Adaptive automata gradually evolve by departing from its initial shape and successively adding productions to or deleting productions from its own current set of productions \( P_n \) as a result of executing adaptive actions.

1. Initialize \( i = 0 \).
2. In situation \((x, e, e)\), search \( P_i \) for some \((y_g, e, e, s, a_i)\), \( A_i \rightarrow (y_g', e', s', a_i') \), \( B \).
3. If no such a production is found, reject the input string and stop.
4. If only one production is found, apply it by proceeding at step 5.
5. If \( n+1 \) productions are found, a non-determinism has been identified.
   5.1. Apply all found productions in parallel.
   5.2. Accept the input string if at least one of the trials succeeds, otherwise reject it.
5.3. Stop.
6. If \( A_i \) is present, perform it, obtaining a new production set \( P_i' \).
7. Search \( P_i' \) for the currently executing production.
8. If the currently executing production has been removed by \( A_i \), go back to step 2.
9. Otherwise, update the current situation, as stated in the current production.
10. If \( B \) is present, perform it, obtaining \( P_i'' \).
11. If a final situation has been reached, accept the input string and stop.
12. Otherwise, increment \( i \) and proceed the recognition in step 2.

The algorithm above may be interpreted as follows:

An adaptive automaton starts executing from an initial situation \((Z_0, e_0, S_0)\), meaning that the pushdown store is empty, the automaton is being fed with the complete input string \( e_0 \), and the underlying state machine is \( S_0 \), in initial state \( e_0 \).

At any instant, being the automaton in situation \((x, e, e)\), the set of productions is searched for some matching production.

A production matches some situation when the current situation of the automaton corresponds exactly to the situation described in the left-hand side of the production.

While searching for such a production, one of the following conditions may occur:

- A single production matches the current situation. In this case, the application of the matching production will be unconditional and deterministic.
- There are more than one matching production. In this case, matching productions must be classified, and if all possibilities are tried non-deterministically, the search of tokens will take place over the others.
- Token-consumptions take precedence over the others.
- No productions are found. Accept the input string if the current state is a final state and the input string is too exhausted leaving empty the pushdown store; reject it if none of the non-deterministic trials accept the input string.

The following text describes how to use the productions in order to make transitions in the adaptive automaton.

The procedure below is performed if the current situation is matched by a production:

\[(y_g, e, e, s, a_i) \rightarrow (y_g', e', s', a_i') \], \( B \).

- If \( A_i \) is present, the corresponding adaptive action will take place first.
- If the execution of \( A_i \) has deleted the current production, abort and start over the process for a new current production.
- If \( A_i \) is present, pop it from the pushdown store.
- If \( G \) is present, push it onto the pushdown store.
- If \( A_i \) is present, consume it and let its right neighbor in the input string be the new current input symbol.
- If \( A_i \) is present, insert it at the left of the current input symbol, and make it the new current input symbol.
- Assign \( e' \) to the current state.
- If \( B \) is present, the corresponding adaptive action will take place last.

We complete this description with the syntax and semantics of adaptive actions. Adaptive actions are interpreted following the definition of a corresponding adaptive function \( F \), which are denoted as tuples with components:

- \( F \) the name of the function
- \( T_1, T_2, \ldots, T_q \), a list of formal parameters
- an optional list of variables
- an optional list of generators
- an optional call to \( f \) (anterior) adaptive action
- a list of basic actions defining inspections, inclusions, deletions of productions
- an optional call to \( g \) (posterior) adaptive action

Variables, parameters and generators are given symbolic names representing values to be used in the productions. Each of these elements are initially undefined, taking at most a single read-only value for each execution of the function. Variables allow referencing some non-constant value. Variables are assigned values as a result of the execution of inspecting or deleting basic adaptive actions.

Generators are similar to variables, but each time they are automatically assigned unique values at the start of the execution of the adaptive function they belong to.

Parameters start undefined when the adaptive functions starts execution, and are always input parameters, staying undefined until some value is attached to them, afterwards they remain read-only throughout the execution of the function.

The pushout mechanism may also assign argument values, taken from the current function call, to the corresponding parameters.

Basic destructive actions (inspection, deletion, inclusion) assume the general forms:

\[ f \sqcap (y_g, e, s, a_i) \rightarrow (y_g', e', s', a_i') \], \( B \), \( f \sqcup (y_g, e, s, a_i) \rightarrow (y_g', e', s', a_i') \], \( B \), \( f \uplus (y_g, e, s, a_i) \rightarrow (y_g', e', s', a_i') \], \( B \)

where the expressions in brackets represent production templates to be respectively searched, deleted or added to the current set of productions of the automaton.

In the particular case of inspection actions, the set of productions will be searched for productions matching the given template.

Variables in the template will be filled with the corresponding values indicated in the production eventually found.

If no matching production exists, all variables used will remain undefined.

If two or more match exist (non-determinism) all matching instances are handled in parallel.

Basic destructive actions do not modify the set of productions in the adaptive automaton.

For basic damaging actions, proceed usually like in the case of basic inspection actions, and delete all matching productions from the set of productions.

If no matching occurs, do not remove any production from the set.

Basic adding actions add the indicated production to the production set if it is not yet there, otherwise do nothing.

The general form for adaptive actions \( A \) or \( B \) is:

\[ F(p_1, p_2, \ldots, p_q) \]

where \( p_1, p_2, \ldots, p_q \) are \( q \) arguments passed to an adaptive function named \( F \).

In parametric adaptive function calls the following conditions must be met:

- use the same name \( F \) to refer to the function in an adaptive action \( A \) or \( B \)
- arguments \( p_1, p_2, \ldots, p_q \) correspond positionally to parameters \( p_1, p_2, \ldots, p_q \)
- each of the arguments \( p_1, p_2, \ldots, p_q \) may be either the name of any element of the adaptive automaton or a symbol of the input alphabet.

Anterior and posterior adaptive actions are usually denoted by calls to (often parametric) adaptive functions.

Operation of adaptive automaton

Adaptive automata are formal devices whose shape evolve, while accepting input sentences, by starting from a fixed initial shape, and changing its set of productions by executing in sequence the adaptive actions attached to the transitions performed.