Proposed Technique

One major issue that must be faced when dealing with natural language inference is how to infer a base language for the desired language. One way consists in adopting an initial hand-made base language definition. Sometimes, however, such a definition is not easily available, suggesting the existence of a previous inference of a regular or a context-free language in the syntax-learning process.

In order to learn a basic underlying syntax from a set of sample sentences, one can start from an initial finite-state automaton which accepts no string at all.

As new sentences are processed, this automaton should be modified to represent increasingly more restrictive acceptors, towards some final version that correctly accepts all the sentences in the sample. By simple generalization procedures, the resulting automaton may be transformed into a device that accepts a wider syntax including the classes to which the sample sentences belong.

It is well known that in many cases this procedure does not generate the correct automaton, leading to a representation of some aspects of the language instead.

Better approximations of the correct automaton may be obtained from that version by successively restricting the language it is able to accept.

This effect may be achieved by narrowing back the syntax accepted by the automaton, by using for that purpose information extracted from a second set of sample strings, whose members do not belong to the language.

As new samples from this second set are processed, the automaton should change into increasingly more restrictive acceptors, towards some final version that correctly describes the desired language.

Obviously the success of the procedure described above depends heavily on the adequacy of the sample used.

Implementation with Adaptive Automata

By means of an adequately designed adaptive automaton, it is possible to perform such a task of constructing or acceptor for the desired language by means of the following steps:

- start by processing samples of the language, that induces the adaptive automaton to configure itself as a finite-state automaton that accepts all given samples.
- after a sufficient sampling is made, the particular tree-shaped finite-state automaton, just obtained by the underlying state-machine of the adaptive automaton, may be transformed into an equivalent more efficient automaton by the use of well-known optimization techniques.
- usual generalization rules may then be applied to the automaton obtained in the previous step, in order to give an acceptor to a more general superset of the desired language, by using less states and transitions.
- further manipulation of the automaton may detect self-embedded constructs, and use them to perform state transformation, leading to a structured pushdown automaton instead of a finite-state automaton.
- negative samples may also be used in order to impose restrictions to the automaton, by specializing the core-free language, until a final formal description of the desired language is obtained.

Examples

In this section, two applications of adaptive automata are presented. For the sake of simplicity, we introduce a simple adaptive automaton, to illustrate the utilization of adaptive action.

Since the application shown in this section involves a lot of steps that transform an initial automaton into a final acceptor representing as close as possible the expected language.

This automaton is better described in [Jos93].

In [Jos93] it is shown how to implement context dependencies in adaptive automata

A similar method allow to perform the equivalent operation from adaptive grammatical formal devices

A simple mapping procedure enables automatic conversion from adaptive automata to adaptive grammar

There is an algorithm that allows automatically generating structured pushdown automata directly from context-free grammars [Jos93]

A similar algorithm allows automatic generation of adaptive grammars from adaptive automata

Applications

In this section, two applications of adaptive automata in syntax learning of regular languages are presented.

For the sake of simplicity, we introduce a simple adaptive automaton to illustrate the functionality of adaptive actions.

Example of adaptive automaton (name collector) - This example is about a name collector. It recognizes identifiers in the input string, classifies them as previously found or not, and modifies the automaton that implements the language in such a way that it can recognize any identifier newly collected [Jos93].

Adaptive functions:

\[ B \left( i, \sigma \right) : \left( \begin{array}{c}
\delta \left( \alpha \right) \\
\left( \alpha \right) \left( \beta \right)
\end{array} \right) \]

\[ + \left( \begin{array}{c}
\left( \alpha \right) \left( \beta \right) \left( \gamma \right)
\end{array} \right) \]

\[ \delta \left( \alpha \right) \rightarrow \left( \alpha \right) \left( \beta \right) \left( \gamma \right) \]

Initial productions:

\[ \left( \alpha \right) : \left( \begin{array}{c}
\left( \alpha \right) \left( \beta \right) \left( \gamma \right)
\end{array} \right) \]

\[ \left( \gamma \right) : \left( \begin{array}{c}
\left( \alpha \right) \left( \beta \right) \left( \gamma \right)
\end{array} \right) \]

\[ \left( \beta \right) : \left( \begin{array}{c}
\left( \alpha \right) \left( \beta \right) \left( \gamma \right)
\end{array} \right) \]

\[ \left( \gamma \right) : \left( \begin{array}{c}
\left( \alpha \right) \left( \beta \right) \left( \gamma \right)
\end{array} \right) \]

\[ \left( \beta \right) : \left( \begin{array}{c}
\left( \alpha \right) \left( \beta \right) \left( \gamma \right)
\end{array} \right) \]

Algorithm to build a composed automaton from prefix- and suffix-automaton

A prefix-free automaton is a tree-shaped automaton that deterministically accepts the finite language representing some positive sample of the sentences of the language.

An algorithm building prefix-free automaton may be designed as such an adaptive automaton that its initial underlying machine shows only one state and one transition with an attached adaptive action (both the origin and the destination of this transition is the same single state, which represents the initial as well as the final state of this automaton).

The adaptive action allow the automaton to grow, driven by the sequence of input symbols, in such a way that at any time of accepted sentence it is a representation of the desired acceptor.

The resulting prefix-free automaton is equivalent to a deterministic finite-state machine that recognizes the finite language corresponding to the set of all input sentences accepted so far.

Suffix-free automata are also tree-shaped automata, and may be built similarly. However, suffix-free automata are interpreted in reverse order, from backward reading of the sentences in the positive sample of the language.

In order to enforce a single initial state, an additional auxiliary state is added as initial state, as well as empty transitions linking it to the starting state of all branches of the tree.

The resulting suffix-free automaton will contain only one initial state and also only one final state.

The map used, adaptive actions have been omitted for clarity.

The composed automaton is built from the composition of the prefix-free automaton with the corresponding suffix-free automata.

It recognizes a superset of the language accepted by those automata, including all sentences in the originating positive sample, and eventual further strings also.