How to split Recursive Automata

Isabelle Tellier

Inria-Lille et LIFO
Université d’Orléans
Available Information

- a set of positive examples
- the target class

First possible strategy: learning by generalization

- build a least general grammar generating the examples
- apply a generalization operator until it belongs to the target class

Second possible strategy: learning by specialization

- the initial hypothesis space is the whole target class
- use the examples to constrain this space until it is reduced to one grammar
Overview of known results

<table>
<thead>
<tr>
<th>class of languages</th>
<th>regular languages</th>
<th>CF languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>is a subclass of</td>
<td>finite state automata</td>
<td>Categorial Grammars</td>
</tr>
</tbody>
</table>
Overview of known results

<table>
<thead>
<tr>
<th>class of language is a subclass of</th>
<th>regular languages</th>
<th>CF languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>representation</td>
<td>finite state automata</td>
<td>Categorial Grammars</td>
</tr>
<tr>
<td>generalization strategy</td>
<td>state fusion (Angluin 81)</td>
<td>unification of categories (Kanazawa 96, 98)</td>
</tr>
</tbody>
</table>

The links between them: in (Tellier 05, 06)
Overview of known results

<table>
<thead>
<tr>
<th>class of languages</th>
<th>regular languages</th>
<th>CF languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>is a subclass of</td>
<td></td>
<td></td>
</tr>
<tr>
<td>representation</td>
<td>finite state automata</td>
<td>Categorial Grammars</td>
</tr>
<tr>
<td>generalization</td>
<td>state fusion</td>
<td>unification of categories</td>
</tr>
<tr>
<td>strategy</td>
<td>(Angluin 81)</td>
<td>(Kanazawa 96, 98)</td>
</tr>
<tr>
<td>specialization</td>
<td>state fission</td>
<td>constraints introduction</td>
</tr>
<tr>
<td>strategy</td>
<td>(Fredouille 00)</td>
<td>(Moreau 04)</td>
</tr>
</tbody>
</table>
## Overview of known results

<table>
<thead>
<tr>
<th>Class of languages</th>
<th>Regular languages</th>
<th>CF languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is a subclass of representation</td>
<td>Finite state automata</td>
<td>Categorial Grammars</td>
</tr>
<tr>
<td>Generalization strategy</td>
<td>State fusion</td>
<td>Unification of categories</td>
</tr>
<tr>
<td>(Angluin 81)</td>
<td></td>
<td>(Kanazawa 96, 98)</td>
</tr>
<tr>
<td>Specialization strategy</td>
<td>State fission</td>
<td>Constraints introduction</td>
</tr>
<tr>
<td>(Fredouille 00)</td>
<td></td>
<td>(Moreau 04)</td>
</tr>
</tbody>
</table>

The links between them: this paper!
1. Introduction
2. Categorial Grammars and Recursive Automata
3. Learning by specialization in both representations
4. Learning from Typed Examples: a new interpretation
5. Conclusion
Definition of a AB-Categorical Grammar

- let $\Sigma$ be a finite vocabulary
- let $B$ be an enumerable set of basic categories, among which is the axiom $S \in B$
- the set of categories $\text{Cat}(B)$ is the smallest set such that:
  - $B \subset \text{Cat}(B)$
  - $\forall A, B \in \text{Cat}(B) : A/B \in \text{Cat}(B)$ and $B\setminus A \in \text{Cat}(B)$
- a Categorical Grammar $G$ is a finite relation over $\Sigma \times \text{Cat}(B)$
- Syntactic rules are expressed by two schemes: $\forall A, B \in \text{Cat}(B)$
  - Forward Application $FA : A/B \xrightarrow{B} B \rightarrow A$
  - Backward Application $BA : B \xrightarrow{B\setminus A} B \rightarrow A$
- $L(G)$ : set of strings corresponding to a sequence of categories which reduces to $S$
Definition of a AB-Categorical Grammar

- Let $\mathcal{B} = \{S, T, CN\}$ where $T$ stands for “term” and $CN$ for “common noun”
- $\Sigma = \{\text{John, runs, a, man, fast}\}$
- $G = \{\langle \text{John, } T \rangle, \langle \text{runs, } T\backslash S \rangle, \langle a, (S/(T\backslash S))/CN \rangle, \langle \text{man, } CN \rangle, \langle \text{fast, } (T\backslash S)\backslash(T\backslash S) \rangle\}$
Definition of Recursive Automata (Tellier06)

- A RA is like a Finite State Automaton except that transitions can be labelled by a state.
- Using a transition labelled by a state $Q$ means producing $w \in L(Q)$.
- There are two distinct kinds of RA:
  - the $RA_{FA}$-kind where the language $L(Q)$ of a state $Q$ is the set of strings from $Q$ to the final state.
  - Every unidirect. $FA$ CG is strongly equivalent with a $RA_{FA}$.
  - the $RA_{BA}$-kind where the language $L(Q)$ of a state $Q$ is the set of strings from the initial state to $Q$.
  - Every unidirect. $BA$ CG is strongly equivalent with a $RA_{BA}$.
  - Every CG is equivalent with a pair $MRA = \langle RA_{FA}, RA_{FA} \rangle$. 

Categorial Grammars and Recursive Automata
1. Introduction
2. Categorial Grammars and Recursive Automata
3. Learning by specialization in both representations
4. Learning from Typed Examples: a new interpretation
5. Conclusion
Inference of rigid CGs from strings (Moreau 04)

- Target Class: rigid Categorial Grammars, i.e. at most one category for each word
- Input: a set of sentences
- Learning Algorithm:
  1. associate a distinct unique variable with each word
  2. for each sentence do
     - try to parse the sentence (CYK-like algorithm)
     - induce constraints on the variables
- Output: (disjunctions of) set(s) of constraints, each set corresponding with a (set of) rigid grammar(s)
Inference of rigid CGs from strings (Moreau 04): example

- input data: The set \( D = \{ \text{John runs, a man runs fast} \} \)
- associate a distinct unique variable with each word:
  \( A = \{ \langle \text{John, } x_1 \rangle, \langle \text{runs, } x_2 \rangle, \langle \text{a, } x_3 \rangle, \langle \text{man, } x_4 \rangle, \langle \text{fast, } x_5 \rangle \} \)
- for every unidirectional CG \( G \), there exists a substitution transforming \( A \) into \( G \)
- \( A \) specifies the set of every unidirectional CGs
- \( A \) can also be represented by a \( MRA = \langle RA_{FA}, RA_{BA} \rangle \):
Learning by specialization

Inference of rigid CGs from strings (Moreau 04) : example

- the only two possible ways to parse “John runs” :

\[
x_1 = S/x_2
\]

\[
x_2
\]

John runs

\[
x_1
\]

John

\[
x_2 = x_1 \setminus S
\]

runs

- to parse “a man runs fast” :

  - theoretically : \(5 \times 2^3 = 40\) distinct possible ways
  - but some couples of constraints are not compatible with the class of rigid grammars

- main problem with this algo : combinatorial explosion
- to limit it : initial knowledge in the form of known assignments
Effects of constraints on a $MRA = \langle RA_{FA}, RA_{BA} \rangle$

- constraints inferred are of the form:
  - $x_k = x_l$ with $x_k$ and $x_l$ already exist: state and/or transition merges in both the $RA_{FA}$ and the $RA_{BA}$
  - or $x_k = X_m/X_n$ (resp. $x_k = X_m\setminus X_n$) with $X_m, X_n \in \text{Cat}(B)$

- the effect of $x_k = X_m/X_n$ (resp. $x_k = X_m\setminus X_n$) in a MRA:
  - $X_m/X_n$ (resp. $x_k = X_m\setminus X_n$) replaces $x_k$ everywhere in the $MRA$
  - every subcategory of $X_m$ and $X_n$ (including themselves) becomes a new state in both the $RA_{FA}$ and the $RA_{BA}$, linked to $F$ (resp. from $I$) by its name
  - in the $RA_{FA}$ (resp. the $RA_{BA}$), a new transition labelled by $X_m/X_n$ (resp. $X_m\setminus X_n$) links the states $X_m$ and $X_n$
  - the states of the same name are merged

- So: a combination of state splits and state merges
- better founded than the state splits in (Fredouille 00)
1. Introduction
2. Categorial Grammars and Recursive Automata
3. Learning by specialization in both representations
4. Learning from Typed Examples: a new interpretation
5. Conclusion
Learning From Typed Examples

Basic ideas (Dudau, Tellier & Tommasi 01)

- cognitive hypothesis: lexical semantics is learned before syntax
- formalization: words are given with their (Montague’s) semantic type
- Types derive from categories by a homomorphism
- Classical example: \( h(T) = e, \ h(S) = t, \ h(CN) = \langle e, t \rangle \) and \( h(A/B) = h(B\setminus A) = \langle h(B), h(A) \rangle \)

- input data: typed sentences are of the form

<table>
<thead>
<tr>
<th>John</th>
<th>runs</th>
<th>a</th>
<th>man</th>
<th>runs</th>
<th>fast</th>
</tr>
</thead>
<tbody>
<tr>
<td>( e )</td>
<td>( \langle e, t \rangle )</td>
<td>( \langle \langle e, t \rangle, \langle e, t \rangle, t \rangle )</td>
<td>( e, t )</td>
<td>( e, t )</td>
<td>( \langle \langle e, t \rangle, \langle e, t \rangle \rangle )</td>
</tr>
</tbody>
</table>
Learning From Typed Examples

Target Class

- The set of CGs such that every distinct category assigned to the same word gives a distinct type
- \( \forall \langle v, C_1 \rangle, \langle v, C_2 \rangle \in G, C_1 \neq C_2 \Rightarrow h(C_1) \neq h(C_2) \)
- Theorem (Dudau, Tellier & Tommasi 03): for every CF-language, there exists a grammar \( G \) generating it and a morphism \( h \) satisfying this condition

General algorithm (Dudau, Tellier & Tommasi 01)

1. initial set of assignments: introduce variables to represent the class
2. for each sentence
   - try to parse the sentence (CYK-like)
   - induce constraints on the variables
3. Output: (disjunctions) of set(s) of constraint(s), each being represented by a least general grammar
Learning From Typed Examples

Example of pre-treatment

- introduce a distinct variable whose possible values are / or \ in front of every subtype
- in our example, the result is of the form:

John runs

\[ e \quad x_1\langle e, t \rangle \]

\[ x_2\langle x_3\langle e, t \rangle, x_4\langle x_5\langle e, t \rangle, t \rangle \rangle \quad x_6\langle e, t \rangle \quad x_1\langle e, t \rangle \quad x_7\langle x_8\langle e, t \rangle, x_9\langle e, t \rangle \rangle \]
Infering constraints by parsing

$$t$$

$$FA : x_1 = \backslash$$

$$e$$

John $x_1(e, t)$ runs

$$t$$

$$FA : x_4 = /$$

$$x_5 = x_9$$

$$x_4(x_5(e, t), t)$$

$$FA : x_2 = /$$

$$x_3 = x_6$$

$$x_2(x_3(e, t), x_4(x_5(e, t), t))$$

a

$$x_6(e, t)$$

man

$$x_1(e, t)$$

runs

$$x_7(x_8(e, t), x_9(e, t))$$

$$BA : x_7 = \backslash$$

$$x_8 = x_1$$

fast
Sum-up

- mix of state splits and state merges
- Types contain in themselves where splits are possible
- not every (complex) state can be merged: states are typed in the sense of (Coste & alii 2004)
- the use of types reduces the combinatorial explosion of possible splits
- types helph to converge to the correct solution quicker
## Learning From Typed Examples

### Sum-up

<table>
<thead>
<tr>
<th>vocabulary</th>
<th>Moreau's initial assignment</th>
<th>target category</th>
<th>pre-treated type</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>$x_1$</td>
<td>$(S/(T\backslash S))/CN$</td>
<td>$x_2\langle x_3\langle e, t \rangle, x_4\langle x_5\langle e, t \rangle, t \rangle \rangle$</td>
</tr>
<tr>
<td>man</td>
<td>$x_2$</td>
<td>$CN$</td>
<td>$x_6\langle e, t \rangle$</td>
</tr>
<tr>
<td>runs</td>
<td>$x_3$</td>
<td>$T\backslash S$</td>
<td>$x_1\langle e, t \rangle$</td>
</tr>
</tbody>
</table>

- there exists a substitution, thus a homomorphism between Moreau’s assignments and categories
- there exists a homomorphism between categories and types (Principle of compositionality)
- the starting point is either a lower bound or an upper bound
- the “good substitution” is well constrained by types
1. Introduction

2. Categorial Grammars and Recursive Automata

3. Learning by specialization in both representations

4. Learning from Typed Examples : a new interpretation

5. Conclusion
Main contributions

- we mainly propose a new perspective on already known algorithms
- the correspondance between Categorial Grammars and recursive automata is fruitful
- MRA can represent sets of grammars corresponding to search spaces
- specialization strategies require additional knowledge (like semantic types)
- natural language is probably learnt by specialization by children
- specialization techniques deserve further investigation (better for incrementality...)

Conclusion