The evolution of an adaptive automaton is performed by steps, through the execution of elementary adaptive actions that

- search the current set of productions for productions matching a given template.
- change the set of productions by adding a new production to the set.
- remove a specified production from the set of productions.

The operation of deterministic adaptive automata is sketched in the following steps:

1. Search the current set of productions of the automaton for a production matching the current situation. If there is no such production, reject the input string and stop execution.
2. From that production, determine the next situation of the automaton and the adaptive actions to be performed before and after the specified transition.
3. If there is an adaptive action to be performed before changing the situation of the automaton, perform it first.
4. If the adaptive action performed in step 3 erased the currently executing production, go back to step 1.
5. Change the current situation of the adaptive automaton according to the next situation extracted in step 2.
6. If there is an adaptive action to be performed after changing the situation of the automaton, perform it now.
7. If the new current situation is not a final situation, go back to step 1, otherwise accept the input string and stop.

Adaptive automata may be interpreted as evolving state machines with an underlying state-machine that start from a
given initial form, operates conventionally by performing either

- a sequence of non-adaptive transitions until reaching the end of the analysis or
- some adaptive transition that may change the adaptive automaton's set of productions by deleting certain

transitions and adding new ones as needed.

So, an adaptive automaton may evolve to a new shape whenever adaptive actions are executed, afterwards resuming

its operation by proceeding from an adequate restarting state in the new state machine. Therefore, the recognition of an input string will be viewed as a recognition path, representing so many intermediate

steps as the number of adaptive transitions performed by the adaptive automaton. Each of these intermediate steps is concerned to the recognition of some input sub-string by means of

- a sequence of non-adaptive transitions, followed by
- a structural self-modification of the automaton, by executing an adaptive action.

Time-complexity analysis
A brief analysis is made in the following discussion, concerning some aspects of the time- and space-behavior of
deterministic adaptive automata, which accept the sentences of their input language without any backtracking.

Non-deterministic automata, because of their characteristics, are of little interest as models that lead to efficient
implementations, and are not considered in this discussion.

Deterministic adaptive automata accept any n-length sentence by means of:

- one transition for each token in the input sentence (n tokens-consuming transitions, each taking n units of time).
- one empty transition that places a symbol onto the pushdown store for each start of a nested construct in the input sentence (maximum (n + 1)/2 transitions, each taking n units of time).
- one empty transition that pop off the pushdown store a previously pushed symbol for each end of a nested construct in the input sentence (maximum (n + 1)/2 transitions, each taking n units of time).
- one self-modifying call to an adaptive function for each yet inexperienced context-dependency detected in the

input sentence (maximum n calls to adaptive functions, one per token).

Assuming that, in the worst case, k is the time response of the most lengthy adaptive action, n will be the time

wasted in adaptive transitions.

So, the maximum total time wasted to accept an n-length sentence will be

\[ \alpha \cdot n + (\beta + \gamma) \cdot (n + 1)/2 \cdot n \]

From this reasoning, we can conclude that any deterministic adaptive automaton accepts an n-length input string in \( \text{O}(n) \), provided that the time taken by any adaptive action is limited to some finite upper bound \( \delta \).

in the general case, time-response of adaptive automaton will depend essentially on the behavior of their adaptive
actions for adaptive actions with some upper bound time response, the time-response function of the adaptive
automaton will essentially follow that boundary function.

Space-complexity analysis
About trace behavior, a similar reasoning may be done:

- assume that the underlying state machine of the adaptive automaton has m productions in its initial production set.
- the total number of production in the set is given at any instant by adding the initial m to the total number of

intermediate productions, and subtracting the total number of deleted productions
- assume that adaptive actions may insert at most P productions, so that P is an upper-bound for the number of

added productions
- in the worst case, an adaptive action will insert p new productions to the current production set without deleting

any production, so P is the maximum value that P can assume.
- assume that adaptive actions may delete at most Q productions each, so that \( Q \) is an upper-bound for the function

of deleted productions.
- in any case, Q is limited: if no deleting actions are performed, \( Q \) will be zero, and in the extreme case of the
deleting action, \( Q \) will assume the cardinality of the current set of productions, which is always finite.

Consequently, \( Q \) is irrelevant for worst-case investigation of space-behavior of adaptive automata.

for bounded productions P, in the worst case, each adaptive action performed will insert p new productions,

leading to a final production set with \( m + P \cdot n \) productions after accepting any n-length sentence.

Therefore, in the hypothesis that P is bounded, the dimension of the production set will also grow as \( \text{O}(n) \) function.

for arbitrary adaptive actions, P may be unbounded, and in this case the set is not guaranteed to remain finite.

Illustrating Example
The behavior of adaptive automata suggest their use as a model for implementing knowledge acquisition devices.

An application of adaptive automata to model learning devices, a little illuminating example in syntax learning is presented in this paper hereafter.

The example sketches the use adaptive automata in learning the syntax of a simple regular language from a set of

representative samples of the language.

Positive samples allow constructing an initial acceptor for a superset of the desired language, and additional negative
samples allow finding restrictions to be imposed to the automaton in order to obtain a correct acceptor for the desired
language.

Syntax Learning: [Dap94, Dap96, Pu75]

Natural languages may be viewed as practical complex cases of context-dependent languages, so context-free
notions are insufficient to denote them, despite the common practice to state context-sensitive languages in terms of

some context-free language and a set of externally defined context-sensitive restrictions.

So, full grammatical formalization of natural languages may be approximated, to some extent, by underlying context-
free grammars, whose productions may be attached context-sensitive restricting rules.

In other words, from automata viewpoint, an acceptor for such a context-dependent language would be built as an

underlying pushdown acceptor, corresponding to the context-free base grammar, and a set of function calls, attached
to its transitions, representing restrictions associated to context-dependencies.

This arrangement may be very naturally implemented by adaptive automata, which have been conceived to have an

underlying structured pushdown automaton, representing the base context-free language, and a set of adaptive actions
attached to its transitions, intended to impose progressive syntactical restrictions to the underlying automaton along
the recognition process.

Hence, adaptive automata may be easily employed to intuitively model devices performing concepts related to formal

definition and processing of natural language syntax.

Furthermore, because of their intrinsic dynamic behavior, adaptive automata seem to be very suitable to conceptually
model the mechanisms of many important learning features that are present in syntax-learning problems.