Single Production Elimination in LR(1) Parsers: A Synthesis

Wilf R. LaLonde

SCS-TR-2
June 1982

School of Computer Science
Carleton University
Ottawa, K1S 5B6
Canada

This research was supported in part by contracts and grants from Carleton University and the Natural Sciences and Engineering Research Council of Canada.

Single Production Elimination in LR(1) Parsers: A Synthesis

Wilf R. LaLonde
School of Computer Science
Carleton University

Abstract

Recently, Soisalon-Soininen provided a necessary and sufficient condition for error entries in any deterministic LR(1) based parser to be classified "don't care". This result allows a previous algorithm by Aho and Ullman for partially solving the single production elimination problem to be used on LALR(1) parsers that employ default reductions.

Careful analysis of the Aho and Ullman algorithm shows that the major transformation step actually consists of two primitive transformations, only one of which is actually required for single production elimination. The other is a space saving transformation. When the algorithm is applied to LALR(1) parsers obtained by extending Knuth LR(0) parsers, it can be seen to emphasize space saving at the expense of single production elimination.

The opposite emphasis can be obtained by first starting with an Anderson LR(0) parser, a space inefficient parser for which all reductions by single productions have been removed, and extending it to an LALR(1) parser in the usual way. Because the Soisalon-Soininen result applies to all LR based parsers, the Aho and Ullman algorithm can be applied. By restricting ourselves to the space saving variant, an LALR(1) parser without reductions by single productions and with default reductions can be obtained.

Index Terms

LR(K) grammars, LALR(k) grammars, parser construction, default reductions, error entries, single productions, compilers.

1. Introduction

In recent years, much attention has been devoted to decreasing the table storage space and increasing the parsing speed of LR(k) parsers; e.g., Aho and Ullman[4,5], Anderson[6], Anderson, Eve, and Horning[7], DeRemer[11,12], Johnson[14], Joliat[15,16], Korenjak[18], LaLonde[21], Pager[23,24,25], Soisalon-Soininen[27], ... One approach for doing this has been to eliminate unnecessary reductions by single productions; i.e., productions of the form A->X where X is either a terminal or a nonterminal; e.g., Aho and Ullman[4,5], Anderson[6], Anderson, Eve, and Horning[7], Pager[22,26], Demers[9,10], Backhouse[8], Joliat[17], Koskimies[19], LaLonde[21], and Soisalon-Soininen[28,29]. Another technique consists of modifying the parsing tables to incorporate default reductions. If one or more reduce actions are described by some table, then one of these reduce actions is chosen for the default reduction which is performed instead of reporting error; e.g., see Pager[22], Aho and Johnson[2], Horning[13], LaLonde[20], and Aho[1].

Anderson[6] along with Anderson, Eve, and Horning[7] have described a method for constructing LR parsers in which all reductions by single productions are eliminated. However, their technique may result in parsers with considerably more tables than would be required had the elimination not been performed. On the other hand, with the techniques of Aho and Ullman[5] and Demers[10], no increase in the number of tables can occur, but the elimination of reductions by single productions is not always guaranteed. Pager[22,23] has extended the method of Aho and Ullman[5] so that all reductions by single productions are eliminated. An additional benefit of the algorithm is that it removes all transitions by nonterminals that occur as the left parts of the single productions. Of all algorithms that solve the general problem in its full generality, this algorithm results in the smallest parsers. LaLonde[21] and Backhouse[8] considered versions of the method of Pager, and Joliat[17] considered a special case of a more general method of Anderson, Eve, and Horning[7].

In some of these techniques, the elimination of reductions by single productions is performed during the construction process; e.g., Anderson[6], Anderson, Eve, and Horning[7], versions of Demers[10] and Pager[22], and LaLonde[21] whereas in others, the parsers are first constructed using one of the standard techniques and then the elimination of the reductions by single productions is effected; e.g. Aho and Ullman[5], versions of Pager[22], and versions of Demers[10].

When constructing such optimized parsers, great care must be taken if modifications to include default reductions are desired lest the parsers lose the ability to detect errors in certain cases; e.g., see Aho and Ullman[3], LaLonde[21], Pager[22], and Soisalon-Soininen[28]. In general, the Pager[22] solution requires the addition of error checks in tables containing default reductions. No other parser implementations require such checks. Soisalon-Soininen[29] solves the problem by eliminating only those single productions for which both default reductions and error detection are not compromised. Although his result allows the elimination of single productions to be maximized, it is unsatisfying because the elimination is not complete.

The problem lies with the LALR(1) parsers constructed by extending the Knuth LR(0) tables; essentially, they are too compact. Short of deriving an algorithm that uncompacts the tables through some state splitting scheme, it is not possible to fully eliminate reductions by single productions while still retaining the ability to use default reductions. In order to take full advantage of the Soisalon-Soininen[29] result, it is essential that variations of the Knuth LR(0) tables be used that are not "too compacted"; e.g., Anderson[6] LR(0) tables. With a simple grammatical transformation, the standard Knuth algorithm can be used to obtain the Anderson tables. More important, the Soisalon-Soininen result should be used for achieving a space optimization on the Anderson tables since reductions by single productions have already been eliminated in the latter. Note that such an optimization was still an open problem prior to the Soisalon-Soininen contribution that solved it.

The rest of the paper is organized as follows. Section 2 contains some terminology and a brief review of the theory of LR parsing relevant to the discussions in subsequent sections. In section 3, we review the technique used by Anderson. Then in section 4, we present the alternative technique.

2. Review of the Problem

In order to produce efficient LALR based parsers with tables stored in list form, two speed increasing transformations are highly desirable: (1) removal of reductions by single productions, and (2) use of default reductions. Consider grammar G1 and its corresponding LALR(1) tables T1 (Figure 1).

```
G1: S -> E; | E
    E -> E+i | i
```

If we remove the reductions by single productions, using Aho and Ullman's [3] algorithm for example, the resulting tables T2 (Figure 2) are obtained. If we modify T1 to have default reductions, the result is T3 (Figure 3). The problem arises when both transformations are applied simultaneously on T1 to get T4 (Figure 4). Using T4, it is easy to see that illegal string i;; is accepted (no error is noticed) even though is is correctly rejected by T1, T2, and T3.

It can be shown that this problem is symptomatic only of LALR parsers, not of SLR parsers [21, 22, 29]. Moreover, it arises because some of the error entries in T1 are essential for error detection [29]. The error entries are those implied transition symbols for which an error would be signaled by the parser. For example, the initial table in Figure 1 has transitions under S, E, and i but not +, ;, or e (these latter three are error entries). In some cases, these error entries are essential (such as in table 0 of T1) and in others, they are not (such as the error entry for i in table 4 as witnessed by the corresponding table in T3). The source of the problem arises from the table merging effect of the single production removal transformation when the distinction between essential error entries and non-essential error entries is ignored. In our example, the semicolon is an essential error entry for table 0 in T1 and a legal transition symbol for tables 3

and 4. Consequently, tables 0, 3, and 4 should not have been merged as was done in T4. Soisalon-Soininen [29] has provided an algorithm which will determine the essential error transitions of any LR based parser. If the Aho and Ullman algorithm is revised so that tables are merged only when there are no incompatibilities between transition symbols in shift, reduce, essential error, and accept contexts, the problems mentioned above vanish. In particular, T5 (Figure 5) results. Unfortunately, the reduction by single production S->E (according to the Aho and Ullman algorithm) could not be removed. As an aside, you might also notice that it is necessary to know which error entries are essential in order to introduce defaults. For instance, transition e in table 2 of T5 was not replaced by a default because ;, +, and i are essential error entries (not because it relates to acceptance). In combined tables 3 and 4 of T5, the default transition replaces the reduce transition under e and the non-essential error transition under i.

Our contribution is to show how the revised Aho and Ullman algorithm may be further modified so that it succeeds in removing the reduction by single production S->E. Additionally, since the Aho and Ullman algorithm is not a general algorithm (there are situations unrelated to the above for which it fails to work), we wish to review a known technique which works in the general case and which, in our opinion, ought to be the method of preference by LR based parser constructor implementors.

3. The Revised Aho and Ullman Algorithm

The actions of an LR parser are restricted to one of shift, reduce, essential error, non-essential error, and accept. We define two transitions under the same symbol to be strongly compatible if either (1) they have identical associated actions (with shift actions implying identical successor tables and reduce actions implying the same production), or (2) one of them is a non-essential error action. They are weakly compatible if either (2) holds or one of them is an error action (irrespective of the class of error action). Two tables are compatible (strongly or weakly) if all transitions under the same symbol are compatible (strongly or weakly respectively).

In general, the Aho and Ullman algorithm applies the following step in a top-down manner to all single productions for which the corresponding reduction is to be removed. That is, a partial order is established such that all A-productions are considered before B-productions if A->B is a single production to be removed.

The Existing Algorithm's Basic Step (Incorporating Non-Essential Error Entries)

If A->B is a single production for which the associated reduction is not desired and T is an LR table with A-successor TA and B-successor TB, then replace all references to TA and TB by T' = TA $\cup$ TB IF (ignoring reduce actions associated with A->B) TA and TB are strongly compatible.

We are claiming that replacing TB by T' = TA $\cup$ TB (with references to

the reduce actions involving A->B removed) is the single production removal transformation whereas replacing TA by T´ is the space optimization (see Figure 6). Intuitively, once a B is generated bottom up, any action following B and the A (since A->B) in that local context is legitimate. The converse may not be true; that is, once an A is generated bottom up, actions following A are legitimate but not necessarily actions following B. This can be seen by looking at T1 in the context S->E. After an E is generated, the shift actions under ; and + are legal and so is the accept action under e (if the reduction by S->E is removed). For the converse, after an S is generated (for instance, using S->E;), the only allowed action is the accept action under e. Neither shift actions under ; and + are allowed. Additionally, we claim that weak compatibility is a sufficient criteria for single production elimination whereas strong compatibility is required for the space optimization. The basic step in the Aho and Ullman algorithm can then be rewritten as the following two steps. Variations of the algorithm can be obtained by removing one or the other of the steps.

The Single Production Elimination Step

If A->B is a single production for which the associated reduction is not desired and T is an LR table with A-successor TA and B-successor TB, then replace all references to TB by TA ∪ TB IF TA and TB are weakly compatible.

The Space Optimization Step

If A->B is a single production and T is an LR table with A-successor TA and B-successor TB such that TB has no reduce action involving B (i.e., previously modified by the single production elimination step), then replace all references to TA by TB (equivalent to TA ∪ TB) IF TA and TB are strongly compatible.

It is straightforward to see that the revised version is correct though tedious to prove formally. Informally, this version has stronger preconditions than the original. Consequently, each step applicable in this version would also be applicable in the original. That the stronger condition is required is manifest by the need to preserve error correction and follows from the Soisalon-Soininen[29] result.

Using the latest version of the algorithm on T1, we find that we can eliminate all reductions by single productions (see T6, Figure 7).

4. A Variation on the Anderson Algorithm

Anderson´s LALR algorithm [6] for directly generating parsers without reductions by single productions completely solves the problem but results in a very large number of tables. The space optimization portion of the revised Aho and Ullman algorithm (actually a direct application of Soisalon-Soininen´s result) can be used to decrease the

table sizes to more reasonable proportions. My intuition indicates that resulting sizes would be similiar to those of standard LALR tables though I lack statistics to support it.

The variation requires a rather simple grammatical transformation (perhaps obvious in hindsight) followed by a straightforward extension of the standard LR table construction technique to handle the transformation.

The Grammatical Transformation (Applied to the Augmented Grammar)

If A->B is a single production to be eliminated, remove it from the list of productions and replace all occurrences of A by {A ¦ B}. If successive applications of this rule results in nested brackets, remove the inner brackets.

With this transformation, our grammar G1 becomes G2. Note: the transformation operates on the augmented grammar ultimately used by the parser constructor.

G2: S´->{S¦E¦i}
S->{E¦i};
E->{E¦i}+i

The LR table construction technique, suitably modified to handle imbedded alternations, is then used. The resulting tables T7 are shown in Figure 8. No space optimizations can be performed. Defaults can be added arbitrarily.

5. Conclusions

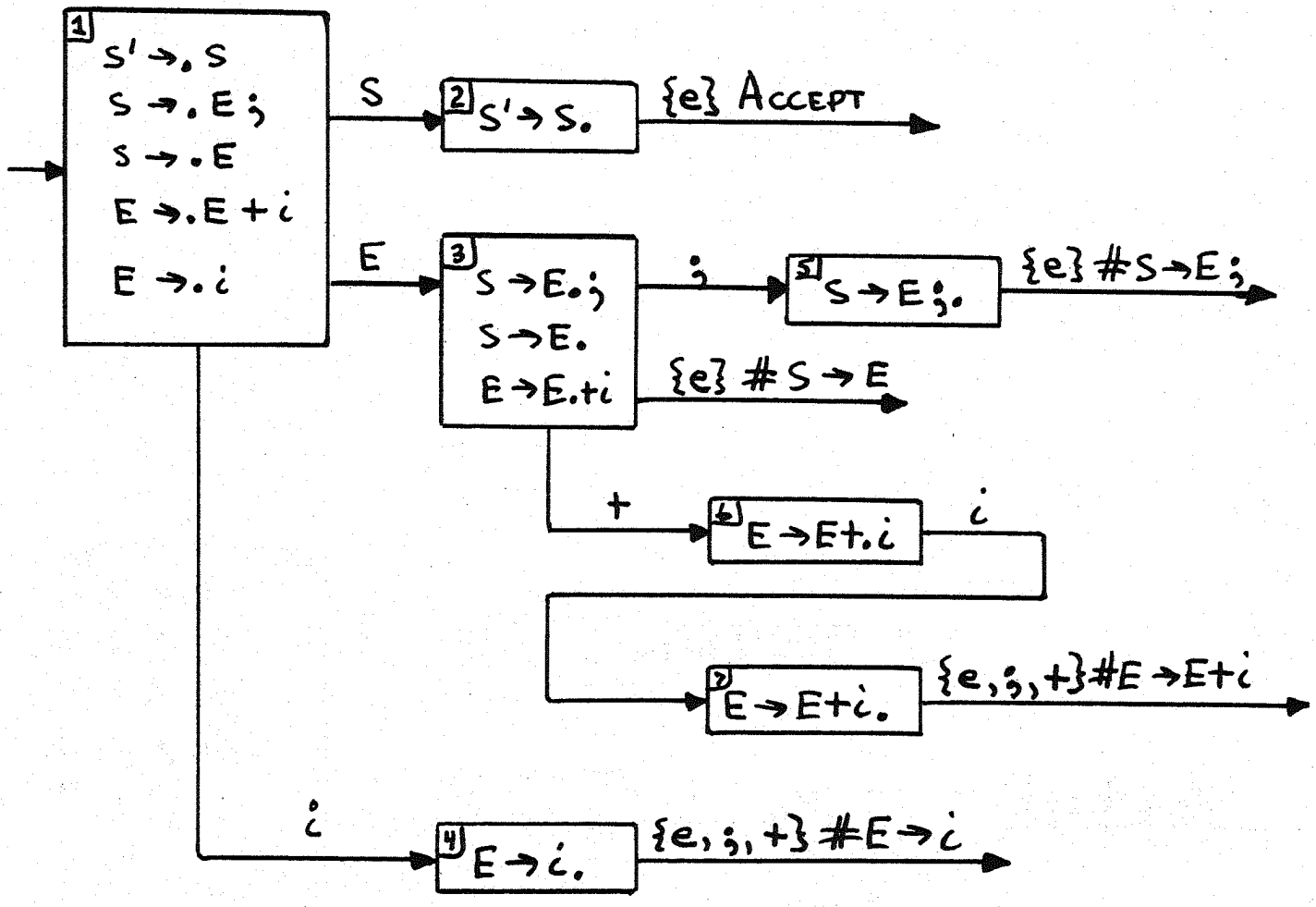
We have clarified the task actually performed by the Aho and Ullman algorithm by showing that it actually consists of two subtasks: one eliminating reductions by single productions and the other performing a space optimization. We have also suggested a variation of the Anderson algorithm for constructing parsers without reductions by any subset of the single productions. The resulting parsers can be compacted by a straightforward space optimization technique directly related to the single production elimination process. In addition to being simple, the technique allows defaults to be used in the LALR parsers. All of these techniques rely on the Soisalon-Soininen algorithm for distinguishing essential error entries from non-essential error entries.

6. References

1. Aho,A.V. Language theory in compiler design. In Applied Computation Theory: Analysis, Design, Modelling (ed. R.T.Yeh), pp. 185-237. Prentice Hall, Englewood Cliffs, N.J., 1976.
2. Aho,A.V., and Johnson,S.C. LR parsing. Computing Surveys 6:2, pp. 99-124, 1974.
3. Aho,A.V., and Ullman,J.D. The Theory of Parsing, Translation and Compiling. Vol 1: Parsing, Vol 2: Compiling. Prentice-Hall Inc., 1972.

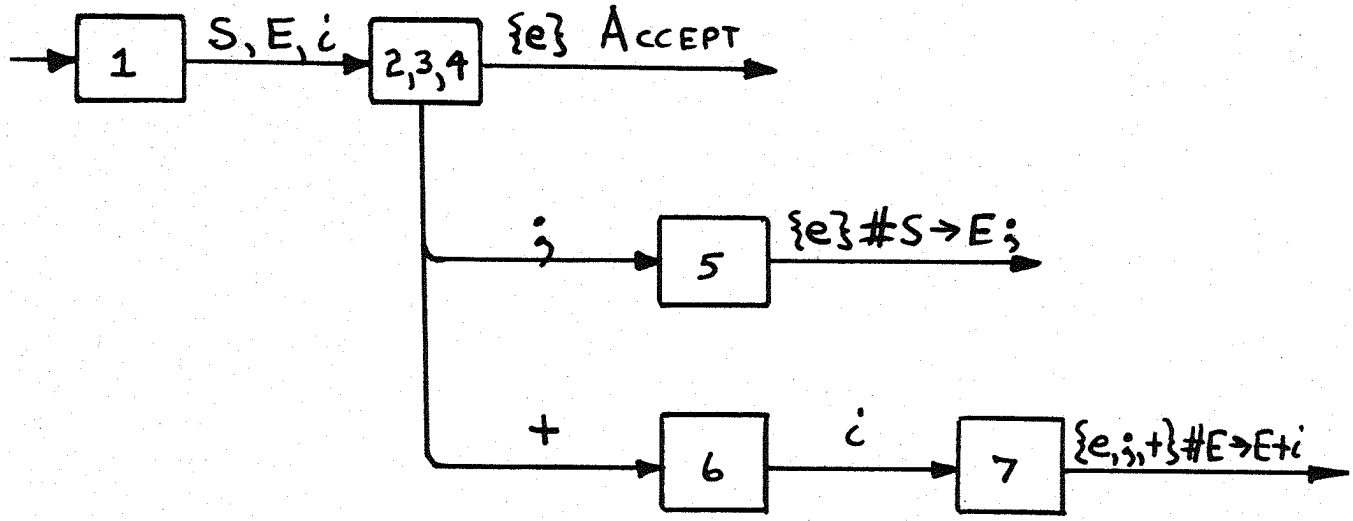
4. Aho,A.V.and Ullman,J.D. Optimization of LR(k) parsers. Journal of Computer and System Sciences 6, pp. 573-602, 1972.
5. Aho,A.V.and Ullman,J.D. A technique for speeding up LR(k) parsers. Siam J. Computing 2:2, pp. 106-127, 1973.
6. Anderson,T. Syntactical analysis of LR(k) languages. Ph.D. thesis, Computing Lab., Univ. of Newcastle upon Tyne, 1972.
7. Anderson,T.,Eve,J., and Horning,J.J. Efficient LR(1) parsers. Acta Informatica 2:1, pp. 12-39, 1973.
8. Backhouse,R.C. An alternative approach to the improvement of LR(k) parsers. Acta Informatica 6:3, pp. 277-296, 1976.
9. Demers,A.J. Skeletal LR parsing. IEEE Conference Record of 15th Annual Symposium on Switching and Automata Theory. pp. 185-198, 1974.
10. Demers,A.J. Elimination of single productions and merging nonterminal symbols of LR(1) grammars. Computer Languages 1:2, pp. 105-119, 1975.
11. DeRemer,F.L.Practical translators for LR(k) languages. Ph.D. thesis, Dept. of Elec. Eng., M.I.T., Cambridge, Mass., 1969.
12. DeRemer,F.L.Simple LR(k) grammars. Comm. ACM 14:7, (July 1971), 453-460.
13. Horning,J.J. LR parsers and analysers, in Compiler Construction, an Advanced Course, Lecture Notes in Computer Science, Vol. 21 (ed. Bauer,F.L., and Eickel,J.), pp. 85-108, Springer-Verlag, Berlin-Heidelberg-New York, 1974.
14. Johnson,S.C. YACC - yet another compiler-compiler. Computing Science Technical Report 32, Bell Laboratories, Murray Hill, N.J, 1975.
15. Joliat,M.L. On the reduced matrix representation of LR(k) parser tables. Technical Report CSRG-28, Computer Systems Research Group, University of Toronto, Oct 1973.
16. Joliat,M.L. The BIGLALR parser generator system. Central Laboratory, Bell-Northern Research Ltd, Ottawa, Ontario, 1975.
17. Joliat,M.L. A simple technique for partial elimination of unit productions from LR(k) parsers. IEEE Trans. Comput. (Corresp.) C-27:7, pp. 763-764, 1976.
18. Korenjak.A.J. A practical method for constructing LR(k) processors. Comm. ACM 12, (November 1969), pp. 613-613.
19. Koskimies,K. Optimization of LR(k) parsers (in Finnish), M.Sc. Thesis, University of Helsinki, Helsinki, 1976.
20. LaLonde,W.R. An efficient LALR parser generator. Technical Report CSRG-2, Computer Systems Research Group, Univ. of Toronto, 1971.
21. LaLonde,W.R. On directly constructing LR(k) parsers without chain reductions. Conference Record Third ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, pp. 127-133, 1976.
22. Pager,D. On eliminating unit productions from LR(k) parsers. Technical Report PE 245, University of Hawaii, Honolulu, 1973.
23. Pager,D. On the decremental approach to left-to-right parsing. Technical Report, University of Hawaii, Honolulu, 1972.
24. Pager,D. On combining compatible states in LR(k) parsing, Technical Report PE 257, University of Hawaii, Honolulu, July 1972.
25. Pager,D. A compaction algorithm for combining the symbol-action lists of an LR(k) parser, Technical Report PE 259, University of Hawaii, Honolulu, July 1972.

26. Pager,D. On eliminating unit productions from LR(k) parsers. in Automata, Languages and Programming, 2nd Colloquium, Lecture Notes in Computer Science, Vol. 14 (ed. Loeckx,J.), pp. 242-254, Springer-Verlag, Berlin-Heidelberg-New York, 1974.
27. Soisalon-Soininen,E. Design of an automatic constructor of LR parsers (in Finnish). Internal report 1976/27, Department of Computer Science, University of Helsinki, Helsinki, 1976.
28. Soisalon-Soininen,E. Elimination of single productions from LR parsers in conjunction with the use of default reductions. Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, pp. 183-193, 1977.
29. Soisalon-Soininen,E. Inessential Error Entries and Their Use in LR Parser Optimization. ACM TOPLAS 4, 2 (April 1982), 179-195.



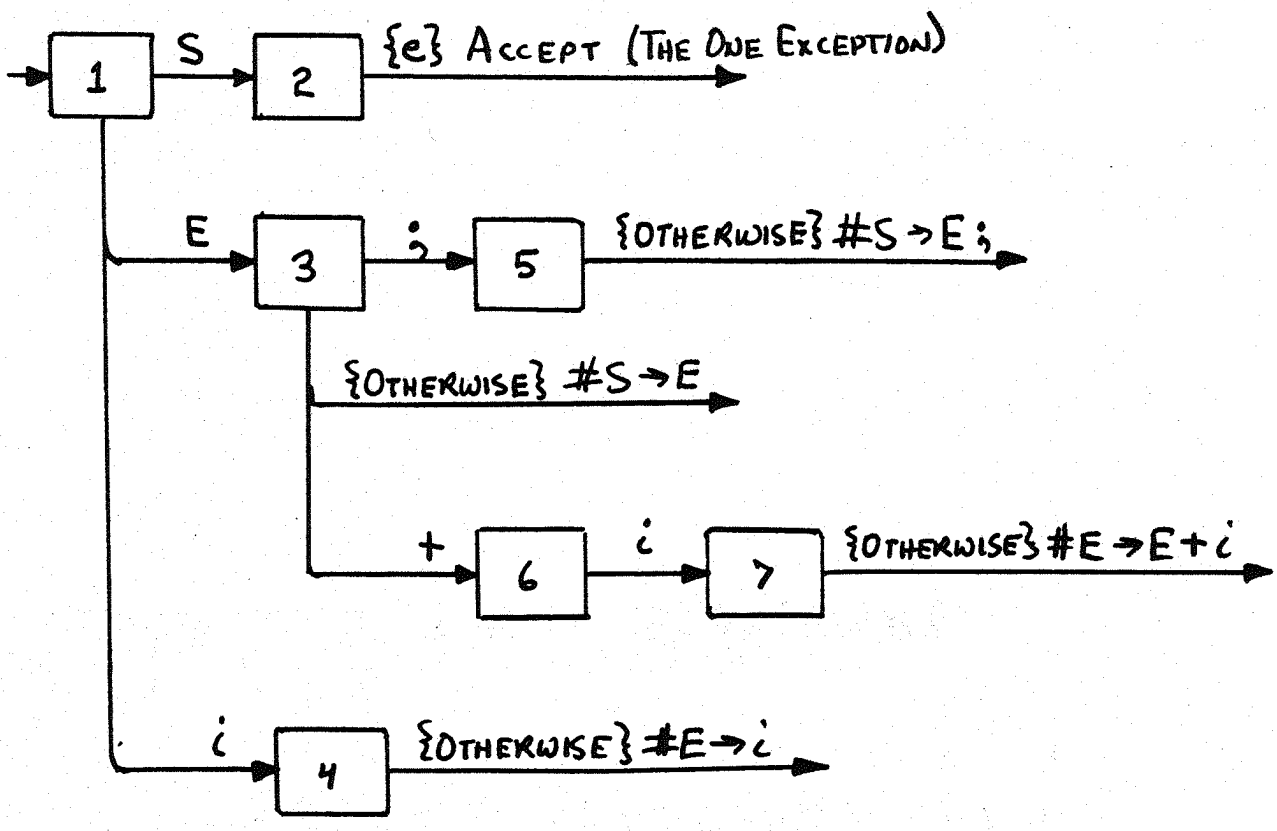
LALR(1) Tables T1 for G1: S->E;|E, E->E+i|i.

Figure 1



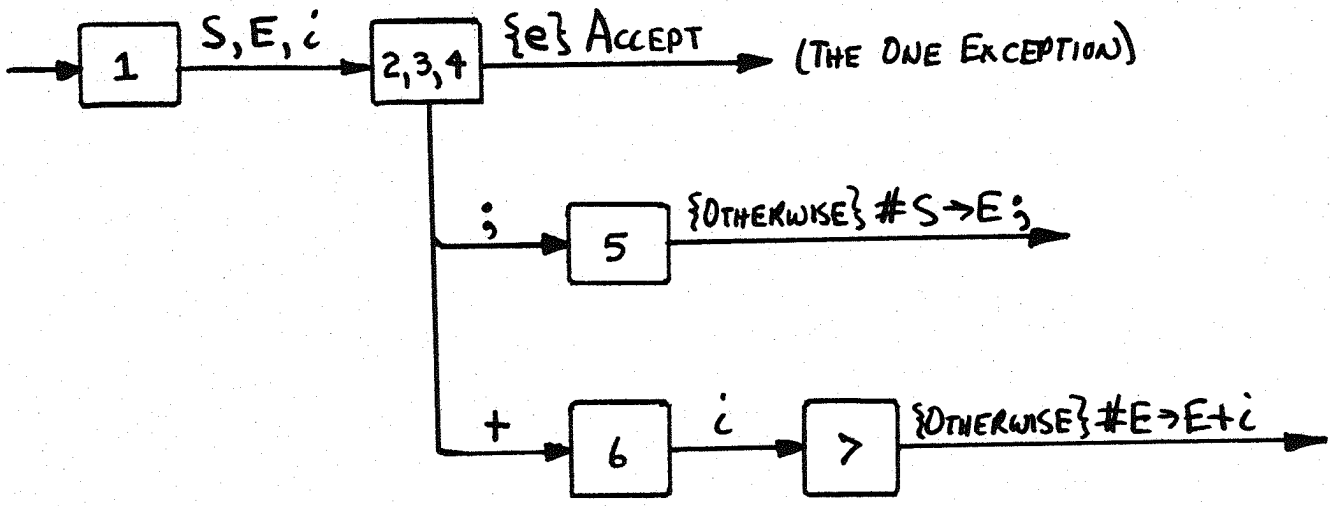
T2 as Modified from T1 by (1) Removing Reductions by Single Productions S->E and E->i.

Figure 2



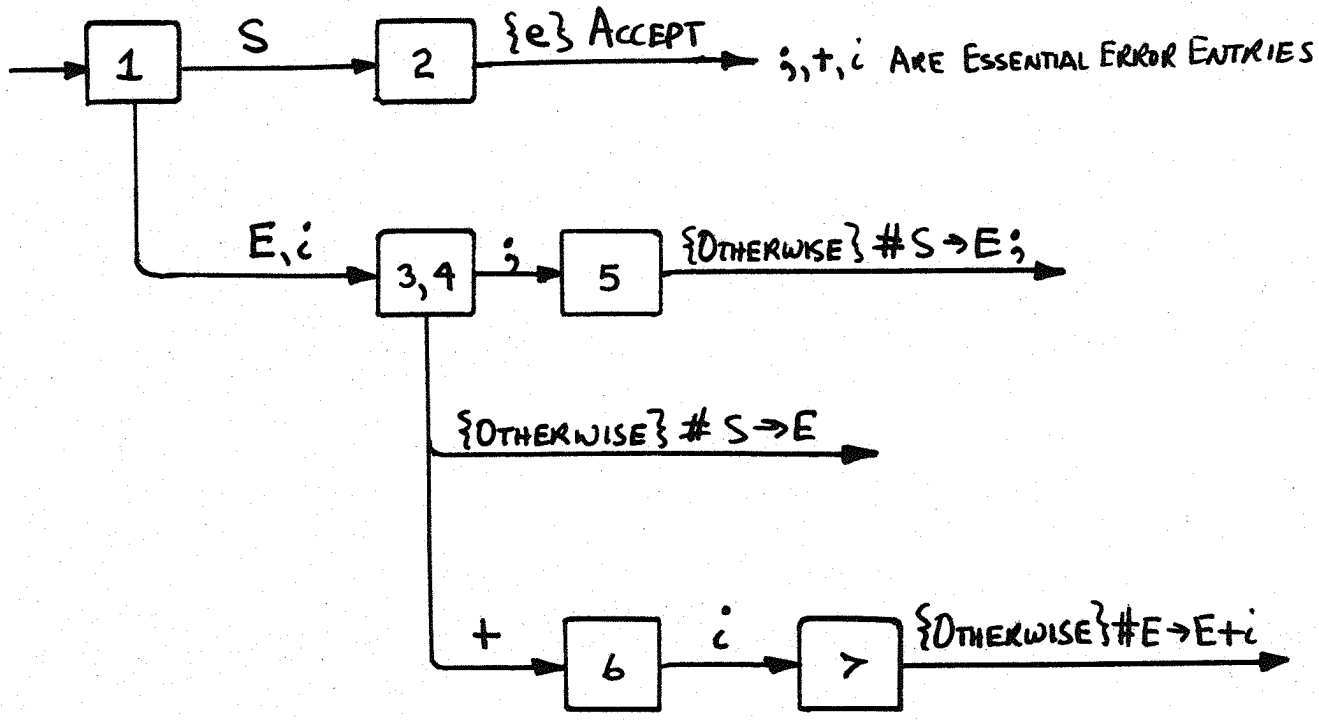
T3 as Modified from T1 by (2) Adding Default Reductions

Figure 3



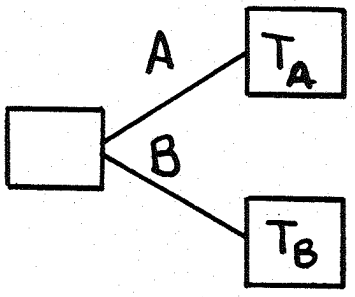
T4 as Modified from T1 by Performing Transformations of the Previous Two Figures (It Incorrectly Accepts i;;, i;+i, i;+i;+i, etc.)

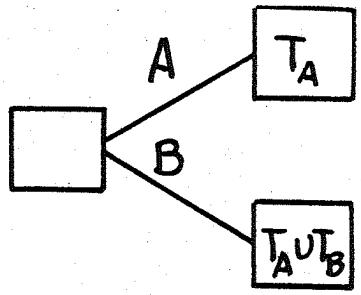
Figure 4



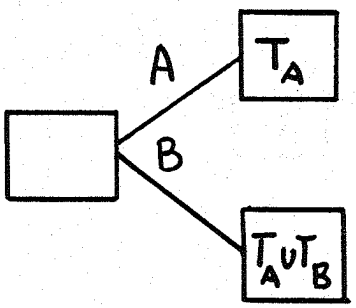
T5 as Modified from T1 Using the Modified Aho and Ullman Algorithm + Defaults

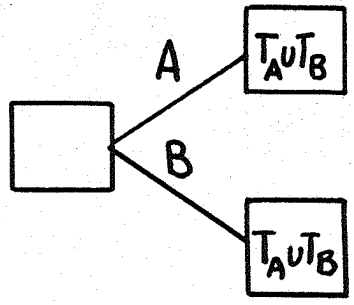
Figure 5





(1) The Single Production Elimination Step

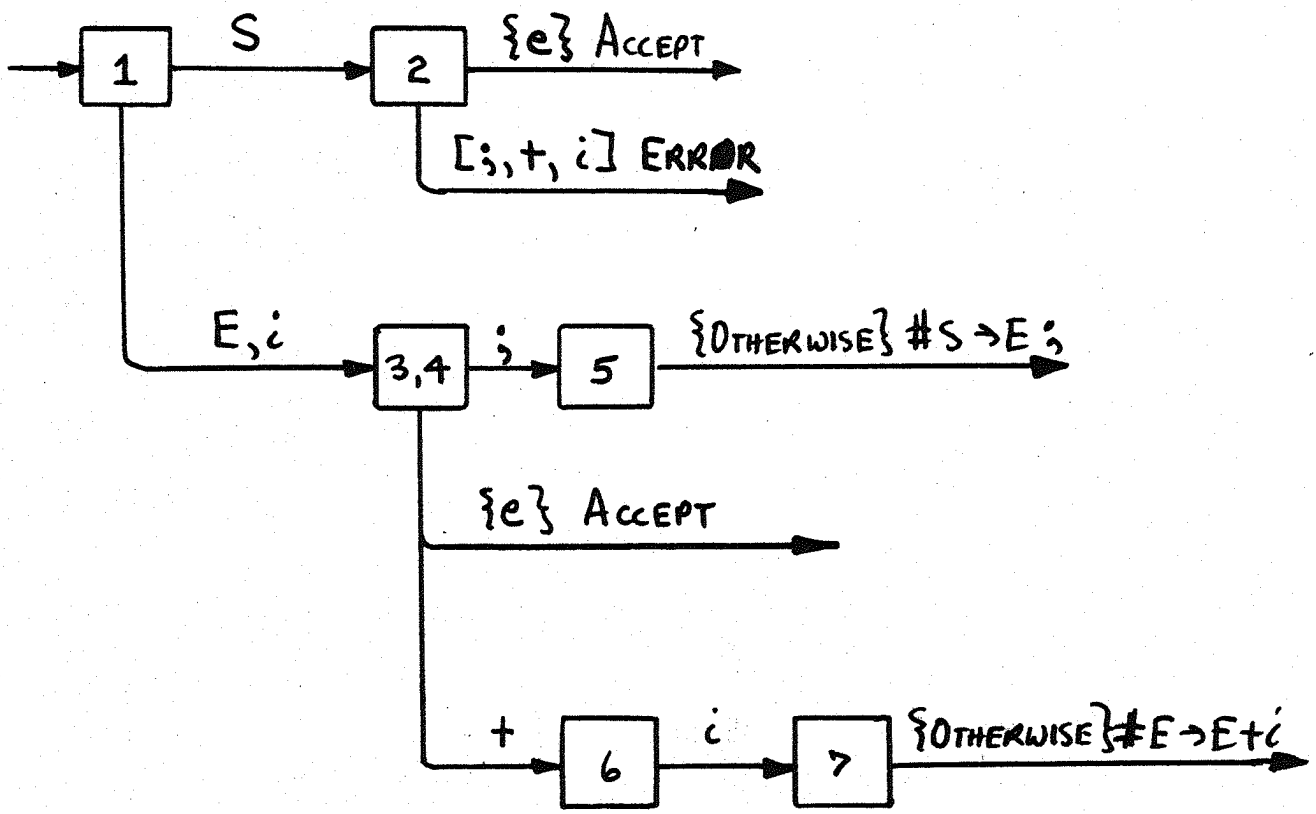




(2) The Space Optimization Step

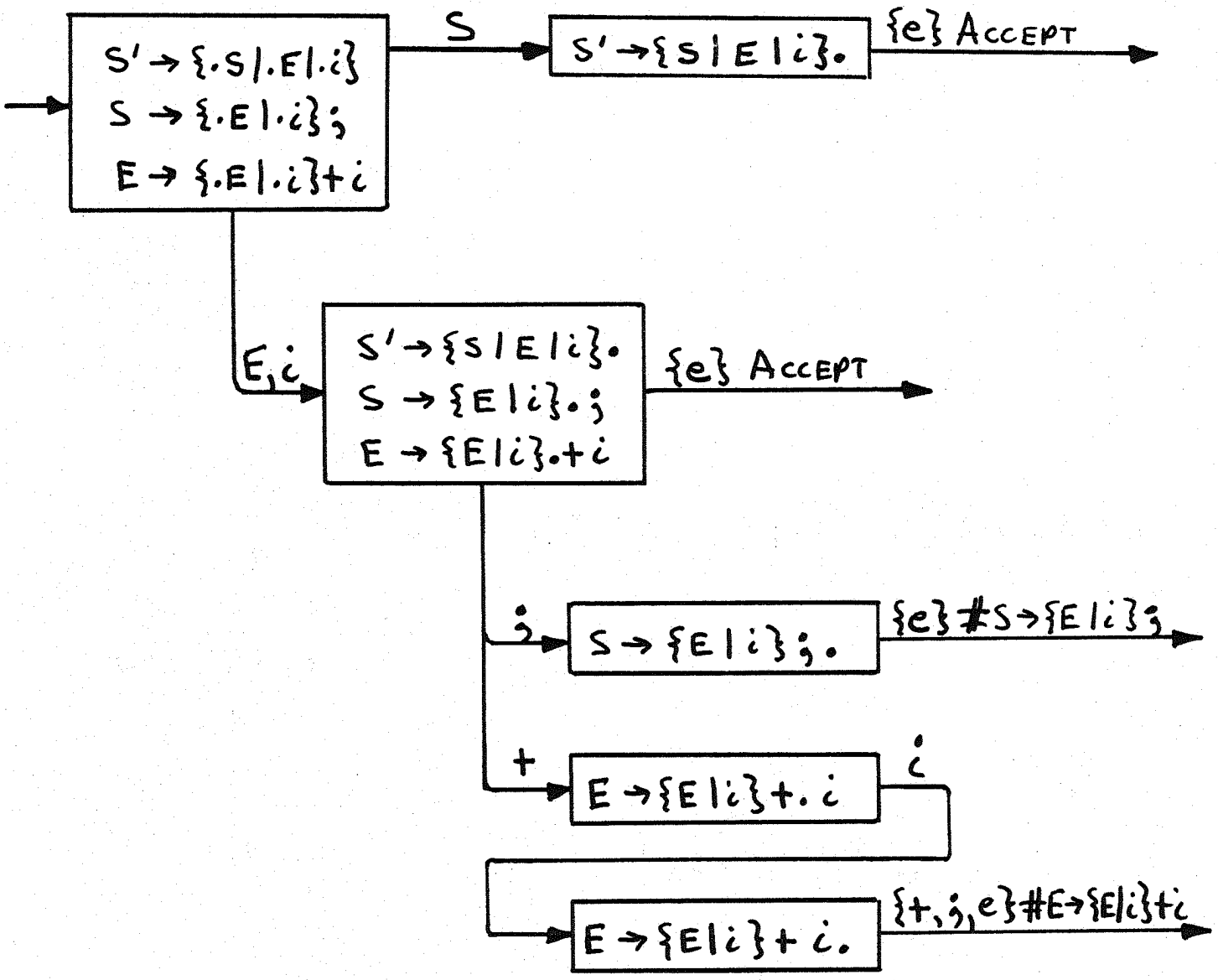
A Decomposition of the Aho and Ullman Algorithm

Figure 6



T6 as Modified from T1 Using the Two-Step Aho and Ullman Algorithm + Defaults

Figure 7



LALR(1) Tables T7 for
G2: (S´->{S|E|i}), S->{E|i};, E->{E|i}+i.

Figure 8

Carleton University
School of Computer Science

Bibliography of SCS reports

SCS-TR-1 THE DESIGN OF CP-6 PASCAL
Jim des Rivieres and Wilf R. Lalonde, June 1982.

SCS-TR-2 SINGLE PRODUCTION ELIMINATION IN LR(1) PARSERS: A SYNTHESIS
Wilf R. Lalonde, June 1982.

SCS-TR-3 A FLEXIBLE COMPILER STRUCTURE THAT ALLOWS DYNAMIC PHASE ORDERING
Wilf R. Lalonde and Jim des Rivieres, June 1982.

SCS-TR-4 A PRACTICAL LONGEST COMMON SUBSEQUENCE ALGORITHM FOR TEXT COLLATION
Jim des Rivieres, June 1982.