Efficient Finite-State Approximation of Context Free Grammars

C.M. Rood
Computer Laboratory
University of Cambridge
`cmr1001@cl.cam.ac.uk`

Abstract. This paper introduces a novel method for constructing finite–state machines recognising context free (CF) grammars. The method utilizes the idea of a *path* through a finite–state machine (FSM). Certain paths form the basis for an *unfolding* process which is applied to the $LR(0)$ characteristic finite–state machine (CFSM) corresponding to the grammar. The next section discusses the approximation algorithm, including this unfolding process, in more detail, and section 3 introduces the concept of an *unfolding criterion*. Section 4 proves the soundness of the approximation method, and its exactness to arbitrary, fixed recursive depths. Section 5 presents initial computational figures resulting from an implementation of the method. A variation of the method that is more computationally feasible is discussed. The final section compares the method with existing research on finite–state approximation of CF grammars, and presents preliminary conclusions regarding the method.

1 THE APPROXIMATION ALGORITHM

The $LR(0)$ CFSM for a CF grammar, G, can be seen as the finite–state controller for a shift–reduce pushdown recogniser, R. Standard results of automata theory tell us that the language recognised by R is the same as that generated by G, $L(G)$ [3].

The process of approximating $L(G)$ with a finite–state machine, M, can be performed directly by applying the following two steps. We start with the $LR(0)$ CFSM for G, M_C.

1. For each state in M_C containing a completed dotted rule, an ϵ–transition is inserted in M_C, in accordance with the corresponding reduce move of R. This process can be termed *flattening* [4].
2. M_C is determinised (and optimised), yielding the resultant FSM, M.

As an example, consider the grammar G_1:

$$S' \to S$$
$$S \to aXa|bXb$$
$$X \to c$$

The CFSM for G_1 appears in Figure 1.

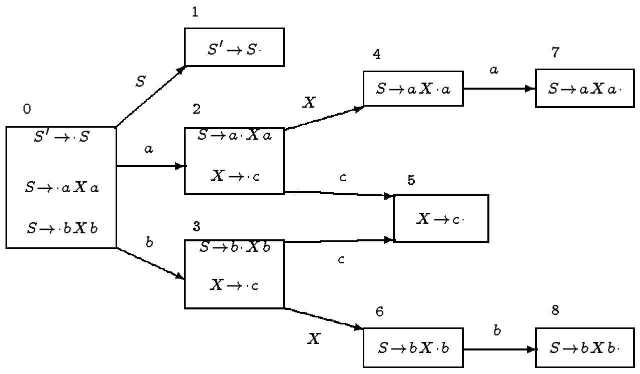

Figure 1. CFSM for G_1.

Note that the completed rule in state 5 necessitates two ϵ–transitions in the flattened machine, to states 4 and 6. When the flattened machine is determinised, the FSM exhibited in Figure 2 results.

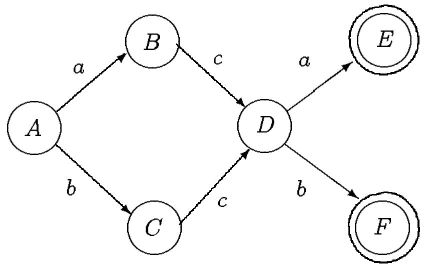

Figure 2. Finite–state approximation to G_1.

The single state D in figure 2 arises from the loss of information in the duplicate ϵ–transitions. This means the resultant machine cannot differentiate the stack configurations corresponding to recogition by R of aca and acb, as well as of

© 1996 C.M. Rood
Proceedings of the ECAI 96 Workshop
Extended Finite State Models of Language
Edited by A. Kornai.

bca and *bcb*.

To remedy this type of failing, we introduce *unfolding* [4] to encapsulate information in the stack R manipulates into the resultant FSM. Our process of unfolding takes place in two stages:

1. All *paths* through M_C are calculated and classified according to an *unfolding criterion.*
2. The states of the unfolded CFSM are generated in accordance with the paths generated in 1.

Each stack configuration of R corresponds to a path. Since there are rarely only a finite number of such paths, unfolding actually takes place according to an *unfolding criterion.* This idea is developed in the next section.

2 THE UNFOLDING CRITERION

Unfolding is required because we need to know of *distinct* paths through the CFSM to any state containing a completed dotted rule.

A *path* through a CFSM is a sequence of alternating state names and transition symbols, *ending in a state name. Unfolding* is the process of identifying possible paths through the CFSM to any state, and splitting this state when distinct paths are found. The split states are labelled with their original names plus the path used to arrive at them.

For example, the unfolded machine corresponding to G_1's CFSM is exhibited in figure 3.

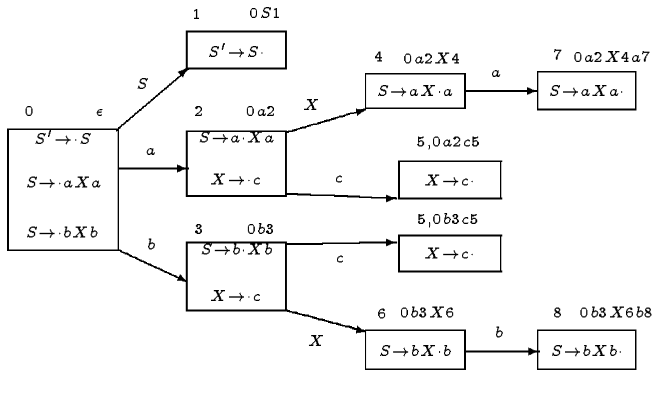

Figure 3. Unfolded CFSM for G_1.

Note that the state 5 has become two states; upon flattening and determinising, we obtain an FSM exactly recognising $L(G_1)$.

Most non-trivial grammars lead to CFSMs that loop back on themselves in several places. This creates "looping" in the paths, and therefore paths that are possibly of infinite length. What is needed is an *equivalence relation* on paths, resulting in a finite number of distinct path equivalence classes.

In terms of paths, a loop is a *repeated sequence of state names and transition symbols.* One equivalence relation on the set of paths through the CFSM arises naturally from the idea of an *unfolding criterion.* The unfolding criterion states *how many loops* a given path may have to be considered legitimate. Any path having more than the required number of loops will belong to the same infinite class of "illegitimate" paths, represented by the path having exactly one more than the specified number of loops.

For example, if we state that N, our unfolding criterion equals 2, then the paths $0a1a1$ (1 loop) and $0a1a1a1$ (2 loops) are legitimate, while $0a1a1a1a1$ (3 loops) is illegitimate.

When implemented, use of the unfolding criterion involves the following stipulations on the unfolded machine:

1. The states of the unfolded machine are pairs $\langle s, [p] \rangle$ of a state and a path equivalence class at that state.
2. The initial state is $\langle s_0, [\epsilon] \rangle$ where ϵ denotes the empty path, and the final states are all those states being the split correlatives of a final state.
3. The transition function, δ_{new} is given by
$$\delta_{new}(\langle s, [p] \rangle, X) = \langle \delta_{old}(s, X), [pX\delta_{old}(s, X)] \rangle \quad (1)$$
where $pX\delta_{old}(s, X)$ is the path obtained by adjoining X and $\delta_{old}(s, X)$ to p.

As an example of the power of the unfolding criterion, consider the grammar G_2:

$$S' \to S$$
$$S \to aSb$$
$$S \to ab$$

Upon unfolding, flattening, and determinising the CFSM for G_2, we obtain the following sequence of automata, for values of $N = 0, 1, 2$.

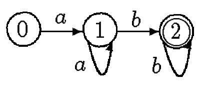

Figure 4. Resultant FSM for G_2 for $N = 1$.

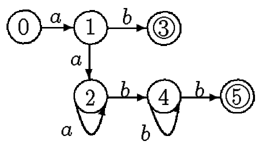

Figure 5. Resultant FSM for G_2 for $N = 2$.

One can easily discern a pattern emerging. For $N = 0$, the finite–state acceptor recognises a^+b^+; for $N = 1$, the acceptor recognises $ab|aa^+b^+b$; for $N = 2$, the acceptor recognises $ab|aabb|aaa^+b^+bb$; and, in general, for $N = k$, the corresponding acceptor will recognise $ab|\cdots|a^kb^k|a^ka^+b^+b^k$. Thus by choosing values of the unfolding criterion, N, one may achieve exactness in the resultant finite–state acceptor to a recursive depth of N.

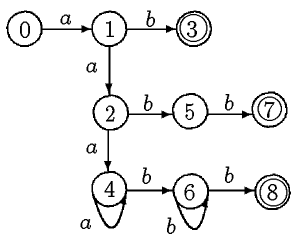

Figure 6. Resultant FSM for G_2 for $N = 3$.

3 THEORETICAL RESULTS

Three technical results follow, involving the stability and computational costs of the approximation method described in the previous two sections.

3.1 Soundness

The goal of this section is the following:

Theorem 1 *Let G be an arbitrary CF grammar, N a natural number, and $w \in A^*$ an arbitrary string (A is the alphabet of terminal symbols of our grammar, G). Then*

$$w \in L(G) \Rightarrow w \in L(M_N) \tag{2}$$

where M_N is the unfolded, flattened FSM corresponding to G and unfolding criterion N.

We consider the non-deterministic resultant FSM for simplicity. Standard results of automata theory give the extension to the corresponding determinised FSM. (See [3] and [1].)

The unfolding criterion, N, induces an equivalence relation on the set $Paths(M, s)$ of all paths through the $LR(0)$ CFSM, M, corresponding to G, which begin at the start state, s_0, and end at the state s.

Let s be a state of M and $p \in Paths(M, s)$. Then

$$p = p_1 X_1 \cdots p_{i-1} X_{i-1} s \tag{3}$$

where the p_k and are state names and the X_k are transition symbols $(1 \leq k \leq (i-1))$.

In order to formalize our notion of "path equivalence", we require the following definition. A *loop*, $l = l_1 Y_1 \cdots l_{i-1} Y_{i-1} l i$ is a *non-empty* path, that begins and terminates at the same state. We introduce the notation $l^* = l - l_i$ on loops to mean that l^* is the loop l stripped of its final state symbol, l_i.

We can now formalize the statement "path p has N loops". A path, p, has N loops iff there is a sub-string, α, of p such that $p = \beta_1 \alpha \beta_2$ for some strings β_1 and β_2, and there is a loop $l = q_1 X_1 \cdots q_{m-1} X_{m-1} q_m$ such that if $l^* = l - q_m$ then $\alpha = (l^*)^N q_m = (l^*)^N q_1$. (That is, if the path p has N *contiguous* loops.)

We can then define an *N-segment (of p)* to be N contiguous loops. A *loop segment (of p)* is an N–segment, α, which is, first of all, *maximal* in the following sense. The path $p = \beta_1 \alpha \beta_2$ (as above), but $\beta_1 \neq \alpha_1 (l^*)^k$ for any $k = 0, 1, 2, \ldots$ and string α_1 and $\beta_2 \neq (l^*)^j \alpha_2$ for any $j = 0, 1, 2, \ldots$ and string α_2. (That is, the N–segment, α, contains as many contiguous occurrences of the loop l as possible at that point within p.)

The N–segment must also have a *defining loop*, given as follows. If α is a loop segment, then $l = q_1 X_1 \cdots q_{m-1} X_{m-1} q_m$ is the (unique) *defining loop (for α)* if, for $l^* = l - q_m$, we have that $\alpha = (l^*)^N q_m$ and there is no loop l_1 such that if $l_1^* = l_1 - q_m$, then $\alpha = (l_1^*)^K q_m$ where $K > N$ and $|l_1| < |l|$. That is, the defining loop l is the *shortest* loop that yields α as a sequence of loops ending at the state q_m. If this defining loop is maximal in p, then the N–segment is also a loop segment.

We define *N–truncation* as follows. Take an M–segment, α_M, that is also a loop segment and $M > N$, form an N–segment, α_N, by truncating all $N+1$ and greater occurences of the defining loop (for α_M). Formally, if

$$l = q_1 X_1 \cdots q_{m-1} X_{m-1} q_m \tag{4}$$

is the defining loop for α_M, and $l^* = l - q_m$ as usual, then $\alpha_M = (l^*)^M q_1 = (l^*)^M q_m$ and $\alpha_N = (l^*)^N q_1 = (l^*)^N q_m$. If $M \leq N$, N–truncation has no effect. In both cases, the resulting N–segment is also a loop segment.

Given a path, $p \in Paths(M, s)$, we define the *N–image* of p to be the path p' obtained by N–truncating every loop segment of p.

As an example, consider the path

$$p = 0a1b2c1b2c1b2c1b2c1d3. \tag{5}$$

It is clear p has four loops. In the definition, let $\beta_1 = 0a$, $\beta_2 = d3$, and $l = 1b2c1$. Thus $l^* = 1b2c$ and hence $\alpha = (l^*)^4 1 = 1b2c1b2c1b2c1b2c1$ is a 4–segment of p. Similarly, $\alpha_1 = 1b2c1b2c1$ is a 2–segment. The path p has only one loop segment, namely α, which is defined by the loop $l = 1b2c1$. Note that $l_1 = 1b2c1b2c1$ is neither the defining loop for the loop segment α (since $|l| < |l_1|$) nor is l_1 a loop segment (it is not maximal in p). The 2–image of p is p' where

$$p' = 0a1b2c1b2c1d3. \tag{6}$$

The equivalence relation induced by the parameter N is given by:

$$p \equiv r \Longleftrightarrow p' = r' (literatim) \tag{7}$$

where p' and r' are the N–images of p and r respectively.

It is straightforward to verify that this congruence is *well–defined*: for each state s of M, $p, r \in Paths(M, s)$, and $a \in A$ if $p \equiv r$ then $pa\delta(s, a) \equiv ra\delta(s, a)$ (δ the transition function of M).

To prove the proposition, we need the following lemma:

Lemma 1 *Given G and M as above and R the corresponding shift–reduce recogniser having finite–state control M, let M_U be the unfolded machine corresponding to M and R_U the recogniser having finite–state control M_U. Then $L(R) = L(R_U)$.*

The action of any shift–reduce recogniser is formalised in a sequence of *configurations*. A *configuration* is a triple,

$$\langle s, \sigma, w \rangle \tag{8}$$

composed of a state (of the finite–state controller), a stack, σ, and a string, $w \in A^*$. A *stack* is a sequence of pairs,

$$\sigma = \langle s_0, X_0 \rangle, \ldots \langle s_n, X_n \rangle \tag{9}$$

of a state and transition symbol, representing the contents of the stack the recogniser manipulates.

A *computation* of the shift-reduce recogniser is a sequence $\eta_0, \ldots, \eta_k$ where $\eta_0 = \langle s_0, \epsilon, w\rangle$ (s_0 is the start state of the finite–state controller, ϵ is the empty stack, and $w \in A^*$), and η_{i+1} is derived from η_i according to the following rules.

1. *Shift Moves* If $\eta_i = \langle s_i, \sigma_i, w_i\rangle$
$$\eta_{i+1} = \langle \delta(s_i, a), \sigma_i\langle s_i, a\rangle, w_{i+1}\rangle \tag{10}$$
where $w_{i+1} = aw_i$, δ is the transition function for the finite–state controller, and $\sigma_i\langle s_i, a\rangle$ is the result of pushing the pair $\langle s_i, a\rangle$ onto the stack σ_i.
2. *Reduce Moves.* If $\eta_i = \langle s_i, \sigma_i, w_i\rangle$ and $X \to X_1 \cdots X_n\cdot$ is a completed dotted rule in the state s_i, then
$$\eta_{i+1} = \langle \delta(s_i, X), \sigma_{i+1}\langle s_i, X\rangle, w\rangle \tag{11}$$
where if $\tau = \langle s_1, X_1\rangle, \ldots, \langle s_n, X_n\rangle$ then $\sigma_{i+1}\tau = \sigma_i$.[1]

The proof of the lemma involves setting up a two–way correspondence between computations of the recogniser, R, having the $LR(0)$ CFSM, M, as its controller, and computations of the recogniser R_U having the unfolded CFSM, M_U as its controller.

The correspondence is as follows. First, we stipulate
$$\langle s_0^M, \epsilon, w\rangle \leftrightarrow \langle s_0^{M_U}, \epsilon, w\rangle \tag{12}$$
where s_0^M is the start state of M, $s_0^{M_U}$ is the start state of M_U ($s_0^{M_U}$ is actually a pair of the required format), ϵ is the empty stack, and $w \in A^*$.

Next, suppose $\eta_0, \ldots \eta_n$ is a computation of R, and fix $i, 0 \le i \le (n-1)$. We define the corresponding computation of R_U inductively as follows.

Case 1 $\eta_i = \langle s_i^M, \sigma_i, w_i\rangle$ and η_{i+1} follows from η_i according to a shift move, i.e, $\eta_{i+1} = \langle s_{i+1}^M, \sigma_{i+1}, w_{i+1}\rangle$ as in equation 10, above. Suppose the configuration of R_U corresponding to η_i is
$$\chi_i = \langle\langle s_i^M, [p_i]\rangle, \Sigma_i, w_i\rangle = \langle s_i^{M_U}, \Sigma_i, w_i\rangle. \tag{13}$$
where if $\Sigma_i = \langle\langle t_0^M, [q_0]\rangle, Y_0\rangle, \ldots, \langle\langle t_k^M, [q_k]\rangle, Y_k\rangle$ then the path $p_i = t_0^M Y_0 \cdots t_k^M Y_k s_i^M$. (Note $s_i^{M_U}$ is shorthand for the pair $\langle s_i^M, [p_i]\rangle$.)
We define
$$\chi_{i+1} = \langle\langle s_{i+1}, [p_{i+1}]\rangle, \Sigma_{i+1}, w_{i+1}\rangle \tag{14}$$
$$= \langle s_{i+1}^{M_U}, \Sigma_{i+1}, w_{i+1}\rangle \tag{15}$$
where
$$s_{i+1}^{M_U} = \delta_{M_U}(s_i^{M_U}, a), \tag{16}$$
$$\Sigma_{i+1} = \Sigma_i\langle\langle s_i^M, [p_i]\rangle, a\rangle, \tag{17}$$
$$aw_{i+1} = w_i, \tag{18}$$
and the path p_{i+1} is given by
$$p_{i+1} = t_0^M Y_0 \cdots t_k^M Y_k s_i^M a s_{i+1}^M. \tag{19}$$
The fact that this is well–defined can be proved using equation 1 (section 3) concerning the transition function of the unfolded machine, and the well–definedness of the path equivalence defined above. Details of the proof are left to the reader.

[1] It may be the case that $X \to \cdot$ is a completed rule in s_i. In this case, $s_{i+1} = s_i$ and $\tau = \epsilon$ is the empty stack.

Case 2 $\eta_i = \langle s_i^M, \sigma_i, w_i\rangle$ and η_{i+1} follows from η_i according to a reduce move, i.e, $\eta_{i+1} = \langle s_{i+1}^M, \sigma_{i+1}, w_{i+1}\rangle$ as in 11, above. In this case, we have (as before)
$$\chi_i = \langle\langle s_i^M, [p_i]\rangle, \Sigma_i, w_i\rangle \tag{20}$$
where if $\Sigma_i = \langle t_0^{M_U}, Y_0\rangle, \ldots, \langle t_k^{M_U}, Y_k\rangle$ ($t_j^{M_U}$ is shorthand for $\langle t_j^M, [q_j]\rangle$, $0 \le j \le k$), then $p_i = t_0^M Y_0 \cdots t_k^M Y_k s_i^M$. However, in this case, we define
$$\chi_{i+1} = \langle\langle s_{i+1}^M, [p_{i+1}]\rangle, \Sigma_{i+1}\langle\langle s_{i+1}^M, [p_{i+1}]\rangle, X\rangle, w_i\rangle \tag{21}$$
where this time, if $X \to X_1 \cdots X_m\cdot$ and
$$\Upsilon = \langle s_{i+1}^{M_U}, X_1\rangle, \langle r_2^{M_U}, X_2\rangle, \ldots, \langle r_m^{M_U}, X_m\rangle, \tag{22}$$
then $\Sigma_{i+1}\Upsilon = \Sigma_i$. Furthermore,
$$p_{i+1} X_1 r_2^M X_2 \cdots r_m^M X_m = p_i. \tag{23}$$
Again, it is left to the reader to verify that this is a viable and sensical correspondence.

We have sketched how the language recognised by R is the same as that recognised by R_U. In the other direction, the correspondence is less involved. Suppose
$$\langle\langle s_0^M, [\epsilon]\rangle, \epsilon, w_0\rangle, \ldots, \langle\langle s_n^M, [p_n]\rangle, \Sigma_n, w_n\rangle \tag{24}$$
is a computation of R_U. The corresponding computation of R is given by
$$\langle s_0^M, \epsilon, w_0\rangle, \ldots, \langle s_n^M, \sigma_n, w_n\rangle \tag{25}$$
where if $\Sigma_j = \langle\langle t_1^j, [q_1^j]\rangle, Y_1\rangle, \ldots, \langle\langle t_{l_j}^j, [q_{l_j}^j]\rangle, Y_{l_j}\rangle$ for $1 \le j \le n$, then the corresponding $\sigma_j = \langle t_1^j, Y_1\rangle, \ldots, \langle t_{l_j}^j, Y_{l_j}\rangle$. (The t_k^j, $1 \le k \le l_j$, $1 \le j \le n$, are states of the original CFSM, M.)

The fact that corresponding computations arrive at accepting configurations simultaneously for the same set of input strings can be seen from the definition of our unfolded machine, M_U, completing the proof of lemma 1.

To prove theorem 1, it remains to define the transition function, Φ, of our unfolded, flattened machine, M_N. Φ is given by:

1. If $\delta_{M_U}(s_1^{M_U}, a) = s_2^{M_U}$, then $s^{M_U} \in \Phi(s_1^{M_U}, a)$.
2. If $s_2^{M_U} \in \mathit{Reduce}(s_1^{M_U})$, then $s^{M_U} \in \Phi(s_1^{M_U}, a)$.

For s^{M_U} a state of our unfolded CFSM, the set $\mathit{Reduce}(s^{M_U})$ contains all states t^{M_U} such that there is a completed dotted rule $X \to X_1 \cdots X_n\cdot$ in s^{M_U} and k states $s_1^{M_U}, \ldots, s_k^{M_U}$ such that
$$s_{i+1}^{M_U} = \delta_{M_U}(s_i^{M_U}, X_i) \tag{26}$$
$$s^{M_U} = s_n^{M_U} \tag{27}$$
$$s_1^{M_U} = \delta_{M_U}(s^{M_U}, X_1) \tag{28}$$
$$t^{M_U} = \delta_{M_U}(s^{M_U}, X) \tag{29}$$
all obtain. ($\mathit{Reduce}(s^{M_U})$ is the set of all states that can be reached from s^{M_U} by a reduce move.)

For $w \in L(G)$, w is accepted by R. Lemma 1 shows w is also accepted by R_U and hence $L(G) \subseteq L(M_N)$, as required by the theorem.[2]

[2] For a more detailled proof of similar results, see [4], pages 249-250.

3.2 Exactness to Arbitrary Recursive Depths

In this section, we will show the following:

Theorem 2 *For an arbitrary CF grammar, G, and string $w \in A^*$ of length $|w| \leq W$, there is a natural number N such that*

$$w \in L(G) \Leftrightarrow w \in L(M_N) \quad (30)$$

where M_N is the unfolded, flattened FSM corresponding to N.

We again consider the case of the non-deterministic FSM, M_N, for simplicity. We have shown in the previous section that $L(G) \subseteq L(M_K)$ for any natural number K, thus we must choose a natural number N, and prove that for this N, $L(M_N) \subseteq L(G)$.

In order to show this, we will need some concept of *distance* within an FSM. Let us consider a *breadth–first* search of an FSM, F. Each state, s_i of F will be visited at least once during our search. We define the *distance* (or *norm*), $\|s_i\|$ of s_i to be the number of steps it takes to reach s_i for the first time from the start state, s_0.

Now let us consider the CFSM for G, M, and its unfolded correlative, M_{U_K}. We use the notation M_{U_K} to indicate the machine resulting when M is unfolded to a depth of K, as opposed to the unfolded, *flattened* machine, M_K. We then have the following result.

Lemma 2 *For any state, $s_i^{M_{U_K}}$, of the unfolded machine, M_{U_K}, such that $\|s_i^{M_{U_K}}\| \leq K$, if p and p' are paths leading from $s_0^{M_{U_K}}$ to $s_i^{M_{U_K}}$, then $p = p'$. (No two* distinct *paths lead to $s_i^{M_{U_K}}$.) Furthermore, the path leading to $s_i^{M_{U_K}}$ has no loops.*

Brief consideration of our unfolding method reveals that both claims must be the case. Each time distinct paths are encountered (including distinct paths owing to repeated traversals of loops), the destination state is split accordingly within the unfolded machine. Note, however, that

1. Paths are obtained in M_{U_K} by following *either* terminals *or* nonterminals, and
2. in actuality, unfolding M takes place by considering K *loops*. In general, therefore, paths may be unique to a distance far surpassing K. However, for paths containing no loops, we may be certain of their uniqueness only to a distance of K from the start state.

Let us now consider the unfolded, *flattened* machine, M_K. Arcs labelled with non–terminals will have been deleted, but we will carry over the function $\|\cdot\|$ from M_{K_U}. That is, if $s_i^{M_K}$ is the correlative of $s_i^{M_{U_K}}$ (we write $s_i^{M_K} \approx s_i^{M_{U_K}}$), then $\|s^{M_K}\| = \|s_i^{M_{U_K}}\|$. This definition makes sense, as the process of flattening has added no new states.

We are now ready to prove theorem 2. Let G be an arbitrary CF grammar. Consider the production rules of G,[3]and choose a production, $X \to X_1 \cdots X_n$ such that for any other production $Y \to Y_1 \cdots Y_m$, $m \leq n$. Let $N = W + n$. We claim M_N is sufficiently unfolded that $w \in L(M_N), |w| \leq W \Rightarrow w \in L(G)$.

[3] We assume in the proof of theorem 2 that there are no transitions of the form $X \to \cdot$. The proof can be extended to cover this case, but requires constraints on the "recursive depth of a string w", i.e, on the number of steps in the shortest derivation of w. Such constraints can be derived from the fact that $|w|$ is finite.

Let $w \in L(M_N)$, $|w| \leq W$, and let $t_0^{M_N}, t_1^{M_N}, \ldots, t_f^{M_N}$ be an accepting path for w in M_N. Recall each state $t_i^{M_N}$ of the unfolded, flattened machine is in fact a pair $\langle s_i^M, [p_i]\rangle$ consisting of a state of the original CFSM for G, M, and a path equivalence class at that state. Then

$$t_0^{M_N} = \langle s_0^M, [\epsilon]\rangle \quad (31)$$

where s_0^M is the start state of M and ϵ is the empty path,

$$t_f^{M_N} = \langle S_f^M, [p_f]\rangle \quad (32)$$

where s_f^M is a final (accepting) state of M, and $t_{i+i}^{M_N}$ follows logically from $t_i^{M_N}$ according to Φ, the transition function of our unfolded, flattened machine (as given at the end of the previous section).

Consider the shift–reduce pushdown recogniser, R_{U_N}, corresponding to our unfolded CFSM M_{U_N}. We will inductively define $\zeta_0, \zeta_1, \ldots, \zeta_f$, an accepting sequence of configurations for R_{U_N}, using the $\langle t_i^{M_K}\rangle_{0 \leq i \leq f}$. Let Let φ be the transition function for M_{U_N} as in equation 1 (section 2).

We define

$$\zeta_0 = \langle t_0^{M_{U_N}}, \epsilon, w\rangle \quad (33)$$

where $t_0^{M_{U_N}} \approx t_0^{M_N}$.

Next, we consider the two types of possible transitions.

Case 1 Given $\zeta_i = \langle t_i^{M_{U_N}}, \Sigma_i w_i\rangle$ and $t_{i+1}^{M_N} \in \Phi(t_i^{M_N}, a)$ for some terminal symbol $a \in A$. In this case, we define

$$\zeta_{i+1} = \langle t_{i+1}^{M_{U_N}}, \Sigma_i \langle t_i^{M_{U_N}}, a\rangle, w_i - a\rangle \quad (34)$$

where $w_i - a$ is obtained by stripping the string w_i of the a appearing at its beginning. We assume (see lemma 3, below) $\varphi(t_i^{M_{U_N}}, a) = t_{i+1}^{M_{U_N}}$, so this definition makes sense.

Case2 Given $\zeta_i = \langle t_i^{M_{U_N}}, \Sigma_i w_i\rangle$ and $t_{i+1}^{M_N} \in \Phi(t_i^{M_N}, \epsilon)$. We assume $\varphi(t_i^{M_{U_N}}, a)$ is undefined for all $a \in A$ and that there is a unique ϵ–transition going from $t_i^{M_N}$ to $t_{i+1}^{M_N}$ in M_N and will prove this assumption below (see lemma 3). In this case, we stipulate

$$\zeta_{i+1} = \langle t^{M_{U_N}}, \Sigma_{i+1}, w_i\rangle \quad (35)$$

where the stack Σ_{i+1} is derived as follows. We know

$$t_{i+1}^{M_N} = \langle s_{i+1}^M, [p_{i+1}]\rangle. \quad (36)$$

Suppose $p_{i+1} = s_0^M X_0 r_1^M X_1 \cdots r_k^M X_k s_{i+1}^M$. then Σ_{i+1} is obtained by stripping Σ_i of pairs $\langle t_j^{M_{U_N}}, Y_j\rangle$ until the first pair $\langle t_J^{M_{U_N}}, Y_J\rangle$ is found such that $t_J^{M_{U_N}} = r_k^M$ (r_k^M is the third to last symbol in the path p_{i+1}). We then push the pair $\langle t_i^{M_{U_N}}, Y_J\rangle$ onto the resultant stack, yielding Σ_{i+1}.

We must now show that the definitions 34 and 35 are well-defined and, in addition, these two cases exhaust the possible situations. To do this, we require the following lemma.

Lemma 3 *Let G, w, N and M_N be as above. Let the sequence $t_0^{M_N}, \ldots, t_f^{M_N}$ be an accepting path for w in M_N (as above). Then $\|t_i^{M_N}\| \leq W$ for $0 \leq i \leq f$.*

We use induction to prove lemma 3. For $t_0^{M_N}$ (the start state of M_N), clearly $\|t_0^{M_N}\| = 0 \leq W$. Suppose $\|t_i^{M_N}\| \leq W$ and consider $t_{i+1}^{M_N}$.

Case 1 $t_{i+1}^{M_N}$ follows from $t_i^{M_N}$ by following an arc labelled with the terminal $a \in A$, where a appears in w. In this case,

$$\|t_{i+1}^{M_N}\| = \|t_i^{M_N}\| + 1. \qquad (37)$$

Note, however, that since $\|w\| \leq W$, there are at most W such transitions. Thus $\|t_{i+1}^{M_N}\| \leq W$ for $0 \leq i \leq f$ just in case the ϵ–transitions (below) add no further depth to the $\langle t_i^{M_N} \rangle_{0 \leq i \leq f}$.

Case 2 $t_{i+1}^{M_N}$ follows from t^{M_N} by following an ϵ–transition. Suppose $\|t_i^{M_N}\| = k$. Now, $t_i^{M_N} = \langle s_i^{M_N}, [p_i] \rangle$, and each ϵ–transition inserted in M_N correspondes to a completed dotted rule contained in the corresponding state of the original CFSM.

Suppose the completed dotted rule in question is of the form $X \to X_1 \cdots X_j$. Let

$$k' = k - j + 1. \qquad (38)$$

Then $t_{i+1}^{M_N}$ can be reached in at least k' steps. (In M_{U_N}, follow the transitions for $t_i^{M_{U_N}}$ up to–but not including–the transition on X_1, then follow X to $t_{i+1}^{M_{U_N}}$ instead.) Hence $\|t_{i+1}^{M_N}\| \leq k' < k \leq W$.

It is this lemma that renders our definition of the sequence $\zeta_0, \ldots, \zeta_f$ defined by equations 34 and 35 both well-defined and exhaustive. Note that $\|t_i^{M_N}\| \leq W$ in particular implies $\|t^{M_N}\| \leq N$ for our choice of N.

First, consider the well-definedness of equations 34 and 35. Since each $t_i^{M_N}$ is such that $\|t_i^{M_N}\| \leq N$, lemma 2 tells us that $\Phi(t_i^{M_N}, a)$ $(a \in A)$ and $\Phi(t_i^{M_N}, \epsilon)$ are singleton sets (i.e, uniquely determined). Thus the $t_i^{M_{U_N}}$ are uniquely determined.

Also, since $\|t_i^{M_N}\| \leq N$, lemma 2 tells us that for each $t_i^{M_N} = \langle s_i^M, [p_i] \rangle$, p_i is uniquely determined, and that it contains no loops (in particular, no repeated states). Thus the stack Σ_{i+1} defined in 35 indeed contains the information the shift–reduce parser $R_{M_{U_N}}$ requires.

Finally, we consider the exhaustiveness of our definition of the sequence $\zeta_0, \ldots, \zeta_f$. Could it occur that there is a shift-reduce ambivalence in the sequence $t_0^{M_N}, \ldots, t_f^{M_N}$? The negative answer arises from the "$+n$" term in our choice of N.

Let us consider a fixed $t_i^{M_N}$. Suppose $X \to X_1 \cdots X_j$ is a production of G having a right side of length $j \leq n$, and the completed dotted rule

$$X \to X_1 \cdots X_f \cdot \qquad (39)$$

is contained in $t_i^{M_N}$. Suppose (in order to derive a contradiction) that there is an ϵ–transition going from state $t_i^{M_N}$ to state $t_{i+1}^{M_N}$, corresponding to the completed dotted rule in. Suppose following each of the X_k on the right hand side of rule "backwards" through M_{U_N} gets us to state $t_l^{M_N}$.

Then $\|t_l^{M_N}\| \leq W$, by lemma 2. There is a transition from on X from the state $t_l^{M_{U_N}}$ to the state $t_{i+1}^{M_{U_N}}$, because our flattening process causes ϵ–transitions to mimic the behaviour of reduce moves. However, this means we now have two paths from $t_0^{M_{U_N}}$ to state $t_{i+1}^{M_{U_N}}$: one via state $t_l^{M_{U_N}}$, and the other via state $t_i^{M_{U_N}}$, contradicting lemma 2. Thus there are no shift-reduce conflicts in the sequence $t_0^{M_N}, \ldots, t_f^{M_N}$, and hence each ζ_j can be defined solely in terms of the mutually exclusive cases 34 and 35.

The fact that ζ_f reaches an accepting state within the unfolded machine M_{U_N} is readily seen from the facts that $t_f^{M_N} \approx t_f^{M_{U_N}}$ and $t_f^{M_N}$ is an accepting state.

Lemma 1 (section 3.2) gives that $L(R_{U_N}) = L(R)$ where R is the original shift–reduce pushdown recogniser corresponding to the CFSM M. Hence $w \in L(R_{U_N}) \Rightarrow w \in L(R)$, completing the proof of theorem 2.

3.3 Worst-Case Costs of Unfolding

For a general FSM, M, the worst-case cost of unfolding occurs when there is a transition from every state to every state (including transitions from a state to itself). Figure 7 illustrates the case for a machine with 3 states.

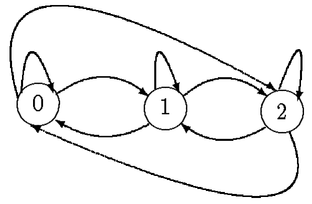

Figure 7. Worst case FSM, $N = 3$

In this case, for a machine with m states and for an unfolding criterion N, there are on the order of m^{m^N} paths that must be generated and compared with the unfolding criterion. Clearly this very quickly outstrips the limits of computational feasibility.

This worst case rarely occurs with the simplest grammars. For example, in the case of the CFSM for grammar G_2, the number of paths through the CFSM that must be considered grows linearly with N.

However, in the general case, the full unfolding costs are problematic for even small CFSMs. such as the one corresponding to the grammar G_3 given by:

$$\begin{gathered} NP \to DetNom|PN \\ Det \to Art|NP's \\ Nom \to N|NomPP|AdjNom \\ PP \to PNP \end{gathered}$$

where the symbols Art, N, PN and P correspond to the parts of speech "article", "noun", "proper noun" and "preposition".[4]

[4] See [4], page 253.

4 COMPUTATIONAL RESULTS

The techniques described in this paper have all been implemented (in ANSI C, running on both PCs and UNIX workstations) and tested on a variety of small grammars.

Unfolding has proven too computationally demanding in most cases of CFSMs having any real interest for natural language processing concerns. The poor computational behaviour of the algorithm leads us to consider a variation on the unfolding algorithm, consisting in following only *terminal* symbols when unfolding the original CFSM. This modification works well in practise:

- No change is made to the unfolding algorithm in the case of grammar G_2.
- For grammar G_3, the revised unfolding method concurs with Pereira and Wright's results and renders unfolding possible to depths surpassing $N = 1000$ on a simple PC.
- For many natural language grammars, unfolding time is significantly reduced.

Table 1 shows preliminary results using both the unmodified and modified unfolding algorithms. The modified unfolding algorithm, which occupies a position between Pereira and Wright's approximation algorithm and our algorithm in terms of representational power.

Table 1. Results on a standard PC (Intel 40486, 25MHz, 8Mb RAM).

Grammar	N	Time (m=minutes, s=seconds)	
		Full Unfolding	Modified Unfolding
G_1	1	$0.23s$	$0.36s$
	k (arbitrary)	$0.23s$	$0.36s$
G_2	1	$1.88s$	$2.56s$
	10	$8.00s$	$5.29s$
	100	$3m12.81s$	$3m43.67s$
G_3	1	$1m23.43s$	$6.83s$
	10	$> 36hours$	$9.56s$
	100	*years*	$1m43.45s$

Initial results for G_{50}, a grammar approximating a large subset of the English language whose CFSM has 50 states, show that the modified unfolding algorithm produces results in $18h40m10.0s$ on a large SUNstation, while the full unfolding algorithm takes longer than $48h$.

5 RELATED WORK AND CONCLUSIONS

In [2], Black mentions that finite–state machines can approximate CF grammars exactly to finite recursive depths. The unfolding criterion can be seen as inducing an algorithmic realisation of this idea. The exponential blow–up of paths that must be generated is a consequence of the inherrent exponential nature of the problem. Unlike Black's method of realising the approximating power of finite–state machines, the method presented here is exact to finite recursives depths, and always sound beyond these depths.

The idea behind flattening, and the term "unfolding" are taken from [4]. Pereira and Wright unfold CFSMs according to "stack congruences". Their method of approximation is exact for left- and right-linear CFGs, as well as a variety of other cases. It fails on G_2, however, and many other self–embedding CFGs.

Pereira and Wright mention that it would be interesting to have some concept of "degree of approximation" of a CF language by a finite–state machine. The current implementation of our unfolding algorithm expands paths to a depth of N along transitions on terminals and to a depth of 0 otherwise. Since the full unfolding method provides exact approximation to a recursive depth of N, this modified version might be called "one–level N–unfolding". By unfolding different sub-machines of the CFSM to different values of N, one can arrive at a series of "M–level N–unfolding" algorithms, providing several heierarchies of approximation to the original language.

ACKNOWLEDGEMENTS

The author wishes to thank the anonymous ACL96 referees whose comments helped improve the clarity of this paper. Similarly, she acknowledges Dr. John Carroll, Oliver Seidel, and all others at the University of Cambridge computer laboratory who helped with the conception and execution of this paper.

REFERENCES

[1] Alfred V. Aho, Ravi Sethi and Jeffrey D. Ullman. 1986. *Compilers: Principles, Techniques, and Tools.* Addison–Wesley, Wokingham, England.

[2] Alan W. Black. 1989. "Finite State Machines from Feature Grammars". In Masaru Tomita, editor, *International Workshop on Parsing Technologies*, pages 277–285. Carnegie Mellon University Press, Pittsburgh, Pennsylvania.

[3] John E. Hopcroft and Jeffrey D. Ullman. 1971. *Introduction to Automata Theory, Languages, and Computation.* Addison–Wesley, Wokingham, England.

[4] Fernando C. N. Pereira and Rebecca N. Wright. 1991. "Finite-State Approximation of Phrase Structure Grammars". *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 246–255, Berkeley, California. Association for Computational Linguistics, Morristown, New Jersey.