1. Introduction

The equivalence problem and the subclass containment problems for deterministic push-down automata (dpda) have received much attention in recent years. The equivalence problem for dpda's is the problem of deciding for any two dpda's whether the languages accepted by them are the same. The containment problem (dpda, \( L \)) is the problem of deciding for an arbitrary dpda \( M \) whether there exists a machine in the class \( L \) accepting the same language as \( M \).

Many contributions have been made to the equivalence problem by many authors. Although the equivalence problem for general dpda's remains open, it has been shown to be decidable for several subclasses of dpda's [KH, RS, V3, V2, B, TK, OH, OIH2, IYI, GF, U].

On the other hand, there are many subclass containment problems that remain open. However, the equivalence problem and the subclass containment problem are closely related to each other. Friedman and Greibach in [FG] have proved that for any subclass of dpda's \( L \) satisfying certain properties, if the containment problem (dpda, \( L \)) is decidable, then it is also decidable whether two dpda's at least one of which belongs to \( L \) are equivalent. Besides this result, some decidability results for the containment problems have also been obtained. For the class of finite automata, Stearns in [S] proved the decidability of the containment problem (dpda, finite-state automata), which is known as the regularity problem. Valiant in [VI] has improved the regularity test to present an exponentially faster algorithm. Greibach in [G2] has shown that it is decidable whether a real-time dpda with empty stack acceptance accepts a linear context-free language.

Oyamaguchi, Inagaki and Honda in [OIH1] and also Itzhaik and Yehudai independently in [IY2] gave a solution to the subclass containment problem (dpda, \( R_0 \)) where \( R_0 \) is the class of real-time dpda's with empty stack acceptance.

In this paper we consider the containment problem (dpda, 1-turn dpda). We shall call this problem the linearity problem for dpda's hereafter. Unfortunately we do not have...
a solution to this problem in general, but we relate it to the concept of on-line regularity indication. A dpda has the on-line regularity indication property if it knows, at each point in the computation, whether the language accepted from here on is regular. We prove that if a dpda may be simulated by a dpda with the on-line regularity indication, then it may be checked for linearity. Using this reduction we show how to decide linearity for a large subclass of the dpda's.

The techniques we use also serve to provide a better understanding of the behavior of dpda's. We get insight into the structure of the stacking and popping moves of the pushdown store when reading different input words, and also the stack movements of different dpda's while reading the same input word.

The paper is organized as follows. Section 2 contains some preliminary definitions, including the concept of a generative quintet and the jump dpda as a normal form for dpda's. In section 3 we introduce the linearity problem and discuss necessary stack usage through the concept of a necessary quintet. We then show an invariant for all dpda's accepting the same language. In section 4 we sketch the linearity test and prove that it works if the dpda in question may be simulated with the on-line regularity indication. We conclude the section with families of dpda's for which this assumption holds, and get a linearity test for them.

The last two sections deal with the finite-turn property and both use the theorems of section 3. Section 5 contains a finite-turn test for strict deterministic real-time dpda's, and in section 6 we find a reduction result of the containment problems of the general case to those of the es-dpda's [F].

2. Preliminaries

We recall some basic definitions regarding deterministic languages. The empty word over an arbitrary alphabet is denoted by $\varepsilon$, and the length of a word $a$ by $|a|$. A deterministic pushdown automaton (dpda) is denoted by a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where $Q$, $\Sigma$, and $\Gamma$ are respectively finite sets of states, input symbols, and stack symbols; $q_0$ is the initial state; $Z_0$ in $\Sigma$ is the initial stack symbols; $F \subseteq Q$ is the set of accepting final states and $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow Q \times \Gamma^*$, the transition function, is a partial function satisfying the determinism condition: if $\delta(q, e, A)$ is defined, then for all $a$ in $\Sigma$, $\delta(q, a, A)$ is not defined. If the determinism condition is not required, then $M$ is a pushdown automaton (pda).

If $\delta(q, \pi, A) = (q', y)$, we usually write $(q, A) \mapsto (q', y)$. This transition rule of $M$ has mode $(q, A)$ and input $\pi$. If $\tau = \varepsilon$, the transition rule is called an $\varepsilon$-rule.

If the only rule with mode $(q, A)$ is an $\varepsilon$-rule, then $(q, A)$ is an $\varepsilon$-mode.