THE LANE TABLE METHOD OF CONSTRUCTING LR(1) PARSERS

ABSTRACT. The first practical application of the LR algorithm was by [1] for the LALR(1) subset of LR(1) grammars. In [2] an efficient method of producing an LR(1) parser for all LR(1) grammars was described which involves resolving conflicts at states of the LR(0) parsing machine, employing two phases. In Phase 1 the contexts of the productions involved in conflicts are evaluated by a process described there called “lane tracing”. If conflicts cannot be resolved by these means, then in Phase 2 the parts of the machine involved in lane tracing are regenerated, avoiding the combination of states that potentially lead to conflicts. Other works along the same lines include [4, 5]. The criterion employed in [2] for determining whether or not states may be combined was that of weak compatibility, as defined in [3]. In this paper we describe an alternative method for determining whether states can be combined. According to testing by [6] this method requires less computation. It is also more efficient. when extending the method from LR(1) to LR(k) parsing as described in [7] where very large grammars may be used for the purposes of natural language translation. Taken together with Phase 1, this new method of Phase 2 will, as before, produce a conflict-free LR(1) parser for all LR(1) grammars.

With each significant increment in processing speed and storage capacity in the last 30 years, there have been articles in the popular computer press discussing how such developments could be put to use. It does not appear to the correspondents that the efficiency of the software of the day will be significantly affected, and the main uses for the enhanced computer power is usually ascribed to resource intensive applications such as weather forecasting. With current processing speed and storage capacity, we have reached a stage where efficient implementations of the original Knuth LR(1) algorithm [6] can, in an acceptable amount of time, produce parsers of acceptable size for most current languages. It may thus appear that the improvement of the original Knuth algorithm described in this paper, which significantly reduces its space and time requirements, is of less interest. But just as the speed and capacity of computers have risen, so have the complexity and scope of computer languages, from the early versions of Fortran and Basic to the languages that are current today. It seems clear that this process will continue, and that more ambitious computer and language development will occur to take full advantage of whatever speed and capacity is available, including forms that we cannot now predict. Examples may include gigantic grammars employed in combination with other techniques for the purpose of natural language translation, or the combination of voice recognition systems with computer and natural language applications.

After a few definitions we describe the new Phase 2 method for the Lane tracing algorithm [2] by means of an example.

Let the conflicting actions at the state considered be $\pi_1,\pi_2,..,\pi_r$. If a state S contains configurations that for $1 \le i \le r$, generate a set of contexts C_i along a lane leading to π_i, then the **collection of contexts generated** by S is defined as the set $\{(C_i, i)|1 \le i \le r\}$.

Note that if the sets $\{C_i|1 \le i \le r\}$ are not pairwise disjoint, then the grammar is not LR(1).

The **collection of contexts associated** with any state S is initially the collection of contexts it generates.

The **criterion** according to which regenerated states may be combined is as follows. Let $\{S_1,\ldots,S_t\}$ be a set of connected regenerated states, and let the (same) collection of contexts associated with $S_1,\ldots, S_r$ be in each case $\{(A_i, i)|1 \le i \le r\}$. If we now regenerate a state T that is a successor of one of $S_1,\ldots,S_t$, then:

1. if there is an existing copy of state T whose associated collection of contexts is $\{(B_i, i)|1 \le i \le r\}$ and the set of states $\{(A_i \cup B_i)|1 \le i \le r\}$ are pairwise disjoint, then this existing copy of state T is taken as the successor involved, and the collection of contexts associated with $\{S_1,\ldots,S_t,T\}$ is defined to be $\{(A_i \cup B_i, i)|1 \le i \le r\}$.

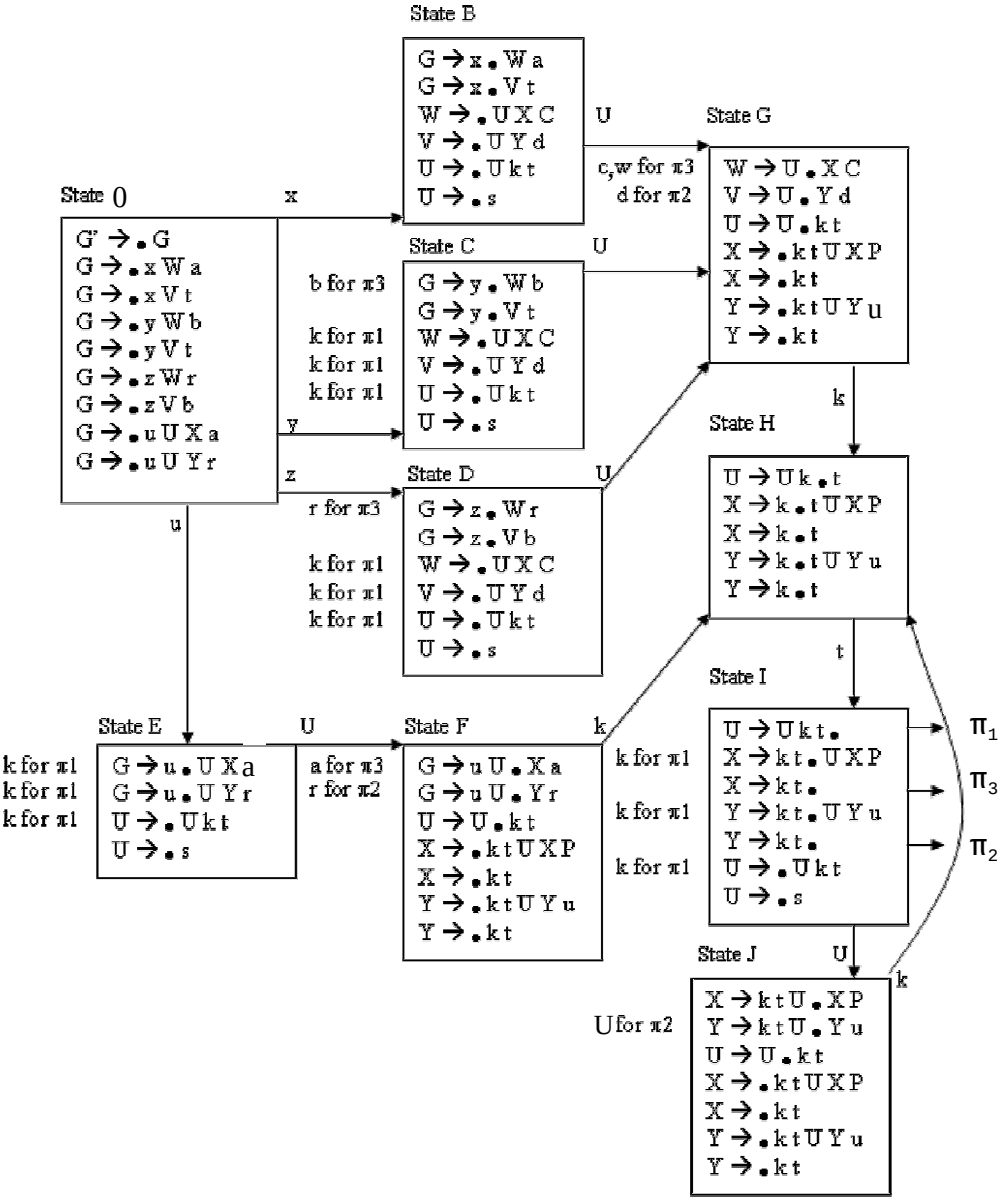
2. otherwise, a new copy T' of T is regenerated as the successor involved, and if the collection of contexts generated by T is $\{(B_i^!, i)|1 \le i \le r\}$, then the collection of contexts associated with $\{S_1, \ldots, S_t,T'\}$ is defined to be $\{(A_i \cup B_i^!, i)|1 \le i \le r\}$. Note that if the sets $\{(A_i \cup B_i^!, i)|1 \le i \le r\}$ are not pairwise disjoint, then the grammar is not LR(1).

Example. Given the following grammar:

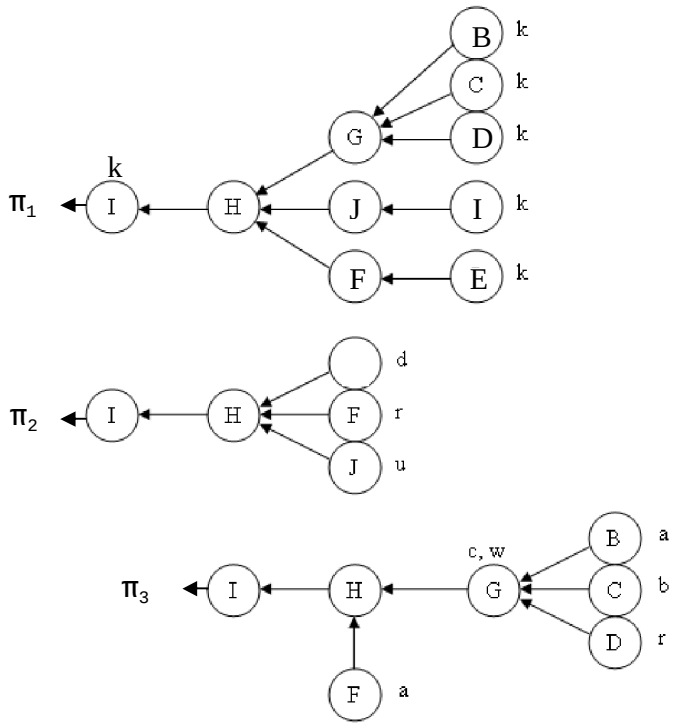
G → xWa | xVt | yWb | yVt | zWr | zVb | uUXa | uUY r
W → UXC
V → UYd
X → ktUXP | kt
Y → ktUYu | kt
U → Ukt | s
E → a | b | c | v
C → c | w
P → z

The part of the LR(0) parsing machine generated for this grammar involved in conflicting lanes (as well as state 0) is shown below. It contains conflicts at state I. At this state π_1, π_2, π_3 are defined as follows:

π_1 is U → UKt
π_2 is Y → kt
π_3 is X → kt


State 0
G' → • G
G → • x W a
G → • x V t
G → • y W b
G → • y V t
G → • z W r
G → • z V b
G → • u U X a
G → • u U Y r
x
y
z
u
State B
G → x • W a
G → x • V t
W → • U X C
V → • U Y d
U → • U k t
U → • s
U
State G
W → U • X C
V → U • Y d
U → U • k t
X → • k t U X P
X → • k t
Y → • k t U Y u
Y → • k t
c,w for π3
d for π2
State C
G → y • W b
G → y • V t
W → • U X C
V → • U Y d
U → • U k t
U → • s
b for π3
k for π1
k for π1
k for π1
State D
G → z • W r
G → z • V b
W → • U X C
V → • U Y d
U → • U k t
U → • s
r for π3
k for π1
k for π1
k for π1
State H
U → U k • t
X → k • t U X P
X → k • t
Y → k • t U Y u
Y → k • t
k
t
State I
U → U k t •
X → k t • U X P
X → k t •
Y → k t • U Y u
Y → k t •
U → • U k t
U → • s
π1
π3
π2
State E
G → u • U X a
G → u • U Y r
U → • U k t
U → • s
k for π1
k for π1
k for π1
State F
G → u U • X a
G → u U • Y r
U → U • k t
X → • k t U X P
X → • k t
Y → • k t U Y u
Y → • k t
a for π3
r for π2
k for π1
k for π1
k for π1
State J
X → k t U • X P
Y → k t U • Y u
U → U • k t
X → • k t U X P
X → • k t
Y → • k t U Y u
Y → • k t
U for π2

Here is a depiction of the states involved in lanes traced and how they are connected to each other. (The lanes are shown in the direction leading to the state with the conflicting actions).



The information collected is stored in a **lane** table as shown:

State	π_1	π_2	π_3	Connected to
B	k		a	*{G}*
C	k		b	*{G}*
D	k		r	*{G}*
E	k			*{F}*
F		r	a	*{H}*
G		d	c, w	*{H}*
H				*{I}*
I	k			*{J}*
J		u		*{H}*

For each of the productions involved in the conflict at a given state (π_1, π_2, π_3 at state I in our example), we regenerate the states involved in lane tracing to that production, starting at the states where lanes in Phase1 lane-tracing ended. Referring to our

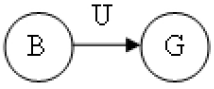
example parsing machine, for π_1 we regenerate B,G,H,I and C,G,H,I and D,G,H,I and E,F,H,I and I,J,H,I. Then we will similarly regenerate the lanes leading to π_2 and π_3. We employ the criterion defined that was defined above for deciding whether copies of the same LR(0) state can be combined or need to be constructed as separate states (so in effect performing state-splitting). For instance in regenerating states B,G,H,I and C,G, H,I we need to determine whether these copies of state G, H, I can be combined, i.e. taken to be the same state.

An example of combining regenerated states is given below. Note that no states other than states B, C,…, J involved in lane tracing are regenerated. For example, the X and Y successors of all the copies of state J are the original X and Y successors of the original state J.

Step 1. Show initially the collection of contexts associated with each state (which at this stage is simply the collection of context generated by the state)[1], as obtained from the Lane table.

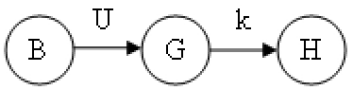
B	(k, 1) (a, 3)	G	(d, 2) (c, w, 3)
C	(k, 1) (b, 3)	H	
D	(k, 1) (r, 3)	I	(k, 1)
E	(k, 1)	J	(u, 2)
F	(r, 2) (a, 3)		

Step 2. Consider the states leading to π_1 starting with B, i.e. BGHI. The collection of contexts associated with B is simply the collection of contexts it generates, i.e. (k, 1) (a, 3) and that associated with G is (d, 2) (c, w, 3). Thus the set of contexts associated with the regenerated connected set of states {B, G} is
(k, 1) (d, 2) (c, w, 3).

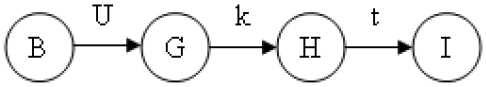


[1] From here on, for the sake of readability, we will omit the curly parentheses, and separating commas in specifying collections of contexts, e.g. instead of ({d},2), ({c,w},3) we will write (d, 2) (c,w,3)

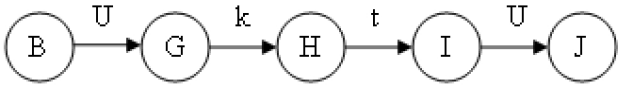
Step 3. Add state H, the successor of state G to the set of regenerated connected states. The collection of contexts associated with {B, G, H} remains as before (k, 1) (d, 2) (c, w, 3).



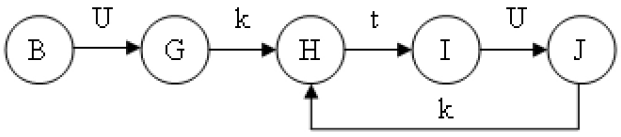
Step 4. Add state I, the successor of state H.
The collection of contexts associated with {B, G, H, I} remains as (k, 1), (d, 2) (c, w, 3).



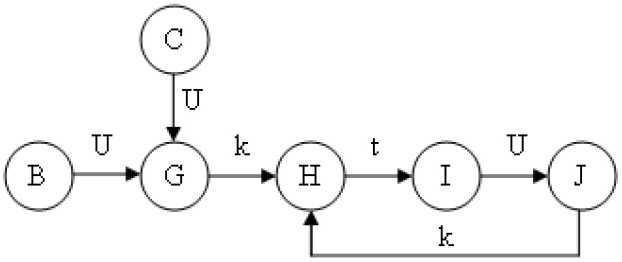
Step 5. Add state J, the successor of state I.
The context sets associated with {B, G, H, I, J} now becomes (k, 1) (d, u, 2) (a, c, w, 3).



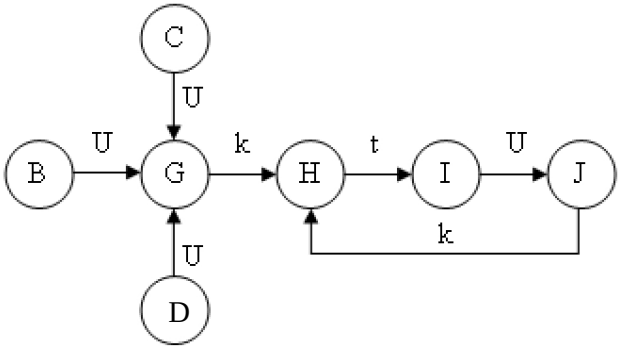
Step 6. Add state H, the successor of state J. State H is already in the set so the context sets associated with {B, G, H, I, J} remains (k, 1) (d, u, 2) (a, c, w, 3).



Step 7. We now consider the states along a lane to π_1 that start with state C, ie. C,G,H,I. The context sets associated (i.e. generated by) C is (k, 1) (a, 3) and that associated with G is (d, 2) (c, w, 3). So the set of contexts associated with {C, G} is (k, 1) (d, 2) (a, c, w, 3). But state G already occurs in the set of regenerated connected states {B, G, H, I, J} considered in Step 6 whose associated set of contexts is (k, 1) (d, u, 2) (a, c, w, 3). We can thus without violating the disjointness between the contexts associated with each of the productions, connect C to {B, G, H, I, J}. In fact the set of contexts associated with {B, C, G, H, I, J} remains as occurred before (k, 1) (d, u, 2) (a, c, w, 3).

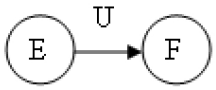


Step 8. We now consider the states leading to π_1 starting at state D. By similar reasoning to that employed in step 7, state D can be added to our set of regenerated connected states. The collection of context sets associated with {B, C, D, G, H, I, J} then becomes (k, 1) (d, u, 2) (a, b, c, r, w, 3).

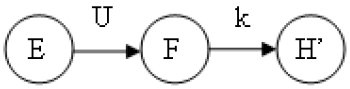


Step 9: We now consider states leading to π_1 starting with E, i.e. EFHI. The set of contexts associated with E is the set it generates, which as shown in the Lane table is (k, 1), while the set of contexts associated with F is (r, 2) (a, 3). So the collection of contexts associated with the connected generated states {E, F} is

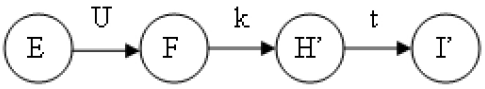
(k,1) (r, 2) (a, 3).



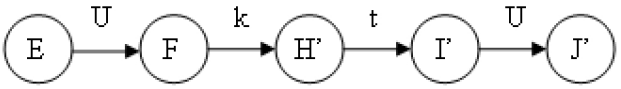
Step 10. Add state H, the successor of state F. Now state H is already in the first set of states. So adding H to the current set of states means we need to consider whether to combine the new set of states with the old one. But the combined contexts would then be (k, 1), (d, r, u, 2), (a, b, c, r, w, 3). This is not pairwise disjoint because "r" is in sets for configurations 2 and 3. So we have to keep the current set separate from the old one, and create a copy H' of state H to insert into the new set. The collection of contexts associated with {E, F, H'} is then (k, 1), (r, 2), (a, 3).



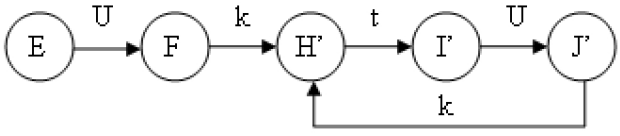
Step 11. Add state I, the successor of state H'. Similarly, we have to create a copy I' of state I to to avoid merging with the old set, which in order to maintain pairwise disjointness of the context sets. The collection of contexts associated with {E, F, H', I'} is then (k, 1), (r, 2), (a, 3).



Step 12. Add state J, the successor of state I'. For the same reason, we need to create a copy J' of state J. The collection of contexts associated with E, F, H', I', J'is then (k, 1), (r, u, 2), (a, 3)

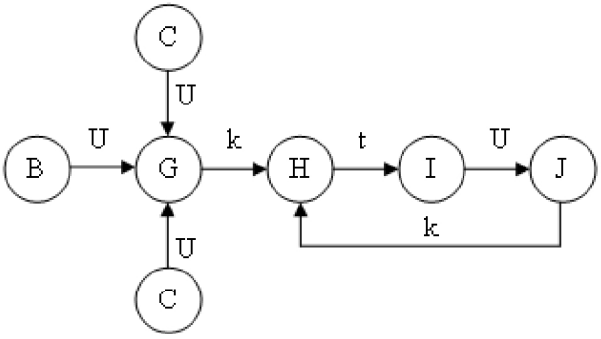


Step 13. Add state H, the successor of state J'. Again, for the same reason, we need to employ a copy H' of H. But there already exists a copy of H in the present set, so we just use that state. The collection of contexts associated with E, F, H', I', J' is then (k, 1), (r, u, 2) (a, 3)



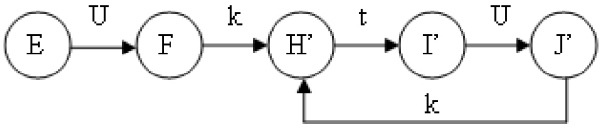
All the states along lanes leading to π_1 and π_2 have now also been generated. The final result of combining regenerated states is then:

Set 1:



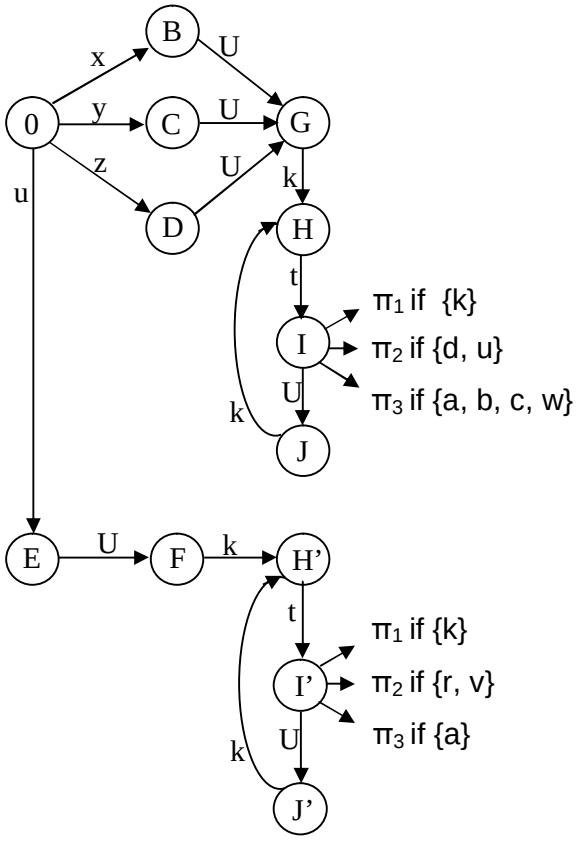
Associated context sets (k, 1) (d, u, 2) (a, b, c, w, 3)

Set 2:



Associated context sets (k, 1) (r, v, 2) (a, 3)

The portion of the parsing machine involved in lane tracing given previously has now been transformed into:



REFERENCES

1. Frank L. DeRemer. Practical translators for LR(k) languages. Ph.D. thesis, MIT, Cambridge, 1969.

2. David Pager. The lane-tracing algorithm for constructing LR(k) parsers and ways of enhancing its efficiency. Information Sciences, 12: 19-42, 1977.

3. David Pager. A practical general method for constructing LR(k) parsers. Acta Informatica, 7: 249-268, 1977.

4. David Spector. Full LR(1) parser generation. ACM SiGPLAN Notices, 58-66, 1981

5. David Spector. Efficient full LR(1) parser generation. ACM SIGPLAN Notices, 23(12), 143-150, 1988.

6. Xin Chen. Measuring and extending LR(1) parser generation. Ph.D thesis, University of Hawaii, 2009. http://sourceforge.net/projects/hyacc/

7. David Pager. Resolving LR type conflicts at compiler or translation time. Tech. Report ICS2009-06-03, ICS Dept., University of Hawaii, June 2009. http://www.ics.hawaii.edu/research/tech-reports