Modelling Morphological Parsers and Grammars

C. Creider, J. Hankamer
and D. Wood

Technical Report # 372
1993
Modelling Morphological Parsers and Grammars

Chet Creider†   Jorge Hankamer‡   Derick Wood§

Abstract

Modelling the morphological structure of natural languages in terms of a nondeterministic finite-state automaton is shown to be insufficient in its handling of some common natural language phenomena. We show that a two-tape nondeterministic automaton is capable of handling these phenomena and discuss the operation of a parser that implements this model. The modelling of the parser is improved by the specification of a new type of automaton, the preset two-head automaton, which we show to be equivalent in expressive power to a linear context-free grammar.

1 Introduction

We attempt to determine an appropriate model for the morphological structure of natural languages. In so doing, we hope to answer, in a preliminary way, the question of what kind of language is required to specify morphological structure. Based on this, we suggest ways of focussing research in

---

*This work was partially carried out under grants from the Social Sciences and Humanities Research Council of Canada, the Natural Sciences and Engineering Research Council of Canada and the Information Technology Research Centre of Ontario

†Department of Anthropology, University of Western Ontario, London, Ontario N6A 5C2 CANADA, creider@csd.uwo.ca.

‡Board of Studies in Linguistics, University of California at Santa Cruz, Santa Cruz CA 95064 USA, hank@ling.ucsc.edu.

§Department of Computer Science, University of Western Ontario, London, Ontario N6A 5B7 CANADA, dwood@csd.uwo.ca.
a domain which has been characterized by a very great diversity of formal approaches. In Section 2, we present a popular mathematical model for morphological structure. In Section 3, we establish the insufficiency of this model and develop, in Section 4, what we claim is a more appropriate one. We present, in Section 5, a particular instantiation of a morphological parser that embodies what we have argued is a reasonable formal model. The formal model is itself improved in Section 6 where we introduce a new type of automaton, the preset two-head automaton. In Section 7, we discuss a variety of relevant linguistic evidence, and, in Section 8, we discuss the kinds of representations for morphological structure that have been envisaged by linguists. The equivalence of preset two-head automata and linear context-free grammars is proved in the Appendix.

Our general approach is familiar to both linguists and computer scientists. The former were informally introduced to the hierarchy of languages by Chomsky (1957) and numerous authors since then have explored this hierarchy. Correspondences between mathematical models, such as automata, and language types have been established by computer scientists and are important, among other reasons, because it is often possible to proceed to a machine implementation of a particular model. (1) gives a matching of these equivalences. See Wood’s text (1987) for extensive discussion and amplification.

<table>
<thead>
<tr>
<th>Language Family</th>
<th>Characterizing Automata Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular languages</td>
<td>Finite-state automata</td>
</tr>
<tr>
<td>Context free languages</td>
<td>Nondeterministic pushdown automata</td>
</tr>
<tr>
<td>Recursive</td>
<td>Decision-making Turing machines</td>
</tr>
<tr>
<td>Recursively enumerable</td>
<td>Turing machines</td>
</tr>
</tbody>
</table>

2 Models of morphological parsers

Although linguists have gone for the higher reaches of the grammar hierarchy in their treatments of morphology (see section 8 below), the authors of morphological parsers have not felt the need to go above the bottom-most level. As noted by Hankamer (1989), three recent approaches to morphological parsing share a representation of the morphotactics (specification of morpheme co-occurrence restrictions) in terms of a finite-state transition network (Kaspar and Weber (1982a; 1982b), Weber, Black and McConnel 1988),
Koskenniemi (1983) and Hankamer (1986)). Hankamer's keçi (1986; 1988) implements this model closely.¹

A nondeterministic finite-state automaton (Wood (1987, 114)) is a model

\[
(2) \quad M = (Q, \Sigma, \delta, s, F),
\]

where

- \( Q \) is a set of state symbols,
- \( \Sigma \) is an alphabet of input symbols,
- \( \delta \subseteq Q \times \Sigma \times Q \) is a transition relation,
- \( s \) in \( Q \) is a start state, and
- \( F \subseteq Q \) is a set of final states.

For a triple \((p, a, q)\) in \( \delta \), \( p \) corresponds to the current state, \( a \) to the current input symbol, and \( q \) to the next state. It is convenient to consider \( \delta \) as a function that gives state sets, defining \( \delta : Q \times \Sigma \to 2^Q \) by (3):

\[
(3) \quad \text{For all } p \text{ in } Q \text{ and for all } a \text{ in } \Sigma: \delta(p, a) = \{ q : (p, a, q) \text{ is in } \delta \}.
\]

The following example in Nandi, a Southern Nilotic language spoken in Western Kenya, illustrates the operation of this automaton.²

\[
(4) \quad Q = \{00, N0, N1, N2, N3, WW\},
\]

\[\Sigma = \{sot, oon, i, cu, cuuk\},\]

\[\delta(00, sot) = N0,\]

\[\delta(N0, oon) = N1,\]

\[\delta(N1, i) = N2,\]

\[\delta(N2, cu) = N3,\]

\[\delta(N3, cuuk) = WW,\]

\[s = \{00\},\]

\[F = \{WW\}.\]

¹The Kaspar and Weber model (AMPLE), however, has a number of other mechanisms for controlling morphotactic relations and is probably too eclectic to categorize neatly. Cf. Hankamer and Black (1993).

²sotoomicuuk 'these gourds of ours': sot 'gourd', oon 'plural', i 'theme', cu 'proximate plural demonstrative', cuuk 'plural possessed + plural possessor'
The nondeterministic finite-state automaton is conceived of as a machine that consists of an input tape of cells, each containing one morpheme, a read head, and a finite-state control with knowledge of the transition function and the current state (Wood (1987, 99)). We can depict the operation of the automaton in terms of the following diagram:

(5)

\[
\begin{array}{cccccc}
\text{sot} & \text{oon} & \text{i} & \text{cu} & \text{cuuk} \\
\end{array}
\]

\[
\text{Finite} \\
\text{control}
\]

We can also represent the nondeterministic finite-state automaton as a labelled directed graph in which the nodes are states and the labelled edges represent the transition function:

(6)

\[
\begin{array}{cccc}
00 & \rightarrow & \text{N0} \rightarrow & \text{N1} \\
\text{sot} & \text{oon} & \text{i} & \text{cu} \\
\text{cuuk} & \rightarrow & \text{N2} \rightarrow & \text{N3} \rightarrow & \text{WW} \\
\end{array}
\]

As can be seen from this example, we associate states with morphological categories and after the identification of the root, suffix morphemes sanction transitions between categories. The model must be nondeterministic because of the need for free jumps between states and for recursion.

3 Insufficiency of the model

Despite agreement among the designers of morphological parsers that efficient and practical parsers can be written using the nondeterministic finite-state automaton model, it has been realized for some time that the model is incapable of handling, or can handle awkwardly at best, a large class of
natural language phenomena. To see this, consider the problem of parsing the Nandi word, ceem-naandi-iin-i-cu ‘these Nandi women’. The root here is naandi, but it alone is incapable of taking the pluralizing suffix -iin. The feminine-forming prefix ceem- must first be affixed to the root in order for -iin to be suffixed. A similar situation in English is discussed in Sproat (1992, 90–92). Consider the English word, en-joy-able. Joy is an adjective, and as the ungrammaticality of *joyable shows, it is not possible to suffix -able to it directly. The verb-deriving prefix en- must precede joy in order to add -able as a suffix. Sproat observes that in order to handle this example within a finite-state model, it is necessary to have two separate dictionaries, each containing joy, where one dictionary contains the case where the prefix en- has not been used and in which the suffix -able is not allowed to follow it, and the other contains entries with a different set of states which allows the sequence of transitions en + joy + able. We would need an automaton such as the following:

\[(7) \quad Q = \{00, V0, V1, WW\},\]
\[\Sigma = \{en, joy1, joy2, able\},\]
\[\delta(00, joy1) = WW,\]
\[\delta(00, en) = V0,\]
\[\delta(V0, joy2) = V1,\]
\[\delta(V1, able) = WW,\]
\[s = \{00\},\]
\[F = \{WW\}.\]

Depicted as a state diagram:

\[(8)\]

We emphasize that the problem here, that of a dependency between a prefix string and a suffix string, is extremely common. While a number of
natural languages are exclusively suffixing or nearly so, and there are no exclusively prefixing languages, a very large number of languages make use of both prefixation and suffixation and interdependencies between the two affix sets are common.

4 An improved model

Suppose, however, that we free ourselves of the requirement to proceed in a strictly left-to-right fashion. In the general case, after identifying the root, we have a string (possibly the null string) of prefix morphemes followed by a root followed by a string (possibly null) of suffixes:

\[(9) \quad p_n + \cdots + p_1 + [\text{root}] + s_1 + \cdots + s_n.\]

Separating the prefix and suffix strings, we obtain:

\[(10) \quad p_n + \cdots + p_1\]
\[\text{and}\]
\[s_1 + \cdots + s_n.\]

Alternatively, as shown in (11), we can picture an automaton that has two tapes and two read heads.

This two-tape nondeterministic automaton, as it is called, reads one tape that consists of the prefix string and another tape that consists of the suffix string. The read heads on the two tapes move independently, and each transition applies to only one tape. If there is no transition for one tape, then the read head associated with that tape does not move. When both tapes have been processed, if the automaton is in a final state, then the parse is successful. This automaton will handle the general problem of prefix/suffix interdependencies. Hopcroft and Ullman (1979, 74) give the following description of a two-tape nondeterministic finite-state automaton:

In a two-tape FA each state is designated as reading tape 1 or tape 2. A pair of strings \((x, y)\) is accepted if the FA, when presented with strings \(x\) and \(y\) on its respective tapes, reaches a final state with the tape heads immediately to the right of \(x\) and \(y\).
Two-tape nondeterministic finite-state automata were investigated by Rosenberg (1967), who proves that there is an equivalence between them and linear context-free grammars. A context-free grammar is linear if there is at most one nonterminal symbol on the right side of every production. They are, in Rosenberg's terms, "the lowest class of nonregular context-free grammars," and thus constitute a minimal means to accommodate the complexities of natural language morphology without departing radically from the simplicity of the finite-state automaton model.

The Nandi word given in Section 3, ceem-naandi-iin-i-cu 'these Nandi women', could be handled with the following two-tape automaton. The transition function, $\delta$, now takes three arguments—state, morpheme, and tape. We will denote the suffix tape by 0 and the prefix tape by 1. Assume the root, naandi, has been identified and has category N0.

\begin{align}
(12) \quad Q &= \{00, N0, N1, N2, N3, WW\}, \\
\Sigma &= \{ceem, naandi, iin, i, cu\}, \\
\delta(00, naandi) &= N0, \\
\delta(N0, ceem, 1) &= N1, \\
\delta(N1, iin, 0) &= N2, \\
\delta(N2, i, 0) &= N3,
\end{align}
\[ \delta(N3, cu, 0) = WW, \]
\[ \tilde{s} = \{00\}, \]
\[ F = \{WW\}. \]

5 Keçi+

Hankamer has written a parser that implements a two-tape nondeterministic automaton. Keçi+ seeks a root beginning at every segment in the input word. Each potential root (each root in the root lexicon meeting initial conditions, which in the present implementation means a match in the first segment) is put through a set of phonological rules applying to roots; the form resulting after the application of root phonology is then compared with the input form to see if there is a match.

When a root is identified as matching at some point in the input form, pointers are set to point to the beginning of the word, the beginning of the root, and the end of the root. A string is assigned the category of the root. An `analyze()` function is called with these pointers passed as parameters; A final parameter is a pointer to the root of a tree structure which is used to record the parse. Figure 1 gives the overall operation of the parser.

```c
/* w points to beginning of input word 
r points to beginning of potential root 
r+rootlength points to end of root 
rootcat is the lexical category of the root (specified in lexical entry) 
parsetree is a pointer to a representation of the parse */

geta word( w )

for each segment in word      /* move r from beginning to end of w */
    if there is a root that matches at r 
       analyze( w, r, r+rootlength, rootcat, parsetree )

Figure 1: Pseudocode for the outer portion of keçi+ parser

`analyze()` looks through the affixes loaded from an affix dictionary. For
any that are of the appropriate category, a function \textit{afmatch()} is called to check whether the affix matches the form. The category type information in the affix entry determines whether the match is tried at the left edge or the right edge of the identified form. If a match is found, the left and right pointers are reset, the category value is reset, and \textit{analyze()} is called again. When the left pointer points to the beginning of the word, the right pointer points at the end of the word, and the category is a final state, there has been a successful parse. At this point the program prints a record of the parse as a bracketed string. As in keći, multiple parses are stored in a stack of windows and can be viewed any time before beginning analysis of a new word. Figure 2 depicts the operation of the \textit{analyze()} function.

\begin{verbatim}
analyze( w, leftedge, rightedge, cat, parsetree )
    if leftedge is at beginning of word and rightedge is at end of word
        and cat is in a final state
            output parse
            for each suffix entry /* incat, outcat, form, gloss */
                if incat equals cat
                    and affix form matches surface representation
                        if prefix
                            reset leftedge
                        else if suffix
                            reset rightedge
                        modify parsetree
                        analyze( w, leftedge, rightedge, outcat, parsetree )
\end{verbatim}

Figure 2: Pseudocode for the \textit{analyze()} function

Keći+ has been adapted by Creider to handle batch input and tested with a \textit{circa} 1000-word file that consists of all of the (unique) words in the first ten chapters of the book of Genesis in the Nandi Bible. On a Sun IPC, this version of keći+ returns a complete and accurate set of parses for all words in less than 2 minutes.

The keći+ implementation, like its predecessor keći, is designed to allow for the operation of phonological rules which convert the underlying repre-
sentations given in the root lexicon and the affix lexicon to surface forms which are then matched with elements of the input word. As our concern is with the specification of structural relations among morphological elements (for which the linear grammar is at least necessary and perhaps sufficient), these phonological rules are not relevant. They are, however, restricted in ways which are worth noting: the rules have access to the underlying form of the morpheme currently under consideration (which they may modify) and to the surface string. They do not have access to underlying representations of preceding or following strings.

6 Further improvements to the model

The keçi+ parser can be modelled with a modified finite-state automaton, which we call a preset two-head automaton, that has two read heads, a left head and a right head. The automaton nondeterministically chooses a starting cell on the input tape; the symbol in the cell is a candidate root segment. The left and right heads are positioned over this chosen cell and the machine's first step is to read the cell's symbol and, if the symbol is acceptable, it then moves both the read heads, the left head to the left and the right head to the right. The left head can move only left and the right head can move only right. At each subsequent machine step, only one of the read heads moves. A computation terminates successfully when both read heads have fallen off the input tape and the automaton is in a final state. Since the machine chooses a "root" cell nondeterministically, it accepts an input string if there is a root-cell choice from which there is a machine computation that terminates successfully.

The relationship between the keçi+ parser and the preset automaton should be clear. The presetting of the two read heads corresponds to the sequential search, in the keçi+ program, for a candidate root segment. The subsequent machine steps correspond to the dependency checking of the prefixes and suffixes.

We also claim that the preset two-head automata provide a new characterization of the linear context-free languages. We sketch the constructive proofs of this fact and provide full proofs in the Appendix.

Let G be a linear context-free grammar in which the productions have one of three forms:
(13) a. $A \to a$, $a$ is a terminal symbol.
b. $A \to aB$, $a$ is a terminal symbol and $B$ is a nonterminal symbol.
c. $A \to Ba$, $a$ is a terminal symbol and $B$ is a nonterminal symbol.

There is no loss of generality when we enforce these restrictions. Observe that productions of type $a$ introduce root segments and cause derivation termination.

We construct a preset two-head automaton from $G$ as follows:

(14) a. $(s, a, a, A)$ is a starting transition if $A \to a$ is a production in $G$.
b. $(B, a, \lambda, A)$ is a continuation transition if $A \to aB$ is in $G$.
c. $(B, \lambda, a, A)$ is a continuation transition if $A \to aB$ is in $G$.

A transition $(p, a, b, q)$ means that, if $M$ is in state $p$ and $a$ is under the left head, and $b$ is under the right head, then $M$ changes state to $q$ and moves the two heads. If one of $a$ and $b$ is $\lambda$, the corresponding head does not read a symbol and does not move. The states of $M$ are the nonterminals of $G$ and a new state $s$, which is the start state. Because state $s$ is new, $M$ can never revisit state $s$ during a computation; thus, the only time that $M$ reads with both heads is at the first step. The sentence symbol $S$ of $G$ is the only final state of $M$. A computation of $M$ that begins in state $s$ and terminates in state $S$, in which both read heads have fallen off the input, mimics, in reverse, a derivation in $G$ that begins with $S$ and terminates with a type $a$ production. Thus a terminal string is accepted by $M$ if and only if it is generated by $G$.

The converse construction is quite similar. We are given a preset two-head automaton $M$ from which we construct a linear context-free grammar $G$ as follows:

(15) a. $p \to a$ is a terminating production if $(s, a, a, p)$ is a transition in $M$.
b. $q \to ap$ is a production if $(p, a, \lambda, q)$ is a transition in $M$.
c. $q \to pa$ is a production if $(p, \lambda, a, q)$ is a transition in $M$.
d. $S \to ap$ is a production if $(p, a, \lambda, q)$ is a transition in $M$, for some final state $q$.
e. $S \to pa$ is a production if $(p, \lambda, a, q)$ is a transition in $M$, for some final state $q$. 

11
Again, a derivation in $G$ mimics, in reverse, a computation in $M$; thus, their languages are identical.

7 Relevant linguistic evidence

It is interesting that there is independent evidence, coming from the study of word-internal phonology, that morphologically complex words are not parsed in a strict left-to-right fashion but are parsed as suggested in Section 5. Consider the case of Nandi vowel harmony. A large number of roots and a small number of suffixes must be specified in the lexicon as having vowels of [+ATR] (Advanced Tongue Root) quality. These items affect all subsequent suffixes, and prefixes as well, ensuring that they also are articulated with [+ATR] vowels. Since the [+ATR] suffixes determine the [ATR] features on prefixes, and never vice versa, it appears that in Nandi these suffixes are to be parsed before prefixes, since otherwise it would be impossible to tell what [ATR] value a prefix should have. As an example of the operation of vowel harmony, contrast (16) with (17):

(16) \[\text{[ki:-go-soːman]}\]
    Past–3rdPers–read
    ‘s/he/they has/have studied (are educated)’

(with [−ATR] vowels) and

(17) \[\text{[ki:-go-soːman-i]}\]
    Past–3rdPers–read–Ipfv
    ‘s/he/they has/have studied (before and will again)’

(with [+ATR] vowels). In the case of (17), the presence of the lexically specified [+ATR] imperfective morpheme, [−i], induces all other vowels in the word to change to their [+ATR] equivalents. The imperfective suffix must be processed first in order to obtain the required [ATR] quality information which is then used in processing the prefixes.

Evidence in support of the need to recognize hierarchical morphological structure which is independent of linear ordering is found in other languages as well. Levergood (1987) discusses the lexical phonology of Arusa, a member of the Maa group of Eastern Nilotic languages, and establishes that there is
an innermost layer of prefixes which are hierarchically closer to the root than
an intermediate layer of suffixes, which are themselves hierarchically closer
to the root than an outer layer of the remaining prefixes. Goldsmith (1990)
presents compelling evidence for the processing of suffixes before prefixes in
Bantu languages. Inkelas (1992) shows that a seemingly intricate complex of
nonlocal dependencies among verbal suffixes in Nimboran, a Papuan language
of New Guinea, is susceptible of an elegant explanation if the whole complex
of suffixes is regarded as hierarchically organized around an abstract core, to
which two of the suffix positions are in fact prefixes. Crucial to the analysis
is that the first position following the core forms an innermost layer, then
the two prefix positions are attached, and finally the outer layer consisting
of the rest of the suffix positions.

8 Formal representations of morphological structure

Linguists have been surprisingly informal (and as a group, eclectic) in their
caracterizations of the morphological component of grammars for natural
languages. Most concern has been with the nature of morphological rules.
Selkirk (1982) has argued that, at least for English, morphological rules can
be specified with a context-free constituent structure grammar. Roeper and
Siegel (1978) have argued that at least some morphological rules are transfor-
mations. Lieber (1992) presents a more recent and more thorough-going ap-
lication of government-binding theory (including the transformation Move-
α) to morphology. Hoeksema (1986) provides a categorial grammar frame-
work for morphological rules. Although Selkirk (1982, 3) asserts that “A
categorial grammar is at best a notational variant of a context-free rewriting
grammar,” recent work (Bentham (1990)) suggests that the exact location of
categorial grammars in the grammar hierarchy is not known. Given the nice
fit of the representation of the transition network with a categorial gram-
mar (Hankamer (1986; 1988)), it would be intriguing to explore the relation
between a linear grammar and the kind of simple categorial grammar it re-
quires.

Given the very good fit of the much more restricted linear grammar with
a wide range of natural language data, we suggest that linguistic research
should concentrate on morphological theories which are similarly restricted. There are some kinds of natural language phenomena for which enhanced models might be required, such as tonal morphemes, morphemic templates, and infixes. Preliminary research on one of these, tonal morphology, suggests that most if not all tonal phenomena can be handled with simple extensions, incorporating the insights of autosegmental phonology (Goldsmith (1990)), to the models (Creider (1991)).

9 Summary

In this article we have suggested that morphological structure may be handled with a linear context-free grammar. We have discussed the operation of a parser that implements the action of a two-tape nondeterministic finite-state automaton, and we have discussed linguistic evidence in support of the appropriateness of this type of parser for natural language morphology.

REFERENCES


Inkelas, Sharon (1992) "Nimboran position class morphology," ms., University of California, Berkeley, California, to appear in *Natural Language and Linguistic Theory*.


Appendix: Preset two-head automata and linear languages

We use $\lambda$ to denote the empty string.

A **preset two-head automaton** $M$ is specified by a tuple $(Q, \Sigma, \delta, s, F)$, where

- $Q$ is a **state alphabet**, 
- $\Sigma$ is an **input alphabet**, and $\Sigma_{\lambda} = \Sigma \cup \{\lambda\}$, 
- $\delta \subseteq Q \times \Sigma_{\lambda} \times \Sigma_{\lambda} \times Q$ is a **finite transition relation**, 
- $s \in Q$ is a **start state**, and 
- $F \subseteq Q$ is a set of **final states**.

A transition of the form $(p, a, \lambda, q)$ is interpreted as: If the current state of the automaton $M$ is $p$, and the symbol under the left read head is $a$, then $M$ enters state $q$ and moves the left head one symbol to the left. Similarly, a transition of the form $(p, \lambda, a, q)$ moves the right head to the right. We allow a transition to move both heads only if the current state is the start state. Moreover, the start state must not be reachable from any other state. To initialize $M$ we move the two read heads to be over the same symbol in the input.
string (if the input string is the empty string, the automaton terminates immediately). The choice of symbol appearance is made nondeterministically. Thus, a transition from the start state has the form \((s, a, a, p)\), where \(p\) is the state to be entered and \(M\) moves both heads, the left head to the left and the right head to the right.

To define the computational sequences of \(M\), we first define its configurations. A tuple \((p, x, a, b, y) \in Q \times \Sigma^* \times \Sigma^\lambda \times \Sigma^\lambda \times \Sigma^*\) is a configuration of \(M\). The left head is over \(a\), the right head is over \(b\), \(xa\) is the unread prefix of the input string, and \(by\) is the unread suffix of the input string. If \(p = s\), then \(a = b\), and \(xay\) is the input string; otherwise, either \(a\) or \(b\) is the empty string. A configuration \((s, x, a, a, y)\) is called an initial configuration and a configuration \((f, \lambda, \lambda, \lambda, \lambda)\) is called a final configuration.

There are three types of computational steps in \(M\):

1. \(M\) is in an initial configuration \((s, x, a, a, y)\) and \((s, a, a, P) \in \delta\); then, a new configuration of \(M\) is \((p, x', c, d, y')\), where \(x = x'c\) and \(y = dy'\). We write \((s, x, a, a, y) \vdash (p, x', c, d, y')\).

2. \(M\) is in a configuration \((p, x, a, b, y)\) with \(p \neq s\) and \((p, a, \lambda, q) \in \delta\); then, a new configuration of \(M\) is \((q, x', c, b, y)\), where \(x = x'c\). We write \((p, x, a, b, y) \vdash (q, x', c, b, y)\).

3. \(M\) is in a configuration \((p, x, a, b, y)\) with \(p \neq s\) and \((p, \lambda, b, q) \in \delta\); then, a new configuration of \(M\) is \((q, x, a, d, y')\), where \(y = dy'\). We write \((p, x, a, b, y) \vdash (q, x, a, d, y')\).

We extend \(\vdash\) to \(\vdash^i\), \(i \geq 0\), \(\vdash^+\), and \(\vdash^*\) in the usual way, see Wood (1987).

A string \(w \in \Sigma^*\) is accepted by a preset two-head automaton if there is a computation

\[(s, x, a, a, y) \vdash^* (f, \lambda, \lambda, \lambda, \lambda),\]

where \(w = xay\) and \(f \in F\). The set of all accepted strings is the language of \(M\), which we denote by \(L(M)\).

We specify a linear (context-free) grammar \(G\) with a triple \((N, \Sigma, P, S)\), where

- \(N\) is the nonterminal alphabet,
\( \Sigma \) is the terminal alphabet,

\[ P \subseteq (N \times \Sigma) \cup (N \times \Sigma N) \cup (N \times N \Sigma) \] is a set of productions, and

\( S \in N \) is a sentence symbol.

We can define a derivation step and a derivation sequence in the standard way, see Wood (1987), for example. We write \( uAv \Rightarrow u' Bv' \) if either \( A \Rightarrow aB \in P, u' = ua \), and \( v' = v \), or \( A \Rightarrow Bb \in P, u' = u \), and \( v' = bv \). In addition, we write \( uAv \Rightarrow uvv \) if \( A \Rightarrow a \in P \). We extend \( \Rightarrow \) to \( \Rightarrow^i \), \( i \geq 0 \), \( \Rightarrow^+ \), and \( \Rightarrow^* \) in the usual way. A terminal string \( x \) is a sentence of \( G \) if \( S \Rightarrow^* x \in G \). The set of sentences, or language, of \( G \) is denoted by \( L(G) \).

Note that we have defined both preset automata and linear grammars such that the empty string is not in their languages. This restriction is convenient from a technical standpoint; it is not necessary.

We now prove that these two models give rise to the same languages; thus, the two models are equivalent in expressive power and the preset automata provide a new characterization of the linear languages.

**Theorem 1** Every linear language is a preset two-head automaton language.

**Proof:** Let \( L \subseteq \Sigma^* \), where \( \Sigma \) is some alphabet, be a linear language. Then, \( L = L(G) \), for some linear grammar \( G = (N, \Sigma, P, S) \). We construct a preset automaton \( M = (N \cup \{s\}, \Sigma, \delta, s, \{S\}) \) such that \( L(M) = L \). The construction is defined as follows.

1. \((s, a, a, A)\) is a starting transition if \( A \Rightarrow a \) is a production in \( G \).
2. \((B, a, \lambda, A)\) is a continuation transition if \( A \Rightarrow aB \) is in \( G \).
3. \((B, \lambda, a, A)\) is a continuation transition if \( A \Rightarrow aB \) is in \( G \).

We claim that \((s, a, a, y) \prec^i (\delta, \lambda, \lambda, \lambda, \lambda) \) if and only if \( S \Rightarrow^i x \text{ay} \). We prove this claim by induction on the number of steps and by strengthening the claim. The strengthened claim is: \((s, a, a, y) \prec^i (A, \lambda, \lambda, \lambda, \lambda) \) if and only if \( A \Rightarrow^i x \text{ay} \).

The basis, \( i = 1 \), follows directly from the construction, namely, \((s, \lambda, a, a, \lambda) \Rightarrow (\lambda, \lambda, \lambda, \lambda) \) if and only if \( A \Rightarrow a \), or \( A \Rightarrow a \) is in \( P \).

We assume that the strengthened claim holds for some \( i \geq 1 \) and we establish that it holds for all computations and derivations of length
If \((s, x, a, a, y) \vdash^{i+1} (A, \lambda, \lambda, \lambda, \lambda)\), then, because \(i + 1 \geq 2\), either \((s, x, a, a, y) \vdash^i (B, \lambda, \lambda, b, \lambda)\) or \((s, x, a, a, y) \vdash^i (B, \lambda, b, \lambda, \lambda)\) for some state \(a\) and symbol \(b\). We consider the first possibility, the second can be dealt with similarly. Now, \((s, x, a, a, y') \vdash^i (B, \lambda, \lambda, \lambda, \lambda)\), where \(y = y'b\); therefore, \(B \Rightarrow^i xay'\), by the induction hypothesis. Immediately, since \((B, \lambda, \lambda, b, \lambda) \vdash (A, \lambda, \lambda, \lambda, \lambda)\), \(A \Rightarrow Bb\) is in \(P\) and \(A \Rightarrow^{i+1} xay\), as desired.

Conversely, if \(A \Rightarrow^{i+1} xay\), then, because \(i + 1 \geq 2\), \(A \Rightarrow Bb \Rightarrow^i xay\) or \(A \Rightarrow bB \Rightarrow^i xay'\) for some \(b\) and \(B\). Again we consider only the first possibility. Clearly, \(B \Rightarrow^i xay'\), where \(y = y'b\), and \((s, x, a, a, y') \vdash^i (B, \lambda, \lambda, \lambda, \lambda)\). Thus, \((s, x, a, a, y) \vdash^{i+1} (A, \lambda, \lambda, \lambda, \lambda)\).

The reverse theorem is proved in a similar manner.

**Theorem 2** Every preset two-head automaton language is a linear language.

**Proof:** Let \(L \subseteq \Sigma^*\), where \(\Sigma\) is some alphabet, be a preset automaton language. Then, \(L = L(M)\), for some preset automaton \(M = (Q, \Sigma, \delta, s, F)\). We construct a linear grammar \(G = (Q \cup \{S\}, \Sigma, P, S)\) such that \(L = L(G)\) as follows:

1. \(p \rightarrow a\) is a terminating production if \((s, a, a, p)\) is a transition in \(M\).
2. \(q \rightarrow ap\) is a production if \((p, a, \lambda, q)\) is a transition in \(M\).
3. \(q \rightarrow pa\) is a production if \((p, \lambda, a, q)\) is a transition in \(M\).
4. \(S \rightarrow ap\) is a production if \((p, a, \lambda, q)\) is a transition in \(M\), for some final state \(q\).
5. \(S \rightarrow pa\) is a production if \((p, \lambda, a, q)\) is a transition in \(M\), for some final state \(q\).

We claim that \(S \Rightarrow^i xay\) if and only if \((s, x, a, a, y) \vdash^i (p, \lambda, \lambda, \lambda, \lambda)\), for some \(p\) in \(F\). We first establish, by induction on the number of steps, that \(p \Rightarrow^i xay\) if and only if \((s, x, a, a, y) \vdash^i (p, \lambda, \lambda, \lambda, \lambda)\). Then, if \(p\) is in \(F\), then from the construction of \(G\), we can replace the first step that uses a \(p\)-production with an \(S\)-production that has the same right-hand side; thus, the claim holds.

The base case of the induction has \(i = 1\). Thus, \((s, \lambda, a, a, \lambda) \vdash (p, \lambda, \lambda, \lambda, \lambda)\) in \(M\) and, by construction, \(p \Rightarrow a\) in \(G\) when \(p\) is in \(F\). Thus,
$S \Rightarrow a$. Consider a computation $(s, x, a, a, y) \vdash^{i+1} (p, \lambda, \lambda, \lambda, \lambda)$. Then, either the $(i+1)$th computational step is $(q, \lambda, b, \lambda, \lambda) \vdash (p, \lambda, \lambda, \lambda, \lambda)$ or $(q, \lambda, \lambda, b, \lambda) \vdash (p, \lambda, \lambda, \lambda, \lambda)$ for some $q$ in $Q$. We consider the first case. Clearly, $(s, x', a, a, y) \vdash^i (q, \lambda, \lambda, \lambda, \lambda)$, where $x = ba'$, and $q \Rightarrow^i a'ay$ in $G$. Moreover, $p \Rightarrow bq$ is in $P$; therefore, $p \Rightarrow^{i+1} xay$ and we have completed the induction step and the proof since $S \Rightarrow^{i+1} xay$ if $p$ is in $F$. \hfill \Box