Proceedings of the Workshop and Tutorial on

Learning Context-Free Grammars

Edited by
Colin de la Higuera, Pieter Adriaans,
Menno van Zaanen, Jose Oncina

14th European Conference on Machine Learning
7th European Conference on Principles and Practice of Knowledge Discovery in Databases

22 – 26 September 2003
Cavtat - Dubrovnik
Croatia

Proceedings of the Workshop and Tutorial on
Learning Context-Free Grammars

Edited by
Colin de la Higuera, Pieter Adriaans, Menno van Zaanen, Jose Oncina

Workshop and Tutorial Chairs:
  Stefan Kramer, Technische Universität München
  Luis Torgo, University of Porto
  Luc Dehaspe, Pharma DM

Publisher: Ruđer Bošković Institute, Bijenička cesta 54, P.O.B. 180, 10002 Zagreb, Croatia


Zagreb 2003
Table of Contents

Preface ................................................................. 1

Slides ...................................................................... 5

Bibliographical Notes ............................................. 19

Papers ................................................................. 29

Inference of a Subclass of Context Free Grammars Using Positive Samples
J. A. LAXMINARAYANA AND G. NAGARAJA ......................... 29

A Hybrid Language Model based on Stochastic Context-free Grammars
D. LINARES, J.-M. BENEDI, AND J.-A. SÁNCHEZ ................... 41

Incremental Learning of Context Free Grammars by Extended Inductive CYK
Algorithm
K. NAKAMURA .......................................................... 53

Leveraging Lexical Semantics to Infer Context-Free Grammars
T. OATES, T. ARMSTRONG, J. HARRIS, AND M. NEJMAN ......... 65

Learning Context Free Grammars in the Limit Aided by the Sample Distribution
Y. SEGNER ............................................................. 77

Left Aligned Grammars—Identifying a class of Context-free Grammar in the
limit from Positive Data
B. STARKIE .......................................................... 89

Offspring-annotated probabilistic context-free grammars
J. L. VERDÚ-MAS, J. CALERA-RUBIO, AND R. C. CARRASCO .... 101

Unsupervised Grammar Induction in a Framework of Information Compression
by Multiple Alignment, Unification and Search
J. G. WOLFF .......................................................... 113
Workshop and Tutorial at ECML/PKDD 2003: Learning Context-Free Grammars

Colin de la Higuera, Pieter Adriaans, Menno van Zaanen and Jose Oncina

- Colin de la Higuera, EURISE, Université de Saint-Etienne, France
  E-mail: cdllh@univ-st-etienne.fr

- Pieter Adriaans, Institute for Language, Logic and Communication (ILLC), Universiteit van Amsterdam, The Netherlands
  E-mail: pietera@science.uva.nl

- Menno van Zaanen, ILK / Computational Linguistics, Tilburg University, The Netherlands
  E-mail: mvzaanen@uvt.nl

- Jose Oncina, Departamento de Lenguajes y Sistemas Informáticos, Universidad de Alicante, Spain
  E-mail: oncina@dsi.ua.es

1 Preface

There has been growing interest over the last few years in learning some formal grammar from text, sequences or structured data, but even if the regular case (corresponding to finite state automata) has been well studied, this has not been the case for more expressive classes such as context-free grammars.

In data mining and machine learning community there is a trend that moves away from single table batch oriented learning tasks to multi-table, on-line learning tasks on top of hybrid data sets that contain tabular data as well as texts, images, sounds etc. This development generates an interest in new classes of learning algorithms and new problem domains. An area that is of vital importance in this context is grammar induction. In this field researchers are beginning to turn their attention to induction of context-free grammars. Induction of classes of languages that are lower in the Chomsky hierarchy is well studied but a real breakthrough in terms of industrial applications and spin-out
to other disciplines, like data mining, is to be expected from a better understanding of the learnability of various subclasses of context-free grammars. In this workshop we will give an overview of the state of the art in this promising field. Special attention will be given to cross-dependencies with existing ML and data mining research. The workshop will especially be of value to researchers with an interest in the intersection of GI, ML and data mining.

1.1 Grammar Induction

Grammar Induction (also referred to as grammatical inference) is a transversal theme. It makes use of results and techniques from formal language theory, computational learning theory, machine learning, statistics to learn, induce or infer a grammar or an automaton from a learning sample. The learning sample is usually composed of strings or other sequential material. Applications of these techniques include Computational Linguistics, Text mining, Speech Recognition, Computational Biology, Web Intelligence or Robotics.

1.2 Context-Free Grammar Learning

Most attention in the field of grammar induction has been set on the problem of learning finite state automata, representing regular languages. Yet long term dependencies, palindromic structures, parenthesis are all internal structures that may appear in a wide range of applications and are better described by context free grammars. The purpose of the workshop is to provide a forum specific to this question, enabling researchers to present their most recent results over the issue of learning context-free grammars.

1.3 Goals for the Workshop

Inside Grammar Induction the topic of context-free grammar learning is recognized as a hard question. There are very few papers published each year on this question and clearly the techniques that have been presented have not been compared with each other. There is no general accepted benchmark, few studies on the critical issues to be dealt with. The workshop will thus aim at:

- providing a clear picture of the field, of the questions that have been solved and of those that need urgent attention
- comparing some of the current techniques
- exploring the issue of learning from bracketed (or semi bracketed) data: this question appears naturally when dealing with XML data
- allowing non specialists to know what is happening in the field
- checking how well these techniques function in practice.
1.4 Topics
The topics on which the call for papers would be issued will be the following:

- General learning results concerning the learnability of context-free grammars (or sub-classes of these).
- Classes of context-free grammars for which positive learning results can be obtained
- heuristics, and results on typical grammars
- validation issues: benchmarks for context-free grammar learning
- applications of context-free grammar learning
- stochastic context-free grammar learning
- learning context-free grammars from structured data, or semi-structured data.
- Learning tree automata.

2 The papers
The organizers of the workshop received a total of 11 submissions, out of which 8 were selected. The numbers may be small, but the average quality was very high and discarding papers was a difficult task!

The papers address most of the various techniques, applications and problems one encounters in context-free grammar learning:

In the theoretical framework Laxminarayana and Nagaraja try to avoid Gold’s negative results concerning identification from positive results alone by proposing a new sub-class of context free grammars, in their paper “Inference of a Subclass of Context-Free Grammars Using Positive Examples”, whereas Seeger (“Learning Context-Free Grammars in the Limit Aided by the Sample Distribution”) introduces a new learning paradigm and uses the distribution over the examples as an extra bias in order to learn.

Learning parsers instead of the actual grammars has historically always been an important direction in the problem of Context-free learning. Nakamura (“Incremental Learning of Context-free grammars by Extended Inductive CYK Algorithm”) presents the system SYNPSE which he has been developing over the past few years. Starkie’s system is still experimental (“Left Aligned Grammars — Identifying a Class of Context-Free Grammar in the Limit from Positive Data”), but has the ambition of learning attribute grammars.

The use of semantics to help the learning process is proposed by Oates et al. in their paper “Leveraging Lexical Semantics to Infer Context-Free Grammars”.

Wolff introduces the ICMAUS framework where alignment is used in order to learn context-free grammars, in “Unsupervised Grammar Induction in a
Framework of Information Compression by Multiple Alignment, Unification and Search”.

The importance of learning context-free grammars from tree banks is due to their capacities in language modeling tasks. Verdú-Mas et al. (“Offspring-Annotated Probabilistic Context-Free Grammars”) propose to put probabilities on a more complex set of non-terminals, given by a more elaborate view over the data. Experiments confirm that the idea is sound. Linares et al. mix stochastic grammars obtained from tree banks with n-grams for language modeling tasks with interesting results (“A Hybrid Language Model Based on Stochastic Context Free Grammars”).
Learning Context-Free Grammars

Colin de la Higuera, University of Saint-Etienne
Jose Oncina, University of Alicante

Outline
1. Context Free grammars
2. Paradigms and theorems
3. Some heuristics
4. Applications
5. Conclusions

What is a context free grammar?

A 4-tuple \((\Sigma, S, V, P)\) such that:
- \(\Sigma\) is the alphabet.
- \(S\) is the start symbol.
- \(V\) is a finite set of non terminals.
- \(P \subseteq V \times (V \cup \Sigma)^*\) is a set of rules.

Example

- The Dyck\(_1\) grammar
  - \((\Sigma, S, V, P)\)
  - \(\Sigma = \{a, b\}\)
  - \(V = \{S\}\)
  - \(P = \{S \rightarrow aSbS, S \rightarrow \lambda\}\)

Derivations and derivation trees

\[S \rightarrow aSbS\]
\[\rightarrow aaSbSbS\]
\[\rightarrow aabbSbS\]
\[\rightarrow aabb\]
\[\rightarrow aabb\]
Why learn context tree grammars (CFG)?

- More expressive than regular grammars: all regular languages are context-free;
- next step up on the Chomsky hierarchy;
- allows to define more precise and expressive structure.

Tree Grammars

- Similar to CFG but the rules have the shape:
  \[ A \rightarrow a \]
  \[ B \rightarrow C \]

Example

- Let \( P = \{ S \rightarrow a, S \rightarrow S, S \rightarrow S \} \)

Skeletons and tree grammars

- Any context free grammar can be transformed in a tree grammar that produces skeletons

A tree automaton

Practise: Parsing

- CYK, Complexity: \( O(n^3) \)
- Earley, Complexity: \( O(n^3) \)
- Valiant, Complexity: \( O(n^{281}) \)
2. Paradigms and results

Identification in the limit
- The examples are provided in sequence.
- For each new example the learning algorithm must provide a hypothesis.
- Success if the sequence of hypothesis converges.

Learning from positives examples
- It is impossible to identify in the limit any super-finite class of language from positive examples only (Gold, 67).
- A super-finite class of languages includes:
  - all the finite language and
  - at least an infinite language.

What can we do?
- Use of some additional help:
  - negative data
  - access to oracles
- Avoid super-finite classes of languages;
- Combinations of both.

Can we learn in the limit context-free languages from...

... positive examples?
- NO (Gold, 67)
  - the class of context free languages is superfinite.
--- positive and negative examples?

**YES** (Gold, 67)
- by an enumeration procedure:
  1. order all the CFG in a list
  2. search the list and return the first grammar consistent with the data.
- this procedure is not exponential!!!
  - complexity is $O(|V| |T| |I|)$

--- skeletons of positive examples?

**NO**, as a consequence of (Gold, 67)
- the class of the tree languages that represent skeletons is super-finite.
**YES** (Sakakibara, 92)
- if the skeletons come from a reversible context-free grammar (normal form)

--- reversible context free languages

- Deterministic bottom-up top-down
  - $A \rightarrow \alpha$ and $B \rightarrow \alpha \Rightarrow A=B$
  - $A \rightarrow \alpha B \beta$ and $A \rightarrow \alpha C \beta \Rightarrow B=C$
- Algorithm
  - Build the grammar that only accepts the sample.
  - Merge pair of non terminals that violate some of the previous rules

--- a distribution?

Not known!
- There are some heuristics...
  - Expectation maximization
  - Inside-Outside

--- queries?

Not (entirely) known!
- The most used queries are:
  - Membership queries
  - Equivalence queries
  - Note that an equivalence query might be non-computable

--- Polynomial identification in the limit
Polynomial Identification

There are several definitions:
- (Pitt, 89) and (Yokomori, 91)
  - Polynomial update time
  - Polynomial hypothesis changes
  - Consistency (Yokomori)
- In polynomial time and data
  (de la Higuera, 97)
  - Polynomial update time
  - Polynomial characteristic sample

Can we learn polynomially in the limit context free languages from ...

... positive and negative examples?

NO,
- if usual cryptographic rules apply.
  (Pitt and Wainwright, 89)
- in polynomial time and data framework,
  linear languages are not identifiable.
  (de la Higuera, 97)
  - the rules have the shape: $A \rightarrow \alpha \beta \gamma \delta A \rightarrow \eta \nu$.

... positive skeletons?

YES,
- provided that the grammar is written in
  reversible normal form (Sakakibara, 92)
  - the regular languages are not identifiable.

... positive and negative skeletons?

YES,
- it is a special case of learning tree
  grammars from positive and negative
  examples
  (Garcia & Oncina, 93)

... positive skeletons and negative examples?

YES
- With a slight modification of the previous
  algorithm.
a distribution?

- There is not even a sensible definition of what this can be.
- The number of examples should be very large in order to have information about the existence of a rule with a very low probability.

Can we learn polynomially in the limit some subclasses of context free languages from ...

- positive examples?

- Very simple grammars (Yokomori, 91)
  - Rules with shape $A \rightarrow a, A \rightarrow aB, A \rightarrow aBC$
  - Globally deterministic in “a”
- Subclasses of even linear languages
  (Takada, 88), (Sempere & Garcia, 94), (Mäkinen, 96)
  - Rules with shape $A \rightarrow aBb, A \rightarrow a, A \rightarrow \lambda$
  - The trick is to transform $A \rightarrow aBb$ to $A \rightarrow aBb$, then we have a regular language.

- positive and negative examples?

- Even Linear Languages
  (Takada, 88), (Mäkinen, 96), (Sempere & Garcia, 94)
  - Same trick as in the previous slide.
- Linear Deterministic Languages
  (de la Higuera & Oncina, 92)
  - Rules with shape $A \rightarrow aBb, A \rightarrow \lambda$
  - $A \rightarrow aBb$ rules deterministic in “a”

- positive skeletons?

- Some classes transposed from regular languages to tree languages and then to context free.
  - k-testable tree languages (Kuutti, 93)
  - (Fernau, 02)
  - (Ishizaka, 89)
  - (Yokomori, 91)
- a distribution?
  - Stochastic
  - Deterministic
  - Linear Languages
    (de la Higuera & Oncina, 03)
  - Identification of the structure
  - Polynomial update time

- Queries?
  - Simple Deterministic Languages
    (Ishizaka, 89)
    - Grammar:
      - rules with shape $A \rightarrow a, A \rightarrow aB, A \rightarrow aBC$
    - Deterministic in "$\eta$"
  - Queries
    - Membership
    - extended equivalence queries
      (counterexample in $L(G) \otimes L$)

- PAC Learning

- Polynomial PAC (Valiant, 84)
  - The examples are taken from an unknown distribution
  - Let $\varepsilon$ be the probability of making a mistake using the hypothesis
  - Success if with a probability smaller than $\delta$ the error is bigger than $\varepsilon$
  - Polynomial in $1/\varepsilon, 1/\delta$ and the target size

Can we PAC learn context free languages from ...

... positive and negative examples?

NO,
  - If usual cryptographic rules apply:
    (Angluin & Kabanov, 91)
... positive examples?

NO,
- A consequence of the previous result.

... positive skeletons?

NO,
- Because regular languages cannot be learned ...

... positive skeletons and negative examples?

probably NO,
- If usual cryptographic rules apply,
  It should be a direct consequence of
  (Angluin & Kharitonov, 91)

3. "Pragmatic" Learning

"Pragmatic" Learning

- Many different ideas:
  - Incremental learning
  - MDL principle
  - Genetic/evolutionary algorithms
  - Reversing the parser
  - Tree automata learning
  - Merging

3.1 SEQUITUR

(http://sequence.rutgers.edu/sequitur/)
(Neville Manning & Witten, 97)
- Idea: construct a CF grammar from a very long string s, such that L(G)={s}
  - No generalization
  - Linear time
  - Good compression rates
Principle

- The grammar w.r.t the string:
  - Each rule has to be used at least twice
  - There can be no substring of length 2 that appears twice

Examples

\[ S \rightarrow abcbd \]  
\[ A \rightarrow bc \]

\[ S \rightarrow AaA \]  
\[ A \rightarrow aab \]

\[ S \rightarrow AbAb \]  
\[ A \rightarrow aa \]

abcabdcabdcabd

In the beginning, God created the heavens and the earth.
And the earth was without form, and void; and darkness was upon
the face of the deep. And the Spirit of God moved upon the face
of the waters.
And God said, Let there be light; and there was light.
And God saw the light, that it was good; and God divided the light
from the darkness.
And God called the light Day, and the darkness he called Night.
And the evening and the morning were the first day.
And God said, Let there be a firmament in the midst of the waters,
and let it divide the waters from the waters.
And God made the firmament, and divided the waters which were
under the firmament from the waters which were above the
firmament: and it was so.
And God called the firmament Heavens. And the evening and the
morning were the second day.

Sequitur options

- appending a symbol to rule \( S \);
- using an existing rule;
- creating a new rule;
- and deleting a rule.
Results

- On text:
  - 2.82 bpc
  - compress 3.46 bpc
  - gzip 3.25 bpc
  - PPNC 2.52 bpc

3.2 Using a simplicity bias

(Langley & Stromsten, 00)
Based on algorithm GRIDS (Wolff, 82)

- Main characteristics:
  - MDL principle
  - Not characterizable
  - Not tested on large benchmarks

Two learning operators

- Creation of non terminals and rules
  \[ NP \rightarrow \text{ART ADJ NOUN} \]
  \[ NP \rightarrow \text{ART ADJ ADJ NOUN} \]
  \[ NP \rightarrow \text{ART API} \]
  \[ NP \rightarrow \text{ART ADJ API} \]
  \[ API \rightarrow \text{ADJ NOUN} \]
  \[ AP2 \rightarrow \text{ADJ API} \]

- Merging two non terminals
  \[ NP \rightarrow \text{ART API} \]
  \[ NP \rightarrow \text{ART AP2} \]
  \[ API \rightarrow \text{ADJ NOUN} \]
  \[ AP2 \rightarrow \text{ADJ API} \]

Results

- Scoring function: MDL: \[ |G| + \sum_{w \in T} |d(w)| \]
- Algorithm:
  - find best merge that improves current grammar
  - if no such merge exists, find best creation
  - halt when no improvement

- On subsets of English grammars (15 rules, 8 non terminals, 9 terminals): 120 sentences to converge
  - on \((ab)^n\): all (15) strings of length \(\leq 30\)
  - on Dyck_5: all (65) strings of length \(\leq 12\)
3.3 Context free grammar induction with genetic/evolutionary algorithms

- (Wyard, 91)
- (Dupont, 94)
- (Kämmerer & Belew, 96)
- (Sakakibara & Kondo, 99)
- (Sakakibara & Muramatsu, 00)

Main issue

- Encoding a context free grammar as a string
- such that after crossovers and mutations the string is still a grammar...
- Some ideas:
  - Fill up with junk dna (Kämmerer & Belew, 96)
  - A grammar is a partition. Encode the partition

4 Applications

- Computational Biology
- Program synthesis, ILP, compiler construction
- Language models, speech & NLP
- Document structure, XML

4.1 Secondary structure predictions

- Why: find the secondary structure
- Concept: a CF grammar
- Data: long tagged strings over a small alphabet: (RNA)
- Difficulties:
  - only positive data: restrict to a subclass of CF grammars or use stochastic CF grammars
- Bibliography: Sakakibara et al. 94, Abe & Mamiya 94
Combining stochastic CFGs and n-grams over BNF sequences (Salvador & Bontempi, 2002)
- CFGs to learn the structure and long-term dependencies
- bigrams for the local relations (non-structured part)
- Sakakibara's algorithm (minimum reversible consistent CFG)
- Probability estimation (inside-outside)

4.2 Inductive logic programming
- Why: learn recursive programs
- Concept: tree automata and grammars
- Input: a transformation of examples and background knowledge into strings (SLD refutations, or terms)

• Difficulties:
  - getting the first order informations into strings/trees
  - regular grammars are very restricted
  - numeric data
  - post-transformation into a logic program
• Bibliography: Merlin, GIFT (Böstrom, 95 & 96, Bernard & cdlh, 99)

Towers
- background knowledge (shape, color)
  - shape(01, triangle), color(01, red)
  - shape(02, square), color(02, red)
  - shape(03, square), color(03, green)
  - shape(04, triangle), color(04, green)
  - shape(05, square), color(05, blue).
- observation set
  - positive examples: tower(05, 03, 01), tower(03, 02, 01), tower(02, 01).
  - negative examples: > tower(04, 06, 03), tower(06, 02, 04, 03).

GIFT
architecture of the system

4.3 Natural Language Processing
[NP [adj][NP [det (bold face)]
 [det (rep 3)]]
 [name (M)]]
 [if (bold concepts)]
 [if (non active)]
System EMILE

Entity Modeling Intelligent Learning Engine
- A context/expression pair is a sentence split into 3 parts: John (makes) tea.
- Makes is an expression
- John (.) tea is a context.
- Identifying contexts, expressions is what EMILE is about.
- How? Through clustering algorithms

An example
- the fox jumped, the dog jumped, the quick brown fox jumped, the lazy dog jumped, the fox jumped over the dog, the dog jumped over the fox, the quick brown fox jumped over the dog, the lazy dog jumped over the fox, the fox jumped over the lazy dog, the dog jumped over the quick brown fox, the lazy dog jumped over the quick brown fox.

Result of EMILE
- [0]→[18] dog jumped.
- [0]→[18] dog jumped over the [4].
- [4]→[fox]
- [4]→[quick brown [4]]
- [18]→[the]
- [18]→[the lazy]

System ABL

(Menno van Zaanen, 00..)
- Uses alignments for grammar construction
- System for unsupervised learning

4.4 Structured documents: XML

- Extract XML schema (Chidlovski 200x)
  <book>
    <part>
      <chapter>
        <sect>
          <title>
          </title>
          <abstract>
          
          </abstract>
        </sect>
      </chapter>
    </part>
  </book>
5 Conclusion

- Theoretical hardness of the polynomial time learning issues;
- determinism and linearity seem to play a strong part;
- algorithms and heuristics are based on very clever ideas;
- not enough comparable work.

Perspectives

- Tasks
- Benchmarks
- Prototypes

- Clearly identifiable open problems

Benchmarks, some features that one expects...

- Small/large/very large alphabets (2, <20, >x0,000)
- All grammars/simple grammars
- Languages or grammars (normal forms?)
- Size
  - of data set
  - of grammars
- No help (only positive data)/some help:
  - Skeletons
  - Partial structure
  - Distribution
- Noise/no noise
- Recognition/tolerance

Prototypes

- Avoid having a repetition of the DFA/stochastic DFA situation: no fixed RPNII/Algolia around;
- distribution of implementations is a necessity;
- distributing your algorithm means extra references!!!

Open problems

- Limits of learning from polynomial data? Comparison between models.
- A plausible model for polynomial identification with probability 1 or something related to this...
- Find a problem solvable on strings for DFA but not solvable on skeletons for CFGs/tree automata.

...
Bibliographical Notes for the Tutorial on Learning Context-free Grammars

Colin de la Higuera

EURISE, Université de Saint-Étienne, 23 rue du Docteur Paul Michelson, 42023 Saint-Étienne, France
cdlh@univ-st-etienne.fr.
WWW home page: http://eurise.univ-st-etienne.fr/~cdlh

Abstract. The following are just some references for the problem of context-free grammar learning. The list of references is short and only includes those references the author believes to be directly concerned with the issue of context free grammar learning. The list should be taken as indicative, as there are many references linked with other research fields (for instance computational linguistics), that are not included.

1. Surveys or introductions to the subject Grammatical inference contain many pointers to context free grammar learning: Lee [Lee96], Sakakibara [Sak97], Honavar and de la Higuera [HdlH01], de la Higuera [dIH00].

2. Teaching Based on teaching models [GK95,GM96], de la Higuera proposes the model of identification in the limit from polynomial time and data [dIH97].

3. Learning with queries A survey paper of the field (by Angluin) is [Ang01], where the openness of the problems related to non deterministic finite automata or context-free grammars is recalled. Membership and equivalence queries together form a Minimal Adequate Teacher. In the case of context-free grammars the negative proofs by Angluin and Kharitonov [AK91] with MATs are related to cryptographic assumptions. On the other hand, if structural information is available, Sakakibara proves the learnability of the class of context-free grammars in this model [Sak90]. The model has received considerable attention and there are many papers on learning with different sorts of queries, Sakakibara [Sak87] learns context-free grammars from queries.

4. Learnable Superclasses One first option to obtain positive results is that of extending results from the class of the regular languages (when represented by DFA). This is the line followed for even linear languages for which a number of results is known: Takada [Tak88], Sempere and García [SG94] and Mäkinen [Mäk96] have all worked on this class and give similar results by different techniques. Even linear languages are generated by grammars where the rules are balanced: the right hands are composed either of terminal symbols only, or are of the form $uTv$ where only $T$ is non-terminal and $u$ and $v$ have identical length. Following this trend other results concerning
this class of languages (or similar) are an extension to a hierarchy of linear
languages (Takada, [Tak94]), the case where only positive strings are avail-
able (Koshiba et al., [KMT97]) (but then only a sub-class is identifiable),
and the case where the positive information is structural (if you know where
the center of the strings is) (Sempere and Nagataja, [SN98]). Different sur-
veys on the subject have been written by Yokomori [Yok89], Lee [Lee96] or
Sakakibara [Sak97]. In [dH00a] a larger class, that of deterministic linear
grammars is proved by de la Higuera and Oncina to be identifiable from
polynomial time and data. A general way of detecting if this is the case for
other classes of grammars is given by the same authors in [dH00b]; Giorda-
no [Gio94] proposes to see the problem as that of an exhaustive search in
a lattice defined by the Reynolds cover over grammars in Chomsky normal
form.

5. Extra information In a series of papers Sakakibara gives techniques to
learn context-free grammars from structured data [Sak90], data containing
positive structured data [Sak92], unstructured data by genetic algorithms
[SK99], and data containing some structure again by genetic algorithms
[SM00]. There are also a number of very specific results that will be of interest
to specialists: Ishizaka [Ish89] learns another restricted class of context-free
grammars, that of the simple deterministic grammars. It should be noticed
that these grammars are not linear. Pure context-free languages are built
from grammars where the non-terminal symbols are also terminal, and their
learnability has been studied by Koshiba et al. [KMT00]. Kremer [Kre97]
proposes a parallel algorithm in order to learn context-free grammars.
A special case should be made of Nevill-Manning and Witten’s algorithm
SEQUITUR [NMW97]. Although it cannot be included into the class of gram-
mar induction algorithms, as it has no generalization capacity, it is an elegant
way of deducing a context-free grammar from just one (usually very long)
sentence. This grammar can then generate just one string; the original one.
Running in linear time and space, SEQUITUR is more than just a compres-
sion technique as it also explains the data it has to compress by giving its
structural hierarchical nature.

6. Tree Automata Tree automata are the direct extension of DFA and NFA for
trees instead of strings. They also provide a smooth link between automata
and context-free grammars, Learning tree automata has already been dealt
with in Fu and Booth’s survey [FB75].
There are very strong links between learning context-free grammars from
bracketed data (or the actual skeletons or parse-trees without inner labels)
and learning regular tree grammars. For instance, Sakakibara [Sak90] uses
algorithms that learn regular tree automata (these can be adapted very often
from algorithms that learn automata from strings), and transforms these into
algorithms that learn context-free grammars from bracketed data.
The advantages are nevertheless that a deterministic case exists, allowing
to re-use results from DFA learning in this setting. An extension of RPNL
in which the best known known algorithm for stochastic DFA (ALERGIA) is proposed by
Carrasco et al. [COCR01]. For the case of learning from positive structural data only, Knuth has presented a state of the art in [KS94]; for the same problem Fernau [Fe02] extends to tree automata results allowing to state when a class is identifiable from positive data only. Calera et al. [CRC98] give a polynomial algorithm computing the relative entropy between regular tree languages (this can be used in a learning algorithm) and use tree automata in compression tasks [CRC00].

7. Artificial Intelligence Approaches The search space for the context-free grammar problems has hardly been studied, but it has been seen as a version space by Vanlehn and Ball in [VB87], described in [Gio94] by Giordano, and also used by Langley and Stromsten [LS00] by means of a simplicity bias and a representation change.

In the case of context-free grammars, genetic algorithms (this time on the rules) were tried by Sakakibara and Kondo [SK99]. Experiments suggest that the knowledge of part of the structure (some parenthesis) may help and reduce the number of generations needed to identify (Sakakibara and Muramatsu, [SM00]).

8. Stochastic Context-Free Grammars The question of inferring probabilistic context-free grammars is going to prove even harder than that of doing the same with finite state machines. Yet the problem has been shown of interest in speech recognition (Wang and Acero, [WA02]) or in computational biology (Sakakibara et al., [SBH+94]) because these grammars can capture the long term dependencies that can arise for instance in folding, and thus in secondary structure. A first problem that requires study is that of checking if a given grammar is consistent: it is easy to derive a set of rule probabilities for which there is a strictly positive probability that the derivations of the grammar do not halt: larger and larger derivation trees are constructed that (with probability one) never stop expanding. Booth and Thompson [BT73] give consistency conditions which are proved to hold if the probabilities are estimated from the data (Sánchez and Benedí, [SB97]). Another “elementary” problem is that of parsing with such a grammar (Stolcke, [Sto05]).

There are two levels of learning problems:

- if you know the grammar rules you can try to estimate the probabilities that fit best. The usual algorithm in that case is the well-known inside-outside algorithm introduced by Baker [Bak79] and studied by Lari and Young [LY90]. Alternative estimation techniques exist (as for example by Ra and Stockman [RS99]), or Sakakibara et al. [SBH+94].

- you can first learn the rules and then the probabilities. If you have additional information about the data, such as some of its structure, you can turn to adapting a tree-automaton learning algorithm (this is done by Sakakibara in [Sak90] and [Sak92]; if this is not the case it may be necessary to learn a simplified automaton, corresponding for instance to a local language, and then estimate the probabilities: this path is taken by Rico et al. [RJCR02]; the direct approach of inferring directly the context free grammars is hard and seems to be attacked only by artifi-
cial intelligence techniques, such as genetic algorithms (Kammeyer and Belew, [KB96]).

9. Categorial Grammars Computational linguists have long been interested in working on grammatical models that would not fit into Chomsky’s hierarchy. Furthermore, their objective is to find suitable models for syntax and semantics to be interlinked, and provide a logic based description language. Key ideas relating such models with the questions of language identification can be found in Kanazawa’s book [Kan98], and discussion relating this to the way children learn language can be found in papers by a variety of authors, as for instance Teller [Tel98]. The situation is still unclear, as positive results can only be obtained for special classes of grammars (see for instance Forest and Le Nir [FN02]), whereas, here again, the corresponding combinatorial problems (for instance that of finding the smallest consistent grammar) appear to be intractable (Costa Florêncio, [Flo02]).

10. Computational Linguistics There has of course been always a lot of interest in relating grammatical inference with natural language. One direction has been taken by Adriaans through shallow grammars [Adr92] (using categorial grammars); this theoretical work is the backbone of the EMILE prototype [AV02].

11. Applications in Computational Biology Molecular biology has necessarily the data and the problems for grammatical inference scientists to work on. For the past 10 years this has been so, and even if the most successful methods in the field are not necessarily those using grammars or automata, there are sufficient features in language theory for work to continue. Brazma et al. propose an overview of the situation [BJVU98], with special emphasis on pattern languages. Determining common patterns in DNA, RNA or protein sequences allows to build alignments, discriminate members of families from non members, and the discovery of new members. Wang et al. [WBS+99] apply such techniques for DNA sequence classification tasks: patterns are induced through a learning stage, and used to score in a classification stage. Sakakibara et al. [SBH+94] learn stochastic context-free grammars from tRNA sequences. The fact the induced grammar is context-free allows to discover and model part of the secondary structure. In the same line of research, secondary structure prediction was detected also by context-free grammars by Abe and Mamitsuka [AM97]. An experimental result by Salvador and Benedí [SB02] is that a combination of context-free grammars and bi-grams obtains good results: they use Sakakibara’s algorithm ([Sak92]) on data that can present some very structured regions isolated and others that are not structured.

A lot of more general work has concerned the study of hidden Markov models, their relationship with grammars. Lyngso et al. study all typical distances between distributions in [LPN99] and prove intractability results in [LP01]. The technique is improved to be able to also compare context-free stochastic grammars [JLP01].

12. ILP Inductive logic programming [Mug99] has several links with grammatical inference. It shares some of its objectives (when learning recursive rules)
and sometimes its techniques, Boström's system MERLIN parses the data by
the background knowledge and uses this information to learn a determinis-
tic finite automaton [Bo96], or a stochastic one [Bo98]. System GIFT by
Bernard et al. [Bde laH01,HBJ02] improves on MERLIN, by learning directly-
tree automata, thus not needing to lose representation capacity by having
to linearize the data.

13. **Document Management** There are a number of possible applications dealing
with documents as data. They either involve constructing dictionaries
(Ahonen et al., [AMN94]) or inferring the grammar generating the tags that
have been used (Young-Lai and Tompa, [YLT00]).

The rise of XML has lead to some new challenges for the field (Fernau points
out some of these in [Fer01]), Chüdiovski obtains some preliminary results
by using context free grammar learning techniques [Chi01]. For these reasons
there has lately been increasing interest in tree automata and tree patterns
(Airumura, [ASA01]).

14. **Compression** SEQUITUR [NMW97] learns a grammar from just one string.
The obtained grammar can then only generate the string SEQUITUR has
learned from. Compression results with this method are comparable to the
best compression methods. Moreover, SEQUITUR extracts the structure of
the text.

N-grams have allowed to build good compression schemes on text. Using the
same sort of ideas, Rico-Juan et al. [RJRC02] first learn a k-testable tree
automaton, and then probabilize this. The obtained model is then used with
very good compression rates on tree like data (XML files, for instance).

**References**


[AFvZ02] P. Adriaans, H. Fernau, and M. van Zaamnen, editors. *Grammatical Inference:
Algorithms and Applications, Proceedings of ICGI '00*, volume 2484 of

[AK91] D. Angluín and M. Kharitonov. When won't membership queries help? In
*Proceedings of 24th ACM Symp. on Theory of Computing*, pages 444–454,


[AM97] N. Abe and H. Mamitsuka. Predicting protein secondary structure using

documents: An application of grammatical inference. In Carrasco and
Oncina [CO94], pages 153–167.


[ASA01] H. Airumura, H. Sakamoto, and S. Arikawa. Efficient learning of semi-
structured data from queries. In Abe et al. [AKZ01], pages 315–331.

In Adriaans et al. [AFvZ02], pages 293–295.


[dIH02b] C. de la Higuera and J. Oncina. On sufficient conditions to identify in the limit classes of grammars from polynomial time and data. In Adriáns et al. [AFvZ02], pages 134–148.


[Flo02] C. Costa Florêncio. Consistent identification in the limit of rigid grammars from strings is NP-hard. In Adriaans et al. [AFvZ02], pages 49–62.


Inference of a Subclass of Context Free Grammars Using Positive Samples

J.A.Laxminarayana and G.Nagaraja*

Computer Science and Engineering Department
Indian Institute of Technology, Bombay
INDIA \{jalana, gn\}@cse.iitb.ac.in

Abstract. It is known that family of context-free languages is not identifiable in the limit from positive data. However, it is known that some useful subclasses of context-free languages are identifiable in the limit. In this work we introduce a new subclass of context-free languages, Terminal Distinguishable Context Free Languages (TDCFL) through a grammatical characterization. The Greibach Normal Form representation is used to represent the rewriting rules of underlying grammar of TDCFL and terminal distinguishability property is employed to merge contextual entities. A polynomial time algorithm is proposed to identify a TDCFL and it is shown that the algorithm correctly identifies any TDCFL in the limit if enough positive samples are given.

1 Introduction

Grammatical inference is a well established research area in machine learning focussing on the problem of identifying a correct representation of the target language from a given set of samples. Since Gold’s [3] seminal work in which he introduced the concept of *identification in limit*, there has been a remarkable amount of work to establish the theory of Grammatical Inference. The works [8] and [4] give an excellent survey and introduction to the subject.

Learning the entire class of the context-free languages using only positive samples is not identifiable in the limit as per Gold’s result. Since it was established by the works of Angluin [1,2] that subclasses of regular languages are identifiable from the positive samples, a natural option is to extend the results to the class of context free languages. The class of even linear languages is a subclass of context free languages and possesses many properties of regular languages. Hence Radhakrishnan and Nagaraja [7], Sempere and Garcia [9] and others have all worked on this class and proposed different inferencing methods. Lee [6] provides a good survey of literature on learning of context free languages.

A context free grammar can be represented in a number of equivalent normal forms. In this work, we propose to use the standard Greibach Normal Form (GNF) to represent the rewriting rules of the grammar. A subclass of the context free language, Terminal Distinguishable Context Free Language (TDCFL)

* corresponding author
is introduced and its grammatical characterization is given. Further a polynomial time algorithm to infer a Terminal Distinguishable Context Free Grammar (TDCFG) is presented with correctness proof.

The paper is organised as follows. Necessary background from language theory is provided in section 2. Definition of the new subclass is presented in section 3 along with its properties. Section 4 deals with the proposed inferencing model for TDCFG. An algorithm to infer TDCFG is given in section 5, whereas section 6 concludes the paper with pointers for future work.

2 Preliminaries

Some basic concepts about formal language theory and formal grammars are introduced. Most of them are found in any introductory book on formal languages and automata eg. [5].

In what follows $\Sigma$ denotes an alphabet and $\Sigma^*$ is the set of words over the alphabet $\Sigma$. Given an alphabet $\Sigma$ and $x \in \Sigma^*$, $|x|$ is used to denote the length of $x$. $\lambda$ denotes the empty string with $|\lambda| = 0$. A prefix of a string is any number of leading symbols of that string, and a suffix is any number of trailing symbols. $\text{Pref}(x)$ and $\text{Pref}(x)$ are used to denote set of all suffixes and prefixes of string $x$ respectively. Given a string $w$ where $w = \sigma_1 \sigma_2 \ldots \sigma_n$ and $\sigma_i \in \Sigma, 1 \leq i \leq n$, the terms $\text{First}(w)$ and $\text{Last}(w)$ denote the characters $\sigma_1$ and $\sigma_n$ respectively.

A grammar is a quadruple $G = (N, \Sigma, P, S)$, where $N$ is the set of non terminal symbols, $\Sigma$ (with $\Sigma \cap N = \emptyset$) the set of terminal symbols, $P$ the rule set and $S \in N$ the start symbol. A grammar is called context free if rules are of the form $A \rightarrow \beta$ where $A \in N$ and $\beta \in \{N \cup \Sigma\}^*$. Given a grammar $G = (N, \Sigma, P, S)$, and two strings $x, y \in \{N \cup \Sigma\}^*$, one can say that 1) $x$ directly derives to $y$ (using rule $\alpha \rightarrow \beta$ ) if $y$ is obtained from $x$ by replacing one occurrence of the subword $\alpha$ in $x$ by $\beta$ and we write $x \Rightarrow y$; 2) $x$ derives to $y$, denoted by $x \Rightarrow^* y$, if $y$ can be obtained by applying to $x$ a finite set of productions of $P$.

The language generated by the grammar $G$ is denoted as $L(G)$ and is defined as the set $L(G) = \{x \in \Sigma^* | S \Rightarrow^*_G x\}$. Generally, we will work with reduced grammars, that is, grammars without useless symbols, unit productions(with the exception of productions with start symbol) or empty productions. $L(G, A)$ denotes the language obtained by the grammar $G_A = (N, \Sigma, P, A)$. It is written as $L(A)$ instead of $L(G, A)$ whenever $G$ is understood.

Definition 1. A context free grammar $G = (N, \Sigma, P, S)$ is in Greibach normal form (GNF), if every rule is of the form:

$$A \rightarrow a\gamma, \text{ where } a \in \Sigma \text{ and } \gamma \in N^*.$$

where $a$ is any terminal and $\gamma$ is a (possibly empty) string of non-terminals. In addition, the rule $S \rightarrow \lambda$ is permitted, where $S$ is the start variable. We state the following theorem which is a re-phrased form of Theorem 4.6 in [5].

Theorem 1. Every context free language without $\lambda$ can be generated by a context free grammar in Greibach Normal Form.
3 Terminal Distinguishable Context Free Languages

Let \( L \) be a context free language and \( G = (N, \Sigma, P, S) \) be CFG in GNF to denote \( L \). Let \( A \rightarrow a\gamma \) be a production rule, where \( \gamma = A_1 A_2 \ldots A_m \) and \( A_j \in N \) for \( 1 \leq j \leq m \). \( L(\gamma) \) is defined as the concatenation of the substrings derivable from each of the nonterminal in \( \gamma \). \( \text{Ter}(x) \) denotes the set of unique symbols in the string \( x \).

**Definition 2.** \( L(\gamma) = \{ x_1 x_2 \ldots x_m \mid x_j \in L(A_j), 1 \leq j \leq m \} \)

Following definitions are used to establish the terminal distinguishable properties of a context free language.

**Definition 3.** A CFG in GNF possesses backward determinism property iff \( B \Rightarrow \) \( w \) and \( C \Rightarrow w \) implies \( B = C \)

**Definition 4.** A CFG in GNF possesses terminal completeness property iff \( \forall x, y \in L(A) \) where \( A \in N \), \( \text{Ter}(x) = \text{Ter}(y) \)

**Definition 5.** A CFG in GNF possesses terminal dissimilarity property if

(i) \( A, B, C \in N - \{ S \} \), \( \gamma_1, \gamma_2 \in N^+ \), \( a \in \Sigma \) and \( A \rightarrow a\gamma_1 \), \( A \rightarrow a\gamma_2 \) are in \( P \) then \( \forall x \in L(\gamma_1) \) and \( \forall y \in L(\gamma_2) \), \( \text{Ter}(x) \neq \text{Ter}(y) \).

(ii) \( S \rightarrow A \) and \( S \rightarrow B \), where \( S, A, B \in N \) and \( S \) is the start symbol, are in \( P \) then \( \forall x \in L(A) \) and \( \forall y \in L(B) \), \( \text{Ter}(x) \neq \text{Ter}(y) \).

**Definition 6.** A Context Free Grammar \( G \) is called a Terminal Distinguishable Context Free Grammar(TDCFG) if it possess backward determinism, terminal completeness and terminal dissimilarity properties.

**Definition 7.** A Context Free Language \( L \) is Terminal Distinguishable Context Free Language(TDCFL) if there exists a TDCFG \( G \) for \( L \).

**Example 1.** The grammar \( G_1 = (\{ S, A, B \}, \{ a, b \}, \{ S \rightarrow aSB | bSA | aB, A \rightarrow a, B \rightarrow b \}, S) \) is a TDCFG and denotes the language \( \{ww^r \mid w \in (a + b)^* \) and \( w^r \) is the reversal of string \( w \} \).

Let \( S^+ = \{ x_1, x_2, \ldots, x_n \} \) be a set of given sample strings. One can define a canonical context free grammar in GNF as follows.

**Definition 8.** The canonical context free grammar in GNF \( G_c = (N_c, \Sigma_c, P_c, S) \), associated with the positive samples \( S^+ \) is defined as follows:

**Step 1:** Examine each \( x_i \in S^+ \) and identify all the distinct terminal symbols used in the generation of the strings of \( S^+ \). Call this set \( \Sigma_c \).

**Step 2:** For each \( x_i = a_{i_1} a_{i_2} \ldots a_{i_m} \), \( x_i \in S^+ \), define the distinct set of rewrite rules. \( S \rightarrow A_{i_1}, A_{i_1} \rightarrow a_{i_1}A_{i_2} \ldots A_{i_m}, A_{i_2} \rightarrow a_{i_2}, A_{i_3} \rightarrow a_{i_3}, \ldots, A_{i_m} \rightarrow a_{i_m} \). Each \( A_i \) represents a new nonterminal symbol.

**Step 3:** The set \( N_c \) consists of \( S \) and all the distinct nonterminal symbols produced by step 2. The set \( P_c \) consists of all the distinct rewrite rules defined by step 2.
Example 2. Suppose that $S^+ = \{ab, aabb, aaabbb\}$. The canonical context free grammar in GNF $G_c$ is

\[
\begin{align*}
\Sigma_c &= \{a, b\}, \quad N_c = \{S, A_{11}, A_{12}, A_{21}, \ldots, A_{24}, A_{31}, \ldots, A_{36}\} \\
P_c &= \{S \rightarrow A_{11} | A_{21} \rightarrow aA_{12}, A_{12} \rightarrow b, \\
& \quad A_{22} \rightarrow a, A_{23} \rightarrow b, A_{24} \rightarrow b, \\
& \quad A_{31} \rightarrow aA_{32}A_{33}A_{34}A_{35}A_{36}, A_{32} \rightarrow a, A_{33} \rightarrow a, A_{34} \rightarrow b, A_{35} \rightarrow b, A_{36} \rightarrow b \} \}
\end{align*}
\]

Construction of a canonical CFG in GNF for any given sample set is trivial. A skeletal structure description (or a GNF-skeleton in short) of a string belonging to a language is a derivation tree with only the terminal symbols. A skeleton exhibits all the grouping structures (phrase structures) of the string without naming the syntactic variables used in the description. Figure 1 shows the GNF-skeletons for a sample set $\{ab, aabb, aaabbb\}$ in which all the internal nodes are represented by corresponding non-terminals.

![Fig. 1. GNF skeletons for the sample set \{ab, aabb, aaabbb\}](image)

The yield of a node $N_{ij}$, $\text{Yield}(N_{ij})$ in a derivation tree is the string obtained by looking over all its immediate children (i.e. siblings in immediate next level) from left to right. It is obvious that every node $N_{ij}$ in a GNF-skeleton has an yield of the form $a\gamma$, where $a \in \Sigma$ and $\gamma \in N^*$. Let $ND(S^+)$ denotes the set of internal nodes found in all GNF-skeletons of $S^+$, where $S^+$ is a set of strings. Formally, $ND(S^+) = \bigcup_{w_i \in S^+} \{N_{ij} | 1 \leq j \leq |w_i| \}$ where $1 \leq i \leq |S^+|$.

The nodes of a GNF-skeleton are divided into two disjoint groups $ND_1$ and $ND_2$ based on the length of the substring $\gamma$. Formally, $ND(S^+) = ND_1 \cup ND_2$ such that $ND_1 \cap ND_2 = \emptyset$, where $ND_1 = \{n \in ND(S^+) | \text{Yield}(n) = a\gamma$ and $|\gamma| > 0 \}$ and $ND_2 = \{n \in ND(S^+) | \text{Yield}(n) = a\gamma$ and $|\gamma| = 0 \}$. The nodes in the set $ND_1$ represent contextual information whereas the set $ND_2$ represent the nodes having single terminal symbol as their yield. By referring Example 2, $ND_1 = \{A_{11}, A_{21}, A_{31}\}$ and $ND_2 = \{A_{12}, A_{22}, A_{23}, A_{32}, A_{33}, A_{34}, A_{35}, A_{36}\}$

Let $x_i \in S^+$ be $i^{\text{th}}$ string such that $x_i = a_{i1}a_{i2} \ldots a_{iK}$ where $K = |x_i|$. In a GNF-skeleton for the string $x_i$, the frontier string of a node $N_{ij}$ denoted by $FS(N_{ij})$ is defined as the string $a_{ij}a_{ij+1} \ldots a_{iK}$. Also, the head string of a node $N_{ij}$, denoted $\text{Head}(N_{ij})$ is defined as the string $a_{i1}a_{i2} \ldots a_{ij-1}$. A skeletal structure node function of a node $N_{ij}$ in a skeleton, denoted by $SSNF(N_{ij})$,
is an ordered pair; $N_{ij}$ is mapped to a value in the range $\Sigma^* \times 2^\Sigma$ as given by $SSNF(N_{ij}) = (\text{Head}(N_{ij}), \text{Ter}(FS(N_{ij})))$. In Figure 1 for node $N_{33}$, $FS(N_{33}) = \text{abb}$, $\text{Head}(N_{33}) = aa$ and $SSNF(N_{33}) = \langle aa, \{a, b\} \rangle$.

4 Strategy to infer TDCFG

It is known that the pumping lemma is used to show that certain languages are not CFL. However, here we use it for the purpose of inductive reasoning on the sample set from a CFL. As per the pumping lemma, for the string $z = uvwxy \in S^+$, we expect strings in the form $uv^ixwz^iy \in L$, where $L$ is the target CFL and $i \geq 1$. This suggests that at least one of the substrings $v, w$ and $x$ should be non-empty and hence there are 7 possible cases to experiment with. We consider the case where $v$ and $x$ are nonempty. However the proposed method can be extended to other cases as well.

To get a desired target CFG, we need to identify more contextual entities in the set $ND(S^+)$. A contextual entity is a non-empty subset of $ND_2$ found in any GNF-skeleton. As a first step, we can construct the canonical CFG in GNF as explained in the previous section. It is proposed to use a pruning model to obtain a set of optimal contextual entities. The pruning model constructs an initial prune set of $ND_2$, to indicate the occurrence of some contextual entities. Here, a prune set is defined as a subset of $2^{ND(S^+)}$. By inducing pumping property into the initial Prune set a final list of contextual entities is obtained using which a modified node list is generated.

4.1 Construction of Initial Prune Sets

In order to construct an initial prune set, we need to define an equivalence relation between the pairs of nodes belonging to the node set $ND(S^+)$. Let $\bowtie$ be an operator to denote an equivalence relation called $P$-Equivalence.

**Definition 9.** Two nodes $N_{ij}, N_{kl} \in ND(S^+)$ are $P$-Equivalent, denoted by $N_{ij} \bowtie N_{kl}$, iff $(\text{Suff}(\text{Head}(N_{ij})) \cap \text{Suff}(\text{Head}(N_{kl})) \neq \emptyset) \land (\text{Pref}(FS(N_{ij})) \cap \text{Pref}(FS(N_{kl})) \neq \emptyset)$

For every pair of nodes $N_{ij}, N_{kl} \in ND_2$, $N_{ij} \bowtie N_{kl}$ is computed and if such equivalence existed, a cluster of nodes is formed. This cluster of nodes is formed so that it exhibits some contextual information. Each such cluster is added into a set called prune set $\mathbb{E}$. This process is shown as procedure Initialise-Prune-Sets.

4.2 Inducing Pumping property

If the initialisation of prune sets is successful, i.e. if $\mathbb{E} \neq \emptyset$, then the pruning model induces pumping property into the contextual entities in $\mathbb{E}$. For each of the cluster $C_1 = N_{ij} \ldots N_u \in \mathbb{E}$ a cluster $C_2$ is formed such that $C_2 = N_{ij-1}C_1N_{u+1}$.
**Procedure 1** Initialise-Prune-Sets

**Input:** The node set $ND(S^+) = ND_1 \cup ND_2$.

**Output:** Prune Set $R$.

$R = \emptyset$.

for all nodes $N_{ij}, N_{kl} \in ND_2$ do

if $N_{ij} \Rightarrow N_{kl}$ then

Compute the length of smallest common suffix of $Head(N_{ij})$ and $Head(N_{kl})$ and store it in $p_1$.

Compute the length of smallest common prefix of $FS(N_{ij})$ and $FS(N_{kl})$ and store it in $p_2$.

$p = \min(p_1, p_2)$

Form clusters of $2p$ nodes like: $(N_{ij-p}N_{ij-p+1} \ldots N_{ij} \ldots N_{ij+p-2}N_{ij+p-1})$ and $(N_{kl-p}N_{kl-p+1} \ldots N_{kl} \ldots N_{kl+p-2}N_{kl+p-1})$

Add the newly formed clusters of nodes to $R$.

end if

end for

**Procedure 2** Induce-Pumping

**Input:** The Prune set $R$.

**Output:** Modified Prune Set $Q$.

$Q = R$.

for all $C_1 = (N_{ij} \ldots N_{il}) \in Q$ do

if $Last(Head(N_{ij})) \neq \lambda$ and $First(FS(N_{il})) \neq \lambda$ then

Form a cluster $C_2 = N_{ij-1}C_1N_{ij+1}$

Add the cluster $C_2$ to $Q$.

end if

end for

where $Last(Head(N_{ij})) \neq \lambda$ and $First(FS(N_{il})) \neq \lambda$. Here, it should be observed that we are interested only in the clusters of nodes belonging to the same parent node. Each new cluster $C_2$ is added to the set $Q$, which represents pruned sets with induced pumping property. Procedure Induce-Pumping shows the construction of modified prune sets.

**4.3 Selecting Contextual Candidates**

The modified prune set resulting from Procedure Induce-Pumping contains all possible contextual entities w.r.t $ND(S^+)$. Since many of the entities are either redundant or overlap each other, there is a need for some selection criterion to select the most useful clusters from the set $Q$ and to discard other inferior clusters.

There are several heuristics to select the most useful clusters from $Q$. We define a term $Result$ which is used in the following discussion of a heuristic for selection.

**Definition 10.** Let $C = N_{ij}N_{ij+1} \ldots N_{il} \in Q$. The $Result(C)$ is defined as $Result(C) = Yield(N_{ij})Yield(N_{ij+1}) \ldots Yield(N_{il})$. 
**Procedure 3 Select-Clusters**

**Input:** Prune set $Q$

**Output:** Modified Prune Set $\hat{Q}$

for all $C_1, C_2 \in Q$ do

$ResC_1 = Result(C_1); ResC_2 = Result(C_2)$.  
if ($ResC_1 = ResC_2$) or ($C_1$ partially overlaps $C_2$) then

Initialise $hit_1$ and $hit_2$ with 0.

for all $C_3 \in Q$ do

if $C_1 \subseteq C_3$ then  
$hit_1 = hit_1 + 1$

end if

if $C_2 \subseteq C_3$ then

$hit_2 = hit_2 + 1$

end if

end for

if $hit_2 > hit_1$ then

swap the clusters $C_1$ and $C_2$.

end if

if $C_1$ and $C_2$ are having same parent node then

Delete $C_2$ from $Q$

end if

end if

end for

The heuristic uses some empirical measures to decide the usefulness of a cluster in $Q$. Whenever two clusters match in their derived Result, one of the clusters is selected to be retained in $Q$ according to the number of appearance it has in other clusters. Again if the selected cluster is $C$, before retaining the cluster in $Q$, a consistency check is carried out and the cluster is added if there is no conflict among existing clusters with $C$. If two clusters partially overlap each other, then it is called a conflict. A procedure Select-Clusters is presented which successfully implements this heuristic and ensures that only useful clusters are retained in $Q$.

### 4.4 Inference of CFG in GNF

Once we have the modified prune sets $Q$, we can venture into the next task of constructing CFG in GNF. For each cluster present in $Q$ we do construct a GNF-skeleton with a unique label given to the root. Each such root label is added to the node set $ND_1$.

Now consider two nodes $N_{ij}, N_{kl} \in ND_1$. Define $N_{ij} \sim N_{kl}$, iff $SSNF(N_{ij}) = SSNF(N_{kl})$ or $FS(N_{ij}) = FS(N_{kl})$.

Again consider two nodes $N_{ij}, N_{kl} \in ND_2$. Define $N_{ij} \simeq N_{kl}$, iff $Yield(N_{ij}) = Yield(N_{kl})$.

By using these equivalence relations on the nodes found in $ND_1$ and $ND_2$, we can get a partition on the set of nodes $ND(S^+)$. Using this partition and $Q$
we can generate the CFG in GNF having terminal distinguishability property. The procedure Infer-Grammar outlines salient steps of this process.

**Algorithm 1: Infer-Grammar**

**Input:** A nonempty set of positive sample \( S^+ \)

**Output:** Inferred TDCFG.

1. Identify \( ND(S^+) = ND_1 \cup ND_2 \) using the GNF-skeletons for \( S^+ \).
2. Using a pruning model obtain a modified prune set \( Q \).
3. For each cluster \( C \in Q \), construct GNF-skeletons for \( \text{Result}(C) \) and add the root labels of such GNF-skeletons to \( ND_1 \).
4. Modify the GNF-skeletons of \( S^+ \) using \( Q \) by replacing each cluster of \( C \in Q \) found in GNF-skeletons by corresponding GNF-skeleton of \( C \).
5. Compute the SSNF and FS of each of the node in \( ND_1 \cup ND_2 \).
6. Compute the equivalence classes of \( \sim \) applied to \( ND_1 \) and \( \equiv \) applied to \( ND_2 \).
7. Enumerate every equivalence class \( S_1, S_2, \ldots, S_i \). The equivalence class which contains the GNF-skeleton roots is labeled with \( S \).
8. Substitute every internal node of GNF-skeleton by its equivalence class (every GNF-skeleton is transformed into a derivation tree).
9. Generate a grammar \( G \) from the derivation trees.

Following example describes the steps in generating TDCFG from a given set of samples using the procedure Infer-Grammar.

**Example 3.** Let the sample set be \( S^+ = \{ab, aabb, aaabbb\} \)

The GNF-skeletons are constructed for the sample set \( S^+ \) as shown in Figure 1. Using these GNF-skeletons a node set \( ND(S^+) \) is constructed: \( ND(S^+) = ND_1 \cup ND_2 = \{N_{11}, N_{21}, N_{31}\} \cup \{N_{12}, N_{22}, N_{23}, N_{24}, N_{32}, N_{33}, N_{34}, N_{35}, N_{36}\} \). Head strings and frontier strings for each of the node in the \( ND(S^+) \) are computed and are shown in Table 1.

Initial prune set is computed using the procedure Initialise-Prune-Sets. We can show that nodes \( N_{12} \) and \( N_{32} \) are P-Equivalent. \( \text{Suff}(\text{Head}(N_{12})) = \text{Suff}(a) = \{a\} \) and \( \text{Suff}(\text{Head}(N_{23})) = \text{Suff}(aa) = \{a, aa\} \). Similarly, \( \text{Pref}(\text{FS}(N_{12})) = \text{Pref}(b) = \{b\} \) and \( \text{Pref}(\text{FS}(N_{23})) = \text{Pref}(bb) = \{b, bb\} \). Since \( \text{Suff}(\text{Head}(N_{12})) \cap \text{Suff}(\text{Head}(N_{23})) \neq \emptyset \) and \( \text{Pref}(\text{FS}(N_{12})) \cap \text{Pref}(\text{FS}(N_{23})) \neq \emptyset \), we can conclude that \( N_{12} \equiv N_{23} \). In a similar way we can show that \( N_{12} \equiv N_{34} \) and \( N_{23} \equiv N_{34} \). This establishment of P-equivalence between the pair of nodes helps in forming initial Prune Set \( R = \{ (N_{11}, N_{12}), (N_{22}, N_{23}), (N_{33}, N_{34}) \} \).

Next step is to induce the pumping property into the set \( R \). Consider a cluster \( C_1 = (N_{22}, N_{23}) \in R \). Observe that \( \text{Last}(\text{Head}(N_{22})) = \text{Last}(a) = a \neq \lambda \) and \( \text{First}(\text{FS}(N_{23})) = \text{Last}(b) = b \neq \lambda \). Since \( \text{Last}(\text{Head}(N_{22})) \neq \lambda \) and \( \text{First}(\text{FS}(N_{23})) \neq \lambda \), we form a new cluster called \( C_2 = (N_{21}, N_{22}, N_{23}, N_{24}) \) and add to the set \( Q \). By using the procedure Induce-Pumping, a modified prune set \( Q \) is generated, i.e., \( Q = \{ (N_{11}, N_{12}), (N_{22}, N_{23}), (N_{33}, N_{34}), (N_{21}, N_{22}, N_{23}, N_{24}), (N_{32}, N_{33}, N_{34}, N_{35}), (N_{31}, N_{32}, N_{33}, N_{34}, N_{35}, N_{36}), (N_{31}, N_{32}, N_{33}, N_{34}, N_{35}, N_{36}) \} \).
Table 1. Computation of head and frontier strings

<table>
<thead>
<tr>
<th>Head</th>
<th>FS</th>
<th>Iter(FS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_{11}$</td>
<td>$a \ b$</td>
<td>${a, b}$</td>
</tr>
<tr>
<td>$N_{12}$</td>
<td>$a \ b$</td>
<td>${b}$</td>
</tr>
<tr>
<td>$N_{21}$</td>
<td>$a \ b$</td>
<td>${a, b}$</td>
</tr>
<tr>
<td>$N_{22}$</td>
<td>$a \ b$</td>
<td>${a, b}$</td>
</tr>
<tr>
<td>$N_{23}$</td>
<td>$a \ b$</td>
<td>${b}$</td>
</tr>
<tr>
<td>$N_{24}$</td>
<td>$ab$</td>
<td>${b}$</td>
</tr>
<tr>
<td>$N_{31}$</td>
<td>$aa\ b$</td>
<td>${a, b}$</td>
</tr>
<tr>
<td>$N_{32}$</td>
<td>$a\ abb$</td>
<td>${a, b}$</td>
</tr>
<tr>
<td>$N_{33}$</td>
<td>$aa\ bb$</td>
<td>${a, b}$</td>
</tr>
<tr>
<td>$N_{34}$</td>
<td>$aa\ bb$</td>
<td>${b}$</td>
</tr>
<tr>
<td>$N_{35}$</td>
<td>$aa\ bb$</td>
<td>${b}$</td>
</tr>
<tr>
<td>$N_{36}$</td>
<td>$aa\ bb$</td>
<td>${b}$</td>
</tr>
</tbody>
</table>

Having formed the set $\mathcal{Q}$, Select-Clusters procedure is used to select most appropriate clusters to represent the syntactic patterns in the sample set. Consider the clusters $C_1 = (N_{11}, N_{12})$ and $C_2 = (N_{22}, N_{23})$ having identical yield. We find that hit value of $C_1$ is 1 and hit value of $C_2$ is 2. Hence $C_2$ is retained and $C_1$ is dropped. By doing this process for all the clusters in the set we get, $\mathcal{Q} = \{(N_{33}, N_{34}), (N_{32}, N_{33}, N_{34}, N_{35}), (N_{31}, N_{32}, N_{33}, N_{34}, N_{35}, N_{36})\}$.

For each of the cluster remaining in $\mathcal{Q}$, a GNF-skeleton can be constructed. The first node $N_{ij}$ of each cluster has to be replaced by an appropriate terminal symbol which is $\text{First}(FS(N_{ij}))$. An unique symbol may be associated with the root of each skeleton and those nodes are added to the node set $ND_1$. In this case we use $T_1, T_2$ and $T_3$ to represent the clusters in $\mathcal{Q}$. Now, the node set $ND_1 = \{N_{11}, N_{21}, N_{31}, T_1, T_2, T_3\}$ and node set $ND_2 = \{N_{12}, N_{22}, N_{23}, N_{24}, N_{32}, N_{33}, N_{34}, N_{35}, N_{36}\}$. By applying the relation $\sim$ on the node set $ND_1$ we get the partition $\pi_1$ having the single block i.e $\pi_1 = \{\{N_{11}, N_{21}, N_{31}, T_1, T_2, T_3\}\}$. Application of $\simeq$ results in a partition $\pi_2 = \{\{N_{12}, N_{23}, N_{24}, N_{34}, N_{35}, N_{36}\}, \{N_{22}, N_{32}, N_{33}\}\}$.

Using these partitions and GNF-skeletons, we can generate following productions. A unique symbol $S_1$ is given to the single block $\pi_1$ and symbols $S_2$ and $S_3$ are used to denote the blocks of $\pi_2$ though $S_2$ is redundant and not used. The productions of TDCFG are as follows. $S \to S_1, S_1 \to aS_3 | aS_1S_3, S_3 \to b$

5 Analysis of Algorithm Infer-Grammar

The properties of the proposed pruning model and Infer-Grammar algorithm are analysed as follows. Let $S'$ be given sample set and $G_c = (N_c, \Sigma, P_c, S_c)$ be canonical CFG in GNF for $S'$. Let $ND(S')$ be the set of nodes in the GNF-skeletons constructed for the samples in $S'$. The term contextual entity is used to denote a cluster of nodes from $ND(S')$. A contextual entity is added to the initial prune set $\mathbb{E}$ if it appears in more than one GNF-skeletons. Following observations are made w.r.t pruning model:
**Observation 1:** Let $\mathbb{R} \neq \emptyset$ be the initial prune set constructed for the $G_e$. Let $C \in \mathbb{R}$ be a cluster of nodes from $ND(S^+)$, $\exists N_{ij} \in (S^+ - C)$ and $\exists N_{kl} \in C$ such that $N_{ij} \simeq N_{kl}$.

**Observation 2:** Clusters in $\mathbb{Q}$ represent optimal collection of contextual entities in $S^+$.

Hence it is obvious that if the set $\mathbb{Q} = \emptyset$, then there exists no contextual entity which is common to more than one GNF-skeleton. In such a case, the Infer-Grammar algorithm outputs a canonical TDCFG, that generates exactly same set of strings of $S^+$.

### 5.1 Convergence Issues

It is to be proved that the proposed algorithm converges to a target grammar if enough information is presented. The information presented to the procedure Infer-Grammar is a set of strings. We need to prove that the procedure identifies any TDCFG grammar in the limit. The proof is based on the existence of a characteristic sample set for any TDCFG grammar in GNF. Following lemma shows the construction of such sample set.

**Lemma 1.** Let $G$ be a TDCFG. There exists a finite set of input samples according to $G$ such that if it is given as input to the proposed inference procedure, then it outputs a grammar $G'$ which is isomorphic to $G$.

**Proof.** Let TDCFG $G = (N, \Sigma, P, S)$. For every auxiliary symbol $A \in N$, its structural information and associated sample string can be calculated as follows:

- Consider all derivation path in which $A$ appears. That is $S \Rightarrow^*_G \alpha A \beta \Rightarrow^*_G w$ with $w \in \Sigma^*$.
- Construct the derivation tree according to previous derivation.
- Compute $SSNF(A)$ at each occurrence of $A$ in the skeleton.
- Compute $FS(A)$ at each occurrence of $A$ in the skeleton.

A set of all such $w$ considered above form a characteristic sample set. Since $G$ is TDCFG, so it is backward deterministic, then the set of skeletons induced by the derivation trees have the property that if the nodes $p$ and $q$ are labeled with different auxiliary symbols $A$ and $B$ respectively then, $SSNF(A) = SSNF(B)$ and $FS(A) = FS(B)$.

When this characteristic sample set is given to the Algorithm 1 as input, the algorithm constructs a modified prune list $\mathbb{Q}$. According to the Observation 2, $\mathbb{Q}$ represent an optimal collection of contextual entities. As per the basic strategy used, a GNF-skeleton is constructed for each of the cluster in $\mathbb{Q}$ and the unique root nodes of such GNF-skeletons are added to $ND_1$. Algorithm 1 constructs a partition on the nodes of $ND_1$ and $ND_2$ where $ND(S^+) = ND_1 \cup ND_2$. Each block of the partition is represented by a set of SSNF’s and set of frontier strings. If the derivation

$$S \Rightarrow_G \alpha_1 A_1 \beta_1 \Rightarrow_G \cdots \Rightarrow_G \alpha_n A_n \beta_n \Rightarrow_G \alpha A \beta \Rightarrow^*_G \alpha w \beta$$
and
\[ S \Rightarrow_G \alpha_1 A_1 \beta_1 \Rightarrow_G \cdots \Rightarrow_G A_n \beta_n \Rightarrow_G \alpha B \beta \Rightarrow_G^* \alpha w' \beta \]
are derivations with \( A \neq B \), then \( \text{Ter}(w) \neq \text{Ter}(w') \) and \( A \) and \( B \) are not related under \( \sim \) or \( \simeq \). This is due to the terminal completeness and terminal dissimilarity properties of TDCF.

On the other hand, let us suppose that, in the skeleton set for the input samples, two different internal nodes \( p \) and \( q \), are labeled by the same auxiliary symbol. Then one of the conditions \( p \sim q \) or \( \exists : r \sim p \land r \sim q \Rightarrow p \sim q \) or \( p \simeq q \) is fulfilled.

From the previous conditions we can affirm that Algorithm 1 correctly relates those internal nodes induced by one auxiliary symbol. We can conclude that if the characteristic sample we have defined is given as input data then every auxiliary symbol of the grammar is represented and the Algorithm 1 correctly distinguishes them. \( \square \)

5.2 Complexity Issues

The time complexity of Algorithm 1 is presented as follows. Let \( S^t = \{w_1, w_2, \ldots, w_n\} \) be the input sample set for the algorithm. Let \( k \) be the size of the input i.e., sum of lengths of all input strings belonging to \( S^t \). It is observed that the maximum possible size of the set \( Q \) is \( k/2 \). The computing time of modules of pruning model viz. Initialise-Prune-sets, Induce-pumping and Select-Clusters are \( O(k^3) \), \( O(k^2) \) and \( O(k^2) \) respectively. All the remaining steps of the algorithm Infer-Grammar, requires \( \leq O(k^3) \) computing time.

Hence we can conclude that total computing time required for the Algorithm 1 is \( O(k^3) \), a polynomial function of \( k \).

5.3 Empirical Results

The proposed Infer-Grammar algorithm has been implemented successfully using a code written in C programming language under RedHat Linux 7.3 system. The main data structure used is doubly linked list to store the information regrading the nodes of GNF-skeletons. Some of the experimental results obtained are tabulated and shown in Table 2.

6 Conclusion

A novel way of identifying a subclass of context free grammars from a set of positive sample is presented. The inference method proposed for the class of TDCFG infers correctly for many languages in the class in polynomial time and data.

The basic strategy can conveniently be modified to handle other remaining cases mentioned in the basic strategy. As a future work, a more detailed study of the characteristics of TDCF can be conducted and exploration of the possibilities of using this inferencing mechanism to possible applications like structured document analysis may be explored.
Table 2. Computation of TDCFG for different sample sets

<table>
<thead>
<tr>
<th>Sample Set</th>
<th>Inferred Grammar</th>
<th>Target Language</th>
</tr>
</thead>
</table>
| $\{ab^2cb^2a, ab^2cb^3a\}$ | $S \rightarrow S_1$  
$S_1 \rightarrow aS_2S_3$, $S_2 \rightarrow bS_4S_5|bS_2S_5$  
$S_3 \rightarrow a$, $S_4 \rightarrow c, S_5 \rightarrow b$ | $\{ab^i cb^j a|i \geq 1\}$ |
| $\{abba, abababa\}$     | $S \rightarrow S_1$  
$S_1 \rightarrow aS_2S_3|aS_1S_3|bS_1S_4$  
$S_2 \rightarrow bS_4$, $S_3 \rightarrow a, S_4 \rightarrow b$ | $\{w(abba)w'|w \in (a+b)^*\}$ |
| $\{aababa, aab^2cb^2a$,  
$\ aab^3cb^3a\}$         | $S \rightarrow S_1S_1 \rightarrow aS_2$  
$S_2 \rightarrow bS_3S_4|aS_2S_3|bS_2S_4$  
$S_3 \rightarrow a$, $S_4 \rightarrow b$ | $\{aw(bab)w'|w \in (a+b)^*\}$ |

References

A Hybrid Language Model based on Stochastic Context-free Grammars*

Diego Linares¹, José-Miguel Benedí², and Joan-Andreu Sánchez²

¹ Pontificia Universidad Javeriana - Cali
Calle 18 No. 118-250 Av. Cañasgordas
Cali (Colombia)
dlinares@dsic.upv.es

² DSIC - Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia (Spain)
{jbenedi, jandreu}@dsic.upv.es

Abstract. This paper explores the use of initial Stochastic Context-Free Grammars (SCFG) obtained from a treebank corpus for the learning of SCFG by means of estimation algorithms. A hybrid language model is defined as a combination of a word-based n-gram, which is used to capture the local relations between words, and a category-based SCFG with a word distribution into categories, which is defined to represent the long-term relations between these categories. Experiments on the UPenn Treebank corpus are reported. These experiments have been carried out in terms of the test set perplexity and the word error rate in a speech recognition experiment.

1 Introduction

Language modeling is an important aspect to consider in the development of speech and text recognition systems. The n-gram models are the most extensively used for a wide range of domains [1]. The n-grams are simple and robust models and adequately capture the local restrictions between words. Also, the estimation of the parameters and the integration of the model in recognition systems are well-known. However, the n-gram models cannot characterize the long-term constraints of the sentences of the tasks. Stochastic Context-Free Grammars (SCFGs) efficiently model long-term relations and have been successfully used on limited-domain tasks of low perplexity. Nevertheless, general-purpose SCFGs work poorly on large vocabulary tasks. The main obstacles to using these models in complex real tasks are the difficulties of learning and integrating SCFGs.

With regard to the learning of SCFGs, two aspects must be considered: first, the learning of the structural component, that is, the rules of the grammar, and second, the estimation of the stochastic component, that is, the probabilities of the rules. Although interesting Grammatical Inference techniques have been proposed elsewhere for learning the rules of the grammar, computational restrictions limit their use in complex real

* This work has been partially supported by the Spanish CICYT under contract (TIC2002/04103-C03-03)
tasks. Taking into account the existence of robust techniques for the automatic estimation of the probabilities of the SCFGs from samples [2–4], other possible approaches for the learning of SCFGs by means of a probabilistic estimation process have been explored [3,5].

All of these estimation algorithms are based on gradient descedent techniques and it is well-known that their behavior depends on the appropriate choice of the initial grammar. When the SCFG is in Chomsky Normal Form (CNF), the usual method for obtaining this initial grammar is a heuristic initialization based on an exhaustive ergodic model [2,5]. This solution is easy to implement but it does not use any structural information of the sample. When a treebank corpus is available, it is possible to directly obtain an initial SCFG in General Format (GF) from the syntactic structures which are present in the treebank corpus [6].

We conjecture that the structural component of SCFGs is very important for the learning of stochastic models. So, in this work, we explore this possibility working with SCFGs in GF by using the Earley algorithm [7] and comparing with SCFGs in CNF.

With regard to the problem of SCFG integration in a recognition system, several proposals have attempted to solve these problems by combining a word n-gram model and a structural model in order to consider the syntactic structure of the language [8,9]. In the same way, we previously proposed a general hybrid language model [10]. This is defined as a linear combination of a word n-gram model, which is used to capture the local relation between words, and a stochastic grammatical model, which is used to represent the global relation between syntactic structures. In order to capture the long-term relations between syntactic structures and to solve the main problems derived from large-vocabulary complex tasks, we proposed a stochastic grammatical model defined by a category-based SCFG together with a probabilistic model of word distribution into the categories.

In the second section of this paper, we present the hybrid language model based on SCFGs, together with the results of their evaluation process. This evaluation process has been done using the UPenn Treebank corpus. Firstly, we compare the obtained SCFGs by an initialization process based on ergodic and treebank initial grammar. Then, we also compare the final hybrid language model with other stochastic language models in terms of the test set perplexity and the word error rate.

2 The Language Model

An important problem related to language modeling is the computation of the expression $\Pr(w_k|w_1 \ldots w_{k-1})$ [11]. In order to calculate this probability, we proposed in [10] a general hybrid language model defined as a linear combination of a word n-gram model, which is used to capture the local relation between words, and a word stochastic grammatical model $M_e$ which is used to represent the global relation between syntactic structures and which allows us to generalize the word n-gram model:

$$\Pr(w_k|w_1 \ldots w_{k-1}) = \alpha \Pr(w_k|w_{k-n+1} \ldots w_{k-1}) + (1-\alpha) \Pr(w_k|w_1 \ldots w_{k-1}, M_e),$$

(1)
where \( 0 \leq \alpha \leq 1 \) is a weight factor which depends on the task. Similar proposals have been presented by other authors [8, 9] along the same line.

The first term of expression (1) is the word probability of \( w_k \) given by the word n-gram model. The parameters of this model can be easily estimated, and the expression \( \Pr(w_k \mid w_{k-n+1} \ldots w_{k-1}) \) can be efficiently computed.

In order to capture long-term relations between syntactic structures and to solve the main problems derived from large vocabulary complex tasks, we proposed a stochastic grammatical model \( M_k \) defined as a combination of two different stochastic models: a category-based SCFG \( (G_c) \) and a stochastic model of word distribution into categories \( (C_w) \). Thus, the second term of the expression (1) can be written in this way: \( \Pr(w_k \mid w_1 \ldots w_{k-1}, G_c, C_w) \).

There are two important questions to consider: the definition and learning of \( G_c \) and \( C_w \), and the computation of the probability \( \Pr(w_k \mid w_1 \ldots w_{k-1}, G_c, C_w) \).

2.1 Learning of the Models

Now, we explain the estimation of the models. First, we introduce some notation. Then, we present the framework in which the estimation process is carried out. Finally, we describe how the parameters of \( G_c \) and \( C_w \) are estimated.

A Context-Free Grammar (CFG) \( G \) is a four-tuple \( (N, \Sigma, P, S) \), where \( N \) is a finite set of non-terminal symbols, \( \Sigma \) is a finite set of terminal symbols, \( P \) is a finite set of rules, and \( S \) is the initial symbol. A CFG is in Chomsky Normal Form (CNF) if the rules are of the form \( A \rightarrow BC \) or \( A \rightarrow a \) (\( A, B, C \in N \) and \( a \in \Sigma \)). We say that the CFG is in General Format (GF) if no restriction is imposed on the format of the rules.

A Stochastic Context-Free Grammar (SCFG) \( G_{sc} \) is defined as a pair \( (G, q) \), where \( G \) is a CFG and \( q : P \rightarrow [0, 1] \) is a probability function of rule application such that \( \forall A \in N: \sum_{\alpha \in (N \cup \Sigma)^+} q(A \rightarrow \alpha) = 1 \). We define the probability of the derivation \( d \) of the string \( x \), \( \Pr(x, d_e \mid G_s) \) as the product of the probability application function of all the rules used in the derivation \( d_e \). We define the probability of the string \( x \) as: \( \Pr(x \mid G_s) = \sum_{\forall d_e} \Pr(x, d_e \mid G_s) \).

Estimation Framework. In order to estimate the probabilities of a SCFG, it is necessary to define both a framework to carry out the optimization process and an objective function to be optimized. In this work, we have considered the framework of Growth Transformations [12] in order to optimize the objective function.

In reference to the function to be optimized, we will consider the likelihood of a sample which is defined as: \( \Pr(\hat{\Omega} \mid G_s) = \prod_{x \in \hat{\Omega}} \Pr(x \mid G_s) \), where \( \hat{\Omega} \) is a multiset of strings.

Given an initial SCFG \( G_s \) and a finite training sample \( \hat{\Omega} \), the iterative application of the following function can be used in order to modify the probabilities (\( \forall (A \rightarrow \alpha) \in P \)):

\[
q'(A \rightarrow \alpha) = \frac{\sum_{x \in \hat{\Omega}} \frac{1}{\Pr(x \mid G_s)} \sum_{\forall d_e} N(A \rightarrow \alpha, d_e) \Pr(x, d_e \mid G_s)}{\sum_{x \in \hat{\Omega}} \frac{1}{\Pr(x \mid G_s)} \sum_{\forall d_e} N(A, d_e) \Pr(x, d_e \mid G_s)} .
\]
The expression \( N(A \rightarrow \alpha, d_x) \) represents the number of times that the rule \( A \rightarrow \alpha \) has been used in the derivation \( d_x \), and \( N(A, d_x) \) is the number of times that the non-terminal \( A \) has been derived in \( d_x \). This transformation optimizes the function \( \Pr(\Omega \mid G_n) \).

Algorithms which are based on transformation (2) are gradient descendent algorithms and, therefore, the choice of the initial grammar is a fundamental aspect since it affects both the maximum achieved and the convergence process. Different methods have been proposed elsewhere in order to obtain the initial grammar.

**Estimation of SCFG in CNF.** When the grammar is in CNF, transformation (2) can be adequately formulated and it becomes the well-known Inside-Outside algorithm [2]. If a bracketed corpus is available, this algorithm can be adequately modified in order to take advantage of this information [3]. The initial grammar for this estimation algorithm is typically constructed in a heuristic fashion from a set of terminals and a set of non-terminals. The most common way is to construct a model with the maximum number of rules which can be formed with a given number of non-terminals and a given number of terminals [2]. Then, initial probabilities which are randomly generated are attached to the rules. This heuristic has been successfully used for real tasks [5]. However, the number of non-terminals is a critical point which leaves room for improvements [13].

**Estimation of SCFG in GF.** When the grammar is in GF, transformation (2) can be adequately computed by using an Earley-based algorithm [4]. This algorithm is based on the definition of the inner probability and outer probability. First, we describe the computation of these values and then we describe the SCFG estimation based on the Earley algorithm.

The Earley algorithm constructs a set of lists \( L_0, \ldots, L_{|\xi|} \), where \( L_i \) keeps track of all possible derivations that are consistent with the input string until \( x_i \). An item is an element of a list and has the form \( j_i A \rightarrow \lambda \cdot \mu \), where \( j \) is the current position in the input and is thereby in the \( L_j \) list. The value \( k \) is the position in the input when the item was selected to expand \( A \). The dot indicates that \( \lambda \) accepts \( x_{k+1} \ldots x_j \) and that \( \mu \) is pending for expansion. This item records the previous history: \( S \Rightarrow x_1 x_2 \ldots x_k A \delta \Rightarrow x_1 x_2 \ldots x_k \lambda \mu \delta \Rightarrow x_1 x_2 \ldots x_k x_{k+1} \ldots x_j \mu \delta \).

The probabilistic version attaches two values called inner probability and outer probability to each item [4].

The inner probability is denoted as \( \gamma(j_i A \rightarrow \lambda \cdot \mu) \). This value represents the sum of probabilities of all partial derivations that begin with the item \( j_i A \rightarrow \lambda \cdot \mu \) and end with the item \( j_i A \rightarrow \lambda \cdot \mu \) generating the substring \( x_{i+1} \ldots x_j \). For each item, the inner probability can be calculated with the following recursive definition:

\[
\gamma(j_i A \rightarrow \lambda \cdot \mu) = q(A \rightarrow \mu), \quad 0 \leq j < n,
\]

\[
\gamma(j_i A \rightarrow \lambda \delta \cdot \mu) = \left\{ \begin{array}{ll}
\gamma(j_i A \rightarrow \lambda \cdot \delta \mu) & \text{if } \delta = x_j, \\
\sum_{k=i}^{j-1} \gamma(k_i A \rightarrow \lambda \cdot \delta \mu) \sum_{C} R_\delta(\delta, C) \gamma(k_i C \rightarrow \sigma \cdot) & \text{if } \delta \in N.
\end{array} \right.
\]
In this expression, \( R_U(A, B) = \Pr(A \Rightarrow_U B) \), which is computed from the probability unit production relation \( \Pr_U(A, B) = q(A \rightarrow B) \). \( \forall A, B \in N. R_U(A, B) = (I - P_U)^{-1} \) when the grammar is consistent [14].

This way, \( \Pr(x|G_S) = \gamma(n^{0}S \rightarrow S) \), where \( S \rightarrow S \) is a dummy rule which is not in \( P \). The expression \( q(S \rightarrow S) \) is always one and it is used for initialization. The time complexity of computing the inner probability is \( O(|P||x|^3) \), and its spatial complexity is \( O(|P||x|^2) \).

The outer probability is denoted as \( \beta(i, j^\delta A \rightarrow \lambda \cdot \mu) \). This value represents the sum of probabilities of all partial derivations that begin with the item \( _i^0S \rightarrow \cdot S \), generate the prefix \( x_1x_2 \ldots x_i \), pass through the item \( _i^iA \rightarrow \nu \mu \), for some \( \nu \), generate the suffix \( x_{j+1} \ldots x_n \) and end in the final item \( _n^0S \rightarrow S \). The outer probability is the complement of the inner probability and, therefore, the choice of the rule \( A \rightarrow \lambda \mu \) is not part of the outer probability.

For each item, the outer probability can be calculated with the following recursive definition:

\[
\beta(_i^0S \rightarrow \cdot S) = 1,
\]

\[
\beta(_i^jA \rightarrow \lambda \cdot \mu) = \begin{cases} 
\beta(_i^{j+1}A \rightarrow \lambda \delta \cdot \mu) & \text{if } \delta \in \Sigma, \\
\sum_{k=j+1}^{n} \beta(_i^kA \rightarrow \lambda \delta \cdot \mu) & \\
\sum_{k=j+1}^{n} \sum_{B \in N} \beta(_i^kB \rightarrow \sigma \cdot C \sigma') \Pr_U(C, A) & \text{if } \delta \mu = \epsilon, \\
0 \leq i < j \leq n.
\end{cases}
\]

The time complexity of the outer probability is \( O(|P||x|^3) \), and its spatial complexity is \( O(|P||x|^2) \).

In order to rewrite (2) in terms of the inner and outer probabilities, we need to note that these definitions associate several items to a rule. Here, we considered only the items with the dot at the beginning of the right side \( _i^iA \rightarrow \cdot \lambda \) to represents the rule \( (A \rightarrow \lambda) \).

Let \( A \rightarrow \lambda \) be a rule, and \( d_2 \) a set of derivations that uses this rule, we assume that the Earley algorithm selects this rule at the \( i+1 \) position to extend derivations from the position \( i \) of the input string.

The algorithm inserts the item \( _i^iA \rightarrow \cdot \lambda \), recording the information: \( S \Rightarrow x_1 \ldots x_i \lambda \eta \), \( \eta \in (N \cup \Sigma)^{+} \). Given that \( A \rightarrow \lambda \) is in \( d_2 \), then, \( S \Rightarrow x_1 \ldots x_i \lambda \eta x_{i+1} \ldots x_n \). And its probability is: \( \Pr(S \Rightarrow x_1 \ldots x_i \lambda \eta x_{i+1} \ldots x_n | A \rightarrow \lambda, G_S) \Pr(U | A \rightarrow \lambda) \).

Using the inner and outer probabilities this expression becomes: \( \beta(_i^iA \rightarrow \cdot \lambda) \gamma(_i^iA \rightarrow \cdot \lambda) \).

This expression adds up the probabilities of all derivations that have selected the rule \( A \rightarrow \lambda \) at the position \( i \). If we sum for all positions, we can rewrite the numerator of (2). And if we sum for all positions and for all rules with the same left non terminal, we can rewrite its denominator. This way, expression (2) can be written as:

\[
\mathcal{F}(A \rightarrow \lambda) = \frac{\sum_{x \in B} \mathcal{F}(x|G_S) \sum_{i=0}^{n-1} \beta(_i^iA \rightarrow \cdot \lambda) \gamma(_i^iA \rightarrow \cdot \lambda)}{\sum_{x \in B} \mathcal{F}(x|G_S) \sum_{i=0}^{n-1} \beta(_i^iA \rightarrow \cdot \lambda) \gamma(_i^iA \rightarrow \cdot \lambda)}. \tag{3}
\]
The time complexity of this transformation per iteration is $O(|\Omega||x||P|)$. However, due to the fact that to inner and outer probabilities are both $O(|P||x|^3)$, the overall time complexity per iteration is $O(|\Omega||x|^3||P|)$.

When a bracketed corpus is available, the inner and outer probabilities can be adequately modified by using a similar function to the one described in [3]. This function filters those derivations (or partial derivations) whose parsing are not compatible with the bracketing defined in the sample, and takes into account only those partial parses which are compatible with the bracketing defined in the strings.

These modifications can affect the time complexity of the algorithm per iteration, which in the worst case is $O(|\Omega||x|^3||P|)$, but in a full bracketed sample the time complexity is $O(|\Omega||x||P|)$.

In this algorithm, the initial grammar can be obtained from a treebank corpus. Each sentence in the corpus is explored and each syntactic structure is considered as a grammar rule. In addition, the frequency of appearance is adequately maintained and, at the end, these values are conveniently normalized. These initial grammars have been successfully used for real tasks [6], since they allow us to parse real test sets.

**Estimation of the Parameters of $C_w$.** We work with a tagged corpus, where each word of the sentence is labeled with part-of-speech tag (POSTag). From now on, these POSTags are referred to as word categories in $C_w$ and are the terminal symbols of the SCFG in $G_c$. The parameters of the word-category distribution, $C_w = \Pr(w|c)$ are computed in terms of the number of times that the word $w$ has been labeled with the POSTag $c$. It is important to note that a word $w$ can belong to different categories. In addition, it may happen that a word in a test set does not appear in the training set, and, therefore, its probability $\Pr(w|c)$ is not defined. We solve this problem by adding the term $\Pr(\text{UNK}|c)$ for all categories, where $\Pr(\text{UNK}|c)$ is the probability for unseen words of the test set.

### 2.2 Integration of the Models

The computation of $\Pr(w_k | w_1 \ldots w_{k-1}, G_c, C_w)$ can be expressed as:

$$\Pr(w_k | w_1 \ldots w_{k-1}, G_c, C_w) = \frac{\Pr(w_1 \ldots w_k | G_c, C_w)}{\Pr(w_1 \ldots w_{k-1} | G_c, C_w)}.$$  

where

$$\Pr(w_1 \ldots w_k | G_c, C_w)$$

represents the probability of generating an initial substring given $G_c$ and $C_w$.

When the grammar is in CNF, a simple modification of the LRI algorithm can be used in order to compute (4) [10].

When the grammar is in GF we introduce an algorithm based on the forward probability defined in [4].
Probability of Generating an Initial Substring with a SCFG in \( G \). The expression (4) can be calculated by means of a simple adaptation of the forward algorithm [4]:

\[
\alpha(0, \$ \rightarrow \cdot S) = 1
\]

\[
\alpha(j, A \rightarrow \cdot) = \sum_{k} \alpha(j, B \rightarrow \lambda \cdot C \mu) R_L(C, A) p(A \rightarrow \sigma), \quad 0 \leq j < n,
\]

\[
\alpha(j, A \rightarrow \lambda \delta \cdot \mu) = \begin{cases} 
\alpha(j-1, A \rightarrow \lambda \cdot \delta \mu) Pr(w_j | \delta) & \text{if } \delta \in C_w, \\
\sum_{k} \alpha(k, A \rightarrow \lambda \cdot \delta \mu) \sum_{C} RU(\delta, C) \gamma(i, C \rightarrow \sigma) & \text{if } \delta \in N, \\
0 \leq k < j \leq n.
\end{cases}
\]

In this expression, \( R_L(A, B) = \text{Pr}(A \Rightarrow_L B) \), which is computed from the probabilistic left-corner relation \( P_L(A, B) = q(A \rightarrow B \lambda) \), \( \forall A, B \in N \). \( R_L(A, B) = (I - P_L)^{-1} \) when the grammar is consistent [14].

In this way \( Pr(w_1 \ldots w_k \ldots | G_C, C_w) = \sum_{k=0}^{k} \sum_{i} \alpha(i, A \rightarrow \lambda \cdot w_k \cdot \mu) \).

3 Experiments with the UPenn Treebank Corpus

In this section, we describe the experiments which were carried out in order to test the language model proposed in the previous section.

The corpus used in the experiments was the part of the Wall Street Journal (WSJ) which had been processed in the UPenn Treebank project [15]. This corpus consists of English texts collected from the Wall Street Journal from editions of the late eighties. It contains approximately one million words distributed in 25 directories. This corpus was automatically labelled, analyzed and manually checked as described in [15]. There are two kinds of labelling: a POS tagging labelling and a syntactic labelling. The size of the vocabulary is greater than 49,000 different words, the POS tag vocabulary is composed of 45 labels, and the syntactic vocabulary is composed of 14 labels. The corpus was divided into sentences according to the bracketing. For the experiments, the corpus was divided into three sets: training (directories 00-20, 42,075 sentences, 1,004,073 words), tuning (directories 21-22, 3,371 sentences, 80,156 words) and test (directories 23-24, 3,762 sentences, 89,537 words). The sentences labeled with POS tags were used to learn the category-based SCFGs, and the sentences labeled with both POS tags and with words were used to estimate the parameters of the hybrid language model.

First, we present perplexity results on the described task. These results are compared with the results obtained by other authors for the same task. Second, we present word error rate results on a speech recognition experiment and the results obtained are compared with the results obtained by other authors.

3.1 Perplexity Results

In order to compare our model with other hybrid models, we carried out the experiments taking into account the restrictions considered in other works [8, 9]. The restrictions

---

3 Release 2 of this data set can be obtained from the Linguistic Data Consortium with Catalogue number LDC94T4B (http://www.ldc.upenn.edu/ldc/noframe.html)
that we considered were the following: all words that had the POS tag CD (cardinal number [15]) were replaced by a special symbol which did not appear in the initial vocabulary; all capital letters were uncapitalized; the vocabulary was composed of the 10,000 most frequent words that appear in the training.

**Baseline Model.** We now describe the estimation of a 3-gram model to be used as both a baseline model and as a part of the hybrid language model. The parameters of a 3-gram model were estimated using the software tool described in [16]. Different smoothing techniques were tested, but we chose the one which provided a test set perplexity which was similar to the one presented in [8, 9]. Linear discounting was used as smoothing technique with the default parameters. The out-of-vocabulary words were used in the computation of the perplexity, and back-off from context cues was excluded. The tuning set perplexity with this model was 160.26 and the test set perplexity was 167.30.

**Hybrid Language Model.** First, the category-based SCFGs ($G_c$) of the hybrid model was obtained. First, we describe the estimation of a SCFG in GF, and then we describe the estimation of a SCFG in CNF.

Given that the UPenn Treebank corpus was used, a treebank grammar was obtained from the syntactic information. The corpus was adequately filtered in order to use only the POS tags and the syntactic tags defined in [15]. Probabilities were attached to the rules according to the frequency of each one in the training corpus. Then, this initial grammar was estimated using the bracketed version of the Earley (bE) algorithm. The perplexity of the POS tag part of the tuning set was computed and the results can be seen in Table 1.

The parameters of an initial SCFG in CNF were estimated using the bracketed version of the IO (bIO) algorithm. The initial grammar had the maximum number of rules which can be created with 35 non-terminals 45 terminal symbols, and the results can be seen in Table 1.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>bE</th>
<th>bIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuning set perplexity</td>
<td>10.53</td>
<td>10.24</td>
</tr>
</tbody>
</table>

We can observe that slightly better results were obtained with the SCFG estimated with the bIO algorithm. This may be due to the fact that did not generalize enough and there was not enough room for improvements [17].

Second, the parameters of the word-category distribution $C_w = \Pr(w|e)$ were computed from the POS tags and the words of the training corpus. The unseen events of the test corpus were considered as the same word **UNK**. We conjectured that the unknown words were not equally distributed among categories, and we assigned a probability
based on the classification of unknown words into categories in the tuning set. A small probability $\epsilon$ was assigned if no unseen event was associated to the category. The percentage of unknown words in the training set was 4.47% distributed in 31 categories, and the percentage of unknown words in the tuning set was 5.53% distributed in 23 categories.

Finally, once the parameters of the hybrid language model were estimated, we applied expression (1). In order to compute expression (4) we used:

- the modified version of the LRI algorithm [10] with the SCFG in CNF which was estimated as we previously described;
- the modified version of the forward algorithm previously described, with the SCFG in GF which was estimated as we previously described.

The tuning set was used in order to determine the best value of $\alpha$ for the hybrid model, and the results can be seen in Figure 1.

![Fig. 1. Tuning set perplexity depending on $\alpha$ for the hybrid language model with the estimated treebank grammar (HLM-bE) and with the estimated SCFG in CNF (HLM-bIO).](image)

The results obtained with the treebank grammar slight improved the results obtained by a grammar heuristically initialized with no structural information. It can be observed that a good structural initial model plays an important role in the estimation process. The minimum tuning set perplexity with the estimated treebank grammar was 133.25 which means a 16.85% of improvement over the 3-gram. Another important aspect to note is that the weight of the grammatical part was approximately 33%, which is less than the values obtained by other authors. However it is important to note that the weight of the grammatical part obtained by the SCFG in CNF was slightly better (35%).

Table 2 shows the test set perplexity obtained for the hybrid language model and the results obtained by other authors who define left-to-right hybrid language models of the same nature [8, 9]. It should be noted that the differences in the perplexity of the trigram model were due mainly to the different smoothing techniques. It can also be
Table 2. Test set perplexity with a 3-gram model and with the hybrid language model and percentage of improvement for different proposals. The first row (CJ00) corresponds to the model proposed by Chelba and Jelinek in [8], the second row (R01) corresponds to the model proposed by Roark in [9], the third row (HLM-bIO) corresponds to the model proposed by [13], and the fourth row (HLM-bE) corresponds to our proposed hybrid language model with the estimated treebank grammar.

<table>
<thead>
<tr>
<th>Model</th>
<th>Perplexity</th>
<th>α</th>
<th>% improv.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Trig.</td>
<td>Interp.</td>
<td></td>
</tr>
<tr>
<td>CJ00</td>
<td>167.14</td>
<td>148.90</td>
<td>0.4</td>
</tr>
<tr>
<td>R01</td>
<td>167.02</td>
<td>137.26</td>
<td>0.4</td>
</tr>
<tr>
<td>HLM-bIO</td>
<td>167.30</td>
<td>142.29</td>
<td>0.65</td>
</tr>
<tr>
<td>HLM-bE</td>
<td>167.30</td>
<td>140.31</td>
<td>0.67</td>
</tr>
</tbody>
</table>

observed that the results obtained by our models are very good, even better if you consider that both the models and their learning methods are simple and well-consolidated. The weight of the structural model of our proposal was less than the other models. This may be due to the fact that the our SCFG were not structurally as rich as those models proposed by other authors.

3.2 Word Error Rate Results

Now, we describe very preliminary speech recognition experiments which were carried out to evaluate the hybrid language model. Given that our hybrid language model is not integrated in a speech recognition system, we reproduced the experiments described in [8, 9, 13] in order to compare our results with those reported in those works.

The experiment consisted of rescoring a list of n best hypotheses provided by the speech recognizer described in [8]. A better language model was expected to improve the results provided by a less powerful language model. In order to avoid the influence of the language model of the speech recognizer it is important to use a large value of n; however, for these experiments, this value was lower.

The experiments were carried out with the DARPA ’93 HUB1 test setup. This test consists of 213 utterances read from the Wall Street Journal with a total of 3,446 words. The corpus comes with a baseline trigram model using a 20,000-word open vocabulary and is trained on approximately 40 million words.

The 50 best hypotheses from each lattice were computed using Ciprian Chelba’s A* decoder, along with the acoustic and trigram scores. Unfortunately, in many cases, 50 distinct string hypotheses were not provided by the decoder [9]. An average of 22.9 hypotheses per utterance were rescored.

The hybrid language model was used in order to compute the probability of each word in the list of hypotheses. The probability obtained with the hybrid language model was combined with the acoustic score and the results can be seen in Table 3 together with the results obtained for different language models.

It can be observed that our language model with the estimated treebank grammar slightly improved the results obtained by the baseline model, in accordance with the
Table 3. Word error rate (WER) results for several models, with different training and vocabulary sizes and the best language model weight. The first row (LT) corresponds to the lattice trigram provided with the HUB1 test, the second row (CJ00) corresponds to the model proposed by [8], the third row (R01) corresponds to the model proposed by [9], the fourth row (BT) corresponds to the baseline trigram, the fifth row (No LM) corresponds to the results without the language model, the sixth row (HLM-bIO) corresponds to the model proposed by [13], and the seventh row (HLM-bE) corresponds to our language model with the estimated treebank grammar.

<table>
<thead>
<tr>
<th>Model</th>
<th>Training Voc.</th>
<th>LM Size</th>
<th>Size</th>
<th>Weight</th>
<th>WER</th>
</tr>
</thead>
<tbody>
<tr>
<td>LT</td>
<td>40M</td>
<td>20K</td>
<td>16</td>
<td>13.7</td>
<td></td>
</tr>
<tr>
<td>CJ00</td>
<td>20M</td>
<td>20K</td>
<td>16</td>
<td>13.0</td>
<td></td>
</tr>
<tr>
<td>R01</td>
<td>1M</td>
<td>10K</td>
<td>15</td>
<td>15.1</td>
<td></td>
</tr>
<tr>
<td>BT</td>
<td>1M</td>
<td>10K</td>
<td>5</td>
<td>16.6</td>
<td></td>
</tr>
<tr>
<td>No LM</td>
<td>0</td>
<td>10K</td>
<td>0</td>
<td>16.8</td>
<td></td>
</tr>
<tr>
<td>HLM-bIO</td>
<td>1M</td>
<td>10K</td>
<td>6</td>
<td>16.0</td>
<td></td>
</tr>
<tr>
<td>HLM-bE</td>
<td>1M</td>
<td>10K</td>
<td>4</td>
<td>16.2</td>
<td></td>
</tr>
</tbody>
</table>

results obtained by other authors. However it is important to note that the estimated SCFG in CNF obtained better results, even if we take into account that the test set perplexity was slightly worst with this model.

An important aspect to be noted is that although the improvement in perplexity is important (the same order of magnitude of other authors [9]), this improvement is not reflected in this error rate experiment. This may be due to the fact that our model is not structurally rich enough, and that more training samples should be used.

4 Conclusions

We have described two combination of a hybrid language, one with SCFG in GF and the other with SCFG in CNF, and results of its evaluation have also been provided. The test set perplexity results were as good as the ones obtained by other authors, even better if you consider that the models are very simple and their learning methods are well-known.

The word error rate results were slightly worse than the ones obtained by other authors. However, we remark that these results tended to improve without including any additional linguistic information.

For future work, we propose extending the experimentation by increasing the size of the training corpus in order to avoid overlearning in accordance with the work of other authors.

5 Acknowledgements

The authors would like to thank Brian Roark for providing them with the $n$-best lists for the HUB1 test set.
References

Incremental Learning of Context Free Grammars
by Extended Inductive CYK Algorithm

Katsuhiko Nakamura

College of Science and Engineering,
Tokyo Denki University, Hatoyama-machi, Saitama-ken,
350-0394 Japan. nakamura@k.dendai.ac.jp

Abstract. This paper describes recent improvements in Synapse system [5, 6] for inductive inference of context free grammars from sample strings. For effective inference of grammars, Synapse employs incremental learning based on the rule generation mechanism called inductive CYK algorithm, which generates the minimum production rules required for parsing positive samples. In the improved version, the form of production rules is extended to include not only $A \rightarrow \beta \gamma$ but also $A \rightarrow \beta$, called extended Chomsky normal form, where each of $\beta$ and $\gamma$ is either terminal or nonterminal symbol. By this extension and other improvements, Synapse can synthesize both ambiguous grammars and unambiguous grammars with less computation time compared to the previous system.

Keywords: incremental learning, inductive CYK algorithm, context free language, unambiguous grammar, iterative deepening

1 Introduction

In this paper, we explore the grammatical inference of context free grammars (CFGs) that is implemented in Synapse (Synthesis by Analyzing Positive String Examples) system. The approach is based on inductive CYK algorithm, incremental learning, and search for rule sets. Since basic methodologies in Synapse have been presented in [5, 6], we focus on recent improvements in Synapse in this paper, especially the extension in the form of the production rules, new performance results, and some heuristics.

The most important problem in machine learning of CFG is the degree of computational complexity required for generating rule sets of the grammar. Among previous theoretical works on grammatical inference, several papers implicitly show that CFGs, or more restricted grammars, are not learnable in polynomial time [1, 8]. For solving this fundamental problem, Synapse employs incremental learning based on the inductive CYK algorithm.

The inductive CYK algorithm is used for generating production rules for each positive sample string. The process of the algorithm succeeds, if a positive sample is derived from the set of rules as in usual CYK (Cocke-Younger-Kasami) algorithm [3]. If the system fails to parse the string, it generates the minimum production rules and adds them to the set of rules from which the sample string
is derived. An important feature of this method is that the system only generates the rules that are applied to in the bottom-up parsing by CYK algorithm.

Synapse uses the incremental learning for two purposes. First, for learning a grammar from its sample strings, the positive samples are given to the rule generation by inductive CYK algorithm in order, until all the positive samples are derived from the set of rules. This method is different from the usual incremental learning as in [7] in that the system checks all the negative samples each time it finds a set of rules for a positive sample. This process continues until the system finds a set of rules from which all the positive samples, but none of the negative samples, are derived. The second use of incremental learning is to synthesize a grammar by adding rules to the rules of previously learned grammars of similar languages.

The system inputs a set of samples and an initial set of rules and searches for the set of rules from which all the positive samples, but no negative samples, are derived. To obtain the minimum sets of rules, the system employs iterative deepening. The sets of rules are searched within a limited number of rules. When the search fails, the system iterates the search with larger limits. The iterative deepening is widely used for finding the optimum solution by the search, for example, in inductive logic programming.

The approaches for grammatical induction as well as machine learning of rule sets in general are classified into two categories, covering-based and generalization-based approaches, which we might call top-down and bottom-up approaches, respectively. The term “covering” is derived from that the learning system gradually covers all the positive samples, starting with empty set of rules [2]. It iteratively synthesizes rules until the rule set derives all the positive sample strings but no negative sample strings. The covering approach is employed by Sakakibara et. al. [9, 10] as well as Synapse. Sakakibara's method is based on both genetic algorithm and CYK algorithm. Their use of CYK algorithm is different from our approach in that possible tables of symbol sets are generated and tested in every generation of the GA process.

Many other grammatical inference systems for CFGs employ generalization-based approach, which is based on generation of rules by analyzing the samples and generalization and abstraction of the generated rules. GRIDS system by Langley and Stromsten [4] uses two operators to create new nonterminal symbols and to merge rules for inferring grammars from natural language sentences. The objective of Emile system [11] is to generate rules by analyzing a large volume of natural language texts. The other differences between these systems and Synapse are that natural languages are the target language and that these are not intended to synthesize simple grammars for general CFGs.

2 CFGs and Chomsky Normal Form

A context free grammar (CFG) is a system $G = (N, T, P, S)$, where $N, T$ and $P$ are finite sets of nonterminal symbols, terminal symbols and production rules (or simply rules), respectively, and $S \in N$ is the starting symbol. The production
rules (or simply rules) are of the form \( A \rightarrow u \) with \( A \in N, u \in (N \cup T)^+ \). Throughout this paper, we represent nonterminal symbols by uppercase characters, and terminal symbol by lower case characters. For any string \( v \) and \( w \in (N \cup T)^+ \), we write \( v \Rightarrow_G w \), if \( w \) can be derived from \( v \) by replacing a nonterminal symbol \( A \) in \( v \) by \( u \) by applying a rule \( A \rightarrow u \) in \( G \). Relation \( \Rightarrow_G \) is the reflective transitive closure of \( \Rightarrow_G \) (the symbol \( G \) may be omitted). The language of \( G \) is the set \( L(G) = \{ w \in T^* | S \Rightarrow_G w \} \). A CFG is ambiguous, if there is a string \( w \) such that there are two or more left derivations\(^1\) from \( S \) for \( w \).

The production rules of Chomsky normal form (CNF) are those of the form either \( A \rightarrow BC \) or \( A \rightarrow a \), \( A, B, C \in N \) and \( a \in T \). The improved version of Synapse employs the normal form,

\[
A \rightarrow \beta \quad \text{and} \quad A \rightarrow \beta \gamma \quad (A \in N, \beta, \gamma \in N \cup T)
\]

called the extended Chomsky normal form (extended CNF). Note that CNF is the special case of the extended CNF. An important feature of the extended CNF is that CYK algorithm can be extended to deal with the rules of this form as shown later. Another feature is that grammars of this form can be simpler than that of Chomsky normal form. This feature is significant for learning rules, since the computation time depends heavily on the number of rules.

Our previous works \([5, 6]\) are concerned with learning grammars of the form \( A \rightarrow \beta \gamma \), called revised CNF. By this form, it is generally possible to simplify not only the sets of rules but also grammatical inference by omitting the rules of the form \( A \rightarrow a \), when the number of terminal symbols is not large. Although no one-letter string can be derived from the revised CNF grammar, the languages defined by this grammar includes all the context free languages of strings with the lengths of two or more.

The following “coding technique” is a method of making up for the problem that this normal form does not have the rules of the form \( A \rightarrow a \).

- Doubling symbols in the strings, so that a string \( a_1a_2\cdots a_n \) is represented by \( a_1a_2a_1a_2\cdots a_na_n \).
- Using rules of the form \( A \rightarrow aa \) instead of \( A \rightarrow a \).

3 Using Synapse

We have two versions of Synapse. One is written in C++ and the other in Prolog. The time data in this section is based on the faster C++ version running on AMD Athlon processor with 1 GHz clock. In this section, as an introduction to Synapse, we outline how to use Synapse.

1. Input a file of positive and negative samples. For efficient synthesis, the samples are ordered by their length.
2. Input a file of initial rules (optional). The initial rules are either those of other grammar or simple supplement rules.

\(^1\) In the left derivation, rules are always applied to the leftmost nonterminal symbols.
3. Select the form of production rules to be generated from the followings, where each of $\beta$ and $\gamma$ is either a terminal or nonterminal symbol.

- Chomsky normal form (CNF): $A \rightarrow BC$.
- Revised CNF: $A \rightarrow \beta \gamma$.
- Extended CNF: $A \rightarrow \beta \gamma$ or $A \rightarrow B$.
- Regular: $A \rightarrow aB$.

Since Synapse currently does not generate rules of the form $A \rightarrow a$ in the case of extended CNF, these rules need to be given as initial rules.

4. Choose which of an ambiguous grammar or an unambiguous grammar Synapse synthesizes.

5. Set the following optional parameters. The functions of these parameters are explained later.

- The maximum limit of the number of rules ($K_{\text{max}}$). The default value is infinity. This value determines the range of search by the iterative deepening.
- The limit $R_{\text{max}}$ of the number of rules to be generated for one positive sample. The default value is equal to $K_{\text{max}}$.
- The limit of the number of nonterminal symbols. This parameter is usually not important, since generation of a new nonterminal symbol is restricted by the form of generated rules (see Section 4).

6. Start synthesis. Every time the system finds a new grammar, it outputs the rule set and pauses. When the user responds by clicking a redo button, the system restarts and tries to find the next solution.

Example 1: The set of strings containing the same number of $a$’s and $b$’s

The followings are the samples for the language $\{w \in \{a, b\}^+ | \#_a(w) = \#_b(w), |w| \geq 2\}$, where $\#_x(w)$ is the number of $x$’s in $w$.

Positive: $ab, ba, baab, aabb, abba, bbba, aaabbb, abaabb, baabab$.

Negative: $aa, bb, aaa, aba, baa, bba, aaaa, aaba, baab, bbaa, bbba, bbbb$.

In this case, no initial rule is necessary. We select revised CNF and ambiguous grammars. Synapse outputs the following grammar with seven rules as the first solution in 0.43 second after generating 2359 rules in the search.

$$S \rightarrow ab \mid ba \mid bC \mid Cb \mid SS, \quad C \rightarrow aS \mid Sa$$

Note that the sample set is not the minimum: The same grammar can be obtained from only the first seven positive samples above and the first four negative samples. In most cases, the result grammars and the computation time does not depend significantly on the number of samples, especially that of positive samples, provided that the input contains a certain set of sufficient samples.

If we chose an unambiguous grammar for this language, Synapse shows that there is no unambiguous grammar having less than 14 rules. In this case, the search takes several hours and needs approximately 100 samples.

The author conjectures that this language is inherently ambiguous.
Example 2: The set of strings containing more $b$'s than $a$'s For the language \( \{ w \in \{ a, b \}^+ \mid \#_a(w) < \#_b(w), \ |w| \geq 2 \} \), Synapse synthesizes the following nine rules in 31 seconds after generating \( 2.4 \times 10^5 \) rules.

\[
S \rightarrow bb \mid SC \mid Sb \mid bC \mid Cb, \ C \rightarrow ab \mid ba \mid Sa \mid aS
\]

Synapse can also incrementally learn a grammar of this language based on the grammar of the similar language in Example 1. By giving the grammar as the initial set of rules after replacing every occurrences of the starting symbol \( S \) by \( S_1 \), Synapse generated the following additional rules in 0.4 second after generating 2549 rules.

\[
S \rightarrow Sb \mid SS_1 \mid bb \mid bS_1 \mid S_1 b.
\]

Note that the number of the total rules \( 7 + 5 = 12 \) is greater than 9 of the grammar above, but the total computation time is much less than the time for the directly obtained grammar.

4 Description of the System

This section describes the improved version of the system. The main differences from the previous version [6] are the form of production rules and the introduction of the parameter \( R_{\text{max}} \).

4.1 Top-Level Procedure in Synapse

Figure 1 shows the top-level procedure in Synapse. It has inputs of ordered sets \( S_P \) and \( S_N \) of positive and negative sample strings, respectively, and an initial set \( P_0 \) of rules for incremental learning of the grammars. The procedure calls inductive CYK algorithm, which is a nondeterministic procedure containing choice points. For the sets \( S_P \) and \( S_N \) with \( S_P \cap S_N = \emptyset \), and for the set \( P_0 \) of rules, the top-level procedure searches for the set \( P \) of rules with \( P_0 \subseteq P \) and the set \( N \) of nonterminal symbols such that \( S_P \subseteq L(G) \) and \( S_N \cap L(G) = \emptyset \) for a CFG \( G = (N, T, P, S) \).

The system controls the search by iterative deepening on the number of rules to be generated. First, the number of the rules in the initial set of rules is assigned to the limit \( K \) of the number of initial rules. When the system fails to generate enough rules to parse the samples within this limit, it increases the limit by one and iterates the search. By this control, it is assured that the procedure finds a grammar with the minimum number of rules at the expense that the system repeats the same search each time the limit is increased.

4.2 Extended Inductive CYK Algorithm

Figure 2 shows the extended inductive CYK algorithm. For inputs of a string \( w \) and a set \( P_0 \) of rules, the procedure is to generate a set \( P_1 \) of rules such that
Input an ordered set \( S_P \) of positive sample strings; an ordered set \( S_N \) of negative sample strings; an initial set \( P_0 \) of rules; and the limit \( K_{\text{max}} \) of the number of rules.

Output A set \( P \) of rules such that all the strings in \( S_P \) are derived from \( P \) but no string in \( S_N \) is derived from \( P \).

Procedure

Step 1: Initialize variables 
\[ P \leftarrow P_0 \text{ (the set of rules)}, \]
\[ N \leftarrow \{S\} \cup \{\text{the set of nonterminal symbols in } P_0\}, \]
\[ K \leftarrow |P_0| \text{ (the limit of the number of rules)}. \]

Step 2: For each \( w \in S_P \), iterate the following operations.
1. Find a set of rules by calling inductive CYK algorithm with the inputs \( w, P, N \) and \( K \). The results are returned to the global variables \( P \) and \( N \).
2. For each \( v \in S_N \), test whether \( v \) is derived from \( P \) by CYK algorithm. If there is a string \( v \) derived from \( P \), then backtrack to the previous choice point.

If no set of rules is obtained, then
1. If \( K \geq K_{\text{max}} \), terminate (no set of rules is found within the limit).
2. Otherwise, add 1 to \( K \) and restart Step 2.

Step 3: Output the result \( P \).

For finding multiple solutions, backtrack to the previous choice point. Otherwise, terminate.

Fig. 1. Top-level Procedure of Synapse

\( w \) is derived from \( P_0 \cup P_1 \). We assume that \( S \) is the starting symbol in every grammar.

This procedure includes CYK algorithm as a sub-procedure, which is extended to deal with extended CNF. In this sub-procedure, the variable \( TS \) is used to keep the test set of symbol pairs \((\beta, \gamma)\), to which a rule \( A \to \beta\gamma \) is applied in the execution of CYK algorithm. The pairs are candidates of the right side of newly generated rules. For generating unambiguous grammars, when ambiguity is detected, the process fails and backtracks to the previous choice point as shown in Step 1, (2) (b).

The procedure has nondeterministic branches, or choice points, to which the control backtracks. At a choice point, the system selects the first element in some set in predefined order, and it selects the next elements in the redoing phase after some process fails and causes backtracking. When the process of the procedure terminates at the success node, it returns sets of rules and nonterminal symbols as a result. The process fails and backtracks to the previous choice point, if it cannot generate any rule set \( P_1 \) such that \( w \) is derived from \( P_0 \cup P_1 \), \(|P_1| \leq R_{\text{max}} \) and \(|P_0 \cup P_1| \leq K\). The process has multiple results for a single set of input, since the redoing processes may generate the second and subsequent results.

It is obvious that inductive CYK algorithm has correctness in the sense that the input string is derived from the output rule set including newly generated rules. It is shown in the previous paper [6] that the algorithm also has the “completeness” property for finding the minimum sets of rules of revised CNF.
Input a string $w$, a set $P$ of rules in extended CNF; a set $N$ of nonterminal symbols; and an integer $K$ (the limit of the number of rules).

($P$ and $N$ are considered as global variables declared in the top-level procedure.)

Output A set of rules in the variable $P$ from which $w$ is derived and a set of nonterminal symbols in the rules in $N$.

Procedure Initialize the variable $TS \leftarrow \emptyset$ (the test set).

Repeat Steps 1 and 2 until $w$ is derived from $P$.

Step 1: (Test whether $w$ is derived from $P$ by CYK algorithm, and at the same time generate a test set $TS$ used in Step 2.)

1. Consider $w$ as the string $a_1a_2 \cdots a_n$. Initialize a 2-dimensional array $T$ by $T[i, 1] = \{a_i\} \cup \{A \mid (A \rightarrow a_i) \in P\}$ for all $1 \leq i \leq n$.

2. (Find every element $T[i, j]$ of $T$ such that $A^* \Rightarrow a_i \cdots a_i \cdot a_{i+j-1}$ for all $A \in T[i, j]$.) Iterate the following processes for $2 \leq j \leq n$ and for $1 \leq i \leq n - j + 1$.

   (a) $T[i, j] \leftarrow \emptyset$.

   (b) For all $k$ ($1 \leq k \leq j - 1$), $\beta \in T[i, k]$, and $\gamma \in T[i + k, j - k]$,

      i. $TS \leftarrow TS \cup \{\beta, \gamma\}$ (adding a pair to the test set).

      ii. if $(B \rightarrow \beta \gamma) \in P$ then (for generating unambiguous grammars, if $B \in T[i, j]$ then backtrack to the previous choice point (failure).)

         $T[i, j] \leftarrow T[i, j] \cup \{\beta, \gamma\} \cup \{A \mid (A \rightarrow \beta) \in P\}$.

3. If $S \in T[1, n]$ then return (success).

4. If (the number of rules generated for $w$) $\geq R_{\text{max}}$ and $|P| \geq K$, then backtrack to the previous choice point (failure).

Step 2: (Generate a rule of the form $A \rightarrow \beta \gamma$ or $A \rightarrow B$ and add it to $P$, where $(\beta, \gamma)$ is a pair contained in the test set $TS$.)

1. Select a pair $(\beta, \gamma) \in TS$ that matches the form of the rules selected by the user.

2. Select a nonterminal symbol $A \in N$ such that $(A \rightarrow \beta \gamma) \notin P$.

3. Perform one of the following operations.

   (a) (Generate a rule of the form $A \rightarrow B$.)

      If $(B \rightarrow \beta \gamma) \in P$, then $P \leftarrow P \cup \{(A \rightarrow B)\}$.

      (b) $P \leftarrow P \cup \{(A \rightarrow \beta \gamma)\}$.

   (c) Generate a new nonterminal symbol $A \notin N$, $N \leftarrow N \cup \{A\}$ and $P \leftarrow P \cup \{(A \rightarrow \beta \gamma)\}$.

Choice points (Nondeterministic branch).

Fig. 2. Procedure of Extended Inductive CYK Algorithm
4.3 Heuristics

**Limitation on the Form of Generated Rules** When the process of inductive CYK algorithm generates a rule $A \rightarrow \beta \gamma$ with a new symbol $A$ in the left hand side, this rule is not effective until a rule containing $A$ in the right hand side is also generated. Therefore, after generating the rule $A \rightarrow \beta \gamma$, we can restrict the rule generation to not terminating until a rule of the form either $B \rightarrow A \eta$ or $B \rightarrow \eta A$ is generated. By adding this heuristics, the synthesis speed increased by a factor of two to twenty.

**Intelligent backtracking** Consider the case where by adding two or more newly generated rules, a positive sample is derived from the set of rules but some negative sample is also derived from this set of rules. In this case, instead of a rule $R$ of the generated rules, another rule may be generated in the redoing process, but the rule $R$ is not be used in the derivation of the negative sample. To avoid this ineffectiveness, the system tests each of the rules in backtracking whether the negative sample is derived from the set of rules without this rule, and regenerates a rule, only if the test succeeds. By adding this heuristics\(^3\), the synthesis speed increased by several times as fast as before.

**Hash memory for avoiding repeated search** The search tree generally has two or more equivalent nodes that correspond to the same rule sets. To avoid repetition of equivalent search, Synapse (C++ version) has a hash memory for checking whether each rule set has been processed, each time the system generates the set. This mechanism\(^4\) reduces the number of rule sets generated in the search at the cost required for storing and checking the hash memory. Some experiments showed that the same rule sets appear frequently in the search, although the reduction in the computation time is not generally large (5 – 35%) because of the overhead for accessing the hash memory.

5 Performance Results

In most of the experiments in this section, we used Synapse system written in C++ and compiled by Windows version Borland C++ Compiler running on AMD Athlon processor with 1 GHz clock. We checked the correctness of all the synthesized grammars by hand.

5.1 Learning Grammars Only from Their Sample Strings

Table 1 shows the grammars synthesized by Synapse only from their sample strings, computation time, and the number GR of generated rules, which represents the size of search tree. These grammars are solutions to exercise problems

\(^3\) This is a well-known technique in logic programming for efficient computation using backtracking.

\(^4\) The similar mechanism has been used in other tree search, e.g. in game programming.
Table 1. Grammars synthesized by Synapse (Computation time in second).

<table>
<thead>
<tr>
<th>Language</th>
<th>Set of rules</th>
<th>R</th>
<th>Time</th>
<th>GR</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) balanced parentheses</td>
<td>$A: S \rightarrow ( )</td>
<td>C )</td>
<td>SS, C \rightarrow ( S \newline U: S \rightarrow ( )</td>
<td>C ), C \rightarrow S(</td>
</tr>
<tr>
<td>(b) regular expressions</td>
<td>$A: S \rightarrow a</td>
<td>b</td>
<td>S+</td>
<td>SS</td>
</tr>
<tr>
<td>(c) $w = w^R$</td>
<td>$A: S \rightarrow aa</td>
<td>bb</td>
<td>bD</td>
<td>Ca, C \rightarrow aa</td>
</tr>
<tr>
<td>(d) $#_a(w) = #_b(w)$</td>
<td>$A: S \rightarrow ab</td>
<td>ba</td>
<td>bC</td>
<td>CB</td>
</tr>
<tr>
<td>(e) $2#_a(w) = #_b(w)$</td>
<td>$A: S \rightarrow bC</td>
<td>CB</td>
<td>SS, C \rightarrow ab</td>
<td>ba</td>
</tr>
<tr>
<td>(f) $ww</td>
<td>w \in {a,b}^+$</td>
<td>$A: S \rightarrow CD</td>
<td>DC</td>
<td>FE</td>
</tr>
</tbody>
</table>

R: num. of generated rules in the grammar. GR: num. of generated rules.
$w^R$: the reversal of $w$.
$\#_x(w)$: the number of $x$'s in $w$.

in the textbook by Hopcroft & Ullman [3], where the language (c) is the set of palindromes, (d) is the set of strings containing the same number of $a$'s and $b$'s, (e) is the set of strings containing twice as many $b$'s as $a$'s, and (f) is the set of strings not of the form $ww$. Each of the grammars is the first result among the multiple results.

All the grammars except (f) are found with $R_{max} = 2$, where the parameter $R_{max}$ is the limit of the of generated rules for one positive sample. The grammar (f) cannot be obtained by setting $R_{max} = 2$ but obtained by $R_{max} = 3$.

For the grammars (b) and (f), we used the coding technique to synthesize the grammars that derive one-symbol strings. The grammar (f) is synthesized by giving Synapse the rules $C \rightarrow aa, D \rightarrow bb, E \rightarrow aa | bb$ as an initial rule set, and setting the form of rules as Chomsky normal form.

5.2 Incremental Learning Based on Similar Grammars

At present, Synapse has not directly synthesized a grammar for the language \{a$^i$b$^j$c$^k$ | i = j or j = k, i, j, k $\geq$ 1\}, which is also in the problems of Hopcroft & Ullman [3], in a reasonable amount of time. Synapse found a grammar for this language by giving the grammars of its subsets $L_1 = \{a^i b^i c^k | i, k \geq 1\}$ and $L_2 = \{a^i b^k c^k | i, k \geq 1\}$ as the initial set of rules. Each of the following grammars of $L_1$ and $L_2$, respectively, is obtained in less than 0.1 second (GR = 599).

$L_1 : S_1 \rightarrow Dc | S_1c, D \rightarrow ab | Eb, E \rightarrow aD.$
$L_2 : S_2 \rightarrow aF | aS_2, F \rightarrow bc | Gc, G \rightarrow bF.$
Given these two grammars as an initial rule set, Synapse (Prolog version) synthesized the remaining four rules

\[ S \rightarrow aF \mid aS_2 \mid Dc \mid S_1c \]

of the revised CNF in less than 820 seconds (GR = 5287). By selecting the extended CNF as the rule form, Synapse found two rules \( S \rightarrow S_1 \mid S_2 \) in 120 seconds (GR = 2365) instead of the four rules\(^5\).

The system also synthesized a grammar of revised CNF for this language \( L_1 \cup L_2 \) from the grammar of its subset \( L_1 \) of the initial rule set. Synapse (C++ version) found the remaining seven rules

\[ S \rightarrow Dc \mid S_1c \mid aG, \ G \rightarrow bc \mid aG \mid bH, \ H \rightarrow Gc \]

in 240 seconds (GR = \(3.8 \times 10^5\)).

### 5.3 Computation Time

The computation time depends on the number \(GR\) of generated rules, the limit \(R_{max}\), the number of samples, and the number of nonterminal symbols. The number \(GR\) generally grows exponentially with the size of the synthesized rule sets. The time for synthesizing a grammar with \(R_{max} = 2\) is three or four times less than that with \(R_{max} = 3\), and approximately six or more times less than without the restriction by this parameter, when the size of the rule set is larger than eight. If the longer positive samples are given before the shorter ones, the parameter \(R_{max}\) need to have a higher value than 3, and the computation time increases considerably.

Figure 3 represents the relation between the number of rules in the generated grammar and the number of all the generated rules for the grammars in Table 1 and grammar \(p\) of the language \(\{a^m b^n \mid 1 \leq m \leq n\}\), grammar \(q\) of the language \(L_1\) above, and grammar \(r\) of the parenthesis language with elements, e.g. \((\ast),(\ast\ast),(\ast\ast\ast),(\ast\ast\ast\ast),\ldots\).

### 6 Conclusion

The grammatical inference in Synapse system is composed of inductive CYK algorithm, which generates rules for positive samples, incremental learning, and search with iterative deepening. The process of inductive CYK algorithm analyzes the bottom-up parsing and generates only rules used in the parsing when the parsing does not succeed. In the extended inductive CYK algorithm, the form of the production rule is extended from the revised Chomsky normal form to include the forms \(A \rightarrow \beta\) as well as \(A \rightarrow \beta
\gamma\). This extension allows simpler sets of rules, and shorter computation times in the induction of grammars for some languages.

\(^5\) At present, the inductive CYK algorithm for extended CNF is implemented only in Prolog version of Synapse.
Another feature of our approach is incremental learning to synthesize new grammars by adding production rules to any existing grammars. This feature enables Synapse to synthesize new grammars from grammars of similar languages.

Synapse system can learn fundamental ambiguous and unambiguous CFGs in a rather short time. It synthesized the grammars for six exercise problems in the textbook by Hopcroft and Ullman [3] only from their samples. The system incrementally learned the remainder of the exercise problems, the grammar of \( \{a^i b^j c^k \mid i = j \text{ or } j = k, \ i, j, k \geq 1 \} \), from its subset grammar as shown in the previous section.

A current restriction of Synapse is that it cannot synthesize grammars with more than about 14 rules from their samples because of the computation cost. To solve this combinatorial problem, Synapse can incrementally learn grammars by generating additional rules based on the similar grammars.

Currently, Synapse has no means to generate the rules of the form \( A \rightarrow a \). Solving this problem is an important problem at present. Other future problems are as follows.

- Applying Synapse to theoretical studies of formal languages, for example, characterization of CFGs by the numbers of the required samples for the syntheses.
- Analysis and further improvement of the synthesis methods, especially those for incremental leaning.

**Fig. 3.** The number of generated rules versus the number of rules in the generated grammars.
Applying the induction methods to synthesizing other classes of grammars such as graph grammars and extending Synapse to learn grammars for natural languages.

Developing a method of representing semantics in sample strings. This is necessary, for example, for synthesizing an unambiguous CFG of arithmetic expressions. The partially structured examples in [10] might be used for this purpose.

Acknowledgements

The author would like to thank Satoshi Matsumoto and Satoru Yoshioka for their help in writing and testing Synapse system. This work is partially supported by Research Institute for Technology of Tokyo Denki University.

References

Leveraging Lexical Semantics to Infer Context-Free Grammars

Tim Oates, Tom Armstrong, Justin Harris, and Mark Nejman

Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County, Baltimore, MD 21250
{oates,arm1,jharri11,mnejmai}@umbc.edu

Abstract. Context-free grammars cannot be identified in the limit from positive examples (Gold, 1967), yet natural language grammars are more powerful than context-free grammars and humans learn them with remarkable ease from positive examples (Marcus, 1993). Identifiability results for formal languages ignore a potentially powerful source of information available to learners of natural languages, namely, meanings. This paper explores the learnability of context-free grammars given positive examples and lexical semantics. That is, the learner has a representation of the meaning of each lexical item.

1 Introduction

Learning formal languages from positive examples is a hard problem. If the language to be learned is finite and every string in the language is guaranteed to be presented at least once, the learner can memorize the strings that it sees. However, if the language can contain infinitely many strings and the learner is given a finite amount of time, then simple memorization will not work. The learner must generalize from the examples it sees. This is precisely the problem facing children learning their native language. For many important classes of languages, including regular and context-free, there are infinitely many languages that include any given set of positive examples. The challenge facing the learner is to avoid overgeneralization, to avoid choosing a language that contains the examples seen thus far and is a superset of the target language.

There is a handful of methods that are typically employed to avoid overgeneralization and establish learnability. If the learner knows that a given string is not in the target language, any overly general languages that include the string can be ruled out. A large number of learnability results reported in the literature depend on negative examples (Angluin, 1987; Gold, 1967; Oncina & Garcia, 1992). In the absence of negative examples, learnability can follow from restrictions on the class of languages from which the target is drawn (Angluin, 1982; Oates et al., 2002; Koshiba et al., 2000) or on the method for selecting examples (Denis et al., 1990; Li & Vitanyi, 1991). Establishing learnability for context-free languages is more difficult than for regular languages. A number of results on the learnability of context-free languages from positive examples require each
string to be paired with its unlabeled derivation tree (Carrasco et al., 1998; Oates et al., 2002; Salakihara, 1992). Such a pair is called a positive structural example. An unlabeled derivation tree is a parse tree in which the non-terminal labels on interior nodes are not present.

The work reported in this paper is part of a larger effort aimed at understanding what is required to allow a robot to learn fragments of natural languages given qualitatively the same inputs available to children—utterances and sensory information about the physical context in which the utterances are heard. Children rarely receive negative examples (i.e., syntactically incorrect utterances marked as such), pay little attention when they do, and have only a few very weak proxies for negative examples (e.g., failure of a caregiver to respond to an utterance as expected) (Marcus, 1993). Therefore, we focus on learning exclusively from positive examples, and on context-free languages because they are sufficiently expressive to represent large fragments of natural languages.

The key to our approach is to recognize that, in addition to hearing utterances, children have sensory access to the world around them. In particular, we assume that the lexicon contains word/meaning pairs, where the meanings have been established, for example, by some associative learning algorithm (Oates, 2001; Roy, 1999). The purpose of natural language communication is to share meanings. Therefore, the goal of the learner is to efficiently arrive at a compact representation of the language that makes it possible to determine which strings are in the language and what their meanings are.

According to Frege’s principle of compositionality, the meanings of phrases and sentences must be a function solely of the meanings of the words involved. Suppose rules for syntactic composition (i.e., productions in the grammar) are in a one-to-one correspondence with rules for semantic composition. For example, if the grammar for English has a syntactic production of the form \( S \rightarrow \text{NP} \text{ VP} \), then there must be a corresponding semantic rule that says how to combine the meaning of a \( \text{NP} \) with the meaning of a \( \text{VP} \) to get the meaning of an \( S \). Therefore, each syntactic parse tree has a structurally equivalent semantic parse tree. More importantly for this paper, each semantic parse tree, which specifies how the meanings of words and phrases are combined to yield the meaning of a sentence, has a structurally equivalent syntactic parse tree. We use lexical semantics to generate possible semantic parse trees, i.e., those that yield a coherent semantics for the sentence, and use these trees as input to an algorithm for learning context-free grammars from positive structural examples.

This paper explores the utility of lexical semantics with respect to inferring context-free languages from positive examples consisting solely of strings. Although unlabeled derivation trees are used by the learning algorithm, they are not part of its input. Rather, they are derived from lexical semantics. The main contributions are (1) the specification of a class of context-free grammars that can be learned from positive string examples and lexical semantics, and (2) a result on the learnability of lexical semantics given a context-free grammar. Our ultimate goal, as discussed in section 5, is to combine these two results into a system that uses lexical knowledge obtained via associative learning to infer
some knowledge of syntax, and uses knowledge of syntax to infer additional lexical knowledge. Over time, such a system could bootstrap itself to increasingly complete knowledge of both syntax and semantics.

The remainder of the paper is organized as follows. Section 2 briefly reviews Sakakibara’s algorithm for inferring context-free languages from positive structural examples (Sakakibara, 1992) and the use of the lambda calculus for lexical and compositional semantics. Section 3 presents a series of results on the learnability of context-free languages from strings and lexical semantics. Section 4 describes a result on the learnability of lexical semantics from syntax. Finally, section 5 summarizes, discusses related work, and points to future research directions.

2 Background

This section reviews both Sakakibara’s algorithm for inferring context-free languages from positive structural examples and the λ-calculus, which is used to represent lexical and compositional semantics.

2.1 Inferring Grammars from Positive Structural Examples

A context-free grammar (CFG) is a four-tuple \( \{N, \Sigma, P, S\} \) where \( N \) is a finite set of non-terminals, \( \Sigma \) is a finite set of terminals, \( P \) is a finite set of productions, and \( S \in N \) is the start symbol. \( N \) and \( \Sigma \) are disjoint. A CFG is in Chomsky Normal Form (CNF) if all productions are of the form \( X \to YZ \) or \( X \to \sigma \), for \( X, Y, Z \in N \) and \( \sigma \in \Sigma \). For the remainder of this paper, the word grammar refers to a CFG in CNF.

Let \( L(G) \) denote the language of grammar \( G \). An unlabeled derivation tree (UDT) for \( s \in L(G) \) is the derivation tree for \( s \) with the non-terminal labels on the interior nodes removed. UDTS can be represented as parenthesized strings. For example, \((a(bc))\) and \(((a)b)c\) are possible UDTS for string \( abc \). In the first UDT, the string \( ab \) is generated by some non-terminal, say, \( X \), and the string \( aX \) is generated by the start symbol.

A grammar is reversible if it is reset-free and invertible. A grammar is reset-free if \( A \to XY \) and \( B \to XY \) in \( P \) implies \( A = B \). A grammar is invertible if \( X \to AY \) and \( X \to BY \) in \( P \) implies \( A = B \). Sakakibara’s algorithm (Sakakibara, 1992) takes a set of positive structural examples and returns the reversible grammar that has the smallest language of all reversible grammars that contain the examples. The algorithm labels the root of each UDT with the start symbol and assigns unique labels to all interior nodes. The union of the productions in the newly labeled trees serve as the initial grammar. Since the generating grammar is known to be reversible, pairs of non-terminals that violate the reset-free and invertible properties are merged until no such pair exists, at which point the algorithm terminates. Let \( GI(E) \) be the output of this algorithm on input \( E \).
2.2 Montague Grammar and the $\lambda$-Calculus

The first formal treatment of natural language semantics is typically attributed to Richard Montague. In three of his last papers, Montague introduced an intensional logic for representing the semantics of natural language strings (Dowty et al., 1981; Partee & Hendriks, 1996). The semantics of words are represented as expressions in the $\lambda$-calculus which are (typically) functions. These functions take arguments and return values, both of which are also expressions in the $\lambda$-calculus. The semantics of strings are computed bottom-up by applying the $\lambda$-calculus expressions of lower-level constituents to one another. Table 1 shows a small lexicon in the Montague framework that will serve as a running example throughout the remainder of this section.

<table>
<thead>
<tr>
<th>Lexeme</th>
<th>$\lambda$-calculus expression</th>
<th>Semantic type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nathaniel</td>
<td>NATHANIEL</td>
<td>e</td>
</tr>
<tr>
<td>Isabel</td>
<td>ISABEL</td>
<td>e</td>
</tr>
<tr>
<td>loves</td>
<td>$\lambda x \lambda y . \text{LOVES}(yx)$</td>
<td>$&lt;e, &lt;e, t &gt;&gt;$</td>
</tr>
<tr>
<td>rests</td>
<td>$\lambda x . \text{RESTS}(x)$</td>
<td>$&lt;e,t&gt;$</td>
</tr>
</tbody>
</table>

The $\lambda$-calculus, developed by Alonzo Church, is a language definition useful for exploring function descriptions and function application. It is a model suited for functional programming used in some languages (e.g., Lisp). Function descriptions are created by performing lambda abstractions. In lambda abstractions a variable argument is given scope and bound to a lambda operator. These anonymous functions can then be applied to other expressions.

Three rules are used to reduce $\lambda$-calculus expressions: $\alpha$, $\beta$, and $\eta$ reductions. A $\lambda$-calculus expression, when fully reduced, will be a first-order logic expression. Let $^{'x/y/z'}$ denote replacing instances of $x$ with $y$ in expression $z$. For any two arbitrary expressions, a lambda-operator may be bound to the same named variable in both. Performing an $\alpha$-reduction on one of the expressions results in a renaming of the variable to a unique name. When an expression is functionally applied to an argument expression it is called $\beta$-reduction. The first lambda-operator and variable is stripped off and the argument is substituted for each instance of the variable. The final operation removes lambda-operators which are bound to variables no longer found in the body of the expression. The three reductions are formally defined below where $\lambda \text{VAR}$ is a lambda-operator bound to a variable and EXPR is an arbitrary expression:

1. $\alpha$-Reduction:
   \[ \lambda \text{VAR} . \text{EXPR} \Rightarrow_\alpha \lambda \text{UNIQUE-VAR} . [\text{VAR/UNIQUE-VAR}] \text{EXPR} \]
2. $\beta$-Reduction:
   \[ (\lambda \text{VAR} . \text{EXPR}_1) \text{EXPR}_2 \Rightarrow_\beta [\text{VAR/EXPR}_2] \text{EXPR}_1 \]
3. $\eta$-Reduction:
   \[ \lambda \text{VAR} . \text{EXPR} \Rightarrow_\eta \text{VAR} \]
For our purposes, and generally, the processes of abstraction and $\beta$-reduction are the most useful tools. Abstractions (i.e., functions) provide the representation for lexical semantics, and $\beta$-reductions are used to compute compositional semantics.

The following example shows the derivation of the meaning (an expression in first-order logic) of the UDT \((\text{Nathaniel} \loves \text{Isabel})\). Note that there is ambiguity for any given pair inside a set of parentheses as to which is the function and which is the argument. For now, we assume the order of application is given and it is as follows: \loves applies to \text{Isabel} and \loves \text{Isabel} applies to \text{Nathaniel}.

1. \((\text{Nathaniel} \loves \text{Isabel}) = (\text{NATHANIEL} (\lambda x \lambda y . \text{LOVES}(y;x) \text{ ISABEL}))\)
2. \((\text{NATHANIEL} (\lambda x \lambda y . \text{LOVES}(y;x) \text{ ISABEL}) \Rightarrow (\text{NATHANIEL} \lambda y . \loves(y;\text{ISABEL})))\)
3. \((\text{NATHANIEL} \lambda y . \text{LOVES}(y;\text{ISABEL})) \Rightarrow \loves(\text{NATHANIEL};\text{ISABEL})\)

From ordered function application we arrive at the standard English meaning of the string.

2.3 Semantic Types

Every $\lambda$-calculus expression has a semantic type, and these types constrain how $\lambda$-calculus expressions compose with one another. Types are either basic or complex. Basic types are defined as e, entities (e.g., Nathaniel, Isabel), or t, truth values (e.g., true, false). Complex types, which represent types of functions, are defined recursively: if $\alpha$ and $\beta$ are types, then $<\alpha,\beta>$ is a type. Semantic types dictate the direction of function application (i.e., when applying a function of type $<\gamma,\delta>$, the argument to the function must have type $\gamma$). Take from Table 1 the types for \text{Isabel} and \text{rests}, e and $<\text{ext}>$ respectively. The order of application must be from \text{rests} to \text{Isabel} to properly compose.

Each expression in the $\lambda$-calculus, being a function description, has a parameter and return value. The semantic type of a function can be represented by $<\alpha,\beta>$ where $\alpha$ and $\beta$ are the semantic types of the parameter and return value respectively. There exists a many-to-one mapping between $\lambda$-calculus expressions and semantic types (e.g., \text{Nathaniel} and \text{Isabel} are both of type \text{e}).

3 Learning Syntax Using Lexical Semantics

In this section we formally explore the utility of UDTs derived from semantic parse trees with respect to grammatical inference.

3.1 Lexical Semantics Reducing Numbers of Parse Trees

How many semantically valid UDTs could there be for a string? For a string of length $n$, the number of possible UDTs is equal to the number of possible binary bracketings of that string. Catalan, an 18th century Belgian mathematician, posed a problem for finding the number of ways to compose factors in calculating
products (Weisstein, 2003). This problem is equivalent to finding the number of binary bracketings. Formally, \( C(n) \), the \( n \text{th} \) Catalan number, is \( \frac{(2n)!}{n!(n+1)!} \). For strings of length \( n \), the number of binary bracketings is equal to the \( n - 1 \text{th} \) Catalan number.

The number of possible parse trees for a string grows exponentially in the length of the string. From a learnability perspective, dealing with large numbers of trees for a single string is not easily handled. Given lexical types for terminal nodes, performing type checking on possible trees will rule out a subset of them as compositionally invalid. Type checking succeeds if after type composition over a tree for a string the root node has semantic type \( t \). Bracketed strings of this form are called semantically valid.

**Table 2. A fragment of the English lexicon**

<table>
<thead>
<tr>
<th>Lexeme</th>
<th>Semantic type</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>(&lt;&lt;e,t&gt;,&lt;&lt;e,t&gt;,t&gt;&gt;)</td>
</tr>
<tr>
<td>dog</td>
<td>(&lt;e,t&gt;)</td>
</tr>
<tr>
<td>cat</td>
<td>(&lt;e,t&gt;)</td>
</tr>
<tr>
<td>hates</td>
<td>(&lt;&lt;&lt;e,t&gt;,t&gt;,&lt;e,t&gt;&gt;)</td>
</tr>
</tbody>
</table>

Take, for example, the string \( s = \text{the cat hates the dog} \) generated by some grammar. Given the length of \( s \) is 5, \( C(4) = 14 \). Of the 14 possible parses, 3 are valid in terms of semantic composition: ((the cat) (hates (the dog))), ((the cat) hates (the dog)) and (((the cat) hates) the dog). As shown above, a first order logic expression for the string can then be constructed following the application orderings for each of the three trees. In one case the expression denotes the cat is the hater of the dog and in others the dog is the hater of the cat. The following is a type-checking evaluation for one possible parse tree.

Take (the cat) in type form as \( \langle<<e,t>,<<e,t>,t>>\rightarrow <e,t>\rangle \) from Table 2. Performing the application in the order given (the cat) has type \( <e,t>,t> \). The type for (the dog) is clearly equivalent. The next bracket combines (hates (the dog)), thus the type is \( \langle<<<e,t>,t>,<e,t>>,<<e,t>,t>\rangle \rightarrow <e,t>,t> \rangle \) resulting in a type of \( <e,t> \). The final combination is between the types of (the cat) and (hates (the dog)) or \( \langle<<e,t>,t>\rightarrow <e,t>\rangle \). The root results in a \( t \) for this bracketing of the string.

### 3.2 Computing Semantically Valid Parses

For string \( s \), type \( \alpha \), and grammar \( G \), let \( T(s) \), \( T_{\alpha}(s) \), and \( T_{G}(s) \) denote sets of unlabeled derivation trees. \( T_{G}(s) \) is the set of trees that are possible given grammar \( G \), \( T(s) \) is the union of \( T_{G}(s) \) for all grammars that generate \( s \). Assuming that lexical semantics are given, \( T_{\alpha}(s) \) is the restriction of \( T(s) \) to those trees whose root have type \( \alpha \). To use semantic parse trees to learn syntax, it must be possible to compute \( T_{\alpha}(s) \) efficiently. We show that this is the case in what follows, starting from an efficient algorithm for computing \( |T(s)| \).
The recurrence below can be used to compute \(|T(s)|\) bottom up, \(M\) is a zero-indexed \(n \times n\) array, where \(|s| = n\), and \(M[i, j]\) is \(\left|T(s[i, \ldots, j])\right|\), i.e., the number of possible UDTs for the substring of \(s\) ranging from position \(i\) to position \(j\). Therefore, \(|T(s)| = M[0, n - 1]\).

\[
M[i, i] = 1
\]

\[
M[i, j] = \sum_{l=2}^{n} \sum_{p=0}^{l-2} \sum_{q=0}^{l-2} M[p, p + q] \cdot M[p + q + 1, p + l - 1]
\]

Assuming the grammar is in CNF, the only way to generate a terminal is via a production of the form \(X \rightarrow \sigma\). Therefore, there is only one unlabeled derivation tree for each substring of \(s\) of length 1 and \(M[i, i] = 1\) for all \(i\). All other entries are computed bottom-up via the second line in the recurrence. The first sum is over substring lengths \(l\), ranging from 2 to \(n\). The second sum is over all starting positions, \(p\), in \(s\) of strings of length \(l\). Together, \(l\) and \(p\) identify a substring of \(s\), namely, \(s[p, \ldots, p + l - 1]\) that must have been generated by a production of the form \(X \rightarrow YZ\). The third sum is over all ways in which \(Y\) and \(Z\) can divide up that substring. For any such division, the number of trees for the substring is the product of the number of trees for the part \(Y\) generates and the number of trees for the part that \(Z\) generates.

Given lexical semantics, it is possible to compute \(T_{\delta}(s)\) using the recurrence below. Rather than \(M\) being an \(n \times n\) array, it is an \(n \times n \times m\) array where \(m\) is the number of distinct types that can label interior nodes in a semantic parse tree. Let \(\delta(\text{expr})\) take on value 1 if \(\text{expr}\) is true, and 0 otherwise. Let \(\text{typeof}(\text{expr})\) return the type (an integer between 0 and \(m - 1\)) of \(\text{expr}\).

\[
M[i, i, l] = \delta(\text{typeof}(s[i, \ldots, i]) = t)
\]

\[
M[i, j, l] = \sum_{t_1, t_2} \sum_{l=2}^{n} \sum_{p=0}^{l-2} \sum_{q=0}^{l-2} M[p, p + q, t_1] \cdot M[p + q + 1, p + l - 1, t_2] \cdot \delta(\text{typeof}(t_1 \oplus t_2) = t \lor \text{typeof}(t_2 \oplus t_1) = t)
\]

There is still only one unlabeled derivation tree for each substring of length 1, but the type of that tree is the type of the lambda expression for the meaning of the terminal. The outer sum in the second line of the recurrence is over all \(m^2\) pairs of types. Given that \(s[p, \ldots, p + l - 1]\) will be split at offset \(q\) (as established by the inner three sums), that split will yield a valid semantic parse only if the types of the two halves are compatible, i.e., if one can be applied to the other. If \(t_1\) is the type of the left half and \(t_2\) is the type of the right half, and applying \(t_1\) to \(t_2\) or applying \(t_2\) to \(t_1\) yields type \(t\), then \(M[i, j, t]\) is updated.

The recurrence above can be computed in \(O(m^2 n^3)\) time. Note that \(m\) will typically be a small constant because interior nodes can only be labeled by lexical types and types out of which lexical types are composed. Also, it is trivial to augment the computation so that \(M\) can be used to extract all semantically valid parses of a given type. This is done by keeping a list of split points, \(q\), for each \((i, j, t)\) that yield the desired type (i.e., \(t\)). This increases the storage required by a multiplicative factor of \(n\).
3.3 Using Semantically Valid Parse Trees to Learn Syntax

Our goal is to use lexical semantics and positive string examples to obtain positive structural examples that can be used to learn syntax. Clearly, if for every string in \( L(G) \) there is a single semantically valid parse, then learning is straightforward. Later in this section we define a class of grammars for which this is the case. What happens, though, if there are multiple semantically valid parses for some string(s) in \( L(G) \)?

Let \( E = \{ s_1, s_2, \ldots \} \) be a set of strings. Let \( T(E) = \cup_{s_i \in E} T(s_i) \), and let \( T_C(E) \) and \( T_0(E) \) be defined similarly. Now consider grammar \( G \) for which \( L(G) = \{ xcd, xab, ycd, yab \} \) and \( T_C(L(G)) = \{ ((xc)d), ((xa)b), (y(cd)), (z(ab)) \} \).

Suppose the type of \( a \) and \( c \) is \( t \), and the type of all other lexical items is \( < t, t > \). Then the semantically valid parses of the strings in \( L(G) \) are as follows:

\[
T_1(L(G)) = \{ ((xc)d), ((xa)b), (y(cd)), (z(ab)), (x(cd)), (x(ab)), (yc)d, (za)b \}
\]

For a reset-free grammar to generate these UDTs, the non-terminal that generates \( cd \) must also generate \( ab \) due to the trees \( (x(ab)) \) and \( (x(cd)) \). Coupled with the fact that \( (y(cd)) \) and \( (z(ab)) \) are in \( L \), the grammar must also generate \( (y(ab)) \) and \( (z(ab)) \), neither of which are in \( L(G) \). That is, learning a grammar based on all semantically valid parse trees can, in some cases, lead to overgeneralization.

Even so, given a sample of strings, \( E \), from the language of some grammar, \( G \), we can establish a few useful properties of the grammar learned from the structural examples contained in \( T_1(E) \). For example, the language of that grammar will always be a subset of the language of the grammar learned from \( E \) with the true UDTs. Somewhat more formally:

\[
L(GI(T_C(E))) \subseteq L(GI(T_1(E)))
\]

To see that this is true, first note that the output of \( GI \) is independent of the order in which merges occur. It then suffices to show that the above holds for a single merge order. Suppose the merges performed by \( GI \) on \( T_1(E) \) are precisely those performed on \( T_C(E) \) up to the point where \( GI(T_C(E)) \) would terminate. If \( GI(T_1(E)) \) were to stop at this point, the language of the resulting grammar would be precisely \( L(GI(T_C(E))) \) because it would contain all of the productions in \( GI(T_C(E)) \) plus productions that produce all and only the trees in \( T_1(E) - T_C(E) \), and these trees correspond to strings in \( E \) that are in \( L(GI(T_C(E))) \).

Additional merging required for \( GI(T_1(E)) \) to terminate can only add strings to the resulting language, so \( L(GI(T_C(E))) \subseteq L(GI(T_1(E))) \).

3.4 Lexically Unambiguous Parses

The previous section established that there exist grammars for which the language inferred using all semantically valid parses of the training examples is a superset of the language inferred using the true semantic parses of the training examples. This section defines a class of grammars for which every string in the
language of such a grammar has a single semantically valid parse. For grammars in this class, lexical semantics ensure learnability.

For a given grammar $G$, let $right(X)$ be the set of terminals and non-terminals that can occur in the rightmost position of any string derivable in one or more steps from $X$. Let $left(X)$ be defined analogously. Let $canapply(A, B)$ return true iff either $A$ is of the type $<\alpha, \beta>$ and $B$ is of the type $\alpha$ (i.e. $A$ can be applied to $B$) or $A$ is of the type $\gamma$ and $B$ is of the type $<\gamma, \delta>$ (i.e. $B$ can be applied to $A$).

**Theorem 1.** If for every production of the form $X \rightarrow YZ$ in grammar $G$ neither condition 1 nor condition 2 below holds, then there is exactly one semantically valid parse of every string in the language generated by $G$.

1. $\exists A \in right(Y) \cup \{Y\} \land \exists B \in left(Z) \land canapply(A, B)$
2. $\exists A \in right(Y) \land \exists B \in left(Z) \cup \{Z\} \land canapply(A, B)$

**Proof.** The proof will be by induction on the height of the derivation tree that generates the string. Because $G$ is in CNF, all productions are of the form $X \rightarrow YZ$ or $X \rightarrow \sigma$. Each non-terminal is generated by a tree of height one, and there is only one way to semantically parse any given non-terminal, so strings derived by a tree of height one have a single semantically valid parse. This is the base case. The inductive assumption is that all strings generated by a tree of height no more than $h$ have a single semantically valid parse.

Suppose some string has a derivation tree of height $h+1$ and the string has multiple semantically valid parses. Let $X$ be the non-terminal that roots the derivation tree, and let $Y$ and $Z$ be the non-terminals that root its left and right subtrees (i.e. the derivation tree was produced by expanding $X$ to $YZ$ via production $X \rightarrow YZ$). The subtrees rooted by $Y$ and $Z$ can have height at most $h$, so by the inductive assumption the strings they generate have a single semantically valid parse.

Therefore, the only way for the string generated by $X$ to have more than one semantically valid parse is for some non-terminal on the right edge of the tree generated by $Y$ to have a type that can be applied to the type of some non-terminal on the left edge of the tree generated by $Z$ (or vice versa). However, conditions 1 and 2 above explicitly disallow this, so there can be no string generated by a tree of height $h+1$ with multiple semantically valid parses. Inductively, the theorem holds for strings generated by derivation trees of any height, i.e. all strings.

The theorem above says that there is a class of grammars for which all strings have a single semantically valid parse. Grammars in this class can be learned by any algorithm that learns CFGs from unlabeled derivation trees.

## 4 Semantic Type Inference

Until now we have assumed that the learning algorithm was given lexical semantics for all terminals. Now let us consider the case where we are given the
UDTs and order of application for internal nodes for a sample of strings in a grammar. From this information we will demonstrate how all semantic types can be inferred using a minimal subset of semantic types.

Let \( X \rightarrow Y Z \) be a production for which the meaning of \( X \) is obtained by applying the meaning of \( Y \) to the meaning of \( Z \). Because the production obeys type constraints, if \( X \) is of type \( < \alpha, \beta > \) then \( Y \) must be of type \( \alpha \) and \( X \) must be of type \( \beta \). Types can be inferred for a given production in CNF in two ways. If the type of \( Y \) is known, the types of \( X \) and \( Z \) can be inferred. If the types of \( X \) and \( Z \) are known, the type of \( Y \) can be inferred. We assume that the start symbol has the same type for all strings. Consider the following grammar assuming left to right function application for all productions (e.g., assuming \( NP \) applies to \( VP \) and not vice versa):

\[
\begin{align*}
S & \rightarrow NP\ VP \\
NP & \rightarrow DET\ N \\
VP & \rightarrow TV\ NP \\
VP & \rightarrow IV
\end{align*}
\]

The start symbol, \( S \), has semantic type of \( t \). For this grammar, if we know the types of the elements of \( \{N \cup T\} \) that are never used as functors in any production, we can infer the types of all other elements of \( \{N \cup T\} \). For the above grammar the known types are of \( S, VP, N \) and \( IV \). Knowing \( S \) and \( VP \) allows inference of \( NP \), which then allows inference of \( DET \) using \( N \), and inference of \( TV \) using \( VP \) and \( NP \).

We say that non-terminal \( Y \) never applies if it never occurs in a production \( X \rightarrow YZ \) or \( X \rightarrow ZY \) for which the meaning of \( Y \) is applied to the meaning of \( Z \) to obtain the meaning of \( X \). The following theorem defines a subset of types that need to be known to ensure that all types are inferable.

**Theorem 2.** For any context-free grammar \( G \) in Chomsky Normal Form, given the types of all non-terminals that never apply, the types of all other non-terminals can be inferred.

**Proof.** Let \( G^* \) be the directed graph created from grammar \( G \) as follows. For each production of the form \( X \rightarrowYZ \) add nodes labeled \( X, Y, \) and \( Z \) to \( G^* \). If \( Y \) applies to \( Z \), add edges from node \( Y \) to both node \( X \) and \( Z \). If \( Z \) applies to \( Y \), add edges from node \( Z \) to both node \( X \) and \( Y \). Each node is marked as either known or unknown, depending on whether the type of the associated non-terminal is known. Nodes with out-degree zero correspond to non-terminals that never apply, and are therefore assumed to be known. All other nodes are initially marked unknown.

If all of the neighbors of a node are marked known, then its type can be inferred and the node marked known. This inference/marking step can be repeated until no new nodes are marked known. Because the number of unknown nodes decreases by at least one on each step, or the algorithm terminates, there can be no more than \( O(|V|) \) iterations, each of which might need to do \( O(|P|) \) work.
Suppose there exists a grammar $G$ for which all nodes in $G*$ are not marked known when the algorithm terminates. Let $X$ be the non-terminal associated with this node. If the type of $X$ cannot be inferred, then the type of at least one of its neighbors, call it $Y$, cannot be inferred. That is, there is path of length 1 from $X$ to a node whose type cannot be inferred. Likewise, because the type of $Y$ cannot be inferred, the type of at least one of its neighbors cannot be inferred, and this node lies on a path of length 2 from $X$. Inductively, there must be a node at the end of a path of length $n$ for all $n \geq 0$ from $X$ whose type cannot be inferred. However, because the graph is acyclic, all paths must terminate in a node with out-degree zero in $O(|V|)$ steps, and the types of all such nodes are known. This is a contradiction. Therefore, the theorem holds.

The importance of theorem 2 is that words with types that never apply are typically those that refer to perceptually concrete aspects of the environment, such as nouns and adjectives. That is, it is possible that an embedded learner might associatively learn the types of these words (Oates, 2001; Roy, 1999), and then use knowledge of syntax to infer the semantic types of all other words in the lexicon.

5 Conclusion

This paper represents the first results of our inquiry into the relationship between meanings and learnability for context-free grammars. Theorem 1 established that there exists a class of grammars whose syntax can be learned from positive string examples and lexical semantics. Theorem 2 established that it is possible to infer lexical semantics given a grammar’s syntax and a small number of lexical types.

The work most similar to ours reported in the literature is that of Tellier and her colleagues (Tellier, 1998; Dudau-Sofronic et al., 2001) who are also interested in the role of lexical semantics in grammatical inference. They propose an algorithm for inferring rigid categorial grammars given strings and their meanings, though the algorithm has exponential complexity and it is unclear (i.e., there is no proof) whether it converges to the target grammar or some grammar containing the target.

Future work will proceed in a number of directions. Ultimately, we want to develop algorithms that will iteratively use incomplete lexical knowledge to infer grammar fragments, and then use these fragments to infer more lexical knowledge. The goal is to have a learner that can provably converge on the correct lexicon and grammar by bootstrapping from some small amount of knowledge about lexical semantics obtained via associative learning.

An important result that we’re currently working toward is an algorithm for inferring syntax when there are multiple semantically valid parse trees for one or more strings. One possible approach is to compute the corresponding semantics of each parse and use the fact that the learner is embedded in an environment to determine which parse is correct, i.e., which one refers to the current state of the world. An alternative approach might involve noticing when a merge would include strings in the grammar that are not observed.
Bibliography


Learning Context Free Grammars in the Limit Aided by the Sample Distribution

Yoav Seginer

ILLC, Universiteit van Amsterdam
yseginer@science.uva.nl

Abstract. We present an algorithm for learning context free grammars from positive structural examples (unlabeled parse trees). The algorithm receives a parameter in the form of a finite set of structures and the class of languages learnable by the algorithm depends on this parameter. Every context free language belongs to many such learnable classes. A second part of the algorithm is then used to determine this parameter (based on the language sample). By Gold’s theorem, without introducing additional assumptions, there is no way to ensure that, for every language, the parameter chosen by the learner will make the language learnable. However, we show that determining the parameter based on the sample distribution is often reasonable, given some weak assumptions on this distribution. Among other things, repeated learning, where one learner learns the language the previous learner converged to, is guaranteed to produce a learnable language after a finite number of steps. This set of limit languages then forms a natural class of learnable languages.

1 Introduction

The problem of learning context free grammars (in the limit) from structural examples has attracted much attention ([6],[7],[2],[4]). A structural example for a sentence consists of the parse tree structure of that sentence without the non-terminal labels. Also known as a tree language, the parse trees of a context free language can be characterized by a tree automaton ([6],[7]). This allows methods for learning regular languages to be extended to the learning of context free languages from structural examples (see [6],[7],[2]).

When Gold [3] first presented his paradigm of learning in the limit, he also showed that within this paradigm the class of context free grammars is not learnable from positive examples. While the use of structural examples does seem to make the learning problem easier, it is still not enough to get around Gold’s theorem and (just as with the class of regular languages) the class of context free languages is not learnable from structural examples. For some, this is sufficient reason to reject Gold’s learning paradigm altogether. Others, who still want to apply this paradigm to the learning of context free grammars, may resort to two methods. The first is to supply the learner with some additional information, the other, to restrict the class of languages which has to be learned to a subclass of the context free languages.
Sakakibara presented solutions of both types. In [6], the learner is allowed to use structural equivalence and membership queries, thus gaining additional information about the language, while in [7] only positive examples are used but the algorithm is restricted to the class of reversible context free grammars. The two approaches can also be combined, of course. Fernau [2] extended [7] to learning the $\delta$-distinguishable tree languages. Every choice of a distinguishing function $\delta$ results in a learnable class. For any language, when the learning algorithm is equipped with an appropriate distinguishing function, it is guaranteed to learn the language correctly from positive examples. Clearly, by Gold’s theorem, to guarantee a correct choice of the distinguishing function $\delta$ for every language, the learner must make use of some information beyond the positive examples for the language.

The algorithm we present in this paper also learns context free grammars from structural examples. The learning algorithm, which is very simple and natural, uses a finite set of structures, the context set. Similarly to Fernau’s distinguishing functions, every context set makes a subclass of the context free languages learnable and together these subclasses cover the whole class of context free grammars. Another property shared with Fernau’s algorithm is that even when a language is outside the class of languages learnable by the algorithm (for a given context set), the algorithm converges to a language containing (overgeneralizing) the original language. The resulting language does not depend on the order in which the language is presented to the learner (compare this with the approximation property in [1]). Beyond these similarities, the learning algorithm itself is of a completely different nature, as the ability to distinguish trees, which is given as an oracle (the distinguishing function) to Fernau’s algorithm, does not exist for our algorithm (until the learning process actually converges). The workings of our algorithm can in some ways be compared with the family of tail algorithms presented in [5], even though these algorithms are not learning algorithms in the sense of Gold (as convergence in the limit is not guaranteed).

In addition to presenting the basic algorithm (section 2) and proving its convergence (section 3) we present (in section 5) a method for choosing the context set to be used, based on the sample being presented to the learner. It relies on the fact that even when the learning process is confined to receiving positive examples, the learner has access not only to the positive examples themselves, but also to the order and frequencies in which they appear. Whether this contains any information about the language being learned depends on the way in which the examples are generated. Usually, when learning within Gold’s paradigm of identification in the limit, the only assumption one makes about the process generating the examples is that every sentence will eventually be generated by it. It has often been observed that this is a very strong requirement (even in Gold’s original paper some alternatives are examined).

The method presented here constructs the context set out of the most frequent structures in (some initial segment of) the language sample. Such frequent structures are clearly evident in natural languages and it has already been observed by linguists [8] that these structures may play an important role in lan-
guage acquisition by children. Whether this procedure produces a good context set for the language to be learned depends on the settings in which we wish to apply the algorithm and is usually an empirical question. There are, however, several properties of the learning algorithm which make this method a good candidate. We discuss this in section 5.

A basic property shown in sections 4 and 5 is that there is a natural process which leads to learnable languages and if a language becomes unlearnable (for whatever reason) there is a way of returning to a learnable language (even if a somewhat different one).Beginning with any language, repeated learning (that is, every learner learning the language which the previous learner converged to) is bound to arrive (after a finite number of steps) at a language learnable by all subsequent learners. The class of these limit languages is then a very natural learnable class.

2 The Learning Algorithm

The learning algorithm receives trees as input and produces a set of rules (the grammar) as output. It learns, therefore, from structural examples. The algorithm creates grammar rules by looking at subtrees of the sample trees and the context trees in which they appear. We begin by defining all these structures.

A tree $T$ is either a single terminal $\sigma \in \Sigma$ or a tuple $\langle T_1, \ldots, T_l \rangle$ ($1 \leq l$) where $T_1, \ldots, T_l$ are trees. We write $T(\Sigma)$ for the set of all trees over the terminal set $\Sigma$. A context is a tree in which a single leaf can be substituted by a tree to create a tree. We assume that $\ast \notin \Sigma$ ($\ast$ indicates the leaf in the context where a tree can be substituted). A context $c$ is either the symbol $\ast$ or $c = \langle T_1, \ldots, T_{i-1}, c', T_{i+1}, \ldots, T_l \rangle$ ($1 \leq l$, $1 \leq i$) where $c'$ is a context and $T_1, \ldots, T_{i-1}, T_{i+1}, \ldots, T_l \in T(\Sigma)$. We write $C_T(\Sigma)$ for the set of all contexts over the terminal set $\Sigma$.

Given a context $c$ and a tree $T$ we write $c(T)$ for the tree created by substituting $T$ for the (unique) leaf labeled $\ast$ in $c$. Given a tree $T$ and a node $\nu$ in this tree, we can always write $T = c(S)$ where $S$ is the subtree rooted at the node $\nu$ (when $\nu$ is the root of the tree $T$, $c = \ast$). A tree $S$ is a subtree of tree $T$ if there exists a context $c$ such that $T = c(S)$. We define $S(T) = \{ S \in T(\Sigma) : \exists c \in C_T(\Sigma) \text{ s.t. } T = c(S) \}$, the set of subtrees of $T$. Given a set of trees $X$ (e.g. a language $L$), the set of subtrees of $X$ is $S(X) = \bigcup_{T \in X} S(T)$.

Central to the operation of the algorithm is a finite context set $C \subseteq C_T(\Sigma)$. This context set $C$ is allowed to change a finite number of times as the language is being presented to the learner, so we have a finite sequence of context sets $C_1, \ldots, C_n$. However, in what follows we will assume that the last context set, $C_n$, is used throughout the presentation of the language sample. This is a legitimate assumption, since, in the worst case, each time the context set changes, the algorithm can rerun all previous learning steps using the updated context set.

For the algorithm presented here it will become clear that much less is needed. The way in which the context set sequence $C_1, \ldots, C_n$ is determined will be discussed later, in section 5. Therefore, from now on we will assume a finite
context set $\mathcal{C}$ which is fixed at the beginning of the learning process. It is also required that $\ast \in \mathcal{C}$.

To construct grammar rules, the algorithm uses the partially ordered set $\mathcal{O} = (2^\Sigma \cup \Sigma, \leq)$ with the subset ordering on the elements of $2^\Sigma$ (the elements in $\Sigma$ are neither comparable with elements in $2^\Sigma$ nor with each other). Every grammar rule is then of the form $(y | x_1, \ldots, x_l)_d$ where $y \in 2^\Sigma$, $x_1, \ldots, x_l \in \mathcal{O}$, $1 \leq l$ and $0 \leq d$. The natural number $d$ is the level of the rule.

Let $R$ be a set of such rules. We define the function $\text{ctx}^R(T)$ which maps a tree $T$ into an element in $\mathcal{O}$ based on the rules in the rule set $R$. If $T = \sigma \in \Sigma$ then $\text{ctx}^R(T) = \sigma$. Otherwise $T = \langle T_1, \ldots, T_l \rangle$ and $\text{ctx}^R(T) = \bigcup \{ y : \exists (y | x_1, \ldots, x_l)_k \in R \text{ s.t. } k \leq \text{depth}(T), x_i \leq \text{ctx}(T_i), 1 \leq i \leq l \}$ where depth$(T)$ is defined recursively as depth$(T) = 1 + \max_{i=1, \ldots, l} \text{depth}(T_i)$ and depth$(\sigma) = 0$ (for a terminal $\sigma \in \Sigma$). The language $L(R)$ generated by a rule set $R$ is defined to be $L(R) = \{ T \in T(\Sigma) : \ast \in \text{ctx}^R(T) \}$.

The algorithm maintains a set of pre-rules, $\mathcal{P}$. A pre-rule is of the form $(y | T_1, \ldots, T_l)$ where $y \in 2^\Sigma$ and $T_1, \ldots, T_l$ are trees. A projection $\pi^R$ from pre-rules to rules, based on the rule-set $R$, is defined. If the pre-rule $P$ is $(y | T_1, \ldots, T_l)$ then $\pi^R(P) = (y | \text{ctx}^R(T_1), \ldots, \text{ctx}^R(T_l))$. The number depth$(\langle T_1, \ldots, T_l \rangle)$ is the depth of the pre-rule $P$, which is equal to the level of the rule $\pi^R(P)$.

The algorithm is initialized with an empty set of pre-rules $\mathcal{P}$ and may change this pre-rule set at every step. For every pre-rule set $\mathcal{P}$, the rule set hypothesized by the algorithm is the unique rule set $R$ such that $R = \pi^R(\mathcal{P}) = \{ \pi^R(P) : P \in \mathcal{P} \}$. This rule set is easy to calculate from $\mathcal{P}$ because of the level assigned to each rule. When computing $\pi^R(P)$ for a a pre-rule of depth $d$, the function $\text{ctx}^R$ makes use only of rules of a level strictly smaller than $d$. Therefore, the computation can advance in a straightforward manner through $\mathcal{P}$, in increasing order of depth.

We now define an ordering of the rules by $(y | x_1, \ldots, x_l)_d \leq (z | w_1, \ldots, w_m)_k$ iff $l = m$, $x_i \leq w_i$ for $i = 1, \ldots, l$, $y \geq z$ and $d \leq k$. After having calculated the rule set $R$ associated with a pre-rule set $\mathcal{P}$, the algorithm removes from $\mathcal{P}$ any pre-rule $P \in \mathcal{P}$ such that the rule $\pi^R(P)$ is not minimal in $R$. Removing pre-rules from $\mathcal{P}$ entails the removal of rules in $R$. However, it is easy to see that, because of the way the rule ordering is defined, this does not change the function $\text{ctx}^R$ and there is therefore no need to recalculate the rule set.

When presented a sample tree $S$, the algorithm traverses all its nodes (except for the leaves) in increasing order of depth (the depth of a node being the depth of the subtree rooted at it). For each such node $\nu$, the tree $S$ can be written as $S = c(T)$, where $T$ is the subtree rooted at $\nu$ and $c$ is an appropriate context. Let $T = \langle T_1, \ldots, T_l \rangle$ ($\nu$ is not a leaf) and let $\mathcal{P}$ and $R$ be the pre-rule set and corresponding rule set as calculated by the algorithm up to this point. The algorithm then performs the following operations: 

1. Check whether there is a rule $R \in R$ such that $R \leq \pi^R(\mathcal{C} \cap \{c\}) | T_1, \ldots, T_l$.

If there is such a rule in $R$ then go on to the next node.
2. Otherwise, if there exists already a pre-rule \((y|T_1, \ldots, T_l)\) in \(P\) then this pre-rule is replaced by the pre-rule \((y \cup (C \cap \{c\})|T_1, \ldots, T_l)\). Otherwise, the pre-rule \((C \cap \{c\}|T_1, \ldots, T_l)\) is added (as is) to \(P\).

3. After the pre-rule set is updated, the corresponding rule set is calculated and all non-minimal rules (and corresponding pre-rules) are removed.

3 Convergence of the Algorithm

Given any finite context set \(C\) and any context free language \(L\), the algorithm will be shown to converge to a language \(L^C\), which is independent of the specific language sample which is presented to the algorithm and such that \(L \subseteq L^C\). Not only the language but also the grammar (rule set) hypothesized by the algorithm will be shown to converge. However, the exact grammar which the algorithm converges to may depend on the order in which the examples are presented to it.

From now on we assume that the context set \(C\) has been fixed. We begin by constructing a rule set \(R_L\) which will be shown to generate the language \(L^C\) to which the algorithm converges. The algorithm need not necessarily converge to the same rule set, but will be shown to converge to an equivalent one. We define the set \(P^L = \{(c \in C : c(T) \in L) | T = (T_1, \ldots, T_l) \in S(L)\}\) of pre-rules and then define \(\hat{R}^L\) to be the unique rule set such that \(\hat{R}^L = \pi^{\hat{R}^L}(P^L)\). Finally, we define \(R^L\) to be the set of minimal elements in \(\hat{R}^L\). Because the language \(L\) is generated by a finite context free grammar, there is a bound on the degree of nodes of the trees in \(L\). From this, together with the fact that \(C\) is finite, it follows that the rule set \(R_L\) is finite.

Lemma 1. For any finite context set \(C\) and context free language \(L\), \(L \subseteq L(R^L)\).

Proof. It follows from the definition of the rule ordering that \(R^L\) and \(\hat{R}^L\) generate the same language. By the construction of \(\hat{R}^L\), for every tree \(T\), \(\{c \in C : c(T) \in L\} \subseteq \text{ctx}^{\hat{R}^L}(T)\). This proves the lemma, since \(*\) \(T\) \(T \in L\) implies \(* \in \text{ctx}^{\hat{R}^L}(T)\) and then, by definition, \(T \in L(\hat{R}^L) = L(R^L)\). \(\square\)

We write \(\text{ctx}^L\) for the function \(\text{ctx}^{R^L}\). For every rule \(R = (y|x_1, \ldots, x_l)_d \in R^L\) and for every \(c \in y\), we define the representative class \(CL(R, c) = \{S \in L : S = c((T_1, \ldots, T_l)) \in L, \text{ctx}^L(T_i) = x_i, 1 \leq i \leq l, \text{depth}((T_1, \ldots, T_l)) = d\}\). There are finitely many such classes. We call a sample \(\hat{S} = \{S_i\}_{i=1}^\infty\) class fat, if it contains infinitely many trees from \(CL(R, c)\) for every such class. To fulfill this condition, trees from each class may be repeated.

The only condition we impose on the language sample is that it be class fat. Since the algorithm can simply remember all examples presented to it and repeat the calculations on them as needed, the class fat sample requirement can be replaced, at the expense of increased memory and time resources, by the requirement that at least one tree from each class appear in the sample. This last condition is even weaker than the standard assumption that all sentences
Lemma 2. For any context free language \( L \), any finite context set \( C \), any class \( \bar{S} \) fat sample of \( L \) and for any \( 0 \leq k \) there is a number \( N_k \) such that for every \( N_k \leq n \), \( R_{k+1} = R_k^\infty \) and for every tree \( T \), \( \text{ctx}^n_k(T) = \text{ctx}_k^\infty(T) \).

Proof. By induction on \( k \). For \( k = 0 \), the claim is immediate from the definitions. We now assume the claim for \( k \) and prove it for \( k + 1 \). Let \( N_k < n \) and let \( R = \{ y \mid x_1, \ldots, x_l \} \in R_k^d \). There exists some \( S = \langle S_1, \ldots, S_l \rangle \in \mathcal{S}(L) \) of depth \( d \leq k + 1 \) such that \( x_i = \text{ctx}^n(S_i) \) \( (1 \leq i \leq l) \) and \( y \subseteq \{ c \in C \mid c(S) \in L \} \). Since \( \text{depth}(S) \leq k + 1 \) and by the induction hypothesis, \( x_i = \text{ctx}^n(S_i) = \text{ctx}^n_k(S_i) = \text{ctx}^n_k(S_i) = \text{ctx}^L(S_i) \) \( (1 \leq i \leq l) \). Therefore, \( \{ c \in C \mid c(S) \in L \} \subseteq R_k^L \) and it follows that there is a rule \( R' \in R_k^L \) such that \( R' \subseteq R \). As this is true for any rule \( R \in R_k^L \), \( \text{ctx}^n_{k+1}(T) \leq \text{ctx}^n_k(T) \) for any tree \( T \).

Next, we show that for every tree \( T \) there is an \( n(T) \) such that for any \( n(T) \leq n \), \( \text{ctx}^n_{k+1}(T) = \text{ctx}^n_{k+1}(T) \). The proof is by induction on the depth of the tree \( T \). For \( T = \sigma \in \Sigma \) (tree of depth 0) the claim is immediate from the definitions. We assume the claim for \( d \) and prove it for a tree \( T = \langle T_1, \ldots, T_l \rangle \) of depth \( d + 1 \). It is immediate from the way the algorithm works and the induction hypothesis on \( k \) that if \( N_k \leq n \) and \( R \in R_k^L \) then there is a rule \( R' \in R_k^L \) such that \( R' \subseteq R \). Therefore, for every tree \( T \), the sequence \( \{ \text{ctx}^n_{k+1}(T) \} \) is monotonically increasing (in \( \sigma \) ). It remains to show that if \( c \in \text{ctx}^n_{k+1}(T) \) then there is an \( N_k \leq n \) such that \( c \in \text{ctx}^n_{k+1}(T) \). By the class fat sample assumption, since \( c \in \text{ctx}^L_{k+1}(T) \), there must be a tree \( S = \langle S_1, \ldots, S_l \rangle \in \mathcal{S}(L) \) such that \( c(S) \in L \), \( \text{depth}(S) \leq \min(k + 1, d + 1) \), \( \text{ctx}^L(S_i) \leq \text{ctx}^L_{k+1}(T_i) \) \( (1 \leq i \leq l) \) and there is some \( N_k, N(T_1), \ldots, N(T_l) < n \) such that at step \( n \), the algorithm tries to add the pre-rule \( \{ c \} | S \rangle \). Therefore, after this step, there is \( R \in R_k^L \) such that \( R \subseteq \{ \{ c \} | S \rangle \text{ctx}^n(S_1), \ldots, \text{ctx}^n(S_l) \rangle \text{depth}(S) \rangle \). By what has been shown in the previous paragraph, the assumptions on \( S \) and the induction hypothesis on \( d \), \( \text{ctx}^n(S_i) = \text{ctx}^n_{k+1}(S_i) \leq \text{ctx}^L_{k+1}(S_i) \leq \text{ctx}^n(S_i) \leq \text{ctx}^L_{k+1}(T_i) = \text{ctx}^n_{k+1}(T_i) \) and therefore the rule \( R \) applies in calculating \( \text{ctx}^n_{k+1}(T) \) and \( c \in \text{ctx}^n_{k+1}(T) \), which completes the proof of the claim.
Since there is a finite number of trees of depth at most $k + 1$, there is an $N$ such that for every $N \leq n$ and every tree $T$ of depth at most $k + 1$, $\text{ctx}^n_{k+1}(T) = \text{ctx}^{n+1}_{k+1}(T)$. For step $N < n$, if the algorithm tries to add a pre-rule $(y|S_1,\ldots,S_l)$ for $S = \langle S_1,\ldots,S_l \rangle$ of depth at most $k + 1$ then, since $\{c \in C : c(S) \in L\} \subseteq \text{ctx}^L(S) = \text{ctx}^{n+1}_{k+1}(S)$, $y \subseteq \text{ctx}^n_{k+1}(S) = \text{ctx}^n_{k+1}(S)$. Since also $|y| \leq 1$, there must be a rule $R \in R^k_{n+1}$ such that $R \leq \pi^R \langle (y|S_1,\ldots,S_l) \rangle$ and the pre-rule is not added. It follows that $R^k_{n+1} = R^N_{n+1}$ for any $N \leq n$. Since for every tree $T$ there is an $N(T)$ such that, for $N(T) \leq n$, $\text{ctx}^n_{k+1}(T) = \text{ctx}^n_{k+1}(T)$ and since the rules of level at most $k + 1$ do not change after step $N$, it follows that $N(T) \leq N$ for every tree $T$. Therefore, we can take $N_{k+1} = N$. \qed

Based on the convergence of levels we can now prove the convergence of the whole rule set.

**Theorem 1.** For any context free language $L$, any finite context set $C$ and any class fat sample $\bar{S}$ of $L$, the algorithm converges to a rule set $R$ such that $L \subseteq L(R^L) = L(\mathcal{R})$.

**Proof.** That $L \subseteq L(R^L)$ is a restatement of Lemma 1. We therefore show that $L(R^L) = L(\mathcal{R})$. Let $k$ be the level of the highest level rule in $R^L$. Using Lemma 2 we take $N = N_k$. Let $N \leq n$ and assume that at step $n + 1$, the algorithm attempts to add the pre-rule $P = (y|S_1,\ldots,S_l)$. Writing $S = \langle S_1,\ldots,S_l \rangle$ we have, by definition, that $y \subseteq \{c \in C : c(S) \in L\}$ and (as in the proof of Lemma 1) $y \subseteq \text{ctx}^n(S)$. By the choice of $k$ and Lemma 2, $y \subseteq \text{ctx}^n(S) = \text{ctx}^n_k(S) = \text{ctx}^n_k(S) \subseteq \text{ctx}^n(S)$. Therefore, since $|y|$ is either 0 or 1, there is a rule $R = (z|x_1,\ldots,x_i) \in R^n \text{ such that } y \subseteq z$ and $x_i \leq \text{ctx}^n(S_i)$ ($1 \leq i \leq l$) and $d \leq \text{depth}(S)$. In other words, $R \leq \pi^R(P)$ and the algorithm does not add the pre-rule. Therefore, the pre-rule set does not change. This shows that the rules set converges at step $N_k$. To complete the proof, it remains to prove that for any tree $T$, $\text{ctx}^{N_k}(T) = \text{ctx}^L(T)$. This follows immediately from Lemma 2, because there are finitely many rules in $R^{N_k}$ and applying Lemma 2 for the level of the highest level rule in $R^{N_k}$ gives the required equality (notice that $R^{N_k}$ may contain rules of level higher than $k$). \qed

Our next step is to prove that for any context free language $L$ there is a context set $C$ such that $L^C = L$. We will actually show that for every finite set of languages we can find a context set which makes them all learnable. From the proof it will immediately be clear that for every language $L$ there are many different context sets which guarantee convergence to $L$. As a result, an algorithm for finding the context set $C$ enjoys a considerable amount of flexibility.

**Theorem 2.** For any finite set set $\mathcal{L}$ of context free languages, there is a finite set of contexts $\mathcal{C}^\mathcal{L}$ such that for any finite set of contexts $\mathcal{C}^\mathcal{L} \subseteq \mathcal{C}$ and any $L \in \mathcal{L}$, $L^{\mathcal{C}} = L$.

**Proof.** It is enough to prove the theorem for $|\mathcal{L}| = 1$ since then we can take $\mathcal{C}^\mathcal{L} = \bigcup_{L \in \mathcal{L}} C(L)$. Fix a context free language $L$. For every tree $T$, we define the
set $C(T) = \{ c \in CT(\Sigma) : c(T) \in L \}$. Since the language $L$ is generated by a finite number of rules, the sub-trees $S(L)$ of $L$ can be assigned only finitely many different types by the context free grammar which generates $L$. Since two trees of the same type appear in the same contexts in the language $L$, there are only finitely many different sets $C(T)$.

We will show that for every tree $T$ of depth at least 1, $ctx^L(T) = C \cap C(T)$. This proves the theorem because trees of depth zero cannot be in any language and for trees of depth at least 1, $T \in L^L \iff T \in L(R^L) \iff * \in ctx^L(T) \iff T \in L$ and therefore $L^L = L$.

If $T = (T_1, \ldots, T_l) \in S(L)$ then, since $\{ c \in C : c(T) \in L \} = C \cap C(T)$, there is a rule $(C \cap C(T)|ctx^L(T_1), \ldots, ctx^L(T_l))_{\text{depth}(T)} \in R^L$ and it follows that $C \cap C(T) \subseteq ctx^L(T)$. If $T \notin S(L)$ then $C \cap C(T) \subseteq ctx^L(T)$ trivially, since $C(T) = \emptyset$.

It remains to show that $ctx^L(T) \subseteq C \cap C(T)$. Since, by definition, $ctx^L(T) \subseteq C$, we show (by induction on the depth of $T$) that $ctx^L(T) \subseteq C(T)$. The claim is immediate for a tree of depth 1, since in this case $T = (\sigma_1, \ldots, \sigma_l)$ where $\sigma_1, \ldots, \sigma_l \in \Sigma$ and there can be at most one rule in $R^L$ which matches this tree, namely, $(C \cap C(T)|\sigma_1, \ldots, \sigma_l)_1$. We now assume the claim for trees of depth $k$ and prove it for trees of depth $k + 1$.

Let $T = (T_1, \ldots, T_l)$ be a tree of depth $k + 1$ and let $c \in ctx^L(T)$. There must be a rule $(y|x_1, \ldots, x_i)_d \in R^L$ such that $c \in y$, $x_i \leq ctx^L(T_i)$ ($1 \leq i \leq l$) and $d \leq k + 1$. Therefore, there is a tree $S = (S_1, \ldots, S_l)$ of depth at most $k + 1$ such that $ctx^L(S_i) = x_i$ for $i = 1, \ldots, l$ and $y = C \cap C(S)$. For any $1 \leq i \leq l$, if depth$(S_i) = 0$ then $x_i = S_i = \sigma_i \in \Sigma$ and therefore also $ctx^L(T_i) = \sigma_i$ which implies that $T_i = S_i$. In the same way, if depth$(T_i) = 0$ then $T_i = S_i$. If $0 < \text{depth}(S_i)$ (and therefore also $0 < \text{depth}(T_i)$) it follows, using the induction hypothesis, that $C \cap C(S_i) = ctx^L(S_i) \subseteq ctx^L(T_i) = C \cap C(T_i)$. By the construction of $C$, this means that $C(S_i) \subseteq C(T_i)$ or, in other words, that $T_i$ can appear in $L$ in any context in which $S_i$ appears. This is, of course, true also for $S_i$ with depth$(S_i) = 0$ since then $S_i = T_i$. Since $c((S_1, \ldots, S_l)) \in L$ this implies that $c((T_1, \ldots, T_l)) \in L$ and repeating this argument we conclude that $c(T) = c((T_1, \ldots, T_l)) \in L$. Therefore, $c \in C(T)$.

The grammars produced by the algorithm are not context free grammars. However, Theorem 2 shows that for any context free grammar, the algorithm can generate a grammar which is strongly equivalent to it (that is, generates the same trees). It is also not too difficult to check that for any rule set $R$, $L(R)$ is a context free language. We omit the proof of this here.

4 Stabilizing to a Learnable Language

As we saw in the previous section, the learning algorithm always converges, but may over-generalize and converge to a language which contains the original
language. In this section we show that repeated learning (that is, each learner learning the language to which the previous learner converged) is bound (after a finite number of steps) to arrive at a language learnable by subsequent learners. It is not even necessary for every learner in the sequence to use the same context set $C$ and the context set can be some random function of the language being learned (as will be discussed in detail later on).

To prove this, we need some notation for the cycles of learning. First, we assume that we have a sequence of finite context sets $\bar{C} = \{C_n\}_{n=0}^\infty$. We then define recursively $L^{\bar{C}(n)} = \left( L^{\bar{C}(n-1)} \right)^{C_{n-1}}$ where $L^{\bar{C}(0)} = L$. When there is no risk of confusion, we omit the $\bar{C}$ superscript and simply write $L^{(n)}$. The language $L^{(n)}$ is the $n$'th generation language of $L$.

Since $L$ will usually be fixed, we (usually) omit it from the notation and write $R^{(n)}$ for $R^{L^{(n)}}$ and $ctx^{(n)}(T)$ for $ctx^{L^{(n)}}(T)$. Note that $R^{(n)}$ is the rule set learned (by the algorithm) from $L^{(n)}$ and that this rule set generates the language $L^{(n+1)}$.

So as to be able to compare context sets in different generations, we define $C_\infty = \bigcup_{0 \leq n} C_n$, which then allows us to define $C^{(n)}(T) = \{ c \in C_\infty : c(T) \in L^{(n)} \}$.

**Lemma 3.** For any context free language $L$, any context set sequence $\bar{C}$, any $0 \leq n$ and any parse tree $T$ with $1 \leq \text{depth}(T)$, $C^{(n)}(T) \cap C_n \subseteq ctx^{(n)}(T) \subseteq C^{(n+1)}(T) \cap C_n$.

**Proof.** The left inclusion is easy, since for $T = \langle T_1, \ldots, T_l \rangle \in S(L^{(n)})$, the rule $(C^{(n)}(T) \cap C_n)[ctx^{(n)}(T_1), \ldots, ctx^{(n)}(T_l)]_{\text{depth}(T)}$ is in $\hat{R}^{L^{(n)}}$.

We now prove the right inclusion. Let $c \in ctx^{(n)}(T)$. We show that $c(T) \in L^{(n+1)}$. Let $T = \langle T_1, \ldots, T_l \rangle$. Since $c \in ctx^{(n)}(T)$, there is a tree $S = \langle S_1, \ldots, S_l \rangle$ such that $c(S) \in L^{(n)}$, $ctx^{(n)}(S_i) \subseteq ctx^{(n)}(T_i)$ ($1 \leq i \leq l$) and $\text{depth}(S) \leq \text{depth}(T)$. This means that any rule which applies to the calculation of $ctx^{(n)}(S)$ also applies to the calculation of $ctx^{(n)}(T)$ and therefore $ctx^{(n)}(S) \subseteq ctx^{(n)}(T)$. Together with $\text{depth}(S) \leq \text{depth}(T)$ this implies that $T$ can appear in $L^{(n+1)}$ in any context in which $S$ can. Since $c(S) \in L^{(n)} \subseteq L^{(n+1)}$, we can conclude that $c(T) \in L^{(n+1)}$.\]

Our main assumption from now on is that $C_\infty$ is finite. In other words, while the context sets used by the algorithm may vary from generation to generation, they are all bounded by a common finite set.

We now define a language $L^\infty$ to which the language sequence $\{L^{(n)}\}_{0 \leq n}$ will be shown to converge. For every tree $T$ with $1 \leq \text{depth}(T)$, let $C^\infty(T) = \bigcup_{0 \leq n} C^{(n)}(T)$ and for $T = \sigma \in \Sigma$ (tree of depth 0) let $C^\infty(T) = \sigma$. Let $S^\infty(L) = S\left( \bigcup_{0 \leq n} L^{(n)} \right)$, that is, the set of all subtrees in all generations. The language $L^\infty$ is defined to be the language generated by the rule set $\hat{R}^\infty(L) = \{ (C^\infty(T), C^\infty(T_1), \ldots, C^\infty(T_l))_{\text{depth}(T)} : T = \langle T_1, \ldots, T_l \rangle \in S^\infty(L) \}$. We take $R^\infty(L)$ to be the set of minimal rules in $R^\infty(L)$. Clearly, $L^\infty = L(R^\infty(L))$. We write $ctx^\infty(T)$ for $ctx^{R^\infty(L)}(T)$. 

Theorem 3. For any context free language $L$ and any context set sequence $\tilde{C}$ such that $\tilde{C}_n$ is finite, there exists an $N$ such that for every $N \leq n$, $L^\infty = L^{(n)}$.

Proof. Let $T \in L^{(n)}$. Then, by Lemma 3, $* \in \text{ctx}^{(n-1)}(T) \subseteq C^{(n)}(T) \cap C_{n-1} \subseteq C^{\infty}(T)$. Therefore $* \in C^{\infty}(T)$. It is easy to check by induction that $C^{\infty}(T) \subseteq \text{ctx}^{\infty}(T)$ and therefore $T \in L^\infty$. This shows that $L^{(n)} \subseteq L^\infty$ for all $n$.

To complete the proof, we show that there exists an $N$ such that for every $N < n$, $L^\infty \subseteq L^{(n)}$. Since $\{L^{(n)}\}_{0 \leq n}$ is an increasing sequence of languages, the assumption that $C_n$ is finite, it follows that for every tree $T$ with $1 \leq \text{depth}(T)$ there is a number $N(T)$ such that for every $N(T) \leq n$, $C^{(n)}(T) = C^{\infty}(T)$. Since for every $d$ the number of trees of depth no greater than $d$ is finite, there is a number $N(d)$ such that for every tree $T \in S^{\infty}(L)$ with $\text{depth}(T) \leq d$ and for every $N(d) \leq n$, $T \in S(L^{(n)})$ and if $1 \leq \text{depth}(T)$ then also $C^{(n)}(T) = C^{\infty}(T)$.

We prove the claim by taking $N = N(k) + 1$ where $k$ is the maximal level of any rule in $\mathcal{R}^{\infty}(L)$.

Fix some $N \leq n$. We prove the claim by constructing a rule set $\mathcal{R}$ such that $L^\infty \subseteq L(\mathcal{R}) \subseteq L^{(n)}$. Let $\phi(x)$ be defined by $\phi(\sigma) = \sigma$ for $\sigma \in \Sigma$ and $\phi(y) = y \cap C_{n-1}$ for $y \subseteq CT(\Sigma)$. For a rule $R = (y|x_1, \ldots, x_i)_d$ we can then define $\phi(R) = (\phi(y)|\phi(x_1), \ldots, \phi(x_i))_d$. The rule set $\mathcal{R}$ is then defined by $\mathcal{R} = \{\phi(R) : R \in \mathcal{R}^{\infty}(L)\}$.

Let $R \in \mathcal{R}$, then there is a rule $R' \in \mathcal{R}^{\infty}(L)$ such that $R = \phi(R')$. By the choice of $k$, $R' = (C^{\infty}(S)|C^{\infty}(S_1), \ldots, C^{\infty}(S_l))$ for some $S = (S_1, \ldots, S_l) \in S^{\infty}(L)$ of depth at most $k$. By the choice of $N$, $S \in S(L^{(n-1)})$ and $\{c \in C_{n-1} : c(S) \in L^{(n-1)}\} = \phi(C^{\infty}(S))$. By Lemma 3 together with the choice of $N$, $\text{ctx}^{(n-1)}(S_i) = \phi(C^{\infty}(S_i))$ ($1 \leq i \leq l$). Therefore, $R \in \mathcal{R}^{L^{(n-1)}}$. This shows that $L(\mathcal{R}) \subseteq L(\mathcal{R}^{(n-1)}) = L^{(n)}$.

We now show that for every tree $T$, $\phi(\text{ctx}^{\infty}(T)) \subseteq \text{ctx}^{\infty}(T)$. This proves that $L^\infty \subseteq L(\mathcal{R})$ because $T \in L^\infty \iff * \in \text{ctx}^{\infty}(T) \iff * \in \text{ctx}^{\infty}(T) \iff T \in L(\mathcal{R})$. We prove the claim by induction on the depth of $T$. For $T = \sigma \in \Sigma$ (depth 0) this is immediate from the definitions. We assume now the claim for $d$ and let $T = (T_1, \ldots, T_l)$ be a tree of depth $d+1$. Let $c \in \phi(\text{ctx}^{\infty}(T))$. There must be a rule $R = (y|x_1, \ldots, x_i)_d \in \mathcal{R}^{\infty}(L)$ ($j \leq d + 1$) such that $c \in y$ and $x_i \leq \text{ctx}^{\infty}(T_i)$ ($1 \leq i \leq l$). Since $\phi(R) \in \mathcal{R}$ and by the induction hypothesis $\phi(x_i) \leq \phi(\text{ctx}^{\infty}(T_i)) \leq \text{ctx}^{\infty}(T_i)$ ($1 \leq i \leq l$), it follows that the rule $\phi(R)$ is used in calculating $\text{ctx}^{\infty}(T)$ and therefore $c \in \phi(y) \subseteq \text{ctx}^{\infty}(T)$. □

We wish to apply this theorem in the following setting. For every context free language $L$ there is a random variable $X(L)$ which selects a context set for $L$ (that is, has values in $2^{CT(\Sigma)}$). We assume that there exists a finite context set $\tilde{C}_\text{max}$ such that for every $L$, $P(X(L) \subseteq \tilde{C}_\text{max}) = 1$. In this case, beginning with any context free language $L$ over $\Sigma$, we can generate the random sequence of languages $\{L^{(n)}\}_{0 \leq n}$ where $\tilde{C}$ is defined by $\tilde{C}_n = X(L^{(n)})$. With probability 1, Theorem 3 is applicable to the sequence generated in this way and we get that the language sequence converges to a language $L^\infty$, that is, there is an $N$ such that for every $N \leq n$, $L^{(n)} = L^\infty$. This means that $L^\infty$ is learnable.
using any $\mathcal{C}_n$ with $N \leq n$. Every such $\mathcal{C}_n$ was generated by the random variable $X(L^{(n)}) = X(L^\infty)$. Moreover, since there are infinitely many such $n$, it follows that, with probability 1, for any $\mathcal{C}$ such that $P(X(L^\infty) = \mathcal{C}) > 0$, $\mathcal{C} = \mathcal{C}_n$ for some $N \leq n$. Therefore, with probability 1, $L^\infty$ is learnable by any context to which $X(L^\infty)$ gives a non-zero probability. This means that if the learner generates the context set used by the algorithm based on the random variable $X$, the language sequence eventually converges to a language which is learnable by this learner with probability 1. In the next section we will see an example of such a learning strategy.

5 Choosing the Context Set

One natural way for the learner to choose the context set to be used with the learning algorithm is to take those contexts which appear most frequently in the language sample. Since for every language there are many different context sets which make it learnable, the algorithm is not too sensitive to moderate (unavoidable) changes in the frequencies and to the exact definition of “most frequent structures”.

One reason for choosing the most frequent structures is that the more frequent the contexts in the context set are, the sooner will the algorithm see all the examples it needs to see in order to converge (since the interesting examples are those in which contexts from the chosen context set appear). We can imagine that some learners may prefer quick convergence even at the expense of inaccuracy of learning (over-generalization). Another reason for taking the most frequent contexts is that it is reasonable to assume that in most settings, the most frequent contexts depend not only on the language structure (grammar) but also on language usage. The influence of language usage on the chosen context set is then described by a random variable $X(L)$. This random variable depends, indeed, on the language being used. However, we assume that even if the language changes (over the generations) some common properties of language usage remain fixed. For example, since the size of a context is proportional to the number of words in the sentence in which it appears and since this directly influences the semantics of the sentence, it seems reasonable to assume that there is a bound on the size of the most frequent contexts, regardless of the specific grammar being used. In this way we satisfy the condition $P(X(L) \subseteq \mathcal{C}_{max}) = 1$ of the previous section for all languages $L$. As we saw before, this condition guarantees that, with probability 1, the language sequence will eventually converge to a language which is learnable with probability 1.

It is possible to come up with different variants of “the most frequent contexts” approach outlined here. The discussion above would apply to any of them. We here present just one, very simple method which is based on the frequencies of contexts in a predetermined initial segment of the sample. The algorithm selects the $k$ most frequent contexts in the first $n$ examples or, alternatively, can take all those contexts which appeared at least $k$ times in the first $n$ examples.
This method is certain to construct a finite context set and to stabilize to a constant set after a finite number of steps.

6 Conclusion

The algorithm presented in this paper is extremely simple and natural. It essentially amounts to reading off rules from examples and discarding rules when they become redundant (non-minimal). This simplicity allows us to distinguish those factors essential to the correct functioning of the algorithm from those details which may be more freely modified. What is essential to the working of the algorithm is the use of a finite set of structures (the context set), the use of a grammar based on a partial order and the use of rules already inferred by the algorithm to infer additional rules (through the \( \text{ctx} \) function). Since inferred rules may initially be incorrect, the rules are layered to ensure correct convergence. Other details of the algorithm can be easily modified without changing the basic results. For example, the use of full contexts can be replaced by sub-structures of those contexts, sub-structures of the parse trees (equipped with an appropriate ordering) or even by information which is external to the sentence structure itself (such as semantic information). All one has to ensure is that a large enough variety of structures is available for the algorithm to be able to distinguish trees of different types (as in Theorem 2). Different choices of such structures can be made, without any significant effect on the analysis given here. In addition, different methods for determining the finite structure set (the context set given as a parameter) can be devised. While all these different variants have the same basic convergence properties, they may differ greatly in their rate of convergence, extent of over-generalization, computational complexity and applicability to actual learning situations.

References

Left Aligned Grammars – Identifying a class of Context-free Grammar in the limit from Positive Data

Bradford Starkie
Telstra Research Laboratories 770 Blackburn Road Clayton
Melbourne Victoria Australia 3127
Brad.Starkie@telstra.com
http://www.cs.mq.edu.au/~bstarkie

Abstract. This paper introduces a class of context-free grammar known as left-aligned grammars, and an algorithm for inferring left-aligned grammars from unlabelled positive examples. The paper includes a proof sketch that the algorithm can identify any given left-aligned grammar in the limit using only unlabelled positive examples from that grammar. Given any set of example sentences, a left-aligned grammar exists that generates those sentences.

1 Introduction

The term “Identification in the limit” [1] refers to a learning scenario where a learner is supplied an infinite stream of example sentences generated according to a grammar G and the learner must identify G. Identification in the limit does not require the learner to know when it has successfully identified the grammar. In his original work Gold [1] examined the Chomsky hierarchy of languages and proved that with the exception of grammars containing a finite number of phrases, no class of languages in the Chomsky Hierarchy can be identified in the limit from arbitrarily presented sets of sentences.

Gold’s findings contradicted the findings of psycholinguists that children are rarely informed of their grammatical errors, yet children do eventually learn natural languages. To reconcile this contradiction Gold suggested three possibilities. One of these possibilities was that even if the general class of regular and context-free languages could not be identified in the limit from arbitrarily presented text, there might be ways in which these classes of language could be restricted such that they could be identified in the limit. In subsequent work Angluin [2] described the conditions under which nonempty recursive formal languages could be identifiable in the limit from positive data. Angluin also described some classes of language that could be identified in the limit, the most notable of which was the K-Reversable class of regular language [3]. Sakakibara [4] extended the work of Angluin and defined a normal form of context-free languages known as k-reversible context-free grammars [5] that can be inferred from positive unlabelled derivation trees. An unlabelled derivation tree is a parse tree in which the nonterminal names attached to edges in the tree are unknown. Thus this algorithm requires some prior knowledge of the structure of the target grammar. Yokomori [6] introduced a class of context-free grammars that
could be identified in the limit from unlabelled positive data known as “very simple
grammars” but it is not guaranteed that a “very simple grammar” exists that can
generate any arbitrary set of sentences.

De la Higuera and Oncina [7] also defined a class of context-free language that can
be identified in the limit from positive unlabelled data, known as deterministic linear
languages. The class of deterministic linear languages have certain properties that
make them more restrictive than the class described in this document. For instance in
deterministic linear grammars each rule can have only one nonterminal (i.e. the
grammar is linear) and each sentence can only have one parse tree assigned to it (i.e.
the grammar is deterministic), Left-aligned grammars do not have these restrictions,
but have other restrictions, most notably the property of universal substitutability.
Both classes of language are subsets of the LL(1) class of language.

Van Zaane [8] introduced a new unsupervised learning framework known as
alignment based learning that is based upon the alignment of sentences. The technique
involves the alignment of pairs of sentences in a corpus. Sentence pairs are
partitioned into substrings that are common and substrings that are not. An assumption
of the technique is that the common substrings are generated by common rules, and
the portions of the sentences that are not common can be represented by rules that are
interchangeable. Alignment based learning suffers from a series of problems. The first
of these problems is that two strings can often be aligned multiple ways and selecting
the correct alignment to identify constituents is nondeterministic. Secondly the
application of alignment-based learning may result in overlapping constituents.
Thirdly it is not guaranteed that substrings used interchangeably in one part of the
language can be interchanged everywhere. Therefore alignment based learning cannot
be used to identify all context free languages in the limit.

This paper proves that a class of context free grammar, known as left aligned
grammars, can be identified in the limit from positive examples using a form of
alignment-based learning. In contrast to reversible context-free grammars, left aligned
grammars can be identified in the limit from unlabelled positive examples. Firstly I
will introduce the left-aligned class of grammars, followed by an algorithm for
inferring left-aligned grammars. I will then introduce a proof sketch that proves that
the class of left-aligned grammars can be identified in the limit from positive examples
using the left-alignment algorithm.

2 Definitions

A context free grammar can be represented by a 4-tuple \( G = (\mathit{N}, \mathit{\Sigma}, \mathit{T}, \mathit{S}) \) where \( \mathit{N} \) is
the alphabet of non-terminal symbols; \( \mathit{\Sigma} \) is the alphabet of terminal symbols such that
\( \mathit{N} \cap \mathit{\Sigma} = \{ \} \); \( \mathit{T} \) is a finite set of productions \( \mathit{P} \) of the form \( \mathit{L} \rightarrow \mathit{S}_\mathit{L}, \mathit{S}_\mathit{R}, \mathit{S}_\mathit{L} \mathit{S}_\mathit{R} \) where \( \mathit{L}, \mathit{S}_\mathit{L}, \mathit{S}_\mathit{R} \in \mathit{N} \) and
each \( \mathit{L} \in (\mathit{N} \cup \mathit{\Sigma}) \); and \( \mathit{S} \) is a special non-terminal called the start symbol that represents
all of the sentences generated by \( G \). In this document I will use lowercase symbols to
represent terminal symbols and uppercase symbols to represent nonterminal symbols.
Italicized uppercase symbols will represent a sequence of zero or more symbols and
bold italicized symbols will represent sequences of one of more symbols. In addition I
will use the symbol \( \mathit{L} \rightarrow \mathit{S}\mathit{S} \rightarrow \mathit{S}_\mathit{L} \mathit{S}_\mathit{R} \) to represent a constituent which is the application of one of
more rules to expand a nonterminal. For instance if a grammar contains the rules
\( \mathit{S} \rightarrow \mathit{a} \mathit{b}, \mathit{a} \mathit{b} \mathit{c} \mathit{d} \rightarrow \mathit{S} \mathit{S} \mathit{c} \mathit{d} \) then one expansion of \( \mathit{S} \) is \( \mathit{a} \mathit{b} \mathit{b} \mathit{c} \mathit{d} \mathit{b} \mathit{c} \mathit{d} \), i.e. \( \mathit{S} \rightarrow \mathit{a} \mathit{a} \mathit{b} \mathit{b} \)
3 Left-Aligned Grammars

I will now describe a class of context-free grammar known as the left-aligned class of grammars that can be identified in the limit from positive data, using a alignment based learning algorithm known as the left alignment algorithm. Given any set of sentences W generated from a left-aligned grammar G:

- The alignments between pairs of sentences in W that should be used to yield the rules of G is known and
- Given any set of candidate constituents derived from aligning pairs of sentences within W, the constituents of sentences within W can be determined deterministically, provided W contains a set of sentences known as a characteristic set of G.

The first of these properties is derived from the fact that left-aligned grammars can be parsed by a non-backtracking top-down parser known as a remainder parser, while the second property is derived from the fact that left-aligned grammars can be parsed by a non-backtracking bottom-up parser known as a left simplex parser. These parsers along with the grammars that can be parsed by them will now be introduced prior to formally introducing the class of left-aligned grammars.

3.1 Remainder Grammars

I will now introduce a class of grammar I will call remainder grammars. The definition of a remainder grammar requires a function called FOLLOW_SYM(X).

**Definition.** If X is a nonterminal of a grammar G and R is a rule of G and G does not have any productions of length zero then FOLLOW_SYM(X,R) is defined such that

- If there exists a rule of the form “A → B X C” other than R, then FOLLOW_SYM(X,R) includes the symbol C
- If there exists a rule of the form “A → B X” other than R, then FOLLOW_SYM(X,R) includes all of the elements of FOLLOW_SYM(A,R)

**Definition.** *A free LL(1) grammar* [9] are context free grammars that have the property that for any two rules of G of the form “A → X” and “A → Y”, “X” and “Y” cannot both derive strings that begin with the same terminal symbol “a”.

**Definition.** A **remainder grammar** is a class of context-free grammar that is comprised of

1. An ε-free LL(1) grammar G₁ and
2. An optional left-recursive rule R₀ of the form “S → S Q” where Q is a sequence of symbols including nonterminals from G and any terminal symbols.
   In addition
3. If R₀ exists then the set FOLLOW_SYM(S, R₀) can only contain nonterminals, and for every nonterminal B in FOLLOW_SYM(S, R₀) there must be a rule of the form “B → Q B’”

**Theorem.** If there exists a remainder grammar R comprising of a LL(1) grammar G₁ and a left recursive rule R₀ of the form “S → S Q’” then the language defined by R is “L(G₁) Q’”.

**Proof.** If the symbol S appears in the right hand side of rule of G₁ then after S has been successfully parsed the sequence “Q’” can be accepted by the grammar. There are however two ways in which this can be parsed. The first way is to use R₀ but
alternatively the next symbol B on the parse stack will be a nonterminal from the set
\texttt{FOLLOW}$\_\text{SYM}(S, R_0)$, therefore the rule "$B \rightarrow Q^b B$" can be used instead.
Therefore all sentences in $L(R)$ except for those with a trailing $Q^b$ can be parsed by a
\texttt{LL(1)} parser using $G_1$.

As a result of theorem 1, a \texttt{LL(1)} parser can be modified to parse sentences generated
from a remainder grammar. I will refer to this modified parser as a remainder parser.
A detailed description of \texttt{LL(1)} parsers can be found in [9]. Although \texttt{LL(1)}
grammars are unambiguous, remainder grammars are not. \texttt{LL(1)} parsers use a stack that defines
either the next expected terminal or the next expected nonterminal to be expanded. At any
point in parsing a sequence of symbols can be constructed from the contents of the
parse stack that describes all of the sequences of symbols that can occur given the
symbols that have been processed. This sequence is the rightmost portion of left-sen-
tential form.
The algorithm for a remainder parser is identical to a \texttt{LL(1)} parser, except that the left
recursive rule $R_0$ is omitted when the \texttt{LL(1)} parse table is constructed, and if the
symbol "$S$", which is used to represent the end of a sentence, is encountered on the
parse stack and the remainder grammar contains a left recursive rule, a remainder
parser pushes the symbols from "$Q$" in reverse order on the stack, and continues
processing. In the same way that the parse stack of an \texttt{LL(1)} parser describes the
language that can be observed given the observed prefix of a sentence, an expression
which I will name a remainder expression can be extracted from the stack of a
remainder parser that defines all of the symbols that can occur given the prefix.
Expressed as a sequence of symbols from left to right, this string is the contents of the
parse stack from top to bottom, with the "$S$" symbol omitted if the remainder grammar
does not contain a left recursive rule, or with "$S$" symbol replaced by the symbol $Q^b$ if
the grammar does contain a left-recursive rule.

\textbf{Definition.} Given a prefix "C" of a constituent "$N_0 \Rightarrow C X_1"$ of a remainder
grammar $G = (N, \Sigma, \Delta, S)$ a \textbf{remainder expression} is a sequence of symbols from the set
$E = (N \cup \Sigma \cup Q^b)$, where $Q^b$ represents a sequence of zero or instances of $Q$. The
symbols of a remainder expression can be expanded to include all of the possible
symbols that can follow $C$.

\textbf{Theorem 2.} Given two expansions of a nonterminal $N_0$ contained within the
remainder grammar $G$, where both expansions contain a common prefix, i.e., "$N_0 \Rightarrow C X_1"$
and "$N_0 \Rightarrow C X_2"$ then the remainder expression that describes $X_1$ is identical to
the remainder expression that describes $X_2$ and that expression can be calculated
without reference to $X_1$ or $X_2$.

\textbf{Proof.} If a remainder parser is used to parse the sequence "$C X_1"$ using $N_0$ as the
start symbol, the stack of the remainder parser can be converted into a remainder
expression, just prior to passing the first symbol in $X_1$ to the parser. This remainder
expression is calculated without reference to $X_1$ and is therefore the same for the
sequences $X_1$ and $X_2$.

\textbf{Definition.} The remainder expression $R$ describing all symbols that can follow
a prefix $C$ when expanding the nonterminal $N_0$ such that "$N_0 \Rightarrow C R"$ can be classified
as either class A, B or C expressions, $R$ can be calculated using a remainder parser by
parsing the first $|C|$ words of "$C X_1"$. 
Class A expressions are of the form $N Q^b$ or $N$ where $N$ is a single nonterminal.

Class B expressions are of the form $X Q^b$ or $X$ where the last rule identified by the parser was of the form “$N \rightarrow C \_X$”

Class C expressions are of the form $X Q^b$ or $X$ where the last rule identified by the parser was not of the form “$N \rightarrow C \_X$”

3.2 Left Simplex Grammars

Definition: A left simplex parser is a parser that uses the ParseLEFT-SIMPLEX algorithm shown in figure 1.

![Figure 1. Algorithm used by a Left-simplex parser](image)

**Definition:** A left simplex grammar is a grammar that

1. contains at least two rules for every nonterminal, with the exception of the top level nonterminal that can have only a single rule,
2. defines a language whose sentences can be parsed with a left-simplex parser using that grammar and,
3. doesn’t contain any redundant rules, where redundant rules are rules that would not be used by a left-simplex parser to parse sentences generated by those rules.

A left simplex parser is the simplest shift/reduce parser that can be built and the parser performs a reduce operation whenever the right hand side of a rule appears on the top of the stack. An algorithm is described in [10] that can be applied to any context-free grammar to determine if it is left-simplex.

**Definition.** A dotted rule describes a partial expansion of a rule of a grammar. The dotted grammar rule “$N_0 \rightarrow C \_X$” shows a partial expansion of the rule “$N_0 \rightarrow C \_X$” where only the sequence $C$ has been expanded.

**Definition.** The simplex sentential form of a sequence of symbols $X$ contained within a grammar $G$ is the value of the stack after parsing $X$ using a left-simplex parser using $G$.

**Definition.** A shift/reduce conflict is a feature of a context free grammar, such that if it were to be used by a shift/reduce parser, the parser would have a choice of either shifting or reducing at a particular point in parsing a sentence.

**Theorem 3.**

Given a constituent of a left-simplex grammar $G$ of the form “$N_i \rightarrow A$” that uses only one instance of rule of the form “$N_i \rightarrow B$” then $B$ is the simplex sentential form of $A$. 
Proof. While parsing a sentence generated by a left-simplex grammar $G$, whose partial derivation tree includes the constituent \((S \ldots (N_0 A) \ldots)\)', the sequence \(A\)' will be reduced to \(B\)', and if the rule \(N_0 \rightarrow B\) doesn't exist a shift/reduce conflict will exist.

An important consequence of theorem 2 is that the right hand side of any rule of a left-simplex grammar cannot be contained within the right hand side of another rule. Similarly the way in which multiple constituents should be substituted into one another to yield the rules of a grammar is known specifically conflicts should be resolved left to right.

**Theorem 4** If a sequence of symbols $X$ appears on the right hand side of a rule of a left-simplex grammar $G$, e.g. \(N_0 \rightarrow X A X B\) then the simple sentential form of any expansion of $X$ is always $X$, even if $X$ is a left sentential form.

**Proof**: If this was not true then at least one sentence generated by $G$ could not be parsed by $G$ using a left-simplex grammar, and the language would therefore not be left-simplex.

### 3.3 Left Aligned Grammars

<table>
<thead>
<tr>
<th>Definition. A <strong>left aligned grammar</strong> is a context-free grammar with the following properties.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Left Aligned grammars are remainder grammars</td>
</tr>
<tr>
<td>2. Left Aligned grammars are left-simplex grammars</td>
</tr>
<tr>
<td>3. Every nonterminal with the exception of $S$ must appear at least once as the last symbol on the right hand side of some rule.</td>
</tr>
</tbody>
</table>

### 4 An algorithm for Inferring Left-Aligned Grammars.

This section of the document will describe an algorithm for inferring left-aligned grammars, which we will call the left-alignment algorithm.

**Definition. Unchunking** is a process of transforming a grammar to ensure that for each nonterminal $N$ at least two rules exist with $N$ on their left hand side.

The left alignment algorithm is as follows

1. Create a prefix tree acceptor from the training examples. Construct a regular grammar that represents the prefix tree acceptor will the exception, that if any state in the prefix tree acceptor is a final state, then do not construct any rules that represent symbols that are accepted after that state
2. Merge rules that have the same right hand sides and delete repeated rules, until no more modifications can be made.
3. Ensure that the right hand side of each rule is not contained within the right hand side of any other rule. If two overlapping candidate constituents exist, then select the leftmost of the two constituents.
4. Merge nonterminals to ensure that the right hand sides of rules are unique, and that no two rules with the same left hand side begin with the same symbol. Unchunk the grammar and repeat steps 2 to 4.
5. Parse the training examples using the inferred grammar. If a sentence cannot be parsed then add a rule of the form $S \rightarrow B$ where $B$ is the simplex sentential form of an unparsable sentence. This will add a left-recursive rule, if it is required.
6. Parse the training examples using a left-simplex parser and delete rules not used to parse the sentences. If there exists a non-terminal that has only one rule then unchunk that non-terminal.

7. Repeat steps 2, 3, 4 & 6 until no more modifications can be made to the grammar.

The left-alignment algorithm is guaranteed to terminate because within each iteration, either the number of rules is decreased or the length of the rules becomes shorter.

At first glance it may not be obvious that the left-alignment algorithm is performing alignment based learning. The construction of the prefix tree acceptor is a process of aligning all sentences from the left. This process is also crucial to eliminating undesirable rules, which are rules that do not exist in the target grammar. The use of the prefix tree acceptor is also considerably faster than aligning each rule individually.

```plaintext
Function: satisfy_LEFT-SIMPLEX_constraints(G, G', G1)
Inputs: a set of candidate rules G, a candidate rule C and a read/write reference to a grammar G'

length = |C|;
for(count = 0; count < length; count++)

    best_rule = [ ]; edge_found = false; best_finish = count;
for( each rule S2 of the form x1-> y where

    Y is a sequence of symbols contained in
    C beginning at candidate rule[count]) {
    if( (|S2|)+count > best_finish) {
        best_finish = |S2| + count;
        best_rule = S2; }
    }
if(edge_found){
    replace |best_rule| symbols of C with
    the left hand side of best_rule starting at
    C[count];
    count = -1; length = |C1; }
add C to G' unless G' already has a rule
with the same right hand side.
```

Figure 4. An algorithm to ensure that the right hand side of a rule is not contained in the right hand side of another rule.

Step 3 in the algorithm is to ensure that the right hand side of each rule is not contained within the right hand side of any other rule, consistent with theorem 2. To do this the handle pruning algorithm of figure 4 is used. The problem of overlapping constituents identified by Van Zaanen [8] is resolved using this algorithm.

Step 4 involves merging non-terminals to ensure that the left-simplex constraints 1 & 3 are satisfied. Specifically,

1. If two rules exist of the form “Nn → C Nj” and “Nn → C Nk” then the non-terminals Nj and Nk are merged.

2. If a rule exists of the form “Nn → Nj” then Nn and Nj are merged.

Step 6 of the algorithm involves parsing the training examples using a left-simplex parser, and deleting unused rules. After deleting redundant rules the grammar is unchunked.
5 Characteristic Sets of Examples of a Left Aligned Grammar

I will now introduce an algorithm that generates a characteristic set of example sentences for every left aligned grammar, such that when these examples are presented to the left alignment algorithm the rules of the grammar when expressed in its normal form will be learnt exactly. The algorithm for generating a characteristic set for any left aligned grammar is shown in figure 3. A characteristic set of a left-aligned grammar contains at most as many sentences as there are rules in the grammar.

```
Function create_characteristic_examples (G, T)
Inputs: a left aligned grammar G
Outputs: A set of characteristic phrases from G
1. Assign each rule a unique number
2. For each rule in G of the form “Nᵢ → X” expand each nonterminal in the sequence X using the function traverse_node;
3. Create a prefix C such that the grammar can generate the phrase “CX”. This is always possible because every nonterminal must appear at the end of the right hand side of a rule. All nonterminals Nᵢ in C are expanded using the function traverse_node,
4. Add “CX” to the characteristic set.
```

Function traverse_node(Nᵢ, G)
Inputs: a left aligned grammar G and a nonterminal symbol Nᵢ
Outputs: one expansion of Nᵢ
1. Let Ry be the rule with the smallest unique number of the form “Nᵢ → X” where X is a sequence of 1 or more symbols and Ry is not recursive w.r.t. nonterminal Nᵢ;
2. Expand all nonterminals in X using traverse_node;

**Figure 5. Algorithm to generate a characteristic set of example phrases of a left-aligned grammar.**

**Theorem 5.** When the left alignment algorithm is presented with a set of example sentences generated from a left aligned grammar G using the create_characteristic_examples algorithm, the left alignment algorithm generates the grammar G up to the naming of nonterminals.

**Proof.** For each pair of rules of the form “Nᵢ → Xᵢ” and “Nᵢ → Xⱼ” the function create_characteristic_examples creates two sentences of the form “CXᵢ” and “CXⱼ”. When these examples are presented to the left alignment algorithm, the algorithm will create the rules “S → CXᵢ”, “Xᵢ → Xⱼ” and “X → Xⱼ”.

For each nonterminal Nᵢ in G, the algorithm traverse_node will return a fixed sequence Eᵢ. This sequence is created by selecting one rule of the form “Nᵢ → Xᵢ” and expanding it using traverse_node, which is the same process used to create the sequence “Xᵢ” in the example “CXᵢ”, therefore it is guaranteed that the characteristic set will include two examples of the form “CEᵢ” and “CXᵢ”. When these examples are passed to the left alignment algorithm, the algorithm creates the two rules “Nᵢ → Eᵢ” and “Nᵢ → Xᵢ”, therefore it can be seen that for each nonterminal Nᵢ the algorithm will generate a rule of the form “Nᵢ → Eᵢ”. The algorithm ConstituentStructure will then replace every instance of Eᵢ by Nᵢ, resolving conflicts in manner consistent with theorem 2. The left recursive rule will be constructed during step 5 due to theorem 3. Therefore given a characteristic set of a left aligned grammar
G the left alignment algorithm will infer a grammar identical to G up to the naming of nonterminals.

6 Preventing Overgeneralization

**Definition.** A super-characteristic set of examples of a left-aligned grammar G is a set of example sentences that is the union of any characteristic set of G generated using the create_characteristic_examples algorithm plus any other set of phrases generated by G.

**Definition.** The rules created by aligning the first i words of an expansion “N_i → C X_j” are generated from a left-aligned grammar G where IC = i can be classified into one of the following classes of rules:

- **Class A rules.** These rules are created if the remainder expression describing X_j is a class A expression and the simplex sentential form of X_j is a single nonterminal N_i. Class A rules represent at least one expansion of a true nonterminal of G.

- **Class B rules.** These rules are created if the remainder expression describing X_j is a class B expression of the form “X Q^m” and the simplex sentential form of X_j is X. In this case, X represents the right corner of the right hand side of a rule.

- **Class C rules.** These rules are created if the remainder expression describing X_j is a class C expression of the form “X Q^m” and one expansion of X is X_j, i.e., (X) → X_j. In this case, X_j represents the right corners of the right hand side of several rules.

- **Class D rules.** All other rules are class D rules. If the remainder expression describing X_j is of the form “X Q^m” then X_j can only be generated by expanding the left-recursive rule.

**Definition.** Two rules “N_i → A” and “N_i → B” are said to be merged if N_i is merged with N_j.

**Theorem 6.** If a left-aligned grammar G_2 is created using steps 1-4 of the left-alignment algorithm and a super-characteristic set T of G_2 and those rules of G_2 not needed to parse T using a left simplex parser are removed from G_2, then only Class C rules of G_2 will be removed, also no class D rules will be added to the grammar. Therefore the resulting grammar will be identical to G up to the naming of nonterminals.

**Proof.** Due to theorem 5 and because T is a super-characteristic set it is known that G_2 will include the rules of G up to the naming of nonterminals. No class D rules will be added to the inferred grammar because the initial grammar is constructed using only transitions of the prefix tree acceptor that occur up to final states (step 1). Therefore each nonterminal in the initial grammar represents a finite sequence of nonterminals and terminals in the target grammar as described by a remainder expression. Because no class D rules are present these remainder expressions do not have any Q^m symbols at the end of them. The right hand side of rules should be reduced to their simplex sentential form (theorem 3). For class A & B rules the remainder expression will be the same as the simplex sentential form describing that rule. Rules will only be merged with other rules that represent the same remainder expression.

Because G_2 contains the rules of G, for all Class C rules of the form “N → A U” there will exist a shift/reduce conflict of the form “N → B A” & “N → A U.” This is because the sequence “A U” contains the right corners of several different
rules. Therefore during step 6 class C rules will never be used to parse the sequence “A U” if it appears at the end of sentence. Therefore class C rules will be deleted during step 6, or will be a benign addition to the grammar and will not cause overgeneralization.

Theorem 7. The class of left-aligned grammars is identifiable in the limit from positive unlabelled data.

Proof. Because a characteristic set T of example sentences can always be generated from any left-aligned grammar G and T is finite, then at some finite point the observed sentences of an infinite stream of unique example sentences from G will be a supercharacteristic set. Theorem 6 states that under these conditions the left alignment algorithm generates a grammar G’ that is identical to G up to the naming of nonterminals.

7 Example

Consider the following training examples: {left left right right, left right left right left, left right left right right, left right left right right right right}. These sentences are created from the following left-aligned grammar:

\{X_1 \rightarrow S \ X_1 \ X_1 \rightarrow right S \rightarrow left X_1 S \rightarrow S S\}. The first three example sentences form a characteristic set of the grammar. After step 1 the initial grammar is \{X_1 \rightarrow left X_2 X_1 \rightarrow right X_3 \rightarrow left right right right X_2 \rightarrow right right S \rightarrow left X_1\}.

It can be seen that this grammar is LL(1) but cannot generate all of the training examples. After step 4 the grammar becomes, \{X_1 \rightarrow S X_4 X_1 \rightarrow right S \rightarrow left X_1\}.

After step 5 the grammar becomes \{X_1 \rightarrow S X_5 X_1 \rightarrow right S \rightarrow left X_4 S \rightarrow S S, S \rightarrow S S S\}.

If the remaining steps of the algorithm are continued the algorithm converges to produce the target grammar.

8 Some Useful Properties of Left-Aligned Grammars

Theorem 8 Given any finite set of sentences a left-aligned grammar can be created that generates at least those sentences.

Proof The following grammar generates sentences of any length of words from a fixed vocabulary of size n in any order: “S \rightarrow S S”, “S \rightarrow x_1”, “S \rightarrow x_2”, …., “S \rightarrow x_n”. This grammar is known as the universal grammar is a left-aligned grammar, and a similar grammar can be constructed from any finite set of sentences, that generates those sentences.

8 Left-Simplex Grammars and The Chomsky Hierarchy

Theorem 9. Figure 6 below shows a Venn diagram showing the relationship between left-aligned Grammars, context free Grammars and regular Grammars.
Figure 6. Left-aligned Grammars and the Chomsky Hierarchy

Proof. The following context-free grammar cannot be converted to a left-aligned grammar.
\[ S \rightarrow a \ b \]  \[ S \rightarrow x \ a \ b \ c \]  
The following grammar is both regular and left-aligned.
\[ S \rightarrow a \ A \]  \[ A \rightarrow b \ A \]  \[ A \rightarrow c \]  
The following regular grammar cannot be converted to a left-aligned grammar.
\[ S \rightarrow a \ a \ a \ A \]  \[ A \rightarrow a \ A \]  \[ A \rightarrow b \]  
The following left-aligned grammar cannot be converted to a regular grammar.
\[ S \rightarrow a \ b \]  \[ S \rightarrow a \ S \ b \]  

9 Experimental Results

As part of an empirical evaluation 800 randomly generated left-aligned grammars were created. The left-alignment algorithm was then presented with both characteristic and supercharacteristic sets of sentences from these target grammars. In all cases the algorithm inferred the target grammars up to the naming of nonterminals.

The algorithm was also tested on a range of sentences generated from non left-aligned grammars. Although the algorithm inferred useful grammars from these examples, for large training sets of similar sentences the inferred grammar often converged to the universal grammar. The algorithm was also tested using non characteristic sets of sentences generated from left-aligned grammars. Although the algorithm is guaranteed to generate a left-aligned grammar when presented with a characteristic set, the inferred grammar is not guaranteed to be left-aligned when presented with arbitrary examples.

10 Conclusions and Further Research

This paper has proved that a class of context-free grammars known as left-aligned grammars can be identified in the limit using the left alignment algorithm. Unlike KL-reversible context-free grammars [4][5] left-aligned grammars can be identified from unlabelled positive examples and unlike very simple grammars [6] given any arbitrary set of sentences a left-aligned grammar exists that generates at least those sentences. The left aligned class of grammars can model a wide range of useful grammars including palindromes, parenthesis, grammars with long-term dependencies, the universal grammar, and grammars with finite cardinality. Left-aligned grammars do however have the property of universal substitutability, which stated informally means that if two constituents of a sentence are interchangeable in one context of a grammar they are always interchangeable. It is known that natural grammars such as English do not in general exhibit this property, although within small enough domains they may exhibit this property. However, there are a range of computational linguistic problems such as speech recognition for which even more simplistic grammars such as n-grams have proven useful. The overgeneralization that might be exhibited by left-aligned grammars that have been inferred from training examples generated from non-left-aligned grammars will be orders of magnitude less than for n-grams. Left-aligned grammars however have some favorable properties that they share in common with n-grams such as fast inference algorithms, and parsing algorithms that operate in time linear to the length of the sentence.
Work has begun on extending the left-alignment algorithm to infer attribute grammars that can be used to attach data structures to sentences using similar techniques to that described in [11] and [12]. It is believed that the new algorithm will be able to infer grammars with violations of universal substitutability. These violations are likely to be identifiable from training examples, if the addition of universal substitutability would result in a grammar inconsistent with the training data.

Acknowledgements

The research in this document has been undertaken as part of the requirements of a Ph.D. at Melbourne University, Melbourne Australia, and their support is hereby acknowledged. The permission of the Chief Technology Officer of Telstra Corporation Ltd. to publish this information is also hereby acknowledged.

References

Offspring-annotated probabilistic context-free grammars

Jose L. Verdú-Mas, Jorge Calera-Rubio, and Rafael C. Carrasco *

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante, E-03071 Alicante, Spain.
{verdu, calera, carrasco}@dlsi.ua.es

Abstract. This paper describes the application of a new model to learn probabilistic context-free grammars (PCFGs) from a treebank corpus. The model estimates the probabilities according to a generalized $k$-gram scheme for trees. It allows for faster parsing, decreases considerably the perplexity of the test samples and tends to give more structured and refined parses. In addition, it also allows several smoothing techniques such as backing-off or interpolation that are used to avoid assigning zero probability to any sentence.

1 Introduction

Context-free grammars may be considered to be the customary way of representing syntactical structure in natural language sentences. In many natural-language processing applications, obtaining the correct syntactical structure for a sentence is an important intermediate step before assigning an interpretation to it. But ambiguous parses are very common in real natural-language sentences (e.g., those longer than 15 words); the fact that many ambiguous parse examples in books sound a bit awkward is due to the fact that they involve short sentences [1, 2, p. 411]. Choosing the correct parse for a given sentence is a crucial task if one wants to interpret the meaning of the sentence, due to the principle of compositionality [3, p. 358], which states, informally, that the interpretation of a sentence is obtained by composing the meanings of its constituents according to the groupings defined by the parse tree.

A set of rather radical hypotheses as to how humans select the best parse tree [6] propose that a great deal of syntactic disambiguation may actually occur without the use of any semantic information; that is, just by selecting a preferred parse tree. It may be argued that the preference of a parse tree with respect to another is largely due to the relative frequencies with which those

* The authors wish to thank the Generalitat Valenciana project CTIDIB/2002/173 and the Spanish CICyT project TIC2000-1599 for supporting this work.

1 The principle of compositionality in natural language is akin to the concept of syntax-directed translation used in compiler construction [4, p. 25] and to the principles which inspire the syntactic transfer architecture used in machine translation systems [5].
choices have lead to a successful interpretation. This sets the ground for a family of techniques which use a probabilistic scoring of parses to the correct parse in each case.

Probabilistic scorings depend on parameters which are usually estimated from data, that is, from parsed text corpora such as the Penn Treebank [7]. The most straightforward approach is that of treebank grammars, [8]. Treebank grammars—which will be explained in detail below—are probabilistic context-free grammars in which the probability that a particular nonterminal is expanded according to a given rule is estimated as the relative frequency of that expansion by simply counting the number of times it appears in a manually-parsed corpus. This is the simplest probabilistic scoring scheme, and it is not without problems; we will show how a set of approximate models, which we will call offspring-annotated models, in which expansion probabilities are dependent on the future expansions of children, may be seen as a generalization of the classic k-gram models to the case of trees, and include treebank grammars as a special case; other models, such as Johnson’s [9] parent-annotated models (or more generally, ancestry annotated models) and IBM history-based grammars [2, p. 423],[10] or Bod’s and Scha’s Data-Oriented parsing [11] offer an alternate approach in which the probability of expansion of a given nonterminal is made dependent on the previous expansions. An interesting property of many of these models is that, even though they may be seen as context-dependent, they may still be easily rewritten as context-free models in terms of specialized versions of the original nonterminals.

This paper is organized as follows: section 2 proposes our generalization of the classic k-gram models to the case of trees, which is shown to be equivalent to having a specialized context-free grammar. A simplication of this model, called the offspring-annotated model or k = 3, for short, is also presented in that section. Experiments using this model for structural disambiguation are presented in section 4 and compared with other family-annotated models, and, finally, conclusions are given in section 5.

2 The model

Let $\Omega = \{\tau_1, \tau_2, \ldots, \tau_m\}$ a treebank, that is, a sample of parse trees.

For all $k > 0$ and for all trees $\tau = \sigma(t_1 \ldots t_m) \in \Omega$ we define the $k$-root of $\tau$ as the tree

$$r_k(\sigma(t_1 \ldots t_m)) = \begin{cases} \sigma & \text{if } k = 1 \\ \sigma(r_{k-1}(t_1) \ldots r_{k-1}(t_m)) & \text{otherwise} \end{cases} \quad (1)$$

The sets $f_k(t)$ of $k$-forks and $s_k(t)$ of $k$-subtrees are defined for all $k > 0$ as follows:

$$f_k(\sigma(t_1 \ldots t_m)) = \cup_{j=1}^m f_k(t_j) \cup \begin{cases} \emptyset & \text{if } 1 + \text{depth}(\sigma(t_1 \ldots t_m)) < k \\ r_k(\sigma(t_1 \ldots t_m)) & \text{otherwise} \end{cases} \quad (2)$$
where $\text{depth}(t)$ denotes the depth of the tree $t$, The depth of a single node tree is zero).

We define the treebank probabilistic $k$ testable grammar $G = (\mathcal{N}, \Sigma, \mathcal{S}, \mathcal{P})$ through:

- $\mathcal{N} = r_{k-1}(f_k(\Omega)) \cup s_{k-1}(\Omega) \cup \{\mathcal{S}\}$;
- $\Sigma$ is the set of lables in $\Omega$;
- $\mathcal{S}$ is the start symbol;
- $\mathcal{P} = \{(r, p(r)) \mid r \in R \land p(r) \in [0, 1]\}$ where $R \subset \mathcal{N} \times (\mathcal{N} \cup \Sigma)^+$ is a set of production rules (usually written as $A \rightarrow \alpha$, where $A \in \mathcal{N}$ and $\alpha \in (\mathcal{N} \cup \Sigma)^+$) and $p(r)$ is the emission probability associated with the rule $r$. The set $\mathcal{P}$ is built as follow:

- for every tree $t \in r_{k}(\Omega)$ add to $\mathcal{P}$ the rule $\mathcal{S} \rightarrow t$ with probability

\[
p(\mathcal{S} \rightarrow t) = \frac{\sum_{\tau \in \Omega} \delta_{t_{r_{k-1}(\tau)}}}{|\Omega|}
\]

(4)

where $\delta_{a,b} = 1$ if $a = b$ and zero otherwise;

- for every tree $\sigma(t_1 t_2 \ldots t_m) \in f_k(\Omega)$ add to $\mathcal{P}$ the rule $r_{k-1}(\sigma(t_1 t_2 \ldots t_m)) \rightarrow t_1 t_2 \ldots t_m$ with probability

\[
p(r_{k-1}(\sigma(t_1 t_2 \ldots t_m)) \rightarrow t_1 t_2 \ldots t_m) = \frac{\sum_{\tau \in \Omega} C(\sigma(t_1 t_2 \ldots t_m), \tau)}{\sum_{\tau \in \Omega} C(r_{k-1}(\sigma(t_1 t_2 \ldots t_m)), \tau)}
\]

(5)

Here $C(t, \tau)$ counts the number of times that the fork $t$ appers in the tree $\tau$;

- for every tree $\sigma(t_1 t_2 \ldots t_m) \in s_k(\Omega)$ add to $\mathcal{P}$ the rule $\sigma(t_1 t_2 \ldots t_m) \rightarrow t_1 t_2 \ldots t_m$ with probability

\[
p(\sigma(t_1 t_2 \ldots t_m) \rightarrow t_1 t_2 \ldots t_m) = 1
\]

(6)

Defined in this way, these probabilities satisfy the normalization constraint

\[
\text{for each } A \in \mathcal{N}: \sum_{\alpha: A \rightarrow \alpha \in \mathcal{P}} p(A \rightarrow \alpha) = 1
\]

(7)

and the consistency constraint. PCFGs estimated from treebanks using the relative frequency estimator always satisfy those constraints [12] [13].

Note that in this kind of models, the expansion probability for a given node is computed as a function of the subtree of depth $k - 2$ that the node generates, i.e., every non-terminal symbol stores a subtree of depth $k - 2$. In the particular case $k = 2$, only the label of the node is taken into account (this is analogous to
the standard bigram model for strings) and the model coincides with the simple rule-counting approach used in treebank grammars.

However, in the case $k = 3$, we get a child-annotated model, that is, non-terminal symbols $\sigma(\sigma_1, \sigma_2 \ldots \sigma_m)$ are defined by:

- the node label $\sigma$,
- the number $m$ of descendents (if any) and
- the labels in the descendents $\sigma_1, \sigma_2, \ldots, \sigma_m$ (if any) and their ordering.

As an illustration, consider a very simple sample with only the tree in the figure 1. If we choose $k = 2$, then

- $r_1(S(NP \ VP)) = S$,
- $f_2(S(NP \ VP)) = \{S(NP \ VP), NP(N), VP(V \ NP), NP(NP PP), PP(PNP)\}$
- $\delta_1(S(NP \ VP)) = \emptyset$

and the CFG is

$$G^{[2]} = (\{S, NP, VP, PP\}, \{N, V, P\}, S, P),$$

with $P$ containing the rules

\[
\begin{align*}
S & \rightarrow NP \ VP \\
NP & \rightarrow NP \ PP \\
NP & \rightarrow N \\
VP & \rightarrow VP \ PP \\
VP & \rightarrow V \ NP \\
PP & \rightarrow P \ NP
\end{align*}
\]

However, for $k = 3$ we obtain

- $r_2(S(NP \ VP)) = S(NP \ VP)$
- $f_3(S(NP \ VP)) = \{S(NP(N) \ VP(V \ NP)), VP(V \ NP(NP PP)), NP(NP PP(P \ NP)), PP(PNP(N))\}$
- $\delta_2(S(NP \ VP)) = \{NP(N)\}$

and the CFG is

$$G^{[3]} = (\{S, S(NP \ VP), NP(N), VP(V \ NP), NP(NP PP), PP(PNP)\}, \{N, V, P\}, S, P),$$

with $P$ containing the rules

\[
\begin{align*}
S & \rightarrow S(NP \ VP) \\
S(NP \ VP) & \rightarrow NP(N) \ VP(V \ NP) \\
VP(V \ NP) & \rightarrow V \ NP(NP PP) \\
NP(NP PP) & \rightarrow NP(N) \ PP(P \ NP) \\
PP(PNP) & \rightarrow P \ NP(N) \\
NP(N) & \rightarrow N
\end{align*}
\]
For comparison, if one uses a parent-annotated version of the grammar (following Johnson [9]), one gets the following rules \(^2\) (where the superindex is the parent’s label).

\[
\begin{align*}
S & \rightarrow ^S\text{NP} ^S\text{VP} \\
^S\text{NP} & \rightarrow N \\
^S\text{VP} & \rightarrow V ^VP\text{NP} \\
^VP\text{NP} & \rightarrow ^NP\text{NP} ^NP\text{PP} \\
^NP\text{NP} & \rightarrow N \\
^NP\text{PP} & \rightarrow P ^PP\text{NP} \\
^PP\text{NP} & \rightarrow N
\end{align*}
\]

\[\text{Fig. 1. A sample parse tree}\]

3 Smoothing

As will be seen in the experiments section, although the child-annotated models yield a good performance (in terms of both parsing and speed), their rules are very specific and, then, some events (subtrees, in our case) in the test set are not present in the training data, yielding zero probabilities. Due to data sparseness, this happens often in practice. However, this is not the case of the unannotated model, \(k = 2\), with total coverage but with worse performance. This justifies the need for smoothing methods.

In the following, three smoothing techniques are described. Two of them are well known: linear interpolation and tree-level back-off. In addition, we introduce a new smoothing technique: rule-level back-off.

\(^2\) As will be seen in section 4, parent-annotated grammars usually have less parameters than child-annotated grammars, contrary to what this example may suggest.
3.1 Linear interpolation

Smoothing through *linear interpolation* [14] is performed by computing the probability of events as a weighted average of the probabilities given by different models. For instance, the smoothed probability of a $k = 3$ model could be computed as a weighted average of the probability given by the model itself, and that given by the $k = 2$ model, that is, the probability of a tree $t$ in the interpolated model is

$$p(t) = \lambda p_3(t) + (1 - \lambda)p_2(t)$$

where $p_3(t)$ and $p_2(t)$ are the probabilities of the tree $t$ in, respectively, the model $k = 3$ and $k = 2$.

In our experiments, the mixing parameter $\lambda \in [0, 1]$ will be chosen to minimize the perplexity of the sample.

3.2 Tree-level back-off

Back-off allows one to combine information from different models. In our case, the highest order model such that the probability of the event is greater than zero is selected. Some care has to be taken in order to preserve normalization.

$$\begin{cases} (1 - \lambda)p_3(t) & \text{if } p_3(t) > 0 \\ \lambda p_2(t) & \text{if } p_3(t) = 0 \end{cases}$$

where

$$\lambda = \frac{\sum_{t; p_3(t) = 0} p_2(t)}{\sum_{t} p_2(t)}.$$  

In our experiments, we will assume that a $\lambda$ may be found such that no sentence $s$ in the test set having a parse-tree with $p_3(t) > 0$ has another parse tree $t'$ with $p(t') > p(t)$. Therefore, $p_2$’s will only be computed for trees with $p_3(t) = 0$. This leads to the following efficient parsing strategy: $k = 2$ (unannotated, fast) parsing is not launched if the $k = 3$ (annotated, fast) parser returns a tree, because the $k = 3$ tree will win out all $k = 2$ trees; therefore, for parsing purposes, the actual value of $\lambda$ is irrelevant.

3.3 Rule-level back-off

Our back-off model builds a new PCFG from the rules of the tree-$k$-grammar models and adding new rules which allow to switch among those models. In particular, the new PCFG consists of three different kinds of rules:

1. $k = 3$ rules with modified probability,
2. back-off rules that allow to switch to the lower model, and,
3. modified $k = 2$ rules to switch-back to the higher model.

This is done as follows:
1. Add the rules of the $k = 3$ model with probability:

$$p(X(X_1 X_2 \ldots X_m) \rightarrow \alpha) = p_3(X(X_1 X_2 \ldots X_m) \rightarrow \alpha)(1 - \lambda(X(X_1 X_2 \ldots X_m)))$$

(11)

2. For each non-terminal symbol of the $k = 3$ model, $X(X_1 X_2 \ldots X_m)$, add a back-off rule $X(X_1 X_2 \ldots X_m) \rightarrow X_1 X_2 \ldots X_m$ with probability:

$$p(X(X_1 X_2 \ldots X_m) \rightarrow X_1 X_2 \ldots X_m) = \frac{\lambda(X(X_1 X_2 \ldots X_m))}{\Lambda(A(X_1 X_2 \ldots X_m))}$$

(12)

where

$$\Lambda(A(X_1 X_2 \ldots X_m)) = 1 - \sum_{X(X_1 X_2 \ldots X_m) \rightarrow \alpha X_1 \alpha X_2 \ldots \alpha X_m \in R^{|X|}} \prod_{i=1}^{m} p_2(X_i \rightarrow \alpha X_i)$$

(13)

3. Add the $k = 2$ rules as unary rules, that is, if the rule is $X \rightarrow X_1 X_2 \ldots X_m$ then, add $X \rightarrow X(X_1 X_2 \ldots X_m)$ with probability:

$$p(X \rightarrow X(X_1 X_2 \ldots X_m)) = p_2(X \rightarrow X_1 X_2 \ldots X_m)$$

(14)

The resulting grammar is normalized provided that parses of the form $X(X_1 X_2 \ldots X_m) \rightarrow X_1 X_2 \ldots X_m \rightarrow \alpha X_1 \alpha X_2 \ldots \alpha X_m$ are assigned a zero probability if $X(X_1 X_2 \ldots X_m) \rightarrow \alpha X_1 \alpha X_2 \ldots \alpha X_m$ exists in the grammar; this must be checked in parse time.

4 Experiments

4.1 General conditions

We have performed a series of experiments to assess the structural disambiguation performance of offspring-annotated models as compared to standard treebank grammars, that is, to compare their relative ability for selecting the best parse tree. To better put these comparisons in context, we have also evaluated Johnson’s [9] parent annotation scheme. To build training corpora and test sets of parse trees, we have used English parse trees from the Penn Treebank, release 3, with small, basically structure-preserving modifications:

- insertion of a root node (ROOT) in all sentences, (as in Charniak [8]) to encompass the sentence and final periods, etc.;
- removal of nonsyntactic annotations (prefixes and suffixes) from constituent labels (for instance, NP-SBJ is reduced to NP);
- removal of empty constituents; and
- collapse of single-child nodes with the parent node when they have the same label.
In all experiments the training corpus, consisted of all of the trees (41,532) in sections 02 to 22 of the Wall Street Journal portion of Penn Treebank, modified as above. This gives a total number of more than 600,000 subtrees. The test set contained all sentences in section 23 having less than 40 words.

All grammar models were written as standard context-free grammars, and Chappelier and Rajman’s [15] probabilistic extended Cocke-Younger-Kasami parsing algorithm (which constructs a table containing generalized items like those in Earley’s [16] algorithm) was used to obtain the most likely parse for each sentence in the training set; this parse was compared to the corresponding gold-standard tree in the test set using the customary PARSEVAL evaluation metric [1, 2, p. 432] after deannotating the most likely tree delivered by the parser. PARSEVAL gives partial credit to incorrect parses by establishing three measures:

- labeled precision (P) is the fraction of correctly-labeled nonterminal bracketing (constituents) in the most likely parse which match the gold-standard parse,
- labeled recall (R) is the fraction of brackets in the gold-standard parse which are found in the most likely parse with the same label, and
- crossing brackets (X) refers to the fraction of constituents in one parse cross over constituent boundaries in the other parse.

The crossing brackets measure does not take constituent labels into account and will not be shown here. Some authors (see, e.g. [17]) have questioned partial-credit evaluation metrics such as the PARSEVAL measures; in particular, if one wants to use a probability model to perform structural disambiguation before assigning some kind of interpretation to the parsed sentence, it may well be argued that the exact match between the gold-standard tree and the most likely tree is the only possible relevant measure. It is however, very well known that the Penn Treebank, even in its release 3, still suffers from problems. One of the problems worth mentioning (discussed in detail by Krotov et al. [18]) is the presence of far too many partially bracketed constructs according to rules like NP → NN NN CC NN NN NNS which lead to very flat trees, when one can, in the same treebank, find rules such as NP → NN NN, NP → NN NN NN NNS and NP → NP CC NP, which would lead to more structured parses such as

```
NP
/ \
NP CC NP
/ \
NN NN NN NN NNS
```

Some of these flat parses may indeed be too flat to be useful for semantic purposes and have little linguistic plausibility; therefore, if one gets a more refined parse, one may consider it to be the one leading to the correct interpretation or not, but it surely contains a more information than the flat, unstructured one.
For this reason, we have chosen to give, in addition to the exact-match figure, the percentage of trees having 100% recall, because these are the trees in which the most likely parse is either exactly the gold-standard parse or a refinement thereof in the sense of the previous example.

4.2 Structural disambiguation results

Here is a list of the models which were evaluated:

- A standard treebank grammar, with no annotation of node labels (NO or \( k = 2 \)), with probabilities for 15,140 rules.
- A child-annotated grammar (CHILD or \( k = 3 \)), with probabilities for 92,830 rules.
- A parent-annotated grammar (PARENT), with probabilities for 23,020 rules.
- A both parent- and child-annotated grammar (BOTH), with probabilities for 112,610 rules.

As expected, the number of rules obtained increases as more information is conveyed by the node label, although this increase is not extreme. On the other hand, as the generalization power decreases, some sentences in the test set become unparsable, that is, they cannot be generated by the grammar.

<table>
<thead>
<tr>
<th>ANNOTATION</th>
<th>( R )</th>
<th>( P )</th>
<th>( f_{R=100%} )</th>
<th>EXACT</th>
<th>PARSED</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO ((k = 2))</td>
<td>70.7%</td>
<td>76.1%</td>
<td>10.4%</td>
<td>10.0%</td>
<td>100%</td>
<td>57</td>
</tr>
<tr>
<td>CHILD ((k = 3))</td>
<td>79.2%</td>
<td>74.2%</td>
<td>19.4%</td>
<td>13.6%</td>
<td>94.6%</td>
<td>9</td>
</tr>
<tr>
<td>PARENT</td>
<td>80.0%</td>
<td>81.9%</td>
<td>18.5%</td>
<td>16.3%</td>
<td>100%</td>
<td>340</td>
</tr>
<tr>
<td>BOTH</td>
<td>80.1%</td>
<td>75.6%</td>
<td>20.5%</td>
<td>14.7%</td>
<td>79.6%</td>
<td>75</td>
</tr>
</tbody>
</table>

Table 1. Parsing results with different annotation schemes: labelled recall \( R \), labelled precision \( P \), fraction of sentences with total labelled recall \( f_{R=100\%} \), fraction of exact matches, fraction of sentences parsed by the annotated model, and average time per sentence in seconds.

The results in table 1 show that

- The parsing performance of parent-annotated and child-annotated PCFG is similar and better than those obtained with the standard treebank PCFG. The performance is measured both with the customary PARSEVAL metrics and by counting the number of maximum-likelihood trees that (a) match their counterparts in the treebank exactly, and (b) contain all of the constituents in their counterpart (100% labeled recall, \( f_{R=100\%} \)). The fact that child-annotated grammars do not perform better than parent-annotated ones may be due to their larger number of parameters compared to parent-annotated PCFG. This makes it difficult to estimate them accurately from currently available treebanks (only about 6 subtrees per rule in the experiments).
– The average time to parse a sentence shows that child annotation leads to parsers that are much faster. This comes as no surprise because the number of possible parse trees considered is drastically reduced; this is, however, not the case with parent-annotated models.

It may be worth mentioning that parse trees produced by child-annotated models tend to be more structured and refined than parent-annotated and unannotated parses which tend to use rules that lead to flat trees in the sense mentioned above. Favoring structured parses may be convenient in some applications (especially where meaning has to be compositionally computed) but may be less convenient than flat parses when the structure obtained is incorrect.

On the other hand, child-annotated models, CHILD and BOTH, were unable to deliver a parse tree for all sentences in the test set (CHILD parses 94.6% of the sentences and BOTH, 79.6%). To be able to parse all sentences, the smoothing techniques described in section 3 were applied.

Those smoothed models were evaluated:

- A linear interpolated model, M1, as described in subsection 3.1 with \( \lambda = 0.7 \) (the value of \( \lambda \) selected to minimize the perplexity).
- A tree-level back-off, M2, as described in subsection 3.2.
- A rule-level back-off, M3, as described in subsection 3.3. This model has 92,830 \( k = 3 \) rules, 15,140 \( k = 2 \) rules and 10,250 back-off rules. A fixed parameter \( \lambda \) (0.005) was selected to maximize labelled recall and precision.

<table>
<thead>
<tr>
<th>MODEL</th>
<th>( R )</th>
<th>( P )</th>
<th>EXACT</th>
<th>PARSED</th>
<th>( I )</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>80.2%</td>
<td>78.6%</td>
<td>17.4%</td>
<td>100%</td>
<td>57</td>
</tr>
<tr>
<td>M2</td>
<td>78.9%</td>
<td>74.2%</td>
<td>17.1%</td>
<td>100%</td>
<td>9.3</td>
</tr>
<tr>
<td>M3</td>
<td>82.4%</td>
<td>81.3%</td>
<td>17.5%</td>
<td>100%</td>
<td>68</td>
</tr>
</tbody>
</table>

**Table 2. Parsing results with different smoothed models.**

The results in table 2 show that:

- M2 is the fastest but its performance is worse than that of M1 and M3.
- M1 and M3 parse sentences at a comparable speed but recall and precision are better using M3.

Compared to un-smoothed models, smoothed ones:

- Cover the whole test set (\( k = 3 \) did not).
- Parsed at reasonable speed (compared to PARENT).
- Achieved acceptable performance (\( k = 2 \) did not).
5 Conclusion

We have introduced a new probabilistic context-free grammar model, offspring-annotated PCFG in which the grammar variables are specialized by annotating them with the subtree they generate up to a certain level. In particular, we have studied child-annotated models (one level) and have compared their parsing performance to that of unannotated PCFG and of parent-annotated PCFG [9]. Offspring-annotated models may be seen as a special case of a very general probabilistic state-based model, which in turn is based on probabilistic bottom-up tree automata. The experiments show that:

- The parsing performance of parent-annotated and the proposed child-annotated PCFG is similar.
- Parsers using child-annotated grammars are, however, much faster because the number of possible parse trees considered is drastically reduced; this is, however, not the case with parent-annotated models.
- Child-annotated grammars have a larger number of parameters than parent-annotated PCFG which may make it difficult to estimate them accurately from currently available treebanks.
- Child-annotated models tend to give very structured and refined parses instead of flat parses, a tendency not so strong for parent-annotated grammars.

As future work we plan to study the use of statistical confidence criteria as used in grammatical inference algorithms [19] to eliminate unnecessary annotations by merging states, therefore reducing the number of parameters to be estimated. Indeed, offspring-annotation schemes (for a value of $k \geq 2$) may be useful as starting points for those state-merging mechanisms, which so far have always been initialized with the complete set of different subtrees found in the treebank (ranging in the hundreds of thousands).

References


Unsupervised Grammar Induction in a Framework of Information Compression by Multiple Alignment, Unification and Search

J Gerard Wolff
CognitionResearch.org.uk,
Telephone: +44(0)1248 712062,
Email: jgw@cognitionresearch.org.uk

Abstract. This paper describes a novel approach to grammar induction that has been developed within a framework designed to integrate learning with other aspects of computing, AI, mathematics and logic. This framework, called information compression by multiple alignment, unification and search (ICMAUS), is founded on principles of Minimum Length Encoding pioneered by Solomonoff and others. Most of the paper describes SP70, a computer model of the ICMAUS framework that incorporates processes for unsupervised learning of grammars. An example is presented to show how the model can infer a plausible grammar from appropriate input. Limitations of the current model and how they may be overcome are briefly discussed.

1 Introduction

This paper describes a novel approach to unsupervised grammar induction that has been developed within a research programme whose overarching goal is the integration of diverse functions—learning, recognition, reasoning and others—within one relatively simple framework. This has had a substantial impact on the way in which the learning processes are organised.

The new framework called information compression by multiple alignment, unification and search (ICMAUS) originated in earlier research developing the SNPR model of grammar induction [25, 24]. Without supervision, the SNPR model successfully learns artificial context-free phrase-structure grammars (CF-PSGs) using a technique of ‘hierarchical chunking’ combined with a search for disjunctive (part of speech) categories and processes for generalising grammatical rules and correcting over-generalisations.

In the ICMAUS programme, the aim has been to match or exceed these capabilities within a system that has been generalised to model a range of other aspects of computing, AI, mathematics and logic. It became apparent at an early stage that this would mean a radical reorganisation of the SNPR model. In the ICMAUS framework a concept of multiple alignment—to be described—has replaced hierarchical chunking as the predominant mode of organisation. With this new orientation, the system provides an interpretation for concepts in
computing, mathematics and logic and it has a range of AI capabilities described in [28] and earlier papers cited there. The present paper describes how the system has been developed for unsupervised learning of grammars.

A much fuller account of the research described here may be found in [27], available from http://www.cognitionresearch.org.uk/papers/ul/ul.htm.

1.1 Relationship with Other Research on Grammar Induction

This research extends the tradition of distributional linguistics pioneered by [8, 6] and others.

At the heart of ICMAUS system are principles of Minimum Length Encoding (MLE) pioneered by [18] (see also [12]). In this framework, grammar induction is conceived as a process of optimisation rather than a process of identifying a target grammar 'in the limit' as postulated by [7]. In the MLE framework, there is no target grammar, merely a process of searching for grammars that are 'good' in terms of MLE principles.

Recent studies that are, perhaps, most closely related to the present research include: [1–3, 5, 9–11, 14–17, 21, 20, 19]. Not all of these studies have adopted MLE principles but they deal with issues and processes that relate to the present research. The idea of combining learning with parsing—to be described—has also been developed by by Nakamura (see [13] and this workshop).

Compared with other work on unsupervised learning of grammar-like structures, the most distinctive features of the ICMAUS research are:

- The integration of learning with other areas of AI, computation, mathematics and logic.
- The multiple alignment concept as it has been developed in the ICMAUS framework, described below. There is, however, a clear affinity with 'alignment-based learning' [21, 20].

2 The ICMAUS Framework

In the ICMAUS framework, all knowledge is stored as patterns: arrays of symbols in one or two dimensions.\footnote{In work to date, the focus has been on one-dimensional patterns.} Despite the simplicity of this format, it is possible within the ICMAUS system to represent several different kinds of knowledge including context-free and context sensitive grammars, networks, trees, if-then rules and others.

Given the generality of this format for knowledge, the learning techniques described in this paper are relevant to the learning of any kind of knowledge, not just 'grammars', narrowly conceived.

The ICMAUS framework is intended as an abstract model of any kind of system for computing or cognition, either natural or artificial. In broad terms, the system works by receiving 'New' information from its environment and transferring it to a repository of 'Old' information. At the same time, it tries to compress
the information as much as possible by finding patterns that match each other and merging or “unifying” patterns that are the same. In these broad terms it is similar to a ZIP program but it differs in the thoroughness of the search for “good” unifications of patterns and in the “multiple alignment” concept, to be described.

2.1 Multiple Alignment

The concept of multiple alignment in the ICMAUS framework has been borrowed from the field of bio-informatics and adapted as described in [28].

An example of an ICMAUS multiple alignment is shown in Figure 1. Row 0 contains the New pattern ‘one of them does’ and all the other rows contain Old patterns, one pattern per row. By convention, the New pattern is always shown in row 0 but otherwise the assignment of patterns to rows is entirely arbitrary.

Fig. 1. A multiple alignment with ‘one of them does’ in New and patterns representing grammatical rules in Old.

Apart from the pattern in row 8, the patterns from Old in this example are like re-write rules in a CCG-PCG with the re-write arrow omitted. If we ignore row 8, the alignment shown in Figure 1 is very much like a conventional parsing, marking the main components of the sentence: words and phrases and the sentence pattern itself (shown in row 5).

Row 8 shows how the ‘discontinuous’ dependency that exists between the singular noun in the subject of the sentence (‘Ns’) and the singular verb (‘Vs’) can be marked within the alignment in a relatively direct manner. Despite the simplicity of the format for representing knowledge, the formation of multiple alignments enables the system to express ‘context sensitive’ aspects of language and other kinds of knowledge.

In each Old pattern there are two kinds of symbols: ID-symbols like ‘<’, ‘N’, ‘Np’, ‘0’ and ‘>’ in ‘< N Np 0 them >’ serve to identify the pattern and the
remaining symbols (‘t h e m’ in this example) are C-symbols that represent the 
contents or substance of the pattern.

Much more detail, with many more examples, may be found in [26].

3 SP70

All the main components of the ICMAUS framework outlined in Section 2 are 
now realised within the SP70 software model (version 9.2). The model is able to 
abstract plausible grammars from sets of simple sentences without prior knowl-
dge of word segments or the classes to which they belong, and the computational 
complexity of the model appears to be acceptable (Section 4). However, in its 
current form, the model has at least two significant shortcomings and some other 
deficiencies, discussed briefly in Section 6.

3.1 Objectives

In the development of this model, the main problems that have been addressed 
are:

- How to identify significant segments in the ‘corpus’ of raw data when the 
  boundary between one segment and the next is not marked explicitly.
- How to identify disjunctive classes of syntactically-equivalent segments (e.g., 
  ‘nouns’, ‘verbs’ and ‘adjectives’).
- How to combine the learning of segmental structure with the learning of 
  disjunctive classes.
- How to learn segments and disjunctive classes through two or more levels of 
  abstraction.
- How to generalize grammatical rules beyond the data and how to correct 
  over-generalizations without feedback from a ‘teacher’ or the provision of 
  ‘negative’ samples or the grading of the data from ‘easy’ to ‘hard’ (cf. [7]).

Solutions to these problems were found in the SNPR model [25, 24] but, as 
noted earlier, the organisation of this model is quite unsuited to the wider goals 
of the present research—integration of diverse functions within one framework. 
The SP70 model (v. 9.2) provides solutions to the first three problems and partial 
solutions to the fourth and fifth problems. Further development is planned as 
indicated in Section 6, below.

3.2 Overall Structure of the Model

Figure 2 shows the high-level organisation of the SP70 model.

The function create_multiple_alignments() referred to in Figure 2 creates zero 
or more multiple alignments, each one comprising the current pattern from New 
(CPFN) and one or more patterns from Old. This function is essentially the 
same as the main component of the SP61 model, described quite fully in [26]. 
Readers are referred to this source for a more detailed description of how multiple 
alignments are formed in the ICMAUS framework.
SP70C
{
  1 Read a set of patterns into New. Old is initially empty.
  2 Compile an alphabet of symbol types in New and, for each type,
    find its frequency of occurrence and the number of bits
    required to encode it (using the Shannon–Fano– Elias method).
  3 While (there are unprocessed patterns in New)
    {
      3.1 Identify the first or next pattern from New as the
          'current pattern from New' (CPPN).
      3.2 Apply the function CREATE_MULTIPLE_ALIGNMENTS() to
          create multiple alignments, each one between the
          CPPN and one or more patterns from Old.
      3.3 During 3.2, the CPPN is copied into Old, one symbol
          at a time, in such a way that the CPPN can be
          aligned with its copy but that any one symbol in
          the CPPN cannot be aligned with the corresponding
          symbol in the copy.
      3.4 Sort the alignments formed by this function in order
          of their compression scores and select the best
          few for further processing.
      3.5 Process the selected alignments with the function
          DERIVE PATTERNS(). This function derives encoded
          patterns from alignments and adds them to Old.
    }
  4 Apply the function SIFTING_AND_SORTING() to create one or
      more alternative grammars for the patterns in New, each
      one scored in terms of MLE principles. Each grammar is
      a subset of the patterns in Old.
}

Fig. 2. The organisation of SP70. The workings of the functions create_multiple_alignments(), derive_patterns() and sifting_and_sorting() are explained in the text.

3.3 Deriving Patterns from Alignments

In operation 3.5 in Figure 2, the derive_patterns() function is applied to a selection of the best alignments formed and, in each case, it looks for sequences of unmatched symbols within the alignment and also sequences of matched symbols.

Consider the alignment shown in Figure 3. From an alignment like that, the function finds the unmatched sequences ‘g i r l’ and ‘b o y’ and, within row 1, it also finds the matched sequences ‘t h a t’ and ‘r u n s’. With respect to row 1, the focus of interest is the matched and unmatched sequences of C-symbols—ID-symbols are ignored.

A copy of each of the four sequences is made; ID-symbols are added to each copy and the copy is added to Old. In addition, another ‘abstract’ pattern is made that records the sequence of matched and unmatched patterns within the alignment. The result in this case is five patterns like those shown in Figure 4.
Fig. 3. A simple alignment from which other patterns may be derived.

\[
\begin{align*}
0 & \text{ that girl \ runs \ 0} \\
1 & < \%19 \text{ that boy \ runs} > 1
\end{align*}
\]

Fig. 4. Patterns derived from the alignment shown in Figure 3.

It should be clear that the set of patterns in Figure 4 is, in effect, a simple grammar for the two sentences in Figure 3, with patterns representing grammatical rules in much the same style as those shown in Figure 1. The abstract pattern \('< \%10 \%20 \%7 > < \%9 > < \%8 > >>' describes the overall structure of this kind of sentence with slots that may receive individual words at appropriate points in the pattern.

Notice how the symbol \('%9' serves to mark 'boy' and 'girl' as alternatives in the middle of the sentence. This is a grammatical class in the tradition of distributional or structural linguistics (see, for example, [8, 6]).

3.4 Sifting and Sorting of Patterns

In the example just shown, all the patterns derived from the alignment are 'correct'. But in many cases, patterns that are derived in this way and added to Old are 'wrong'. The wrong patterns are weeded out in the sifting_and_sorting() stage of processing (operation 4 in Figure 2), where the system develops one or more alternative grammars for the patterns in New in accordance with MLE principles. Figure 5 shows the overall structure of the sifting_and_sorting() function.

Compiling a Set of Alternative Grammars A set of alternative grammars for the patterns in New that are good in terms of MLE principles are derived (in the compile_alternative_grammars() function) in operation 4 of Figure 5. Each grammar is a subset of the patterns that have been added to Old during operation 3 of Figure 2.

The process of compiling good grammars is essentially a hill-climbing search through the abstract space of alternative grammars, trying to minimise \((G + E)\) for each grammar, where \(G\) is the size of the given grammar (in bits) and \(E\) is the size of all the New patterns (in bits) after they have been encoded in terms of the grammar. Minimising \((G + E)\) is, of course, the central idea in grammar induction using MLE principles. In what follows, \((G + E)\) is abbreviated as \(T\).
SIFTING AND SORTING

1. For each pattern in Old, set its frequency of occurrence to 0.
2. While (there are still unprocessed patterns in New)
   2.1 Identify the first or next pattern from New as the CPPN.
   2.2 Apply the function CREATE_MULTIPLE_ALIGNMENTS to create multiple alignments, each one between the CPPN and one or more patterns from Old.
   2.3 From amongst the best of the multiple alignments formed, select ‘full’ alignments in which all the symbols of the CPPN are matched and all the C-symbols are matched in each pattern from Old.
   2.4 For each pattern from Old, count the maximum number of times it appears in any one of the full alignments selected in operation 2.3. Add this count to the frequency of occurrence of the given pattern.

3. Compute frequencies of symbol types and their encoding costs. From these values, compute encoding costs of patterns in Old and new compression scores for each of the full alignments created in operation 2.
4. Using the alignments created in 2 and the values computed in operation 3, COMPILE_ALTERNATIVE_GRAMMAR.

Fig. 5. The organisation of the sifting_and_sorting() function. The compile_alternative_grammars() function is described in the text.

The grammars are built in stages, at first trying to minimise T for the first New pattern alone, then trying to minimise T for the first and second New pattern, followed by the first, second and third, and so on.

4 Computational Complexity

In a serial processing environment, the time complexity of SP70 is approximately O(N^2) where N is the number of patterns in New. In a parallel processing environment, the time complexity may approach O(N), depending on how well the parallel processing is applied. In serial or parallel environments, the space complexity should be O(N).

The time complexity of the program may be improved when it has been developed, as envisaged, so that the New patterns are processed in batches, with a purging of Old between each batch to remove all patterns except those in the best grammar. In this case, the time complexity should be O(N).

5 Example

When New contains the eight sentences shown in Figure 6, the best grammar found by SP70 is the one shown in Figure 7.
that boy runs
that girl runs
that boy walks
that girl walks
some boy runs
some girl runs
some boy walks
some girl walks

Fig. 6. Eight sentences supplied to SP70 as New.

< %22some >
< %23that >
< %15boy >
< %16girl >
< %34runs >
< %37walks >
< 1 < %2 < %1 < %3 > >

Fig. 7. The best grammar (in terms of MLE principles) that is found by SP70 when New contains the eight sentences shown in Figure 6.

5.1 Intermediate Results

As the first phase of learning proceeds (operation 3 of Figure 2), intermediate results are often much less tidy than the example shown in Section 3.3. For example, when Old contains only the first pattern shown in Figure 6, the only alignment it can create is:

0 that boy runs 0
1 < %19 that boy runs > 1

Notice that the Old pattern (in row 1) is, in effect, the same pattern as the New pattern (in row 0) so it is not permissible to match 'o' in the New pattern, for example, with 'o' in the Old pattern because that would mean matching a given symbol with itself.

From the alignment just shown, the program derives 'bad' patterns like '< %3 14 that >', '< %4 18 boy runs >' and '< %4 17 hat boy runs >' and these are added to Old. However, as later patterns are processed, the repository of Old patterns begins to accumulate enough patterns that are good in MLE terms so that it is able to create quite respectable looking parsings like this.
In the `sifting_and_sorting()` and sorting phase, all the ‘bad’ patterns are discarded and the ‘good’ patterns are cleaned up by removing unnecessary ID-symbols and renaming the retained ID-symbols in a tidy manner.

5.2 Values for $G$, $E$, $T$ and Compression

Figure 1 shows changing values for $G$, $E$ and $T$ for the best grammar found (in terms of MLE principles) as successive patterns from New are processed in `compile_alternative_grammars()`. It is interesting to see that, as successive patterns are processed, progressively more compression is achieved, represented by the falling values for ($T$ / ‘original’), shown in the last column.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>$G$</th>
<th>$E$</th>
<th>$T$</th>
<th>Original</th>
<th>Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>790,49</td>
<td>25.78</td>
<td>790,27</td>
<td>7943,70</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>11065,38</td>
<td>191.29</td>
<td>11276,67</td>
<td>16209,42</td>
<td>0.68</td>
</tr>
<tr>
<td>3</td>
<td>14005,26</td>
<td>302.09</td>
<td>14097,35</td>
<td>25195,14</td>
<td>0.59</td>
</tr>
<tr>
<td>4</td>
<td>14005,26</td>
<td>307.57</td>
<td>15002,83</td>
<td>34502,87</td>
<td>0.44</td>
</tr>
<tr>
<td>5</td>
<td>17690,37</td>
<td>563.32</td>
<td>18213,39</td>
<td>42888,08</td>
<td>0.42</td>
</tr>
<tr>
<td>6</td>
<td>17690,37</td>
<td>731.75</td>
<td>18303,82</td>
<td>51155,30</td>
<td>0.36</td>
</tr>
<tr>
<td>7</td>
<td>17690,37</td>
<td>887.00</td>
<td>18337,67</td>
<td>69822,52</td>
<td>0.31</td>
</tr>
<tr>
<td>8</td>
<td>17690,37</td>
<td>1044.92</td>
<td>18094,99</td>
<td>69171,76</td>
<td>0.27</td>
</tr>
</tbody>
</table>

Table 1. Cumulative values (in bits) of $G$, $E$ and $T$ for the best grammar found as successive patterns from New are processed in `compile_alternative_grammars()`. For comparison purposes, the cumulative sizes of the original patterns (excluding ID-symbols) are shown in the ‘original’ column and values for compression ($T$ / ‘original’) are shown in the last column.

6 Discussion

6.1 Evaluation

In accordance with the ‘looks-good-to-me’ approach to the evaluation of grammar induction systems [20], the grammar shown in Figure 7 looks like an appropriate grammar for the patterns shown in Figure 6.\footnote{A possible improvement might be a grammar that isolates the ‘s’ in ‘runs’ and ‘was’ in ‘kiss’ as a separate morpheme.} This may seem like a sloppy
method of evaluation but it should not be forgotten that the human brain is, by a wide margin, the best learning system on the planet. This provides a justification for using human judgement of what does or does not ‘look good’ as a means of evaluating the output of artificial learning systems. With any system that is sufficiently robust to be applied to realistic samples of natural language, then there is no alternative to (human) judgements about what is or is not a ‘correct’ grammar for a given language or (human) conventions about how language is segmented into words. Statistical tests may be applied to establish whether or not there is a significant level of agreement between structures established by human judgement and the results of artificial learning [22, 23].

Notice that the use of a ‘target’ grammar as a criterion of success (as in Gold’s approach to learning [7]) does not overcome the problem that, for any given language sample, there are many alternative grammars that are compatible with the sample and some are ‘better’ than others.

6.2 Reorganisation Needed

The example in the previous section is good enough to show that the approach is sound but experiments with other examples have shown that the model suffers from two main weaknesses:

- Although the model in its current form can isolate basic segments and tie them together in an overall abstract structure, it is not good at finding intermediate levels of abstraction.
- In the development of the model to date, no attempt has been made to enable the system to detect discontinuous dependencies such as number dependency between the subject of a sentence and its main verb (as mentioned in Section 2.1). Although this kind of capability may seem like a refinement that we can afford to do without at this stage of development, a deficiency in this area seems to have an impact on the program’s performance at an elementary level.

A possible solution to both problems is a reorganisation of the model so that learning is integrated even more closely with parsing. Recent work has shown that operation 2.2 in the sifting_and_sorting() function (Figure 5) can be omitted—the multiple alignments from operation 3.2 in Figure 2 can be used instead. It is also envisaged that New patterns will be processed in batches and that, after each batch, sifting_and_sorting() will be applied and Old patterns that are not proving useful will be discarded.

7 Conclusion

SP70 is not yet an ‘industrial strength’ system for unsupervised learning but I believe the framework has considerable potential and provides a sound basis for further development.
A key attraction of this approach to learning is that the ICMAUS framework provides a unified view of a variety of issues in AI thus facilitating the integration of grammar induction with other aspects of intelligence. Given the generality of the framework, the learning techniques described here are relevant to the learning of any kind of knowledge, not just grammars.

Acknowledgements

I am grateful to Pat Langley and Merno van Zaanen for constructive comments on the report on which this paper is based. The paper has also benefitted from comments and suggestions by an anonymous referee.

References


