CONTEXT-FREENESS OF THE LANGUAGE ACCEPTED BY MARCUS' PARSER

R. Nozohoor-Farshi
School of Computing Science, Simon Fraser University
Burnaby, British Columbia, Canada V5A 1S6

ABSTRACT

In this paper, we prove that the set of sentences parsed by Marcus' parser constitutes a context-free language. The proof is carried out by constructing a deterministic pushdown automaton that recognizes those strings of terminals that are parsed successfully by the Marcus parser.

1. Introduction

While Marcus [4] does not use phrase structure rules as base grammar in his parser, he points out some correspondence between the use of a base rule and the way packets are activated to parse a construct. Charniak [2] has also assumed some phrase structure base grammar in implementing a Marcus style parser that handles ungrammatical situations. However neither has suggested a type for such a grammar or the language accepted by the parser. Berwick [1] relates Marcus' parser to LR(k,t) context-free grammars. Similarly, in [5] and [6] we have related this parser to LRRL(k) grammars. Inevitably, these raise the question of whether the string set parsed by Marcus' parser is a context-free language.

In this paper, we provide the answer for the above question by showing formally that the set of sentences accepted by Marcus' parser constitutes a context-free language. Our proof is based on simulating a simplified version of the parser by a pushdown automaton. Then some modifications of the PDA are suggested in order to ascertain that Marcus' parser, regardless of the structures it puts on the input sentences, accepts a context-free set of sentences. Furthermore, since the resulting PDA is a deterministic one, it confirms the determinism of the language parsed by this parser. Such a proof also provides a justification for assuming a context-free underlying grammar in automatic generation of Marcus type parsers as discussed in [5] and [6].

2. Assumption of a finite size buffer

Marcus' parser employs two data structures: a pushdown stack which holds the constructs yet to be completed, and a finite size buffer which holds the lookaheads. The lookaheads are completed constructs as well as bare terminals. Various operations are used to manipulate these data structures. An "attention shift" operation moves a window of size k=3 to a given position on the buffer. This occurs in parsing some constructs, e.g., some NP's, in particular when a buffer node other than the first indicates start of an NP. "Restore buffer" restores the window to its previous position before the last "attention shift". Marcus suggests that the movements of the window can be achieved by employing a stack of displacements from the beginning of the buffer, and in general he suggests that the buffer could be unbounded on the right. But in practice, he notes that he has not found a need for more than five cells, and PARSIFAL does not use a stack to implement the window or virtual buffer.

A comment regarding an infinite buffer is in place here. An unbounded buffer would yield a parser with two stacks. Generally, such parsers characterize context-sensitive languages and are equivalent to linear bounded automata. They have also been used for parsing some context-free languages. In this role they may hide the non-determinism of a context-free language by storing an unbounded number of lookaheads. For example, LR-regular [3], BCP(m,n), LR(k,∞) and FSPA(k) parsers [8] are such parsers. Furthermore, basing parsing decisions on the whole left contexts and k lookaheads in them has often resulted in defining classes of context-free (context-sensitive) grammars with undecidable membership. LR-regular, LR(k,∞) and FSPA(k) are such classes. The class of GLRRL(k) grammars with unbounded buffer (defined in [5]) seems to be the known exception in this category that has decidable membership. Walters [9] considers context-sensitive grammars with deterministic two-stack parsers and shows the undecidability of the membership problem for the class of such grammars.

In this paper we assume that the buffer in a Marcus style parser can only be of a finite size b (e.g., b=5 in Marcus' parser). The limitation on the size of the buffer has two important consequences. First, it allows a proof for the context-freeness of the language to be given in terms of a PDA. Second, it facilitates the design of an effective algorithm for automatic generation of a parser. (However, we should add that: 1- some Marcus style parsers that use an unbounded buffer in a constrained way, e.g., by restricting the window to the k rightmost elements of the buffer, are equivalent to pushdown automata. 2- Marcus style parsers with unbounded buffer, similar to GLRRL parsers, can still be constructed for those languages which are known to be context-free.)

3. Simplified parser

A few restrictions on Marcus' parser will prove to be convenient in outlining a proof for the context-freeness of the language accepted by it.

(i) Prohibition of features:
Marcus allows syntactic nodes to have features containing the grammatical properties of the constituents that they represent. For implementation purposes, the type of a node is also considered as a feature. However, here a distinction will be made between this feature and others. We consider the type of a node and the node itself to convey the same concept (i.e., a non-terminal symbol). Any other feature is disallowed. In Marcus' parser, the binding of traces is also implemented through the use of features. A trace is a null deriving non-terminal (e.g., an NP) that has a feature pointing to another node, i.e., the binding of the trace. We should stress at the outset that Marcus' parser outputs the annotated surface structure of an utterance and traces are intended to be used by the semantic component to recover the underlying predicate/argument structure of the utterance. Therefore one could put aside the issue of trace registers without affecting any argument that deals with the strings accepted by the parser, i.e., frontiers of surface structures. We will reintroduce the features in the generalized form of PDA for the completeness of the simulation.

(ii) Non-accessibility of the parse tree:
Although most of the information about the left context is captured through the use of the packeting mechanism in Marcus' parser, he nevertheless allows limited access to the nodes of the partial parse tree (besides the current active node) in the action parts of the grammar rules. In some rules, after the initial pattern matches, conditional clauses test for some property of the parse tree. These tests are limited to the left daughters of the current active node and the last cyclic node (NP or S) on the stack and its descendants. It is plausible to eliminate tree accessibility entirely through adding new packets and/or simple flags. In the simplified parser, access to the partial parse tree is disallowed. However, by modifying the stack symbols of the PDA we will later show that the proof of context-freeness carries over to the general parser (that tests limited nodes of parse tree).

(iii) Atomic actions:
Action segments in Marcus' grammar rules may contain a series of basic operations. To simplify the simulation, we assume that in the simplified parser actions are atomic. Breakdown of a compound action into atomic actions can be achieved by keeping the first operation in the original rule and introducing new singleton packets containing a default pattern and a remaining operation in the action part. These packets will successively deactivate themselves and activate the next packet much like "run <rule> next"s in PIDGIN. The last packet will activate the first if the original rule leaves the packet still active. Therefore in the simplified parser action segments are of the following forms:

(1) Activate packets1; [deactivate packets2].
(2) Deactivate packets1; [activate packets2].
(3) Attach ith; [deactivate packets1]; [activate packets2].
(4) [Deactivate packets1]; create node; activate packets2.
(5) [Deactivate packets1]; cattach node; activate packets2.[1]
(6) Drop; [deactivate packets1]; [activate packets2].
(7) Drop into buffer; [deactivate packets1]; [activate packets2].
(8) Attention shift (to ith cell); [deactivate packets1]; [activate packets2].
(9) Restore buffer; [deactivate packets1]; [activate packets2].

Note that forward attention shift has no explicit command in Marcus' rules. An "AS" prefix in the name of a rule implies the operation. Backward window move has an explicit command "restore buffer". The square brackets in the above forms indicate optional parts. Feature assignment operations are ignored for the obvious reason.

4. Simulation of the simplified parser

In this section we construct a PDA equivalent to the simplified parser. This PDA recognizes the same string set that is accepted by the parser. Roughly, the states of the PDA are symbolized by the contents of the parser's buffer, and its stack symbols are ordered pairs consisting of a non-terminal symbol (i.e., a stack symbol of the parser) and a set of packets associated with that symbol.

Let N be the set of non-terminal symbols, and Σ be the set of terminal symbols of the parser. We assume the top S node, i.e., the root of a parse tree, is denoted by S_0, a distinct element of N. We also assume that a final packet is added to the PIDGIN grammar. When the parsing of a sentence is completed, the activation of this packet will cause the root node S_0 to be dropped into the buffer, rather than being left on the stack. Furthermore, let P denote the set of all packets of rules, and 2^P the powerset of P, and let $P, P_1, P_2, \ldots$ be elements of 2^P. When a set of packets P is active, the pattern segments of the rules in these packets are compared with the current active node and contents of the virtual buffer (the window). Then the action segment of a rule with highest priority that matches is executed. In effect the operation of the parser can be characterized by a partial function M from active packets, current active node and contents of the window into atomic actions, i.e.,

$$M: 2^P \times N^{(1)} \times V^{(k)} \rightarrow \text{ACTIONS}$$

[1] "Cattach" is used as a short notation for "create and attach".

where $V = N \cup \Sigma$, $V(k) = V0+V1+...+Vk$ and ACTIONS is the set of atomic actions (1) - (9) discussed in the previous section.

Now we can construct the equivalent PDA $A=(Q,\Sigma,\Gamma,\delta,q_0,Z_0,f)$ in the following way.

Σ = the set of input symbols of A, is the set of terminal symbols in the simplified parser.
Γ = the set of stack symbols [X,P], where $X \in N$ is a non-terminal symbol of the parser and P is a set of packets.
Q = the set of states of the PDA, each of the form $\langle P_1,P_2,\text{buffer}\rangle$, where P_1 and P_2 are sets of packets. In general P_1 and P_2 are empty sets except for those states that represent dropping of a current active node in the parser. P_1 is the set of packets to be activated explicitly after the drop operation, and P_2 is the set of those packets that are deactivated. "buffer" is a string in $(|^{(1)}V)^{(m)}|V^{(k)}$, where $0 \le m \le b-k$. The last vertical bar in "buffer" denotes the position of the current window in the parser and those on the left indicate former window positions.
q_0 = the initial state = $\langle \emptyset,\emptyset,|\lambda\rangle$, where λ denotes the null string.
f = the final state = $\langle \emptyset,\emptyset,|S_0\rangle$. This state corresponds to the outcome of an activation of the final packet in the parser. In this way, i.e., by dropping the S_0 node into the buffer, we can show the acceptance of a sentence simultaneously by empty stack and by final state.
Z_0 = the start symbol = $[S_0,P_0]$, where P_0 is the set of initial packets, e.g., {SS-Start, C-Pool} in Marcus' parser.
δ = the move function of the PDA, defined in the following way:

Let P denote a set of active packets, X an active node and $W_1W_2...W_n$, $n \le k$, the content of a window. Let $\alpha|W_1W_2...W_n\beta$ be a string (representing the buffer) such that: $\alpha \in (|^{(1)}V)^{(b-k)}$ and $\beta \in V^*$ where $\text{Length}(\alpha' W_1W_2...W_n\beta) \le b$, and α' is the string α in which vertical bars are erased.

Non-λ-moves: The non-λ-moves of the PDA A correspond to bringing the input tokens into the buffer for examination by the parser. In Marcus' parser input tokens come to the attention of parser as they are needed. Therefore, we can assume that when a rule tests the contents of n cells of the window and there are fewer tokens in the buffer, terminal symbols will be brought into the buffer. More specifically, if $M(P,X,W_1...W_n)$ has a defined value (i.e., P contains a packet with a rule that has pattern segment $[X][W_1]...[W_n]$), then
$\delta(\langle \emptyset,\emptyset,\alpha|W_1...W_j\rangle, W_{j+1}, [X,P]) =$
$(\langle \emptyset,\emptyset,\alpha|W_1...W_jW_{j+1}\rangle, [X,P])$ for all α, and for $j = 0, ..., n-1$ and $W_{j+1} \in \Sigma$.

λ-moves: By λ-moves, the PDA mimics the actions of the parser on successful matches. Thus the δ-function on λ input corresponding to each individual atomic action is determined according to one of the following cases.

Cases (1) and (2):
If $M(P,X,W_1W_2...W_n)$ = "activate P_1; deactivate P_2" (or "deactivate P_2; activate P_1"), then
$\delta(\langle \emptyset,\emptyset,\alpha|W_1W_2...W_n\beta\rangle,\lambda,[X,P]) =$
$(\langle \emptyset,\emptyset,\alpha|W_1W_2...W_n\beta\rangle,[X,(P \cup P_1)-P_2])$ for all α and β.
Case (3):
If $M(P,X,W_1W_2...W_i...W_n)$ = "attach ith (normally i is 1); deactivate P_1; activate P_2", then
$\delta(\langle \emptyset,\emptyset,\alpha|W_1...W_i...W_n\beta\rangle,\lambda,[X,P]) =$
$(\langle \emptyset,\emptyset,\alpha|W_1...W_{i-1}W_{i+1}...W_n\beta\rangle, [X,(P \cup P_2)-P_1])$ for all α,β.
Cases (4) and (5):
If $M(P,X,W_1...W_n)$= "deactivate P_1; create/cattach Y; activate P_2", then
$\delta(\langle \emptyset,\emptyset,\alpha|W_1...W_n\beta\rangle,\lambda,[X,P]) =$
$(\langle \emptyset,\emptyset,\alpha|W_1...W_n\beta\rangle, [X,P-P_1][Y,P_2])$ for all α and β.
Case (6):
If $M(P,X,W_1...W_n)$ = "drop; deactivate P_1; activate P_2", then
$\delta(\langle \emptyset,\emptyset,\alpha|W_1...W_n\beta\rangle,\lambda,[X,P]) = (\langle P_2,P_1,\alpha|W_1...W_n\beta\rangle,\lambda)$ for all α and β, and furthermore
$\delta(\langle P_2,P_1,\alpha|W_1...W_n\beta\rangle,\lambda,[Y,P']) =$
$(\langle \emptyset,\emptyset,\alpha|W_1...W_n\beta\rangle, [Y,(P' \cup P_2)-P_1])$ for all α and β, and $P' \in 2^P$, $Y \in N$.
The latter move corresponds to the deactivation of the packets P_1 and activation of the packets P_2 that follow the dropping of a current active node.
Case (7):
If $M(P,X,W_1...W_n)$ = "drop into buffer; deactivate P_1; activate P_2", (where $n < k$), then
$\delta(\langle \emptyset,\emptyset,\alpha|W_1...W_n\beta\rangle,\lambda,[X,P]) = (\langle P_2,P_1,\alpha|XW_1...W_n\beta\rangle,\lambda)$ for all α and β, and furthermore
$\delta(\langle P_2,P_1,\alpha|XW_1...W_n\beta\rangle,\lambda,[Y,P']) =$
$(\langle \emptyset,\emptyset,\alpha|XW_1...W_n\beta\rangle, [Y,(P' \cup P_2)-P_1])$ for all α and β, and for all $P' \in 2^P$ and $Y \in N$.
Case (8):
If $M(P,X,W_1...W_i...W_n)$ = "shift attention to ith cell; deactivate P_1; activate P_2", then
$\delta(\langle \emptyset,\emptyset,\alpha|W_1...W_i...W_n\beta\rangle,\lambda,[X,P]) =$
$(\langle \emptyset,\emptyset,\alpha|W_1...|W_i...W_n\beta\rangle, [X,(P \cup P_2)-P_1])$ for all α and β.
Case (9):
If $M(P,X,W_1...W_n)$= "restore buffer; deactivate P_1; activate P_2", then
$\delta(\langle \emptyset,\emptyset,\alpha_1|\alpha_2|W_1...W_n\beta\rangle,\lambda,[X,P]) =$
$(\langle \emptyset,\emptyset,\alpha_1|\alpha_2,W_1...W_n\beta\rangle, [X,(P \cup P_2)-P_1])$ for all α_1,α_2, and β such that α_2 contains no vertical bar.

Now from the construction of the PDA, it is obvious that A accepts those strings of terminals that are parsed successfully by the simplified parser. The reader may note that the value of δ is undefined for the cases in which $M(X,P,W_1...W_n)$ has multiple values. This accounts for the fact that Marcus' parser behaves in a deterministic way. Furthermore, many of the states of A are unreachable. This is due to the way we constructed the PDA, in which we considered activation of every subset of P with any active node

and any lookahead window.

5. Simulation of the general parser

It is possible to lift the restrictions on the simplified parser by modifying the PDA. Here, we describe how Marcus' parser can be simulated by a generalized form of the PDA.

(i) Non-atomic actions:
The behaviour of the parser with non-atomic actions can be described in terms of $M' \epsilon M^*$, a sequence of compositions of M, which in turn can be specified by a sequence δ' in δ^*.

(ii) Accessibility of descendants of current active node, and current cyclic node:
What parts of the partial parse tree are accessible in Marcus' parser seems to be a moot point. Marcus [4] states

> *"the parser can modify or directly examine exactly two nodes in the active node stack... the current active node and S or NP node closest to the bottom of stack... called the dominating cyclic node... or... current cyclic node... The parser is also free to examine the descendants of these two nodes..., although the parser cannot modify them. It does this by specifying the exact path to the descendant it wishes to examine."*

The problem is that whether by descendants of these two nodes, one means the immediate daughters, or descendants at arbitrary levels. It seems plausible that accessibility of immediate descendants is sufficient. To explore this idea, we need to examine the reason behind partial tree accesses in Marcus' parser. It could be argued that tree accessibility serves two purposes:
(1) Examining what daughters are attached to the current active node considerably reduces the number of packet rules one needs to write.
(2) Examining the current cyclic node and its daughters serves the purpose of binding traces. Since transformations are applied in each transformational cycle to a single cyclic node, it seems unnecessary to examine descendants of a cyclic node at arbitrarily lower levels.

If Marcus' parser indeed accesses only the immediate daughters (a brief examination of the sample grammar [4] does not seem to contradict this), then the accessible part of the a parse tree can represented by a pair of nodes and their daughters. Moreover, the set of such pairs of height-one trees are finite in a grammar. Furthermore, if we extend the access to the descendants of these two nodes down to a finite fixed depth (which, in fact seems to have a supporting evidence from X theory and C-command), we will still be able to represent the accessible parts of parse trees with a finite set of finite sequences of fixed height trees.

A second interpretation of Marcus' statement is that descendants of the current cyclic node and current active node at arbitrarily lower levels are accessible to the parser. However, in the presence of non-cyclic recursive constructs, the notion of giving an exact path to a descendant of the current active or current cyclic node would be inconceivable; in fact one can argue that in such a situation parsing cannot be achieved through a finite number of rule packets. The reader is reminded here that PIDGIN (unlike most programming languages) does not have iterative or recursive constructs to test the conditions that are needed under the latter interpretation.

Thus, a meaningful assumption in the second case is to consider every recursive node to be cyclic, and to limit accessibility to the subtree dominated by the current cyclic node in which branches are pruned at the lower cyclic nodes. In general, we may also include cyclic nodes at fixed recursion depths, but again branches of a cyclic node beyond that must be pruned. In this manner, we end up with a finite number of finite sequences (hereafter called forests) of finite trees representing the accessible segments of partial parse trees.

Our conclusion is that at each stage of parsing the accessible segment of a parse tree, regardless of how we interpret Marcus' statement, can be represented by a forest of trees that belong to a finite set $T_{N,h}$. $T_{N,h}$ denotes the set of all trees with non-terminal roots and of a maximum height h. In the general case, this information is in the form of a forest, rather than a pair of trees, because we also need to account for the unattached subtrees that reside in the buffer and may become an accessible part of an active node in the future. Obviously, these subtrees will be pruned to a maximum height h-1. Hence, the operation of the parser can be characterized by the partial function $\underline{M}$ from active packets, subtrees rooted at current active and cyclic nodes, and contents of the window into compound actions, i.e.,

$$\underline{M}: 2^{P} \times (T_{N,h} \cup \{\lambda\}) \times (T_{C,h} \cup \{\lambda\}) \times (T_{N,h-1} \cup \Sigma)^{(k)} \rightarrow \text{ACTIONS}^*$$

where $T_{C,h}$ is the subset of $T_{N,h}$ consisting of the trees with cyclic roots.

In the PDA simulating the general parser, the set of stack symbols Γ would be the set of triples $[T_Y, T_X, P]$, where T_Y and T_X are the subtrees rooted at current cyclic node Y and current active node X, and P is the set of packets associated with X. The states of this PDA will be of the form <X,P₁,P₂,buffer>. The last three elements are the same as before, except that the buffer may now contain subtrees belonging to $T_{N,h-1}$. (Note that in the simple case, when h=1, $T_{N,h-1}$=N). The first entry is usually λ except that when the current active node X is dropped, this element is changed to T'_X. The subtree T'_X is the tree dominated by X, i.e., T_X, pruned to the height h-1.

Definition of the move function for this PDA is very similar to the simplified case. For example, under the

assumption that the pair of height-one trees rooted at current cyclic node and current active node is accessible to the parser, the definition of $\underline{\delta}$ function would include the following statement among others:
If $\underline{M}(P,T_X,T_Y,W_1...W_n)$ = "drop; deactivate P_1; activate P_2". (where T_X and T_Y represent the height-one trees rooted at the current active and cyclic nodes X and Y), then
$\underline{\delta}(<\lambda,\emptyset,\emptyset,\alpha|W_1...W_1\beta>, \lambda,[T_Y,T_X,P]) =$
$(<X,P_2,P_1,\alpha|W_1...W_1\beta>,\lambda)$ for all α and β. Furthermore,
$\underline{\delta}(<X,P_2,P_1,\alpha|W_1...W_1\beta>, \lambda,[T_Y,T_Z,P]) =$
$(<\lambda,\emptyset,\emptyset,\alpha|W_1...W_1\beta>, [T_Y,T_Z,(P' \cup P_2)-P_1])$ for all (T_Z,P') in $T_{N,1}{}^{X2^P}$ such that T_Z has X as its rightmost leaf.
In the more general case (i.e., when h > 1), as we noted in the above, the first entry in the representation of the state will be T_X, rather than its root node X. In that case, we will replace the rightmost leaf node of T_Z, i.e., the nonterminal X, with the subtree T_X. This mechanism of using the first entry in the representation of a state allows us to relate attachments. Also, in the simple case (h=1) the mechanism could be used to convey feature information to the higher level when the current active node is dropped. More specifically, there would be a bundle of features associated with each symbol. When the node X is dropped, its associated features would be copied to the X symbol appearing in the state of the PDA (via first $\underline{\delta}$-move). The second $\underline{\delta}$-move allows us to copy the features from the X symbol in the state to the X node dominated by the node Z.

(iii) Accommodation of features:
The features used in Marcus' parser are syntactic in nature and have finite domains. Therefore the set of attributed symbols in that parser constitute a finite set. Hence syntactic features can be accommodated in the construction of the PDA by allowing complex non-terminal symbols, i.e., attributed symbols instead of simple ones.

Feature assignments can be simulated by replacing the top stack symbol in the PDA. For example, under our previous assumption that two height-one trees rooted at current active node and current cyclic node are accessible to the parser, the definition of $\underline{\delta}$ function will include the following statement:
If $\underline{M}(P,T_{X:A},T_{Y:B},W_1...W_n)$ = "assign features A' to current active node; assign features B' to current cyclic node; deactivate P_1; activate P_2" (where A,A',B and B' are sets of features), then
$\underline{\delta}(<\lambda,\emptyset,\emptyset,\alpha|W_1...W_1\beta>,\lambda, [T_{Y:B},T_{X:A},P]) =$
$(<\lambda,\emptyset,\emptyset,\alpha|W_1...W_1\beta>, [T_{Y:B \cup B'},T_{X:A \cup A'},(P \cup P_2)-P_1])$ for all α and β.

Now, by lifting all three restrictions introduced on the simplified parser, it is possible to conclude that Marcus' parser can be simulated by a pushdown automaton, and thus accepts a context-free set of strings. Moreover, as one of the reviewers has suggested to us, we could make our result more general if we incorporate a finite number of semantic tests (via a finite oracle set) into the parser. We could still simulate the parser by a PDA.

Furthermore, the pushdown automaton which we have constructed here is a deterministic one. Thus, it confirms the determinism of the language which is parsed by Marcus' mechanism. We should also point out that our notion of a context-free language being deterministic differs from the deterministic behavour of the parser as described by Marcus. However, since every deterministic language can be parsed by a deterministic parser, our result adds more evidence to believe that Marcus' parser does not hide non-determinism in any form.

It is easy to obtain (through a standard procedure) an LR(1) grammar describing the language accepted by the generalized PDA. Although this grammar will be equivalent to Marcus' PIDGIN grammar (minus any semantic considerations), and it will be a right cover for any underlying surface grammar which may be assumed in constructing the Marcus parser, it will suffer from being an unnatural description of the language. Not only may the resulting structures be hardly usable by any reasonable semantic/pragmatics component, but also parsing would be inefficient because of the huge number of non-terminals and productions.

In automatic generation of Marcus-style parsers, one can assume either a context-free or a context-sensitive grammar (as a base grammar) which one feels is naturally suitable for describing surface structures. However, if one chooses a context-sensitive grammar then one needs to make sure that it only generates a context-free language (which is unsolvable in general). In [5] and [6], we have proposed a context-free base grammar which is augmented with syntactic features (e.g., person, tense, etc.) much like attributed grammars in compiler writing systems. An additional advantage with this scheme is that semantic features can also be added to the nodes without an extra effort. In this way one is also able to capture the context-sensitivity of a language.

6. Conclusions

We have shown that the information examined or modified during Marcus parsing (i.e., segments of partial parse trees, contents of the buffer and active packets) for a PIDGIN grammar is a finite set. By encoding this information in the stack symbols and the states of a deterministic pushdown automaton, we have shown that the resulting PDA is equivalent to the Marcus parser. In this way we have proved that the set of surface sentences accepted by this parser is a context-free set.

An important factor in this simulation has been the assumption that the buffer in a Marcus style parser is bounded. It is unlikely that all parsers with unbounded buffers written in

this style can be simulated by deterministic pushdown automata. Parsers with unbounded buffers (i.e., two-stack parsers) are used either for recognition of context-sensitive languages, or if they parse context-free languages, possibly to hide the non-determinism of a language by storing an unlimited number of lookaheads in the buffer. However, this does not mean that some Marcus-type parsers that use an unbounded buffer in a constrained way are not equivalent to pushdown automata. Shipman and Marcus [7] consider a model of Marcus' parser in which the active node stack and buffer are combined to give a single data structure that holds both complete and incomplete subtrees. The original stack nodes and their lookaheads alternately reside on this structure. Letting an unlimited number of completed constructs and bare terminals reside on the new structure is equivalent to having an unbounded buffer in the original model. Given the restriction that attachments and drops are always limited to the k+1 rightmost nodes of this data structure, it is possible to show that a parser in this model with an unbounded buffer still can be simulated with an ordinary pushdown automaton. (The equivalent condition in the original model is to restrict the window to the k rightmost elements of the buffer. However simulation of the single structure parser is much more straightforward.)

ACKNOWLEDGEMENTS

The author is indebted to Dr. Len Schubert for posing the question and carefully reviewing an early draft of this paper, and to the referees for their helpful comments. The research reported here was supported by the Natural Sciences and Engineering Research Council of Canada operating grants A8818 and A9203 at the universities of Alberta and Simon Fraser.

REFERENCES

[1] R.C. Berwick. **The Acquistion of Syntactic Knowledge.** MIT Press. 1985.

[2] E. Charniak. A parser with something for everyone. **Parsing natural language**, ed. M. King, pp. 117–149. Academic Press, London. 1983.

[3] K. Culik II and R. Cohen. LR-regular grammars: an extension of LR(k) grammars. **Journal of Computer and System Sciences**, vol. 7, pp. 66–96. 1973.

[4] M.P. Marcus. **A Theory of Syntactic Recognition for Natural Language.** MIT Press, Cambridge, MA. 1980.

[5] R. Nozohoor-Farshi. LRRL(k) grammars: a left to right parsing technique with reduced lookaheads. Ph.D. thesis, Dept. of Computing Science, University of Alberta. 1986.

[6] R. Nozohoor-Farshi. On formalizations of Marcus' parser. COLING-86. 1986.

[7] D.W. Shipman and M.P. Marcus. Towards minimal data structures for deterministic parsing. IJCAI-79. 1979.

[8] T.G. Szymanski and J.H. Williams. Non-canonical extensions of bottom-up parsing techniques. **SIAM Journal of Computing**, vol. 5, no. 2, pp. 231–250. June 1976.

[9] D.A. Walters. Deterministic context-sensitive languages. **Information and Control**, vol. 17, pp. 14–61. 1970.