# PARALLEL DECOMPOSITION OF LR(k) PARSERS[*]
(Extended Abstract)

David B. Benson[**] and Ralph D. Jeffords[***]

Computer Science Department
Washington State University
Pullman, Washington 99164/USA

## Introduction

The practical goal of our work is to produce highly compact tables for table-driven parsers. The motivation lies in the recent hardware developments of fast ROM and microcomputers. The theoretical goal is a better understanding of the algebra--particularly algebraic decomposition--of syntax and parsing machines [4,5,6]. This paper considers LR(k) parsers, but we firmly believe the same techniques will work at least as well for the other standard table-drive techniques (LL(k), precedence, etc.).

Series decomposition of parsers is standard. One usually has a lexical analysis whose output is input to the syntactic analysis. Also, Lorenjak's LR(k) analysis [13] resembles a serial decomposition although the results are recombined before run-time. Parallel decomposition has been explored for sequential machines [9,11], but not for parsers, although the motivation for this work stemmed from reading [14,15]. The notion of parallelism used in [10] requires having the entire input available at once, a different notion from parallel decomposition. It appears that the technique in [10] and the parallel decomposition presented here could be combined to provide a parallel decomposition of parsers working on distinct input segments.

To have a parallel decomposition of a language, it suffices to have a monofunctor

$$f : \underset{\sim}{S} \to \underset{\sim}{S}_1 \times \underset{\sim}{S}_2$$

from the syntax category, S, of the original grammar to the product of the syntax categories for the "quotient" grammars $\underset{\sim}{S}_1$ and $\underset{\sim}{S}_2$ [5,6]. (Although parallel decomposition into only two components is covered in this paper, the generalization to more than two components is straightforward.) However, for our explorations in this paper, we obtain tractability by additional restrictions on f:

$$f = (h_1,h_2), \text{ with } h_i : \underset{\sim}{S} \to \underset{\sim}{S}_i,\ i = 1,2,$$

where each $h_i$ is a length preserving homomorphism on strings and carries rules bijectively to rules. Thus, $h_i$ preserves the length of derivations and the resulting system can be viewed as encoding each symbol in the original alphabet $V = N \cup \Sigma$ as a pair of

[*] Research supported in part by NSF Grant MCS 73-03497A01.

[**] Current address: Computer Science, University of Colorado, Boulder, CO 80309/USA

[***]Current address: Computer Science, School of Engineering, University of Mississippi, University, MS 38677/USA

of symbols in $V_1 \times V_2$.

There are two approaches to constructing parallel LR(k) parsers:

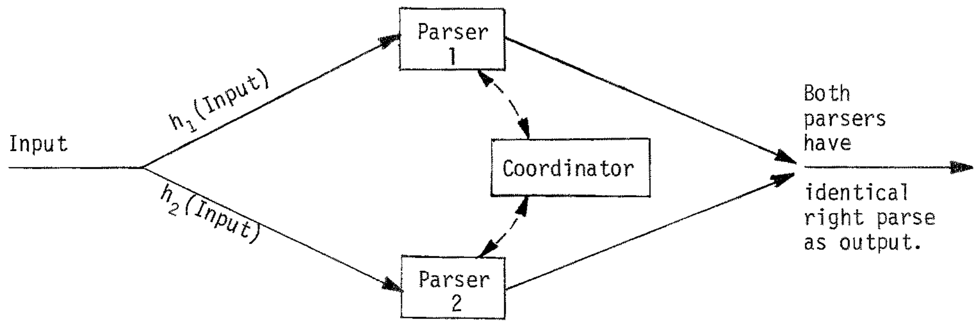
(1) Decomposition of the language (grammar).

A modification of the LR(k) parser construction algorithm is used to construct a parser for each of the "quotient" grammars, $G_1$ and $G_2$, of the respective decomposed syntax categories, $\underset{\sim}{S}_1$ and $\underset{\sim}{S}_2$.

(2) Decomposition of the parser.

Column and (optionally) row mergers are applied in two different fashions to the LR(k) driving matrices to obtain the two component parsers.

In either case the resulting component parsers are essentially mergers of the columns and rows of the ordinary LR(k) parsing matrices (with possible duplication). Standard coding results then lead one to believe that the number of columns needed in LR(k) driving matrices should substantially decrease at least in the case of parsing action matrices, which are usually quite sparse. There is a cost however. The parallel parser will, on a uniprocessor, run at about one-third of the speed.



The two parsers do not run independently in contradistinction to the case of sequential machines. At each step they must be coordinated which requires about as much time as each parser step.

## Grammar Decomposition Approach to a Parallel LR(k) Parser

The details consist of creating such a parallel parser which runs nondeterministically and then establishing the additional restrictions necessary to obtain determinism once again.

Given an LR(k) grammar $G = (N,\Sigma,P,S)$ where P is an indexed set of productions with indices $1,\ldots,r,\ldots,|P|$, and surjective $h_i: V \rightarrow V_i$, $i = 1,2$, let

$$G_i = (N_i, \Sigma_i, P_i, S_i),\ i = 1,2$$

be grammars such that

(1) $N_i = \{h_i(A) \mid A \rightarrow \beta \in P\}$

(2) $\Sigma_i = h_i(V) - N_i$

(3) $V_i = N_i \cup \Sigma_i$,

(4) $P_i$ is indexed by $1, \ldots, |P|$ with productions

$h_i(r\colon A \to \beta) = r\colon h_i(A) \to h_i(\beta)$,

(5) $S_i = h_i(S)$ .

Then $h_i$ extends to a syntax functor from $\underset{\sim}{S}(G)$ to $\underset{\sim}{S}(G_i)$ [5,6]. We now require $(h_1, h_2)\colon V_1 \to V_1 \times V_2$ injective to obtain the following fundamental

Lemma 1: $\alpha \Rightarrow^{\pi} \beta$ if and only if $h_1(\alpha) \Rightarrow^{\pi} h_1(\beta)$ and $h_2(\alpha) \Rightarrow^{\pi} h_2(\beta)$ , where $\pi$ is the right parse of $\beta$ to $\alpha$. □

Thus compatible derivations in the image syntaxes uniquely determine the original derivation. For notational convenience, let $\gamma_i$ denote $h_i(\gamma)$ for each string $\gamma$.

Let $T_i$ be LR(k) parsers for $G_i$--it doesn't matter that the $G_i$ are not necessarily LR(k), just construct the tables for a nondeterministic k-lookahead parser using the LR construction methods. Each row $s_i$ of the $T_i$ matrices consists of the parsing action entry $f_i(s_i,-)$ and the goto entry $g_i(s_i,-)$, [2]. For lookahead u the entry $f_i(s_i,u_i)$ is a <u>set</u> of actions since $T_i$ is nondeterministic. (This is the same type of construction used in [1] for constructing parsers for "ambiguous grammars".) The coordination consists of taking $f_1(s_1,u_1) \cap f_2(s_2,u_2)$ when $T_i$ is in state $s_i$, i = 1,2. The resulting set of actions is used by both parsers. For various reasons, $|f_1(s_1,u_1) \cap f_2(s_2,u_2)| \geq 1$ in some cases and the parallel parser runs nondeterministically. We can show that it is correct but lacks immediate error detection, as defined in [4].

To obtain determinism and immediate error detection, additional information and restrictions are required. To explain these, some notation is desirable. Let $W = \{\gamma \mid V_k^G(\gamma) \neq \emptyset\}$ be the set of viable prefixes. Define $\gamma \doteq \beta$ if and only if $V_k^G(\gamma) = V_k^G(\beta)$. Now let $\Gamma = W/\doteq$. $\Gamma$ will be used to index the sets of valid items $V_k^G$. Define the domain of f (parsing action function) to be $\Gamma \times \Sigma^{*k}$. Define $g\colon \Gamma \times V \to \Gamma$ to be $g([\gamma],X) = [\beta]$ if and only if $\mathrm{GOTO}(V_k^G(\gamma),X) = V_k^G(\beta)$. With this notation the rows of the parsing matrix are indexed by $\Gamma$. Given $\gamma_1 = h_1(\gamma)$ and $\gamma_2 = h_2(\gamma)$ the original string $\gamma$ is uniquely determined due to the injectivity of $(h_1,h_2)\colon V \to V_1 \times V_2$.

To ensure that Lemma 1 is satisfied, the construction algorithm for sets of valid items (Algorithm 5.8 in [2]) is modified to rule out certain items that are never associated with any compatible derivations. Furthermore, additional information consisting of original parser state (ST part) and lookahead (LA part) must be attached to each parsing action within an entry. This gives parsing action entries in the matrix $f_i$ of the form

$$f_i([\gamma_i],u_i) = \bigcup \{(f([\beta],v),[\beta],v)\},$$

where $[\beta]$ is the ST part and v is the LA part. The union is over lookaheads v such that $v_i = u_i$ and over states $[\beta]$ of the canonical LR(k) parser related in a rather complicated way to the component parser state $[\gamma_i]$. Even with this added information and the modified construction algorithm the component parsers may not operate deterministically. The details are given in Section 4.2 of [12].

## Direct Parallel Decomposition of the Ordinary LR(k) Parser

As in the decomposed grammar parallel parser construction method, original parser state and lookahead must be associated with each parsing action value.

Definition: $\hat{f}([\gamma],u) \equiv (f([\gamma],u),[\gamma],u)$.

This isn't bad as it seems since the $\hat{f}$ triples can be nicely encoded. See the example at the end of the paper.

Similarly, the goto requires the original state to maintain determinism in the general case--the symbol X is not required since it is uniquely determined by $g([\gamma],X)$ and $[\gamma]$.

Definition: $\hat{g}([\gamma],X) \equiv (g([\gamma],X),[\gamma])$.

The columns of the parsing action (goto) matrices of the ordinary LR(k) parser are merged according to the partition of V induced by the respective $h_i$ to obtain the respective columns of the component parsing action (goto) matrices. The merger of the rows (which incidentally defines the "states" of the component parsers) may be done in any fashion which preserves the correct determination of the state of the ordinary parallel parser for each reachable component-state pair.

By proper row merging (with duplication) one obtains exactly the parser of the preceding section (in this case the ST part of the goto function is unnecessary). If no row mergers are performed then each component parser has the same states as the ordinary LR(k) parser. The simple guaranteed approach employed in this section is to let J be any system of representatives of $\Gamma$ and to merge rows $\alpha \in J$ and $\beta \in J$ if $\alpha_i = \beta_i$.

With these restrictions the parsing action matrix and goto matrix for each component of the parallel parser are defined as the merge of all rows and columns (of the ordinary parsing action and goto matrices, respectively) whose indices are identical under the component parser's homomorphism.

Definition:

$$f_i([\gamma_i],u_i) \equiv \{\hat{f}([\beta],w) \mid \beta \in J,\ \beta_i = \gamma_i,\ w_i = u_i,\ f([\beta],w) \neq \underline{\text{error}}\}\ ,$$
$$g_i([\gamma_i],X_i) \equiv \{\hat{g}([\beta],Y) \mid \beta \in J,\ \beta_i = \gamma_i,\ Y_i = X_i,\ g([\beta],Y) \neq \underline{\text{error}}\}\ ,$$

Lemma 2: (Determinism under intersection)

$$f_1([\gamma_1],u_1) \cap f_2([\gamma_2],u_2) = (f([\gamma],u),\ [\gamma],\ u),$$
$$g_1([\gamma_1],X_1) \cap g\ ([\gamma_2],X_2) = (g([\gamma],X),\ [\gamma]). \ \square$$

The parallel parsing algorithm with each component parser i employing its respective $f_i$ and $g_i$ matrices is the following.

Algorithm 3: (from Aho and Ullman [2, p.375])

Input: A string $z \in \Sigma^*$ to be parsed. Initially each parser i has state $[\gamma]$ (the initial null string state symbol) on its stack and has $z_i$ as input.

Output: The right parse of z if $z \in L(G)$, error otherwise.

Method: Do until accept or error;

(1) Let $[\gamma_i]$ be respective states on top stacks.

(2) Determine respective lookahead symbols $u_i$ from next k input symbols of $z_i$.

(3) Compute $f_i([\gamma_i],u_i)$ and extract $f([\gamma],u)$ from it.

(a) If $f([\gamma],u)$ = shift then the next input symbol (say $a_i$) is read and pushed onto stack i. Now compute $\bigcap g_i([\gamma_i],a_i) = ([\gamma a],[\gamma])$ and push $[\gamma_i a_i]$ onto stack i. Return to step (1).

(b) If $f([\gamma],u)$ = reduce r and r: $A \rightarrow \alpha$ then $2|\alpha|$ symbols are popped from stack i and r is written once to the common output. Let $[\beta_i]$ be the new state on top of stack i. Compute $\bigcap g_i([\beta_i],A_i) = ([\beta A],[\beta])$ and push $[\beta_i A_i]$ onto stack i. Return to step (1).

(c) If $f([\gamma],u)$ = accept $\equiv$ reduce 0 then announce success.

(d) If $f_i([\gamma_i],u_i) = \emptyset$, announce error.

The parallel parser simulates the ordinary LR(k) parser,

Theorem 4:

The ordinary LR(k) parser will be in configuration $(\gamma^0 X^1 \gamma^1 \ldots X^j \gamma^j,\ x,\ \pi)$ if and only if the parallel component parsers are in the respective configurations $(\gamma_i^0 X_i^1 \gamma_i^1 \ldots X_i^j \gamma_i^j,\ x_i,\ \pi)$, i = 1,2 (the equivalence class brackets, [,], have been dropped for clarity). $\square$

## An Example of a Parallel LR(1) Parser

The example is based on a partially simplified parser from Anderson et al., [3]. Figure 1 is taken from their paper. Figure 2 gives the parsing action (only shift and error entries), Figure 3 the corresponding terminal goto matrix and the nonterminal goto matrix, and each of Figures 2 and 3 show three vectors associated with the "states" of the parser:

(PA) Applicable row of parsing action or terminal goto for the parser states 1 through 17.

(PR) Production associated with a state.

(NG) Applicable row of nonterminal goto matrix for the given state.

In the two goto matrices, positive numbers, n, are next-states and negative numbers, -n, are collapsed reduce rows, signifying a reduction by the production |-n|. Elimi-

| Parser State | Parsing Action | Goto |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 1 | 2 |
| 3 | 2 | 1 |
| 4 | 2 | 2 |
| 5 | 2 | 3 |
| 6 | 2 | 4 |
| 7 | 3 | 1 |
| 8 | 4 | 1 |
| 9 | 5 | 3 |
| 10 | 6 | |
| 11 | 7 | |
| 12 | 8 | |
| 13 | 9 | |
| 14 | 10 | |
| 15 | 11 | |
| 16 | 12 | |
| 17 | 13 | |

| | id | + | ( | ) | * | if | then | or | else | := | $\perp$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 10 | | | | | 8 | | | | | |
| 2 | *10 | | 4 | | | | | | | | |
| 3 | | | | | | | | | 2 | | -14 |
| 4 | *12 | | | | | | | | | | |
| 5 | 10 | | | | | | | | | | |
| 6 | | | | | | | | | | 3 | |
| 7 | | | | | | | 9 | 14 | | | |
| 8 | | 5 | | | | | | | -3 | | -3 |
| 9 | | 5 | | | 6 | | | | -3 | | -3 |
| 10 | *11 | | | | | | | | | | |
| 11 | | 5 | | *9 | | | | | | | |
| 12 | | 5 | | *9 | 6 | | | | | | |
| 13 | | -6 | | -6 | 6 | | | | -6 | | -6 |

Parsing Action

| | S | A | I | E | T | P | B | L |
|---|---|---|---|---|---|---|---|---|
| 1 | -0 | -0 | -0 | 12 | 13 | 13 | 11 | -4 |
| 2 | -13 | -13 | -13 | 15 | 16 | 16 | | |
| 3 | | 7 | | | 17 | 17 | | |
| 4 | | | | | | -8 | | |

Goto

Figure 1.

nation of LR(0) reduce states, partial chain derivation elimination, and reordering after merger of rows has already been done. The important point is that any simplifications not affecting the number of columns may be performed before applying the parallel decomposition.

All reduce actions (represented by production numbers) have been deleted from the parsing action and placed in the PR vector since there is at most one production associated with a given state and each nonshift entry for that state may default to that particular reduction. In general, if the grammar is weak precedence (all weak precedence grammars are SLR(1) [8, p.209]) and no chain derivation elimination has been performed, then each state of the canonical SLR(1) parser will have at most one type of reduction, [12].

| Parser State | PA | PR | NG |
|---|---|---|---|
| 1 | 1 | | 1 |
| 2 | 1 | | 2 |
| 3 | 2 | | 1 |
| 4 | 2 | | 2 |
| 5 | 2 | | 3 |
| 6 | 2 | | 4 |
| 7 | 3 | 14 | 1 |
| 8 | 4 | | 1 |
| 9 | 5 | | 3 |
| 10 | 6 | | |
| 11 | 7 | | |
| 12 | 8 | 3 | |
| 13 | 9 | 3 | |
| 14 | 10 | | |
| 15 | 11 | | |
| 16 | 12 | | |
| 17 | 13 | 6 | |

| Parser State | id | + | ( | ) | * | if | then | or | else | := |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | S | | | | | S | | | | |
| 2 | S | | S | | | | | | | |
| 3 | | | | | | | | | S | |
| 4 | S | | | | | | | | | |
| 5 | S | | | | | | | | | |
| 6 | | | | | | | | | | S |
| 7 | | | | | | | S | S | | |
| 8 | | S | | | | | | | | |
| 9 | | S | | | S | | | | | |
| 10 | S | | | | | | | | | |
| 11 | | S | | S | | | | | | |
| 12 | | S | | S | S | | | | | |
| 13 | | | | | S | | | | | |

Parsing Action

Figure 2

Normally one combines the terminal portion of the goto matrix with the parsing action [1, p.108] but this is not desirable here since we want just one symbol in the parsing action matrix to keep the example simple. Furthermore, the entire goto matrix can be merged compactly without regard to blank entries, which are inaccessible [2, p.589].

The parallel parser is constructed by merging just terminal columns of the parsing action and terminal goto matrix according to $h_1, h_2$ given in Figure 4. Merging rows can also be considered but makes the construction much more difficult. Without merged rows the ST parts in both parsing action and goto matrices are superfluous.

The parsing action is sparse enough that the LA parts of $\hat{f}([\gamma_i], u_i)$ are unnecessary in all but row 12 in Figure 4. For row 12 a secondary shift symbol $\hat{S}$ is sufficient to encode the necessary information from the LA part required for unique determination of the original parsing action.

In row 12,

$f_1(12,2) \cap f_2(12,2) = S$ ((2,2) encodes ")")

while

$f_1(12,1) \cap f_2(12,2) = \emptyset$ ((1,2) encodes "else")

| Parser State | PA | PR | NG |
|---|---|---|---|
| 1 | 1 | | 1 |
| 2 | 1 | | 2 |
| 3 | 2 | | 1 |
| 4 | 2 | | 2 |
| 5 | 2 | | 3 |
| 6 | 2 | | 4 |
| 7 | 3 | 14 | 1 |
| 8 | 4 | | 1 |
| 9 | 5 | | 3 |
| 10 | 6 | | |
| 11 | 7 | | |
| 12 | 8 | 3 | |
| 13 | 9 | 3 | |
| 14 | 10 | | |
| 15 | 11 | | |
| 16 | 12 | | |
| 17 | 13 | 6 | |

| Parser State | id | + | ( | ) | * | if | then | or | else | := |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 10 | | | | | 8 | | | | |
| 2 | -10 | | 4 | | | | | | | |
| 3 | | | | | | | | | 2 | |
| 4 | -12 | | | | | | | | | |
| 5 | 10 | | | | | | | | | |
| 6 | | | | | | | | | | 3 |
| 7 | | | | | | | 9 | 14 | | |
| 8 | | 5 | | | | | | | | |
| 9 | | 5 | | | 6 | | | | | |
| 10 | -11 | | | | | | | | | |
| 11 | | 5 | | -9 | | | | | | |
| 12 | | 5 | | -9 | 6 | | | | | |
| 13 | | | | | 6 | | | | | |

Terminal Goto

| | S | A | I | E | T | P | B | L |
|---|---|---|---|---|---|---|---|---|
| 1 | -0 | -0 | -0 | 12 | 13 | 13 | 11 | -4 |
| 2 | -13 | -13 | -13 | 15 | 16 | 16 | | |
| 3 | | 7 | | | 17 | 17 | | |
| 4 | | | | | | -8 | | |

Nonterminal Goto

Figure 3.

signifying error.

As the terminal goto matrix is sparse, it may be merged using the indirect addressing methods in [3]. The result is given in Figure 4. The five entries with multiple values should be kept in an auxiliary table. In other cases it may be preferable to use the parallel decomposition only for the parsing action and run all subparsers from a single goto matrix.

The parsing action decomposition is essentially a decomposition-encoding done for each separate row. Briefly state, a decomposition-encoding of a binary-valued vector $(V_0, V_1, \ldots, V_{n-1})$ is two vectors $F_1$ and $F_2$ such that

$$\bigvee_w (F_1(h_1(j)) \wedge_w F_2(h_2(j))) = V_j$$

($\bigvee_w$ represents a w-input OR function giving a single bit result; $\wedge_w$ represents w

| ROW NUMBER | $h_1$ | | | | $h_2$ | | |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 0 | 1 | 2 |
| 1 | S | | | | S | | S |
| 2 | S | | | | S | S | |
| 3 | | S | | | | | S |
| 4 | S | | | | S | | |
| 5 | S | | | | S | | |
| 6 | | | S | | S | | |
| 7 | | | | S | S | S | |
| 8 | | | S | | | S | |
| 9 | | S | S | | | S | |
| 10 | S | | | | S | | |
| 11 | | | S | | | S | S |
| 12 | | Ŝ | S | | | SŜ | S |
| 13 | | S | | | | S | |

Parsing Action

| ROW NUMBER | $h_1$ | | | | $h_2$ | | |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 0 | 1 | 2 |
| 1 | 10,8 | | | | 10 | | 8 |
| 2 | -10,4 | | | | -10 | 4 | |
| 3 | | 2 | | | | | 2 |
| 4 | -12 | | | | -12 | | |
| 5 | 10 | | | | 10 | | |
| 6 | | | 3 | | 3 | | |
| 7 | | | | 9,14 | 9 | 14 | |
| 8 | | | 5 | | | 5 | |
| 9 | | 6 | 5 | | | 5,6 | |
| 10 | -11 | | | | -11 | | |
| 11 | | | 5,-9 | | | 5 | -9 |
| 12 | | 6 | 5,-9 | | | 5,6 | -9 |
| 13 | | 6 | | | | 6 | |

Terminal Goto

| $h_1$ | | | | $h_2$ | | |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 0 | 1 | 2 |
| 10,8 | | | | 10 | | 8 |
| -10,4 | 2 | | | -10 | 4 | 2 |
| -12 | 6 | 5,-9 | | -12 | 5,6 | -9 |
| | | 3 | | 3 | | |
| | | | 9,14 | 9 | 14 | |
| -11 | | | | -11 | | |

Terminal Goto (Merged)

| | id | + | ( | ) | * | if | then | or | else | := | $\perp$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $h_1$ | 0 | 2 | 0 | 2 | 1 | 0 | 3 | 3 | 1 | 2 | 1 |
| $h_2$ | 0 | 1 | 1 | 2 | 1 | 0 | 0 | 1 | 2 | 0 | 0 |

Figure 4.

parallel 2-input AND functions). This encoding is nearly as compact as simply packing bits into a word but does not require any shift operation to access a desired bit. For further details see [7,12]. A second decomposition-encoding or packing may be performed to merge the rows into a more compact form. Since LR(k) parsers for actual programming languages allow a shift for only a very few lookaheads in a given state, a parallel decomposition should introduce a relatively small number of additional shift symbols like $\hat{S}$. By trying various sets of homomorphisms with the injective property it may be possible to minimize the number of additional shift symbols required (equivalently the number of bits required in the decomposition-encoding). A heuristic procedure for testing such homomorphisms in a practical way is left for further investigation. For example, it is possible by a clever choice of homomorphisms to encode the parsing action of Figure 2 using only a single shift symbol, as shown in Figure 5. This gives a total parsing action storage requirement of $13 \times 7 = 91$ bits as compared to the original matrix of $13 \times 10 = 130$ bits. In general, it should be possible to encode the entire parsing action matrix in considerably less space than required by other methods.

| | $\bar{h}_1$ | | | | $\bar{h}_2$ | | |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 0 | 1 | 2 |
| 1 | S | | | | S | | S |
| 2 | S | | | | S | S | |
| 3 | | | S | | | | S |
| 4 | S | | | | S | | |
| 5 | S | | | | S | | |
| 6 | | S | | | | | S |
| 7 | | | S | S | | S | |
| 8 | | S | | | S | | |
| 9 | | S | S | | S | | |
| 10 | S | | | | S | | |
| 11 | | S | | S | S | | |
| 12 | | S | S | S | S | | |
| 13 | | | S | | S | | |

Parsing Action

| | id | + | * | ) | ( | $\perp$ | then | or | if | := | else |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\bar{h}_1$ | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 |
| $\bar{h}_2$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 |

Figure 5.

## Conclusions

In conclusion, parallel decomposition appears to offer economies of space for LR(k) parsing. However, this must still be checked in cases of practical interest. As a theoretic tool, parallel decomposition increases our understanding of the nature of parsing. It has already led to other simplifications of the parsing process as reported in [7].

## References

[1] Aho, A. V., and Johnson, S. C. LR parsing. ACM Computing Surveys 6,2 (June 1974), 99-123.

[2] Aho, A. V., and Ullman, J. D. The Theory of Parsing, Translating, and Compiling, Vol. I and II. Prentice-Hall, Englewood Cliffs, N. J., 1972, 1973.

[3] Anderson, J., Eve, J., and Horning, J. J. Efficient LR(1) parsers. Acta Informatica 2,1 (1973), 12-39.

[4] Benson, D. B. An abstract machine theory for formal language parsers. Acta Informatica 3,2 (1974), 187-202.

[5] Benson, D. B. The basic algebraic structures in categories of derivations. Information and Control 28,1 (May 1975), 1-29.

[6] Benson, D. B. Some preservation properties of normal form grammars. SIAM J. Computing 6,2 (1977), to appear.

[7] Benson, D. B., and Jeffords, R. D. A vector encoding technique applicable to tabular parsing methods, submitted to Comm. ACM.

[8] DeRemer, F. L. Practical translators for LR(k) languages, Tech. Report MAC TR-65, Project MAC, M.I.T., Cambridge, Mass., Oct. 1969.

[9] Eilenberg, S. Automata, Languages, and Machines, Vol. B. Academic Press, New York, 1976.

[10] Fischer, C. On parsing context free languages in parallel environments. Cornell U. TR75-237 (1975).

[11] Hartmanis, J. and Stearns, R. E. Algebraic Structure Theory of Sequential Machines. Prentice-Hall, Englewood Cliffs, N.J., 1966.

[12] Jeffords, R. D. Algebraic decomposition of parsers. Ph.D.Diss., Washington State Univ., Pullman, Wash., Feb. 1977.

[13] Korenjak, A. J. A practical method for constructing LR(k) processors. Comm. ACM 12,11 (Nov. 1969), 613-623.

[14] Schnorr, C. P., and Walter, H. Pullbackkonstruktionen bei Semi-Thuesystemen. E.I.K. 5,1 (1969), 27-36.

[15] Walter, H. Verallgemeinerte Pullbackkonstruktionen bei Semi-Thuesystemen und Grammatiken. E.I.K. 6,4 (1970), 239-254.