JOURNAL OF COMPUTING TECHNOLOGIES

ISSN 2278 – 3814

Available online at www.jctjournals.com

Volume 1 , Issue 2 , June 2012

Improved LALR (1) Parsing

Mr. Om Singh Parihar[1], Mrs. Dhawal Gupta[2]
[1&2]Computer Science & Engineering,
[1&2]RGPV Bhopal (M.P.), India
[1&2]Gyan Ganga College of Technolgy,Jabalpaur(M.P.),India
[1]*ommsingh@gmail.com*
[2]*guptadhawal12@gmail.com*

Abstract—**LALR (1) is one of most popular bottom up parsing method. Various methods have been given to remove shift-reduce reduce-reduce conflict during parsing. These methods are efficient but detect conflicts in later stage of parsing. In this paper method that is being proposed detects conflicts in earlier stage of parsing so a parsing table that contains conflicts using existing approaches of table construction now can avoid conflicts earlier. Method that is being proposed here concentrates on the way table is constructed for LALR (1) parser. Slight change in LALR (1) table construction can avoid conflicts in better way.**

Keywords— ***LALR (1) parsing, Conflicts, LALR (1) parsing table.***

1. INTRODUCTION

Working with LALR (k) grammars has great advantage that they can be used by parser generators to automatically produce fully efficient and operational parsers, encoded in languages like C, C++, Java, Haskell, etc. Examples of LALR parser generators are CUP [CUP 2007], YACC [Johnson 1979], Frown [Frown 2007], among others. During parsing problems arises while we have two alternatives to apply to determine nest parsing decision and we are not sure which one is correct to choose. This happens due to the recurrent existence of conflicts, i.e., nondeterministic points in the parser. By analyzing Output file created by parser generator conflicts can be removed usually. This output consists of a considerable amount of textual data, from the numerical code associated to grammar symbols to the grammar and LALR automaton itself. Using the Notus language as an example, the Bison parser generator (the GNU version of YACC) dumps a 54 Kb file, containing 6244 words and 2257 lines. The big amount of data and the fact that none of it is interrelated – hyperlinks are not possible in text files, make it very difficult to browse. The level of abstraction in these log files is also a problem, since non experts in LALR parsing may not interpret them accordingly. When facing these difficulties, these users often migrate to LL parser generators. Despite their simplified theory, this approach is not a real advantage, since LL languages are a proper subset of the LALR ones. Even for experts users, removing conflicts in such harsh environment causes a decrease of productivity.

To face this scenario, in this paper we present a methodology for removing conflicts in non LALR(1) grammars. This methodology consists of a set of steps whose intention is to capture conflictsat in earlier stage of LALR(1) generator: we using these steps to accomplish this goal(I) create set of LR(1) items; (ii)create LALR(1) parsing table following rule as merge only those do not cause conflict, state those causes conflcits are not merged. This article is organized as follows: Section 2 gives the necessary background to understand the formulations used in later sections; Section 3 discusses conflicts in LR and LALR parsing; Section 4 presents the proposed methodology.

2. BACKGROUND

Before we present the methodology itself, it is necessary to establish some formal concepts, conventions, definitions and theorems. Most of the subject defined here is merely a reproduction or sometimes a slight variation of what is described in [Charles 1991], [DeRemerandPennello [1982], [Aho and Ullman 1972] and[KristensenandMadsen [1981]. It is assumed that the reader is familiar with LR and LALR parsing. A context free grammar (CFG) is given by G = (N, Σ, P,S) N is a finite set of nonterminals, Σ the finite set of terminals, P the set of rules in G and finally S ∈ N is the start symbol. V = N ∪ Σ is said to be the vocabulary of G. When not mentioned the opposite, a given grammar is considered to be in its augmented form, given by (N , Σ , P , S), where

N = {S } ∪ N ,

Σ = {$} ∪ Σ,

P = {S → S$} ∪ P ,

considering that S ∈ N and $ ∈ Σ.The following conventions are adopted: lower case greek letters (α, β, ...) define strings in V ∗ ; lower case roman letters from the beginning of the alphabet(a, b, ...) and t, bold strings and operator characters (+, −, =, ., etc) represent symbols in Σ, whereas letters from the end of the alphabet (except for t) denote elements in Σ ∗ ; upper case letters from the beginning of the alphabet (A, B, ...) and italic strings represent nonterminals in N , while those near the end (X, Y ,...) denote symbols in V . The empty string is given by λ and the EOF markerby $. The length of a string γ is denoted as |γ|. The symbol Ω stands for the–undefined constant‖. An LR(k) automaton is defined as a tuple LRAk = (Mk , V, P, IS, GOT Ok ,REDk), where Mk is the finite set of states, V and P are as in G, IS is the initial state, GOT Ok : Mk × V ∗ → Mk is the transition function and REDk : Mk ×∗∗Σk → P(P) is the reduction function, where Σk = {w | w ∈ Σ ∗ ∧ 0 ≤ |w| ≤ k}.

A state, either a LR or LALR one, is a group of items. An item is an element in N × V ∗ × V ∗and denoted as A → α • β.The usual way to build the LALR (k) automaton is to calculate the LRA0 automaton first. For such, let the components of LRA0 be defined. The set of states is generated by the following equation:

M0 = {F −1 (CLOSURE ({S → •S$}))}∪{F −1 (CLOSURE(F (q))) | q ∈ SUCC(p) ∧ p ∈ M0 }

where F is a bijective function that maps a state to a set of items (excluded the empty set) and

CLOSURE(is) = is ∪ {B → •β | A → α • Bω ∈ is ∧ B → β ∈ P } SUCC(p)= {F −1 (ADVANCE(p, X)) | X ∈ V } ADVANCE(p, X) = {A → αX • β | A → α • Xβ ∈ F (p)}

The initial state (IS) is obtained by F −1 (CLOSURE({S → •S$})). RED0 (q, λ) is stated as

RED0 (q, λ) = {A → γ | A → γ• ∈ F (q)}

GOTOk , ∀k ≥ 0, can be defined as: GOTOk (p, λ) = p

GOTOk (p, X) = F −1 (CLOSURE (ADVANCE (p, X)))

GOTOk (p, Xα) = GOTOk (GOTOk (p, X), α), ∀α =

λ

From this point, when mentioning a state p, it will be known from the context whether it refers to the number or to the set of items of the state. The LALR (k) automaton, LALR (k) , is a tuple (M0 , V, P, IS, GOTOk , REDk),Where except for REDk , all components are as in LRA0 . Before considering REDk , it is necessary to model a function to capture all predecessor states for a given state q, under a sentential form α. Let PRED be such function:

P RED(q, α) = {p | GOT Ok (p, α) = q} Then, REDk (q, w) = {A $\rightarrow \gamma$ | w $\in$ LAk (q, A $\rightarrow \gamma\bullet$)}

where LAk is the set of lookahead strings of length not greater than k that may follow a processed right hand side of a rule. It is given by LAk (q, A $\rightarrow \gamma$) = {w $\in$ F IRSTk (z) | S $\Rightarrow$ αAz $\wedge$ $\alpha\gamma$ access q} rm$*$ where F IRSTk (α) = {x | ($\alpha \Rightarrow x\beta \wedge |x| = k$) $\vee$ ($\alpha \Rightarrow x \wedge |x| < k$)}lm$*$ $*$ and $\alpha\gamma$ access q iff P RED(q, $\alpha\gamma$) = $\emptyset$. For k = 1, DeRemer and Pennello proposed an algorithm to calculate the lookaheads in LA1 [DeRemer and Pennello 1982] and it still remains as the most efficient one [Charles 1991]. They define the computation of LA1 in terms of FOLLOW1 : (M0 $\times$ N $\times$ M0) $\rightarrow$ P(Σ). The domain (M0 $\times$ N $\times$ M0) is said tobe the set of nonterminal transitions. The first component is the source state, the second the transition symbol and the last one the destination state. For presentation issues, transitions will be written as pairs if destination states areirrelevant. F OLLOW1 (p, A) models the lookahead tokens that follow A when ω becomes the current handle, as long as A $\rightarrow \omega \in$ P . These tokens arise in three possible situations [DeRemer and Pennello 1982]:

a) $\exists$ C $\rightarrow \theta \bullet$ B$\eta \in$ p, such that p $\in$ PRED(q, β), B $\rightarrow$ βA$\gamma \in$ P and $\gamma \Rightarrow \lambda$.
In this case, FOLLOW1 (p, B) $\subseteq$ F OLLOW1 (q, A). This situation is captured by a relation named includes: (q, A) includes (p, B) iff the previous conditions are respected;
(b) given a transition (p, A), every token that is directly read from a state q, as long as GOT O0 (p, A) = q, is in LA1 (p, A). This is modeled by the direct read function:
DR(p, A) = {t $\in \Sigma$ | GOT O0 (q, t) = $\Omega \wedge$ GOT O0 (p, A) = q}
c) Given (p, A), every token that is read after a sequence of nullable nonter minal transitions is in LA1 (p, A). To model the sequence of nullable transi- tions the reads relation is introduced: (p, A) reads (q, B) iff GOT O0 (p, A) =

$*$

q e B $\Rightarrow \lambda$.
The function READ1 (p, A) comprises situations (b) and (c):READ1 (p, A) = DR(p, A) $\cup${READ1 (q, B) | (p, A) reads (q, B)}From this and (a), FOLLOW1 is written as:

FOLLOW1 (p, A) = READ1 (p, A)∪Finally,LA1 (q, A → ω) ={FOLLOW1 (p, A) | p ∈ PRED(q, ω)} {FOLLOW1 (q, B) | (p, A) includes (q, B)}

3. CONFLICTS IN NON LALR (*K*) GRAMMARS

Conflicts arise in grammars when, for a state q in the LALR(k) automaton and
a lookahead string w ∈ Σ ∗ , such that |w| ≤ k, at least one condition is satisfied:
a) |REDk (q, w)| ≥ 2: reduce/reduce conflict;
b) |REDk (q, w)| ≥ 1 ∧ ∃ A → α • β ∈ q ∧ w ∈ F IRSTk (β): shift/reduceconflict.
If one of these conditions is true, *q* is said to be an inconsistent state. A grammar is LALR(*k*) if its correspondent LALR(*k*) automaton has no inconsistent states.A conflict is caused either by ambiguity or lack of right context, resulting in four possible situations. Ambiguity conflicts are the class of conflicts caused by the use of grammar rules that result in at least two different parsing trees for a certain string. These conflicts cannot be solved by increasing the value of *k*; in fact there isn't a *k* (or *k* = ∞) such that the grammar is LALR(*k*). Some of these conflicts are solved by rewriting some grammar rules in order to make it LALR(*k*), according to the *k* used by the parser generator (situation (i)). As an example, consider the *dangling-else* conflict. It is well known that its syntax can be expressed by a non ambiguous LALR(1) set of rules, although is more probable that one will first write an ambiguous specification. Some ambiguity conflicts, on the other hand, simply cannot be removed from the grammar without altering the language in question (situation (ii)). These conflicts are due to the existence of inherently ambiguous syntax constructions. number of **b**'s when a conflict involving the item *B*2 → **b**• is reported. The only possible solution for this example is to rewrite the grammar. For this simple case, such rewrite definitely exists, because *L* is a regular language. Nevertheless, it should be pointed out that this kind of solution is not always possible. The mentioned four situations exhaust all possibilities of causes of conflicts in LALR(*k*) parser construction. These situations of conflicts are also applicable to LR(*k*) parser generation. One type of reduce/reduce conflict is, however, *LALR specific*. It arises when calculating *LAk* for reduction items in states in *M*0. Such calculation can be seen as generating the *LRA*1 automaton and merging states with the same item set; lookaheads of reduction items in the new state are given by the union of the lookaheads in each reduction item in each merged state. When performing the merge, reduce/reduce conflicts, not present in the LR(1) automaton, can emerge. Specific LALR reduce/reduce conflicts occur if the items involved in the conflict do not share the same left context, i.e., a sentential formobtained by concatenating each entry symbol of the states in the path from *IS* to *q*, the inconsistent state. As a consequence, these conflicts do not representambiguity, but do not imply in the existence of a *k*.

4. PROPOSED METHODOLOGY

Parsing table plays very important role in LALR (1) parsing. Its used to make parsing decision during parsing telling parser program whether a input symbol to be shifted onto stack or to reduce an existing substring in stack by matching production rule. Problem arises while there are two entries in parsing table and it cannot be decided which entry is right choice to make. A wrong one chosen can cause failure of parsing further. We are showing here a(1) case of conflicts. Suppose there are two states I_2 I_1 while we are using existing algorithm to make LALR(1) parsing table than merging states I_2 and I_1 will cause shift – reduce conflicts because as wether to shift = input symbol onto stack or to ruduce L by R in below set of LR(1) items.

Example: 1

I_2: S → L. =R, $

I_1 R → L., $

The Core of A Set of LR (1) Items are calculated as below:

The core of a set of LR (1) items is the set of its first component.

Ex: S → L. =R, $	S→L.=R
R → L., $	R → L.

We will find the states (sets of LR (1) items) in a canonical LR(1) parser with same cores. Then we will merge them as a single state.

I_1:L → id.,= L → id.,$

A new state:

I_{12}: L → id.,=/$

have same core, merge them. We will do this for all states of a canonical LR(1) parser to get the states of the LALR parser.

Method that we are proposing makes some changes in the way table is constructed. First we see how LALR(1) is constructed using existing method-

1. Create the canonical LR(1) collection of the sets of LR(1) items for the given grammar.

2. For each core present; find all sets having that same core; replace those sets having same cores with a single set which is their union. $C=\{I_0,...,I_n\}$ ➔ $C'=\{J_1,...,J_m\}$ where $m \le n$

3. Create the parsing tables (action and goto tables) same as the construction of the parsing tables of LR (1) parser.

3.1 Note that: If $J=I_1 \cup ... \cup I_k$ since $I_1,...,I_k$ have same cores
Cores of $goto(I_1,X),...,goto(I_2,X)$ must be same.

3.2 So, $goto(J,X)=K$ where K is the union of all sets of items having same cores as $goto(I_1,X)$.
If we follow this approach than merging of two states cause conflict in case of example 1 as shown above.

So we are giving here a new way to construct LALR(1) parsing table that will merge only those states that do not cause conflict and all other states that are causing conflict in merging will we left as they are. In order to get such effect we are proposing modified LALR (1) parsing table construction algorithm.

5. PROPOSED METHOD FOR LALR (1) PARSING TABLE

1. Create the canonical LR (1) collection of the sets of LR (1) items for the given grammar.

2. Create LR (1) parsing table.

3. Now merge states of LR(1) parsing table having same core LR(0) items and different look ahead symbols in DFA for LR(1) , into a single state where all those rows don't have entry into location Rn Cm and Rx Cm belong to all rows. Where R=Row and C=column, m, n, x are positive numbers.

4. For each core present; find all sets having that same core and does not cause multiple entry into table replace those sets having same cores with a single set which is their union. $C=\{I_0,...,I_n\}$ ➔ $C'=\{J_1,...,J_m\}$ where $m \le n$

5. Create the parsing tables (action and goto tables) same as the construction of the parsing tables of LR (1) parser.

   1. If $J=I_1 \cup ... \cup I_k$ since $I_1,...,I_k$ have same cores.

   Cores of $goto(I_1,X),...,goto(I_2,X)$ must be same.

   2. So, $goto(J,X)=K$ where K is the union of all sets of items having same cores as $goto(I_1,X)$.

6. CONCLUSION

In this paper, we presented the problem of conflict removal in non LALR(1).After going through this presentation my conclusion is that existing LALR(1) parsing method up to date is very effective but it resolves conflict after generation of LALR(1) parsing table , as I have shown LAR(1) paring table is made from CLR(1) parsing table so at the time of construction of table LALR(1) from CLR(1) table if check for multiple entries is put then resultant LALR(1) parsing can be more efficient because it is an attempt to solve problem before it generation at the time it is detected.

REFERENCES

1. Journal of Universal Computer Science, Vol. 13, No. 6 (2007), 737-752 submitted: 19/1/07, accepted: 22/3/07, appeared: 28/6/07 © J.UCS A Methodology for Removing LALR (k) Conflicts Leonardo Teixeira Passos (Federal University of Minas Gerais, Brazil leonardo@dcc.ufmg.br) Mariza A. S. Bigonha (Federal University of Minas Gerais, Brazil mariza@dcc.ufmg.br) Roberto S. Bigonha (Federal University of Minas Gerais, Brazil bigonha@dcc.ufmg.br).

2. Journal of Universal Computer Science 735–752 (2007) Teixeira Passos, L. Bigonha, M.A., Bigonha, R.: A methodology for removing LALR(k) conflicts.

3. Conference of the Centre for Advanced Studies on Collaborative Research (CASCON 2006). pp. 39–53. Toronto (2006) Thurston, A.D., Cordy, J.R.: A backtracking LR algorithm for parsing ambiguous context-dependent languages.

4. Conference sponsored by IEEE, New York) MICKUNAS, M D., AND SCHNEIDER, V.B. On the ability to cover LR (k) grammars with LR(1), SLR(1), and (1, 1) bounded-context grammars Proc of the 14th Ann Symp. on Switching and Automata Theory, Iowa City, Oct 1973, pp. 109-121 .

5. Conference on Functional programming (ICFP'02), Bryan Ford. Packrat parsing: simple, powerful, lazy, linear time. In Proceedings of the seventh ACM SIGPLAN international.