A Methodology for Removing LALR(k) Conflicts

Leonardo Teixeira Passos
Mariza A. S. Bigonha
Roberto S. Bigonha
{leonardo, mariza, bigonha}@dcc.ufmg.br

Federal University of Minas Gerais – UFMG
Programming Languages Laboratory

SBLP 2008

Content

Introduction

Proposed Methodology

SAIDE
- Automatic Conflict Removal
- Methodology Realization
- Architecture

Conclusion

Main References

LALR Grammars

LALR(k) grammars are the standard way to automatically produce parsers by using tools such as CUP [3], Bison [1], YACC [5], etc.

Despite the usefullness of the LALR parsing method, writing a LALR(k) grammar is not a trivial task, specially when $k = 1$.

LALR Grammars

The difficulty in writing LALR(k) grammars is due to the frequent occurrence of *conflicts*.

Grammar	Prods.	Terms.	Non terms.	Conflicts	Conflicts/Prods (%)
Algol-60	131	58	67	61	47
Scheme	175	42	83	78	45
Oberon-2	213	75	112	32	15
Notus	236	77	110	575	41

Table: Conflicts in some test grammars.

Problem

Since the introduction of LALR parsing in 1969, conflicts continue to be removed in an old fashion and primitive manner.

Common approach: analyse the log file dumped by the LALR parser generator.

Problem

Example: Notus programming language log file has 6244 words and 2257 lines.

How one removes the 575 conflicts in the grammar?

- analyse the log file content:
  - difficulty in browsing: a single file containg a huge amount of data, without linkage of them – hyperlinks are not possible in pure text files;
  - difficulty in interpreting its content: non experts in LALR parsing cannot relate the cause of a conflict just facing the LALR automaton.
- migrate to LL parsing: the theory of these parsers is simple and intuitive. But, does it really solve the problem?

Solution

How conflicts can be removed in such harsh environment?

By using a sistematic way = **methodology**

1. understand
2. classify
3. remove the conflict
4. test solution

Content

Introduction

Proposed Methodology

SAIDE
- Automatic Conflict Removal
- Methodology Realization
- Architecture

Conclusion

Main References

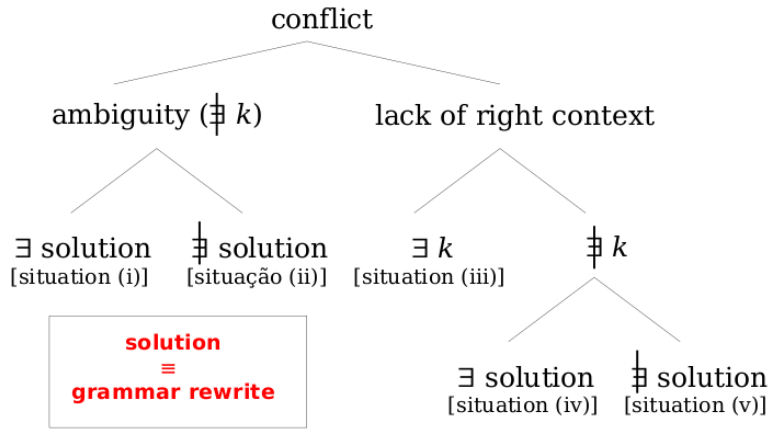
Understanding

Goal: provide faster browsing and better intuition of the cause of conflict by

- modularizing the visualization of the data recorded in the log file;
- interrelating the divided data using hyperlinks;
- presenting conflicts using low-level (LALR automaton) and high level structures (derivation trees).

Classification

Goal: find one of the possible categories to which the conflict reside.



Conflict Removal & Testing

Conflict removal

Goals:

- automatically remove conflicts from situations (iii);
- assist the user with examples that match the ambiguity from situation (i).

Testing

Goal:

- provide an environment in which the user can attest the removal of the conflict.

Content

Introduction

Proposed Methodology

SAIDE
Automatic Conflict Removal
Methodology Realization
Architecture

Conclusion

Main References

SAIDE

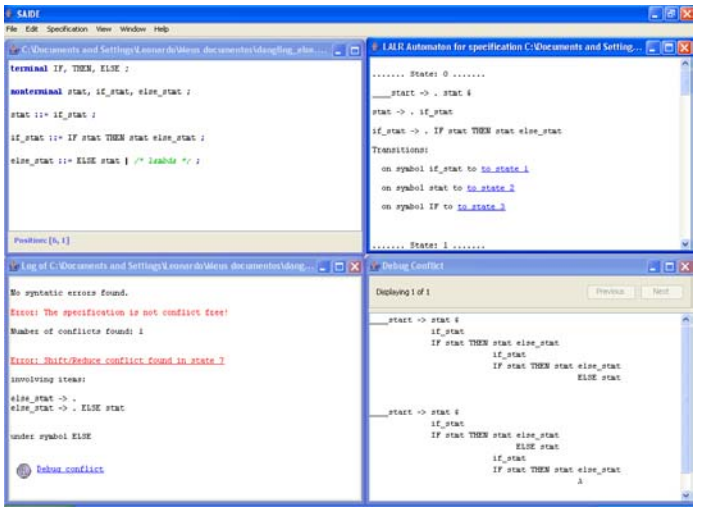
SAIDE: acronym of *Syntax Analyser Integrated Development Environment*.

SAIDE is a parser generator.

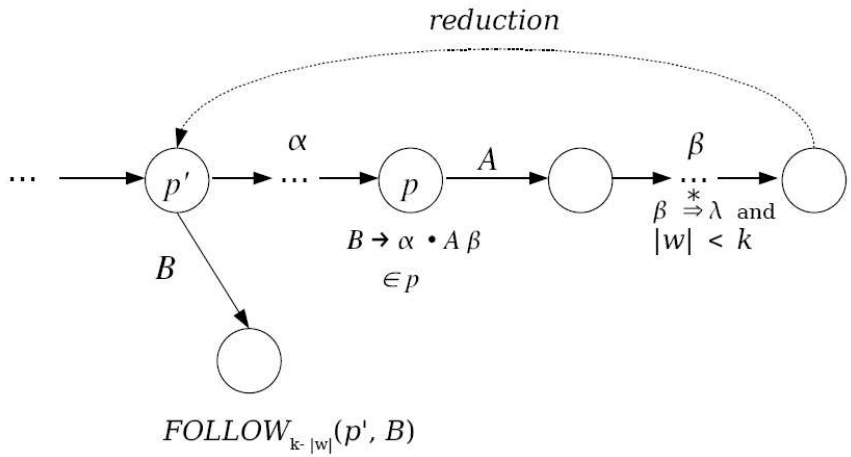
Main features:

- automatically eliminates some conflicts resulted from the lack of right context;
- supports the proposed methodology for manual removal;
- permits the interpretation of grammars written in a variety of specification languages.

SAIDE



Automatic Conflict Removal



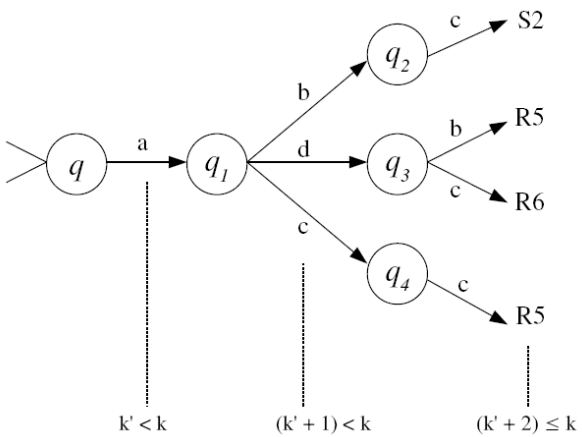
Automatic Conflict Removal

$$
\begin{aligned}
FOLLOW_0(p, A) &= \{\lambda\} \\
FOLLOW_k(p, A) &= READ_k(p, A) \\
&\cup \bigcup \{FOLLOW_k(p', B) \mid (p, A) \textit{ includes } (p', B)\} \\
&\cup \bigcup \{CONCAT(\{w\}, FOLLOW_{k-|w|}(q, B)) \mid \\
&\qquad B \to \alpha \bullet A\beta \in p, \\
&\qquad w \in FIRST_k(\beta), \\
&\qquad |w| < k, \\
&\qquad w \neq x\$, \\
&\qquad q \in PRED(p, \alpha), \\
&\qquad B \neq S\}
\end{aligned}
$$

Major problem: $FOLLOW_{k'}(p, A)$ cannot be reused when calculatig $FOLLOW_{k''}(p, A)$, with $k'' > k'$

Automatic Conflict Removal

Solution: calculate the minimal set of lookaheds w incrementally building and extending *Lookahead Finite Automatons* using a modified algorithm from the one used in [2]:

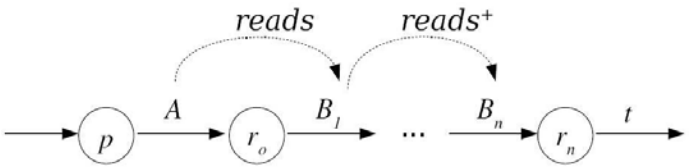


Understanding

S'	\$					
δ_1	B_1	v_1				
	δ_2	B_2	v_2			(a)
		...				
		δ_n	B_n	v_n		
			α	B	β_1	
					β_2	
					...	(b)
					β_{m-1}	
					$t\ \beta_m$	
			α_1	A_1	γ_1	
				α_2	A_2	γ_2 (c)
					...	
					α_{s-1}	A_{s-1} γ_{s-1}
						α_s

Understanding

reads relation [4]



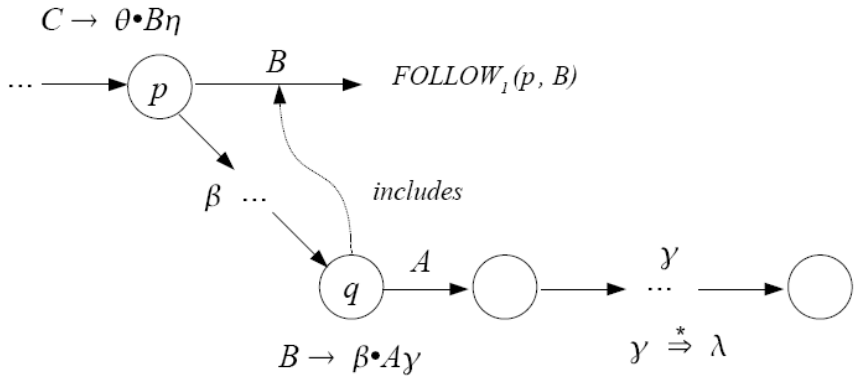
$$READ_1(p, A) = DR(p, A) \cup \bigcup\{READ_1(q, B) \mid (p, A)\ reads\ (q, B)\}$$

$$DR(p, A) = \{t \in \Sigma \mid GOTO_0(q, t) \neq \Omega \ \wedge \ GOTO_0(p, A) = q\}$$

$GOTO_0$ is the transition function of the LALR automaton.

Understanding

includes relation [4]



Understanding

Algorithm part (c) [4]:

for every (q', A_{s-1}) transition representing a node v in the includes grah, given that $A_{s-1} \rightarrow \alpha_s \bullet$ is in q and q' is a predecessor of q under α_s, do:

> traverse the includes graph in a BFS manner from v until a node representing a nonterminal transition (p, B) is found, such that t is in $\text{READ}_1(p, B)$. The state p contains one or more items of the form $B_n \rightarrow \alpha B \beta_1$, where t is in $\text{FIRST}_1(\beta)$.

Understanding

Algorithm part (b) [4]:

calculate the set E, given by:

$$\begin{aligned} E \quad = \quad & \{B_n \rightarrow \alpha B \bullet \beta_1\} \\ & \cup \{A \rightarrow \delta X \bullet \eta \mid A \rightarrow \delta \bullet X\eta \in E \ \wedge \ X \overset{*}{\Rightarrow} \lambda\} \\ & \cup \{C \rightarrow \bullet\alpha \mid A \rightarrow \delta \bullet C\eta \in E \ \wedge \ C \rightarrow \alpha \in P\} \end{aligned}$$

Map each addition to E back to the items that generated it.

Understanding

Algorithm part (c) [4]:

find the shortest path xi from the start item of the LALR automaton to the state that contains $B_n \rightarrow \alpha B \beta_1$

calculte E', given by

$$\begin{aligned} E' \quad = \quad & \{(S' \rightarrow \bullet S\$, 1)\} \\ & \cup \{(C \rightarrow \bullet\alpha, j) \mid (A \rightarrow \delta \bullet C\eta, j) \in E' \wedge \ C \rightarrow \alpha \in P)\} \\ & \cup \{(A \rightarrow \delta X \bullet \eta, j+1) \mid (A \rightarrow \delta \bullet X\eta, j) \in E' \wedge X = \xi_j \wedge j \leq |\xi|\} \end{aligned}$$

in a BFS way, linking additions to E' back to the pairs that generated them

stop as soon as $(B_n \rightarrow \alpha \bullet B\beta_1, |\xi|)$ appears.

Understanding

- **Debug tree for reduce/reduce conflicts**:

  apply sequence (c), (b), (a) to each reduction item.

  Problem: conflicts specific to LALR parser construction.

- **Debug tree for shift/reduce conflicts**:

  apply sequence (c), (b), (a) to the reduction item.

  make $\xi = \delta_1\delta_2\delta_n\alpha\alpha_1...\alpha_s$ and apply the E' set calculation algorithm to every shift item.

Classification

Currently performed manually.

Hints to help user:

- shows whether the conflict is in LALR or LR;
- presents the set of strings of length up to k_{max} that were tried, but still were not able to remove the conflict;
- examples of some ambiguity examples according to some known patterns.

Classification

Ambiguity known patterns
(assuming $S' \stackrel{*}{\Rightarrow} \xi' P \xi''$, $\delta_i \stackrel{*}{\Rightarrow} \lambda$ and $P_i \stackrel{*}{\Rightarrow} P$):

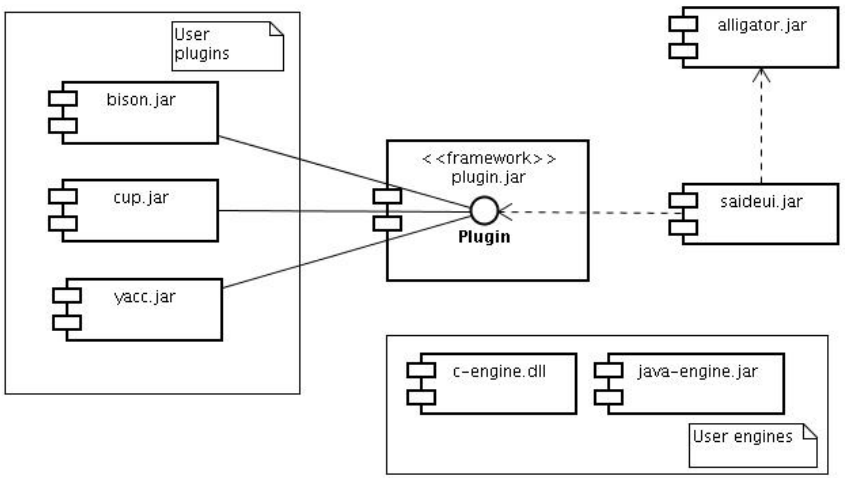
- $P \stackrel{*}{\Rightarrow} \delta_1 P_1 \delta_2 \alpha \delta_3 P_2 \delta_4$ and $P \stackrel{*}{\Rightarrow} \delta_6 \beta \delta_7 \delta_8$: captures ambiguous constructions such as $E \rightarrow E + E \mid t$;
- $P \stackrel{*}{\Rightarrow} \delta_1 \alpha \delta_2 P_1 \delta_3$ and $P \stackrel{*}{\Rightarrow} \delta_5 \alpha P_2 \delta_6 \beta \delta_7 P_3 \delta_8$: dangling-else's instances;
- $P \stackrel{*}{\Rightarrow} \delta_1 \alpha \delta_2 P_1 \delta_3$ and $P \stackrel{*}{\Rightarrow} \delta_5 P_2 \delta_6 \beta \delta_7$: captures ambiguous constructions such as the rules *exp* $\rightarrow$ **let** *dcl* **in** *exp* **where** *exp* and *exp* $\rightarrow$ *exp* **where** *exp*;
- $P \stackrel{*}{\Rightarrow} \delta_1 \alpha \delta_2 P_1 \delta_3 \beta \delta_4$ and $P \stackrel{*}{\Rightarrow} \delta_6 \alpha \delta_7 P_2 \delta_8 \beta \delta_9$: captures alias between nonterminals.

Testing

Resubmit the specification to SAIDE to check if the conflict has been eliminated, by reanalysis of the list of conflicts.

Upon failure, restart from one of the points before testinge, or remove the next conflict, if it exists.

Plugin Facility (up to date)



Content

Introduction

Proposed Methodology

SAIDE
- Automatic Conflict Removal
- Methodology Realization
- Architecture

Conclusion

Main References

Conclusion

Conflict removal is difficult and takes a great amount of time during LALR parser construction.

The usual way to remove conflict is not adequate.

Proposed solution: a methodology for manual conflict removal.

Proposed tool: SAIDE.

Contributions & Future Work

Main contributions are:

- the proposed methodology systemizes the steps in conflict removal;
- an informal proof, described in the article, that an algorithm based on the $FOLLOW_k$ equation would terminate;
- SAIDE provides a uniform environment without any concern of the specification language used.

Future work:

- add some level of automation to the classification phase;
- perform validation tests on SAIDE.

Content

Introduction

Proposed Methodology

SAIDE
- Automatic Conflict Removal
- Methodology Realization
- Architecture

Conclusion

Main References

Bison.
Bison - gnu parser generator.
http://www.gnu.org/software/bison/.
Last access: 24/04/2007.

P. Charles.
A Pratical Method for Constructing Efficient LALR(k) Parsers with Automatic Error Recovery.
PhD thesis, New York University, 1991.

CUP.
Cup: Lalr parser generator in java.
http://www2.cs.tum.edu/projects/cup/.
Last access: 13/01/2007.

F. DeRemer and T. Pennello.
Efficient computation of lalr(1) look-ahead sets.
ACM Trans. Program. Lang. Syst., 4(4):615–649, 1982.

S.C Johnson.
Yacc: Yet another compiler compiler.
In *UNIX Programmer's Manual*, volume 2, pages 353–387.
Holt, Rinehart, and Winston, 1979.