KNOWLEDGE ACQUISITION THROUGH NATURAL LANGUAGE CONVERSATION AND CROWDSOURCING

Luka Bradeško

Doctoral Dissertation
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia

Supervisor: Doc. Dunja Mladenić, Jožef Stefan Institute, Ljubljana, Slovenia

Evaluation Board:
Dr. Michael Witbrock, Chair, IBM, New York, New York
Prof. Erjavec, Member, Jožef Stefan Institute, Ljubljana, Slovenia
Prof. Iztok Savnik, Member, Univerza v Novi Gorici, Nova Gorica, Slovenia

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL

Luka Bradeško

KNOWLEDGE ACQUISITION THROUGH NATURAL LANGUAGE CONVERSATION AND CROWDSOURCING

Doctoral Dissertation

PRIDOBIVANJE STRUKTURIRANEGA ZNANJA SKOZI POGOVOR TER S POMOČJO MNOŽIČENJA

Doktorska disertacija

Supervisor: Doc. Dunja Mladenić

Ljubljana, Slovenia, April 2017

To the world...

Acknowledgments

Thank everyone who contributed to the thesis: - EU Projects - Cyc - Dave - Michael - Vanessa - Dunja - Coworkers

Abstract

The English abstract should not take up more than one page.

Povzetek

Povzetek v slovenščini naj ne bo daljši od ene strani.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

AST	...	Abstract Syntax Tree
CC	...	Curious Cat (a name of the knowledge acquisition application and platform that is a side result of this thesis)
CSK	...	Common Sense Knowledge
CYC	...	An AI system (Inference Engine and Ontology), developed by Cycorp Inc.
CycKB	...	Cyc Knowledge Base (Ontology part of Cyc system)
CycL	...	Cyc Lanugage
FOIL	...	First Order Inductive Learner
GAC	...	Generic Artificial Consciousness
GOKC	...	Goal-Oriented Knowledge Collection
GWAP	...	Games With A Purpose
HMI	...	Human to Machine Interaction
JSI	...	Jožef Stefan Institute
KA	...	Knowledge Acquisition
KB	...	Knowledge Base
KDML	...	Knowledge SDatabase Mark-up Language
LSA	...	Latent Semantic Analysis
MHI	...	Machine to Human Interaction
MIT	...	Massachusetts Institute of Technology
MMHHI	...	Machine Mediated Human to Human Interaction
MPI	...	Max Planck Institute
MSR	...	Microsoft Research
NL	...	Natural Language
NLP	...	Natural Language Processing
NP	...	Noun Phrase
NTU	...	National Taiwan University
OIE	...	Open Information Extraction
PMI	...	Pointwise Mutual Information
POI	...	Point of Interest
POS	...	Part of Speech
POS:X	...	Abbreviatios for Part of Speech tags used by POS parsers
POS:CC	...	Coordinating conjunction
POS:CD	...	Cardinal Number
POS:DT	...	Determiner
POS:EX	...	Existential there
POS:FW	...	Foreign Word
POS:IN	...	Preposition or subordinating conjunction
POS:JJ	...	Adjective
POS:JJR	...	Adjective, comparative
POS:JJS	...	Adjective, superlative
POS:LS	...	List item marker

POS:MD ... Modal
POS:NN ... Noun, singular or mass
POS:NNS ... Noun, plural
POS:NNP ... Proper noun, singular
POS:NNPS ... Proper noun, plural
POS:PDT ... Predeterminer
POS:POS ... Possesive ending
POS:PRP ... Personal pronoun
POS:PRP$ ... Possesive pronoun
POS:RB ... Adverb
POS:RBR ... Adverb, comparative
POS:RBS ... Adverb, superlative
POS:RP ... Particle
POS:SYM ... Symbol
POS:TO ... to
POS:UH ... Interjection
POS:VB ... Verb, base form
POS:VBD ... Verb, past tense
POS:VBG ... Verb, gerund or present participle
POS:VBN ... Verb, past participle
POS:VBP ... Verb, non-3rd person singular present
POS:VBZ ... Verb, 3rd person singular present
POS:WDT ... Wh-determniner
POS:WP ... Wh-pronoun
POS:WP$ ... Possesive wh-pronoun
POS:WRB ... Wh-adverb
PTT ... Taiwanese Bulletin Board System
SKSI ... Semantic Knowledge Source Integration
SPD ... Staypoint Detection
TUW ... The University of Waikato, New Zealand
UL ... University of Leipzig
UoM ... University of Manheim
UoR ... University of Rochester
UW ... University of Washington

Symbols

$\in$ ... Element of. Stating $x \in S$ means that x is an element of the set S.

$\wedge$ ... logical conjunction (and). The statement $A \wedge B$ is true if both A and B are true, otherwise it is false.

$\forall$ ... Universal Quantifier (for all). One of the quantifiers used to be able to convert atomic formulas into propositions. For example, $\forall x \in S : P(x)$ means that the propositional function $P(x)$ is true for every x in the set S. Or shorter. $\forall x : P(x)$, means this is true in the universal set. Given yet another example, non propositional atomic formula $x > 5$ which is neither true or false, can be converted into true/false proposition by adding a quantifier: $\forall x : x > 5$.

$\exists$ ... Existential Quantifier (there exists). One of the quantifiers, that can be used to convert atomic formulas into propositions. For example, $\exists x \in S : P(x)$ means that there exists at least one x in the set S, for which the propositional function $P(x)$ is true. Or shorter, $\exists x : P(x)$ means that there exists at least one x in the universal set, for which the porpositionalfunction is true. Given yet another, non propositional atomic formula $x > 5$ is neither true or false. But when converted to proposition by adding a quantifier it becomes either true or false in the given set: $\exists x : x > 4$.

$\Longrightarrow$ ... *Material implication*, also known as *Material conditional* or simlply *implication* is a logical connective that is used to form the statements like $p \implies f$, which can be read as "if p is true, then also q is true". If p is true and q is false, then the whole statement $p \implies q$ is false.

$\Longleftrightarrow$ ... *Material Equivalence* is a biconditional logical connective between two logical statements. In english it can read as "Id, and only if.", or "the same as". The statement $A \iff B$ is true only if both A and B are false, or both A and B are true.

Glossary

Antecedent (predicate logic) is a first half of the hypotetical proposition. It is a p part of the implies statement (see symbol $\implies$ in Chapter Symbols for explanation. In an implication $p \implies q$, p is an antecedent.

Arity (predicate logic) is a property of predicate that defines the number of parameters or operands that predicate can operate with. For example, if a predicate P has *arity* of 2, valid statements using this predicate can only be the ones with exactly 2 operands $(P(x, y), P(a, b), ...)$. Statement $P(x)$ is in this case not a valid statement, since it uses the predicate with only one parameter.

AST (Abstract Syntax Tree) is an abstract representation of Wikipedia page as parsed from DBPedia parser. Something like DOM tree for Wikipedia instead for pure HTML

Atomic Formula (predicate logic). If the predicate P has arity n, then P followed by n constants and variables is an atomic formula. Examples: $P(a), P(x), P(x, y)D(a, x)$.

Consequent (predicate logic) is a second half of the hypotetical proposition. It is a q part of the implies statement (see symbol $\implies$ in Chapter Symbols for explanation. In an implication $p \implies q$, q is a consequent or apodosis.

Constant (predicate logic) is besides *variables*, *predicates*, and *quantifiers* one of the atomic parts of the *predicate logic* sentences. For example, in a sentence $P(a, x)$, a serves as a constant. Constants are usually marked with the letters from the beginning of the alphabet. In this thesis, also predicate is a constant and all constants are written either with letters or their $Names$.

Existential Quantifier ($\exists$). For explanation see the symbol $\exists$ in the chapter Symbols.

First order logic can also be called *Predicate logic*. See this term for more refined definition

Material Equivalence ($\iff$). For explanation see the symbol $\iff$ in the chapter Symbols.

Material implication ($\implies$). For explanation, see $\implies$ in the Chapter Symbols.

OIE (Open Information Extraction is a paradigm introduced by Oren Etizoni in his TextRunner system. The main idea of this paradigm is that the knowledge acquisition system is not pre-determined to extract some specific facts, patterns, etc, but is open-ended, extracting large set of relational tuples without any human input.

PMI (Pointwise Mutual Information is a measure which captures co-occurence relationsip between terms in a big corpus.

predicate is a a term used in predicate logic, representing a verb template that desribes properties of objects, or relationships between multiple objects.

Predicate logic, called also *First order logic* is a formal system that uses quantification over variables. This makes this logic more expressive than the *Propositional logic*. In some limited sense, *Predicate logic* could be defined as *Propositional logic* with quantifiers.

proposition. This term is often synonim for a logical *statement*, but can also mean more abstract meaning that two different statements with the same meaning represent. In *Propositional logic*, a proposition is the smalles syntactic unit. On the other hand, in *Predicate logic*, statements/sentences are broken into *constatants, variables, predicates* and *quantifiers*.

Propositional function, is an atomic function in from the *Predicate logic* which is open ended (missing quantifiers) and thus cannot count as proposition. For example, $P(x)$ is a propositional function, while $\forall x P(x)$ is a proposition.

Propositional logic, also known as *sentential* or *statement logic*, is the branch of logic that operates with entire propositions/statements/sentences to form more complicated propositions/statements/sentences, and also logical relationships and properties derived from combining or altering this statements.

Quantifier (logical). Quantifiers in *Predicate logic* convert propositional functions (open ended) into proper propositions which can be true or false. For example, $P(x)$ is a propositional function, which can get converted into proper proposition using one of the quantifiers: $\forall x P(x)$. For more info look for the terms *Universal Quantifiier* and *Extistential Quantifier*.

Sentential logic. See the term *Propositional logic*.

SKSI (Semantic Knowledge Source Integration) is a *Cyc* sub-system for external knowledge integration.

Statement logic. Synonim for *Propositional logic*. For description see the glossary for this term.

Upper Ontology (also top-level, foundation or core ontology) is the part of ontology (or knowledge base), which defines the core objects that serve as a main knowledge building blocks to construct the full knowledge base.

Universal Quantifier (∀). For explanation check the symbol ∀ in the chapter Symbols.

Chapter 1

Introduction

An intelligent being or machine solving any kind of a problem needs knowledge to which it can apply its intelligence while coming up with an appropriate solution. This is especially true for the knowledge-driven AI systems which constitute a significant fraction of general AI research. For these applications, getting and formalizing the right amount of knowledge is crucial. This knowledge is acquired by some sort of Knowledge Acquisition (KA) process, which can be manual, automatic or semi-automatic. Knowledge acquisition using an appropriate representation and subsequent knowledge maintenance are two of the fundamental and as-yet unsolved challenges of AI. Knowledge is still expensive to retrieve and to maintain. This is becoming increasingly obvious, with the rise of chat-bots and other conversational agents and AI assistants. The most developed of these (Siri, Cortana, Google Now, Alexa), are backed by huge financial support from their producing companies, and the lesser-known ones still result from 7 or more person-years of effort by individuals Finish :

Knowledge acquisition and subsequent knowledge maintenance, are two of the fundamental and as-yet not-completely-solved challenges of Artificial Intelligence (AI).

We propose and implement novel approach to automated knowledge acquisition using the user context obtained from a mobile device and knowledge based conversational crowdsourcing. The resulting system named Curious Cat has a multi objective goal, where KA is the primary goal, while having an intelligent assistant and a conversational agent as secondary goals. The aim is to perform KA effortlessly and accurately while having a conversation about concepts which have some connection to the user, allowing the system (or the user) to follow the links in the conversation to other connected topics. We also allow to lead the conversation off topic and to other domains for a while and possibly gather additional, unexpected knowledge. For illustration see the example conversation sketch in Table I, where topic changes from a specific restaurant to a type of dish. In this example case, the conversation is started by the system when user stays at the same location for 5 minutes.

1.1 Scientific Contributions

This section gives an overview of scientific and other contributions of this thesis to the knowledge acquisition approaches.

1.1.1 Novel Approach Towards Knowledge Acquisition

Traditionally KA (knowledge acquisition) approach focuses on one type of acquisition process, which can be either Labor, Interaction, Mining or Reasoning(**Zang2013**). In

this thesis we propose a novel, previously untried approach that intervenes all aforementioned types with current user context and crowdsourcing into a coherent, collaborative and autonomous KA system. It uses existing knowledge and user context, to automatically deduce and detect missing or unconfirmed knowledge(reasoning) and uses this info to generate crowdsourcing tasks for the right audience at the right time(labor). These tasks are presented to users in natural language (NL) as part of the contextual conversation (interaction) and the answers parsed (mining) and placed into the KB after consistency checks(reasoning). The approach contribution can be summed up as a) definition of the framework for autonomous and collaborative knowledge acquisition with the help of contextual knowledge (chapter X), and b) demonstrate and evaluate the contributions of contextual knowledge and approach in general chapter X.

1.1.2 Knowledge Acquisition Platform Implementation as Technical Contribution

Implementation of the KA framework as a working real-world prototype which shows the feasibility of the approach and a way to connect many independent and complex subsystems. Sensor data, natural language, inference engine, huge pre-existing knowledge base (Cyc)(**Lenat1995**), textual patterns and crowdsourcing mechanisms are connected and interlinked into a coherent interactive application (Chapter X).

1.1.3 A Shift From NL Patterns to Logical Knowledge Representation in Conversational Agents

Besides the main contributions presented above, one aspect of the approach introduces a shift in the way how conversational agents are being developed. Normally the approach is to use textual patterns and corresponding textual responses, sometimes based on some variables, and thus encode the rules for conversation. As a consequence of natural language interaction, the proposed KA framework is in some sense a conversational agent which is driven by the knowledge and inference rules and uses patterns only for conversion from NL to logic. This shows promise as an alternative approach to building non scripted conversational engines (Chapter X).

1.2 Thesis structure

The rest of the thesis is structured in to chapters covering specific topics. Chapter X introduces

Chapter 2

Background and Problem Definition

This chapter describes the challenges and components that Knowledge Acquisition system such is presented in this work (*Curious Cat*) have to address and bring together into a working workflow in order to be a coherent KA system able to satisfy the goals of general (common) Knowledge Acquisition.

Curious Cat is a KA system making use of existing knowledge, logical inference, crowd-sourcing and mobile context, to trigger natural language questions at user-appropriate moments and then incorporate the answers consistently into the existing KB. To successfully do this, there are many inter connected steps addressing a broad range of as yet not completely solved problems from multiple fields of artificial intelligence, machine learning, natural language processing and human computer interaction. Additionally to that, given that the approach uses crowd-sourcing, there is an additional complexity of technical implementation and scalability.

For more structured explanation of the approach and challenges involved, this section is grouped into sub-sections describing main challenges. Each section then references our approach to it (implementation) and also related works that gives overview of similar approaches.

Curiois Cat is knowledge driven, meaning that knowledge is connecting all of the components, including the user interaction, and storing of the results into the KB (section 2.1). Its user context is obtained through a mobile sensor mining in a real world application that monitors the user's activity and location through mobile GPS and accelerometer sensors. This raw data is then corrected, clustered, classified and enriched before inserted into the KB as knowledge (section 2.2. The newly asserted context can trigger forward chaining operation of the inference engine (section 2.1) which can results in logical representation of a new question (section 2.3) or a statement that the system intends to show to the user. The aforementioned logical formula is then converted to natural language (sections 2.4) and presented to the user through a mobile app. When the user answers, his NL answer is converted back to logic (section 2.4), checked by the inference engine agains the existing knowledge for consistency, and inserted as new peace of knowledge into the KB (section 2.3). After iteraction like that, the system determines whether to continue the conversational path with the user or not. Newly acquired knowledge is then used to check with other users for validity (section 2.5) and is then used by the inference to produce new questions/comments/suggestions (section 2.3).

2.1 Knowledge Representation, Engineering and Inference

dada

2.2 Context (Information) Extraction

dada

2.3 Knowledge Acquisition

dada

2.4 Natural Language Processing

dada

2.5 Crowdsourcing

dada

Chapter 3

Related Work

In this chapter we will give an overview of approaches and related works on broader knowledge acquisition research field, information extraction, crowdsourcing and geo-spatial context mining.

Knowledge Acquisition has been addressed from different perspectives by many researchers in Artificial Intelligence over decades, starting already in 1970 as a sub-discipline of AI research, and since then resulting in a big number of types and implementations of approaches and technologies/algorithms. The difficulty of acquiring and maintaining the knowledge was soon noticed and was Goined as *Knowledge Acquisition Bottleneck* in 1977(**Feigenbaum1977**). In more recent survey of KA approaches (**Zang2013**), authors categorize all of the KA approaches into four main groups, regarding the source of the data and the way knowledge is acquired:

- *Labour Acquisition.* This approach uses human minds as the knowledge source. This usually involves human (expert) ontologists manually entering and encoding the knowledge.
- *Interaction Acquisition.* As in Labour Acquisition, the source of the knowledge is coming from humans, but in this case the KA is wrapped in a facilitated interaction with the system, and is sometimes implicit rather than explicit.
- *Reasoning Acquisition.* In this approach, new knowledge is automaticaGly inferred from the existing knowledge using logical rules and machine inference.
- *Mining Acquisition.* In this approach, the knowledge is extracted from some large textual corpus or corpora.

We believe this categorization most accurately reflects the current state of machine (computer) based knowledge acquisition, and we decided to use the same classification when structuring our related work, focusing more on closely related approaches and extending where necessary. According to this classification, our work presented in this thesis, fits into a hybrid approach combining all four groups, with main focus on interaction and reasoning. We address the problem by combining the labour and interaction acquisition (users answering questions as part of NL interaction aimed at some higher level goal, such as helping the user with various tasks), adding unique features of using user context and existing knowledge in combination with reasoning to produce a practically unlimited number of potential interaction acquisition tasks, going into the field of crowd-sourcing by sending these generated tasks to many users simultaneously.

Previous works that can compare with our solution is divided into the systems that exploit existing knowledge (generated anew during acquisition or pre-existing from before

Fix this, refe
to chapters i
to specifici w

in other sources) (**Singh2002a**; **Witbrock2003**; **Forbus2007**; **Kvo2010**; **Sharma2010**; **Mitchel2015**), reasoning (**Witbrock2003**; **Speer2007**; **Speer2008**; **Kuo2010**), crowd-sourcing (**Singh2002**; **Speer2009**; **Kuo2010**; **Pedro2012a**; **Pedro2013**), acquisition through interaction (**Speer2009**; **Pedro2012**; **Pedro2013**), acquisition through labour(add, probably rather refer to subsections) () and natural language conversation(**Pedro2012**; **Speer2007**; **Speer2009**; **Witbrock2003**; **Kuo2010**).

Test referencing table (see Table 3.1).

Table 3.1: Structured overview of related KA systems

System	Parent	Reference	Category	Source	Representation	Prior K.	Crowds.	Context
Cyc project (Cycorp)	/	(**Lenat1995**)	Labour	K. Exp.	CycL	/	/	/
ThoughtTrasure(Signiform)	/	(**Mueller2003**)	Labour	K. Exp.	LAGS	/	/	/
HowNet (Keen.)	/	(**Dong2010**)	Labour	K. Exp.	KDML	/	/	/
OMCS/ConceptNet (MIT)	/	(**Singh2002a**)	Labour	Public	ConceptNet	/	✓	/
GAC/Mindpixel(McKinstry)	/	(**McKinstry2008**)	Labour	Public	MindPixel	/	✓	/
SKSI (Cycorp)	Cyc	(**Masters2007**)	Labour/Integration	K. Exp.	Structured	✓	/	/
KRAKEN (Cycorp)	Cyc	(**Panton2002a**)	Interaction	D. Exp	CycL	✓	/	/
UIA (Cycorp)	Cyc	(**Witbrock2003UIA**)	Interaction	D. Exp	CycL	✓	/	/
Factivore (Cycorp)	Cyc	(**Witbrock2005**)	Interaction	D. Exp	CycL	✓	/	/
Predicate Populator (Cycorp)	Cyc	(**Witbrock2005**)	Interaction	D. Exp	CycL	✓	/	/
CURE (Cycorp)	Cyc	(**Witbrock2010**)	Interaction	D. Exp	CycL	✓	/	/
OMCommons (MIT)	OMCS	(**Speer2007**)	Interaction	Public	ConceptNet	✓	✓	/
Freebase (Metaweb/Google)	/	(**Bollacker2008**)	Interaction	Public	RDF	/	/	/
20 Questions (MIT)	OMCS	(**Speer2009**)	Game	Public	ConceptNet	/	/	/
Verbosity (CMU)		(**VonAhn2006a**)	Game	Public	/	/	✓	/
Rapport (NTU)	ConceptNet	(**Kuo2009**)	Game	Public	ConceptNet	/	✓	/
Virtual Pet (NTU)	ConceptNet	(**Kuo2009**)	Game	Public	ConceptNet	/	✓	/
GOKC (NTU)	ConceptNet	(**Kuo2010**)	Game	Public	ConceptNet	✓	✓	/
Collabio (MS)	/	(**Bernstein2010**)	Game	Public	/	/	✓	/
AIML (Alice foundation)	/	(**Wallace2003**)	Chatbot	/	AIML	/	/	/
Chatscript (Brilligunderstanding)	/	(**Wilcox2011**)	Chatbot	/	ChatScript	/	/	/
CyN (Daxtron Labs)	Cyc+AIML	(**Wilcox2011**)	Chatbot	/	AIML+Cyc	✓	/	/
PCW (Cycorp)	Cyc	(**Matuszek2004**)	Mining	Web Search	AIML+Cyc	✓	/	/
Learning Reader (NU)	Cyc	(**Forbus2007**)	Mining	Web	CycL	✓	/	/
NELL (CMU)	/	(**Mitchell2015**)	Mining	Web	Predicate l.	✓	✓	/
KnowItAll(UW)	/	(**Etzioni2004**)	Mining	Web Search	text	/	/	/
Probase (MSR)	/	(**Wu2012**)	Mining	Web	Propriatery	/	/	/
TextRunner (UW)	KnowItAll	(**Soderland2007**)	Mining	Web	text	/	/	/
ReVerb (UW)	TextRunner	(**Fader2011**)	Mining	Web	text	/	/	/
R2A2 (UW)	ReVerb	(**Etzioni2011**)	Mining	Web	text	/	/	/
ConceptMiner (MIT)	ConceptNet	(**Eslick2006**)	Mining	Web Search	ConceptNet	✓	/	/
DBPedia (UL&UoM)	Wikipedia	(**Lehmann2015**)	Mining	Wikipedia	RDF	/	✓	/
YAGO (MPI)	Wikipedia	(**Suchanek2008**)	Mining	Wikipedia	RDF	✓	/	/
Cyc+Wiki (TUW)	Cyc/Wikipedia	(**Medelyan2008**)	Mining	Wikipedia	CycL	/	✓	/
KNEXT (MPI)		(**Schubert2002**)	Mining	Penn Treebank	/	/	/	/
P. Populator+FOIL (Cyc)	Predicate Populator	(**Witbrock2005**)	Reasoning	Induction	CycL	✓	/	/
PIP (NU)	Cyc	(**Sharma2010**)	Reasoning	Induction	CycL	✓	/	/
AnalogySpace (MIT)	OMCS	(**Speer2008**)	Reasoning	Analogy	ConceptNet	✓	/	/

3.1 Labour Acquisition

This category consists of KA approaches which rely on explicit human work to collect the knowledge. A number of expert (or also untrained) ontologists or knowledge engineers is employed to codify the knowledge by hand into the given knowledge representation (formal language). Labour acquisition is the most expensive acquisition type, but it gives a high quality knowledge. It is often a crucial initial step in other KA types as well, since it can help to have some pre-existing knowledge to be able to check the consistency of the newly acquired knowledge. Labour Acquisition is often present in other KA types, even if not explicitly mentioned, since it is implicitly done when defining internal workings and structures of other KA processes. While we checked other well known systems that are result of Labour Acquisition, Cyc (mentioned below) is the most comprehensive of them and was picked as a starting point and main background knowledge and implementation base for this work.

3.1.1 Cyc

The most famous and also most comprehensive and expensive knowledge acquired this way, is Cyc KB, which is part of Cyc AI system (**Lenat1995**). It started in 1984 as a research project, with a premise that in order to be able to think like humans do, the computer needs to have knowledge about the world and the language like humans do, and there is no other way than to teach them, one concept at a time, by hand. Since 1994, the project continued through Cycorp Inc. company, which is still continuing the effort. Through the years Cyc Inc. employed computer scientists, knowledge engineers, philosophers, ontologists, linguists and domain experts, to codify the knowledge in the formal higher order logic language CycL (**Matuszek2006a**). As of 2006 (**Matuszek2006**), the effort of making Cyc was 900 non-crowdsourced human years which resulted in 7 million assertions connecting 500,000 terms and 17,000 predicates/relations (**Zang2013**), structured into consistent subtheories (Microtheories) and connected to the Cyc Inference engine and Natural Language generation. Since the implemtentation of our approach is based on Cyc, we give a more detailed description of the KB and its connected systems in section 5.1 on page 45. Cyc Project is still work in progress and continues to live and expand through various research and commercial projects.

3.1.2 ThoughtTreasure

Approximately at the same time(1994) as Cyc Inc. company was formed, Eric Mueller started to work on a similar system, which was inspired by Cyc and is similar in having a combination of common sense knowledge concepts connected to their natural language presentations. The main differentiator from Cyc is, that it tries to use simpler representation compared to first-order logic as is used in Cyc. Additionally, some parts of *ThoughtTreasure* knowledge can be presented also with finite automata, grids and scripts(**Mueller1999**; **Mueller2003**). In 2003 the knowledge of this system consisted of 25,000 concepts and 50,000 assertions. ThoughtTreasure was not so successfull as Cyc and ceased all developments in 2000 and was open-sourced on Github in 2015. link as footnote.

3.1.3 HowNet

started in 1999 and is an on-line common-sense knowledge base unveiling inter-conceptual relationships and inter-attribute relationships of concepts as connoting in lexicons of the Chinese and their English equivalents. As of 2010 it had 115,278 concepts annotated with Chinese representation, 121,262 concepts with English representation, and 662,877

knowledge base records including other concepts and attributes (**Dong2010**). HowNet knowledge is stored in the form of concept relationships and attribute relationships and is formally structured in KDML (Knowledge Database Mark-up Language), consisting of concepts (called semens in KDML) and their semantic roles.

3.1.4 Open Mind Common Sense (OMCS)

is a crowdsourcing knowledge acquisition project that started in 1999 at the MIT Media Lab(**Singh2002a**). Together with initial seed and example knowledge, the system was put online with a knowledge entry interface, so the entry was crowd-sourced and anyone interested could enter and codify the knowledge. OMCS supported collecting knowledge in multiple languages. It's main difference from the systems described above (Cyc, HowNet, ThoughtTreasure) is, that it used deliberate crowdsourcing and that it's knowledge base and representation is not strictly formal logic, but rather inter-connected pieces of natural language statements. As of 2013 (**Zang2013**), OMCS produced second biggest KB after Cyc, consisting of English (1,040,067 statements), Chinese (356,277), Portuguese (233,514), Korean (14,955), Japanese (14,546), Dutch (5,066), etc. Initial collection was done by specifying 25 human activities, where each activity got it's own user interface for free form natural language entry and also pre-defined patterns like "A hammer is for _____", where participants can enter the knowledge. Although OMCS started to build KB from scratch it shares a similarity to our CC system in a sense that it is using crowd-sourcing and also natural language patterns with empty slots to fill in missing parts. OMCS was later used in many other KA approaches as a prior knowledge, similar way as we use Cyc. After a few versions, OMCS was taken from public access and merged with multiple KBs and KA approaches into an ConceptNet KB[1] (**Speer2016**), which is now (in 2017) part of Linked Open Data (LOD) and maintained as open-source project.

3.1.5 GAC/MindPixel

Generic Artificial Consciousness was a bold try to make a general AI based on the premise that human thinking can be simulated with answering binary yes/no questions or decisions(**McKinstry20**
where humans have a natural bias toward yes answers. With this in mind, McKinstry founded GAC (later renamed to MindPixel), where internet crowd would answer yes/no questions for shares in the company which would commercially exploit the GAC AI. Initially, one such answer which contributed to GAC knowledge base was called MindPixel, but after a while this became name for the whole system. With this idea, McKinstry's MindPixel was one of the first crowd-sourced efforts for collaborative KB building. While MindPixel shut down after McKinsey stopped working on it, it inspired *OMCS, later OpenMind* and started the crowdsourcing and crowd based cross validation of the facts. Besides the *OMCS*, a similar YES/NO crowdsourcing system lives under the name *Weegy* [2].

3.1.6 Semantic Knowledge Source Integration (SKSI)

No matter how big the underlying knowledge base is, there will always be some missing knowledge, that exists somewhere else. While with knowledge acquisition it is possible to add these missing peaces, sometimes it makes more sense to keep this knowledge externally and only link it properly so it can be used. This is especially true for fast changing data such as stock values, weather forecast information, real-time measurements of traffic/ production line, etc. In such cases it is beneficial if one can link and address this data in the same way

[1] http://conceptnet.io/
[2] http://www.weegy.com

as it would have been included in the original KB. This is the goal of *SKS*(**Masters2007**) In order to connect the external source, a "wapper" knowledge needs to be defined in *Cyc* KB, which describes which concepts and instances external source represents, and how to access them (http request, SQL Query, etc.).

In *SKS* system, this "wrapper" knowledge is asserted as normal *CycL* assertions using special predicates and concepts. The descriptions are structured into three layers, where first (access layer) describes how to access the data (ie. how to connect, send queries and retrieve the content). Then the second (physical schema layer), describes how the data is structured inside the original source. The third layer (logical schema) describes in *CycL*, how the data connects to *CycKB*. Example of the logical layer is semantics on how specific columns translate into the KB. For example, table *Securities*, column *Name*, translates into instance of Cyc concept *#$Equity-Security* with the *#$nameString* linked to the value of the table cell. With this approach, it is possible to seamlessly link the existing KB with external sources, and use *Cyc* inference engine and querying mechanisms on externally connected data without even noticing it's external.

3.2 Interaction Acquisition

Similarly as with Labour KA, interaction Acquisition gets the knowledge from human minds, but in this case the acquisition is an intended side effect, while users are interacting with the software as part of some other activity/task, or as part of a motivation scheme, such as knowledge acquisition games. Besides games, the interaction could be some other user interface for solving specific tasks, or a Natural Language Conversation. This type of acquisition is most strongly correlated with the approach described in this thesis, since Curious Cat uses points (gaming), to motivate users and it interacts with user in NL, while discussing various topics (concepts). It uses the conversation to set up the context and acquire (remember) user's responses and places them properly in to the KB. Sometimes the acquired knowledge is paraphrased and presented back to user to show the 'understanding', which was first tried in OSMC (section 3.1, (**Singh2002b**)), but there only in non-conversational way as part of the input forms.

3.2.1 Interactive User Interfaces

Interactive user interfaces are the most common representation of interaction acquisition, where the user interface is constructed in a way to help user enter the data and thus make the acquisition much faster and cheaper. Historically, these systems were developed to help the labour acquisition systems, or on top of them, after parent systems reached some sort of maturity and initial knowledge stability. This is the reason why all of these systems rely or are build on top of labour acquisition (section 3.1) or mining acquisition (section 3.3) systems.

3.2.1.1 KRAKEN

system was a knowledge entry tool which allows domain experts to make meaningful additions to CYC knowledge base, without the training in the areas of artificial intelligence, ontology development, or knowledge representation(**Panton2002a**). It was developed as part of DARPA's Rapid Knowledge Formation (RKF) project in 2000. As its goal was to allow knowledge entry to non-trained experts, it started to use natural language entry and is as this, a first pre-cursor to Curious Cat system and a seed idea for it. It consists of creators, selectors, modifiers of Cyc KB building blocks, tools for consistency checks and tools for using existing knowledge to infer new things to ask. This tool, together with it's

derived solutions was later re-written and integrated into Cyc as CURE system (see below). While KRAKEN and later CURE already used Natural Language generation and parsing, and started with the idea of natural language dialogue for doing the KA, the interaction, it was missing user context (user's had to select or search the concept of interest), and also crowdsourcing aspects. Kraken was also missing rules for explicit question asking. The questions were all related to the selected concept and given as a list of natural language forms.

3.2.1.2 User Interaction Agenda (UIA)

was a web based user interface for KRAKEN KA tool(**Panton2002a**; **Witbrock2003UIA**). It worked inside a browser and it worked as responsive web-app (in 2001) by automatically triggering refresh functionality of the browser. It consisted of a menu of tools that is organized according to the recommended steps of the KE process, text entry box (query, answer, statement), center screen for the main interaction with the current tool, and a summary with a set of colored steps needed to complete current interaction. Similarly as KRAKEN itself, this interface was later improved and integrated into main Cyc system as part of CURE tool.

3.2.1.3 Factivore

This application was a Java Applet user interface for an extended KRAKEN system, meant for quick facts entering (**Witbrock2005**). On the back-end it used the same mechanisms and logical templates, while in the front-end it only allowed facts entering, as opposed to UIA, which also allowed rules (which ended up as not being useful).

3.2.1.4 Predicate Populator

Predicete Populator is a similar tool as *Factivore*, which instead of only collecting instances, allows to add general knowledge about classes. For example, instead of describing facts for a specific restaurant, it can collect general knowledge that is true for all restaurants (**Witbrock2005**). The context of the KA in this case, is given by class concept, a predicate and a web-site which is parsed into CycL concepts. These are then filtered out if they do not match argument constraints of the predicate and then shown to user for selection. As part of the validation, this tool had some problems with correctly acquired knowledge. One of the proposed solutions (never implemented), was to start using volunteers to vote about the correctness. This is already a pre-cursor idea for crowd-sourced voting mechanisms that we used in Curious Cat.

3.2.1.5 Freebase

Freebase started in 2007(**Bollacker2008**) and was a large (mostly instance based) crowd-sourced graph database for structured general human knowledge. Initially it was acquired from multiple public sources, mostly Wikipedia. The initial seed was then constantly updated and corrected by the community. On the user interface side, Freebase provides an AJAX/Web based UI for humans and an HTTP/JSON based API for software access. For finding knowledge and also software based editing, it uses Metaweb Query Language (MQL). A company behind freebase was bought by Google in 2010 and incorporated into a Google Knowledge Graph. In 2016 Freebase was incorporated into the Wikidata platform and shut down by Google and is no longer maintained.

3.2.1.6 OMCommons (Open Mind Commons)

This system is an interactive interface to OMCS which can respond with a feedback to user answers and maintain dialogue (**Speer2007**). This is similar approach as we do with Curious Cat and shows understanding of the knowledge users enter. The mechanisms behind is by using inference engine to make analogical inferences based on the existing knowledge and new entry. Then it generates some relevant questions and asks user to confirm them. For example, as given from the original paper, *OMCommons* asks: "A bicycle would be found on the street. Is this common sense?". This is then displayed to the user with the justification for the question: "A bicycle is similar to a car. I have been told that a car would be found on the street". Users then click on "Yes/No" buttons to confirm or reject the inferred statement. The interactive interface also allows its users to refine the knowledge entered by other users and see the ratings. Users can also explore what new inferences are result of their new contributions.

3.2.2 Games

Games are a specific sub-section of interaction acquisition, where the actual acquisition is hidden or transformed into much more enjoyable process, maximizing the entertainment of the users. This type of KA was first officially introduced by Luis von Ahn in 2006 (**VonAhn2006**; **VonAhn2008**) under the name 'Games with Purpose' paradigm.

3.2.2.1 20Q (20 Questions)

This is a game with intentional knowledge acquisition task which focuses to the most salient properties of concepts. The game itself is a standard 20 questions game which aims to make one player figure out the concept of discussion by asking yes/no questions and then infer from the answers what the concept could be. The only difference is that the player which is asking is a computer based on OMCS knowledge base. It generates questions in NL, and according to what a player answers, it attempts to guess the concept. To decide what questions to ask, it uses statistical classification methods (**Speer2009**), to discover the most informative attributes of concepts in OMCS KB. After the user answers all the questions, including whether the detected concept was right or not, the concept and the answers will be assigned to proper cluster and thus the characteristics of the object are learned.

3.2.2.2 Verbosity

. Similarly as Q20 above, Verbosity is a spoken game for two persons randomly selected online. It was inspired by Taboo board game(**TabooGame**) which required players to state common sense facts without mentioning the secret concept. While having similar game-play as aforementioned board game, Verbositywas developed with the intent to collect common sense knowledge (**VonAhn2006a**). One player (narrator), gets a secret word concept and needs to give hints about the word to the other player (guesser), who must figure out the word that is described the hints. The hints take the form of sentence templates with blanks to be filled in. For example, if the word is "CAR", the narrator could say "it has wheels." In the experiments, a total of 267 people played the game and collected 7,871 facts. While these facts were mostly a good quality and it was proven that the game can be used successfully, these facts were natural language snippets and were not incorporated into any kind of structure or formal KB.

3.2.2.3 Rapport

This is a KA game based on Chinese OMCS questions, but implemented as a Facebook game to make use of the social connections inside social network. The Game helps users to make new friends or enhance connections with their existing social network by asking and answering questions and matching the answers to other users(**Kuo2009**). This game aims to enhance the experience and community engagement and thus functionality of aforementioned *Verbosity* game, by employing simultaneous interaction between all the players versus only 1 to 1 interaction between 2 community members. For evaluation, the answers where multiple users answered the same were considered valid. This game had a similarity with Curious Cat in a sense that it employed the voting mechanism for the same answers, and the repetitive questioning of the same question to multiple users. Authors found out that the agreement between same answers of the repetitive question and voting is 80% or more after at least 2 repetitions of the same question. In 6 months, *Rapport* collected 14,001 unique statements from 1,700 users. Normalized, this is 8.2 answers per user.

3.2.2.4 Virtual Pet

This is a similar game as *Rapport* in a sense that it uses *OMCS* patterns, is in Chinese and is developed by the same authors (**Kuo2009**). Instead of Facebook platform, *Virtual Pet* uses PTT (Taiwanese bulleting board system in Chinese language). Instead of direct interaction between the users themselves, users interact with virtual pet and can ask it questions and answer it's questions. In the back-end, the questions the pet asks, are actually questions from other users. This game in 6 months collected 511,734 unique pieces of knowledge from 6,899 users. Normalized this is 74,1 answers per user. While this game attracted much more answers than *Rapport*, the quality of the answers was slightly lower. Authors argue that the reasons behind both is, that users didn't interact directly, but through the virtual pet, so they were less careful whether answers are correct or not.

3.2.2.5 Goal Oriented Knowledge Collection (GOKC)

. This game builds on the findings and approach of *Virtual Pet* KA game. The main improvement is to try and actually make use of the new knowledge inside a given domain (picked by the initial seed questions), to infer new questions. With this the authors tried to fix a drawback of *Virtual Pet*, that through time, the questions and answers become saturated, and the number of new questions and answers falls exponentially through time, with respect to the number of already collected knowledge peaces. This approach is also aligned with the CC approach, which uses existing+ context and new knowledge, to drive the questions. First part of the *GOKC* paper describes analysis of the knowledge collected by *Virutal Pet* game. The second part is a description and evaluation of GOKC KA approach, where authors did 1 week experiment to show that the approach works. During that week the system inferred created 755 new questions, out of which, 12 were reported as bad. Out of these questions 10,572 answers were collected where 9,734 were voted as good. This results in the 92,07% precision. Compared to the game without question expansion (*Virtual Pet*), which has precision of 80.58%, this is an improvement.

3.2.2.6 Collabio (Collbaorative Biography)

. This is also a Facebook based game, with the intention to collect user's tags. While the gathered knowledge is more a set of person's tags than knowledge, it served as an inspiration to *Rapport* and *Virtual Pet*. During the experiment, *Collabio* users tagged 3,800 per-

sons with accurate tags with information that cannot be found otherwise(**Bernstein2009**; **Bernstein2010**).

3.2.3 Interactive Natural Language Conversation

Natural Language Knowledge Acquisition methods are special case of Interaction Acquisition systems. While almost all of the approaches already described above (under Interactive User Interfaces and Games subsections) use natural language to some extent, the language processing used is based on relatively small amount of textual patterns, or statements which are not necessary connected into a conversation. Common denominator of these systems is that they intentionally try to acquire knowledge and then use natural language statements to do this. As a side effect and as motivation for users, sometimes consequent questions and answers give a feeling of conversation. On the other side chat-bots, start with the intention to maintain an interesting conversation with the users, and have to do knowledge acquisition only to remember facts and parts of the past conversations to be able to be smart enough, so users do not lose interest. Starting with Eliza(**Weizenbaum1966**), these systems evolved, mostly directed by Turning Tests(**Turing?**), implemented as Loebner competitions, trying to pass it[3]. Through the measure of these tests(**Bradesko2012**), among a few propriatery chat-bots, two technologies evolved (*AIML, ChatScript*) to be general enough and can be used for conversational engine (chat-bot) construction and also NL knowledge acquisition.

3.2.4 AIML(Artificial Intelligence Mark-up Language

is an XML based scripting language. It allows developers of chat-bots, to construct a pre- defined natural language patterns and their responses. These definitions are then fed into an AIML engine, which can match user inputs with the patterns and figure out what response to write. AIMLs syntax consists mostly of input rules (categories) with appropriate output. The pattern must cover the entire input and is case insensitive. It is possible to use a wildcard (*) which binds to one or more words. The simplest example of AIML pattern with appropriate response is presented in Table 3.2. This pattern detects user's questions like "Do you have something on the menu?" and responds with "We have everything on the menu."

Table 3.2: AIML Example

```
<Category>
   <pattern> Do you have * on the menu </pattern>
   <template>
      We have everything on the menu.
   </template>
</Category>
```

AIML allows recursive calls to its own patterns, which allows for some really complicated and powerful patterns, covering many examples of input. Regarding the knowledge acquisition, AIML has an option to store parts of the textual patterns as variables and thus store information for later.

The AIML example on Table 3.3 can remember keywords following "I just ate" pattern, like "I just ate pizza". If user at some point later says "I am hungry", the bot is able to

[3]http://www.loebner.net

Table 3.3: AIML example of saving info to variables

```
<category>
    <pattern>I just ate *</pattern>
    <template>
        Nice choice! <set name = "food"> <star/></set>
    </template>
</category>
<category>
    <pattern>I am hungry</pattern>
    <template>
        Eat another <get name = "food"/>?
    </template>
</category>
```

respond with "Eat another pizza". In a combination with "<that>" tag, which matches previous computer's response, AIML can be used to construct specific knowledge acquisition questions (Table 3.4). The given example is using AIML 1.0, which was later improved with AIML2.0(**Wallace2013**) which introduced the *<Learn>* tag, but mechanism stayed mostly the same.

Table 3.4: AIML Example of remembering answers on specific questions

```
<category>
    <pattern>*</pattern>
    <that>What did you order</that>
    <template>
        Was it good? <set name = "menuItem"><star/></set>
    </template>
</category>
```

While AIML language with appropriate engine can remember specific facts, the mechanism is purely keyword based and cannot really count as structured knowledge. Additionally, since it only remembers direct facts, it would be really hard to construct an acquisition of all types of food for example. AIML based chatbots were winning Loebner's competitions in the years from 2000 to 2004, but were later outcompeted by Chatscript based bots and propriatery solutions.

3.2.5 ChatScript

is an NLP expert system consisting of textual patterns rules. It was designed by Bruce Wilcox (**Wilcox2011**) and besides patterns it has mechanisms for defining concepts, triple store for facts, own inference engine POS tagger and parser. From the measure of how close the system is to pass the Turing Test as measured by Loebner's competitions, *ChatScript* surpassed *AIML*, and is its successor, since both systems are open sourced. It was designed purposely to be simpler to use and have more powerful tools for NLP and knowledge acquisition which is integral part chatbot systems. A simple example from AIML (Table 3.2) can be re-written in much shorter form as ChatScript rule (Table 3.5).

Similarly the example from Table 3.3 can be writen as:

Table 3.5: Simple ChatScript example

```
?: (do you have * on the menu) We have everything on the menu.
```

Table 3.6: Simple ka (remembering) example

```
s: (I just ate _*) Nice choice!
s: (I am hungry) East another _0?
```

Similarly, example from Table 3.4 in ChatScript looks like:

Table 3.7: Simple ChatScript question/answer/remember example

```
t: What did you order?
        a: (_*) \$menuItem=_0 Was it good?
```

While the above examples repeats the functionality of AIML, ChatScript is more powerful and can remember facts in the shape of (subject verb object) and act on them. This is done with using *ĉreatefact* and *f̂indfact* functions.

3.2.6 CyN

CyN is an AIML interpreter implementation with additional functionality to be able to access Cyc inference engine and KB for both, storing the knowledge and also for querying(**Coursey2004**). This was done by introduction of new AIML tags:

- <*cycterm*> Translates an English word/phrase into a Cyc symbol.
- <*cycsystem*> Executes a CycL statement and returns the result.
- <*cycrandom*> Executes a CycL query and returns one response at random.
- <*cycassert*> Asserts a CycL statement.
- <*cycretract*> Retracts a CycL statement.
- <*cyccondition*> Controls the flow execution in a category template.
- <*guard*> Processes a template only if the CycL expression is true

3.3 Mining Acquisition

This category of KA systems try to make use of big text corpus-es available online or otherwise on some digital media. Because the core idea of writing is to share information, there is a lot of knowledge in the texts that can be extracted and converted into a structured knowledge that can later be used by computers. Due to wast size and availability of the data on the internet, mining is most often done on the web resources. Since most of the data format from these corpuses is text, these techniques are particularly strong in the using various NLP techniques, which are often combined with existing knowledge to correct mistakes and check consistency.

3.3.1 Populating Cyc from the Web (PCW)

Since whole idea of Cyc system is to gain enough knowledge through manual work, to be able to learn on itself after some point, the Cyc team is looking int into other means of knowledge acquisition which can automatize or speed-up the KA process. One of the approaches is by mining facts from the Internet by issuing appropriate search engine queries(**Matuszek2004**).

Because Cyc KB is really big, the first step of this approach is to select appropriate part of the kb (concepts and related queries) which are in the interest of the system. For initial experiments a set of 134 binary predicates was selected. These predicates were then used to scan the KB and find the missing knowledge, which was converted to CycL queries. These queries were then converted to NL and queried on a web search engine. The results are then converted back to CycL through NL to logic engine of Cyc. After the conversion these are converted back to NL and re-searched on the web, to check whether the results still hold, and then as the last step, the results are checked for consistency (whether they can be asserted into Cyc). From the initial 134 predicates, the system generated 348 queries, 4290 searches. It found 1016 facts, out of which 4 were rejectec due to inconsistency, 566 rejected by search engine (not same results), 384 were already known to Cyc, and finally 61 new consistent facts were detected as valid. After human review, the findings were that only 32 facts were actually correct.

3.3.2 Learning Reader

Learning Reader(**Forbus2007**) is a prototype knowledge mining system that combines NLP, large KB (CycKB) and analogical inference into an automated knowledge extraction system that works on simplified language texts. The system uses Direct Memory Access Parsing (DMAP(**Martin1986**)) to parse text and convert it into CycL concepts which are then checked by the inference engine whether they can form correct CycL statements. The prototype consisted of 30,000 NDAP patterns (a quick approximation would be to imagine CyN patterns- chapter 3.2.6). After parsing and syntax checks, found CycL sentences were checked by the inference within CycKB, whether knowledge is new or not. Only new logical statements were then asserted. Additional feature of this prototype system is that it includes question answering mechanism, which can be used by evaluators to check what the system learned. This same mechanism is also used to try to generate new facts (elaborate) and also questions based on newly acquired knowledge. The experiment on 62 written stories improved the recall from 10% to 37% and kept the accuracy as 99.7% compared to original 100%. By using additional inference and conjecture based inference, the recall raised to 60% while accuracy dropped to 90.8%.

3.3.3 Never Ending Language Learner (NELL)

NELL(**Mitchell2015**) is a text mining KA system running 24/7 with the goal to extract knowledge, use this knowledge to improve itself and extract more knowledge. NELL was started in January 2010 and as of 2015 acuired 80 million confidence weighted new beliefs. NELL consists of many different learning tasks (for different types of knowledge), where each task also consist of the performance metrics, so the system can asses itself and check if the learning task itself is also improving through time. In 2015 it consisted of 2500 learning tasks. Some of learning task examples:

- Category Classification
- Relation Classification

- Entity Resolution
- Inference Rules among belief triples

After learning tasks, there is a *Coupling Constraings* component, which combines results of learning tasks. The potentially useful knowledge gets asserted into the KB as candidates, where the assertions are checked by knowledge integrator module which integrates the assertions into the KB, or rejects them.

After some initail initial KB had been gathered, the CMU text-mining knowledge NELL also started to apply a crowdsourcing approach(**Pedro2012a**), using natural language questions to validate its KB. In a similar fashion as Curious Cat, NELL can use newly acquired knowledge, to formulate new representations and learning tasks. There is, however, a distinct difference between the approaches of NELL and Curious Cat. NELL uses information extraction to populate its KB from the web, then sends the acquired knowledge to Yahoo Answers, or some other Q/A site, where the knowledge can be confirmed or rejected. By contrast, Curious Cat formulates its questions directly to users (and these questions can have many forms, not just facts to validate), and only then sends the new knowledge to other users for validation. Additionally, Curious Cat is able to use context to target specific users who have a very high chance of being able to answer a question.

3.3.4 KnowItAll

KnowItAll(**Etzioni2004**) is a domain independent web fact extraction system that s specific search engine queries to find new instances of specific classes. It starts it extraction with a small seed of class names and NL patterns like "NP1 such as NP2". The classes and patterns are then used to find instances, new classes and also new extraction phrases by analyzing the results of the web search engines. For example, Googling: "Cities such as *", will return a lot of statements with instances of cities. After these cities are extracted, the names can be used for furhter Googling and by analyzing the phrases in which these cities appear, new patterns can be found, and so on. As part of the experiment, KnowItAll ran for 5 days and extracted over 50,000 instances of cities, states, countries, actors and films. To asses the correctnes of the extractions, the system can fill-in the instances into the various patterns and check the hit-count returned by the search engine./ This then compares by the hit-count of the instance itself and uses this to asses the probability of the instance really belonging to the detected class. For example: comparing the hit-count of "Ljubljana", "Cities such as Ljubljana" and "Planets such us Ljubljana", the system can figure out that Ljubljana is most likely indeed an instance of the class city.

KnowItAll was the first of the systems that inspired an *Open Information Extraction (Open IE)* paradigm(**Etzioni2011**) which resulted in many other IE systems such us TextRunner, ReVerb and R2A2. The main ide of this paradigm is to avoid hand labeled examples and domain specific verbs and nouns when approaching textual patterns which can lead to open (without specifying the targets) knowledge extraction on a web scale.

3.3.5 Probase

Probase is a probabilistic taxonomy of concepts and instances consisting of 2.7 million of concepts extracted from 1.68 billion of web pages (**Wu2012**). The main difference between Probase and other KBs is that Probase is probabilistic as opposed of "black and white" KB. On the other hand, even if it has much more concepts, it is sparse in the knowledge, since it only uses isA relation (taxonomy). Probase was compared to other taxonomies such as WordNet, YAGO and Freebase in the sense of recall and precition of isA relations.

Probase was found to be most comprehensive (biggest recall), while losing at precision measure against YAGO (92.8% vs 95%).

Probase was later renamed as *Microsoft Concept Graph*, and has accessible API[4] which in 2017 consists of 5,401,933 concepts, 12,551,613 instances and 87,603,947 isA relations.

3.3.6 TextRunner

TextRunner is a successor of KnowItAll system (**Soderland2007**) and is the first to introduce Open Information Extraction (OIE) paradigm, which's main idea is that it is open-ended and can extract information autonomously without any human intervention which would fix the system to some specific domain or set of concepts/relations. *TextRunner* was ran through over 9 million of web pages, and compared to *KnowItAll* reduced the error rate for 33% on comparable set of extractions. Throughout the experiments, *TextRunner* collected 11mio of high probability tuples and 1 mio concrete facts. *TextRunner* consists of three components.

Self-Supervised Learner is a component started first which takes a small corpus of documents as an input and then outputs a classifier that can detect candidate extractions and classify them as trustworthy or not.

Single-Pass Extractor is a component that makes a single pass through the full corpus and extracts tuples for all possible relations. These tuples are then sent to the classifier trained before, which then marks them as trustworthy or not. Only trustworthy tuples are retained.

Redundancy-Based Assesor is the last step which assigns a probability to each retained tuple, based on the probabilistic model or redundancy (**Downey2005**).

3.3.7 ReVerb

With the experiments done with *TextRunner* and *WAE*, it became obvious that OIE systems have a lot of noise and inconistencies in the results. For this reason two syntactical and lexical constraints were introduced in *ReVerb* OIE system(**Fader2011**). This helps with removing the incoherent extractions such as "recalled began" which was extracted from sentence "They recalled that Nungesser began his career as precint leader", or uninformative extractions like "Faust, made a deal" extracted from "Faust made a deal with the devil".(**Fader2011**). When started, *ReVerb* first identifies relation phrases that match the constraints, then if finds approriate pair of appropriate noun phrase rguments for each identified phrase. The resulting extractions are then given a confidence score using logistic regression classifier.

3.3.8 R2A2

R2A2 is another improvement in OIE paradigm, since previous systems assumed that relation arguments are only simple noun phrases. Analysis of *ReVerb* errors showed that 65% of errors is on the arguments side (the relation was ok). To fix this, *R2A2* system goes somehow into the direction of kb based KA sytems like Curious Cat with argument constraings.(**Etzioni2011**). The difference is that *R2A2* is not using hard logic and inference, but rather statistical classifier to detects class constraints (bounds) of the arguments. Compared to *ReVerb*, *R2A2* has much higher precision and recall.

[4]https://concept.research.microsoft.com

3.3.9 ConceptMiner

ConceptMiner is a KA system built by Ian Scott Eslick as part of his master thesis(**Eslick2006**), with the main hipothesis that the seed knowledge collected from vounteers can be then used to bootstrap automatic knowledge acquisition. *ConceptMiner* specifically focuses on binary semantic relationships such as cause, effect, intent and time. The system relies on the prior volunteer knowledge from *ConceptNet* and tests its hipothesis with experimental extractions of knowledge aroun three semantic relations: desire, effect and capability.

As a first step, the systemn uses knowledge around predicates *DesireOf*, *EffectOf* and *CapableOf* from *ConceptNet*, to construct web-search queries. The results of these are then used to derive general patterns for aforementioned relations. For example an existing knowledge *(DesireOf "dog" "attention")*, when converted to search engine query: "dog * bark", results in patterns like:

- "My/PRP$ dog/NN loves/VBZ attention/NN ./."
- "Horseback/NN riding /VBG dog /NN attracts/VBZ attention/NN."

While not all of the patterns are of the same quality, with the sheer number of repetitions, it is possible to extract more probable ones. This then results in general patterns such as *<X>/NN loves/VBZ <Y>/NN*. These can be then used to issue a lot of search queries with various combinations of wors, to extract instances of 'who desires what'. These potential instances then go into the last step (filtering). As part of this step, *ConceptMiner* removes badly formed statements, concepts not included in *ConceptNet* KB, and concepts with low PMI score (see abbreviations and glossary).

3.3.10 DBPedia

DBPedia is crowd-sourced RDF KB, extracted from Wikipedia pages and made publicly available(**Lehmann2015**). As of 2017 the English DBPedia contains 4.58 million knowledge pieces, out of which 4.22 million in a consistent ontology, including 1,445,000 persons, 735,000 places, 411,000 creative works (123,000 music albums, 87,000 films and 19,000 video games), 241,000 organizations (58,000 companies, 49,000 educational institutions), 251,000 species and 6,000 diseases[5]. *DBPedia* is also localized into 125 languages, so all-together it consists of 38.3 million knowledge pieces. It is also linked to YAGO categories.

Acquisition mechanism is automatic and consists of the following steps:

- Wikipedia pages are downloaded from dumps or through API and parsed into an Abstract Syntax Tree (AST)
- AST is forwarded to various extractor modules. For example, extractor module can find labels, coordinates, etc. Each of the extractor modules can convert it's part of AST into RDF triples.
- The collection of RDF statements as returned from the extractors is written into an RDF sink, supporting various format such as NTriples, etc.

3.3.11 YAGO (Yet Another Great Ontology)

YAGO is an ontology built automatically from *WordNet* and *Wikipedia*(**Suchanek2008**). Latest version *YAGO3* is built from multiple languages and as of 2015 consist of 4,595,906 entities, 8,936,324 facts, 15,611,709 taxonimy facts and 1,398,837 labels (**Mahdisoltani2015**).

[5] http://wiki.dbpedia.org/about

The facts were extracted from Wikipedia category system and info boxes, using a combination of rule-based and heuristic methods, and then enriched with hierarchy (taxonomic) relations taken from WordNet. Since building YAGO is automatized, each next run of the script can use existing knowledge for type and consistency checking. This kind of type checking helps YAGO to maintain its precision at 95%(**Suchanek2008**).

3.3.12 KNEXT

With the premise that there is a lot of general knowledge available in texts, which lays beneath explicit assertional content, Schubert build *KNEXT* KA system(**Schubert2002**) which extracts *general "possibilistic" propositions* from text. The main difference towards other KA mining systems is that before combining meanings from a phrase, the meanings are abstracted (generalized) and simplified. For example, abstraction of "a long, dak corridor" yields "a corridor". Or "a small office at the end of a long dark corridor" yields "an office". This kind of abstraction, together with weakening of relations into a possibilistic form, starts to represent presumptios about the world. The extraction follows five steps:

- Pre-process and POS tag the input
- Apply a set of ordered patterns to the POS tree recursively
- For each successfully matched subtree, abstract the interpretations using semantic rule patterns
- Collect the phrases expected to hold general "possibilistic" propositions
- Formulate the propositiions and output these together with simple English representations.

From an example input statement "Blanche knew something must be causing Stanley's new, strange behavior but she never once connected it with Kitti Walker.", the output looks like this:

```
A female-individual may know a preposition
(:Q DET FEMALE-INDIVIDUAL) KNOW[V] (:Q DET PROPOS)
something may cause a behavior
(:F K SOMETHING[N]) CAUSE[V] (:Q THE BEHAVIOR[N])
a male-individual may have a behavior
(:Q DET MALE-INDIVIDUAL) HAVE[V] (:Q DET BEHAVIOR[N])
a behavior can be new
(:Q DET BEHAVIOR[N]) NEW[A])
a behavior can be strange
(:Q DET BEHAVIOR[N]) STRANGE[A])
a female-individual may connect a thing-reffered-to with a female
   -individual
(:Q DET FEMALE-INDIVIDUAL) CONNECT[V]
(:Q DET THING-REFERRED-TO)
(:P WITH[P] (:Q DET FEMALE-INDIVIDUAL)))
```

The authors position their system as an addition to systems like *Cyc*, and conducted their KA experiments on Treebank corpora resulting in around 60% of propositions marked as "reasonable general claims"(**Schubert2003**).

3.4 Reasoning Acquisition

Compared to other types of KA, *Reasoning Acquisition* in its essence, doesn't need any external data-sources, but it uses existing knowledge and machine inference to automatically infer additional facts from the existing knowledge. While deductive reasoning can come with new facts from the premises and rules, the reasoning that has a chance to produce higher value (non obvious) findings is inductive and analogical resasoning. Analogical reasoning can find new facts of some concept based on properties of similar concepts. On the other side, infuctive reasoning can find find new probable rules based on current observations of the KB.

3.4.1 Cyc Predicate Populator + FOIL

With the initial experiments conducted from Labour Rule Acquisition done as part of *Factivore* and *Predicate Populator*(**Witbrock2005**), it was found that getting inference rules from crowdsourcing and untrained human labour is ineffective and slow. For this reason, inductive logic inference mechanism based on FOIL (First Order Inductive Learner) approach (**Quinlan1995**) was added to the systen. Experiments were conducted on a set of 10 predicates from the KB, which generated 300 new rules. Of these rules, 7.5% were found to be correct and 35% correct with minor editing to make them well formed (assertible to Cyc). This way, rule acquisition was speed up for quite a lot, since previous experiments showed that human experts produce rules with the rate around three per hour, while with FOIL, they can review and double check for correctnes around twenty rules per hour.

3.4.2 Plausible Inference Patterns (PIP)

The main idea of *PIP* system is to learn new plausible inference rules (patterns of plausible reasoning) by combining existing knowledge and reinforcement learning and thus improve the system's question answering abilities(**Sharma2010**). It is based on *Cyc* knowledge base, especially its predicate hierarchy represented by *#$genlPred* predicate, and *PredicateType* which represents second order collection of predicates that can be grouped together by some common features. For example *(#$genlPreds #$holds #$touches)*, means that each time something is holding something else, it is also touching it. The system scans the KB for rules containing predicates and tries to generate *PIPs* out of them, meaning that it tries to replace the predicates in the rules with the appropriate *PredicateType* which is linked to prior predicate. If there are more than 5 ways to generate the same PIP (from various predicate instances), then this new rule is accepted as "valid". Example of such PIP is:

$$(\#\$familyRelationsSlot(?x, ?y) \land \#\$familyRelationSlot(?y, ?z)) \Longrightarrow \#\$personalAssociationPredicate(?y, ?z)$$

where *#$familyRelationsSlot* and *#$personalAssociationPredicate* are instances of *#$PredicateType* and thus collections of other predicates, and *?x,?y,?z* are variables representing other concepts (**sharma2010**). This PIP or inference rule, tells the special inference algorithm that two predicates of type *#$familyRelationsSlot* cand imply *#$personalAssociationPredicate*, whent a proper combination of antecedent bindings on *?x,?y, ?z* is can be proved. Part of the *PIP* system is also an inference algorithm (FPEQ), which can make use of the above rule and come up with the possible solutions. As the first step, it constructs a graph connected to the concepts used in the Q/A query. Then, the algorithm searches

for the assertions in the graph matching a set of existing PIP inference rules. As the last steps, it returns the maches (i.e. filled-in PIPs with specific predicates and concepts) which can answer the query. As an additional step, authors introduuced reinforcement learning, where a small amount of user feedback (+1 or -1 voting) for final answers, can improve the PIP selection and also the level of generalizations when generating PIPs. Overall, the experiments showed that the approach increased Q/A recall for quite a lot (120% improvement), while minimally reduced the precision (94% of the baseline)(**Zang2013**).

3.4.3 AnalogySpace

This system is meant to work over big, but unexact knowledge bases (such as OMCS), where the need it to be able to make rough conclusions based on similarities, as opposed on hard and absolute logical facts. *AnalogySpace* does just that, by performing analogical closure by dimensionality reduction of semantic network (KB graph)(**Speer2008**). This system tried to represent a new synthesis between a standard symbolic reasoning and statistical methods. For this, a similar technique to Latent Semantic Analysis (LSA) is being used, where strong assertions are used as opposed to weak semantics of word co-occurences in the document. In the LSA matrix, on one axis are concepts from *OMCS*, and on the other axis a features of these concepts, which yelds a sparse matrix of very high dimension. Then Singular Value Decomposition (SVD) is used on the matrix to reduce the dimensionality. This results in *principal components*, which represent the most important aspects of the knowledge. Then semantic similarity can be determined using linear operations over the resulting vectors. In a sense, this dimensionality reduction is acting as a generalization process for the kb. This way it is easier to calculate similarities between resulting concept vectors and can thus make generalizations based on these similarities, even if original concept didn't have some of the exact assertions that would enable inference engine to use it in the inference process and thus come with good answer. Results of experminents have shown that more than 70% of the resulting assertions were marked as true by human validators.

3.4.4 Cyc Wiki

Given that Wikipedia contains a lot of knowledge which is not structured in a way to be useful for inference engines directly, it makes sense to either try to structure it (as was done with *YAGO*), or link it with some existing ontology such as *CycKB*, which was done as part of this research task (**Medelyan2008**). Authors first filtered out of pootential pool of Cyc concepts all non-common sense concepts, which ended with 83,897 of them. Then these concepts were string matched with Wikipedia concept names based on their name directly, or based on *#$nameString* predicate. For example, Cyc concept *#$Virgo-Constellation* would be mathced to Wikipeadia page *Virgo (Constellation)*. After this step, the mappings that have 1 to 1 relationship with Wikipedia pages and do not point to Wiki disambiguation pages, are considered properly aligned. But the mappings that have more than 1 Wikipedia result then go into the disambiguation phase. With this approach, authors were able to get the precision of 96.2% and recall 64.0% when no disambiguation was needed, and precision 93% and recall 86.3% with the disambiguation part.

3.5 Acquistion of Geospatial Context

This sub section covers the works and approaches that can relate to our context mining/acquisition implementation. While the approaches themselves are more from the data-mining domain, as opposed to knowledge acquisition, in *Curious Cat*, the results are converted

in the knowledge and asserted directly as a contextual knowledge about the users. For this reason, this related work is in its own subsection. In the Table 3.1, the approaches here are not listed, since they don't count as a knowledge acquisition, but more a custom data-mining solutions which support our proposed KA approach.

3.5.1 Extracting Places from Traces of Locations

This algorithm (we refer to it as SPD1 - Staypoint Detection), is one of the first papers/approaches that started to mine the raw GPS data into more user friendly notion of location - place.(**Kang2005**) The main idea of the algorithm is to be able to cluster the raw GPS locations into the particular places (home, work, specific restaurant, etc.) that user visited. This is achieved with a combination of radius(D_t) and time-based (T_t) clustering tresholds, where a cluster is a set of GPS coordinates that are within radius ($dist(loc_1) - dist(loc_n) < D_t$) during a time which is longer than a threshold ($time(loc_n) - time(loc_1) > T_t$). With this approach, it is possible to cluster locations based on how long one stays within a region, wihtout knowing the number of clusters in front (as is necessary with more standard clustering approaches). The core of the algorithm has a benefit to be really simple and can be easily ran on the phone.

The algorithm pseudocode is shown below (algorithm 3.1), where CL represents currently detecting cluster (staypoint), VP is detected cluster (staypoint or Visit Point) and *plocs* is temporal locations to be discarded or added to cluster later. Regarding the thresholds, T_t is Time threshold, T_d is distance treshold.

Algorithm 3.1: Staypoint Detection Algorithm 1 (SPD1)

```
Data: raw GPS coordinates
Result: cluster representing one staypoint
if distance(CL, loc) < D_t then
    add loc to CL;
    clear plocs;
else
    if plocs.length > 1 then
        if duration(cl) > T_t then
            add CL to VP;
        end
        clear CL;
        add plocs.end to CL;
        clear plocs;
        if distance(CL,loc) < D_t then
            add loc to CL;
            clear plocs;
        else
            add loc to plocs;
        end
    else
        add loc to plocs;
    end
end
```

While the above alogorithm robustly detects stay-points, it completely ignores paths

between. Additional weak-point is that it doesn't hadndle well when more than one GPS coordinate wrongly jumps due to GPS accuracy. This was fixed by the improved algorithm we developed as part of *Curious Cat* system, and simultanously, the GPS accuracy part as well by the algorithm described below (subsection 3.5.2).

3.5.2 Discovery of Personal Semantic Places based on Trajectory Data Mining

This work(**Lv2016**)(SPD2), builds on top of the first *SPD* algoritm described in subsection 3.5.1. It improves the stay-point detection and incorporates it into the broader system which is able to map particular visits into the exact places (points of interest - POIs) and then additionally able to predict next locations of tracked users. With this functionality (especially mapping to exact points of interest), this approach shares a lot of similarities with the context mining approach that is employed within the *Curious Cat* system. The main improvement of this algorithm, is that it takes into the account the discontinuous characteristics of phone GPS signal (it gets lost inside buildings, its accuracy drops under the trees). This is done by introducing a second T_{dt} threshold, which is must be higher than T_d, and sets a tolerated distance between two consequent stay-points to be merged on the fly. Additionally, these thresholds are not fixed, but dynamically calculated based on the distribution of distances between raw GPS points.

The simplified algorithm (removed unnecessary if/else statements) is shown below (algorithm 3.2), where CL represents currently detecting cluster (staypoint), PC is previous cluster and VP is detected cluster (staypoint or Visit Point). Regarding the thresholds, T_t is Time threshold, T_d is distance treshold and T_{dt} is tolerated distance threshold for later merging.

Algorithm 3.2: Staypoint Detection Algorithm 2 (SPD2)

Data: raw GPS coordinates
Result: cluster representing one staypoint

if *distance(CL, loc)* $< T_d$ **then**
  add loc to CL;
else if *duration(CL)*$>T_t$ **then**
  append CL to VP;
  CL = 0, PC = 0;
else if *interval(CL, PC)* $> T_t$ **and** *distance(CL,PC)*$< T_dt$ **then**
  CL = combine (CL, PC);
  append CL to VP;
  PC = 0;
else
  PC=CL;
  CL=0;
end

Similarly as *SPD1*, also *SPD2* has a prblem of ignoring the GPS coordinates that are part of paths (or moves) between stay-points.

3.5.3 Applying Commonsense Reasoning to Place Identification

Clustering raw sensor data into locations, paths, activities and travel modes is often not enough. Besides knowing the location, time of arrival and duration of stay, contextual

knowledge benefits from the information about the type of place and name, or even the exact ID from some database, where additional information can be looked up. This is part of the *CuriousCat* system, which was inspired by and is based on the related work of Marco Mamei(**Mamei2010**). In this work, the author shows how *Cyc* knowledge base can be used to improve automatic place identification. The approach is using probabilistic ranking the list of candidates (retrieved from *Foursquare* in consideration of the common sense likelihood, which is based on the user profile, features of the day (exact time, noon, afternoon, morning, midday,...), previous visits and type of probable place. The approach was validated and given that the GPS accuracy of N95 phones (used in the experiment) is worse than that of phones in 2017, it achieved the accuracy of 75% for business type of places. ——————————————————————————————-

Chapter 4

Knowledge Acquisition Approach

This chapter defines the terms, formal structure and steps that form our proposed KA approach. First it introduces the general architecture and interaction loop that defines the sequence of interactions and steps involved in the process(section 4.1). In the second part, it formalizes the upper ontology and logical constructs required for the KA approach (section 4.2). After that, each of the crucial steps is described in more detail through examples and additions to the core logical structure defined earlier.

To make it easier to explain, formalize and understand the proposed approach for automated knowledge acquisition with prior knowledge, user context and conversational crowdsourcing, we will guide the explanations and formalizations through an example KA conversation depicted in tables 4.1 and 4.2.

Table 4.1: Minimal example of Curious Cat and user interaction.

Step num.	Interaction	
	Curious Cat	User1
1	Where are we? Are we at Joe's Pizza restaurant?	Yes.
2	We've never been here before. What kind of place this is?	Restaurant.
3	Does Joe's Pizza have Wi-Fi?	Yes.
4	Is it fast enough to make Skype calls?	I don't know.
5	What's on the menu in Joe's Pizza?	Pizzas.
...Some time passes while the user eats...		
6	What did you order?	A car.
7	I've never heard of food called 'car' before. Are you sure it's a type of food?	No.
8	What did you order then?	Pizza Deluxe
9	I've never heard of 'Pizza Deluxe' before. What kind of thing is it?	Pizza.

When the system is convesing with the user, it uses all the knowledge gathered in prior conversations, other user's conversations and also this conversation, and is able to use it to further generate new comments and related questions. This is evident in the interaction step number 4 in Table 4.1. At the same time, or later, when some other user is in a similar context, the knowledge can be double checked with another user, as shown in Table 4.2. Based on the votes (confirmations or rejections) from the crowd (other users), the system can decide whether to believe the new knowledge in general, or only when it interacts with

this particular user. This is explained in more detail in section ref.

Table 4.2: Minimal example of Truth checking interaction with the help of crowdsourcing.

Step num.	Interaction	
	Curious Cat	User2
10	Where are we? Are we at Joe's Pizza restaurant?	Yes.
11	We've never been here before. What kind of place this is?	Restaurant.

The resulting implementation (described at the end in chapter 5) of the described approach, named Curious Cat has a multi objective goal, KA is the primary goal, while having an intelligent assistant and a conversational agent are secondary goals. The aim is to perform knowledge acquisition effortlessly and accurately as a side effect, while having a conversation about concepts which have some connection to the user. At the same time, the approach allows the system (or the user) to follow the links in the conversation to other connected topics, covering and collecting more knowledge. For illustration see the example conversation sketch in]hlTable I, where the topic changes from a specific restaurant to a type of dish.

4.1 Architecture

The proposed KA system consists of multiple interconnected technologies and functionalities which we grouped into logical modules according to the problems they are solving (as also defined in chapter 2). This was done in order to minimize the complexity, improve the maintenance costs and allowing switching the implementations of separate sub-modules. Additionally such logical grouping increases the explainability and general understanding of the system.

On Figure 4.1 these modules are represented with the boxes, and their functionality groups are presented with the colors (see the figure legend). Arrows represent the interaction and workflow order and initiation (the interaction is initiated/triggered from the origin of the arrow).

We can see that the central core of the system is the knowledge base (modules marked in purple and letter A). The knowledge base consists of *Upper Ontology* gluing everything together, *Common Sense Knowledge* to be able to "understand" user's world and check the answers for consistency, *Meta Knowledge* for enabling inference about its internal structures, *User Context KB* to hold current user context and *Knowledge Acquisition Rules* to drive the KA process from within the KB, using logical inference.

Next to the KB, is an *Inference Engine* that performs inference over the knowledge from the KB. Its modules are represented with the red color and letter B. The inference engine needs to be general enough to be able to perform over full KB, and should be capable of meta-reasoning (over the meta-knowledge and KA knowledge in the KB) about the KB's internal knowledge structures. In cases when the inference engine have some missing functionalities, some of these tasks can be supplemented by the *Procdeural Support* module. In the proposed system, inference engine handles almost all of the core KA operations, which can be separated into the following modules:

- *Consistency Checking* module which can asses the user's answers and check whether they fit within the current KB knowledge.

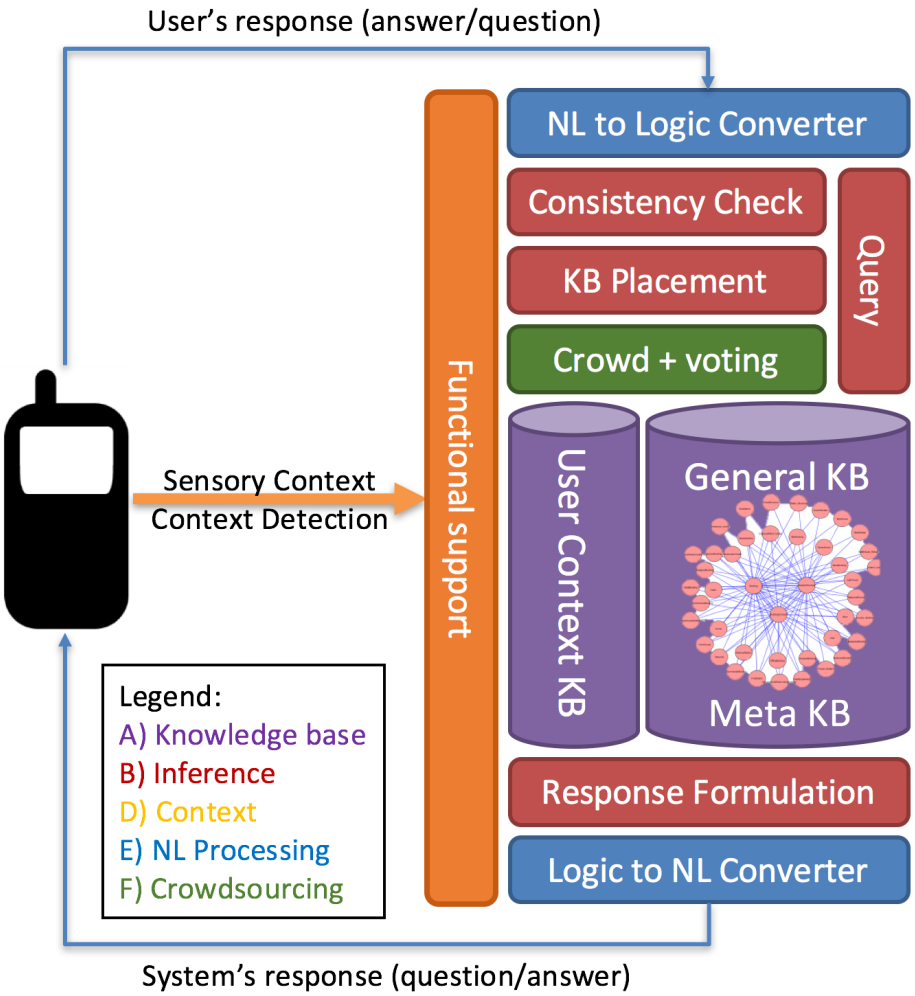

Figure 4.1: General Architecture of the KA system, with an interaction loop presented as arrows.

- *KB Placement* module which decides where into the subtree of the KB the answer should be placed.
- *Querying* module, which employs the inference engine to answer questions that are coming from the user through NL to Logic converter.
- *Response Formulation* module, which employs the KA meta-knowledge and do inference about what to say/ask next. Results of this module are then forwarded to the Logic to NL converter and then to the user.

Tightly integrated with the knowledge base and inference engine is a *Crowdsourcing Module*, which monitors crowd (multipl user's) answers and is able to remove (or move to different contexts) the knowledge from the KB, based on its consistency among multiple users. If some piece of knowledge inside the KB is questionable, the module marks it as such and then *Response Formulation* module checks with other users whether it's true or not and should maybe be removed or only kept in the one user's part of the KB. This module is represented in Green color and letter F.

At the entry and exit point of the system workflow, there are NLP processing modules which can convert logic into the natural language and vice versa. These modules are used

for natural language communication with the users. Tese two modules are represented in Blue and letter E.

On the side of the Figure, there is a procedural module (depicted with Orange color and letter D), which is a normal software module (in our implementations written in procedural programming language), which glues everything together. It contains a web-server, authentication functionalities, machine learning capabilities, connections to external services and context mining and other functions that are hard to implement using just logic and inference. This module is taking care of the interactions between submodules.

All of the modules are triggered either through the contextual triggers (also internal, like when timer detects the specific hour or time of day), or by the users. When the context changes, it causes the system to use inference engine to figure out what to do. Usually, as a consequence it results in a multiple options like questions or comments. Then it picks one and sends a request to the user. This triggering is represented with the arrows, where the blue arrows represent natural language interaction, and the orange one represents structured or procedural interaction, when the procedural module classifies or detects any useful change in the sensor data sent into the system py the part running on the mobile phone.

4.1.1 Interaction Loop

As briefly already mentioned above, besides architecture, Figure 4.1 also indicates a system/user interaction loop represented by arrows. Orange arrow (pointing directly from the phone towards the system) represents the automatic interaction or triggers that the phone (client) is sending to the system all the time. This provides one part of the user context. After the precedural part analyses the data (as described in ref to SPD) and enter findings into the KB as context, this often triggers the system to come up with a new question, or context related info. Example of such a trigger is, when user changes a location and the system figures out the name and type of the new place. On the other side, Blue arrows represent the Natural Language interaction which can happen as a result of automatic context (Orange arrow), or some other reason causing new knowledge appearing in the KB. New knowledge can appear as a consequence of answering a question from the same user, or some other user. This shows, how the actual knowledge (even if entered automatically through procedural component) is controlling the interaction, and explains how the system is initialized and how its main pro-activity driver is implemented. Examples of such initialization of the interaction is presented in Script 1 and Script2. Additionally, the user can trigger a conversation at any point in time either by continuing the previous conversation or simply starting a new one.

According to the interactions described above, the proposed KA system have two options for the interaction. Human to machine (HMI), when users initiate interaction, and machine to human (MHI), when the system initiates the interaction. The specifics of both, which cannot fit in Figure 4.1 are explained in the following sub-sections 4.2.1 and 4.2.2. On top of this, the design of the system allows a novel type of interaction which combines multiple users and machine into one conversation, while presenting this to the users as a single conversation track with the machine. This becomes useful when the system doesn't have enough knowledge to be able to answer user's questions, but it has just enough to know which other users to ask (i.e. when someone is asking a question about specigic place and there is no answer in the KB, *Curious Cat* can ask other users that it knows had been there). This type of the interaction can be called Machine Mediated Human to Human interaction (MMHHI). This allows the system to answer questions also when it doesn't know them, while simultaneously also store and remember the answers, either parsing them and assert them directly to KB, or leave them in NL for later Knowledge

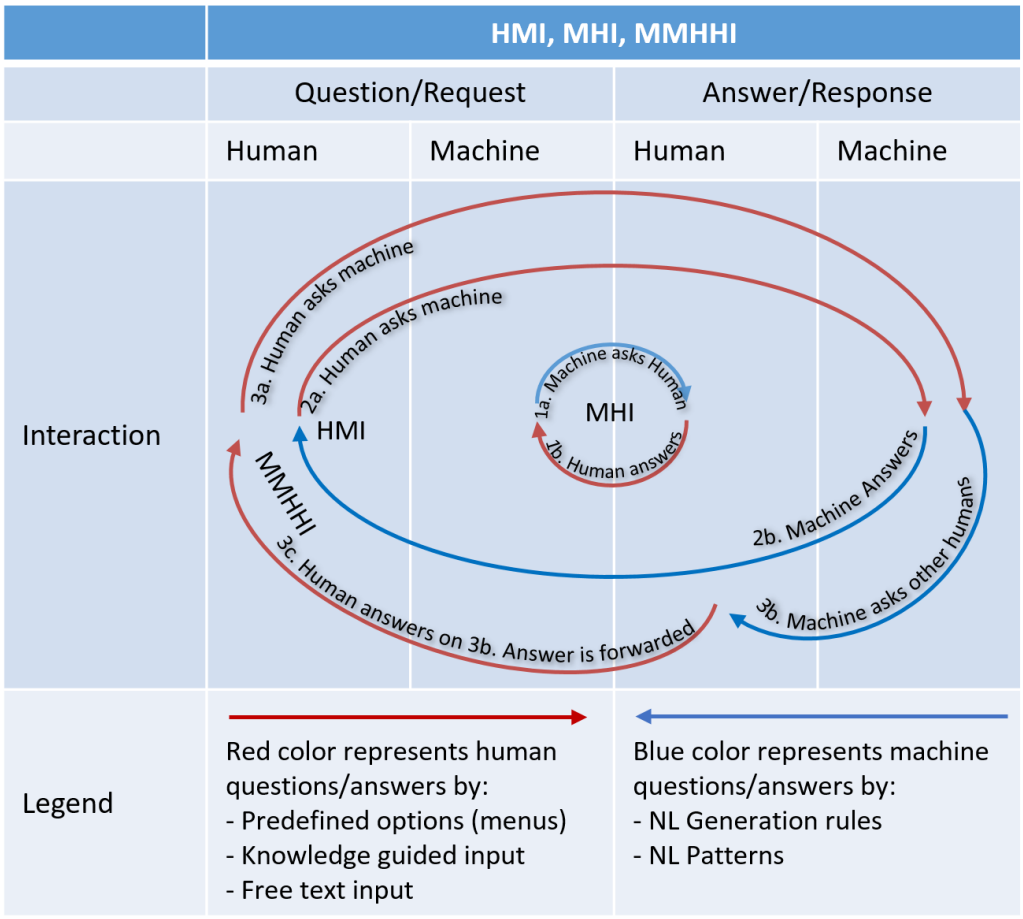

Figure 4.2: Possible interaction types between the user and Curious Cat KA System.

Mining analysis. The possible interaction types are also presented on Figure 4.2.

4.1.1.1 Machine to Human Interaction (MHI)

The most basic form of interaction between the CC system and the user, which we also use the most, is when something triggers a change in the KB and CC decides it's the time to ask or tell something. On the Figure 4.2, this is represented by the most inner loop (MHI). Example of this is when the context part of *Procedural* module classifies a new location and then asserts it into the KB. This then triggers Inference Engine which results in a new user query (same as defined in ref to CCWantsToAskLocation).

$$ccWantsToAsk(CCUser1, (userLocation(CCUser1, CCLoc1)))$$

This query then goes through logic to NL ref conversion, which is then presented to the user in NL like "Where are we? Are we at the restaurant X?". This is represented on Figure 4.2 with a blue arrow on the inner circle, marked with 1a. The presentation is handled by the client and can be in a written form, or through the text-to-speech interface. User can then answer this question and thus close the interaction loop (blue arrow marked with 1b), possibly causing a new one with her answer.

For easier answering, the KA system can use existing KB to generate a set of possible answers at the qustion generation time. These can be then picked by the user instead of writing. THe guidance can consis of variations of these:

- A fixed set of pre-defined options that user can pick from, generated from the KB.
- A set of pre-defined options with an additional free text field when the set of possible answers is big or inifite. In this cases the text field is connected to the KB providing auto-complete options for valid answers.
- Completely free text were user can write anything. This is essentially the same as for the HMI interaction described below (subsubsection 4.1.1.2).

The example of mediated answer guidance can be seen in Fig. 3, where the system presented a set of possible answers while still allowing a free text which will be autocompleted with the food types that the system knows about. If the user enters something new, the system will accept that (as was shown in the step 6 in Table I).

The inference triggering, language rules and mechanisms for context detection are described in more detail in sections ref, ref, ref respectively. This type of the interaction is where the users answer questions and is thus part of the main research topic of this thesis.

4.1.1.2 Human to Machine Interaction (HMI)

The second type of interaction is, when user initiates the conversation. If this is done at some point as the answer to an old question, the process is the same as described above in subsubsection 4.1.1.1. But users can also enter a free text, asking a question or stating something. In this case, this goes through the *NL to Logic Converter* module which tries to convert the text into logical query or assertion. The complexity of converting NL into logic is a lot higher than in the opposite direction, since the language is not as exact. In CC implementation ref to CC converter, we handled this to some extent by using SCG system (**Schneider2015**), where the text is matched to NL patterns which are linked to appropriate logical structures. This would be used for example, when user, instead of simple Pizza Deluxe (step 5 in Table I), would say They sell pizzas or something even more complex (see section 4.5.2). After the text is converted into logic, inference engine can use it to query it against the KB, and show the answers back to user, again converted into NL through *Logic to NL Converter* module. This type of interaction is depicted on Figure 4.2 by the middle arrow circle started by humans (arrow 2a), where machine provides the response (arrow 2b).

4.1.1.3 Machine Mediated Human to Human Interaction (MMHHI)

Both of the interactions described in sections 4.1.1.1 and 4.1.1.2 pre-supposes that the receipient of the query, knows how to answer it, or respond otherwise. In the cases when, let's say machine doesn't have any answer (The NL question gets converted into the logical query, which doesn't retrieve any aswers from within the KB). It could respond with "I don't know", which is a valid response. Thile this allows for the conversation to continue, it doesn't help the user to get the answer, also does not benefit to knowledge acquisition. The only thing the system can learn from this, is that user is interested in the object of the question. This doesn't have to end there though, since CC has access to other users and knowledge about their past and current contexts. Based on the topic of interes from the user query, the system can easily find users which might know the answer (inferred from their past whereabouts, answers, etc.). Once such an user is deteced, the original question can simply be forwarded to him, as it would be asked by CC itself. Once he, or one of the users answers, CC can forward it to the original user. On top of that, CC can parse the answer the same way as described above for HMI and MHI (sections 4.1.1.1, 4.1.1.2), and remember it, placing it into the KB. In the cases when the language of the answer is too

complex, it can be stored in its original format, for later text-mining apporach which can lead to learning of new-patterns as well as the knowledge hiding in the answers. On top of this, CC can also remember the question itself, and place it on specific type of concepts, as an important quetsion to ask. On Figure 4.2, MMHHI approach is depicted by the outer circle of arrows, where 3a is original user's question, which is forwarded to other users when CC doesn't know how to answer (3b). After one or more of the users answer, the answer is forwarded back to the original user (3c).

4.2 Knowledge Base

As visible in Figure 4.1, Knowledge Base is the central part of the proposed KA system. Internally KB has three components. The main part, which should in any real implementation of the system also be the biggest, is the common-sense knowledge, and its upper ontology over which we operate. This part of the system contributes the most to the ability to check the answers for consistency. The more knowledge already exists, the easier becomes to assess the answers, come up with new questions and also propose possible answers in the guided interaction(ref).

The second part is the user Context KB, which stores the contextual knowledge about the user. This covers the knowledge that the user has provided about himself (ref) and the knowledge obtained by mining raw mobile sensors (ref). On Figure 4.1, This part of the KB is represented as the left-most KB, sitting between the main KB and the *Procedural Module*.The sensor based context allows the system to proactively target the right users at the right time and thus improve the efficiency and accuracy and also stickiness of the KA process.

The third KB part, is the meta-knowledge and KA rules that drive the dialog and knowledge acquisition process (ref). Although in our implementation we used Cyc KB (ref and also tested Umko KB (ref), the approach is not fixed to any particular knowledge base. But the KB needs needs to be expressive enough to be able to cover the intended knowledge acquisition tasks and meta-knowledge needed for the system's internal workings.

Because the full Curious Cat system including the KB is too big and complex to be fully explained here (the KA Meta Knowledge alone consists of 12,353 assertions and rules), we will focus on the fundamentals of the idea and approach, and define the simplest possible logic to explain the workings through the examples given in the ref and ref Table II.

The logic examples are given in formal higher order predicate logic, which we later replace with a more compact notation, with a slight change in the way the variables are presented. For better readability, instead of x, y, z, we mark variables with a question mark (?) followed by a name that represents the expected type of the concepts the variable represents. For example, when we see a variable in a logical formula like $CCUser(?PERSON)$, we immediately know that $?PERSON$ can be only replaced with (bound to) instance or subclasses of the concept $Person$. We start predicate names with a small letter (*predicate*) and the rest of the concepts with a capital letter (*Concept*). At this point it is worth noting that while our logical definitions and formalization are strongly influenced by Cyc(**Lenat1995**), and while the approach is based on the Cyc upper ontology, the approach is general and not bound to any particular implementation, and our notation below reflects but is not tightly bound to that of OpenCyc.

Add footnote the paper

4.2.1 Upper Ontology

First we introduce the vocabulary or terms (constants/concepts) that will allow us to construct the upper ontology which is the glue of any knowledge base that can be used for

machine inference:

$$S_{Constants} = \{Something, Class, Number, Predicate, subclass, \\ is, arity, argClass, argIs\} \tag{4.1}$$

In the standard *predicate logic*, $P(x)$ notation tells us that whathever the x stands for, it has the property defined by the predicate P. For example, the following propositional function:

$$Person(x) \tag{4.2}$$

is stating that something (x) is a person, or more precisely, x is an instance of a class $Person$. In order to be able to construct logical statements ranging over classes and their instances in a more controlled and transparent way, we use a constant is, and define it as a predicate denoting that something is an instance of some class.

Definition 4.1 (predicate "is"). Predicate $is(x, y)$ denotes that x is an instance of y. For example, stating $is(John, Human)$ defines John as one instance of the class of humans.

Now, to make things clearer and more precise, instead of writing instance relation through custom predicate $Person(x)$, we can use more precise syntax, which allow us to specify what is instance of which class: $is(x, Person)$.

As visible in the Equation 4.2, in predicate logic, predicates are defined only with usage. Everything that we write as a predicate in a similar formula is then defined as a predicate. This is not a desired behavior in our KA approach, since the knowledge will be coming from the users with various backgrounds and without any idea of predicate logic. For this reason we need more control of what can be used where, if we want to be able to check for consistencies and have control of our KB with the inference engine. For this reason, we enforce a constraint (rule).

Definition 4.2 (Predicate Constraint). Everything that we want to use in the KB as a predicate, must first be defined as an instance of the $Predicate$ class: $\forall x \in S_{Predicates} : is(x, Predicate)$. From the other side, set of predicates can be defined as $S_{Predicates} = \{x : is(x, Predicate)\}$. This constraint can be inserted into our KB as a *material implication* rule, which needs to be true at all times, to serve as a constraint:

$$\forall P \forall x_{1...n}(P(x_1...x_n) \implies is(P, Predicate)) \tag{4.3}$$

Now, careful reader might notice, that we actually cannot use is as a predicate, since nowhere in our KB is stated that this is actually a predicate. To fix this error and make our KB consistent with its constraints, we need to add an assertion defining what term is stands for:

$$is(is, Predicate) \tag{4.4}$$

At the time the above assertion (Assertion 4.4) is asserted into the KB, it also becomes valid assertion, since it complies with the constraint defined in Definition 4.2 and thus our Constraint Rule 4.3 is true. After this assertion is in the KB, ir can be used as a predicate because it is an instance of the term $Predicate$ and complies with our constraints.

At this point we can define(assert) the rest of our predicates:

$$is(subclass, Predicate) \wedge is(arity, Predicate) \wedge is(argClass, Predicate) \\ \wedge is(argIs, Predicate) \tag{4.5}$$

And also the rest of our terms, which we define as instances of term $Class$.

$$\begin{aligned} &is(Class, Class) \wedge is(Predicate, Class) \\ &\wedge is(Number, Class) \end{aligned} \tag{4.6}$$

In Predicate logic, predicates have a property called arity, which defines number of arguments that the predicate can have. For example, if predicate P has arity of 1, then it can only take one operand (variable or term). In this case only $P(x)$ or $P(a)$ are valid statements, and $P(x, y)$ is not. In our KB, arity is defined using *arity* predicate, which itself was defined in Assertion 4.5.

Definition 4.3 (predicate "*arity*"). Predicate $arity(x, y)$ denotes that predicate x has arity of y.

Similarly, as with the constraint that all predicates need to be defined as such (Definition 4.2), we gain more control over KB and make things easier for the inference engine and KA approach, if we limit the assertions, to "obey" the predicate arities. For this reason our KB has additional constraint.

Definition 4.4 (Arity Constraint). All assertions in CC KB are valid only, when the predicates used in the assertion have the same number of arguments as defined with their *arity* assertions:

$$\forall P \forall n \forall x_{1...n} : (P(x_1, ..., x_n) \implies arity(P, n)) \tag{4.7}$$

After we add the above rule (Constraint Rule 4.7), our KB is not consistant anymore, since all the $is(x, y)$ assertions (Assertions 4.4, 4.5) violate the constraint. We fix this by adding the following assertion:

$$arity(arity, 2) \wedge arity(is, 2) \tag{4.8}$$

This makes the KB consistant again, because we defined all the arities of predicates *arity* and *is*, which we have used so far in our KB, as well as defined them with $is(x, Predicate)$ assertions. We can now continue with defining the arities of the rest of the predicates:

$$arity(subclass, 2) \wedge arity(argClass, 3) \wedge arity(argIs, 3) \tag{4.9}$$

We can see now that the arity of *is* predicate is defined as 2 (same as for *subclass* and *arity*, which can be used to define arity of itself), and can confirm that all the logical formulas in the definitions up to now are correct.

To be able to describe the world in more detail, we define the *subclass* predicate, which handles the hierarchy relations between multiple classes (unlike *is*, which handles relationships between classes and their instances).

Definition 4.5 (predicate "*subclass*"). Predicate $sublcass(x, y)$ denotes that x is a subclass of y. For example, asserting $subclass(Dog, Animal)$, is specifying all dogs, to be a sub-class of animals, and $subclass(Terrier, Dog)$ is speficfying that all terriers are sub-class of dogs. At this point we might notice that while it is logical to us that terriers are also a sub-class of animals, there is no way for the machine inference to figure that out. For this reason we need to introduce a "subclass transitivity" inference rule:

$$\forall x \forall y \forall z : ((subclass(x, y) \wedge subclass(y, z)) \implies subclass(x, z)) \tag{4.10}$$

The rule above is basically saying that if a first thing is a sub-class of a the second thing, and then thesecond thing is a subclass of the thirdthing, then the third thing is a sub-class of the first thing as well.

Because we want to be able to prevet our system from acquiring incorrect knowledge, we need to limit the domains and ranges of the predicates (arguments). This could be done by adding a speciic constraint rules (material implication rule without the power to make new assertions). For example, for both *subclass* arguments, to only allow instances of a $Class$, we could assert:

$$\forall x_1 \forall x_2 : (subclass(x_1, x_2) \implies (is(x_1, Class) \wedge is(x_2, Class)) \tag{4.11}$$

Because the rule (Rule 4.11 is only true if the right part (the consequent) is true, or the left part (the antecedent) is false, its inclusion in the KB (as with other constraint rules) forces the KB to not allow the arguments of subclass to be anything else than an instance of a class $Class$. It would be hard to construct a large KB, by writing the rule like this for each of the thousands potential predicates. To make this easier, following Cyc practice(**Lenat1995**), we will introduce $argIs$ predicate (Definition 4.5). To make this definition more understandable, let's first expand the example (Rule 4.11 above:

$$\begin{gathered} \forall x_1 \forall x_2 : (subclass(x_1, x_2) \implies (is(x_1, Class) \wedge is(x_2, Class)) \\ \iff \\ argIs(subclass, 1, Class) \wedge argIs(subclass, 2, Class)) \end{gathered} \tag{4.12}$$

This rule above (Rule 4.12 states, that the constraint rule (Rule 4.11 can be written as 2 $argIs$ assertions. Instead of writing full rule, the constraint for the argunent of *subclass* can be written simply as $argIs(subclass, 1, Class)$. To make this hold for all the combinations of predicates (not just *subclass* from example), we can re-phrase the rule to be general, and also define the $argIs$ predicate.

Definition 4.6 (predicate "$argIs$"). Predicate argIs(x,y,z) denotes that the $y-th$ argument of predicate x, must be an instance of z. For example, asserting $argIs(subclass, 1, Class)$, states that the first argument of predicate *subclass* must be an instance of $Class$. This is enforced by the following constraint rule:

$$\begin{gathered} \forall P \forall n \forall x_1 ... x_n \forall C_{1...m} : \\ ((arity(P, n) \wedge P(x_1, ..., x_n)) \implies (is(x_1, C_{1...m}) \wedge ... \wedge is(x_n, C_{1...m})) \\ \iff \\ argIs(P, 1, C_{1...m}) \wedge ... \wedge argIs(P, n, C_{1...m})) \end{gathered} \tag{4.13}$$

These definitions allow us to use simple $argIs$ assertions, instead of complicated rules. For the cases, when the arguments shouldn't be instances of a $Class$, but its subclasses, we can define similar predicate and its constraint rules also fro $argClass$:

Definition 4.7 (predicate "$argClass$"). Predicate $argClass(x, y, z)$ denotes that the $y-th$ argument of predicate x, must be a subclass of z. For example, asserting $argClass(servesCuisine, 2, Restaurant)$, states that the first argument of predicate *servesCuisine* must be a subclass of *Restaurant*. This is enforced by the following constraint rule (similar as for $argIsa$, but for $argClass$):

$$\begin{gathered} \forall P \forall n \forall x_1 ... x_n \forall C_{1...m} : \\ ((arity(P, n) \wedge P(x_1, ..., x_n)) \implies (subclass(x_1, C_{1...m}) \wedge ... \wedge subclass(x_n, C_{1...m})) \\ \iff \\ argClass(P, 1, C_{1...m}) \wedge ... \wedge argClass(P, n, C_{1...m})) \end{gathered} \tag{4.14}$$

Before we can assign argument constraints to our existing predicates and have all the KB valid, we need to define a special class $Number$:

Definition 4.8 (class $Number$ and its instances). All natural numbers are instances of the class $Number$. Formally, this can be asserted into a KB as:

$$\forall x \in \mathbb{N} : is(x, Number) \tag{4.15}$$

This now allows us to use $argIs$ and $argClass$ predicates instead of complicated rules, to define types of arguments inside any predicate used in the KB. By using these two newly defined predicates, we can now proceeed to assert the argument limits of the $argIsa$ predicate itself:

$$argIs(argIs, 1, Predicate) \wedge argIs(argIs, 2, Number) \wedge argIs(argIs, 3, Class) \tag{4.16}$$

We can see that the first argument must be an instance of $Predicate$, which holds in all three cases ($argIs$ is a first argument of the Assertion 4.16 above), since it was defined in Assertion 4.5. Similary, the second argument is a valid number, in all three cases as defined in Definition 4.8. Also the third arguments ($Predicate$, $Number$,$Class$), are all instances of the $Class$ as defined in Assertion 4.6, so the assertion 4.16 can be asserted and it doesn't invalidate itself throigh argument contraint rule (Constraint 4.13).

Similary, we can now proceed to define the rest of our predicates. Starting with the most similar $argClass$:

$$\begin{aligned} argIs(argClass, 1, Predicate) \wedge argIs(argClass, 2, Number) \\ \wedge argIs(argClass, 3, Class) \end{aligned} \tag{4.17}$$

Since we didn't yet assert any direct $argClass$ constraint, this predicate at this point defines the constraints, without any danger to invalidate our current KB.

Continuing with $subclass$. On the Definition 4.5, we can see that this predicate defines sub-class relationships between the classes. For this reason it makes sense to only allow instances of $Class$ for its arguments:

$$argIs(subclass, 1, Class) \wedge argIs(subclass, 2, Class) \tag{4.18}$$

Same as with the Assertion 4.17, we didn't yet assert any direct assertions about something being a sub-class of something, this assertion doesn't affect yet the validity of our KB, while prevents future assertions of $subclass$ predicate on anything but $Class$ instances.

For the $arity$ predicate, we can check our existing assertions (4.8, 4.9), and see that as the first argument we always have an instance of $Predicate$, while as the second argument we have an instance of a $Number$. According to this, it serves our purpose and is safe to limit the arguments of $arity$ to:

$$argIs(arity, 1, Predicate) \wedge argIs(arity, 2, Number) \tag{4.19}$$

We can now proceed to define our last (is) predicate constraints, which is a bit more complicated. If we look at our existing is assertions (4.4, 4.5, 4.15), we can see that as a second argument we always have an instance of a $Class$ ($Predicate$, $Class$), but for the first argument we can actually put in anything (is, $Class$, $Predicate$, $Number$, $\mathbb{N}$). From instance of a $Class$, instance of a $Predicate$, to an instance of a $Number$. For the second argument we can imediatelly asert

$$argIs(is, 2, Class) \tag{4.20}$$

In order to be able to say that some argument can be anything, we followed a Cyc example and introduce term *Something*, first mentioned in the set of our terms in Assertion 4.1. We set this as the constraint for the first argument of *is* predicate:

$$argIs(is, 1, Something) \tag{4.21}$$

But this assertion above (4.21), invalidates the correctnes of all of our *is* assertions, since none of the current first arguments are instances o *Something* (see Assertions 4.4, 4.5 and 4.15). To fix this, we need to be able at least to say that things are instances of *Something* ($is(x, Something)$). According to the *is* argument constraint assertion above (4.20), *Something* must be an instance of a *Class*. So we define it as so:

$$is(Something, Class) \tag{4.22}$$

Now, as a consequence of this, we could assert for all of the arguments that are used in *is* (*is*, *Class*, *Predicate*, *Number*, $\mathbb{N}$), that they are an instances of the *Something*. This would be highly unpractical, since we would need to do this for every future constant to be used by *is* predicate (especially unpractical for infinite number of $\mathbb{N}$).

Instead, since we know that *Something* is an instance of a *Class*, we can use it in our *subclass* assertions (a consequence of constraint Assertion 4.18) and state that *Class* is a subclass of *Something*:

$$subclass(Class, Something) \tag{4.23}$$

A consequence of this assertion is (because of inference rule 4.10), that every sub-class of anything that exists in our KB (because we can only use instances of *Class* in *subclass* assertions), is a sub-class of *Something* as well. This doesn't yet seem to help us make our 1st *is* predicate arguments instances of *Something*, but it will, after we address another weaknes in our current KB. Consider the continuation of the example we started in *subclass* definition (Definition 4.5, where a terrier is a sub-class of dog, and dog is a sub-class of animals ($subclass(Dog, Animal)$, $subclass(Terrier, Dog)$). If we introduce an instance of the class *Terrier*, let's say, a real dog named "Spot" ($is(Spot, Terrier)$), we can see that there is a logical problem, since Spot is a terrier in our KB, but not a dog, or even an animal (there is nothing to support $is(Spot, Animal)$ assertion. This can be fixed by introducing the following Inference Rule:

$$\forall x \forall y \forall z : ((is(x, y) \wedge subclass(y, z)) \implies is(x, z)) \tag{4.24}$$

This rule is basically saying, that if there is an instance of a class, and this class is a sub-class of another class, then this instance is also an instance of the other class. Now, this inference rule (4.24), together with the fact that *Class* is a sub-class of *Something*, and the *subclass* transitivity inference rule (4.10), makes everything that is instance of a class, or its sub-class (which is everything in our KB), also an instance of *Something*, and thus proving the assertion 4.21 correct and consequently our KB fully consistant again.

At this point our Upper Ontology is defined (it is visually presented on Figure 4.3) and ready to build upon as will be described in the next chapters. Since Curious Cat main implementation is based on Cyc, and it was inspired by the way Cyc ontology is constructed and being used, this upper ontology reflects the main part of Cyc upper ontology (see Chapter 5, implementation on how this formalization maps to Cyc). While our upper ontology logical definitions and formalizations are strongly influenced by Cyc, the approach is general and not bound to any particular implementation, and our notation reflects but is not tightly bound to that of Cyc. For example, the usage of *argIs* and *argClass* can be replaced by *domain* and *range* when using a RDF schema, such as was done in our RDF prototype implenentation(**Bradesko2012a**), or a completely custom constraints can

be used in specific ontologies, so this upper ontology is more of a guidence and a tool to be able to explain apporach, than a fixed ontology that needs to be implemented.

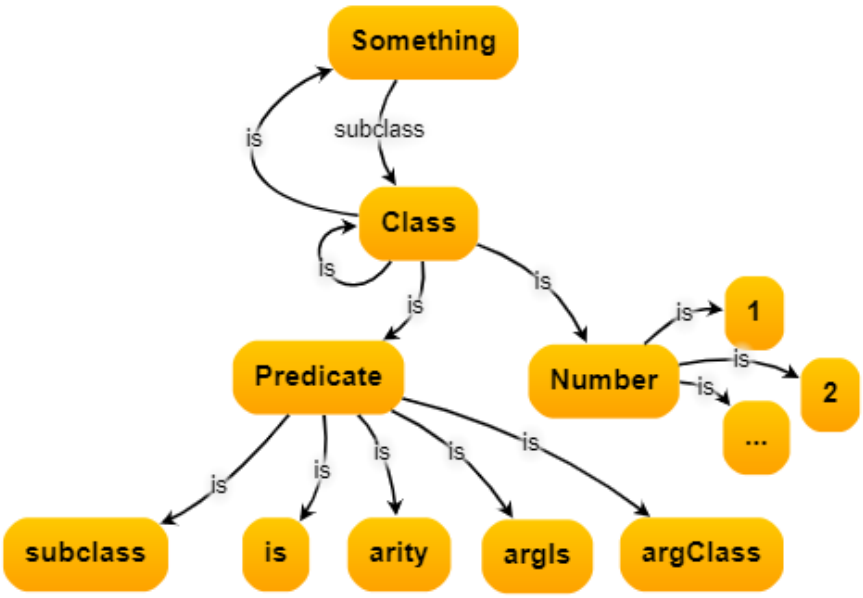

Figure 4.3: Upper ontology terms, with 'is' and 'subclass' relations.

4.2.2 Existing Knowledge

This sub-chapter extends our upper ontology example with additional knowledge that allows us to explain the system through the examples given in Table 4.1 and Table 4.2.

As also visible in the Architecture schematic (**??**, KB with background knowledge is one of the most crucial elements of proposed approach. It serves both, as the driving force behind the source of the questions, since with the help of contextual knowledge triggers the infrence to produce logical queries for the missing parts of the knowledge. At the same time it is the drive behind the proactive user interaction, sarting either as a question or a suggestion. Finally, the background knowledge is also used for validation of answered questions. If the answers are not consistent according to the existing KB, users are required to re-formulate, or repeat the answer.

The main *Curious Cat* implementation, uses an extended full Cyc ontology and KB, similar to that released as ResearchCyc, as a common sense and background knowledge base. This is far too big (millions of assertions), to be explained in any detail here with the general approach (it is explained in more detail in Chapter 5, Implementation). In this chapter we define only the concepts and predicates and thus construct the minimal example KB that is necessary for explaining the p roposed system.

First we introduce the set of new concepts that we need for the food part of the ontology. These concepts are defined on top of the existing Upper Ontology, so the new parts of the KB should maintain the consistency of the already described parts. The set of terms S_{Food}, needed to describe this part of the KB is as follows:

$$S_{Food} = \{FoodOrDrink, Food, Bread, Baguette, Drink, Coffee\} \tag{4.25}$$

Where each of these terms is an instance of a $Class$:

$$\forall x \in S_{Food} : isa(x, Class) \tag{4.26}$$

Then, these classes are connected into a class-hierarchy, which is done with a *subclass* predicate:

$$\begin{gathered} subclass(Drink, FoodOrDrink) \wedge subclass(Coffee, Drink) \wedge \\ subclass(Food, FoodOrDrink) \wedge subclass(Bread, Food) \wedge \\ subclass(Baguete, Bread) \end{gathered} \tag{4.27}$$

Which is also presented graphically on Figure 4.4 below.

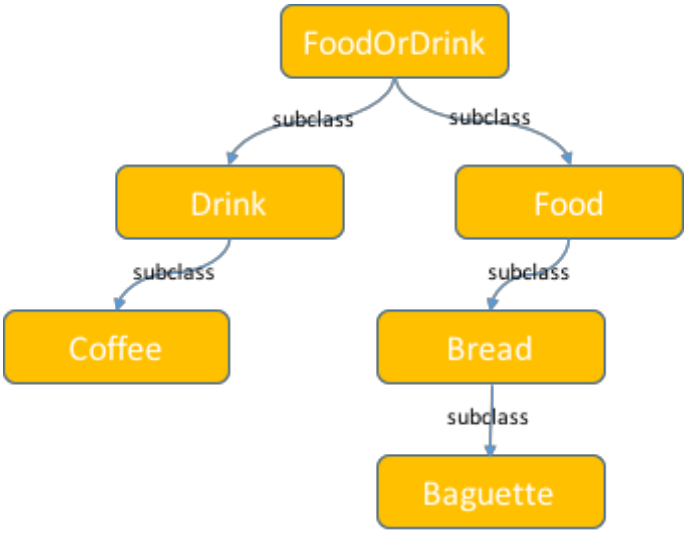

Figure 4.4: Hierarchy of food related terms, specified by subclass predicates.

The second part of ontology (KB) constructing knowledge about places consist of the following terms:

$$S_{Place} = \{Placa, PublicPlace, PrivatePlace, Restaurant\} \tag{4.28}$$

Where each of these terms is also an instance of a *Class*:

$$\forall x \in S_{Place} : isa(x, Class) \tag{4.29}$$

Then, these classes are connected into a class-hierarchy, which is done with a *subclass* predicate:

$$\begin{gathered} subclass(PrivatePlace, Place) \wedge subclass(PublicPlace, Place) \wedge \\ subclass(Restaurant, PublicPlace) \wedge subclass(Home, PrivatePlace) \end{gathered} \tag{4.30}$$

Which is also presented graphically on Figure 4.5 below.

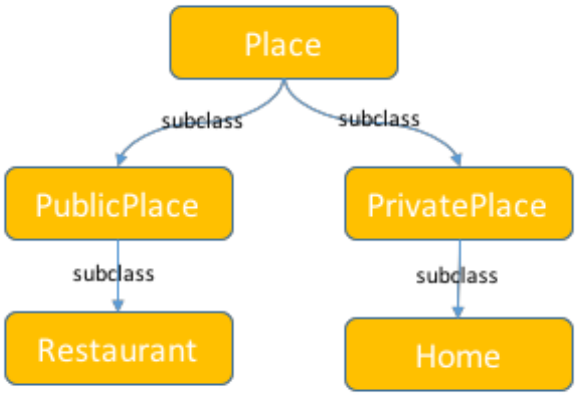

Figure 4.5: Hierarchy of place related terms, specified by subclass predicates.

Then the last part (excluding predicates which are defined separately), of our *Existing Knowledge* consist of the following terms:

$$S_{Reest} = \{Service, WirelessService, Vehicle, Car, Animal, Duck, Human, User\} \tag{4.31}$$

Where each of these terms is also an instance of a $Class$:

$$\forall x \in S_{Rest} : isa(x, Class) \tag{4.32}$$

And a class-hierarchy:

$$\begin{aligned} &subclass(Car, Vehicle) \wedge subclass(WirelessService, Service) \wedge \\ &subclass(Duck, Animal) \wedge subclass(Human, Animal) \wedge \\ &subclass(User, Human) \end{aligned} \tag{4.33}$$

Which is also presented graphically on Figure 4.6 below.

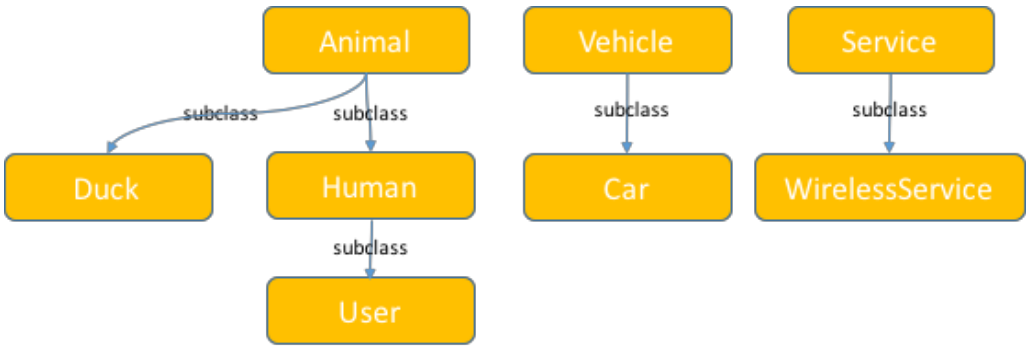

Figure 4.6: Hierarchy of the rest of the terms from our *existing knowledge*.

We can see that all these new terms add on top of the existing set of terms from upper ontology ($S_{Constants} = S_{upper} \cup S_{Food} \cup S_{Place} \cup S_{Rest}$).

For defining the predicates, we need a bit more detailed definitions, since we also want to represent the constraints which will serve for the inference engine to check the validity of the answers (as explained in ref to constraints def. in UpperOnto).

Definition 4.9 (predicate *menuItem*). Predicate $menuItem(x, y)$ denotes that place x has a menu item y on its menu.Formally it is defined as:

$$\begin{aligned} &is(menuItem, Predicate) \wedge \\ &argIs(menuItem, 1, Restaurant) \wedge \\ &argClass(menuItem, 2, FoodOrDrink) \end{aligned} \tag{4.34}$$

As we see defined above (assertion 4.34), this predicate constraints allow us to only use instances of class *Restaurant* for the first argument, and sub-classes of class $FoodOrDrink$ for the second. Besides the items on the menu, to explain our examples, we also need a way to tell that a place provides some services.

Definition 4.10 (predicate *providesServiceType*). Predicate $providesServiceType(x, y)$ denotes that place x provides service y. Formally it is defined as:

$$\begin{aligned} &is(providesServiceType, Predicate) \wedge \\ &argIs(providesServiceType, 1, Place) \wedge \\ &argClass(providesServiceType, 2, Service) \end{aligned} \tag{4.35}$$

As defined in the Assertion 4.35, *providesServiceType* constraints allow us to only use instances of class *Place* (Public,Private, Home,Restaurant for the current stage of the KB) for the first argument, and sub-classes of class *Service* for the second.

Definition 4.11 (predicate *userLocation*). Predicate $userLocation(x, y)$ denotes that user x is located at place y. Formally it is defined as:

$$\begin{aligned} &is(userLocation, Predicate) \wedge \\ &argIs(userLocation, 1, User) \wedge \\ &argIs(userLocation, 2, Place) \end{aligned} \tag{4.36}$$

As defined in the Assertion 4.36, *userLocation* constraints allow us to only use instances of class $User$ for the first argument, and instances of *Place* as a second.

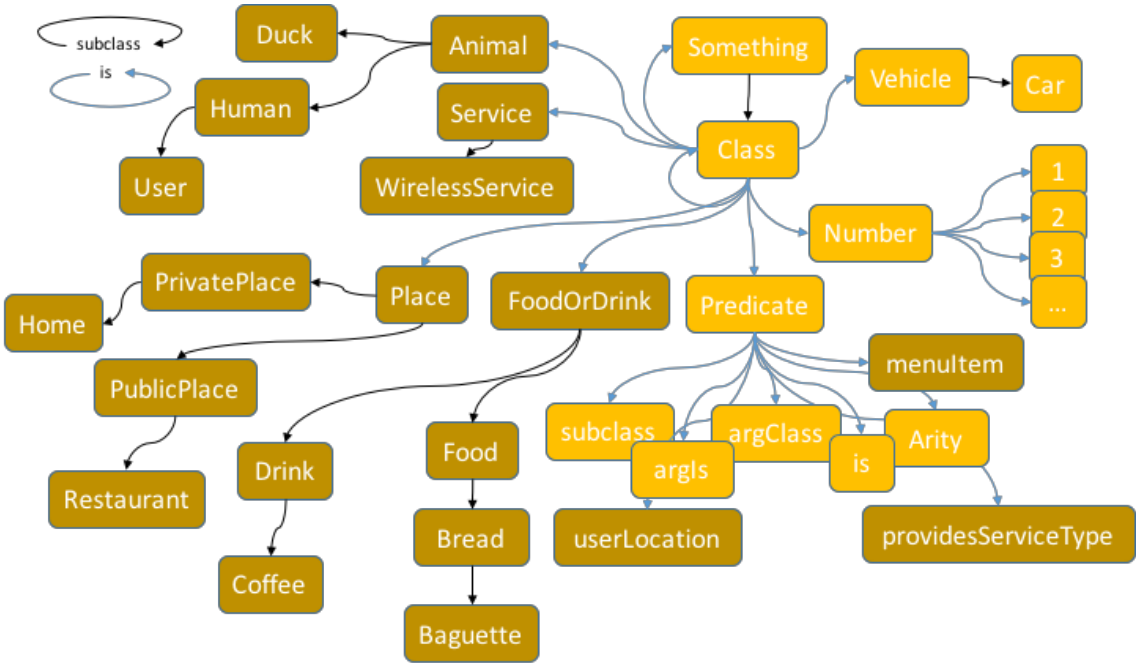

Figure 4.7: Current "existing knowledge" on top of upper ontology

At tjhis point we have a formal KB structure representing a minimal *existing knowledge* required to explan the examples from Table 4.1 and Table 4.2. The KB is graphically presented on the Figure 4.7, where the upper ontology is presented in lighter color and new KB parts in darker. Due to lack of space, the only relations are of *is* (represented by blue arrows) and *subclass* (represented with black arrows) predicates.

4.2.3 KA Knowledge

In the previous two sections we defined the upper ontology (Section 4.2.1) and then using its vocabulary to define the pre-existing knowledge (Section 4.2.2). This will suffice to support the explanation of the proposed KA approach. Similar as with the other parts of the KB, listing full set of *Curious Cat* KA rules would not fit in the paper. Instead we define an example set of the KB, sufficient for describing the approach and keeping the explanation as simple as possible.

As a starting point, we need to define a main KA meta-class $Formula$, which's is a special class (like $Number$).

Definition 4.12 (class $Formula$ and its instances). All logical formulas (assertions, queries) are instances of the class $Formula$. Formally, this can be represented as:

$$\forall P \forall x_{1...n} : is(P(x_1, ..., x_n), Formula) \tag{4.37}$$

Basically all of the content of the KB and queries are instances of $Formula$. For example, assertion $userLocation(User1, Ljubljana)$ is an instance of $Formula$ and consequentially the statement $is(userLocation(User1, Ljubljana), Formula)$ is true.

Then we need to define additional KA meta predicates:

Definition 4.13 (predicate "$known$"). Special meta-predicate $known(x)$ denotes that the formula x can be proven in the KB. This predicate is non-assertible, meaning that

it cannot be used to add things into the KB, but it can be used in inference rules. For example, since the Assertion 4.22 is already asserted in the KB and true, the query/unassertible statement $known(is(Something, Class))$ is also true. The predicate is formally defined as follows:

$$is(known, Predicate) \wedge arity(known, 1) \wedge argIs(known, 1, Formula) \tag{4.38}$$

In a similar fashion as predicate *known*, we define *unknown* predicate.

Definition 4.14 (predicate "*unkknown*"). Special meta-predicate $unknown(x)$ denotes that the formula x can **not** be proven in the KB. This predicate is non-assertible, meaning that it cannot be used to add things into the KB, but it can be used in inference rules. Building on the example given in the Definition 4.13 (*known* predicate), the query $unkown(is(Something, Class))$ is **not true**, since the assertion exist an is known. On the other hand, $unknown(is(Human, Duck))$ is true, since there is no knowledge in the KB which would support the encapsulated *is* statement. The predicate is formally defined as follows:

$$is(unknown, Predicate) \wedge arity(unknown, 1) \wedge argIs(unknown, 1, Formula) \tag{4.39}$$

And now one of the main KA predicates, which is used to provide the CC system with the formulas which need to be converted to NL and presented to the user.

Definition 4.15 (predicate "*ccWantsToAsk*"). One of the main KA predicates written as $ccWantsToAsk(x, y)$, denotes that the *Curious Cat* system wants to ask user x a question represented by the formula y. For example, assertion

$$ccWantsToAsk(User1, userLocation(User1, x))$$

tells CC system to ask user the question $userLocation(User1, x)$, which, after it goes through logic to NL conversion ref is paraphrased as "Where are we?", as hinted in the step 1 in the Table 4.1. The predicate is formally defined as follows:

$$\begin{gathered} is(ccWantsToAsk, Predicate) \wedge arity(ccWantsToAsk, 2) \wedge \\ argIs(ccWantsToAsk, 1, User) \wedge argIs(ccWantsToAsk, 2, Formula) \end{gathered} \tag{4.40}$$

4.2.4 Context

Definition 4.16 (predicate "*probableUserLocation*"). DadaMaybe define this one at the end, since it could also be defined in the Context section

Chapter 5

Real World Knowledge Acquisition Implementation

5.1 Cyc

TBW

5.1.1 SCG

TBW

5.2 Mobile Client

TBW

Chapter 6

Evaluation

TBW
chapters that

Chapter 7

Conclusions

We came to the following conclusions ...

Bibliography

Publications Related to the Thesis

All publications related to the thesis should be referenced in the text.

Other Publications (optional)

...

Biography

The author of this thesis ...