• adaptive automata are based on structured pushdown automata, that are in turn built up from simple finite-state automata, allowing it to achieve high time performance in deterministic cases
• adaptive automata may be easily extended to implement transducers, allowing the generation of parse trees as side effect of parsing sentences
• general adaptive automata may be restricted to simpler models according to the strict needs of each particular language, allowing to minimize unnecessary resource wasting in the formal model, so leading to cheaper implementations
• natural language processing is a practical problem in Artificial Intelligence that may explore adaptive formal devices as an alternative to existing usual solutions for all these features

In this paper we show that it is possible to make practical use of adaptive formal devices in the field of syntax learning, one of the hundreds of problems that arise in natural language processing research.

No attempt will be made to propose efficient methodologies or to evaluate the degree of usability of the proposed solution.

Instead, we will concentrate in showing it is possible to efficiently use adaptive concepts to solve fundamental problems in syntax learning, so applying adaptive formal devices to problem solving in an important research field. An adaptive automaton may be viewed as an underlying evolving structured pushdown automaton, represented by an initial fixed state machine implementing the adaptive automaton, and a set of adaptive functions. Adaptive functions are intended to perform, along the automaton's operation, appropriate dynamic changes to the current version of its underlying state machine at that moment, so leading to a new version of that state machine.

In the next sections, a simplified description of the notation and the semantics of adaptive automata is given. Then, an example is shown to illustrate its application to solve a typical problem in syntax-learning of regular languages.

Adaptive automata
Finite-state and pushdown automata are known to be inadequate to represent context-dependent languages, because of their lack of resources for storing context-sensitive features. Infinite-state automata can perform such a task, but by their nature they represent a non-practical solution to the problem.

Many abstract devices have been proposed in the literature, each having its own features and limitations, and, due to their complexity, seldom leading to easy creation of efficient implementations. Adaptive automata, as originally proposed [Joshi94] are devices that may be designed to operate extensively as finite-state or as pushdown automata.

An exceptional case occurs strictly at the first time syntactical constructs that have not been handled previously by the automaton. In response, the adaptive automaton executes self-modifying adaptive actions that change its own shape into another one for accepting the novel construct. By designing in such a way an adaptive automaton, the task of accepting context-dependent sentences is limited to some combination of three types of activities:
• handling a context-dependency not previously experienced by the acceptor
• handling an occurrence of a non-regular context-free construction in the sentence
• handling the remaining regular portions of the input string

Structure of adaptive automata
Structured pushdown automata are a class of pushdown automata in which we find:
• a finite set of states, including its initial state, and
• one or more final states;
• an input alphabet and
• a pushdown alphabet;
• a special symbol, indicating that the pushdown store is empty;
• a set of the sub-machines; and
• a set of productions, specifying all allowed transitions.

A set of productions denoting some structured pushdown automaton represents, at any instance, the current state of the machine implementing the adaptive automaton.

Let a configuration of the adaptive automaton be a triple, indicating the contents of the top of the pushdown store \((g, s')\), a state \((s, s')\) and the next input symbol to be consumed \((\sigma, \sigma')\). Each production assumes the general form
\[
(\gamma, s, s', \sigma, \sigma', s') \rightarrow (\gamma, s, s', \sigma', s')
\]
whose left- and right-side triples denote respectively the configurations of the adaptive automaton immediately before and immediately after the application of the production.

Symbols \(s\) and \(\sigma\) represent the invisible part of the pushdown store and the not yet consumed part of the input string, respectively.

Optional adaptive actions \(A\) and \(B\) may be attached to the productions, representing calls to adaptive functions to be executed before and after the production is applied, respectively.

By restricting the format of the productions, one may also restrict hierarchically the class of languages the automaton can describe by eliminating adaptive actions, no context-dependencies are allowed, so context-free languages may be represented; by further eliminating references to a pushdown store, nested constructs are no more accepted, so only regular languages will be represented.

Eliminating the references to a pushdown store without forbidding the use of adaptive actions allow specifying context-dependent languages through a subclass of the adaptive automata in which the underlying state machine is restricted to a finite-state machine.

An adaptive automaton may be stated as an evolving formal device that starts from an initial underlying structured pushdown automaton

\[
E_0 = (Q_0, \Sigma M_0, \Sigma, \Gamma, P_0, Z_0, q_0, F)
\]

where:
- \(Q_0\) is a set of states, including a single initial state \(q_0\) of \(Q_0\)
- \(F\) is the subset of final states \(F \subseteq Q_0\)
- \(\Sigma\) is the input alphabet,
- \(\Gamma\) is the pushdown store alphabet,
- \(Z_0 \in \Gamma\) indicates empty pushdown store
- \(M_0\) is the set of sub-machines constituting \(E_0\)
- \(P_0\) is the adaptive automaton's initial set of productions

whose most general form has the case

\[
(\gamma, s, s) \rightarrow (\gamma, s', s')
\]

specifying a transition the adaptive automaton may execute from situation \((g, s, s)\). At each instance, the transitions describing the underlying structured pushdown automaton are represented by productions of the form

\[
(\gamma, s, s) \rightarrow (\gamma, s', s')
\]

where:
- \((g, s, s)\) is the situation of the adaptive automaton before applying the production
- \(g\) is the contents of the pushdown store top before applying the production
- \(s\) is the state of the automaton before applying the production
- \(\sigma\) is the input symbol to be consumed next
- \(g, s, s)\) is the situation after applying the production. Primed symbols denote the same elements above.

Optional actions \(A\) and \(B\) specify adaptive function calls to be executed respectively before and after the transition is performed.

Different classes of languages may be properly described from this single general form of productions by imposing restrictions adequate to each language's needs:
• General context-sensitive languages will need adaptive actions. Writing onto the input string is optional, as well as the use of a pushdown store.
• General context-free languages will need a pushdown store. No adaptive actions are used, nor writing onto the input string.
• Regular languages do not need a pushdown store. Adaptive actions and writing onto the input string are forbidden.