Abstract

We introduce a method for the automatic construction of error correcting parsers and the notion of skeletal set of a language constitutive for the method. The method is studied completely in the framework of formal language theory.

Introduction

We present a method for the automatic construction of error correcting parsers. The construction proceeds in two phases which are illustrated by Fig. 1.

The first step can be applied to any deterministic pushdown parser $M$ (cf. Fig. 1a). We assume that the parser is given by a set of instructions in the form $(q,a,Z) \rightarrow (p,a,\beta)$ (cf. [1]). The meaning of such an instruction is that if the parser is in the state $q$, reading the input symbol $a$, and the pushdown symbol $Z$ then it replaces $Z$ by the string $\beta$ on the output tape and enters the state $p$. Now the first phase of the construction results in removing some (maybe none) of the instructions of $M$ (symbolized by the part cut out in Fig. 1b). The outcome of this phase, a simplified parser $M'$ equivalent to $M$, is sure to stop at the first syntactic error in the input string.

In the course of the second, crucial phase of our construction some
new instructions are appended to $M'$ (symbolized by the hatched area in Fig. 1c). These instructions add the error correcting capacity to $M'$. 

Even such a rough characterization of our method shows that for correct strings the error correcting parser $M''$ performs the same computation as the original parser $M$. In other words, the error handling capacity of $M''$ causes no overhead for correct strings.

Moreover, unlike many contemporary methods our method requires no extra space (for copying the pushdown store, tracing back the input etc.) during error correction. This all follows from the observation that the parser in Fig. 1c has the form of a deterministic pushdown transducer.

To describe other characteristic features of our method let us look more closely on the structure of the parser $M''$ (cf. Fig. 2).

![Diagram of parser structure]

Note that the E-part of the parser (i.e. the appended instructions handling errors) has no access to the output and cannot manipulate the pushdown store. It co-operates with the $M'$-part in the following way.

1) The $M'$-part reads the input until it finds the first error (recall that $M'$ stops on it). Then it passes the control to the E-part together with the information about its current state $q$ and the top pushdown symbol $Z$.

2) The E-part skips input symbols until it finds the first $s$ from a predefined set of skeletal symbols. Skeletal symbols determine recovery points in the input strings, and we shall discuss their properties later on.

3) The E-part feeds a string $u$ into the input channel of the $M'$-part and passes the control back to it.

The third step is repeated until $M'$ is ready to accept the symbol $s$.

There is a perceptible affinity between the behaviour of the parser $M''$ and parsers constructed by Röhrich's method. Actually, Röhrich's remarkable paper [6], in particular his penetrating analysis of the