use_can[x]) {
      switch (sym→enumerated_id()) {
      case T_Enum::T_rule_def_:
        {
          if (sr_t ≡ 0) return 0;    /* no rhs with refered rules */
          r_t = new AST(*sym);
          AST::join_pts(*r_t, *sr_t);
          return r_t;
        }
      case T_Enum::T_T_subrule_def_:
        {
          if (rr_t ≡ 0) break;    /* no refered rules */
          AST * srt = new AST(*sym);
          AST::join_pts(*srt, *rr_t);
          lk_rr_t = 0;
          rr_t = 0;
          if (sr_t ≡ 0) {
            sr_t = srt;
            lk_sr_t = srt;
          }
          else {
            AST::join_sts(*lk_sr_t, *srt);
            lk_sr_t = srt;
          }
          break;
        }
      case T_Enum::T_refered_rule_:
        {
          AST * rrt = new AST(*sym);
          if (rr_t ≡ 0) {
            rr_t = rrt;
            lk_rr_t = rrt;
          }
```

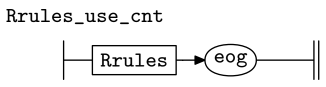
```
            else {
              AST::join_sts(*lk_rr_t, *rrt);
              lk_rr_t = rrt;
            }
            break;
          }
        }    /* switch */
      }    /* for */
      return 0; }
```

8. Crules_use_cnt user-prefix-declaration directive.

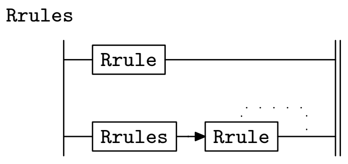
⟨ Crules_use_cnt user-prefix-declaration directive 8 ⟩ ≡

```
#include "o2_externs.h"
  extern int MAX_USE_CNT_RxR(NS_yacco2_terminals::rule_def * Rule_use,
      NS_yacco2_terminals::rule_def * Against_rule);
```

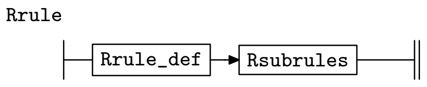
9. *Rrules_use_cnt* **rule.**



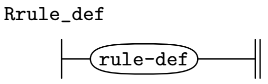
10. *Rrules* **rule.**



11. *Rrule* **rule.**



12. *Rrule_def* **rule.**

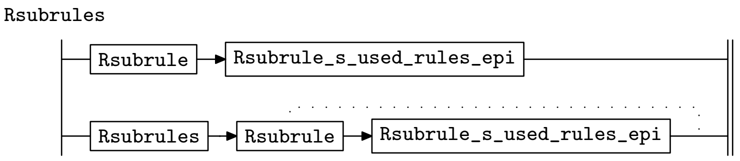


Post the rule's used cnt within its definition to be used in generating the recycle tables. Remember the current rule is assessing its indirect rule usage. It is the rule usage definition that gets updated for its recycle table requirements and not the current rule.

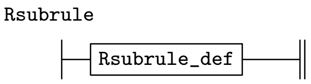
⟨ Rrule_def subrule 1 op directive 12 ⟩ ≡

```
Crules_use_cnt *fsm = ( Crules_use_cnt * ) rule_info__.parser__→fsm_tbl__;
fsm→rule_def_ = sf→p1__;
std::list < NS_yacco2_terminals::rule_def *> ::iterator i = fsm→rules_list_for_use_cnt_.begin( );
std::list < NS_yacco2_terminals::rule_def *> ::iterator ie = fsm→rules_list_for_use_cnt_.end( );
for ( ; i ≠ ie; ++i) {
  rule_def *for_rule = *i;
  int use_cnt = MAX_USE_CNT_RxR(for_rule, fsm→rule_def_);
  if (for_rule→rhs_use_cnt( ) < use_cnt) {
    for_rule→rhs_use_cnt(use_cnt);
  }
}
```

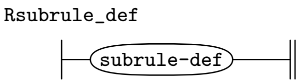
13. *Rsubrules* **rule.**



14. *Rsubrule* **rule.**



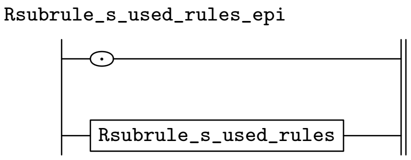
15. *Rsubrule_def* **rule.**



⟨ Rsubrule_def subrule 1 op directive 15 ⟩ ≡

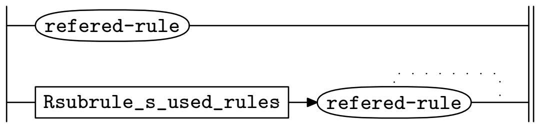
```
Crules_use_cnt *fsm = ( Crules_use_cnt * ) rule_info__.parser__→fsm_tbl__;
fsm→subrule_def_ = sf→p1__;
```

16. *Rsubrule_s_used_rules_epi* **rule.**

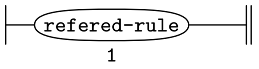


17. *Rsubrule_s_used_rules* **rule.**

`Rsubrule_s_used_rules`

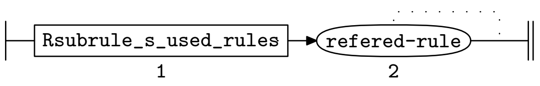


18. *Rsubrule_s_used_rules*'**s subrule 1.**



⟨ Rsubrule_s_used_rules subrule 1 op directive 18 ⟩ ≡
  Crules_use_cnt ∗ *fsm* = (*Crules_use_cnt* ∗) *rule_info__.parser__*→*fsm_tbl__*;
  fsm→*mark_recursion_rule_use*(*sf*→*p1__*);

19. *Rsubrule_s_used_rules*'**s subrule 2.**



⟨ Rsubrule_s_used_rules subrule 2 op directive 19 ⟩ ≡
  Crules_use_cnt ∗ *fsm* = (*Crules_use_cnt* ∗) *rule_info__.parser__*→*fsm_tbl__*;
  fsm→*mark_recursion_rule_use*(*sf*→*p2__*);

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

```
  /*
   File: rules_use_cnt.fsc
   Date and Time: Fri Jan  2 15:33:55 2015
  */
  transitive   n
  grammar-name "rules_use_cnt"
  name-space   "NS_rules_use_cnt"
  thread-name  "Crules_use_cnt"
  monolithic   y
  file-name    "rules_use_cnt.fsc"
  no-of-T      569
  list-of-native-first-set-terminals 1
     rule_def
  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
  "Optimization: Count ‘‘rules used’’ \n to lower new / delete rule cycles while
parsing."
```

21. Lr1 State Network.

$\Rightarrow$ State: 1 state type: ^{s}

	← rule →	R#	sr#	Po	← subrule element →	Brn	Gto	Red	LA
c	Rrule_def	4	1	1	rule-def	1	2	2	
c	Rrules_use_cnt	1	1	1	Rrules $\underline{eog}$	1	3	4	
c	Rrules	2	2	1	Rrules $\underline{Rrule}$	1	3	5	
c	Rrules	2	1	1	Rrule	1	17	17	
c	Rrule	3	1	1	Rrule_def $\underline{Rsubrules}$	1	6	8	

$\Rightarrow^{rule-def}$ State: 2 state type: ^{r}

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

$\Rightarrow^{Rrules}$ State: 3 state type: ^{s}

	← rule →	R#	sr#	Po	← subrule element →	Brn	Gto	Red	LA
t	Rrules_use_cnt	1	1	2	eog	1	4	4	
c	Rrule_def	4	1	1	rule-def	3	2	2	
t	Rrules	2	2	2	Rrule	1	5	5	
c	Rrule	3	1	1	Rrule_def $\underline{Rsubrules}$	3	6	8	

$\Rightarrow^{eog}$ State: 4 state type: ^{r}

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

$\Rightarrow^{Rrule}$ State: 5 state type: ^{r}

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

$\Rightarrow^{Rrule\_def}$ State: 6 state type: ^{s}

	← rule →	R#	sr#	Po	← subrule element →	Brn	Gto	Red	LA
c	Rsubrule_def	7	1	1	subrule-def	6	7	7	
t	Rrule	3	1	2	Rsubrules	3	8	8	
c	Rsubrules	5	2	1	Rsubrules $\underline{Rsubrule}$	6	8	11	
c	Rsubrules	5	1	1	Rsubrule $\underline{Rsubrule\_s\_used\_rules\_epi^{\epsilon}}$	6	15	16	
c	Rsubrule	6	1	1	Rsubrule_def	6	14	14	

$\Rightarrow^{subrule-def}$ State: 7 state type: ^{r}

	← rule →	R#	sr#	Po	← subrule element →	Brn	Gto	Red	LA
t	Rsubrule_def	7	1	2		6	0	7	4

$\Rightarrow^{Rsubrules}$ State: 8 state type: $^{s/r}$

	← rule →	R#	sr#	Po	← subrule element →	Brn	Gto	Red	LA
t	Rrule	3	1	3		3	0	8	3
c	Rsubrule_def	7	1	1	subrule-def	8	7	7	
t	Rsubrules	5	2	2	Rsubrule $\underline{Rsubrule\_s\_used\_rules\_epi^{\epsilon}}$	6	9	11	
c	Rsubrule	6	1	1	Rsubrule_def	8	14	14	

$\Rightarrow^{Rsubrule}$ State: 9 state type: $^{s/r}$

	← rule →	R#	sr#	Po	← subrule element →	Brn	Gto	Red	LA
c	Rsubrule_s_used_rules_epi	8	1	1	ϵ	9	0	9	5
c	Rsubrule_s_used_rules	9	1	1	refered-rule	9	10	10	

	rule	R#	sr#	Po	subrule element	Brn	Gto	Red	LA
t	Rsubrules	5	2	3	Rsubrule_s_used_rules_epi	6	11	11	
c	Rsubrule_s_used_rules_epi	8	2	1	Rsubrule_s_used_rules	9	12	12	
c	Rsubrule_s_used_rules	9	2	1	Rsubrule_s_used_rules $\underline{refered-rule}$	9	12	13	

$\Rightarrow^{refered-rule}$ State: 10 state type: ^{r}

	← rule →	R#	sr#	Po	← subrule element →	Brn	Gto	Red	LA
t	Rsubrule_s_used_rules	9	1	2		9	0	10	4

$\Rightarrow^{Rsubrule\_s\_used\_rules\_epi}$ State: 11 state type: ^{r}

	← rule →	R#	sr#	Po	← subrule element →	Brn	Gto	Red	LA
t	Rsubrules	5	2	4		6	0	11	5

$\Rightarrow^{Rsubrule\_s\_used\_rules}$ State: 12 state type: $^{s/r}$

	← rule →	R#	sr#	Po	← subrule element →	Brn	Gto	Red	LA
t	Rsubrule_s_used_rules_epi	8	2	2		9	0	12	5
t	Rsubrule_s_used_rules	9	2	2	refered-rule	9	13	13	

$\Rightarrow^{refered-rule}$ State: 13 state type: ^{r}

	← rule →	R#	sr#	Po	← subrule element →	Brn	Gto	Red	LA
t	Rsubrule_s_used_rules	9	2	3		9	0	13	4

$\Rightarrow^{Rsubrule\_def}$ State: 14 state type: ^{r}

	← rule →	R#	sr#	Po	← subrule element →	Brn	Gto	Red	LA
t	Rsubrule	6	1	2		8	0	14	4

$\Rightarrow^{Rsubrule}$ State: 15 state type: $^{s/r}$

	← rule →	R#	sr#	Po	← subrule element →	Brn	Gto	Red	LA
c	Rsubrule_s_used_rules_epi	8	1	1	ϵ	15	0	15	5
c	Rsubrule_s_used_rules	9	1	1	refered-rule	15	10	10	
t	Rsubrules	5	1	2	Rsubrule_s_used_rules_epi	6	16	16	
c	Rsubrule_s_used_rules_epi	8	2	1	Rsubrule_s_used_rules	15	12	12	
c	Rsubrule_s_used_rules	9	2	1	Rsubrule_s_used_rules $\underline{refered-rule}$	15	12	13	

$\Rightarrow^{Rsubrule\_s\_used\_rules\_epi}$ State: 16 state type: ^{r}

	← rule →	R#	sr#	Po	← subrule element →	Brn	Gto	Red	LA
t	Rsubrules	5	1	3		6	0	16	5

$\Rightarrow^{Rrule}$ State: 17 state type: ^{r}

	← rule →	R#	sr#	Po	← subrule element →	Brn	Gto	Red	LA
t	Rrules	2	1	2		1	0	17	3

22. Index.

`rules_use_cnt Grammar`

`Date: January 2, 2015 at 15:39`

`File: rules_use_cnt.lex`

`Ns: NS_rules_use_cnt`

Version: 1.0 Debug: false

Grammar Comments: Type: Monolithic

Optimization: Count “rules used” to lower new / delete rule cycles while parsing.