REGULAR CLOSURE OF DETERMINISTIC LANGUAGES*

EBERHARD BERTSCH† AND MARK-JAN NEDERHOF‡

Abstract. We recall the notion of regular closure of classes of languages. We present two important results. The first one is that all languages which are in the regular closure of the class of deterministic (context-free) languages can be recognized in linear time. This is a non-trivial result, since this closure contains many inherently ambiguous languages.

The second one is that the class of deterministic languages is contained in the closure of the class of deterministic languages with the prefix property, or stated in an equivalent way, all LR(k) languages are in the regular closure of the class of LR(0) languages.

Key words. context-free languages, regular languages

AMS subject classifications. 68Q45, 68Q15

1. Introduction. In spite of important insights in applying sophisticated matrix operations to recognition techniques [16], the known upper bound on the time complexity of context-free language recognition still exceeds $\mathcal{O}(n^2)$, measured in the length of the input string. On the other hand, there are many languages whose time complexity has been shown to be linear by means of *specialized* algorithms but which are not recognized in linear time by *general* recognition algorithms. The frontier of knowledge in this area is moved on by the results of the present article.

We introduce a special class of two-level automata. Their upper level is constituted by a (classical) finite automaton whose transitions are, however, not enabled by a single terminal symbol but by any element of a given (lower level) language. All languages at the lower level are assumed to have the restrictive LR(k)-property. Due to the well-known correspondence between regular expressions and finite automata, the new class of languages may thus be stated to result from the set of deterministic (i.e. LR(k)) languages by recursively applying concatenation, union, and Kleene star to given languages in that class.

The new class contains some notorious specimens such as $\{a^m b^m c^n\} \cup \{a^m b^n c^n\}$, an inherently ambiguous language [8]. As will be developed in the body of this paper, a linear upper time bound on recognizing such languages can be established.

The results are related to previous work by the authors [13] on efficient recognition of language suffixes which had in turn been motivated by practical syntax error detection (more precisely, by non-correcting error recovery [15]).

The article may be outlined as follows: After introducing the definitional framework in Section 2, we show that all deterministic context-free languages can be constructed from prefix-free deterministic languages by regular operations (Section 3). This follows from a detailed decomposition of pushdown computations into sequences of moves that do not discard more than one element of the stack they started with.

---

*Portions of this paper appear in a modified form as "M.J. Nederhof and E. Bertsch, An innovative finite state concept for recognition and parsing of context-free languages. In András Kornai (ed): *Extended finite state models of language*, Cambridge University Press, 1998", an abstract of which has appeared in *Natural Language Engineering*, 2(4), 1996, pp. 381–382.

†Ruhr University, Faculty of Mathematics, D-44780 Bochum, Germany (`bertsch@lpi.ruhr-uni-bochum.de`).

‡DFKI, Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany (`nederhof@dfki.de`). Research by the second author was carried out within the framework of the Priority Programme Language and Speech Technology (TST), while working at the Faculty of Arts of the University of Groningen. The TST-Programme is sponsored by NWO (Dutch Organization for Scientific Research).

The most important proposition of Section 4 is that each deterministic pushdown automaton can be transformed into an equivalent one which is "loop-free". This requires a fairly deep discussion of individual pushing and popping moves. In essence, automata are changed in such a way that the stack contents after certain moves must reflect the *amount* of processed input. Using tabulation, the transformed pushdown automata can be simulated at all input positions in linear time. This result is part of a proof of the fact that languages in the regular closure of the deterministic languages can be recognized in linear time, by means of a two-level device, to be defined in Section 5.

Section 6 deals with various syntheses and applications of the obtained results. In particular, "on-line" and "off-line" variants of the two-level device are presented and compared. The difference between these notions is rather similar to the one between Earley's recognition algorithm [7] and the Cocke-Kasami-Younger technique [8]. With the latter approach, partial recognition results are collected without reference to their left context.

Although the concept of parse tree is less immediate for the new kind of language description than for ordinary grammars, we are able to sketch an efficient transduction procedure yielding representations of the syntactic structure of given inputs (Section 7).

Two applications are presented in Section 8. First we prove that suffix recognition is possible in linear time. This new proof is much shorter than recently published proofs of this fact. We then describe an application in pattern matching.

2. Notation. A finite automaton $\mathcal{F}$ is a 5-tuple (S, Q, q_s, F, T), where S and Q are finite sets of input symbols and states, respectively; $q_s \in Q$ is the *initial* state, $F \subseteq Q$ is the set of *final* states; the transition relation T is a subset of $Q \times S \times Q$.

An input $b_1 \cdots b_m \in S^*$, is *recognized* by the finite automaton if there is a sequence of states $q_0, q_1, \ldots, q_m$ such that $q_0 = q_s$, $(q_{k-1}, b_k, q_k) \in T$ for $1 \leq k \leq m$, and $q_m \in F$. For a certain finite automaton $\mathcal{F}$, the set of all such strings w is called the language *accepted by* $\mathcal{F}$, denoted $L(\mathcal{F})$. The languages accepted by finite automata are called the *regular* languages.

In the following, we describe a type of pushdown automaton without internal states and with very simple kinds of transition. This is a departure from the standard literature (e.g. [9]), but considerably simplifies our definitions and proofs in the remainder of the paper. That the generative capacity of this type of pushdown automaton is not affected with respect to any of the more traditional types can be argued by the fact that every context-free language is accepted by a nondeterministic LR recognizer of a form very similar to our type of pushdown automaton [14]. See also [13].

Thus, we define a pushdown automaton (PDA) $\mathcal{A}$ to be a 5-tuple $(\Sigma, \Delta, X_{initial}, F, T)$, where Σ, Δ and T are finite sets of input symbols, stack symbols and transitions, respectively; $X_{initial} \in \Delta$ is the *initial* stack symbol, $F \subseteq \Delta$ is the set of *final* stack symbols.

We consider a fixed input string $a_1 \cdots a_n \in \Sigma^*$. A *configuration* of the automaton is a pair (δ, v) consisting of a stack $\delta \in \Delta^*$ and the remaining input v, which is a suffix of the original input string $a_1 \cdots a_n$.

The *initial* configuration is of the form $(X_{initial}, a_1 \cdots a_n)$, where the stack is formed by the initial stack symbol $X_{initial}$. A *final* configuration is of the form $(\delta X, \epsilon)$, where the element on top of the stack is some final stack symbol $X \in F$.

The transitions in T are of the form $X \overset{z}{\mapsto} XY$, where $z = \epsilon$ or $z = a$, or of the

form $XY \overset{\epsilon}{\mapsto} Z$.

The application of such a transition $\delta_1 \overset{z}{\mapsto} \delta_2$ is described as follows. If the topmost symbols on the stack are δ_1, then these may be replaced by δ_2, provided either $z = \epsilon$, or $z = a$ and a is the first symbol of the remaining input. If $z = a$ then furthermore a is removed from the remaining input.

Formally, for a fixed PDA we define the binary relation $\vdash$ on configurations as the least relation satisfying $(\delta\delta_1, v) \vdash (\delta\delta_2, v)$ if there is a transition $\delta_1 \overset{\epsilon}{\mapsto} \delta_2$, and $(\delta\delta_1, av) \vdash (\delta\delta_2, v)$ if there is a transition $\delta_1 \overset{a}{\mapsto} \delta_2$.

In the case that we consider more than one PDA at the same time, we use symbols $\overset{z}{\mapsto}_{\mathcal{A}}$ and $\vdash_{\mathcal{A}}$ instead of $\overset{z}{\mapsto}$ and $\vdash$ if these refer to one particular PDA $\mathcal{A}$.

The recognition of a certain input v is obtained if starting from the initial configuration for that input we can reach a final configuration by repeated application of transitions, or, formally, if $(X_{initial}, v) \vdash^* (\delta X, \epsilon)$, with some $\delta \in \Delta^*$ and $X \in F$, where $\vdash^*$ denotes the reflexive and transitive closure of $\vdash$ (and $\vdash^+$ denotes the transitive closure of $\vdash$). For a certain PDA $\mathcal{A}$, the set of all such strings v which are recognized is called the language *accepted by* $\mathcal{A}$, denoted $L(\mathcal{A})$. A PDA is called *deterministic* if for all possible configurations at most one transition is applicable. The languages accepted by deterministic PDAs (DPDAs) are called *deterministic languages*.

We may restrict deterministic PDAs such that no transitions apply to final configurations, by imposing $X \notin F$ if there is a transition $X \overset{z}{\mapsto} XY$, and $Y \notin F$ if there is a transition $XY \overset{\epsilon}{\mapsto} Z$. We call such a DPDA *prefix-free*. The languages accepted by such deterministic PDAs are obviously *prefix-free*, which means that no string in the language is a prefix of any other string in the language. Conversely, any prefix-free deterministic language is accepted by some prefix-free DPDA, the proof being that in a deterministic DPDA, all transitions of the form $X \overset{z}{\mapsto} XY$, $X \in F$, and $XY \overset{\epsilon}{\mapsto} Z$, $Y \in F$, can be removed without consequence to the accepted language if this language is prefix-free.

In compiler design, the deterministic languages are better known as LR(k) languages, and the prefix-free deterministic languages as LR(0) languages [9].

A prefix-free DPDA is in normal form if, for all input v, $(X_{initial}, v) \vdash^* (\delta X, \epsilon)$, with $X \in F$, implies $\delta = \epsilon$, and furthermore F is a singleton $\{X_{final}\}$. Any prefix-free DPDA can be put into normal form. (See [9, Theorem 5.1] for a proof of a related result.) We define a *normal PDA* (NPDA) to be a prefix-free deterministic PDA in normal form.

We define a subrelation $\models^+$ of $\vdash^+$ as: $(\delta, vw) \models^+ (\delta\delta', w)$ if and only if $(\delta, vw) = (\delta, z_1 z_2 \cdots z_m w) \vdash (\delta\delta_1, z_2 \cdots z_m w) \vdash \ldots \vdash (\delta\delta_m, w) = (\delta\delta', w)$, for some $m \geq 1$, where $|\delta_k| > 0$ for all k, $1 \leq k \leq m$. Informally, we have $(\delta, vw) \models^+ (\delta\delta', w)$ if configuration $(\delta\delta', w)$ can be reached from (δ, vw) without the bottom-most part δ of the intermediate stacks being affected by any of the transitions; furthermore, at least one element is pushed on top of δ. Note that $(\delta_1 X, vw) \models^+ (\delta_1 X\delta', w)$ implies $(\delta_2 X, vw') \models^+ (\delta_2 X\delta', w')$ for any δ_2 and any w', since the transitions do not address the part of the stack below X, nor read the input following v.

3. Meta-deterministic languages. In this section we define a new sub-class of the context-free languages, which results from combining deterministic languages by the operations used to specify regular languages.

We first define the concept of *regular closure* of a class of languages.[1] Let $\mathcal{L}$ be a

[1] This notion was called *rational* closure in [3].

class of languages. The regular closure of $\mathcal{L}$, denoted $C(\mathcal{L})$, is defined as the smallest class of languages such that:

(i) $\emptyset \in C(\mathcal{L})$,
(ii) if $l \in \mathcal{L}$ then $l \in C(\mathcal{L})$,
(iii) if $l_1, l_2 \in C(\mathcal{L})$ then $l_1 l_2 \in C(\mathcal{L})$,
(iv) if $l_1, l_2 \in C(\mathcal{L})$ then $l_1 \cup l_2 \in C(\mathcal{L})$, and
(v) if $l \in C(\mathcal{L})$ then $l^* \in C(\mathcal{L})$.

Note that a language in $C(\mathcal{L})$ may be described by a regular expression over symbols representing languages in $\mathcal{L}$.

Let $\mathcal{D}$ denote the class of deterministic languages. Then the class of *meta-deterministic* languages is defined to be its regular closure, $C(\mathcal{D})$. This class is obviously a subset of the class of context-free languages, since the class of context-free languages is closed under concatenation, union and Kleene star, and it is a *proper* subset, since, for example, the context-free language $\{ww^R \mid w \in \{a,b\}^*\}$ is not in $C(\mathcal{D})$. (w^R denotes the mirror image of w.)

Finite automata constitute a computational representation for regular languages; DPDAs constitute a computational representation for deterministic languages. By combining these two mechanisms we obtain the meta-deterministic automata, which constitute a computational representation for the meta-deterministic languages.

Formally, a meta-deterministic automaton $\mathcal{M}$ is a triple $(\mathcal{F}, A, \mu)$, where $\mathcal{F} = (S, Q, q_s, F, T)$ is a finite automaton, A is a finite set of deterministic PDAs with identical alphabets Σ, and μ is a mapping from S to A.

The language accepted by such a device is composed of languages accepted by the DPDAs in A according to the transitions of the finite automaton $\mathcal{F}$. Formally, a string v is *recognized by* automaton $\mathcal{M}$ if there is some string $b_1 \cdots b_m \in S^*$, a sequence of PDAs $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m \in A$, and a sequence of strings $v_1, \ldots, v_m \in \Sigma^*$ such that

(i) $b_1 \cdots b_m \in L(\mathcal{F})$,
(ii) $\mathcal{A}_k = \mu(b_k)$, for $1 \leq k \leq m$,
(iii) $v_k \in L(\mathcal{A}_k)$, for $1 \leq k \leq m$, and
(iv) $v = v_1 \cdots v_m$.

The set of all strings recognized by automaton $\mathcal{M}$ is called the language *accepted by* $\mathcal{M}$, denoted $L(\mathcal{M})$.

EXAMPLE 3.1. As a simple example of a language accepted by a meta-deterministic automaton, consider $L = L_1 \cup L_2$, where $L_1 = \{a^m b^n c^n \mid n, m \in \{0, 1, \ldots\}\}$ and $L_2 = \{a^m b^m c^n \mid n, m \in \{0, 1, \ldots\}\}$. It is well-established that L is not a deterministic language [9, Example 10.1]. However, it is the union of two languages L_1 and L_2, which are by themselves deterministic. Therefore, L is accepted by a meta-deterministic automaton $\mathcal{M}$ which uses two DPDAs $\mathcal{A}_1$ and $\mathcal{A}_2$, accepting L_1 and L_2, respectively.

We may for example define $\mathcal{M}$ as $(\mathcal{F}, \{\mathcal{A}_1, \mathcal{A}_2\}, \mu)$ with $\mathcal{F} = (S, Q, q_s, F, T)$, where

(i) $S = \{b_1, b_2\}$,
(ii) $Q = \{q_s, q_f\}$,
(iii) $F = \{q_f\}$,
(iv) $T = \{(q_s, b_1, q_f),\ (q_s, b_2, q_f)\}$, and
(v) $\mu(b_1) = \mathcal{A}_1$ and $\mu(b_2) = \mathcal{A}_2$.

A graphical representation for $\mathcal{M}$ is given in Figure 3.1. States $q \in Q$ are represented by vertices labelled by q, triples $(q, b, p) \in T$ by arrows from q to p labelled by $\mu(b)$. □

That the meta-deterministic automata precisely accept the meta-deterministic

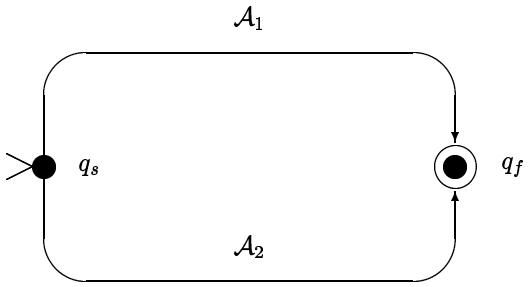

FIG. 3.1. *A meta-deterministic automaton.*

languages is reflected by the following equation.

$$C(\mathcal{D}) = \{L(\mathcal{M}) \mid \mathcal{M} \text{ is a meta-deterministic automaton}\}$$

This equation straightforwardly follows from the equivalence of finite automata and regular expressions, and the equivalence of deterministic pushdown automata and deterministic languages.

Let $\mathcal{N}$ denote the class of prefix-free deterministic languages. In the same vein, we have

$$C(\mathcal{N}) = \{L(\mathcal{M}) \mid \mathcal{M} = (\mathcal{F}, A, \mu) \text{ is a meta-deterministic automaton where } A \text{ is a set of normal PDAs}\}$$

In the sequel, we set out to prove a number of properties of languages in $C(\mathcal{D})$, represented by their meta-deterministic automata (i.e. their corresponding recognition devices). The DPDAs in an arbitrary such device cause some technical difficulties which may be avoided if we restrict ourselves to meta-deterministic automata which use only normal PDAs, as opposed to arbitrary deterministic PDAs. Fortunately, this restriction does not reduce the class of languages that can be described, or in other words, $C(\mathcal{N}) = C(\mathcal{D})$. We prove this equality below.

Since $C(\mathcal{N}) \subseteq C(\mathcal{D})$ is vacuously true, it is sufficient to argue that $\mathcal{D} \subseteq C(\mathcal{N})$, from which $C(\mathcal{D}) \subseteq C(C(\mathcal{N})) = C(\mathcal{N})$ follows using the closure properties of C, in particular monotonicity and idempotence.

We prove that $\mathcal{D} \subseteq C(\mathcal{N})$ by showing how for each DPDA $\mathcal{A}$ a meta-deterministic automaton $\rho(\mathcal{A}) = (\mathcal{F}, A, \mu)$ may be constructed such that A consists only of prefix-free deterministic PDAs, and $L(\rho(\mathcal{A})) = L(\mathcal{A})$. This construction is given by:

CONSTRUCTION 1 (DPDA to meta-deterministic automaton). Let $\mathcal{A} = (\Sigma, \Delta, X_{initial}, F_{\mathcal{A}}, T_{\mathcal{A}})$ be a deterministic PDA. Construct the meta-deterministic automaton $\rho(\mathcal{A}) = (\mathcal{F}, A, \mu)$, with $\mathcal{F} = (S, Q, q_s, F_{\mathcal{F}}, T_{\mathcal{F}})$, where

(i) $S = \{b_{X,Y} \mid X, Y \in \Delta\} \cup \{c_{X,Y} \mid X, Y \in \Delta\}$,
(ii) $Q = \Delta$,
(iii) $q_s = X_{initial}$,
(iv) $F_{\mathcal{F}} = F_{\mathcal{A}}$,
(v) $T_{\mathcal{F}} = \{(X, b_{X,Y}, Y) \mid X, Y \in \Delta\} \cup \{(X, c_{X,Y}, Y) \mid X, Y \in \Delta\}$.

The set A consists of (prefix-free deterministic) PDAs $\mathcal{B}_{X,Y}$ and $\mathcal{C}_{X,Y}$, for all $X, Y \in \Delta$, defined as follows.

Each $\mathcal{B}_{X,Y}$ is defined to be $(\Sigma, \{X^{in}, Y^{out}\}, X^{in}, \{Y^{out}\}, T)$, where X^{in} and Y^{out} are fresh symbols, and where the transitions in T are

$$X^{in} \overset{z}{\mapsto}_{\mathcal{B}_{X,Y}} X^{in}Y^{out} \quad \text{for all } X \overset{z}{\mapsto}_{\mathcal{A}} XY, \text{ some } z$$

Each $\mathcal{C}_{X,Y}$ is defined to be $(\Sigma, \Delta \cup \{X^{in}, Y^{out}\}, X^{in}, \{Y^{out}\}, T)$, where X^{in} and Y^{out} are fresh symbols, and where the transitions in T are those in $T_{\mathcal{A}}$ plus the extra transitions

$$\begin{array}{lll} X^{in} & \overset{z}{\mapsto}_{\mathcal{C}_{X,Y}} & X^{in}Z \quad \text{for all } X \overset{z}{\mapsto}_{\mathcal{A}} XZ, \text{ some } z \text{ and } Z \\ X^{in}Z & \overset{\epsilon}{\mapsto}_{\mathcal{C}_{X,Y}} & Y^{out} \quad \text{for all } XZ \overset{\epsilon}{\mapsto}_{\mathcal{A}} Y, \text{ some } Z \end{array}$$

The function μ maps the symbols $b_{X,Y}$ to automata $\mathcal{B}_{X,Y}$ and the symbols $c_{X,Y}$ to automata $\mathcal{C}_{X,Y}$.

Each automaton $\mathcal{B}_{X,Y}$ mimics a single transition of $\mathcal{A}$ of the form $X \overset{z}{\mapsto}_{\mathcal{A}} XY$. Formally, $\mathcal{B}_{X,Y}$ recognizes a string z if and only if $(X, z) \vdash_{\mathcal{A}} (XY, \epsilon)$.

Each automaton $\mathcal{C}_{X,Y}$ mimics a computation of $\mathcal{A}$ that replaces stack element X by stack element Y. Formally, $\mathcal{C}_{X,Y}$ recognizes a string v if and only if $(X, v) \models^{+}_{\mathcal{A}} (XZ, \epsilon) \vdash_{\mathcal{A}} (Y, \epsilon)$, for some $Z \in \Delta$.

For the proof, consider that recognition of v by $\mathcal{C}_{X,Y}$ means that $(X^{in}, v) \models^{+}_{\mathcal{C}_{X,Y}} (X^{in}Z, \epsilon) \vdash_{\mathcal{C}_{X,Y}} (Y^{out}, \epsilon)$, some Z, due to the nature of its transitions. This is equivalent to $(X^{in}, z) \vdash_{\mathcal{C}_{X,Y}} (X^{in}W, \epsilon)$, $(W, v') \vdash^{*}_{\mathcal{C}_{X,Y}} (Z, \epsilon)$, $(X^{in}Z, \epsilon) \vdash_{\mathcal{C}_{X,Y}} (Y^{out}, \epsilon)$, and $v = zv'$, some z, v' and W, due to the definition of $\models^{+}$. This is again equivalent to $(X, z) \vdash_{\mathcal{A}} (XW, \epsilon)$, $(W, v') \vdash^{*}_{\mathcal{A}} (Z, \epsilon)$, $(XZ, \epsilon) \vdash_{\mathcal{A}} (Y, \epsilon)$, and $v = zv'$, by virtue of the construction of $\mathcal{C}_{X,Y}$ from $\mathcal{A}$. Finally, this conjunction is equivalent to $(X, v) \models^{+}_{\mathcal{A}} (XZ, \epsilon) \vdash_{\mathcal{A}} (Y, \epsilon)$.

Note that the languages recognized by some of the $\mathcal{B}_{X,Y}$ and $\mathcal{C}_{X,Y}$ may be the empty set.

That all $\mathcal{B}_{X,Y}$ and $\mathcal{C}_{X,Y}$ are deterministic follows from the fact that by assumption $\mathcal{A}$ is deterministic. That all $\mathcal{B}_{X,Y}$ and $\mathcal{C}_{X,Y}$ are prefix-free follows from the fact that no transitions apply when a final stack symbol is on top of the stack.

To prove that $L(\rho(\mathcal{A})) = L(\mathcal{A})$ we have to show that

1. any sequence of transitions of the form $(X_{initial}, a_1 \cdots a_n) \vdash^{*}_{\mathcal{A}} (\delta X, \epsilon)$, with $X \in F_{\mathcal{A}}$, can be decomposed into a list of m sequences of transitions $(X^{in}_{k-1}, v_k) \vdash^{*}_{\mathcal{A}_k} (\delta_k X^{out}_k, \epsilon)$, $1 \leq k \leq m$, $X_0 = X_{initial}$ and $X_m = X$, using a list of m automata $\mathcal{A}_1, \ldots, \mathcal{A}_m \in A$ that recognize strings $v_1, \ldots, v_m$, respectively, where $a_1 \cdots a_n = v_1 \cdots v_m$ and such that the string $b_1 \cdots b_m$ with $\mu(b_k) = \mathcal{A}_k$, for $1 \leq k \leq m$, is recognized by $\mathcal{F}$.

2. Conversely, if we have a string $b_1 \cdots b_m$ recognized by $\mathcal{F}$, then we must show that any list of m sequences of transitions $(X^{in}_{k-1}, v_k) \vdash^{*}_{\mathcal{A}_k} (\delta_k Y^{out}_k, \epsilon)$ for automata $\mathcal{A}_k = \mu(b_k)$, $1 \leq k \leq m$, can be composed into a single sequence $(X_0, v_1 \cdots v_m) \vdash^{*}_{\mathcal{A}} (\delta Y_m, \epsilon)$, with $X_0 = X_{initial}$ and $Y_m \in F_{\mathcal{A}}$, using $X_k = Y_k$, $1 \leq k \leq m$.

The intuition behind decomposing a sequence of transitions for DPDA $\mathcal{A}$ is conveyed by Figure 3.2. We see the development of the stack, of which the height alternately increases and decreases during performance of the transitions. We assume the input is recognized at t9, where some final stack symbol X_9 is on top of the stack. We now locate the point in time t8 where the stack was of the same height for the last time before t9. Assume that at t8 some stack symbol X_8 is on top of the stack. The sequence of transitions from t8 to t9 is of the form $(X_8, v) \models^{+}_{\mathcal{A}} (X_8 Z, \epsilon) \vdash_{\mathcal{A}} (X_9, \epsilon)$, for some $Z \in \Delta$, which means that the stack development can be mimicked by the PDA $\mathcal{C}_{X_8,X_9}$. In Figure 3.2 the step between t7 and t8 is of the form $(X_7, z) \vdash_{\mathcal{A}} (X_7 X_8, \epsilon)$, which is mimicked by the PDA $\mathcal{B}_{X_7,X_8}$.

This can be continued until the complete sequence of transitions has been decomposed into a list of 9 sequences which are mimicked by PDAs of the form $\mathcal{B}_{X_{k-1},X_k}$ or $\mathcal{C}_{X_{k-1},X_k}$. The corresponding string $b_1 \cdots b_9$, where $b_k = b_{X_{k-1},X_k}$ or $b_k = c_{X_{k-1},X_k}$,

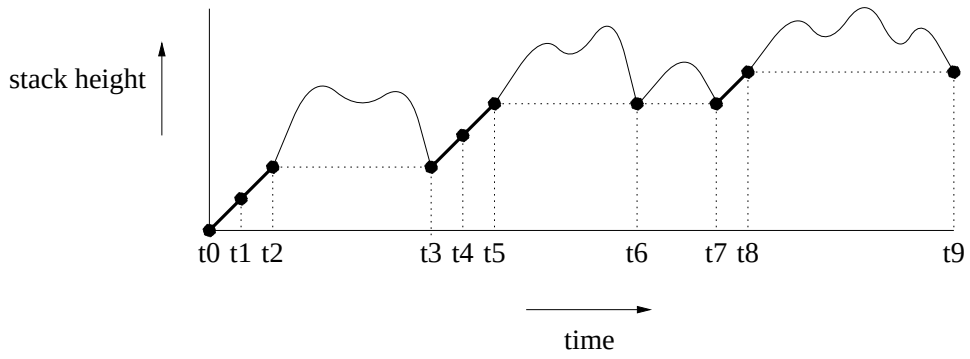

FIG. 3.2. *A stack development of a DPDA $\mathcal{A}$ on input $v_1 \cdots v_9$.*

$1 \leq k \leq 9$, then allows state $X_9 \in F_{\mathcal{F}} = F_{\mathcal{A}}$ to be reached from state q_s, or in other words, this string is recognized by finite automaton $\mathcal{F}$.

The proof of the general case uses induction on t. We show that if we have a sequence of transitions

$$(X_0, z_1 z_2 \cdots z_t) \vdash_{\mathcal{A}} (\delta_1 X_1, z_2 \cdots z_t) \vdash_{\mathcal{A}} \ldots \vdash_{\mathcal{A}} (\delta_{t-1} X_{t-1}, z_t) \vdash_{\mathcal{A}} (\delta_t X_t, \epsilon)$$

then for some m there are

(i) a list $Y_0, \ldots, Y_m \in \Delta$, with $Y_0 = X_0$ and $Y_m = X_t$,

(ii) a list $b_1, \ldots, b_m \in S$, such that $(Y_0, b_1, Y_1), (Y_1, b_2, Y_2), \ldots, (Y_{m-1}, b_m, Y_m) \in T$, and

(iii) a list $v_1, \ldots, v_m \in \Sigma^*$, such that $v_1 \cdots v_m = z_1 \cdots z_t$ and v_k is recognized by PDA $\mu(b_k)$, for $1 \leq k \leq m$.

The case that $t = 0$ can trivially be solved with $m = 0$; for $t > 0$ we distinguish between two cases:

(i) $|\delta_{t-1} X_{t-1}| < |\delta_t X_t|$, or in other words, the last step used a pushing transition, or

(ii) $|\delta_{t-1} X_{t-1}| > |\delta_t X_t|$, or in other words, the last step used a popping transition.

In the first case we may assume by definition that the automaton $\mathcal{B}_{X_{t-1}, X_t}$ recognizes z_t. The induction hypothesis for $t-1$, applied to $(X_0, z_1 z_2 \cdots z_{t-1}) \vdash^*_{\mathcal{A}} (\delta_{t-1} X_{t-1}, \epsilon)$, provides the required 3 lists with some $m-1$ instead of m. We set $Y_m = X_t$, $b_m = b_{X_{t-1}, X_t}$ (so that $\mu(b_m) = \mathcal{B}_{X_{t-1}, X_t}$), $v_m = z_t$, which gives us the required 3 lists for t. Note that $(X_0, z_1 \cdots z_{t-1}) \vdash^*_{\mathcal{A}} (\delta_{t-1} X_{t-1}, \epsilon)$ and $(X_0, z_1 \cdots z_{t-1} z_t) \vdash^*_{\mathcal{A}} (\delta_{t-1} X_{t-1}, z_t)$ are equivalent.

In the second case we may assume that there is a maximal $t' < t$ such that $|\delta_{t'} X_{t'}| = |\delta_t X_t|$. Note that then $|\delta_{t''} X_{t''}| > |\delta_t X_t|$, for $t' < t'' < t$, which means we have $(\delta_{t'} X_{t'}, z_{t'+1} \cdots z_t) \models^+_{\mathcal{A}} (\delta_{t'} X_{t'} X_{t-1}, \epsilon) \vdash_{\mathcal{A}} (\delta_{t'} X_t, \epsilon)$, which is equivalent to $(X_{t'}, z_{t'+1} \cdots z_t) \models^+_{\mathcal{A}} (X_{t'} X_{t-1}, \epsilon) \vdash_{\mathcal{A}} (X_t, \epsilon)$. By definition we get that the automaton $\mathcal{C}_{X_{t'}, X_t}$ recognizes $z_{t'+1} \cdots z_t$. The induction hypothesis for t', applied to $(X_0, z_1 z_2 \cdots z_{t'}) \vdash^*_{\mathcal{A}} (\delta_{t'} X_{t'}, \epsilon)$, provides the required 3 lists with some $m-1$ instead of m. We set $Y_m = X_t$, $b_m = c_{X_{t'}, X_t}$ (so that $\mu(b_m) = \mathcal{C}_{X_{t'}, X_t}$), $v_m = z_{t'+1} \cdots z_t$, which gives us the required 3 lists for t. Note that $(X_0, z_1 \cdots z_{t'}) \vdash^*_{\mathcal{A}} (\delta_{t'} X_{t'}, \epsilon)$ and $(X_0, z_1 \cdots z_{t'} z_{t'+1} \cdots z_t) \vdash^*_{\mathcal{A}} (\delta_{t'} X_{t'}, z_{t'+1} \cdots z_t)$ are equivalent.

We now give a proof of the converse, viz. that if we have a string $b_1 \cdots b_m \in L(\mathcal{F})$,

then a list of m sequences of transitions for automata $\mathcal{A}_k = \mu(b_k)$ recognizing strings v_k, $1 \leq k \leq m$, can be composed into a single sequence for $\mathcal{A}$ recognizing $v_1 \cdots v_m$.

Because of the definition of $\rho(\mathcal{A})$, this list of m sequences consists of sequences of the form $(X_{k-1}^{in}, v_k) \vdash_{\mathcal{A}_k}^* (\delta_k' X_k^{out}, \epsilon)$, $k = 1, 2, \ldots, m$, where $\delta_k' = X_{k-1}^{in}$ if $\mathcal{A}_k$ is of the form $\mathcal{B}_{X_{k-1},X_k}$, and $\delta_k' = \epsilon$ if $\mathcal{A}_k$ is of the form $\mathcal{C}_{X_{k-1},X_k}$. The existence of these sequences implies the existence of sequences $(X_{k-1}, v_k) \vdash_{\mathcal{A}}^* (\delta_k X_k, \epsilon)$, where $\delta_k = X_{k-1}$ if $\mathcal{A}_k$ is of the form $\mathcal{B}_{X_{k-1},X_k}$, and $\delta_k = \epsilon$ if $\mathcal{A}_k$ is of the form $\mathcal{C}_{X_{k-1},X_k}$.

We can place these m sequences after one another to obtain $(X_0, v_1 \cdots v_m) \vdash_{\mathcal{A}}^* (\delta_1 \cdots \delta_m X_m, \epsilon)$, making use of the fact that $(X_{k-1}, v_k) \vdash_{\mathcal{A}}^* (\delta_k X_k, \epsilon)$ implies $(\delta_1 \cdots \delta_{k-1} X_{k-1}, v_k v_{k+1} \cdots v_k) \vdash_{\mathcal{A}}^* (\delta_1 \cdots \delta_{k-1} \delta_k X_k, v_{k+1} \cdots v_k)$. Since $b_1 \cdots b_m \in L(\mathcal{F})$ and therefore $X_0 = q_s = X_{initial}$ and $X_m \in F_{\mathcal{F}} = F_{\mathcal{A}}$, this sequence recognizes $v_1 \cdots v_k$.

From the above we conclude

THEOREM 3.2. $C(\mathcal{N}) = C(\mathcal{D})$

This theorem can be paraphrased as "The class of LR(k) languages is contained in the regular closure of the class of LR(0) languages".[2]

EXAMPLE 3.3. We demonstrate Construction 1 by means of an example. Consider the language $L_{Pal} = \{wcw^R \mid w \in \{a,b\}^*\}$, where w^R denotes the mirror image of string w. This language consists of palindromes in which a symbol c occurs as the center of each palindrome.

Now consider the language $L_{PrePal} = \{v \mid \exists w[vw \in L_{Pal}]\}$, consisting of all prefixes of palindromes. This language, which is obviously not prefix-free, is accepted by the PDA $\mathcal{A}_{PrePal} = (\Sigma, \Delta, I, F, T)$, with $\Sigma = \{a, b, c\}$, $\Delta = \{I, A, B, C, \overline{A}, \overline{\overline{A}}, \overline{B}, \overline{\overline{B}}\}$, $F = \{I, A, B, C\}$, and T consists of the following transitions:

$$
\begin{array}{lcll}
X & \overset{a}{\mapsto} & XA & \text{for } X \in \{I, A, B\} \\
X & \overset{b}{\mapsto} & XB & \text{for } X \in \{I, A, B\} \\
X & \overset{c}{\mapsto} & XC & \text{for } X \in \{I, A, B\} \\
C & \overset{a}{\mapsto} & C\overline{A} & \\
C\overline{A} & \overset{\epsilon}{\mapsto} & \overline{\overline{A}} & \\
A\overline{\overline{A}} & \overset{\epsilon}{\mapsto} & C & \\
C & \overset{b}{\mapsto} & C\overline{B} & \\
C\overline{B} & \overset{\epsilon}{\mapsto} & \overline{\overline{B}} & \\
B\overline{\overline{B}} & \overset{\epsilon}{\mapsto} & C &
\end{array}
$$

The automaton operates by pushing each a or b it reads onto the stack in the form of A or B, until it reads c, and then the symbols read are matched against the occurrences of A and B on the stack. Note that F is $\{I, A, B, C\}$, which means that a recognized string may be the prefix of a palindrome instead of being a palindrome itself.

The upper level of the meta-deterministic automaton $\rho(\mathcal{A}_{PrePal})$ is shown in Figure 3.3. (Automata accepting the empty language have been omitted from this representation, as well as vertices which after this omission do not occur on any path from I to any other final state.)

[2] Not all definitions of LR(0) in the literature are equivalent. E.g. [8] allows some LR(0) languages that are not prefix-free. Theorem 13.3.1 of that book implies that the alternative class of LR(0) languages is contained in the regular closure of the class of what we call LR(0) languages.

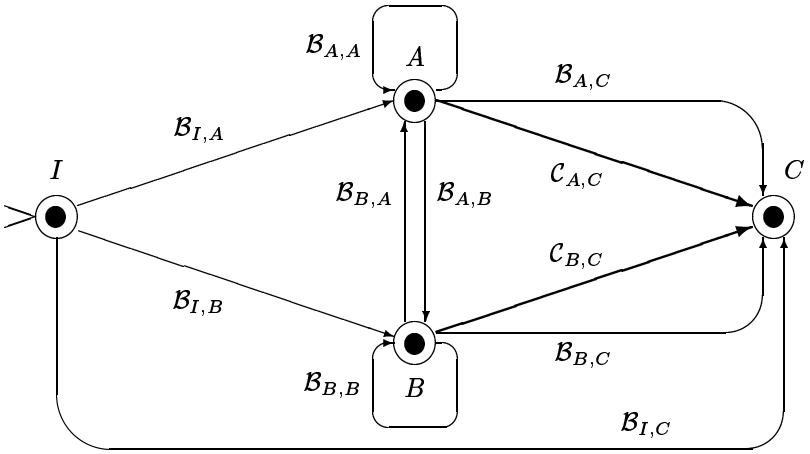

FIG. 3.3. *Meta-deterministic automaton* $\rho(\mathcal{A}_{PrePal})$.

The automaton $\mathcal{B}_{A,B}$ accepts the language $\{b\}$, since the only pushing transition of $\mathcal{A}_{PrePal}$ which places B on top of A reads b. As another example of a lower level automaton, automaton $\mathcal{C}_{A,C}$ accepts the language $\{wa \mid w \in L_{Pal}\}$, since $(A, v) \models^{+}_{\mathcal{A}} (AZ, \epsilon) \vdash_{\mathcal{A}} (C, \epsilon)$, some Z, only holds for v of the form wa, with $w \in L_{Pal}$; for example $(A, bcba) \vdash_{\mathcal{A}} (AB, cba) \vdash_{\mathcal{A}} (ABC, ba) \vdash_{\mathcal{A}} (ABC\overline{B}, a) \vdash_{\mathcal{A}} (AB\overline{\overline{B}}, a) \vdash_{\mathcal{A}} (AC, a) \vdash_{\mathcal{A}} (AC\overline{A}, \epsilon) \vdash_{\mathcal{A}} (A\overline{\overline{A}}, \epsilon) \vdash_{\mathcal{A}} (C, \epsilon)$.

Note that Construction 1 together with a mechanical transformation from finite automata to regular expressions (e.g. [9, Theorem 2.4]) gives us a method for obtaining a regular expression over LR(0) languages, given an LR(k) language. For example, the equation $L_{PrePal} = \{a, b\}^{*}(\epsilon \cup L_{Pal})$ may be derived from investigating Figure 3.3.

□

4. Recognizing fragments of a string. In this section we investigate the following problem. Given an input string $a_1 \cdots a_n$ and an NPDA $\mathcal{A}$, find all pairs of input positions (j, i) such that substring $a_{j+1} \cdots a_i$ is recognized by $\mathcal{A}$; or in other words, such that $(X_{initial}, a_{j+1} \cdots a_i) \vdash^{*} (X_{final}, \epsilon)$. It will be proved that this problem can be solved in linear time.

For technical reasons we have to assume that the stack always consists of at least two elements. This is accomplished by assuming that a fresh stack symbol $\perp$ occurs below the bottom of the actual stack, and by assuming that the actual initial configuration is created by an imaginary extra step $(\perp, v) \vdash (\perp X_{initial}, v)$.

The original problem stated above is now generalized to finding all 4-tuples (X, j, Y, i), with $X, Y \in \Delta$ and $0 \leq j \leq i \leq n$, such that $(X, a_{j+1} \cdots a_i) \models^{+} (XY, \epsilon)$. In words, this condition states that if a stack has an element labelled X on top then the pushdown automaton can, by reading the input between j and i and without ever popping X, obtain a stack with one more element, labelled Y, which is on top of X. Such 4-tuples are henceforth called *items*.

The items are computed by a dynamic programming algorithm based on work from [1, 11, 6, 12].

ALGORITHM 1 (Dynamic programming). Consider an NPDA and an input string $a_1 \cdots a_n$.

1. Let the set $\mathcal{U}$ be $\{(\perp, i, X_{initial}, i) \mid 0 \leq i \leq n\}$.

2. Perform one of the following two steps as long as one of them is applicable.

push 1. Choose a pair, not considered before, consisting of a transition $X \overset{z}{\mapsto} XY$ and an input position j, such that $z = \epsilon \vee z = a_{j+1}$.
2. If $z = \epsilon$ then let $i = j$, else let $i = j + 1$.
3. Add item (X, j, Y, i) to $\mathcal{U}$.

pop 1. Choose a triple, not considered before, consisting of a transition $XY \overset{\epsilon}{\mapsto} Z$ and items $(W, h, X, j), (X, j, Y, i) \in \mathcal{U}$.
2. Add item (W, h, Z, i) to $\mathcal{U}$.

3. Finally, define the set $\mathcal{V}$ to be $\{(j, i) \mid (\bot, j, X_{final}, i) \in \mathcal{U}\}$.

It can be proved [1, 11] that Algorithm 1 eventually adds an item (X, j, Y, i) to $\mathcal{U}$ if and only if $(X, a_{j+1} \cdots a_i) \models^+ (XY, \epsilon)$. Specifically, $(\bot, j, X_{final}, i) \in \mathcal{U}$ is equivalent to $(\bot, a_{j+1} \cdots a_i) \vdash (\bot X_{initial}, a_{j+1} \cdots a_i) \vdash^* (\bot X_{final}, \epsilon)$. Therefore, the existence of such an item $(\bot, j, X_{final}, i) \in \mathcal{U}$, or equivalently, the existence of $(j, i) \in \mathcal{V}$, indicates that substring $a_{j+1} \cdots a_i$ is recognized by $\mathcal{A}$, which solves the original problem stated at the beginning of this section.

If no restrictions apply, the number of 4-tuples computed in $\mathcal{U}$ can be quadratic in the length of the input. The central observation is this: It is possible that items $(X, j, Y, i) \in \mathcal{U}$ are added for several (possibly linearly many) i, with fixed X, j and Y. This may happen if $(\bot, a_h \cdots a_j \cdots a_{i_m}) \vdash^* (\delta X, a_{j+1} \cdots a_{i_m}) \models^+ (\delta XY, a_{i_1+1} \cdots a_{i_m})$ and $(Y, a_{i_1+1} \cdots a_{i_m}) \vdash^+ (Y, a_{i_2+1} \cdots a_{i_m}) \vdash^+ \ldots \vdash^+ (Y, a_{i_{m-1}+1} \cdots a_{i_m}) \vdash^+ (Y, \epsilon)$, which leads to m items (X, j, Y, i_1), ..., (X, j, Y, i_m). Such a situation can in the most trivial case be caused by a pair of transitions $X \overset{z}{\mapsto} XY$ and $XY \overset{\epsilon}{\mapsto} X$; the general case is more complex however.

On the other hand, whenever it can be established that for all X, j and Y there is at most one i with (X, j, Y, i) being constructed, then the number of entries computed in $\mathcal{U}$ is linear in the length of the input string, and we get a linear time bound by the reasoning presented at the end of this section.

The following definition identifies the intermediate objective for obtaining a linear complexity. We define a PDA to be *loop-free* if $(X, v) \vdash^+ (X, \epsilon)$ does not hold for any X and v. The intuition is that reading some input must be reflected by a change in the stack.

Our solution to linear-time recognition for automata which are not loop-free is the following: We define a language-preserving transformation from an arbitrary NDPA to a loop-free NDPA. (A similar transformation for the purpose of recognizing suffixes of strings in linear time was described in [13].) Intuitively, this is done by pushing extra elements $\overline{X}$ on the stack so that we have $(X, v) \vdash^+ (\overline{X}X, \epsilon)$ instead of $(X, v) \vdash^+ (X, \epsilon)$, where $\overline{X}$ is a special stack symbol to be defined shortly.

As a first step we remark that for a normal PDA we can divide the stack symbols into two sets $PUSH$ and POP, defined by

$$PUSH = \{X \mid \text{there is a transition } X \overset{z}{\mapsto} XY\}$$
$$POP = \{Y \mid \text{there is a transition } XY \overset{\epsilon}{\mapsto} Z\} \cup \{X_{final}\}$$

It is straightforward to see that determinism of the PDA requires that $PUSH$ and POP are disjoint. We may further assume that each stack symbol belongs to either $PUSH$ or POP, provided we assume that the PDA is *reduced*, meaning that there are no transitions or stack symbols which are useless for obtaining the final configuration from an initial configuration.[3]

[3]Note that each PDA may be turned into a reduced PDA accepting the same language by just

CONSTRUCTION 2 (NPDA transformation). Consider an NPDA $\mathcal{A} = (\Sigma, \Delta, X_{initial}, \{X_{final}\}, T)$ of which the set of stack symbols Δ is partitioned into $PUSH$ and POP, as explained above. From this NPDA a new PDA $\tau(\mathcal{A}) = (\Sigma, \Delta', X'_{initial}, \{X'_{final}\}, T')$ is constructed, $X'_{initial}$ and X'_{final} being fresh symbols, where $\Delta' = \Delta \cup \{X'_{initial}, X'_{final}\} \cup \{\overline{X} \mid X \in PUSH\}$, $\overline{X}$ being fresh symbols, and the transitions in T' are given by

$$
\begin{array}{llll}
XY & \overset{\epsilon}{\mapsto}_{\tau(\mathcal{A})} & Z & \text{for } XY \overset{\epsilon}{\mapsto}_{\mathcal{A}} Z \text{ with } Z \in POP \\
XY & \overset{\epsilon}{\mapsto}_{\tau(\mathcal{A})} & \overline{Z} & \text{for } XY \overset{\epsilon}{\mapsto}_{\mathcal{A}} Z \text{ with } Z \in PUSH \\
\overline{X} & \overset{\epsilon}{\mapsto}_{\tau(\mathcal{A})} & \overline{X}X & \text{for } X \in PUSH \\
\overline{X}Y & \overset{\epsilon}{\mapsto}_{\tau(\mathcal{A})} & Y & \text{for } X \in PUSH, Y \in POP \\
X & \overset{z}{\mapsto}_{\tau(\mathcal{A})} & XY & \text{for } X \overset{z}{\mapsto}_{\mathcal{A}} XY
\end{array}
$$

and the two transitions $X'_{initial} \overset{\epsilon}{\mapsto}_{\tau(\mathcal{A})} X'_{initial}X_{initial}$ and $X'_{initial}X_{final} \overset{\epsilon}{\mapsto}_{\tau(\mathcal{A})} X'_{final}$.

To provide an intuitive explanation of this construction, we observe that the unwanted sequences of transitions have the property of replacing a push symbol X by itself without affecting the part of the stack under it in the course of doing so. The transformation has the effect that instead of X, a padded form of it consisting of two symbols $\overline{X}X$ is produced by the corresponding new transition sequence. So for example, a sequence $(X, v_1v_2) \models^+ (XY_1, v_2) \vdash (X, v_2) \models^+ (XY_2, \epsilon) \vdash (X, \epsilon)$ in the original automaton, is turned into a sequence $(X, v_1v_2) \models^+ (XY_1, v_2) \vdash (\overline{X}, v_2) \vdash (\overline{X}X, v_2) \models^+ (\overline{X}XY_2, \epsilon) \vdash (\overline{X}\,\overline{X}, \epsilon) \vdash (\overline{X}\,\overline{X}X, \epsilon)$ in the transformed automaton.

The padding has to be gotten rid of later on, viz. when some genuine pop symbol is on top of it. We could for example obtain $(\overline{X}\,\overline{X}X, z) \vdash (\overline{X}\,\overline{X}XY, \epsilon) \vdash (\overline{X}\,\overline{X}Z, \epsilon) \vdash (\overline{X}Z, \epsilon) \vdash (Z, \epsilon)$, where the original automaton would do $(X, z) \vdash (XY, \epsilon) \vdash (Z, \epsilon)$, assuming $Z \in POP$.

EXAMPLE 4.1. We demonstrate this construction further by means of a more elaborate example.

Consider the NPDA $\mathcal{A} = (\{a, b\}, \{X, Y, Z, P\}, X, \{P\}, T)$, where T contains the transitions given in the left half of Figure 4.1. It is clear that $\mathcal{A}$ is not loop-free: we have $(X, a) \vdash (XY, \epsilon) \vdash (X, \epsilon)$. If the input $a_1 \cdots a_n$ to Algorithm 1 is a^n, then $(\bot, a_{j+1} \cdots a_i) \models^+ (\bot X, \epsilon)$ and therefore $(\bot, j, X, i) \in \mathcal{U}$, for $0 \leq j \leq i \leq n$. This explains why the time complexity is quadratic.

We divide the stack symbols into $PUSH = \{X\}$ and $POP = \{Y, Z, P\}$. Of the transformed automaton $\tau(\mathcal{A}) = (\{a, b\}, \{X, Y, Z, P, X', P', \overline{X}\}, X', \{P'\}, T')$, the transitions are given in the right half of Figure 4.1. That the complexity of Algorithm 1 is no longer quadratic but linear for the transformed PDA is proved in the remainder of this section.

The recognition of aab by $\mathcal{A}$ and $\tau(\mathcal{A})$ is compared in Figure 4.2. □

We now set out to prove that τ has the required properties.

LEMMA 4.2. *If $\mathcal{A}$ is an NPDA, then $\tau(\mathcal{A})$ is an NPDA.*

Proof.

To check the NPDA property, we must establish that $\tau(\mathcal{A})$ is deterministic, prefix-free, and in normal form. We discuss these points in sequence.

Determinism in the case of $X'_{initial}$ is obvious since only one transition applies. This also holds for X_{final}, for which there were no applicable transitions in $\mathcal{A}$ due to

omitting the useless transitions.

$\mathcal{A}$			$\tau(\mathcal{A})$		
			X'	$\overset{\epsilon}{\mapsto}$	$X'X$
X	$\overset{a}{\mapsto}$	XY	X	$\overset{a}{\mapsto}$	XY
XY	$\overset{\epsilon}{\mapsto}$	X	XY	$\overset{\epsilon}{\mapsto}$	$\overline{X}$
			$\overline{X}$	$\overset{\epsilon}{\mapsto}$	$\overline{X}X$
X	$\overset{b}{\mapsto}$	XZ	X	$\overset{b}{\mapsto}$	XZ
XZ	$\overset{\epsilon}{\mapsto}$	P	XZ	$\overset{\epsilon}{\mapsto}$	P
			$\overline{X}P$	$\overset{\epsilon}{\mapsto}$	P (Some other transitions of this form have been omitted, because they are useless.)
			$X'P$	$\overset{\epsilon}{\mapsto}$	P'

FIG. 4.1. *The transformation τ applied to a NPDA $\mathcal{A}$.*

$\mathcal{A}$		$\tau(\mathcal{A})$	
stack	input	stack	input
X	aab	X'	aab
		$X'X$	aab
XY	ab	$X'XY$	ab
X	ab	$X'\overline{X}$	ab
		$X'\overline{X}X$	ab
XY	b	$X'\overline{X}XY$	b
X	b	$X'\overline{X}\,\overline{X}$	b
		$X'\overline{X}\,\overline{X}X$	b
XZ		$X'\overline{X}\,\overline{X}XZ$	
P		$X'\overline{X}\,\overline{X}P$	
		$X'\overline{X}P$	
		$X'P$	
		P'	

FIG. 4.2. *The sequences of configurations recognizing aab, using $\mathcal{A}$ and $\tau(\mathcal{A})$.*

it being prefix-free by assumption. No transitions apply when X'_{final} is on top of the stack.

For each symbol $\overline{X}$ on top of the stack exactly one pushing transition may be applied, while for each pair of symbols $\overline{X}\,Y$ on top of the stack, with $Y \in POP$, exactly one popping transition may be applied.

The other cases of XY on top with pop symbol Y produce either Z or $\overline{Z}$ deterministically, depending on whether Z is a push or pop symbol.

For push symbols $X \in \Delta$ on top of the stack, the unique push move also available in $\mathcal{A}$ is the only possibility.

Prefix-freeness follows from the fact that no transitions with a final stack symbol on top of the stack are possible. On the same line, the property of being in normal form means that the unique final stack symbol can only be at the bottom. This is guaranteed by producing it only from the initial stack symbol which is itself not produced by any transition. □

LEMMA 4.3. *If $\mathcal{A}$ is an NPDA, then $\mathcal{A}$ and $\tau(\mathcal{A})$ accept the same language.*

Proof. We first prove that for all stack symbols $X_1, \ldots, X_m$ from Δ we have

$(X_{initial}, v) \vdash^*_{\mathcal{A}} (X_1 \cdots X_m, \epsilon)$ if and only if $(X_{initial}, v) \vdash^*_{\tau(\mathcal{A})} (\alpha_1 X_1 \alpha_2 \cdots \alpha_m X_m, \epsilon)$ where for all i, $1 \leq i \leq m$, we have $\alpha_i = \overline{Y_{i,1}}\,\overline{Y_{i,2}} \cdots \overline{Y_{i,r_i}}$ for some $r_i \geq 0$.

"only if": The proof is given by induction on the number of steps used in $(X_{initial}, v) \vdash^*_{\mathcal{A}} (X_1 \cdots X_m, \epsilon)$.

1. If zero steps are involved then we have $(X_{initial}, \epsilon) \vdash^*_{\mathcal{A}} (X_{initial}, \epsilon)$. By definition we have also $(X_{initial}, \epsilon) \vdash^*_{\tau(\mathcal{A})} (X_{initial}, \epsilon)$.

2. Suppose that the last step is a push, then we have $(X_{initial}, v) \vdash^*_{\mathcal{A}} (X_1 \cdots X_{m-1}, z) \vdash_{\mathcal{A}} (X_1 \cdots X_m, \epsilon)$, where the last transition used is $X_{m-1} \overset{z}{\mapsto}_{\mathcal{A}} X_{m-1}X_m$. The induction hypothesis informs us that $(X_{initial}, v) \vdash^*_{\tau(\mathcal{A})} (\alpha_1 X_1 \alpha_2 \cdots \alpha_{m-1} X_{m-1}, z)$. Because also $X_{m-1} \overset{z}{\mapsto}_{\tau(\mathcal{A})} X_{m-1}X_m$ we have $(X_{initial}, v) \vdash^*_{\tau(\mathcal{A})} (\alpha_1 X_1 \alpha_2 \cdots \alpha_{m-1} X_{m-1} X_m, \epsilon)$.

3. Suppose that the last step is a pop, then we have $(X_{initial}, v) \vdash^*_{\mathcal{A}} (X_1 \cdots X_{m-1} X'_m X_{m+1}, \epsilon) \vdash_{\mathcal{A}} (X_1 \cdots X_{m-1} X_m, \epsilon)$, where the last transition used is $X'_m X_{m+1} \overset{\epsilon}{\mapsto}_{\mathcal{A}} X_m$. The induction hypothesis informs us that $(X_{initial}, v) \vdash^*_{\tau(\mathcal{A})} (\alpha_1 X_1 \alpha_2 \cdots \alpha_m X'_m \alpha_{m+1} X_{m+1}, \epsilon)$, with $\alpha_{m+1} = \overline{Y_1} \cdots \overline{Y_r}$, for some $r > 0$. We first have $(\alpha_1 X_1 \alpha_2 \cdots \alpha_m X'_m \overline{Y_1} \cdots \overline{Y_r} X_{m+1}, \epsilon) \vdash_{\tau(\mathcal{A})} (\alpha_1 X_1 \alpha_2 \cdots \alpha_m X'_m \overline{Y_1} \cdots \overline{Y_{r-1}} X_{m+1}, \epsilon) \vdash_{\tau(\mathcal{A})} \cdots \vdash_{\tau(\mathcal{A})} (\alpha_1 X_1 \alpha_2 \cdots \alpha_m X'_m X_{m+1}, \epsilon)$, using the transitions $\overline{Y_j} X_{m+1} \overset{\epsilon}{\mapsto}_{\tau(\mathcal{A})} X_{m+1}$, $1 \leq j \leq r$, which exist since $X_{m+1} \in POP$. Subsequently, there are two possibilities:

(i) If $X_m \in PUSH$ then $X'_m X_{m+1} \overset{\epsilon}{\mapsto}_{\tau(\mathcal{A})} \overline{X_m}$ and $(\alpha_1 X_1 \alpha_2 \cdots \alpha_m X'_m X_{m+1}, \epsilon) \vdash_{\tau(\mathcal{A})} (\alpha_1 X_1 \alpha_2 \cdots \alpha_m \overline{X_m}, \epsilon) \vdash_{\tau(\mathcal{A})} (\alpha_1 X_1 \alpha_2 \cdots \alpha_m \overline{X_m} X_m, \epsilon)$. In the last configuration, $\alpha_m \overline{X_m}$ is a sequence of "barred" symbols as desired.

(ii) If $X_m \in POP$ then $X'_m X_{m+1} \overset{\epsilon}{\mapsto}_{\tau(\mathcal{A})} X_m$ and $(\alpha_1 X_1 \alpha_2 \cdots \alpha_m X'_m X_{m+1}, \epsilon) \vdash_{\tau(\mathcal{A})} (\alpha_1 X_1 \alpha_2 \cdots \alpha_m X_m, \epsilon)$.

"if": Analogously to the "only if" part, the proof is given by induction on the number of steps used in $(X_{initial}, v) \vdash^*_{\tau(\mathcal{A})} (\alpha_1 X_1 \alpha_2 \cdots \alpha_m X_m, \epsilon)$. The following cases are possible.

1. Suppose the last transition used was $\overline{X_m} \overset{\epsilon}{\mapsto}_{\tau(\mathcal{A})} \overline{X_m} X_m$, then we have $(X_{initial}, v) \vdash^*_{\tau(\mathcal{A})} (\alpha_1 X_1 \alpha_2 \cdots X_{m-1} \alpha'_m XY, \epsilon) \vdash_{\tau(\mathcal{A})} (\alpha_1 X_1 \alpha_2 \cdots X_{m-1} \alpha'_m \overline{X_m}, \epsilon) \vdash_{\tau(\mathcal{A})} (\alpha_1 X_1 \alpha_2 \cdots X_{m-1} \alpha'_m \overline{X_m} X_m, \epsilon)$, where $\alpha_m = \alpha'_m \overline{X_m}$, and the second-last transition used was $XY \overset{\epsilon}{\mapsto}_{\tau(\mathcal{A})} \overline{X_m}$ for some X and Y. The induction hypothesis informs us that $(X_{initial}, v) \vdash^*_{\mathcal{A}} (X_1 \cdots X_{m-1} XY, \epsilon)$. From $XY \overset{\epsilon}{\mapsto}_{\tau(\mathcal{A})} \overline{X_m}$ we conclude the existence of $XY \overset{\epsilon}{\mapsto}_{\mathcal{A}} X_m$. Therefore $(X_1 \cdots X_{m-1} XY, \epsilon) \vdash_{\mathcal{A}} (X_1 \cdots X_{m-1} X_m, \epsilon)$.

2. Suppose the last transition used was $\overline{X} X_m \overset{\epsilon}{\mapsto}_{\tau(\mathcal{A})} X_m$, then we have $(X_{initial}, v) \vdash^*_{\tau(\mathcal{A})} (\alpha_1 X_1 \alpha_2 \cdots \alpha_m \overline{X_m} X_m, \epsilon) \vdash_{\tau(\mathcal{A})} (\alpha_1 X_1 \alpha_2 \cdots \alpha_m X_m, \epsilon)$. Since $\alpha_m \overline{X_m}$ is a sequence of barred symbols, the induction hypothesis informs us that $(X_{initial}, v) \vdash^*_{\mathcal{A}} (X_1 \cdots X_m, \epsilon)$.

3. Suppose the last transition used was $XY \overset{\epsilon}{\mapsto}_{\tau(\mathcal{A})} X_m$ for X and Y from Δ. Then we have $(X_{initial}, v) \vdash^*_{\tau(\mathcal{A})} (\alpha_1 X_1 \alpha_2 \cdots X_{m-1} \alpha_m XY, \epsilon) \vdash_{\tau(\mathcal{A})} (\alpha_1 X_1 \alpha_2 \cdots X_{m-1} \alpha_m X_m, \epsilon)$. From the induction hypothesis, $(X_{initial}, v) \vdash^*_{\mathcal{A}} (X_1 \cdots X_{m-1} XY, \epsilon)$. The transition $XY \overset{\epsilon}{\mapsto}_{\mathcal{A}} X_m$ is identically available in $\mathcal{A}$, thus providing the desired sequence of transitions.

4. The argument for a transition $X \overset{z}{\mapsto}_{\tau(\mathcal{A})} XY$, $X \in \Delta$, is analogous to case 3 because again the transition is also available in the old automaton.

From the definition of τ it is clear that $(X'_{initial}, v) \vdash^*_{\tau(\mathcal{A})} (X'_{final}, \epsilon)$ is only possible if $(X_{initial}, v) \vdash^*_{\tau(\mathcal{A})} (X_{final}, \epsilon)$. From the "if" part we conclude that $(X_{initial}, v) \vdash^*_{\tau(\mathcal{A})}$

(X_{final}, ϵ) implies $(X_{initial}, v) \vdash^*_{\mathcal{A}} (X_{final}, \epsilon)$.

From the "only if" part we conclude that $(X_{initial}, v) \vdash^*_{\mathcal{A}} (X_{final}, \epsilon)$ implies $(X'_{initial}, v) \vdash_{\tau(\mathcal{A})} (X'_{initial} X_{initial}, v) \vdash^*_{\tau(\mathcal{A})} (X'_{initial} \alpha X_{final}, \epsilon) \vdash^*_{\tau(\mathcal{A})} (X'_{initial} X_{final}, \epsilon) \vdash_{\tau(\mathcal{A})} (X'_{final}, \epsilon)$, for some $\alpha = \overline{Y_1} \cdots \overline{Y_r}$, using the transitions $\overline{Y_j} X_{final} \overset{\epsilon}{\mapsto}_{\tau(\mathcal{A})} X_{final}$, $1 \leq j \leq r$, which exist since $X_{final} \in POP$.

This proves the equivalence of the two accepted languages. □

LEMMA 4.4. *If $\mathcal{A}$ is an NPDA, then $\tau(\mathcal{A})$ is loop-free.*

Proof. Consider the set of stack symbols of $\tau(\mathcal{A})$. We define a partial ordering $<$ on these symbols as the least ordering satisfying:

$$\begin{aligned} X'_{final} &< X'_{initial} \\ X &< \overline{Y} \quad \text{for all } X \in POP, Y \in PUSH \\ \overline{Y} &< Z \quad \text{for all } Y, Z \in PUSH \\ X &< Z \quad \text{for all } X \in POP, Z \in PUSH \end{aligned}$$

Note that this relation is transitive and irreflexive. Below we prove that if $(X, v) \vdash^+_{\tau(\mathcal{A})} (Z, \epsilon)$ then $Z < X$. This is sufficient to prove that $\tau(\mathcal{A})$ is loop-free, since $<$ is irreflexive.

Consider $(X, v) \models^+_{\tau(\mathcal{A})} (XY, \epsilon) \vdash_{\tau(\mathcal{A})} (Z, \epsilon)$, using some transition $XY \overset{\epsilon}{\mapsto}_{\tau(\mathcal{A})} Z$ for the last step. It is obvious that $X \notin POP$ since otherwise Y could not have been on top of X. There are three remaining cases:

(i) If $X = X'_{initial}$ then Z must be X'_{final}. Therefore $Z < X$.

(ii) If $X \in PUSH$ then either $Z \in POP$ or Z is of the form $\overline{Z'}$, with $Z' \in PUSH$, according to the definition of τ. Therefore in either case $Z < X$.

(iii) If X is of the form $\overline{X'}$, with $X' \in PUSH$, then the transition $XY \overset{\epsilon}{\mapsto}_{\tau(\mathcal{A})} Z$ must be of the form $\overline{X'} Y \overset{\epsilon}{\mapsto}_{\tau(\mathcal{A})} Y$, with $Y = Z \in POP$. Therefore $Z < X$.

Since each sequence $(X, v) \vdash^+_{\tau(\mathcal{A})} (Z, \epsilon)$ can be split up into smaller sequences (in this case at most two, leading from a symbol in $PUSH$ to a barred symbol and then to a symbol in POP) of the form $(X, v) \models^+_{\tau(\mathcal{A})} (XY, \epsilon) \vdash_{\tau(\mathcal{A})} (Z, \epsilon)$, and since $<$ is transitive, the required result follows. □

We now return to the issue of the time complexity of Algorithm 1. We start with a minor result.

LEMMA 4.5. *Let $\mathcal{U}$ be computed using Algorithm 1, for a loop-free NPDA and certain input. There can be at most one item of the form $(X, j, Y, i) \in \mathcal{U}$ for each X, Y and j.*

Proof. The existence of an item $(X, j, Y, i) \in \mathcal{U}$ requires that $(X, a_{j+1} \cdots a_i) \models^+ (XY, \epsilon)$. Because the NPDA is (by definition) deterministic, the existence of two items $(X, j, Y, i_1), (X, j, Y, i_2) \in \mathcal{U}$ (say $i_1 < i_2$) requires that $(X, a_{j+1} \cdots a_{i_1}) \models^+ (XY, \epsilon)$ and $(Y, a_{i_1+1} \cdots a_{i_2}) \vdash^+ (Y, \epsilon)$, because of the definition of $\models^+$. However, $(Y, a_{i_1+1} \cdots a_{i_2}) \vdash^+ (Y, \epsilon)$ is not possible if the NPDA is loop-free. □

THEOREM 4.6. *For a loop-free NPDA, Algorithm 1 has linear time demand, measured in the length of the input.*

Proof. Let the input be $a_1 \cdots a_n$. Let $|\Delta|$ denote the number of stack symbols. We investigate how many steps are applied in the process of computing $\mathcal{U}$.

push Since the PDA is deterministic, there are $\mathcal{O}(|\Delta| \cdot n)$ combinations of a stack symbol X and an input position i such that there is a transition $X \overset{z}{\mapsto} XY$ with $z = \epsilon \vee z = a_i$. Therefore, the pushing step is applied $\mathcal{O}(|\Delta| \cdot n)$ times.

pop There are $\mathcal{O}(|\Delta|^2 \cdot n)$ items of the form $(W, h, X, j) \in \mathcal{U}$, because of Lemma 4.5. For each of these, there are $\mathcal{O}(|\Delta|)$ items of the form $(X, j, Y, i) \in \mathcal{U}$, again

because of Lemma 4.5. The popping step is therefore applied $\mathcal{O}(|\Delta|^3 \cdot n)$ times.

Together, this yields $\mathcal{O}(|\Delta|^3 \cdot n)$ steps. Computing $\mathcal{V}$ from $\mathcal{U}$ can straightforwardly be done within $|\Delta|^2 \cdot n$ steps, since there are at worst that many elements in $\mathcal{U}$, according to Lemma 4.5. □

COROLLARY 4.7. *For any NPDA $\mathcal{A}$ and input $a_1 \cdots a_n$, the set $\{(j,i) \mid a_{j+1} \cdots a_i \in L(\mathcal{A})\}$ can be computed in linear time.*

Proof. From Lemmas 4.2 and 4.4 and Theorem 4.6 we conclude that $\{(j,i) \mid a_{j+1} \cdots a_i \in L(\tau(\mathcal{A}))\}$ can be computed in linear time. According to Lemma 4.3, this is the same set as $\{(j,i) \mid a_{j+1} \cdots a_i \in L(\mathcal{A})\}$. □

5. Meta-deterministic recognition. With the results from the previous section we can prove that the recognition problem for meta-deterministic languages can be solved in linear time, by giving a tabular algorithm simulating meta-deterministic automata.

Consider a meta-deterministic automaton $\mathcal{M} = (\mathcal{F}, A, \mu)$. Because of Theorem 3.2 we may assume without loss of generality that the DPDAs in A are all normal PDAs. Because of the existence of transformation τ, we may furthermore assume that those NPDAs are all loop-free.

For deciding whether some input string $a_1 \cdots a_n$ is recognized by $\mathcal{M}$ we first determine which substrings of the input are recognized by which NPDAs in A. Then, we traverse the finite automaton, identifying the input symbols of $\mathcal{F}$ with automata which recognize consecutive substrings of the input string. In order to obtain linear time complexity, we again use tabulation, this time by means of pairs (q,i), which indicate that state q has been reached at input position i.

The complete algorithm is given by

ALGORITHM 2 (Meta-dynamic programming). Consider a meta-deterministic automaton $\mathcal{M} = (\mathcal{F}, A, \mu)$, where $\mathcal{F} = (S, Q, q_s, F, T)$ and A is a finite set of loop-free NPDAs, and consider an input string $a_1 \cdots a_n$.

1. Construct the tables $\mathcal{V}_\mathcal{A}$ as the sets $\mathcal{V}$ in Algorithm 1, for the respective $\mathcal{A} \in A$ and input $a_1 \cdots a_n$.

2. Let the set $\mathcal{W}$ be $\{(q_s, 0)\}$. Perform the following as long as it is applicable.

1. Choose a quadruple not considered before, consisting of
   - (i) a pair $(q,j) \in \mathcal{W}$,
   - (ii) a PDA $\mathcal{A} \in A$,
   - (iii) a pair $(j,i) \in \mathcal{V}_\mathcal{A}$, and
   - (iv) a state $p \in Q$,

   such that $(q,b,p) \in T$ for some b with $\mu(b) = \mathcal{A}$.
2. Add (p,i) to $\mathcal{W}$.

3. Recognize the input when $(q,n) \in \mathcal{W}$, for some $q \in F$.

THEOREM 5.1. *Algorithm 2 recognizes $a_1 \cdots a_n$ if and only if $a_1 \cdots a_n \in L(\mathcal{M})$.*

Proof. First we prove that Algorithm 2 eventually adds an item (q,i) to $\mathcal{W}$ if and only if there is some string $b_1 b_2 \cdots b_m \in S^*$, a sequence of states $q_0, \ldots, q_m \in Q$, a sequence of PDAs $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m \in A$, and a sequence of strings $w_1, \ldots, w_m \in \Sigma^*$ such that

- (i) $q_0 = q_s$, $(q_{k-1}, b_k, q_k) \in T$, for $1 \leq k \leq m$, and $q_m = q$,
- (ii) $\mathcal{A}_k = \mu(b_k)$, for $1 \leq k \leq m$,
- (iii) $w_k \in L(\mathcal{A}_k)$, for $1 \leq k \leq m$, and
- (iv) $a_1 \cdots a_i = w_1 \cdots w_m$.

The "if" part of the proof is by induction on m: Suppose that the above 4 conditions

hold, then in particular $(q_{m-1}, b_m, q_m) \in T$, $\mathcal{A}_m = \mu(b_m)$, and $w_m = a_{j+1} \cdots a_i \in L(\mathcal{A}_m)$, some j. This last condition is equivalent to $(j,i) \in \mathcal{V}_{\mathcal{A}_m}$. The above 4 conditions for m imply the same conditions for $m-1$ with j instead of i, and therefore we may use the induction hypothesis to derive that (q_{m-1}, j) is added to $\mathcal{W}$. We conclude that the conditions are fulfilled under which the algorithm adds (q_m, i) to $\mathcal{W}$.

The "only if" part of the proof is very similar. Proof by induction can be applied here if we assume each item is given a "time stamp" identifying the point in time when that item is (first) added to $\mathcal{W}$.

From the above characterization, the correctness of $(q,n) \in \mathcal{W}$ as criterion for recognition of the input immediately follows. A similar property is proved in full detail in [4]. □

We now get the main theorem of this paper.

THEOREM 5.2. *Recognition can be performed in linear time for all meta-deterministic languages.*

Proof. It is sufficient to prove that Algorithm 2 operates in linear time. Because of Theorem 4.6 and since there is a finite number of NPDAs in A, the tables $\mathcal{V}_{\mathcal{A}}$, $\mathcal{A} \in A$, can be constructed in linear time.

Further, the table $\mathcal{W}$ is constructed in a linear number of steps, since each step corresponds with one quadruple $((q,j),\ \mathcal{A},\ (j,i),\ p)$, with $(j,i) \in \mathcal{V}_{\mathcal{A}}$, of which there are at most $|Q|^2 \cdot |A| \cdot n$. Note that prefix-freeness of each $\mathcal{A}$ implies that for any j there is at most one i such that $(j,i) \in \mathcal{V}_{\mathcal{A}}$. □

6. On-line simulation. The nature of Algorithm 2 as simulation of meta-deterministic automata is such that it could be called an *off-line* algorithm. A case in point is that it simulates steps of PDAs at certain input positions where this can never be useful for recognition of the input if the preceding input were taken into account. By processing the input strictly from left to right and by computing the table elements in a demand-driven way, an *on-line* algorithm is obtained, which leads to fewer table elements, although the *order* of the time complexity is not reduced.

The realisation of this on-line algorithm consists of two steps: first we adapt the pushing step so that the PDAs by themselves are simulated on-line, and second, we merge Algorithm 1 and Algorithm 2 such that they cooperate by passing control back and forth concerning (1) where a PDA should start to try to recognize a subsequent substring according to the finite automaton, and (2) at what input position a PDA has succeeded in recognizing a substring. Conceptually, the finite automaton and the PDAs operate in a routine-subroutine relation.

ALGORITHM 3 (On-line meta-dynamic programming). Consider a meta-deterministic automaton $\mathcal{M} = (\mathcal{F}, A, \mu)$, with $\mathcal{F} = (S, Q, q_s, F, T)$ and A is a finite set of loop-free NPDAs, and consider an input string $a_1 \cdots a_n$.

1. Let the set $\mathcal{W}$ be $\{(q_s, 0)\}$.
2. Let the sets $\mathcal{U}_{\mathcal{A}}$ be $\emptyset$ for all $\mathcal{A} \in A$.
3. Perform one of the following four steps as long as one of them is applicable.

down 1. Choose a pair, not considered before, consisting of
(i) a pair $(q,i) \in \mathcal{W}$ and
(ii) a PDA $\mathcal{A} \in A$,
such that $(q,b,p) \in T$ for some b with $\mu(b) = \mathcal{A}$ and some p.
2. Add $(\perp, i, X_{initial}, i)$ to $\mathcal{U}_{\mathcal{A}}$.

push 1. For some PDA $\mathcal{A} \in A$, choose a pair, not considered before, consisting of a transition $X \overset{z}{\mapsto}_{\mathcal{A}} XY$ and an input position j, such that there is an item $(W, h, X, j) \in \mathcal{U}_{\mathcal{A}}$, for some W and h, and such that $z = \epsilon \vee z = a_{j+1}$.
2. If $z = \epsilon$ then let $i = j$, else let $i = j + 1$.
3. Add item (X, j, Y, i) to $\mathcal{U}_{\mathcal{A}}$.

pop 1. For some PDA $\mathcal{A} \in A$, choose a triple, not considered before, consisting of a transition $XY \overset{\epsilon}{\mapsto}_{\mathcal{A}} Z$ and items $(W, h, X, j), (X, j, Y, i) \in \mathcal{U}_{\mathcal{A}}$.
2. Add item (W, h, Z, i) to $\mathcal{U}_{\mathcal{A}}$.

up 1. Choose a quadruple, not considered before, consisting of
(i) a pair $(q, j) \in \mathcal{W}$,
(ii) a PDA $\mathcal{A} \in A$,
(iii) an item $(\perp, j, X_{final}, i) \in \mathcal{U}_{\mathcal{A}}$, and
(iv) a state $p \in Q$,
such that $(q, b, p) \in T$ for some b with $\mu(b) = \mathcal{A}$.
2. Add (p, i) to $\mathcal{W}$.

4. The input is recognized when $(q, n) \in \mathcal{W}$, for some $q \in F$.

The popping and pushing steps, simulating the PDA steps, operate much as in Algorithm 1. An important difference is that the pushing step no longer operates irrespective of preceding input: it only simulates a push on some stack element X, if it has been established with regard to previously processed input that such an element may indeed appear on top of the stack.

A second difference is that the PDA steps are simulated by starting at input positions computed by the "down" step, which adds $(\perp, i, X_{initial}, i)$ to $\mathcal{U}_{\mathcal{A}}$ only if recognition of a substring recognized by $\mathcal{A}$ is needed from position i in order to enable a transition to a next state in the finite automaton.

The "up" step constitutes a shift of control back to the finite automaton after some PDA has succeeded in recognizing a substring.[4]

The characterization of the elements in $\mathcal{W}$ we gave after Algorithm 2 is still valid for the above on-line algorithm. The characterization of the elements in the sets $\mathcal{U}_{\mathcal{A}}$ is more restricted than before, however. Relying on a standard result for on-line tabular simulation of PDAs [11], one can prove that Algorithm 3 eventually adds an item (X, j, Y, i) to $\mathcal{U}_{\mathcal{A}}$, for some $\mathcal{A} \in A$, if and only if there is some $h \leq j$ and some state q such that

1. $(q, h) \in \mathcal{W}$, for some q with $(q, b, p) \in T$, some b with $\mu(b) = \mathcal{A}$ and some p,
2. $(\perp, a_{h+1} \ldots a_j) \vdash^*_{\mathcal{A}} (\delta X, \epsilon)$, for some δ, and
3. $(X, a_{j+1} \ldots a_i) \models^+_{\mathcal{A}} (XY, \epsilon)$.

The first condition states that the finite automaton is in need of a substring recognized by PDA $\mathcal{A}$ starting from position h. The second condition states that some configuration can be reached from an initial configuration by reading the input from position h up to position j, and in this configuration, an element labelled X is on top of the stack. The third condition is as before.

A device which recognizes some language by reading input strings from left to right is said to satisfy the *correct-prefix property* if it cannot read past the first incorrect symbol in an incorrect input string. A different way of expressing this is that if it has

[4] If μ is bijective, and if $(q, b, p) \in T$ is unique for each b, then the condition $(\perp, j, X_{final}, i) \in \mathcal{U}_{\mathcal{A}}$ may be replaced by $(Y, j, X, i) \in \mathcal{U}_{\mathcal{A}}$ with X final in $\mathcal{A}$, and mention of $(q, j) \in \mathcal{W}$ may be omitted. This generalization allows arbitrary PDAs as opposed to NPDAs. In particular, nondeterministic PDAs may be used. For deterministic, loop-free (but not necessarily normal or prefix-free) PDAs the on-line algorithm then still has a linear time complexity. We do not pursue this option because the resulting recognition algorithm cannot be turned into a parsing algorithm; see also Section 7.

succeeded in processing a prefix w of some input string wv, then w is a prefix of some input string wv' which can be recognized.

A consequence of the on-line property of Algorithm 3 is that it satisfies the correct-prefix property, provided that both the finite automaton $\mathcal{F}$ and the PDAs in A satisfy the correct-prefix property. A straightforward proof can be obtained from the characterizations of the elements in $\mathcal{W}$ and $\mathcal{U}_\mathcal{A}$, $\mathcal{A} \in A$, given above.

7. Producing parse trees. We have shown that meta-deterministic recognition can be done efficiently. The next step is to investigate how the recognition algorithms can be extended to be parsing algorithms.

The approach to tabular context-free parsing in [11, 6] is to start with pushdown transducers. A pushdown transducer can be seen as a PDA of which the transitions produce certain *output symbols* when they are applied. The *output string*, which is a list of all output symbols which are produced while successfully recognizing an input, is then seen as a representation of the parse.

If the pushdown transducers are to be realized using a tabular algorithm such as Algorithm 1 then we may apply the following to compute all output strings without deteriorating the time complexity of the recognition algorithm. The idea is that a context-free grammar, the *output grammar*, is constructed as a side-effect of recognition. For each item (X, j, Y, i) added to the table, the grammar contains a nonterminal $A_{(X,j,Y,i)}$. This nonterminal is to generate all lists of output symbols which the pushdown transducer produces while computing $(X, a_{j+1} \cdots a_i) \models^+ (XY, \epsilon)$. The rules of the output grammar are created when items are computed from others. For example, if we compute an item (W, h, Z, i) from two items $(W, h, X, j), (X, j, Y, i) \in \mathcal{U}$, using a popping transition $XY \overset{\epsilon}{\mapsto} Z$ which produces output symbol a, then the output grammar is extended with rule $A_{(W,h,Z,i)} \to A_{(W,h,X,j)}\ A_{(X,j,Y,i)}\ a$.

The start symbol of the output grammar is $A_{(\perp,0,X_{final},n)}$, for recognition of the complete input. For Algorithm 1 however, which recognizes fragments of the input, we have several output grammars, of which the start symbols are of the form $A_{(\perp,j,X_{final},i)}$. The sets of rules of these grammars may overlap.

The languages generated by output grammars consist of all output strings which may be produced by the pushdown transducer while successfully recognizing the corresponding substrings. In the case of deterministic PDAs, these are of course singleton languages.

In a straightforward way this method may be extended to off-line simulation of a meta-deterministic automaton $\mathcal{M} = (\mathcal{F}, A, \mu)$, where A is now a set of push-down transducers:

1. We create subgrammars for v and the respective automata in A separately, following the ideas above.

2. We merge all grammar rules constructed for the different automata $\mathcal{A} \in A$. We assume the sets of stack symbols from the respective automata are pairwise disjoint, in order to avoid name clashes.

3. For each automaton $\mathcal{A} \in A$ we add rules $A_{(\mathcal{A},j,i)} \to A_{(\perp,j,X_{final},i)}$, if $A_{(\perp,j,X_{final},i)}$ is a nonterminal found while constructing $\mathcal{U}_\mathcal{A}$.

4. While constructing table $\mathcal{W}$ the output grammar may be extended with a rule $A_{(p,i)} \to A_{(q,j)}\ A_{(\mathcal{A},j,i)}$, when a pair (p, i) is derived from a pair $(q, j) \in \mathcal{W}$ and a pair $(j, i) \in \mathcal{V}_\mathcal{A}$.

5. We extend the output grammar with all rules of the form $S \to (q, n)$, where $q \in F$. S is the start symbol of the grammar.

(For on-line processing similar considerations apply.)

In this way, we may produce a context-free grammar reflecting the structure of the input string, without deteriorating the time complexity of the recognition algorithm.

8. Applications.

8.1. Suffix recognition. For a language L we define the language $\mathit{suffix}(L) = \{v \mid \exists w[wv \in L]\}$. A member of $\mathit{suffix}(L)$ will be called a *suffix*. In this section we will assume that L is a deterministic language.

Only recently [2] has it been shown that suffixes can be recognized in linear time. In [13] it was shown that furthermore *parsing* of suffixes is possible in linear time. Here we give an alternative proof of this result as a corollary to the previous sections. This we do by providing a transformation from a deterministic language L, specified in the form of a deterministic PDA $\mathcal{A}$, to a meta-deterministic automaton $\sigma(\mathcal{A})$ recognizing suffixes.

For technical reasons, we assume that the PDA $\mathcal{A}$ satisfies a property called *pop-realistic*, which means that if it can pop a number of elements off a stack, then those elements may indeed occur on top of a stack in a configuration reachable from an initial configuration. Formally, we say that a PDA is *pop-realistic* if $(\delta, v) \vdash^* (X, \epsilon)$, some δ, v, X, implies $(X_{initial}, w) \vdash^* (\delta'\delta, \epsilon)$ for some w and δ'.

The assumption that $\mathcal{A}$ is pop-realistic is not a theoretical restriction, since any PDA can be mechanically transformed into one that is pop-realistic and that accepts the same language [5]; nor is it a practical restriction since many naturally occurring PDAs realizing e.g. top-down or LR recognition already satisfy this property.

CONSTRUCTION 3 (Suffix recognition). Let $\mathcal{A} = (\Sigma, \Delta, X_{initial}, F_{\mathcal{A}}, T_{\mathcal{A}})$ be a deterministic PDA which is pop-realistic. Construct the meta-deterministic automaton $\sigma(\mathcal{A}) = (\mathcal{F}, A, \mu)$, with $\mathcal{F} = (S, Q, q_s, \{q_f\}, T_{\mathcal{F}})$, where

(i) $S = \{e\} \cup \{b_{X,Y} \mid X, Y \in \Delta\} \cup \{c_X \mid X \in \Delta\}$,

(ii) $Q = \Delta \cup \{q_s, q_f\}$,

(iii) q_s and q_f are fresh symbols,

(iv) $T_{\mathcal{F}} = \{(q_s, e, X) \mid X \in \Delta\} \cup \{(X, b_{X,Y}, Y) \mid X, Y \in \Delta\} \cup \{(X, c_X, q_f) \mid X \in \Delta\}$.

As set of PDAs we take $A = \{\mathcal{E}\} \cup \{\mathcal{B}_{X,Y} \mid X, Y \in \Delta\} \cup \{\mathcal{C}_X \mid X \in \Delta\}$. These PDAs are defined as follows.[5]

$\mathcal{E}$ is defined to be $(\Sigma, \{\diamond\}, \diamond, \{\diamond\}, \emptyset)$, where $\diamond$ is a fresh symbol.

Each $\mathcal{B}_{X,Y}$ is defined to be $(\Sigma, \Delta \cup \{X^{in}, X^{out}\}, X^{in}, \{X^{out}\}, T)$, where X^{in} and X^{out} are fresh symbols, and where the transitions in T are those in $T_{\mathcal{A}}$ plus the extra transitions

$$\begin{array}{lll} X^{in} & \overset{\epsilon}{\mapsto}_{\mathcal{B}_{X,Y}} & X^{in}X \\ X^{in}X' & \overset{\epsilon}{\mapsto}_{\mathcal{B}_{X,Y}} & X^{out} \quad \text{for all } ZX' \overset{\epsilon}{\mapsto}_{\mathcal{A}} Y, \text{ some } X' \text{ and } Z \end{array}$$

Each $\mathcal{C}_X$ is defined to be $(\Sigma, \Delta, X, F_{\mathcal{A}}, T_{\mathcal{A}})$.

The function μ maps the symbol e to automaton $\mathcal{E}$, the symbols $b_{X,Y}$ to automata $\mathcal{B}_{X,Y}$, and the symbols c_X to automata $\mathcal{C}_X$.

The automaton $\mathcal{E}$ accepts the singleton language containing the empty string. Its use at a transition $(q_s, e, X) \in T_{\mathcal{F}}$ is to mimic an arbitrary computation of $\mathcal{A}$ leading to a configuration where X is on top of the stack. During this computation an unknown input string is read, and the rest of the stack is composed of an unknown combination of stack symbols.

[5] Note that the languages recognized by some of these automata may be the empty set.

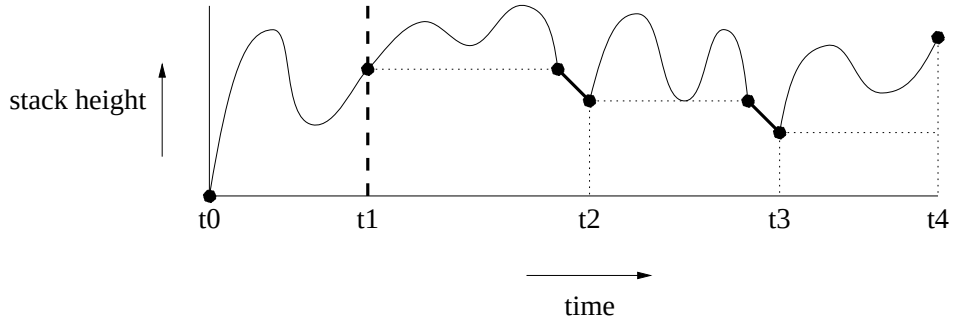

FIG. 8.1. *A stack development of a DPDA $\mathcal{A}$, divided into recognition of a prefix and recognition of the remaining suffix.*

Each automaton $\mathcal{B}_{X,Y}$ mimics all computations beginning with X on top of the stack and ending with the first configuration where the stack shrinks to below the original height; at that point Y is on top. Formally, $\mathcal{B}_{X,Y}$ recognizes a string v if and only if there are X' and Z such that $(X, v) \vdash^* (X', \epsilon)$ and $(ZX', \epsilon) \vdash (Y, \epsilon)$; first, the stack may grow and shrink while reading v, replacing X by some element X', and then X' and an element Z beneath it in the stack are replaced by Y.

Each automaton $\mathcal{C}_X$ mimics all computations of $\mathcal{A}$ that start with X on top of the stack and eventually reach a final configuration without ever having a stack of which the height is one less than that of the original stack. Formally, $\mathcal{C}_X$ recognizes a string v if and only if $(X, v) \vdash^*_{\mathcal{A}} (\delta Y, \epsilon)$ for some $Y \in F_{\mathcal{A}}$.

For a complete proof that $L(\sigma(\mathcal{A})) = \mathit{suffix}(L(\mathcal{A}))$, which is similar to the proof in Section 3, we refer to [5]. In the present paper we merely convey the intuition.

Figure 8.1 suggests how the stack may alternately grow and shrink while $\mathcal{A}$ recognizes some input. From t0 to t1 some prefix of the input is read. Acceptance of the remainder of the input, the suffix, is achieved between t1 and t4. Suppose that the stack shrinks to maximally two elements below the height it had at t1: at t2 the stack shrinks to one element below the original height, and at t3 the stack shrinks one element further. The stack development between t1 and t2 is mimicked by automaton $\mathcal{B}_{X_1,X_2}$, where we assume X_1 and X_2 are on top of the stack at t1 and t2 respectively. Similarly, the development between t2 and t3 is mimicked by $\mathcal{B}_{X_2,X_3}$. The final part, between t3 and t4, is mimicked by $\mathcal{C}_{X_3}$.

Conversely, if we have consecutive segments of the input recognized by a sequence of automata $\mathcal{B}_{X_1,X_2}$, $\mathcal{B}_{X_2,X_3}$ and $\mathcal{C}_{X_3}$, then composition of the three sequences of transitions leads to a development of the stack as suggested between t1 and t4 in Figure 8.1. The existence of the required sequence of transitions between t0 and t1 follows from the assumption that $\mathcal{A}$ is pop-realistic.

The above and Theorem 5.2 together imply

COROLLARY 8.1. *Recognition of suffixes can be performed in linear time for all deterministic languages.*

8.2. Generalized pattern matching. In [10] the following problem is treated. Given are a finite set of input symbols Σ, an input string $a_1 \cdots a_n \in \Sigma^*$ and a pattern $b_1 \cdots b_m \in \Sigma^*$. To be decided is whether $a_1 \cdots a_n = v b_1 \cdots b_m w$, some $v, w \in \Sigma^*$, or in words, whether $b_1 \cdots b_m$ is a substring of $a_1 \cdots a_n$.

This problem can also be stated as follows. To be decided is whether $a_1 \cdots a_n$ is a member of the language $\Sigma^*\{b_1 \cdots b_m\}\Sigma^*$. This language is described as a regular

expression over deterministic languages, i.e. Σ and $\{b_1 \cdots b_m\}$, and therefore this language is meta-deterministic. Consequently, the algorithms in this paper apply.

The time demand can then be shown to be $\mathcal{O}(n \cdot m)$, which is, of course, $\mathcal{O}(n)$ if n is taken as sole parameter. This is in contrast to the algorithm in [10], which provides a complexity of $\mathcal{O}(n + m)$. This seems a stronger result if time complexity is the only matter of consideration. From a broader perspective however, one finds that our approach allows a larger class of problems to be solved.

For example, the substring problem can be generalized as follows. Given are a finite set of input symbols Σ, an input string $a_1 \cdots a_n \in \Sigma^*$ and a deterministic language $L \subseteq \Sigma^*$. To be decided is whether $a_1 \cdots a_n = uvw$, some $u, w \in \Sigma^*$ and $v \in L$, or in words, whether some substring of $a_1 \cdots a_n$ is in L. As before, the problem can be translated into a membership problem of some string in a meta-deterministic language, and therefore our approach allows this problem to be solved in $\mathcal{O}(n)$ time.

9. Conclusions. We have introduced a new subclass of the context-free languages, the meta-deterministic languages, which include the deterministic languages properly. We have given recognition algorithms for this class, and have shown that they have a linear time complexity. Our results are non-trivial since this class contains inherently ambiguous languages. It is still an open problem whether a constructive definition exists for *all* context-free languages which can be recognized in linear time.

Acknowledgements. The second author has had fruitful discussions with Joop Leo about linear-time recognizability of subclasses of context-free languages.

The authors acknowledge help from two referees. The presentation of several proofs was improved due to their suggestions.

REFERENCES

[1] A. Aho, J. Hopcroft, and J. Ullman, *Time and tape complexity of pushdown automaton languages*, Information and Control, 13 (1968), pp. 186–206.
[2] J. Bates and A. Lavie, *Recognizing substrings of LR(k) languages in linear time*, ACM Transactions on Programming Languages and Systems, 16 (1994), pp. 1051–1077.
[3] J. Berstel, *Transductions and Context-Free Languages*, B.G. Teubner, Stuttgart, 1979.
[4] E. Bertsch, *An asymptotically optimal algorithm for non-correcting LL(1) error recovery*, Bericht Nr. 176, Fakultät für Mathematik, Ruhr-Universität Bochum, Apr. 1994.
[5] E. Bertsch and M. Nederhof, *Regular closure of deterministic languages*, Bericht Nr. 186, Fakultät für Mathematik, Ruhr-Universität Bochum, Aug. 1995.
[6] S. Billot and B. Lang, *The structure of shared forests in ambiguous parsing*, in 27th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Vancouver, British Columbia, Canada, June 1989, pp. 143–151.
[7] J. Earley, *An efficient context-free parsing algorithm*, Communications of the ACM, 13 (1970), pp. 94–102.
[8] M. Harrison, *Introduction to Formal Language Theory*, Addison-Wesley, 1978.
[9] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.
[10] D. Knuth, J. Morris, Jr., and V. Pratt, *Fast pattern matching in strings*, SIAM Journal on Computing, 6 (1977), pp. 323–350.
[11] B. Lang, *Deterministic techniques for efficient non-deterministic parsers*, in Automata, Languages and Programming, 2nd Colloquium, Saarbrücken, vol. 14 of Lecture Notes in Computer Science, 1974, Springer-Verlag, pp. 255–269.
[12] M. Nederhof, *Linguistic Parsing and Program Transformations*, PhD thesis, University of Nijmegen, 1994.
[13] M. Nederhof and E. Bertsch, *Linear-time suffix parsing for deterministic languages*, Journal of the ACM, 43 (1996), pp. 524–554.

[14] M. Nederhof and G. Satta, *Efficient tabular LR parsing*, in 34th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Santa Cruz, California, USA, June 1996, pp. 239–246.

[15] H. Richter, *Noncorrecting syntax error recovery*, ACM Transactions on Programming Languages and Systems, 7 (1985), pp. 478–489.

[16] L. Valiant, *General context-free recognition in less than cubic time*, Journal of Computer and System Sciences, 10 (1975), pp. 308–315.