A Prefix-Correct Earley Recognizer for Multiple Context-Free Grammars

Makoto Kanazawa
National Institute of Informatics
2–1–2 Hitotsubashi, Chiyoda-ku, Tokyo, 101–8430, Japan
kanazawa@nii.ac.jp

Abstract

We present a method for deriving an Earley recognizer for multiple context-free grammars with the correct prefix property. This is done by representing an MCFG by a Datalog program and applying generalized supplementary magic-sets rewriting. To secure the correct prefix property, a simple extra rewriting must be performed before the magic-sets rewriting. The correctness of the method is easy to see, and a straightforward application of the method to tree-adjoining grammars yields a recognizer whose running time is $O(n^6)$.

1 Deriving an Earley-style recognizer by magic-sets rewriting

We use the following 2-MCFG generating $\mathrm{RESP}_+ = \{\, \mathsf{a}_1^m \mathsf{a}_2^m \mathsf{b}_1^n \mathsf{b}_2^n \mathsf{a}_3^m \mathsf{a}_4^m \mathsf{b}_3^n \mathsf{b}_4^n \mid m, n \geq 1 \,\}$ as our running example:[1]

(1) $S(x_1 y_1 x_2 y_2) :- P(x_1, x_2),\ Q(y_1, y_2).$
$P(\mathsf{a}_1\mathsf{a}_2, \mathsf{a}_3\mathsf{a}_4).$
$P(\mathsf{a}_1 x_1 \mathsf{a}_2, \mathsf{a}_3 x_2 \mathsf{a}_4) :- P(x_1, x_2).$
$Q(\mathsf{b}_1\mathsf{b}_2, \mathsf{b}_3\mathsf{b}_4).$
$Q(\mathsf{b}_1 y_1 \mathsf{b}_2, \mathsf{b}_3 y_2 \mathsf{b}_4) :- Q(y_1, y_2).$

The equivalence between this MCFG and the following *Datalog* program (i.e., function-free Horn clause logic program) is straightforward:

(2) $S(i, m) :- P(i, j, k, l),\ Q(j, k, l, m).$
$P(i, k, l, n) :- \mathsf{a}_1(i, j),\ \mathsf{a}_2(j, k),\ \mathsf{a}_3(l, m),\ \mathsf{a}_4(m, n).$
$P(i, l, m, p) :- \mathsf{a}_1(i, j),\ P(j, k, n, o),\ \mathsf{a}_2(k, l),\ \mathsf{a}_3(m, n),\ \mathsf{a}_4(o, p).$
$Q(i, k, l, n) :- \mathsf{b}_1(i, j),\ \mathsf{b}_2(j, k),\ \mathsf{b}_3(l, m),\ \mathsf{b}_4(m, n).$
$Q(i, l, m, p) :- \mathsf{b}_1(i, j),\ \mathsf{b}_2(k, l),\ \mathsf{b}_3(m, n),\ Q(j, k, n, o),\ \mathsf{b}_4(o, p).$

Nonterminals and terminals of the grammar become *intensional* and *extensional* predicates of the Datalog program, respectively. The program (2) together with the *extensional database* $\{a_1(0, 1), \ldots, a_n(\boldsymbol{n-1}, \boldsymbol{n})\}$ derives $S(0, \boldsymbol{n})$ if and only if $a_1 \ldots a_n$ is in the language of the grammar (1).[2] Programs like (2) may be used as *deduction systems* (Shieber et al., 1995) or *uninstantiated parsing systems* (Sikkel, 1997) for chart parsing.

As demonstrated by Kanazawa (2007), Datalog offers an elegant unifying treatment of parsing for various string and tree grammars as well as tactical generation (surface realization) from logical forms reprensented by lambda terms.[3] Since deduction systems for parsing can be thought of as Datalog programs, we may view various *parsing schemata* (Sikkel, 1997) (i.e., mappings from grammars to deduction systems) as transformations of Datalog programs.

Magic-sets rewriting of Datalog programs is a technique to allow bottom-up evaluation to incorporate top-down prediction. As is well-understood, if we apply *generalized supplementary magic-sets rewriting* (Beeri and Ramakrishnan, 1991) to a Datalog program representing a context-free grammar, the result is essentially the deduction system for Earley's algorithm. Let us see how this technique applies to the program (2) (see Ullman (1989a; 1989b) for exposition).

First, *adornments* are attached to predicates, which indicate the free/bound status of each argu-

[1] Note that we are using the notation of *elementary formal systems* (Smullyan, 1961; Arikawa et al., 1992) aka *literal movement grammars* (Groenink, 1997), instead of that of Seki et al. (1991), to represent MCFG rules.

[2] If i is a natural number, we let "$\boldsymbol{i}$" stand for the constant symbol representing i. We let "0", "1", etc., stand for themselves.

[3] Not only does the Dataog representation imply the existence of a polynomial-time algorithm for recognition, but it also serves to establish the tight complexity bound, namely *LOGCFL*, which presumably is a small subclass of P. See Kanazawa (2007) for details.

1: $S^{bf}(i,m)$:– $P^{bfff}(i,j,k,l)$, $Q^{bbbf}(j,k,l,m)$.
2: $P^{bfff}(i,k,l,n)$:– $\mathtt{a}_1^{bf}(i,j)$, $\mathtt{a}_2^{bf}(j,k)$, $\mathtt{a}_3^{ff}(l,m)$, $\mathtt{a}_4^{bf}(m,n)$.
3: $P^{bfff}(i,l,m,p)$:– $\mathtt{a}_1^{bf}(i,j)$, $P^{bfff}(j,k,n,o)$, $\mathtt{a}_2^{bf}(k,l)$, $\mathtt{a}_3^{fb}(m,n)$, $\mathtt{a}_4^{bf}(o,p)$.
4: $Q^{bbbf}(i,k,l,n)$:– $\mathtt{b}_1^{bf}(i,j)$, $\mathtt{b}_2^{bb}(j,k)$, $\mathtt{b}_3^{bf}(l,m)$, $\mathtt{b}_4^{bf}(m,n)$.
5: $Q^{bbbf}(i,l,m,p)$:– $\mathtt{b}_1^{bf}(i,j)$, $\mathtt{b}_2^{fb}(k,l)$, $\mathtt{b}_3^{bf}(m,n)$, $Q^{bbbf}(j,k,n,o)$, $\mathtt{b}_4^{bf}(o,p)$.

Figure 1: Adorned Datalog program.

ment in top-down evaluation of the program (Figure 1). These adornments determine what arguments newly created predicates take in the new program.

There are two classes of new predicates. For each intensional predicate A, a corresponding *magic predicate* m_A is created, which takes only the bound arguments of A as its arguments. For each rule with n subgoals, *supplementary predicates* $sup_{i.j}$ for $j = 1, \ldots, n-1$ are introduced, where i is the rule number. The set of arguments of $sup_{i.j}$ is the intersection of two sets: the first set consists of the bound variables in the head of the rule and the variables in the first j subgoals, while the second set consists of the variables in the remaining subgoals and the head. The new program is in Figure 2. The rules for the magic predicates express top-down prediction, while the remaining rules serve to binarize the original rules, adding magic predicates as extra subgoals.

The program in Figure 2 can be used as a correct recognizer in combination with the control algorithm in Kanazawa (2007). This algorithm, however, reads the entire input string before accepting or rejecting it, so it cannot satisfy the *correct prefix property* with any program.[4] For this reason, we use the following alternative control algorithm in this paper, which is designed to reject the input as soon as the next input symbol no longer immediately contributes to deriving new facts. Note that the input string $a_1 \ldots a_n$ is represented by an extensional database $\{a_1(0,1), \ldots, a_n(\boldsymbol{n-1}, \boldsymbol{n})\}$.

Chart recognizer control algorithm

1. (INITIALIZE) Initialize the chart to the empty set, the agenda to the singleton $\{m\_S(0)\}$, and i to 0.
2. Repeat the following steps:
   (a) Repeat the following steps until the agenda is exhausted:
      i. Remove a fact from the agenda and call it the trigger.
      ii. Add the trigger to the chart.
      iii. (PREDICT/COMPLETE) Generate all facts that are immediate consequences of the trigger together with all facts in the chart, and add to the agenda those generated facts that are neither already in the chart nor in the agenda.
   (b) If there is no more fact in the input database, go to step 3.
   (c) i. Remove the next fact $a_{i+1}(\boldsymbol{i}, \boldsymbol{i+1})$ from the input database and call it the trigger.
      ii. (SCAN) Generate all facts that are immediate consequences of the trigger together with all facts in the chart, and add the generated facts to the agenda.
      iii. If the agenda is empty, reject the input; otherwise increment i.
3. If $S(0, \boldsymbol{i})$ is in the chart, accept; otherwise reject.

The trace of this recognizer on input $\mathtt{a}_1\mathtt{a}_2\mathtt{a}_3\mathtt{a}_4$ is as follows, where the generated facts are listed in the order they enter the agenda (assuming that the agenda is first-in first-out), together with the type of inference, rule, and premises used to derive them:

1.	$m\_S(0)$	INITIALIZE	
2.	$m\_P(0)$	PREDICT, $r_1, 1$	
3.	$sup_{2.1}(0,1)$	SCAN, $r_6, 2, \mathtt{a}_1(0,1)$	
4.	$sup_{3.1}(0,1)$	SCAN, $r_9, 2, \mathtt{a}_1(0,1)$	
5.	$m\_P(1)$	PREDICT, $r_3, 4$	
6.	$sup_{2.2}(0,2)$	SCAN, $r_7, 3, \mathtt{a}_2(1,2)$	
7.	$sup_{2.3}(0,2,2,3)$	SCAN, $r_8, 6, \mathtt{a}_3(2,3)$	!!
8.	$P(0,2,2,4)$	SCAN, $r_{21}, 7, \mathtt{a}_4(3,4)$	
9.	$sup_{1.1}(0,2,2,4)$	COMPLETE, $r_5, 1, 8$	
10.	$m\_Q(2,2,4)$	PREDICT, $r_2, 9$	

[4] A recognizer is said to have the *correct prefix property* or to be *prefix-correct* if it processes the input string from left to right and rejects as soon as the portion of the input that has been processed so far is not a prefix of any element of the language.

r_1: $m\_P(i) :- m\_S(i).$
r_2: $m\_Q(j,k,l) :- sup_{1.1}(i,j,k,l).$
r_3: $m\_P(j) :- sup_{3.1}(i,j).$
r_4: $m\_Q(j,k,n) :- sup_{5.3}(i,j,k,l,m,n).$

r_5: $sup_{1.1}(i,j,k,l) :- m\_S(i),\ P(i,j,k,l).$
r_6: $sup_{2.1}(i,j) :- m\_P(i),\ \mathtt{a}_1(i,j).$
r_7: $sup_{2.2}(i,k) :- sup_{2.1}(i,j),\ \mathtt{a}_2(j,k).$
r_8: $sup_{2.3}(i,k,l,m) :- sup_{2.2}(i,k),\ \mathtt{a}_3(l,m).$
r_9: $sup_{3.1}(i,j) :- m\_P(i),\ \mathtt{a}_1(i,j).$
r_{10}: $sup_{3.2}(i,k,n,o) :- sup_{3.1}(i,j),\ P(j,k,n,o).$
r_{11}: $sup_{3.3}(i,l,n,o) :- sup_{3.2}(i,k,n,o),\ \mathtt{a}_2(k,l).$
r_{12}: $sup_{3.4}(i,l,m,o) :- sup_{3.3}(i,l,n,o),\ \mathtt{a}_3(m,n).$
r_{13}: $sup_{4.1}(i,j,k,l) :- m\_Q(i,k,l),\ \mathtt{b}_1(i,j).$
r_{14}: $sup_{4.2}(i,k,l) :- sup_{4.1}(i,j,k,l),\ \mathtt{b}_2(j,k).$
r_{15}: $sup_{4.3}(i,k,l,m) :- sup_{4.2}(i,k,l),\ \mathtt{b}_3(l,m).$
r_{16}: $sup_{5.1}(i,j,l,m) :- m\_Q(i,l,m),\ \mathtt{b}_1(i,j).$
r_{17}: $sup_{5.2}(i,j,k,l,m) :- sup_{5.1}(i,j,l,m),\ \mathtt{b}_2(k,l).$
r_{18}: $sup_{5.3}(i,j,k,l,m,n) :- sup_{5.2}(i,j,k,l,m),\ \mathtt{b}_3(m,n).$
r_{19}: $sup_{5.4}(i,l,m,o) :- sup_{5.3}(i,j,k,l,m,n),\ Q(j,k,n,o).$

r_{20}: $S(i,m) :- sup_{1.1}(i,j,k,l),\ Q(j,k,l,m).$
r_{21}: $P(i,k,l,n) :- sup_{2.3}(i,k,l,m),\ \mathtt{a}_4(m,n).$
r_{22}: $P(i,l,m,p) :- sup_{3.4}(i,l,m,o),\ \mathtt{a}_4(o,p).$
r_{23}: $Q(i,k,l,n) :- sup_{4.3}(i,k,l,m),\ \mathtt{b}_4(m,n).$
r_{24}: $Q(i,l,m,p) :- sup_{5.4}(i,l,m,o),\ \mathtt{b}_4(o,p).$

Figure 2: The result of applying generalized supplementary magic-sets rewriting to the program in Figure 1.

Although the input is correctly rejected, the correct-prefix property is violated at line 7. The problem comes from the fact that in rule r_8 of Figure 2, both arguments of $\mathtt{a}_3$ are free (see the adornment on $\mathtt{a}_3$ in rule 2 of Figure 1). This means that $\mathtt{a}_3$ is predicted somewhere, but not necessarily at the current position in the input string. So after $\mathtt{a}_3$ is scanned at position 2, there is no guarantee that the input that has been processed so far is a correct prefix. In fact, the problem is even worse, as this particular recognizer fails to accept any input string. On input $\mathtt{a}_1\mathtt{a}_2\mathtt{b}_1\mathtt{b}_2\mathtt{a}_3\mathtt{a}_4\mathtt{b}_3\mathtt{b}_4$, for example, the recognizer proceeds as above up to line 6, but then rejects the input, since no scan move is possible on $\mathtt{b}_1$.[5]

2 Securing the correct prefix property by adding redundant subgoals

In order to produce a prefix-correct recognition algorithm by magic-sets rewriting, it is necessary to ensure that in the program to be rewritten, the first argument of all extensional predicates is adorned as bound. To achieve this, we need an extra rewriting of the Datalog program corresponding to the given MCFG before applying magic-sets rewriting.

In the Datalog program representing an MCFG, occurrences of variables in the body of a rule come in pairs, with each pair corresponding to an occurrence of a symbol (terminal or string variable) in the head of the corresponding MCFG rule. We will rewrite the Datalog program in such a way that the modified program satisfies the following property:

- The order of (the first occurrences of) the pairs of variables in the body of a rule correspond to the order of the corresponding symbol occurrences in the MCFG rule.

This will make sure that the first arguments of all extensional predicates are adorned as bound.[6]

In order to achieve this, we split each 4-ary intensional predicate $R(i,j,k,l)$ into two predicates, $R_1(i,j)$ and $R(i,j,k,l)$. The predicate $R(i,j,k,l)$ retains its original meaning, while the new predicate $R_1(i,j)$ intuitively means $\exists kl.R(i,j,k,l)$. Where an old rule has $R(i,j,k,l)$ in its right-hand side, the new rule has $R_1(i,j)$ and $R(i,j,k,l)$ in its right-hand side; the positions of $R_1(i,j)$ and $R(i,j,k,l)$ will be dictated by the positions of the symbols corresponding to (i,j) and (k,l) in the MCFG rule. Since $R_1(i,j)$ is derivable whenever $R(i,j,k,l)$ is, this will not alter the least fixpoint semantics of the rule.

For instance, this procedure rewrites the third rule of the original program (2) as follows:

(3) $P(i,l,m,p) :- \mathtt{a}_1(i,j),\ P_1(j,k),\ \mathtt{a}_2(k,l),\ \mathtt{a}_3(m,n),\ P(j,k,n,o),\ \mathtt{a}_4(o,p).$

[5] The problem with the program in Figure 2 is essentially the same as the one that Johnson (1994) discusses in the context of top-down recognition for tree-adjoining grammars, first noted by Lang.

[6] This assumes a normal form for MCFGs characterized by the following condition:

- If $A(t_1,\ldots,t_r) :- B_1(x_{1,1},\ldots,x_{1,r_1}),\ \ldots,\ B_m(x_{m,1},\ldots,x_{m,r_m})$ is a rule, then $t_1\ldots t_r \in (\Sigma\cup X)^* x_{i,j} (\Sigma\cup X)^* x_{i,k} (\Sigma\cup X)^*$ implies $j<k$, where $X=\{\,x_{i,j} \mid 1\le i\le m, 1\le j\le r_i\,\}$.

This normal form corresponds to what Villemonte de la Clergerie (2002a; 2002b) called *ordered simple RCG*.

$S(i,m) :- P_1(i,j),\ Q_1(j,k),\ P(i,j,k,l),\ Q(j,k,l,m).$
$P_1(i,k) :- aux_2(i,k).$
$P(i,k,l,n) :- aux_2(i,k),\ \mathsf{a}_3(l,m),\ \mathsf{a}_4(m,n).$
$aux_2(i,k) :- \mathsf{a}_1(i,j),\ \mathsf{a}_2(j,k).$
$P_1(i,l) :- aux_3(i,j,k,l).$
$P(i,l,m,p) :- aux_3(i,j,k,l),\ \mathsf{a}_3(m,n),\ P(j,k,n,o),\ \mathsf{a}_4(o,p).$
$aux_3(i,j,k,l) :- \mathsf{a}_1(i,j),\ P_1(j,k),\ \mathsf{a}_2(k,l).$
$Q_1(i,k) :- aux_4(i,k).$
$Q(i,k,l,n) :- aux_4(i,k),\ \mathsf{b}_3(l,m),\ \mathsf{b}_4(m,n).$
$aux_4(i,k) :- \mathsf{b}_1(i,j),\ \mathsf{b}_2(j,k).$
$Q_1(i,l) :- aux_5(i,j,k,l).$
$Q(i,l,m,p) :- aux_5(i,j,k,l),\ \mathsf{b}_3(m,n),\ Q(j,k,n,o),\ \mathsf{b}_4(o,p).$
$aux_5(i,j,k,l) :- \mathsf{b}_1(i,j),\ Q_1(j,k),\ \mathsf{b}_2(k,l).$

Figure 3: Rewritten Datalog program.

Note the correspondence with the MCFG rule:

(4) $P(\underset{i}{\mathsf{a}_1}\ \underset{j}{x_1}\ \underset{k}{\mathsf{a}_2}\ \underset{l}{\ ,\ }\ \underset{m}{\mathsf{a}_3}\ \underset{n}{x_2}\ \underset{o}{\mathsf{a}_4}\ \underset{p}{\ }) :- P(x_1,x_2).$

A rule for P_1 is obtained from (3) by discarding the last three subgoals, which pertain to string positions in the second argument of the head of (4):

(5) $P_1(i,l) :- \mathsf{a}_1(i,j),\ P_1(j,k),\ \mathsf{a}_2(k,l).$

We then fold the common part of (5) and (3), creating an auxiliary predicate aux_3 (the subscript indicates the rule number from the original program):

$$P_1(i,l) :- aux_3(i,j,k,l).$$
$$P(i,l,m,p) :- aux_3(i,j,k,l),\ \mathsf{a}_3(m,n),\ P(j,k,n,o),\ \mathsf{a}_4(o,p).$$
$$aux_3(i,j,k,l) :- \mathsf{a}_1(i,j),\ P_1(j,k),\ \mathsf{a}_2(k,l).$$

The above description is for 2-MCFGs, but the procedure is applicable to the Datalog program representing any MCFG. In the general case, a $2m$-ary predicate $R(i_1,\ldots,i_{2m})$ is split into m predicates, $R_1(i_1,i_2),\ldots,R_{m-1}(i_1,\ldots,i_{2m-2})$, $R(i_1,\ldots,i_{2m})$, and $m-1$ auxiliary predicates are introduced, one for each R_i. We call this rewriting procedure *redundancy introduction*.

The result of applying redundancy introduction to the program (2) is in Figure 3. Figure 4 shows the adornments. Note that the adornments on the boldface predicate occurrences in Figure 4 are adjusted to "less bound" patterns in order to satisfy the *unique binding property*. This is justified by

1: $S^{bf}(i,m) :- P_1^{bf}(i,j),\ Q_1^{bf}(j,k),\ P^{bbbf}(i,j,k,l),\ Q^{bbbf}(j,k,l,m).$
2: $P_1^{bf}(i,k) :- aux_2^{bf}(i,k).$
3: $P^{bbbf}(i,k,l,n) :- \boldsymbol{aux_2^{bf}}(i,k),\ \mathsf{a}_3^{bf}(l,m),\ \mathsf{a}_4^{bf}(m,n).$
4: $aux_2^{bf}(i,k) :- \mathsf{a}_1^{bf}(i,j),\ \mathsf{a}_2^{bf}(j,k).$
5: $P_1^{bf}(i,l) :- aux_3^{bfff}(i,j,k,l).$
6: $P^{bbbf}(i,l,m,p) :- \boldsymbol{aux_3^{bfff}}(i,j,k,l),\ \mathsf{a}_3^{bf}(m,n),\ P^{bfff}(j,k,n,o),\ \mathsf{a}_4^{bf}(o,p).$
7: $aux_3^{bfff}(i,j,k,l) :- \mathsf{a}_1^{bf}(i,j),\ P_1^{bf}(j,k),\ \mathsf{a}_2^{bf}(k,l).$
8: $Q_1^{bf}(i,k) :- aux_4^{bf}(i,k).$
9: $Q^{bbbf}(i,k,l,n) :- \boldsymbol{aux_4^{bf}}(i,k),\ \mathsf{b}_3^{bf}(l,m),\ \mathsf{b}_4^{bf}(m,n).$
10: $aux_4^{bf}(i,k) :- \mathsf{b}_1^{bf}(i,j),\ \mathsf{b}_2^{bf}(j,k).$
11: $Q_1^{bf}(i,l) :- aux_5^{bfff}(i,j,k,l).$
12: $Q^{bbbf}(i,l,m,p) :- \boldsymbol{aux_5^{bfff}}(i,j,k,l),\ \mathsf{b}_3^{bf}(m,n),\ Q^{bbbf}(j,k,n,o),\ \mathsf{b}_4^{bf}(o,p).$
13: $aux_5^{bfff}(i,j,k,l) :- \mathsf{b}_1^{bf}(i,j),\ Q_1^{bf}(j,k),\ \mathsf{b}_2^{bf}(k,l).$

Figure 4: Adorned version of the program in Figure 3.

viewing, for instance, $\boldsymbol{aux_2^{bf}}(i,k)$ as an abbreviation for $aux_2^{bf}(i,k')$, $k' =^{bb} k$.[7] Generalized supplementary magic-sets rewriting applied to Figure 4 results in Figure 5.

The following shows the trace of running the chart recognizer using the program in Figure 5 on input $\mathsf{a}_1\mathsf{a}_2\mathsf{a}_3\mathsf{a}_4$:

1.	$m\_S(0)$	INITIALIZE
2.	$m\_P_1(0)$	PREDICT, $r_1, 1$
3.	$m\_aux_2(0)$	PREDICT, $r_5, 2$
4.	$m\_aux_3(0)$	PREDICT, $r_7, 2$
5.	$sup_{4.1}(0,1)$	SCAN, $r_{22}, 3, \mathsf{a}_1(0,1)$
6.	$sup_{7.1}(0,1)$	SCAN, $r_{26}, 4, \mathsf{a}_1(0,1)$
7.	$m\_P_1(1)$	PREDICT, $r_{10}, 6$
8.	$m\_aux_2(1)$	PREDICT, $r_5, 7$
9.	$m\_aux_3(1)$	PREDICT, $r_7, 7$
10.	$aux_2(0,2)$	SCAN, $r_{39}, 5, \mathsf{a}_2(1,2)$
11.	$P_1(0,2)$	COMPLETE, $r_{37}, 2, 10$
12.	$sup_{1.1}(0,2)$	COMPLETE, $r_{17}, 1, 11$
13.	$m\_Q_1(2)$	PREDICT, $r_2, 12$
14.	$m\_aux_4(2)$	PREDICT, $r_{11}, 13$
15.	$m\_aux_5(2)$	PREDICT, $r_{13}, 13$

The algorithm correctly rejects the input without making any SCAN moves on a_3. If the input is

[7] For the sake of simplicity, we defer explicit use of equality until Section 4.

r_1: $m\_P_1(i) :- m\_S(i).$
r_2: $m\_Q_1(j) :- sup_{1.1}(i, j).$
r_3: $m\_P(i, j, k) :- sup_{1.2}(i, j, k).$
r_4: $m\_Q(j, k, l) :- sup_{1.3}(i, j, k, l).$
r_5: $m\_aux_2(i) :- m\_P_1(i).$
r_6: $m\_aux_2(i) :- m\_P(i, k, l).$
r_7: $m\_aux_3(i) :- m\_P_1(i).$
r_8: $m\_aux_3(i) :- m\_P(i, l, m).$
r_9: $m\_P(j, k, n) :- sup_{6.2}(i, j, k, l, m, n).$
r_{10}: $m\_P_1(j) :- sup_{7.1}(i, j).$
r_{11}: $m\_aux_4(i) :- m\_Q_1(i).$
r_{12}: $m\_aux_4(i) :- m\_Q(i, k, l).$
r_{13}: $m\_aux_5(i) :- m\_Q_1(i).$
r_{14}: $m\_aux_5(i) :- m\_Q(i, l, m).$
r_{15}: $m\_Q(j, k, n) :- sup_{12.2}(i, j, k, l, m, n).$
r_{16}: $m\_Q_1(j) :- sup_{13.1}(i, j).$

r_{17}: $sup_{1.1}(i, j) :- m\_S(i),\ P_1(i, j).$
r_{18}: $sup_{1.2}(i, j, k) :- sup_{1.1}(i, j),\ Q_1(j, k).$
r_{19}: $sup_{1.3}(i, j, k, l) :- sup_{1.2}(i, j, k),\ P(i, j, k, l).$
r_{20}: $sup_{3.1}(i, k, l) :- m\_P(i, k, l),\ aux_2(i, k).$
r_{21}: $sup_{3.2}(i, k, l, m) :- sup_{3.1}(i, k, l),\ \mathsf{a}_3(l, m).$
r_{22}: $sup_{4.1}(i, j) :- m\_aux_2(i),\ \mathsf{a}_1(i, j).$
r_{23}: $sup_{6.1}(i, j, k, l, m) :- m\_P(i, l, m),\ aux_3(i, j, k, l).$
r_{24}: $sup_{6.2}(i, j, k, l, m, n) :- sup_{6.1}(i, j, k, l, m),\ \mathsf{a}_3(m, n).$
r_{25}: $sup_{6.3}(i, l, m, o) :- sup_{6.2}(i, j, k, l, m, n),\ P(j, k, n, o).$
r_{26}: $sup_{7.1}(i, j) :- m\_aux_3(i),\ \mathsf{a}_1(i, j).$
r_{27}: $sup_{7.2}(i, j, k) :- sup_{7.1}(i, j),\ P_1(j, k).$
r_{28}: $sup_{9.1}(i, k, l) :- m\_Q(i, k, l),\ aux_4(i, k).$
r_{29}: $sup_{9.2}(i, k, l, m) :- sup_{9.1}(i, k, l),\ \mathsf{b}_3(l, m).$
r_{30}: $sup_{10.1}(i, j) :- m\_aux_4(i),\ \mathsf{b}_1(i, j).$
r_{31}: $sup_{12.1}(i, j, k, l, m) :- m\_Q(i, l, m),\ aux_5(i, j, k, l).$
r_{32}: $sup_{12.2}(i, j, k, l, m, n) :- sup_{12.1}(i, j, k, l, m),\ \mathsf{b}_3(m, n).$
r_{33}: $sup_{12.3}(i, l, m, o) :- sup_{12.2}(i, j, k, l, m, n),\ Q(j, k, n, o).$
r_{34}: $sup_{13.1}(i, j) :- m\_aux_5(i),\ \mathsf{b}_1(i, j).$
r_{35}: $sup_{13.2}(i, j, k) :- sup_{13.1}(i, j),\ Q(j, k).$

r_{36}: $S(i, m) :- sup_{1.3}(i, j, k, l),\ Q(j, k, l, m).$
r_{37}: $P_1(i, k) :- m\_P_1(i),\ aux_2(i, k).$
r_{38}: $P(i, k, l, n) :- sup_{3.2}(i, k, l, m), \mathsf{a}_4(m, n).$
r_{39}: $aux_2(i, k) :- sup_{4.1}(i, j),\ \mathsf{a}_2(j, k).$
r_{40}: $P_1(i, l) :- m\_P_1(i),\ aux_3(i, j, k, l).$
r_{41}: $P(i, l, m, p) :- sup_{6.3}(i, l, m, o), \mathsf{a}_4(o, p).$
r_{42}: $aux_3(i, j, k, l) :- sup_{7.2}(i, j, k), \mathsf{a}_2(k, l).$
r_{43}: $Q_1(i, k) :- m\_Q_1(i), aux_4(i, k).$
r_{44}: $Q(i, k, l, n) :- sup_{9.2}(i, k, l, m), \mathsf{b}_4(m, n).$
r_{45}: $aux_4(i, k) :- sup_{10.1}(i, j), \mathsf{b}_2(j, k).$
r_{46}: $Q_1(i, l) :- m\_Q(i), aux_5(i, j, k, l).$
r_{47}: $Q(i, l, m, p) :- sup_{12.3}(i, l, m, o), \mathsf{b}_4(o, p).$
r_{48}: $aux_5(i, j, k, l) :- sup_{13.2}(i, j, k), \mathsf{b}_2(k, l).$

Figure 5: The result of applying generalized supplementary magic-sets rewriting to the program in Figure 4.

$\mathsf{a}_1\mathsf{a}_2\mathsf{b}_1\mathsf{b}_2\mathsf{a}_3\mathsf{a}_4\mathsf{b}_3\mathsf{b}_4$ instead, the execution of the algorithm continues as follows:

16.	$sup_{10.1}(2, 3)$	SCAN, r_{30}, 14, $\mathsf{b}_1(2, 3)$
17.	$sup_{13.1}(2, 3)$	SCAN, r_{34}, 15, $\mathsf{b}_1(2, 3)$
18.	$m\_Q_1(3)$	PREDICT, r_{16}, 17
19.	$m\_aux_4(3)$	PREDICT, r_{11}, 18
20.	$m\_aux_5(3)$	PREDICT, r_{13}, 18
21.	$aux_4(2, 4)$	SCAN, r_{45}, 16, $\mathsf{b}_2(3, 4)$
22.	$Q_1(2, 4)$	COMPLETE, r_{43}, 13, 21
23.	$sup_{1.2}(0, 2, 4)$	COMPLETE, r_{18}, 12, 22
24.	$m\_P(0, 2, 4)$	PREDICT, r_3, 23
25.	$sup_{3.1}(0, 2, 4)$	COMPLETE, r_{20}, 24, 10
26.	$sup_{3.2}(0, 2, 4, 5)$	SCAN, r_{21}, 25, $\mathsf{a}_3(4, 5)$
27.	$P(0, 2, 4, 6)$	SCAN, r_{38}, 26, $\mathsf{a}_4(5, 6)$
28.	$sup_{1.3}(0, 2, 4, 6)$	COMPLETE, r_{19}, 23, 27
29.	$m\_Q(2, 4, 6)$	PREDICT, r_4, 28
30.	$sup_{9.1}(2, 4, 6)$	COMPLETE, r_{28}, 29, 21
31.	$sup_{9.2}(2, 4, 6, 7)$	SCAN, r_{29}, 30, $\mathsf{b}_3(6, 7)$
32.	$Q(2, 4, 6, 8)$	SCAN, r_{44}, 31, $\mathsf{b}_4(7, 8)$
33.	$S(0, 8)$	COMPLETE, r_{36}, 28, 32

Needless to say, the program that we obtain with our method has room for optimization. For instance, rules of the form $m\_aux(i) :- R(i, j, k)$ are useless in the presence of $m\_aux(i) :- R_1(i)$, so they can be safely removed. Nevertheless, the recognizer produced by our method is always correct and satisfies the correct prefix property without any such fine-tuning.

3 Correctness of the method

It is easy to prove that the Datalog program $\mathbf{P}$ that we obtain from an MCFG G after redundancy introduction and magic-sets rewriting is correct in the sense that for any string $a_1 \dots a_n$,

$$\mathbf{P} \cup \{ m\_S(0), a_1(0, 1), \dots, a_n(\boldsymbol{n-1}, \boldsymbol{n}) \} \vdash S(0, \boldsymbol{n}) \quad \text{iff} \quad a_1 \dots a_n \in L(G).$$

Since the initial Datalog program is correct (in the sense of the above biconditional with $m\_S(0)$

omitted) and magic-sets rewriting preserves correctness (modulo $m\_S(0)$), it suffices to prove that the redundancy introduction transformation preserves correctness.

Let $\mathbf{P}$ be a Datalog program representing a 2-MCFG and let $\mathbf{P}'$ be the result of applying redundancy introduction to $\mathbf{P}$. Let R be any 4-ary intensional predicate in $\mathbf{P}$, D be an extensional database, and c, d, e, f be constants in D. It is easy to see that

$$\mathbf{P}' \cup D \vdash R(c,d,e,f) \quad \text{implies} \quad \mathbf{P}' \cup D \vdash R_1(c,d),$$

and using this, we can prove by straightforward induction that

$$\mathbf{P} \cup D \vdash R(c,d,e,f) \quad \text{iff} \quad \mathbf{P}' \cup D \vdash R(c,d,e,f).$$

The general case of m-MCFGs is similar.

It is also easy to see that our control algorithm is complete with respect to any Datalog program $\mathbf{P}$ obtained from an MCFG by redundancy introduction and magic-sets rewriting. Observing that all facts derivable from $\mathbf{P}$ together with $\{m\_S(0), a_1(0,1), \ldots, a_n(\boldsymbol{n-1}, \boldsymbol{n})\}$ have the form $R(\boldsymbol{i_1}, \ldots, \boldsymbol{i_m})$ where $i_1 \leq \cdots \leq i_m$, and rules involving an extensional predicate a all have the form $P(\ldots, j) :- R(\ldots, i), a(i, j)$, we can prove by induction that our control algorithm generates all derivable facts $R(\boldsymbol{i_1}, \ldots, \boldsymbol{i_m})$ before making any scan moves on a_{i_m+1}.

It remains to show the correct prefix property. We call an MCFG *reduced* if every nonterminal denotes a non-empty relation on strings. Let $\mathbf{P}$ be the Datalog program obtained by applying redundancy introduction to a program representing a reduced 2-MCFG. By the correspondence between magic-sets rewriting and *SLD-resolution* (Brass, 1995), it suffices to show that SLD-resolution (with the leftmost selection function) using program $\mathbf{P}$ has a property which corresponds to prefix-correctness.

Let Γ and Δ denote negative clauses, and let $\square$ denote the empty clause. We write

$$\Gamma \overset{\mathbf{P},D}{\Longrightarrow} \Delta$$

to mean that there exists an SLD-derivation starting from goal Γ and ending in goal Δ, using rules in $\mathbf{P}$ and *exactly* the facts in D as input clauses. We call D a *string database* if D is isomorphic to $\{a_1(0,1), \ldots, a_n(\boldsymbol{n-1}, \boldsymbol{n})\}$. It is easy to see that if $S(0,x) \overset{\mathbf{P},D}{\Longrightarrow} \Gamma$, then D is a string database.

Theorem 1. *If* $S(0,x) \overset{\mathbf{P},D}{\Longrightarrow} \Gamma$, *then* $\Gamma \overset{\mathbf{P},D'}{\Longrightarrow} \square$ *for some* D' *such that* $D \cup D'$ *is a string database.*

This is the desired property corresponding to prefix-correctness. The theorem can be proved with the help of the following lemma.

Lemma 2. *Let* R *be a 4-ary intensional predicate in* $\mathbf{P}$. *For every string database* D *and constants* c, d *in* D *such that* $\mathbf{P} \cup D \vdash R_1(c,d)$, *if* e *is a constant not in* D, *there exists a string database* D' *whose constants are disjoint from those of* D *such that* $\mathbf{P} \cup D \cup D' \vdash R(c,d,e,f)$ *for some constant* f *in* D'.

Again, the general case of m-MCFGs can be treated similarly.

4 Application to tree-adjoining grammars

So far, we have implicitly assumed that the empty string ϵ does not appear as an argument in the head of MCFG rules. Since ϵ can be eliminated from any MCFG generating an ϵ-free language (Seki et al., 1991), this is not an essential restriction, but it is often convenient to be able to handle rules involving ϵ directly, as is the case with 2-MCFGs representing TAGs. To translate an MCFG rule with ϵ into a Datalog rule, we use range-restricted equality as an extensional predicate. For example, a 2-MCFG rule

$$A(x, \epsilon) :- B(x).$$

is translated into Datalog as follows:[8]

$$A(i,j,k,l) :- B(i,j), k = l.$$

Rewritten Datalog programs will now involve equality. We continue to represent the input string $a_1 \ldots a_n$ as an extensional database $\{a_1(0,1), \ldots, a_n(\boldsymbol{n-1}, \boldsymbol{n})\}$, but modify our control algorithm slightly:

Chart recognizer control algorithm (revised) Same as before, except for the following two steps:

1. (INITIALIZE) Initialize the chart to the empty set, the agenda to $\{m\_S(0), 0 = 0\}$, and i to 0.
2. (c) iii. If the agenda is empty, reject the input; otherwise increment i, and then add the fact $\boldsymbol{i} = \boldsymbol{i}$ to the agenda.

[8] Since equality is treated as an extensional predicate, rules like this are *safe* in the sense that they can derive ground facts only.

To obtain an $O(n^6)$ prefix-correct Earley recognizer for TAGs by our method, we translate each nonterminal node of an elementary tree into a 2-MCFG rule. For each such node M, the 2-MCFG has a distinct nonterminal symbol M, whose arity is either 2 or 1 depending on whether the node M dominates the foot node or not.

Let M be a node dominating a foot node having children $L^1, \ldots, L^j, N, R^1, \ldots, R^k$, of which N is the child on the path to the foot node. For each elementary tree γ with root node T that can adjoin into M, the 2-MCFG has the rule

(6) $M(w_1 x_1 \ldots x_j z_1, z_2 y_1 \ldots y_k w_2) :-$
$T(w_1, w_2),\ L^1(x_1), \ldots, L^j(x_j),\ N(z_1, z_2),$
$R^1(y_1), \ldots, R^k(y_k).$

If adjunction is optional at M, the 2-MCFG also has the rule

$M(x_1 \ldots x_j z_1, z_2 y_1 \ldots y_k) :-$
$L^1(x_1), \ldots, L^j(x_j),\ N(z_1, z_2),\ R^1(y_1), \ldots, R^k(y_k).$

Let F be a foot node. For each elementary tree γ with root node T that can adjoin into F, the 2-MCFG has the rule

$F(w_1, w_2) :- T(w_1, w_2).$

If adjunction is optional at F, the 2-MCFG also has the rule

$F(\epsilon, \epsilon).$

We omit the other cases, but they are all straightforward.

The translation into Datalog of the 2-MCFG thus obtained results in a variant of Lang's Horn clause axiomatization of TAGs (discussed by Johnson (1994)). For example, consider (6) with $j = k = 2$:

(7) $M(w_1 x_1 x_2 z_1, z_2 y_1 y_2 w_2) :- T(w_1, w_2),\ L^1(x_1),$
$L^2(x_2),\ N(z_1, z_2),\ R^1(y_1),\ R^2(y_2).$

The Datalog representation of (7) is the following:

(8) $M(i, m, n, r) :- T(i, j, q, r),\ L^1(j, k),\ L^2(k, l),$
$N(l, m, n, o),\ R^1(o, p),\ R^2(p, q).$

Redundancy introduction rewrites (8) into three rules:

(9) $M_1(i, m) :- aux(i, j, l, m).$
$M(i, m, n, r) :- aux(i, j, l, m),\ N(l, m, n, o),$
$R^1(o, p),\ R^2(p, q),\ T(i, j, q, r).$
$aux(i, j, l, m) :- T_1(i, j),\ L^1(j, k),\ L^2(k, l),$
$N_1(l, m).$

$m\_aux(i) :- m\_M_1(i).$
$m\_aux(i) :- m\_M(i, m, n).$
$m\_N(l, m, n) :- sup_{2.1}(i, j, l, m, n).$
$m\_R^1(o) :- sup_{2.2}(i, j, m, n, o).$
$m\_R^2(p) :- sup_{2.3}(i, j, m, n, p).$
$m\_T(i, j, q) :- sup_{2.4}(i, j, m, n, q).$
$m\_T_1(i) :- m\_aux(i).$
$m\_L^1(j) :- sup_{3.1}(i, j).$
$m\_L^2(k) :- sup_{3.2}(i, j, k).$
$m\_N_1(l) :- sup_{3.3}(i, j, l).$

$sup_{2.1}(i, j, l, m, n) :- m\_M(i, m, n),\ aux(i, j, l, m).$
$sup_{2.2}(i, j, m, n, o) :- sup_{2.1}(i, j, l, m, n),\ N(l, m, n, o).$
$sup_{2.3}(i, j, m, n, p) :- sup_{2.2}(i, j, m, n, o),\ R^1(o, p).$
$sup_{2.4}(i, j, m, n, q) :- sup_{2.3}(i, j, m, n, p),\ R^2(p, q).$
$sup_{3.1}(i, j) :- m\_aux(i),\ T_1(i, j).$
$sup_{3.2}(i, j, k) :- sup_{3.1}(i, j),\ L^1(j, k).$
$sup_{3.3}(i, j, l) :- sup_{3.2}(i, j, k),\ L^2(k, l).$

$M_1(i, m) :- m\_M_1(i),\ aux(i, j, l, m).$
$M(i, m, n, r) :- sup_{2.4}(i, j, m, n, q),\ T(i, j, q, r).$
$aux(i, j, l, m) :- sup_{3.3}(i, j, l),\ N_1(l, m).$

Figure 6: The result of applying generalized supplementary magic-sets rewriting to the three rules in (9).

Finally, the generalized supplementary magic-sets rewriting yields the rules in Figure 6. Each rule in Figure 6 involves at most 6 variables, while the arity of predicates is at most 5. It is easy to see that this holds in general; it follows that the time and space complexity of the recognizer for TAGs produced by our method is $O(n^6)$ and $O(n^5)$, respectively.

5 Comparison with previous approaches

Prefix-correct Earley-like recognizers for MCFGs have been presented before (Matsumura et al., 1989; Harkema, 2001; Albro, 2002; Villemonte de la Clergerie, 2002a; Villemonte de la Clergerie, 2002b). The recognizer obtained by our method seems to be slightly different from each of them, but the main advantage of our approach lies not in the resulting recognizer, but in *how* it is obtained. Unlike previous approaches, we borrow a well-known and well-understood technique from deductive database theory, namely magic-sets rewriting, to automatically *derive* an Earley-style recognizer. Since the parsing schema for Earley's algorithm can be regarded as a special case of generalized supplementary magic-sets rewriting, there

is a precise sense in which our recognizer may be called an *Earley* recognizer. We have used an ad hoc but simple and easy-to-understand rewriting (redundancy introduction) to secure the correct prefix property, and it is the only step in our approach specifically tailor-made for MCFGs.

The application of our method to TAGs in turn uses a completely straightforward encoding of TAGs into Datalog programs (via 2-MCFGs), which is close to Lang's Horn clause axiomatization. (Lang's encoding itself can be used to the same effect.) The resulting recognizer for TAGs is prefix-correct and runs in time $O(n^6)$ and space $O(n^5)$, which is the same as the best known bound for prefix-correct recognizers for TAGs (Nederhof, 1999). The behavior of our recognizer on Nederhof's (1999) example roughly corresponds to that of Nederhof's recognizer, but there is a significant difference between the two in the indices involved in some of the items. More importantly, unlike Nederhof's, our recognizer is a special case of a more general construction, and the time and space complexity bounds are obtained without any fine-tuning.

Since it involves very little non-standard technique, we believe that our method is easier to understand and easier to prove correct than previous approaches. For this reason, we also hope that this work serves useful pedagogical purposes.

References

Albro, Daniel M. 2002. An Earley-style parser for multiple context-free grammars. Unpublished manuscript, UCLA.

Arikawa, Setsuo, Takeshi Shinohara, and Akihiro Yamamoto. 1992. Learning elementary formal systems. *Theoretical Computer Science*, 95(1):97–113.

Beeri, Catriel and Raghu Ramakrishnan. 1991. On the power of magic. *Journal of Logic Programming*, 10(3–4):255–299.

Brass, Stefan. 1995. Magic sets vs. SLD-resolution. In Eder, J. and L. A. Kalinichenko, editors, *Advances in Databases and Information Systems*, pages 185–203. Springer, Berlin.

Groenink, Annius. 1997. *Surface without Structure*. Ph.D. thesis, Utrecht University.

Harkema, Hendrik. 2001. *Parsing minimalist languages*. Ph.D. thesis, UCLA.

Johnson, Mark. 1994. Logical embedded push-down automata in tree-adjoining grammar parsing. *Computational Intelligence*, 10(4):495–505.

Kanazawa, Makoto. 2007. Parsing and generation as Datalog queries. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, pages 176–183.

Matsumura, Takashi, Hiroyuki Seki, Mamoru Fujii, and Tadao Kasami. 1989. Some results on multiple context-free grammars. *IEICE Technical Report*, COMP 88–78:17–26. In Japanese.

Nederhof, Mark-Jan. 1999. The computational complexity of the correct-prefix property for TAGs. *Computational Linguistics*, 25(3):345–360.

Seki, Hiroyuki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229.

Shieber, Stuart M., Yves Schabes, and Fernando C. N. Pereira. 1995. Principles and implementations of deductive parsing. *Journal of Logic Programming*, 24(1–2):3–36.

Sikkel, Klaas. 1997. *Parsing Schemata*. Springer, Berlin.

Smullyan, Raymond M. 1961. *Theory of Formal Systems*. Princeton University Press, Princeton, N.J.

Ullman, Jeffrey D. 1989a. Bottom-up beats top-down in Datalog. In *Proceedings of the Eighth ACM Symposium on Principles of Database Systems*, pages 140–149.

Ullman, Jeffrey D. 1989b. *Principles of Database and Knowledge-Base Systems. Volume II: The New Technologies*. Computer Science Press, Rockville, M.D.

Villemonte de la Clergerie, Éric. 2002a. Parsing MCS languages with thread automata. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6)*, pages 101–108.

Villemonte de la Clergerie, Éric. 2002b. Parsing mildly context-sensitive languages with thread automata. In *Proceedings of the 19th International Conference on Computational Linguistics*, pages 1–7.