MINISTERE DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ FERHAT ABBAS – SÉTIF
UFAS (ALGÉRIE)

THESE

Présentée devant la Faculté des Sciences
Département d'Informatique
Pour l'obtention du diplôme de

DOCTORAT D'ÉTAT

OPTION : INTELLIGENCE ARTIFICIELLE

Par

Chafia KARA-MOHAMED (épouse HAMDI-CHERIF)

THEME

A development environment integrating algorithms, inferences and learning – *ESLIM* Project

Soutenue le : 01/07/2012 Devant la commission d'examen

A. BOUKERRAM	Maître de Conférences	U. FA, Sétif	Président
A. HAMDI-CHERIF	Professeur	Qassim U., Arabie S.	Rapporteur
M. BENMOHAMMED	Professeur	U. Constantine	Examinateur
A. REFOUFI	Maître de Conférences	U. FA, Sétif	Examinateur
M. ALIOUAT	Maître de Conférences	U. FA, Sétif	Examinateur

الإهداء

إلى *"الذِينَ يَبِيتُونَ لِرَبِّهِمْ سُجَّدًا وَ قِيَـــمًا"* ...

... ساعين للفرقان بين الحقّ و الباطل،

... عرفانًا مني، تأسيا بهم و تقديرًا لهم...ثم إلى...

... والديّ الكريمين الذين من ضحّيا كثيرًا من أجلي

... زوجي الذي علّمني حسن الظن **بالله**

... أخواتي اللواتي علّمنني كيف تكون الأسرة واحدة

... أخي الكبير الذي علّمني الشّهامة و العزّة و الصّبر

... أخي الصّغير الذي علّمني الإرادة القويّة

... و أولادي الذين علّموني كيف يمكن للصّغير أن يكون كبيرًا

This work is dedicated to:

…Those who "*spend their nights prostrated or standing in prayers*"

Those whose knowledge came from the Source-Of-All

In human and intelligible form

Springing out of the profoundness of Self

For guidance, light and salvation

Far from the whims of mundane glory …

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABREVIATIONS .... *I*

LIST OF FIGURES .... *IV*

LIST OF TABLES .... *VI*

LIST OF ALGORITHMS .... *VII*

ACKNOWLEDGEMENTS .... *IX*

ملخص .... *X*

ABSTRACT .... *XI*

RESUME .... *XII*

CHAPTER 1 INTRODUCTION *1*

1. *PRELIMINARIES* 1
2. *MOTIVATIONS* 2
3. *BACKGROUND AND OBJECTIVES* 3
   - **3.1 Process of inference** 3
     - *3.1.1 Inference in symbolic settings* 3
     - *3.1.2 Inference in knowledge-based systems (KBSs)* 4
     - *3.1.3 Inference in learning settings* 4
   - **3.2 Specific goals** 5
     - *3.2.1 Avoiding the "general problem solving (GSP)" syndrome* 5
     - *3.2.2 Syntactic level - first* 6
   - **3.3 Main tools** 7
     - *3.3.1 Grammars and parsing* 7
     - *3.3.2 Declarative programming and FOL* 8
4. *ORGANIZATION OF THE MANUSCRIPT* 8

CHAPTER 2 SOME CONCEPTS OF FORMAL LANGUAGES **11**

1. *INTRODUCTION* 11
2. *PRELIMINARIES* 12
3. *LANGUAGES* 13
   - **3.1 Operations on languages** 13
   - **3.2 languages models** 14
     - *3.2.1 Formal grammars* 14
     - *3.2.2 Automata* 14
       - *3.2.2.1 Finite state automata (FSA)* 15
       - *3.2.2.2 Push-down automata (PDA)* 15
     - *3.2.3 Regular expression* 16

3.2.4 Topological consideration 17
4. CHOMSKY LANGUAGES HIERARCHY *17*
4.1 Type 3 - Regular languages 17
4.2 Type 2 - Context-free languages 17
4.3 Type 1 - Context-sensitive languages 18
4.4 Type 0 - Unrestricted (free) languages 18
5. REGULAR LANGUAGES *19*
5.1 Introductory example 19
5.2 Characteristics of regular languages 20
6. CONTEXT-FREE LANGUAGES (CFLS) *21*
6.1 Examples of CFLs 21
6.2 Applications of CFLs 22
6.3 Characteristics of CFLs 22
6.4 Relationship between regular and CFLs 23
7. PARSING *24*
7.1 Top-down parsing 24
7.2 Bottom-up parsing 25
7.3 Hybrid parsing 25
8. CONCLUSION *25*

CHAPTER 3 STATE OF THE ART OF GRAMMATICAL INFERENCE **27**

INTRODUCTION *27*
2. THEORETICAL MODELS FOR GRAMMAR INFERENCE *28*
2.1. Identification in the limit (learning from text) 29
2.1.1 Definition 29
2.1.2 Characteristics 30
2.2 Active learning 30
2.2.1 Definition 31
2.2.2 Characteristics of active learning 32
2.3 PAC learning 32
2.3.1 Definitions 32
2.3.2 Characteristics 33
2.4 Relation between active learning and PAC learning 33
3. ALGORITHMS FOR GI *34*
3.1 Algorithms for regular grammars 34
3.1.1 Complexity for inferring regular grammars 35
3.1.2 Learning FA 36
3.1.2.1 Trakhtenbrot and Barzdin 36
3.1.2.2 Gold's algorithm 36
3.1.2.3 RPNI algorithm 36
3.1.2.4 Traxbar algorithm 37
3.1.2.5 Dupont's lattice setting 37
3.1.2.6 Evidence Driven State Merging (EDSM) Heuristic 37
3.1.2.7 Data-driven heuristic 38
3.1.3 Learning non-deterministic finite state automata NFA 38
3.1.4 Learning quantum finite automata 39
3.2. Algorithms for CFGs 39

3.2.1 Difficulty of CFG inference 40
3.2.2 Algorithms for CFG inference 41
3.2.2.1 Complexity 41
3.2.2.2 Patterns in strings 42
3.2.2.3 Extension of regular languages 'results to CFLs 42
3.2.2.4 Use of artificial intelligence techniques 43
3.2.2.5 Stochastic CFGs (SCFGs) 43
3.2.2.6 Algorithms that uses alternative representations for languages 44
3.2.2.7 Algorithms that rely on structured data 45
3.2.2.8 ILSGInf : Inductive Learning System for Grammatical Inference 45
4. APPLICATIONS OF GRAMMATICAL INFERENCE TECHNIQUES 47
4.1 Structured pattern recognition 47
4.2 Computational linguistics 47
4.3 Speech recognition 48
4.4 Automatic translation 48
4.5 Document management 48
4.6 Data and text mining 49
4.6.1 Text mining 49
4.6.2 Text compression 49
4.6.3 RPNI and structure induction 50
4.7 Biological interfaces 50
4.7.1 Grammatical structures in biological sequences 50
4.7.2 DNA computing 50
4.8 Map learning 51
4.9 Self assembling 51
4.10 Software engineering 52
4.11 Soft computing and evolutionary multiobjective optimization (EMO) 52

CHAPTER 4 GRAMMATICAL INFERENCE WITH GASRIA 53

1. INTRODUCTION 53
2. PROBLEM FORMULATION AND BASIC METHODS 54
2.1 GASRIA Objectives 55
2.2 Methods used 56
3. RELATED WORKS: THREE INTERCONNECTED FIELDS 56
3.1 Formal languages approach 56
3.2 Machine Learning (ML) 58
3.2.1 Inductive and deductive learning 58
3.2.2 Some ML/data mining methods 58
3.3 Inductive logic programming (ILP) 59
4. GI VS. ILP 60
4.1 Problem of inductive inference 60
4.1.1 Inductive inference and normal semantics 60
4.1.2 Inductive inference and definite semantics 62
4.2 Formalized ILP approach 63
4.3 GI formulated in ILP framework 64
4.4 GI - ILP interplay 64
5. GASRIA ARCHITECTURE 65

5.1 *GASRIA* modes of operation 65
5.1.1 Overall block diagram 65
5.1.2 GASRIA class diagram 67
5.2 Learning mode: *ILSGInf* 67
5.3 Exploitation mode: *EXINF* 68
5.4 Fact base 68
5.4.1 Initial symbol and the grammar of the language 68
5.4.2 Additional information 69
5.5 Rule base 69
5.5.1 Vocabulary and rule base syntax 69
5.5.5.1 Vocabulary 69
5.5.5.2 Rule base syntax 70
5.5.2 Automatic syntactic analysis 70
6. PARSING 71
6.1 Notation 71
6.2 Earley's algorithm 71
6.2.1 The idea 71
6.2.2 Detailed steps of Earley's algorithm 73
6.2.3 Correctness 74
6.2.4 Earley and CYK algorithms 74
6.3 Additional definitions 75
6.3.1 Types of sentences and partial derivatives (PaDe's) 75
6.3.2 Derivation trees 75
6.4 Motivation for using PaDe's 76
7. LEARNING IN GASRIA 77
7.1 Learning characteristics 77
7.2 Learning strategy implementation 77
8. RESULTS AND DISCUSSION 77
8.1 *GASRIA* implementation 77
8.2 Example 78
8.2.1 Learning phase: ILSGInf use 79
8.2.2 Exploitation phase: EXINF use 80
9. CONCLUSION 81

CHAPTER 5 INFERENCES WITH EXINF INTELLIGENT PARSING ISSUES 83

1. INTRODUCTION 83
2. EXINF OBJECTIVES 84
2.1 Inferential characteristics 85
2.2 Parsing characteristics 85
2.3 Complementary characteristics 86
3. FIRST-ORDER LOGIC (FOL) CONSIDERATIONS 86
3.1 Rule-based deduction systems 86
3.1.1 Rules and operation 86
3.1.2 Basic components of rule-based systems 87
3.2 Knowledge-base engineering issues 88
3.2.1 Knowledge acquisition 88
3.2.2 Knowledge explanation 89
3.3 Forward chaining (FC 89

3.4 Backward chaining (BC) 90
3.5 Backward chaining *vs.* forward chaining 91
4. EXINF ARCHITECTURE *92*
4.1 Design diagrams 92
4.1.1 Use case diagram *92*
4.1.2 Class diagram *93*
4.3 The three *EXINF* layers 93
4.3.1 EXINF first layer *93*
4.3.2 EXINF second layer *94*
4.3.3 EXINF third layer *94*
5. EXINF - KBS USED FOR PARSING *97*
5.1 *EXINF* as a knowledge-based system (KBS) 97
5.2 Declarative Earley's algorithm: rule base 97
5.2.1 Summarized Earley's algorithm *97*
5.3 *EXINF* reasoning mechanism 98
5.3.1 Forward chaining implementation *99*
5.3.2 Example *100*
6. APPLICATIONS *101*
6.1 Problem 1: regular language 101
6.1.1 EXINF first and second layers *101*
6.1.2 EXINF third layer *104*
6.2 Problem 2 : context-free language (CFL) 104
6.2.1 EXINF 2^{nd} layer *104*
6.2.2 EXINF with counter example *106*
6.2.3 EXINF third layer for CFL *107*
7. CONCLUSION *107*

CHAPTER 6 AN INDUCTIVE LEARNING SYSTEM FOR GRAMMATICAL INFERENCE - **ILSGINF** ***109***

1. INTRODUCTION *109*
2. RELATED WORKS *110*
2.1 ML and human interaction 110
2.2 Algorithm types 111
3. ILSGINF OBJECTIVES *112*
4. ILSGINF LEARNING SOLUTION *112*
4.1 Basic properties 112
4.2 *ILSGInf* architecture 113
4.3 General structure of *ILSGInf* learning strategy 115
4.3.1 Strategy overview and complexity *115*
4.3.2 Refinement cycle *116*
4.4 Validation procedure 117
5. ILSGINF IMPLEMENTATION *117*
5.1 Initial grammar construction 117
5.2 Partial parsing 119
5.3 Detailed refinement cycle 120
5.3.1 Generalization *120*
5.3.2 Partial derivatives (PaDe's) construction *121*

5.3.3 One PaDe construction for a sub-sentence ... 122
5.3.4 Heuristics for sorting PaDe's ... 122
6. TESTED EXAMPLE ... *123*
6.1 *PPA* use ... 123
6.2 Discussions ... 124
7. CONCLUSION ... *125*

CHAPTER 7 *GASRIA/ILSGINF* INTERACTIONS WITH SYSTEMS CONTROL ***127***

1. INTRODUCTION ... *127*
2. ILSGINF AND CONTROL SYSTEMS INTERACTION ... *128*
2.1 The basic control methodology ... 128
2.1.1 Negative feedback dynamic control ... 128
2.1.2 Control laws construction ... 129
2.2 Motivations for grammatical control approach ... 130
2.3 Using grammars to control machine drives ... 131
2.4 Steps for using GI in control systems ... 132
2.4.1 Quantification of the variables ... 132
2.4.2 Production rules ... 132
2.4.3 Learning ... 133
2.5 *EXINF/ILSGInf* in control of machine drives ... 133
2.6 Comparing GI-controlled systems with other methods ... 134
3. SELF-ASSEMBLY ISSUE ... *134*
3.1 Self-assembly as a process ... 134
3.2 Modes of self-assembly ... 136
3.3 Self-assembly central issue ... 136
3.4 Graph grammars ... 137
3.4.1 Definition of graph grammar ... 137
3.4.2 Application of graph grammars in self-assembly ... 137
4. FROM STRING GI TO GRAPH GI ... *138*
4.1 Four methodological levels for solution ... 138
5 CONCLUSION ... *139*

CONCLUSION ... ***141***

1. FIRST-ORDER LOGIC (FOL) AND GRAMMATICAL INFERENCE (GI) ... *141*
2. INFERENCES AND "INTELLIGENT" PARSING ... *142*
3. PARTIAL PARSING ALGORITHM ... *142*
4. PERFORMANCE CRITERIA ... *143*
5. GI, CONTROL AND SELF-ASSEMBLY ... *144*
6. PROSPECTS ... *144*
6.1 Parsing ... 144
6.2 GI-based control and self-assembly ... 144
7. FURTHER… FOR THE FUTURE ... *145*
7.1 Computer algebra system (CAS) improvement ... 145
7.2 Semantic level of programming languages ... 145

7.3 Grammars and bioinformatics 145
REFERENCES ***147***
GLOSSARY ***155***
APPENDIX 1 - CLASS OF LANGUAGES INFERRED BY *GASRIA* ***161***
APPENDIX 2 – *ILSGINF* CLASS DIAGRAM ***163***
APPENDIX 3 COMPLEXITY OF ILSGINF LEARNING ALGORITHM ***164***
INDEX ***165***

LIST OF SYMBOLS AND ABREVIATIONS

a_i	Character in a string (i=1,2,3…)
$a_1 \ldots a_n$	Sequence of characters
ω	String: a sequence of characters $\omega = a_1 \ldots a_n$
ω, r, l, x, v	Strings of terminals
$\lvert\omega\rvert$	Length of string ω
ω^R	Reversal of $\omega = a_n \ldots a_1$
$\lvert\omega\rvert_a$	Number of character a in the string ω
$\leq_{alpha}$	Alphabetical order over elements of Σ
$\leq_{length\text{-}lex}$	Length-lexical order over strings
$\leq_{lex}$	Lexical order over strings
$\leq_{prefix}$	Prefix order over strings
$\leq_{subseq}$	Subsequence order over strings
$\rightarrow$	Symbol separating left- and right-hand-side of a production
$\Rightarrow$	Single derivation
$\overset{*}{\Rightarrow}$	Multiple derivation
$\models$	Entailment
α, β	Strings formed by terminals and non-terminals
$a_1 a_2 \ldots a_n$	A string formed by n characters
B	Background (prior) knowledge
BC	Backward chaining
B_{pop}	State with branch and read symbol from stack operation
B_{read}	State with branch and read symbol from input operations
BNF	Backus Naur Form
C_G, C_S	Conjunction of clauses
CFG	Context-free grammar
CFL	Context-free language
CL	Class of languages
CNF	Chomsky normal form
CRS	Conflict resolution set
CYK	Cocke-Younger-Kasami algorithm
CSP	Constraint satisfaction problem
D_g, D_g'	Most general concatenation of all sub-sentences
δ_N	Transition function
DPDA, PDA	Deterministic push-down automaton, non deterministic
DSL	Domain-specific language
$E = E^+ \cup E^-$	Evidence as the concatenation of positive and negative evidence
F_A	Set of accepting states, a subset of Q

FA or DFA	Finite automaton (deterministic)
F_R	Set of rejecting states, a subset of Q
FC	Forward chaining
FOL	First order logic
|G|	Size of G
G	Symbol for grammar
G_0	Initial inferred grammar
G_i	Inferred grammar at stage i (i=1,2,…) of the inference process
GI	Grammar inference (or induction)
g	Set of grammars
G_h	Hypothesis grammar
G_t	Target grammar
H	Set of hypotheses
h	One hypothesis, element of H
Γ	Set of symbols in the stack
I	Set of initial states, a subset of Q
IE	information extraction
IR	information retrieval
ILP	Inductive logic programming
KB	Knowledge base
KBS	Knowledge-based system
LHS	Left-hand side
L	Language defined over an alphabet
L^*	$L^* = \cup_{i \in N} L^i$
$L(G)$	Language generated by grammar G
λ, ε	Empty character
L'	Complement of L
L_1/L_2	Complement of L_2 in L_1
LBA	Linear bounded automaton
$L+$	Set of positive examples of sentences
$L-$	Set of negative examples or counter examples of L
L^n	Power set of L, $L^0 = \{\lambda\}$ and $L^{n+1} = L.L^n$
MAT	Minimum adequate teacher
MCV	Most constrained variable
MQ	Membership query
MRV	Minimum remaining value, used in constraint satisfaction problem
NB_{push}	States with no branching but only push operations
NFA	Non deterministic finite automaton
N	Set of non-terminals or variables
P	Set of productions or rules

PAC	Probabilistic approximately correct
PaDe	Partial derivative
PPA	Partial parsing algorithm
PTA	Prefix tree acceptor
Q	Set of states
R	Inductive inference rule
RHS	Right-hand side
Σ	Set of characters (terminals) called alphabet
$\Sigma*$	All string of different lengths (including λ) formed over Σ
S	Starting symbol, special symbol in V
STA	Skeletal tree automata
SCFG	Stochastic context-free grammar
TM	Text mining
TL	Target language
U	Control variable
y	Output (controlled) variable

LIST OF FIGURES

CHAPTER 2: SOME CONCEPTS OF FORMAL LANGUAGES

Figure 2.1 DIAG21 - DFA that recognizes strings containing 001

Figure 2.2 DIAG22 - PDA recognizing $\{\omega\omega^R \mid \omega \in \{0, 1\}^*\}$

CHAPTER 3: SURVEY OF GRAMMATICAL INFERENCE (GI)

Figure 3.1 Methodology 31 - METH31 Methodological steps: inference problem

Figure 3.2 Methodology 32 - METH32 Combinatorial probl. associated with DFA

Figure 3.3 Diagram 31 - DIAG31 – *ILSGInf* within existing GI methods

CHAPTER 4: GRAMMATICAL INFERENCE WITH *GASRIA*

Figure 4.1 Methodology 41 – METH41 Methodological steps used in *GASRIA*

Figure 4.2 Methodology 42 – METH42 Inductive inference and normal semantics

Figure 4.3 Methodology 43 – METH43 Inductive inference and definite semantics

Figure 4.4 Methodology 44 – METH44 General ILP approach

Figure 4.5 Methodology 45 – METH45 GI problem formulated as an ILP problem

Figure 4.6 Architecture 41 – ARCH41 *GASRIA* architecture

Figure 4.7 Architecture 42 – ARCH42 *GASRIA* class diagram

Figure 4.8 Methodology 46 – METH46 Fact base syntax

Figure 4.9 Methodology 47 – METH47 Fact base structure

Figure 4.10 Methodology 48 – METH48 Syntax used by *EXINF*

Figure 4.11 Diagram 41 – DIAG41 Derivation tree of G

Figure 4.12 Methodology 49 – METH49 Grammar generation

Figure 4.13 Methodology 4.10 – METH410 Refining cycle in grammar generation

CHAPTER 5: INFERENCES WITH *EXINF* - INTELLIGENT PARSING ISSUES

Figure 5.1 Methodology 51 - METH51 Fact base
Figure 5.2 Methodology 52 - METH52 Rule base
Figure 5.3 Methodology 53 - METH53 Heuristics for learning from an expert

Figure 5.4 Methodology 54 - METH54 Heuristics in a rule-base system
Figure 5.5 Methodology 55 - METH55 Backward chaining *vs.* forward chaining
Figure 5.6 Architecture 51 - ARCH51 *EXINF* Use case diagram
Figure 5.7 Architecture 52 - ARCH52 *EXINF* as a three-layered system
Figure 5.8 Architecture 53 - ARCH53 *EXINF* as a detailed three-layered system
Figure 5.9 Application 51 - APPL51 Example of facts and rules
Figure 5.10 Application 52 - APPL52 Fact base for RL $L_1 = \{ w = (ab)^n, n>=1\}$
Figure 5.11 Application 53 - APPL53 Construction of list l_0*
Figure 5.12 Application 54 - APPL54 Fact base for CFL $L_2 = \{ w = (a^n b^n, n>=1\}$
Figure 5.13 Application 55 - APPL55 Fact base for CFL L_2 with counter example

CHAPTER 6: *ILSGInf* - AN INDUCTIVE LEARNING SYSTEM FOR GI

Figure 6.1 Diagram 61 – DIAG61 *ILSGINF* block diagram

Chapter 7: *GASRIA/ILSGInf* INTERACTIONS WITH SYSTEMS CONTOL

Figure 7.1 Diagram 7.1 – DIAG71 GI control in open-loop/closed-loop modes
Figure 7.2 Methodology 71 – METH71 Adapted GI control system methodology

LIST OF TABLES

CHAPTER 2: GRAMMATICAL INFERENCE WITH *GASRIA*
Table 2.1 TAB21 – Chomsky languages hierarchy

CHAPTER 4: GRAMMATICAL INFERENCE WITH *GASRIA*
Table 4.1 TAB41 Partial derivative construction for (a+b) sentence based on a+b

CHAPTER 5: INFERENCES WITH *EXINF* - INTELLIGENT PARSING ISSUES
Table 5.1 TAB51 Progressive construction of sub-lists
Table 5.2 TAB52 Construction of sub-lists for L_2 = { w = ($a^n b^n$, n>=1}
Table 5.3 TAB53 Construction of sub-lists for L_2 with counter example

CHAPTER 6: *ILSGInf* - AN INDUCTIVE LEARNING SYSTEM FOR GI
Table 6.1 TAB61 – Progressive construction of sub-lists

APPENDIX 1: CLASS OF LANGUAGES LEARNED BY *GASRIA*
Table A1 TABA1 - Class of languages inferred by *GASRIA*

LIST OF ALGORITHMS

CHAPTER 4: GRAMMATICAL INFERENCE WITH *GASRIA*

Algorithm 4.1 ALGO41 – Earley's algorithm

CHAPTER 5: INFERENCES WITH *EXINF* - INTELLIGENT PARSING ISSUES

Algorithm 5.1 ALGO51 Declarative Earley's algorithm

Algorithm 5.2 ALGO52 Implemented forward chaining

CHAPTER 6: *ILSGInf* - AN INDUCTIVE LEARNING SYSTEM FOR GI

Algorithm 6.1 ALGO61 *ILSGInf* learning strategy

Algorithm 6.2 ALGO62 *ILSGInf* refinement cycle

Algorithm 6.3 ALGO63 Main steps in partial parsing algorithm

Algorithm 6.4 ALGO64 Algorithm for initial grammar construction

Algorithm 6.5 ALGO65 Partial parsing algorithm

Algorithm 6.6 ALGO66 Generalization

Algorithm 6.7 ALGO67 Partial derivatives (*PaDe's*) construction

Algorithm 6.8 ALGO68 PaDe's construction for a sub-sentence

Algorithm 6.9 ALGO69 Heuristics for *PaDe's* sorting

ACKNOWLEDGEMENTS

Praise to Almighty *Allah*, Source and Goal of our true search in this life; praise for sustaining us in all our endeavors – throughout our lives. Although many people and bodies have to be thanked for their help in the accomplishment of this work, some have to be singled out. I would like to thank my supervisor Dr. A. Hamdi-Cherif for his guidance and unfading support throughout this many-year research. A special thank go to the members of the examination panel, namely Dr. A. Boukerram, Prof. M. Benmohamed, Dr. M. Aliouat and Dr. A. Refoufi; all of them got into the details of deciphering the manuscript and sending valuable constructive criticisms. My special thanks go to the administrative and scientific bodies of Université Ferhat Abbas, Sétif, especially Computer Science Department, where this work was initially formulated and came into conclusion. I also thank Computer College at Qassim University, Saudi Arabia, where some parts of research have been conducted, although, in most times at odd hours and within the hard constraints of a very busy working life. Many thanks go to our long standing family's friend Mohamed-Najib Harmas for the difficult administrative cobweb-like tasks he went through before this research was allowed to be defended, several months after its total completion. My heartfelt thanks are addressed to all members of my small and larger families for all their support and patience; some of these literally grew with this research like Mohamed, Saliha, Zineb, and Khadidja.

Grateful thanks to all…

ملخص

إن غالبية لغات البرمجة تقوم على قواعد نحوية مستقلة عن السياق. و إن الغرض من الاستدلال النحوي هو استنباط قواعد اللغة من مجموعة مدخلة من الجمل الصحيحة و أحيانا غير صحيحة. إننا نهتم في دراستنا هذه بالنحو المستقل عن السياق. وبما أن القواعد الشكلية المستنبطة في هذا النوع من النحو لا يدل فقط على طريقة تركيب الجمل بل على العلاقة بين الوحدات المختلفة المكونة للجملة و بالتالي يساعد على فهم المعنى.

بناء على ما سبق، نقترح إنتاج بيئة متبوعة بتنفيذ، من شأنها توحيد الجوانب المختلفة للبرمجة في إطار التعلم الآلي. إن الفكرة المحورية للعمل المقترح هي استخدام الاستدلال النحوي بوصفه إطارًا موحدًا لتحقيق هذا التكامل. بما أن أي برنامج هو أساسًا مجموعة من السلاسل، فإننا نبين أن استخدام الاستدلال النحوي يُمْكنه، زيادة على المساهمة في تكامل الجوانب المختلفة للبرمجة، أن يمتد أيضًا إلى مجالات أخرى أوسع نطاقاً.

يتمحور العمل حول المساهمات التالية :

- دراسة نظرية للغات البرمجة؛
- دراسة الاستدلال النحوي؛
- دراسة و تنفيذ لبيئة تدمج التعلم الآلي والمنطق من الدرجة الأولى؛
- دراسة و تنفيذ نظام مبني على منطق الدرجة الأولى لاستعماله في تحليل الجمل بطريقة منفردة أو بالاعتماد على التعلم؛
- دراسة و تنفيذ لخوارزم مبني على الحدسيات و استعماله لتحسين عملية التعلم في إطار الاستدلال النحوي، و في زمن محدود.
- التداخل بين الاستدلال النحوي و أنظمة التحكم الآلي.

إن هذا العمل يفتح مجالا واعدا للبحث في إطار المساهمة في تكامل لغات البرمجة، هادفًا إلى إثرائها بإضافة مستوى خاص بالتعلم الآلي.

الكلمات المفتاحية

تصنيف اللغات، لغات التصميم، النحو و أساليب إعادة الكتابة، تحليل الجمل، اللغات الصورية، ذكاء اصطناعي، استنتاج و برهنة القوانين، محرك الاستدلال، التعلم، اكتساب اللغة.

ABSTRACT

Most programming languages are based on context free grammars (CFGs). The purpose of grammatical inference is to infer a grammar, in our situation a CFG, from positive examples of sentences and possibly incorrect ones, for a given language. Based on these two fundamental definitions, we propose an environment followed by an implementation unifying different aspects of programming in machine learning settings. The central idea of this work is to use grammatical inference (GI) as a unifying framework for achieving this integration. Because any program can be considered as a string of characters, we show that the use of grammatical inference can not only unify different aspects of programming but also extend to wider areas of applications. The work sums up the following contributions:

- State of the art of language theory and of grammatical inference;
- Design and development of an environment integrating machine learning and first-order logic (FOL);
- Design and development of a FOL system for parsing sentences independently or with a learning module;
- Design and development of a heuristics-based polynomial-time complexity algorithm enhancing the learning process in grammatical inference.
- Interaction between grammatical inference and control systems.

The present work bears a promising line of research, contributing further to programming languages integration, aiming at the improvement of these languages with a machine learning layer.

ACM Categories and Subject Descriptors

D.3.1 [Formal definitions and theory], **D.3.2** [Language classifications], *Design languages*, **F.4.2** [Grammars and other rewriting systems], *Parsing,* **F.4.3** [Formal Languages], **I.2** [Artificial intelligence], **I.2.3** [Deduction and theorem proving], *Inference engine*, **I.2.6** [Learning], *Language acquisition.*

RESUME

La majorité des langages de programmation est basée sur les grammaires à contexte libre (CFG). Le but de l'inférence grammaticale est d'inférer une grammaire, en l'occurrence à contexte libre (CFG), à partir d'exemples de phrases correctes et éventuellement incorrectes, d'un langage donné. Partant de ces deux définitions fondamentales, nous proposons un environnement suivi d'une implémentation unifiant des aspects différents de la programmation dans le cadre d'apprentissage automatique. L'idée centrale du travail est donc d'utiliser l'inférence grammaticale comme trame unificatrice pour réaliser cette intégration. Dans la mesure où tout programme peut être considéré comme une suite de caractères, nous montrons que l'utilisation de l'inférence grammaticale peut non seulement unifier des aspects différents de la programmation mais aussi s'étendre à d'autres domaines plus vastes.

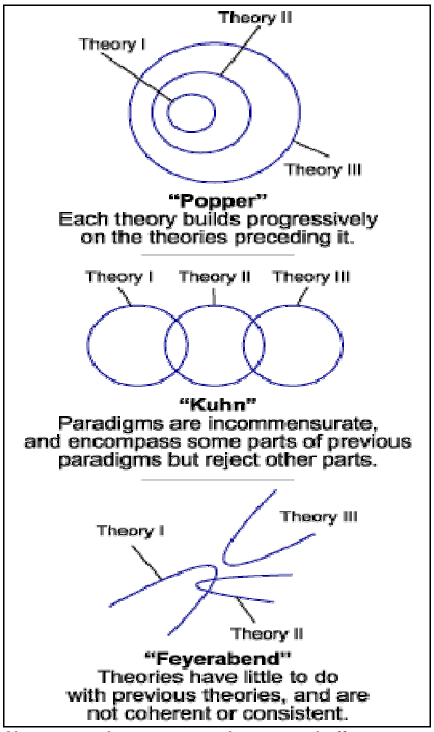
Le travail s'articule autour des contributions suivantes :

État de l'art de la théorie des langages ; État de l'art de l'inférence grammaticale ; Étude et développement d'un environnement intégrant apprentissage et logique du premier ordre ; Étude et développement d'un système fonctionnant en logique du premier ordre agissant comme analyseur syntaxique autonome ou en collaboration avec un module d'apprentissage ; Étude et implémentation d'un algorithme à complexité polynomiale, basé sur des heuristiques et destiné à l'amélioration du processus d'apprentissage dans le cadre de l'inférence grammaticale ; Interaction avec les systèmes de commande automatique.

Le présent travail est porteur d'une ligne prometteuse de recherche, et contribue davantage à l'intégration des langages de programmation, projetant de les enrichir par la caractéristique d'apprentissage qui leur fait actuellement défaut.

Catégories et descripteurs de sujets de ACM

D.3.1 [Définitions formelles], **D.3.2** [Classifications de langages], *conception des langages*, **F.1.1** [Modèles de calcul], **F.4.2** [Grammaires et systèmes de réécriture], *analyse syntaxique*, **F.4.3** [Langages formels], **I.2** [Intelligence artificielle], **I.2.3** [Déduction et démonstration de théorèmes], *moteur d'inférence*, **I.2.6** [Apprentissage], *acquisition de langages*



No matter where you stand, you need effort.

Diagram from : http://en.wikipedia.org/wiki/Portal:Scientific_method

CHAPTER 1
INTRODUCTION

1. Preliminaries

Most programming languages, whether imperative or declarative, are based on context-free grammars (CFGs). This remains true at a more refined level, with CFGs present in procedural, object-oriented, functional, logic programming and multi-paradigmatic languages. A sketchy summary of programming languages can be summarized as follows:

- *Conventional imperative languages:* These incorporate structured and/or object-oriented approaches with the high-level built-in functions and provide numerical processing like *FORTRAN*, *PASCAL* or *C/C++*, among others.

- *Advanced imperative approach*: These languages include numerical systems exemplified by the matrix environments like *MATLAB*™[1] supported by various visual programming aids like *Simulink*™ or symbolic general-purpose computer algebra systems (CASs) like *Mathematica*™[2] or *Maple*™[3] and their various corresponding toolboxes. Sophisticated CASE (computer aided software engineering) tools are also available, *e.g. Rational Rose*™[4]. Whether they are designed for number-crunching calculations or for symbolic processing or for modeling and implementation, these systems can be considered as one layer above the previous one.
- *Declarative approach:* The declarative approach focuses on what computational processes to undertake and not on how to perform them. This approach is represented by subcategories of functional programming (*e.g., LISP*) and logic programming (*e.g., Prolog*). On top of these, we find expert systems shells or generators like *NASA CLIPS*[5], essentially based on inductive logic programming (ILP), or its offshoots. This layer is still even more powerful in handling imprecise, non-numerical, and linguistic data. These environments/shells represent the favourite setting for *knowledge base* (KB) construction and inference engineering, a sub-filed of knowledge engineering.

2. Motivations

As far as scientific computation is concerned, most programming, modeling and simulation environments that have been developed in the last two decades or so, heavily concentrated on the following topics: matrix environments, computer algebra software (CAS), visual programming, object-oriented programming (OOP)

---

[1] MATLAB™ is a trademark of the Mathworks, http://www.mathworks.com
[2] Mathematica™ is a trademark of Wolfram Research, Inc., http://www.wolfram.com/mathematica
[3] Maple™ is a trademark of Maplesoft, http://www.maplesoft.com
[4] Rational Rose is a trademark of IBM™ http://www-01.ibm.com/software/rational/
[5] NASA CLIPS http://www.siliconvalleyone.com/clips.htm

simulation environments, coupled or hybrid systems that attempted to combine both numerical systems with advanced expert systems development aids. However sophisticated these systems might be, none considered the possibility of incorporating the learning layer in their implementation. Therefore none of these rightly deserves the overly-used appellation of intelligent system. For approximately five decades, these programming languages and environments contributed lines of implementation from basic algorithmic settings, incorporating sophisticated numerical and symbolic methods, to inferential / declarative methods. Notoriously, machine learning methods have not yet been fully applied in this domain. Our aim is to contribute towards this end using one machine learning approach, namely grammatical inference (GI).

3. Background and objectives

3.1 Process of inference

3.1.1 Inference in symbolic settings

In logic-based symbolic environments, the word *inference* is defined as the process of *reasoning* logically building new knowledge on the basis of available rules and facts. This process requires a problem-solving model, or paradigm, that organizes and controls the steps taken to solve the problem. One powerful paradigm involves the chaining of IF-THEN rules to form a given line of reasoning. There are three modes of chaining. If the chaining starts from a set of conditions and moves toward some conclusion, the method is called *forward chaining*. If the conclusion is known, for example, a goal to be achieved, but the path to that conclusion is not known, then reasoning backwards is used, resulting in *backward chaining*. *Hybrid chaining* is a combination of both; it might start with forward and shift to backward chaining.

These problem-solving methods are built into program modules known as *inference engines* that manipulate and use knowledge in the KB to form a line of reasoning.

One of the most important results of this problem-solving method is the emergence of *expert systems*. In symbolic settings, an expert system is a program that incorporates two main components - an inference engine, responsible for reasoning by entailing new facts, and a KB containing both *factual* and *heuristic* knowledge. *Factual knowledge* is that specific knowledge of the task domain that is widely shared, typically consisting of printed material like textbooks or journals, multimedia support found in Websites or any other electronic support. This knowledge is commonly agreed upon by those knowledgeable in the particular field. *Heuristic knowledge* is the less rigorous, more experiential, more judgmental knowledge of performance. In contrast to factual knowledge, heuristic knowledge is rarely discussed, and is largely individualistic. It is the knowledge of good practice, good judgment, and plausible reasoning in the field and mainly describes personal rules of thumb encompassing an "art of good guessing", personally acquired over lifetime training. As a result, expert systems are normally used to model the human decision-making process. Although expert systems contain algorithms, many of those algorithms tend to be static, *i.e.* they do not change over time.

3.1.2 Inference in knowledge-based systems (KBSs)

Abusively, knowledge-based systems (KBSs) are considered as synonymous of expert systems. In our account, we will make a distinction between the two categories programs and consider expert systems as a particular form of KBS. Expert systems usually rely on rule as a form of knowledge representation formalism. Obviously, not all knowledge is expressible as rules. That is why we need other types of KBs like neural networks, case-based reasoning genetic algorithms, intelligent agents, data mining, and intelligent tutoring systems [KC07].

3.1.3 Inference in learning settings

In learning settings, a program is intended to infer (or induce) an unknown result based on some past data. This operation involves a metric for attesting the quality of the results. In this context, *inference* implies the identification of a hidden function, given a set of its values. In particular, the learning of the syntax of the language is usually referred to as *grammatical inference* or *grammar induction* (GI); an important domain for both cognitive and psycholinguistic domain as well as for the domain of engineering and computation. GI deals with the problem of inferring (or learning or inducing) a grammar from some given data. Data, whether sequential or structured are composed from a finite alphabet, and may have unbounded string-lengths. By grammars, we intend only deterministic finite automata DFA, equivalent to regular grammars [Sip06] and some context free grammars (CFGs). If we refer to Chomsky hierarchy, only type-3 and subclasses of type-2 grammars, respectively, are concerned. In a machine learning perspective, we need the grammar, *i.e.* the concept learned, to predict and classify unseen data. The inferred grammar is also used as a model or a compressed representation of the input data. Early work in the field was set out in [Fu74]. But since 1994, more interests have been given to the field. An *International Conference on Grammatical Inference* (*ICGI*) is held every two years. The last one was held on September 2010 in Valencia, Spain. This increasing interest in the field is probably due to the following reasons:

- *Need for a more elaborate theory*; the GI community became aware of the fact that the hardness of even the easiest problem needs more theoretical attention and developments.
- *Expansion of applications*; the new fields where GI techniques can be applied are increasing every year.

3.2 Specific goals

3.2.1 Avoiding the "general problem solving (GPS)" syndrome

The question that interests us is: "How to integrate a GI-based machine learning layer in programming languages?" If we were to realize this, then solving similar problems using this type of programming languages will take less and less time to be solved, thanks to learning from examples of problems. However, this is a very distant end. We want to avoid the "general problem solving (GSP)" syndrome. Developed in the fifties, in the early days of artificial intelligence (AI), GPS was a program that tried to solve a very broad class of problems from theorem proof, geometric problems to chess playing [NS72]. GPS solved simple problems that could be sufficiently formalized such as the Towers of Hanoi. However, it could not solve any real-world problems because search was easily lost in the combinatorial explosion of intermediate states. In our account, we will therefore study only the syntactic level of languages.

3.2.2 Syntactic level - first

As a first step towards the realization of the objective of adding a learning layer to programming, we propose to start at the syntactic level. Because any program can syntactically be considered as a string of characters, we show that the use of GI can not only unify different aspects of programming but also extend to wider areas of applications such as control systems and self-assembly. As a result, the central idea for answering the central question above is to use grammatical inference (GI) as a unifying framework.

The purpose of GI is to infer a grammar, in our situation a context-free grammar (CFG), from positive examples of sentences and possibly incorrect ones, for a given language. In the attempt to address our fundamental issue, we propose an environment followed by an implementation. We show how the issue of GI can be reduced to learning heuristics. We describe our *GASRIA* GI system; fully designed, developed and tested as a system for GI capable of learning inductively a broad class of CFGs. The overall work consists of:

– The design and development of a first-order logic (FOL) environment used for parsing;

– The design and development of a knowledge base (KB) consisting of a rule base and a fact base describing the grammar rules under consideration;

– The design and implementation of the inductive learning *partial parsing algorithm* (PPA); an Earley-like algorithm capable of parsing sentences not as whole but as parts; [HH07b]

– The integration of FOL and an inductive learning within a coherent system; [HH07a]

– The study of some interactions between GI self-assembly and control systems; this latter being usually handled by matrix environments, [HH09a], [HH09b].

3.3 Main tools

The main tools can be summarized in two categories, namely, grammars and first-order logic (FOL).

3.3.1 Grammars and parsing

Grammars can be regular, context-free, context-sensitive and unrestricted. Context-sensitive and unrestricted grammars are more expressive, because the left-hand side of the productions can be more than just a single non-terminal. To start with, however, we aim at learning regular and CFGs, which have single non-terminals on the left side of production rules. The result is a reasoning or "intelligent" syntactic analyzer capable of inductive learning. One of the most important properties is that grammars have the ability to generalize over a specific language, *i.e.* to learn by induction. Therefore, it is possible to learn a grammar based on a set of sample sentences. We do not need to specify every sentence in a given language. This is the observation that led us to explore the possibility of using GI as a machine learning paradigm. Indeed, GI like most machine learning algorithms objective is to generalize over a set of (a preferably small number of) examples in order to obtain a more general model, by induction. Moreover, we need to handle strings of characters; hence the use of grammars and not other machine learning methods. On the other hand, the number of training examples has to be preferably small - less than six examples, in our tested cases.

3.3.2 Declarative programming and FOL

In addition, we combine GI with the declarative programming approach and specifically with first-order logic (FOL), to infer and use the grammar that has been produced for syntactic purposes. Declarative programming encompasses many different sub-fields such as constraint programming, domain-specific languages (*e.g.* *SQL*-based, *XML*-based), functional programming (*e.g.* *Lisp*, *Scheme*), and logic programming (*e.g.* *Prolog*).

The motivation for using the declarative approach is that this paradigm requires what computation should be performed and not how to compute it. It has a clear correspondence with mathematical logic and specifically with FOL. The knowledge base containing FOL-based rules and facts allows the entailment of new facts, thus contributing to the GI process.

4. Organization of the manuscript

In this manuscript, we explain the main building blocks of the proposed solution; each one of these blocks in an independent chapter. The work is structured around the following components:

- *State of the art of language theory:* Chapter 2 describes the theory of languages that is necessary for explaining the main results.
- *State of the art of GI:* Chapter 3 reports the theoretical background of GI and discusses the most important related algorithms, systems and applications.
- *GASRIA:* In an attempt to integrate GI and FOL, Chapter 4 explains the design and development of an architecture, namely *GASRIA* as a complete and integrated system for GI. Its main modules are explained in two subsequent independent chapters. The main idea is based on a novel machine learning algorithm, namely the *partial parsing algorithm* (PPA), coupled with a FOL-based system.
- *EXINF:* Chapter 5 describes aspects related to first-order logic (FOL) and declarative systems. It discusses an in-depth description of one of the components

of the solution, namely the design and development of *EXINF* as a FOL-based system. *EXINF* characteristics are the possibility of use as a stand-alone system or as a support for partial parsing. *EXINF* is presented as a knowledge-based system (KBS) using dynamic facts, necessary for parsing. These facts are the translation of input sentences into syntactical rules. As shown in the examples, important parsing steps are undertaken using *EXINF*.

- *ILSGInf:* Chapter 6 reports the design and implementation of one machine learning environment called *ILSGInf*. It is based on the *partial parsing algorithm* (PPA). The chapter explains specific aspects of grammar inference, including regular and CFGs. It also describes the experimental PPA capability and validation as a core component of *ILSGInf*.
- *Interactions:* Chapter 7 reports application areas of some of our results. Control systems, mainly, and self-assembly, peripherally, are discussed as possible applications fields.

The work ends with a conclusion summing up results and recommendations with prospective developments to address open issues.

CHAPTER 2
SOME CONCEPTS OF FORMAL LANGUAGES

1. Introduction

The elaboration of the theoretical "Universal-Algorithm Machine" and the invention of the vacuum tube gave birth to the idea of a stored-program computer. The goal was to convert the electronic computer to a real-life model of the "Universal-Algorithm Machine". Along with the concept of programming a computer, came the question: "What is the 'best' language in which to write programs"? As a result, different programming languages were developed, but they apparently shared the same possibilities and limitations.
Many questions rose: what is language in general? How do people learn it? Linguists created the subject of mathematical models for the description of languages to answer these questions. Consequently, the computer took on linguistic abilities. It became a word processor, a translator, an interpreter of simple grammar, a compiler of a programming language, a speech recognizer, and now we try to give it the

ability to learn languages, under the constraint that we are not yet able to understand how human do that.

2. Preliminaries

We start by giving some mathematical definitions, which are of interest to us. They can be found in any book dealing with concepts of formal language [Gdd08], [deH10] [Sip06].

- An *alphabet* is a finite non-empty set of symbols or letters, often denoted by Σ.
- A *string* ω over an *alphabet* Σ is a sequence $\omega = a_1 \ldots a_n$ of letters $a_i \in \Sigma$.
- *Length* of ω, noted $|\omega|$ is the number of letters constructing it, in this example $|\omega| = n$.
- *Number of occurrences*: Given $a \in \Sigma$, $|\omega|_a$ denotes number of occurrences of the letter a in the string ω.
    - The *empty string* denoted by λ (or by ε) such that $|\lambda| = 0$.
    - Given two strings u and v, we define $u.v$ (or simply uv) as the *concatenation* of u and v and $|uv| = |u| + |v|$.
- If ω is a string, $\omega = a_1 \ldots a_n$ we note $\omega^R = a_n \ldots a_1$ as the *reversal* of ω.
- Σ^* is the set of all finite strings over Σ. We define $\Sigma^+ = \{x \in \Sigma^* : |x| > 0\}$ and $\Sigma^{<n} = \{x \in \Sigma^* : |x| < n\}$
- The string u is a *substring* of a string x if there are two strings l and r such that $x = l.u.r$.
- We define $|x|_u$ as the number of occurrences of the substring u in the superstring x.

- The string u is a *subsequence* of a string x if u is obtained by removing some letters from x. More precisely, u is a subsequence of x if there is a sequence of indices $i=(i_1,\ldots, i_{|i|})$ where $1\leq i_1 \leq\ldots\leq i_{|i|}\leq|x|$ and $u_j = x_{i_j}$. We note $u = x(i)$.
- *Orders in strings*: there are four ordering relations between strings based on the *total order relation* over elements of Σ, noted $\leq_{alpha}$ called alphabetical order. These four ordering relations are defined as:
    - *Prefix order*: $x \leq_{pref} y$ *if* $\exists w \in \Sigma^*$ such that $y = xw$.
    - *Lexicographical order*: $x\leq_{lex}y$ *if* $x\leq_{pref}y$ or ($x=uav$, $y=ubw$ *and* $a\leq_{alpha}b$)
    - *Subsequence order:* $x\leq_{subseq}y$ if x is a subsequence of y
    - *Length-lex order :* $x\leq_{length\text{-}lex}y$ if $|x|<|y|$ or ($|x|=|y|$ and $x\leq_{lex}y$)

    We can assign with all these orders the corresponding *strict orders*

    $<_{alpha}$, $<_{pref}$, $<_{subseq}$, $<_{length\text{-}lex}$.

3. Languages

A language is a certain specified set of *strings*, where strings have symbols from a specific alphabet. A language L over Σ, $L\subseteq\Sigma^*$.

3.1 Operations on languages

Certain operations can be done on languages: let L_1, L_2 be two languages

- *Union:* $L_1\cup L_2=\{ x\in\Sigma^*: x\in L_1$ OR $x\in L_2\}$
- *Intersection:* $L_1\cap L_2=\{x\in\Sigma^*: x\in L_1$ AND $x\in L_2\}$
- *Product*: $L_1.L_2 = \{ uv : u\in L_1, v\in L_2\}$
- *Powerset:* $L^0=\{\lambda\}$, $L^{n+1} = L^n.L=L.L^n$
- *Star*: $L^* = \cup_{i\in N} L^i$, where N is the set of positive or null integers.
- *Complement*: $L' = \{w\in\Sigma^*: w\notin L\}$, $L_1\setminus L_2$ is the complement of L_2 in L_1

- *Symmetric difference*, $L1 \oplus L2 = L_1 \backslash L_2 \cup L_2 \backslash L_1$

3.2 Languages models

There are different ways to allow computation of languages. Hence, we find methods to generate grammars, to recognize finite automata, to define regular expressions, and recently to use topological operations to represent a language. The work in [Cho59] was the first to classify languages into four classes using four types of grammars.

3.2.1 Formal grammars

Definition 1 - A formal grammar G has four components $G=<\Sigma, N, P, S>$ where

- Σ is an *alphabet*, called also *set of terminals*.
- N a set of symbols, called *non-terminals* or *variables*, with the restriction that Σ and N are disjoint.
- S a special non-terminal symbol, called a *start symbol*.
- P is a *set of production rules*, each one is of the form $\alpha \rightarrow \beta$ or sometimes noted (α,β).

Definition 2 - A *regular grammar* is a formal grammar where:

$$P \subset (N \times \Sigma^*) \cup (N \times \Sigma^*.N) \cup (N \times N.\ \Sigma^*)$$

Definition 3 - A *context-free grammar* (CFG) is a formal grammar where:

$$P \subset N \times (\Sigma \cup N)^*$$

Definition 4 - A *context-sensitive grammar* is a formal grammar where:

$P \subset (N \cup \Sigma)^*.N.(N \cup \Sigma)^* \times (\Sigma \cup N)^+$, where for each (α, β) in P, $|\alpha| \leq |\beta|$

Definition 5 **-** An *unrestricted grammar* is a formal grammar where $P \subset N^+ x(\Sigma \cup N)^*$

3.2.2 Automata

We can informally define an *automaton* (plural *automata*) as a mathematical model of a machine that recognizes a set of strings. There are different types of such models that differ from each other essentially in the amount of memory they use. These are finite state automata (FSA) and push-down automata (PDA).

3.2.2.1 Finite state automata (FSA)

Finite state automata (FSA) were developed in 1950's. There two types of finite state automata, namely:

- *Non deterministic finite automaton (NFA)* is a sextuple $A = <\Sigma, Q, I, F_A, F_R, \delta_N>$ where:

- Σ is an *alphabet*,
- Q is a finite set of *states*,
- $I \subseteq Q$ the set of *initial states*,
- $F_A \subseteq Q$ is the set of *final accepting states*,
- $F_R \subseteq Q$ is the set of *final rejecting states*,
- $\delta_N : Q \; x \; (\Sigma \cup \{\lambda\}) \rightarrow 2^Q$, is the *transition function*, and 2^Q is the *powerset* of Q.

- *A deterministic finite automaton (DFA* or *FA)* is obtained from an NFA if I is reduced to only one initial state, and the image given by δ_N is only one state, and hence $\delta_N : Q \; x \; (\Sigma) \rightarrow Q$. Note that the empty transition is also excluded.
- A string $\omega = a_1 \ldots a_n$ is recognized by an automaton A, if there is a sequence of states starting at an initial state $q_0, \ldots, q_m$ and a sequence of letters $b_1 \ldots b_m$, b_i in $\Sigma \cup \{\lambda\}$ (in the case of NFA) or in Σ (in the case of FA) and $a_1 \ldots a_n = b_1 \ldots b_m$ such that $\forall j \in [1..m]$, $q_j \in \delta_N(q_{j-1}, b_j)$. $q_0 \in I$ and $q_m \in F_A$.

We note that for any NFA, there is an FA which recognizes the same language (FA = NFA).

3.2.2.2 Push-down automata (PDA)

Here, we need memory to keep some intermediate information. Push-down automata (PDA) uses memory that has a last-in first-out structure, LIFO or stack. A PDA is an FA with a stack. A PDA is eight-tuple = <Σ, Γ, I, F_A, F_R, NB_{PUSH}, B_{READ}, B_{POP}> where:

- Σ is the *alphabet of input data*,
- Γ is the *alphabet of the stack*,
- I is the *initial state*,
- F_A is the set of *accepting states*,
- F_R is the set of *rejecting states*,
- NB_{PUSH} is the set of *non-branching states* that only push letter in the stack,
- B_{READ} is the set of branching states that *read letters from the input*, and
- B_{POP} is the set of branching states that *read letters from the stack.*

PDA can be divided into two categories based on determinism:

- A PDA is said to deterministic (DPDA), if for each input string there is only one way in the machine. Otherwise, it is non-deterministic and it is simply noted PDA. Unlike FAs, DPDA is not equivalent to PDA. Non-determinism adds a significant power to PDA.
- A string $\omega = a_1 \ldots a_n$ is recognized by a PDA if, starting at initial state and following a path of labelled and unlabelled edges according to different read input letters and stack characters, the process ends at accepting state.

3.2.3 Regular expression

A regular expression over Σ is defined recursively as follows:

- the empty set ϕ, the empty character λ and $\forall a \in \Sigma$ are regular expressions over Σ.

- if r_1, r_2 are two regular expressions, then (r_1), $r_1.r_2$, r_1+r_2, r_1^* are regular expressions.

Regular expressions are equivalent to FA and to NFA, by Kleene's theorem.

3.2.4 Topological consideration

After defining some metrics and distances over string and especially the edit distance, a language can be considered as a topology. Hence, the notion of ball can be introduced. Ball of strings is the set of all strings presenting a distance from special string (the centre) less or equal to some value *r* (the radius of the ball) [deH10].

4. Chomsky languages hierarchy

Chomsky [Cho59] defined four classes of languages as a hierarchy. These classes of languages are from the bottom regular languages (type-3), context-free languages (type-2), context-sensitive languages (type-1) and recursive enumerable languages (type-0).

Because it is a hierarchy, each language in a class is also an element of the superior class. The distinction between language classes can be done by examining the structure of the production rules of their corresponding grammars, or the nature of the machines which can be used to recognize them.

4.1 Type 3 - Regular languages

A language *L i*s a regular language if it can be generated by a regular grammar. This class of languages can be defined by regular expressions and can be recognized by an FA. Any finite language is regular.

4.2 Type 2 - Context-free languages

A language *L* is a context-free language (CFL) if it can be generated by a context-free grammar (CFG). This class of languages is recognized by PDAs. Deterministic PDAs recognize a subclass of CFLs called deterministic CFLs while nondeterministic PDAs can recognize larger class of CFLs.

For type 1 and 0 languages, we just cite them as elements of Chomsky hierarchy. We do not expand our study to these because they are not studied in grammatical inference (GI) due to their complexity.

4.3 Type 1 - Context-sensitive languages

A language *L* is a context-sensitive language if it can be generated by a context-sensitive grammar (CSG). Since more than one symbol is permitted on the left hand side, symbols surrounding the non-terminal concerned by the replacement are known as *context*. The automaton which recognizes a context-sensitive language (CSL) is called a linear-bounded automaton (LBA) *i.e.* basically an NFA/FA which can store symbols in a list.

4.4 Type 0 - Unrestricted (free) languages

A language *L* is an *unrestricted language* if it can be generated by an unrestricted grammar. Free grammars have absolutely no restrictions on their grammar rules, except of course, that there must be at least one non-terminal on the left-hand-side. The languages generated by such grammars are *recursively enumerable* (RE). The type of automata which can recognize such a language is basically an NFA/FA with an *infinitely-long list*. This is called a Turing machine (TM).

The hierarchy can be summarized in the table below. Type-1 and Type-0 languages are recognized by Turing machines (not studied here) which were developed in 1930's and 1940's.

Table 2.1 TAB21 – Chomsky languages hierarchy

Type	***Language Class***	***Grammar***	***Automaton***
3	Regular language	Regular	NFA or FA
2	Context-free language	Context-free	Push-down automaton (PDA)
1	Context-sensitive language	Context-sensitive	Linear-bounded automaton
0	Recursive enumerable language	Unrestricted (free)	Turing machine (TM)

In the following sections, we concentrate our study on regular and context-free languages because of their wide implications in different learning methods and programming languages.

5. Regular languages

5.1 Introductory example

A regular language is any language that can be recognized by an automaton, defined by a regular expression or generated by a regular grammar. In general, we can use regular languages whenever we need a limited amount of memory. For examples, we use them in text editors, automated opening doors, elevators, to cite but a few.

For example, we give here a language and its three equivalent representations using Kleene's theorem, for simplicity we consider $\Sigma = \{0, 1\}$; with L accepting strings containing 001 **.**

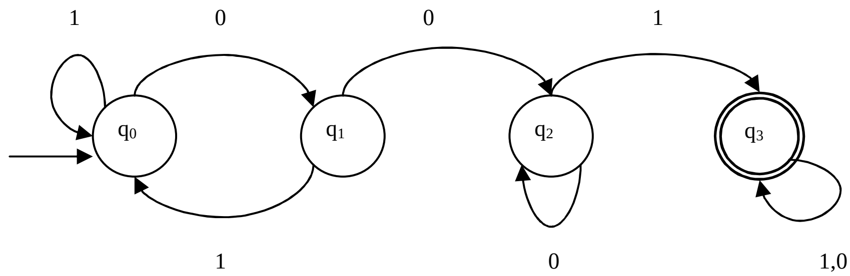


Figure 2.1 DIAG21 An FA that recognizes strings containing 001

A regular expression that defines L is (1* + (01)*) 00 0* 1(0+1)*

A regular grammar that generates L is:

$$S \rightarrow 1S \mid 0A \; ; A \rightarrow 0B \mid 1S ; B \rightarrow 0B \mid 1C ; C \rightarrow 0C \mid 1C \mid 0 \mid 1;$$

5.2 Characteristics of regular languages

Regular languages are closed under union, intersection, Kleene star, concatenation and complementation. We can consider union, star and concatenation as regular operations. The following definitions summarize the main characteristics of regular languages [Sip06].

- *Quotient:* if *L1* is regular, *L2* is any language, then *Pref(L2 in L1)* is also regular, where *Pref(L2 in L1)* is the set of all strings that can be placed in front of some elements in *L2* to produce some elements in *L1*.
- *Equivalence:* two NFAs are equivalent if they recognize the same language. This problem is decidable. Equivalence between two regular expressions is also decidable.
- *Finiteness:* whether an NFA accepts a finite or infinite language is decidable. If an NFA has N states then it accepts an infinite language if and only if it accepts an input string with ω such that $N \leq |\omega| < 2N$.

- *Emptiness:* if an NFA has N states, then if it accepts any word then it accepts words of length less or equal to N.
- *Membership problem:* it is the problem of deciding if some string is recognized (defined or generated) respectively by a NFA, (regular expression or regular grammar). This problem is decidable.
- *Pumping lemma:* if L is a regular language, then there is a number p (the pumping length) where, if w is any string in L of length at least p, then w may be divided into three pieces, $w = xyz$, satisfying the following conditions:
    1. For each $i \geq 0$, $xy^iz \in L$,
    2. $|y| > 0$, and
    3. $|xy| \leq p$

P is always taken as number of states in the automaton that recognizes the language.

6. Context-free languages (CFLs)

Any language that can be recognized by a PDA or generated by a CFG is a CFL. The set of CFLs is larger than that of regular languages.

6.1 Examples of CFLs

- For $\Sigma = \{a, b\}$, $L1 = \{a^nb^n, n \geq 0\}$
- $L2 = \{\omega \in \Sigma^* \mid \omega$ has same number of a and $b\}$ is a CFL.
- $L3$ can be generated by the CFG $S \to aSb \mid SS \mid \lambda$.
- $L4 = \{\omega\omega^R \mid \omega \in \{0, 1\}^*\}$ can be recognized by the PDA described in Figure 2.2 below.

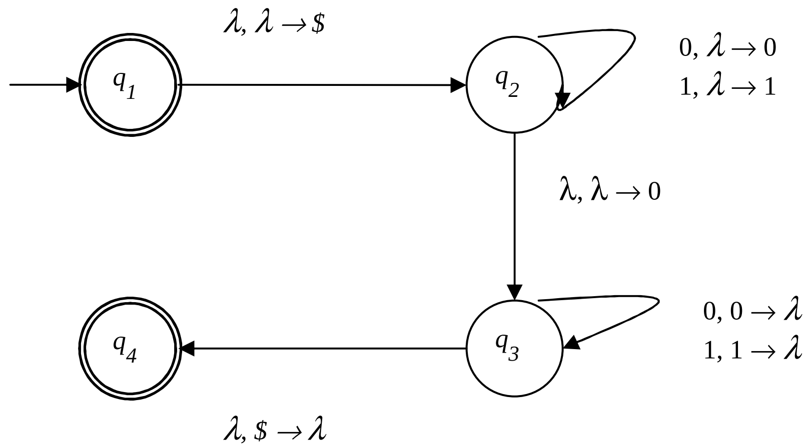


Figure 2.2 DIAG22 PDA recognizing $\{\omega\omega^R \mid \omega \in \{0, 1\}^* \}$ [Sip06]

We can interpret this figure as starting by pushing the symbols that are read onto the stack. At each point, non-deterministically guess that the middle of the string has been reached and then change its behavior into pop operation. For each symbol that has been read, check its similarity with the popped symbol.

6.2 Applications of CFLs

All programming languages and compilers are based on CFLs. CFGs were first used in the study of human languages. CFLs have been applied to a variety of fields from user behavior modeling to DNA (DeoxyriboNucleic Acid) structure. Note that these complex systems can be interpreted as languages, in general and grammars, in particular.

6.3 Characteristics of CFLs

- CFLs are closed under union, product and Kleene star operations.

- *Complements:* complement of a CFL may not be a CFL. This type of language is not closed under complementation.
- *Intersection:* CFLs are not closed under intersection. However, intersection of a CFL and a regular language is always a CFL.
- *Equivalence:* a CFG is equivalent to PDA but deterministic PDA is not equivalent to PDA.
- *Finiteness and emptiness:* it is decidable whether a CFG generates a finite or an infinite language and whether it generates any string (if $L(G) = \{\}$).
- *Membership:* Membership tells whether a string belongs to a given language. This is done through parsing.
- *Empty production:* if L is a CFL generated by a CFG that includes λ-productions, then there is a different CFG with no such productions and that generates L or L-$\{\lambda\}$.
- *Chomsky Normal Form (CNF):* for any CFL L, the non-empty strings of L can be generated by a CFG with each production is one of the forms $A \rightarrow BC$ or $A \rightarrow a$.
- *Pumping Lemma:* if L is a CFL, then there is a number p called the pumping length, such that, if w is any string in L of length at least equal to p, w may be divided into five substrings $u\ v\ x\ y\ z$ satisfying the following three conditions:
  - $|vy| > 0$
  - $|vxy| \leq p$
  - for each $i \geq 0$, uv^ixy^iz in L

Pumping lemma for regular languages (*resp.* CFLs) is in general used to prove that a language is not regular (*resp.* CFL).

6.4 Relationship between regular and CFLs

- All regular languages can be generated by CFGs (they are CFLs)
- If all the productions in a given CFG fit one of the two forms:

$A \to \omega B$ or $A \to \omega$ or $A \to \lambda$, where A and B are nonterminals and $\omega \in \Sigma^*$, then the language generated by this CFG is regular.

- A CFG is called a *regular grammar* if each of its productions is of one of the two forms $A \to \omega B$ or $A \to \omega$ where A and B are nonterminals and $\omega \in \Sigma^*$.

7. Parsing

Parsing a sentence using a grammar is determining how this sentence could be formed from the rules of the grammar starting at the special non-terminal.

Derivation is the sequence of applications of the rules that produces the specified string of terminals from the starting symbol.

Example

Let the productions be: $S \to aS$ (1)

$S \to \lambda$ (2)

Generate the sentence *aaaaaa*

$$S \Rightarrow aS \Rightarrow aaS \Rightarrow aaaS \Rightarrow aaaaS \Rightarrow aaaaaS \Rightarrow aaaaaaS \Rightarrow aaaaaa$$

All strings of terminals and non-terminals in the derivation and before reaching the final sentence are called *working strings.* This derivation can be traced as a tree called *parse tree.* We concentrate here on syntactic parsing of formal languages. There are three different approaches

7.1 Top-down parsing

Starting with the symbol S, we try to find some sequence of productions that generates the target word. This is done by checking all possibilities for left-most derivations. We follow each branch until it becomes clear that this branch can no longer present a viable possibility.

A general form of a top-down parsing is known as *recursive-descent parsing* that may involve *backtracking.*

In some cases, we can write grammars such that a recursive-descent parsing can be applied with no backtracking. This type of parsing is called *predictive parsing*.

7.2 Bottom-up parsing

Starting with the word, we try to find the last few productions to reach the starting symbol. A general form of *bottom-up parsing* is known as *shift-reduce parsing.*

7.3 Hybrid parsing

The first and the second approaches are combined so that the parsing is optimized. An important bibliographical study of parsing algorithms can be found in [ALS07].

8. Conclusion

In this chapter, we have summarized the most important notions of formal languages of interest to us. The central ideas remain those related to parsing and CFGs. The next chapter is dedicated to grammatical inference *i.e.* how to infer a grammar for a language from a set of examples (or sentences).

CHAPTER 3
STATE OF THE ART OF GRAMMATICAL INFERENCE

1. Introduction

In order to study the state of art of grammatical inference, we proceed as follows. In Section 2, we describe the theoretical models available for GI. We start with the *identification in the limit*, as defined in the late sixties in [Gol67], followed by the seminal contributions of the eighties represented by the so-called active learning as defined in [Ang81], and ending with PAC (probably approximately correct) learning due to [Val84]. Section 3 reports the main algorithms used in GI. We only stress those that deal with regular grammars and CFGs. Section 4 is devoted to applications of GI. Given the range of these applications, it clearly appears that it is a multidisciplinary domain spanning pattern recognition [Cas90], bioinformatics

[Coh04], syntactic pattern recognition [Luc94], DNA computers [Adl94], and robotics [Kla07], among others.

2. Theoretical models for grammar inference

A computer program is said to *learn* from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E* [Mit97].

In GI, experience *E* is the linguistic input, the task *T* is a grammar, and performance measure *P* is any metric that provides a measure of difference between the grammars inferred and a target grammar. Learning languages are based on inductive inference [AS83]. We can specify a classical inference problem by the following points, expanded in Figure 3.1 below.

/* Methodology 3.1 */
/* METH31 */

/* Methodological Steps – Inference Problem */

- **What is the class of concepts or rules being considered?**
- **What is the hypothesis space (descriptions)?**
- **Find an admissible presentation: the information and the way it is presented in.**
- **What is the class of methods under consideration?**
- **What is the criterion of a successful inference, *i.e.* convergence?**

Figure 3.1 METH31 Methodological Steps – Inference problem

For GI, the hypothesis can represent FAs, regular expressions, regular grammars, CFGs, or tree-grammars. The examples are typically strings or some special graphs. Obviously, the methods used to tackle these different hypotheses are different and so are the algorithms used. However, we can single out three main theoretical models that were established for this purpose.

2.1. Identification in the limit (learning from text)

2.1.1 Definition

The seminal work in [Gol67] established a theoretical model for on-line and incremental learning destined to learning languages.
This contribution asserts the following points:

1. A *presentation* is a function f: N→ X
    - Where N is the set of integers and X is any enumerable set,
    - f is associated to a language L through a function *yields(f)=L*.
    - If $f(N) = g(N)$ then *yields(f) = yields(g)*.
2. A presentation is a *text* or an *informant*
    - A *text presentation* of a language $L \subseteq \Sigma^*$ is a function $f : N \rightarrow \Sigma^*$, $f(N)=L$, with f an infinite succession of elements of L, where each one must appear at some instant.
    - An *informant presentation* $f: N \rightarrow \Sigma^* X \{+,-\}$ such that $f(N) = (L,+) \cup (L,-)$.
      In this case, f is an infinite succession of labelled examples, positive or negative elements of Σ^*, and where each one must appear at some instant t.
3. A *learning function*, called *inductive machine*, which, after each example, returns a hypothesis.
    - The *learning function* takes as input n elements $(e_1,\ldots, e_n)$ of f.

- It returns some hypothesis $H_n\ (e_1, \ldots, e_n)$.
- The target language is identified in a finite time t, if the learning function attains a fixed point, *i.e.* a point in time after which it does not change with the new inputs.

4. In this case, we say that the class of languages to which the target language belongs is *identifiable in the limit*.

2.1.2 Characteristics

- *Identifiability* is a property of a class of languages, not of an individual language. It is the characteristics of a class of languages for being *identifiable*. We say that a class of languages CL is *identifiable* if and only if a learning function that identifies CL exists.
- A *learning function* LF identifies a class of languages CL if and only if it identifies any language L of the class CL.
- A learning function identifies a language L if and only if it identifies any presentation of the language.
- A learning function identifies a presentation f, if and only if, the learning function converges to h and $yields(f) = yields(h)$.
- If we are given examples and counter-examples of the language to be identified, and each individual string is sure of appearing, then at some point the inductive machine will return the correct hypothesis.
- If we are given only the examples of the target, then identification is impossible for any super finite class of languages, *i.e.* a class containing all finite languages and at least one infinite language.

2.2 Active learning

This model was set out in [Ang81]. This framework concerns the learning with additional information, queries asked from an *oracle*.

- The *oracle* is a device that knows the target language. When it is asked, the oracle gives correct answers with no probabilities.
- Different types of queries are established: let a string w (in general), a target language TL and a grammar G.
  - Membership queries: the question asked to the oracle is "Is $w \in TL$ true?"

    For a *membership query*, we have $MQ: \Sigma^* \rightarrow$ {yes, no}.
  - *Equivalence queries*: the question asked to the oracle is "Is $L(G) = TL$?"

    * *Weak equivalence query WEQ:* $g \rightarrow$ {yes, no} or

    * *Strong equivalence query SEQ:* $g \rightarrow \{yes\} \cup \Sigma^*$
  - *Inclusion queries*: the question asked to the oracle is "*Is* $L(G) \subseteq TL$?"

    Inclusion query: *SSQ:* $g \rightarrow \{yes\} \cup \Sigma^*$
- Different system depends on the type of queries used.
  - *Only membership queries* $\Gamma = \{MQ\}$
  - *All types of queries* $\Gamma = \{MQ, WEQ, SSQ\}$
  - *Minimum adequate teacher* MAT with $\Gamma = \{MQ, EQ\}$.

2.2.1 Definition

A class of grammars g is identifiable with a polynomial number of queries if there is an algorithm *alg* such that:

- For each grammar G in g, *alg* identifies G with polynomial (in $|G|$) number of queries in Γ.

- This algorithm does each update in time polynomial (in $|G|$) and in the length of the longest counter-example.

2.2.2 Characteristics of active learning

- With an MAT, we can learn FA and also a variety of other classes of grammars.
- It is difficult to see how powerful is really an MAT.
- It is easy to find a class, a set of queries and provide and algorithm that learns using them.
- It cannot learn FA from (a polynomial number of) membership queries alone or from equivalence queries alone.
- With only a polynomial number of examples, or with a polynomial number of mind changes, learning FA is not possible.

2.3 PAC learning

2.3.1 Definitions

Probably approximately correct (PAC) learning was proposed as an alternative model for identification in the limit [Val84]. While in this latter, it is assumed that a finite time for learning an *exact hypothesis*, PAC allows for a hypothesis to identify a target language with certain probability and this identification is performed in polynomial time.

A hypothesis *h* is said to be *approximately correct* if and only if $Pr_D([h(x)\neq L(x)]<\varepsilon$

Where:

- C is a class of languages and H is a set of hypothesis.
- $L \in CL$ and $h \in H$.
- ε is some positive value.

PAC-learnability

Let us take CL to be defined over a set of example sentences from the alphabet Σ of length n. CL is said to be *PAC-learnable* by the learner if, for all grammars $g \in C$, given a distribution D of examples over Σ^*, ε and δ constrained by $\varepsilon > 0$ and $\delta < 0.5$, the learner will, with $Pr_D > (1 - \delta)$, output a hypothesis grammar g_h with $g_h \in G$ such that $error_D(g_h) < \varepsilon$.

This means that the inference is done with a probability Pr_D with an error as small as prescribed. For so doing, we need to measure the difference between the target and the inferred grammars using an error metric. The *error*, denoted $error_D(g_h)$, of the hypothesis grammar g_h with respect to the target grammar g_t is the probability that g_h and g_t disagree on the classification of randomly-drawn instances x from distribution D.

2.3.2 Characteristics

- A class CL is *polynomially PAC-learnable* if it is PAC-learnable in a polynomial time in $1/\varepsilon$, $1/\delta$, n and the size of g.
- *PAC-learning of FA* is still an open problem but it is believed to be impossible.
- Assumption that the PAC learning will be held under any distribution can lead to abnormal examples.

2.4 Relation between active learning and PAC learning

A class is polynomially identifiable by equivalence queries if and only if it is polynomially PAC-learnable [Ang88].

3. Algorithms for GI

Classes of grammars are studied at levels in reverse order of their classification in the Chomsky hierarchy [Cho59]. A lot of work is done in the field of regular grammars (type 3) and less work for the class of CFGs (type 2). Some works have considered the possibility of extracting grammars from programs [CMZ05]. These two types concern formal languages. Important interest is given to these two classes because there are efficient algorithms that solve the decidable problem of membership of an element to the associated languages. Case-sensitive grammars (type-1) and unrestricted grammars (type-0) are generally used for natural language processing. In the following, we only concentrate our survey on grammars for formal languages.

3.1 Algorithms for regular grammars

Regular grammars are widely studied in the domain of grammar inference for several reasons:

- They are simple.
- They are important in syntactic pattern recognition.
- They have a well-known set of properties such as decidability of membership and equivalence questions.
- There exist efficient parsers for them.

For each regular grammar, there exist a set of finite state automata which recognize language of this grammar. The problem of inferring a regular grammar is that of learning a finite state automaton from both positive and negative data. This problem can be formally established as a decision problem as described in Fig. 3.2 below.

/* Methodology 3.2 */
/* METH32 */

/* Methodological Steps – Inference Problem */

- **Given**
  - **a finite alphabet**
  - **two disjoint sets of examples *D*+ and *D*-**
  - **an integer *n***
- **Find**
  A deterministic finite state automaton (*FA*) consistent with *D*+ and *D*-
- **Subject to the constraints**
  A number of states less than or equal to *n*.

Figure 3.2 METH32 Combinatorial problem associated with a FA.

This is known as the combinatorial problem associated with a FA. It was proved that this problem is NP-complete [Gol67]. The problem of finding polynomially larger FA than the minimum FA, consistent with the input data, is NP-hard [PW93]. The learning of FA is also extended to the non-deterministic finite state automata NFA. We give below some algorithms concerning the two recognizers.

3.1.1 Complexity for inferring regular grammars

The search space of regular grammar inference depends on the total number of states in the maximal canonical automaton. We usually build a lattice. However, even for a small number of states it is not practical to explicitly build the lattice. For example, with only 4 states, 15 different automata can be obtained by merging states.

With 10 states the number of different automata is increased to 115,975. To overcome this problem, we usually rely on heuristic [Sav04] or incremental methods [PV96].

3.1.2 Learning FA

The importance of work on FA is justified by the fact that the algorithms treating the inference problem for FA can be adapted for larger classes of grammars, for instance even linear grammars [Tak88], sub-sequential transducers [Knu94] or tree grammars. They can even be transposed to solve the inference problem for CFGs, when the data is presented as unlabelled trees [Sak92].

3.1.2.1 Trakhtenbrot and Barzdin [TB73]

In [TB73], the authors study the case where all data length is greater than a certain value. For this case, there exists an algorithm that identifies FA. They describe a greedy learning algorithm with polynomial-time complexity for constructing the smallest FA consistent with complete labelled training set. The input is the prefix tree acceptor (PTA). This tree is collapsed into a smaller graph by merging all pairs of states that represent compatible mappings from string suffixes to labels. This process is called contraction procedure.

3.1.2.2 Gold's algorithm [Gol78]

This algorithm tries to find the minimum FA compatible with the data. The states of the FA are strings or prefixes of strings. An *observation table OT(S,E)* is constructed and contains the whole information. *S* is a set of states and *E* is some experiment. The algorithm will find the correct automaton when a characteristic sample is included in the data. It has a polynomial-time complexity.

3.1.2.3 RPNI algorithm [OG92]

A regular positive negative inference (RPNI) algorithm is based on state merging method, [OG92]. In this case also, a prefix acceptor automaton is initially constructed on the basis of positive data set. An iterative merging process is performed but corrected by a set of negative data. Many other algorithms followed RPNI, intending to improve the order of states to be merged. For instance, BLUE* [Seb03] is an adaptation of RPNI that deals with noisy data.

3.1.2.4 Traxbar algorithm [Lan92]

Traxbar algorithm is a variant of the algorithm exposed above [TB73]. It is used in the case where both target machine and training set are drawn randomly by a uniform distribution [Lan92]. In this work, it is experimentally shown that *Traxbar* can learn approximately a FA if the training set and the machine are generated randomly instead of being chosen by an adversary. This had a great impact on the induction community since languages of infinite size become learnable.

3.1.2.5 Dupont's lattice setting [DMV94]

This work considers the grammar inference as a "generalization of search" problem, inferring a grammar is reduced to the process of searching for a target grammar in the search space. Regular inference may be defined as the discovery of an unknown automaton *A* from which an observed positive sample *I*+ is supposed to have been generated. Given the additional hypothesis of structural completeness of *I*+, this problem is considered as a search through a Boolean lattice built from the positive information.

3.1.2.6 Evidence Driven State Merging (EDSM) Heuristic [LPP98]

The main idea in the so-called evidence-driven state merging *(EDSM)* algorithm [LPP98] is to try all possible merges and keep only the merge with the high score. It was realized that an effective way to choose which pair of nodes to merge next within the augmented prefix tree acceptor (APTA) would simply involve selecting the pair of nodes whose sub-trees share the most similar labels. To improve the running time of *EDMS*, *window-EDMS* (*W-EDMS)* was suggested where only nodes that lie within a fixed-sized window from the root node of the APTA are considered for merging. An analytical study of *W-EDMS* shows that it is better than its full-width counterpart [CK02].

EDMS won the Abbadingo learning competition (http://abbadingo.cs.unm.edu/), in 1998. This competition's topic is average case learnability of FA from given training data. The basic setup is based on 16 benchmark problems. Each problem consists of a secret randomly generated FA which serves as a target concept, a set of training strings which have been labeled by that target concept, and a set of unlabeled testing strings. The task is to predict the labels that the target concept would assign to the testing strings. Each problem will be considered solved by the first competitor who demonstrates a test set error rate of 1% or less.

3.1.2.7 Data-driven heuristic

This represents a new framework for learning FA, where the quantity of data is used as heuristic to drive the learning process [deH96]. Any data-independent ordering will allow for identification in the limit. Here, a heuristic is chosen. It tries to merge those two states for which most evidence is available. Based on this heuristic, it is proved that the algorithm identifies in the limit. However, the characteristic set associated to this heuristic can be exponential. The learning algorithm is called *data-independent* if it does not need information about the data of positive and negative examples to return its result. Otherwise it is *data-dependent*. Results obtained assert that *polynomial identification from given data* is a non-trivial condition leading to interesting algorithms in GI.

3.1.3 Learning non-deterministic finite state automata NFA

Inferring NFA is not polynomially possible from given data [deH97]. In [DLT01], it is proposed to learn cheaper structure than FA; looking for an NFA seems to be a promising way. A sub-class of FA called *residual finite state automata* (RFSA) is studied. RFSA shares the property of existence of a canonical representation with FA. They define the system called *DeLeTe* that builds the canonical representation from any sample containing S_A, where S_A is a characteristic sample with polynomial cardinal associated with a FA.

3.1.4 Learning quantum finite automata

Equivalence between quantum automata [Moo00] and quantum grammars on one side and FA and grammars are studied in [KW97]. The importance of quantum automata is due to their lower space complexity (fewer states, fewer steps) and their capacity to recognize some non-regular and non-CFLs. In [RG01], it is shown that quantum and classical learning are information-theoretical equivalent. However, apparent computational advantages of the quantum model yield to efficient quantum learning algorithms which seem, up to now, to have no equivalent in classical counterparts such as those proposed in [BJ99].

3.2. Algorithms for CFGs

After spending almost three decades on regular grammar inference, it was natural to move to the next class in the Chomsky hierarchy, *i.e.* the CFGs. That was first set in the *European Conference on Machine Learning (ECML2003)*. Another motivation to study the domain was the limitations of regular grammars in some new domains like genetic structures, *XML* and its technology, text compression, and the like. CFGs are more expressive than regular ones. Learning the entire class of CFLs is until now an intractable problem, *i.e.* the time required solving instances of the problem growth exponentially with the size of instances of the problem. Providing additional information or avoiding super-finite classes can help to identify this class in the limit. In order to avoid the negative result of impossibility of inferring a class of languages from positive examples alone, some methods have been set out. On top of positive examples, additional information can be negative examples, use of an oracle, and knowledge on structures or *ad hoc* heuristics.

3.2.1 Difficulty of CFG inference

A tentative of synthesizing most problems in GI of CFLs is detailed in [Eyr06]. These problems can be summarized as follows:

- CFLs are not stable for a set of algebraic operations like intersection and complementation. The use of negative examples is not useful because they have not the same structure as the hypothesis to be learned.
- It was proved that the class of CFLs is not identifiable in the limit, polynomially in time and data using a sample of positive and negative examples. This is due to the undecidability of equivalence problem in the class of CFLs [deH97].
- Contrarily to regular languages where the entire class is recognized by FA, CFLs can be recognized by non-deterministic push-down automata (PDA) Determinism is an essential point in learning, so nondeterminism and ambiguity of CFGs represent an important problem within the inference process.
- Some CFGs have a huge "expansibility". Indeed, the number of productions grows exponentially with the size of a sentence. For example, the simple deterministic grammar:

$$Gn = (\{a\}, \{N_i, i <= n\}, P, N_0),$$

where:

$$P = \{ N_i \rightarrow a\ N_{i+1}\ \ N_{i+1} \} \cup \{ N_n \ \ \rightarrow a \}.$$

For this grammar, the equivalence problem is decidable but the number of productions used is exponential in the size of the grammar. So inferring it in polynomial time is impossible.

- Indivisibility of the CFGs is another problem for the learning process. Any update in the productions can affect the totality of the language; there is no separate ways of derivations.

3.2.2 Algorithms for CFG inference

Due to the serious theoretical limitations of learning the entire CFLs, different practical techniques are established to obtain positive results. So classifying these algorithms is a difficult task. This may explain why there are only very few number of surveys of the field. To our knowledge, there are only a couple of these, [Lee96] and [deH05]. Recently, a book was published for learning automata and grammars [deH10]. We give below a tentative classification of the most important algorithms.

3.2.2.1 Complexity

The complexity of CFGs is obviously is worse than the complexity of regular grammars exposed above. Indeed, the search space for (CFG) inference is even larger [CMZ05]. For a given positive sentence, we need to find the different derivation trees. Using CNF, the number of all possible binary trees with *n* internal nodes is given by the *n*-th Catalan number. An additional issue is that internal nodes (nonterminals) need to be properly labeled. The number of possible labeling of nonterminals is defined by Bell numbers. As a result, the construction of derivational trees with proper labeling of nonterminals contributes to an immense search space. For instance, a statement with 5 terminals (4 nonterminals) can be parsed by 210 different derivation trees, while this number increases to 1.9479161E9 for a statement with 11 terminals (10 nonterminals) [SF01].

3.2.2.2 Patterns in strings

In general, this type of algorithms is popular in the pattern recognition community. A pattern is a special substring. These algorithms deal with learning from *text*, *i.e*. a set positive data and eventually negative ones. This approach is limited by Gold's theorem. The first algorithm is reported in [Sol59] while [Tan87] gives an algorithm that learns CFGs from positive and negative examples of strings. The technique presented is to remove self-embedding structures from a finite sample, infer a linear grammar from the sample, and compose the inferred linear grammars to create a CFG. Once again, the learnability from positive examples only is not guaranteed for all CFLs.

The work in [Ang80] gives some sufficient and (or) necessary conditions for this purpose. However, the use of negative examples seems also unnatural. As stressed earlier, when a child learns a language, he receives only correct sentences from that language and needs no incorrect ones. These points motivate research for tools other than negative examples.

3.2.2.3 Extension of regular languages 'results to CFLs

The class of regular languages is a subset of CFLs. One natural way to upgrade to CFG inference is the extension of techniques used for regular grammar inference. We have seen that the lack of linearity and determinism represent a problem in CFG inference. This has motivated the study of linear and even linear languages. The GI problem for even linear languages can be solved by reducing it to the GI problem for regular languages [Tak88]. [Mäk96] introduces subclasses of even linear languages for which there exist inference algorithms using positive samples only; this is done *via* Szilard languages [Ros97].

k-bounded CFGs are identifiable in polynomial time using equivalence queries and *non-terminal membership* queries [Ang87]. Non-terminal membership queries propose a string *w* and a non-terminal *A*; the answer is "*yes*" if *w* is derivable from *A* and "*no*" otherwise. In effect, the learner is allowed to ask about the structure of the target grammar. A larger class of the deterministic linear grammar is proved to be identifiable from polynomial time and data [deH02].

Simple deterministic languages (SDLs) are used in such a way that non-terminal membership queries are no longer needed [Ish90]. Instead, the algorithm is allowed *extended equivalence queries*, which propose a grammar *G*, where *G* does *not* have to be a grammar for an SDL; the answer is "*yes*" if the target grammar is equivalent to *G*.

Other subclasses of CFLs that have been shown to be learnable are *structurally reversible* languages, *one-counter* languages (languages accepted by deterministic one-counter automata), pivot languages, very simple languages, and terminal distinguishable CFLs [LN03].

3.2.2.4 Use of artificial intelligence techniques

Here the inference problem is seen as a search in the space of possible grammars. The main problem to study is the size of the search space. For CFGs, the search space has been seen as a version space [Lan00]. Search algorithms like hill-climbing or genetic algorithms are used. We can use genetic algorithm on the rules of grammars on the condition that some help is provided from structures of data to reduce the size of the population [Sak00]. Other techniques like the use of an intelligent backtracking or the prior conflict diagnosis or heuristics are of a great utility.

3.2.2.5 Stochastic CFGs (SCFGs)

There is sometimes a need to deal with noisy data for example in speech recognition or in computational biology. Here stochastic CFGs (SCFGs) are used. SCFGs are CFGs where a probability is associated to each production so that the sum of probabilities of all productions with the same left hand side is one. One problem in this approach is how to decide of the correctness of these probabilities. The second is with parsing such grammars. Here, all algorithms attempt to search the space of all SCFGs, either exhaustively, *i.e.*, by *enumeration*, or by some sort of heuristic search. An enumerative algorithm is developed that identifies SCFG's in the limit with probability one from stochastic data [Hor72]. The approach of inferring directly the CFGs is hard. It seems that artificial intelligence techniques like genetic algorithms can be of great help in solving this problem [Sak00].

3.2.2.6 Algorithms that uses alternative representations for languages

Instead of representing a language by a grammar from the Chomsky hierarchy, it is represented in different ways: context-free expression, pattern languages, and categorical grammars.

The first representation is used by [Yok88] and is inspired by learning regular expression. The author gives an NP-complete algorithm that learns *context-free expressions.* Pattern languages are first studied by [AS83], defining a pattern as the concatenation of constants and variables, and the language of a pattern as the set of strings obtained by substituting constant strings for variables. Introduction of types [Kos95] or using only one pattern [ERS97] are ways to simplify the problem of inference. Pattern languages have been also used with probabilities in [RZ01].

Grammars in Chomsky hierarchy deal only with syntax. For linguistics, learning a language concerns both syntax and semantics. Categorial grammars are grammars where syntax is attributed some semantics [Kan98]. Important role of semantics and context in the early stages of children's language acquisition, especially in the 2-wordstage has motivated the work in [BA08].

3.2.2.7 Algorithms that rely on structured data

We saw that additional information is needed along with positive examples to achieve learnability of CFGs. Important information concerns the *structure of the data*. This structural information is known as *derivation trees*. Structural data can be represented by strings generated by a *parenthesis grammar* or by *skeleton.*

For any CFG G, the corresponding parenthesis grammar (G) is formed from G by replacing every production $A \rightarrow \alpha$ by $A \rightarrow (\alpha)$. On the other hand, *skeletons* are derivation trees with the non-terminal labels removed. The key property of skeletons is that they are exactly the set of trees accepted by *skeletal tree automata* (STA), a variation of finite automata that take skeletons as input. There are very strong relations between learning CFGs from parenthesized data or skeletons and learning regular tree grammars.

Learning FA has been extended to the identification of STA in polynomial time, although this requires being able to ask *structural membership* and *structural equivalence* queries [Sak92]. As a result, inference is made possible for *reversible CFGs* in the limit from positive structural data alone by adapting the technique for reversible automata [Ang82]. Skeletons are also used to infer terminal distinguishable CFGs [LN03].

3.2.2.8 ILSGInf : Inductive Learning System for Grammatical Inference

Derivation trees and the so-called partial derivatives heuristic construction is at the heart of our method, used in the development of *ILSGInf* [HH07b], detailed in Chapter 6.

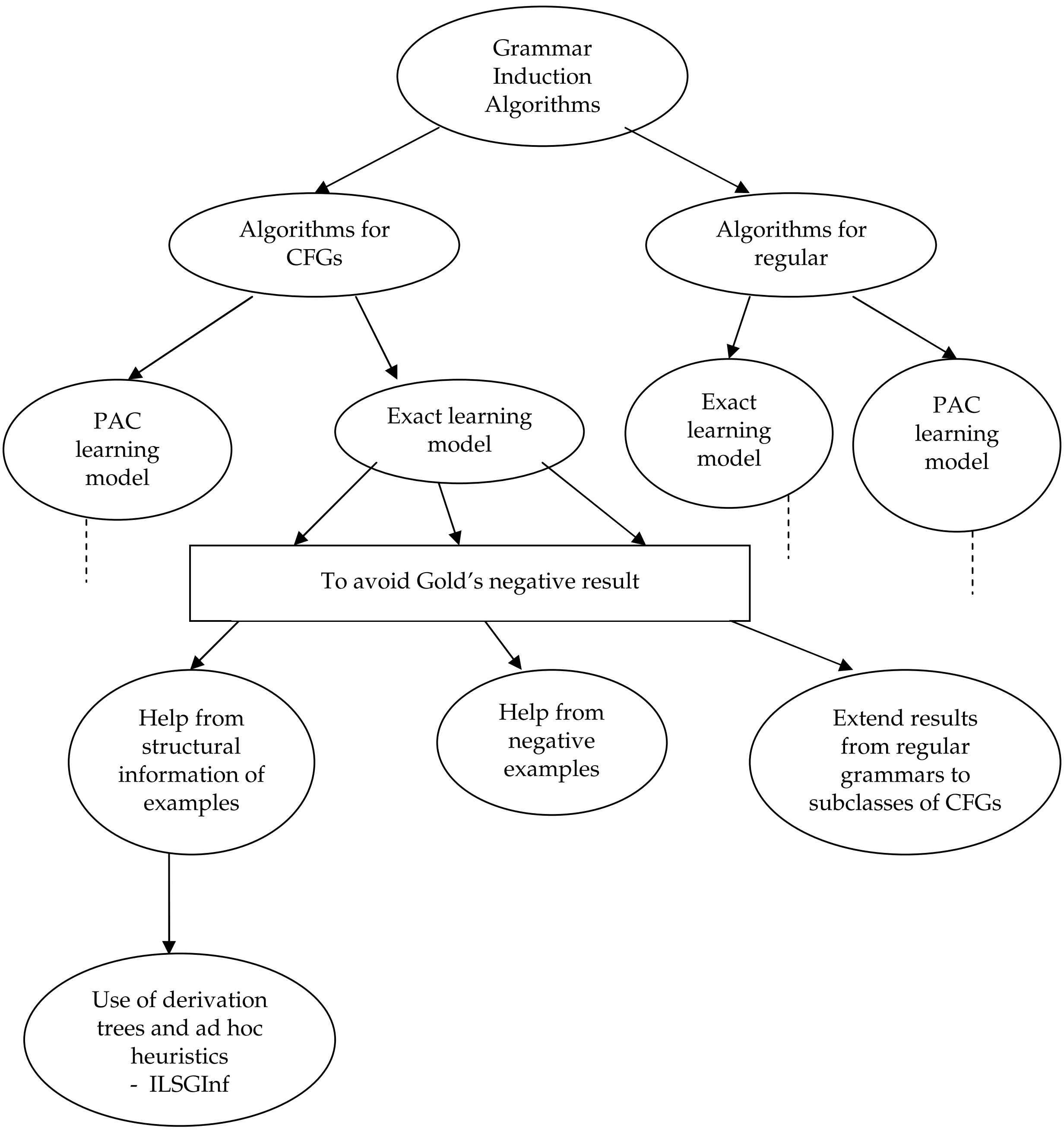


Figure 3.3 DIAG31 – *ILSGInf* : a system for GI within existing GI methods

4. Applications of grammatical inference techniques

GI techniques are widely used in different domains. We survey a set of such applications in different fields such as pattern recognition, language processing, data processing, robotics, and software engineering, to cite just a few of these numerous applications.

4.1 Structured pattern recognition

Pattern recognition is the field where GI was applied first. Sometimes objects with no independent measurable properties are recognizable by their structural configuration. Structures are described using grammars where terminals are the set of recognizable pieces, and productions encode the different configuration. Then classification is equivalent to parsing. GI is present when we want to infer the global structure of a set of instances. It was applied to textures in images, image contours [Luc94], fingerprints classification, recognition of pictures of industrial objects, character recognition by learning stochastic finite automata.

4.2 Computational linguistics

One of the earliest motivations of GI was to understand human language acquisition. While GI deals only with syntax, human language acquisition takes also semantics in consideration. *EMILE* prototype [AV02] is a toolbox for natural language processing. It is intended to help researchers to analyse the grammatical structure of free text. This work is based on categorical (or categorial) grammars which are most suitable for linking syntax with semantics. Another worthy application is shallow parsing, *i.e.* the task of dividing sentences into a sequence of simple phrases [Tho02]. Shallow parsing can be used to index internet pages, for instance.

4.3 Speech recognition

Speech is the domain where noise is an important characteristic. Probabilistic grammar inference is used *via* two models: hidden Markov models (HMMs), *i.e.* automata with probability, and n-grams models. One of the earliest models was used by [MB95] to focus on a description of the hybrid HMM/artificial neural networks method. In this work, the authors also began to look at the connectionist inference of language models, including phonology, from data. This step is required in order to take advantage of locally discriminant probabilities rather than simply translating to likelihoods. GI techniques were also used to language simplification trough error-correcting [ASV01].

4.4 Automatic translation

Usually a transduction is viewed as a string to string function *f* ("My red car") = "Ma voiture rouge". Automata with outputs are used. We can cite the improvement of the *OSTIA* algorithm. The input of learning is represented by pairs of strings (input string and the associated output). *Multiplicity automata* are used to deal with ambiguity. Alignment techniques were used with dictionaries to improve the learning of sub-sequential transducers [Vil00].

4.5 Document management

In recent years, writing, storing, and retrieving documents in electronic form has become popular. These documents are structured. The common way to describe the structure of similar documents is the use of grammars. *Extended markup language XML* has been recently used for text element markup. The extraction of schematic information from *XML* documents often requires certain generalisation of input data. Among existing conceptual approaches to the *XML*, the grammar-based one seems to be the most promising for the schema extraction. An *XML* data is equivalent to a derivation tree of a CFG without non-terminal labels in GI theory. This extraction was addressed as a GI approach [Chi01].

4.6 Data and text mining

4.6.1 Text mining

Both information extraction (IE) and information retrieval (IR) belong to the broader field of text mining (TM). In information retrieval, we seek to recover information from a subset of documents that are hopefully relevant to a query, based on keywords searching, usually augmented by a thesaurus. In information extraction, the goal is to extract from the documents, which may be in a variety of languages, important facts about *ad hoc* types of events, entities or relationships. These facts are then usually entered automatically into a database, which may then be used to analyze the data for trends, to give a natural language summary, or simply to serve for on-line access. Information extraction consists in finding subtle or at least non-trivial knowledge from text. Automatic information extraction is still in the making despite the fact that there are many public Web-based platforms that can be used for this purpose, *e.g.* GATE[6] platform.

4.6.2 Text compression

Grammars have the potential of representing infinite information using only finite set of rules. As a result of this property, one can consider a grammar as a compression tool of the whole language. For example, *Sequitur* is a compression system that was developed based on the idea that a good grammar is a compact

[6]GATE was developed at Sheffield University, England, http://gate.ac.uk/ie/

grammar [NW97]. It requires no input except a single text and it produces a grammar that generates only the input text. It is clear that that *Sequitur* cannot be considered as a GI system but it has an important role to compress an input text.

4.6.3 RPNI and structure induction

In [KR07], the authors study the use of RPNI algorithm, described in Section 3.1.2.3 above, to infer information extraction models from positive and negative examples. In [Sai06], GI is applied to text corpus. These techniques attempt to induce the structures of a source data by a set of production rules of regular grammar.

4.7 Biological interfaces

4.7.1 Grammatical structures in biological sequences

The huge amount of data about genes and proteins and the availability of complete genomes offer the possibility to study more globally the interaction between bio-entities in complex cellular processes. Many efforts focused on the decoding of complete genomes and assignment of functions to genes and proteins. The result is the birth of the field of bioinformatics. Its principal goal is to bridge the gap between biology and computer science to understand cell behaviour and to develop systems that link computational techniques and biology, among others. Bioinformatics is facing new challenges in analyzing the functioning of biochemical networks and molecular biology. GI techniques are expected to find many useful grammatical structures in biological sequences [Coh04].

4.7.2 DNA computing

DNA computing began by the demonstration that DNA can be used as a form of computation for solving the seven-point Hamiltonian path problem [Adl94]. DNA computing and parallel computing are fundamentally similar. Indeed, in DNA computing, many different molecules of DNA try many different possibilities at once. In this novel computer architecture, simple biological operations are coded as simple instructions. DNA sequences are used to encode information and enzymes can be employed to simulate basic computations. As a result, a DNA computer was constructed and coupled with an input and output module, which would theoretically be capable of diagnosing cancerous activity within a cell, and releasing an anti-cancer drug upon diagnosis [BGB04]. It has been demonstrated that DNA array can implement a cellular automaton, which generates a fractal called the Sierpinski gasket. This shows that computation can be incorporated into the assembly of DNA arrays, increasing its scope beyond simple periodic arrays [RPW04].

4.8 Map learning

In their article [DBK92], the authors present a robot with automatic learning abilities based on GI in the field of map learning. It is useful for a robot to construct a spatial representation of its environment from experiments and observations. Probabilistic GI techniques are used to infer the global structure of the environment from a sample of experiences.

4.9 Self assembling

In self assembly, a collection of particles spontaneously arrange themselves into some coherent structure. In one approach, each particle is provided with a local interaction rule, based on graph grammar [KGL06]. The main problem is to infer a global behaviour of a system by means of local rules. The approach shows that we can refer to grammars approaches to precisely predict and control the emergent behaviour of self-organizing system. Some aspects of grammars are used to model dynamical systems and self-organized systems are described in Chapter 7.

4.10 Software engineering

Extracting grammars from programs attracts researchers from software engineering. In this field, we want to recover a grammar from legacy systems in order to automatically generate various software analysis and modification tools. The so-called memetic algorithm *MAGIc* improves current results in grammar inference of domain-specific language (DSL) grammars from example of DSL programs. The result is a tool support for DSL development, assisting domain experts and software language engineers in developing a DSL and its implementation. A semiautomatic grammar-driven system, called *MARS*, uses GI techniques to recover metamodels from instance models developed on a network metamodel [MHB09].

4.11 Soft computing and evolutionary multiobjective optimization (EMO)

Learning can be reduced to finding solutions using evolutionary multiobjective optimization (EMO). In this framework, the different solutions are handled by the standard evolutionary operators such as selection, crossover, and mutation. The improvement of the solution is handled by the construction and comparison of the Pareto fronts using the various fitness (objective) functions. This framework was used and tested on a medical classification problem and gave satisfactory results [HH11][7].

---

[7] Part of this work has been published under the title: “Evolutionary multiobjective optimization for medical classification”, *2011 IEEE GCC Conference & Exhibition*, *"For Sustainable Ubiquitous Technology"*, Dubai, United Arab Emirates, pp. 441-444, 19-22 February 2011, http://www.ieeexplore.org

CHAPTER 4

GRAMMATICAL INFERENCE WITH GASRIA[8]

1. Introduction

As stressed in Chapter 2, many methods and systems have been developed for GI for more than half a century. As far as this chapter is concerned, the proposed contribution falls at the intersection of three major fields of research, namely formal languages, machine learning (ML) with special emphasis on grammatical inference (GI), and inductive logical programming (ILP). We know that these fields of research historically evolved independently, although it can be well be argued that they are naturally related since both GI and ILP are considered as integral parts of ML. On the other hand, formal languages are described using grammars. Now each of these areas has now its own scientific community with its *ad hoc* periodicals, its scientific meetings and its specialized conferences. Because the system we propose is based on one logic-based

[8] Part of this chapter has been published under the title "Apprentissage inductif de grammaires: Le système GASRIA. (Inductive learning for grammars: The GASRIA System)", In *Revue d'Intelligence Artificielle*, Hermes-Lavoisier Edition, Paris, France, ISSN: 0992499X, 21(2):223-253, March-April 2007
http//ria.revuesonline.com, http://www.revuesonline.com, http://ria.revuesonline.com/article.jsp?articleId=9770

environment and one inductive learning module, we attempt a useful rapprochement between ILP and GI.

Specifically, our main problem deals with parsing. In classical parsing, a sentence is either recognized or refused. In other words, parsing is stopped, perhaps at the outset, due to the first unrecognized character - with no further search. This limitation characterizes all existing methods like Earley's algorithm [Ear70] or its offshoots [Lee92]. In a more general context involving learning, as the one we are considering, this limitation is a truly severe drawback [MGZ03]. Indeed, we want, for example, to know whether at least some part(s) of the sentence is (are) correct without getting ousted by the first unrecognized character. Therefore, we apply a method to parse all that is parsable using partial derivation. In this way, we are able to draw maximum syntactic knowledge from the sentence under consideration. In order to address this issue, we introduce the concept of *partial parsing* and its corresponding algorithm, the so-called *partial parsing algorithm* (PPA) [HH07a].

This chapter is organized as follows. Section 2 formulates the problem, putting forward the objectives to be realized and the available methods for doing so. Section 3 describes related works from three different perspectives, namely GI, machine learning and ILP. In Section 4, GI and ILP approaches are defined and compared and GI is formulated in an ILP framework. *GASRIA* architecture is described in Section 5 while Section 6 reports relevant parsing issues. Section 7 describes the learning process in *GASRIA* and Section 8 reports the backbone of the implementation and operation of *GASRIA* on an illustrative example. The chapter ends up with a conclusion and perspectives for further developments.

2. Problem formulation and basic methods

One of the reasons hindering coupling a first-order logic-based environment with a learning system for grammar acquisition lies in the structural and functional differences between these two types of systems. We develop a synergy between both systems in order to induce, or infer, one possible grammar. We concentrate on CFGs because they are used to specify the majority of programming languages. The other reason is that CFGs inference is still a challenging issue.

2.1 GASRIA Objectives

A complex and multidisciplinary environment is the intelligent and synergetic interaction of basic and modular building blocks tied together in a coherent action towards the achievement of the most practical and lesser-effort design. For so doing the overall environment is to comply with the methodological steps depicted in Figure 4.1 below.

/* Methodology 4.1 */
/* METH41 */

/ Methodological steps used in GASRIA */*

1. Goal : GASRIA level

Design an integrated architecture and develop a system based on coupling inductive learning and first-order logic (FOL) for the purpose of grammatical inference for some CFGs.

2. Modules

EXINF **Module /* Chapter 5 */**

Design and develop an FOL-based module for addressing both traditional or "crude" parsing and "intelligent" parsing issues based on an original declarative Earley-like algorithm.

ILSGINF **Module /* Chapter 6 */**

Design and develop an inductive learning module for solving the following incremental learning problem:

From a set of positive strings with respect to a given language, induce one possible CFG that generates all strings acceptable by the given language.

Figure 4.1 METH41 Methodological steps used in GASRIA

2.2 Methods used

We rely on methods drawn from parsing and from inductive logic programming (ILP). Parsing is used to recognize and/or classify a string. ILP is used to make the required inferences during the learning process.

3. Related works: three interconnected fields

There are many approaches that can be used to meet the methodological steps described in Figure 4.1 above. We stress the important fields that are of interest to us. We thus report some aspects of formal languages, as the basis for parsing, before concentrating on machine learning and ILP.

3.1 Formal languages approach

This approach has been addressed in details in Chapter 2, specifically dealing with formal languages and grammars. We further summarize the basic concepts we need for our work. Intuitively speaking, a *language* is a complex system of structured messages that enables humans, or other species, to *communicate* what they know about the world. *Communication* is meant as the intentional exchange of information that is brought about by the *production* and perception of signs drawn from a shared system of conventional signs. Particularly for humans, language is at the root of thinking. That is why the so-called Turing Test, used for the definition and examination of machine intelligence is, above all, based on language. A *formal language* is the eventually infinite set of strings, each of which is the concatenation of *terminal symbols*, usually called words in natural languages. For instance, in the language of arithmetic, the terminal symbols include real numbers, or symbols representing them, and other symbols like the + sign, the – sign, the = sign. In this case, if *a* and *b* are two numbers then "*a*+*b*" is a member of the arithmetic language, "+*a*,*b*-" is not. Formal languages, like first-order logic have strict mathematical definitions. A *grammar* is a finite set of rules that specifies a given language. Formal languages always have a precise, official grammar, specified in manuals or books. Both formal and natural languages associate a meaning or *semantics* to each valid string. For instance, in the language of arithmetic, a grammatical rule would specify that if *"a"* and *"b"* are expressions then *"a+b"* is also an expression whose semantics is the sum of both *a* and *b*. *Pragmatics* is a characteristic of natural language which consists of the interpretation of a given string according to the situation or context.

Most grammar rule formalisms are based on the idea of *phrase structure* – that strings are composed of substrings called *phrases*, which can be expressed in different categories, known as *noun phrase (NP), verb phrase (VP).* For example the *NP* "The paper" can be concatenated with the *VP* "is excellent" to produce the *sentence S* "The paper is excellent". The category names such as *VP*, *NP* and *S* are called *non-terminal symbols* or simply *non-terminals*. Grammars define *non-terminals* using *rewrite rules*, usually described in Backus-Naur Form (BNF), previously known as Backus Normal Form. In that case, the previous sentence can be expressed in the form $S \rightarrow NP\ VP$ meaning that any sentence *S* can be written as any *NP* followed by any *VP*. *Parsing* is the process of building a *parse tree*, composed of a root *S*, interior nodes composed of *non-terminals* and *leaves* composed of words as *terminals*. For example, the previous sentence "The paper is excellent" would have *S* as root with one left-child *NP* and one right-child *VP*. The *NP* node would have a left-child *Article* and a right-child *Noun*. The *VP* node would have a left-child *Verb* and a right-child *Adjective.* Further down in the tree we would have all the words composing the sentence, as leaves. The only child of *Article* is *The*. Likewise, *Noun* is instantiated by *paper*, *Verb* by *is*, and *Adjective* by *excellent*. The result is that the parsed sentence appears at the bottom of the tree. This process is called *top-down parsing*. Conversely, if we start from any sentence, we try, in *bottom-up* process to go up to the start symbol *S*. If we succeed in doing this, then the sentence is said to be correct, *i.e.* the sentence belongs to the language; otherwise, it is incorrect. The process of moving "upwards" in the parse tree from the leaves to the immediate level above is referred to as "*tokenization*". Therefore, the instantiation of tokens ends up with terminals [RN03].

3.2 Machine Learning (ML)

3.2.1 Inductive and deductive learning

As a broad subfield of artificial intelligence, ML is concerned with the design and development of algorithms and techniques that allow computers to improve their processing through training. At a general level, there are two types of learning: inductive and deductive. Inductive methods extract rules and patterns out of massive data sets. The major focus of ML research is to extract information from data automatically, by computational and statistical methods. ML is therefore closely related to not only theoretical computer science but also to data mining and statistics. ML has a wide spectrum of applications including natural language processing, syntactic pattern recognition, search engines, medical diagnosis, bioinformatics, brain-machine interfaces, detecting credit card fraud, stock market analysis, classifying DNA sequences, speech and handwriting recognition, object recognition in computer vision, game playing and robot locomotion, among others [Mit97].

3.2.2 Some ML/data mining methods

The main traditional methods available in ML are decision tree learning (DTL), neural networks, Bayesian learning, instance-based learning, genetic algorithms, rule learning, analytical learning, and reinforcement learning. Among the most well-known algorithms, we can cite symbolic rule-learning algorithm such as CN2 [CN89], and C4.5 [Qui93]. When rules have to be learned from extremely large data sets, specialized algorithms stressing computational efficiency may also be used. Other machine learning algorithms commonly applied to this kind of problems include inductive logic programming [Mug99], neural networks, and Bayesian learning algorithms. The textbook [Mit97] describes a broad range of machine learning algorithms, as well as the statistical principles on which they are based. The field of ML has borne the explosive field of data mining, sometimes called knowledge discovery from databases, or advanced data analysis. It has already produced practical applications in such areas as analyzing medical outcomes, detecting credit card fraud (*e.g.* using the so-called White

Hat Google™ Hacking), predicting customer purchase behavior, predicting the personal interests of Web users, and optimizing manufacturing processes, among others. This is so because data mining algorithms enable discovery of important "regularities" in large data sets. A more recent survey describes most systems and algorithms of data mining [MR11] and some textbooks describe applied data mining platforms such as the Weka[9] platform [WFH11]. The study of ML has also led to a set of fundamental scientific and epistemological questions about how computers might automatically learn from experience and subsequently improve behavior.

3.3 Inductive logic programming (ILP)

ILP aims to construct a set of hypotheses to enrich available background knowledge using predicate logic. In the case where positive examples are not entailed by background knowledge, the idea is to construct a new set of hypotheses to extend background knowledge in order to make this entailment possible.

From ML, ILP inherits the goals, namely to develop tools and techniques to induce hypotheses from observations (examples) and to synthesize new knowledge from experience. By using computational logic as the representational mechanism for hypotheses and observations, ILP can overcome the two main limitations of classical ML techniques, namely the use of limited knowledge representation formalism encoded as a propositional logic, on the one hand, and the difficulties in using substantial background knowledge in the learning process, on the other hand [Mug99]. As an interaction with grammars, we can refer to the specific application where ILP has been applied to the problem of learning a grammar that is augmented with semantics. Since an augmented grammar is a Horn clause logic program, techniques of ILP are found appropriate. As an example, *CHILL* [ZM96] is an ILP program that learns a grammar and a specialized parser for that grammar from examples. The target domain is natural language database queries. *CHILL*'s task is to learn the predicate *Parse(words, query)*

[9] Weka is a Web-based platform developed at Waikato University, New Zeland; http://www.cs.waikato.ac.nz/ml/weka

that is consistent with examples and, hopefully, generalizes well to other examples. For instance, the query "what is the capital of the state with the largest population?" is transformed into *"Answer(c, Capital(s,c) AND Largest(p, State(s), AND Population(s,p)))*. Applying ILP directly to learn this sort of predicate results in poor performance since the induced parser has only about 20% accuracy. Fortunately, ILP learners can improve by adding knowledge through the construction of hypotheses. In this case, most of the *Parse* predicate was defined as a logic program, and *CHILL*'s task was reduced to inducing the control rules that guide the parser to select one parse over another. With this additional background knowledge, *CHILL* achieves 70 to 85% of accuracy on various database query tasks. This is obviously based on the assumption that the problem can be expressed in a predicate form; an assumption that might turn out difficult to be realized in some situations.

4. GI *vs.* ILP

4.1 Problem of inductive inference

Inductive learning's task, at large, is based on the idea of fitting a set of instances (or examples) into a more general framework. This is equivalent to identifying a relationship between some variables, given some observed results. It can be set in a variety of manners, but the question ends up with an identification of some hidden relationship between the known inputs and the produced outputs.

4.1.1 Inductive inference and normal semantics

We are given a background (prior) knowledge *B* and evidence *E*. This evidence is described by the union of two disjoint subsets of positive evidence (E^+) and negative evidence (E^-). Assume that we have evidence $E = E^+ \mathbf{V}\, E^-$ a background theory and some hypotheses all expressed as *well-formed formula* (*wff*). We can formulate the general problem of inductive inference as described in Figure 4.2 below.

/* Methodology 4.2 */
/* METH42 */

/ General problem of inductive inference (GPII) */*
/ Normal semantics */*

Given

- **A background (prior) knowledge *B***
- **Evidence *E***
- $E = E^{+} \cup E^{-}$ **/* Positive and negative evidence */**

Find

One hypothesis *H*

Constraints

1. *Prior satisfiability*

$B \cap E^{-} \mid\neq \square$

/* *i.e.* The conjunction of background (prior) knowledge *B* and negative evidence does not entail the inconsistent clause. */

2. *Posterior satisfiability (or consistency with negative evidence)*

$B \cap H \cap E^{-} \mid\neq \square$

/* ***i.e.*** **The conjunction of all knowledge except the positive evidence does not entail the inconsistent clause**. */

3. *Prior necessity*

$B \mid\neq E^{+}$

/* ***i.e.*** **Background knowledge alone does not entail positive evidence, which means that we need additional knowledge from constructed hypotheses */**

4. *Posterior sufficiency (or completeness with regard to positive evidence)*

$(B \cap H) \models E^{+}$

/* *i.e.* **Background knowledge and all constructed hypotheses entail all positive evidence. */**

Figure 4.2 METH42 Inductive inference and normal semantics

By *satisfiability*, it is meant that the inconsistency clause cannot be entailed from background knowledge and negative evidence. This is true for *prior satisfiability*, *i.e.*

before the introduction of any hypothesis. It remains true *after* the introduction of hypotheses, for the case of *posterior satisfiability*.

4.1.2 Inductive inference and definite semantics

In most ILP systems, background theory and hypotheses are restricted to being definite, thus simplifying the general setting. Indeed, a definite clause theory T has a unique minimal Herbrand model $M^+(T)$, and any logical formulae is either true or false in the minimal model. The above general problem can be redefined with adapted constraints as follows.

/* Methodology 4.3 */

/* METH43 */

/ General problem of inductive inference (GPII) */*
/ Definite semantics */*

Solve the same problem as for normal semantics above

Constraints

1. ***Prior satisfiability***

$\forall\, e \in E^-, e \text{ false in } M^+(B)$

/ i.e. Background knowledge cannot support negative evidence*/*

2. ***Posterior satisfiability (or consistency with negative evidence)***

$\forall\, e \in E^-, e \text{ false in } M^+(B \cap H)$

/ i.e. A fortiori nor can negative evidence be supported by both background knowledge and hypotheses, supposed to enrich background knowledge */*

3. ***Prior necessity***

$\exists\, e \in E^+ / e \text{ false in } M^+(B)$

/ Some positive evidence is false in background knowledge. Otherwise, we would not need additional hypotheses to try to establish that it is true */*

4. ***Posterior sufficiency (or completeness with regard to positive evidence)***

$\forall\, e \in E^+, e \text{ true in } M^+(B \cap H)$

/ i.e. All positive evidence is contained in background knowledge and all constructed hypotheses. */*

Figure 4.3 METH43 Inductive inference and definite semantics

The general case of the definite semantics, where the evidence is restricted to true and false ground facts (examples), is called example setting. Notice that the example setting is equivalent to the normal semantics, where *B* and *H* are definite clauses and *E* is a set of ground unit clauses. The example setting is the main setting of ILP. It is used by the large majority of existing ILP systems.

4.2 Formalized ILP approach

The general ILP approach works as follows. It keeps track of a queue of candidate hypotheses *QH*. It repeatedly deletes a hypothesis *H* from the queue and expands that hypothesis using inference rules. The expanded hypotheses are then added to the queue of hypotheses *QH*, which may be pruned to discard unpromising hypotheses for further investigation. This process is continued until the stop-criterion is satisfied.

/* Methodology 4.4 */
/* METH44 */
/* The general ILP approach */

QH := **Initialize** */* Set of starting hypotheses */*
REPEAT
Delete *H* from *QH*
/* **Delete** *influences search strategy. Can realize depth-first (LIFO), breadth-first (FIFO), best-first. */*

Choose the inference rules $r_1,\ldots r_k$ in *R* to be applied to *H*
/ R is the set of rules to be applied */*
/* **Choose** *determines the inference rule to be applied on H */*
Apply the rules to *H* to yield $H_1,\ldots,H_n$
Add $H_1,\ldots,H_n$ to *QH*

Prune *QH*
/* **Prune** *determines which candidates hypothesis are to be deleted from the queue. Can rely on user as "oracle" */*
UNTIL stop-criterion (QH) satisfied
/* *Conditions under which algorithm stops. When either solution or failure is found from current queue */*

/ Combining* **delete** *and* **prune** *it is easy to obtain advanced search strategies such as hill-climbing, beam-search, best-first, etc… */*

Figure 4.4 METH44 General ILP approach

4.3 GI formulated in ILP framework

We can express our positive-example-based grammatical inference problem (PIB-GIP) within an ILP framework, as follows:

/* Methodology 4.5 */
/* METH45 */

/* GI Problem formulated as an ILP problem */

/* Positive-example-based grammatical inference problem (PEB-GIP) */

Given

- A logic program representing background knowledge
  /* In our context, background knowledge corresponds to CFG definitions and parsing methods. */
- A set of positive examples (or sentences) D^+

Find
- In ILP framework, construct additional rules describing a CFG that generates this specific set of data.
- In GI framework, departing from ILP, the aim is also to generate other similar data not necessarily given at the outset.

Figure 4.5 METH45 GI problem formulated as an ILP problem

In any of the frameworks of Figure 4.5, there remains the delicate operation of reducing the number of relevant hypotheses to construct. In our case, the partial parsing algorithm (PPA), described in forthcoming Section 5.2 of Chapter 6 reduces drastically this number since it searches within known sub-sentences. This step represents a useful contribution. To our knowledge, no absolute minimization method exists regarding the number of hypotheses to consider.

4.4 GI - ILP interplay

As can be easily seen from the literature, ILP [Mug99] has several links with GI. When learning recursive rules, ILP shares some of GI's objectives and sometimes its techniques. For instance, *MERLIN*[10] (Model Extraction by Regular Language INference)

[10] http://people.dsv.su.se/~henke/ML/MERLIN.html

system parses the data by the background knowledge and uses this information to learn a deterministic finite automaton or a stochastic one [Bos98]. *MERLIN* 2.0 is an inductive logic programming (ILP) system that uses a general hypothesis in the form of a logic program together with sets of positive and (optionally) negative examples in order to find an inductive hypothesis which entails all positive examples but no negative examples. *MERLIN* has been improved resulting the *GIFT* system [BH01]. This latter builds on *MERLIN* by learning directly tree automata, thus not needing to lose representation capacity by having to linearize the data. However, systems like *MERLIN* and *GIFT* use GI as the inference engine of logic programs; they do not combine GI with existing ILP systems.

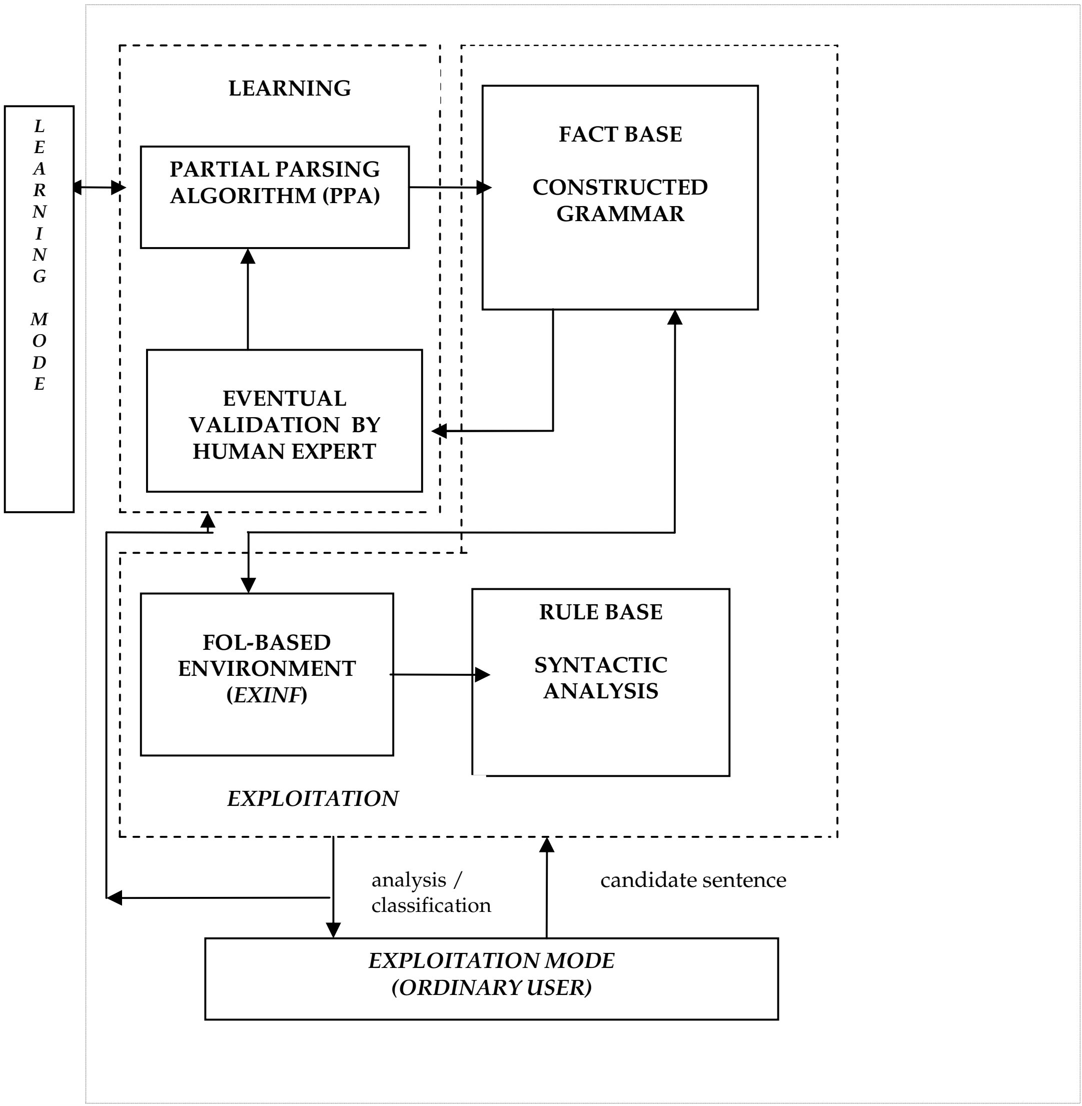
5. GASRIA Architecture

5.1 *GASRIA* modes of operation

5.1.1 Overall block diagram

Figure 4.6 shows the overall architecture of *GASRIA* system. As shown, the proposed system is based on two main components: the learning module *ILSGInf* and an FOL-based environment called *EXINF* containing Earley parsing rules and the facts concerning the grammar and the sentence to be parsed. Each component is associated with one specific mode of operation. As indicated, there are two modes (or sessions) of possible operation, namely the learning or training mode destined to the expert or teacher, and linked to the *ILSGInf* module and the exploitation or testing mode destined to the ordinary user, linked to the analysis / classification of sentences to be parsed. We begin by describing the learning mode, and then the exploitation mode.

Figure 4.6 ARCH41 - GASRIA architecture



5.1.2 *GASRIA* class diagram

Figure 4.7 below describes the main classes used in *GASRIA*. It depicts the overall class diagram of system and is used for reusability, readability and easier maintenance.

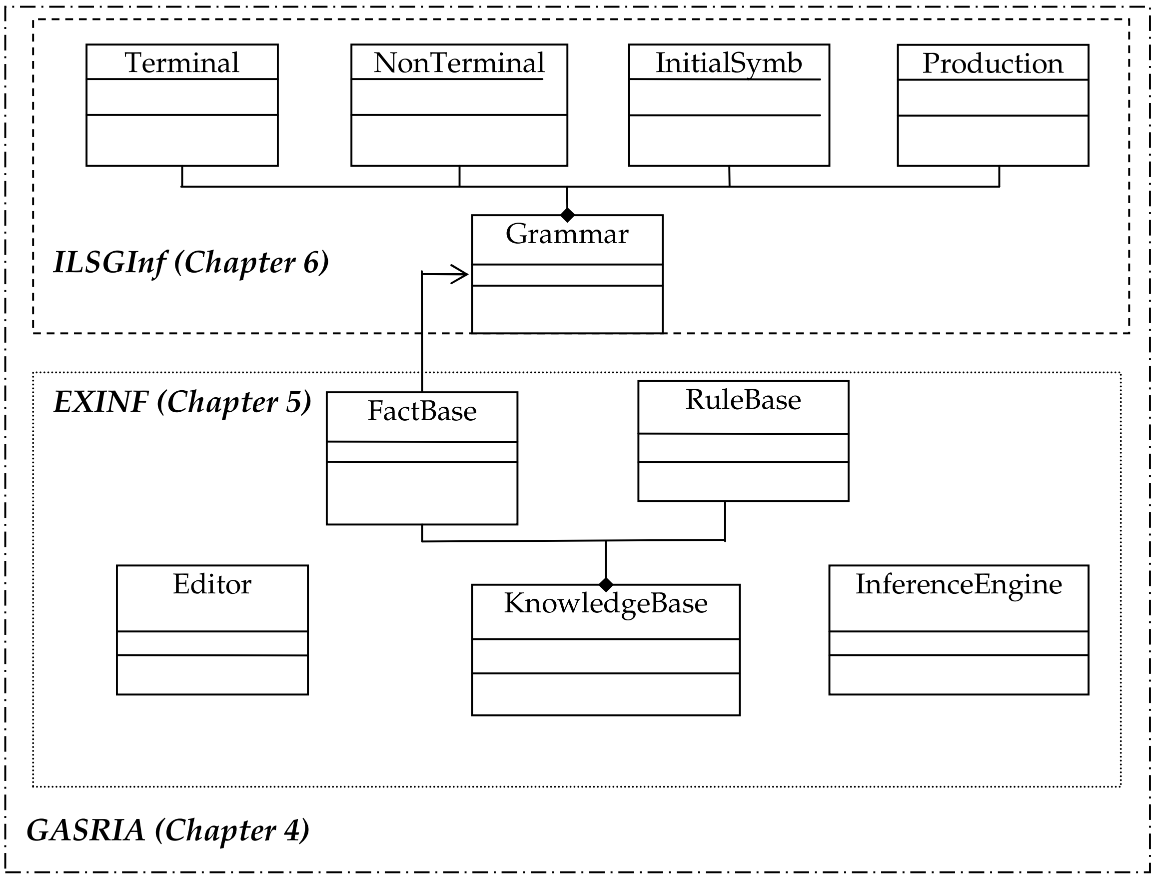


Figure 4.7 ARCH42 *GASRIA* class diagram

5.2 Learning mode: *ILSGInf*

In this mode, the system acquires knowledge from examples introduced by the human expert or teacher, with an exclusive interest in positive examples. At the beginning of the training, the *ILSGInf* learning module receives, one by one, human expert-chosen sentences of a given language and thus enriches its fact base, initially empty. Starting from this set of sentences, this module builds a CFG that generates the language. The fact base is automatically and incrementally filled with the grammar rules describing the language. This eventually completes the session with the expert. The learning mode is further detailed in Chapter 6.

5.3 Exploitation mode: *EXINF*

Because any incremental learning mode requires by its nature the integration of an element of exploitation, we use for that purpose a first-order logic (FOL) programming environment, called *EXINF* working in forward chaining fashion. This form of chaining is used because syntactic analysis is a bottom-up approach. Parsing starts with facts and ends up with goals. *EXINF* allows a specification of the expert knowledge using production rules and plays the role of a parser. In this mode, the available knowledge is used to classify the new sentence. The sentences introduced by the user are syntactically analyzed and the result is displayed indicating whether they belong to the language. The blocks involved in this mode are the inference engine, the fact base and the rule base. The exploitation mode is further explained in Chapter 5.

5.4 Fact base

The fact base consists of a CFG for a given language. The main components of the fact base consist of the two components depicted below.

5.4.1 Initial symbol and the grammar of the language

These are represented by a set of production rules written using the syntax described in Figure 4.8 below.

/* Methodology 4.6 */
/* METH46 */

/* Fact base syntax */

RULE FACT <rule right-hand side> < rule left-hand side >

FACT initial-symbol < initial symbol >

Figure 4.8 METH46 Fact base syntax

5.4.2 Additional information

This concerns the string to be analyzed, such as the string itself and its length (*i.e.* the number of symbols). Figure 4.9 below shows the fact base structure

```
/* Methodology 4.7 */
/* METH47 */

/* Fact base structure */

FACT string <string >
FACT length < length >
```

Figure 4.9 METH47 Fact base structure

5.5 Rule base

The rule base consists of a set of production rules describing a parser such as Earley's parser. It is written using the language accepted by *EXINF*, detailed in Chapter 5. The rule base is used by the exploitation mode.

5.5.1 Vocabulary and rule base syntax

The language of expression allows communication with the expert. This language is used to describe the rule base. Like any language, it is described by a vocabulary and grammar.

5.5.5.1 Vocabulary

The vocabulary includes:

- The *identifiers* in the form of strings that represent predicates;
- The *variables* represented by alphanumeric identifiers preceded by the symbols "**?**", in the case of a single variable (*i.e.* substituted by one string), or by "**&**" in the case of many variables (*i.e.* substituted by more than one string);
- Reserved words that have a specific meaning for the system: **IF, THEN RULE, FACT, ADD, EXECUTE, DELETE, END.**

5.5.5.2 Rule base syntax

The rule base syntax (or more precisely the rules of production) that generates the language is written in the normal form of Backus-Naur form (BNF) as expressed in Figure 4.10 below.

/* Methodology 4.8 */

/* Syntax used by EXINF */
/* METH48 */

```
<declarations> :: = <declaration> [ <declaration> ]* END
<declaration> :: = < rule-declaration> | < fact-declaration>
<rule-declaration> :: = RULE [ <name> ]* <rule>
<name> :: = string of 5 characters
<rule> :: = IF <antecedents> THEN <consequents>
<antecedents> :: = ( <premise> ) [ <antecedents> ]*
<premise> :: = <predicate> <element>+
<predicate> :: = classical identifier
<element> :: = <constant> | <variable>
<constant> :: = classical identifier
<variable> :: = ?<constant> | &<constant> | ?- | &-
<consequents> :: = {<conclusion> |<action>}[ <consequents>]*
<conclusion> :: = ADD ( <predicate> <element>+) | DELETE
            (<predicate> <element>+)
<action> :: = EXECUTE (<expression>)
<expression> :: = write ( message ) | <variable> |
        {<variable> | <constant>} <operation> {<variable> |
        <constant>}
<operation> :: = arithmetic operation
< fact-declaration> :: = FACT <fact>
<fact> :: = <predicate> [ <constant> ]+
```

Standard notations
*- Symbol * indicates existence of 0 or more symbol (s)*
- Symbol + indicates existence of 1 or more symbol(s)
- Symbol **`?identifier`** *concerns only one* **`identifier`** *variable*
- Symbol **`&identifier`** *concerns more than one* **`identifier`** *variable*

Figure 4.10 – METH48 Syntax used by *EXINF*

5.5.2 Automatic syntactic analysis

Once learning is finished, *GASRIA* is ready to work as a simple syntactic analyzer *i.e.* switches to the exploitation mode of operation. In this case, the user is supposed to learn a language from the system. Thus, the user supplies new sentences to be recognized. *EXINF* deals with these sentences as a syntactic analyzer, or rule base, using the grammar of the language *i.e.* the content of the fact base which has been updated during the learning phase. *GASRIA* operates a classification on the membership of these new sentences and informs the user. In addition, the system always questions the results obtained because it has to rely on experience. For that, the system keeps track of all details of the session with the user and transmits it to the expert for a possible validation of responses, thus enriching the language. The eventual mistakes are corrected using the *ILSGInf* module. Note that these errors affect the answers provided by *GASRIA* that the expert has refuted.

6. Parsing

6.1 Notation

In all subsequent analysis, we use the following notation:

Symbols $A, B, C,\ldots$ to range over non-terminals N, with symbols $a, b, c, \ldots$ to range over the input alphabet Σ.

Symbols X, Y range over $(N \cup \Sigma)$.

Symbols α, β, γ range over $(N \cup \Sigma)^*$

Symbols $v, w, x,\ldots$ range over Σ^*

For a fixed grammar, the binary relation $(\Rightarrow)$ is defined over $(N \cup \Sigma)^*$ such that

$\gamma A\delta \Rightarrow \gamma\alpha\delta$ whenever $(A \rightarrow \alpha) \in P$.

Multiple derivation, closure of $(\Rightarrow)$, is denoted $(\overset{*}{\Rightarrow})$.

6.2 Earley's algorithm

6.2.1 The idea

We briefly present here the Earley's algorithm, before introducing our declarative adaptation, detailed in Chapter 5. Let $G = (N, \Sigma, P, S)$ be a CFG. We associate with G a set of symbols, called *dotted items*, specified as:

$$I_E = \{ [A \rightarrow \alpha \bullet \beta] \mid (A \rightarrow \alpha\beta) \in P\}.$$

Dotted items are used to represent intermediate steps in the process of recognition of a production of the grammar. The sequence of symbols between the arrow and the dot indicates the sequence of constituents recognized so far at consecutive positions within the input string. More precisely, given a production:

$$p : (A \rightarrow X_1X_2 \ldots X_r),\ r >= 0,$$

the process of recognition of the right-hand side of p is carried out in several steps. We start from item $(A \rightarrow \bullet X_1X_2 \ldots X_r)$, attesting that the empty sequence of constituents has been collected so far. This item represents a prediction for p. We then proceed with item $(A \rightarrow X_1 \bullet X_2 \ldots X_r)$,after the recognition of a constituent X_1 , and so on. Production p has has been fully recognized only if we reach item $(A \rightarrow X_1\ X_2 \ldots X_r\bullet)$, attesting therefore the complete recognition of a constituent A.

Given a string $w = a_1a_2\ldots a_n$, with $n >= 0$ and each a_i a terminal symbol, a position within w is any integer i such that $0 =< i <= n$. In what follows, E is a square matrix whose entries are subsets of I_E and are addressed by indices that are positions within the input string. Entries are denoted as $E_{i,j}$. The insertion by the algorithm of item $[A \rightarrow \alpha \bullet \beta]$ in $E_{i,j}$, $i =< j$, attests the fact that the sequence of constituents in α exactly spans the substring $a_{i+1}\ldots a_j$ of the input. Control flow is not specified in the method below, since it is usually regulated by means of a data structure called *agenda*, which directs the incremental construction of the table by means of an iteration: starting from an empty table, items are added as long as needed, and with the desired priority.

6.2.2 Detailed steps of Earley's algorithm

/* Algorithm 4.1 */
/* ALGO41 */

/* Earley's Algorithm */

/* In a list I_p where *p*> 0, item "$A \rightarrow \alpha \bullet \beta, q$ " has the meaning described below */

//* By neglecting the first *q* symbol(s) of the sub-string, we can parse the string α in the case where the string β comes after it (β is called a prevision string). Therefore, in the case where $\beta = \varepsilon$ (empty string), we say that the string α is parsed by neglecting *q* symbols. *//

1 Construction of I_0

1.1- **FOR** every rule $S \rightarrow \alpha$ in *P*, **ADD** [$S \rightarrow \bullet \alpha$, 0] in I_0.

1.2- **IF** the item is of the form [$B \rightarrow \gamma \bullet, 0$] in I_0,
THEN FOR every item of the form [$A \rightarrow \alpha \bullet B \beta$, 0] in I_0,
ADD item [$A \rightarrow \alpha B \bullet \beta$, 0] to I_0.

1.3- **IF** [$A \rightarrow \alpha \bullet B \beta$, 0] is in I_0
THEN FOR every rule of the form $B \rightarrow \gamma$
ADD item [$B \rightarrow \bullet \gamma$, 0] to I_0.

1.4- **REPEAT** 12 et 13 **UNTIL** no item can be added

2 Construction of I_p from lists $I_0,\ldots,I_{p-1}$

2.1- **FOR** every item of the form [$B \rightarrow \alpha \bullet a \beta, q$] in I_{p-1} such that a= a_p in ω,
ADD [$B \rightarrow \alpha a \bullet \beta, q$] in I_p

2.2- **FOR** every item of the form [$A \rightarrow \gamma \bullet, q$] in I_p,
AND FOR every item of the form [$B \rightarrow \alpha \bullet A \beta, k$] in I_q,
ADD [$B \rightarrow \alpha A \bullet \beta, k$] to I_p

2.3- **FOR** every item of the form [$A \rightarrow \alpha \bullet B \beta, q$] in I_p,
AND FOR every rule of the form $B \rightarrow \gamma$ in P,
ADD [$B \rightarrow \bullet \gamma, p$] to I_p

2.4- **REPEAT** 22 et 23 **UNTIL** no item can be added

3 Eventual acceptance of a string of length n

3.1 **IF** n+1 lists are constructed
AND an item of the form [$S \rightarrow \alpha \bullet, 0$] is found in I_n
THEN string is accepted
3.2 **ELSE** string is refused.

Algorithm 4.1 - ALGO41 Earley's algorithm

6.2.3 Correctness

The string w is accepted if and only if $[\ S \rightarrow\ \bullet\ \alpha\ ] \in E_{0,n}$ for some $(\ S \rightarrow\ \alpha\ ) \in P$,. The correctness of the algorithm immediately follows from the property below.

Property: in Earley's algorithm described above, an item $[\ A \rightarrow \alpha \bullet \beta\ ]$ is inserted in $E_{i,j}$ if and only if the following conditions hold:

C1. $S \overset{*}{\Rightarrow} a_1a_2\ldots a_iA\ \gamma$, for some γ, and

C2. $\alpha \overset{*}{\Rightarrow} a_{i+1}\ldots a_j$

For methods cruder than the Earley's algorithm, membership of an item in some entry may merely be subject to condition C2, which is sufficient for determining the correctness of the input. However, Earley's algorithm is more selective, as is apparent from condition C1, which characterizes the so-called top-down filtering capability of the method. Condition C1 guarantees that only those constituents are predicted that are compatible with the portion of the input that has been read so far. Assuming the working grammar is fixed, a simple analysis reveals that the considered algorithm runs in time $O(n^3)$.

6.2.4 Earley and CYK algorithms

Earley's algorithm is an example of chart parser class. Cocke-Younger-Kasami algorithm (CYK) is another example (Manacher, 1978). These algorithms are both based on dynamic programming. The choice of Earley's algorithm is dictated by considerations related to complexity and simplicity of implementation. The time complexity of both algorithms is $O(n^3)$ where n is the length of the sentence. However, Earley's algorithm performs better in most situations. Indeed, it reaches $O(n^2)$ for unambiguous grammars and $O(n)$ for $LR(k)$. For the space complexity, Earley's consumes $O(n)$, while *CYK* needs $O(n^2)$. Earley's algorithm can parse all CFGs, but *CYK* parses only grammars in Chomsky normal form (CNF). For these reasons, we have used Earley's parser for our system and not *CYK*.

6.3 Additional definitions

6.3.1 Types of sentences and partial derivatives (PaDe's)

(1) Let *C* be a *global sentence* defined as a blank-free string of characters in any artificial language.

(2) A *sub-sentence* of a given global sentence *C* is any recognized sub-sequence of characters in this global sentence.

(3) A *partial derivative* (*PaDe*) of C is the parse sub-tree of any sub-sentence.

(4) Any parsing based on *PaDe's* is termed *partial parsing* and its corresponding algorithm called *partial parsing algorithm* (PPA).

(5) *A list* (resp. *sub-list*) is the result of parsing using Earley's algorithm for a global sentence (*resp.* sub-sentence).

(6) *More general PaDe :* we say of a *PaDe* that it is more general than another if the former contains the minimum number of terminals *i.e.* the maximum of terminals are transformed into non-terminals. The resulting *PaDe* is therefore smaller.

(7) *More general grammar:* In order to obtain a more general grammar, it is necessary to add a more general rule to each step of the generalization process. The rule to be added is always of the form "$S \rightarrow DP_i$" where DP_i is the concatenation of *PaDe's*.

6.3.2 Derivation trees

We need derivation trees [ALS07] for the construction of our grammar from the initial stage to the final stage. A labeled and ordered tree D is said to be a derivation tree for a CFG of the form G = (N, Σ, P, S) if :

1- The root of D is labeled by S ;

2- $D_1,\ldots, D_k$ are sub-trees of direct descendents of S and the roots of D_i are x_i, then $S \rightarrow x_1 \ldots x_k$ is a production rule in P. D_i must be a derivation tree for G = (N, Σ, P, x_i) if x_i is a non-terminal and D_i is a unique node (named x_i if it is a terminal).

3- D_1 is the only sub-tree of D, the root of D_1 is ε . In this case, the production rule $S \rightarrow \varepsilon \in P$.

Example: G = (N, {a, b, ε}, P, S) where :

N = { S }

P = { S $\rightarrow$ a SbS, S $\rightarrow$ bSaS, S $\rightarrow$ ε }

Among the syntactic trees of this grammar, we find those of Figure 4.11.

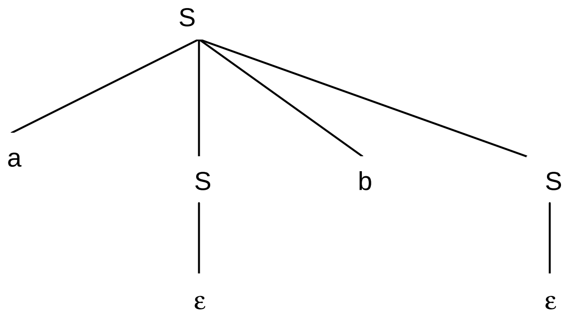


Figure 4.11 – DIAG41 – A derivation tree of G

6.4 Motivation for using PaDe's

Now, we use the additional definitions above to proceed further through an example of *PaDe* use. For example, we have the following problem:

Initially recognized global sentence: a+b

New global sentence to be recognized: (a+b)

How can we handle this new sentence? In classical parsing: new global sentence refused because of first unrecognized character "(".

With the use of *PaDe's*:

PaDe1 = (

PaDe2 = a+b

PaDe3 =)

Result: *Accept* DP2. *Reject* all other sub-sentences.

Head of sub-sentence in global sentence	Sub-sentence length	*PaDe's* of sub-sentence : Result is a dynamic string	
0	1	D	(

1	3	S	a+b
4	1	E	)

Table 4.1 TAB41 PaDe's construction for (a+b) based on a+b

7. Learning in GASRIA

7.1 Learning characteristics

It is useful to make the following remarks concerning learning in *GASRIA.*

- No pre-classification is required from the expert when supplying the sentences for training. Therefore, the system does not need to make any search in the sentences space.
- The system gradually builds a grammar that generates these sentences.
- For the validation of any learning system, we need an exploitation module to check whether learning has been done correctly. We use the module *EXINF* for parsing.
- We use the property that rules can be written in the forms $A \rightarrow BC$, or $A \rightarrow a$.

7.2 Learning strategy implementation

Implementation concerns the development of all required phases, *i.e.* those that take in charge the initial grammar construction, partial parsing, the refinement cycle and the treatment of partial derivatives. All these phases are described in details in Chapter 6 concerning *ILSGInf* module.

8. Results and discussion

8.1 *GASRIA* implementation

A program has been developed using Microsoft Visual C++ release 6.0 (MVC++6.0™) under Microsoft Windows XP™. This program takes full advantage of the object-oriented method. Grammar generation follows the steps described in Figure 4.12 below:

/* Methodology 4.9 */
/* METH49 */

/ Grammar generation */*

```
- Read first positive sentence
- Generate an initial grammar
- Use refinement cycle
      -- Read a new positive sentence
      -- Generalize this grammar
- Give results
- Test grammar validity on additional sentences, with
  eventual recourse to human expert
```

Figure 4.12 METH49 Grammar generation

Refinement cycle for grammar generation follows the steps described in Figure 4.13, but with no specialization.

/* Methodology 4.10 */
/* METH410 */

/ Refinement cycle */*

```
- Use result given by PPA
- Generalize grammar if result gives failure for a positive
  example
- Specialize grammar if result gives success for a negative
  example
```

Figure 4.13 METH410 Refinement cycle in grammar generation

8.2 Example

The details of how to operate the complete system is described in the two forthcoming chapters. We only give here the basic steps as building blocks of the grammar induction as performed by *GASRIA*, on a simple example. The class of languages learned by GASRIA are given in Appendix 1.

Problem Statement

1. *Given* a set of positive examples $S^+ = \{ a+b, (a+b)\}$ from a language L.
2. *Infer* a grammar that can generate L in the limit.
3. *Describe* all the steps of the induction process and consider both learning phase and exploitation phase.

8.2.1 Learning phase: *ILSGInf* use

1. *Initial sentence introduction:* the expert introduces the sentence: a+a
2. *Initial grammar generation*

The program generates the following initial grammar $G_0 = (N_0, \Sigma_0, P_0, S)$ where:

$N_0 = \{A, B, S, C\}$; $\Sigma_0 = \{a, +\}$; S initial symbol in N_0 .

$P_0 = \{ A \rightarrow a, B \rightarrow +, C \rightarrow AB , S \rightarrow CA \}$

3 *Parsing of the new sentence*: *EXINF* as parser rejects the new sentence (a+a) according to G_0 since the opening parenthesis "(" and the closing one ")"are not recognized by G_0.

4. *Refinement cycle*

3.1 *Sub-lists construction*: the partial parsing algorithm (PPA) uses all sub-lits for all sub-strings for analysis.

3.2 *PaDe's construction*: all *PaDe's* are obtained.

3.3. *Generalization*: The program selects the most general rule which is the concatenation of the most general *PaDe's*. This selection gives: $S \rightarrow DSE$

3.4 *Grammar induction*: The program generates the following induced grammar:

$G_1 = (N_1, \Sigma_1, P_1, S)$ where:

$N_1 = \{A, B, S, C, D, E\}$; $\Sigma_1 = \{a, +, (,)\}$;

$P_1 = \{ A \rightarrow a, B \rightarrow +, C \rightarrow AB , S \rightarrow CA, D \rightarrow (, E \rightarrow) , S \rightarrow DSE\}$

3.5 *Introduction of new positive sentence*: the expert introduces (a+a)+(a+a)

3.6 *Parsing of the new sentence*: Go to Step 3 above.

3.7 *Generation of the third grammar of the form*:

$G_2 = (N_2, \Sigma_2, P_2, S)$

$N_2 = \{A, B, S, C, D, E, F\}$; $\Sigma_2 = \{a, +, (,)\}$;

$P_2 = \{ A \to a, B \to +, C \to AB, S \to CA, D \to (, E \to) , S \to FBF, F \to DSE, S \to F\}$

4. *Grammar transformation to Chomsky normal form (CNF)*

The grammar is improved using the CNF as follows:

Rule $F \to DSE$ is replaced by : $F \to DH$ and $H \to SE$

Rule $S \to FBF$ is replaced by : $S \to FG$ and $G \to BF$

The actual grammar is now the most general grammar since it can generate all (infinite) strings of the form : a+a, (a+a), (((a+a))), (((a+a))+(a+a)), …

Formally the actual grammar generates the following language:

expression → a+a

expression → (expression)

expression → expression + expression

Discussion

Only three positive examples a + a, (a+a), (a+a) + (a+a) are needed to infer a grammar that generates all strings belonging to L. Our method does not produce any counter examples; which represents an important result. Chapter 6 provides more details of how this is done by *ILSGInf.*

8.2.2 Exploitation phase: *EXINF* use

The grammar G_2 is introduced in the fact base of *EXINF*. At this stage *EXINF* is able to parse any sentence of the language L.

1. *Recognized sentence*: ((((a+b)+(a+b)+(a+b)+(a+b)))). The analysis gives success.

2. *Unrecognized sentence*: ((a+b)+a+b. The analysis gives failure.

Chapter 5, Section 5 describes in more details of how this is done by *EXINF*.

9. Conclusion

In this chapter, we reported an early attempt in bridging the gap between GI and first-order logic (FOL). Based on this idea, *GASRIA* has been designed and developed as a GI system that can infer some CFG's from positive examples. Thus, the system behaves as a parser with the ability to learn inductively, with the learning module, and to reason through an FOL-based programming environment, *EXINF*, developed for a broader context. For the tested languages, the number of examples required for induction is very small, here not exceeding five examples. On the other hand, the generated language is not empty since it contains at least the introduced examples, and generates no counter example. The combination of GI and FOL can be regarded as an important step towards "intelligent" compilers. The results obtained in this chapter are further expanded in Chapter 5, reporting in details the parsing problem using logic, and complemented by learning in Chapter 6.

CHAPTER 5

INFERENCES THROUGH *EXINF* INTELLIGENT PARSING ISSUES

1. Introduction

This chapter is concerned with coupling first-order logic (FOL) and grammatical inference (GI) aiming to construct an intelligent parser (IntPar). Our goal here is to establish the "methodological production" rule *FOL and GI → IntPar*. We mainly build our contribution on methods drawn from FOL as applied to parsing. Starting from truly first principles, we design and develop a rule-based first-order deduction system, called *EXINF*, and couple it with a learning module, called *ILSGInf*, for the purpose of GI. While we stress the importance of the logic-based methods used for implementation, we also raise the issues imposed by such a coupling. Although *EXINF* is used here for parsing, it can also be used as a stand-alone inferential system. On top of that general-purpose usage, the application of *EXINF* is two-fold; it can be considered as an ordinary sentence parser, or as an extended Earley's parser for a given grammar. More importantly, *EXINF* can contribute to the inference of one unknown grammar from positive examples in conjunction with the learning module *ILSGInf*, described in Chapter 6. In summary, *EXINF* can be used as a stand-alone inference engine implementing both forward and backward chaining, as a "crude" parser or an "intelligent" parser. All these issues are addressed in this chapter. The resulting implementation gives a powerful unified framework able to meet one of the challenges of GI.

The chapter is structured as follows. In Section 2, we formulate our problem by specifying the refined objectives. *EXINF* parsing capabilities are described in Section 3 while Section 4 explains its reasoning mechanisms based on forward chaining. Section 5 is devoted to the implementation of the system and to experimental results. Finally, lessons learned are drawn from the actual results and proposals are highlighted pointing towards the improvement of the actual work.

2. *EXINF* objectives

The objective is to concentrate on the description of a first-order rule-based or logic programming environment, called *EXINF*, capable of reasoning on assertions related to an unknown grammar to be induced. While the operation of the complete system, inferential and learning has been reported in Chapter 4, we here stress the importance of the logic programming environment *EXINF*. The main objectives of this system are:

(i) *Stand-alone inferences capability*, *i.e. EXINF* is a system based on FOL that can infer knowledge for general-purpose application. In this respect, *EXINF* can be compared to those available over the Web, *e.g. NASA CLIPS* rule-based language.

(ii) *Simple parsing*, *i.e. EXINF* can be used to parse any language based on a CFG.

(iii) *"Intelligent" parsing*, *i.e. EXINF* can infer one unknown CFG from positive examples, in conjunction with a learning module, namely *ILSGInf*.

(iv) Moreover, *EXINF* is a system developed from scratch and, as such, is easier to update and to adapt for special applications such as the one we are dealing with. Our developed logic programming environment has the inferential and complementary characteristics described below.

2.1 Inferential characteristics

The central process in any intelligent system is *inference*, defined here as the ability to add valid new propositions to a knowledge base or to derive the truth of propositions not explicitly contained within the knowledge base.

(i) Rule-based system: Knowledge is rule-based *i.e.* it is represented by production rules.

(ii) First-order, predicate logic: Reasoning is based on first-order or predicate logic.

(iii) Variables: Use of variables are allowed. These are instantiated (or bound) by constants from the fact base.

(iv) Closed world assumption: Like many systems (*e.g.* Prolog), our system works with the *closed world assumption*, *i.e.* a goal that is not explicitly expressed in the fact base, or that cannot be inferred from it, is considered as false. This assumption does not reduce the capabilities of our system since the grammar contains all information concerning the language under consideration. Indeed, any grammar generates all the instances of the corresponding language. The difficulty resides in inferring a grammar, not in using it.

(v) Backtrack characteristics: in the case of failure, search for a new solution is done by returning to the state preceding actual failure.

(vi) *Resolution principle*: The system does not use the Robinson's resolution principle. Therefore, it can be easily adapted.

(vii) *Forward chaining and backward chaining*: The system uses both forward and backward chaining for deriving or proving new knowledge. Only forward chaining is used and described in this chapter.

2.2 Parsing characteristics

A problem that often faces a learning system designer lies in the difference between the types of representations used to describe the examples, on the one hand, and the concepts describing these examples, on the other. In our case, an example is a string. As for the concepts or generalizations, it consists of a context free grammar (CFG).

It is clear that the difference between a string and a grammar is important. For minimizing this difference, we rely on syntax trees which are located halfway between these two approaches. We use the link between a string of characters and a grammar as a means of transforming examples from string representation to a closer representation with respect to a grammar. This transformation can be seen as a process of interpretation. Thus, in learning mode, the parser is used for this rapprochement.

2.3 Complementary characteristics

(i) *Parsing*: We use an adapted version of Early's algorithm for parsing [Ear70].
(ii) *Learning*: A description of the learning module *ILSGInf* is given in Chapter 6.
(iii) *Integration*: An integrated implementation involving both learning and parsing is reported in [HH07a].

3. First-order logic (FOL) considerations

3.1 Rule-based deduction systems

3.1.1 Rules and operation

Rule-based problem-solving deduction-oriented systems are built using rules of the form:

<*if* antecedent…*then* conclusion>.

The antecedent is also known as premise, condition or left-hand side (*LHS*). The conclusion is also known as consequent, action or right-hand side (*RHS*). The rules are therefore interchangeably called *if-then* rules or *antecedent-consequent* rules *condition-action* rules [Win93].

Rule-based systems can either work in a forward or backward chaining mode. In the first mode, we move from the *LHS* to the *RHS*. We therefore use the *condition* pattern to identify the *action* pattern. During the forward chaining mode, whenever a *RHS* pattern is observed to match a fact in the fact base, the condition is *satisfied*. A rule is *triggered* whenever all *RHS* patterns are satisfied. When a triggered rule establishes a new fact, the rule is said to be *fired*. In deduction systems, all triggered rules generally fire. In the case where many rules need to be fired, a *conflict-resolution* procedure is needed to decide which rule to fire. All deduction systems whether forward or backward comprise an inference cycle consisting of three phases, namely:

Detection → conflict resolution → execution or firing

During the first phase, which is the detection phase, a conflict resolution set (*CRS*) is constructed and which consists of all candidates rule. The second phase is conflict resolution proper *i.e.* the choice of the rule to trigger. The last phase is the deduction phase during which the chosen rule is finally fired. A termination procedure is used to end the search.

3.1.2 Basic components of rule-based systems

The basic components of a rule-based problem-solving deduction system are a rule base and a fact base [Win93].

(i) The fact base

/* Methodology 5.1 */
/* METH51 */

/ Fact Base*/*

```
Lexically: There are application-specific symbols
       and pattern symbols.
Structurally: assertions are application-specific
       symbols and patterns are application-specific
       symbols and pattern symbols.
Semantically: the assertions denote facts in some
       world. Facts cannot be false but assertion
       can.

Constructors
       Add an assertion to working memory.
Readers
       Produce a list of matching assertions
       in fact base given a pattern.
```

Figure 5.1 – METH51 Fact base

/* Methodology 5.2 */
/* METH52 */

/ Rule Base */*

```
Lexically: There are application-specific symbols
    and pattern symbols.
Structurally: Patterns are lists application-
    specific symbols and pattern symbols, and rules
    consist of patterns. Some of these patterns
    constitute the LHS of the rule and the others
    constitute the RHS of the rule.
Semantically: Rules denote constraints that enable
    procedures to seek new assertions or to
    validate a hypothesis.

Constructors
    Construct a rule, given an ordered list of LHS
    patterns and a RHS pattern.
Readers
    Produce a list of a given rule's RHS patterns.
    Produce a list of a given rule's LHS patterns.
```

Figure 5.2 – METH52 Rule base

3.2 Knowledge-base engineering issues

3.2.1 Knowledge acquisition

To acquire or extract the necessary knowledge from a human expert in order to code it as rules understandable by a computer, the following strategy is used, as decribed in Figure 5.3.

/* Methodology 5.3 */
/* METH53 */
/ Heuristics for learning from an expert */*

```
-  Ask about specific situations to learn the
   expert's general knowledge

-  Ask about situations pairs that look identical
   but that are handled differently, so that the
   expert's vocabulary becomes understandable.
```

Figure 5.3 – METH53 Heuristics for learning from an expert

3.2.2 Knowledge explanation

In order to answer a question about the behavior of a rule-base deduction system, the following heuristics are used, as explained in Figure 5.4 below.

/* Methodology 5.4 */
/* METH54 */

/ Heuristics for explaining results given by a rule-base system */*

```
To answer a question about the reasoning done by a
rule-base deduction system:

IF the question is a HOW question,
THEN report the assertions connected to the RHS of
     the rule that established the assertion
     referenced in the question.

IF the question is a WHY question,
THEN report the assertions connected to the LHS of
     the rule of all rules that used the assertion
     referenced in the question.
```

Figure 5.4 METH54 Heuristics for explaining results given by a rule-base system

3.3 Forward chaining (FC

The forward chaining is based on the *modus ponens* rule which states that:

$$((\ p \rightarrow q\ )\ and\ p\ ) \models (q)$$

The symbol $\models$ represents entailment. In this logical expression, the *RHS*, *q,* is said to be entailed, inferred or derived from the LHS, $((\ p \rightarrow q\ )\ and\ p\ )$. Both *LHS* and *RHS* are related by two fundamental theorems:

Deduction theorem: ($LHS \models RHS$) $\leftrightarrow$ ($LHS \rightarrow RHS$ is valid or is a tautology).

Contradiction theorem: ($LHS \models RHS$) $\leftrightarrow$ (*LHS* AND NOT(*RHS*) is unsatisfiable).

In our situation, parsing is a bottom-up process since parsing begins from the facts and tries to attain some specified goals. Therefore, it is more suitable to use forward chaining. We are in a situation where the goal is not precisely known. Indeed, at the outset, the system ignores whether or not a given sentence belongs to the language under consideration.

3.4 Backward chaining (BC)

Backward chaining is goal-driven reasoning approach. It attempts to answer a question of the form: "*how* did we reach this conclusion (goal)?" Starting from this specific conclusion, the premise(s) is (are) tried as sub-goals to be proved by tracing back to eventually meet facts. Therefore, this approach works back from the conclusion or query. If this query is true then no proof is needed. Otherwise, the algorithm finds those implications in the knowledge base that conclude the query. All premises become sub-goals to be proved. If all the premises of one of these implications can be proved true, by backward chaining, then the query is true. The process is therefore repeated until it reaches a set of known facts that forms the basis of the proof. In backward chaining, *modus ponens* is run in reverse. Backward chaining is a *sound inference* rule *i.e.* a rule that yields true derived conclusions provided that the conditions are true. It is useful to distinguish between reasoning with backward chaining, and *reasoning backwards*, starting from known consequents to unknown antecedents. To be specific, by reasoning backwards we mean if the consequent of a rule is known to be true, then the antecedent will be true as well. This is usually referred to as *plausible reasoning*. This can be expressed in the form $((p \rightarrow q) \text{ and } q) \equiv (p)$ and is known as *logical abduction*. For example, from the sentence "all Gamma Computers are fast" and the "My computer is fast", we can infer the eventually false sentence "My computer is Gamma Computer". Proof by contradiction is an example of use of backward chaining. It can alternatively be expressed by the so-called *modus tollens* rule which states that:

$$(p \rightarrow q) \equiv (\neg q \rightarrow \neg p).$$

Because the backward chaining is goal-directed, we have therefore to establish a list containing the goal and all relevant sub-goals. Although *EXINF* implements also the backward chaining, it will not be described here, because of no concern.

3.5 Backward chaining *vs.* forward chaining

Choosing one mode of chaining depends on the problem under consideration. We can use some rules of thumb or heuristics to find an acceptable choice. Let us define a *meta-heuristic i.e.* a heuristic of how to choose heuristics themselves. Any meta-heuristic has to produce a heuristic that reduces the search state space of the problem. Applying this meta-heuristic, we readily find the steps of choosing between the two modes of chaining. Whenever the rules are such that a typical set of facts can lead to many conclusions, we say that the system exhibits a high degree of *fan out*. In this case, we choose a backward mode. Alternatively, whenever the rules are such that a typical hypothesis can lead to many questions, the system is said to exhibit a high degree of *fan in*, which argues for the use of forward chaining. Of course, in many situations, these concepts of fan in and fan out cannot be used since no one dominates. In this case, we have to use other heuristics such as *amount of facts* heuristics*. The meta-heuristic* is described in Figure 5.5 below [Win93].

/* Methodology 5.5 */
/* METH55 */

/ Backward chaining vs. forward chaining * /*
/ BC vs. FC */*

/* Level 0 : META-HEURISTIC //

/* Heuristic has to reduce the solution state space */

/ * Level 1 : Choose *fan in* and *fan out* heuristics */

***1 fan in* and *fan out* calculation**

1.1 **FOR** every rule base find the *fan in*, alternatively find the number of consequents that can be instantiated

1.2 **FOR** every rule base find the *fan out*, alternatively find the number of premises that can be instantiated.

2* Comparison between *fan in* and *fan out

```
IF fan in = fan out
   THEN choice between BC and FC is done with equal
       weight
        ELSE
                IF fan in > fan out THEN choose FC
                ELSE choose BC

/ * Level 2 : Choose the amount of facts heuristics */

IF no facts are available

AND interest is in whether one of many possible
   conclusions is true

    THEN use BC
IF all possible facts are available

AND interest is in deriving all possible conclusions
   from those facts

    THEN use FC
```

Figure 5.5 Backward chaining *vs*. forward chaining

4. EXINF Architecture

4.1 Design diagrams

4.1.1 Use case diagram

There are two external modes when using *EXINF*. These modes are referred to as exploitation and learning modes. Figure 5.6 shows the use cases describing both of them in relation with the two main actors *i.e.* the human expert teaching the system *EXINF* in the quest of grammar construction and the ordinary user, looking for sentences parsing.

- *Exploitation mode*: it concerns any user interested in parsing a given sentence using a given grammar.
- *Learning mode*: it concerns a human expert acting as a teacher *via ILSGInf*.

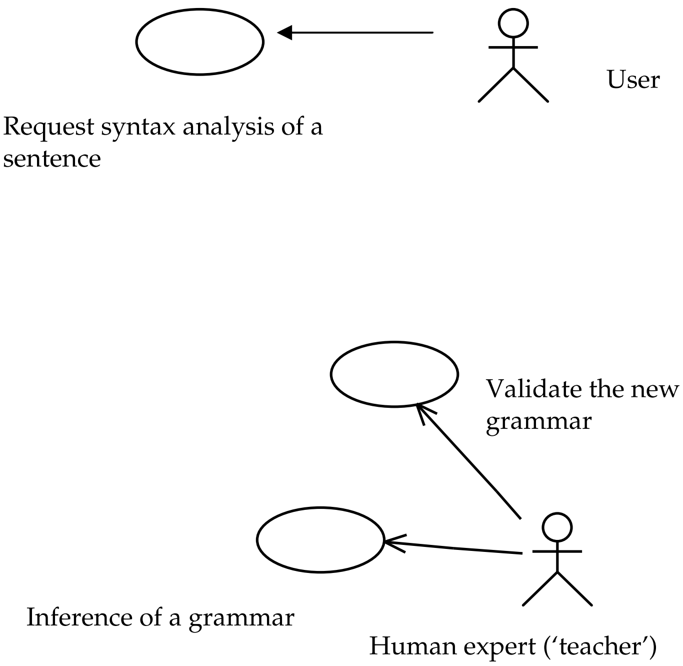


Figure 5.6 ARCH51 *EXINF* Use case diagram

4.1.2 Class diagram

Figure 4.7, in Chapter 4 described the main classes in *EXINF* class diagram. It depicts the overall architecture of *EXINF* with the broader system. It is mainly used for readability and maintenance.

4.3 The three *EXINF* layers

EXINF can be used for three different purposes, specified as layers. As a result, *EXINF* is a three-layered system, as depicted in Figure 5.7 and Figure 5.8. Only two of these are of interest to us *i.e.* the second and third layers.

4.3.1 *EXINF* first layer

Here *EXINF* can be used as a general purpose first-order logic (FOL) expert system shell, or inferential system, for knowledge-base systems development. It allows the user to introduce both rules and facts concerning a given problem. This is a general issue not discussed here.

4.3.2 *EXINF* second layer

This layer is more specialized than the first one. Here, the knowledge base is a set of parsing rules based on declarative form of Earley's algorithm. This layer is concerned with parsing a given sentence using a given grammar, introduced manually by the user. Here, *EXINF* is used as a "crude" parser or sentence recognizer like any other parser.

4.3.3 *EXINF* third layer

In the third layer, *EXINF* is used as a system that can infer a grammar from positive examples, or as "intelligent" parser. However, this issue cannot be undertaken by *EXINF* alone. It is resolved in conjunction with the learning module *ILSGInf*.

Figure 5.7 ARCH52 *EXINF* as a three-layered system

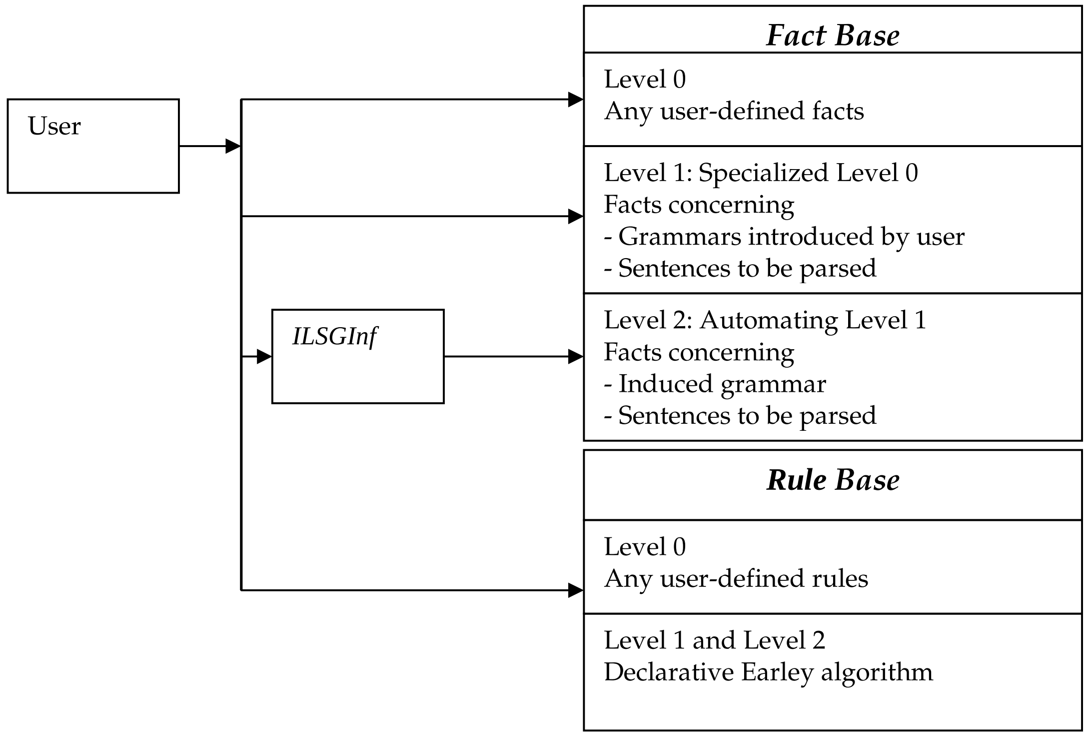
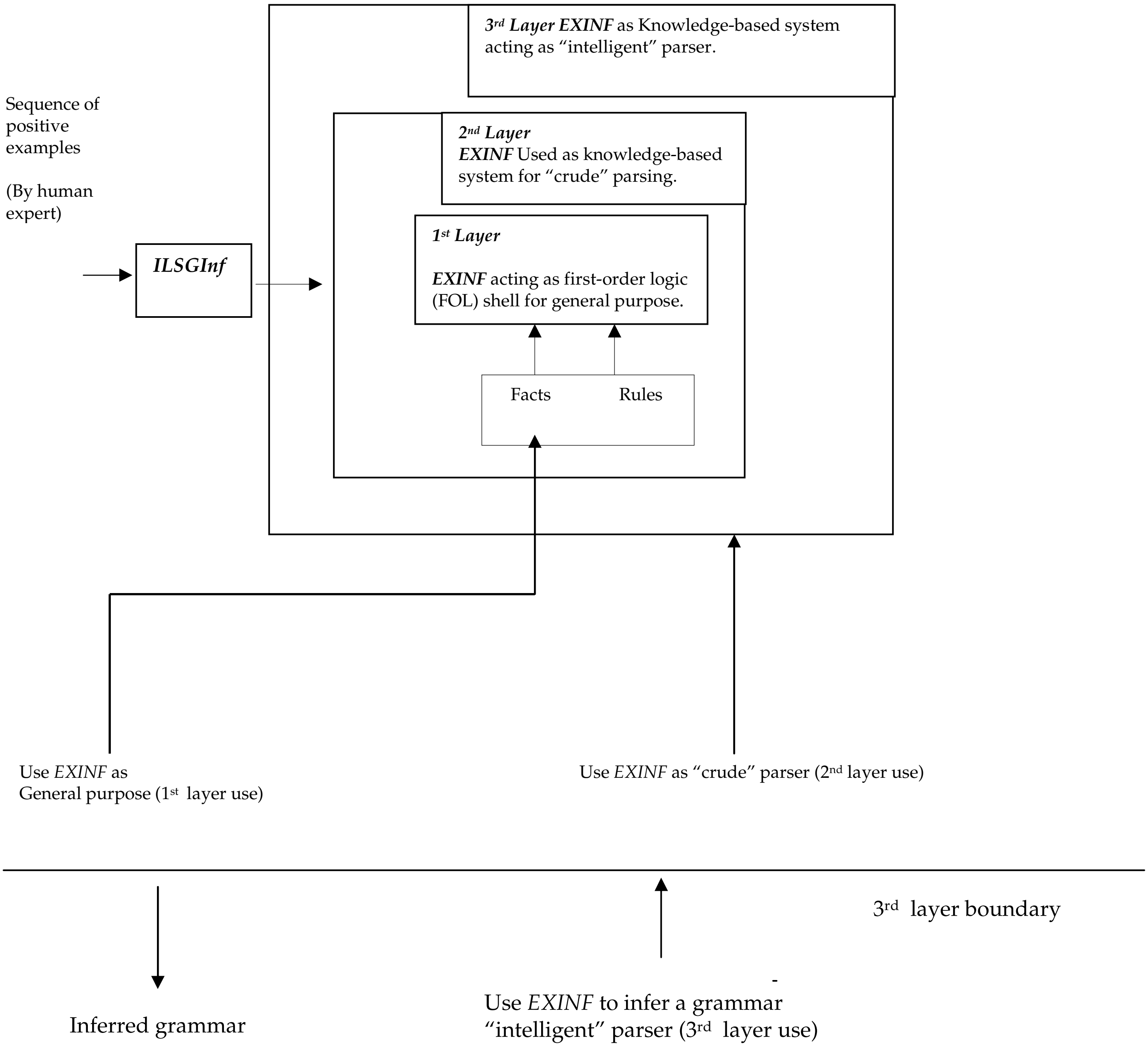


Figure 5.8 ARCH53 *EXINF* as a detailed three-layered system



5. *EXINF* - KBS used for parsing

5.1 *EXINF* as a knowledge-based system (KBS)

As a knowledge-based system (KBS) for parsing, *EXINF* is composed of:

1. Knowledge base which consists of :
   1.1. A fact base that contains the generated grammar and the sentence to be parsed.
   1.2. A rule base which contains the declarative form of Earley's algorithm.
2. Inference engine relying on:
   2.1. Forward chaining as far as parsing is concerned.
   2.2. Backward chaining, for other problems.

5.2 Declarative Earley's algorithm: rule base

EXINF rule base is built on Earley's algorithm ALGO41 described in Chapter 4 Section 6.2. The idea is to translate this algorithm into a declarative form.

5.2.1 Summarized Earley's algorithm

Let $G = (N, \Sigma, P, S)$ be a CFG. Let $w = a_1a_2\ldots a_n$, be an input string, $n >= 0$, and $a_i \in N$ for $1 = < i = < n$.

Compute the least $(n + 1)* (n + 1)$ table E such that the following conditions hold:

$[S \rightarrow \bullet\, \alpha] \in E_{0,0}$ for each $(S \rightarrow \bullet\, \alpha) \in P$, and

1. $[A \rightarrow \bullet\, \gamma] \in E_{j,j}$. if $[B \rightarrow \alpha \bullet A\beta] \in E_{i,j}$, $(A \rightarrow \gamma) \in P$;
2. $[A \rightarrow \alpha\, a_j \bullet \beta] \in E_{i,j}$ if $[A \rightarrow \alpha \bullet a_j\, \beta] \in E_{i,j-1}$;
3. $[A \rightarrow \alpha B \bullet \beta] \in E_{i,j}$ if $[A \rightarrow \alpha \bullet B\, \beta] \in E_{i,k}$, $[B \rightarrow \gamma \bullet\,] \in E_{k,j}$

Write an algorithm that undertakes this task declaratively.

5.2.2 Declarative Earley's algorithm

The solution is the declarative Earley's algorithm as described in Algorithm 5.1 below.

/* Algorithm 5.1 */
/* ALGO51 */
/ Declarative Earley's algorithm */*

```
RULE 1 /*construction of list l0*/
IF (RULE ?symbol &part)
  (initial_symbol ?symbol)
THEN ADD (I0 [ ?symbol → • &part , 0 ] )

RULE 2 /*construction of list l0/
IF (I0 [ ?symbol1 →  &- • , 0 ] )
  (I0 [ ?symbol2 →  &- • ?symbol1 &- , 0 ] )
THEN ADD (l0 [ ?symbol2  →  &- ?symbol1  • &-, 0])

RULE 3 /* construction of list l0 */
IF (I0 [ ?symbol1  →  &-•  ?symbol2 &-, 0])
  (rule ?symbol2 &part)
THEN ADD (I0 [ ?symbol2  → • &part, 0])

RULE 4 /* going from Ip-1 to lp : a character is recognized*/
IF ( I?p-1 [ ?symbol1 →  &part1 • ?a &part2, ?q])
  (string ?a ? string_remainder)
THEN EXECUTE ( ?p ?(p -1) + 1)
ADD (I?p [ ?symbol1  →  &part1 ?a •  &part2, ?q])
DELETE (string ?a &string_remainder)
ADD (string & string_remainder)

RULE 5 /*Filling list lp */
IF (I?p [?symbol1  → &- • , ?q])
(I?q [?symbol2 →  &part1 • ?symbol1 &part2, ?k])
THEN ADD (I?p [?symbol2 →  &part1?symbol1 •  &part2,?k])

RULE 6 /* Filling list lp */
IF (I?p [ ?symbol1  →  &- • ?symbol2 &-, ?q])
(RULE?symbol2 &part)
THEN ADD (I?p [?symbol2  → • &part, ?p])
RULE 7 /*Parsing of complete string*/
IF (string)
(length ?n)
(I?n [?symbol  → &part •, 0])
(initial_symbol ?symbol)
THEN ADD (write ("parsing is successfully achieved"))
DELETE (string)
```

Algorithm 5.1 - ALGO51 Declarative Earley's algorithm

5.3 *EXINF* reasoning mechanism

Once parsing characteristics have been settled, we now introduce the inference engine reasoning mechanism, based on forward chaining. This process handles parsing based on the declarative approach.

5.3.1 Forward chaining implementation

The following steps, describing the forward chaining, are a standard method of reasoning. For instance refer to [Win93].

/* Algorithm 5.2 */
/* ALGO52 */
/ Implemented forward chaining */*

```
UNTIL no rules produces new assertions,

/* Detection : Conflict Resolution Set  (CRS) Construction */
FOR each rule
  Try to match the first antecedent with an existing assertion.
  Create a new binding set with variable bindings established by
  the match.

  Using the existing variable bindings, try to match the next
  antecedent with an existing assertion. If any new variables appear
  in this antecedent, augment the existing variable bindings.

/* Conflict Resolution Phase or Execution Phase */

   REPEAT the previous step for each antecedent, accumulating
          variable bindings incrementally
   UNTIL
    • There is no match with any existing assertion using the binding
      set established so far. In this case, back up to previous match
      of an antecedent to an assertion, looking for an alternative
      match that produces an alternative, workable binding set.
    • There are no antecedents to be matched. In this case,
       - Use binding set in hand to instantiate the consequent,
       - Determine if the instantiated consequent is already
         asserted. If not, assert it.
       - Back up to the most recent match with unexplored bindings,
         looking for an alternative match that produces a workable
         binding set
/* Termination Test */
    • There are no more alternative matches to be explored at any
      level.
```

Algorithm 5.2 - ALGO52 Implemented forward chaining

5.3.2 Example

Assume we have the following knowledge base, given in Figure 5.9 below:

/* Application 5.1 */
/* APPL51 */

/ Fact base */*

```
R(a)
F(b)
```

/ Rule Base */*

```
Rule1    IF     R(?x)
         AND    F(?y)
            THEN   M(?x)

Rule2    IF    A(?x)
         AND   R(?x)
            THEN  print ("end of program")
```

Figure 5.9 APPL51 Example of facts and rules

In this case, we can see that **RULE1** is a potential candidate for triggering. Indeed, all its premises are satisfied by the fact base. But **RULE2** is not a candidate since the condition **A(?x)** cannot be bound with any fact in the fact base. The construction of the conflict resolution set (CRS) is based on the variables that can actually be instantiated. In our case, two types of variables are considered.

- The first type is called simple variable and is preceded by "**?**", *e.g.* **?x**. It captures one simple item of the data.
- The second, called commentary variable, is preceded by "**&**", *e.g.* **&y**. It incorporates a list of items.

Consider the following filter: **R(?x, ?y, a, &z).**

Consider the following data: **R** (*This is a good example*).

After filtering, the simple variables **?x**, and **?y** are respectively instantiated by *"This"* and *"is"*. The constant *"a"* is identical to the given constant. The variable **&z** is instantiated by *"good example"*. The overall result is: *"This is a good example"*.

6. Applications

We incrementally use all layers of *EXINF* to solve the problems described below.

6.1 Problem 1: regular language

We have a regular language of the form $L_1 = \{ w = (ab)^n, n>=1\}$. Use *EXINF* as a "crude" parser based on a grammar introduced as facts and on the rules embodied in declarative Earley's algorithm. The grammar is to be introduced manually by the user.

6.1.1 *EXINF* first and second layers

Since we are concerned with parsing, only the second layer is of interest to us. A possible grammar for L_1 is:

$$G_1 = (N_1, \Sigma_1, S, P_1)$$
$$\Sigma_1 = \{a, b\}$$
$$N_1 = \{A, B, S\}$$
$$P_1 = \{ \; A \rightarrow a, \; B \rightarrow b, \; S \rightarrow AB, \; S \rightarrow SS \; \}$$

(1) Filling the fact base

EXINF stores this grammar as facts as shown in Figure 5.10 below:

/* Application 5.2 */
/* APPL52 */

/ Fact Base for Tested Example 1*/*

/ Production rules stored as facts */*

```
FACT  RULE A  a                    // Fact1 //
FACT  RULE B  b                    // Fact2 //
```

```
FACT  RULE S  A  B                    // Fact3 //
FACT  RULE S  S  S                    // Fact4 //
FACT  initial_symbol S                // Fact5 //

        /* Sentence to be parsed and its length */

FACT  string   ababab                 // Fact6 //
FACT  length  6                       // Fact7 //
```

Figure 5.10 APPL52 Fact base for regular language $L_1 = \{ w = (ab)^n, n>=1\}$

EXINF represents each production rule in the grammar as a fact (Fact1, 2, 3, 4, 5). The sentence to be parsed and its length are also introduced in the fact base (Fact6, 7). Parsing is processed by *EXINF* as a sequence of forward chaining inference cycles.

(2) EXINF Typical Inference Cycle

1st Step: Detection

As described in Algorithm 5.2 above, this step involves the so-called detection or construction of conflict resolution set CRS.

CRS(0) = {RULE1}. In this special case, only RULE1 has all its premises instantiated with some facts and therefore RULE1 is the only candidate for eventual triggering. We use RULE1 for instantiation, *i.e.*, we use the description given in Figure 5.11 below:

```
              /* Application 5.3 */

                 /* APPL53 */

RULE1 /*construction of list l0*/

IF (RULE ?symbol &part)
(initial_symbol ?symbol)
THEN ADD (I0 [ ?symbol  → • &part ,0])
```

Figure 5.11 APPL53 Construction of list l_0*

2nd Step: Execution / conflict resolution

(i) Matching

First premise (RULE ?symbol &part) can be matched by FACT 1,2,3,4.

The second premise can be matched with FACT5.

(ii) Heuristics for premise choice

Now the obvious question is: "which premise to evaluate at this step"? Consider this question as a constraint satisfaction problem (CSP). All CSP search algorithms generate successors by considering possible assignments for only a single variable at each node in the search tree. The so-called minimum remaining value (MRV) is a common heuristic used in CSP. Like any heuristics, its aim is to reduce the search space. MRV heuristic chooses an unassigned variable that has the minimum number of remaining values, at some stage of the assignment process. Here the number of values assignable to a given premise has to be minimum. MRV heuristic is also called the most constrained variable (MCV) or fail-first heuristic; the latter because it picks a variable that is most likely to cause a failure soon, thereby pruning the search tree. If there is a variable *X* with zero legal values remaining, the MRV heuristics will select *X* and failure will be detected immediately—avoiding pointless searches through other variables which always will fail when *X* is finally selected.

(iii) Instantiation

Here the variable ?symbol is instantiated with value S.

(iv) Propagation

The last instantiation is then propagated in the entire rule.

The first premise will be (RULE S &part).

After propagation, the only facts that can be instantiated with this premise are now FACT3 and FACT4. Choose the first fact in list which is FACT3 and the variable &part is instantiated with A B.

(v) Conclusion execution

Now all premises of the rule are instantiated, therefore the system executes the rule's conclusion which is the insertion of the fact:

I_0 [?symbol $\rightarrow$ • &part , 0] as I_0 [S $\rightarrow$ • A B, 0] in the fact base.

(vi) Rule saturation

EXINF is based on rule saturation, *i.e.* it explores all possible inductions. It therefore tries to match the second premise with FACT4. So &part is instantiated with SS and the fact I_0 [S → •S S,0] is inserted in the fact base. Now there is no other choice and the first cycle is finished.

(vii) Termination

This basic cycle is repeated until no other new derivations are available.

(3) Parsing final result

The final result is presented in Table 5.1.

Table 5.1: TAB51 Progressive construction of sub-lists for $L_1 = \{ w = (ab)^n, n>=1\}$.

	sub-list 0	*sub-list 1*	*sub-list 2*	*sub-list 3*	*sub-list 4*	*sub-list 5*	*sub-list 6*
Sentence **ababab**	**I_{01}** S →•SS, 0 S →•AB, 0 A → • a, 0	**I_{11}** A → a •, 0 S →A•B, 0 B → • b,1	**I_{21}** B → b •,1 S → AB •,0 S → S •S,0 S → •AB,2 S → •SS,2 A → •a,2	**I_{31}** A → a •,2 S →A •B,2 B → •b,3	**I_{41}** B → b •,3 S →AB •,2 S →SS •,0 S →S •S,2 S →S •S,0 S →• AB,4 S → •SS,4 A → • a,4	**I_{51}** A → a •,4 S→A •B,4 B → • b,5	**I_{61}** B → b •,5 S→AB •,4 S →SS •,2 ***S →SS •,0*** S → S•S,4 S → S•S,2 S → S•S,0 S → •SS,6 S →• AB,6 A → • a, 6

Discussions and decisions

Decision: The introduced sentence ababab is accepted because in sub-list 6, we find the item S → SS •,0.

6.1.2 EXINF third layer

The issue is to automatically classify any unknown sentence using *EXINF* as "intelligent" parser. This phase is not treated here since it relies on the learning module *ILSGInf*.

6.2 Problem 2 : context-free language (CFL)

6.2.1 *EXINF* 2nd layer

Use *EXINF* 2nd layer in order to parse a CFL of the form:

$$L_2 = \{ w = a^n b^n, n>=1\}$$

A possible grammar for L_2 is:

$G_2 = (N_2, \Sigma_2, S, P_2)$

$\Sigma_1 = \{a, b \}$

$N_2 = \{A, B, C, S\}$

$P_2 = \{ A \rightarrow a$

$B \rightarrow b$

$S \rightarrow AB$

$C \rightarrow AS$

$S \rightarrow CB$

$\}$

(1) Filling the fact base

EXINF stores this grammar as facts as explained in Figure 5.12 below.

/* Application 5.4 */
/* APPL54 */

/ Fact Base for Tested Example 2*/*

/ Production rules stored as facts */*

```
FACT  RULE A  a                     // Fact1 //
FACT  RULE B  b                     // Fact2 //
FACT  RULE S  A  B                  // Fact3 //
FACT  RULE S  C  B                  // Fact4 //
FACT  RULE C  A  S                  // Fact5 //
FACT  initial_symbol S              // Fact6 //
```

/ Sentence to be parsed and its length */*

```
FACT  string   aaabbb              // Fact7 //
FACT  length   6                   // Fact8 //
```

Figure 5.12 APPL54 Fact base for the CFL $L_2 = \{ w = (a^n b^n, n>=1\}$

EXINF represents each production rule in the grammar as a fact (Fact1, 2, 3, 4, 5, 6). The sentence to be parsed and its length are also introduced in the fact base (Fact7, 8).

(2) Inference Cycles

As in Problem 1 above

(3) Parsing final result

Table 5.2 describes the final result

Table 5.2: TAB52 Progressive construction of sub-lists for L_2 = { w = ($a^n b^n$, n>=1}

	sub-list 0	*sub-list 1*	*sub-list 2*	*sub-list 3*	*sub-list 4*	*sub-list 5*	*sub-list 6*
Sentence **aaabbb**	**I_0** S →•CB, 0 S →•AB, 0 C →•AS, 0 A → • a, 0	**I_1** A → a •, 0 S →A•B, 0 C →A•S, 0 B → • b,1 S → •AB,1 S → •CB,1 A → •a,1 C → •AS,1	**I_2** A → a•,1 S → A•B,1 C → A•S,1 B → •b,2 S → •AB,2 S → •CB,2 A → •a,2 C → •AS,2	**I_3** A → a •,2 S →A •B,2 C → A•S,2 B → •b,3 S → •AB,3 S → •CB,3 A → •a,3 C → •AS,3	**I_4** B → b •,3 S →AB •,2 C → AS•,1 S →C•B,1 B → •b,4	**I_5** B → b •,4 S→ CB •,1 C → AS •,0 S → C •B,0 B → •b,5	**I_6** B → b •,5 ***S→ CB •,0***

Discussions and decisions

Decision: The introduced sentence aaabbb is accepted because in sub-list 6, we find the item S →CB •,0.

6.2.2 *EXINF* with counter example

Let's consider the same language L_2 as above but with a counter example of the form aabbb.

(1) Fact Base

The fact base is described in Figure 5.13 below:

/* Application 5.5 */
/* APPL55 */
/ Fact Base for Tested Counter Example 1*/*

/ Production rules stored as facts */*

```
FACT  RULE A  a                       // Fact1 //
FACT  RULE B  b                       // Fact2 //
FACT  RULE S  A  B                    // Fact3 //
FACT  RULE S  C  B                    // Fact4 //
FACT  RULE C  A  S                    // Fact5 //
FACT  initial_symbol S                // Fact6 //
```

/ Sentence to be parsed and its length */*

```
FACT  string   aabbb                  // Fact7 //
FACT  length  5                       // Fact8 //
```

Figure 5.13 APPL55 Fact base for the CFL language L_2 with counter example

(2) Inference cycles

As in Problem 1 above

(3) Parsing final result

Table 5.3: TAB53 Construction of sub-lists for language L_2 with counter example

	sub-list 0	*sub-list 1*	*sub-list 2*	*sub-list 3*	*sub-list 4*	*sub-list 5*
Sentence **aabbb**	I_0 S →•CB, 0 S →•AB, 0 C →•AS, 0 A → • a, 0	I_1 A → a •, 0 S →A•B, 0 C →A•S, 0 B → • b,1 S → •AB,1 S → •CB,1 A → •a,1 C → •AS,1	I_2 A → a•,1 S → A•B,1 C → A•S,1 B → •b,2 S → •AB,2 S → •CB,2 A → •a,2 C → •AS,2	I_3 B → b•,2 S →AB •,1 C → AS•,0 S → C•B,0 B → •b,3	I_4 B → b •,3 S →CB•,0	I_5 empty

Discussions and decisions

Decision: The introduced sentence aabbb is NOT accepted because sub-list 5 is empty.

6.2.3 *EXINF* third layer for CFL

As for the regular case, the issue relies on the learning module *ILSGInf* and is treated in Chapter 6. The processes described above remain exactly the same, but when using *ILSGInf*, the grammar is not introduced by the user but automatically generated by *ILSGInf*.

7. Conclusion

We have described the design, development and test of a rule-based deductive system, called *EXINF* and its coupling with a learning module capable of helping in grammatical inference. Although the developed system can be used as a general-purpose first-order logic programming environment, implementing both forward chaining and backward chaining, its main use here is in parsing. In this regard, at the most basic or “crude” level, it can parse sentences of a given language. But its most important aspect is that it is used as an “intelligent” parser *i.e.* as a grammar constructor in conjunction with the learning module *ILSGInf.* Advanced integration of first-order logic (FOL) and grammar inference (GI) represents an early step towards truly intelligent parsers. In Chapter 6, we describe *ILSGInf* as a useful contribution towards this distant end.

CHAPTER 6 *ILSGInf*

AN INDUCTIVE LEARNING SYSTEM FOR GRAMMATICAL INFERENCE[11]

1. Introduction

In Chapter 4, we described the building blocks of a grammatical inference system or the so-called *GASRIA* system. These building blocks mainly involve an FOL-based system, *EXINF*, used for parsing, coupled with an inductive learning system for grammatical inference, called *ILSGInf*. Both systems collaborate with each other. While Chapter 5 described *EXINF* in detail, this chapter describes the learning solution provided by *ILSGInf*. Here, we are concerned with the learning aspect in the proposed GI system. As an in-depth description of the work presented in the previous chapters, principally Chapter 4, we now discuss the details of how *GASRIA* operates through its learning module *ILSGInf*, ending up with an induced grammar from positive examples.

[11] Part of this chapter has been published under the title "*ILSGInf* : Inductive learning system for grammatical inference" In *WSEAS Trans. on Comp.*, ISSN: 1991-8755, 6(6):991-996, July 2007, http://www.wseas.org

Some machine learning systems attempt to eliminate the need for human intuition in the analysis of the data, while others adopt a collaborative approach between human and machine; this latter is what interests us in this chapter. This is so, because human intuition cannot be entirely eliminated since the designer of the system must specify how the data is to be represented and what mechanisms will be used to search for a characterization of the data. This aspect of machine learning can be viewed as an attempt to automate parts of the scientific method.

The chapter is structured as follows. The problem is formulated in Section 2 while Section 3 deals with some related works. The proposed solution is described in Section 4, and implemented in Section 5. Our solution is based on the novel partial parsing algorithm (PPA) and its implementation. Tested examples are treated in Section 6. The chapter ends with a conclusion reporting the main advantages of the method with possible future extensions.

2. Related works

2.1 ML and human interaction

Broadly speaking, machine learning (ML) is a field that attempts to develop algorithms that not only helps in taking the proper action at the actual step but also in improving future actions. In addition, it is true that many efforts were also provided with an aim to bring closer machine learning methods and grammars [CK03], or to integrate these last two topics within expert systems framework. In spite of the panoply of methods which exist in the attempt to mimic human knowledge by the machine [Lar02] and to integrate learning and reasoning [KR97], or to theorize the dynamics of acquisition of languages by evolution equations [KNN01], a problem still remains open. We specifically mean the automatic acquisition of the knowledge required by GI. In this attempt, our primary interest is to study GI from positive data, following [KMT00] and [Sak97].

2.2 Algorithm types

The computational analysis of machine learning algorithms and their performance is a branch of theoretical computer science known as computational learning theory. Because training sets are finite and the future is uncertain, learning theory usually does not yield absolute guarantees of the performance of algorithms. Instead, probabilistic bounds on the performance are quite common.

In addition to performance bounds, computational learning theorists study the time complexity and feasibility of learning. In computational learning theory, a computation is considered feasible if it can be done in polynomial time. There are two kinds of time complexity results. Positive results show that a certain class of functions is learnable in polynomial time; negative results show that certain classes cannot be learned. Machine learning algorithms are organized into taxonomy, based on the desired outcome of the algorithm. We report the main algorithm types.

- *Supervised learning*, in which the algorithm generates a function that maps inputs to desired outputs. One standard formulation of the supervised learning task is the classification problem: the learner is required to learn (to approximate) the behavior of a function which maps a vector $[X_1, X_2,,...X_n]$, into one of several classes by looking at several input-output examples of the function.
- *Unsupervised learning*, in which an agent which models a set of inputs has no knowledge of labeled examples because they are not available.
- *Semi-supervised learning* which combines both labeled and unlabeled examples to generate an appropriate function or classifier.
- *Reinforcement learning*, in which the algorithm learns a policy of how to act, given an observation of the world. Every action has some impact in the environment, and the environment provides feedback that guides the learning algorithm.
- *Transduction*, similar to supervised learning, but does not explicitly construct a function. Instead, it tries to predict new outputs based on training inputs, training outputs, and test inputs which are available while training.

- *Learning to learn* in which the algorithm learns its own inductive bias based on previous experience.

3. ILSGInf objectives

ILSGInf is an inductive learning system for GI based on the partial parsing algorithm (*PPA*). The main idea behind the *PPA* is to take full advantage of the syntactic structure of available sentences. It is based on Earley's algorithm but divides the sentence into sub-sentences using partial derivative (*PaDes*). Given a recognized sentence as reference, *PPA* is able to recognize part of the sentence (or sub-sentence(s)) while rejecting the other unrecognized part. Moreover, *PPA* contributes to the resolution of a difficult problem in inductive learning and allows additional search reduction in the partial derivatives space which is to equal to the length of the sentence, in the worst case.

4. ILSGInf learning solution

4.1 Basic properties

Inductive learning is a bottom-up process. The process of learning begins with specific instances and constructs a generalization. Therefore, in order to learn inductively, we parse all that is parsable in a global sentence. Like most inductive systems, *ILSGInf* receives the training instances (here through a human expert), then builds a sufficient knowledge stored in *EXINF* facts base, to infer one possible grammar. Thus, *ILSGInf* constructs a CFG capable of generating and/or recognizing all possible sentences produced by the language under consideration. As an example from the literature, the task undertaken by *SubdueGL* [Jon04] follows a somewhat similar technique and attempts to discover common structures in graphs from examples. In our case, it is useful to consider the following points, as stressed above:

- *ILSGInf* relies on a human expert who sequentially introduces chosen instances. In our actual work, we obviously suppose that the human expert acts as a cooperative teacher, *i.e.* that the teacher avoids giving, on purpose, examples that make the system wander away from the solution.
- *ILSGInf* gradually constructs a grammar that generates these examples.
- An initial grammar is generated and eventually updated until the most general grammar is obtained.
- For the validation of the learning process, our learning system relies on an inference mechanism. Thus, *ILSGInf* uses *EXINF* - a first-order general-purpose inference engine, developed as a stand-alone system.
- We take advantage of the fact that rules are written in the form $A \rightarrow BC$, or $A \rightarrow a$.

Search is undertaken in the space of rules in order to infer a grammar capable of generating these instances and eventually other similar ones.

4.2 *ILSGInf* architecture

By receiving a series of examples chosen by the expert and using the knowledge available, *ILSGInf* improves the facts *i.e.* the grammar of the language. So it builds the CFG that generates all the examples. Figure 6.1 shows its block diagram. The class diagram of *ILSGInf* is depicted in Appendix 2.

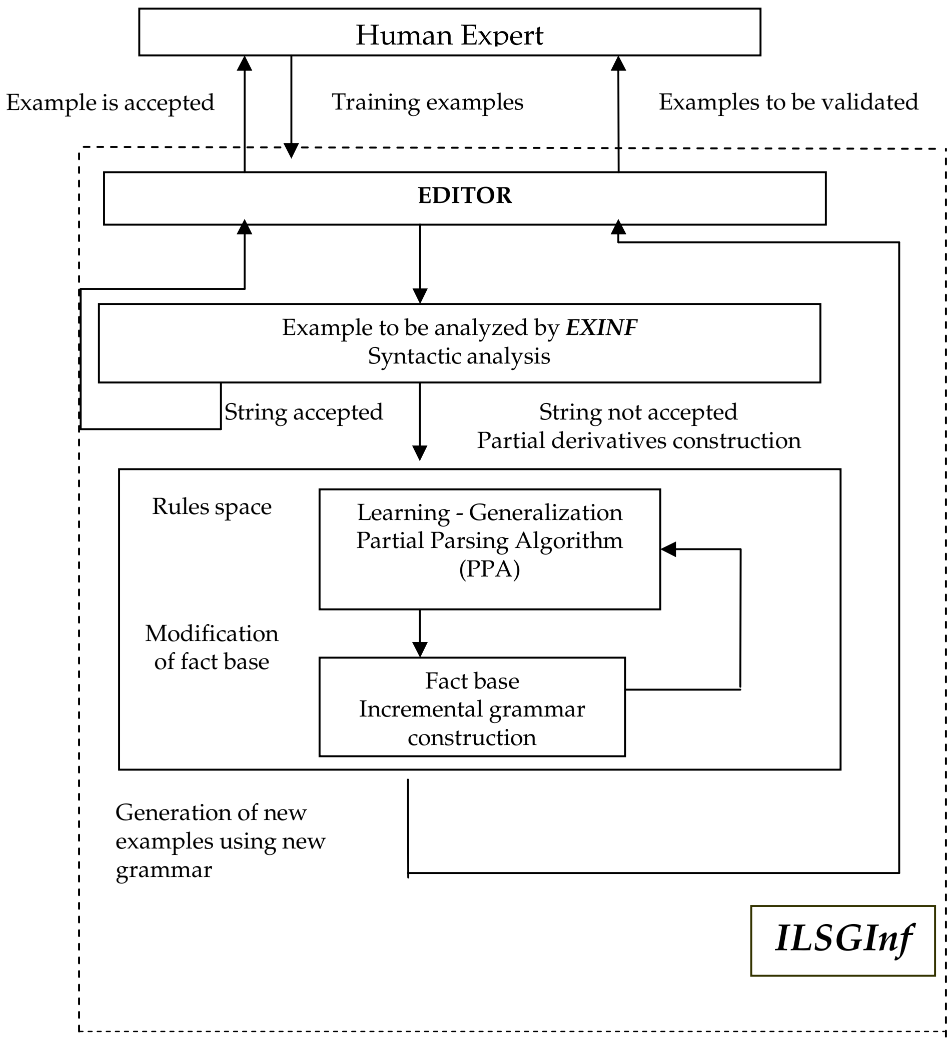


Figure 6.1 DIAG61 - *ILSGInf* block diagram

4.3 General structure of *ILSGInf* learning strategy

4.3.1 Strategy overview and complexity

At the beginning of the learning process, when no syntactical knowledge about the language is available, the system makes a direct memorization of the information provided in the form of initial grammar that is automatically generated. Then it is refined with the presentation with new sentences. Algorithm 6.1 below shows the steps involved in *ILSGInf* learning process. The time complexity of *ILSGInf* is $O(n^3)$ as shown in Appendix 3.

```
/* Algorithm 6.1 */
/* ALGO61 */

/* Learning Strategy */
BEGIN
Learning system receives a sentence

        initial  grammar construction (ALGO64)

       WHILE  system receives a sentence DO

               Refinement cycle (ALGO62)

        ENDWHILE

         Give improved grammar of the language
 END
```

Algorithm - 6.1 ALGO61 - *ILSGInf* learning strategy

4.3.2 Refinement cycle

The refinement cycle is summarized in Algorithm 6.2 below.

/* Algorithm 6.2 */
/* ALGO62 */

/ Refinement cycle */*

```
BEGIN
        /* Partial Parsing Algorithm (PPA) */

                    CALL PPA (ALGO65)

 IF sentence is positive
 AND analysis gives failure THEN

               Generalization of G (ALGO68)

 IF sentence is negative AND analysis gives success
 THEN

        Specialization of G (left as perspective)
END
```

Algorithm 6.2 - ALGO62 *ILSGInf* refinement cycle

Algorithm 6.2 describes the refinement cycle. When a given sentence (*sentence*) is received, the *PPA* is called using the current grammar (*grammar*). The result of the analysis is placed in the variable *analysis*. Two cases might occur which are:

1. First case: failure to recognize a recognizable sentence. We are then dealing with a grammar which does not recognize a correct sentence. This grammar should be *generalized* so that it can generate more sentences than currently done.
2. Second case: recognize a counter-example. This requires a *specialization* of the grammar since it recognizes more than needed.

Note that both generalization and specialization represent difficult and current problems. Here, we are only concerned with generalization, since specialization deals with counter-examples, not considered in our work. On the other hand, no counter-examples are generated by our system.

/ Algorithm 6.3 */*

/ ALGO63 */*

/ Main steps in Partial Parsing Algorithm (PPA) */*

```
FOR each recognizable sub-sentence

        PARSE sub-sentence using Earley's algorithm

      Construct the PaDe's using each parse tree ALGO67
ENDFOR
```

Algorithm 6.3 - ALGO63 Main steps in partial parsing algorithm (PPA)

4.4 Validation procedure

The grammar built is then used to generate a series of sentences that are validated by the human expert. This validation constitutes a guarantee that the integration of the new rule in the grammar does not conflict with its consistency. The system rejects the new rule as soon as the verification process detects an incorrect string. If no counter-example is generated, the grammar is considered correct. Otherwise, the level of generalization is reduced. This represents a form of specialization.

5. *ILSGInf* implementation

ILSGInf implementation is based on the requirements for obtaining partial parsing for a given global sentence. We start with the *PPA* and describe the heuristics for sorting partial derivatives (*PaDe's*) and conclude with the generalization process.

5.1 Initial grammar construction

Initial grammar is of the form : $G_0 = (N_0, \Sigma_0, P_0, S)$ where :

N_0 = {A / A non-terminal of derivative tree}

Σ_0 = {a / a is a symbol of input character string}

P_0 = {/ R rule of the form A → BC ; or A → a } with A, B, C non-terminals in derivation tree.

S = initial symbol.

/* *Algorithm 6.4* */
/* *ALGO64* */

/* *Algorithm for the construction of initial grammar* $G_0 = (N_0, \Sigma_0, P_0, S)$ */

```
Begin
string[i]                          /* table containing the string example */
n                                 /* length of initial global string */
Initial_symbol:="S"               /*creation of initial symbol, by convention "S"*/
i:=1, k:=1                        /*indices*/

                    /* Associate to each terminal one non-terminal */
                    /* create the set of  initial rules as follows */
for i=1 to n do
      if string[i] is not yet associated with a non-terminal
      then create_the_rule non-terminal(k) →  string[i]
      k:=k+1
      endif
endfor

if n<= 2             /* Derivation from S* /
        then  create_rule S → <non-terminal(1)> <non-terminal(2)>

         else      /*Construction of derivation tree from bottom to top */
         create_the_rule non-terminal(k)    →   <non-terminal(1)> <non-terminal(2)>
         i:=3; k:=k+1

    while i<n do
     create_the_rule non-terminal(k) → <non-terminal(k-1)> <non-terminal(i)>
     k:=k+1; i:=i+2
     endwhile

              /* For string to be recognized, it must derive from root * /

     create_rule S  →   <non-terminal(k-1)> <non-terminal(i)>

endif
end
```

Algorithm 6.4 - ALGO64 Algorithm for initial grammar construction

5.2 Partial parsing

The detailed steps of the partial parsing algorithm are described in Algorithm 6.4 below.

/* Algorithm 6.5 */

/* ALGO65 */

/ Partial Parsing Algorithm */*

```
FinalParse := empty     /*a global sentence to be parsed*/
i:=1               /* index for spanning the global sentence */
head := 1         /* head of a sub-sentence to be parsed */
read (car)       /* read character car  to be parsed */
while car <> end of sub-sentence do

        /* for delimiting the sub-sentence to be parsed */

  while car <> end of sub-sentence and car accepted do
    sub-sentence = sub-sentence + car
     /* generation of sub-sentence sub-sentence  */
    i:=i+1
    read(car)
  endwhile
  if car refused then

        /*  Result is complete parsing of sub-sentence */

    Earley (sub-sentence (head, i-1), result)
    Concatenate (FinalParse, result, car [refused])
    head := i+1 /*Start over with sub-sentence following refused
                character*/
    i:=1    /*Consider another sub-sentence */
   else        /* it is the end of global sentence*/
    Earley (sub-sentence (head, i-1), result)
```

```
    Concatenate (FinalParse, result, empty);
   endif
endwhile
```

Algorithm 6.5 - ALGO65 Partial parsing algorithm

5.3 Detailed refinement cycle

5.3.1 Generalization

In our context, we follow [Mug99] for defining generalization as corresponding to induction and specialization to deduction. The generalization algorithm is described in Algorithm 6.6 below.

Definition 1: A hypothesis H_G is more general than a hypothesis H_S if and only if H_G entails H_S. We also say that H_S is more specific than H_G.

Example

For search algorithms, the notion of generalization and specialization are incorporated using inductive and deductive inference rules.

Definition2: A deductive inference rule r maps a conjunction of clauses C_G onto a conjunction of clauses C_S such that C_G entails C_S; r is called a specialization rule.

Examples

Resolution is a deduction rule.

Dropping a clause from a hypothesis realizes a specialization.

Definition3: An inductive inference rule r maps a conjunction of clauses C_S onto a conjunction of clauses C_G such that C_G entails C_S; r is called a generalization rule.

Example

Absorption rule is an inductive inference rule. In the absorption rule the conclusion entails the condition. Note that applying the absorption rule in the reverse direction, *i.e.* applying resolution, is a deduction rule.

/* Algorithm 6.6 */

/* ALGO66 Generalization */

For each sub-sentence
- Construct the list of partial derivatives (*PaDe's*)
  - **sort** these *PaDe's* by increasing order of generality
  - **choose** as hypothesis the rule $S \rightarrow D_g$, where S is the initial symbol and D_g the most general concatenation of all sub-sentences
  - **add** this rule to the set P of current grammar rules
  - **use** this grammar to generate a set of sentences called *test sentences*
  - **if** this generated set is accepted by a human expert
    then accept this new grammar
  - **else** start again with rule $S \rightarrow D_g'$, where D_g' is less general than D_g
  - **if** no *PaDe* has allowed acceptance of this generated set
    then consider it in the same way as an initial grammar

Algorithm 6.6 - ALGO66 Generalization

5.3.2 Partial derivatives (PaDe's) construction

The basics of partial derivatives (*PaDe's*) have been treated previously. Construction of the *PaDe's* for a given string reduces this latter. Thus, it replaces the parsed parts by the corresponding non-terminals. The steps of the construction of a *PaDe* are described in Algorithm 6.7 below.

/* Algorithm 6.7 */

/* ALGO67 PaDe's construction */

/ This technique is based on the use of lists produced by the syntactic analyzer */*

- **if** partial parsing algorithm (*PPA*) generates k sets of sub-lists
  t**hen** we have k sub-strings(s) analyzed separately
- each sub-string of length m is analyzed by a sub-list $I_0...I_m$
- in each sub-string, we have:

```
• the list I0 is always present
• if I1 is empty
 then symbol a1 of sub-string is not recognized, therefore, the
      length of sub-string is equal to 1.
• if the sub-list contains at least  I0 and I1,
  then we have found part of the string that is recognized and
       which contains at least one symbol.
```

Algorithm 6.7 - ALGO67 PaDe's construction

5.3.3 One *PaDe* construction for a sub-sentence

For each given sub-string, we need the construction of a *PaDe*. We proceed using Algorithm 6.8 as follows:

/ Algorithm 6.8 */*

/ ALGO68 PaDe's construction for a given sentence */*

```
For each sub-list do
 if I1 is empty, then the character is not recognized and no
       transformation is needed.
 if Ik exists for a sub-string of length k,
       and if item "S →α•,0" is in it,
       then sub-string is totally recognized and transformed into S.
 if Imax for the string of length k exists,
       and if 0 <max <=k then we proceed as follows :
       for  jmax to  j = 1
       Consider the items of the form  "A→α•,i" for increasing i
       Treat these items starting from the most specific α to the most
       general
```

Algorithm 6.8 - ALGO68 PaDe's construction for a given sub-sentence

5.3.4 Heuristics for sorting *PaDe's*

There are two levels when sorting PaDe's, as explained in Algorithm 6.9 below.

/* Algorithm 6.9 */

/* ALGO69 Heuristics for PaDe's sorting */

/ Level 1 local sorting */*

```
for all sub-sentences
order in a decreasing fashion of generality all PaDe's
```

/ Level 2 global sorting */*

```
order in increasing fashion of length all sub-sentences of global sentence
```

/ Heuristics search for the adequate rule */*

```
initially choose rule whose RHS is the concatenation of the most  general
   PaDe's for all sub-sentences produced in Level l above
   test this new grammar by generating new sentences
   if all generated sentences are accepted
        then new rule is accepted
        else modify RHS of the rule by considering the following PaDe of
             the following sub-sentence
```

Algorithm 6.9 - ALGO69 Heuristics for PaDe's sorting

6. Tested example

6.1 *PPA* use

Given the following CFG: G = (N, Σ, P, S), where :

N = {S, A, B}, Σ ={a, +}, P = {S → A B, A → a, B→+ A}

Let w= (a+a)+(a+a) be a global sentence to be parsed. The sub-sentences are:

C_1 = (, C_2 = a + a, C_3 =), C_4 = +, C_5 = (, C_6 = a + a, C_7=)

Our partial parsing algorithm gives the following results of sub-lists and sub-sentences:

Table 6.1 TAB61 Progressive construction of sub-lists

	sub-list 0	*sub-list 1*	*sub-list 2*	*sub-list 3*
sub-sentence 1	I_{01} S → •AB, 0 A → • a, 0	I_{11} empty	I_{21} empty	I_{31} empty
sub-sentence 2	I_{02} S →• AB, 0 A → • a, 0	I_{12} A → a • , 0 S →A•B , 0 B →• +A, 1	I_{22} B →+•A, 1 A → • a , 2	I_{32} A → a • , 2 B →+A•, 1 S →AB•, 0
sub-sentence 3	I_{03} S →•AB, 0 A → • a , 0	I_{13} empty	I_{23} empty	I_{33} empty
sub-sentence 4	I_{04} S →•AB, 0 A → • a , 0	I_{14} empty	I_{24} empty	I_{34} empty
sub-sentence 5	I_{05} S →•AB, 0 A → • a , 0	I_{15} empty	I_{25} empty	I_{35} empty
sub-sentence 6	I_{06} S →• AB, 0 A → • a, 0	I_{16} A → a • , 0 S →A•B , 0 B→•+A,1	I_{26} B→ +•A, 1 A→ • a , 2	I_{36} A → a • , 2 B →+A•, 1 S →AB•, 0
sub-sentence 7	I_{07} S →•AB, 0 A→ • a , 0	I_{17} empty	I_{27} empty	I_{37} empty

6.2 Discussions

For the sub-sentences 1, 3, 4, 5 and 7, we note that:

(i) I_{1x} (x=1,3,4,5,7) is empty. In this case, while no classical algorithm (*eg* Earley-like) proceeds further, the *PPA* looks for other *PaDe's*. Because sub-sentences are refused, then no transformation is needed.

(ii) In sub-sentences 2, 6 all I_{3x} (x=2,6) are accepted. In each of these, we find an item of the form "S→α•,0" which is "S→AB•,0". Then respective sub-sentences are totally accepted and transformed as S.

(iii) PaDe's of the global sentence "(a+a)+(a+a)" have the form : "D = (S)+(S)" Other PaDe's of "a+a" are :

a+A from item $A \rightarrow a\bullet,2$ in I_{3x}, (x=2,6)

aB from item $B \rightarrow +A\bullet,1$ in I_{3x}, (x=2,6)

A+a from item $A \rightarrow a\bullet,0$ in I_{1x} (x = 2,6)

AB from item $A \rightarrow a\bullet,0$ in I_{1x} and I_{3x}, (x=2,6)

(iv) Local sorting is done as follows: S, AB, aB, a+A, A+a.

7. Conclusion

We have designed, developed and tested an inductive system for grammar inference. The central idea is the so-called *partial parsing algorithm* (PPA) that can parse sentences not parsed by traditional methods. Comparatively, inductive logic programming (ILP) requires a prohibitive number of hypotheses to construct a grammar. Our method suggests a drastic reduction in the number of relevant hypotheses to be considered while inferring a grammar. Moreover, in our approach, at each step, the system takes advantage of the syntactic knowledge contained in the global sentence. In this way, the system avoids the construction of redundant rules and thus improves the quality of the inferred grammar. In this regard, our implemented and tested system addresses a difficult issue while proposing a real application with tangible results.

CHAPTER 7

GASRIA/ILSGInf INTERACTIONS WITH SYSTEMS CONTOL[12]

1. Introduction

In this chapter, we report a framework for inductive learning as used in two different fields of applications, very far away from formal languages, namely control of machine drives and robotic self-assembly. We present an alternative method for tackling the control problem using GI, instead of control law generation using traditional state-space methods such as state-feedback or adaptive control methods, for instance. We fully describe one example issued from the first field and give the methodological steps for solving inference problems for the other field. We rely on graph grammars for robotic

[12] - Part of this chapter has been published under the title "Grammatical inference for robotic self assembly – basic methodology", Invited conference paper In: *Recent Advances in Artificial Intelligence, Knowledge Engineering and Database (AIKED'09)*", Cambridge, UK, February 21-26, 2009, pp. 447-452, ISBN: 978-960-474-051-2, ISSN: 1790-5109, http://www.worldses.org/online/2009.htm, http://portal.acm.org/citation.cfm?id=1554004
- Above article extended under the title "Grammatical inference methodology for control systems", *WSEAS Trans. on Comp.*, ISSN: 1991-8755, 8(4):610-619, April 2009, http://www.wseas.us/e-library/transactions/computers/2009/29-113.pdf http://portal.acm.org/citation.cfm?id=1558760

self-assembly applications. We further propose a four-level methodology for addressing the issue of GI-based control and self-assembly ending with graph grammatical inference.

The Chapter is organized as follows. In Section 2, the issue of controlling a physical system, namely machine drives, is addressed with concentration on the integration of GI within the control loop. Section 3 discusses the self-assembly issue. Section 4 describes the methodological steps to follow in order to solve the GI-based control problem and robotic self-assembly problem using graph grammars, as an ultimate result of the actual work.

2. *ILSGInf* and control systems interaction

2.1 The basic control methodology

Before considering tackling self-assembly issues using graph grammars, we describe a simple control problem related to machine drives. For that, we need an introductory account of control systems and their interplay with grammars.

2.1.1 Negative feedback dynamic control

Control is an interdisciplinary branch of engineering and mathematics, which deals with the behavior of dynamical systems. The desired output of a system is taken as a reference to be attained or maintained at a specific value. When one or more output variables of a system need to follow a certain reference over time, a controller generates the control law (or strategy) necessary to obtain the desired effect on the output of the system. This is usually done using negative feedback, *i.e.* a procedure whereby the actual value is subtracted from the desired value to create the error signal which is amplified by the controller to allow correction to be undertaken at subsequent stages. This procedure is therefore done in closed-loop form.

A thermostat is a simple example for a closed-loop negative feedback control system: it constantly measures the actual temperature and controls the heater's valve setting to increase or decrease the room temperature according to the user-defined setpoint. A simple method, called control law or control strategy, switches the heater either completely ON, or completely OFF, and an overshoot or undershoot of the controlled temperature is to be expected, dictated by the physical inertia of the system. A more expensive method varies the amount of heat provided by the heater depending on the difference between the required temperature, or setpoint and the actual temperature. This minimizes over/undershoots.

An anti-lock braking system (ABS) used in vehicle braking technology is a more complex example, consisting of multiple inputs, conditions and outputs. The aim of the system is to avoid the brakes from locking irrespective of the external conditions such as speed of the vehicle, weather conditions, road surface, among others.

2.1.2 Control laws construction

Whatever control strategy is used, the resulted control system must first guarantee the stability of the closed-loop behavior, *i.e.* preventing that the system state or output take unacceptable values. For linear systems, this can be obtained by directly placing the poles of the closed-loop transfer function. For multiple-input multiple output (MIMO) systems, pole placement can be performed mathematically using a state space representation of the open-loop system and calculating a feedback matrix assigning poles in desired location of the s-plane for continuous systems or the or z-plane for discrete systems. This is usually done by computer aided control systems design (CACSD) methods and tools and capabilities [Ham94].

Whatever methods are used for linear systems, one cannot always ensure robustness, *i.e.* the ability in coping with small differences between the true system and the nominal model used for design. Furthermore, all system states cannot in general be measured and so estimators must be included and incorporated in pole placement design. The estimators are either observers of Luenberger type for deterministic control or Kalman filters for stochastic control.

2.2 Motivations for grammatical control approach

By grammatical control, we mean the use of GI to, either generate the control law or to detect faulty operating conditions through the detection of abnormal input-output pairs. GI as applied to control systems at large is relatively a new area of research. As an indication, a rapid search in *IEEE* site (http://www.ieee.org) using *ieeexplore* search engine and keywords (formal language control + dynamical systems + grammatical inference) hits one journal paper [MDP01] and two conferences papers. Subsequent efforts remain quite isolated, [HH09a], [HH09b], [CKR10].

Any grammar codes for the class of all possible syntactical patterns that belong to the language produced by the grammar. The basic idea is to design a parser (or classifier) that recognizes strings accepted by the grammar. There is a mapping signals-to-strings. Each signal is quantized and each value is given a terminal symbol. Under normal operations, signals are compatible with the grammar. Once the grammar is learnt, it is used as a reference by the nominal system. If at a later time, there is some faulty output from the dynamical system then the faulty generated signals are translated as "odd" strings, reporting abnormal behavior resulting in anomaly detection. An input of non-terminals is used for both the nominal and actual dynamical systems. An error is evaluated between the strings generated by both systems. Two modes are possible. In the open-loop mode, the grammar generates the working patterns imposed by the external input command. If this error exceeds some threshold, a fault is reported. A closed-loop control is used when the control U is generated for an output y to be within some prescribed values [Ham10]. The basic procedure is described in Figure 7.1 below.

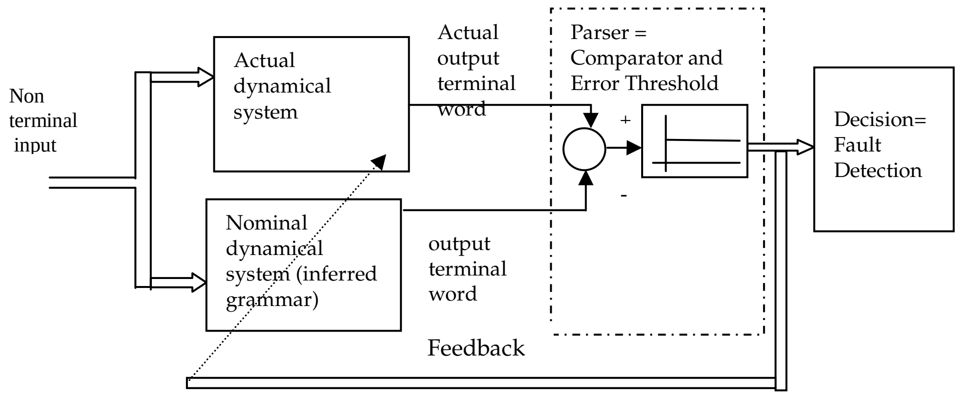


Figure 7.1 DIAG71 - Grammatical control used in open-loop/closed-loop modes

As exposed in Chapter 6, *ILSGInf* classifies negative examples correctly (*i.e.* as negative) but does not take them into account for improving the grammar it generates. In other words, the positive examples help *ILSGInf* in improving the generated grammar, but the negative ones do not contribute to this improvement. Now, we discuss the application of GI to a context-free language (CFL) as a prelude to a grammatical-based control. We must notice that, although the control system under consideration is simple, it requires a context-sensitive grammar inference. This is obviously outside the scope of *ILSGInf*. Therefore, we need additional knowledge in the form of *p*-production as explained below.

2.3 Using grammars to control machine drives

Before discussing self-assembly, we describe the interaction between a simpler control problem and GI, namely the control of machine drives. Control of machine drives is a specialized subject in its own right, usually studied within traditional disciplines such as electrical and / or mechanical / industrial engineering. Based on mathematical models, this subject encompasses a tremendous body of knowledge since the early days of cybernetics going back to the late 1940's. To dynamically control a machine drive is to let it follow an imposed behavior, automatically calculated in real-time. The main methodology of dynamic control is therefore to produce the so-called prescribed feedback control law on the basis of output observations, as and when needed. If the environment is unknown, we use adaptive control. For the purpose of this specific application, we are only concerned with control, using grammars as a methodology. So far in this thesis, by GI, we intended only deterministic finite automata FA, equivalent to regular grammars, on the one hand and some context-free grammars (CFGs), on the other hand. If we refer to Chomsky hierarchy, only type-3 and subclasses of type-2 grammars, respectively, are concerned; as described in Chapter 2. Now, in order to control drives, these classes of grammars are not sufficient. We need to include larger classes of grammars such as context-sensitive grammars or type-1. This is a real challenge since there remain many obstacles in inferring DFAs, let alone context-sensitive grammars. Because of the difficulty in handling this type of problems, supplementary human-supplied expert codification is needed in order to account for this kind of induction.

2.4 Steps for using GI in control systems

To develop a grammatical description and a GI algorithm for controlled dynamical systems three steps are required [MDP01].

2.4.1 Quantification of the variables

Quantification refers to the creation of alphabets for the output (controlled) variable y and the control variable U. The objective is to generate the control U in order to maintain the output y within some prescribed values. A terminal alphabet Σ is associated to the output variable y and the nonterminal alphabet N to the control variable U. The feedback control law generates the required value of the input U so as to keep the output y within a specified range. For so doing, a quantification of the variables is made, in a discrete way, dividing the variables range into equal intervals and associating each interval to a symbol in the alphabet.

2.4.2 Production rules

p-type productions are defined by the human expert to be some substitution rules of a given form. This human-supplied codification is necessary. A p-type production codes the evolution of the output variable, depending on its p past values and on the value of the control variable U. There is, therefore, a functional relationship between the dynamics of the system and the p-type productions. Note that p-type productions as described here are not the Proportional-control or P-control action.

2.4.3 Learning

A learning algorithm is necessary to extract the productions from the experimental data. To obtain a sample of the language, a sequence of control signals is applied to the system in such a way that the output variable y takes values in a sufficiently wide region. The signal evolution is then quantified as described above, and a learning procedure is followed.

2.5 *EXINF/ILSGInf* in control of machine drives

Since we are at the beginning of the applied work, results mainly concern the applicability of GI to machine drives as an introductory application of GI-based control methodology.

In GI control systems, GI is used as an algorithm through which a grammar is inferred from a set of sample words produced by the dynamical system considered as the linguistic source. Therefore in order to apply GI, a dynamical system must be considered as a linguistic source capable of generating a specific language. The set of productions encodes the dynamics of the system that generates the language. Any word that can be derived from the start symbol S followed by a sequence of productions of the grammar is said to be within the language generated by the dynamical system.

/* Methodology 7.1 */

/* METH71 */

/* GI Control systems drive */

```
1. Pre-processing phase
   1.1 Quantification of variables
   1.2 Production rules
        Call first level of EXINF (see Fig. 5.9)
        / * instead of manually-introduced expertise * /
2. Learning using GI
        Call ILSGInf
        /* Third level of EXINF is implicitly used */
```

Figure 7.2 DIAG 71 - Adapted GI control system methodology

From quantification, we derive the alphabet of the language. The operation of the drive system gives the words that are classified by the human expert as correct, for the case of positive examples only. Based on these elements, *ILSGInf,* with the help of a knowledge base in *EXINF*, as described in Chapter 6, automatically generates the grammar from the given examples.

2.6 Comparing GI-controlled systems with other methods

A useful methodological comparison can be made between grammatical methods and other methods such as observer-based methods of control and soft computing, *e.g.* fuzzy control [Hag07].

3. Self-assembly issue

3.1 Self-assembly as a process

In addition to the use of GI in machine drives, GI can be used in self-assembly. Self-assembly is the process in which a disordered system of preexisting shapes or components forms an organized structure or pattern as a consequence of specific, local interactions among the components themselves, without external direction. It is a phenomenon in which a collection of particles spontaneously arrange themselves into a coherent structure. In nature, self-assembly is ubiquitous. For example, cell membranes, and tissues are self-assembled from smaller components in a decentralized fashion. It is common to encounter, in the natural world members of decentralized systems that self-organize in response to environmental stimuli and to each other to produce complex global behaviors. This is referred to as flocking. Birds and bacteria group behavior are among the most common examples. Flocking has been used as a metaphor for the study and development of artificial swarm intelligence-based systems. Self-assembly, as a facet of flocking is beginning to find its way into science and engineering, through various disciplines ranging from molecular application encountered in bioinformatics [Win00], to robot reconfiguration, and stochastic self-assembly, among others. Assembling geometrical shapes into whatever desired shape is still considered as a challenging control problem. Assembling shapes into a given pattern can be seen as a *language* where the individual shapes are the *words* and the obtained pattern correspond to a *sentence* obeying some specific rules or *grammar* for generating grammatically correct sentences. The process of self-assembly can therefore be seen as the automatic generation of a language. One of the central questions for robotic self-organized systems is to know whether it is possible to synthesize a set of local controllers that produce a prescribed global behavior that is sufficiently robust to uncertainties about the environmental conditions. Since assembling geometrical shapes into some desired shape can be viewed as a set of sentences of a language, it is therefore not surprising to address this issue from the standpoint of grammars. More precisely, we propose to make use of GI. Ultimately, graph grammars are considered as an emerging field that is believed promising [Kla07].

3.2 Modes of self-assembly

Self-assembly, as defined above, comes in two modes, passive and active. In passive self-assembly, particles interact according to their geometry or surface chemistry and stay in a thermodynamic equilibrium, once this steady-state is reached. Particles behavior in chemical reactions can be classified in this mode. The geometrical patterns in the natural world give a clear indication that self-organized systems are omnipresent, from leaves to snowflakes, all governed by emergence of global patterns based on smaller patterns or fractals. In active self-assembly, each particle may use energy to accept some interactions with other particles while rejecting others, according to a controlling program. Typical examples are multi-robot systems, where small groups of robots determine the outcome of encounters according to their internal programming [Kla07]. In our work, we focus on this latter mode of self-assembly.

3.3 Self-assembly central issue

As stressed above, the main question in programmed self-organization concerns the ability to design rules that govern the global behavior of a system by means of local rules. In a wide variety of settings, we can design local rules that yield a specified behavior, with the ability to reason about the correctness of the result. In some circumstances, we can provide algorithms that automatically generate such a set of rules. Recent results are obtained in diverse areas ranging from algorithmic self-assembly of DNA [Win00], to the formation stabilization of multiple agents using decentralized navigation functions [TK05]. These results indicate that the emergent behavior of a self-organizing system can be precisely predicted and controlled, although there is much work to be done to understand the physics, dynamics, and implementation of self-organization. Progress in this area promises to open up new vistas for a completely new era of bottom-up engineering of systems, ranging from programmable nano-scale molecular machines to controlled swarms of interacting autonomous robots [KGL06].

3.4 Graph grammars

3.4.1 Definition of graph grammar

Graphical structures of various kinds, like graphs, diagrams, visual sentences are very useful to describe complex structures and systems in a direct and intuitive way. *Graph grammars* have been invented in the early seventies in order to generalize Chomsky's (string) grammars. This generalization consists in gluing graphs instead of concatenating strings. Graph grammars are evolving graphs from some starting graph, and whose evolution follows specified production rules.

A *graph* is a pair (V, E) where:

- V is a finite set called *vertices*
- E is a finite set with elements in $V \times V$, called *edges*.

A *graph grammar* is a pair (Gr_0, P) where:

- Gr_0 is called the *starting graph*
- P is a *set of production rules*

Similarly to a language generated by string grammars, a language generated by a graph grammar is the *set of graphs* that can be derived from the starting graph and applying rules in P. Mathematical accounts of graph grammars are based on algebraic representation [Ehr79].

3.4.2 Application of graph grammars in self-assembly

From the point of view of graphical programming languages, graph grammars are useful especially in the storage level. Thus, instead of storing all these graphical structures as individual objects, we store only their grammar for reasons of compact size and generative power. While earlier mathematical work focused on string grammars, more interest is recently based on tree and graph grammars [Hof00]. In self-assembly applications, graph grammars are used to model the physics of the particles by describing the outcomes of interactions among them. When used to program the desirable outcomes of interactions among particles, a graph grammar represents a description of a communication protocol and is thus intended to be coupled with a physical model of the environment that mediates the interactions. In particular, a suitably designed graph grammar can precisely describe and direct the changing network topology of a self-organizing system [MKE07].

4. From string GI to graph GI

4.1 Four methodological levels for solution

We propose here a set of steps we believe can handle the issue of GI-based control starting from string grammars to graph grammars.

1. ***Level 1****:* Extension of known techniques used in GI to graph grammars
    - 1.1 State of the art in GI for regular languages and CFLs
    - 1.2 Concentration on on structural methods such as tree and graph grammars
    - 1.3 Graph grammars and their algebra
    - 1.4 Investigation of the use of inference in graph grammars
2. ***Level 2:*** Formal languages for systems control

The main issue here is to consider how formal languages can help in developing novel techniques in system control. It can be structured as follows:

- 2.1 Current methods for system control based on formal languages
- 2.2 Control methods based on (string) grammar inference
    - 2.2.1 Extend and apply *ILSGInf-EXINF* to control drives
    - 2.2.2 Extend *ILSGInf-EXINF* application to robot control

Level 3: **Robotics self-assembly and graph grammars**

The main issue here is to study the phenomenon of self-organizing systems and robotics self-assembly using graph grammars. It is structured as follows:

- 3.1 Graph grammars for robotic self-assembly
- 3.2 Inference in graph grammars for robotics self-assembly

Level 4: **GI-based control *vs.* other control methods**

- 4.1 GI-based *vs.* state-feedback control methods (*e.g.* observer-based)
- 4.2 GI-based *vs.* soft computing-based control (*e.g.* neural nets and genetic-based)

4.3 Recommendations and feasibility study

5 Conclusion

The present chapter paves the way towards an objective evaluation and an introductory study of the effectiveness and usefulness of GI as applied in control systems settings. It represents an early contribution as far as graph grammars inference integration is concerned. A unification of the diversified works dealing with robotic self-assembly while concentrating on graph grammars as an alternative control method is made possible. This is done using an incremental methodology for control and self-assembly, starting with string grammatical inference and ultimately leading to inference in graph grammars. However, the results report only a tiny aspect of the overall issue, since these describe only the case of context-free language (CFL) inference as (an incomplete) part of the control of machine drives. Much work is still required on both sides, *i.e.* control and formal languages, for the development of fully-integrated systems that scale up to real-life applications that use context-sensitive grammars.

CONCLUSION

1. First-order logic (FOL) and grammatical inference (GI)

In this research, we investigated an early attempt in bridging the gap between inferences as produced by first-order logic (FOL) and machine learning processes as undertaken by grammatical inference (GI). The aim is programming languages improvement with a learning layer. For the purpose of integrating the inferential or declarative approach, as exemplified by *Prolog*-like logic programming, with machine learning methods such as those used in GI, we have designed, fully implemented and tested various algorithms. Specifically, we studied, from design to testing and debugging, an inductive learning environment *ILSGInf* supported by, and coupled with a rule-based deductive reasoning environment, called *EXINF*. The result of this integration is the so-called *GASRIA* system that has been designed and developed as a GI system for the induction of some CFG's from positive examples using heuristics. Thus, the proposed system behaves as a parser with the ability to learn a grammar by induction, supported by the learning environment *ILSGInf*, and reasoning through *EXINF*, a FOL-based programming environment. As a result, *GASRIA* takes a set of sentences from a human teacher and generates a grammar from it. The overall system has been successfully applied to various artificial formal languages ending with a class of context-free languages (CFLs).

2. Inferences and "intelligent" parsing

Parsing according to a specified grammar is a field of many practical applications. Both programming and natural languages parsing represent the most obvious examples. One of the major characteristics of grammars is that they have the ability to generalize over a specific language. This characteristic is very useful, since it offers the possibility to learn a grammar based on a set of sample sentences without the need to specify every sentence of a language. This is accomplished by all machine learning algorithms since they seek to generalize over a set of examples in order to obtain a more general model.

In our case, the general model or inferred grammar is obtained using two environments; one deductive and the other inductive. Although the deductive environment *EXINF* can be used as a general-purpose FOL programming environment, implementing both forward chaining and backward chaining, its main use here is in parsing. In this regard, at the most basic or "crude" level, *EXINF* can parse sentences of a given language. But its most important role is that it is used as an "intelligent" parser *i.e.* as a grammar constructor in conjunction with the inductive environment *ILSGInf.* Further integration of FOL and GI represents an important step towards truly intelligent parsers. Chapter 6 described *ILSGInf*, a useful contribution towards this distant end.

3. Partial parsing algorithm

In our parsing approach, the central idea is the so-called *partial parsing algorithm* (PPA). In this work, the *PPA* contributes to infer a CFG and is capable of parsing sentences that, in our learning settings, are not parsable by existing methods. This is done through the use of partial derivatives, representing the different items that can be isolated in the derivation tree of the sentence under analysis. The *PPA*, which is designed and described in detail, is validated using a set of experiments.

4. Performance criteria

In evaluating results of this kind, we can rely on criteria that are traditionally considered important.

- How efficient and incremental the method/system is?
- How precisely and naturally its generalization process is, after the introduction of any additional example.
- How well it obtains correct identification in the limit.
- How natural and useful the inferred grammatical rules are.

As shown in the results, the answers to all these questions are satisfactory. Indeed, the developed overall system is both efficient and incremental. Our method suggests a drastic reduction in the number of relevant hypotheses needed for inferring a grammar. Moreover, in our approach, at each step, the system takes advantage of the syntactic knowledge contained in the global sentence with the help of partial derivatives. In this way, the system avoids the construction of redundant rules and thus improves the quality of the inferred grammar.

On the other hand, some methods suffer from the "curse of dimensionality". For instance, i*nductive logic programming* (ILP) requires a prohibitive number of hypotheses to construct a grammar. In our case, the tested languages required a reduced number of examples for induction, not exceeding five to six examples attesting that the generalization is realized quite rapidly with no generation of counter examples. It is shown that this leads, in polynomial time, to correct identification in the limit of the regular languages and some CFLs, as detailed in the examples treated in the text. On the other hand, the generated language is not empty since it contains at least the introduced examples. In this regard, the proposed approach successfully addresses a difficult issue. Our additional asset is the use of FOL within the declarative approach in parsing. Avenues for other applications such as control systems is also made possible.

5. GI, control and self-assembly

In addition to intelligent parsing through the integration of FOL and GI, we studied also applications that are usually considered far from formal languages, namely control systems and self-assembly. For GI-based dynamical control systems, original knowledge in the form of signal from sensors is translated into rules and facts in the form of grammar to be induced. For self-assembly systems, graph grammars are used instead, because they are more suitable to describe geometrical patterns. In both cases, we are in face of context-sensitive grammar whose inference is not possible by existing methods. We therefore need additional human expertise. In GI-based control systems, for instance, we need the humanly-supplied *p*-type productions. These have to be coded, updated and used in the inference process. Hence, the use of the declarative approach in handling this kind of knowledge. We have taken advantage of the integration of GI and FOL to contribute to the development to GI-based control systems and self-assembly, as described in Chapter 7.

6. Prospects

6.1 Parsing

Prospectively, much effort is still needed in order to address the difficult issue of intelligent parsing so as to scale up to real life applications such as development of a new type of compilers. The combination of GI and FOL can be regarded as one important step towards the design of intelligent compilers.

6.2 GI-based control and self-assembly

GI-based control is still in its infancy. For the time being, this approach does not compare well with the so-called soft computing approach, which is based on methods such as neural networks, fuzzy systems, genetic algorithms, and similar methods. However, the integration of GI and FOL can open new vistas for novel algorithms on the basis that FOL-based declarative environments are very powerful in the manipulation of knowledge and its update through inference.

7. Further… for the future

The results obtained can be taken as a good starting point for contributions towards the following directions of research:

7.1 Computer algebra system (CAS) improvement

In today's CASs, any problem (integration, differentiation, solution of algebraic equations…) is solved in the same fixed way irrespective of the number of times it solves it. A learning layer will make the system solve problems differently on the basis of previous problems.

7.2 Semantic level of programming languages

So far, we only considered the syntactic level of languages. A good line of research would be to devise methods that address the semantic level as well. GI helps us to identify hierarchical structures in programs. These structures identify not only different units but also how these units interact. Understanding how interaction between parts of a program helps in adding learning to programming, as one possible future line of research.

7.3 Grammars and bioinformatics

An interesting theme concerns the interaction between GI and gene expression in the human cell. Blending methods from control systems and GI will improve our knowledge of gene regulatory networks (GRNs) whose faulty functioning is responsible for many devastating human diseases, such as cancer, to cite but one.

How much knowledge in GI, control systems, and other computerized medical fields with their various interactions do we need in order to eradicate just one of these human diseases?

Obviously, this is another story.

This thesis extracted a very tiny drop from the vast ocean of knowledge that can hopefully help in elucidating this question – for the welfare of all…

REFERENCES

[Adl94] Adleman, L. M. "Molecular computation of solutions to combinatorial problems", *Science*, ISSN (print): 0036-8075, ISSN (online):1095-9203, 266(5187):1021-1024, 1994

[ALS07] Aho, A.V., M. S. Lam, R. Sethi, & J. D. Ullman "*Compilers: Principles, Techniques, & Tools",* 2nd Edition, Addison-Wesley, ISBN: 9780321547989, 2007

[Ang80] Angluin, D. "Inductive inference of formal languages from positive data" *Inform. and Control*, ISSN: 0019-9958, 45:117-135, 1980

[Ang81] Angluin, D. "A Note on the number of queries needed to identify regular languages", *Inform. and Control*, ISSN: 0019-9958, 51:76-87, 1981

[Ang82] Angluin, D. "Inference of reversible languages", *J. ACM,* ISSN: 0004-5411, 29(3):741-765, 1982

[Ang87] Angluin, D. "Learning *k*-bounded CFGs", *Yale Tech. Rept.* RR-557, 1987

[Ang88] Angluin, D. "Identifying languages from stochastic examples", *Technical Report YALEU/DCS/RR-614*, Yale University, March 1988

[ASV01] Amengual, J.-C., A. Sanchis, E. Vidal & J.-M. Benedi, "Language simplification trough error-correcting and grammatical inference techniques, *Machine Learning*, ISSN (print): 0885-6125, ISSN (online): 1573-0565, 44(1-2):143-159, 2001

[AS83] Angluin, D. & C. Smith, "Inductive inference: theory and methods", *ACM Computer Surveys,* ISSN (print): 0360-0300, ISSN (online): 1557-7341, 15 (3):237-269, 1983

[AV02] Adriaans, P. & M. Vervoort, "The EMILE 4.1 grammar induction toolbox", *Proc. of ICGI02*, *LNAI*, Springer, 2484: 293-295, 2002

[BA08] Bacerra, L., A. Angluin, "Learning semantics before syntax", *Proc. of ICGI08, LNAI*, pp. 1-14, Springer, Berlin, 2008

[BGB04] Benenson, Y., B. Gil, U. Ben-Dor, R. Adar, E. Shapiro, "An autonomous molecular computer for logical control of gene expression". *Nature* **429** (6990):423–429, 2004

[BH01] Bernard, M. & C. de la Higuera, "Apprentissage de programmes logiques par inférence grammaticale", *Revue d'Intelligence Artificielle,* Hermes-Lavoisier Edition, Paris, France, ISSN: 0992499X, 14(3):375–396, 2001

[Bos98] Boström, H. "Predicate Invention and Learning from Positive Examples Only", *Proc. of the Tenth European Conference on Machine Learning*, Springer Verlag, pp. 226-237, 1998

[BJ99] Bshooty, N. & J. Jackson, "Learning DFA over the uniform distribution using a quantum example oracle", *SIAM J. Comput.*, ISSN (electronic): 1095-7111, 28(3):1136-1153, 1999

[Cas90] Casacuberta, F. "Some relations among stochastic finite state networks used in automatic speech recognition", *IEEE Trans. PAMI*, ISSN: 0162-8828, 12(7):691-695, 1990

[Chi01] Chidlovskii, B., "Schema Extraction from XML data: a grammatical inference approach", *Proc. 8th Int. Worksh. on Knowledge Repres. Meets Databases (KRDB'01) CEUR Worksh. Proc.*, CiteSeerX 10.1.1.2.4760, vol. 45, 2001

[Cho59] Chomsky, N. "On certain formal properties of grammars", *Inform. and Control*, ISSN: 0019-9958, 137-167, 1959

[CK02] Cicchello, O. & S. Kremer, "Beyond EDMS", *Proc. of ICGI00, LNAI*, 2484:28-48, Springer, Berlin 2002

[CK03] Cicchello O., Kremer S. C., "Inducing grammars from sparse data sets: a survey of algorithms and results", *JMLR,* ISSN (online):1533-7928, 4:603-632, 2003

[CKR10] Chakraborty, S., E. Keller, A. Ray, J. Mayer, "Symbolic identification of dynamical systems: theory and experimental validation", *American Control Conference,* Baltimore, MD, USA, June 30-July 02, 2010

[CMZ05] Črepinšek, M., M. Mernik, & V. Žumer, "Extracting grammar from programs: brute force approach", *ACM SIGPLAN Notices*, ISSN: 0362-1340, 40(4):29-38, 2005

[CN89] Clark, P., & T. Niblett, "The CN2 induction algorithm", *Machine Learning*, ISSN (print): 0885-6125, ISSN (online): 1573-0565, 3:261-283, 1989

[Coh04] Cohen, J., "Bioinformatics - an introduction for computer scientists", *ACM Computing Surveys*, ISSN (print): 0360-0300, ISSN (online): 1557-7341, 36(2): 122–158, June 2004

[deH97] de la Higuera, C. "Characteristic sets for polynomial grammatical inference", *Machine Learning*, ISSN (print): 0885-6125, ISSN (online): 1573-0565, 27:125-137, 1997

[deH05] de la Higuera, C. "A bibliographical study of grammatical inference", *Pattern Recognition*, ISSN: 0031-3203, 38:1332-1348, 2005

[deH10] de la Higuera, C. "*Grammatical Inference - Learning Automata and Grammars*", Cambridge University Press, ISBN 978-0-521-76316-5, 2010

[deH96] de la Higuera, C., J. Oncina & E. Vidal, "Identification of DFA: data-dependent versus data-independent algorithm", *Proc. of ICGI96*, *LNAI*, Springer, 1147:313-325, 1996

[deH02] de la Higuera, C. & J. Oncina, "On sufficient conditions to identify in the limit classes of grammars from polynomial time and data", *Proc. of ICGI96*, *LNAI*, Springer, 2484:134-148, 2002

[DLT01] Denis, F., A. Lemay & A. Terlutte, "Learning regular languages using RFSA", *"Proc. of ALT2001"*, *LNCS*, Springer, 2225:348-363, 2001

[DBK92] Dean, T., K. Basye, L. Kaelbling, E. Kokkevis, O. Maron, D. Angluin & S. Engelson, "Inferring finite automata with stochastic output functions and an application to map learning", *Proc. of the 10th Nat. Conf. on AI*, pp. 208–214, 1992

[DMV94] Dupont, P., L. Miclet & E. Vidal, "What is the search space of the regular inference?", *Proc. of ICGI94*, *LNAI*, Springer, 862:25-37, 1994

[Ear70] Earley, J., "An efficient context-free parsing algorithm" *Comm. ACM*, ISSN: 0001-0782, 13(2):94-102, 1970. {url : www.acm.org}

[Ehr79] Ehrig, E., "Introduction to the algebraic theory of graph grammars", In V. Claus, H. Ehrig, and G. Rozenberg, (Eds.), "*Graph-Grammars and Their Application to Computer Science and Biology*", Springer-Verlag, pp. 1–69, 1979.

[ERS97] Erlebach, T., P. Rossmanith, H. Stadtherr, A. Steger & T. Zeugmann, "Learning one-variable pattern languages very efficiently on average, in parallel and by asking queries", *Proc. of ALT97, LNCS*, Springer, 1316:260-276, 1997

[Eyr06] Eyraud, R., "*Inference Grammaticale de Langages Hors-Contexte*", PhD Thesis, Faculté des Sciences et Techniques de Saint-Etienne, 2006

[Fu74] K. S. Fu, "*Syntactic Methods in Pattern Recognition*", Academic Press, New York, 1974.

[Gdd08] Goddard, W. "*Introducing the Theory of Computation*", Jones and Bartlett Publishers Inc., ISBN: 9780763741259, 2008

[Gol67] Gold, E. M. "Language identification in the limit", *Inform. and Control*, ISSN: 0019-9958, 10(5):447-474, 1967

[Gol78] Gold, E. M. "Complexity of automata identification from given data", *Inform. and Control*, ISSN: 0019-9958, 37:302-320, 1978

[Hag07] Hagras, H. "Type-2 FLCs: a new generation of fuzzy controllers", *Computational Intel. Mag., IEEE*, ISSN: 1556-603X, 2(1):30-43, Feb. 2007

[Ham94] Hamdi-Cherif, A. "The CASCIDA Project - A computer-aided system control for interactive design and analysis", *Proc. of IEEE / IFAC Joint Symposium on CACSD (CASCD'94),* Tucson, AZ, USA, p.247-251,1994

[Ham10] Hamdi-Cherif, A. "Towards robotic manipulator grammatical control", Invited Book Chapter In: Suraiya Jabin (Ed.) *"Robot Learning"*, SCIYO Pub., ISBN 978-953-307-104-6; pp. 117-136, October 2010

[HH07a] Hamdi-Cherif, C., & A. Hamdi-Cherif, "Apprentissage inductif de grammaires: Le système GASRIA. (Inductive learning for grammars: The GASRIA System)", *Revue d'Intelligence Artificielle*, Hermes-Lavoisier Edition, Paris, France, ISSN: 0992499X, 21(2):223-253, March-April 2007

[HH07b] Hamdi-Cherif, C. & A. Hamdi-Cherif, "*ILSGInf* : Inductive learning system for grammatical inference", *WSEAS Trans. on Comp.*, ISSN: 1991-8755, 6(6):991-996, July 2007, http://www.wseas.org

[HH09a] Hamdi-Cherif, A. & C. Hamdi-Cherif, "Grammatical inference methodology for control systems", *WSEAS Trans. on Comp.*, ISSN: 1991-8755, 8(4):610-619, April 2009

[HH09b] Hamdi-Cherif, A., C. Kara-Mohammed (*alias* Hamdi-Cherif), "Grammatical inference for robotic self-assembly: basic methodology", *Recent Advances in Artificial Intelligence, Knowledge Engineering and Database (AIKED'09)*", Cambridge, UK, February 21-26, 2009, ISBN: 978-960-474-051-2, ISSN: 1790-5109, pp. 447-452, 2009

[HH11] Hamdi-Cherif, A., C. Kara-Mohammed (*alias* Hamdi-Cherif), "Evolutionary multiobjective optimization for medical classification", *2011 IEEE GCC Conference & Exhibition, "For Sustainable Ubiquitous Technology"*, Dubai, United Arab Emirates, pp. 441-444, 19-22 February 2011

[Hof00] Hoffmann B. "Hierarchical graph transformation" *Int. J. of Comp. and Syst. Sci.*, ISSN (print): 1064-2307, ISSN (online): 1555-6530, pp. 98-113, 2000

[Hor72] Horning, J. J. "A procedure for grammatical inference" *Information Processing* ISSN: 0162-8828, 71:519-523, 1972

[Ish90] Ishizaka, H. "Polynomial time learnability of simple deterministic languages", *Machine Learning,* ISSN (print): 0885-6125, ISSN (online): 1573-0565, 2(2):151-164, 1990

[Jon04] Jonyer, I. "MDL-based context-free graph grammar induction and applications", *Int. J. on AI Tools*, ISSN: 1793-6349, 13(1):65-73, 2004

[KC07] Kendal, S.L. & M. Creen, *"An Introduction to Knowledge Engineering"*, ISBN 13: 978-1-84628-475-5, 2007

[KGL06] Klavins, E., R. Ghrist, D. Lipsky, "A grammatical approach to self-organizing robotic systems", *IEEE Trans. Automat. Contr.,* ISSN: 0018-9286, 51(6):949-962, June 2006

[Kla07] Klavins, E. "Programmable self-assembling", *IEEE Control Syst. Mag.*, ISSN: 0272-1708, 27(4):43-56, Aug. 2007

[KMT00] Koshiba T., Mäkinen E., Takada Y., "Inferring pure context-free languages from positive data", *Acta Cybernetica*, ISSN: 0324-721X, 14(3):469-477, 2000

[Kan98] Kanazawa, M. "Learnable classes of categorial grammars", *CSLI Publications*, Stanford, CA, 1998

[KNN01] Komarova N.L., Niyogi P., Nowak M.A., "The Evolutionary dynamics of grammar acquisition", *J. theor. biology*, ISSN: 0022-5193, 209(1): 43-59, 2001

[Kos95] Koshiba, T., "Typed pattern languages and their learnability", *Proc. of Euro COLT95, LNAI*, Springer, 904:367-379, 1995

[KR07] Kermorvant, C., & A. Rafrafi, "Automata learning for numerical entities extraction from OCR output", *Proc. of the ICML Worksh. on Challenges and App. of Grammar Induction*, 2007

[KR97] Khardon R., Roth D., "Learning to reason", *J. of the ACM*, ISSN: 0004-5411, 44(5):697-725, 1997

[KW97] Kondas, A. & J. Watrous, "On the power of quantum finite state automata", *Proc. of 38th of IEEE Conf. on Foundations of Computer Science (FOCS97)*, ISBN: 0-8186-8197-7, pp. 66 – 75, 1997

[Lan92] Lang, K. "Random DFA's can be approximately learned from sparse uniform examples", *Proc. of COLT*, pp. 45-52, 1992

[Lar02] Larichev O.I., "Close imitation of expert knowledge : the problem and methods", *Int. J. of Inf. Tech. &Decision Making (IJITDM),* ISSN (print): 0219-6220. ISSN (online): 1793-6845, 1(1):27-42, 2002

[Knu94] Knuutila, T. & M. Steinby, "Inference of tree languages from a finite sample: an algebraic approach", *Theoret. Comput. Sci.*, ISSN: 0304-3975, 129:337-367, 1994

[LPP98] Lang, K.J., B.A. Pearlmutter & R.A. Price, "Results of the Abbadingo One : DFA learning competition and a new evidence-driven state merging algorithm", *Proc. of ICGI98, LNAI*, Springer, 1433:1-12, 1998

[Lan00] Langley, P. & S. Stromsten, "Learning context-free grammars with a simplicity bias", *Proc. of ECML2000, 11th Eur. Conf. on Machine Learning, LNCS*, Springer, 1810:220-228, 2000

[Lee92] Leermakers, R., "Recursive ascent parsing: from Earley to Marcus". *Theoret. Comp. Sc.*, ISSN: 0304-3975, 104:299-312, 1992

[Lee96] Lee, L. "Learning of context-free languages: a survey of the literature", Technical Report RT-12-96, Center for Research in Computing Technology, Harvard University, Cambridge, MA, 1996

[Luc94] Lucas, S., E. Vidal, A. Amari, S. Hanlon & J.C. Amengual, "A comparison of syntactic and statistical techniques for offline OCR", *Proc. of ICGI94*, *LNAI*, Springer, 862:168-179, 1994

[LN03] Laxminarayana J. A. & G. Nagaraja, "Inference of a subclass of context-free grammars using positive examples", *ECML Worksh. on Learning Context-Free Grammars*, pp. 29-40, 2003

[Mäk96] Mäkinen, E. "A note on the grammatical inference problem for even linear languages", *Fundam. Inf.*, ISSN: 0169-2968, 25(2):175-182, 1996

[MB95] Morgan, N. & H. Bourlard, "Continuous speech recognition", *IEEE Sig. Process. Mag.*, ISSN: 0018-9294, 12(3):25-42, May 1995

[MDP01] Martins, J. F., J.A. Dente, A.J. Pires, and R. Vilela Mendes "Language identification of controlled systems: modeling, control, and anomaly detection", *IEEE Trans. On Syst. Man and Cyb. – Part C: Appl. And Rev.* ISSN: 1094-697731, (2):234-242, 2001

[MGZ03] Mernik, M., G. Gerlic, V. Zumer, & B.R. Bryant, "Can a parser be generated from examples?" *Proc. ACM Symp. On Appl. Comp.*, pp. 1063-1067, 2003

[MHB09] Mernik, M., D. Hrnčič, B.R. Bryant, A.P. Sprague, J. Gray, Q. Liu & F. Javed, "Grammar inference algorithms and applications in software engineering", *XXII Int. Symp. on Inf., Comm. and Automation Tech., ICAT*, Print ISBN: 978-1-4244-4220-1, pp. 1 – 7, 29-31 October 2009

[Mit97] Mitchell, T.M. "*Machine Learning*", McGraw-Hill, New York, ISBN 10: 0-07 042807-7, 1997

[MKE07] McNew, J. M., E. Klavins, & M. Egerstedt, "Solving coverage problems with embedded graph grammars", In A. Bemporad, A. Bicchi, and G. Buttazzo, (Eds.), "*Hybrid Systems: Computation and Control*", Springer-Verlag, pp. 413-427, 2007

[Moo00] Moore, C. *et al.*, "Quantum automata and quantum grammars", *Theoret. Comput. Sci.*, ISSN: 0304-3975, 237:275-306, 2000

[MR11] Mikut, R. & M. Reischl, "Data mining tools", *Wiley Interdisciplinary Reviews Data Mining and Knowledge Discovery*, ISSN: 1942-4795, 1(5):431–443, Sept./Oct. 2011

[Mug99] Muggleton S., "Inductive logic programming: issues, results and the challenge of learning language in logic", *Artificial Intelligence*, ISSN: 0004-3702, 114:283-296, 1999

[NS72] Newell, A., & H. A. Simon, *"Human Problem Solving"* Prentice-Hall, 1972

[NW97] Nevill-Manning, C., & I. Witten, "Identifying hierarchical structure in sequences: a linear-time algorithm", *J. Artif. Intell. Res.*, ISSN: 1076-9757, 7:67-82, 1997

[OG92] Oncina, J. & P. Garcia, "Inferring regular languages in polynomial updated time", In P. de la Blanca, Sanfeliu & E. Vidal (Eds.), "*Pattern Recognition and Image Analysis*", World Scientific, ISBN**:** 9789812797902, 1992

[Onc98] Oncina, J. "The data driven approach applied to the OSTIA algorithm", *Proc. of ICGI98, LNAI*, Springer, 1433:50-56, 1998

[PV96] Parekh, B. & V. Honavar, "An incremental interactive algorithm for grammar inference", *Proc. of ICGI03*, *LNAI*, Springer, pp. 238-249, 1996

[PW93] Pitt, L., M. Warmuth, "The minimum consistent DFA problem cannot be approximated within any polynomial", *J. ACM*, ISSN: 0004-5411, 40(1):95-142, 1993

[Qui93] Quinlan, J. R. "*C4.5: Programs for Machine Learning*", Morgan Kaufmann, ISBN: 1558602380, 1993

[RG01] Rocco, A., S. Servedio & J. Gortler, "Quantum versus classical learnability", *16th Annual IEEE Conf. on Computational Complexity (CCC'01),* ISBN: 0-7695-1053-1, pp. 138-148, 2001

[RPW04] Rothemund, P. W. K., N. Papadakis & E. Winfree, "Algorithmic self-assembly of DNA Sierpinski triangles", *PLoS Biology,* ISSN (print): 1544-9173, ISSN (online): 1545-7885, **2**(12):e424, 2004

[RN03] Russell, S., P. Norving, "*Artificial Intelligence, A Modern Approach*", Chapter 22 Communication, pp. 791-833, Prentice Hall, 3rd Edition 2003

[Ros97] Rosenberg, R. & A. Salomaa, "*Handbook of Formal Languages, Vol. 1 Word, Language, Grammar*", Springer 1997

[RZ01] Rossmanith, P. & A. Zeugmann, "Stochastic finite learning of the pattern languages", *Machine Learning,* ISSN (print): 0885-6125, ISSN (online): 1573-0565, 44(1):67-91, 2001

[Sai06] Saidi, A.S. "Using Grammatical inference in structure induction", *Proc. 15th Int. Conf. On Computing (CIC06),* ISBN: 0-7695-2708-6, pp. 92-104, 2006

[Sak92] Sakakibara, Y. "Learning context-free grammars from structural data in polynomial time," *Theoret. Comp. Sci.*, ISSN: 0304-3975, 76:223-242, 1992

[Sak97] Sakakibara Y., "Recent advances of grammatical inference", *Theoretical Computer Science*, 185:15-45, 1997

[Sak00] Sakakibara, Y. & H. Muramatsu, "Learning context-free grammars from partially structured examples", *Proc. of ICGI00, LNAI*, Springer, 1891:229-240, 2000

[Sav04] Savchenko, S. "Regular expression mining" *Dr Dobbs Journal,* ISSN 1044-789X, 29(2):46–48, 2004

[Seb03] Sebban, M., & Janodet, J., "On state merging in grammatical: a statistical approach for dealing with noisy data", *20th Int. Conf. on Machine Learning*, pp. 688-695, 2003

[SF01] Stanley, R. P. & S. Fomin, "*Enumerative Combinatorics*" Volume 2, Cambridge University Press, 2001

[Sip06] Sipser, M. "*Introduction to the Theory of Computation*", Thomson Course Technology; 2nd Edition, ISBN-13: 978-0534947286, 2006

[Sol59] Solomonoff, R. J. "A new method for discovering the grammars of phrase structure languages," *Proc. Int. Conf. Inf. Process.*, New York, UNESCO, 1959

[Tak88] Takada, Y. “Grammatical inference for even linear languages based on control sets”, *Inform. Process. Lett.*, ISSN: 0020-0190, 28:193-199, 1988

[Tan87] Tanatsugu, K. “A grammatical inference for context-free languages based on self-embedding", *Bull. Informatics and Cybernetics,* ISSN: 0286-522X, 2(3-4):149-163, 1987

[Tho02] F. Thollard, A. Clark, “Shallow parsing using probabilistic grammatical inference”, *Proc. of ICGI00, LNAI*, , Springer, 2484:269-282, 2002

[TK05] Tanner, H.G., A. Kumar, “Formation stabilization of multiple agents using decentralized navigation functions,” In S. Thrun, G. Sukhatme, S. Schaal, and O. Brock, (Eds.) *Robotics: Science and Systems I*, MIT Press, pp. 49–56, 2005

[TB73] Trakhtenbrot, B. & Y. Barzdin, *“Finite Automata: Behaviour and Synthesis”*, North-Holland Pub. Co., ISBN 0-444-10418-6, 1973

[Val84] Valiant, L. G. “A theory of the learnable”, *Comm. ACM,* ISSN: 0001-0782, 27(11):1134-1142, 1984

[Vil00] Vilar, J. M. “Improve the learning of sub-sequential transducers by using alignment and dictionaries”, *Proc. of ICGI00, LNAI*, Springer, 1891:298-312, 2000

[Win00] Winfree, E. “Algorithmic self-assembly of DNA: theoretical motivations and 2-D assembly experiments,” *J. Biomolec. Struct. Dyn.*, **ISSN**: 0739-1102, 11(2):263–270, May 2000.

[Win93] Winston P.H. “*Artificial Intelligence*”, 3rd Edition, *Addison-Wesley Series in Computer Science*, ISBN-10: 0201533774, ISBN-13: 978-0201533774, 1993

[WFH11] Witten, I.H., E. Frank, M.A. Hall, *“Data Mining: Practical Machine Learning Tools and Techniques”*, 3rd Edition, Morgan Kaufmann, ISBN 978-0-12-374856-0, January 2011

[Yok88] Yokomori, T. “Inductive inference of context-free languages based on context-free expressions”. *Int. J. Computer Math.,* ISSN (print): 0020-7160. ISSN (online): 1029-0265, 24, 115-140, 1988

[ZM96] Zelle, I. & R.I. Mooney, “Learning to parse database queries using inductive logic programming”, *Proc. Of the Thirteen Nat. Conf. on AI*, 2:1050-1055, 1996

GLOSSARY

English	Français	عربي
Alphabet	Alphabet	أبجدية
Alphabetical order	Ordre alphabétique	ترتيب أبجدي
Automata	Automates	أتمتة
Automaton	Automate	أتمتة
Automaton, deterministic push-down	Automate à pile	أتمتة مكدس
Automaton, finite (deterministic)	Automate fini (déterministe)	أتمتة محدودة (قطعية)
Automaton, finite (non deterministic)	Automate fini non déterministe	أتمتة محدودة غير قطعية
Automaton, linear bounded	Automate linéaire borné	أتمتة خطية محدودة
Automata, skeletal tree	Automate d'arbre squelettique	أتمتة لشجرة هيكلية
Background (or prior) knowledge	Connaissance de fond (*a priori)*	المعرفة الخلفية (المسبقة)
Backus Naur Form	Forme de Backus-Naur	شكل باكوس - نور
Character in a string	Caractère dans une chaine	حرف من السلسلة
Chomsky normal form	Forme normale de Chomsky	الشكل النظامي لشومسكي
Chaining	Chainage	تسلسل
Chaining, backward	Chainage arrière	تسلسل خلفي
Chaining, forward	Chainage avant	تسلسل أمامي
Chaining, hybrid	Chainage hybride	تسلسل هجين
Cocke-Younger-Kasami algorithm	Algorithme de Cocke-Younger-Kasami	خوارزم كوك - يونغر- كازامي
Complement of a language	Complément d'un langage	مكمّل للغة
Concatenation of positive and negative evidence	Concaténation de preuves positive et négative	ترصيص الأدلة الموجبة و السالبة
Clause	Clause	فقرة
Clauses, conjunction of	Conjonction de clauses	وصل الفقرات
Clauses, disjunction of	Disjonction de clauses	فصل الفقرات
Conflict resolution set	Ensemble de résolution de conflit	مجموعة اختزال التعارض

Constraint satisfaction problem	Problème à satisfaction de contraintes	مسألة تحقيق القيود
Control variable	Variable de commande	متغير التحكم
Definite semantics	Sémantique définie	دلالات معرفة
Empty character	Caractère vide	الحرف الفارغ
Emptiness	Vide	الفراغ
Entailment	Implication	استلزام
Equivalence	Equivalence	التكافؤ
Evidence	Preuve	دليل
"Fail-first" heuristic	Heuristique du premier échec	بديهية أول رسوب
Finiteness	Finitude	محدودية
Grammar	Grammaire	النحو
Grammar, context-sensitive	Grammaire à contexte sensitif	النحو الحساس للسياق
Grammar, context-free	Grammaire à contexte libre	النحو المستقل عن السياق
Grammar, formal	Grammaire formelle	النحو الشكلي
Grammar, hypothesis	Grammaire hypothèse	النحو المفروض
Grammar inference (or induction)	Inférence (ou induction) grammaticale	الاستدلال (أو الاستقراء) النحوي
Grammar, regular	Grammaire régulière	النحو المنتظم
Grammar, size of a	Grammaire, taille d'une	حجم النحو
Grammar, stochastic context-free	Grammaire stochastique a contexte libre	النحو العشوائي المستقل عن السياق
Grammar, target	Grammaire cible	النحو الهدف
Grammar, unrestricted (free)	Grammaire, non restreinte (libre)	النحو غير المقيد، الحر
Inductive inference and definite semantics	Inférence inductive et sémantique définie	استدلال تراجعي و دلالات معرّفة
Inductive inference and normal semantics	Inférence inductive et sémantique normale	استدلال تراجعي و دلالات نظامية
Inductive inference rule	Règle d'inférence inductive	قانون استدلال تراجعي
Inductive logic programming	Programmation logique inductive	برمجة منطقية استدلالية
Inferred grammar at a given stage of the inference process	Grammaire inférée à un niveau donné du processus d' inférence	نحو مستدل عنه في مستوى معين من مرحلة الاستدلال
Information extraction	Extraction d'information	استخراج المعلومة
Information retrieval	Recherche d'information	استعلام عن المعلومة

Initial inferred grammar	Grammaire initiale inférée	النحو الابتدائي المستدل عنه
Knowledge base	Base de connaissances	قاعدة المعرفة
Knowledge-based system	Système à base de connaissance	نظام قاعدة المعرفة
Language	Langage	لغة
Language, context-free	Langage, à contexte libre	اللغة المستقلة عن السياق
Language, domain-specific	Langage spécifique au domaine	اللغة الخاصة بالميدان
Language defined over an alphabet	Langage défini selon un alphabet	لغة معرفة حسب أبجدية
Language, formal	Langage, formel	اللغة الشكلية
Language generated by a given given grammar	Langage généré par une grammaire donnée	لغة مولدة بنحو معين
Language, regular	Langage, régulier	اللغة المنتظمة
Language, recursive enumerable	Langage énumérable récursif	لغة عودية، قابلة للعد
Language, target	Langage cible	اللغة الهدف
Learning	Apprentissage	التعلم
Learning, machine	Apprentissage automatique	التعلم الآلي
Learning, semi-supervised	Apprentissage semi-supervisé	تعلم بإشراف جزئي
Learning, supervised	Apprentissage supervisé	التعلم بإشراف
Unsupervised learning	Apprentissage non supervisé	التعلم بغير إشراف
Left- and right-hand-side of a production	Partie gauche et partie droite de la production	الجهة اليسرى و اليمنى للإنتاج
Length of string	Longueur de la chaine	طول السلسلة
Lexical order over strings	Ordre lexical dans les chaines	ترتيب معجمي في السلاسل
Logic	Logique	منطق
Logic, first order	Logique du premier ordre	منطق الدرجة الأولى
Logic, propositional	Logique des propositions	منطق القضايا
Membership query	Requête d'appartenance	استعلام عن الانتماء
Membership problem	Problème d'appartenance	مسألة الانتماء
Minimum adequate teacher	Enseignant adéquat minimal	أصغر معلم مناسب
Minimum remaining value	Valeur minimale restante	أقل قيمة متبقية
Most constrained variable	La variable la plus contrainte	المتغير الأكثر قيودًا
Most general concatenation of all sub-sentences	La concaténation la plus générale de toutes les sous-phrases	الترصيص الأعم لكل الجمل الجزئية
Multiple derivation	Dérivation multiple	اشتقاق متعدد

Non-terminal	Non terminal	غير نهائي
Number of character in the string	Nombre de caractères dans la chaine	عدد الحروف في السلسلة
Operation	Operation	عملية
Operation, complement	Operation, complement	عملية المكمل
Operation, intersection	Operation, intersection	عملية التقاطع
Operation, product	Operation, produit	عملية الضرب
Operation, symmetric difference	Operation, difference symétrique	عملية الطرح المتناظر
Operation, union	Operation, union	عملية الإتحاد
Output (controlled) variable	Variable (commandée) de sortie	متغير المخرج (المتحكَّم فيه)
Parsing	Analyse syntaxique	التحليل النحوي
Parsing, bottom-up	Analyse syntaxique, ascendante	التحليل التصاعدي
Parsing, hybrid	Analyse syntaxique, hybride	التحليل الهجين
Parsing, top-down	Analyse syntaxique, descendante	التحليل التنازلي
Partial derivative	Dérivée partielle	المشتق الجزئي
Partial parsing algorithm	Algorithme à analyse syntaxique partielle	خوارزم التحليل النحوي الجزئي
Posterior satisfiability (consistency with negative evidence)	Satisfiabilité *a posteriori* (consistance avec l'évidence négative)	القابلية للتحقق الملحَق (انسجام مع الدليل السلبي)
Posterior sufficiency (or completeness with regard to positive evidence)	Suffisance *a posteriori* (complétude *vis-à-vis* de l'évidence négative)	الكفاية الملحَقة (كمال بالنسبة للدليل السلبي)
Power set	Ensemble puissance	قُوى مجموعة
Programming	Programmation	برمجة
Programming, declarative	Programmation déclarative	البرمجة التصريحية
Programming, imperative	Programmation impérative	البرمجة الآمرة
Programming, functional	Programmation fonctionnelle	البرمجة الوظيفية
Programming, procedural	Programmation procédurale	البرمجة الإجرائية
Programming, object-oriented	Programmation orientée objet	البرمجة الشيئية
Prior necessity	Nécessité *a priori*	ضروري مسبقاً
Prior satisfiability	Satisfiabilité *a priori*	القابلية للتحقق المسبق
Probabilistic approximately correct	Probablement approximativement correct	القريب من الصحيح احتمالاً

Pumping lemma	Pumping lemma	مأخوذ الضخ
Reversal of a string	Inversion de chaine	معكوس السلسلة
Resolution principle	Principe de résolution	مبدأ الإختزال
Sequence of characters	Séquence de caractères	متتالية من الحروف
Set of accepting states	Ensemble des états acceptants	مجموعة الأحوال المتقبِّلة
Set of characters (terminals) or alphabet	Ensemble de caractères (terminaux) ou alphabet	مجموعة الحروف (النهائية) أو أبجدية
Set of hypotheses	Ensemble des hypothèses	مجموعة الفرضيات
Set of initial states	Ensemble des états initiaux	مجموعة الأحوال الابتدائية
Set of positive examples	Ensemble des exemples positifs	مجموعة الأمثلة الموجبة
Set of negative examples or counter examples	Ensemble des exemples négatifs ou contre-exemples	مجموعة الأمثلة السالبة أو الأمثلة المضادة
Set of non-terminals or variables	Ensemble des non terminaux ou variables	مجموعة اللانهائية أو المتغيرات
Set of positive (or negative) examples of sentences	Ensemble des exemples positifs (ou négatifs) de phrases	مجموعة الأمثلة الموجبة (أو السالبة) من الجمل
Set of productions or rules	Ensemble de productions ou de règles	مجموعة منتجات أو من قوانين
Set of rejecting states	Ensemble des états de rejet	مجموعة الأحوال الرافضة
Set of states	Ensemble des états	مجموعة الأحوال
Set of symbols in the stack	Ensemble des symboles dans la pile	مجموعة الرموز في المكدس
Single derivation	Dérivation simple	اشتقاق واحد
Starting symbol	Symbole initial	الرمز الابتدائي
State with branch and read from input	État de branchement et de lecture des entrées	حالة قفز و قراءة من المُدخَلات
State with branch and read from stack	État de branchement et lecture de pile	حالة قفز و قراءة من المكدس
State with no branching but only with push	État sans branchement mais avec empilement seul	حالة بلا قفز و لكن بتكديس فقط
Strings of terminals	Chaine de terminaux	سلسلة من النهائيات
Symbol	Symbole	رمز
Terminal	Terminal	نهائي
Text mining	Fouille de texte	التنقيب عن النصوص
Transition function	Fonction de transition	دالة الانتقال

APPENDIX 1

CLASS OF LANGUAGES INFERRED BY *GASRIA*

Table A1 below gives some of the grammars inferred by *GASRIA*. For each language, the first row contains a description of the language, the second column contains the set *L+* of positive examples and the third column gives the most general grammar inferred by the system. Then a number of rows follow containing the sequence of grammars generated. For each grammar, we give only the set of productions. *S* is the initial symbol. We can conclude that the subclass of languages learned by our algorithm is the linear languages, which incorporate even linear and regular languages. For the search space, the choice of Chomsky normal form for describing the grammar and the collection of non-terminal two by two from left to right, we have reduced the search space to only one possible grammar. Of course, it may not be the best one always.

Table A1 – TABA1: Class of languages inferred by *GASRIA*

Language	*L+*	Most general grammar
$a^n b^n$, $n \geq 1$	ab, aabb, aaabbb	G_1
G_0 = S → AB , A → a, B → b		
G_1= S → AB , A → a, B → b C → AS, S → CB		
S→ x | y | z | S+S | S*S | S-S | S/S | (S) This grammar generates arithmetic expressions using x,y,z variables.	x, y, z, x+y, x-y, x*y, x/y, (x), (x+(x-y)/(z*y-x))	G_7
G_0 = S → x		
G_1= S → x, S → y		

G_2= S → x, S → y, S→ z		
G_3= S → x, S → y, S→ z, S → BS, B → SA, A → +		
G_4= S → x, S → y, S→ z, S → BS, B → SA, A → +, C → -, D → SC, S → DS		
G_5= S → x, S → y, S→ z, S → BS, B → SA, A → +, C → -, D → SC, S → DS, E → *, F → SE, S → FS		
G_6= S → x, S → y, S→ z, S → BS, B → SA, A → +, C → -, D → SC, S → DS, E → *, F → SE, S → FS, G → /, H → SG, S → HS		
G_7= S → x, S → y, S→ z, S → BS, B → SA, A → +, C → -, D → SC, S → DS, E → *, F → SE, S → FS, G → /, H → SG, S → HS, I → (, J →), K → IS, S → KJ		
$b^n ab^{2n}$ $n \geq 1$	babb, bbabbbb, bbbabbbbbb, bbbbabbbbbbbb	G_1
G_0= A → b, B → a, C → AB, D → CA, S → DA,		
G_1= A → b, B → a, C → AB, D → CA, S → DA, E → AS, F → EA, S → FA		
$b^n abcb^{3n}$ $n \geq 0$	abc, babcbbb, bbbabcbbbbbbbbb	G_1
G_0= A → a, B → b, C → c, D → AB, S → DC		
G_1= A → a, B → b, C → c, D → AB, S → DC, E → BS, F → EB, G → FB, S → GB		
aaabbbbb, aab	aaabbbbb, aab	G_1
G_0= A → a, B → b, C → AA, D → CA, E → DB, F → EB, G → FB, H → GB, S → HB		
G_1= S → CB		

APPENDIX 2 – *ILSGINF* CLASS DIAGRAM

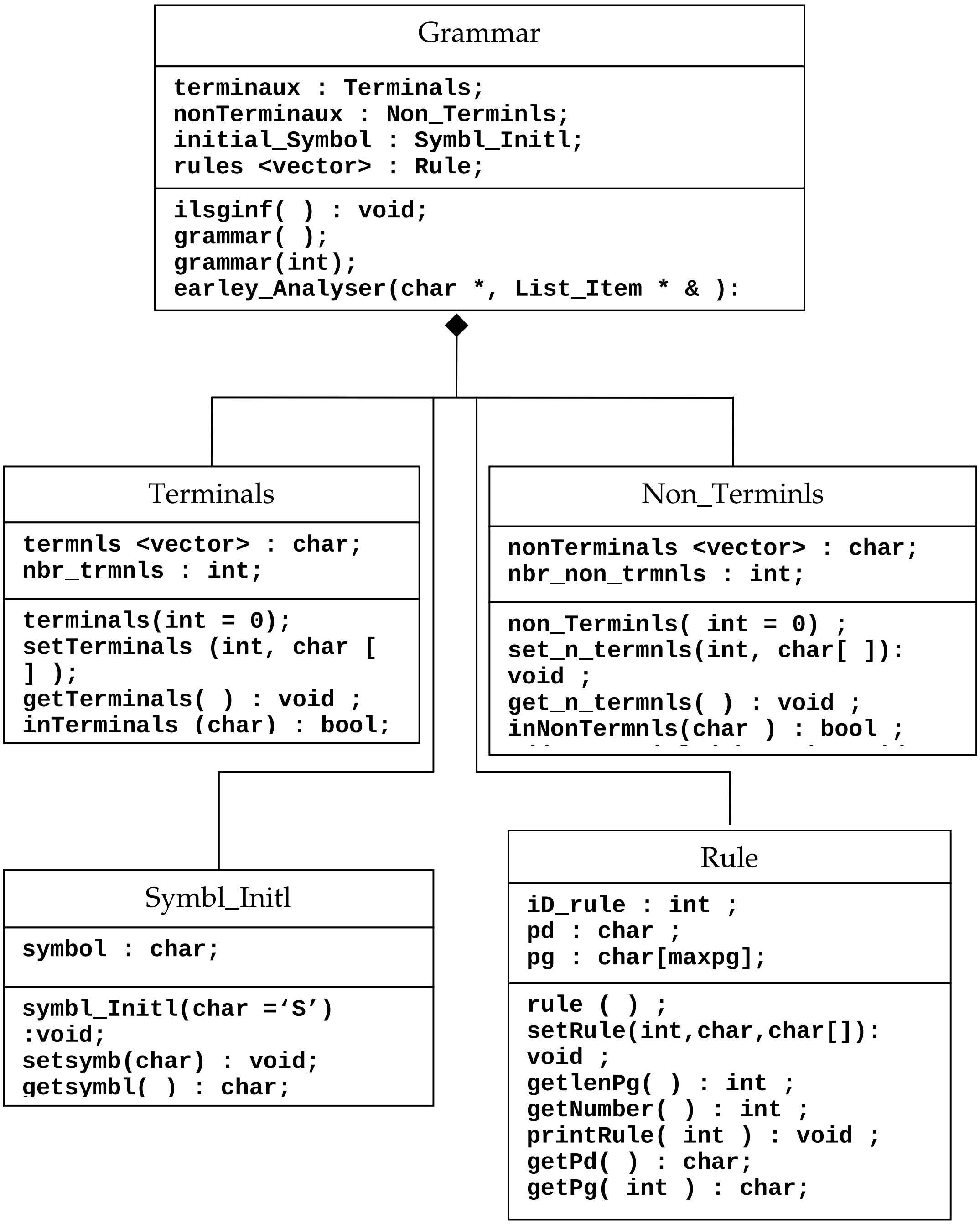


Figure A2 – DIAG A/2 : ILSGInf class diagram

APPENDIX 3 COMPLEXITY OF *ILSGINF* LEARNING ALGORITHM

For complexity calculation of *ILSGInf*, assume that n is the maximum size of examples in the input sample, and x is the number of examples in it. The estimated time complexity **T(.)** of the algorithm is polynomial with respect to the maximum length of examples in the input sample. The cardinality of the input sample also increases the time complexity until the most general grammar is found.

T(**ILSGInf**, *n*) =

constant+T(**Generate_first_grammar**, *n*)+(*x*-1)*(const+T(***PPA*_Parse**, *n*) + T(**Generalize**, *n*)) (1)

Where:

T(**Generate_first_grammar**, *n*) = *O(3n) = O(n)*

T(***PPA*_Parse**,*n*)=max(T(**Earley_algorithm**,*n*),

$\{\max(\sum_{i=1}^{n} k_i^{3}, \quad k \in [0,n] / \sum_{i=1}^{n} k_i = n\}$ + T(***PaDe*_sorting**, *n*))) (2)

T(**Earley_algorithm**, *n*) = $O(n^3)$ (Earley algorithm known complexity)

T(***PaDe*_sorting**,*n*) = $O(n^2)$ (sorting algorithm known complexity)

$\{\max(\sum_{i=1}^{n} k_i^{3}, \quad k \in [0,n] / \sum_{i=1}^{n} k_i = n\}$ = $O(n^3)$

Thus (2) gives

T(**PPA_Parse**,*n*) = max($O(n^3)$, $O(n^3)+O(n^2)$) = $O(n^3)$

T(**Generalize**, *n*) = $O(n)$

The final result giving the complexity of *ILSGInf* is given by:

T(***ILSGInf***,*n*) = *O(n) + (x-1) * (O(n3) + O(n)) = (x-1) ** $O(n^3)$ = $O(n^3)$

Although, we have been successful in generating a subclass of CFLs in polynomial time, the actual method cannot deal with more complex CFG's such as $\omega\omega^R$. We are now developing adequate heuristics to improve the proposed method to enlarge the set of learned languages.

INDEX

Abbadingo learning competition, 36
absorption rule, 117
accepting strings, 19
Active learning, 28
adaptive control, 123, 128
adaptive control methods, 123
AI, 6, 144, 146, 149
All types of queries, 29
alphabet, 5
antecedent, 83, 87, 96
approximately correct, iii, 25, 30, 154
artificial intelligence, II, 6, 41, 42, 56
automata, I, II, iii, 14, 15, 17, 32, 33, 36, 37, 38, 39, 41, 43, 46, 63, 145, 146, 147
automaton, i, ii, 14, 17, 18, 20, 32, 33, 34, 35, 49
background knowledge, 57, 60
background theory, 58, 60
Backtrack characteristics, 82
backward chaining, 3, 80, 82, 83, 87, 104, 138
BLUE*, 35
C4.5, 56, 148
CFG, II, i, xi, xii, 6, 13, 17, 20, 21, 22, 23, 38, 39, 40, 43, 47, 53, 62, 65, 66, 69, 73, 79, 81, 82, 94, 108, 109, 119, 137, 138, 160, 167
CFGs, II, xi, 1, 5, 6, 7, 9, 21, 23, 24, 25, 27, 32, 34, 37, 38, 39, 40, 41, 42, 43, 53, 72, 128, 143, 167
CFL, IV, i, v, 17, 20, 21, 22, 23, 101, 102, 103, 127, 135
CFLs, II, 17, 20, 21, 22, 23, 37, 38, 39, 40, 41, 134, 138, 139, 160
Chaining, 151
CHILL, 57
Chomsky hierarchy, 5, 17, 32, 37, 42, 128
class diagram, iv, 65, 90, 109
classes of languages, 16
classification of sentences, 63
Closed world assumption, 82
closed-loop control, 126
CN2, 56, 144
CNF, i, 22, 39, 72, 78
commentary variable, 97
complementation, 19, 22, 38
computer algebra software, 2
concatenation, i, 11, 19, 42, 54, 73, 77, 117, 119, 153
conclusion, V, ix, 3, 9, 52, 68, 83, 87, 100, 106, 117
condition-action rules, 83
conflict-resolution procedure, 84
conjunction of clauses, 116, 117
Constraint satisfaction problem, i, 152
constraint satisfaction problem (CSP), 99
Constructors, 84, 85
context free grammars, xi, 5, 167
context-free expressions, 42, 149
context-free grammar, iii, 6, 13, 17
context-free grammars, 1, 128, 146, 147, 148
context-free language, IV, 17, 101, 127, 135
Context-free language, i, 18
context-free languages, 16, 18, 138, 146, 147, 149
context-sensitive, 7
context-sensitive grammar, 13, 17, 127, 140
context-sensitive language, 17
Context-sensitive language, 18
context-sensitive languages, 16
Contradiction theorem, 86
control law, 123, 124, 125, 126, 127, 128
control of machine drives, V, 123, 127, 135
control strategy, 125
control systems, V, xi, 6, 7, 123, 124, 125, 126, 129, 135, 140, 141, 142, 145, 167
controlling program, 132
counter-example, 30, 113, 114
counter-examples, 28, 113
data analysis, 56
data mining, III, 4, 56
Data-driven heuristic, II
decision tree learning, 56
decision-making process, 4
declarative, 1, 2, 3, 8, 53, 69, 91, 94, 95, 98, 137, 140, 141, 154
Declarative programming, I, 8
deduction, IV, 80, 83, 84, 86, 116, 117
Deduction theorem, 86
DeLeTe, 37
derivation, i, 23, 39, 43, 47, 52, 69, 73, 74, 114, 139, 153, 155
determinism, 15, 40
Determinism, 38
deterministic CFLs, 17
deterministic finite automaton, 14, 63
DFA, ii, iv, 5, 14, 143, 144, 146, 148
DNA, III, 21, 26, 48, 56, 132, 148, 149
domain-specific language, 50

domain-specific languages, 8
DPDA, i, 15
DSL, i, 50
DTL, 56
dynamic control, V, 127
dynamical system, 126, 129
Earley's algorithm, III, IV, vii, 52, 69, 71, 72, 73, 91, 94, 95, 98, 108, 113
Earley's parser, 67, 72, 80
ECML2003, 37
EMO, III, 50
empty string, 11, 71
entailment, 8, 57, 86
enumerative algorithm, 42
Equivalence, 20, 22, 29, 37, 152
even linear grammars, 34
evidence, i, 35, 36, 58, 59, 60, 61, 146, 151, 154
evidence-driven state merging *(EDSM)* algorithm, 35
evolutionary multiobjective optimization, III, 50
exact hypothesis, 30
EXINF, III, IV, V, iv, v, vi, vii, 8, 53, 63, 66, 67, 68, 69, 75, 77, 78, 79, 80, 81, 87, 89, 90, 91, 92, 93, 94, 95, 97, 98, 99, 100, 101, 102, 103, 104, 105, 108, 109, 129, 130, 134, 137, 138
expert systems, 2, 3, 4, 106
exploitation mode, 63, 66, 67, 68
extended equivalence queries, 41
fact base, 6, 65, 66, 67, 69, 78, 82, 83, 84, 94, 97, 98, 99, 100, 102, 103
facts, v, 3, 4, 8, 9, 47, 61, 63, 66, 84, 86, 87, 88, 90, 97, 98, 99, 100, 102, 103, 108, 109, 140
Factual knowledge, 4
fail-first heuristic, 100
finite automata, II, 5, 13, 43, 45, 128, 144
finite language, 16
Finiteness, 20, 22, 152
`firing`, 84
first-order logic, xi, 6, 7, 8, 52, 53, 54, 66, 79, 80, 90, 104, 137, 167
FOL, I, IV, V, ii, xi, 6, 7, 8, 53, 63, 66, 79, 80, 81, 83, 90, 104, 105, 137, 138, 140, 141, 167
forward chaining, IV, v, vii, 3, 66, 81, 82, 83, 86, 88, 89, 95, 96, 99, 104, 138
Forward chaining, IV, ii, 82, 86, 94, 96
FSA, I, 14
functional programming, 2, 8
GASRIA, III, IV, V, VI, iv, v, vi, vii, 6, 8, 51, 52, 53, 63, 64, 65, 68, 75, 76, 79, 105, 123, 137, 145, 157
general problem solving, I, 5, 6
generalization, 35, 73, 108, 113, 114, 116, 117, 133, 139
Generalization, V, vii, 77, 112, 117
genetic algorithms, 4, 41, 42, 56, 141
GI, II, III, IV, V, VI, ii, iv, v, vi, vii, xi, 3, 5, 6, 7, 8, 17, 25, 26, 27, 36, 38, 40, 44, 45, 46, 47, 48, 49, 50, 51, 52, 58, 62, 79, 80, 104, 105, 106, 108, 123, 124, 126, 127, 128, 129, 130, 131, 134, 135, 137, 138, 140, 141, 142, 167
GIFT, 63
grammatical inference, II, III, V, xi, 3, 5, 6, 17, 24, 25, 45, 51, 53, 62, 80, 104, 105, 124, 126, 135, 137, 143, 144, 145, 146, 147, 148, 149, 167
graph grammar, V, 49, 133, 146
Graph grammars, V, 133, 134
graphs, 27, 108, 133
heuristic, 4, 34, 36, 42, 43, 88, 100, 152
hidden Markov models, 46
hierarchy, I, vi, 16, 18
HMMs, 46
Horn clause logic program, 57
hypotheses, ii, 27, 57, 58, 59, 60, 61, 62, 121, 139, 155
hypothesis, ii, 26, 27, 28, 30, 31, 35, 38, 59, 60, 61, 63, 85, 88, 116, 117, 152
ICGI, 5
identifiable in the limit, 28, 38
identifiers, 67
ILP, III, ii, iv, 2, 51, 52, 54, 57, 60, 61, 62, 121, 139
ILSGInf, II, III, IV, V, VI, iv, v, vi, vii, 9, 43, 44, 63, 65, 69, 75, 77, 78, 80, 81, 83, 89, 91, 101, 104, 105, 108, 109, 110, 111, 112, 114, 123, 124, 127, 129, 130, 134, 137, 138, 145, 159, 160
imperative, 1, 2, 154
imperative languages, 1
implementation, IV, V, xi, 2, 3, 6, 7, 9, 50, 52, 72, 80, 81, 83, 106, 114, 132, 167
Inclusion queries, 29
inconsistency clause, 59
incremental, 27, 34, 53, 66, 70, 135, 139, 148
induction, III, ii, 5, 7, 35, 76, 77, 79, 116, 128, 137, 139, 143, 144, 146, 148, 152
inductive learning, 7, 52, 53, 105, 108, 123, 137
inductive logic programming, 2, 54, 56, 63, 121, 139, 149
inductive machine, 27, 28
inference, I, II, III, ii, iii, iv, xi, 2, 3, 4, 5, 9, 26, 31, 32, 33, 34, 35, 37, 38, 39, 40, 41, 42, 43, 46, 50, 53, 58, 59, 60, 61, 63, 66, 80, 82, 84, 87, 95, 99, 104, 105, 109, 116, 117, 121, 123, 127, 134, 135, 140, 141, 143, 144, 145, 147, 148, 149, 152, 167
inference problem, 26, 34

Inference problem, 26
inferred grammar, ii, 5, 121, 138, 139, 153
informant presentation, 27
information extraction, ii, 47, 48
information retrieval, ii, 47
Initial grammar generation, 77
initial state, 14, 15
input data, 5, 15, 33, 46
Integration, 83
intelligent agents, 4
intelligent system, 3, 82
intersection, 19, 22, 38, 51, 154
intractable problem, 37
KB, ii, 2, 4, 6
k-bounded, 41, 143
KBS, IV, ii, 4, 9, 94
KBSs, I, 4
Kleene star, 19, 22
knowledge, 2, I, IV, i, 2, 3, 4, 6, 8, 9, 38, 39, 47, 52, 56, 57, 58, 59, 60, 62, 63, 65, 66, 81, 82, 85, 87, 90, 91, 94, 96, 106, 107, 108, 109, 111, 121, 127, 130, 139, 140, 141, 142, 146, 151
knowledge base, 2, 6, 8, 82, 87, 91, 96, 130
knowledge engineering, 2
language, III, i, iii, xi, 5, 6, 7, 8, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 22, 23, 24, 27, 28, 32, 39, 40, 42, 45, 46, 47, 50, 53, 54, 55, 56, 57, 65, 66, 67, 68, 73, 77, 78, 79, 81, 82, 87, 103, 104, 108, 109, 111, 112, 126, 129, 130, 131, 133, 138, 139, 147,鬱151, 157, 167
Language defined over an alphabet, ii, 153
Language generated by a given grammar, 153
language theory, xi, 8, 167
lattice, II, 33, 35
LBA, ii, 17
learnability, 31, 36, 40, 43, 146, 148
Learning, II, III, IV, V, VI, v, vi, vii, xi, 26, 34, 36, 37, 43, 50, 56, 65, 75, 77, 83, 89, 108, 112, 130, 143, 144, 145, 146, 147, 148, 149, 153, 160, 167
learning abilities, 49
learning from examples, 6
learning function, 27, 28
learning heuristics, 6
learning inductively, 6
learning layer, xi, 3, 5, 6, 137, 141, 167
learning mode, 63, 66, 83
learning process, xi, 36, 39, 52, 54, 57, 109, 111, 167
LIFO or stack, 15
linear-bounded automaton, 17
local controllers, 131
local rules, 132
Logic, 153
logic programming, III, ii, 1, 2, 8, 57, 81, 137, 147, 152
machine learning, xi, 3, 5, 7, 8, 9, 51, 52, 54, 56, 106, 107, 137, 138, 167
machine learning algorithms, 7, 56
matrix environments, 2, 7
Membership, ii, 20, 22, 29, 153
membership query, 29
MERLIN, 62
Minimum adequate teacher, ii, 29, 153
minimum remaining value, 99
modus ponens, 86, 87
modus tollens, 87
most constrained variable (MCV), 100
multiple-input multiple output (MIMO), 125
negative examples, ii, 36, 38, 40, 48, 63, 127, 155
negative feedback control system, 124
neural networks, 4, 46, 56, 141
NFA, II, ii, 14, 16, 17, 18, 20, 33, 37
Non deterministic finite automaton, ii, 14
non-terminal, 7
non-terminal symbols, 55
object-oriented programming, 2
observation table, 34
observer-based methods of control, 130
one-counter languages, 41
Only membership queries, 29
OOP, 2
oracle, 29, 38, 61, 143
orders, 12
PAC, II, iii, 25, 30, 31
PaDe's, IV, V, vii, 73, 74, 75, 77, 113, 117, 118, 119, 121
PaDe's, V, 114
parenthesis grammar, 43
parse tree, 23, 55, 113
parser, 57, 66, 67, 72, 77, 79, 80, 83, 91, 98, 101, 104, 126, 137, 138, 147
parsing, I, II, IV, V, VI, iii, vii, xi, 6, 7, 8, 9, 22, 23, 24, 42, 45, 52, 53, 54, 55, 62, 63, 73, 74, 75, 77, 79, 80, 81, 83, 86, 89, 91, 94, 95, 98, 104, 105, 106, 108, 113, 114, 115, 116, 118, 120, 121, 138, 140, 144, 147, 149, 154, 167
parsing algorithms, 24
Partial derivative, iii, vi, 154
partial parsing algorithm, 52, 73, 138
Pattern languages, 42
PDA, I, i, iv, 14, 15, 18, 20, 21, 22, 38
PDAs, 17
pivot languages, 41
plausible reasoning, 4, 87
pole placement design, 125
polynomial identification from given data, 36
positive examples, ii, xi, 6, 38, 40, 43, 57, 63, 65, 77, 78, 79, 80, 81, 91, 105,

127, 130, 137, 147, 155, 157, 167
posterior satisfiability, 60
PPA, V, iii, 7, 8, 9, 52, 62, 73, 76, 77, 106, 108, 112, 113, 114, 118, 119, 121, 138, 160
p-production, 127
Pragmatics, 55
predicate logic, 57, 82
prefix tree acceptor, 34, 35
premises, 87, 88, 97, 99, 100
Prior necessity, 59, 60, 154
prior satisfiability, 59
Probably approximately correct, 30
programming language, 10
programming languages, VI, xi, 1, 3, 5, 10, 18, 21, 53, 133, 137, 167
Propagation, 100
propositional logic, 57
p-type production, 128
p-type productions, 128, 140
Pumping lemma, 20, 23, 155
Push-down automata, 14
query, ii, 29, 47, 57, 87, 153
Quotient, 19
Readers, 84, 85
reasoning, IV, 3, 4, 7, 81, 86, 87, 95, 96, 106, 137
Recognized sentence, 78
Recursive enumerable language, 18
recursive enumerable languages, 16
regular expression, 15, 18, 19, 20, 42
regular expressions, 13, 15, 16, 20, 27
regular grammar, 13, 16, 18, 19, 20, 23, 32, 33, 37, 40, 48
regular grammars, II, 5, 25, 27, 32, 37, 39, 128
Regular inference, 35
regular language, IV, 16, 18, 20, 22, 98
Regular language, 18
regular languages, II, 16, 18, 19, 20, 23, 38, 40, 134, 139, 143, 144, 148, 157
Reinforcement learning, 107
Reserved words, 67
residual finite state automata, 37
Resolution principle, 82, 155
RFSA, 37, 144
robotics, 26, 45, 134
RPNI algorithm, II, 34, 48
rule base, III, IV, 6, 66, 67, 68, 69, 84, 88, 94
Rule saturation, 100
rules, V, ii, v, 3, 4, 6, 7, 8, 9, 13, 16, 17, 23, 26, 41, 47, 48, 49, 54, 55, 56, 58, 61, 62, 63, 65, 66, 67, 68, 75, 82, 83, 84, 85, 86, 88, 90, 91, 96, 97, 98, 102, 103, 109, 114, 116, 117, 121, 128, 130, 131, 132, 133, 139, 140, 155
satisfiability, 59, 60, 154
self assembly, 49, 123
self-assembly, V, VI, 6, 7, 9, 123, 124, 127, 131, 132, 133, 134, 135, 140, 145, 148, 149
Self-assembly, V, 130, 131, 132
semantics, III, iv, 42, 45, 55, 57, 59, 60, 61, 143, 152
Semi-supervised learning, 107
sequence, i, 11, 12, 14, 23, 24, 45, 70, 73, 99, 129, 157
set of states, 14, 34
Set of symbols in the stack, ii, 155
shells, 2
Simple deterministic languages, 41
simple variable, 97
soft computing, 130, 134, 140
software engineering, 2, 45, 50, 147
specialization, 76, 113, 114, 116, 117
Stand-alone inferences capability, 81
start symbol, 13, 55, 129
starting graph, 133
state-feedback, 123, 134
state-space methods, 123
string, V, i, iii, xi, 5, 6, 11, 12, 14, 15, 16, 20, 21, 22, 23, 28, 29, 34, 41, 46, 54, 55, 67, 68, 70, 71, 72, 73, 74, 82, 83, 94, 95, 98, 102, 103, 113, 114, 117, 118, 133, 134, 135, 151, 153, 154, 155, 167
string-lengths, 5
Strong equivalence query, 29
structural completeness, 35
structural membership, 43
structurally reversible languages, 41
SubdueGL, 108
Supervised learning, 107
Symbol, i, ii, 68, 155
symbolic environments, 3
symbolic processing, 2
syntactic, III, 6, 7, 8, 23, 26, 32, 52, 56, 66, 68, 74, 108, 118, 121, 139, 141, 147
syntactic level, 6
syntax, III, iv, 5, 42, 45, 66, 68, 83, 143
target language, 28, 29, 30
Terminal, 155
terminal distinguishable CFGs, 43
terminal distinguishable CFLs, 41
terminal symbols, 54
terminals, i, ii, iii, 7, 13, 23, 39, 45, 55, 69, 73, 114, 117, 126, 155
text presentation, 27
Transduction, 107
Transition function, i, 155
Traxbar algorithm, II, 35
Turing machine, 17, 18
type-0, 16
type-1, 16, 128
type-2, 5, 16, 128

type-3, 5, 16, 128
union, 19, 22, 58, 154
Unrecognized sentence, 78
unrestricted, 7, 17, 32, 152
unrestricted grammar, 13, 17
Unsupervised learning, 107, 153
variables, V, ii, 13, 42, 58, 67, 82, 96, 97, 100, 124, 128, 130, 155, 157
visual programming, 2
Weak equivalence query, 29
window-EDMS (*W-EDMS)*, 36

Abstract – Résumé - ملخص

ملخص

إن غالبية لغات البرمجة تقوم على قواعد نحوية مستقلة عن السياق. و إن الغرض من الاستدلال النحوي هو استنباط قواعد اللغة من مجموعة مدخلة من الجمل الصحيحة و أحيانا غير صحيحة. إننا نهتم في دراستنا هذه بالنحو المستقل عن السياق. وبما أن القواعد الشكلية المستنبطة في هذا النوع من النحو لا يدل فقط على طريقة تركيب الجمل بل على العلاقة بين الوحدات المختلفة المكونة للجملة و بالتالي يساعد على فهم المعنى.

بناء على ما سبق، نقترح إنتاج بيئة متبوعة بتنفيذ، من شأنها توحيد الجوانب المختلفة للبرمجة في إطار التعلم الآلي. إن الفكرة المحورية للعمل المقترح هي استخدام الاستدلال النحوي بوصفه إطارًا موحدًا لتحقيق هذا التكامل. بما أن أي برنامج هو أساسًا مجموعة من السلاسل، فإننا نبين أن استخدام الاستدلال النحوي يُمْكنه، زيادة على المساهمة في تكامل الجوانب المختلفة للبرمجة، أن يمتد أيضًا إلى مجالات أخرى أوسع نطاقاً.

يتمحور العمل حول المساهمات التالية :

- دراسة نظرية للغات البرمجة؛
- دراسة الاستدلال النحوي؛
- دراسة و تنفيذ لبيئة تدمج التعلم الآلي والمنطق من الدرجة الأولى؛
- دراسة و تنفيذ نظام مبني على منطق الدرجة الأولى لاستعماله في تحليل الجمل بطريقة منفردة أو بالاعتماد على التعلم؛
- دراسة و تنفيذ لخوارزم مبني على الحدسيات و استعماله لتحسين عملية التعلم في إطار الاستدلال النحوي، و في زمن محدود.
- التداخل بين الاستدلال النحوي و أنظمة التحكم الآلي.

إن هذا العمل يفتح مجالا واعدا للبحث في إطار المساهمة في تكامل لغات البرمجة، هادفًا إلى إثرائها بإضافة مستوى خاص بالتعلم الآلي.

الكلمات المفتاحية

تصنيف اللغات، لغات التصميم، النحو و أساليب إعادة الكتابة، تحليل الجمل، اللغات الصورية، ذكاء اصطناعي، استنتاج و برهنة القوانين، محرك الاستدلال، التعلم، اكتساب اللغة.

Abstract

Most programming languages are based on context free grammars (CFGs). The purpose of grammatical inference is to infer a grammar, in our situation a CFG, from positive examples of sentences and possibly incorrect ones, for a given language. Based on these two fundamental definitions, we propose an environment followed by an implementation unifying different aspects of programming in machine learning settings. The central idea of this work is to use grammatical inference (GI) as a unifying framework for achieving this integration. Because any program can be considered as a string of characters, we show that the use of grammatical inference can not only unify different aspects of programming but also extend to wider areas of applications. The work sums up the following contributions:

- State of the art of language theory and of grammatical inference;
- Design and development of an environment integrating machine learning and first-order logic (FOL);
- Design and development of a FOL system for parsing sentences independently or with a learning module;
- Design and development of a heuristics-based polynomial-time complexity algorithm enhancing the learning process in grammatical inference.
- Interaction between grammatical inference and control systems.

The present work bears a promising line of research, contributing further to programming languages integration, aiming at the improvement of these languages with a machine learning layer.

ACM Categories and Subject Descriptors

D.3.1 [Formal definitions and theory], **D.3.2** [Language classifications], *Design languages*, **F.4.2** [Grammars and other rewriting systems], *Parsing*, **F.4.3** [Formal Languages], **I.2** [Artificial intelligence], **I.2.3** [Deduction and theorem proving], *Inference engine*, **I.2.6** [Learning], *Language acquisition.*

Résumé

La majorité des langages de programmation est basée sur les grammaires à contexte libre (CFG). Le but de l'inférence grammaticale est d'inférer une grammaire, en l'occurrence à contexte libre (CFG), à partir d'exemples de phrases correctes et éventuellement incorrectes, d'un langage donné. Partant de ces deux définitions fondamentales, nous proposons un environnement suivi d'une implémentation unifiant des aspects différents de la programmation dans le cadre d'apprentissage automatique. L'idée centrale du travail est donc d'utiliser l'inférence grammaticale comme trame unificatrice pour réaliser cette intégration. Dans la mesure où tout programme peut être considéré comme une suite de caractères, nous montrons que l'utilisation de l'inférence grammaticale peut non seulement unifier des aspects différents de la programmation mais aussi s'étendre à d'autres domaines plus vastes. Le travail s'articule autour des contributions suivantes :

État de l'art de la théorie des langages ; État de l'art de l'inférence grammaticale ; Étude et développement d'un environnement intégrant apprentissage et logique du premier ordre ; Étude et développement d'un système fonctionnant en logique du premier ordre agissant comme analyseur syntaxique autonome ou en collaboration avec un module d'apprentissage ; Étude et implémentation d'un algorithme à complexité polynomiale, basé sur des heuristiques et destiné à l'amélioration du processus d'apprentissage dans le cadre de l'inférence grammaticale ; Interaction avec les systèmes de commande automatique.

Le présent travail est porteur d'une ligne prometteuse de recherche, et contribue davantage à l'intégration des langages de programmation, projetant de les enrichir par la caractéristique d'apprentissage qui leur fait actuellement défaut.

Catégories et descripteurs de sujets de ACM

D.3.1 [Définitions formelles], **D.3.2** [Classifications de langages], *conception des langages*, **F.1.1** [Modèles de calcul], **F.4.2** [Grammaires et systèmes de réécriture], *analyse syntaxique*, **F.4.3** [Langages formels], **I.2** [Intelligence artificielle], **I.2.3** [Déduction et démonstration de théorèmes], *moteur d'inférence*, **I.2.6** [Apprentissage], *acquisition de langages*