University of Miskolc

Faculty of Mechanical Engineering and Informatics

Ontology-based Semantic Annotation and Knowledge Representation in a Grammar Induction System

Ph.D. Dissertation

Author:

Erika Baksáné Varga

MSc in Information Engineering

"József Hatvany" Doctoral School
of Information Science, Engineering and Technology

Research Area

Applied Computational Science

Research Group

Data and Knowledge Bases, Knowledge Intensive Systems

Head of Doctoral School:

Prof. Tibor TÓTH

Head of Research Area:

Prof. Jenő SZIGETI

Head of Research Group:

Prof. László CSER

Academic Supervisor:

Dr. László KOVÁCS

Miskolc, 2011

Declaration

The author hereby declares that this thesis has not been submitted, either in the same or different form, to this or any other university for a Ph.D. degree.
The author confirms that the work submitted is her own and the appropriate credit has been given where reference has been made to the work of others.

Nyilatkozat

Alulírott Baksáné Varga Erika kijelentem, hogy ezt a doktori értekezést magam készítettem, és abban csak a megadott forrásokat használtam fel. Minden olyan részt, amelyet szó szerint vagy azonos tartalomban, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Miskolc, 2011. október 20.

Baksáné Varga Erika

A disszertáció bírálatai és a védésről készült jegyzőkönyv megtekinthető a Miskolci Egyetem Gépészmérnöki és Informatikai Karának Dékáni hivatalában, valamint a doktori iskola weboldalán az Értekezések menüpont alatt: *http://www.hjphd.iit.uni-miskolc.hu.*

Témavezető ajánlása

Baksáné Varga Erika: "Ontology-based semantic annotation and knowledge representation in a grammar induction system" című PhD értekezéséhez

Baksáné Varga Erika a diploma megszerzése után a tanszékünkön helyezkedett el tanársegédként. Sokat dolgoztunk együtt az adatbázis kezelés területeihez tartozó tantárgyak oktatásában. Pozitív tapasztalataim alapján örömmel vállaltam el témavezetését, amikor 2003-ban jelentkezett a doktori képzésre. Témakörként olyan területet választottunk ki, mely egyrészt kapcsolódott a tanszék korábbi kutatási munkáihoz, másrészt kellően ígéretesnek mutatkozott. A számítógépes nyelvészet napjainkra valóban egy kurrens területté vált, melyen belül a statisztikus nyelvtan tanulást jelöltük ki a kutató munkához. Az első évek alatt kiderült, hogy milyen fontos szerepe van a szemantikai oldalnak, a hagyományos szintaktika orientált megközelítések nem jelentenek elég hatékony megoldást. Emiatt a kutatásban a hangsúly a szemantikai oldal felé helyeződött át, középpontba helyezve a szemantikai és a szintaktikai komponensnek kapcsolatát.

A feladat megoldásához a jelölt a mesterséges intelligencia egy napjainkban felfutó ágának, az ontológiának a lehetőségeit és módszereit használta fel és ötvözte sokoldalúan. A elvégzett vizsgálatok eredményeként széleskörű és lényegi elemzés született a kapcsolódó nemzetközi irodalomról, külön kiemelve az egyetemünkön folytatott kapcsolódó kutatások szerepét is. A kidolgozott módszerek alkalmazhatók többek között nyelvtantanuló ágensek belső motorjának részeként a hatékony nyelvtantanulás biztosításához. A dolgozat különböző részterületen elért eredményeket fog össze, melyek közül kiemelném az ECG szemantikai modell kidolgozását, a szemantikai hálókban megvalósuló tanulási folyamatok (általánosítás, fogalomképzés) modelljét, valamint a szemantikai modell és a nyelvtan fák (TAG) kapcsolatára elkészített algoritmust.

Az értekezés tézisei és a témához kapcsolódóan megjelent publikációk igazolják, hogy a jelölt sikeresen végrehajtotta a kitűzött célt. A jelölt a kutatás eredményeiről rendszeresen beszámolt angol nyelvű folyóiratokban, illetve hazai és külföldi konferenciákon, ezáltal eleget téve a Hatvany József Informatikai Tudományok Doktori Iskola publikációs követelményeinek. Az eredmények igazolják, hogy a jelölt képes színvonalas, önálló kutatómunkára, munkáját a rendszeresség és a teljességre törekvés jellemzi. Maga az értekezés gondos és szerteágazó tudományos munkát, a szakirodalom alapos feltérképezését tükrözi. Az értekezés Baksáné Varga Erika saját eredményeit tartalmazza és a Hatvany József Informatikai Tudományok Doktori Iskola által megkövetelt tartalmi és formai követelményeknek mindenben megfelel. Mindezekre tekintettel a jelölt számára a PhD cím odaítélését messzemenően támogatom.

Miskolc, 2011. október 20.

Dr. habil. Kovács László
tudományos vezető

There are no shortcuts to any place worth going.

Az olyan helyekre vezető utakat, ahová érdemes eljutni, nem lehet lerövidíteni.

Helen Keller

Acknowledgements

The dissertation exposes many years of work which I could not be able to achieve without the support of others.

First of all, I owe my deepest gratitude to **Professor Tibor Tóth**, Head of "József Hatvany" Doctoral School for giving me support and the necessary permissions to the accomplishment of the doctoral procedure.

I am heartily thankful to my supervisor, **Dr. László Kovács**, whose guidance and support from the initial to the final level enabled me to develop an understanding of the subject.

I am indebted to many of my colleagues at the Department of Information Technology for helping me in organizing the events connected to the doctoral procedure. Also, I would like to thank **Kornél Dluhi** for his work in the beginning of the software development phase, and **Almée Réti** for reviewing and correcting my English.

At last but not least, this thesis would not have been possible without the unfailing encouragement and support of my family. I am especially greatful to my husband Attila, for his invaluable help in the completion of the project, and I truly appreciate the efforts of my parents who spared no pains to create an untroubled atmosphere during my working hours.

I dedicate this work to my daughters Anna and Rozi, wishing them an unclouded childhood and power to make their dreams come true.

List of Notations

Miscellaneous

O_d, O_i	dynamic / immediate object
I_d, I_i, I_f	dynamic / immediate / final interpretant
$\mathfrak{I}$	interpretation
$\hat{I}$	interpretation function
Δ	interpretation domain
$\mathbf{n}, \mathbf{n}', \mathbf{n}'', \mathbf{m}', \mathbf{e}, \mathbf{f}', \mathbf{g}, \mathbf{h}$	semantic equivalence assignment functions

Predicate logic

t	term
f	function symbol
p	predicate symbol
α, β	formulas
x, y	variable symbols
ν	variable assignment function
$d \in \Delta$	data value
$D_O \in \Delta$	object domain
Θ	general relation symbol
$\mathcal{F}$	set of statements, formulas (in general)
$\varsigma \in \mathcal{F}$	general statement
$\mathcal{O}$	set of operations over $\mathcal{F}$
$\sigma \in \mathcal{O}$	operation
Q	general quantifier symbol

Description logic

Δ_D	interpretation domain of datatypes
dt	datatype
i	individual, instance name
C	concept name
R	role, relation name
R_D	datatype property in OWL
R_O	object property in OWL
$\top$	universal concept symbol in OWL
$\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$	DL knowledge base

$\mathcal{T}$	TBox, terminological knowledge base
$\mathcal{A}$	ABox, assertional knowledge base

Formal grammar and language

$\mathcal{G}, \mathcal{G}'$	formal grammars
$\mathcal{L}, \mathcal{L}(\mathcal{G})$	language, the language generated by $\mathcal{G}$
$\mathcal{V}$	vocabulary, set of valid sentences in $\mathcal{L}$
$\tau : \mathcal{L}_1 \rightarrow \mathcal{L}_2$	transformation function
PR	set of production rules
NT	set of nonterminal strings
$N \in NT$	nonterminal string
$\mathbf{S} \in NT$	start symbol
TS	set of terminal strings
$w \in TS$	terminal string, word
δ, η, ξ	arbitrary strings of terminals and nonterminals
ε	empty string
Σ	alphabet (character set) of a language
$\tilde{c} \in \Sigma$	character symbol
W	set of words
$w \in W$	word
$\mathcal{S}$	set of sentences
$s \in \mathcal{S}$	sentence

ECG model

U	problem domain
U_I	set of object instances
U_C	set of concepts
U_A	set of agents
Γ	environment
S	snapshot
H	history
$I \subseteq U_I,\ I_\Gamma$	set of object instances in the environment
$\iota \in U_A$	agent
$\Lambda_{\pi,\iota}$	primary knowledge model of agent ι
Λ_ι	extended knowledge model of agent ι
$\tilde{C} \subseteq U_C,\ \tilde{C}_{\Lambda_{\pi,\iota}} \subseteq \tilde{C}_{\Lambda_\iota}$	set of concepts in a knowledge model
$\chi_{\pi,\iota}$	primary conceptualization mapping
$\tilde{R}$	set of derivation rules

TAG formalism

$TAG(\mathcal{G})$	TAG grammar
$\mathcal{L}(TAG(\mathcal{G}))$	language generated by a TAG grammar
$T(I)$	set of initial trees

$T(A)$	set of auxiliary trees
$\mathbf{T}(TAG(\mathcal{G}))$	phrase-structure tree set of a TAG grammar

ECG-TAG & S-ECG-TAG formalisms

$ECG\text{-}TAG(\mathcal{G})$	ECG-TAG grammar
E, $\bar{E}$, $\tilde{E}$	sets of undirected edges
C	set of ECG concepts
$c \in C$	ECG concept
$CC \subseteq C$	set of category concepts
$PC \subseteq C$	set of predicate concepts
RS	set of ECG relationships
$RR \subseteq RS$	set of semantic role relations
$SR \subseteq RS$	set of specialization (isa) relationships
$T(S)$	single-element set of the base **S**-type tree
$T(D)$	single-element set of the derivation tree
$S\text{-}ECG\text{-}TAG(\mathcal{G})$	S-ECG-TAG grammar
SN	set of symbolic level nodes
$sn \in SN$	symbolic level node
ST	set of symbolic terms
$st \in ST$	symbolic term

ECG diagram graph

$\Gamma = \langle V, A, R \rangle$	ECG diagram
V	set of vertices
$v \in V$	vertex
A	set of directed edges (arrows)
$a \in A$	directed edge, arrow
R	set of semantic roles
$r \in R$	semantic role
$fi(a_i)$	incidence function
$d_{(\Gamma)}(v)$	number of incidence edges with vertex v
$d^{-}_{(\Gamma)}(v)$, $d^{+}_{(\Gamma)}(v)$	number of incoming/outgoing edges to/from vertex v

ECG graph matching

EC	set of element categories
$ec \in EC$	element category
T	set of element category types
$type_c \in T$	element category type
$T_{cc} \subset T$	set of category concept types
$T_{pc} \subset T$	set of predicate concept types
$T_{rr} \subset T$	set of semantic role relation types
$T_{sr} \subset T$	set of specialization relation types
EI	set of element instances

$ei \in EI$	element instance
T'	set of element instance types
$type_i \in T'$	element instance type
$[type_i]$	category type of an element instance type
$\{type_i\}$	numeric code of an element instance type
Φ	set of correlations between two element instances
$\varphi \in \Phi$	correlation between two element instances
$><$	semantic comparability relation of two instances
$<>$	semantic incomparability relation of two instances
$\sim$	category equivalence relation of two instances
$=$	equivalence relation of two instances
$\prec$	generalization relation of two instances
SC	set of semantically comparable instance pairs
SNC	set of semantically incomparable instance pairs
CTE	set of category equivalent element instance pairs
EQV	set of equivalent element instance pairs
GEN	set of instance pairs in generalization relation
Φ	set of relations between two ECG diagrams
$\phi \in \Phi$	relation between two ECG diagrams
$\bowtie$	semantic comparability of two ECG diagrams
$\lhd\rhd$	semantic incomparability of two ECG diagrams
$\equiv$	equivalence relation of two ECG diagrams
$\preceq$	generalization relation of two ECG diagrams
$\simeq$	isomorphical relation of two ECG diagrams
M	set of mapping elements (alignment)
$m \in M$	mapping element
μ	fitness value

ECG concept lattices and generalization

$UNIV$	suprenum element of a lattice
NIL	infinum element of a lattice
$lcg()$	least common generalization function
$gcs()$	greatest common specialization function
$(T, <)$	element category type lattice
$(T', \prec)$	element instance type lattice
$\gamma, \gamma^* \in \Gamma$	subgraph of an ECG diagram graph
$\mathbf{\Gamma}$	set of primary-level ECG diagram graphs
$\mathbf{\Gamma}(\mathbf{A})$	set of accumulated ECG diagram graphs
$\boldsymbol{\gamma}$	set of subgraphs resulting from graph intersection
$\cap, \cap^*$	ECG graph intersection operator and its extension

List of Abbreviations

AI	Artificial Intelligence
CFG	Context-Free Grammar
CG	Conceptual Graph (Sowa)
CSG	Context-Sensitive Grammar
DG	Dependency Grammar
DL	Description Logic
ECG	Extended Conceptual Graph (Baksáné & Kovács)
EHOPL	Extended Higher-Order Predicate Logic
FOPL	First-Order Predicate Logic
FTAG	Feature-Based Tree Adjoining Grammar
HOPL	Higher-Order Predicate Logic
IDE	Integrated Development Environment
KB	Knowledge Base
KIF	Knowledge Interchange Format
KR	Knowledge Representation
LTAG	Lexicalized Tree Adjoining Grammar
MTAG	Multicomponent Tree Adjoining Grammar
NL	Natural Language
NLI	Natural Language Interface
NLP	Natural Language Processing
OKR	Ontology Level Knowledge Representation (Muresan)
OWL	Web Ontology Language
PAC	Probably Approximately Correct (learning)
PCFG	Probabilistic Context-Free Grammar
PL	Predicate Logic
POS	Part-Of-Speech
RDF	Resource Description Framework
RDFS	RDF Schema
SDM	Semantic Data Model
STAG	Synchronous Tree Adjoining Grammar
TAG	Tree Adjoining Grammar
TKR	Text Level Knowledge Representation (Muresan)
UML	Unified Modeling Language
WFF	Well-Formed Formula

W3C	World Wide Web Consortium
XML	EXtensible Markup Language

ECG model

ECG Diagram	Graphical representation of the ECG model
ECG-HOPL	Logic-based representation of the ECG model
ECG-TAG	Semantic-level TAG formalism of the ECG model
S-ECG-TAG	ECG-TAG formalism with syntactic-level

AMR	Abstract Role Relationship
AMCR	Abstract Category Concept
AMPR	Abstract Predicate Concept
FMI	Specialization Relationship
FMR	Primary Role Class Relationship
FSR	Primary Role Single-Instance Relationship
FICN	Primary Unnamed Category Instance Concept
FICR	Primary Permanent-Named Category Instance Concept
FICT	Primary Temporary-Named Category Instance Concept
FMCR	Primary Category Concept
FMPR	Primary Predicate Concept

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The dissertation examines a possible interconnection between two apparently distant branches of artificial intelligence (AI): ontology and rule induction. Recently, ontology gains an ever-wider range of applications, mainly in areas where the use of semantic information is presumably benefitial. Statistical rule induction is devoted to find characteristic or frequent rule sets. We have an intuition that the idea of combining statistical approaches with semantics in rule learning is advantageous concerning the efficiency and accuracy of the learning algorithms, so it is worth investigating. The motivation for accommodating the research in a grammar induction framework is the fact that symbolic languages have the most complex systems of rules (grammars), therefore they must be considered when developing a general methodology for rule learning.

1.1 Preliminaries

1.1.1 Ontology and its applications

Most computers today are connected in networks to share data, information and knowledge. Due to the overwhelming amount of information that is continually being generated, effective processing and use or reuse of knowledge is essential. Therefore researchers in AI first developed ontologies to facilitate sharing, processing and reuse of knowledge.

The term 'ontology' in philosophy is the science of what is, of the kinds and structures of objects, properties and relations in every area of reality. According to a widely accepted definition for ontology in information science, "an ontology is a formal, explicit specification of a shared conceptualization" [Gruber, 1993]. In this context, *conceptualization* refers to an abstract model of some phenomenon in the world that identifies that phenomenon's relevant concepts. *Explicit* means that the type of concepts used and the constraints on their use are unambiguously defined, and *formal* means that the ontology should be machine understandable. *Shared* reflects the notion that an ontology captures consensual knowledge – that is, it is not restricted to some individual but is accepted by a group.

When two agents need to communicate or exchange information, the prerequisite is that a consensus has to be formed between them. Ontologies are specifically designed to provide

the common semantics for agent communication. In order to be able to play this important role a joint standard needs to be developed for specifying and exchanging ontologies. Therefore researches in ontology are aimed, on the one hand, at defining a standard ontology language, and on the other, at developing tools and methods for ontology design and verification, and finally at creating ontology libraries.

Since the beginning of the 1990s, ontologies have become a popular research topic, and several AI research communities – including knowledge engineering, natural language processing, and knowledge representation – have investigated them. More recently, the notion of ontology is becoming widespread in fields such as information integration and retrieval, and knowledge management. Ontologies are becoming popular largely because of what they promise: a shared and common understanding that reaches across people and application systems.

1.1.2 Rule induction

Rule induction (see [Rückert, 2008] for details) is a popular learning scheme in machine learning. This branch of AI aims to provide computational methods for improving the performance of knowledge acquisition from experimental data by discovering and exploiting regularities. Generally speaking, the task of a learner (computer program) is to induce a predictive model from the training data whose predictions are as accurate as possible while being comprehensible for humans.

In traditional rule induction, knowledge exploited is stored in condition-action symbolic structures (if-then rules). As long as the rule set is not too large and the rules are not too complicated, the result can be easily understood and analyzed by humans. For that reason, small (i.e. simple) rule sets are considered to be among the best comprehensible and interpretable representations in machine learning. A major issue concerning rule induction is the determination of the learner's complexity. Overfitting happens whenever the model induced is expressive enough to incorporate noisy or random patterns which appear in the training data, but are not present in the underlying data generation process. In order to avoid overfitting the expressive power of the rule set must be restricted. On the other hand, if the rule set induced is very simple, it may not be able to catch all important regularities and it might be characterized by bad predictive accuracy.

In typical machine learning applications the experimental data are very often imperfect or noisy, which means that the available information is generally too sparse to draw justified deductive conclusions. Hence, many machine learning systems rely on statistical and probabilistic methods, which can express the randomness and the probabilities of the events and decisions involved. Early works on statistical rule induction were aimed at finding algorithms that are provably predictive for large training sets; while the probably approximately correct (PAC) learning framework [Valiant, 1984] also considers the time complexity of computing model representations.

A different approach to statistical inference has been followed by Bayesian statistics. From this perspective, the main goal is to find an algorithm that predicts well on unseen data, that is its accuracy on the test set is maximal, or in other words, the test error is minimal. Unfortunately, at learning time neither the test data, nor the underlying distribution are known, so the error cannot be optimized directly, based on the training set. Therefore, instead of aiming for an algorithm with low test error, the original problem can be reformulated as finding an algorithm that has low true error on the training data. In practice, however, the true error can never be obtained, because the underlying distribution is unknown. Only an estimation can be given for the distribution from a (preferably large) test set. Another disappointing result is that there is no guarantee that a certain finite training set size leads to algorithms with reasonably low true error; namely the Bayes error can only be achieved in the limit.

Theoretically, the crux of machine learning is to find algorithms whose biases match well with the learning problems that are frequently encountered in practice. However, the experimental results show that rather than looking for the 'universally best' learning algorithm, one should search for an algorithm whose bias matches well with the learning problem at hand, in order to achieve good performance.

1.2 Aims and scope

The main motivation for the research is to develop a new general rule learning methodology that alloys statistics with semantics. With that, our aim is to improve the performance of statistical rule induction by utilizing semantic information in the learning process. The actual learning problem is chosen to be grammar induction, because symbolic languages have the most complex systems of rules (grammars), so they must be considered when developing a general methodology. In grammar induction, the aim is to learn the formal grammar generating the language of the input data. Accordingly, the general schema of the grammar induction system (or agent) investigated can be found in Figure 1.1.

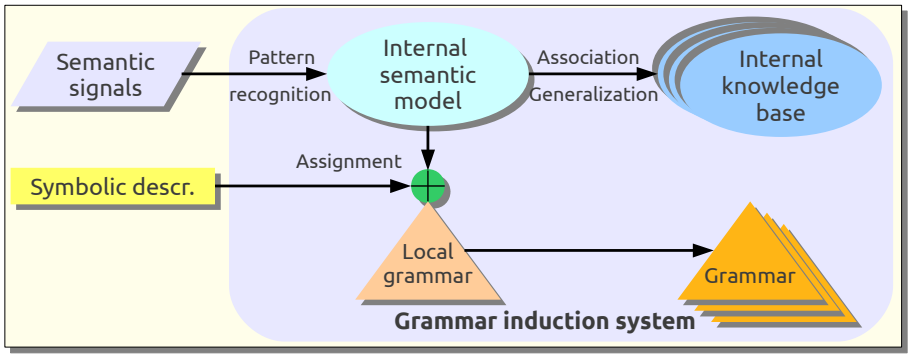

Figure 1.1 *Schematic description of the grammar induction system investigated*

The dissertation covers the first phase in the development of the system outlined in Figure 1.1, that is the specification and deep examination of an appropriate semantic representation optimized for grammar induction.

The capabilities of the grammar learning agent are fixed in advance, which are

1. pattern recognition: the ability to recognize the objects of its direct environment and their relations;
2. association: the ability of relating pieces of information based on its stored knowledge; and
3. generalization: the ability of creating abstract concepts by extracting the common characteristics of existing knowledge items.

In order to be able to achieve these tasks, the learning agent needs a semantic representation model that satisfies the following basic requirements:

- its main building blocks should be concepts and their relationships,
- it should be predicate-centered, where predicate is a type of concept,
- it should have an apriori knowledge regarding the model elements,
- it should make a clear distinction between apriori and learned elements,
- it should reflect the multi-layered nature of conceptualization, and
- it should provide high levels of flexibility and extendibility.

A semantic model satisfying the above requirements can be considered as an ontology representation language. Thus, in the system examined ontologies are in the first place used for semantic annotation, and secondly as the representation of the knowledge base of the grammar induction system.

In the field of ontology-based semantic annotation the interest of researchers has arisen only in recent years, mainly in connection with the semantic web [Berners-Lee et al., 2001] and is oriented towards the automatic creation of semantic annotation for web documents. These projects require an upper ontology that defines the taxonomy of concepts in a problem domain. The process of annotation is realized by finding matching ontology concepts and words in documents. The distinguishing feature of this project is the annotation method, namely symbolic language sentences are annotated with instance-level ontologies.

In the field of grammar induction, only one project is found that uses ontology as a repository of semantic knowledge supporting the inference of a constraint-based grammar (see Section 2.3.4 on page 26). This attempt is again an example of word-based semantic description, while the aim of the present research is to support the automatic learning of symbolic language sentence units (word sequences) and their ordering.

Therefore, based on the available literature, the present research can be considered as a novel approach to grammar induction, a novel approach to semantic annotation, and also a novel approach to the application of ontologies.

Ontologies play an important role in semantic modeling. However, at present there does not exist a general graphical format for their representation. Other researches in the literature use existing conceptual models for this purpose. The comparative analysis of these graphical semantic models also forms part of the present research in view of the question whether they are suitable for representing the knowledge base of the modeled grammar induction system. Section 2.2.4 on page 18 exposes the results of this analysis with the conclusion that all the examined existing conceptual models have some shortcomings concerning the actual task. Accordingly, the tasks to be solved during the completion of the project can be summarized in the following points.

1. The first task of the research is to develop a semantic model for graphical knowledge (ontology) representation that fulfils the requirements declared for the actual task; as well as the analysis of its expressive power.
2. The second task is to find an appropriate grammar formalism that is able to represent the semantic model developed and its symbolic language description jointly, in a common framework, in support of the training of the grammar induction system modeled.
3. The third task is to describe how the knowledge base of the grammar induction system examined builds up. This means the modeling of the process of conceptualization within the semantic model introduced.
4. Finally, a test system should be implemented for the verification of the theoretical results.

1.3 Dissertation guide

Following the tasks to be solved, the dissertation consists of the following chapters.

Chapter 2 gives a thorough introduction to the topic of the dissertation through the examination of the information processing model of agents. In this process the first stage to be studied is conceptualization, the second is knowledge representation and the third is grammar learning. Within the field of knowledge representation, special attention is paid to ontologies and the forms of their specification. This chapter also includes a comparative analysis of the applicability of existing graphical conceptual models in the present research. In connection with grammar induction, the techniques and problems of the creation and use of semantic annotation are in the focus of the discussion.

Chapter 3 exposes the development of a novel semantic model, which is designed to fulfil the declared requirements of the knowledge representation format in the grammar induction system investigated. The chapter starts with the analysis of how much expressive power is needed for the problem at hand. The result is that first-order predicate logic is too restricted, and also higher-order predicate logic needs some extensions. With these extensions, the new semantic model is defined in two forms: it has a logic-based and a graphical representation, and is named Extended Conceptual Graph (ECG).

Chapter 4 aims at finding a grammar formalism that is able to represent the semantic model developed and its symbolic description in a common framework. The new formalism introduced is an edge-labeled lexicalized tree-based representation combining the levels of semantics and syntax. The semantic level is constructed from ECG diagram graphs, where the nodes correspond to ECG concepts and the edges represent ECG relationships. At the symbolic level the nodes include the word sequences assigned to ECG concepts, while the edges are labeled by precedence relations representing the order of the word sequences in the corresponding symbolic sentence. Thus, the symbolic level encodes word order locally and discontinuous constructions are represented by sibling nodes. This formalism supports the learning of the association rules between the syntactic and semantic levels of language, therefore it makes the generation of a symbolic grammar possible.

Chapter 5 models the processes of conceptualization – association, abstraction and generalization – using ECG diagram graphs. Association is defined as the incorporation of new information elements into the knowledge base, which raises the problem of matching ECG diagram graphs. The matching problem poses several questions concerning the aspects of comparing ECG diagram graphs and their elements. In support of these comparisons, lattices are defined for storing concept generalization structures. Abstraction and generalization are implemented in one operation, embedded in the association algorithm, that is defined as the process by which new ECG concepts are created in the knowledge base incorporating frequently occuring existing concepts. The two operations (association and generalization) together accomplish the process of conceptualization, at the idealized end of which stands the generalized accumulated ECG diagram graph (representing the knowledge of the learning agent) built up from a set of primary-level ECG diagram graphs.

Chapter 6 introduces the test system implemented in which two applications of the ECG model are demonstrated: the generation of the set of semantically annotated training samples, and the simulation of the conceptualization process on this data set.

Chapter 7 summarizes the new scientific results achieved during the completion of the project. It also gives some application areas of the results, and outlines the directions of future investigations.

Chapter 2

Background

The ability of computers to process language as skillfully as we do will signal the arrival of truly intelligent machines.

D. Jurafsky and J.H. Martin: Speech and Language Processing (2000), Prentice Hall

Ontology, knowledge representation, grammar induction and agent technology are all concepts used in AI. According to a general definition, "an agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors" [Futó, 1999]. The goal of the research is to study a human agent who observes signals from the environment and after processing the information received, is able to express its perceptions with linguistic symbols (see Figure 2.1). For the examination of this information processing model the following intermediate stages should be studied: the process of conceptualization, the representation of knowledge, and its mapping to a symbolic language. This latter stage requires the knowledge of the given language: vocabulary and grammar rules for building expressions, which is the result of a learning process. In the following sections, these stages are discussed in detail.

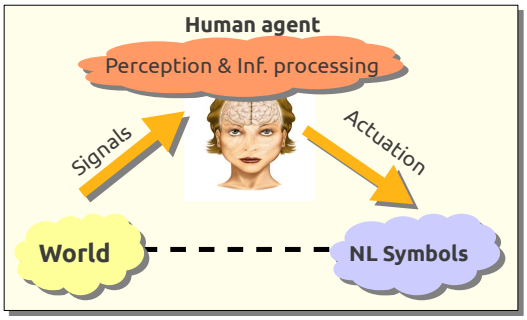

Figure 2.1 *Information processing model of human agents*

2.1 The process of conceptualization

The study of signs, called semiotics (or semiology), was developed independently by the Swiss linguist, Ferdinand de Saussure and by the American logician and philosopher Charles Sanders Peirce. The term comes from the Greek sêma (sign). Saussure saw the new science as a "science which studies the life of signs at the heart of social life".

Peirce extended this definition, and described semiotics as the science that studies the use of signs by "any scientific intelligence". By that term, he meant "any intelligence capable of learning by experience", including animal intelligence and even mindlike processes in inanimate matter [Sowa, 2000b], [Sowa, 2006]. How understanding evolves from signs of objects can be demonstrated by a triangle, which has a long history going back to as far as Aristotle. He distinguished objects, the words that refer to them, and the corresponding experiences in the psyché. Peirce (1867) adopted that three-way distinction from Aristotle and used it as the semantic foundation for his system of logic.

In **Peirce's theory** [Hartshorne et al., 1958] a sign is defined as the signifier of an object in the world, while interpretant is considered as the understanding that we have of the sign-object relation. The importance of the interpretant for Peirce is that signification is not a simple dyadic relationship between sign and object: a sign signifies only in being interpreted. This makes the interpretant central to the content of the sign, in that the meaning of a sign is manifested in the interpretation that it generates in sign users. In his final account (1906-10), Peirce found parallels between the semiotic process and the process of inquiry, which is an end-directed process. Depending on the stages of the semiotic process, he distinguished different types of objects and interpretants. Peirce made a distinction between the object of the sign as we understand it at some given point in the semiotic process (immediate object O_i), and the object of the sign as it stands at the idealized end of the process (dynamic object O_d). Therefore, the immediate object is not an additional object distinct from the dynamic object but is an informationally incomplete copy of the dynamic object generated at some interim stage in the chain of signs.

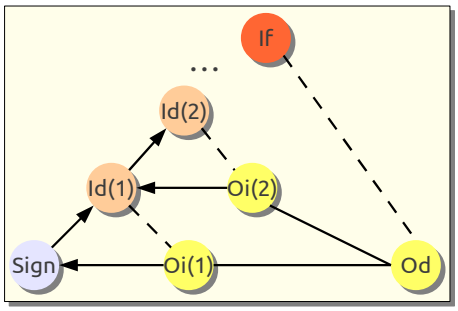

Figure 2.2 *Peirce's final account of the semiotic process [Hartshorne et al., 1958]*

At the same time, Peirce identifies three different ways in which we grasp a sign's standing for its object. The immediate interpretant I_i is a general, definitional understanding of the sign. The dynamic interpretant I_d, on the other hand, is our understanding of the sign at some actual instance in the semiotic process (the "effect actually produced on the mind"). Thus it provides an incomplete understanding of O_d, while an O_i in the sign chain consists of the dynamic interpretants from earlier stages. The final interpretant I_f, then, is what our understanding of the dynamic object would be at the end of inquiry, that is, if we were to reach a full and true understanding of the dynamic object. These three types of interpretants were introduced on the basis of the three levels of understanding (grades of clarity). That is, a full understanding of some concept involves 1) familiarity with it in

day-to-day encounters, 2) the ability to offer some general definition of it (logical analysis), and 3) knowing what effects to expect from holding that concept to be true (pragmatic analysis). Accordingly, dynamic interpretant I_d corresponds to the 1^{st} grade, immediate interpretant I_i corresponds to the 2^{nd} grade, while final interpretant I_f corresponds to the 3^{rd} grade of clarity [Atkin, 2007]. Figure 2.2 shows a detailed form of Peirce's final account of the semiotic process. The dashed lines between the interpretants and the objects reflect the implicit nature of these relations.

On the basis of Peirce's theory, **Ogden & Richards** drew their meaning triangle in 1923 [Ogden & Richards, 1923], which is a model of how linguistic symbols are related to the objects they represent. In the model the components of the process of understanding include: referents, that are the "objects that are perceived and that create the impression stored in the thought area"; symbol, that stands for "the word that calls up the referent through the mental processes of the reference"; and reference (or thought) which "indicates the realm of memory where recollections of past experiences and contexts occur". Thus, the meanings of words are determined by the past (and current) experiences of speakers who encounter these words in specific contexts. Since speakers interpret words with a background of unique experiences, each and every speaker is bound to interpret the same word in a unique and different way; that is speakers have different references for the same symbol. This definition implies that the referent of a symbol is relative to different speakers. As a consequence, in the semiotic triangle of Ogden & Richards there is no direct connection between the referent (object in the world) and the symbol. Figure 2.3 shows the difference between Peirce's and Ogden & Richards' approaches to semiotics.

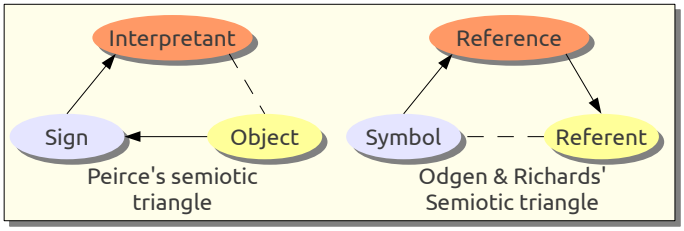

Figure 2.3 *Semiotic triangles of Peirce*[1] *and Ogden&Richards*[2]

In terms of the dissertation, the process of conceptualization stands parallel with the signal processing model of Sieber [Sieber, 2008], in which an agent is viewed as a discrete unit in the world that can act as a recipient and/or as a sender. Accordingly, each agent owns a decoding engine with sensors for constructing an internal model of the world based on the signals received, which are external data instances constituting the agent's environment. This internal knowledge model is changing over time, and can be represented by a kind of semantic network. At the same time, each agent has an encoding engine provided with actuators for transforming its internal knowledge model into signals.

[1] [Sowa, 2000b]
[2] [Ogden & Richards, 1923]

The signal processing model displayed in Figure 2.4 is taken as a basis, where the bold arrow indicates that this study concentrates on the examination and description of how agents build up their internal knowledge base (**KB**) from the signals received of the objects in their environment. The dashed lines indicate that the objects are not covered by the investigations.

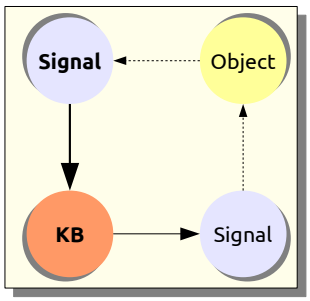

Figure 2.4 *The signal processing model of agents*

Following the multi-layer semantic data model [Kovács & Sieber, 2009], the internal knowledge base of an agent is built up in several stages; that is the process of transforming signals into concepts (conceptualization) occurs at several levels. The ability of agents to distinguish the elements of their environment (object detection) corresponds to the immediate interpretant of Peirce, which is an unanalyzed impression of input signals. The mapping of environment objects and their relations into knowledge base concepts and their relations can be viewed as the first dynamic interpretant. Then, the final interpretant, that is the true interpretation of the input signals, is constructed in several interpretation stages. As the internal knowledge model is dynamically evolving in time, in the present work Ogden & Richard's assumption – that the meaning of signals (their mapping to knowledge base concepts) is determined by the previous states of the agent's internal knowledge model – is accepted.

2.2 Knowledge representation

The next question is how to represent 'knowledge' in the knowledge base of the agent investigated. According to [Davis et al., 1993] the driving preoccupation of the field of knowledge representation (KR) should be understanding and describing the richness of the world. In this paper, the authors argue that a knowledge representation plays five distinct roles, each important to the nature of representation and its basic tasks. Thus, a knowledge representation is

1. most fundamentally a surrogate that substitutes the things in the world;
2. a set of ontological commitments;
3. a fragmentary theory of intelligent reasoning;
4. a medium for pragmatically efficient computation, i.e. the computational environment in which thinking is accomplished;

5. a medium of human expression, i.e. a language in which we say things about the world.

Later, Sowa defines knowledge representation in [Sowa, 2000a] as a multidisciplinary subject that applies theories and techniques from three other fields: 1) logic provides the formal structure and rules of inference; 2) ontology defines the kinds of things that exist in the application domain; and 3) computation supports the applications that distinguish knowledge representation from pure philosophy. For an extensive survey and historical overview on knowledge representation languages and applications, see [Jurafsky & Martin, 2000]. Also, [Jurafsky & Martin, 2000] lists the computational purposes a knowledge representation should serve.

2.2.1 Ontology as a knowledge representation model

Within the framework of the dissertation, ontology is the word used to describe a domain-specific knowledge representation model. Originally, ontology as a branch of philosophy is the science of what is, of the kinds and structures of objects, properties and relations in every area of reality. The term 'ontology' (or ontologia) was itself coined in 1613, independently by two philosophers, Rudolf Göckel (Goclenius), in his *Lexicon philosophicum* and Jacob Lorhard (Lorhardus), in his *Theatrum philosophicum.*

Formal ontology can be defined as taxonomic hierarchies of classes [Szeredi et al., 2005], or vocabularies of terms defined by human-readable text, together with sets of formal constraining axioms [Sántáné-Tóth, 2006]. In the philosophical sense, an ontology can be referred to as a particular system of categories accounting for a certain vision of the world. As such, this system does not depend on a particular language. On the other hand, in its most prevalent use in AI, an ontology refers to an engineering artifact, constituted by a specific vocabulary used to describe a certain reality, plus a set of explicit assumptions regarding the intended meaning of the vocabulary words. This set of assumptions usually has the form of a first-order logical theory, where vocabulary words appear as unary or binary predicate names, respectively, called concepts and relations. In the simplest case, an ontology describes a hierarchy of concepts related by subsumption relationships; in more sophisticated cases, suitable axioms are added in order to express other relationships between concepts and to constrain their intended interpretation [Guarino, 1998]. Practically speaking, an ontology is structured knowledge, or a logical subset of general knowledge that defines a set of domain concepts through characteristic relations.

There are several formal representations that are used for modeling ontologies, and for expressing knowledge based on the ontology. Accordingly, ontologies can be represented either in a textual format, or by a graphical representation format.

The proposed logic-based standards for storing ontologies in textual format are the Knowledge Interchange Format and the OWL web ontology language of W3C. In addition to logic-based representations there are several other formats, which include representation

languages based on logic programming such as F-logic; frame-based languages such as Ontolingua; and XML-related languages like RDF and its ontology-style specialization, the RDF Schema (RDFS).

At present, there does not exist a general graphical format for modeling ontologies. Since conceptual data schemes and ontologies share many similarities, there are proposals of using existing conceptual methodologies and tools for ontology modeling (mainly UML, see [Cranefield & Purvis, 1999], [Wang & Chan, 2001], [Xueming, 2007], [Jarrar et al., 2003]). They are examined in Section 2.2.4 on page 18 in terms of their applicability for the problem at hand.

For an extensive state-of-the-art survey on ontology representation languages, the interested reader should refer to [Bechhofer, 2002], [Calí et al., 2005] and [Scriptum, 2005]. For the present research, the standard logic-based ontology languages are studied the logical foundations of which are summarized hereinafter.

2.2.2 Logical representation models

First-order predicate logic (FOPL) is a flexible, well-understood and computationally tractable approach to knowledge representation, which uses a wholly unambiguous formal language interpreted by mathematical structures [Jurafsky & Martin, 2000]. It is a system of deduction that extends propositional logic by allowing quantification over individuals of a given domain of discourse. The syntax of FOPL is built up of a vocabulary consisting of non-logical and logical symbols over a given alphabet. The set of non-logical symbols includes function symbols with a fixed arity ≥ 0, and the collection of variable and constant symbols. The set of logical symbols comprises predicate symbols with a fixed arity ≥ 0, the Boolean connectives ($\wedge$, $\vee$, $\neg$, $\rightarrow$), and the quantifiers ($\forall$ and $\exists$). As its name implies, FOPL is organized around the notion of predicate. Predicates are symbols that refer to the relations that hold among some fixed number of objects in a given domain. Objects are represented by terms, which can be defined as constants, functions or variables. FOPL constants refer to exactly one object, and are conventionally depicted as single capitalized letters. Functions also refer to unique objects, while variables, which are normally denoted by single lower-case letters allow us to make statements about unnamed objects (free variables) and also to make statements about some/all objects in some arbitrary world being modeled (bound variables in the scope of a quantifier). Formally,

- all variable symbols are terms;
- if $t_1, \ldots, t_n$ are terms and f is a function symbol with arity n, then $f(t_1, \ldots, t_n)$ is also a term.

A statement is expressed in the form of formulas, which are defined as follows.

- If p is a predicate symbol with arity n, and $t_1, \ldots, t_n$ are terms, then $p(t_1, \ldots, t_n)$ is an atomic formula.
- If t_1 and t_2 are terms, then $t_1 = t_2$ is an atomic formula.

– If α and β are formulas then so are $\neg\alpha$, $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, and $(\alpha \rightarrow \beta)$.

– If α is a formula, and x is a variable, then both $\forall x.\alpha$ and $\exists x.\alpha$ are formulas.

– A sentence is a formula without free variables.

The syntax of FOPL defines the set of well-formed formulas (WFFs), while the semantics of FOPL determines the truth value of an arbitrary formula in a given model or interpretation (an abstract realization of a situation). Formally, an interpretation $\mathcal{I} = \langle \Delta, \hat{I} \rangle$ consists of a domain Δ and an assignment function $\hat{I}$ which assigns

– an $f^{\hat{I}}$ function with arity n to every function symbol f with arity n, where: $f^{\hat{I}} : \Delta \times \ldots \times \Delta \mapsto \Delta$, and

– a $p^{\hat{I}}$ relation with arity n to every predicate symbol p with arity n, where: $p^{\hat{I}} \subseteq \Delta \times \ldots \times \Delta$.

With the aid of interpretation an element of Δ can be assigned to every variable-free expression. Similarly, a truth value can be assigned to every sentence. For the interpretation of expressions with variables, and formulas with free variables a variable assignment function is required. This ν function assigns an element of Δ to each variable symbol x, so that $\nu(x) \in \Delta$. Given an interpretation $\mathcal{I} = \langle \Delta, \hat{I} \rangle$ and a variable assignment ν the $t^{\nu,\hat{I}}$ meaning of an arbitrary term t is defined as follows.

– If x is a variable, then $x^{\nu,\hat{I}} = \nu(x)$.

– If $t_1, \ldots, t_n$ are terms and f is a function symbol with arity n, then $f(t_1, \ldots, t_n)^{\nu,\hat{I}} = f^{\hat{I}}(t_1^{\nu,\hat{I}}, \ldots, t_n^{\nu,\hat{I}})$.

Given an interpretation $\mathcal{I} = \langle \Delta, \hat{I} \rangle$ and a variable assignment ν the truth value of an arbitrary α formula is defined as $\mathcal{I} \models_\nu \alpha$, that is the interpretation satisfies the formula. Regarding the different types of formulas this definition is the following.

– $\mathcal{I} \models_\nu p(t_1, \ldots, t_n)$ iff $\langle d_1, \ldots, d_n \rangle \in p^{\hat{I}}$ and $d_i = t_i^{\nu,\hat{I}}$.

– $\mathcal{I} \models_\nu t_1 = t_2$ iff $d_1, d_2 \in \Delta$ and for both $d_i = t_i^{\nu,\hat{I}}$ where $d_1 = d_2$.

– $\mathcal{I} \models_\nu \neg\alpha$ iff not $\mathcal{I} \models_\nu \alpha$.

– $\mathcal{I} \models_\nu \alpha \wedge \beta$ iff $\mathcal{I} \models_\nu \alpha$ and $\mathcal{I} \models_\nu \beta$.

– $\mathcal{I} \models_\nu \alpha \vee \beta$ iff $\mathcal{I} \models_\nu \alpha$ or $\mathcal{I} \models_\nu \beta$.

– $\mathcal{I} \models_\nu \alpha \rightarrow \beta$ iff not $\mathcal{I} \models_\nu \alpha$ or $\mathcal{I} \models_\nu \beta$.

– $\mathcal{I} \models_\nu \forall x.\alpha$ iff for all $d \in \Delta$ $\mathcal{I} \models_{\nu[x \mapsto d]} \alpha$.

– $\mathcal{I} \models_\nu \exists x.\alpha$ iff for some $d \in \Delta$ $\mathcal{I} \models_{\nu[x \mapsto d]} \alpha$.

Where $\nu[x \mapsto d]$ is the variable assignment which assigns $d \in \Delta$ to x, while assigning the same value to every other variable as ν does [Szeredi et al., 2005].

Description logics (DL) [Baader et al., 2003] is considered the most important knowledge representation formalism unifying and giving a logical basis to the well-known traditions of semantic networks, frame-based systems, semantic data models (SDMs) and object-oriented representations [Bognár, 2000]. It is semantically based on, and hence is a subset (a decidable fragment) of FOPL. In comparison with SDM languages, the infiniteness of the interpretation domain and the open-world assumption (i.e. if a statement cannot be proved to be true using our current knowledge, we cannot draw the conclusion that the statement is false) are the distinguishing features of DL. The DL syntax [Baader et al., 2003] contains two disjoint alphabets of symbols that are used to denote atomic concepts, designated by unary predicate symbols, and atomic roles, designated by binary predicate symbols; where the latter are used to express relationships between concepts. Terms are then built from the basic symbols using several kinds of constructors. In the syntax of DL, concept expressions are variable-free and they are given a set-theoretic interpretation: a concept is interpreted as a set of individuals while roles are interpreted as sets of pairs of individuals. DL semantics is defined by interpretations: $\mathfrak{I} = \langle \Delta, \hat{I} \rangle$, where Δ is the domain (a non-empty set) and $\hat{I}$ is an interpretation function that maps

- a concept name C to $C^{\hat{I}} \subset \Delta$,
- a role name R to a binary relation $R^{\hat{I}} \subset \Delta \times \Delta$, and
- an individual name i to $i^{\hat{I}} \in \Delta$.

This interpretation function extends to concept expressions in an obvious way, where C, C_1 and C_2 are concept symbols, i, i_1 and i_2 are individual names, while nR denotes cardinality restrictions on binary relations.

- $(C_1 \sqcup C_2)^{\hat{I}} = C_1^{\hat{I}} \cup C_2^{\hat{I}}$.
- $(C_1 \sqcap C_2)^{\hat{I}} = C_1^{\hat{I}} \cap C_2^{\hat{I}}$.
- $(\neg C)^{\hat{I}} = \Delta \setminus C^{\hat{I}}$.
- $\{i\}^{\hat{I}} = \{i^{\hat{I}}\}$.
- $(\exists R.C)^{\hat{I}} = \{i_1 \mid \exists i_2 \ \ \langle i_1, i_2 \rangle \in R^{\hat{I}} \wedge i_2 \in C^{\hat{I}}\}$.
- $(\forall R.C)^{\hat{I}} = \{i_1 \mid \forall i_2 \ \ \langle i_1, i_2 \rangle \in R^{\hat{I}} \Rightarrow i_2 \in C^{\hat{I}}\}$.
- $(\leqslant nR)^{\hat{I}} = \{i_1 \mid \ \mid \{i_2 \mid \langle i_1, i_2 \rangle \in R^{\hat{I}}\} \mid \leqslant n\}$.
- $(\geqslant nR)^{\hat{I}} = \{i_1 \mid \ \mid \{i_2 \mid \langle i_1, i_2 \rangle \in R^{\hat{I}}\} \mid \geqslant n\}$.

Within a general knowledge base there is a clear distinction between intensional knowledge, or general knowledge about the problem domain, and extensional knowledge, which is specific to a particular problem. Analogously, a DL knowledge base comprises two components: a TBox and an ABox. The TBox contains intensional knowledge in the form of a terminology with subsumption relationships between the concepts. The ABox contains extensional knowledge (also called assertional knowledge) about the domain of discourse, that is assertions about individuals.

2.2.3 Logic-based standard ontology languages

A proposed standard for storing ontologies in textual format is the Knowledge Interchange Format, which is based on first-order logic. Another widely-spread ontology language is OWL, the standard web ontology language of W3C, which is based on description logics and is a revision of the DAML+OIL web ontology language.

Knowledge Interchange Format (KIF) [Genesereth, 1998] is a language designed for use in the interchange of knowledge among disparate computer systems, but is not intended as an internal representation for knowledge. Rather, it provides for the representation of knowledge about knowledge. Its language is logically comprehensive, i.e. it provides for the expression of arbitrary logical sentences; and has declarative semantics, so there is no need for an interpreter to understand the meaning of expressions. Although KIF is a highly expressive language, its main disadvantages are that 1) it complicates the job of building fully conforming systems, and 2) the resulting systems tend to be heavyweight, i.e. they are larger and in some cases less efficient than systems that employ more restricted languages.

The grammatically legal expressions of KIF are formed from lexemes, which are built up of characters. There are three disjoint types of expressions in the language: terms, sentences, and definitions. Terms are used to denote objects in the world being described; sentences are used to express facts about the world; and definitions are used to define constants. Definitions and sentences are called forms, and a knowledge base is a finite set of forms. There are all together nine types of terms, six types of sentences and three types of definitions in KIF.

The basis for the semantics of KIF is a conceptualization of the world in terms of objects and relations among those objects. A universe of discourse is the set of all objects presumed or hypothesized to exist in the world. Relationships among objects take the form of relations. Formally, a relation is defined as an arbitrary set of finite lists of objects. A function is a special kind of relation. For every finite sequence of objects (called the arguments), a function associates a unique object (called the value). More formally, a function is defined as a set of finite lists of objects, one for each combination of possible arguments. In each list, the initial elements are the arguments, and the final element is the value.

Web Ontology Language (OWL) [Bechhofer et al., 2004] is a W3C standard family of knowledge representation languages for authoring ontologies. It can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. Thus, an OWL ontology consists of a set of axioms which place constraints on sets of individuals (called classes) and the types of relationships permitted between them (see Table 2.1 for the list of OWL class constructors and axioms). It applies an open world assumption, but no unique name assumption.

OWL provides three increasingly expressive sublanguages. OWL-Lite supports a classification hierarchy and simple constraints. OWL-DL, which is based on $\mathcal{SHIQ}$ description logic, supports maximum expressiveness while retaining computational completeness and decidability. It includes all OWL language constructs, but they can be used only under certain restrictions. OWL-Full, which is a union of OWL syntax and RDF, allows for maximum expressiveness without computational guarantees.

Table 2.1 *OWL class constructors and axioms*

Class constructors		Axioms	
Constructor	DL syntax	Axiom	DL syntax
intersectionOf	$C_1 \sqcap \cdots \sqcap C_n$	subClassOf	$C_1 \sqsubseteq C_2$
unionOf	$C_1 \sqcup \cdots \sqcup C_n$	equivalentClass	$C_1 \equiv C_2$
complementOf	$\neg C$	disjointWith	$C_1 \sqsubseteq \neg C_2$
oneOf	$\{i_1\} \sqcup \cdots \sqcup \{i_n\}$	sameIndividualAs	$\{i_1\} \equiv \{i_2\}$
allValuesFrom	$\forall R.C$	differentFrom	$\{i_1\} \sqsubseteq \neg\{i_2\}$
someValuesFrom	$\exists R.C$	subPropertyOf	$R_1 \sqsubseteq R_2$
maxCardinality	$\geqslant nR$	equivalentProperty	$R_1 \equiv R_2$
minCardinality	$\leqslant nR$	inverseOf	$R_1 \equiv R_2^-$
		transitiveProperty	$R^+ \sqsubseteq R$
		functionalProperty	$\top \sqsubseteqslant 1R$
		inverseFunctionalProperty	$\top \sqsubseteqslant 1R^-$

A first question is always to consider which sublanguage to use in ontology development. The choice between OWL-Lite and OWL-DL depends on the extent to which users require the more expressive constructs provided by OWL-DL; while the choice between OWL-DL and OWL-Full depends on the extent to which users require the meta-modeling facilities of RDF Schema. For the problem at hand maximum expressiveness is needed with computational effectiveness, therefore OWL-DL is chosen, which benefits from many years of DL research:

- it has a well-defined semantics,
- its formal properties are well understood (complexity, decidability), and
- there are known algorithms and highly optimized implemented systems for authoring and reasoning.

OWL-DL is equivalent to $\mathcal{SHOIN}(D)$ description logic, where

- $\mathcal{S} \equiv \mathcal{ALC}_{\mathcal{R}+}$, i.e. $\mathcal{ALC}$ description logic extended with transitive roles;
- $\mathcal{H}$: role hierarchies;
- $\mathcal{O}$: instance concepts (objects);
- $\mathcal{I}$: inverted roles;
- $\mathcal{N}$: cardinality restrictions;
- $\mathcal{Q}$: qualified cardinality restrictions;
- (D): datatypes.

OWL supports XML Schema primitive datatypes, and there is a clear distinction between object classes and datatypes. There is a disjoint interpretation domain Δ_D for datatypes, so that

- for a datatype dt, $dt^{\hat{I}} \subseteq \Delta_D$;
- and $\Delta_D \cap \Delta = \emptyset$.

Also, there exist disjoint object and datatype properties, so that

- for a datatype property R_D, $R_D^{\hat{I}} \subseteq \Delta \times \Delta_D$;
- for an object property R_O and datatype property R_D, $R_O^{\hat{I}} \cap R_D^{\hat{I}} = \emptyset$.

An OWL ontology can be mapped to a DL knowledge base: $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ (Tbox) is a set of axioms of the form:

- $C_1 \sqsubseteq C_2$ (concept inclusion),
- $C_1 \equiv C_2$ (concept equivalence),
- $R_1 \sqsubseteq R_2$ (role inclusion),
- $R_1 \equiv R_2$ (role equivalence),
- $R^+ \sqsubseteq R$ (role transitivity),

and $\mathcal{A}$ (Abox) is a set of axioms of the form:

- $i \in C$ (concept instantiation),
- $\langle i_1, i_2 \rangle \in R$ (role instantiation).

An interpretation $\mathcal{I}$ satisfies (models) an axiom (denoted by $\mathcal{I} \models$):

- $\mathcal{I} \models C_1 \sqsubseteq C_2$ iff $C_1^{\hat{I}} \subseteq C_2^{\hat{I}}$.
- $\mathcal{I} \models C_1 \equiv C_2$ iff $C_1^{\hat{I}} = C_2^{\hat{I}}$.
- $\mathcal{I} \models R_1 \sqsubseteq R_2$ iff $R_1^{\hat{I}} \subseteq R_2^{\hat{I}}$.
- $\mathcal{I} \models R_1 \equiv R_2$ iff $R_1^{\hat{I}} = R_2^{\hat{I}}$.
- $\mathcal{I} \models R^+ \sqsubseteq R$ iff $(R^{\hat{I}})^+ \subseteq R^{\hat{I}}$.
- $\mathcal{I} \models i \in C$ iff $i^{\hat{I}} \in C^{\hat{I}}$.
- $\mathcal{I} \models \langle i_1, i_2 \rangle \in R$ iff $\langle i_1^{\hat{I}}, i_2^{\hat{I}} \rangle \in R^{\hat{I}}$.

An interpretation $\mathcal{I}$ satisfies a Tbox $\mathcal{T}$ ($\mathcal{I} \models \mathcal{T}$) iff $\mathcal{I}$ satisfies every axiom in $\mathcal{T}$. Similarly, $\mathcal{I}$ satisfies an Abox $\mathcal{A}$ ($\mathcal{I} \models \mathcal{A}$) iff $\mathcal{I}$ satisfies every axiom in $\mathcal{A}$. Consequently, $\mathcal{I}$ satisfies a knowledge base $\mathcal{K}$ ($\mathcal{I} \models \mathcal{K}$) iff $\mathcal{I}$ satisfies both $\mathcal{T}$ and $\mathcal{A}$ [Szeredi et al., 2005].

The OWL language is defined in terms of an abstract syntax [Patel-Schneider et al., 2004]; and OWL ontologies are most commonly serialized using RDF/XML syntax (but frame-based and functional syntax are also defined). The absence of visual syntax motivated several proposals to use software engineering techniques (especially UML) in the ontology development process for graphical representation.

2.2.4 Graphical representation models

Graph-based models play an important role in modeling due to their intuitive nature. Visual languages for knowledge representation are examined deeply in [Kremer, 1998]. To see the differences between the underlying ideas and capabilities of some existing conceptual models, consider the following example. Given the investigated grammar learning agent, in its direct environment a black circle is located in a white triangle. Let us assume that the agent has the ability to detect the individual objects (shapes) of the environment together with their color attributes, and is able to recognize the binary relation of inclusion between them (immediate interpretant). This situation can be phrased as "A black circle is in a white triangle".

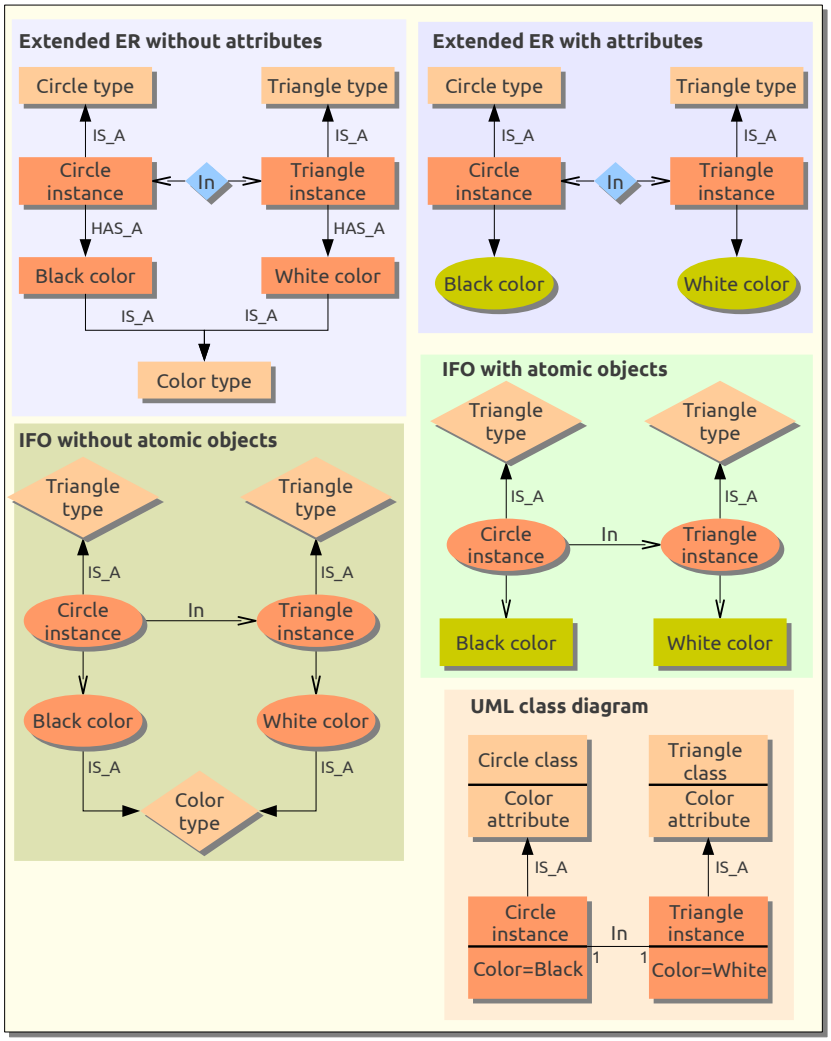

Figure 2.5 *SDM models for "A black circle is in a white triangle"*

Semantic data models (SDMs) provide a data-oriented description of the world 'in by means of a simple graphical tool set which is close to the human way of thinking (see [Kovács, 2004]). They focus on grasping the inner structure of objects. Instances sharing commonalities are grouped under general concept types. An agent provided with an SDM would therefore create a 'circle type' and a 'triangle type' with a color attribute

in its mind. SDM models for the given example can be found in Figure 2.5. The general drawbacks of applying an SDM for the present knowledge representation task are 1) the sharp distinction between concept types and concept instances, 2) the lack of attribute value representation (except for UML), and 3) the ambiguous representation of relations: partly because they can be viewed as distinct conceptual units, and partly because the role of the participants cannot be specified.

Semantic networks were developed after the work of Quillian [Quillian, 1968], with the goal of characterizing knowledge by means of network-shaped cognitive structures. Here, concepts are represented as nodes in a graph and the binary semantic relations between the concepts are represented by named and directed edges between the nodes. There are several different types of semantic network implementations (see [Sowa, 1991] and [Sowa, 1992]), varying in the kind of relation they emphasize. What is common to all semantic networks is a declarative graphical representation that can be used either to represent knowledge or to support automated systems for reasoning about knowledge. Among the numerous variants, assertional (also called propositional) networks are of interest for the purposes of the present research, some of which have been proposed as models of the conceptual structures underlying natural language semantics. As their name implies, they are designed to assert propositions. From this group of semantic networks RDF graphs and Conceptual Graphs are selected for deeper analysis.

RDF graph [Klyne & Carroll, 2004] is a significant example of semantic networks. An RDF graph is a set of triples, each consisting of a subject, a predicate and an object. Each triple represents a statement of a relationship (predicate) between the concepts (subject and object) denoted by the nodes that are connected by a directed link (pointing to the object) (see Figure 2.6).

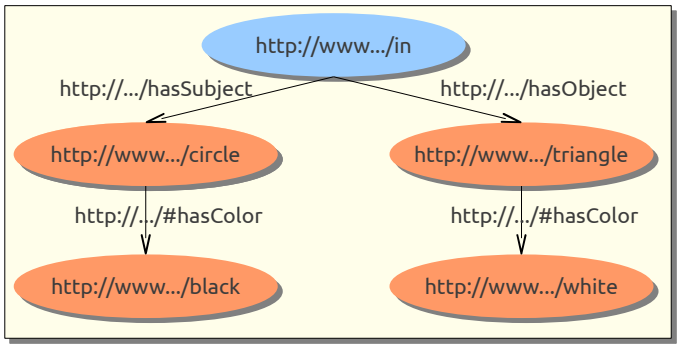

Figure 2.6 *RDF representation of "A black circle is in a white triangle"*

The disadvantages of the RDF approach from the point of view of the present research are the following: 1) predicates are resources themselves, as a consequence of which their distinction from concept resources is not evident; 2) all components are uniformly handled unique resources with unique identifiers; and 3) predicates are restricted to connecting two concept resources. In practice, the need for representing n-ary relations cannot be avoided. RDF provides reification [Hayes, 2004] as a solution. Reification enables us to make the

stating of an RDF triple a distinct resource, then further information can be added about this resource. In n-ary relations, however, additional arguments in the relation do not usually characterize the statement but rather provide additional information about the relation instance (predicate) itself. Consider, for example that the black circle is in the middle (not in the upper corner) of the white triangle. This fact adds further information to the inclusion relation and not to the overall statement. This situation cannot be naturally handled within the RDF framework.

Conceptual Graph (CG) is a logical formalism that includes classes, relations, individuals and quantifiers [Sowa, 1976], [Sowa, 1984]. This formalism is based on semantic networks, and possesses both a graphical representation, called display format and a textual representation, called linear format. In its graphical notation, a conceptual graph is a bipartite directed graph where instances of concept types are displayed as rectangles and conceptual relations are displayed as ellipses (the set of which corresponds to thematic roles [Fillmore, 1968] in linguistics). Directed edges then link these vertices and denote the existence and orientation of the relation. From a linguistic point of view, "conceptual relations link the concept of a verb to the concepts of the participants in the occurrent expressed by the verb" [Sowa, 2000a]. As a consequence of its strong relation to linguistics, concept types can take part in several conceptual relations. The only restrictive factors are human language and understanding.

From the point of view of the present research, the main drawback of CGs is their linguistic approach. Namely, a CG model's appearance depends on the phrasing of the statement's predicate. That is, if we take our example of "A black circle is in a white triangle" and we replace the predicate with the verb "include" so that "A black circle is included in a white triangle", the resulting CGs will be distinct because of the different conceptual relations (see Figure 2.7). In other words, two semantically equivalent statements yield different CG graphs due to their surface (syntactic) differences.

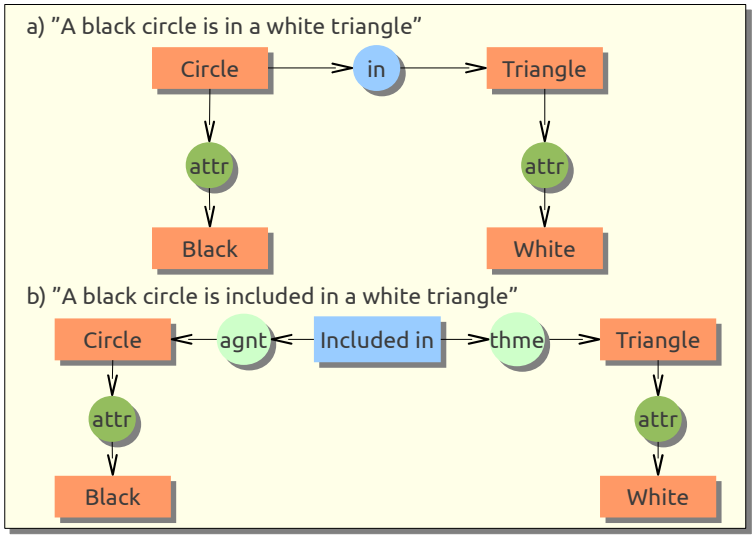

Figure 2.7 *CG representations of "A black circle is in a white triangle"*

Frame-based systems use entities like frames and their properties as a modeling primitive. The notion of frame was originally introduced by Minsky. According to his definition, a frame is a data structure for representing a concept, which can be unique or generic. In [Minsky, 1975] Minsky proposed a knowledge representation scheme that used frames for organizing knowledge. Frames, following the object-oriented approach, are supposed to capture the essence of concepts or stereotypical situations by clustering all relevant information for these situations together. Collections of such frames are to be organized in frame systems in which the frames are interconnected by means of slots. The power of frame theory lies in the inclusion of presumptions: its essence is that there are stored frame structures (default and observed) which can be recalled and adopted to the actual observations by changing the details as necessary. The disadvantage of frames is the ambiguous representation of relations, which can be connecting slots or distinct frames (see Figure 2.8).

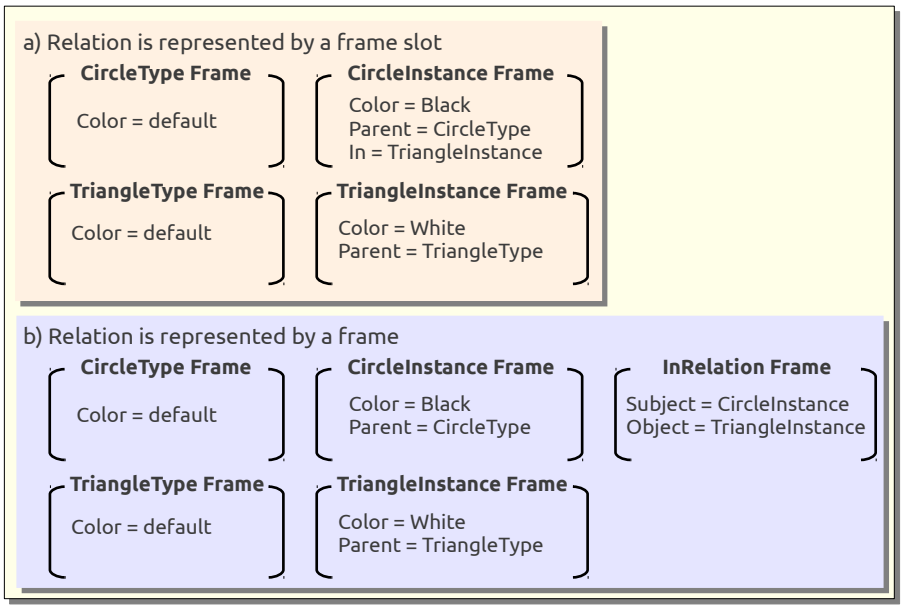

Figure 2.8 *Representation of "A black circle is in a white triangle" with frames*

2.2.5 Evaluation of graphical representation models

For representing the knowledge base of the investigated grammar learning agent a semantic model is required, which supports a formal specification that enables the mapping of semantics into a symbolic representation of the conceptualization process. The capabilities of the agent are fixed in advance, which are

1. pattern recognition: the ability to recognize the objects of its direct environment and their relations;
2. association: the ability of relating pieces of information based on its stored knowledge; and
3. generalization: the ability of creating abstract concepts by extracting the common characteristics of existing knowledge items.

In order to be able to achieve these tasks, the agent needs a representation model that satisfies the following basic requirements:

- its main building blocks should be concepts and their relationships,
- it should be predicate-centered, where predicate is a type of concept,
- it should have an apriori knowledge regarding the model elements,
- it should make a clear distinction between apriori and learned elements,
- it should reflect the multi-layered nature of conceptualization, and
- it should provide high levels of flexibility and extendibility.

Accordingly, semantic data models are not adequate for the present task most importantly because the roles the entities are playing in a relationship cannot be specified, which means that they are not predicate-centered and they have no separate representation forms for apriori and learned elements.A frame-based system is an appropriate representation form for a frame-based ontology language. However, the ontology language is chosen to be logic-based for the graphical representation of which a semantic network would be a better choice.Even so, RDF does not fully satisfy the declared requirements because it does not differentiate between predicates and other concepts, and it has no separate representation forms for apriori and learned elements. The problem of representing n-ary relations also underlies its disability.CG – with some extensions – would be a better candidate for the actual task if it were not so strongly connected to the syntactic level of language. This connection implies that semantic analysis must be preceded by syntactic analysis in natural language processing (NLP) tasks.

2.3 Grammar learning

2.3.1 Formal grammars and languages

In Encyclopaedia Britannica, grammar is defined as the "rules of a language governing its phonology, morphology, syntax, and semantics". A formal grammar, on the other hand, is a set of formation rules that describe which strings formed from the alphabet of a language are syntactically valid within the language, without describing anything else about the language. Thus, sentences that can be derived by a formal grammar are in the language defined by that grammar, and are called grammatical sentences. Sentences that cannot be derived by the grammar are not in the language defined by that grammar, and are referred to as ungrammatical. The hard line between *in* and *out* of a language characterizes all formal languages, but is only a very simplified model of how natural languages really work. This is because determining whether a given sentence is part of a given natural language often depends on the context [Jurafsky & Martin, 2000].

A formal grammar $\mathcal{G}$ is a finite formal description that generates a language $\mathcal{L}$ over some finite vocabulary $\mathcal{V}$; i.e. it defines the set of valid sentences in $\mathcal{L}$. In other words, grammars are language definition meta-languages. A grammar is a 4-tuple: $\mathcal{G} = \langle NT, TS, PR, \mathbf{S} \rangle$,

where NT is the finite set of nonterminals (or variables, denoted by single uppercase letters); TS is the finite set of terminals (denoted by single lowercase letters); PR is the finite set of production rules; and $\mathbf{S} \in NT$ is the start symbol. The general assumptions are made that $\mathcal{V} = TS \cup NT$ and $TS \cap NT = \emptyset$. Greek letters denote arbitrary strings over the vocabulary $\mathcal{V}$. $\mathcal{L}(\mathcal{G})$ is the language generated by $\mathcal{G}$. Thus $\mathcal{L}(\mathcal{G})$ is the set of all possible terminal strings that can be derived by starting at $\mathbf{S}$ and repeatedly applying production rules; i.e. $\mathcal{L}(G) = \{w \in TS \cup \{\varepsilon\} \mid \mathbf{S} \overset{*}{\Rightarrow} w\}$, where $\overset{*}{\Rightarrow}$ means to derive in zero-or-more steps and ε is the empty string [Harrison, 1978], [Bach, 2004].

A wide range of grammar formalisms proposed for natural language processing were developed with the idea that the formalism itself should characterize the class of formal languages natural languages belong to. Many formalisms, however, are much more powerful than necessary for modeling natural languages. In [Chomsky, 1956] Chomsky introduced four types of formal grammars, known as the Chomsky hierarchy, in terms of their generative power. Here, the distinction between languages can be seen by examining the structure of the production rules of their corresponding grammars.

0. Recursively enumerable or **unrestricted grammars** – including all formal grammars – with no restrictions on the form that rules can take; i.e. the rules have the form $\delta \to \eta$, where δ and η are arbitrary strings over the vocabulary $\mathcal{V}$ and $\delta \neq \varepsilon$, where ε is the empty string.

1. **Context-sensitive grammars** (CSG) with the restriction that the right-hand side of the production must contain at least as many symbols as the left-hand side. Formally, these grammars have rules of the form $\delta N \eta \to \delta \xi \eta$, where $N \in NT$, $\delta, \eta \in \mathcal{V}$, $\delta, \eta \neq \varepsilon$ and $\xi \in \mathcal{V} \cup \{\varepsilon\}$; or of the form $\mathbf{S} \to \varepsilon$ as long as $\mathbf{S}$ is not on the right side of a production.

2. **Context-free grammars** (CFG) are defined by rules of the form $N \to \delta$, where $N \in NT$ is a nonterminal and $\delta \in \mathcal{V} \cup \{\varepsilon\}$ is a string of terminals and nonterminals; or of the form $\mathbf{S} \to \varepsilon$ as long as $\mathbf{S}$ is not on the right side of a production.

3. **Regular grammars**. There are two kinds of regular grammar: right-linear (right-regular), with rules of the form $N_1 \to wN_2$ or $N_1 \to w$; and left-linear (left-regular), with rules of the form $N_1 \to N_2 w$ or $N_1 \to w$, where $N_1, N_2 \in NT$ and $w \in TS$. A rule of the form $\mathbf{S} \to \varepsilon$ is also allowed as long as $\mathbf{S}$ is not on the right side of a production.

Formal grammars and languages are ordered within a subset relationship along a scale of restrictiveness: Type 3 $\subset$ Type 2 $\subset$ Type 1 $\subset$ Type 0. To which class a language belongs is determined by the simplest grammar that can generate the language. Thus, for a given language two questions must be examined: 1) it must be determined to which class the grammar belongs that generates the language, and 2) it must be shown that there is no other grammar generating the same language that belongs to a simpler class.

A hotly contested issue over several decades has been the question where natural languages are located within this hierarchy. Chomsky showed in [Chomsky, 1957] that natural languages are not regular and he also presumed that natural languages are not entirely context-free. Later, [Shieber, 1985] proved that there are context-sensitive constructions in natural languages that cannot be adequately described with a CFG. Thus, how much more power beyond CFG is necessary to describe these phenomena is an important question. Linguists have found that while the context-sensitive languages seem to be a large enough class to describe natural languages, they also seem to contain a much bigger class of formal languages than that. Even worse, they can take up to exponential time to simulate on ordinary computers, making them totally unworkable for practical use. All available evidence suggests that a very cautious extension of CFG is sufficient to accommodate all linguistic phenomena. The first approach is to extend CFG with transformations or feature structures. The second approach is to replace CFG. Joshi proposed in [Joshi, 1985] that the class of grammars that is necessary for describing natural languages might be characterized as mildly context-sensitive grammars. This class of grammars is located between the classes of CFG and CSG grammars. Most researchers today propose no wider grammar class for the handling of natural languages, but there are some who are still not satisfied with this answer and keep on searching for methods proving that natural languages can be described by even stricter grammars.

2.3.2 Grammar induction

Grammar induction refers to the process of learning grammars from language data, and is a particular instance of inductive learning. The general aim of statistical grammar induction is to find a grammar that has the highest probability, given a set of training data. Early works on grammar induction emphasized heuristic structure search, where the primary induction is done by incrementally adding new production rules to an initially empty grammar. In the early 1990s, attempts were made to do grammar induction by parameter search, where the broad structure of the grammar is fixed in advance and only parameters are induced.

CFG plays a dominating role in grammar induction as it has reasonable complexity concerning the cost of generation and learning. According to [Jurafsky & Martin, 2000], CFG is the backbone of many models of natural and computer language syntax. It is powerful enough to express sophisticated relations among the words in a sentence, yet computationally tractable enough that efficient algorithms exist for parsing sentences with it. Although natural languages are of higher complexity and expressive power, it is widely assumed that practically CFG is an acceptable approximation of natural language. A proof of this is that very often natural language interfaces for computer applications apply some controlled language based on some kind of CFG. Concerning its learnability, [Gold, 1967] publishes the disappointing result that it is impossible to identify any of the four classes of languages in the Chomsky hierarchy in the limit if the training data consists only of strings in the language being inferred. If only positive examples are given, then only the finite cardinality languages are learnable. It has been proved that deterministic finite

state languages are the largest class that can be efficiently learned by provably converging algorithms; and there is no context-free grammatical inference theory which provably converges, if the language defined by the grammar is infinite [Jaworski & Unold, 2007]. Therefore, building algorithms that learn CFGs is one of the open and crucial problems of grammar induction. The most evident approaches taken have been to provide learning algorithms with more helpful information, such as negative examples or structural information (called annotation).

2.3.3 Annotation techniques

Existing grammar learning methods can be grouped as supervised or unsupervised, depending on the type of training data. Supervised methods learn from labeled (annotated) data, while unsupervised methods can learn from plain text as well. Due to the difficulty of the task, the majority of statistical grammar induction methods have focused on supervised learning [Charniak, 1996], [Manning & Schütze, 1999]. In practice, supervised methods (see [McEnery et al., 2005] for an overview) generate better results, since they can adapt their output to the structured examples from the initialization phase, whereas unsupervised methods do not have any idea what the output should look like. Despite this hurdle the natural language learning community has witnessed rapid advances in unsupervised grammar induction, mainly because of the lack of labeled data sets in many languages (for a review see [Clark, 2001], [Roberts & Atwell, 2002] and [1], [2]). [11] documents an attempt to induce probabilistic context-free grammar (PCFG) from an unlabeled positive data set.

By definition, linguistic annotation covers any descriptive or analytic notations applied to raw language data. Their role is to add information about the linguistic form. A collection of texts with linguistic annotations is known as a corpus. In practice, syntactic and semantic annotation schemes are distinguished. Syntactic annotation [Atwell et al., 2000] is realized by either adding a part-of-speech (POS) tag to each word in a text (indicating the grammatical role the word plays in its sentence), or determining each word's dependency from the head (typically the main verb) of its sentence. Semantic annotation of texts can also be performed in two ways [Reeve & Han, 2005]. Either, the semantic role of a word is determined in its sentence, or each word is annotated on the basis of a given (usually domain specific) ontology.

Supervised grammar induction methods are usually based on grammatically annotated corpora. For an overview of grammatical annotation schemes, see [Atwell et al., 2000]. The main problems with this kind of annotation are the following:

- a wide variety of annotation schemes exists,
- creation of annotated corpora is very expensive and time consuming,
- limited availability of annotated corpora,
- the language of most existing corpora is English,
- existing corpora are domain specific,

- human-tagged corpora are subject to inconsistencies,
- machine-tagged corpora are subject to errors.

Ontology-based semantic annotation can be accomplished manually using authoring tools which provide an integrated environment for simultaneously authoring and annotating text; or semi-automatically. The fully automatic creation of semantic annotations is not possible, because natural language is semantically ambiguous. For an extensive survey on semantic annotation platforms, see [Reeve & Han, 2005]. Characteristics of this kind of annotation are the following:

- there is no standard for ontology (annotation scheme) creation,
- annotation is expensive and time consuming,
- limited availability of ontologies,
- the language of most existing ontologies is English,
- existing ontologies are domain specific,
- manual annotation is often fraught with inconsistencies and errors,
- automated annotation has the potential benefits of consistently applying ontologies, and using multiple ontologies to annotate a single document.

In the present research, the primary information source of the agent is the visual representation of its environment. These snapshot images are associated with symbolic language sentences. The basic assumption is that the semantic contents of the images and the corresponding descriptive sentences are the same, that are represented by instance-level ontologies. The task of the agent is to learn the grammar of the given language using the semantic information stored in the ontologies. Thus, in terms of grammar induction, the problem can be phrased as learning grammar from symbolic language sentences augmented with ontologies carrying semantic information. The distinguishing features of the semantic annotation scheme proposed in the dissertation are the following:

- instance-level ontology-based semantic annotation is created automatically,
- assignment of annotation is sentence-based.
- the language of the ontology is indifferent.

2.3.4 Related work

In [Muresan, 2006] the author defines a new type of constraint-based grammar, which allows deep language understanding and is learnable. The grammar models both syntax and semantics, and has constraints at the rule level for semantic composition and semantic interpretation, which is ontology-based. The thesis also proposes a new relational learning model, where the learner is presented with a small set of positive representative examples consisting of utterances paired with their semantic representations. Ontology is used as a repository of meanings, which are connected to the grammar through a set of constraints. The focus on ontology is motivated by the need to interpret language in terms of the language-independent concepts of some underlying domain of discourse.

Similarities with the approach of the present research are:

- semantic interpretation is ontology-based,
- ontology-level knowledge representation is realized by a directed acyclic graph where nodes are concepts (or instances of concepts) and edges are semantic roles,
- the training samples are positive representative examples consisting of utterances paired with their semantic representations,
- the basic criteria; i.e. expressive power of semantic representation, computational tractability and learnability of the grammar.

Characteristics of Muresan's work are:

- the grammar is constraint-based and models both syntax and semantics,
- the learner has apriori knowledge about utterance categories (syntactic and functional) and agreement among categories,
- the learner is presented with a representative sublanguage for generalization,
- the use of a robust parser in the first phase of grammar induction,
- a natural language string is represented by a flat semantic molecule, which is a type of feature structure denoted by $\binom{h}{b}$, where h (head) encodes the information required for composition while b (body) is the actual semantic representation of the string; consequently semantic representations are linked to grammatical information,
- the ontology is frame-based,
- the semantic representation can be considered as an ontology query language,
- the semantic representation is appropriate for semantic role labeling.

Distinguishing features of the present research are:

- the agent has no apriori syntactic knowledge,
- the agent is augmented with cognitive abilities, such as pattern recognition, association and generalization,
- the use of statistical methods in grammar learning,
- sentence-level assignment of semantic and syntactic representations,
- semantic representation is ontology-based: a graphical model is also developed for the logic-based textual format.

Figure 2.9 shows Muresan's ontology level (OKR) and text level (TKR) representations for our example, where each concept (denoted by #) is represented by a semantic molecule. Thus, semantic molecules correspond to terminal symbols (words or lexical items). In order to establish this correspondence, a set of elementary semantic molecule templates are considered that correspond to lexical categories (parts-of-speech). Within Muresan's learning framework, the lexicon and these templates are apriori given as background knowledge.

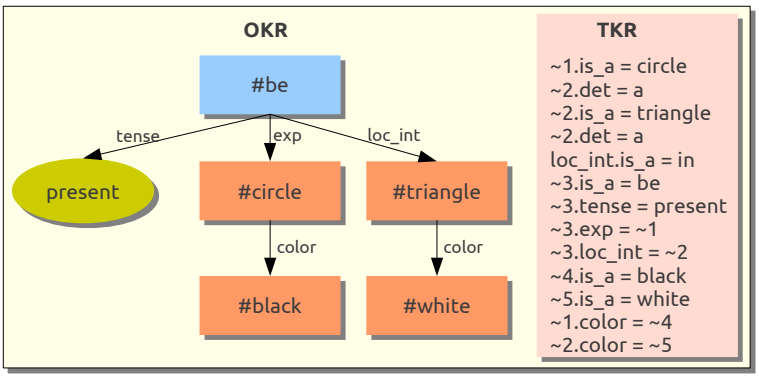

Figure 2.9 *Muresan's representation of "A black circle is in a white triangle"*

2.4 Conclusions

The main objective of the research is to develop a new general rule learning methodology that alloys statistics with semantics in order to improve the performance of statistical rule induction. In the literature, only one project has been found with the same motivation but with a different solution. This project proves the initial assumption that the use of semantic information in statistical rule learning is benefitial concerning the efficiency and accuracy of the learning algorithm.
Within the framework of the present research, semantic information is represented by instance-level ontologies. For the ontology, a logic-based representation has been chosen because of its mathematical foundations. OWL-DL is preferred over KIF, because KIF is not primarily designed for being an internal representation of knowledge. On the other hand, OWL-DL is a standard highly expressive language with facilities for expressing meaning and semantics.

As declared in Section 1.2 on page 3, the first task to be solved is to find a conceptual ontology modeling language that can be used to describe the semantics of the investigated grammar learning agent's internal knowledge model the basic requirements of which are:

- its main building blocks should be concepts and their relationships,
- it should be predicate-centered, where predicate is a type of concept,
- it should have an apriori knowledge regarding the model elements,
- it should make a clear distinction between apriori and learned elements,
- it should reflect the multi-layered nature of conceptualization, and
- it should provide high levels of flexibility and extendibility.

The analysis of the existing conceptual models (see also [3], [4] and [5]) shows that they all have some shortcomings from the aspects of the requirements. They cannot provide a clear separation of logical and conceptual elements, and are not flexible enough to be applied as a general model. Therefore, a novel semantic model is proposed for representing the knowledge base in the modeled grammar induction system, which is exposed in the next chapter.

Chapter 3

Developing a Novel Semantic Representation Model

There are many languages in the brain, and each brain language has its own alphabet.

J.D. Norseen: Images of Mind – The Semiotic Alphabet (1996)

Within the frames of the project an important stage is to find an appropriate formalism for the semantic model that can represent the knowledge base of the grammar learning agent investigated. According to the analyses in Section 2.2.4 on page 18, traditional semantic modeling languages are not adequate for reflecting the process of conceptualization, they cannot provide a clear separation of logical and conceptual elements, and are not flexible enough to be applied as a general model. Therefore, the first task of the research is to develop a semantic ontology representation language that fulfils the requirements identified in Section 1.2 on page 3. Accordingly, in this chapter the development of a novel semantic model is discussed, the main characteristics of which should be the following:

- its main building blocks should be concepts and their relationships,
- it should be predicate-centered, where predicate is a type of concept,
- it should have an apriori knowledge regarding the model elements,
- it should make a clear distinction between apriori and learned elements,
- it should reflect the multi-layered nature of conceptualization, and
- it should provide high levels of flexibility and extendibility.

Beyond the above characteristics, the new semantic model should possess sufficient expressive power to represent symbolic language semantics, because it will be used for knowledge representation in a grammar induction system. Since the aim is to develop a general model, the most complex symbolic language must be taken into consideration, which is natural language (NL). Generally, in NLP tasks the predicate calculus is used for representing NL semantics, though the opinions in the related literature are quite diverse in terms of the suitability of first-order versus higher-order logic because of the conflicting demands of expressivity and computational effectiveness. This chapter starts with the analysis of how much expressive power is needed for the problem at hand. The result is that first-order predicate logic is too restricted, and also higher-order predicate logic needs some

extensions. With these extensions, the new semantic model is defined in two forms: it has a logic-based and a graphical representation, and is named Extended Conceptual Graph (ECG). Although the name might be confusing, the model hasn't got any connections with the extended (refined) versions of Sowa's CG model.

3.1 Representation of natural language

Natural languages are built up of sequences of words which are finite sequences of symbols over a given alphabet. Syntax is the term which defines the set of rules telling us how words may be combined to form sentences. Formally,

- the main building blocks of natural languages are characters (symbols) $\tilde{c} \in \Sigma$, where Σ denotes the finite character set (alphabet) of the language;
- words are finite sequences over Σ, that is every $w \in W \subseteq \Sigma^*$, where W denotes the set of words;
- sentences are finite sequences over W, that is every $s \in \mathcal{S} \subseteq W^*$, where $\mathcal{S}$ denotes the set of sentences.

Natural languages are infinite recursive systems, hence on the basis of understanding a finite number of words we can understand and construct an infinity of sentences recursively applying the rules of syntax [Keenan, 2005].

NL semantics is concerned with the relation between language and the 'world'. Hence, the meaning of a sentence determines the conditions under which it is true. Since, by definition sentences are finite sequences of words, and as a consequence of the recursive nature of language, the meaning of a word will determine what contribution it makes to the truth conditions of the sentences in which it occurs [Blackburn & Bos, 2003]. This is called the principle of compositionality. For correct interpretation, however, we also need to have world knowledge. Without context, that is without defining the domain of discourse, many human language sentences could be assigned several meanings. This ambiguity may result from the lexical ambiguity of words, or from the syntactic ambiguity of sentences (word combinations). In other words, NL sentences are built up of word constituents bearing a set of possible meanings which are made concrete by the actual context.

3.2 Semantic equivalence of NL and predicate logic

"In philosophy and linguistics the predicate calculus is used for analyzing the semantics and logic of natural language... The way expressions and structures contribute to the meaning of a natural language sentence is supposed to be determined and shown by means of its translation into the calculus" [Ben-Yami, 2004]. The advantages of predicate logic (PL) are the use of a simple and exact notation and interpretation system, the standard formalism and general applicability, the ability for reasoning and rule validation, and its convertability to other symbolisms.

By definition, two statements in the same system are logically equivalent if, for all possible values of the variables involved, both statements are true or both are false. If α and β are equivalent, we write $\alpha \equiv \beta$. Formally, given an interpretation $\mathfrak{I} = \langle \Delta, I \rangle$ and a variable assignment ν, formula α and formula β are equivalent if for all $\mathfrak{I}$ and ν

- $\mathfrak{I} \models_\nu \alpha$ and $\mathfrak{I} \models_\nu \beta$, or
- not $\mathfrak{I} \models_\nu \alpha$ and not $\mathfrak{I} \models_\nu \beta$.

If the two statements are in the same set of statements ($\mathcal{F}$), then semantic equivalence is a binary relation over the given set, denoted by $\Theta \subseteq \mathcal{F} \times \mathcal{F}$. If Θ is reflexive, symmetric and transitive, then is said to be an equivalence relation.

1. Θ is reflexive if $\forall \varsigma \in \mathcal{F}$ $(\varsigma \Theta \varsigma)$ holds.
2. Θ is symmetric if $\forall \varsigma_1, \varsigma_2 \in \mathcal{F}$ $(\varsigma_1 \Theta \varsigma_2 \Rightarrow \varsigma_2 \Theta \varsigma_1)$ holds. Thus, if ς_1 is the semantic equivalent of ς_2, then the opposite is also true.
3. Θ is transitive if $\forall \varsigma_1, \varsigma_2, \varsigma_3 \in \mathcal{F}$ $(\varsigma_1 \Theta \varsigma_2 \wedge \varsigma_2 \Theta \varsigma_3 \Rightarrow \varsigma_1 \Theta \varsigma_3)$ holds. Thus, if ς_1 is the semantic equivalent of ς_2 and ς_2 is the semantic equivalent of ς_3, it entails that ς_1 is the semantic equivalent of ς_3.

Taking the sets of NL sentences and PL formulas all three properties evidently hold, therefore semantic equivalence can be considered as equivalence relation over both sets. An equivalence relation divides a set into a number of non-empty, pairwise disjoint subsets (equivalence classes). The statement sets constructed from these semantic equivalence classes can be denoted by NL/Θ and PL/Θ, respectively. Between the two sets, a semantic equivalence relation $\mathbf{n} : NL/\Theta \to PL/\Theta$ is defined as follows.

1. $\mathbf{n}$ is a mapping: if $\forall s \in NL/\Theta$, $\exists \alpha \in PL/\Theta$ so that $(s, \alpha) \in \mathbf{n}$, and $\forall s \in NL/\Theta$, $\forall \alpha_1, \alpha_2 \in PL/\Theta$ $((s, \alpha_1), (s, \alpha_2) \in \mathbf{n} \Rightarrow \alpha_1 = \alpha_2)$. Thus, every NL sentence should have a corresponding PL formula.
2. $\mathbf{n}$ is injective: if $\forall s_1, s_2 \in NL/\Theta$, $\forall \alpha \in PL/\Theta$ $((s_1, \alpha), (s_2, \alpha) \in \mathbf{n} \Rightarrow s_1 = s_2)$. Thus, every PL formula can have only one corresponding NL sentence (but it is not necessary to have any).
3. $\mathbf{n}$ is surjective: if $\forall \alpha \in PL/\Theta$, $\exists s \in NL/\Theta$ so that $(s, \alpha) \in \mathbf{n}$. Thus, every PL formula should have one corresponding NL sentence.
4. $\mathbf{n}$ is bijective: if $\mathbf{n}$ is injective and surjective.

The attention is restricted to logical forms reflecting syntactic structure. So, the definition of equivalence of two different notation systems is given by introducing the definition of a composition preserving transformation.

Definition 1. *Given two languages $\mathcal{L}_1(\mathcal{F}_1, \mathcal{O}_1)$ and $\mathcal{L}_2(\mathcal{F}_2, \mathcal{O}_2)$, where $\mathcal{F}$ denotes the set of formulas and $\mathcal{O}$ denotes the set of operations over $\mathcal{F}$, the transformation $\tau : \mathcal{L}_1 \to \mathcal{L}_2$ is said to be composition preserving if*

$$\tau(\sigma(\varsigma_1, \varsigma_2, ...)) \equiv \tau(\sigma)(\tau(\varsigma_1), \tau(\varsigma_2), ...), \tag{3.1}$$

i.e. $\tau(\sigma(\varsigma_1, \varsigma_2, ...))$ *and* $\tau(\sigma)(\tau(\varsigma_1), \tau(\varsigma_2), ...)$ *are equivalent in all interpretations.*

Without the criterion of composition preserving, an $\alpha(w_1, w_2, ...)$ general PL formula could be assigned to any arbitrary $s = w_1, w_2, ...$ NL sentence. In this case however, the semantic interpretation of the logical formula is not easier than that of the NL sentence.

In [10] the semantic equivalence assignments between NL/Θ and PL/Θ of different order are thoroughly examined. The conclusion is that $\mathbf{n} : NL/\Theta \rightarrow FOPL/\Theta$ is not a mapping because there are some linguistic phenomena that cannot be represented in standard FOPL at all, or not with the precondition of composition preserving. If the set of NL sentences is restricted though to those that can be represented in FOPL, $\mathbf{n}'$ is a multivalued mapping. On the other hand, $\mathbf{n}$" : $FOPL/\Theta \rightarrow$ *restricted* NL/Θ would be a surjective mapping if the criterion of composition preserving is ignored. Therefore the semantic content set $FOPL/\Theta$ is able to cover is narrower than that of NL/Θ.

We can go beyond FOPL in two directions. On the one hand, calculuses of higher order can be introduced, in which propositions or propositional functions (and therefore sets) can appear as arguments to other functions. On the other hand, higher (constructive and nonconstructive) methods can be used like recursive numerical functions, symbolic structures, and semantic methods. In some ways intermediate between these are systems in which numbers are explicitly introduced (as primitives) into the domain of arguments [Hinman, 2005]. The most obvious differences between HOPL and FOPL are that 1) HOPL uses variables that range over sets instead of discrete variables; and 2) in HOPL predicates can be arguments of predicates and values of variables (i.e. quantification over predicates is allowed). In other words, higher-order logics allow for quantification not only of elements of the domain of discourse, but subsets of the domain of discourse, sets of such subsets, and other objects of higher type (such as relations between relations, functions from relations to relations between relations, etc.). The semantics are defined so that, rather than having a separate domain for each higher-type quantifier to range over, the quantifiers instead range over all objects of the appropriate type [Shapiro, 2001].

The necessity of HOPL in representing NL semantics is proved in view of the arguments against it. Firstly, reification [Jurafsky & Martin, 2000] is a technique used for representing all concepts that one wants to make statements about as objects in FOPL, instead of using higher-order predicates. In this case, however, new relations need to be introduced which in fact do not solve, but only shift the problem. Moreover, the resulting valid FOPL formulas will not be in accordance with the precondition of composition preserving. Secondly, [Peregrin, 1997] states that FOPL is sufficient, since HOPL formulas can be converted into FOPL formulas. In the proposed formalism, an arbitrary $p_1(p_2(x))$ second-order statement can be transformed into a $p_1(y_1) \wedge p_2(y_2) \wedge p'(y_1, y_2, x)$ FOPL formula; because $\forall p.p(x)$ is rendered into $\forall y.p(y) \rightarrow p'(y, x)$. This solution formally results in valid FOPL formulas, but the criterion of composition preserving is violated because the resulting formulas ignore the subordination relation between the NL constituents: all

elements are at the same level, and the original structure is obscured. This problem can be eliminated by the use of higher-order predicates. In general, a higher-order predicate of order n takes one or more $(n-1)^{th}$-order predicates as arguments, where $n > 1$. Thirdly, since FOPL restricts the use of quantifiers to $\exists$ and $\forall$ HOPL is needed to introduce primitive quantifier expressions, as well as symbols for handling counting quantifiers [Barwise & Cooper, 1981]. Although higher-order logics are more expressive, in [10] it is shown that this formalism also needs further extensions in order to comply with the criterion of composition preserving.

3.3 ECG: the new semantic representation model

The present project aims at studying a learning agent that is able to learn the grammar rules of the language of the input data corresponding to the observations of the agent. In the first stage, the expressions examined are restricted to the observations which are related to definite, unambiguously interpretable situations. Consequently, the sentences describing these situations are factual assertions with true logical values. Therefore the following linguistic phenomena are beyond the scope of the investigations:

- if-then structures and conditionals,
- imperative, optative, exclamatory and interrogative sentences,
- probability and other certainty/uncertainty factors,
- intentional secondary meaning (pragmatics).

On the other hand, linguistic phenomena that need to be studied are as follows:

- domain types,
- referring expressions,
- adverbs,
- adjectives,
- numerical expressions and cardinality,
- quantification,
- logical connectives,
- historical (temporal) sequences, and
- causality.

For the composition preserving logical representation of the linguistic phenomena examined, the following HOPL extensions are proposed.

1) Arbitrary predicates (relations) are allowed, denoted by capitalized words. Domain types are assigned to the arguments of predicates, which specify the semantic roles these arguments play. The fixed set of roles (analogously to thematic roles [Fillmore, 1968] in linguistics) are associated with and determined by the predicate.

1.1/a Peter loves Mary.

1.1/b Love(Subject: Peter, Object: Mary).

2) Concepts are regarded as sets. Constants referring to specific objects (concepts) are single-element sets denoted by capitalized words, while constants referring to general concepts are multiple-element sets denoted by lower-case words. An element of a set c is denoted by the $isa(c)$ function (functions are denoted by lower-case words). An object can be referred by the : operator.

2.1/a Peter reads a book.

2.1/b Read(Subject: Peter, Object: isa(book)).

2.2/a Peter reads a/the book Tom likes.

2.2/b Read(Subject: Peter, Object: isa(book):x |
Like(Subject: Tom, Object: x)).

From the set-based treatment of concepts follows that plural forms, when used for referencing objects in general, are represented as abstract concepts, i.e. multiple-element sets.

2.3/a Peter likes books.

2.3/b Like(Subject: Peter, Object: book).

3) By the representation of adverbs a distinction should be made between those that describe the circumstances of the action or state expressed by the predicate, and those that add extra conditions connected with the basic assertion. The latter is represented by the use of the *Happens* relation. The difference is clearly seen in the second example.

3.1/a Peter travels by train.

3.1/b Travel(Subject: Peter, Instrument: isa(train)).

3.2/a Peter often travels by train.

3.2/b Happens(Subject: Travel(Subject: Peter, Instrument: isa(train)),
Time: Often).

3.2/c *Travel(Subject: Peter, Instrument: isa(train), Time: Often).*

The logical form in 3.2/c is incorrect, because from its truth it does not follow that *Travel(Subject: Peter, Time: Often)* is true.

The next example is an illustration for the ambivalent nature of NL, where it cannot be decided which predicate the adverb is linked to.

3.3/a I see you running today.

3.3/b See(Subject: I, Object: Run(Subject: You, Time: Today)).

3.3/c See(Subject: I, Object: Run(Subject: You), Time: Today).

4) Adjectives can be added to the assertion by the use of the *Property* relation.

4.1/a Peter reads a scientific book.

4.1/b Read(Subject: Peter, Object: isa(book):x |
Property(Subject: x, Object: Scientific)).

5) For the treatment of numerical expressions, numerical relations and numerical primitives are needed to be introduced, as well as the $some(c)$ function for creating a group of objects.

5.1/a Peter reads two books.

5.1/b Read(Subject: Peter, Object: some(isa(book)):x |
Property(Subject: x, Object: Two)).

5.2/a Peter reads more books than magazines.

5.2/b Read(Subject: Peter, Object: (some(isa(book)):x,
some(isa(magazine)):y) | More(Subject: x, Object: y)).

5.3/a Peter reads more books than Tom.

5.3/b (Read(Subject: Peter, Object: some(isa(book)):x),
Read(Subject: Tom, Object: some(isa(book)):y) |
More(Subject: x, Object: y)).

6) Existential and universal quantifiers are defined similarly by means of the $some(c)$ and $all(c)$ functions, respectively.

6.1/a There is a book on a/the table.

6.1/b Is(Subject: isa(book), Location: isa(table)).

6.2/a There are some books on a/the table.

6.2/b Is(Subject: some(isa(book)), Location: isa(table)).

6.3/a All books are on a/the table.

6.3/b Is(Subject: all(isa(book)), Location: isa(table)).

7) Logical operators can be applied to predicates or to arguments of predicates. When they refer to predicates it should be noted that *and* means the presence of multiple predicates (they can be connected with the , operator), while *or* means the uncertainty of the observation (which is beyond the scope of the investigations).

7.1/a Peter reads and laughs.

7.1/b (Read(Subject: Peter), Laugh(Subject: Peter)).

7.2/a Peter reads not laughs. ≡ Peter reads.

7.2/b Peter does not read.

7.2/c NotRead(Subject: Peter).

In the latter example, case (a) demonstrates that the examinations are restricted to observations and the addition of extra information is not allowed. Case (b) states that Peter is not doing something without stating what he is doing. As a result, case (c) shows that an observation is uninterpretable without a specific predicate, thus *not* can only be allowed if included in the predicate.

The same applies when logical operators are related to arguments of predicates. Here *and* is represented by the grouping of the corresponding arguments, and *or* means uncertainty which is not covered by the investigations. Also, negation either expresses that an argument is not something without saying what it is, which is uninterpretable in the present framework; or it states what the argument is, in which case the negation is extra information (e.g.: Peter reads not a book but a magazine. = Peter reads a magazine.).

8) Temporal aspects can only be studied when more observations are compared on a historical basis. In this case the former observation(s) must have a tense preceding the latter observation(s). The observation at the end of the history demonstrates the actual (present) state of the system.

8.1/a Peter gives Tom a book. Tom reads the book.

8.1/b Give(Subject: Peter, Object: isa(book):x, Recipient: Tom) →
Read(Subject: Tom, Object: x).

8.1/c Tom reads the book that Peter gave him.

9) The examination of causes and results leads us back again to the *Happens* relation.

9.1/a Peter cannot sleep because Tom is dancing.

9.1/b Happens(Cause: Dance(Subject: Tom),
Result: NotSleep(Subject: Peter)).

The new formalism that has been developed by applying the above extensions to HOPL (called EHOPL) is the logic-based representation of the ECG semantic model (called ECG-HOPL). From the above analysis it can be seen that in view of the criterion of composition preserving, ECG-HOPL approximates NL better than HOPL without these extensions. Therefore, considering the assignment $\mathbf{m}'$: $ECG\text{-}HOPL/\Theta \rightarrow NL/\Theta$ (where $ECG\text{-}HOPL/\Theta$ denotes the set of EHOPL statements constructed from semantic equivalence classes) it can be stated that $\mathbf{m}'$ is a surjective mapping. This result is summarized in the next statement.

Statement 1. *Every ECG-HOPL statement can be semantically unambiguously rendered into an NL sentence examined, that is every ECG-HOPL statement can have only one corresponding NL formulation (with the assumption that semantically identical NL sentences examined are considered as one). On the other hand, every NL sentence examined can be approximated by an ECG-HOPL statement if the pragmatic level of language is not taken into account.*

Consequence. *ECG can be used as a sentence-level semantic annotation language.*

3.3.1 Formal definition of model elements

The main components of the problem area are learning agents and the observed environment. It is assumed that the agents are able to detect objects, their attributes and some given kinds of relationships between them within the environment. This set of relationships depends on the agent's type; that is different agent classes may have different sets of recognizable relations. Based on the observations, each agent develops a knowledge model that describes the semantics of its observations and consequently that of the environment. Therefore, the model language proposed defines a high-level, graph-based semantic model that can be used to describe the different phases of conceptualization within the agents examined. The ECG model is based on the following base sets:

- the set of all (static and dynamic) object instances, denoted by U_I,
- the set of concepts, denoted by U_C, and
- the set of agents, denoted by U_A.

It is assumed that these base sets are non-empty sets. In the detailed description of the model language, the following terms and formal definitions are used.

1) The term environment denotes the context in which the agents exist. The environment, denoted by Γ, is represented in a graph formalism as the key elements of the model are objects and their relationships. The environment is shared by different agents. Formally,

$$\begin{aligned} \Gamma &= \Gamma(I, \rightarrow_I), \\ I &\subseteq U_I, \\ \rightarrow_I &\subseteq I \times I^2. \end{aligned} \tag{3.2}$$

The node component of the Γ environment is denoted by I_Γ and the edge component has a notation of $\rightarrow_{I_\Gamma}$.

2) The Γ' sub-environment is a subset of the Γ environment, $\Gamma' \subseteq \Gamma$, so that the following properties are met:

$$\begin{aligned} I_{\Gamma'} &\subseteq I_\Gamma, \\ \rightarrow_{I_{\Gamma'}} &\subseteq \rightarrow_{I_\Gamma}, \\ \rightarrow_{I_{\Gamma'}} &\subseteq I_{\Gamma'} \times I_{\Gamma'}^2. \end{aligned} \tag{3.3}$$

3) The characteristic environment of an agent ι, denoted by Γ_ι, is a sub-environment that is assigned to the agent. This sub-environment is used to describe the direct environment of the agent, as different agents may have access to different subparts of the universal environment. The role of the characteristic environment is to capture all elements that an agent has access to.

4) The environment snapshot for an agent ι is a sub-environment of its Γ_ι characteristic environment, denoted by S_{Γ_ι}. The snapshot refers to the environment accessed at a

given point of time. The snapshot is a static mapping of the environment. The list of environment snapshots is defined as the environment history and is denoted by H_{Γ_ι}, where

$$\bigcup_{S \in H_{\Gamma_\iota}} S = \Gamma_\iota. \tag{3.4}$$

5) A primary knowledge model describes a concept-architecture. This level corresponds to the lowest level of concepts, i.e. instance concepts. A knowledge model always belongs to a single agent. The elements of the primary knowledge model are symbolic representations of the characteristic environment of the agent. The primary knowledge model is given by a graph where the nodes are called concepts and the edges are the relationships between them. The primary knowledge model of an agent α is denoted by $\Lambda_{p,\alpha}$, where

$$\begin{aligned} \Lambda_{\pi,\iota} &= \{\tilde{C}, \rightarrow_{\tilde{C}}\}, \\ \tilde{C} &\subseteq U_C, \\ \rightarrow_{\tilde{C}} &\subseteq \tilde{C} \times \tilde{C}^2. \end{aligned} \tag{3.5}$$

The $\tilde{C}$ concept part of the $\Lambda_{\pi,\iota}$ knowledge model is denoted by $\tilde{C}_{\Lambda_{\pi,\iota}}$.

6) Since the primary knowledge model corresponds to the actual characteristic environment, a mapping function is defined from the environment to the concepts. This lowest level of conceptualization is called signal level layer in [Sieber, 2008]. In the present model, the signal level layer corresponds to the primary knowledge model. The mapping from the environment to the primary knowledge model has an important role, and is defined as primary conceptualization mapping. This is a surjective, but usually non-injective function from the agent's characteristic environment onto the primary knowledge model – that is different objects may be mapped to the same concept – denoted by

$$\chi_{\pi,\iota} : \Gamma_\iota \rightarrow \Lambda_{\pi,\iota}, \tag{3.6}$$

where instances are mapped to concepts and environment relationships correspond to concept relationships:

$$\begin{aligned} \chi_{\pi,\iota}(I_{\Gamma_\iota}) &= \tilde{C}_{\Lambda_{\pi,\iota}}, \\ \chi_{\pi,\iota}(\rightarrow_{I_{\Gamma_\iota}}) &= \rightarrow_{\tilde{C}_{\Lambda_{\pi,\iota}}}. \end{aligned} \tag{3.7}$$

7) A primary knowledge model snapshot for an agent is a primary conceptualization mapping of its characteristic environment snapshot, denoted by $S_{\Lambda_{\pi,\iota}}$. The snapshot belongs to a given point of time. During the lifetime of an agent, the agent processes a sequence of snapshots. This sequence is called the history of the agent, denoted by $H_{\Lambda_{\pi,\iota}}$, where

$$\bigcup_{S \in H_{\Lambda_{\pi,\iota}}} S = \Lambda_{\pi,\iota}. \tag{3.8}$$

The history of the primary knowledge model snapshots is the prime source of the agent's conceptualization process. Agents can store not only the primary knowledge models but

the corresponding history, too. In addition to the primary knowledge model, agents can also manage higher-level concepts. These concepts are defined as the extension of the primary knowledge model. The knowledge model snapshots describe simple static information modules where temporal aspects of the events are ignored.

8) Temporal and other complex relationships are managed at a higher level of conceptualization. The Λ_ι extended knowledge model of agent ι is defined to describe compound, abstract concepts, so that

$$\begin{aligned} \Lambda_\iota &= \{\tilde{C}, \rightarrow_{\tilde{C}}\}, \\ \tilde{C} &\subseteq U_C, \\ \rightarrow_{\tilde{C}} &\subseteq \tilde{C} \times \tilde{C}^2. \end{aligned} \tag{3.9}$$

The Λ_ι model is an extension of the primary knowledge model, thus

$$\begin{aligned} \tilde{C}_{\Lambda_{\pi,\iota}} &\subseteq \tilde{C}_{\Lambda_\iota}, \\ \rightarrow_{\tilde{C}_{\Lambda_{\pi,\iota}}} &\subseteq \rightarrow_{\tilde{C}_{\Lambda_\iota}}. \end{aligned} \tag{3.10}$$

9) In order to keep predicates in the center of modeling, a special type of concepts is introduced, the so-called predicate concepts which are the kernels of atomic propositions. The set of predicate concepts is open. Predicate concepts with the corresponding relationships are based on the primary knowledge model and are defined with patterns. Non-predicate concepts are called category concepts. Derivation rules are used to define category concepts from primary knowledge concepts. The set of derivation rules is agent dependent. This rule set is denoted by

$$\tilde{R}_\iota : \Lambda_{\pi,\iota} \rightarrow \Lambda_\iota. \tag{3.11}$$

10) An extended knowledge model fragment is a sub-model that contains exactly one predicate concept as its dedicated core (or kernel) predicate concept and some other concepts, linked to this predicate concept, which may be predicate concepts as well as category concepts. The graph of the fragment is a connected graph. The fragment is used to describe a statement or proposition.

3.3.2 Basic building blocks of the model

The model should be able to support model validation and reasoning processes as well. Thus, a key feature of the model is its strong relationship to the logic-based predicate formalism; that is the model can be constructed from atomic predicates and it can be transformed into logical formulas. The logic-based formalism provides a more powerful mechanism for model evaluation and model processing.
Beside the usual modeling elements, additional elements are introduced that enable a more efficient and powerful description of the conceptualization process. The proposed ECG model contains three primitive types: concepts, relationships and containers. Based on their behavior, the following concept sub-types are defined.

1. According to the grade of identification:

    - (N) Noname concept: is a primary concept that has no context-unique identification name.
    - (R) Permanent-named concept: is a concept having a context-unique name. A permanent concept is associated with an implicit definition that enables the identification of its instances in the environment.
    - (T) Temporary-named concept: is a concept occurring in some previous snapshot(s) of the history as a noname concept.

2. Categories on a logical basis:

    - (P) Predicate concept: is a concept that is used to denote predicates that are usually given by verbs in sentences. Predicate concepts can be the kernels of model fragments.
    - (C) Category concept: is the term covering all non-predicate concepts. Category concepts can denote various attributes for example. Each category concept defines a subset of instances that match this category concept.

3. According to the model level:

    - (F) Primary concept: is a concept at the instance level. Primary concepts correspond to instances of the agent's environment.
    - (A) Abstract (derived) concept: a higher-level concept in the agent's extended knowledge model. The derivation rule is defined with a sequence of snapshots.

4. Categories on the basis of cardinality:

    - (I) Single instance concept: is a concept that identifies only one object.
    - (M) Class concept: is a concept that covers several instances.

Due to the semantic integrity constraints, only the following concept types are allowed: FICN, FICT, FICR, FMCR, FMPR, AMCR and AMPR.

Relationship types are categorized as follows.

1. According to the model level:

    - (F) Primary relationship: is a relationship that can be recognized by the agent. It is detected usually in the environment.
    - (A) Abstract (derived) relationship: is a relationship that is based on primary relationships. The derivation rule is defined with a sequence of snapshots.

2. According to the logical level:

    - (I) Specialization relationship: is equivalent to the usual ISA relationship. It provides inheritance. A concept may have multiple parents.
    - (R) Role relationship: is a relation representing an arbitrary attribute of a predicate concept.

3. Categories on the basis of cardinality:

    - (S) Single instance relationship: only one object cluster is identified by this relation.
    - (M) Class relationship: several instances can belong to the relationship.

The relationship types allowed are FMI, FSR, FMR and AMR.

The group of container elements includes structure modules.

- Model snapshot: is a model segment that corresponds to a given point of time. It contains only static elements.
- Model fragment: is a special part of the entire model. A fragment should contain exactly one predicate concept as its dedicated kernel concept. A knowledge model is usually transported to other agents in fragment units.
- Model history: is a sequence of successive snapshots in time. It can describe events and other time-dependent phenomena.

Between the container elements a special relationship is defined, called derivation relationship. It is a relationship that can be used to describe derivation rules. It chains the derived element with a sequence of corresponding snapshots.

3.3.3 Graphical representation of ECG

In order to present the elements of the model in an easy to understand formalism, a graphical language is defined (called ECG diagram). The graphical elements of the model are listed in Figure 3.1.

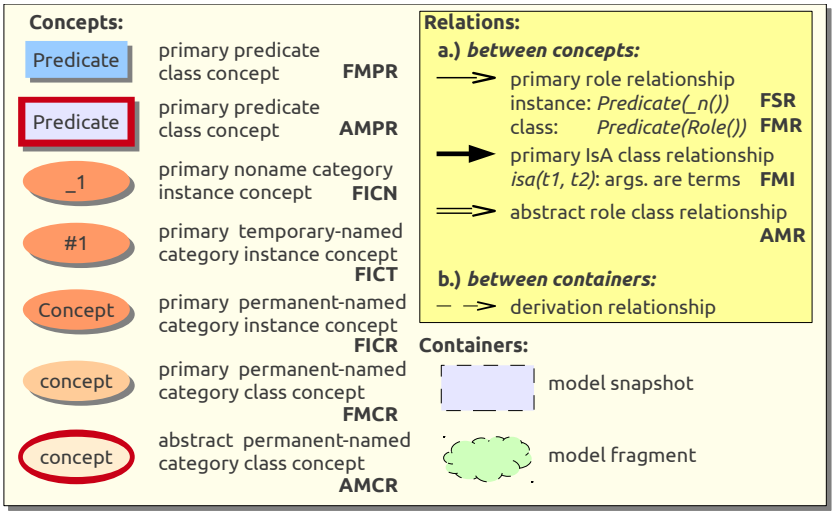

Figure 3.1 *Graphical components of ECG diagram*

- FICN is an instance constant with a name of the form $\_n$, where n is a number.
- FICT is an instance constant with a name of the form $\#n$, where n is a number.

- FICR is an instance constant with a name of capital letters.
- FMCR is a class constant with a name of lower-case letters.
- FMPR is a predicate with a name of capital letters.
- AMCR is a class constant with a name of lower-case letters and with an implication rule.
- AMPR is a predicate with a name of capital letters and with an implication rule.
- FMI is a predicate of the form $isa(t_1, t_2)$, where arguments t_1 and t_2 are terms.
- FSR is a function used as predicate argument, where the name is given in the form $\_n$, where n is a number.
- FMR is a function used as predicate argument, where the name is given in capital letters.
- AMR is a function used as predicate argument, where the name is given in capital letters and with an implication rule.

The use of the graphical elements is demonstrated by the example described in Section 2.2 on page 10. The agent's characteristic environment contains an environment snapshot with a black circle located in the middle of a white triangle. The agent is defined so that it can detect the individual objects of the environment, together with their color attributes, and is able to recognize the binary relation of inclusion between them. Let us assume that agents can observe the signals coming from their characteristic environment through different sensors each dedicated to a particular sense. That is, for example agents have a separate sensor for detecting the color attribute of objects. Thus, observing the color of an object is a built-in ability of agents together with the knowledge that signals coming from this sensor are colors (i.e. colors are permanent-named concepts). Based on its observations, the agent constructs the primary knowledge model with primary concepts and relationships, which is displayed in Figure 3.2. The present model is built up of a model snapshot with one fragment, where the dedicated predicate concept is marked by *.

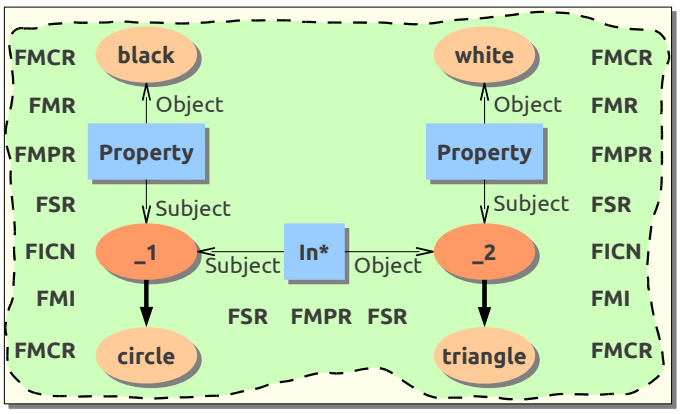

Figure 3.2 *ECG diagram representation of "A black circle is in a white triangle"*

3.3.4 Equivalence of ECG-HOPL and ECG diagram

In Definition 1 a composition preserving transformation is defined, and it is stated that two statements from different notation systems are said to be equivalent if there exists a composition preserving transformation between them by means of which the two formulas are equivalent in all interpretations. The following analysis goes through the linguistic phenomena identified in Section 3.3 on page 33, and specifies the composition preserving transformation of ECG-HOPL statements into ECG diagram graphical structures. Figure 3.3 shows the basic ECG diagram structures identified.

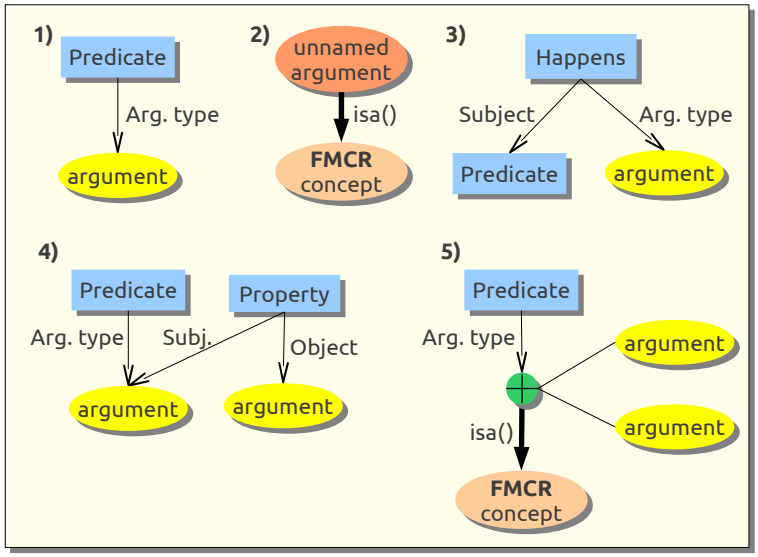

Figure 3.3 *Basic ECG diagram graphical structures*

Domain types: 1) shows a predicate with a typed argument, where types correspond to semantic roles. The fixed set of roles are associated with and determined by the predicate. Arguments can be arbitrary ECG concepts, including predicate concepts as well.

Referring expressions: Objects are represented by different types of category concepts (see Figure 3.1). Accordingly, there is a distinction between concepts referring to concrete objects (FICR), concepts referring to a collection of objects (FMCR), concepts referring to unreferenced unnamed objects (FICN), and concepts referring to referenced unnamed objects (FICT). The two latter serve for making a distinction between the use of indefinite and definite articles, respectively (see Figure 3.2). 2) illustrates how an unnamed object is associated with a collection of named objects through the *isa*() relationship.

Adverbs: Adverbs connected to the predicate are considered to be extra arguments of the predicate as in 1). On the other hand, 3) demonstrates how adverbs associated not only with the predicate itself but with the whole assertion are handled.

Adjectives: 4) shows the treatment of adjectives as arguments of the *Property* predicate.

Numerical expressions: 5) displays how groups of objects can be composed. If an adjective indicating the cardinality of the group is also present, then a *Property* predicate with an argument needs to be added to the construction.

Causality: Causality can be traced back to 3) where the *Happens* predicate has a *Cause-type* and a *Result-type* predicate argument.

Quantifiers, logical connectives, temporal sequences: The handling of quantifiers, logical connectives and temporal sequences originates in the previously discussed basic structures with the extension that also predicates can comprise a group.

From these basic structures an ECG diagram, which is actually a semantic network, with arbitrary complexity can be built. For illustration, see Figure 3.2.

This analysis proves that the assignment $\mathbf{e} : \textit{ECG-HOPL}/\Theta \rightarrow \textit{ECG diagram}$ is a bijective function, therefore $\mathbf{e^{-1}} : \textit{ECG diagram} \rightarrow \textit{ECG-HOPL}/\Theta$ also exists.

Statement 2. *The ECG-HOPL and the ECG diagram formalisms are semantically equivalent, that is the same semantic content can be represented by both symbolisms. Therefore the ECG diagram can be viewed as the graphical counterpart of the ECG-HOPL language.*

As a consequence, the assignment $\mathbf{f'} : \textit{ECG diagram} \rightarrow NL/\Theta$ is a surjective mapping, just like $\mathbf{m'} : \textit{ECG-HOPL}/\Theta \rightarrow NL/\Theta$.

3.3.5 Visualization of ECG ontologies

Generally speaking, ontology is structured knowledge that defines a set of domain concepts through characteristic relations. Within the frames of the dissertation, semantic information is represented by the ECG model which bears the features of ontologies, therefore it can be considered as an ontology modeling language. In this project ECG ontologies are stored in standard OWL DL textual format. The steps of their construction can be found in Appendix B. They are visualized by ECG diagram graphs, the creation of which is performed by Algorithm 3.1.

The cost of the algorithm is determined in function of the size of the input (denoted by n), which is the total number of concepts and relations in the ontology to be displayed. This equals the number of OWL elements in the input file. Summing up the costs of the instructions, the cost of the algorithm is $2n + (n/2)^2$, where the number of relations is $n/2$; which can be approximated by $O(n^2)$.

```
Input: ECG-HOPL ontology in textual OWL DL format
Output: ECG diagram graph
kp = Find(hasKernelPredicate)
Level = 0
Draw(kp)
Nodes = kp
Relations = Find all relations where hasDomain=kp
while Relations is not empty do
    Level = Level + 1
    LevelRelations = { }
    foreach r ∈ Relations do
        domain = Find(hasDomain)
        range = Find(hasRange)
        if range is not in Nodes then
            Draw(range)
            Draw(r) starting from domain
            Nodes = Nodes + range
            LevelRelations = LevelRelations + range
        if domain is not in Nodes then
            Draw(domain)
            Draw(r) pointing to range
            Nodes = Nodes + domain
            LevelRelations = LevelRelations + domain
    Relations = Find all relations where hasDomain or hasRange is in LevelRelations
```

Algorithm 3.1 *Generation of ECG diagram graphs from OWL ontologies*

3.3.6 Model evaluation

ECG is a conceptual modeling language designed in view of the requirements defined in Section 1.2 on page 3. The main characteristics of the model can be summarized as follows.

Main building blocks of the model The main building blocks of the ECG model are concepts, relationships, and containers which serve for structuring the model. ECG differentiates between several categories of concept and relationship types which constitute the container types. The 'world' is built up of interconnected ECG model fragments representing separate observations, containing exactly one kernel predicate (denoted by *) and having 'true' truth value.

Predicate-centeredness Predicate concepts, which are distinguished from non-predicate concepts, are the kernels of atomic propositions. In the center of an ECG model fragment stays the kernel predicate, and each basic ECG graphical structure is organized around a predicate.

Multiple conceptualization levels In the ECG model, the process of conceptualization occurs at two levels. At the primary level the direct and static mapping of the objects and relations within an observation takes place. At the extended or abstract level temporal and other complex relationship types are also managed.

Apriori knowledge regarding model elements The ECG model is designed for knowledge representation in learning agents. The capabilities of the agents are fixed in advance, which are pattern recognition, association and generalization. Agents are defined so that they are able to detect objects in the environment, their attributes, and the relationships between them; where the set of recognizable attributes and relationships are pre-defined.

Distinction between apriori and learned elements The primary level of the ECG model serves for the direct mapping of environment objects and relationships into primary-level knowledge items. The abstract level of the ECG model provides abstract elements for representing the phases of conceptualization.

Flexibility The ECG model is able to grasp the semantic content of situations. The elements of the environment can be represented by the relatively small, fixed set of ECG model elements. This means that several environment elements are mapped to the same ECG model element, which has therefore flexible semantic assignment.

Extendibility The ECG model is a recursive, compositional system: that is infinitely many statements can be constructed from the small finite set of model elements. Consequently, the more extended an ECG model is, the better it is able to approximate NL.

According to the evaluation of the model, it can be stated that ECG satisfies the declared requirements of the knowledge representation form in the grammar induction system investigated, and therefore it can be used to describe the semantics of the examined grammar learning agent's internal knowledge model.

3.4 Formalizing ECG-HOPL with CFG

From a practical point of view, it is important to see that the syntax of the proposed semantic representation language is simple enough so that it can be generated by a powerful, but computationally tractable grammar. Since CFG is the most commonly used mathematical system, or meta-language for modeling constituent structure in NLs, the goal of this section is to show that the syntax of ECG-HOPL can be given by context-free rules (see also [9]).

The syntax of FOPL formulas is given by the $\mathcal{G}' = (\Delta, NT, TS, PR, \mathbf{S})$ grammar. The well-formed formulas can be generated by starting from $\mathbf{S} \in NT$ and applying the PR production rules. NT is the set of nonterminals, TS is the set of terminals; and $TS \cap NT = \emptyset$ holds. FOPL statements are interpreted over the $D_O \in \Delta$ domains comprising atomic objects. Production rules of context-free grammars are of the form $N \to \delta$, where N stands for an arbitrary nonterminal, while δ is an arbitrary sequence of terminals and nonterminals which can even be empty. Accordingly, the formal definition of FOPL with CFG is as

follows (where $f : D_{O_1} \times D_{O_2} \cdots \to \Delta$ is the function symbol and $p \subseteq D_{O_1} \times D_{O_2} \ldots$ is the predicate symbol).

$$
\begin{aligned}
\mathbf{S} &= \{\ \text{Formula}\ \} \\
NT &= \{\ \text{Formula},\quad \text{Atom},\quad \text{Term}\ \} \\
TS &= \{\ \text{connective},\quad \text{quantifier},\quad \text{constant},\quad \text{variable},\quad \text{predicate},\quad \text{function}\ \} \\
PR &= \{\ \text{Formula} \to \text{Atom} \,|\, \neg\,\text{Formula} \,|\, \text{Formula connective Formula} \,|\, \text{quantifier variable Formula} \\
&\qquad \text{Atom} \to \text{predicate(Term, } \ldots\text{)} \\
&\qquad \text{Term} \to \text{function(Term, } \ldots\text{)} \,|\, \text{constant} \,|\, \text{variable} \\
&\quad \}
\end{aligned}
$$

$$
\begin{aligned}
\Delta &= \bigcup\nolimits_{D_O \in \Delta} D_O \\
&\quad D_{\text{connective}} = \{\ \wedge,\quad \vee,\quad \Rightarrow\ \} \\
&\quad D_{\text{quantifier}} = \{\ \exists,\quad \forall,\ \} \\
&\quad D_{\text{constant}} = \{\ A,\quad B,\quad \ldots\ \} && \in D_{O_i} \\
&\quad D_{\text{variable}} = \{\ x,\quad y,\quad \ldots\ \} && \in D_{O_j} \\
&\quad D_{\text{predicate}} = \{\ \text{Read},\quad \text{Give},\quad \ldots\ \} && \in D_{O_k} \\
&\quad D_{\text{function}} = \{\ \text{fatherOf},\quad \text{friendOf},\quad \ldots\ \} && \in D_{O_l}
\end{aligned}
$$

The most significant differences between HOPL and FOPL are that

1. HOPL uses variables that range over sets instead of discrete variables, and
2. in HOPL predicates can be arguments of predicates and values of variables (that is quantification over predicates is allowed).

Accordingly, the formal definition of HOPL with CFG is as follows (where $f : D_{O_1} \times D_{O_2} \cdots \to \Delta$ is the function symbol and $p \subseteq D_{O_1} \times D_{O_2} \ldots$ is the predicate symbol).

$$
\begin{aligned}
\mathbf{S} &= \{\ \text{Formula}\ \} \\
NT &= \{\ \text{Formula},\quad \text{Atom},\quad \text{Term}\ \} \\
TS &= \{\ \text{connective},\quad \text{quantifier},\quad \text{constant,variable},\quad \text{predicate},\quad \text{function}\ \} \\
PR &= \{\ \text{Formula} \to \text{Atom} \,|\, \neg\,\text{Formula} \,|\, \text{Formula connective Formula} \,|\, \text{quantifier variable Formula} \\
&\qquad \text{Atom} \to \text{predicate(Term, } \ldots\text{)} \,|\, \text{predicate(Atom, } \ldots\text{)} \\
&\qquad \text{Term} \to \text{function(Term, } \ldots\text{)} \,|\, \text{function(Atom, } \ldots\text{)} \,|\, \text{constant} \,|\, \text{variable} \\
&\quad \}
\end{aligned}
$$

$$
\begin{aligned}
\Delta &= \bigcup\nolimits_{D_O \in \Delta} D_O \\
&\quad D_{\text{connective}} = \{\ \wedge,\quad \vee,\quad \Rightarrow\ \} \\
&\quad D_{\text{quantifier}} = \{\ \exists,\quad \forall,\ \} \\
&\quad D_{\text{constant}} = \{\ A,\quad B,\quad \ldots\ \} && \in D_{O_i} \\
&\quad D_{\text{variable}} = \{\ x,\quad y,\quad \ldots\ \} && \in D_{O_j} \cup D_{\text{predicate}} \\
&\quad D_{\text{predicate}} = \{\ \text{Read},\quad \text{Give},\quad \ldots\ \} && \in D_{O_k} \\
&\quad D_{\text{function}} = \{\ \text{fatherOf},\quad \text{friendOf},\quad \ldots\ \} && \in D_{O_l}
\end{aligned}
$$

The most important extensions of HOPL introduced are that

1. the arguments of predicates are typed,
2. constants are differentiated as single-element and multiple-element sets,
3. formulas can also be arguments of predicates, and
4. quantifiers are eliminated.

Among these, the first three are real extensions of FOPL, thus FOPL formulas can be converted to a form complying to them. For the elimination of quantifiers the lemma is used that to every FOPL formula a logically equivalent prenex formula can be constructed [Varga & Várterész, 2003], which has the form

$$Q_1 x_1 Q_2 x_2 \dots Q_n x_n \alpha, \tag{3.12}$$

where $(n \geq 0)$ and Q denotes the quantifiers and α is quantifier-free. Then the *some*() and *all*() functions can be applied to this formula. Accordingly, the formal definition of ECG-HOPL with CFG is the following (where $f : D_{O_1} \times D_{O_2} \cdots \rightarrow \Delta$ is the function symbol and $p \subseteq D_{O_1} \times D_{O_2} \dots$ is the predicate symbol).

$\mathbf{S} = \{$ Formula $\}$

$NT = \{$ Formula, Atom, Term, Compoundformula $\}$

$TS = \{$ connective, object constant, abstract constant, variable, predicate, function, type $\}$

$PR = \{$ Formula $\rightarrow$ Atom $|$ Compoundformula $|$Formula connective Formula
Atom $\rightarrow$ predicate(type: Term, ...) $|$predicate(type: Formula, ...)
Term $\rightarrow$ function(Term, ...) $|$function(Atom, ...) $|$object constant $|$ abstract constant
Compoundformula $\rightarrow$ predicate(type: Term, ...,
type: function(abstract constant):variable, ... $|$
predicate(type: Term, ...,type: variable, ...))
$\}$

$\Delta = \bigcup_{D_O \in \Delta} D_O$

$D_{\text{connective}} = \{$, $\rightarrow$ $\}$

$D_{\text{type}} = \{$ Subject, Object, ... $\}$

$D_{\text{object constant}} = \{$ Peter, Often, ... $\}$ $\in D_{O_i}$

$D_{\text{abstract constant}} = \{$ book, train, ... $\}$ $\in D_{O_j}$

$D_{\text{variable}} = \{$ x, y, ... $\}$ $\in D_{O_k}$

$D_{\text{predicate}} = \{$ Read, More, ... $\}$ $\in D_{O_l}$

$D_{\text{function}} = \{$ isa, some, all, ... $\}$ $\in D_{O_m}$

Statement 3. *ECG-HOPL can be converted to CFG, which proves that the syntax of the semantic representation language proposed is simple enough so that a computationally effective learning algorithm can be constructed for inducing a set of rules from ECG.*

3.5 Related work

In [Ilieva, 2007] a universal graphical notation is defined, which is equally valid for the presentation of linguistic knowledge and a problem domain knowledge. Using a unified graphical representation for natural language and the knowledge it carries, the authors are literally able to draw text as a picture. The process is based on a deep syntactic analysis (including part-of-speech tagging, morphological analysis, memory-based parsing and chunking) and a clear representation of the text, using a limited set of meaningful graphical symbols.

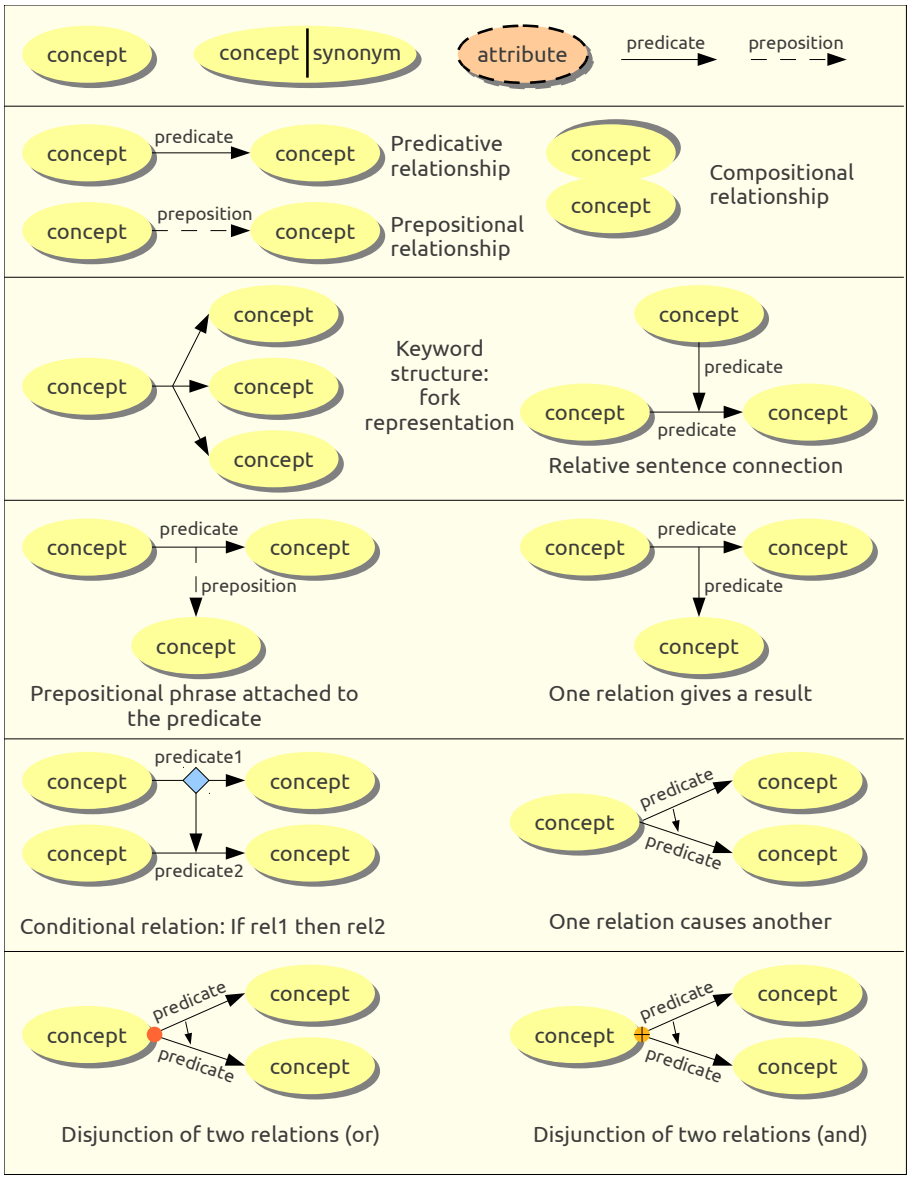

Figure 3.4 *Ilieva's basic graphical notations [Ilieva, 2007]*

As a result of the syntactic analysis, a natural language sentence is divided into three basic groups according to the function each one performs: Su(bject), Pr(edicate) and Ob(ject). The subject and the object are noun groups; while the predicate is a verb group consisting of a main verb, and its corresponding adverb, modality, infinitive, and auxiliary verbs.

Several sentences can be connected with conjunctions or relative pronouns in order to produce a compound sentence.

The basic building blocks of the graphical language are concepts (entities) and the relations between them (see Figure 3.4). The concepts are nouns in the natural language sentence taking on the role of subject or object. In the graph only the name of the concept is displayed in an oval form, while all other information is kept in the tabular presentation of the text, which serves as a knowledge base for all kinds of syntactic and semantic knowledge extracted during the analysis. The relations between the concepts are presented as a pointed and labeled arc. For the demonstration of the model's graphical elements, see Figure 3.5.

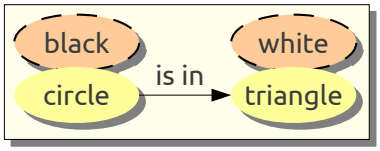

Figure 3.5 *Ilieva's representation of "A black circle is in a white triangle"*

Similarities with the approach of the present research are:

- the purpose of representing knowledge in a graphical form,
- the philosophy behind the model is that there is a relation between the basic building blocks of language and knowledge,
- the triple division of "knowledge" into subject, predicate and object,
- the basic building blocks of the graphical language are concepts and relations,
- the graphical language is a kind of semantic network (an assertional network),
- the model attaches to each entity all the relations it is involved in,
- entities can play different roles in different relations.

Characteristics of Ilieva's work are:

- the graphical language is used as an intermediate language between a natural language and UML (its main purpose is to automatically translate textual user requirements written in uncontrolled natural language into UML, or other types of software engineering diagrams),
- a graph is built after a syntactic analysis,
- semantic knowledge is stored in a structured tabular presentation,
- the model can also be considered as an executable or hybrid semantic network.

Distinguishing features of the present research are:

- the ECG language is predicate-centered,
- predicates are concepts as well, which are graphically distinguished from other types of concepts,
- several categories of concept and relation types are distinguished.

3.6 Summary of the results

In this chapter, the developed ECG semantic model has been introduced and analyzed concerning expressiveness. The new scientific results can be summarized as follows.

Thesis 1. *[8],[9],[10]*
A novel semantic model is developed, called ECG, which has a logic-based ECG-HOPL and a semantically equivalent graphical ECG diagram representation. The model satisfies the requirements of the knowledge representation format in the investigated grammar induction system, and can be used as an ontology modeling language because its main building blocks are concepts and their relationships. It is predicate-centered and it defines two levels and distinct elements for describing the different phases of conceptualization. It provides high levels of functionality, flexibility and extendibility. It is computationally tractable while highly expressive, that is it covers a wide range of linguistic phenomena.

Consequences

1. ECG can be used to describe the semantics of the examined grammar learning agent's internal knowledge model.

2. ECG can be applied as a sentence-level semantic annotation language, because every ECG-HOPL statement can be semantically unambiguously rendered into an NL sentence examined and every NL sentence under examination can be approximated by an ECG-HOPL statement.

   $< Statement\ 1 >$

3. Since ECG can be considered as an ontology modeling language, ECG diagram can be used for visual ontology representation. The generation of ECG diagram graphs can be accomplished by an $O(n^2)$ algorithm.

4. ECG-HOPL can be defined with CFG, which proves that the syntax of ECG is simple enough so that a computationally effective learning algorithm can be constructed for inducing a set of grammar rules from ECG, and consequently from sentences annotated by ECG. $< Statement\ 3 >$

Chapter 4

Developing a Grammar Representation Model

Combine the extremes, and you will have the true center.

Karl Wilhelm Friedrich Schlegel

As mentioned earlier, the aim is to develop a new general statistical rule learning methodology that utilizes semantic information as well. In the scope of the dissertation, the rules to be learnt are the grammars of natural languages, since these are the most complex rule systems and are therefore suitable for being the basis of a general model. For the grammar induction system investigated, a common formalism is needed to represent instance-level ontologies in conjunction with the corresponding symbolic descriptions of the agent's observations.

In terms of the present research, semantic models are represented by ECG diagrams, the nodes of which are labeled with ECG concepts, and the edges of which are labeled with semantic roles. The basic ECG diagram structures are shown in Figure 3.3 on page 43. Note here, that in practice the node and edge labels are ECG identifiers (see Chapter 5). Their symbolic names serve only better understanding.

Symbolic descriptions are given by a restricted NL, that is only expressions being definite and unambiguously interpretable are taken into account. Consequently, the sentences involved are factual assertions with true logical values. In terms of the actual task, the symbolic language applied should meet the following requirements:

- the range of linguistic phenomena covered by the symbolic language should not exceed that of the ECG-HOPL language (see in Section 3.3 on page 33);
- the symbolic language should consist of symbolic terms, referring to ECG concepts, and true symbolic assertions on terms; and
- each symbolic term should have exactly one conceptual representative within a description.

For practical purposes, the formalism that is able to represent semantic and symbolic information jointly, in a common framework – which will be the input of grammar induction – is desired to be a grammar formalism. By definition, grammar formalisms are

artificial languages whose purpose is to characterize precisely other artifical or natural languages. As mentioned in Section 2.3 on page 22, formal grammars describing natural languages lie between the context-free and context-sensitive grammars in the Chomsky hierarchy. A proposed formalism is tree-adjoining grammars (TAGs) [Joshi et al., 1975], which are mildly context-sensitive and have several variants. In NLP, the most influential ones are lexicalized TAG (LTAG) [Joshi & Schabes, 1997], feature-based TAG (FTAG) [Vijay-Shanker, 1987], and synchronous TAG (STAG), which is specifically designed for machine translation [Shieber & Schabes, 1990]. They are brought into focus for the purposes of the present research first of all because they satisfy the criterion of computational efficiency while covering a wide range of natural language constructions; and their tree structure resembles the ECG graph representation of ontology models.

The weak points of the original TAG formalism are the representation of discontinuous constructions and word order. In this chapter, it is shown that an adequate extension of the TAG formalism makes it applicable for the problem at hand, that is for being the common framework for the representation of ECG diagrams and the corresponding symbolic language units. However, the implementation of the algorithms developed will take place in a later stage of the project, directly connected to grammar induction.

4.1 Introduction to the TAG formalism

A $TAG(\mathcal{G})$ grammar is defined as a quadruple $\langle NT, TS, T(I), T(A)\rangle$, so that TS and NT are disjoint alphabets of terminals and nonterminals, respectively; $T(I)$ is a finite set of initial trees, and $T(A)$ is a finite set of auxiliary trees. The trees in $T(I) \cup T(A)$ are called elementary trees. Initial trees have the characteristics that all internal nodes are labeled by nonterminals; and all leaf nodes are labeled by terminals, or by nonterminal nodes marked for substitution ($\downarrow$). Auxiliary trees possess the same characteristics as initial trees do, except for exactly one nonterminal node, called the foot node (marked by $*$), which must match the root node and which can only be used to adjoin the tree to another node.

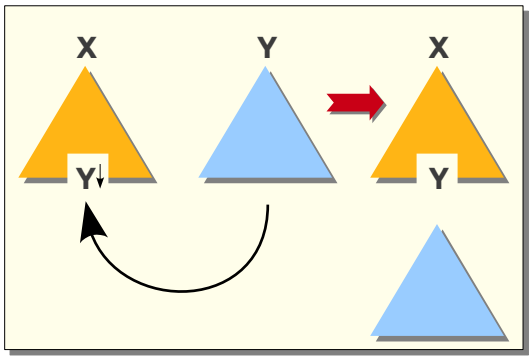

Figure 4.1 *Substitution operation*[3]

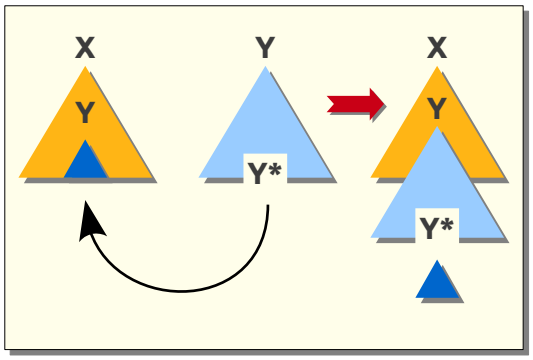

Figure 4.2 *Adjunction operation*[3]

[3] [Joshi & Schabes, 1997]

Two operations are defined in the TAG formalism, substitution and adjunction. In the substitution operation (Figure 4.1), the root node of an initial tree is merged into a nonterminal leaf node marked for substitution (↓) in another initial tree, where the merger nodes must match. In an adjunction operation (Figure 4.2), an auxiliary tree is grafted onto a nonterminal node anywhere in an initial tree. The root and foot nodes of the auxiliary tree must match the node of the initial tree at which the auxiliary tree adjoins. The subtree of the initial tree, the root of which is the node where the auxiliary tree adjoins, is removed from the initial tree, and the auxiliary tree is substituted for it instead; while this subtree is substituted in the foot node of the auxiliary tree.

The phrase-structure tree set of a $TAG(\mathcal{G})$, $\mathbf{T}(TAG(\mathcal{G}))$ is defined to be the set of all derived trees starting from **S**-type initial trees in $T(I)$ whose frontier consists of terminal nodes (all substitution nodes having been filled). The string language generated by a $TAG(\mathcal{G})$, $\mathcal{L}(TAG(\mathcal{G}))$ is defined to be the set of all terminal strings on the frontier of the trees in $\mathbf{T}(TAG(\mathcal{G}))$.

4.2 Grammar representation of the semantic model

Grammars, and theories of grammar, can be classified according to whether the basic unit of sentence structure is the phrase, or the dependency between two sentence elements. CFG and TAG are examples of phrase-structure grammars. On the other hand, dependency grammar (DG) [Tesniére, 1959] is a dependency-based linguistic approach to the description and analysis of natural language syntax. It is constituted by distinguishing a head-dependent asymmetry, and describing the relations between a head (usually the main verb) and its dependents in terms of semantically motivated dependency relations.

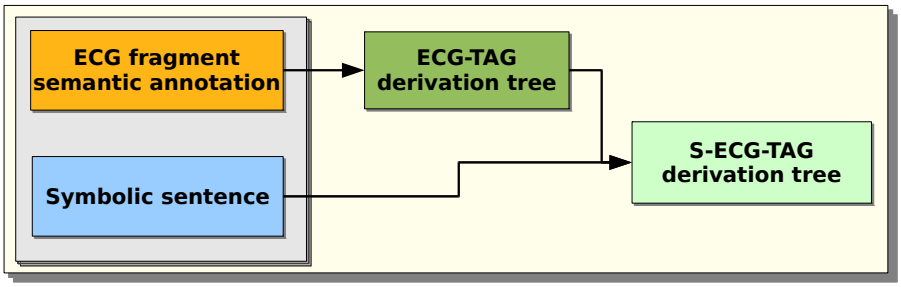

Figure 4.3 *Joint representation of annotation and symbolic description*

In NLP, the usual problem is how to map a syntactic dependency structure to a semantic one. In other words, how to make use of the result of syntactic analysis in semantic analysis. In the training phase of grammar induction, symbolic descriptions are assigned to instance-level ECG diagrams representing the agent's observations. Thus, the distinguishing feature of the present work is the aim of finding a mapping from semantic (conceptual) dependency structure represented by the ECG ontologies to a syntactic one. This means in practice that a correspondence is sought for between ECG concepts and the syntactic units of symbolic descriptions. The task can only be accomplished within a common framework

that is based on ECG and that combines the levels of semantics and syntax (see Figure 4.3). For this purpose, a new tree-based formalism has been developed. In order to find a matching between ECG concepts and the syntactic units of symbolic descriptions, an ECG diagram needs to be converted first into the newly defined tree structure the nodes of which are ECG concepts. The next step is then to find an algorithm that maps the syntactic units of the corresponding symbolic description to the leaf nodes of the resulting derivation tree.

4.2.1 Analysis of ECG diagram graphs

In the scope of the dissertation, semantic models are represented by ECG diagrams, the nodes of which are labeled with ECG concepts, and the edges of which are labeled with semantic roles.

Definition 2. *An ECG diagram $\Gamma = \langle V, A, R \rangle$ is a directed edge-labeled graph, where $v \in V$ vertices are ECG concepts, and $a \in A$ edges (arrows) are semantic relations labeled with $r \in R$ semantic roles.*

Table 4.1 *Incoming and outgoing edges of ECG diagram vertices*

Vertex type	Incoming edges	Outgoing edges
Instance category concept (FICN, FICT, FICR)	≥ 1 semantic role relations	≥ 1 isa relations
Primary class category concept (FMCR)	≥ 1 any relations	≥ 0 isa relations
Abstract class category concept (AMCR)	≥ 1 isa relations and ≥ 0 semantic role relations	≥ 0 isa relations
Kernel predicate concept (FMPR*)	none	≥ 1 semantic role relations and ≥ 0 isa relations
Primary predicate concept (FMPR)	≥ 0 semantic role relations	≥ 1 semantic role relations and ≥ 0 isa relations
Abstract predicate concept (AMPR)	≥ 1 isa relations and ≥ 0 semantic role relations	≥ 0 any relations

Table 4.2 *Edges connecting ECG diagram vertices*

Relation type	Source vertex	Target vertex
Specialization	FICN, FICT, FICR	FMCR
	FMCR, AMCR	AMCR
	FMPR, AMPR	AMPR
Semantic role	FMPR, AMPR	FICN, FICT, FICR
	FMPR, AMPR	FMCR, AMCR
	FMPR, AMPR	FMPR, AMPR

Within the diagrams of ECG fragments, $d^-_{(\Gamma)}(v_{kernelpredicate}) = 0$, that is the number of incoming edges into vertices representing kernel predicate concepts equals 0. Otherwise,

there may be more than 1 parent nodes (i.e. incoming edges) for any other node. Also, the number of outgoing edges from any vertex is arbitrary. The incoming and outgoing edges of the vertices are summarized in Table 4.1, while Table 4.2 shows which vertex types are allowed to be connected by each edge type.

By definition, an ontology is based on a taxonomy, that is a hierarchy of concepts. Since ECG is an ontology model, the next statement holds.

Statement 4. *The diagram of a primary-level ECG fragment – representing an actual observation of the agent – is an acyclic graph, thus it can be converted to a tree structure the root of which is the kernel predicate.*

4.2.2 Definition of the ECG-TAG formalism

For the tree-based representation of ECG diagrams, first of all the TAG formalism needs to be amended by edge labels. Furthermore, the new ECG-TAG model re-defines the trees (nodes and edges) of the TAG formalism, so that node and edge labels are coming from ECG ontologies. On the other hand, the new formalism inherits the basic TAG operations of substitution and adjunction.

Definition 3. *$ECG\text{-}TAG(\mathcal{G}) = \langle V, E, R^+, T(S), T(I), T(A)\rangle$ is an ECG-TAG grammar, so that V is the set of vertices where $V = C \cup \{\mathbf{S}\}$, where C is a finite set of ECG concepts and $\mathbf{S}$ is the start symbol; E is the set of edges where $E = RS \cup \bar{E}$, where RS is a finite set of ECG relationships and $\bar{E}$ is a finite set of edges for connecting predicate concepts; edges in E are labeled with elements from $R^+ = R \cup \{predicate\}$ respectively, where R is a finite set of semantic roles; $T(S)$ is the single-element set of the base $\mathbf{S}$-type tree, $T(I)$ is a finite set of initial trees, and $T(A)$ is a finite set of auxiliary trees.*

The C set of ECG concepts consists of two subsets: CC denotes the subset of category concepts, while PC is the subset of predicate concepts. Similarly, the RS set of ECG relationships can be divided into two subsets: RR denotes the subset of semantic role relations, while SR is the subset of specialization (isa) relationships. The ECG-TAG derivation trees are constructed from the following trees by using the operations of substitution and adjunction defined in the original TAG formalism.

- The root node of the base **S**-type tree is denoted by **S** (start symbol). One leaf node of the tree is the kernel predicate from PC, while all the other leaf nodes are elements of C and the edges are elements of $RR \cup \{\bar{e}\}$, where $\bar{e} \in \bar{E}$. The leaf nodes may be marked for substitution ($\downarrow$).
- Initial trees are constructed on the basis of the specialization relationships in the underlying ECG ontology. Therefore the nodes are elements of C and the edges are elements of SR; and the leaf nodes may be marked for substitution ($\downarrow$).
- Auxiliary trees have exactly one foot node (marked $*$), which must match the root node and which can only be used to adjoin the tree to another node of another

tree. Auxiliary trees are constructed on the basis of the non-kernel predicates of the underlying ECG ontology. Therefore one leaf node of each auxiliary tree is a non-kernel predicate from PC, while all the other leaf nodes are elements of C and the edges are elements of $RR \cup \{\bar{e}\}$, where $\bar{e} \in \bar{E}$; and the leaf nodes may be marked for substitution ($\downarrow$).

Statement 5. *The resulting formalism is, at the same time, dependency-based in the sense that edge labels represent semantic dependency relations.*

In the ontology model, the kernel predicate is seen as the highest level concept (in the scope of a fragment), governing a set of complements, which govern their own complements themselves. Therefore each semantic relation represents the local semantic dependency of the child node from its parent node.

4.2.3 Mapping ECG diagram into ECG-TAG formalism

Mapping ECG diagram structures into ECG-TAG structures involves the mapping of ECG concepts and relationships, so that ECG concepts correspond to nodes and relationships correspond to labeled edges in the ECG-TAG trees; where an ECG-TAG derivation tree describes an ECG fragment. Other ECG container elements can be represented by the combinations of derivation trees.
This section examines the assignment $\mathbf{g} : \mathit{ECG\ diagram} \rightarrow \mathit{ECG\text{-}TAG}$. The assignment is said to be a mapping, if every ECG diagram has a corresponding ECG-TAG structure. This mapping is surjective, if also every ECG-TAG structure has a corresponding ECG diagram. It is bijective, if the mapping is mutually unambiguous. For the analysis, the basic ECG diagram structures (see Figure 3.3 on page 43) are considered and mapped into ECG-TAG structures.

1) Representation of predicates and arguments.
When representing predicate schemas, there is a difference between the mapping of the kernel predicate and other predicates. Namely, the kernel predicate is always included in the base **S**-type ECG-TAG tree, while other predicates form parts of separate ECG-TAG auxiliary trees the roots (and the identical foot nodes) of which are arbitrary leaf nodes in other trees. See Figure 4.4.

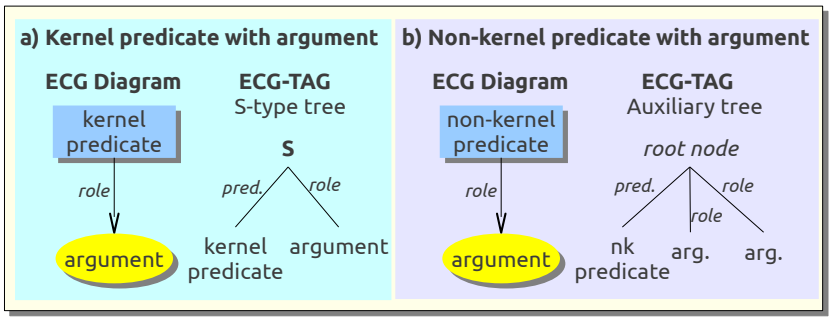

Figure 4.4 *Mapping ECG diagram predicates and arguments to ECG-TAG*

2) Assigning concepts to higher-level concepts.

Any concept can be assigned to a higher-level concept of corresponding type through a specialization relationship. This kind of structure shows differences in the ECG diagram, because of the dissimilar representation of predicate and non-predicate concepts. However, their mapping to ECG-TAG yields the same initial tree structure with distinct nodes. See the illustration in Figure 4.5.

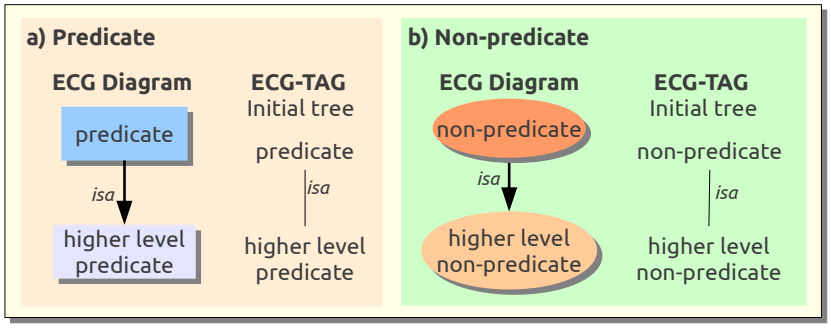

Figure 4.5 *Mapping ECG diagram specialization relationships to ECG-TAG*

3) Representing predicate as argument.

In the ECG model, non-kernel predicates can be arguments of other predicates. Again, there is a difference in mapping between the cases when the kernel predicate has a predicate argument, or when a non-kernel predicate has a predicate argument. See Figure 4.6. The mapping of this structure resembles the case in 1).

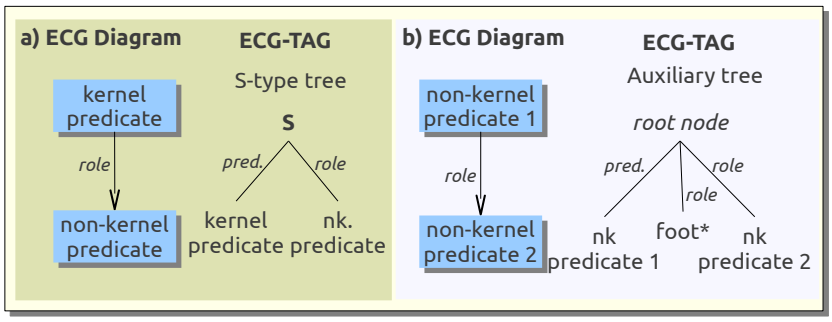

Figure 4.6 *Mapping ECG diagram predicates as arguments to ECG-TAG*

4) Representing common arguments.

Figure 4.7 shows a complex predicate schema, where the kernel predicate and a non-kernel predicate share the same argument. This structure could be further complicated by assuming that both predicates are non-kernel predicates. In this case two auxiliary trees need to be constructed, where the common argument is the root and foot nodes in one of them. The root and foot nodes of the other auxiliary tree are not specified in the example.

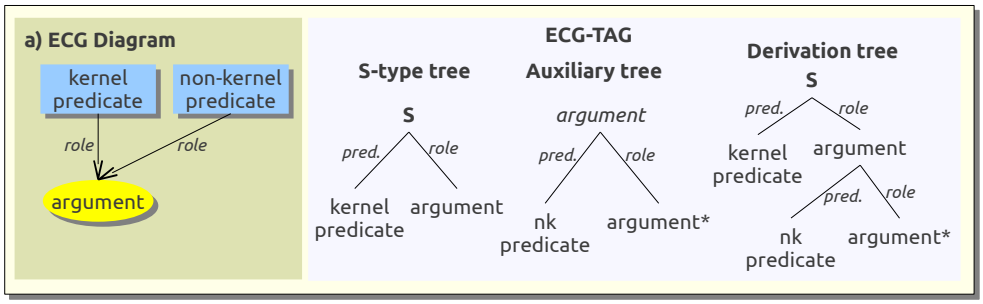

Figure 4.7 *Mapping ECG diagram komplex predicate schemas to ECG-TAG*

5) Representing groups of arguments.

Figure 4.8 displays the mapping of argument groups. Again, a distinction should be made between the case when the group of arguments is connected to the kernel predicate, and the case when it belongs to a non-kernel predicate. This structure should be extended with the remark that non-kernel predicates can also be elements of argument groups.

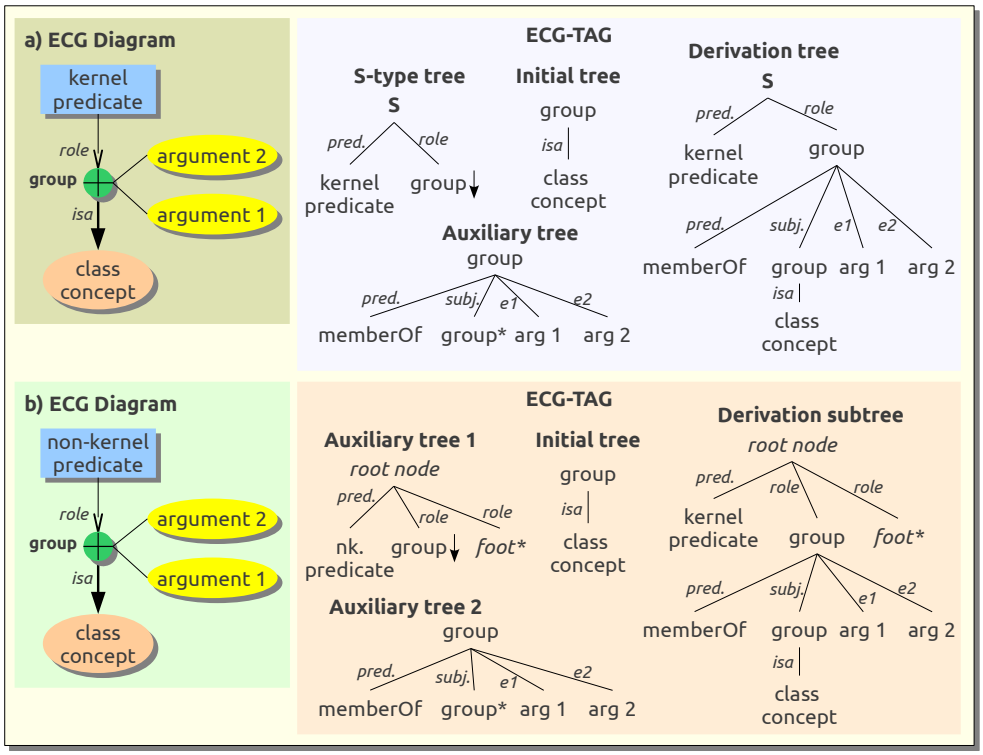

Figure 4.8 *Mapping ECG diagram groups of arguments to ECG-TAG*

Statement 6. *This analysis shows that every ECG diagram structure can be mapped into a corresponding ECG-TAG structure, that is* **g** *is a lossless mapping.*

If ECG diagram constituents (concepts and relationships) are given by their category types, an ECG-TAG structure may have more than one corresponding ECG diagram structures. In this case **g** is a surjective mapping. On the other hand, if ECG diagram components are given by their identification codes, every ECG-TAG structure has exactly one ECG diagram correspondent. In this case **g** is a bijective function, therefore

$\mathbf{g^{-1}}$: *ECG-TAG* $\rightarrow$ *ECG diagram* also exists that is again an unambiguous mapping. See the difference between the two representations of ECG elements in Chapter 5.

The algorithms specifying the steps of mapping an ECG fragment diagram into an ECG-TAG derivation tree can be given as follows.

1. Construct a base **S**-type tree with root **S** (see Algorithm 4.1), the leaf nodes of which are the kernel predicate and the (arbitrary number) connected concepts. The edges are labeled according to the semantic role relationships. The tree is built up from left to right so that the first node is the predicate concept with the label 'predicate'. The order of the other nodes is arbitrary.

Input: ECG fragment diagram
Output: **S**-type ECG-TAG tree
Stree.root = **S**
$i = 1$
Stree.leaf[i] = kernelpredicate
Stree.edge[i] = 'predicate'
relation[ ] = {Find all role relations where domain = kernelpredicate}
foreach $j \in$ *relation* **do**
    $i = i + 1$
    Stree.leaf[i] = relation[j].range
    Stree.edge[i] = relation[j].label

Algorithm 4.1 *Creation of the base* **S***-type ECG-TAG tree*

2. Create the initial tree set based on the specialization relations in the ECG diagram (see Algorithm 4.2).

Input: ECG fragment diagram
Output: ECG-TAG initial tree set
relation[ ] = {Find all isa relations}
foreach $j \in$ *relation* **do**
    Itree[j].root = relation[j].domain
    Itree[j].leaf = relation[j].range
    Itree[j].edge = 'isa'

Algorithm 4.2 *Creation of the ECG-TAG initial tree set*

3. Create the auxiliary tree set (see Algorithm 4.3) on the basis of the non-kernel predicate concepts having arguments in the ECG diagram. These trees are built up from left to right so that the first node is the predicate concept, and the edges are labeled according to the semantic role relationships. The root of the auxiliary tree must match a leaf node in another tree, and the frontier node with the same label is its foot node.

```
Input: ECG fragment diagram, S-type ECG-TAG tree, ECG-TAG initial tree set
Output: ECG-TAG auxiliary tree set
nodelist = {Find all leaf nodes in Stree and Itree[ ]}
i = 1
nkp_nodes[ ] = {Find all nonkernel predicates in ECG diagram}
foreach k ∈ nkp_nodes do
    relation[ ] = {Find all role relations in ECG diagram where domain = nkp_nodes[k]}
    if relation is not empty then
        Atree[k].leaf[i] = nkp_nodes[k]
        Atree[k].edge[i] = 'predicate'
        if nkp_nodes[k] is in nodelist then
            Atree[k].root = nkp_nodes[k]
        foreach j ∈ relation do
            i = i + 1
            Atree[k].leaf[i] = relation[j].range
            Atree[k].edge[i] = relation[j].label
            if Atree[k].root is empty AND relation[j] is in nodelist then
                Atree[k].root = relation[j].range
```

Algorithm 4.3 *Creation of the ECG-TAG auxiliary tree set*

4. Build the derivation tree (see Algorithm 4.4) starting from the **S**-type tree, by applying the operations of tree adjoining and substitution successively, in this order. Both operations are implemented as functions returning the modified versions of the trees given as their first argument. The returned tree structures are used in the succeeding iteration step. During tree adjoining, it may happen that an auxiliary tree cannot be adjoined for the first time. In this case the tree must be added to the end of the auxiliary tree set to get another chance to be adjoined later.

```
Input: ECG-TAG S-type tree, ECG-TAG initial tree set, ECG-TAG auxiliary tree set
Output: ECG-TAG derivation tree
foreach i ∈ Atree do
    foreach j ∈ Dtree.leaf do
        if Atree[i].root == Dtree.leaf[j] then
            Adjoin(Dtree.leaf[j], Atree[i])
            break
    Atree = Atree + Atree[i]
foreach i ∈ Itree do
    foreach j ∈ Dtree.leaf do
        if Itree[i].root == Dtree.leaf[j] then
            Substitute(Dtree.leaf[i], Itree[j])
            break
```

Algorithm 4.4 *Construction of the ECG-TAG derivation tree*

The mapping algorithms can be evaluated by the number of execution steps. The size of the input ECG fragment diagram can be characterized by the number of ECG diagram elements (n), amongst which the number of concepts and the number of relations can be approximated by $n/2$, respectively. Using these estimations, the **S**-type ECG-TAG tree and also the ECG-TAG initial tree set are created with a cost of $O(n)$. For the number

of elements in the **S**-type ECG-TAG tree (k), and for that of the ECG-TAG initial tree set (l) hold, that $k \leq n$ and $l \leq n$. Thus, the ECG-TAG auxiliary tree set is constructed with a cost of $2n + n + (n/2)n + (n/2)^2 \approx O(n^2)$.
The ECG-TAG derivation tree is constructed from the tree set generated in Algorithms 1-3. The number of trees in the ECG-TAG initial tree set must be equal to the number of specialization relationships in the ECG diagram, which is $n/2$. Similarly, the number of trees in the ECG-TAG auxiliary tree set must equal the number of non-kernel predicates in the ECG diagram, which is approximated by $n/2$. However, in the case of tree adjoining the sequence in which the auxiliary trees are adjoined does matter. The algorithm solves this problem in a way that virtually doubles the size of the auxiliary tree set to n in the worst case. The number of leaf nodes in the ECG-TAG derivation tree must be less than or equal to the number of concepts in the ECG diagram, which is $n/2$. Therefore the cost of the algorithm constructing the ECG-TAG derivation tree can be given as $n(n/2) + (n/2)^2 \approx O(n^2)$.

The cost of mapping ECG fragment diagrams into ECG-TAG derivation trees results from the summation of the costs of Algorithms 4.1-4.4, which yields an approximation of $O(n^2)$. Appendix C shows two examples of ECG-TAG derivation tree construction.

4.3 Grammar representation of the symbolic description

4.3.1 Representation of symbolic language

The symbolic representation of ECG model fragments – i.e. of the agent's observations – is defined by a symbolic language, which is specified by the following characteristics.

- It is a restricted NL, that is only expressions being definite and unambiguously interpretable are taken into account. Consequently, the sentences involved are factual assertions with true logical values.
- The range of linguistic phenomena covered by the symbolic language is the same as that of the ECG-HOPL language (see in Section 3.3 on page 33).
- It consists of symbolic expressions of two kinds:
  - symbolic terms, referring to ECG concepts, and
  - true symbolic assertions on terms, i.e. sentences.

The ECG model in Figure C.1 can be given by several true symbolic assertions, e.g.:

1. A circle is in a triangle.
2. A circle is located in a triangle.
3. A black circle can be found in the triangle.
4. A black colored circle is in a triangle to be found.
5. In a triangle stands a circle.
6. There is a circle in the triangle.

The symbolic terms in these examples and the referred ECG concepts are listed in Table 4.3. From this, the following characteristics of symbolic terms can be inferred:

- they may be multi-word representations, or
- their components may not appear continuously.

It is worth mentioning here that there may be ECG concepts that do not have a representant at the symbolic level at all.

Statement 7. *The assignment of symbolic terms to ECG concepts, that is* $\mathbf{h} : ST \rightarrow C$ *is a mapping which is neither surjective, nor injective. Namely, from the characteristics of symbolic terms follows that* $\forall st \in ST \;\; \exists c \in C$*, while* $\neg \forall c \in C \;\; \exists st \in ST$*.*

Since, in symbolic expressions both the symbols and the arrangement of symbols communicate meaning, the following problems need to be addressed:

- the mapping of symbolic terms into ECG concepts, and
- the arrangement (order) of symbolic terms.

Table 4.3 *Correspondence of ECG concepts and symbolic terms*

ECG concept	Symbolic term
FMPR_1: isIn	is in is located in can be found in is in . . . to be found in . . . stands there is . . . in
FMPR_2: hasColor	colored
FMPR_3: hasColor	–
FMCR_1: Circle	a circle a . . . circle
FMCR_2: Triangle	a triangle the triangle
FMCR_3: Black	black
FMCR_4: White	–

4.3.2 Definition of the S-ECG-TAG formalism

For handling the problem of mapping symbolic terms into ECG concepts, which are the nodes of the ECG-TAG derivation trees, each ECG-TAG derivation tree (representing the semantic level) needs to be vertically expanded by a symbolic level. As a consequence of the characteristics of symbolic terms, at this level one or more nodes may correspond to each symbolic term, which are all linked to the leaf nodes of the semantic level. This extended formalism combines the levels of semantics and syntax, and got the name S-ECG-TAG, that is the ECG-TAG formalism is extended with a symbolic level.

Formally, the S-ECG-TAG formalism extends the ECG-TAG formalism with the SN finite set of symbolic-level nodes. A $sn \in SN$ symbolic-level node is represented by a finite sequence of words delimited by spaces, i.e. $sn = w_1 \quad w_2 \quad \ldots$, where w words are finite sequences over Σ, where Σ denotes the finite character set of the symbolic language. A $st \in ST$ symbolic term corresponding to an ECG concept (and an ECG-TAG semantic-level node) is defined as a finite set of symbolic-level nodes, that is $st = \langle sn_1, sn_2, \ldots \rangle$ which can be an empty set.

The edges connecting symbolic-level nodes to ECG-TAG semantic-level leaf nodes are labeled by precedence relations representing the true order of word sequences in the corresponding symbolic sentence.

Definition 4. *S-ECG-TAG*$(\mathcal{G}) = \langle V, E, R^{+n}, T(D) \rangle$ *is a S-ECG-TAG grammar, so that* V *is the set of vertices where* $V = C \cup \{\mathbf{S}\} \cup SN$*, where* C *is a finite set of ECG concepts,* $\mathbf{S}$ *is the start symbol and* SN *is a finite set of symbolic-level nodes;* E *is the set of edges where* $E = RS \cup \bar{E} \cup \tilde{E}$*, where* RS *is a finite set of ECG relationships,* $\bar{E}$ *is a finite set of edges for connecting predicate concepts and* $\tilde{E}$ *is a finite set of edges for connecting symbolic-level nodes; edges in* E *are labeled with elements from* $R^{+n} = R \cup \{predicate\} \cup \{n_1 \ldots n_k\}$ *respectively, where* R *is a finite set of semantic roles; and* $T(D)$ *is the single-element set of the derivation tree.*

4.3.3 Assignment of symbolic terms to ECG concepts

The aim is to represent symbolic terms and ECG concepts in a common formalism. After mapping ECG diagrams to ECG-TAG trees, ECG concepts are the nodes of the corresponding ECG-TAG derivation tree. The actual task is to develop an algorithm that automatically assigns the terms of the corresponding symbolic sentence to the leaf nodes of the ECG-TAG derivation trees and codes word order locally. This mapping algorithm is based on statistical methods, that is there are no predefined global rules for the assignment and the agent has no apriori knowledge about the grammar of the symbolic language. The agent does not perform the syntactic and morphological analyses of the symbolic description, either. The algorithm that solves the problem involves the following steps (see Algorithm 4.5).

1. Select an ECG-TAG derivation tree ($T(D)_{base}$) and the corresponding symbolic sentence the units of which are to be mapped into ECG concepts.
2. Search in the S-ECG-TAG database for matching patterns. If the search fails, continue with the next step.
3. Find a set of reference ECG-TAG derivation trees ($\mathbf{T}(\mathbf{D})_{ref}$) so that the following requirements are met.
    - The number of reference ECG-TAG derivation trees in $\mathbf{T}(\mathbf{D})_{ref}$ should be equal to the number of ECG concepts in $T(D)_{base}$.

- The reference ECG-TAG derivation trees should be selected in accordance with the criterion that the number of ECG concepts in each $T(D)_{ref} \in \mathbf{T}(\mathbf{D})_{ref}$ equals the number of ECG concepts in $T(D)_{base}$.
- Each $T(D)_{ref} \in \mathbf{T}(\mathbf{D})_{ref}$ should differ from $T(D)_{base}$ in exactly one ECG concept, so that all other $T(D)_{ref} \in \mathbf{T}(\mathbf{D})_{ref}$ contains this ECG concept.

```
Input: ECG-TAG derivation tree and its symbolic description
Output: S-ECG-TAG derivation tree

if Find(DT-base, S-base, S-ECG-TAG-DB) then
    return S-ECG-TAG
else
    S-ECG-TAG = DT-base
    words[ ] = Decompose(S-base)
    repeat
        failure = false
        if Generate(DT-ref[ ],S-ref[ ]) then
            foreach i ∈ DT-ref do
                diff-concept = Compare(DT-base, DT-ref[i])
                diff-units[ ] = Compare(S-base, S-ref[i])
                foreach j ∈ diff-units do
                    if diff-units[j] is not in a symbolic node then
                        CreateSymbolicNode(diff-units[j], S-ECG-TAG)
                        edgelabel = Search(diff-units[j], S-base)
                        LinkSymbNode(diff-units[j], diff-concept, edgelabel, S-ECG-TAG)
                    else
                        failure = true
                        break
            foreach k ∈ words do
                if words[k] is not in a symbolic node then
                    if words[k + 1] is in a symbolic node and linked to a category concept
                    then
                        ModifySymbNode(attach words[k] to words[k + 1] in S-ECG-TAG)
                    else
                        failure = true
        else
            exit
    until failure is true
```

Algorithm 4.5 *Generation of the S-ECG-TAG derivation tree*

4. Compare $T(D)_{base}$ to each $T(D)_{ref} \in \mathbf{T}(\mathbf{D})_{ref}$. In each iteration step, the two ECG-TAG derivation trees differ in only one node and consequently the differing symbolic word sequence(s) (which are determined by string comparison) should correspond to the symbolic terms representing the differing nodes. In the S-ECG-TAG derivation tree each word sequence is located in a separate symbolic-level node which should be linked to the corresponding semantic-level node. The edges connecting symbolic-level nodes to semantic-level nodes need to be labeled according to the order of word sequences in the symbolic sentence.

5. A problematic phenomenon is the handling of articles. In most cases, articles do not change together with the symbolic representants of ECG concepts. Thus, they are not assigned to any of the semantic-level nodes. To solve this problem, the algorithm uses the following assumption. If – in the symbolic sentence – the word directly preceding the symbolic representant of an ECG category concept is not assigned to any of the S-ECG-TAG symbolic-level nodes, then this word should be concatenated with the symbolic term representing the ECG category concept examined.

6. If not all symbolic sentence units could be unambiguously located in one of the nodes of the S-ECG-TAG derivation tree, select a new $(\mathbf{T}(\mathbf{D})_{ref})$ reference set and repeat steps 4 and 5. If a reference set cannot be generated, select another ECG-TAG derivation tree $(T(D)_{base})$ and execute the algorithm again.

If s denotes the size of the S-ECG-TAG database, m is the number of words in the symbolic sentence attached to the ECG-TAG derivation tree under examination, and n denotes the number of leaf nodes in this tree, the cost of the algorithm is calculated as $s + (n \times (m/n)) + m$, where m/n is the average number of sentence units (word sequences) per leaf node. Initially, when the S-ECG-TAG database includes none or a small number of successful assignments, the performance of the algorithm depends to a large extent on the method of selecting an appropriate reference ECG-TAG derivation tree set. Later on, as the size of the S-ECG-TAG database increases, the chance of finding matching patterns increases too, and thus the number of execution steps of the algorithm can be more precisely characterized by the cost of searching for an exact match in the S-ECG-TAG database. Appendix D shows an example assignment.

4.3.4 Learning word orderings

Knowledge of a given language involves knowing the vocabulary and the rules for building expressions from vocabulary items. In the actual task, word sequences compose the vocabulary of the given language that are stored in the symbolic-level nodes of the S-ECG-TAG derivation trees. Here, the labels of the edges connecting symbolic-level nodes to semantic-level nodes represent a valid ordering of the given word sequences.

The task of grammar induction is to infer a grammar (a set of rules) that can generate all the valid sentences in the given symbolic language. For this, the list of valid orderings needs to be generated from the S-ECG-TAG derivation tree database. Taking a valid ordering, the sentence determines a valid sequence of state transitions where a state corresponds to a word sequence (stored in a symbolic-level node sn). It is assumed that every state may occur only once, there is no repetition. If sn_i denotes a state, the sequence of n events is represented as [Jurafsky & Martin, 2000]: $sn_1, \ldots, sn_n$ or sn_1^n. The probabilities of sequences can be computed by using the chain rule of probability for decomposition [Manning & Schütze, 1999]:

$$P(sn_1^n) = P(sn_1)P(sn_2|sn_1)P(sn_3|sn_1^2)\ldots P(sn_n|sn_1^{n-1}) = \Pi_{k=1}^{n} P(sn_k|sn_1^{k-1}). \quad (4.1)$$

The bigram model is an N-gram model where the probability of a state is approximated by the preceding state [Jurafsky & Martin, 2000]:

$$P(sn_n|sn_1^{n-1}) \approx P(sn_n|sn_{n-1}). \tag{4.2}$$

The property that the probability of a state depends only on the previous state is called Markov property and has the following formulation [Krenn & Samuelsson, 1997]:

$$P(\zeta_{t+1} = sn_{i_{t+1}}|\zeta_1 = sn_{i_1}, \ldots, \zeta_t = sn_{i_t}) = P(\zeta_{t+1} = sn_{i_{t+1}}|\zeta_t = sn_{i_t}), \tag{4.3}$$

where $\zeta_1, \ldots, \zeta_n$ are random variables. If the probability of a state depends only on the previous state, the model is called first-order Markov model. The Markov model method [Markov, 1913] is essential in speech recognition, handwriting recognition, machine translation, spelling correction, part-of-speech tagging, natural language generation and in any tasks where words have to be identified from noisy, ambiguous input.
In a Markov chain each state is associated with a finite set of signals. After each transition, one of the signals is emitted. In the present model, the signal-set of a state contains only one signal that corresponds to the state itself. As an example, consider the following two sentences where the identified word sequences are separated by brackets.

1. (A black) (circle) (stands in) (a white) (triangle).
2. (A white) (triangle) (includes) (a black) (circle).

The state transition matrix related to these sentences can be found in Table 4.4. In this matrix, the **S** symbol denotes the start state, the beginning of the sentence. The rows of the matrix correspond to the initial states and the columns denote the end states. The values in the matrix cells are equal to the frequency of the corresponding state transition. Taking the restriction that a state can not be repeated, this matrix will generate exactly the given training sentences.

Table 4.4 *State transition matrix for the given sentences*

	S	a black	circle	stands in	a white	triangle	includes
S		1			1		
a black			2				
circle				1			
stands in					1		
a white						2	
triangle							1
includes		1					

4.4 General assessment

The S-ECG-TAG grammar formalism is an edge-labeled lexicalized tree-based representation having two levels. At the semantic level the nodes correspond to ECG concepts, while the edges represent ECG relationships (specialization or semantic role relations). At the symbolic level the nodes include word sequences, while the edges are labeled by precedence relations representing the true order of word sequences in the corresponding symbolic sentence. The most characteristic features of the formalism can be summarized as follows.

1. The new formalism is TAG-based because it uses the same tree set (with different interpretation) and the same operations for tree construction as the original TAG formalism.
2. At the same time, it is also dependency-based in the sense that edge labels represent semantic dependency relations (at the semantic level) (*Statement 5*).
3. The S-ECG-TAG formalism combines the levels of semantics and syntax. Semantic-level trees are constructed from ECG diagram graphs with an $O(n^2)$ algorithm where n is the number of ECG diagram elements (*Statements 4 and 6*). Symbolic terms are then assigned to the semantic-level tree nodes by an incremental learning process.
4. At the symbolic level, the formalism can represent discontinuous constructions by sibling nodes, and can encode word order locally for each node by edge labels.

4.5 Related work

The weak points of the original TAG formalism are the representation of discontinuous constructions and word order. [Rambow & Joshi, 1997] compares TAGs and dependency grammars from the perspective of word order variation. It has been observed that the TAG derivation tree resembles dependency structure, so it can be used as a dependency tree. In TAG-based dependency grammar [Joshi & Rambow, 2003] the idea is to define a grammar with elementary dependency trees which encode both dependency and word order, and which are combined using well-defined new operations. This approach means that all word order phenomena are expressed locally in the elementary trees; there are no global word order rules. The trees (in their graphical representation) fully specify the ordering of all the nodes in the tree. The work was originally inspired by the desire to co-locate more than one word in a structure in order to represent multi-word lexemes and idioms.

The TAG formalism was originally designed for the representation of syntax. In recent years, Kallmeyer and her collaborators have been involved in research on logic-based TAG semantics [Kallmeyer, 2002] through a series of papers using LTAG, FTAG, multicomponent TAG (MTAG) and STAG. They work on a syntax-semantics computation system, defining a set of desirable properties that such a system should have [Kallmeyer, 1997]:

1. computation of the semantics of a sentence should rely only on the relationships expressed in the TAG derivation tree;
2. the generated semantics should compactly represent all valid interpretations of the input sentence; and
3. the formalism should not increase the expressivity of the TAG formalism.

The present approach is the opposite of the one outlined in [Kallmeyer, 1997], by computing a syntactic level on the basis of semantics using statistical methods. Nevertheless, the system modeled can be evaluated according to the above mentioned requirements.

1. The generated syntactic level encodes the true structure of the corresponding symbolic sentence via edge labels.
2. A state transition matrix stores all the valid orderings of the word sequences.
3. The expressiveness of the S-ECG-TAG and the ECG-TAG formalisms is the same.

The TAG-based and dependency-based representations of the ECG model are thoroughly examined and described in [7]. In [6], a dependency-based mapping between symbolic language elements and ECG concepts is investigated, and an algorithm for learning the ordering and transformation of words is also given. [11] specifies an algorithm for inferring a PCFG grammar from unannotated positive linguistic data, that can serve control purposes in comparative analyses.

4.6 Summary of the results

In this chapter, the grammar formalism proposed for representing the semantic model developed and its symbolic language description in a common framework has been introduced, and the algorithms executing the assignments have been theoretically specified. Their implementation will be performed in a later stage of the research project, directly connected to grammar induction. The new scientific results can be summarized as follows.

Thesis 2. *[7]*
ECG fragment diagrams are acyclic graphs, therefore they can be converted to a tree structure the root of which is the kernel predicate. The mapping is proved to be lossless and is accomplished by an $O(n^2)$ algorithm, where n is the number of ECG diagram elements. The new ECG-TAG formalism consists of edge-labeled lexicalized tree structures, the nodes of which correspond to ECG concepts, while the edges represent ECG relationships. The formalism is TAG-based, because it uses the same tree set (with different interpretation) and the same operations for tree construction as the original TAG formalism. At the same time, it is also dependency-based in the sense that edge labels represent semantic dependency relations.

$< Statements\ 4, 5, 6 >$

Thesis 3.

The algorithm that performs the assignment of symbolic sentence units to ECG concepts results in a new grammar formalism, called S-ECG-TAG, which combines the levels of semantics and syntax. The formalism extends the ECG-TAG formalism with a symbolic level, where the nodes include word sequences, while the edges are labeled by precedence relations representing the true order of word sequences in the corresponding symbolic sentence. Hence, the symbolic level encodes word order locally and discontinuous constructions are represented by sibling nodes.

< *Statement* 7 >

Consequences

1. The S-ECG-TAG formalism can be applied as a common framework for representing ECG diagrams and the corresponding symbolic sentences.

2. The S-ECG-TAG formalism can be applied as a formal grammar to be learnt in the grammar induction process, because word sequences that are stored in the symbolic-level nodes of the S-ECG-TAG derivation trees compose the vocabulary of a given language, while the labels of the edges connecting symbolic-level nodes to semantic-level nodes represent a valid ordering of the given word sequences.

Chapter 5

Conceptualization Using ECG Diagram Graphs

Learning without thought is labor lost.

Confucius

In terms of machine learning, concept formation (or conceptualization) is the process by which an agent learns to sort specific experiences into general rules or classes. In order to make learning feasible in complex domains, abstraction and generalization operators are often applied to make the problem tractable. In [Ponsen et al., 2010], abstraction is given as a technique to reduce the complexity of a problem by filtering out irrelevant properties while preserving all the important ones necessary to still be able to solve a given problem. At the same time, generalization is defined as a technique to apply knowledge previously acquired to unseen circumstances or extend that knowledge beyond the scope of the original problem. Humans show great capability in abstracting and generalizing knowledge in everyday life. A learning agent needs abstraction and generalization as well to deal successfully with contemporary technological challenges, given the huge state and action spaces that characterize real world problems. As a consequence, abstraction and generalization have received significant attention in the machine learning research community, recently.

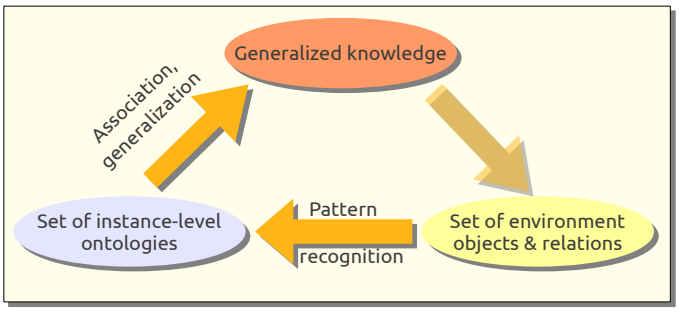

Figure 5.1 *Conceptualization in the grammar learning agent examined*

Projecting Peirce's approach (see Section 2.1 on page 7) to the actual task, the process of conceptualization in the grammar learning agent examined is shown in Figure 5.1. Here, the dynamic object is the set of objects and relations existing in the investigated agent's environment. The immediate interpretant is the set of observations taken by the agent (by detecting and mapping the objects and relations of the environment), which are given

by instance-level ontology models. Dynamic interpretants are created after the successive incorporation (association) and generalization of each perception. At the idealized end of the process stands the final interpretant, which is a generalized description of the environment elements observed. As can be seen, the two operations of association and generalization together accomplish the process of conceptualization.

For the simulation of this process, the grammar learning agent examined is endowed with the capabilities of pattern recognition, association and generalization. Thus, it is able to

- recognize and map the objects of its direct environment and their relations into knowledge base items;
- incorporate and relate the information elements observed to its existing (and continuously evolving) knowledge base; and
- create new (not observed) higher-level concepts by extracting the common characteristics of existing knowledge items.

The agent maps its observations into internal knowledge items which are instance-level ontologies represented by ECG diagram graphs. Generally speaking, association is the process of forming connections between new information elements observed and existing items in the knowledge base. In the case of ECG diagram graphs, association is defined as follows.

Definition 5. *Association is the process by which an ECG diagram graph is matched to and inserted in the knowledge base of previously matched and connected ECG diagram graphs.*

Within the framework of the dissertation, the notions of abstraction and generalization are not separated, they are implemented in one operation. Namely, generalization is considered as the process of extracting common significant elements by abstraction, where abstraction is the process by which new higher-level concepts are derived from literal concepts or other abstractions. For a given finite training set generalization is a finite gradual process. Therefore generalized knowledge can only be obtained through several stages. The ECG model defines three abstract elements which can be used in generalization, that is

- AMCR: abstract category concept,
- AMR: abstract semantic role relationship,
- AMPR: abstract predicate concept.

Table 5.1 *Abstract element insertion rules*

ECG graph element type	**Inserted abstract element type**
Category concept types (FICN, FICT, FICR, FMCR, AMCR)	AMCR
Semantic role relation types (FSR, FMR, AMR)	AMR
Predicate concept types (FMPR, AMPR)	AMPR

Thus, the process of generalization in ECG diagram graphs means the derivation of abstract concepts and relationships from corresponding primary-level or other abstract elements (see Table 5.1). Accordingly, within the ECG model generalization can be interpreted in three levels.

1. At the first level higher-level concepts can be revealed based on the common characteristics of existing concepts.
2. At the second level the substitutability of objects can be learned on the basis of the semantic role relations defined by the predicate.
3. At the third level the frequent predicate schemas can be explored.

In the dissertation the first level is examined, therefore the following definition is used for generalization.

Definition 6. *Generalization is the process by which new higher-level ECG concepts are created in the knowledge base of ECG diagram graphs incorporating the common significant (frequently occuring) characteristics of the existing concepts, where the derivation relationship is represented by specialization.*

This process reduces the complexity of the knowledge base of ECG diagram graphs, since the new higher levels include an ever decreasing number of concepts and relationships than the underlying layers.

The aim of this chapter is to show how the knowledge base of the grammar learning agent examined is built up of instance-level ontologies (represented by ECG diagram graphs), each describing a snapshot taken of the environment. For this, the two operations of association (i.e. matching and connecting ECG diagram graphs) and generalization which involves abstraction (i.e. creating higher-level concepts) must be defined on ECG diagram graphs. Prior to these definitions, a review of the literature concerning graph matching and generalization is exposed.

5.1 Related works

5.1.1 Graph matching

Generally speaking, the graph matching problem (for unlabeled undirected graphs) can be stated as follows. Given two graphs $G_1 = \langle V_1, E_1 \rangle$ and $G_2 = \langle V_2, E_2 \rangle$, the problem is to find a bijective mapping $f : V_1 \to V_2$ so that $(u, v) \in E_1$ iff $(f(u), f(v)) \in E_2$. When such a mapping exists, this is called an isomorphism, and G_1 is said to be isomorphic to G_2. If $|V_1| = |V_2|$, the problem is said to be exact graph matching. If $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$, the problem is said to be subgraph matching (or subgraph isomorphism). For labeled graphs, the following requirements also need to be met:

– the label of vertex u must be the same as that of $f(u)$ for all $u \in V_1$;
– the label of edge (u, v) must be the same as that of $(f(u), f(v))$ for all $(u, v) \in E_1$.

A graph matching problem is considered to be inexact when isomorphism cannot be expected and the matching aims at finding the best, not necessarily bijective correspondence between G_1 and G_2. The best correspondence of a graph matching problem is defined as the optimum of some objective function (fitness function) which measures the similarity between matched vertices and edges.
According to [Bodon, 2010], for the whole category of graph matching problems there does not exist a general polynomial time algorithm. What is more, it has not been proven yet that the complexity of the whole problem type is NP-complete. However, in the case of small specific graphs there may exist algorithms by means of which isomorphism can be decided in polynomial time. At the same time, the complexity of subgraph matching problems is proved to be NP-complete. The two most well-known algorithms for deciding subgraph isomorphism are backtracking [Ullmann, 1976] and Nauty [McKay, 1981].

[Carroll, 2001] discusses the matching of RDF graphs. The paper traces the problem back to the standard graph isomorphism problem, where a graph is considered to be unlabeled and undirected. It argues that, within RDF models it is possible to encode an unlabeled directed graph by using a single property label (e.g. $rdf : value$) for the edges and anonymous resources for each vertex. Undirected graphs can be obtained by encoding each edge of the graph as two RDF triples, one in each direction. In this way, the standard graph isomorphism algorithms, developed in the 1970's, can be used effectively for comparing RDF graphs.

Similarly, ECG diagrams are labeled directed graphs. They can be constructed as undirected graphs by encoding the direction of an edge in its label. Edge labeling can also be eliminated by encoding connections in vertex labels. Vertex labels, however, cannot be omitted, therefore the standard graph isomorphism algorithms are not applicable for the problem at hand. Since ECG diagrams are small specific graphs with rich labeling, it is assumed that there exists an algorithm that decides isomorphism within acceptable time.

5.1.2 Generalization

Formal concept analysis [Ganter & Wille, 1999] gave birth to the notion of concept lattices, which are used in many application areas to represent conceptual hierarchies among objects and can also be used for representing concept generalization structures. In the literature, there are two main variants of concept set building algorithms. The methods of the first group work in batch mode, assuming that every element of the context table is already present before starting the concept lattice building. The other group of proposals uses an incremental lattice building method. In this case, the concept set is immediately updated when the context is extended with a new object (see [Godin et al., 1995]).
One of the biggest problems in creating concept lattices is the large number of attributes. Most of the proposals in the literature cope with this problem by eliminating the attributes with low relevance value, thereby providing better efficiency at the expense of information loss. [Kovács, 2006] presents a lattice building algorithm that does not use attribute reduction. In this proposal, it is assumed that there exists a lattice (called attribute lattice)

containing the attributes from the objects. This lattice can be considered as a thesaurus with generalization relationships among the attributes, and can be used to improve the quality and usability of the lattice to be generated from the objects.

In terms of the actual task, ECG diagram graphs can be considered as objects and ECG diagram elements (concepts and relations) can be seen as their attributes. In this case, the attribute lattice shows the specialization and generalization relationships among ECG elements, and the approach of [Kovács, 2006] could be adapted for the classification of ECG diagram graphs.

[Fargues et al., 1986] describes a similar task to the one discussed in the dissertation. This paper outlines the principles of a Prolog-like deductive system based on Sowa's conceptual graphs (CGs) within which a generalization algorithm is implemented. This CG processor is the main component of the KALIPSOS general system for knowledge acquisition from texts that is developed at IBM Paris Scientific Center. The success of this project motivates the effort of simulating the process of conceptualization using ECG diagram graphs, specified in the following sections.

5.2 The problem of matching ECG diagram graphs

Association is the process by which an ECG diagram graph is matched to the knowledge base of previously matched and connected ECG diagram graphs. In other words, in this process an ECG diagram graph is compared to a graph created incrementally from ECG diagram graphs. Thus, the algorithm realizing this comparison belongs to the class of subgraph matching algorithms. Prior to the specification of the association operation which is based on the ECG diagram graph matching algorithm, the comparison of ECG diagram graph elements needs to be studied.

5.2.1 ECG element category type lattice

In the ECG model, concepts and relations (elements) are given by two attributes: a type and a caption, in the form of $type : caption$. Formally, each $ec \in EC$ element category in the ECG model is given as $ec = type_c : caption$, where $type_c \in T$ and $caption$ is a string representing the name of the corresponding element category. The T set of ECG element category types (see the ECG model terminology in Appendix A on page 101) is the union of the subsets listed in Table 5.2.

Table 5.2 *Classification of ECG element category types*

Subset of T	**Element category types**
T_{cc}: category concept types	FICN, FICT, FICR, FMCR, AMCR
T_{pc}: predicate concept types	FMPR, AMPR
T_{rr}: semantic role relation types	FSR, FMR, AMR
T_{sr}: specialization relation type	FMI

Statement 8. *ECG element category types can be merged in a lattice* $(T, <)$*, whose partial ordering relation* $<$ *can be interpreted as a categorical generalization relation.*

The top and the bottom element categories of the $(T, <)$ lattice are UNIV (the suprenum element) and NIL (the infinum element), respectively. The generated lattice is displayed in Figure 5.2. For reading the lattice, the notion of 'restriction' must be introduced. Thus, we say that an ECG element category ec_2 can be 'restricted to' another ECG element category ec_1, or, in other words, ec_1 is a 'restriction of' ec_2 if $type_{c_1} < type_{c_2}$ holds according to the element category type lattice.

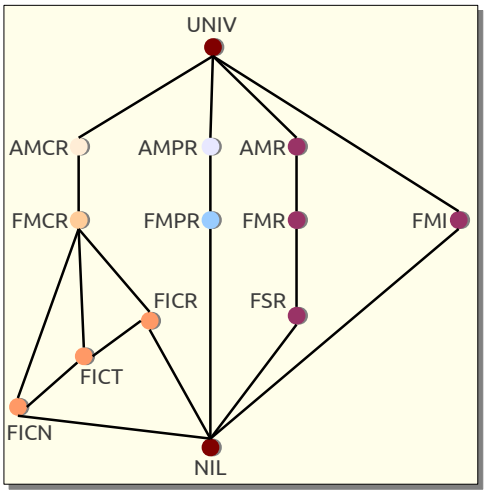

Figure 5.2 *ECG element category type lattice*

Based on the lattice, it is possible to exhibit the least common generalization lcg, and the greatest common specialization gcs of two element categories ec_1 and ec_2:

$$lcg(ec_1, ec_2) = \min\{type_c \mid type_{c_1} <= type_c \text{ and } type_{c_2} <= type_c\}; \tag{5.1}$$

$$gcs(ec_1, ec_2) = \max\{type_c \mid type_c <= type_{c_1} \text{ and } type_c <= type_{c_2}\}. \tag{5.2}$$

5.2.2 Correlations between ECG element instances

Analogously to the definition of ECG element categories, in an instance-level ECG diagram graph each $ei \in EI$ element instance is given as $ei = type_i : caption$, where $type_i \in T'$ and $caption$ is a string representing the name of the corresponding element instance. The members of T' are constructed from the members of T augmented with a number. Thus, an element instance type has the form $type_i = type_c\_n$, where $type_c$ is the corresponding element category type and n is a numeric code, so that $type_i = type_c\_n$ is a unique identifier within the problem domain. In this way, element instances in an ECG diagram graph can be unambiguously identified by their type attributes alone. The caption attribute serves only better understanding and legibility. The element category type part of an element instance type is denoted by $[type_i] = type_c$, while the numeric code of an element instance type can be obtained as $\{type_i\} = n$.

Given two element instances ei_1 and ei_2, on the basis of the element category type lattice the following $\varphi \in \Phi$ correlations can be identified between them.

1. Semantic comparability:

$$ei_1 >< ei_2 \text{ if } lcg([type_{i_1}], [type_{i_2}]) \neq UNIV. \tag{5.3}$$

   The set of element instance pairs generated by $\varphi =$ ' $><$ ' is denoted by SC.

2. Semantic incomparability:

$$ei_1 <> ei_2 \text{ if } lcg([type_{i_1}], [type_{i_2}]) = UNIV. \tag{5.4}$$

   The set of element instance pairs generated by $\varphi =$ ' $<>$ ' is denoted by SNC.

3. Category-equivalence:

$$ei_1 \sim ei_2 \text{ if } [type_{i_1}] = [type_{i_2}]. \tag{5.5}$$

   The set of element instance pairs generated by $\varphi =$ ' $\sim$ ' is denoted by CTE.

4. Equivalence:

$$ei_1 = ei_2 \text{ if } type_{i_1} = type_{i_2}. \tag{5.6}$$

   The set of element instance pairs generated by $\varphi =$ ' $=$ ' is denoted by EQV.

5. Generalization:

   Statement 9. *ECG element instance types can be merged in a lattice* $(T', \prec)$*, whose partial ordering relation* $\prec$ *can be interpreted as a categorical generalization relation.*

   The top and the bottom elements of the $(T', \prec)$ lattice are UNIV (the suprenum element) and NIL (the infinum element), respectively. In terms of [Kovács, 2006], this lattice is the attribute lattice on the basis of which we say that

$$ei_1 \prec ei_2 \text{ if } type_{i_1} \prec type_{i_2}. \tag{5.7}$$

   If $ei_1 \prec ei_2$ holds, we say that element instance ei_2 can be 'restricted to' element instance ei_1, or in other words ei_1 is a 'restriction of' ei_2. This relation is asymmetric, therefore $ei_2 \prec ei_1$ needs to be also checked. The set of element instance pairs generated by $\varphi =$ ' $\prec$ ' is denoted by GEN.

Accordingly, all element instance pairs in a problem domain U fall into one of the disjunct sets generated by the above defined φ correlations. The relationships between the sets are:

$$\begin{aligned} U = \quad & SC \cup SNC, \qquad SC \cap SNC = \emptyset \\ SC \supset \quad & CTE \cup GEN, \qquad CTE \cap GEN = \emptyset, \quad EQV \subset CTE. \end{aligned} \tag{5.8}$$

Based on the element instance type lattice, it is possible to exhibit the least common generalization *lcg*, and the greatest common specialization *gcs* of two element instances ei_1 and ei_2:

$$lcg(ei_1, ei_2) = \min\{type_i \mid type_{i_1} \prec= type_i \text{ and } type_{i_2} \prec= type_i\}; \tag{5.9}$$

$$gcs(ei_1, ei_2) = \max\{type_i \mid type_i \prec= type_{i_1} \text{ and } type_i \prec= type_{i_2}\}. \tag{5.10}$$

5.2.3 Matching ECG diagram graphs

By definition, the matching operation determines an alignment for a pair of ontologies, where alignment is the task of creating links (i.e. a set of mapping elements) between two consistent ontologies. In the case of ECG diagram graph matching the aim is to find the ϕ relation between two given graphs. The constituents of the graphs are given in the form of *type* : *caption*, where the type attribute uniquely identifies the element instances. Therefore ECG diagram graph matching can be traced back to the matching of element instance types.

Definition 7. *An ECG diagram graph can be defined as* $\Gamma = \langle V, A, R \rangle$. *$V$ is the set of vertices containing e_i element instances where* $[type_i] \in T_{cc} \cup T_{pc}$. *$A$ is the set of arrows (directed edges) containing e_i element instances where* $[type_i] \in T_{rr} \cup T_{sr}$. *$R$ is the set of semantic roles with which the arrows in A are labeled. Thus, the f incidence function assigns an ordered pair of vertices in V and a semantic role in R to each arrow in A, that is* $f(a_i) = (v_i, v_j, r_k)$.

The ECG diagram graph matching algorithm takes two ECG diagram graphs Γ_1 and Γ_2 as input. The output of the ECG diagram graph matching algorithm is a set of mapping elements denoted by M and a fitness value μ. A mapping element $m \in M$ is defined as a triplet $\langle ei_1, ei_2, \varphi \rangle$, where

- $ei_1 \in \Gamma_1$ and $ei_2 \in \Gamma_2$ are the aligned element instances of the two ontologies, and
- φ is the correlation between ei_1 and ei_2.

The sequence in which the correlations are examined between two element instances – on the basis of the element instance type lattice – should follow the steps below because for a given element instance pair there may be more than one candidate correlations but only one is chosen for the alignment. In this way, the resulting assignment is univalent.

1. Check $ei_1 >< ei_2$. If true then goto 2. If false then $\varphi =$ ' <> '.
2. Check $ei_1 \sim ei_2$. If true then goto 3. If false then goto 4.
3. Check $ei_1 = ei_2$. If true then $\varphi =$ ' = '. If false then $\varphi =$ ' $\sim$ '.
4. Check $ei_1 \prec ei_2$. If true then $\varphi =$ ' $\prec$ '. If false then goto 5.
5. Check $ei_1 \succ ei_2$. If true then $\varphi =$ ' $\succ$ '. If false then $\varphi =$ ' >< '.

The order of precedence among the φ relations is: $=, (\prec, \succ), \sim, ><, <>$, where $\prec$ and $\succ$ are at the same precedence level. For the alignment, the first three relations would be of

interest. However, in order to obtain an unambigous (bijective) mapping only the mapping elements where $\varphi \in \{=\}$ are included in the alignment. Therefore the method belongs to the class of exact matching algorithms.

For describing the result of the matching, an alignment measure is introduced. Let l denote the number of mapping elements in an alignment M. For a given pair of aligned ECG diagram graphs Γ_1 and Γ_2, the fitness value μ is calculated as:

$$\mu(\Gamma_1, \Gamma_2) = \frac{l}{\frac{j+k}{2}}, \tag{5.11}$$

where j is the number of element instances in graph Γ_1, while k is the number of element instances in graph Γ_2. In this way, the fitness value falls into the $[0, 1]$ interval.

Given two ECG diagram graphs $\Gamma_1 = \langle V_1, A_1, R_1 \rangle$ and $\Gamma_2 = \langle V_2, A_2, R_2 \rangle$, their $\phi \in \Phi$ relation needs to be determined. According to the value of μ, they can be

- semantically comparable, indicated by $0 < \mu(\Gamma_1, \Gamma_2) < 1$:

$$\Gamma_1 \bowtie \Gamma_2 \text{ if } V_1 \cap V_2 \neq \emptyset; \tag{5.12}$$

- semantically incomparable (or disjunct), indicated by $\mu(\Gamma_1, \Gamma_2) = 0$:

$$\Gamma_1 \lhd\rhd \Gamma_2 \text{ if } V_1 \cap V_2 = \emptyset; \tag{5.13}$$

- equivalent, indicated by $\mu(\Gamma_1, \Gamma_2) = 1$:

$$\Gamma_1 \equiv \Gamma_2 \text{ if } V_1 = V_2 \text{ and } A_1 = A_2. \tag{5.14}$$

5.2.4 Evaluation of the matching algorithm

A well-known and widely-accepted classification of matching approaches is provided by [Shvaiko & Euzenat, 2004]. The authors specify three dimensions along which matching algorithms can be classified. According to this, the main characteristics of the matching procedure developed are as follows.

1. *Input dimensions*
   - The ECG diagram graph matching algorithm works on instance-level ECG graphs, which are instantiations of the ECG conceptual model (shared ontology) in which the ontologies are expressed.
   - The matching algorithm relies on four information sources: 1) the type attribute of graph constituents (element-level information), 2) the element category type lattice (schema-level external source), 3) the element instance type lattice (schema-level external source), and 4) the relations between the graph elements (structure-level information).

2. *Process dimensions*

   – The computation of mapping elements is exact.
   – The input is interpreted on the basis of the ECG semantic model.

3. *Output dimensions*

   – The form of the resulting alignment is a one-to-one correspondence between the element instances.
   – The method delivers a graded answer, i.e. an alignment is given with a fitness value in the $[0, 1]$ range.
   – The algorithm examines four $\phi \in \Phi$ relations between two ECG graphs: equivalence ($\equiv$), semantic comparability ($\bowtie$), disjunction ($\lhd\rhd$), and restriction ($\preceq$).

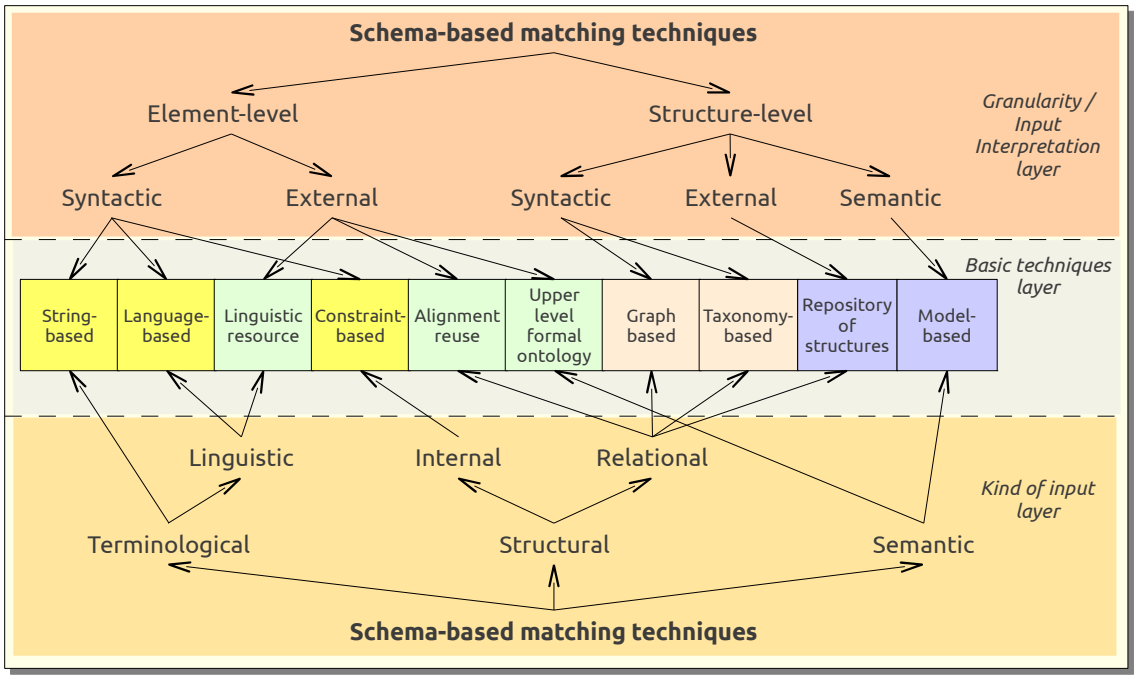

Figure 5.3 *Classification of matching algorithms [Shvaiko & Euzenat, 2004]*

The classification of matching algorithms in [Shvaiko & Euzenat, 2004] differentiates between elementary matchers, which include element-level and structure-level matchers, and combining matchers which combine several approaches (see Figure 5.3). Combination can be done in two ways: a hybrid matcher integrates multiple matching criteria, while a composite matcher combines the results of independently executed matchers.

The matching algorithm developed can be considered to be a hybrid matcher that applies the following techniques:

– element-level syntactic string-based matching for the determination of equivalence of element instances,

– element-level external matching using the element category type lattice for the determination of the $\varphi = \{><, <>, \sim\}$ relations between element instances,

– element-level external matching using the element instance type lattice for the determination of the generalization relation between element instances,

– structure-level syntactic graph-based matching in examining the $\preceq$ restriction relation between two ECG diagram graphs.

The matching method developed could be further refined by using an element-level external repository of existing mapping elements (alignment reuse), and a structure-level external repository of previously aligned ontologies along with a similarity coefficient.

5.3 The association operation

It is possible to insert ECG diagram graphs incrementally into an initially empty knowledge base, which is itself another (accumulated) ECG diagram graph and which is built up by the successive insertion of primary-level ECG diagram graphs. This corresponds to association in the process of conceptualization, that is incorporating new information items into the existing knowledge base. In this operation, the ECG diagram graph to be inserted (Γ_2) must be matched to the knowledge base (Γ_1) according to the following algorithm (see Algorithm 5.1).

1. The M mapping (alignment) of the two ECG diagram graphs must be performed resulting in μ.
2. If $\mu(\Gamma_1, \Gamma_2) = 0$ then $\Gamma_1 \cap \Gamma_2 = \emptyset$. In this case Γ_2 is inserted into the knowledge base in a disjunctive way.
3. If $\mu(\Gamma_1, \Gamma_2) = 1$ then $\Gamma_1 \equiv \Gamma_2$, that is $V_1 = V_2$ and $A_1 = A_2$. In this case Γ_2 does not need to be inserted into the knowledge base.
4. If $0 < \mu(\Gamma_1, \Gamma_2) < 1$ then $\Gamma_1 \cap \Gamma_2 \neq \emptyset$. In this case $\forall v_{2i}, a_{2i} \in \Gamma_2 \mid v_{2i}, a_{2i} \notin \Gamma_1$ are inserted into the knowledge base in a conjunctive way, if Γ_2 is not a subgraph of Γ_1.

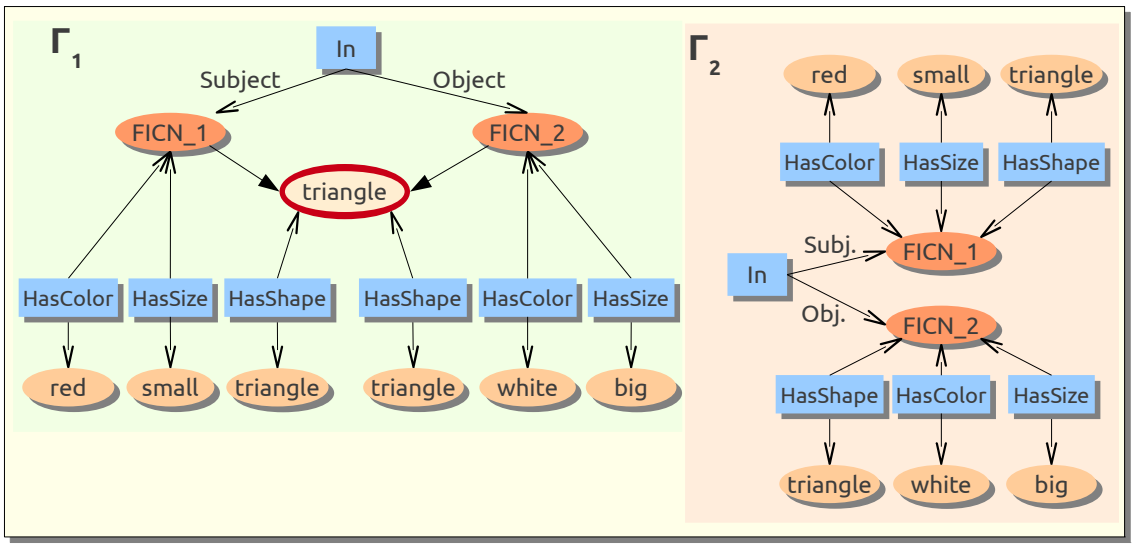

Figure 5.4 *Isomorphic ECG diagram graphs*

An ECG diagram graph Γ_2 is a subgraph of Γ_1 if

- Γ_1 contains a subgraph Γ_1' which is identical to Γ_2, i.e. $\Gamma_1' \equiv \Gamma_2$; or
- Γ_1 contains a subgraph Γ_1' which is isomorphic to Γ_2, i.e. $\Gamma_1' \simeq \Gamma_2$, where 'isomorphic' means that graph Γ_2 can be obtained from Γ_1' by restricting some of the element instances of Γ_1' based on the $(T', \prec)$ element instance type lattice (see the illustration in Figure 5.4 where $\Gamma_1 \equiv \Gamma_1'$).

```
Input: Γ1, Γ2
Output: Γ1

(M,μ) = Match(Γ2, Γ1)
if μ == 1 then
    return
if μ == 0 then
    Insert Γ2 into Γ1
if 0 < μ < 1 then
    if Γ2 ⪯ Γ1 then
        return
    else
        foreach i ∈ Γ2 do
            foreach m ∈ M do
                if i ∉ m then
                    Insert i into Γ1
```

Algorithm 5.1 *The association algorithm*

The association algorithm is based on ECG diagram graph matching. Therefore computing the cost of the algorithm includes the cost of matching, plus the cost of inserting graph elements. In matching ECG diagram graphs Γ_2 is matched to Γ_1. Let j denote the number of element instances in Γ_1, and k denote the number of element instances in Γ_2. Matching is accomplished by comparing each element instance in Γ_2 to each element instance of Γ_1, the cost of which is $k \cdot j$.
If $\mu == 0$, the cost of insertion is k. If $0 < \mu < 1$, the two graphs have common element instances. The cost of checking relation $\Gamma_2 \preceq \Gamma_1$ is $k \cdot l \leq k^2$ where l is the number of mapping elements in M. If Γ_2 is a subgraph of Γ_1, that is $\Gamma_2 \preceq \Gamma_1$ holds, thus all element instances of Γ_2 are in alignment M, no insertion is executed. If Γ_2 is not a subgraph of Γ_1, the cost of inserting the element instances of Γ_2 that are not included in Γ_1 is less than k. Summing up the costs of the instructions, the association algorithm has an approximated cost of $k \cdot j + k^2 + 2 \cdot k$.

5.4 The generalization operation

The association operation does not involve generalization. It covers only the accumulation of incoming information. However, by the increase of the amount of incoming data the knowledge base would be subtle and computationally intractable without the use of generalization.

The present investigation focuses on concept generalization within ECG diagram graphs, which is defined as the process by which new higher-level ECG concepts are created in the knowledge base of ECG diagram graphs incorporating the common characteristics of existing concepts. This process reduces the complexity of the knowledge base of ECG diagram graphs, since the new higher levels include an ever decreasing number of concepts and relationships than the underlying layers.

The higher-level concepts to be introduced are pre-defined in a domain-specific concept lattice, which is the element instance type lattice. Its generation from the given domain is called *abstraction*. This concept lattice is built in batch mode using the existing concepts of the samples, and also the abstract category concepts which are manually added to the lattice.

The generalization algorithm gives as result the least common generalized graph that can be obtained from two ECG graphs. The algorithm involves the following steps.

1. The first task is to find frequent knowledge patterns the context of which is similar, that is ECG diagram subgraphs which differ in only one semantically comparable concept node.
2. Instead of the differing concepts a new concept is introduced. Otherwise, the operation cannot be performed. In this way, the similar subgraphs can be united under the new concept and the differing concepts are connected to the new concept via specialization relationships.
3. The relationships of the generalized graph should be updated.

Definition 8. *Two ECG diagram subgraphs $\gamma_1 \in \Gamma_1$ and $\gamma_2 \in \Gamma_2$ are said to be similar subgraphs if $\gamma_1 \equiv \gamma_2$ or $\gamma_1 \simeq \gamma_2$, and they are connected to differing but semantically comparable ECG concept nodes ($ei_1 >< ei_2$).*
Two similar subgraphs are considered as maximal similar subgraphs if they cannot be extended further without violating the criterion of similarity.

Thus, the maximal similar subgraphs are searched for in Γ_1 and Γ_2. For this, the operation of ECG graph intersection must be introduced. The intersection of two ECG graphs Γ_1 and Γ_2 is the set of identical or isomorphic connected subgraphs. Formally,

$$\Gamma_1 \cap \Gamma_2 = \{\gamma_1, \gamma_2, \ldots, \gamma_k\} \text{ where } \quad \forall \gamma_i \ : \ \gamma_i \in \Gamma_1 \wedge \gamma_i \in \Gamma_2 \text{ or } \quad \gamma_i \in \Gamma_1 \wedge \gamma_i' \in \Gamma_2 \text{ where } \gamma_i \simeq \gamma_i'. \tag{5.15}$$

Note here, that on the set of all ECG diagram graphs in a given domain, the operation of graph intersection can be recursively performed, thereby determining the set of all common subgraphs. In this way a lattice can be built, at the lower levels of which are individual

(infrequent specialized) ECG diagram graphs, while at the top levels of which are frequent general subgraphs.

Statement 10. *On the union of the set of primary-level ECG diagram graphs (**Γ**) and the set of subgraphs resulting from intersection (**γ**) the ⊆ set relation effects a lattice structure.*

Input: Γ_1, Γ_2
Output: Γ_1

(M,μ) = Match(Γ_2, Γ_1)
if μ == *1* **then**
    return
if μ == *0* **then**
    Insert Γ_2 into Γ_1
if *0* < μ < *1* **then**
    if $\Gamma_2 \preceq \Gamma_1$ **then**
        return
    else
        `// Begin generalization`
        Search for maximal similar subgraphs in Γ_1, Γ_2
        foreach (γ_1^*, γ_2^*) **do**
            if $ei_1 >< ei_2$ **then**
                if $lcg(ei_1, ei_2) \neq UNIV$ **then**
                    if $lcg(ei_1, ei_2) \notin \Gamma_1$ **then**
                        Insert $lcg(ei_1, ei_2)$ into Γ_1
                    if $lcg(ei_1, ei_2) \neq ei_1$ **then**
                        Connect ei_1 to $lcg(ei_1, ei_2)$ by FMI
                    Update relations of ei_1 in Γ_1
                    if $ei_2 \notin \Gamma_1$ **then**
                        Insert ei_2 into Γ_1
                    if $lcg(ei_1, ei_2) \neq ei_2$ **then**
                        Connect ei_2 to $lcg(ei_1, ei_2)$ by FMI
                    Update relations of ei_2 in Γ_1
        `// End generalization`
        foreach $i \in \Gamma_2$ **do**
            foreach $m \in M$ **do**
                if $i \notin m$ **then**
                    Insert i into Γ_1

Algorithm 5.2 *Association with generalization*

The extension of the ECG graph intersection operation results in the pairs of maximal similar subgraphs of Γ_1 and Γ_2 (see Definition 8). Formally,

$$\Gamma_1 \cap^* \Gamma_2 = \{(\gamma_{11}^*, \gamma_{12}^*), (\gamma_{21}^*, \gamma_{22}^*), \ldots, (\gamma_{k1}^*, \gamma_{k2}^*)\} \text{ where} \quad (5.16)$$
$$\forall(\gamma_{i1}^*, \gamma_{i2}^*) \ : \ \gamma_{i1}^* \cup \gamma_{i2}^* = \gamma_i \cup \{ei_1, ei_2\} \mid$$
$$\gamma_{i1}^*, ei_1 \in \Gamma_1 \ , \gamma_{i2}^*, ei_2 \in \Gamma_2 \text{ and } \gamma_i \in \Gamma_1 \cap \Gamma_2.$$

For the differing concepts $ei_1 >< ei_2$ should be checked, where $ei_1 \in \Gamma_1$ and $ei_2 \in \Gamma_2$. If semantic comparability holds between the instances – that is $lcg([type_{i_1}], [type_{i_2}]) \neq UNIV$

on the basis of the element category type lattice – a new concept needs to be introduced from the element instance type lattice determined as $lcg(ei_1, ei_2)$ (if this is not the $UNIV$ top element). It is possible, that $lcg(ei_1, ei_2)$ results in one of its arguments. In this case actually no insertion occurs. Finally, the differing concepts are connected to the new concept via specialization relationships and the other relationships originally in connection with the differing concepts should also be updated.

In practice, the problem of finding similar subgraphs can be solved by parallel depth-first searches in Γ_1 and Γ_2. Optimally, the generalization operation is implemented as part of the process of association. More precisely, it should be accomplished within the instruction block where $0 < \mu < 1$ and $\Gamma_2 \preceq \Gamma_1$ is false, that is the two graphs have common element instances but Γ_2 is not a subgraph of Γ_1. The modified algorithm can be found in Algorithm 5.2.

Consequently, the cost of the conceptualization process results from the sum of the cost of the association algorithm ($k \cdot j + k^2 + 2 \cdot k$) and that of the generalization algorithm which involves similar subgraph search and the search for the least common generalized element in the element instance type lattice. Let k denote the number of element instances in Γ_2, and j denote the number of element instances in Γ_1; where Γ_2 is a primary-level ECG diagram graph, while Γ_1 is the accumulated ECG diagram graph. Thus, in the long run $j >> k$, and m denotes the size of the element instance type lattice. In the case of sequential search the cost of similar subgraph search is $k \cdot j$ and the cost of determining the least common generalized element of the two differing elements is $2 \cdot m$. When using an indexed tree for the searches the costs can be reduced to $k \cdot \log j$ and $2 \cdot \log m$, respectively. However, in this case the cost of generating the indexed tree should also be taken into account.
Considering a smaller domain and sequential searches, the cost of the conceptualization process adds up to $2 \cdot (k \cdot j) + k^2 + 2 \cdot k + 2 \cdot m$. Initially, when the size of the accumulated ECG diagram graph is null the first element of the summation can be neglected and the cost of the method is mostly characterized by the size of the element instance type lattice. Later on, as the size of Γ_1 increases, the effect of the first element will be the most significant on the cost of the whole process of conceptualization.

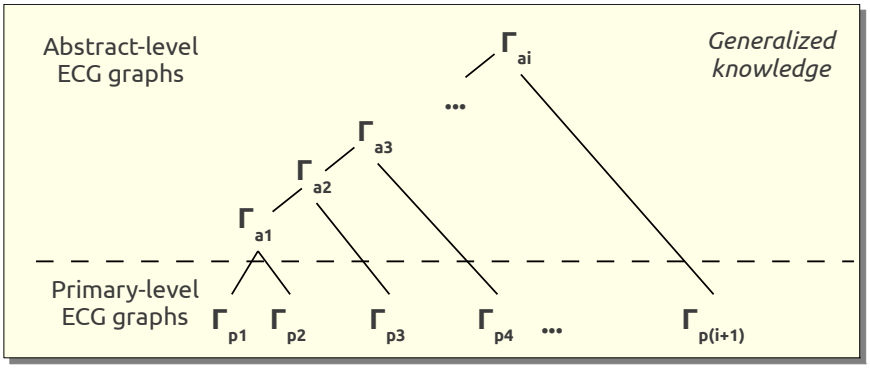

Figure 5.5 *The process of generalization*

At the end of the generalization process (see Figure 5.5) stands the generalized knowledge an agent can obtain from the samples observed. This can be formulated as recursively determining the least common generalization of the previous abstract-level ECG graph and a primary-level ECG graph, that is

$$\Gamma_{a_i} = lcg(\Gamma_{a_{i-1}}, \Gamma_{p_{i+1}}). \tag{5.17}$$

Similarly to the formulation of the least common generalization of two ECG graphs, it is possible to give the greatest common specialization of two ECG graphs. This latter can be defined as the maximal common restriction of the two graphs, that is the union of the maximal similar subgraphs of the two graphs. Formally,

$$gcs(\Gamma_1, \Gamma_2) = \bigcup(\{\max(\gamma^*_{i1}, \gamma^*_{i2})\}). \tag{5.18}$$

Though it is possible that the greatest common specialization of two ECG graphs results in an empty graph, the least common generalization and the greatest common specialization of two ECG graphs always exist and can be computed. Therefore, the definition of the $\prec$ relation on element instances can be extended to a partial relation $\preceq$ on ECG diagram graphs and the term 'restriction of' can be used to describe this relation. Accordingly, an ECG diagram graph Γ_2 is a restriction of ECG diagram graph Γ_1, that is $\Gamma_2 \preceq \Gamma_1$ if graph Γ_2 is more 'specialized' than graph Γ_1.

Let $\mathbf{\Gamma}$ denote the set of primary-level ECG diagram graphs (representing environment snapshots) that are matched to and incorporated in the knowledge base of the agent, and $\mathbf{\Gamma}(\mathbf{A})$ denote the set of accumulated ECG diagram graphs resulting from the conceptualization (association and generalization) steps executed.

Statement 11. *The $\preceq$ relation effects a lattice structure on the union of $\mathbf{\Gamma}$ and $\mathbf{\Gamma}(\mathbf{A})$.*

The top element of the lattice $(\mathbf{\Gamma} \cup \mathbf{\Gamma}(\mathbf{A}), \preceq)$ symbolizes the accumulated knowledge of the agent at the end of the conceptualization process. The bottom element is NIL, the infinum element.

5.5 Summary of the results

This chapter has discussed the process of conceptualization by means of ECG diagram graphs, that is how the knowledge base of the grammar inducing agent investigated is built up from primary-level ECG graphs. This process involves not only knowledge accumulation and incorporation (association), but also abstraction and generalization, where the operation of association can always be accomplished, but generalization does not necessarily occur in each step. The new scientific results can be summarized as follows.

Thesis 4.

A method is developed for the execution of the conceptualization process within the learning agent examined, which involves the operations of association and generalization. According to the association algorithm, primary-level ECG diagram graphs are matched to and incorporated in an initially empty knowledge base, which is itself another (accumulated) ECG diagram graph. The matching of ECG diagram graphs is based on a hybrid context-dependent ECG diagram graph matching algorithm, and is traced back to the matching of element instances, for the examination of which an element category type lattice is defined. The generalization algorithm is implemented as part of the association process and proceeds by introducing new (not observed) higher-level concepts into the knowledge base. First, the algorithm searches for maximal similar subgraphs which differ in only one ECG diagram graph node. For their exploration the intersection operation of two ECG diagram graphs and its extension are defined. If the differing nodes are semantically comparable on the basis of the element category type lattice, a new concept is inserted from the element instance type lattice determined as the least common generalization of the differing concepts. Finally, the relationships are updated in the knowledge base.

$< \textit{Statements}\ 8, 9 >$

Consequences

1. Recursively performing the operation of graph intersection on the set of ECG diagram graphs and on the resulting sets of common subgraphs, a lattice can be built. The lower-level nodes of the lattice include individual (infrequent specialized) ECG diagram graphs, while at the top levels of the lattice frequent general subgraphs are located. The relation between the nodes of this lattice is $\subseteq$.

   $< \textit{Statement}\ 10 >$

2. The two operations of association and generalization together accomplish the process of conceptualization. At the end of the process, the accumulated knowledge of the agent can be obtained as the top element of the lattice constructed from the set of primary-level ECG diagram graphs and the set of accumulated ECG diagram graphs resulting from the conceptualization (association and generalization) steps executed, the relation between the nodes of which is $\preceq$.

   $< \textit{Statement}\ 11 >$

Chapter 6

Applications of the Theoretical Results

The fourth task of the dissertation is to build a test system within which the applicability of the theoretical results can be verified. Accordingly, this chapter introduces the semantic annotation framework [12] by the application of which the set of training samples (symbolic assertions semantically annotated with ECG ontologies) has been generated automatically for modeling the process of conceptualization.

6.1 Semantic annotation framework

The process of building a knowledge-based system is composed of three stages. First, the domain ontology (terminological database) should be created which then should be extended with the assertions or statements on the instances of the domain. Finally, to the knowledge given in this way a problem solving method should be assigned that defines the control structure (operational model) of the system [Futó, 1999].

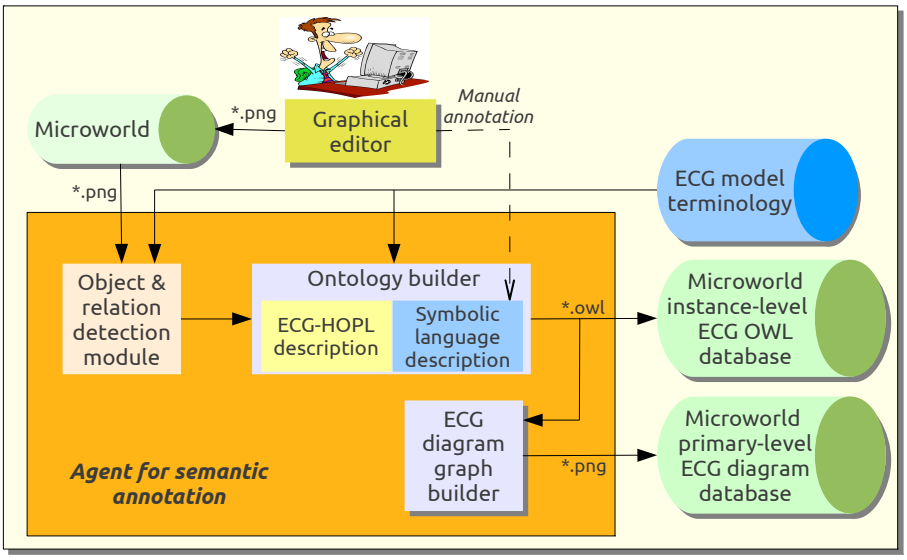

Figure 6.1 *Operational model of the the semantic annotation framework*

For the present investigations, the terminological database has been created in OWL DL (*ECGmodel.owl* on the DVD attached) on the basis of the ECG model definition using Protégé, which is a free, open source ontology editor and knowledge-base framework

developed at Stanford University (*http://protege.stanford.edu/*). For generating the assertions on domain instances, a semantic annotation framework has been implemented in NetBeans IDE 6.9 integrated development environment using Java 1.6.0_20 version Java HotSpot(TM) 64-Bit Server VM 16.3-b01 [12]. The operational model of the system is shown in Figure 6.1.

The system represents an agent for semantic annotation. Its environment is the graphical microworld created by a graphical editor. The environment snapshots (static observations) are processed by the sensor of the agent, which is the object and relation detection modul. Its task is to recognize the objects of the environment and their relations, and to map them to the internal semantic representation of the agent using the terminological database (ECG model definition). Next, the ontology builder automatically generates the instance-level ontologies (assertions on the environment instances describing the semantics of the observations) in OWL DL textual format. The assertions are also expressed in ECG-HOPL statements and in symbolic sentences. Finally, the ECG diagram graph builder creates the graphical diagrams for the OWL descriptions.

6.1.1 Graphical editor

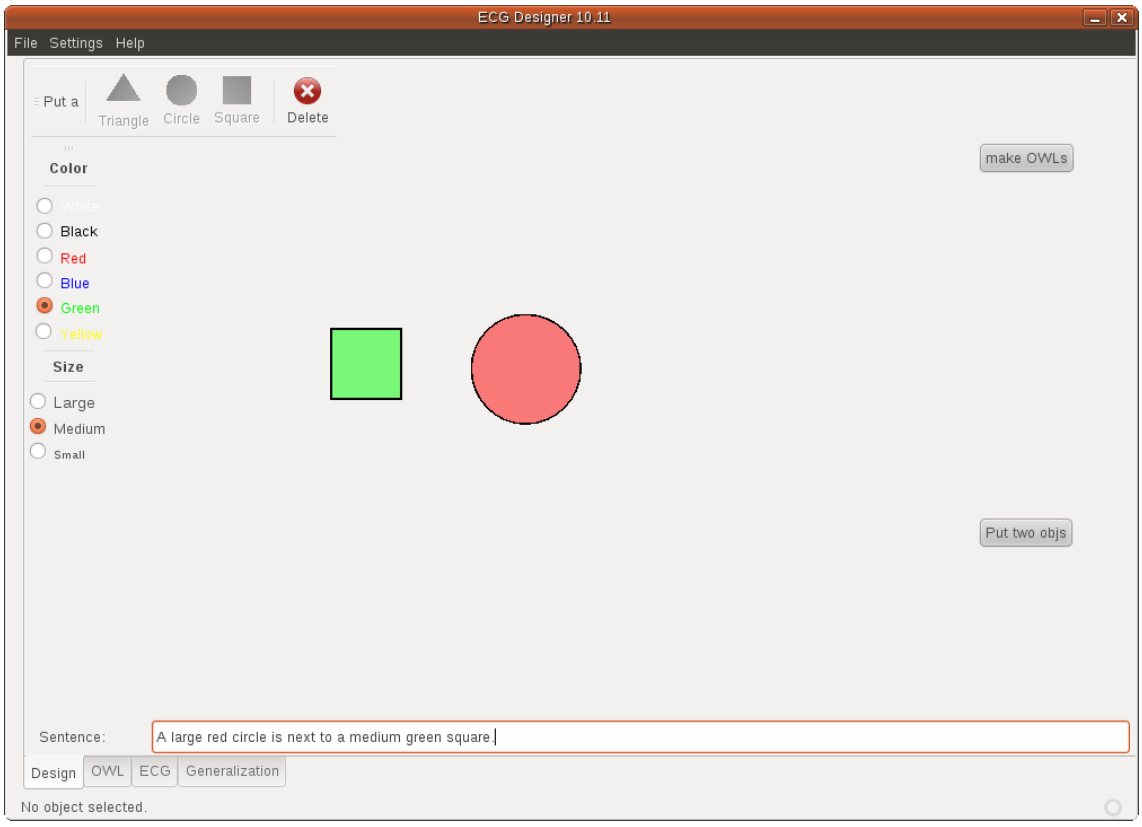

Figure 6.2 *User interface of the graphical editor modul*

The environment of the agent is the graphical microworld created by the graphical editor. The objects of the environment include two-dimensional instances which can be characterized by their shape (circle, square and triangle), their color (white, black, red, blue,

green, yellow) and their size (large, medium, small). The relationship of two objects can be of geometrical or proportional type. The geometrical relationship of two objects can be inclusion, intersection or non-intersection. The non-intersection relationship can be further divided into four (semantically three) cases: above, below and next to. Since the relations of the two objects are determined relatively to one another, the inverse of the relationships is also defined which can be given by changing the roles of the objects within the relation. Thus, all together six semantically different geometrical relationships can be identified. The graphical user interface of the graphical editor modul (`Design` tab in the semantic annotation framework) is displayed in Figure 6.2.

Environment snapshots created manually are stored in a `tmp` directory. However, by clicking on the `Put two objs` button the graphical editor automatically generates all possible, semantically valid and unique snapshots in `png` format. The combinatorically estimated upper value for the number of snapshots in the given microworld is 11664. However, when two objects with the same shape and size are in inclusion relationship they are actually in coverage, so these cases (648) are semantically uninterpretable and they should be subtracted from the estimated value resulting in 11016, which gives the number of valid cases. Graphical environment snapshots, which are also stored in `txt` format for the sake of reloadability, are stored in an `OWL` directory groupped by the shape, size and color of the objects observed.

6.1.2 Object and relation detection modul

Environment snapshots are processed by the object and relation detection modul, which represents the sensor of the agent. Its task is to recognize the objects of the environment together with their properties and their relationships, and to map the observations to the internal semantic representation of the agent using the ECG model definition (see Table 6.1). The capabilities of the agent are predefined. Accordingly, the set of recognizable environment elements is the following.

Table 6.1 *Mapping rules of recognizable environment elements*

Element recognized	**Internal identifier**	**Constraints**
assertion on two objects	FMPR	Within a fragment only one kernel predicate can be defined.
object	FICN	Objects are linked to the kernel predicate via FSR relationships in Subject or Object role.
shape	FMCR	Shapes are connected to a FICN object via the *HasShape* predicate in FMR Object role, while the FICN object plays the Subject role.
color	FMCR	Colors are connected to a FICN object via the *HasColor* predicate in FMR Object role, while the FICN object plays the Subject role.
size	FMCR	Sizes are connected to a FICN object via the *HasSize* predicate in FMR Object role, while the FICN object plays the Subject role.

- Shapes recognized: circle, square, triangle.
- Colors recognized: white, black, red, blue, green, yellow.
- Sizes recognized: large, medium, small.
- Relationships between two objects recognized:
  - inclusion and its inverse relation;
  - intersection;
  - non-intersection: above/below, next to;
  - bigger/smaller.

The identification of the geometrical relationships is based on the position of the geometrical centers of the objects. The algorithm proceeds by first examining whether the object displayed first is *in* the object displayed second. If this is the case, the *inverse inclusion* relation also holds by changing the roles of the objects. If the relation between the two objects is not inclusion, the possibility of *intersection* should be examined. If the result is negative, a non-intersection relation must hold. For deciding the type of a non-intersection relation a coordinate system turned by 45° should be placed to the center of the object displayed first. If the center of the object displayed second is located in the upper/lower quarter, then the first object is *below/above* the second. Its inverse is that the second object is *above/below* the first (determined by changing the roles of the objects). If the center of the second object is located in the left or right quarter, the relationship between the objects is *next to*. Finally, the proportional relationship is examined between the objects on the basis of their size. If the first object is *bigger/smaller* than the second, then the second is evidently *smaller/bigger* than the first. The result of this analysis is stored for each snapshot in textual format (see an example in Figure 6.3).

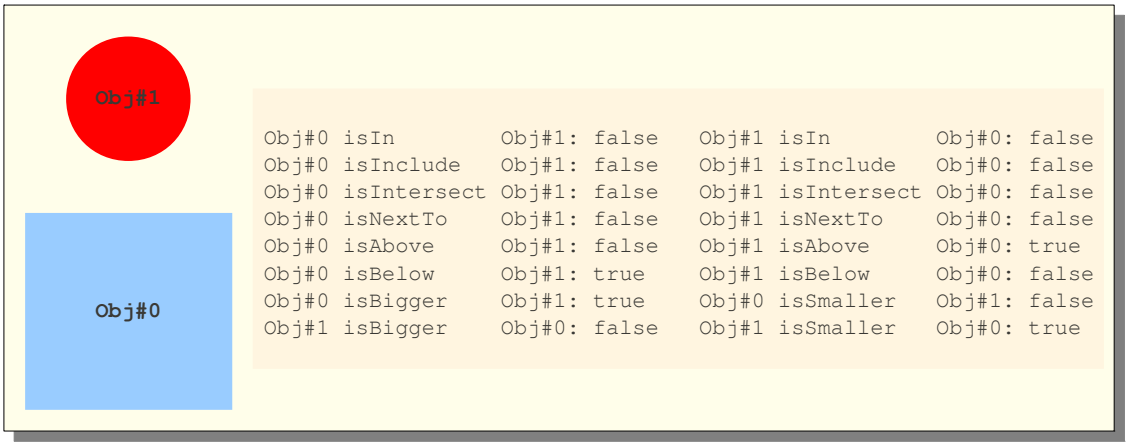

Figure 6.3 *Relation test of the given snapshot*

6.1.3 Ontology builder

The task of this modul is to automatically generate the instance-level ontologies (i.e. the semantic descriptions of the environment snapshots detected) in OWL DL textual format (see Figure 6.4) and to assign ECG-HOPL statements and symbolic sentences to the ontologies (which are also stored in the OWL files).

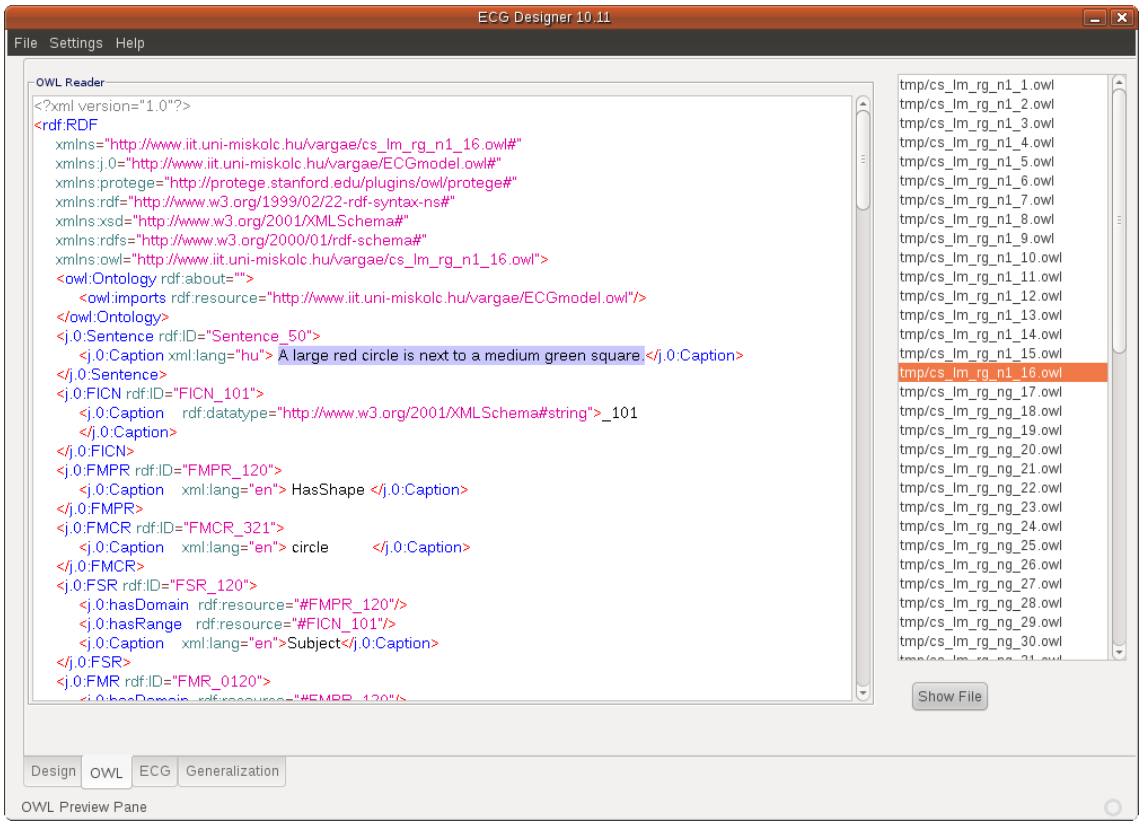

Figure 6.4 *Display of an instance-level OWL ontology*

The `OWL` tab of the semantic annotation framework shows the list of instance-level OWL ontologies generated. By choosing one from the list and clicking on the `Show file` button, it can be displayed in the main panel. For each environment snapshot and for each kernel predicate detected in the snapshot 16 OWL files are generated and stored in the same directory as the graphical snapshot itself. In the simplest case, the OWL ontology includes only the two objects (without their properties) and their relationship. The other files extend this information with the size and/or color of the objects. The program modul creates all possible variations. Among the 11016 graphical environment snapshots 3240 display objects with the same size, in the case of which the proportional relationship can not be interpreted. Therefore 51840 OWL ontologies are generated for these snapshots. For the other 7776 environment snapshots including dissimilar sized objects 248832 OWL ontologies are created. Thus, the number of OWL files comprising the training set adds up to circa 300 thousand.

The steps of constructing instance-level ontologies are summarized in Appendix B. In manual microworld creation mode symbolic sentences can be added manually to the graphical snapshots, the language of which is indifferent. In automatic mode, however, the ontology builder assigns an English sentence to each snapshot automatically. Figure 6.5 shows an example for determining the ECG-HOPL statement of a snapshot.

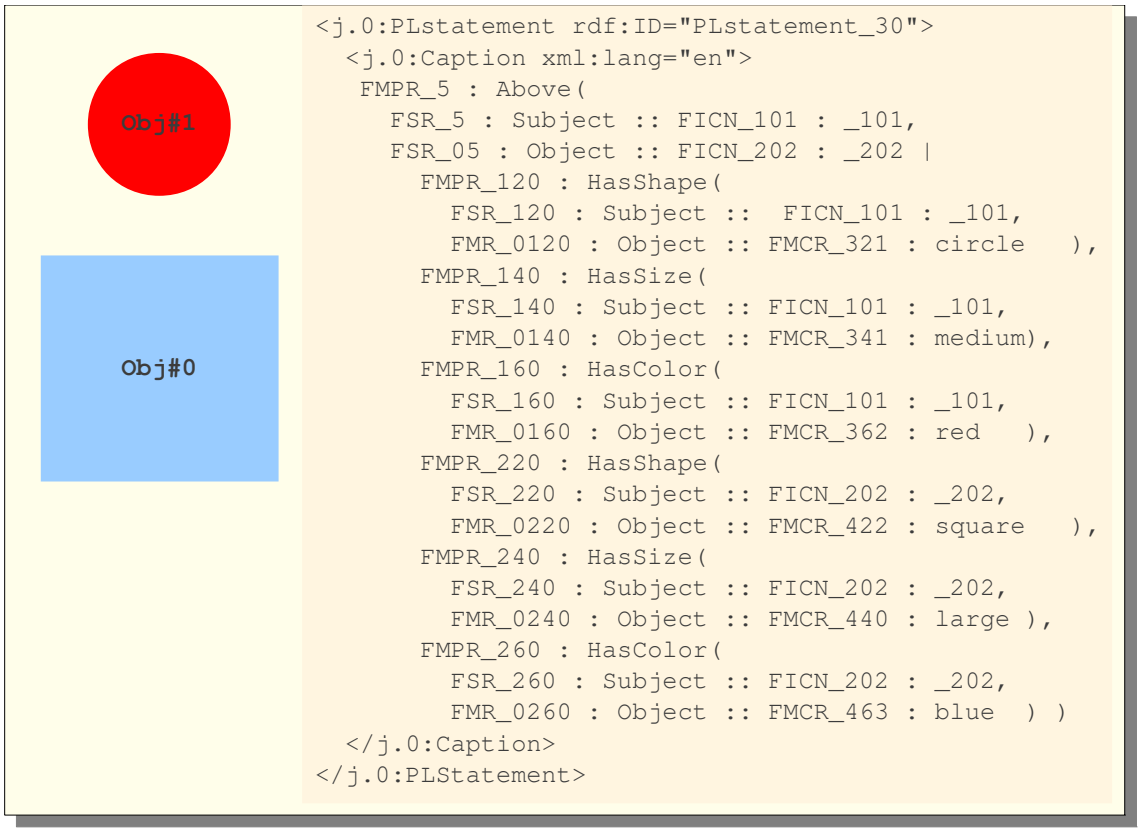

Figure 6.5 *ECG-HOPL statement of the given snapshot*

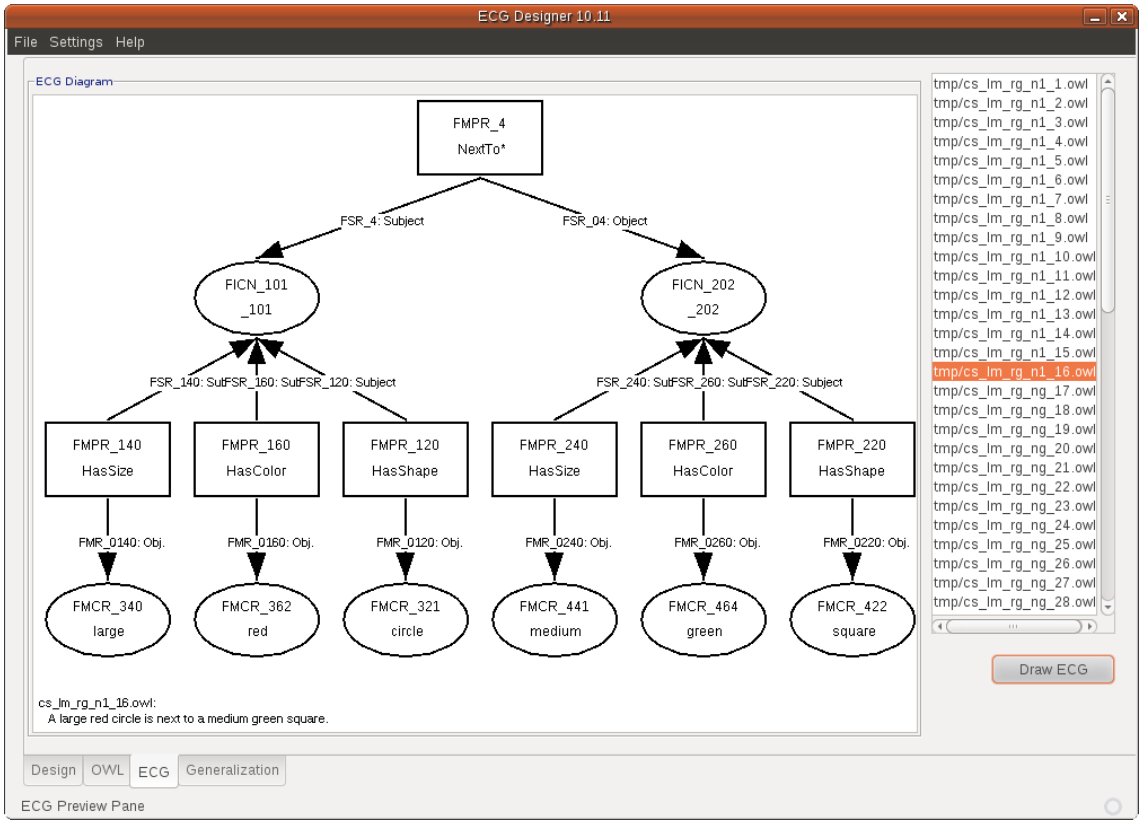

Figure 6.6 *Display of an ECG diagram graph*

6.1.4 ECG diagram graph builder

The task of this modul is the visualization of the textual ontologies. ECG diagram graphs are created automatically using Algorithm 3.1 on page 45 from which one can be chosen to be displayed in the `ECG` tab of the semantic annotation framework by clicking on the `Draw ECG` button (see Figure 6.6).

6.2 Modeling the process of conceptualization

The base set for modeling the process of conceptualization includes approximately 300 thousand ECG diagram graphs generated automatically by the semantic annotation framework. As described in Chapter 5, the process of conceptualization involves the processes of association, abstraction and generalization.

6.2.1 Association

Association is defined as the process by which ECG diagram graphs are incrementally matched to and inserted in the knowledge base of the agent investigated, according to Algorithm 5.1 on page 82. In the `Generalization` tab of the system implemented the ECG diagram graphs can be chosen and inserted into the graph set by clicking on the `Insert ECG` button. The main panel displays the current state of the knowledge base. On this panel, by moving the cursor to any node of the accumulated graph the status bar shows the description of the concept included in that node. The detailed description of the relationships are given in the message box under the graph (see Figure 6.7).

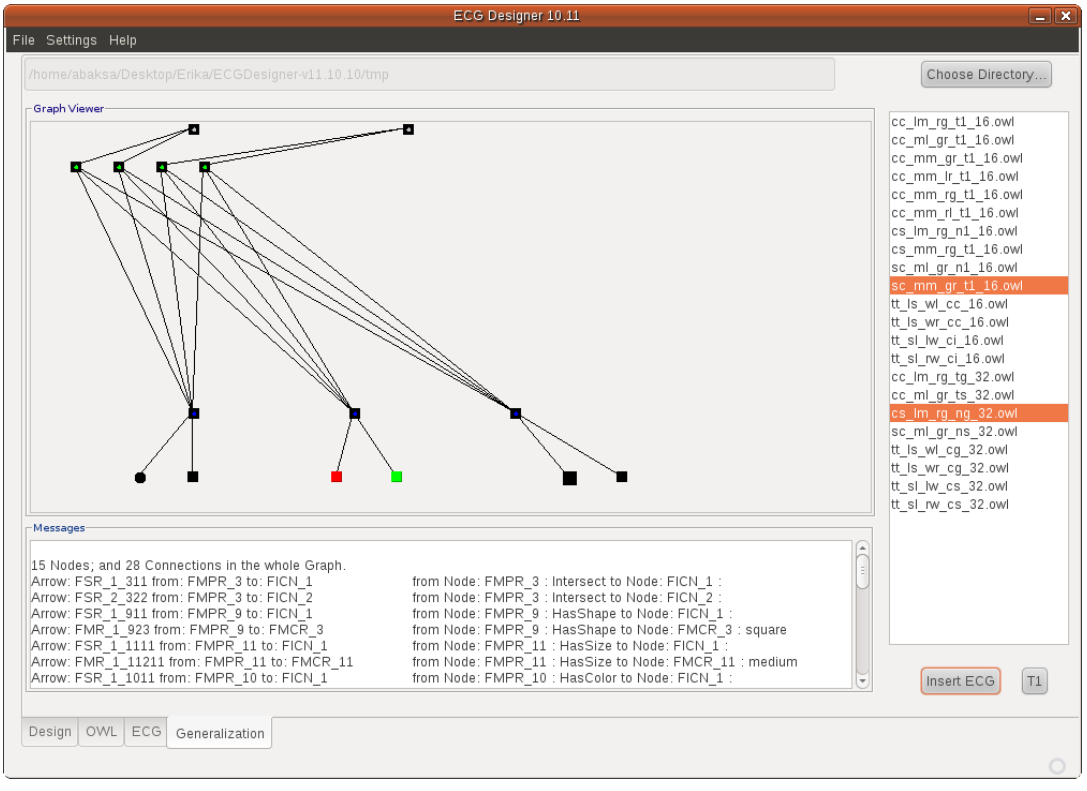

Figure 6.7 *Illustration of association*

6.2.2 Abstraction

Abstraction is the process by which higher-level concepts are generated from a given domain. These concepts can be structured in an element instance type concept lattice, which is built in batch mode and to which abstract category concepts can be added manually. A demonstrative segment of the lattice generated on the basis of the training data set is shown in Figure 6.8.

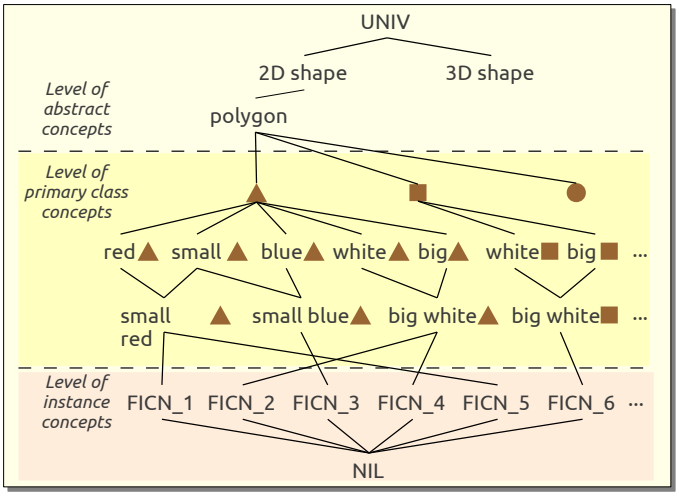

Figure 6.8 *A segment of the element instance type lattice*

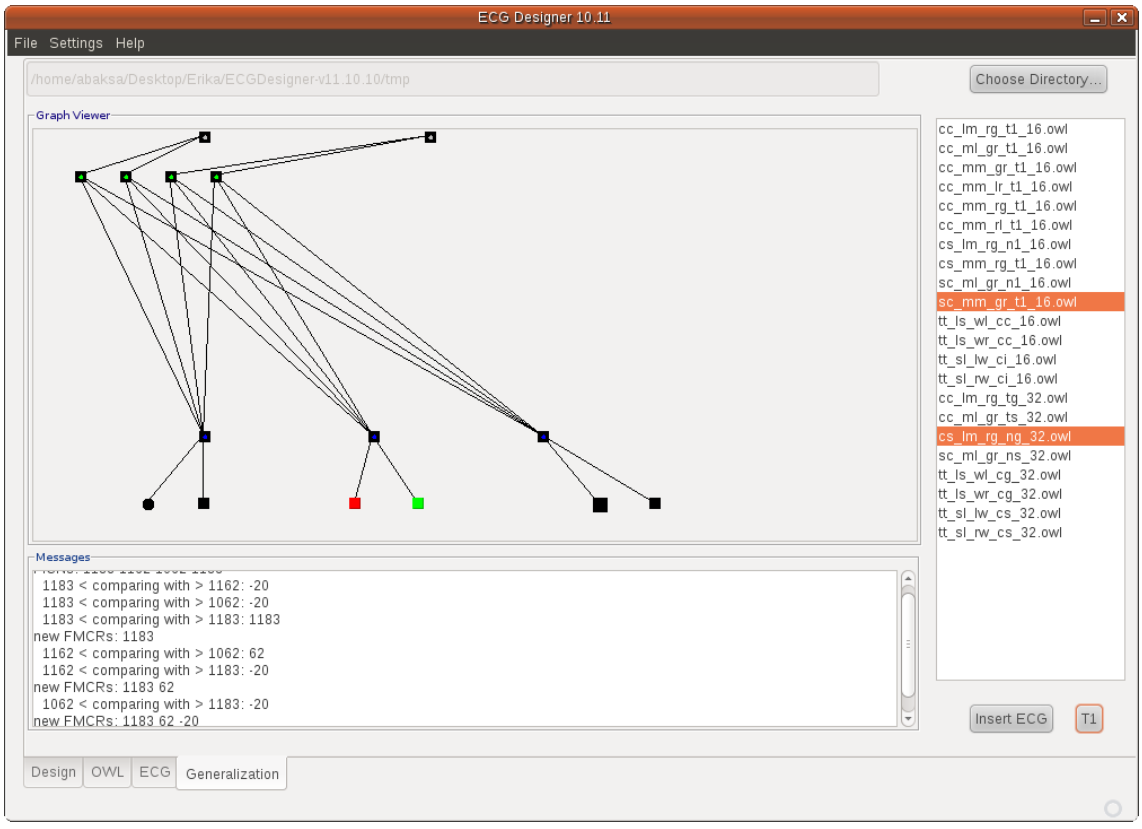

Figure 6.9 *Illustration of generalization*

6.2.3 Generalization

Generalization is the process by which new higher-level ECG concepts are created in the knowledge base of ECG diagram graphs incorporating the common significant (frequently occuring) characteristics of the existing concepts. The higher-level concepts to be introduced are pre-defined in the element instance type lattice.

Generalization is implemented as part of the association algorithm and can be accessed through the `Generalization` tab of the system implemented. The algorithm searches for maximal similar subgraphs in the two graphs to be joined, which differ in only one semantically comparable concept node. Instead of the differing concepts a new concept is introduced determined as the least common generalization of the differing concepts in the lattice (see Figure 6.8), and displayed in the message box under the accumulated graph. In other words, the generalization algorithm gives as result the least common generalized graph that can be obtained from two ECG graphs. See the illustration in Figure 6.9. A demonstrative example for the process of conceptualization (involving association and generalization) can be found in Appendix E.

Chapter 7

Summary

In the dissertation the semantic model which can be applicable in the grammar induction system in Figure 1.1 on page 3 has been investigated. The semantic model of a learning agent has two major roles: 1) it is the internal representation of the signals coming from the agent's environment, and 2) it is used to build up the knowledge base of the agent. Accordingly, the scope of the present research has covered the following tasks.

1. First, the semantic model for knowledge representation in the grammar induction system investigated had to be developed and analyzed (see Chapter 3).
2. Second, the grammar formalism that is able to represent symbolic language sentences and their semantic descriptions in a common framework had to be defined, in support of grammar induction (see Chapter 4).
3. Third, the build-up of the knowledge base of the grammar learning agent examined had to be modeled through the process of conceptualization (see Chapter 5).
4. Finally, a test system had to be implemented for verifying the applicability of the theoretical results (see Chapter 6).

7.1 Contributions

The new scientific results achieved during the completion of the project are summarized as follows.

Thesis 1: [8], [9], [10]
A novel semantic model is developed, called ECG, which has a logic-based ECG-HOPL and a semantically equivalent graphical ECG diagram representation. The model satisfies the requirements of the knowledge representation format in the investigated grammar induction system, and can be used as an ontology modeling language because its main building blocks are concepts and their relationships. It is predicate-centered and it defines two levels and distinct elements for describing the different phases of conceptualization. It provides high levels of functionality, flexibility and extendibility. It is computationally tractable while highly expressive, that is it covers a wide range of linguistic phenomena.

Consequences of Thesis 1:

1. Since ECG can be considered as an ontology modeling language, ECG diagram can be used for visual ontology representation. The generation of ECG diagram graphs can be accomplished by an $O(n^2)$ algorithm, where n is the number of OWL elements to be displayed.

2. ECG can also be applied as a sentence-level semantic annotation language, because every ECG-HOPL statement can be semantically unambiguously rendered into an NL sentence examined and every NL sentence under examination can be approximated by an ECG-HOPL statement.

3. ECG-HOPL can be defined with CFG, which proves that the syntax of ECG is simple enough so that a computationally effective learning algorithm can be constructed for inducing a set of grammar rules from ECG, and consequently from the sentences annotated by ECG.

Thesis 2: [7]

ECG fragment diagrams are acyclic graphs, therefore they can be converted to a tree structure the root of which is the kernel predicate. The mapping is proved to be lossless and is accomplished by an $O(n^2)$ algorithm, where n is the number of ECG diagram elements. The new ECG-TAG grammar formalism consists of edge-labeled lexicalized tree structures, the nodes of which correspond to ECG concepts, while the edges represent ECG relationships. The formalism is TAG-based, because it uses the same tree set (with different interpretation) and the same operations for tree construction as the original TAG formalism. At the same time, it is also dependency-based in the sense that edge labels represent semantic dependency relations.

Thesis 3:

The next task is to represent the semantic models and their symbolic language descriptions in a common framework. The algorithm that performs the assignment of symbolic sentence units to ECG concepts results in a new grammar formalism, called S-ECG-TAG, which combines the levels of semantics and syntax. The formalism extends the ECG-TAG formalism with a symbolic level, where the nodes include word sequences, while the edges are labeled by precedence relations representing the order of word sequences in the corresponding symbolic sentence. Hence, the symbolic level encodes word order locally and discontinuous constructions are represented by sibling nodes.

Consequences of Thesis 3:

1. The S-ECG-TAG formalism can be applied as a common framework for representing ECG diagrams and the corresponding symbolic sentences.

2. The S-ECG-TAG formalism can be applied as a formal grammar to be learnt in the grammar induction process because word sequences stored in the symbolic-level

nodes of the S-ECG-TAG derivation trees compose the vocabulary of a given language, while the labels of their edges represent a valid ordering of the word sequences.

Thesis 4: [13]
A method is developed for the execution of the conceptualization process within the learning agent examined, which involves the operations of association and generalization. According to the association algorithm, primary-level ECG diagram graphs are matched to and incorporated in an initially empty knowledge base, which is itself another (accumulated) ECG diagram graph. The matching of ECG diagram graphs is based on a hybrid context-dependent ECG diagram graph matching algorithm, and is traced back to the matching of element instances, for the examination of which an element category type lattice is defined.
The generalization algorithm is implemented as part of the association process and proceeds by introducing new (not observed) higher-level concepts into the knowledge base. First, the algorithm searches for maximal similar subgraphs which differ in only one ECG diagram graph node. For their exploration the intersection operation of two ECG diagram graphs and its extension are defined. If the differing nodes are semantically comparable on the basis of the element category type lattice, a new concept is inserted from the element instance type lattice determined as the least common generalization of the differing concepts. Finally, the relationships are updated in the knowledge base.

Consequences of Thesis 4:

1. The two operations of association and generalization together accomplish the process of conceptualization. At the end of the process, the generalized knowledge of the agent can be obtained as the top element of the lattice constructed from the set of primary-level ECG diagram graphs and the set of accumulated ECG diagram graphs resulting from the association and generalization steps executed.

2. Recursively performing the operation of graph intersection on the set of ECG diagram graphs and on the resulting sets of common subgraphs, a lattice can be built. The lower-level nodes of the lattice include individual (infrequent specialized) ECG diagram graphs, while at the top levels of the lattice frequent general subgraphs are located.

7.2 Directions of future investigations

As declared in Chapter 5, not only the least common generalization but also the greatest common specialization of two ECG graphs can be computed. Therefore the first task to be accomplished in the future is the incorporation of this computation into the system developed.
As a second task, training samples for a greater and more appropriate domain should be generated for modeling the process of generalization on all three levels.

After implementing the grammar learning algorithms of Chapter 4, the grammar induction system investigated can be developed. At this stage, there are several possible directions of research depending on the choice of symbolic language and formal grammar.

If the grammar induction system is extended to be cooperate with a sentence generation agent (which is able to assign symbolic sentences to ontology models), the system modeled in Figure 7.1 will be applicable as a natural language interface (NLI) for the ECG model and other semantic models after conversion, or even for image recognition systems.

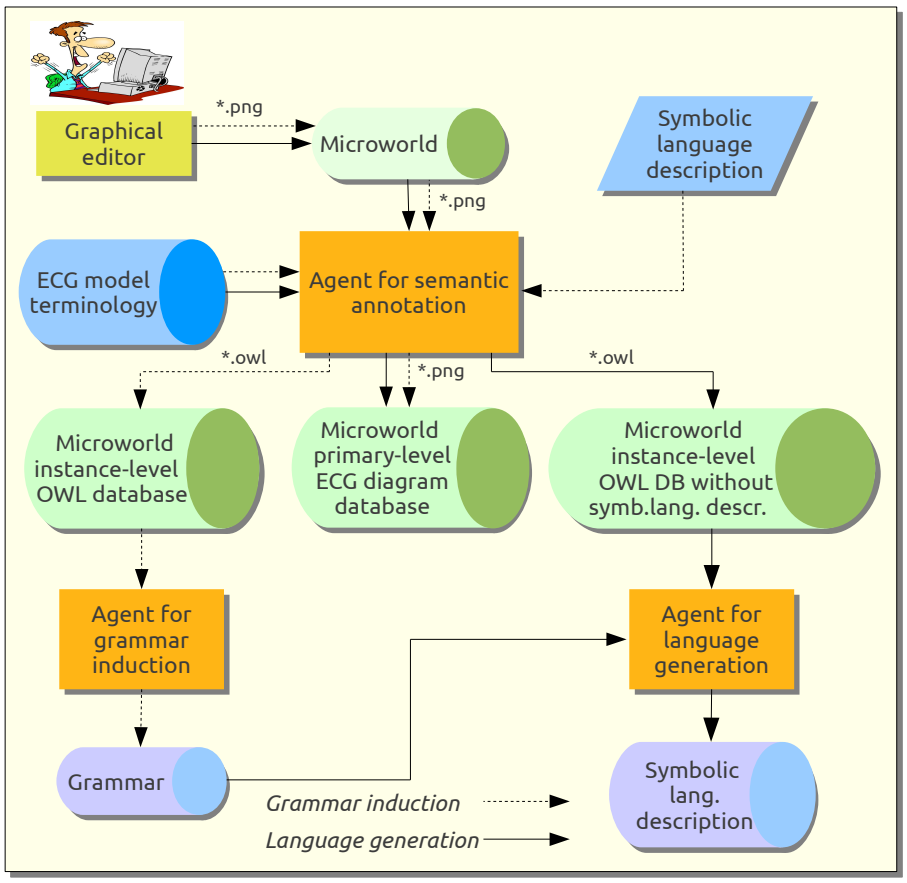

Figure 7.1 *System plan for grammar induction extended with sentence generation*

The extended system can also be used for supporting machine translation. Based on the fact that semantically equivalent assertions have identical ontology representations independent of the syntactic level of language, assertions given in two different languages can be compared and their similarity can be determined according to their semantic annotations. After that, applying the grammar of the target language (which should be learned previously) the system can generate the semantically equivalent target language sentences for the assertions in the source language.

Appendix A

DL Definition of the ECG Model

Main building blocks of the model

- $T \equiv Concept \sqcup Relationship \sqcup Container \sqcup Sentence \sqcup PLstatement$
- $Concept \equiv T \sqcap \neg Relationship \sqcap \neg Container \sqcap \neg Sentence \sqcap \neg PLstatement$
- $Relationship \equiv T \sqcap \neg Concept \sqcap \neg Container \sqcap \neg Sentence \sqcap \neg PLstatement \sqcap =1\ hasRange.Concept$
- $Container \equiv T \sqcap \neg Concept \sqcap \neg Relationship \sqcap \neg Sentence \sqcap \neg PLstatement$
- $Sentence \equiv T \sqcap \neg Concept \sqcap \neg Relationship \sqcap \neg Container \sqcap \neg PLstatement$
- $PLstatement \equiv T \sqcap \neg Concept \sqcap \neg Relationship \sqcap \neg Container \sqcap \neg Sentence$

Containers

- $History \sqsubseteq Container$
- $Snapshot \sqsubseteq History \sqcap \exists hasCategoryConcept.CategoryConcept \sqcap \exists hasPredicateConcept.PredicateConcept$
- $Fragment \equiv Snapshot \sqcap =1\ hasKernelPredicate.PredicateConcept \sqcap =1\ hasDescription.Sentence \sqcap =1\ hasPLDescription.PLstatement$

Concepts

Category concepts

- $CategoryConcept \equiv Concept \sqcap \neg PredicateConcept$
- $PrimaryCategoryConcept \equiv CategoryConcept \sqcap \neg AbstractCategoryConcept \sqcap =1\ Primary.\{true\}$
- $AbstractCategoryConcept \equiv CategoryConcept \sqcap \neg PrimaryCategoryConcept \sqcap =1\ Primary.\{false\}$
- $ClassPrimaryCategoryConcept \equiv PrimaryCategoryConcept \sqcap \neg InstancePrimaryCategoryConcept \sqcap =1\ Multiple.\{true\}$
- $InstancePrimaryCategoryConcept \equiv PrimaryCategoryConcept \sqcap \neg ClassPrimaryCategoryConcept \sqcap =1\ Multiple.\{false\}$
- $FICN \equiv InstancePrimaryCategoryConcept \sqcap =1\ Named.\{no\}$
- $FICT \equiv InstancePrimaryCategoryConcept \sqcap =1\ Named.\{temporary\}$

- FICR ≡ InstancePrimaryCategoryConcept ⊓ =1 Named.{permanent}
- FMCR ≡ ClassPrimaryCategoryConcept ⊓ =1 Named.{permanent}
- AMCR ≡ AbstractCategoryConcept ⊓ =1 Multiple.{true}
  ⊓ =1 Named.{permanent}

Predicate concepts

- PredicateConcept ≡ Concept ⊓ ¬ CategoryConcept
  ⊓ =1 Multiple.{true} ⊓ =1 Named.{permanent}
- PrimaryPredicateConcept ≡ PredicateConcept ⊓ ¬ AbstractPredicateConcept
  ⊓ =1 Primary.{true}
- AbstractPredicateConcept ≡ PredicateConcept ⊓ ¬ PrimaryPredicateConcept
  ⊓ =1 Primary.{false}
- FMPR ≡ PrimaryPredicateConcept
- AMPR ≡ AbstractPredicateConcept

Relationships

- IsaRelationship ≡ Relationship ⊓ =1 Typed.{isa} ⊓ =1 hasDomain.Concept
- RoleRelationship ≡ Relationship ⊓ =1 Typed.{role}
  ⊓ =1 hasDomain.PredicateConcept
- FMI ≡ IsaRelationship ⊓ =1 Multiple.{true} ⊓ =1 Primary.{true}
- FSR ≡ RoleRelationship ⊓ =1 Multiple.{false} ⊓ =1 Primary.{true}
- FMR ≡ RoleRelationship ⊓ =1 Multiple.{true} ⊓ =1 Primary.{true}
- AMR ≡ RoleRelationship ⊓ =1 Multiple.{true} ⊓ =1 Primary.{false}

Object property definitions

- hasDescription : Fragment → Sentence
- hasPLDescription : Fragment → PLstatement
- hasDomain : Relationship → Concept
- hasRange : Relationship → Concept
- hasCategoryConcept : Fragment → CategoryConcept
- hasPredicateConcept : Fragment → PredicateConcept
- hasKernelPredicate ⊑ hasPredicateConcept
- isKernelOf ≡ hasKernelPredicate^{-}

Datatype property definitions

- Caption : Concept ⊔ Relationship ⊔ Sentence ⊔ PLstatement → {string}
- Primary : Concept ⊔ Relationship → {boolean}
- Multiple : Concept ⊔ Relationship → {boolean}
- Named : Concept → {string:<no,temporary,permanent>}
- Typed : Relationship → {string:<isa,role>}

Appendix B

Instance-Level ECG Ontology Construction

Header of the OWL file

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns="http://www.iit.uni-miskolc.hu/vargae/filename.owl#"
    xmlns:j.0="http://www.iit.uni-miskolc.hu/vargae/ECGmodel.owl#"
    xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xml:base="http://www.iit.uni-miskolc.hu/vargae/filename.owl">
<owl:Ontology rdf:about="ECG model">
    <owl:imports
     rdf:resource="http://www.iit.uni-miskolc.hu/vargae/ECGmodel.owl"/>
</owl:Ontology>
```

Fragment description

```
<j.0:Fragment rdf:ID="Fragment_n"/>
<j.0:Sentence rdf:ID="Sentence_n">
    <j.0:Caption xml:lang="hu">...</j.0:Caption>
</j.0:Sentence>
<j.0:PLstatement rdf:ID="PLstatement_n">
    <j.0:Caption xml:lang="hu">...</j.0:Caption>
</j.0:PLstatement>
```

Specifying the kernel predicate

```
<j.0:FMPR rdf:ID="FMPR_n">
    <j.0:isKernelOf rdf:resource="#Fragment_n"/>
    <j.0:Caption xml:lang="en">...</j.0:Caption>
</j.0:FMPR>
```

Specifying a non-kernel predicate

```
<j.0:FMPR rdf:ID="FMPR_n">
   <j.0:Caption xml:lang="en">...</j.0:Caption>
</j.0:FMPR>
```

FICN category concept definition

<j.0:FICN rdf:ID="**FICN_n**">
<j.0:Caption xml:lang="en">**_n**</j.0:Caption>
</j.0:FICN>

FICT category concept definition

<j.0:FICT rdf:ID="**FICT_n**">
<j.0:Caption xml:lang="en">**#n**</j.0:Caption>
</j.0:FICT>

FICR category concept definition

<j.0:FICR rdf:ID="**FICR_n**">
<j.0:Caption xml:lang="en">**...** </j.0:Caption>
</j.0:FICR>

FMCR category concept definition

<j.0:FMCR rdf:ID="**FMCR_n**">
<j.0:Caption xml:lang="en">**...** </j.0:Caption>
</j.0:FMCR>

Specifying an Isa relation

<j.0:FMI rdf:ID="**FMI_n**">
<j.0:hasDomain rdf:resource="**#...**"/>
<j.0:hasRange rdf:resource="**#...**"/>
</j.0:FMI>

Specifying a semantic role relation

<j.0:FMR rdf:ID="**FMR_n**">
<j.0:hasDomain rdf:resource="**#...**"/>
<j.0:hasRange rdf:resource="**#...**"/>
<j.0:Caption xml:lang="en">**...** </j.0:Caption>
</j.0:FMR>

Building a fragment

<j.0:Fragment rdf:resource="**#Fragment_n**">
<j.0:hasDescription rdf:resource="**#Sentence_n**"/>
<j.0:hasPLDescription rdf:resource="**#PLstatement_n**"/>
<j.0:hasKernelPredicate rdf:resource="**#FMPR_n**"/>
<j.0:hasPredicateConcept rdf:resource="**#FMPR_n**"/>
...
<j.0:hasCategoryConcept rdf:resource="**#...**"/>
...
</j.0:Fragment>

Closing the file

</rdf:RDF>

Appendix C

Examples for ECG-TAG Derivation Tree Construction

Figure C.2 illustrates the steps of constructing the ECG-TAG derivation tree for the example in Figure C.1 describing the observation "A black circle is in a white triangle".

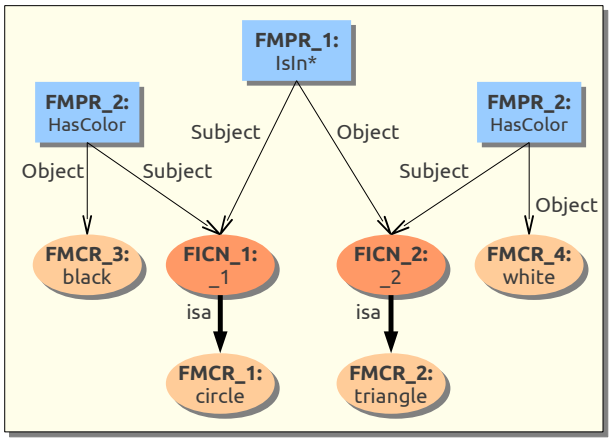

Figure C.1 *ECG diagram for "A black circle is in a white triangle"*

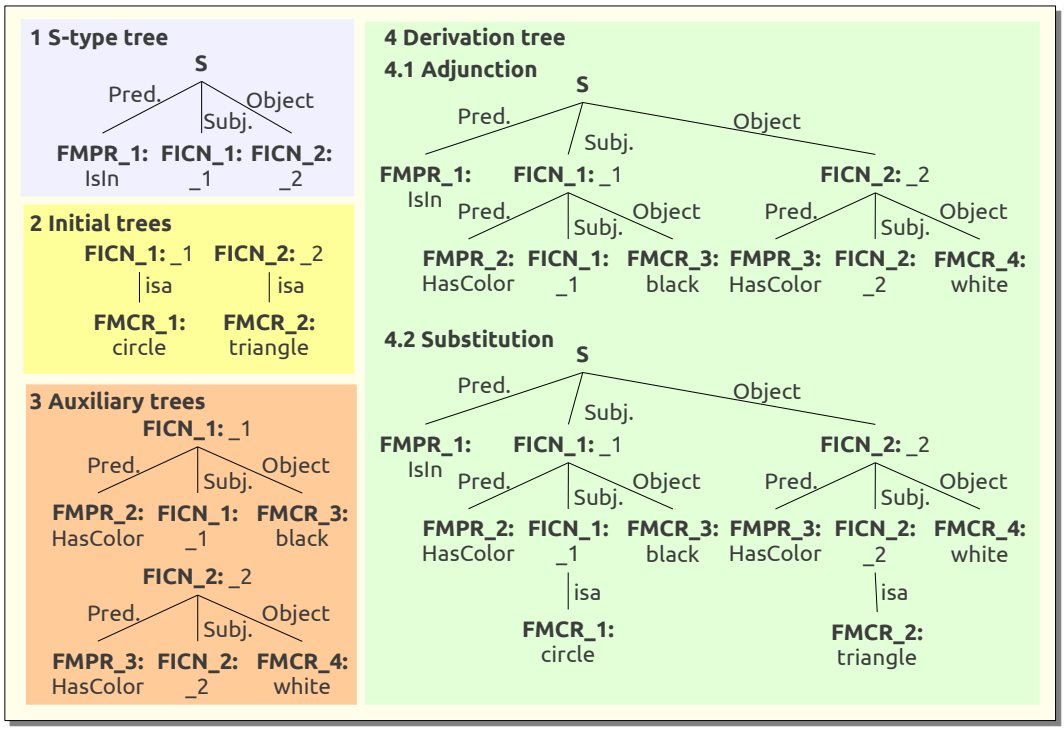

Figure C.2 *Construction of the ECG-TAG derivation tree for Fig. C.1*

A more complex example can be seen in Figure C.3 representing the ECG fragment where "A black circle is in a big white triangle". Here, two auxiliary trees with the same root node are created during the mapping process. The steps of the mapping and the resulting ECG-TAG derivation tree are shown in Figure C.4.

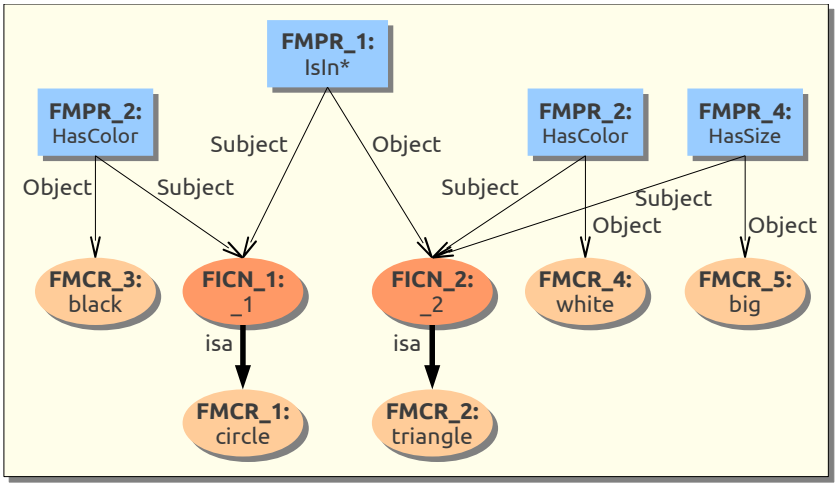

Figure C.3 *ECG diagram for "A black circle is in a big white triangle"*

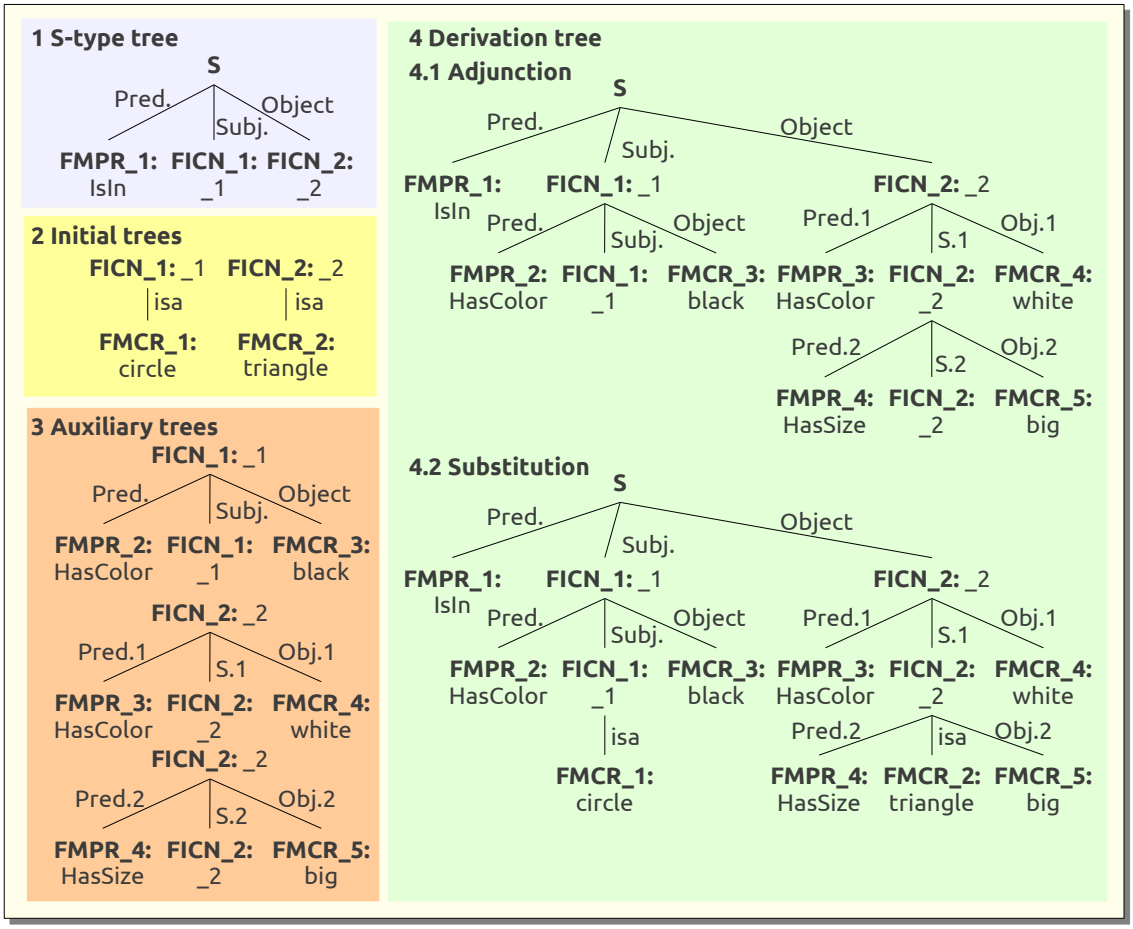

Figure C.4 *Construction of the ECG-TAG derivation tree for Fig. C.3*

Appendix D

Example for S-ECG-TAG Derivation Tree Construction

Figure D.1 shows an example of how symbolic terms are assigned to ECG concepts.

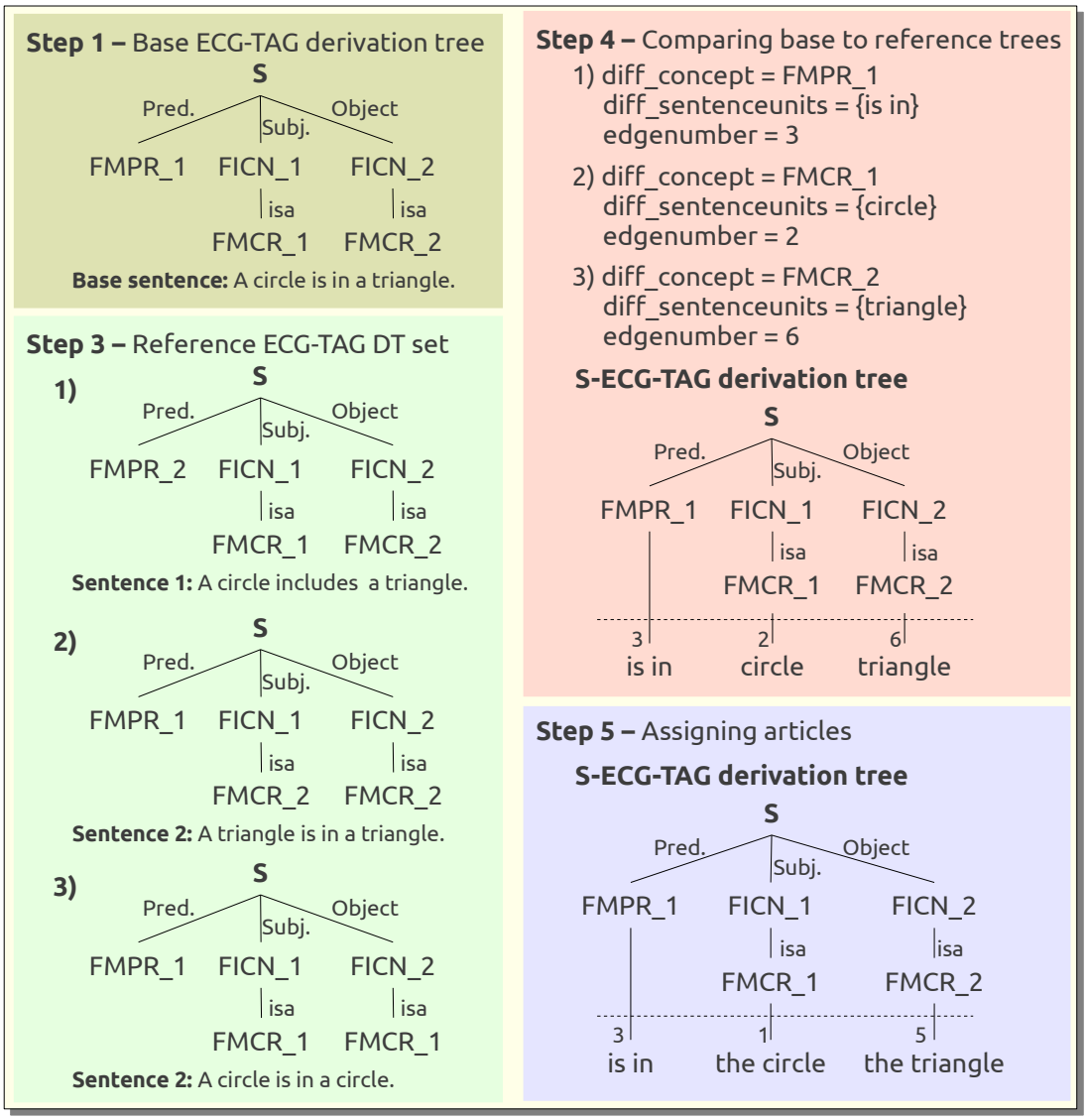

Figure D.1 *Assigning symbolic terms to ECG concepts*

Appendix E

Example for the Process of Conceptualization

Considering the example illustrated in Figures E.1 – E.5, let Γ_1 denote the knowledge base of the agent already containing one observation and Γ_2 denote the observation to be inserted into the knowledge base. Two similar subgraphs are found indicated by identical colors. The new concepts inserted from the element instance type lattice (see Figure 6.8 on page 95) are indicated by red-bordered ellipses. In the next stage Γ_3 denotes the current state of the knowledge base and Γ_4 is the new observation to be inserted. Again, two similar subgraphs are identified and two new concepts are inserted. As a result, after processing three observations Γ_5 represents the actual state of the knowledge base.

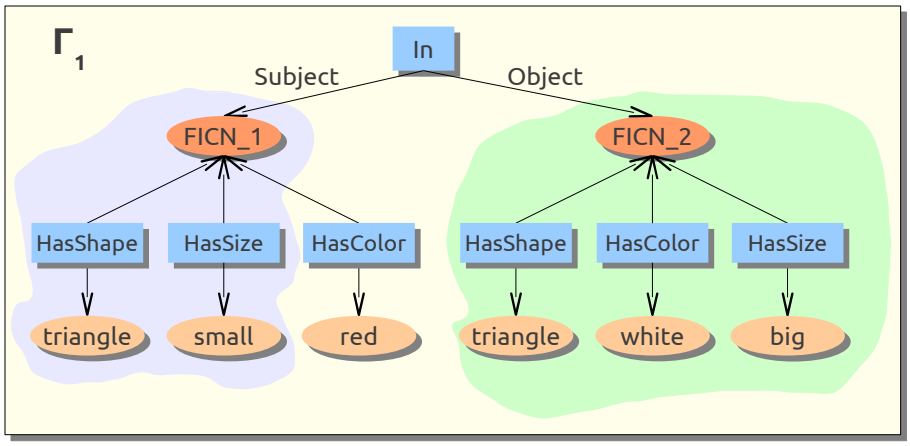

Figure E.1 *Initial state of the knowledge base containing one observation*

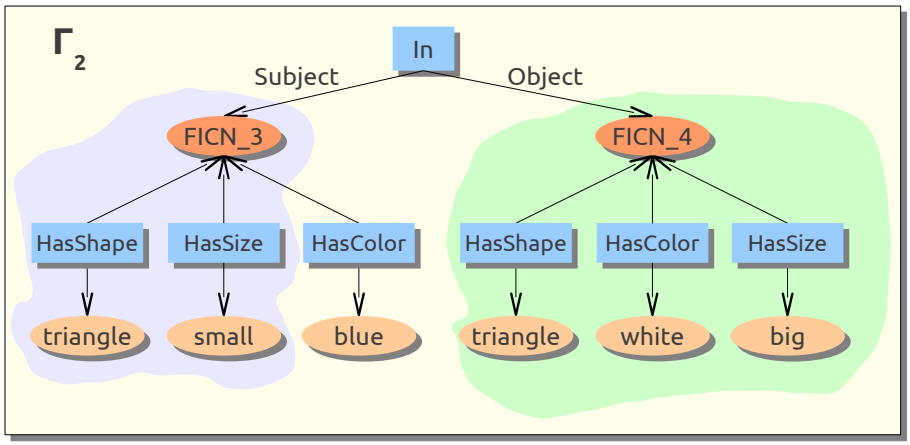

Figure E.2 *First new observation to be inserted into the knowledge base*

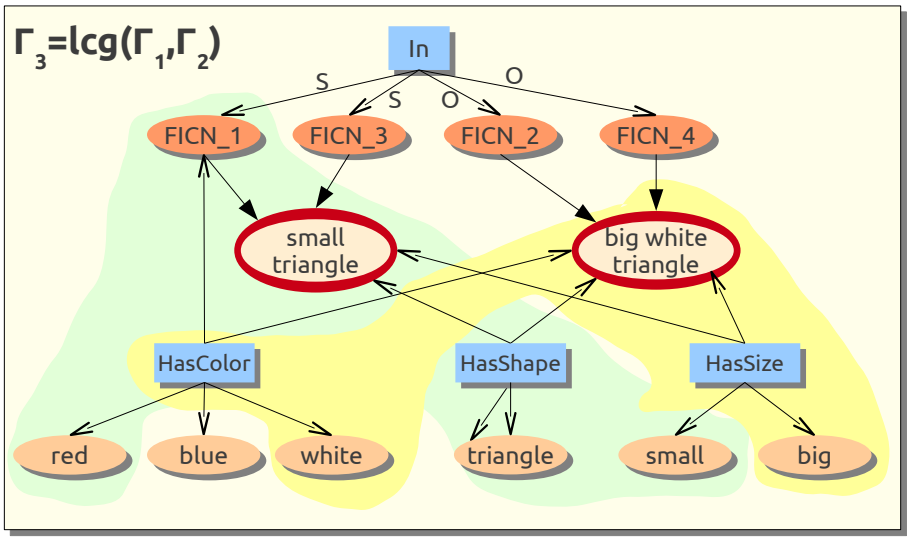

Figure E.3 *Current state of the knowledge base containing two observations*

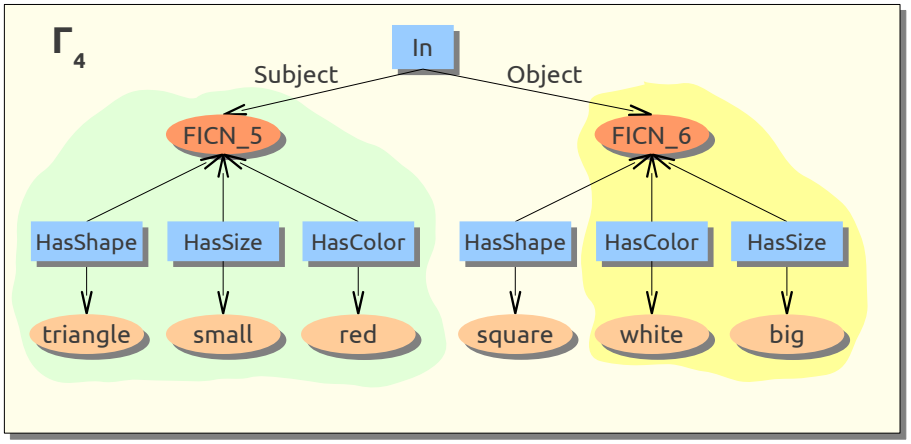

Figure E.4 *Second new observation to be inserted into the knowledge base*

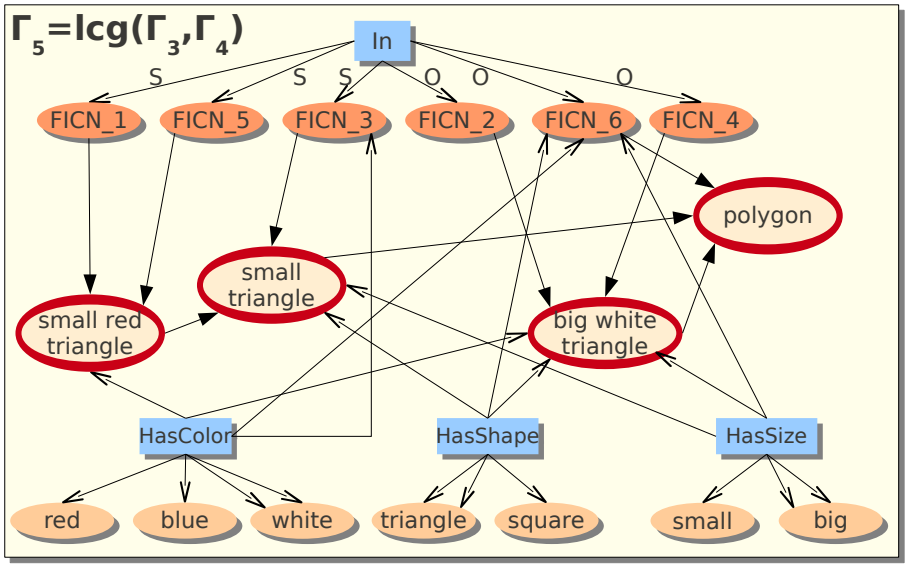

Figure E.5 *Current state of the knowledge base containing three observations*

Reference List

[Atkin, 2007] Atkin, A. (2007). Peirce's Theory of Signs. In The Stanford Encyclopedia of Philosophy.

[Atwell et al., 2000] Atwell, E., Demetriou, G., Hughes, J., Schiffrin, A., Souter, C., & Wilcock, S. (2000). A comparative evaluation of modern English corpus grammatical annotation schemes. ICAME Journal, 24, pp. 7–23.

[Baader et al., 2003] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., & Patel-Schneider, P. (2003). The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press.

[Bach, 2004] Bach, I. (2004). Formális nyelvek. Budapest: Neumann Kht.

[Barwise & Cooper, 1981] Barwise, J. & Cooper, R. (1981). Generalized quantifiers and natural language. Linguistics and Philosophy, 4, pp. 159–219.

[Bechhofer, 2002] Bechhofer, S. (2002). Ontology Language Standardization Efforts. Technical Report IST Project IST-2000-29243, Information Management Group, Department of Computer Science, University of Manchester, UK.

[Bechhofer et al., 2004] Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D., Patel-Schneider, P., & Stein, L. (2004). OWL Web Ontology Language Reference, W3C Recommendation.

[Ben-Yami, 2004] Ben-Yami, H. (2004). Logic & Natural Language. Ashgate.

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. Scientific American.

[Blackburn & Bos, 2003] Blackburn, P. & Bos, J. (2003). Computational semantics. Theoria, 18, pp. 27–45.

[Bodon, 2010] Bodon, F. (2010). Adatbányászati algoritmusok, tanulmány. http://www.cs.bme.hu/~bodon/magyar/adatbanyaszat/tanulmany/adatbanyaszat.pdf.

[Bognár, 2000] Bognár, K. (2000). Leíró logikák az ismeretábrázolásban. Alkalmazott Matematikai Lapok, 20(2), pp. 183–193.

[Calí et al., 2005] Calí, A., Calvanese, D., Grau, B. C., Giacomo, G. D., Lembo, D., Lenzerini, M., Lutz, C., Milano, D., Möller, R., Poggi, A., & Sattler, U. (2005). State of the art survey. Technical Report WP1 – Assessment of Fundamental Ontology Based Tasks, FP6-7603 Thinking ONtologiES (TONES) project.

[Carroll, 2001] Carroll, J. (2001). Matching RDF Graphs. Technical Report HPL-2001-293, Information Infrastructure Laboratory, HP Laboratories Bristol.

[Charniak, 1996] Charniak, E. (1996). Statistical Language Learning. Cambridge, MA: MIT Press.

[Chomsky, 1956] Chomsky, A. (1956). Three models for the description of language. IRE Transactions on Information Theory, 2(2), pp. 113–123.

[Chomsky, 1957] Chomsky, A. (1957). Syntactic Structures. The Hague: Mouton & Co.

[Clark, 2001] Clark, A. (2001). Unsupervised Language Acquisition: Theory and Practice. PhD thesis, COGS, University of Sussex.

[Cranefield & Purvis, 1999] Cranefield, S. & Purvis, M. (1999). UML as an ontology modeling language. In In Proceedings of the Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence (IJCAI-99): pp. 46–53.

[Davis et al., 1993] Davis, R., Shrobe, H., & Szolovits, P. (1993). What is a knowledge representation? AI Magazine, 14(1), pp. 17–33.

[Fargues et al., 1986] Fargues, J., Landau, M., Dugourd, A., & Catach, L. (1986). Conceptual graphs for semantics and knowledge processing. IBM J. Res. Develop., 30(1).

[Fillmore, 1968] Fillmore, C. (1968). The case for case. In E. Bach & R. Harms (Eds.), Universals in Linguistic Theory pp. 1–88. New York: Holt Rinehart and Winston.

[Futó, 1999] Futó, I., Ed. (1999). Mesterséges Intelligencia. Aula Kiadó.

[Ganter & Wille, 1999] Ganter, B. & Wille, R. (1999). Formal Concept Analysis, Mathematical Foundations. Springer Verlag.

[Genesereth, 1998] Genesereth, M. R. (1998). Knowledge Interchange Format. Draft proposed American National Standard (dpANS). NCITS.T2/98-004.

[Godin et al., 1995] Godin, R., Missaoui, R., & Alaoui, H. (1995). Incremental concept formation algorithms based on Galois lattices. Computational Intelligence, 11(2), pp. 246–267.

[Gold, 1967] Gold, E. (1967). Language identification in the limit. Information Control, 10, pp. 447–474.

[Gruber, 1993] Gruber, T. (1993). A translation approach to portable ontology specifications. Knowledge Acquisition, 5(2), pp. 199–220.

[Guarino, 1998] Guarino, N. (1998). Formal Ontology and Information Systems. In N. Guarino (Ed.), Formal Ontology in Information Systems (FOIS'98) Trento, Italy: IOS Press, Amsterdam pp. 3–15.

[Harrison, 1978] Harrison, M. (1978). Introduction to Formal Language Theory. Addison-Wesley.

[Hartshorne et al., 1958] Hartshorne, C., Weiss, P., & Burks, A., Eds. (1931–1958). Collected Papers of C. S. Peirce. Cambridge, MA: Harvard University Press.

[Hayes, 2004] Hayes, P. (2004). RDF Semantics. W3C Recommendation.

[Hinman, 2005] Hinman, P. (2005). Fundamentals of Mathematical Logic. A.K. Peters.

[Ilieva, 2007] Ilieva, M. (2007). Graphical notation for natural language and knowledge representation. In 19th SEKE.

[Jarrar et al., 2003] Jarrar, M., Demey, J., & Meersman, R. (2003). On using conceptual data modeling for ontology engineering. Journal on Data Semantics, pp. 185–207.

[Jaworski & Unold, 2007] Jaworski, M. & Unold, O. (2007). Improved TBL algorithm for learning context-free grammar. In International Multiconference on Computer Science and Information Technology: pp. 267–274.

[Joshi, 1985] Joshi, A. (1985). Natural Language Parsing, chapter Tree-adjoining grammars: How much context sensitivity is required to provide reasonable structural descriptions?, pp. 206–250. Cambridge University Press.

[Joshi et al., 1975] Joshi, A., Levy, L., & Takahashi, M. (1975). Tree adjunct grammars. Journal Computer Systems Science, 10(1).

[Joshi & Rambow, 2003] Joshi, A. & Rambow, O. (2003). A formalism for Dependency Grammar based on Tree Adjoining Grammar. MTT 2003 Paris.

[Joshi & Schabes, 1997] Joshi, A. & Schabes, Y. (1997). Handbook of Formal Languages, chapter Tree-Adjoining Grammars, pp. 69–123. Springer: Berlin.

[Jurafsky & Martin, 2000] Jurafsky, D. & Martin, J. (2000). Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. New Jersey: Prentice Hall.

[Kallmeyer, 1997] Kallmeyer, L. (1997). A syntax-semantics interface with synchronous tree description grammars. Formal Grammar 1997. Linguistic Aspects of Logical and Computational Perspectives on Language, pp. 112–124.

[Kallmeyer, 2002] Kallmeyer, L. (2002). Enriching the TAG derivation tree for semantics. In 6. Konferenz zur Verarbeitung natürlicher Sprache, KONVENS 2002 Saarbrücken: pp. 67–74.

[Keenan, 2005] Keenan, E. (2005). How much logic is built into natural language? In P. D. et al (Ed.), Fifteenth Amsterdam Colloquium: ILLC, University of Amsterdam pp. 39–45.

[Klyne & Carroll, 2004] Klyne, G. & Carroll, J. (2004). Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation.

[Kovács, 2004] Kovács, L. (2004). Adatbázisok tervezésének és kezelésének módszertana. Budapest: ComputerBooks.

[Kovács, 2006] Kovács, L. (2006). Concept lattice structure with attribute lattices. Production Systems and Information Engineering, 4, pp. 65–81.

[Kovács & Sieber, 2009] Kovács, L. & Sieber, T. (2009). Multi-layered semantic data models. In Encyclopedia of Artificial Intelligence pp. 1130–1135. Hersey (USA): IGI Global Publisher.

[Kremer, 1998] Kremer, R. (1998). Visual languages for knowledge representation. In 11th Workshop on Knowledge Acquisition, Modeling and Management (KAW'98) Banff, Alberta, Canada.

[Krenn & Samuelsson, 1997] Krenn, B. & Samuelsson, C. (1997). The Linguistic's Guide to Statistics.

[Manning & Schütze, 1999] Manning, C. & Schütze, H. (1999). Foundations of Statistical Natural Language Processing. Cambridge, MA: MIT Press.

[Markov, 1913] Markov, A. (1913). An example of statistical investigation in the text of 'Eugene Onyegin' illustrating coupling of 'tests' in chains. Proc. of the Academy of Sciences, St. Petersburg, VI(7), pp. 153–162.

[McEnery et al., 2005] McEnery, A., Xiao, R., & Tono, Y. (2005). Corpus-Based Language Studies: An Advanced Resource Book. Routledge Applied Linguistics. Routledge.

[McKay, 1981] McKay, B. D. (1981). Practical graph isomorphism. Congressus Numerantium, 30.

[Minsky, 1975] Minsky, M. (1975). A Framework for Representing Knowledge. In P. Winston (Ed.), The Psychology of Computer Vision. New York: McGraw-Hill.

[Muresan, 2006] Muresan, S. (2006). Learning Constraint-based Grammars from Representative Examples: Theory and Applications. PhD thesis, Columbia University, NY.

[Ogden & Richards, 1923] Ogden, C. & Richards, I. (1923). The Meaning of Meaning: A Study of the Influence of Language Upon Thought and of the Science of Symbolism. London: Routledge & Kegan Paul.

[Patel-Schneider et al., 2004] Patel-Schneider, P., Hayes, P., & Horrocks, I. (2004). OWL Web Ontology Language Semantics and Abstract Syntax , W3C Recommendation.

[Peregrin, 1997] Peregrin, J. (1997). What does one need when she needs 'higher-order' logic? In Filosofia: LOGICA'96, Praha.

[Ponsen et al., 2010] Ponsen, M., Taylor, M., & Tuyls, K. (2010). Abstraction and Generalization in Reinforcement Learning: A Summary and Framework. ALA Workshop, Adaptive and Learning Agents (LNAI Journal).

[Quillian, 1968] Quillian, M. (1968). Semantic Information Processing, chapter Semantic Memory, pp. 216–270. MIT Press: Cambridge, MA.

[Rambow & Joshi, 1997] Rambow, O. & Joshi, A. (1997). Recent Trends in Meaning-Text Theory, chapter A formal look at dependency grammars and phrase-structure grammars, with special consideration of word-order phenomena, pp. 167–190. John Benjamins: Amsterdam and Philadelphia.

[Reeve & Han, 2005] Reeve, L. & Han, H. (2005). Survey of semantic annotation platforms. In 2005 ACM Symposium on Applied Computing Santa Fe, New Mexico: pp. 1634–1638.

[Roberts & Atwell, 2002] Roberts, A. & Atwell, E. (2002). Unsupervised Grammar Inference Systems for Natural Language. Technical Report 2002.20, University of Leeds, School of Computing.

[Rückert, 2008] Rückert, U. (2008). A Statistical Approach to Rule Learning. PhD thesis, Technischen Universität München.

[Sántáné-Tóth, 2006] Sántáné-Tóth, E. (2006). Ontológia – Oktatási segédlet.

[Scriptum, 2005] Scriptum (2005). Ontológia-építő nyelvek értékelése, elemző összehasonlítása. Technical Report MEO projekt, Scriptum Rt.

[Shapiro, 2001] Shapiro, S. (2001). The Blackwell Guide to Philosophical Logic, chapter Classical logic II: Higher order logic. Blackwell Publishers.

[Shieber, 1985] Shieber, S. (1985). Evidence against the context-freeness of natural language. Linguistics and Philosophy, (8), pp. 333–343.

[Shieber & Schabes, 1990] Shieber, S. & Schabes, Y. (1990). Synchronous Tree-Adjoining Grammars. COLING: pp. 253–258.

[Shvaiko & Euzenat, 2004] Shvaiko, P. & Euzenat, J. (2004). A Survey of Schema-based Matching Approaches. Technical Report DIT-04-087, Informatica e Telecomunicazioni, University of Trento.

[Sieber, 2008] Sieber, T. (2008). Ontology-based modeling and reuse of technical documentation. PhD thesis, József Hatvany Doctoral School for Information Science, Engineering and Technology, University of Miskolc, Hungary.

[Sowa, 1976] Sowa, J. (1976). Conceptual graphs for a database interface. IBM Journal of Research and Development, 20(4), pp. 336–357.

[Sowa, 1984] Sowa, J. (1984). Conceptual Structures: Information Processing in Mind and Machine. Reading, MA: Addison-Wesley.

[Sowa, 1991] Sowa, J., Ed. (1991). Principles of Semantic Networks: Explorations in the Representation of Knowledge. San Mateo, CA: Morgan Kaufmann Publishers.

[Sowa, 1992] Sowa, J. (1992). Encyclopedia of Artificial Intelligence (ed. by S.C. Shapiro), chapter Semantic Networks. Wiley, 2nd edition.

[Sowa, 2000a] Sowa, J. (2000a). Knowledge Representation: Logical, Philosophical, and Computational Foundations. Pacific Grove, CA: Brooks Cole Publishing Co.

[Sowa, 2000b] Sowa, J. (2000b). Ontology, Metadata, and Semiotics. In Conceptual Structures: Logical, Linguistic, and Computational Issues, number 1867 in Lecture Notes in AI pp. 55–81. Berlin: Springer-Verlag.

[Sowa, 2006] Sowa, J. (2006). Peirce's contributions to the 21st century. In 14th International Conference on Conceptual Structures (ICCS 2006) Aalborg, Denmark: pp. 54–69.

[Szeredi et al., 2005] Szeredi, P., Lukácsy, G., & Benkő, T. (2005). A szemantikus világháló elmélete és gyakorlata. Budapest: Typotex.

[Tesniére, 1959] Tesniére, L. (1959). Éléments de syntaxe structurale. Paris: Klincksieck.

[Ullmann, 1976] Ullmann, J. R. (1976). An algorithm for subgraph isomorphism. ACM, 23(1).

[Valiant, 1984] Valiant, L. G. (1984). A theory of the learnable. Communications of the ACM, 27(11), pp. 1134–1142.

[Varga & Várterész, 2003] Varga, K. P. & Várterész, M. (2003). A matematikai logika alkalmazásszemléletű tárgyalása. Budapest: Panem.

[Vijay-Shanker, 1987] Vijay-Shanker, K. (1987). A Study of Tree Adjoining Grammars. PhD thesis, University of Pennsylvania, Philadelphia, PA, USA.

[Wang & Chan, 2001] Wang, X. & Chan, C. (2001). Ontology modeling using UML. In 7th International Conference on Object Oriented Information Systems Conference (OOIS'2001: pp. 59–68.

[Xueming, 2007] Xueming, L. (2007). Using UML For Conceptual Modeling: Towards An Ontological Core. PhD thesis, Memorial University of Newfoundland.

Author's Publications

[1] Varga, E. & Kovács, L. (2005). Review of Unsupervised Grammar Induction Systems. In: 5th International Conference of PhD Students, Miskolc, Hungary, pp. 201–206.

[2] Varga, E. & Kovács, L. (2005). Quality Measures of Language Learning Systems. In: 5th International Conference of PhD Students, Miskolc, Hungary, pp. 207–212.

[3] Baksa-Varga, E. & Kovács, L. (2008). A Semantic Model for Knowledge Base Representation in a Grammar Induction System. In: 1st Workshop on Computational Intelligence in Measurement, Control and Instrumentation (CIMCI 2008), Timisoara, Romania, 3, pp. 27–32.

[4] Kovács, L. & Baksa-Varga, E. (2008). Logical Representation and Assessment of Semantic Models for Knowledge Base Representation in a Grammar Induction System. In: 7th International Conference on Renewable Sources and Environmental Electrotechnologies (RSEE 2008), Oradea, Romania, pp. 48–53.

[5] Kovács, L. & Baksa-Varga, E. (2008). Logical Representation and Assessment of Semantic Models for Knowledge Base Representation in a Grammar Induction System. Journal of Computer Science and Control Systems, University of Oradea, Romania, pp. 48–53.

[6] Kovács, L. & Baksa-Varga, E. (2008). Dependency-Based Mapping between Symbolic Language and Extended Conceptual Graph. In: 6th International Symposium on Intelligent Systems and Informatics (SISY 2008), Subotica, Serbia, pn. 13.

[7] Baksáné Varga, E. & Kovács, L. (2008). Ontológia-alapú nyelvtantanuló rendszer nyelvtanmodellje. A Dunaújvárosi Főiskola Közleményei, A Magyar Tudomány Hete 2008 konferenciasorozat, Informatikai konferencia (DFTH 2008), XXX/1, pp. 219–226.

[8] Baksa-Varga, E. & Kovács, L. (2008). Knowledge Base Representation in a Grammar Induction System with Extended Conceptual Graph. Transactions on Automatic Control and Computer Science, Scientific Bulletin of "Politehnica" University of Timisoara, Romania, 53(67), pp. 107–114.

[9] Baksáné Varga, E. (2009). Magasabb rendű logika a természetes nyelvek szemantikájának reprezentálásánál. A Gépipari Tudományos Egyesület Műszaki Folyóirata (GÉP), LX. évfolyam, 2009/6, pp. 49–55.

[10] Baksa-Varga, E. & Kovács, L. (2009). Semantic Representation of Natural Language with Extended Conceptual Graph. Journal of Production Systems and Information Engineering, Vol. 5, pp. 19–39.

[11] Kovács, L. & Baksa-Varga, E. (2010). Induction of Probabilistic Context-Free Grammar Using Frequent Sequences. Journal of Advanced Computational Technologies, *in press.*

[12] Baksáné Varga, E. (2010). Ontológia-alapú szemantikai annotálást végző ágens dokumentációja. Projektjelentés. ME Általános Informatikai Tanszék, Tanszéki Közlemények. http://www.iit.uni-miskolc.hu/iitweb/opencms/research/TechReports/.

[13] Baksa-Varga, E. & Kovács, L. (2011). Generalization and Specialization Using Extended Conceptual Graphs. In: 11th International Scientific Conference on Informatics (INFORMATICS'2011), Rožňava, Slovakia, *in press.*