Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2004-033
AITR-2004-001

May 18, 2004

Learning Commonsense Categorical Knowledge in a Thread Memory System

Oana L. Stamatoiu

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Learning Commonsense Categorical Knowledge in a Thread Memory System

by

Oana L. Stamatoiu

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2004

© Massachusetts Institute of Technology 2004. All rights reserved.

Certified by: Patrick H. Winston
Ford Professor of Artificial Intelligence and Computer Science
Thesis Supervisor

Accepted by: Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Learning Commonsense Categorical Knowledge in a Thread Memory System

by

Oana L. Stamatoiu

Submitted to the Department of Electrical Engineering and Computer Science on May 6, 2004, in partial fulfillment of the requirements for the degree of Master of Engineering in Computer Science and Engineering

Abstract

If we are to understand how we can build machines capable of broad purpose learning and reasoning, we must first aim to build systems that can represent, acquire, and reason about the kinds of commonsense knowledge that we humans have about the world. This endeavor suggests steps such as identifying the kinds of knowledge people commonly have about the world, constructing suitable knowledge representations, and exploring the mechanisms that people use to make judgments about the everyday world. In this work, I contribute to these goals by proposing an architecture for a system that can learn commonsense knowledge about the properties and behavior of objects in the world. The architecture described here augments previous machine learning systems in four ways: (1) it relies on a seven dimensional notion of context, built from information recently given to the system, to learn and reason about objects' properties; (2) it has multiple methods that it can use to reason about objects, so that when one method fails, it can fall back on others; (3) it illustrates the usefulness of reasoning about objects by thinking about their similarity to other, better known objects, and by inferring properties of objects from the categories that they belong to; and (4) it represents an attempt to build an autonomous learner and reasoner, that sets its own goals for learning about the world and deduces new facts by reflecting on its acquired knowledge. This thesis describes this architecture, as well as a first implementation, that can learn from sentences such as "A blue bird flew to the tree" and "The small bird flew to the cage" that *birds can fly*. One of the main contributions of this work lies in suggesting a further set of salient ideas about how we can build broader purpose commonsense artificial learners and reasoners.

Thesis Supervisor: Patrick H. Winston
Title: Ford Professor of Artificial Intelligence and Computer Science

Acknowledgments

I would like to thank first of all my thesis advisor, Patrick Winston, for his mentoring, his guidance, and his support of my work. Most of all, I owe to him my interest in AI and my passion for understanding human intelligence, as well as any conceptual and writing clarity of the work I present here.

I also want to thank the members of the Genesis Group at the MIT Artificial Intelligence and Computer Science Laboratory, whose ideas have deeply shaped my goals and my approach of understanding intelligence from a computational point of view. I want to extend special thanks to Ian Eslick; I have found our discussions about symbolic learning and reasoning especially useful in my work.

Looking back on my years at MIT, I find that I have been deeply changed, as a thinker and as a person, by several professors whom I've had the honor to listen to and learn from. Among them, I'd like to thank professors Patrick Winston and Marvin Minsky in the computer science field for inspiring my curiosity and passion for artificial intelligence and cognition, professor James Munkres in the mathematics department for nurturing my early inclinations as a mathematician, and professor Ned Hall in the philosophy department for teaching me to look at the world in more than one way.

I feel my current work has been made possible above all by the affection and support of my parents and grandparents whom I deeply love and respect.

Finally, I would like to dedicate this work to my grandfather, Sergiu Caster, who has inspired my early interest in books, my passion for opera, and my genuine curiosity to discover the inner workings of the human mind.

Contents

1 Introduction **8**
- 1.1 Building a Learner of Commonsense Knowledge 9
- 1.2 Outline 11

2 Grounding in Previous Work **12**
- 2.1 Context in Machine Learning 12
- 2.2 Similarity and Categorization 17
- 2.3 The Problem of Representation 21
- 2.4 Knowledge Acquisition and Learning 23
- 2.5 The Bridge Project 27

3 System Design **30**
- 3.1 System Specifications 31
- 3.2 The Representations and Implicit Assumptions 34
- 3.3 Design Overview 36
- 3.4 Using Context for Learning and Question Answering . . 38
- 3.5 The Conceptual Model for Learning 42
  - 3.5.1 The Similarity Mechanism 44
  - 3.5.2 Stereotypes for Classes of Objects 46
  - 3.5.3 The Memory Module 48
  - 3.5.4 The Query Answering Methods 49

4 Implementation and Results **54**
- 4.1 The Squirrel Implementation 54
  - 4.1.1 The Implementation of the Backend 55
  - 4.1.2 The Implementation of the User Interface 57
- 4.2 Concrete Learning Examples 59
  - 4.2.1 Robins fly because they are birds, and birds can fly 59

4.2.2 Planes fly because they are similar to robins, and robins fly . . . 63
4.2.3 A robin is made of feathers, has wings, and is a bird . . . 65

5 Evaluation and Discussion **67**
5.1 System Evaluation . . . 67
5.1.1 Penguins can't fly . . . 68
5.1.2 Contributions of the Learning Architecture . . . 68
5.2 Surprises . . . 71

6 Salient Ideas for Learning with Context **73**
6.1 Richer Representations of the World . . . 73
6.2 Better Use of Context . . . 74
6.3 Better Learning and Reasoning Heuristics . . . 74
6.4 Better Models of Memory . . . 76
6.5 Autonomy... and All That Jazz . . . 77

7 Contributions **78**

A Examples of Squirrel Code **80**
A.1 Pseudocode for the Assessing the Similarity of Two Things 80
A.2 Code for Calculating Stereotypes . . . 83
A.3 Pseudocode for the Reflection Thread of Memory . . . 88
A.4 Code Template for the Learn Methods of the Learner . 89
A.5 Code of the Qualify Method of the Learner . . . 90

List of Figures

2.1 An example of a Thing structure. 29

3.1 An MDD showing the main modules of the learning system. 37
3.2 An MDD showing the layering of the learning system, by module function. 38
3.3 A pictorial representation of the Context object. 40
3.4 The steps involved in processing an input sentence. . . . 43
3.5 Calculating the stereotype of a class of objects from the absolute and relative stereotypes. 47
3.6 The threaded structure of the `Memory` module. 49

4.1 A screenshot of the Squirrel GUI. 57

List of Tables

3.1 The classes of knowledge Squirrel learns. 32
3.2 The kinds of questions Squirrel can answer. 33
3.3 The methods used to answer queries about objects' properties. 52
3.4 Priority of deduction methods for each kind of user query. 53

Chapter 1

Introduction

"My aim is to put down on paper what I see and what I feel in the best and simplest way." (Ernest Hemingway)

If we are to understand how we can build machines capable of broad purpose learning and reasoning, we must first aim to build systems that can represent, acquire, and reason about the kinds of commonsense knowledge that we, humans, have about the world. This endeavor suggests steps such as identifying the kinds of knowledge people commonly have about the world, constructing suitable knowledge representations, and exploring the mechanisms that people use to make judgments about the everyday world. In this work, I contribute to these goals by proposing an architecture for a rational system that can learn commonsense knowledge about the properties and behavior of objects in the world. In this thesis I describe this architecture and the fundamental ideas in which it grounds; I present an implementation, called Squirrel, which embodies these ideas; and I discuss how the architecture can be used to support broader purpose learning about the world.

The architecture described here augments previous machine learning systems in four ways.

First, it uses a notion of context, built from information recently given to the system, to learn and reason about objects' properties. The context has seven dimensions, where each dimension represents a kind of property that an object has. Examples of these properties are an object's capabilities, what an object is made of, and what categories of objects it belongs to.

Second, the architecture includes multiple methods that it can use to reason about objects, so that when one method fails, it can fall back

on others. This is important because the ability to think about things in different ways is likely one of the hallmarks of human intelligence.

Third, the architecture illustrates the usefulness of reasoning about objects by thinking about their similarity to other, better known objects, and by inferring properties of objects from the categories that they belong to. Similarity and categorization appear to be fundamental cognitive processes whose understanding may help us better understand human cognition in general. As such, it is important to illustrate how well a system can reason about the world by using judgments based on similarity and categorization.

Fourth, this architecture represents an attempt to build a autonomous learner and reasoner, that sets its own goals for learning about the world and deduces new facts by reflecting on its acquired knowledge.

This thesis describes this architecture as well as a first implementation, called Squirrel, that can learn from sentences such as "A blue bird flew to the tree" and "The small bird flew to the cage" that *birds can fly*. One of the main contributions of this work lies in suggesting a further set of salient ideas about how we can build broader purpose commonsense artificial learners and reasoners.

1.1 Building a Learner of Commonsense Knowledge

The grand vision of the work described here is to understand how computers can be built to learn and reason about the world, and thus how they can be turned into *intelligent* or *rational* entities. This project represents a step toward this vision, that of developing an architecture suitable for accumulating and generalizing knowledge about the properties and behavior of objects in the world, from simple English sentences. I describe both the architecture and an implementation. Throughout this thesis, I call the architecture *the architecture* or *the system*, and I call its current implementation *Squirrel*.

Before describing the system in any more depth, I want to clarify what I mean when I say the system acquires "commonsense knowledge" about the world, and what I mean when I call it a "learning system".

I use "commonsense knowledge" throughout this thesis to refer to general knowledge that we humans have about the objects around us and their properties. Other authors use the term "commonsense" to refer to a broader kind of knowledge about the world [1]. Specific examples of what I call commonsense knowledge include knowledge that: people can walk, people have legs, birds can fly, birds have wings,

and a bird is an animal. The knowledge the system described here accumulates does not include, for example, social knowledge such as knowledge that people like to talk to each other.

I call the system I have built a "learning system" because its main purpose is to deduce new facts about the world from old facts. Upon making these deductions, the system stores the new facts in its knowledge base. In this way, the system grows its knowledge about the world by deducing new knowledge from old. This is the meaning that "to learn" has in this thesis.

Concretely, the system is an incremental learner as defined by (Devaney and Ram 1996, [23]). It receives input English sentences one at a time, and incorporates knowledge for each sentence in its knowledge base. The system refines what it knows about the world dynamically, from each new sentence and from queries about objects and their properties, posed by a human user. For example, the system generalizes knowledge such as that birds can fly from input sentences such as "A blue bird flew to a tree", "The small bird flew from the cage to the river". The system's architecture is modular and contains four layers: a layer for representing objects and events in the world, a layer for language parsing and understanding, a layer for learning, and a layer for storing, retrieving, and manipulating long term knowledge. These layers interact with each other primarily by using English phrases.

The main contributions of this architecture are that:

- It uses similarity and categorization judgments that rely on the context built from information recently supplied to the system, to reason about objects and their properties,
- It illustrates the usefulness of having multiple ways to reason about situations so that when one method fails, the system can try others instead,
- It illustrates the role that communication has in prompting a system to consolidate its knowledge about the world and deduce new knowledge, and
- It uncovers a set of salient ideas about the properties an architecture should have in order to support broader purpose learning and reasoning about the world,

One of the main surprises I have found while designing and implementing this learning system is the usefulness of building the system to use English expressions to 'talk' to itself internally.

One of the salient ideas that has emerged out of this learning architecture is that the notion of context is essential to the quality of learning and reasoning in an artificial system.

1.2 Outline

This thesis is organized as follows. In chapter 2, I describe the previous work that this thesis grounds in. I discuss the use of context in machine learning, psychology research into similarity and categorization, and artificial intelligence research in the areas of knowledge representation and acquisition. I also describe the Bridge architecture [13] that my learning system is built on top of. The sections on context (section 2.1), knowledge acquisition (section 2.4), and the Bridge architecture (section 2.5) are crucial background for the remainder of this thesis.

In chapter 3, I describe the design of the learning system. I start by pointing out the fundamental ideas and specifications I desire the system to have. I then describe the knowledge representations I use and the underlying assumptions these representations imply. I further describe the main modules of the system, their interactions, and the roles they play in learning.

In chapter 4, I describe Squirrel, a first implementation of this system. I describe the user interface and backend implementation and present three concrete learning examples from Squirrel's execution.

In chapter 5, I evaluate the learning system and discuss the surprises I have found while designing and implementing Squirrel.

In chapter 6, I outline the salient ideas that have emerged from the current work, about what properties that an architecture should have in order to support broader purpose learning and reasoning about the world. Together, these ideas outline a platform for learning broader purpose knowledge about the world.

In chapter 7, I present the contributions that I have made toward building intelligent artificial systems.

Chapter 2

Grounding in Previous Work

"The secret of genius is to carry the spirit of childhood into maturity." (Thomas Henry Huxley)

A first step to designing an artificial learner consists of answering the following four fundamental questions: (1) what is the context in which learning happens, (2) what kinds of representations are best suited for the information that is to be learned, (3) what kinds of knowledge a machine can attempt to learn, and (4) what learning principles can be applied to deduce general information about the world around.

In this chapter, I describe previous work in the areas of context in machine learning, similarity and categorization as methods for reasoning about the world, knowledge representation, and knowledge acquisition. For each of these areas, I show how my current work grounds in the work described and how it contributes to further research in that area.

2.1 Context in Machine Learning

The use of context in machine learning is a relatively new topic of research in AI. Since the 1990s, context has been the topic of several conferences and workshops, such as the International and Interdisciplinary Conference on Modeling and Using Context (1997), and the International Conference on Machine Learning (1996). Though as of yet, there seems to be no universally agreed upon definition of con-

text (Bigolin, Brezillon 1997, [26]), several authors (Devaney and Ram 1996, [23], Matwin and Kubat 1996, [34], Turney 1996, [42], Lenat 1998, [22]) have advocated for the usefulness of context in concept learning as well as in expert domains (Agabra, Alvarez, Brezillon 1997, [15]), and moreover for the necessity of using context for machine learning tasks (Matwin and Kubat 1996, [34]). Researchers have concretely experimented with using context in a variety of domains, from studying wine fermentation (Agabra, Alvarez, Brezillon 1997, [15]), to using context to improve the formalization of natural language, namely translating from system requirements expressed in natural language to a conceptual model for the system (Bigolin, Brezillon 1997, [26]), to using context to improve knowledge discovery (Sala 1997, [36]).

I review here representative work which motivates the use of context in machine learning and illustrates the role of using context in concept learning and classification tasks. From this work, I argue that using context can substantially improve machine learning algorithms, and I give a brief overview of my use of context in this thesis.

Several authors have been interested in using context in concept learning (Devaney and Ram 1996, [23], Matwin and Kubat 1996, [34]), where an artificial learner takes as input a set of instances described in terms of a number of characteristics and aims to generalize a set of concepts from this input. (Devaney and Ram 1996, [23]) define the goal of such a learner to be to improve the prediction of the characteristics or behavior of instances unseen by the learner. Devaney and Ram advocate using context by arguing that in order to produce a robust concept learner, the learner must describe the instances it classifies in terms of attributes which are relevant to the problem at hand. They claim that context should include the goals and tasks of the learner, as well as knowledge about its environment, and that as context changes, the learner should also change the characteristics which it uses to describe instances.

To illustrate the usefulness of context in artificial learning, Devaney and Ram present an implementation of what they call an "attribute-incremental" concept learner, called AICC. An attribute-incremental learner is incremental, in that it incorporates inputs gradually and dynamically changes its concepts based on the inputs. It is attribute-incremental because it modifies not only its concepts, but also the attributes used to describe those concepts, so it is able to respond to changes in context. The main contribution of this learner is that, in response to changes in the attributes set, it produces a new set of concepts by modifying existing ones, in less time than it takes traditionally

systems to build the same new concepts from scratch. They are thus able to bootstrap new concepts from old by using context, a result especially noteworthy for situations where there is a considerable cost associated with the acquisition of information (Devaney and Ram 1996, [23]).

Devaney and Ram also situate the use of context in a larger picture, by arguing that context is important for learners that are placed in situations similar typical for human concept learners. They conclude that use of prior greatly increases the efficiency of the learning process, without detracting from the performance of the learner (Devaney and Ram 1996, [23]).

Like Devaney and Ram, (Matwin and Kubat 1996, [34]) are interested in using context for concept learning, and propose that context deserves more attention in the machine learning community. Furthermore, they argue that context is useful in learning and that learning tasks cannot be solved satisfactorily while context is ignored. They identify the most important question in systems which take context into account to be that of how to take into account the *change* in context between the training test and the testing set provided to the learner.

To support their point, they cite three examples of machine learning applications where the use of context is "either necessary or at least highly beneficial" (Matwin and Kubat 1996, [34]). The examples are: (1) a calendar application which schedules meetings of a university professor; the context here consists of the personal priorities of the professor, and experiments show that the poorest performance of the system correlates with the semester boundaries, where it takes the system some time to learn the new priorities of the professor; (2) Matwim and Kubat's application for detecting oil spills on the sea surface; the context here includes factors such as meteorological conditions, and they show that taking this information into account improves the prediction accuracy of the system; (3) a sleep analysis performed by Kubat, Pfurtscheller, and Flotzinger in 1994; in this experiment, the biological signals studied are subject-dependent, and the context is the sleeper; in this case, experiments proved that it was useful to look at several sleepers in order to infer information about the others.

(Matwin and Kubat 1996, [34]) argue that context is "inextricable from many classification tasks". They observe that context influences the saliency of different features of an object at different times. They further cite Katz, Gately, and Colling (1990) who provide several strategies for using context in learning and "demonstrate that the use

of context can result in substantially more accurate classification."

One of the major contributions of the paper by Matwin and Kubat [34] is providing a division of concepts into three groups, based on their sensitivity to context. The groups they identify are: (1) absolute concepts, that don't depend on any context, such as the notion of *number*, (2) relative concepts, that possess different properties in different circumstances, such as the notion of *poverty*, and (3) partially relative concepts, that have a set of properties which are always present, and a set of properties that change with context, such as a *swimming suit* that always has the same basic components, but whose color and size vary widely. Matwin and Kubat conclude that using context is necessary in many practical applications.

(Agabra, Alvarez and Brezillon 1997, [15]) provide an example of such a practical application, in the domain of wine making. They design a system that uses contextual knowledge, such as the weather at grapes' harvest time, to help predict a stop in the normal fermentation process of the wine. Their idea is that using contextual knowledge can help account for the causes of stuck fermentation and provide a correct explanation of why it happened. Their goal is to build a contextual model of the wine making process, run a simulation and obtain a result which indicates the risk of stuck fermentation and test the sensibility of the result to various conditions that appear along the wine making process. In their design, they use the onion metaphor, initially proposed by Tichiner, which models contextual knowledge as a core problem to solve, with contextual knowledge organized in layers around this core. similarly, Agabra, Alvarez, and Brezillon's system has layers of context, one for each step of the wine making process, before the fermentation step.

Through their work, (Agabra, Alvarez, Brezillon 1997, [15]) demonstrate that using context can be useful not only in everyday activities, but also in expert domains. They conclude that contextual knowledge can be essential to knowledge based systems, and that in expert domains, the expert knowledge can serve as the context for explaining a problem or evaluating a risk.

The work of Turney (Turney 1996, [42], Turney 1996, [43]) provides a look at context from yet another perspective, that of identifying context sensitive features in order to improve supervised learning from examples. (Turney 1996, [42]) provides a formal definition of "what it means for a feature to be context-sensitive to another feature", and claims that research on context demonstrates that exploiting contex-

tual information "can improve the performance of machine learning algorithms".

(Turney 1996, [42]) proposes a classification of object features resembling the classification of concepts by (Matwin and Kubat 1996, [34]). This classification includes primary features, that are useful for classifying object even in isolation from other information; contextual features, that are useful for classification only in combination with other features; and irrelevant features, that are never useful. Turney further argues that it is possible to use our commonsense knowledge about the world to distinguish between these primary, contextual, and irrelevant features.

Finally, Lenat's work in the context of the Cyc system (Lenat 1998, [22], Lenat, Guha, Pittman, Pratt, Shepherd 1990, [39], Guha, Lenat 1994, [30]) is worthy of attention. (Lenat 1998, [22]) discusses the use of context from the point of view of the problem of acquiring commonsense knowledge, and extending the Cyc base of knowledge (Lenat, Guha, Pittman, Pratt, Shepherd 1990, [39]). Lenat proposes an internal structure for context, composed of twelve "mostly-independent" dimensions along which contexts vary, and where each region of the space defines a context. Lenat argues in favor of this model on the grounds of efficiency: "it should enable a much more efficient, much more focused sort of virtual lifting of assertions from one context to another and [...] it should make it easier to specify the proper context in which an assertion (or question) should be stated".

The design and use of context in the system I present in this thesis draws on several observations and results described above. Specifically, I:

- Use context to illustrate its usefulness for both concept learning and classification tasks, as well as for learning and question answering. My learning system is in a sense a concept formation system, because it forms concepts about what objects in the world are like and what properties they have. The learner performs classification tasks in that it decides whether a given object belongs to a class by using similarity judgments between objects, which are heavily influenced by the context of the judgment,

- Design and build a learner with attribute incremental features, which describes the objects it learns about in terms of the kinds of knowledge present currently in the context of the conversation with the human user,

- Use Matwin and Kubat's (Matwin and Kubat 1996, [34]) division of concepts into absolute, relative, and partially relative concepts. As described in more detail in chapter 3, my system bases many of its judgments on the stereotype of a class of objects, which essentially is the union of an absolute concept with several relative concepts.

2.2 Similarity and Categorization

My thesis also grounds in previous work on judgments of similarity and categorization of objects. Many noted researchers in artificial intelligence and psychology (Tversky 1977, [44], Lakoff 1987, [21], Rosch 1978, [33], Rosch 1975, [32], Tenenbaum 2000, [41], Gentner and Markman 1997, [11], Winston 1980, [47]) have argued that similarity and categorization are instrumental in cognitive processing and concept formation.

Research on similarity was most influenced by Tversky's seminal work on the features of similarity. (Tversky 1977, [44]) argued that similarity is an organizing principle which people use to classify objects, to form concepts, and to make generalizations. He proposed a model of similarity where objects are represented as a set of features, and the similarity between objects is represented as a linear combination of their common and distinctive features. Tversky points out that similarity should not be thought of as a symmetric relation, because when confronted with stereotypes of a class of objects and with examples of objects that belong to the class, people tend to believe that the examples are more similar to the stereotype than vice versa. He cites Rosch's work on categorization in support of this observation. He further argues that similarity should not be though of as transitive either, at least in the case where the features on the basis of which similarity between pairs is asserted are different for the two pairs.

Tversky argues that the classical geometric model of similarity, where objects are represented as points in a two dimensional space, and the differences between them are represented as the euclidean distances between these points, is not adequate. Instead, he proposes the contrast model, which he derives from qualitative axioms about the similarity of objects. In this model, all features are given equal weight when calculating the similarity as a linear combination of common and distinctive features. This method yields reasonable success, but Tversky notes that the prediction of similarity improves when con-

text is taken into account, where context consists of the "frequency of mention" of the features.

(Tversky 1977, [44]) goes on to analyze the relations between people's judgments of similarity and difference, as well as the effects of context on similarity judgments. On the topic of similarity versus difference, he provides experimental evidence that people observe similar features more than distinctive features: "the relative weight of the common and the distinctive features varies with the task and [the data] support the hypothesis that people attend more to the common features in judgments of similarity than in judgments of difference". On the topic of similarity in context, he states that judgments of similarity depend on context in that the salience of object features depends on the context of the comparison.

Shortly after, (Shepard 1980, [37]) also contains ground breaking work on similarity. He is primarily interested in how living things respond to stimuli, and how a response learned to one stimulus may generalize to any other stimulus. He presents several models for assessing similarity and argues that different models are appropriate for different types of data, and that even for the same data, using different models may help bring out different but equally important properties of the input. In his later work, (Shepard 1987, [38]) is interested in similarity from the point of view of generalization. He claims that people generalize from one situation to another because we believe similar situations are likely to have the same consequences. He goes even further to assert that generalization should be the first law of psychology and he envisions generalization as a universal law the governs the functioning of the universe, and which is not restricted to the behavior of living things.

The work by (Gentner and Markman, 1990, [11]) sheds light on the cognitive processes that account for similarity and analogy judgments. Gentner and Markman were primarily interested in explaining the mechanisms of similarity and analogy, and argued that both involve the same process of structural alignment and mapping.

Their motivation for studying similarity was that similarity is important for many areas of cognition, and that understanding the process which accounts for people's similarity judgments may enable us to better understand human thinking in general. Gentner and Markman, as well as other authors (Bassok 1990, [5], Holyoak and Koh 1987, [14], Kolodner 1993, [18], Novick 1990, [27], Winston 1980, [47]) have argued that similarity helps us solve new problems, by employing procedures that we have used to solve prior similar problems. (Gentner

and Markman 1990, [11]) also argue that similarity has a central role in categorization judgments, citing, among others, Rosch's work for support: "similarity is often given a central role in categorization [...] It is common to assume that objects can be categorized on the basis of perceptual, behavioral, or functional commonalities with the category representation (e.g. robins are seen as birds because of their perceptual and behavioral similarity to a prototype bird or to many other birds that have been encountered)." Gentner and Markman situate their work on similarity in a broader context by arguing that comparison processes foster insight into problems, they lead us to new inferences, and ultimately spark to creative thinking: "comparison processes foster insight. They highlight commonalities and relevant differences, they invite new inferences, and they promote new ways of constructing situations. This creative potential is easiest to notice when the domains compared are very different [...] even prosaic similarity comparisons can lead to insights."

Research on categorization was headed primarily by Eleanor Rosch. In her seminal work on categorization (Rosch 1965, [32], Rosch 1978, [33]), she outlines the following two principles for the formation of categories. The first is that the main goal of forming categories is to provide the maximum information with the least cognitive effort. The second is that this maximum is achieved when the categories formed map the perceived structure of the world as closely as possible. Rosch also notes that context affects how we categorize objects in categories.

A large part of Rosch's work is dedicated to investigating and explaining the role that prototypes (more commonly referred to as stereotypes) play in our similarity and categorization judgments. She motivates the emergence of prototypes to describe categories of objects as the need to increase the distinctiveness between different categories (Rosch 1978, [33]). This means that we use prototypes when we reason about classes of objects because when we compare two prototypes the distinction between them is clear cut - in other words, prototypes are far enough from the boundaries of the categories to create no confusion. Furthermore, she observes that people rate prototypes reliably: "subjects overwhelmingly agree in their judgments of how good an example or clear a case members are of a category, even for categories about whose boundaries they disagree" (Rosch 1978, [33]).

From her experiments, Rosch synthesizes three major conclusions about the roles of prototypes: (1) prototypes don't impose any sort of processing model for categories, but they do pose a constraint on the processing, namely that process models should not be inconsistent with

the facts we know about prototypes; (2) prototypes don't constitute a theory of representation of categories, but the facts we know about prototypes can constrain the models of representation; and (3) prototypes don't constitute any particular theory of category, although they must be learned if a system is to have adequate knowledge of the world.

(Lakoff 1987, [21]) cites Rosch's work on prototypes in his own inquiry on categorization. He observes that we humans use categorization all the time, for example in recognizing the objects around us, and in speaking even the simplest phrases. Therefore, he argues that understanding the principles behind categorization is crucial for understanding the nature of man: "An understanding of how we categorize is central to any understanding of how we think and how we function, and therefore central to an understanding of what makes us human" (Lakoff 1987, [21]).

(Anderson 1991, [4]) points out yet another advantage of categorization, namely that knowing that an object belongs to a category means we can predict a lot about the properties and behavior of that object.

Last but not least, Simon Ullman's work on using intermediate complexity features in image recognition and classification (Ullman et al. 2002, [35]) suggests a practical guideline for categorizing objects into classes: by focusing not on the most specific features of the object, nor on the most general, but rather on the objects' intermediate features. (Ullman et al. 2002, [35]) show that features of intermediate complexity are more informative for classification than the very simple or the very complex features.

The learning system I describe here draws on the research described above in the following ways:

- I design my learning system to use both similarity and category judgments in order to deduce properties of objects,
- I use a linear measure of similarity of objects, like the one described by (Tversky 1977, [44]), enhanced to weigh features differently, based on the context of the comparison; my system compares objects and uses their similarity to deduce what properties objects have,
- I use intermediate complexity properties of objects to describe objects and to infer new properties, and

- I use comparison of objects with prototypes to determine whether objects belong to a class, and to further infer what properties they may have in virtue of belonging to that class. I take Rosch's work (Rosch 1978, [33]) a step further, by investigating how well a learner can do by making inferences about particulars objects from prototypes.

2.3 The Problem of Representation

The very need for knowledge representation (KR) in areas such as problem solving, natural language understanding, vision and many more, has generated much controversy among scientists in the field of AI: is representation necessary for an intelligent machine, or can one manage just fine without it? Brooks has been a devoted proponent of the latter point of view (Brooks 1991, [8]) and argued that progress in the field has been held back by the search for the appropriate representation, and that one should strictly rely on interfacing to the real world, through perception and action. "Connectionist" and "situated" theories of learning also embrace this point of view. ([12]) Connectionism postulates that intelligent cognitive behavior can arise directly from neural-like mechanisms, such as neural nets, and thus bypasses the need for representation. Situatedness also bypasses representation by trying to make behavior emerge solely from interaction with the environment, à la (Brooks 1991, [8]).

On the other side of the bastion, researchers (Brachman 1990, [7], Davis, Shrobe, Szolovits 1993, [28], Minsky 1985, [24], [29]) have argued that knowledge representation acts as a fundamental abstraction of the world around and that achievement in artificial intelligence is largely due to the computers' ability to represent knowledge internally. For example, (Brachman 1990, [7]) points out that KR allows computational systems to reason about their environments, their goals, and themselves. He cites Minsky's work on the frames representation (Minsky 1985, [24]) as a promise that using representations to reason about the world will have a major influence on artificial systems.

But even among the promoters of KR, the question of just what role representations play, and which representation is most appropriate for various machine learning tasks is still under extensive research.

(Davis, Shrobe, Szolovits 1993, [28]) clarified the role of KR by stating that a knowledge representation has five roles: (1) it serves as a surrogate for the real world, (2) it reflects a set of ontological commitments about how to see the world, namely which things the

representation pays attention to and which things it blurs from view, (3) it represents an embedded theory of intelligent reasoning, because when you choose a particular representation, you automatically commit to a particular theory about what it means to reason intelligently, (4) it acts as a medium for efficient computation, by defining what is easy (i.e. polynomial) and what is hard (i.e. exponential) to computer in this representation, and (5) it acts as a medium of expression, by prescribing what things are easy to express and what things are hard to express in that representation.

Based on work in KR, in this thesis I take the stand that having a representation, and, moreover, having the appropriate representation, is crucial for building an artificial learning system. The architecture presented here uses a representation first proposed by (Vaina and Greenblatt 1979, [19]). Vaina and Greenblatt were primarily interested in implementing a semantic memory. They aimed to study how the way that knowledge is arranged influences how people understand the world, how we solve problems, how we remember old facts, and how we learn new facts. Their model of human memory, called "thread memory" is especially well-suited for learning. It postulates a thread data structure, that is a multi-link, loop-free chain of semantic nodes. The semantic nodes represent the categories to which an object belongs. For example, a robin can be represented by a thread that looks like this:

```
living thing -> animal -> bird -> robin
```

The most important features of thread memory are:

- its capacity to learn, which (Vaina and Greenblatt 1979, [19]) illustrate with a running program that learns concepts using children's' books. The researchers assert that the thread memory model provides a better base for a learning system than other models,
- threads encode information in the order from the most general piece of information to the most specific; Vaina and Greenblatt argue that, in the context of classifying an object in the categories it belongs to, this organization makes information come out in the right order, because people think of the most general category of an object first, before thinking about more specific categories,
- thread memory has redundancy, in that information such as that a bird is an animal is presumably encoded on the threads of all

things which are birds. Namely all these threads contain the nodes `bird -> animal` Thus, by looking at any of these birds, one can see that a bird is an animal,

- the representation is easy to maintain, a property that has important implications for computational efficiency. For example, adding more knowledge to a system that uses this representation involves no modifications to the existing knowledge. They further point out that this property may simplify implementing context, and
- the thread memory representation is well suited to compare and contrast judgments between objects, and to recognition tasks.

Based on Vaina and Greenblatt's work, I design my learning system to use threads like the ones in the thread memory representation to record properties of objects. In my work, I describe the merits of the thread memory representation in terms of the five roles defined by Davis et al. (Davis, Shrobe, Szolovits 1993, [28]), and illustrate the usefulness of thread memory for implementing context and for supporting the operations of learning and question answering with context.

2.4 Knowledge Acquisition and Learning

In building a learning system, it is important to clearly define what kind of knowledge the learner is meant to acquire. This topic also has sparked debates in the field of artificial intelligence.

Several researchers have recently insisted that we should aim to build Systems with *commonsense knowledge* (Singh 2002, [40], [1], Lenat, Guha, Pittman, Pratt, Shepherd 1990, [39], Chklovski 2003, [9], Chklovski 2003, [10]). The most prominent such effort has been led by Minsky and Singh, who laid down the basis of the Open Mind Common Sense project (OMCS, [1]).

The motivation for this work is that computers lack commonsense knowledge about the world, and that they lack the ability to do commonsense reasoning, which means the ability to use commonsense knowledge to solve the kind of problems we encounter every day in our lives. Minsky and Singh ([1]) define commonsense knowledge to encompass common knowledge about the world (such as knowledge that if you're going to someone's birthday party, you should bring a gift), but also knowledge about how to use, combine, and classify the knowledge one has in order to solve certain problems.

OMCS has inspired the recent work of (Chklovski 2003, [10], Chklovski 2003, [9]) on acquiring commonsense knowledge from human contributors. Chklovski's goal was to capture the commonsense knowledge of non-expert contributors, as a step toward constructing a machine that could reason about the everyday world (Chklovski 2003, [10], Chklovski 2003, [9]). His method was to use volunteer contributors over the Internet to acquire assertions for his knowledge base (KB). This approach represented an innovation over previous approaches (Lenat, Guha, Pittman, Pratt, Shepherd 1990, [39], Guha, Lenat 1994, [30]) where knowledge was accumulated with an effort of many experts, over a long period of time. Chklovski also envisioned a repository of commonsense knowledge to be a crucial step toward breakthroughs in fields such as natural language processing and information retrieval. The seed for Chklovski's KB was taken from the OMCS KB.

Chklovski's system acquired its knowledge from assertions provided by volunteer contributors to the KB, and from its own algorithms to reason by analogy: given an assertion about a topic, the system guessed which other topics it already knew about were most similar to it, and proceeded to ask questions about the new topic, based on its similarity with these other, known topics. Chklovski's learning system exhibited bootstrapping qualities, in the sense that the more knowledge it had, the more knowledge it could base its questions on.

One of the main contributions of Chklovski's work, and directly relevant to the work I present here, is his discovery that the commonsense knowledge volunteered by online contributors falls nicely into thirteen classes. Chklovski notes that there currently is not universally agreed upon classification for assertions. Chklovski's classification, reported in his PhD thesis (Chklovski 2003, [10]), contains the following classes of knowledge, in the order from the most populated to the least populated:

1. ACTION, as in "Birds fly", accounted for 30.5% of all assertions,
2. Qualified-ACTION, as in "Planes fly very fast", accounted for 17.2% of all assertions,
3. PROPERTY, as in "A swan is white", accounted for 9.7% of all assertions,
4. ISA, as in "A bird is an animal", accounted for 9.0% of all assertions,
5. ACTION-ON, as in "Horses can be ridden", accounted for 6.6% of all assertions,

6. FUNCTION, as in "Planes are used for transportation", accounted for 5.0% of all assertions,

7. PART-OF, as in "A wings is part of a bird", accounted for 3.3% of all assertions,

8. Qualified-ISA, as in "Swans are white birds", accounted for 2.7% of all assertions,

9. REQUIRES, as in "Flying requires wings", accounted for 2.3% of all assertions,

10. DEFINITION, as in "Phones are devices for making calls", accounted for 1.9% of all assertions,

11. COMPARATIVE, as in "Horses are faster than people", accounted for 1.5% of all assertions,

12. MADE-OF, as in "A bird is made of feathers", accounted for 0.8% of all assertions, and

13. POSSIBLE-STATE, as in "Birds can be flying", accounted for 0.4% of all assertions.

Chklovski argues that this classification suggests that, in order to acquire a significant sample of the possible assertions about objects and their properties, contributors should be queried for knowledge that falls into all of the strongly populated classes enumerated above.

While evaluating his system, Chklovski points out that two important limitations (Chklovski 2003, [10]). One is due to the fact that his parser accepts only assertions in the form of single sentences which are syntactically valid and interpretable in isolation from other sentences or phrases. Furthermore, his system's goal is to handle knowledge about classes of objects, rather than about particular objects. These factors constrain the kinds of assertions the system can be given, and subsequently the kinds of knowledge it has about the world. The second limitation is due to the simplicity of the internal representations used. Chklovski argues that if more, or more complex representations were used, more kinds of things could be learned. For example, some important kinds of commonsense knowledge that Chklovski's system does not capture are knowledge about causes and about the motives of actions taken by people (as exemplified by the sentence"A person stands on a chair to change a light bulb"), or about how knowledge can be combined (such as reasoning from "Birds fly" and "Dead things cannot

fly" to "Dead birds cannot fly").

Chklovski's work profoundly guided the design of my system, as I describe at the end of this section. However, several other research directions in the area of learning and knowledge acquisition also deserve brief mention here.

In his work on the Cyc system (Lenat, Guha, Pittman, Pratt, Shepherd 1990, [39], Guha, Lenat 1994, [30]), Lenat suggests that there is a fundamental difference between humans and computers, in terms of both knowledge content and in how knowledge is used. He notes that humans are equipped to deal with new and perhaps unexpected situations as they arise, whereas computers cannot dynamically adjust to a new situation, when it exceeds their limitations. For example, humans can find more than one way to solve the same problem, they can ask for advice, they can read about the problem, or, when all else fails, they can fall back on commonsense. Lenat believes that it is precisely this difference that accounts for humans' superior intelligence.

Yet other authors such as Valiant (Valiant 1984, [45]) have looked at machine learning from a theoretical perspective. Valiant proposed a model of learning with three properties, namely that (1) learning is probabilistic, that is the hypothesis data is generated randomly; (2) learning is computationally efficient, that is at most polynomial in n, the number of observations that serve as input data, and (3) the learning algorithm is appropriately general, that is it can generalize well given a reasonably large input set. Valiant further demonstrated that it is possible indeed to build a system with all three characteristics above.

(Pitt, Valiant 1988, [20]) have also been concerned with the computational limitations of learning, in particular for learning from examples by way of generalization. The basic idea of learning from examples was described by Winston (Winston 1993, [48]). It consists of forming a description of a category of objects based on examples known to belong to that class; as more examples are presented to the system, the system expands its description of the category. In 1988, Pitt and Valiant (Pitt, Valiant 1988, [20]) showed that learning from example is computationally feasible, in the sense that it is possible to learn characterizable classes of concepts, where the classes are non-trivial for general purpose knowledge, and the computational processes that acquire these classes require a polynomial number of steps. Pitt and Valiant further point out that there may be variations in meaning in populations of

learners that use generalization.

My learning system draws on the previous work described above in the following ways:

- My system aims to accumulate commonsense knowledge about objects and their properties, such as knowledge that birds are animals, and that birds can fly.
- My system has several methods that it can use to answer a query posed by the human user, so it has several ways to "solve a problem" or to "deal with an unexpected situation".
- The knowledge my system accumulates about objects falls into seven of Chklovski's categories. Ultimately, one of my goals is to evaluate how well my system can reason and learn about the world, by using these classes of knowledge.
- My system, like Chklovski's, exhibits bootstrapping properties: the more knowledge it has, the more knowledge it can use to answer queries. Furthermore, once it deduces an answer to a question posed by the user, the system immediately adds the new information to the knowledge base, for further use.
- Following Chklovski, I analyze my system's performance in light of the limitations imposed by its language parser. The parser I use can take in single sentences or phrases conjoined by "and then". It can also handle statements about particular objects, not just classes of objects. Still, the sentences it can parser are still meant to be interpretable without much context, and this biases somewhat the kinds of things the learner can learn about the world. I discuss this in more detail in the chapter 5.

2.5 The Bridge Project

My learning system also grounds in work on the Bridge project [46]. The Bridge project is a project developed in the Genesis group at the MIT Computer Science and Artificial Intelligence Laboratory, under the guidance of Professor Patrick Winston. The main goal of Bridge [46] is to understand the computational nature of intelligence, and particularly the way in which the different faculties (vision, motor, linguistic) cooperate to facilitate human intelligence. The learning system I describe in this thesis is designed and implemented on top of the Bridge

infrastructure. I therefore describe the relevant Bridge representations here, and explain how they are used in my system.

This Bridge infrastructure [13] uses one main abstraction to represent the world, called a Thing. This abstraction actually comes in three flavors: a Thing, a Relation, and a Sequence. Bridge provides modules that implement each of these abstractions. Here is a brief description of the role of each.

- A Thing represents an object in the world, for example a bird. A Thing is made of a bundle of Threads. The Bridge Thread structure is an implementation of the thread structure for the thread memory model, described in (Vaina and Greenblatt 1979, [19]). In my learning system, each Thing is equipped at creation with a bundle of seven Threads, where each Thread stores a specific type of knowledge. The types of knowledge are those listed in table 3.1. Each thread has a thread label that identifies the type of knowledge stored on it. The thread labels are listed in the first column of table 3.1. Figure 2.1 exemplifies the structure of a Thing; that particular Thing instance represents a robin, for which the system knows some *isa*, *can*, and *has* facts.

- A Relation inherits from a Thing. It indicates how an object is related to another. In the context of language understanding, it indicates how the subject of a sentence (for example, the bird object from the sentence "A bird flew to a tree") relates to the object (in this case, the tree object.) In the learning system, there is a Relation for each sentence the system receives as input.

- A Sequence also inherits from a Thing. It is a structure which can contain an arbitrary number of elements. The Bridge parser usually creates Sequences of Relations for sentences with several phrases. For example, given the input sentence "A bird flew to a tree and then it flew to the lake", the parser builds a Sequence of two Relations, for each phrase of the sentence.

An important property of the Bridge representation is its focus on transitions. Bridge "views" the world in terms of objects moving along paths (or trajectories), a representation initially proposed by (Jackendoff 1983, [16]). Trajectories emphasize the movements of objects in the physical world. To make this more concrete, a Sequence represents a path, which tells how an object (usually the subject of a sentence) moves in time. The natural language parser the learning system uses

Figure 2.1: An example of a Thing structure.

is also part of the Bridge system. This parser accepts simple English sentences as input and outputs Sequences representing Jackendoff trajectories.

In the chapter 3, I discuss at length the assumptions and biases the Bridge infrastructure (specifically the Thread, Thing, and trajectory representations and the parser limitations) imposes on the learning system.

Chapter 3

System Design

"A designer knows that he has achieved perfection not when there is nothing left to add, but when there is nothing left to take away." (Antoine de Saint-Exupéry)

This chapter describes the design of the learning system. I first describe three fundamental ideas that guided my work. Then, I describe the system's specifications and discuss the representations the system uses and their implications for the learning the system aims to do. The bulk of this chapter is dedicated to describing the conceptual design and the software design of the learning and reasoning algorithms.

I designed the learning system I describe in this thesis with three fundamental ideas in mind. I describe these here, before delving into the details of the design, and I comment briefly on how the system design incorporates each of these ideas.

The first idea is that part of the thinking and learning that go on in an intelligent system stem from communication with the outside world. In common terms, this asserts that speaking with others makes us smarter, because it forces us to use our language apparatus together with our other cognitive and reasoning facilities. (I believe this idea was first formulated by my advisor, Patrick Winston. It has surfaced numerous times in the conversations we have had about the role of language, vision, and motor abilities in intelligence.) Speaking with our peers also makes us learn new facts, and leads us to think up new ideas that we may not have come up with on our own. The current learning system can respond to queries posed by a human user in English. It reflects the idea that communication contributes to intelligence in the way in which it responds to these queries. Responding to a query

involves a process of deducing an answer (which is the most accurate answer the system can give based on the contents of its knowledge base), and then learning a new property or classification of the object from that answer. Thus communicating with a human user usually results in learning a new fact about the world.

The second idea is that you can't learn anything unless you almost know it already. Credit belongs to William Martin. This idea suggests that learning is incremental, and happens as small leaps of inference from things that we know to things we don't know but which can be easily guessed. The current learning system does just this: it looks at objects it already knows about and tries to deduce properties for objects it knows less about, but which are, nonetheless, similar to the objects it knows.

The third idea is that in order to learn and to be able to conduct an intelligent conversation, one needs to have a notion of context. As argued in the previous chapter, context is all around us. Everything we do, say, and think is based on some context, though we're seldom explicitly aware of it. The main goal of this work was to carry over the notion of context to the machine learning system that I describe here. Section 3.5 below describes in detail how I represent context in this system, and how I use context in both learning and question answering.

3.1 System Specifications

I present here the specifications of the learning system that pertain to the system's interface, the learning it does (namely, what kinds of things it learns), and its question answering capabilities (namely, what kinds of questions it can answer).

First, I'd like to clarify what this system does and, most importantly, what it *does not* focus on. The main focus is on the learning and question answering algorithms. The system is an incremental learner, in the sense described by (Devaney and Ram 1996, [23]). It receives inputs one at a time, and refines its knowledge of the world dynamically, with each input received. In this work, the focus is *not* on natural language parsing. For this reason, the language that the system can understand is fairly limited. For input sentences, the system uses the Bridge parser discussed in chapter 2. For queries, this system uses a very simple and specialized parser. This parser recognizes only the kinds of queries I describe in the paragraphs below. The syntax of these queries is simplified particularly because this parser is just a

utility, it is not one of the main focuses of my research. In chapter 5 I argue that the learning architecture I have built is general enough that the language parsing limitations do not impose a hard cap on what the system can learn.

Here are the specifications for the system's interface, learning capabilities, and question answering capabilities. The system interacts with the human user by taking as input simple English sentences. These sentences are either about particular objects, as in "The blue bird flew to the tree" or about classes of objects, as in "Birds can fly." Out of these sentences, the system constructs its store of knowledge. The system also accepts and answers queries posed in simple English, for example "Do birds fly?"

The system learns seven kinds of knowledge about objects. As discussed in chapter 2, I have lifted these kinds of knowledge from Tim Chklovski's recent work (Chklovski 2003, [10], Chklovski 2003, [9]) on acquiring commonsense knowledge from human contributors, by way of an online acquisition system. From among Chklovski's fourteen classes of knowledge about objects, I have picked the seven classes pictured in table 3.1 below.

Type of knowledge	**Example sentence**	**% in Chklovski's classification**
action	Birds fly.	47.7%
property	Robins are small.	9.7%
isa	A robin is a bird.	9.0%
has	A bird includes wings.	3.3%
can	Birds can fly.	5.0%
requires	Birds require wings to fly.	2.3%
madeof	Birds are made of feathers.	0.8%

Table 3.1: The classes of knowledge Squirrel learns.

The *action* class of knowledge in table 3.1 includes Chklovski's ACTION and QUALIFIED-ACTION classes. The *isa* class includes Chklovski's ISA and QUALIFIED-ISA classes. The *has* class of knowledge is a reformulation of Chklovski's PART-OF class. The *can* class is a generalization of Chklovski's FUNCTION class.

Together, the seven classes of knowledge in table 3.1 encompass

≈87% of all assertions provided to Chklovski's knowledge acquisition system. I have chosen to focus specifically on these seven classes in my work because from these classes, my system can acquire a representative sample of the common knowledge that people have about objects in the world.

The system can answer four different kinds of questions, as shown in table 3.2 below. Note that the first two types of questions ask directly about what properties an object has. These properties can refer to any of the seven kinds of knowledge from table 3.1. For example, the "what is X" category of questions includes all of the following: "what is a bird?", "what can a bird do?", and "what is a bird made of?"

Question type	**Example input to Squirrel**	**Purpose**
what is X	What is a bird?	deduce and learn new fact about an object
does X do Y	Can planes fly?	deduce and learn new fact about an object
is X similar to Y	Is a plane similar to a bird?	compare objects to learn a new fact about one of them
describe X	Describe birds.	synthesize known facts about an object

Table 3.2: The kinds of questions Squirrel can answer.

The questions the system aims to answer are meant to make the system deduce information about classification of objects (this information falls into the *isa* knowledge class), and to deduce properties of objects (which fall into one of the *action*, *property*, *has*, *can*, *requires*, *madeof*).

I have chosen the queries above to demonstrate the usefulness of this learning architecture. I believe that a system that can efficiently make these kinds of deductions and judgments about objects can be extended to reason about objects and their properties in a broader sense.

3.2 The Representations and Implicit Assumptions

Researchers have argued that the representation of the world that an artificial intelligence system uses has crucial implications for the kind of reasoning the system can perform (Davis, Shrobe, Szolovits 1993, [28]). Thus, before describing the design of the learner, I point out the implicit assumptions that result out of the representations my system uses.

The main representations, presented in chapter 2, are: the Thread, a module which is part of the Bridge system, ([13]) and implements a thread as described by (Vaina and Greenblatt, [19]), the Thing, Relation, and Sequence modules which are part of the Bridge system ([13]), and the trajectory representation ([16]), output by the Bridge parser.

The Thread and Thing abstractions, the focus on trajectories of objects as a way to understand the world, and the seven classes of knowledge the system is designed to learn have profound implications for the system.

First, these representations prescribe in what terms the system views the world. The system views each object in terms of the seven types of knowledge described in table 3.1, namely what the object does, what properties it has, what classes of objects it belongs to, what parts it has, what its capabilities are, what it requires, and what it is made of. These types of knowledge capture some but not all the knowledge people have about objects; there are some kinds of knowledge that the current system does not capture and therefore cannot reason about. I will argue later, during the evaluation of the system, that the design is modular and extensible enough to support broader purpose learning.

Second, the use of Threads is important because of the four main properties of the thread memory representation (Vaina and Greenblatt 1979, [19]): redundancy, storing category information in the order from the most general category an object belongs to down to the most specific, ease of maintaining a knowledge base expressed as thread memory, and the suitability of threads for compare and contrast operations. These properties shape the computational environment on top of which the learning and reasoning algorithms are built. The redundancy of information on things' *isa* threads eases the computation the system has to do. For example, consider all the objects the system knows about which are birds. Then, the information that a bird is an animal is stored on most of these Things, in the form `animal -> bird`. Thus, the system can access this information by looking at any of these Things, and

needs not do any involved computation to search for the information.

Third, the fact that the system's view of the world is focused on the movement of objects along trajectories has implications for the kinds of things it is easy to express, and maybe more importantly for the kinds of things it is hard to express in the system. For example, it is especially hard to capture the notion of time. There is no easy way to express how objects and their properties change with time, so the system doesn't learn to take time into account.

Fourth, the representation implies an underlying model of intelligent reasoning. Specifically, the system assumes that humans use certain methods to learn and make judgments about the properties of objects, and it aims to explore how an artificial learner could use these same methods. For example, I assume that humans use categorization of objects and similarity of objects to infer object properties. Consequently, the learning and question answering algorithms I describe in section 3.5 make use of exactly these principles.

Finally, an important fact to keep in mind about the Thing representation is that the system I describe here takes this representation for granted. That is, this work does not focus on exploring how it is that representations can arise in a system, from interaction with the world. This is indeed a crucial problem in artificial intelligence in general, and in the area of knowledge representation in particular. However, this problem is outside the scope of this work. This work rather assumes the Thing representation is available to the system, and focuses on showing what the system can learn about and reason about by using it.

Before moving on, here are brief explanations of several terms which I will use often throughout the remainder of this thesis:

- a *type* is the name of a category of objects, or something that represents a property of an object. In Squirrel, types are recorded on the threads of Thing structures. For example: *bird* is a classification which may be recorded on the *isa* thread of an object which is a robin. Another example is *fly*, which is a capability of robins, so it may be recorded on the *can* thread of an object which is a robin.

- *Thing* refers to the data structure I've described above and which represents an object in the world. When thing appears spelled all in lowercase, it should be interpreted as synonym for object.

- *Event* and *event* are wrappers around Relations from a software perspective, and represent events in the world (such as "A bird

flew to a tree") from a conceptual perspective.

- I use *stereotype* to refer to what Rosch calls a prototype (Rosch 1978 [33]), that is, an object which is representative of an entire class of objects. For example, I refer to the stereotypical bird as an object that has all the generic properties that we associate with birds, such as the fact that a bird is an animal, that it has wings, and that it can fly.

3.3 Design Overview

The learning system I have built has three main capabilities: (1) building a knowledge base, by recording information from input sentences and query answers, (2) deducing answers to queries, by inspecting the knowledge base, and (3) learning new facts from input sentences and query answers. The system learns from input sentences and query answers by recording pieces of information it extracts from these sentences or answers in a memory object. Learning from query answers effectively acts as a cache, in that the piece of information in the answer can later be used without having to be deduced again. The first capability of the system, that of building a knowledge base, is a side effect of the learning algorithm.

The main pieces that accomplish the three system capabilities are: the interaction between the three main modules in the system-the learning module, `Learner`, the memory module, `Memory`, and the natural language utility, `ParsingEngine`), the use of context, and the methods for reasoning about objects by using similarity and categorization.

I first describe the system modules and their interaction. Figure 3.1 represents a Module Dependency Diagram (MDD) for the learning system.

The most important dependencies are those of the `Learner`, `Memory`, and `ParsingEngine` modules. The diagram shows that the `Learner` depends on the `ParsingEngine`, `Memory`, `Thing`, `Event`, `Context`, and `ThingFactory`. The `ParsingEngine` depends on the parsing modules in the Bridge infrastructure, and on the `Event` and `ThingFactory` modules. The `Memory` module depends on `Thing` module.

The `Learner`, `Memory`, and `ParsingEngine` modules encompass the learning, storing, and language parsing capabilities of the learning system. The objects in the world are represented by `Thing` structures. A Thing contains several `Threads`, which implement the thread structure described in (Vaina and Greenblatt 1979, [19]). The `Thing` and `Thread`

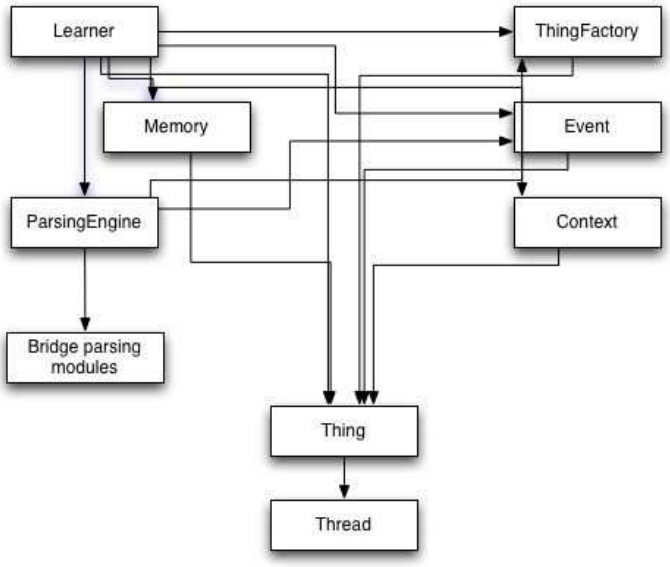

Figure 3.1: An MDD showing the main modules of the learning system.

modules are part of the Bridge infrastructure ([13]), described in chapter 2. Nearly all modules use `Thing` structures, because they are the main abstraction of the world that exists in the system. The following is a description of the roles of the main modules in the system.

- The `Learner` and `ParsingEngine` modules call on the `ThingFactory` module to produce Thing structures. For example, the `ParsingEngine` produces Things that correspond to the objects in a given input sentence.
- The `ParsingEngine` uses a parser module from the Bridge infrastructure ([13]) to parse input sentences into `Event` structures. The `Events` become input to the `Learner` module. The `Learner` extracts the Things representing the sentence subject, verb, and object from the `Event`, and stores them in the `Memory` module.
- The `Context` module represents the notion of context in learning and thinking about the world. The `Learner` manipulates the `Context`. It uses it and modifies it during learning and question answering. Section 3.4 explains in detail what the role of the context is, and how it is used in the system.

Figure 3.2 highlights the functions the systems' modules serve. The figure reveals that the system is nicely layered. At the bottom level, the `Thing` and `Thread` structures encompass the representation of the world. At the second level, the `ParsingEngine` encompasses natural language processing. This level is necessary because the system receives input in the form of simple English sentences. The system also "talks to itself" using English sentences, as will be explained shortly. The third level represents the system's memory. The fourth level represents the cognitive capabilities of the system, namely the learning. The main merit of figure 3.2 is that it can be interpreted as the architecture for a broader-purpose learner and reasoner about the world.

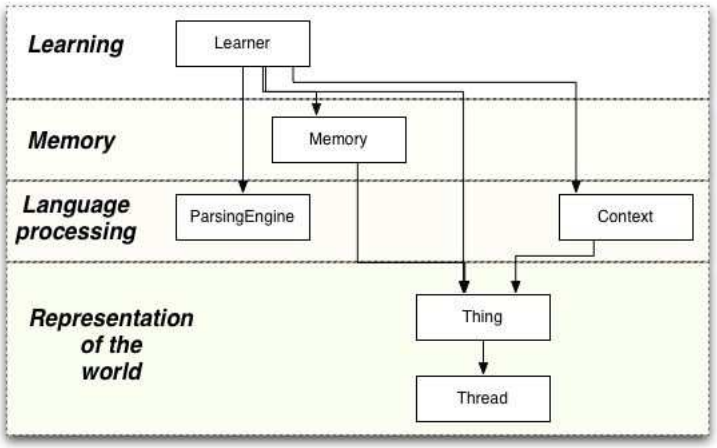

Figure 3.2: An MDD showing the layering of the learning system, by module function.

The learning and question answering algorithms of the system are built on top of the basic modules described above. The next two sections discuss the representation and use of context in the system, and the algorithms that deduce information about objects in order to respond to user queries about the system's knowledge.

3.4 Using Context for Learning and Question Answering

As humans, we use context in virtually all of our everyday activities. We use it while speaking to each other and while learning. Most of the

time, we use context unconsciously, because we are unaware of how our minds supply it. Sometimes, we are instructed to use a specific context, for example in stories which start with "Once upon a time, in a land far, far away, ..." At other times, our minds supply a default context, which consists of our commonsense knowledge about the world. But just what does the notion of 'context' refer to?

I define context (informally) in my system as temporal proximity of information. What this means is best illustrated by a few examples. Suppose two people carry on a conversation about robins. They talk about the color of their wings, how fast they fly, and whether their legs are long or short. Then, the context of the conversation contains the information most recently exchanged between the speakers. In this case, the context includes robins (the object which is the subject of the conversation), as well as robins' properties (the color of their wings, plus other properties we know robins have), their capabilities (such as fly, walk, etc), and the parts robins are made of (such as their legs). For each speaker, the context might also include memories of particular robins the speaker has seen at some previous time. The main idea is that the conversation about robins primes the speakers to think about robins and their properties, and may also influence how the speakers reason. For example, if speaker A asks speaker B what sparrows are like, then B will likely describe sparrows in comparison to robins, by pointing out the similarities and differences between sparrows and robins. This scenario suggests that capturing the context of a conversation may help an artificial system reason more intelligently about the world.

Here is how I incorporate the use of context in my system. The 'conversation' in this case consists of the human user giving the system input sentences and input questions. The idea is to: build up and modify context based on the input sentences, to answer queries by bootstrapping from the current context, and to modify the context after deducing answers to the queries posed.

I represent context as a container with four elements: a type, a fixed-length queue of thread labels, a Thing instance, and a Relation instance. The type is a category of objects, such as *robin* in the scenario above. With the type, I mean to capture the subject of the conversation, in other words the thing the speakers talk about. The thread labels are the names of the seven threads I use in my system, previously described in table 3.1. These labels are: *action*, *property*, *has*, *can*, *requires*, *madeof*, *isa*. The context contains a queue of fixed length of labels. The length is set to 3 in the implementation I describe in the next chapter, but can be changed. With these labels, I

mean to capture the kinds of object properties the speakers talk about, such as the subject's capabilities, or the parts it is made of, or the categories of objects it belongs to. This queue is also important because it allows the system to describe objects in terms of the threads whose labels are in context. This idea is based on the assumption that the information most relevant to the speakers is the information most recently exchanged between them. Describing things in different ways depending on the threads in context makes the learning system an attribute-incremental system, as described by (Devaney and Ram 1996, [23]). Figure 3.3 shows a picture of a context object.

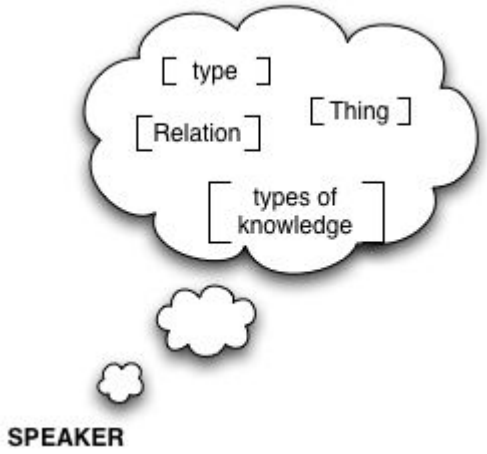

Figure 3.3: A pictorial representation of the Context object.

From a software engineering point of view, the context is a singleton. The `Learner` and `Memory` modules change the `Context`, based on the operations they perform. Conceptually, the `Learner` changes the context while processing an input sentence, and while deducing an answer to a user query. The `Memory` changes the context while "reflecting" on the knowledge base (KB). (I describe reflection in the next section.) The following pseudocode illustrates how context changes.

```
input sentence -> Learner.learn(){
    extract things from Event;
    modify and store things in Memory;
    change context;
}
query to system -> Learner.answer(){
    formulate answer based on context;
    modify context;
    answer -> Learner.learn();
}
Memory.reflect(){
    decide what to think about based on context;
    deduce some new information;
    change context;
    new information -> Learner.learn();
}
```

The system is designed to change the context as follows:

- When learning from an input sentence, if the sentence is an assertion about a class of objects, as in "Birds can fly", the `Learner` sets the context type to the name of that class, (here, *bird*). Otherwise, if the sentence is an assertion about a particular object, as in "A bird flew to the tree", the `Learner` sets the context type to the most narrow category that it knows the object belongs to. For example, if the bird in the previous sentence was known to be a robin, the context type would be *robin*; otherwise, it would be *bird*. I treat *isa* assertions separately from the other six types of assertions: in *isa* assertions, I set the context type to be the syntactic object of the sentence. For example, given "A bird is an animal", the `Learner` sets the context type to *animal*.
  The `Learner` further sets the Thing in context to point to the Thing structure which represents the subject of the sentence, and sets the Relation in context to point to the Relation structure which corresponds to the given sentence.
  The `Learner` also pushes the thread label corresponding to the given sentence on the queue of thread labels in context. So for example, given the sentence "A bird is an animal" the `Learner` pushes *isa* on the queue.

- When the system has to deduce an answer to a query, it tries several methods. The `Learner` changes the context depending on

which method was used. I explain this in more detail below, in the section that describes the model of learning.

- After the system deduces the answer to a query, it expresses the answer as an English sentence. The `Learner` changes the context as if this sentence was an input sentence, so it follows the guideline described above.

I argue here informally that the way in which the `Learner` builds and changes the context is useful. I defer a full discussion and evaluation of the system's learning performance until chapter 5. Changing the context according to the guidelines above, in the case of input sentences, seems to be useful because it models exactly the simple conversation scenario I described at the beginning of this section: after each sentence is processed, the context contains the subject of the sentence, it contains the kind of information about the subject that was expressed in the sentence, and it contains the relation corresponding to the sentence. The goal of my system is to explore this use of context and to illustrate how it improves concept formation about objects and their properties, by way of similarity and categorization judgments.

I now go on to describe the design of the learning and question answering algorithms, all of which make use of and change the information in context.

3.5 The Conceptual Model for Learning

The learning is localized in the `Learner` module. When the system receives as input an English sentence, the `Learner` sends it to the `ParsingEngine` for parsing. The `ParsingEngine` returns an `Event` object, which is a wrapper around the Relation corresponding to the sentence. The `Learner` extracts the subject, the verb, and the object of the sentence from the `Event`. It then learns by recording the property predicated of the subject on one of the subject Thing's threads. For example, given the sentence "Birds are animals", the `Learner` stores the type *animal* on the thread labeled *isa* of the Thing structure for *bird*. Once the `Learner` has recorded this information on the subject's thread, it stores the subject in the system's memory (by making a call to the `Memory` module). The last action the `Learner` does is to change the context object, according to the guidelines described above.

The `Learner` also learns some information about the verbs in the sentence [1]. The system records Thing structures which represent verbs,

[1] Squirrel, the implementation described in chapter 4 does not use the knowledge

and modifies their threads in the same way as it does for Things that represent objects. A verb has three threads, labeled *isa*, *requires*, and *example_of*. Given the sentence "A bird flew to the tree," the system adds the type *bird* to the *example_of* thread of the Thing representing the verb fly.

Figure 3.4 shows the conceptual steps in processing an input sentence. The steps are numbered 1 through 6, indicating the order in which the relevant operations are performed.

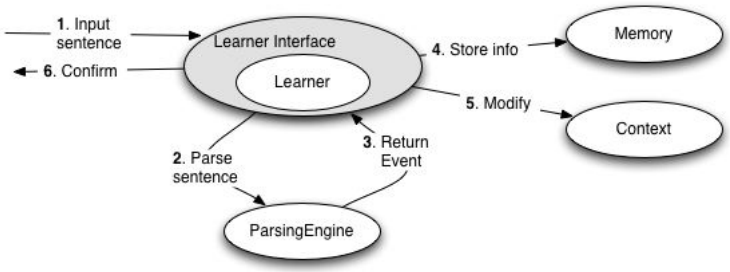

Figure 3.4: The steps involved in processing an input sentence.

One important observation is the following. The system can accept sentences about both particular objects and about classes of objects. A sentence such as "A bird flew to a tree" is interpreted to be an assertion about a particular bird. As a result, once the `Learner` gets the event from the `ParsingEngine`, it builds a new Thing structure to represent this bird. It then adds a trajectory representing moving to a tree to this bird's *action* thread, and stores the Thing structure in `Memory`. On the other hand, sentences such as "Birds are animals" are interpreted as assertions about whole classes of objects. The system keeps one Thing structure for each class of objects. Thus, when a sentence such as "Birds are animals" comes in, the system first tries to retrieve the general bird Thing from its memory. If no such objects exists, then it creates a new Thing for this class. Here is a piece of pseudo-code for the `learn()` method of the `Learner` module:

```
Learner.learn(event){
  Thing subject = event.getSubject();
```

about verbs for reasoning about objects' properties. The system is designed with the infrastructure already in place for this, so adding reasoning about verbs to the system is easy. Chapter 5 discusses the merits of reasoning from knowledge about verbs.

```
Thing object = event.getObject();
Relation rel = event.getRelation();
Thing t = null;
String type = null;
if (isGenericStatement(event)){
  t = memory.getGenericThing(subject.getName());
  type = getClassName(subject);
}else type = getMostNarrowClass(subject);
  t.addType(object, thread_label);
  memory.store(t);
  memory.store(rel);
  context.setParameters(type, thread_label, t, rel);
}
```

When the system receives a query, such as "Can robins fly?" it tries to reason about it based on the information currently available in its memory (KB), and deduce an answer. The system reasons by using categorization and similarity. The idea is to try to deduce a property of an object by looking at what classes of objects it belongs to (reasoning from categorization), or by looking at other objects similar to it, and about which the system knows more information (reasoning from similarity). This design is based on an implicit assumption that (Gentner and Markman 1990, [11]) point out, namely: when reasoning by similarity, we assume that if two things are similar in some ways, then they are also similar in other ways. This is exactly the assumption my system makes–when presented with an object and a question about whether the object has some property, one of the ways it tries to reason about it is by finding another object which is similar to the given object, and seeing whether that other object has the desired property. If the objects are similar, and if that other object does have the desired property, then the system concludes that the object of interest also has the property. These 'other' objects can be either particular Things that the system has learned about at some previous time, or stereotypes of a class of objects.

Before explaining the reasoning algorithms in detail, I explain the three fundamental ingredients: how the system assesses similarity of Things, how it represents and manipulates stereotypes, and the structure and role of the `Memory` module.

3.5.1 The Similarity Mechanism

The system compares Thing structures to see if they are similar using a measure of similarity à la (Tversky 1977, [44]). The measure of

similarity is a linear combination of the types the two Things have in common. More specifically, the system calculates a total similarity score for the two Things. The total similarity score is a weighted sum of the seven similarity scores, one for each pair of threads with the same thread label. Each pair of threads is given a default weight of 1, and the threads whose labels are in context are weighed more than the others by a factor of 2. (The values of these weights are adjustable. In chapter 5, I explain which parameters are changeable, and what can be accomplished by changing them.) For each pair of threads, the similarity score contains the count of the types that are on both threads. Thus, the formulas for the similarity score for a pair of threads and for the total similarity score can be written as shown below. In these equations, s is the total similarity score. t is the similarity score for a pair of threads with the same label. w_i is the weight of the pair i of threads.

$$s = Sum_{i=1\ to\ 7}(t_i \times w_i)$$
$$t = Sum_{i=1\ to\ number\ of\ shared\ types}(1)$$

Once the system calculates the total similarity score for the Things, it compares it against the total similarity score that would result if the two Things contained exactly the same types (that is, if both Things contained the union of their types), call it s_{max}. (If the thread weights were normalized, the maximum similarity score would be 1, but in this implementation it is more than 1.) At this point, there are three possibilities:

- If $s \geq .7 \times s_{max}$, the system considers that the two Things are similar.
- Else, if $s \leq .3 \times s_{max}$, the system considers that the two Things are dissimilar.
- Finally, if neither of the above is true, the system concludes that it 'doesn't know' whether the Things are similar. In other words, it doesn't have enough information to believe that they are similar.

This raw measure of similarity I use relies on the types that the Things have in common. The idea, already described above, is that if two Things have many properties in common, then they are likely to have other properties in common too, possibly exactly the one property we are interested in. Appendix A gives pseudocode for the methods that calculate similarity, as explained above.

It is important to comment here on the decision to weigh threads differently according to whether or not their labels are in context. Weighing the threads differently is important because it makes the threads in context (which represent the kind of information most recently received by the system) more relevant than the rest. Essentially, the effects of these weights is that when the system compares two Things, it 'cares more' about certain kinds of properties than others. Thus, the system can judge that two objects are similar in one context, yet dissimilar in another. Consider the following example. Suppose the system receives information about birds and fish, such as: "Birds can fly," "Birds have wings," "Birds are made of feathers," "Fish can swim," "Fish are made of scales." When asked whether birds are similar to fish, the system would answer no, because birds and fish are described to be made of different things, and they have different capabilities. Now suppose the system receives more information, such as: "Birds are animals," "Fish are animals," "Birds breathe," "Fish breathe." Now, when the system is asked again whether fish are similar to birds, it would think they are, because the more important properties to consider now are what kinds of things birds and fish are, and what actions they do. This example illustrates how weighing threads differently can essentially change the system's perspective of the world.

3.5.2 Stereotypes for Classes of Objects

The system represents stereotypes as Thing structures. The system differentiates between two kinds of properties of that a stereotypical object such as 'bird' has. The first type contains properties learned from assertions such as "Birds are animals"; the system knows this assertion is about a class of objects, so it considers that it holds true for all birds that it knows about. The second type contains properties learned from assertions such as "A bird flew to the tree." This assertion is about a particular bird, and so it does not automatically hold true for other birds. (Matwin and Kubat 1996, [34]) call the first type of properties *absolute properties*, that don't depend on any context or any other information. They call the second type *relative properties*, because the system attributes the second kind of properties to the stereotypical bird if and only if a majority of all the birds the system knows about have those properties.

The following example illustrates. Suppose the system knows about 10 particular birds, and that 9 of these have the type *wing* on their *has* thread. Suppose also that the 10th bird has no types on its *has* thread. Then, when asked whether a bird (meaning the stereotypical bird) has

wings, the system will conclude that it does, because a majority of the birds it knows about (9 out of 10) do indeed have wings. The process by which the system looks at all the birds and decides which properties belong to a majority of them, was called 'finding the thickest bundle', by (Vaina and Greenblatt 1979, [19]).

To summarize: a stereotype for a class of objects consists of the union of an absolute stereotype (represented by a Thing structure in `Memory` and which contains on its threads all the properties learned from assertions about that class of objects) and a relative stereotype (that is a Thing with all the properties that belong to a majority of the particular objects of that class.) The relative stereotype is calculated on the fly, so that the properties it includes change as the system receives more input. For example, at some time t, the system may know 4 birds that are gray colored and one that is blue. Suppose that between t and some later time t', the system learns about another 10 blue birds. Then, at t', there are 11 blue birds and 4 gray birds, for a total of 15. Then, by t', the concept of bird will have changed from birds being gray to being blue. The result of the union of the absolute stereotype with the relative stereotype is what (Matwin and Kubat 1996, [34]) call a *partially relative* concept. Figure 3.5 illustrates the stereotypes the system contains and how it forms the partially relative stereotype. Appendix A shows pseudocode for the methods that calculate the relative stereotype and the partially relative stereotype.

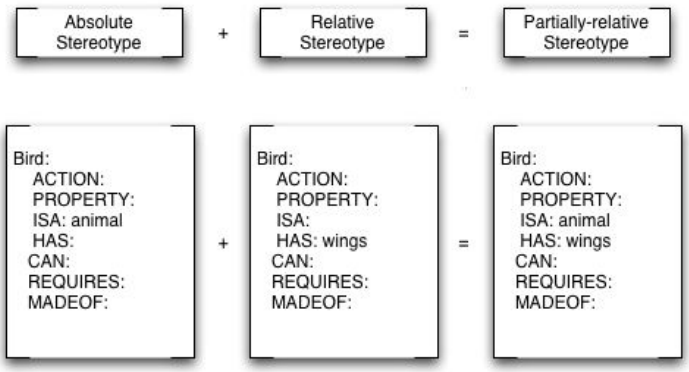

Figure 3.5: Calculating the stereotype of a class of objects from the absolute and relative stereotypes.

Throughout the rest of this thesis, I use 'stereotype' to refer to the partially relative stereotype.

3.5.3 The Memory Module

The main role of the `Memory` module is to store Thing instances and to provide an interface for updating the KB and for retrieving Things from the KB.

The `Memory` module is designed to contain two threads of operation [2]. In one thread, called the I/O thread, the `Memory` responds to the `Learner`'s requests to retrieve Things and to calculate stereotypes. This happens, for example, when the `Learner` wants to look at a stereotype to answer some question. In the second thread, called the reflection thread, the `Memory` runs a process of *thinking*, or *reflection on its knowledge*. This reflection consists of the following steps:

1. The thread picks a Thing or a class of objects to 'think about'. What is picked depends on the current context, so for example if the context type is *bird*, then the system is likely to pick the class of birds to 'think about'. 'Thinking about' a particular Thing or a class of objects means that the system tries to deduce a property of that Thing or class that it doesn't already know (i.e. a type that is not already recorded on any of the Thing's threads.) Conceptually, in this step the system sets a goal for itself (to deduce a new property of an object), and expresses that goal internally, as an English query.

2. In the second step, the system tries to produce an answer to the query in the same way that it deals with queries from the human user in the I/O thread.

3. In the third step, the system has deduced an answer, and proceeds to learn from this answer as if from an input sentence.

Appendix A contains pseudocode for another possible implementation of the reflection mechanism, that picks two random Things from the system's memory, compares them, and if they are similar, attributes a property of one of the Things to the other.

There are two important differences between the operations inside the I/O thread and the reflection thread: (1) the operation in the reflection thread is prompted by the system's *autonomously* picking something to think about; this turns the system into a rational agent, capable of setting goals for itself and carrying them out, and (2) whereas the I/O thread can be idle for long periods of time, the reflection thread

[2] Squirrel, the running instance of this learning system which I describe in chapter 4, does not implement the reflection thread. However, it implements the infrastructure on top of which the reflection thread can be easily added.

runs continuously. Thus, in effect, the system is always thinking about something and trying to consolidate its knowledge by deducing new properties of objects [3]. There is also an important similarity between the I/O thread and the reflection thread, and that is that both are primed by the current context: the system answers questions by using and modifying the information in context (as explained below), and it also picks the objects it thinks about based on the context. Figure 3.6 shows the threaded structure of `Memory`.

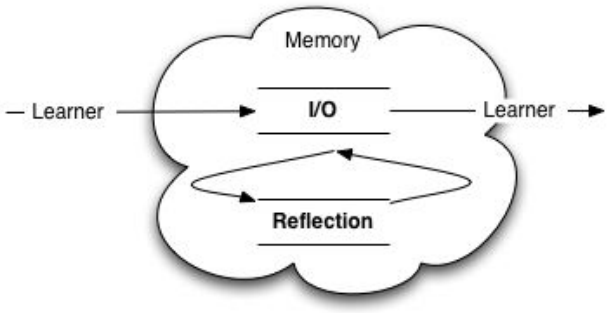

Figure 3.6: The threaded structure of the `Memory` module.

3.5.4 The Query Answering Methods

The main idea for the design of the query answering methods is to give the system several different 'ways to think' about the query, and thus several 'ways to deduce an answer' to the query. This idea has its seeds in the ideology of the Open Mind Commonsense Project [1], which argues that part of the commonsense knowledge that today's computers lack has to do with knowing different ways to attack and solve the same problem. A second important design decision was for the system to try to use the information in context to guide its reasoning about objects. Thus, the system design contains four methods for answering a query. Suppose the query is about a Thing t, for example, "What can t do?" or "Is t an animal?". Then, these methods are:

- Look at t's threads and see if they contain a type that can answer the question. This is the simplest thing to try; if the information

[3] Of course, there are a couple of technical details, such as the fact that if the system sees no input, then it doesn't really have anything to think about, and if it has very little input, then it may exhaust the things it can think about fairly quickly. These are low-level details that I do not discuss at length in this thesis.

asked for is already recorded on the t's threads, then the system has to look no further.

- Look at the stereotype for the type in context. (If the context type is *bird*, then look at the stereotypical bird.) See if t is similar to this stereotype. If it is, and if the stereotype has the property that the query asks about, then conclude that t has it also. (For example, given the query "Is t an animal?", if the stereotype of the type is similar to t, and is an animal, then conclude that t is also an animal.) The hope here is to answer the query by taking advantage of the context of the 'conversation'. Also, the hope is that, if t and the stereotype are similar in other respects, then they may also be similar with respect to the property the query is about.

- Look at the thing in context and apply the same similarity judgment as above: if the thing in context is similar to t, and if the thing in context has the property asked about, then conclude that t also has this property. Again, this method aims to take advantage of the context and of reasoning from similarity.

- Look at the types on t's *isa* thread, that is, at the categories of objects that t is known to belong to. Then, calculate the stereotypes of these types. If any of these stereotypes has the property in question, then conclude that t also has this property. The hope here is that t will have the property in virtue of belonging to the class, as a first approximation [4]. The main point here is that the learner first picks a type of intermediate complexity from the *isa* thread. This type is supposed to be a category that is not too general, nor too specific. For example, when thinking about robins, and intermediate category would be *bird*, as compared to *living_thing* which is too general, or *non-cavity_nesters* which is too specific. The hope is that the intermediate complexity category will yield the optimal information that will allow the system to answer the query.

In turn, the methods that assess the similarity between two objects (such as t and a stereotype), do so by trying one of three things:

- Look only at the types on the threads of the two Things, and calculate a similarity score for the Things according to the method described in the subsection on similarity above.

[4] Of course this is a gross judgment, which only takes into account the genus, not the specific differences. More elaborate algorithms would have to also assess the similarity of the two objects.

- Look at the stereotype of the context type. If both Things are similar to the stereotype, conclude that they are also similar to each other. Here, the similarity between each Thing and the stereotype is calculated by looking only at the Things' threads, as in the bullet above.

- Look at the Thing in context. If both Things are similar to it, then conclude they are also similar to each other.

The specification for the methods listed above says that the method either returns a positive answer (something like "t is an animal"), or returns a don't know answer (something like "I don't know enough about t to answer your question.") For each question received, the Squirrel implementation described in chapter 4 tries all of the methods in the order listed above. That is, it first tries to answer the question by looking only at t's threads. If this method succeeds to produce a positive answer, it stops, returns the answer to the user, and proceeds to learn from it. Otherwise, it goes on to the next method, and so on until the very last. If the last method produces a don't know answer also, the system returns this answer to the user and learns nothing from the answer. I choose to use this simple ordering of the methods because the overall focus of this project is a proof of concept that context can be designed and implemented meaningfully and usefully on top of the thread memory representation. Chapter 5 discusses the possibility of ordering the methods more intelligently, for example by choosing their priority based on the information in context. Such an ordering may lead to surprising improvements in the quality of the learning and reasoning the system performs.

Table 3.3 describes in more detail the specifications for each method and, most importantly, it shows how the learner changes the context, according to which method the system found successful. Table 3.4 gives the order in which the system tries the methods, for each type of query from table 3.2.

Method	**Sample positive answer**	**Change in context**
Deduction from t's thread	*t is an animal.*	Push the thread label onto the context queue. If answer is positive, set the context Thing to t.
Compare t with the stereotype of the type in context	*t is an animal because it is similar to a bird, and a bird is an animal.*	Push the thread label onto the context queue. If the answer is positive, set the context Thing to the stereotype.
Compare t with the Thing in context, call it x	*t is an animal because it is similar to x, and x is an animal.*	Push the thread label onto the context queue. If the answer is positive, if the Thing in context is a stereotype, set the context type to the stereotype's name; else set the context type to the most narrow category the Thing belongs to.
Deduction from the categories of objects t belongs to	*t is an animal because t is a bird, and a bird is an animal.*	Push the thread label onto the context queue. if the answer is positive, set the Thing in context to the stereotype that helped deduce the answer.

Table 3.3: The methods used to answer queries about objects' properties.

Query type	**Priority of deduction methods**
what is X	1. Look at X's threads, 2. Look at stereotype of type in context, 3. Look at Thing in context, 4. Look at X's categories.
does X do Y	1. Look at X's threads, 2. Look at stereotype of type in context, 3. Look at Thing in context, 4. Look at X's categories.
is X similar to Y	1. Assess similarity from X and Y's threads, 2. See if X and Y are similar to the stereotype of the type in context, 3. See if X and Y are similar to the Thing in context.
describe X	1. Look at X's threads.

Table 3.4: Priority of deduction methods for each kind of user query.

Chapter 4

Implementation and Results

"The best evidence for such a proposal is a working computer program; this is the only convincing way to show that a theory of learning is effective, complete, and applicable – not to mention its practical utility." (Lucia Vaina and Richard Greenblatt)

In this chapter, I present a program called Squirrel, that implements the learning architecture described in chapter 3. I describe here several implementation details about the backend and the user interface and show two examples from Squirrel's execution, which illustrate the learning and reasoning capabilities of the system.

4.1 The Squirrel Implementation

Squirrel implements the learning architecture described in chapter 3. The entire implementation was done in Java and Java Swing. Its backend consists of the `Learner`, `Memory`, `ParsingEngine`, `Event`, `Context`, and `ThingFactory` modules, plus a few other modules that interact with the Bridge infrastructure and implement helper functionality. Squirrel has a graphical user interface (GUI) that enables the human user to interact with the learning system. The next two subsections describe the backend and GUI in detail.

4.1.1 The Implementation of the Backend

Below is an explanation for the implementation of the most important backend modules. One of the merits of this implementation is that it allows for a runtime scenario where several `Learner`s are instantiated, each with its own `Memory`, `ParsingEngine`, and `Context`. These `Learner`s could be used for a Kirby-like experiment (Kirby 1998, [17]). I find this a very interesting project to explore. I comment on it briefly and offer some suggestions in chapter 6.

- The `Learner` class implements the learning and reasoning capabilities of the system. The `Learner` is the only module that the user directly interacts with (through the user interface). The `Learner` receives the user's inputs and dispatched to the other modules for computation; once the necessary computation is finished, the `Learner` returns a confirmation to the user, through the system's GUI.
  The `Learner` contains one `Memory`, `Context`, `ParsingEngine`, and `ThingFactory` instance. The `Learner` keeps a list of all the `Event`s that it learned from in a given session. The `Learner` implements a `learn()` method that takes as input a list of `Event`s (possibly of length 1), and for each event, it delegates to one of seven methods, according to the kind of object property that each event expresses. These methods are:
  `learnFromClassification()`,
  `learnFromCapability()`, `learnFromProperty()`,
  `learnFromAction()`, `learnFromHas()`,
  `learnFromRequires()`, and `learnFromMadeof()`.
  These seven methods contain the same code template: each of them extracts the subject, verb, and object out of the event, each updates the subject and verb's threads, each stores subject and object in memory, and each updates the context before returning. Appendix A shows template for the code of these methods.
  The `Learner` also implements an `answerQuery()` method, that takes in a query expressed as an English question, delegates to one of four methods called `qualifyX()`, `qualifyXY()`, `compareXY()`, and `desctibeThing()`, returns an answer to the user, and possibly learns from the answer (if the answer was a positive one.) The `qualifyX()` methods tries to answer "what is X" questions; `qualifyXY()` tries to answer "Does X do Y" questions;
  `compareXY()` tries to answer "Is X similar to Y" questions; and `describeThing()` tries to answer "Describe X" questions. I show the code for `qualifyX()` in appendix A.

- The `Memory` class stores Thing and Relation instances of the Java Hashtable class. `Memory` provides methods to retrieve Things name, and provides an `update()` method that takes in a Thing structure and stores it in the appropriate hashtable. The most interesting method of `Memory` is the method that calculates a stereotype from the absolute stereotype and relative stereotype for the given class of objects. This method produces a new Thing, which is the union (i.e. contains all the types of) the absolute stereotype and the relative stereotype, both of which are possibly empty (i.e. have no types). Appendix A shows the code for this method. `Memory` also contains methods to initialize itself by initializing and starting up the I/O and reflection threads.

- The `ParsingEngine` class implements methods that call on the Bridge parser class to parse English sentences input to the system. The `ParsingEngine` then retrieves the trajectory structures output by this parser and wrap them into `Event` instances that can be passed to the `Learner` module.

- The `Event` class implements methods for retrieving the subject and object of the underlying sentence and the Relation corresponding to the sentence.

- The `Context` class implements the context object as described in chapter 3. It contains a String instance, a Thing instance, a relation instance, and an instance of the class `LStringsQueue` that implements a queue for thread labels. The `Context` is a singleton and provides a `getContext()` method, as well as setter methods for changing its String, Thing, Relation, and queue elements.

- The `ThingFactory` class contains Strings (declared public, static, and final) representing the labels of the Threads attached to Thing structures. The main role of the `ThingFactory` class is to take in Things coming from the Bridge parser and attach to them the seven object threads (or the three verb threads) that this learning system needs.

- The `Thing` class implements several static methods that manipulate the threads of the Bridge Thing structures.

- The `Utility` class implements a library of methods used by several other classes in the system.

4.1.2 The Implementation of the User Interface

Squirrel's GUI was implemented in Java Swing. Through this interface, the human user can provide input sentences (or text files with sentences) and queries to the `Learner`. The `Learner`, in turn, displays its output to the GUI. Figure 4.1 shows a screenshot of the Squirrel GUI.

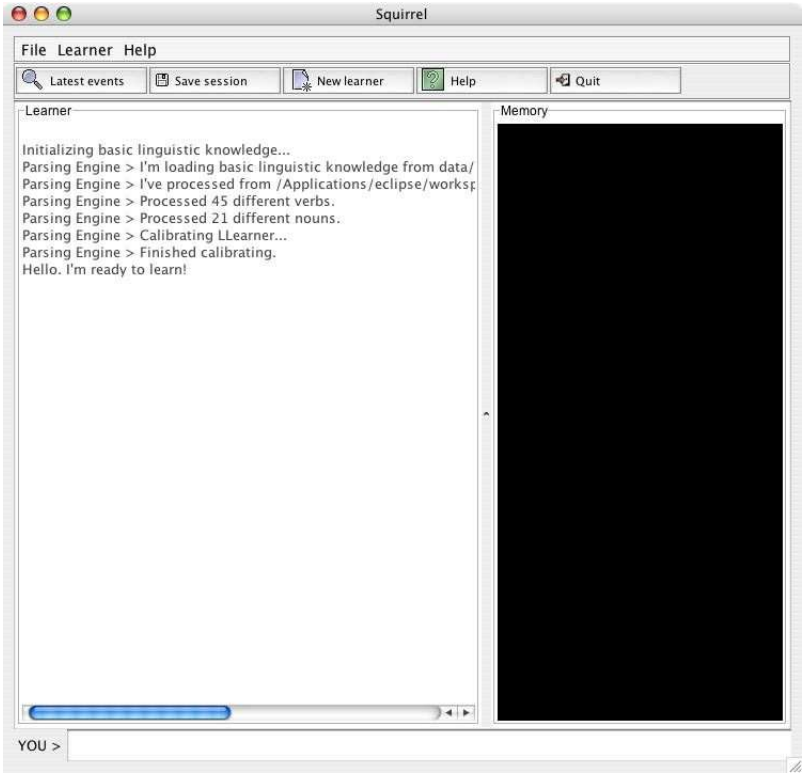

Figure 4.1: A screenshot of the Squirrel GUI.

The GUI contains two text areas. The white area on the left displays the output of the `Learner` module (this includes output of the modules manipulated by the `Learner`, such as the `ParsingEngine` and `Context` modules). The black area on the right displays output of the `Memory` module. This includes, for example, detailed output that the `Memory` provides while calculating the stereotype for a class of objects. At the bottom of the GUI window, there is a text field where the user can

type text to interact with the system.

The user interacts with the system either by typing in the text field or by invoking one of the actions of the toolbar buttons or menu items. The `Labels` class contains the strings that can be presented to the system in order to ask the system to learn from sentences or answer to queries. Here are a few examples of what the user can type into the GUI's text field:

- `learn:` followed by an English sentence, for example `learn: A bird flew to a tree.` This tells the system to learn from the given sentence. As the systems processes this sentence, it displays output in the left text area. For example, it shows the `Event` corresponding to the sentence, and prints a confirmation message to the user.

- `learn from file:` followed by a file name, for example `learn from file: afile.txt` This tells the system to read the file and learn from all sentences in the file, in the order in which they appear in the file.

- `answer:` followed by a query, for example `answer: can bird fly?` This tells the system to answer the given query.

The following is an explanation of the functionality present in the GUI's menubar and toolbar.

The menubar contains menus titled File, Learner, and Help. From the File menu, the user can:

- Choose the New Learner option. This instantiates a fresh `Learner` object, with a fresh memory, parsing engine, and context. This effectively means that the user starts interacting with a new system, which has no knowledge about the world.

- Choose the Save session option. This saves the output of the system to a text file. The output contains the outputs of the `Learner`, `Memory`, `ParsingEngine`, `Context`, and `Event` modules.

- Choose the Quit option, which quits the program.

From the Learner menu, the user can:

- Choose the Retrieve option. This lets the user view: all the Things in memory, all the Relations in memory, all the stereotypes in memory, all the events the learner has learned from during the current session, or the very last events that the learner learned from.

- Choose the Mode option. This lets the user choose various settings for the running program, such as controlling the amount of output that the learner and memory instances provide.

From the Help menu, the user can:

- Choose the Show Help option, to view detailed help information about how to interact with the system.
- Choose the About option, to view information about the current Squirrel version.

The toolbar contains buttons that provide shortcuts for: viewing the last events the `Learner` learned from, saving the output of the current session to a text file, reseting the `Learner`, viewing the Help dialog, and quitting the program.

4.2 Concrete Learning Examples

To illustrate the learning and reasoning capabilities of the system, I present here three examples from Squirrel's execution. The first example, described in subsection 4.2.1, illustrates how the system deduces that robins fly by reasoning that robins are birds and that birds can fly. The second example, described in subsection 4.2.2, illustrates how the system deduces that planes fly by reasoning that planes are similar to robins and that robins fly. The third example, described in subsection 4.2.3 illustrates how the system picks different attributes to describe a robin, according to which object properties are rendered relevant by the context.

4.2.1 Robins fly because they are birds, and birds can fly

In the first example, the learner first reads and learns from a text file containing several sentences about properties of birds and robins. The input file contains the following sentences. (The syntax of the sentences is a little strange, because the parser is not very sophisticated. When the subject noun has plural form, the sentences is interpreted to be an assertion about a class, although the verb is in the singular form.)

```
Birds is an animal.
Birds has wings.
Birds has legs.
```

```
Birds is made of feathers.
Birds fly.
Birds walk.
Birds is small.
Birds is black.

Robins is an animal.
Robins is a bird.
Robins is a wild_bird.
Robins has wings.
Robins is made of feathers.
```

Here is output from the actual execution of the program. The first few lines give the user feedback about the setting up and calibration of the system. At this time, the system reads two files that contain basic linguistic knowledge for its natural language parser. The next chunk of output shows the events that correspond to the sentences in the file above, that the learner learns from. I show only the first two such events here.

```
Initializing basic linguistic knowledge...
Parsing Engine > I'm loading basic linguistic knowledge
from data/wordk
Parsing Engine > I've processed from
bridge/experiments/learner/data/wordk
5162 Things.
Parsing Engine > Processed 45 different verbs.
Parsing Engine > Processed 21 different nouns.
Parsing Engine > Calibrating LLearner...
Parsing Engine > Finished calibrating.
Hello. I'm ready to learn!

I'm trying to learn from a file called data/story1
I'm trying to learn from file
bridge/experiments/learner/data/story1

I'm learning from the event:
Event: classification-8619
 With relation: thing: classification-8619
(thing intangiblething event classification,
features complete)
 thing: birds-8625 (subject)
 (thing animal birds, features classification)
```

```
 thing: animal-8624 (object)
 (thing animal, features identifier an complete)
 With subject: thing: birds-8625
(thing animal birds, features classification)
 First object: thing: animal-8624
(thing animal, features identifier an complete)
 All object(s):
(
thing: animal-8624
(thing animal, features identifier an complete)
)

I've learned that bird-8748 is a animal

I'm learning from the event:
Event: have-8631
 With relation: thing: have-8631
(thing event relation have)
 thing: birds-8628 (subject)
 (thing birds, features noun identifier complete)
 thing: wings-8632 (object)
 (thing wings, features noun identifier complete)
 With subject: thing: birds-8628
(thing birds, features noun identifier complete)
 First object: thing: wings-8632
(thing wings, features noun identifier complete)
 All object(s):
(
thing: wings-8632
(thing wings, features noun identifier complete)
)

I've learned that bird-8748 has wings
```

The output shown above appears in the white text area of the GUI window. While the system processes the input file, the black text area of the GUI window shows output of the `Memory` and `Context` modules. For example, the following output is displayed in the right area after the processing of each of the first three sentences in the file above.

```
 <<< Context: type=animal thread labels=isa
Thing=bird-8748 Relation=classification-8619 >>>
```

```
<<< Context:  type=bird  thread labels=has isa
  Thing=bird-8748  Relation=have-8631 >>>

<<< Context:  type=bird  thread labels=has isa
  Thing=bird-8748  Relation=have-8640 >>>
```

Once the system parsed the input file, I asked it "What does robin do?" Here is the corresponding output.

```
You've asked: what does robin do?. I'm thinking...
Let's see. I'm looking at the threads of robin-8757
to see what it does
I don't know because I haven't learned what robin-8757
does
Let's see. I'm thinking about whether I can say anything
about robin-8757 by looking at the type in context...
Let's see. I'm comparing the threads of robin-8757
and robin-8760
I think that robin-8757 is similar to robin
Let's see. I'm looking at the threads of robin-8760
to see what it does
I don't know because I haven't learned what robin-8760
does
I don't know because I can't deduce anything from
the type robin
Let's see. I'm trying to think about whether I can deduce
anything about robin-8757
by looking at whether it's similar to the thing
in context...
I don't know because I can't deduce anything from
the thing in context.
Let's see. I'm trying to think about what robin-8757
does by looking at what robin-8757 is...
I'm trying to think about robin-8757  by looking at the
intermediate type: bird
Let's see. I'm looking at the threads of bird-8763
to see what it does
bird-8763 does fly because I've learned that bird-8763
does fly
robin-8757 does fly because robin-8757 is a bird and
I know that bird-8763 does fly
robin-8757 does fly because I've looked at
what kind of thing robin is.
```

```
I conclude that robin-8757 does fly
** I will learn that: robins  fly
I'm trying to learn from the phrase ``robins  fly''

I'm learning from the event:
Event: fly-8772
 With relation: thing: fly-8772
(thing event go fly)
 thing: robins-8768 (subject)
 (thing robins, features noun identifier complete)
 thing: path-8771 (object)
 (thing path, features empty)
  sequence (0 elements)
 With subject: thing: robins-8768
(thing robins, features noun identifier complete)
 No objects.

I've learned that robin-8754 fly
```

To answer the question, the system tries the methods described in chapter 3. As the output shows, the system tries several dead ends until it gets to thinking about what kind of thing a robin is. Then, it reasons that a robin is a bird, and remembers that birds fly. Therefore, it concludes that robins also fly, in virtue of being birds. Once the system obtains the answer, it proceeds to learn from it, as it would from an input sentence.

4.2.2 Planes fly because they are similar to robins, and robins fly

In the second example the system reads an input file with the following sentences.

```
Birds is an animal.
Birds has wings.
Birds has legs.
Birds is made of feathers.
Birds fly.
Birds walk.
Birds is small.
Birds is black.

Robins is an animal.
```

```
Robins is a bird.
Robins is a wild_bird.
Robins has wings.
Robins is made of feathers.

Planes has wings.
Planes is black.
Planes is big.
Planes requires wings.
Robins requires wings.
```

Once the system learned from the sentences, I asked it "Can planes fly?" Here is the output for this question.

```
You've asked: can plane fly?. I'm thinking...
Let's see. I'm thinking about whether plane-8865 can
fly by looking at its threads...
I don't know because I haven't learned whether
plane-8865 can fly
Let's see. I'm thinking about whether I can say
anything about plane-8865 by looking at the type
in context...
Let's see. I'm comparing the threads of
plane-8865 and robin-8868
I think that plane-8865 is similar to robin
Let's see. I'm thinking about whether robin-8868
can fly by looking at its threads...
robin-8868 can fly because I've learned that
robin-8868 can fly
plane-8865 can fly because plane-8865 is a robin
and I know that robin can fly
plane-8865 can fly because I've compared plane
with the type in context.
I conclude that plane-8865 can fly
** I will learn that: planes can fly
I'm trying to learn from the phrase ``planes can fly''

I'm learning from the event:
Event: fly-8878
 With relation: thing: fly-8878
(thing event go fly, features can)
 thing: planes-8873 (subject)
 (thing planes, features noun identifier complete)
```

```
 thing: path-8877 (object)
 (thing path, features empty)
  sequence (0 elements)
 With subject: thing: planes-8873
(thing planes, features noun identifier complete)
 No objects.

I've learned that plane-8803 can fly
```

The system answers this question by comparing planes with the context type, which happens to be robin. It discovers that planes are similar to robins, and concludes that planes fly because it knows that robins fly.

4.2.3 A robin is made of feathers, has wings, and is a bird

In the third example, the system first receives the following sentences.

```
Robins is an animal.
Robins is a bird.
Robins is a wild_bird.
Robins has wings.
Robins is made of feathers.
```

After learning from these sentences, the context contains the thread labels *madeof*, *has*, and *isa*. Once the system processed these sentences, I asked it: `Describe robin.` Here is the corresponding output.

```
You've asked: describe robin. I'm thinking...
Let's see. I'm thinking about robin
I conclude that robin is made of feathers,
has wings, is bird
```

The output shows that the system describes robin in terms of the threads in context. That is, it describes robin in terms of what it is made of, what parts it has, and what kind of thing it is. At this point, I entered two more input sentences:

```
Robins can fly.
Robins requires wings.
```

Once the system processed these new sentences, I asked it again: `Describe robin.` The output is shown bellow.

```
You've asked: describe robin. I'm thinking...
Let's see. I'm thinking about robin
I conclude that robin requires wings,
can fly, is made of feathers
```

The output shows that the system's description of a robin has now changed, because of the change in context. After the system processes the last two input sentences above, the context contains the thread labels *requires*, *can*, and *madeof*. Thus, the relevant properties of an object are now what it requires, what it can do, and what it is made of, and the system's answer reflects this change.

Chapter 5

Evaluation and Discussion

"The man with insight enough to admit his limitations comes nearest to perfection." (Johann Wolfgang von Goethe)

In this chapter, I evaluate the learning architecture I described and the implementation I built with respect to the kinds of learning it supports and its extensibility to broader purpose learning and reasoning. At the end, I point out the surprises I have come upon while working on this project.

5.1 System Evaluation

The examples from the previous chapter illustrate how Squirrel proceeds to learn the properties about objects specified in table 3.1. Furthermore, the conceptual design of the learning and reasoning methods allows this architecture to learn broader purpose knowledge about objects, and to reason more abstractly about the world. The software design of the modules that comprise the system is modular enough to render the system easy to understand and to extend with more layers of abstract reasoning.

For example, the system can learn about actions (verbs) by using the same infrastructure currently in place for learning about objects. I believe that learning about actions is important because it can help with the particularly puzzling case of the penguin that is a bird and is similar to birds in many respects, yet does not fly. How could the

system described here arrive to know that penguins can't fly? I consider this example important enough to dedicate the next subsection to suggesting a solution to this 'puzzle'.

5.1.1 Penguins can't fly

In the current setup, the system is likely to deduce that penguins fly, either by reasoning that penguins are birds and birds fly, or by reasoning that penguins are similar to birds (or to some particular species of birds). Also, the system described so far is unsupervised, so there is no way for the human to interact and tell the system it has drawn a wrong conclusion. So how could the system then learn that penguins cannot fly?

One way this may be accomplished is by augmenting the system to also reason about objects from the information it has about actions. Suppose the system knew that *flying* requires the flying animal to have hollow bones, so as to minimize its weight. One way it could deduce this fact in the first place is by noticing that all objects that are known to fly have *hollow bones* on their *has* or *property* threads. Then, the system could reason that because penguins have solid bones, they cannot fly. More generally, the system could learn exceptions to a rule by bootstrapping from the knowledge actions and objects.

This solution is actually an instance of the generalization and specialization mechanisms for machine learning proposed by (Winston 1993, [48]). In this case, the *requires* thread acts as a *requires link* which is part of the description of the class of objects that are birds.

5.1.2 Contributions of the Learning Architecture

The main contributions of the learning architecture and its Squirrel implementation are the following:

- Squirrel illustrates the benefits of using context to learn and reason about objects' properties. The examples from chapter 4 show that context provides the learner with the appropriate information it needs in order to think about objects and to reason about their properties. The learner could have reasoned by using a traditional rule-based mechanism or a statistical algorithm. I believe, however, that using context (possibly layered on top of traditional methods) can significantly improve the learning capabilities of a system.

- Squirrel illustrates the benefits of being able to 'think' about the world in more than one way. The examples in chapter 4 show that Squirrel is able to try several ways to deduce whether an object has a particular property. The system can try to reason about what kind of thing the object is, or it can compare it to other objects it knows more about. I believe that all of these are sensible ways in which we, humans, also attack the same problems.

- Squirrel offers reasons as support for the judgments it makes about objects. As the output from chapter 4 shows, Squirrel 'knows' why it concluded that an object has a particular property. For example, if it knows that a robin is a bird and a bird can fly, then it concludes that a robin can also fly *because* a robin is a bird. Thus, Squirrel is able to cite support for its conclusions about how the world works.

- Squirrel illustrates how a system can reason about the world by relying on the fundamental cognitive processes of similarity and categorization. Reasoning by similarity helps the system build its knowledge of the world incrementally, by inferring new properties of an object from its similarity with another, better known object. Reasoning by categorization helps the system abstract its knowledge of the world, by creating stereotypes to represent groups of objects.

- Squirrel represents an attempt to create a system that sets its own goals for learning about the world. The system is designed to 'reflect' or to 'think' by choosing a category of objects or a particular object that the system has learned about in the past and trying to deduce new properties of that category or object. Thus, provided a reasonably large input to start off with, a system like Squirrel can be expected to consolidate its knowledge by deducing new facts and making new predictions about the world.

An important observation about the learning architecture described in chapter 3 and its implementation described in chapter 4 is that while the architecture relies on the existence of several parameters (such as the weights of the threads for calculating similarity scores), the implementation chooses concrete values for these parameters. These values can vary in different implementations. By varying them, one can hope to optimally tune the learning system. I believe that one of the best ways to accomplish this tuning is by using a parameter adjusting scheme (such as a neural net, a genetic algorithm, or any other

computational scheme) to learn the 'best' values for these parameters based on the kinds of sentences the system receives as input and on the context the system builds from the input.

Besides the merits described above, Squirrel also has several limitations. These limitations stem from: (1) the underlying Thing and trajectory representations that the system uses to model the real world, and (2) from the restrictions that the language parser places on the kind of inputs that can be presented to the system.

In general, having only one kind of representation of the world restricts a system to being able to reason efficiently only about some (not all) aspects of the world. Squirrel reasons very efficiently about categories that objects belong to, about what parts they have, and so on, but cannot reason, for example, about object properties that change with time. This is a limitation inherent in the underlying Thing/Thread representation. (Chklovski 2003, [10]) also discusses the importance of using multiple representations. He asserts that his learner could reason better given more (or more detailed) representations, and also given a more detailed model of which properties of objects imply the presence or absence of which other properties (Chklovski 2003, [9]). This is especially true of my system. I believe the system could learn more effectively if it had a way to tell, for example, that the presence of types on some thread t1 implies the same types should be present on thread t2. A concrete example is that types that appear on the *action* thread of a Thing should also appear on the *can* thread of that Thing. One could accomplish this kind of learning by designing the system to compare the types on pairs of threads and to see which threads are similar, meaning which threads have approximately the same types, across classes of objects. What this scenario would require is the 'right' notion of similarity for threads.

The Bridge parsing module also restricts the kinds of things my system can learn because it imposes restrictions on the kinds of sentences the system can receive as input. For example, the sentences have to be assertions that are interpretable in isolation (such as "A bird flew to the tree"), or one can string several assertions together to refer to objects already spoken about (such as "A bird flew to the tree *and then* the bird landed on a rock.") From the point of view of the learning and reasoning the system is designed to do, I believe these limitations are only superficial. That is, the framework for learning with context, and the design of the learner to reason from similarity, categorization, and stereotypes is general enough that the system can acquire broader kinds of knowledge once we substitute in a more sophisticated parser.

From a practical point of view, it is interesting to point out what tasks this system could be used for. For one thing, the examples in chapter 4 show that Squirrel is an effective inference engine, so it could be used to reason about expert domains. I believe that a more exotic task that Squirrel could perform well on is recognizing metaphors in natural language. The system could base its metaphor recognition on lack of similarity between two things. The following example illustrates. Suppose the system receives as input a phrase such as "Words are daughters of the earth", or "language is the instrument of science" [1]. The system could compare the subject and object of the sentence, that is compare word with daughter, or language with instrument. In cases where the comparison indicates the two things are dissimilar, the system could conclude the sentence contains a metaphor, because it brings together things that the system knows are dissimilar in the real, objective world.

One final note. The design of the learning architecture includes a reflection thread in the system's memory, whose role is to pseudo-randomly pick things to think about (I say pseudo-randomly because what is picked is meant to be primed on the context at the time the choice is made.) This source of randomness could lead one to observe interesting differences in the knowledge acquired by different `Learner` instances. For example, one learner may think a lot about robins and arrive to be quite an 'expert' on what robins are, how they behave, etc, while another learner may arrive to know a lot about mallards. This 'evolution' could be exploited by setting up the learners to 'talk' to each other, by uttering assertions to each other; if the kinds of assertions a learner utters are primed on the kinds of things the learner thinks about or knows a lot about, then this setup would effectively accomplish the transfer of knowledge from one learner to another.

5.2 Surprises

I have had several surprises while working on the design and implementation of this learner. I have found that:

- Context helps significantly with query answering, because it captures the relevant information that the system needs to bootstrap its reasoning from,
- Reasoning by similarity and categorization is highly effective and can form the basis for a more abstract reasoning platform,

[1] Taken from Samuel Johnson, quoted in Minsky 1988, [25]

- Using intermediate complexity categories to classify objects and deduce their properties seems to work better than using more specific or more general categories,

- It was very convenient to build the system to 'talk' to itself while processing information. Squirrel uses English phrases to pass information around internally–for example, once it deduces an answer to a query, it expresses it in English, and passes this English phrase on to its learning method. English is also used in the Bridge system as an intermediary for translating from Jackendoff trajectory representations (Jackendoff 1983, [16]) to Borchardt change frames (Borchardt 1993, [6]). The translator was designed and implemented by Seth Tardiff, a member of the Genesis group. My system as well as the translator thus suggest that simple English is a clean interface not only between human and machine, but also within the machine.

- The idea about using a measure of similarity of threads to deduce which properties imply the presence (or absence) of which other properties lead me to an interesting observation about the role of rules in a learning or reasoning system. The observation is that rules act as a *shortcut* for reasoning that is *learned* by the system *throughout the system's evolution*, rather than as a primary mechanism for learning. In the past, authors have asserted (Minsky 1988, [25]) that, although we do seem to use rules, rules are probably not hard-coded in the brain because it is evolutionarily more efficient to learn them.

The evaluation of the current system, coupled with the surprises I have observed while building it, leads me to suggest several salient ideas for how to build a architecture to support broader purpose learning and reasoning about the world. I describe these ideas in the next chapter.

Chapter 6

Salient Ideas for Learning with Context

"The best way to predict the future is to invent it." (Alan Kay)

While the previous chapters have described and evaluated the system I have built and hinted that it is possible to build a broader purpose learner and reasoner, this chapter suggests the steps to get there. The most important steps have to do with using more and richer representations of the world, making better use of context in reasoning about the world, designing better learning heuristics, and using a better model for the system's memory.

6.1 Richer Representations of the World

I believe that having several representations of the world instead of just one, and having the ability to reason about which representation is most appropriate for a given problem is one of the keys to achieving human-like intelligence, and beyond. I have already argued in chapter 5 that richer internal representations can help improve the learning and reasoning of the system. Here, I have two concrete suggestions in mind, namely using Borchardt transition frames to represent events in the world (Borchardt 1993, [6]) and using Allen time representations (Allen 1983, 1984 [2, 3]) to capture the notion of time passing and of the world changing with time. I think that experimenting with various representations can yield surprising insight into the problem of building an architecture capable of more abstract reasoning about the world.

6.2 Better Use of Context

The current work is a step toward incorporating context in machine learning. Context, however, can be exploited much more.

For example, context can be used by the architecture described in this thesis not only to provide information (a type, a Thing, a Relation) to guide each question answering method, but furthermore to guide the system to choose *which method is most appropriate* for answering a given question, or what is *the most appropriate priority queue* for the methods.

Also, as mentioned before, context can be used to prime the system's 'thinking'. Just like question answering takes context into account, it's natural for the reflection that the system does to be guided by the context. As a concrete example, if the system has recently seen a lot of input information about birds, then it should be likely to think about birds in the near future. I believe that implementing thinking in this way will improve the quality of the information that the system deduces and maintains in its KB.

From a software engineering perspective, a better way to implement context in a learning system such as the one described in this thesis is the following. The context is essentially an observer of the information that passes around the system and is a blackboard for the modules that need context information to do their job. The context should be able to see the input that the system receives for learning or the information that the system deduces, and the responses it gives to queries. The context changes itself based on the information it sees pass around. For example, it keeps a history of the latest system utterances that it observed and updates its components according to these utterances. With this design, the other modules in the system that need to make decisions based on context can query the context for its value, then make their decisions. Moreover, these modules need not worry about updating the context themselves. The main advantage of this approach is that the context object becomes a useful abstraction from a software engineering perspective, rather than being just a container of several data types.

6.3 Better Learning and Reasoning Heuristics

There are several respects in which the system's heuristics for reasoning and learning can be improved. The most important are:

- The system could learn that the presence of certain properties implies the presence or absence of other properties. I have discussed this in more detail in chapter 5.

- The system relies on several parameters for learning and reasoning. I have discussed in chapter 5 that the optimal values for these parameters could be learned by a parameter adjusting scheme, instead of being hard-coded in the implementation.

- As discussed in chapter 5, the system can be augmented to learn about actions in the same way that it learns about objects. Besides expanding the kinds of knowledge the system has about the world, learning about actions can contribute to make the learning about objects and their properties more efficient. For example, in a scenario where the system knows that *fly* is a *move* kind of action, the system could deduce that a bird can move from having seen it fly.

- Another important aspect of how we humans learn, as Minsky and Singh describe in the OMCS paradigm ([1]), is that we tend remember the ways in which we reasoned about things and solved problems in the past. This helps us solve similar problems more easily in the future. Accordingly, it may turn out to be useful to design the learning system to keep track of which methods (out of all the methods in its repertoire) worked to reason about particular situations or answer particular queries.

- The current system is unsupervised. I believe it would be interesting to experiment with supervised learning in this architecture. The supervised mode can be set up such that when the learner deduces a piece of information, it asks the human user for confirmation. The merit of this mode is that it enables the system to quickly validate the inferences it makes about the world. For example, suppose that when asked, "Is a robin an animal?", the system sees that some (but not most) robins are birds and that all birds are animals. Then, the system can ask, "Is a robin a bird?". An affirmative answer gives the system the necessary information on the spot. In the unsupervised mode, this scenario has a different outcome: the system answers the question "Is a robin an animal?" in the negative, until it sees a majority of robins which are birds, so that its concept of the stereotypical robin changes.

- I believe a lot could be learned from performing a Kirby-like experiment (Kirby 1998, [17]) that involved a population of learners,

each with the architecture presented here. Each learner could 'broadcast' its utterances (that is, the deductions the learner makes during reflection, or the answers it deduces to queries) to the entire population. Each learner could take another's utterances as its own new inputs. The idea of performing such an experiment comes from the general belief in psychology that developments in human intelligence were importantly shaped by social complexities ([12]). The Machiavellian Intelligence Hypothesis ([12], Byrne and Whiten 1988, [31]) also supports this idea. It states that the advanced cognitive processes of primates are primarily adaptations to the special complexities of their social lives, rather than non-social environmental problems such as finding food. I believe such an experiment could turn out surprising results about the rate at which 'correct' facts about the world can be learned collectively versus individually. It would also be fascinating to investigate whether the 'chatting' of the learners can give rise to differences in the concepts learned.

6.4 Better Models of Memory

The design of the system does not currently include a thorough model of memory. In essence the `Memory` module implements a long term memory. I believe there might be something to be gained from designing the memory more carefully. For example, it may be useful to distinguish between implicit and explicit memory. Implicit memory allows us to learn new skills based on previous experience, without our recollecting any particular events, or without our even realizing that prior knowledge is used. Explicit memory has to do with explicit recollection of events or points in time.

Explicit memory could help with reasoning about objects. The system could keep records of the judgments it makes, where the records contain the method (or methods) the system finds useful in answering a particular type of questions. For example, suppose the system finds it useful to answer questions about robins by looking at what properties they inherit from birds. Then, when presented with similar questions, the system could 'recall' the methods it used in the past, and attempt to use them again to answer the new questions. In the robin and bird example above, the system could attempt to answer new questions about robins by thinking again about birds.

6.5 Autonomy... and All That Jazz

Finally, the bigger vision I have started out with was to create a rational learner. 'Rational' traditionally means that it can be expected to achieve its goals, given its available knowledge base ([12]). Currently, Squirrel maintains a repository of knowledge that it can expand and reason about. The next grand step is: how can Squirrel be made to 'act' based on its knowledge? That is, how can it set goals for itself and carry them through? I believe part of this lies in the design and implementation of the reflection thread. To reflect, the system picks an object to think about, or poses itself a question to answer about some object's property. This can essentially be seen as as setting a goal, and answering the question can be seen as carrying the goal through. I believe, however, that there is much more to do in this research direction.

Chapter 7

Contributions

"There is a single light of science, and to brighten it anywhere is to brighten it everywhere." (Isaac Asimov)

In this thesis, I have made the following contributions toward building a machine that can learn and reason about common aspects of the world.

- I have designed and implemented a learning architecture that uses similarity and categorization judgments, and the context it builds from information recently received, to reason and learn about objects and their properties. This architecture provides the basic infrastructure on top of which more efficient learners can be built.
- I have designed this architecture to be modular, and easily extensible to support supervised learning, learning about actions, and concurrent reflection and forgetting, that have potential to reveal new surprises about artificial learning.
- In the area of context in machine learning, I contributed by defining context as temporal proximity of information. I have implemented this notion of context on top of a thread memory system. In this implementation, the context has seven dimensions, with each dimension representing a particular kind of property that an object can have, such as a capability of the object, or an *isa* property.
- In the area of similarity and categorization, I have contributed by illustrating how a system can reason about the world by re-

lying on these fundamental cognitive processes. I showed that reasoning by similarity helps the system build its knowledge of the world incrementally, by inferring new properties of an object from its similarity with another, better known object. I further shows that reasoning by categorization helps the system abstract its knowledge of the world, by creating stereotypes to represent groups of objects.

- In the area of acquiring commonsense knowledge, I have contributed by building a system that can learn seven of the thirteen kinds of commonsense knowledge that people have about objects in the world. I further illustrated the benefits of being able to think and reason about the world in more than one way. This idea of having multiple methods to think about something or to solve a problem is one of the hallmarks of the Open Mind Common Sense paradigm.

- In the area of learning, I have contributed the design of a reflection mechanism that allows a system to set its own learning goals by picking things to think about and by posing itself questions about the world. The result of reflecting is that the system can deduce new information about objects given an initial input of reasonable size, such as an input file with twenty simple English sentences.

- Finally, I have suggested a set of salient properties for building a more efficient platform for learning with context. The most important are the use for multiple representations and the idea of priming the system's reasoning and learning on the context built from information recently received or information recently exchanged with a human user or with another system.

Appendix A

Examples of Squirrel Code

A.1 Pseudocode for the Assessing the Similarity of Two Things

The method `Learner.compareFromThreadsXY()`, whose code is given below, compares two Things according to a linear measure of the types their threads have in common. It calls on the methods `LThing.countCommonTypes` and `LThing.countMinTypes`, also shown below. The `LThing` module implements several useful methods that manipulate Things and Threads.

```
// requires x, y not null
private String compareFromThreadsXY(Thing x, Thing y){
  /* This method compares the x and y by looking at
   the types on their threads;
   the threads in context weigh more in the
   comparison than the others;
   the weights and similarity threshold can be
   changed (in the future they can be learned by
   a parameter adjusting algorithm)

   Calculate a similarity score for the two things:
   simScore = Sum_threads-in-common{threadSimScore};
   threadSimScore = (number of types in common) *
                    (thread weight)
```

```
 this method doesn't run into trouble if one tries
 to compare things with different threads.
 In that case, there will be fewer threads
 in common which will drive the overall
 similarity score down.

 if simScore > high_threshold,
   return LLabels.similarLabel;
 if simScore < low_thresholds,
   return LLabels.differentLabel;
 else return LLabels.dontknowLabel;

 Possible extension: if during the calculation
 of the per thread similarity scores, a thread
 that is not currently in context proves
 to have a very high score (that is, many types
 are in common), then weigh it more, and add it
 to the context.

 When the two things are indeed similar,
 picks one randomly and sets the context
 thing to it.
*/

String answer = ``'';
print(``Let's see. I'm comparing the threads of''
 +x.getName()+``and'' +y.getName());

List threadsInCommon =
 LThing.getCommonThreads(x, y);
if (threadsInCommon.size() == 0)
 return(LLabels.differentLabel);
else {
  double simScore = 0.0;
  // similarity score of x and y
  // over all threads
  double maxScore = 0.0;
  // maximum possible similarity score
  // for these x and y
  int commonTypes = 0;
  int totalTypes = 0;
  double threadWeight = 0;
```

```
    for (int i=0; i<threadsInCommon.size(); i+=2){
      Thread t1 = (Thread)threadsInCommon.get(i);
      Thread t2 = (Thread)threadsInCommon.get(i+1);

      commonTypes = LThing.countCommonTypes(t1, t2);
      totalTypes = LThing.countMinTypes(t1, t2);

      if (context.containsLabel(LThing.getLabel(t1)))
      threadWeight = contextThreadWeight;
      else threadWeight = defaultThreadWeight;
      simScore += commonTypes * threadWeight;
      maxScore += totalTypes * threadWeight;
    }

    if (simScore > highSimThreshold*maxScore){
      answer = LLabels.similarLabel;
      Random generator = new Random();
      int index = generator.nextInt(1);
      context.setThing(index == 0 ? x : y);
    }
    else if (simScore < lowSimThreshold*maxScore)
     answer = LLabels.differentLabel;
    else answer = LLabels.dontknowLabel;
  }
  return(answer);
}

/**
 * Returns the number of types (besides the thread label)
 * that th1 and th2 have on common.
 *
 * @exception NullPointerException if any arg is null
 *            IllegalArgumentException if th1 and th2
 *            do not have the same label.
 */
public static int countCommonTypes(Thread th1,
                                   Thread th2){
  if (th1.size() == 0) return(0);
  else if (th2.size() == 0) return(0);
  int count = 0;
  List types1 = getTypes(th1);
```

```
  List types2 = getTypes(th2);
  String label1 = (String)types1.remove(0);
  String label2 = (String)types2.remove(0);
  if (!label1.equals(label2))
    throw new IllegalArgumentException();
  for (int i=0; i<types1.size(); i++){
    String type = (String)types1.get(i);
     if (types2.contains(type)) count++;
  }
  return(count);
}

/**
 * Returns the minimum of the number of types,
 * on th1 and th2 excluding the thread labels.
 *
 * @exception NullPointerException if any arg is null
 */
public static int countMinTypes(Thread th1,
                                Thread th2){
  int count1 = th1.size() - 1;
  int count2 = th2.size() - 1;
  if (count1 < count2) return(count1);
  return(count2);
}
```

A.2 Code for Calculating Stereotypes

The following is the code of the `Memory.findGenericThing` method, that calculates the partially relative stereotype of a class of objects. The partially relative stereotype is the union of an absolute stereotype and a relative stereotype. The union is calculated by the method `LThing.unionOfThings`, also shown bellow. The relative stereotype contains the types that belong to most of the exemplars of the class. Its calculation relies on the `LThing.intersectThreads` method, also shown bellow.

```
/**
 * Returns the stereotype Thing for the given type,
 * and stores this stereotype in generics;
 * if type is null or empty, returns null and
 * stores nothing.
```

```
 * The stereotype is the union of the generic
 * absolute for that type (if it exists), and a
 * stereotype calculated from the frames in things.
 */
 // This method is designed for isa types;
 // if type is something that would go an a
 // different thread, this method might not give
 // the result you expect
public synchronized Thing findGenericThing(String type){
  if (type == null || type.equals(``'')) return null;
  type = type.trim().toLowerCase();

  Thing absolute = getAbsStereotype(type);
  Collection things = getThings(type);
  if (verbose)
    print(``Building'' +type+
     ``stereotype from'' + things.size()+
    ``things and'' +
    (absolute == null ? ``no'' : ``one'')+
    ``absolute stereotype.''+
    (things.size() == 0 ? ``'' :
    ``The things are:\n''+
     LUtility.collectionToString(things)));

  if (things.size() == 0){
    Thing st = factory.createGeneric(type);
    if (absolute != null){
      st = LThing.unionOfThings(st, absolute);
      LThing.removeType(st, LLabels.genericAbs);
      st.setName(type+st.getNameSuffix());
    }
    storeGeneric(st);
    return st;
  }else {
    Thing temp = factory.createGeneric();
    Bundle b = temp.getBundle();
    Iterator it = b.iterator();
    while(it.hasNext()){
      Thread th = (Thread)(it.next());
      if (th.contains(LLabels.generic))
        th.remove(LLabels.generic);
        List intersection =
```

```
          LThing.intersectThreads(things,
                                  LThing.getLabel(th),
                                  threshold);
        th = LThing.copyTypes(intersection, th);
    }
    temp.addType(LLabels.generic, LThingFactory.ISA);
    if (absolute != null){
      temp = LThing.unionOfThings(temp, absolute);
      LThing.removeType(temp, LLabels.genericAbs);
    }
    temp.setName(type+ ``-''+
     LThing.extractSuffix(temp.getName()));
    storeGeneric(temp);
    return temp;
  }
}

/**
 * Returns a new Thing, which contains,
 * on each thread, a union of the t1's and t2's
 * types on the threads with the same label.
 *
 * @exception NullPointerException if any
 *            arg is null
 * @exception IllegalArgumentException if t1
 *            and t2 don't both have
 *            threads with the same labels
 *
 * @modifies nothing
 */
public static Thing unionOfThings(Thing t1, Thing t2){
  if (! (isThing(t1) && isThing(t2) ||
         isVerb(t1) && isVerb(t2)))
    throw new IllegalArgumentException(``Things''+
  t1.getName()+ ``and'' +
  t2.getName()+
  ``contain different labeled threads''+
  `` and cannot be ORed:\n''+
  t1.toString(true)+ ``\n''+t2.toString(true));

  Bundle b = new Bundle();
  Bundle t1b = t1.getBundle();
```

```
  for (int i=0; i<t1b.size(); i++){
    Thread th1 = (Thread)(t1b.get(i));
    b.add(unionOfThreads(th1,
    t2.getThread(getLabel(th1))));
  }
  Thing newt = new Thing();
  newt.setBundle(b);
  return newt;
}

/**
 * Returns a new Thread, which contains
 * all the types on t1 and t2.
 *
 * @requires no arg is null
 */
public static Thread unionOfThreads(Thread t1, Thread t2){
  List types1 = getTypes(t1);
  List types2 = getTypes(t2);
  for (int i=0; i<types2.size(); i++){
    String tp = (String)(types2.get(i));
    if (!types1.contains(tp)) types1.add(tp);
  }
  Thread newt = copyTypes(types1, new Thread());
  return newt;
}

/**
 * Returns a List which contains the types
 * contained by a majority of the threads with
 * the given label, of the things in the given
 * collection; threadLabel is the first element
 * in the list; threshold represents the proportion
 * that determines the ‘‘majority’’;
 * if things or threadLabel is empty, or
 * if none of the things in things have threads
 * labeled threadLabel, returns null.
 *
 * @exception NullPointerException if any arg is null
 * @requires no arg is null, threshold is between 0, 1.
 */
public static List intersectThreads(Collection things,
```

```
                                    String threadLabel,
                                    double threshold){
  if (things.size() == 0 || threadLabel.equals(``'')){
    if(verbose)
      print(``Empty collection or thread label''+
       ``in intersectThreads()'');
    return null;
  }
  Collection threads = new ArrayList();
  Iterator it = things.iterator();
  while(it.hasNext()){
    Thing t = (Thing)(it.next());
    Thread th = t.getThread(threadLabel);
    if (th != null) threads.add(th);
  }
  if (threads.size() == 0){
    if (verbose)
      print(``No thing contained threads with label'' +
           threadLabel);
      return null;
  }

  List visited = new ArrayList();
  List newTypes = new ArrayList();
  newTypes.add(threadLabel);
  it = threads.iterator();
  while(it.hasNext()){
    Thread th = (Thread)it.next();
    List types = getTypes(th);
    types.remove(threadLabel);
    for (int i=0; i<types.size(); i++){
      String type = (String)types.get(i);
      if (!visited.contains(type)){
        if (mostHaveType(threads, type, threshold))
          newTypes.add(type);
          visited.add(type);
      }
    }
  }
  return newTypes;
}
```

```
/*
 * Returns true if a proportion equal to threshold if
 * the given threads contain type.
 *
 * @requires the args are not null, threshold is between
 *           0 and 1.
 */
private static boolean mostHaveType(Collection threads,
                                    String type,
                                    double threshold){
  if (threads.isEmpty() || type.equals(‘‘’’)) return false;
  int total = threads.size();
  double num = 0.0;
  Iterator it = threads.iterator();
  while(it.hasNext()){
    Thread th = (Thread)it.next();
     if (th.contains(type)) num += 1.0;
  }
 if (num/total >= threshold) return true;
   return false;
}
```

A.3 Pseudocode for the Reflection Thread of Memory

The following is pseudocode for the reflection thread, as designed in the first version of Squirrel, version 1.0. I have built Squirrel 1.0 in the spring and summer of 2003. The version described in this thesis is version 2.0, built throughout the fall of 2003 and spring of 2004.

The reflection code bellow represents the first idea I have had about how to build a reflection mechanism. Its behavior is simpler than the behavior described in chapter 3 of this thesis: the thread picks two Things at random from memory, called a datum and a pattern. The goal is to learn something new about the datum by looking at the pattern. To do this, the system first sees if the two Things are similar. If they are, then it assigns the datum a property of the pattern. The pattern can be either a particular Thing or a stereotype. This algorithms is perhaps similar to the Self Organizing Maps paradigm.

```
reflect(){
  pick a Thing; // this is the datum
```

```
  pick either another Thing or a stereotype;
  // this is the pattern
  if (compareFromThreadsXY(datum, pattern).equals(
    LLabels.similarLabel)){
    pick a thread label;
    pick a type on pattern's thread with this label;
    // repeat the above until you find a type that
    // belongs to the pattern but not the datum
    record the type on the datum's thread
}
```

A.4 Code Template for the Learn Methods of the Learner

The following is a code template for the `learnFromXYZ()` method of the `Learner` class.

```
// Learns from the given event;
// if e==null or e.isEmpty(), does nothing
private void learnFromXYZ(LEvent e){
  // extract the subject and object
  // (if the object is a sequence, this methods
  // assumes the first object is the desired one);
  // see if the sentence is a statement about
  // generic objects or a particular object;
  // if the there is a Thing in LMemory with the
  // subject's name, retrieve it, else initialize
  // a new Thing;
  // add the object's type to the subject's isa thread
  Thing subject = e.getSubject();
  Thing object = e.getObject();
  String name = subject.getName();
  String type = subject.getType();
  Thing t = null;
  if (engine.isGenericStatement(e)){
    String tp = engine.getGenericName(type);
    t = memory.getAbsStereotype(tp);
    if (t == null){
      if (engine.isVerb(tp))
       t = factory.createGenericVerb(tp);
      else t = factory.createGenericAbsolute(tp);
```

```
    }
  }else{
    String nm = factory.getMapping(name);
    if (nm != null) t = memory.getThing(nm);
    if (t == null) t = factory.createThing(subject);
  }
  String what = object.getType();
  t.addType(what, LThingFactory.xyz_thread_label);
  memory.update(t);
  Relation rel = e.getRelation();
  memory.update(rel); // record the Relation
  print(``I've learned that'' + t.getName()+
        xyz_thread_label +
        object.getType() + ``\n'');

  // change the context according to the input
  context.setParameters(what,
    LThingFactory.xyz_thread_label, t, rel);
}
```

A.5 Code of the Qualify Method of the Learner

The following is the code of the `Learner.qualifyFromThingX()` method. This is one of the methods called by `Learner.qualifyX()` while trying to answer a query like "What is X?"

```
// requires thread_label is one of
// LThingFactory.THING_THREADS,
// x != null
private String qualifyFromThingX(String thread_label,
                                 Thing x){
  // this method tries to return a type from
  // the thread with the given thread label,
  // from the thing in context.
  // it adds thread_label to the context.
  // if there is no such type, returns the empty string;
  // if there is such a type, it changes the type
  // in context to the most specific type of the
  // thing in context.
  //
```

```
String name = x.getName();
String answer = LLabels.dontknowLabel;
String reason =
 ‘‘I can’t deduce anything from the thing in context.’’;

print(‘‘Let’s see.’’+
‘‘I’m trying to think about whether I can deduce’’+
‘‘ anything about’’+name+
‘‘ by looking at whether it’s similar to’’+
‘‘the thing in context...’’);
context.addLabel(thread_label);
Thing toc = context.getThing();

if (toc != null && !LThing.thingEquals(toc, x)){
  String toc_name = toc.getName();
  String temp = name+
    LThingFactory.threadToWord(thread_label);

  if (LThing.isStereotype(toc) &&
  !LThing.hasType(x,
    LThing.getMostSpecificType(toc), LThingFactory.ISA)){}
  else {
    // proceed
    String result = compareFromThreadsXY(x, toc);
    if (! result.equals(LLabels.similarLabel)){
      // don’t know whether they are similar,
      // or know they are dissimilar
      // answer = LLabels.dontknowLabel;
      reason = toc_name+ ‘‘tells me nothing about’’ +name;
    }else {
      // x and toc are similar,
      // so check what toc has on its thread
      print(‘‘I think that’’ +name+ ‘‘is similar to’’
       +toc_name);
      String a_type = LThing.getAType(toc, thread_label);
      if (a_type == null){}
      else {
        answer = temp+a_type;
        reason = name+toc_name+
         ‘‘are similar and I know that’’ +toc_name+
        LThingFactory.threadToWord(thread_label)+ a_type;
        context.setType(LThing.isStereotype(toc) ?
```

```
          LThing.extractNameFromSuffixedName(toc.getName()) :
          LThing.getMostSpecificType(toc));
        }
      }
    }
  }
  print(answer+ ``because'' +reason);
  return(answer);
}
```

Bibliography

[1] The open mind commonsense project. Online resource, at: `http://commonsense.media.mit.edu/`.

[2] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[3] J. F. Allen. A general model of action and time. *Artificial Intelligence*, 23(2), 1984.

[4] J. R. Anderson. The adaptive nature of human categorization. *Psychological Review*, 98(3):409–429, 1991.

[5] M. Bassok. Transfer of domain-specific problem-solving procedures. *Journal of Experimental Psychology: Learning, Memory and Cognition*, (16):522–533, 1990.

[6] G. C. Borchardt. Casual reconstruction. A.I.Memo No. 1403, MIT Artificial Intelligence Laboratory, 1993.

[7] R. Brachman. The future of knowledge representation. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 1990.

[8] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[9] T. Chklovski. Learner: A system for acquiring commonsense knowledge by analogy. In *Proceedings of the international conference on Knowledge capture*, pages 4–12, 2003.

[10] T. Chklovski. Using analogy to acquire commonsense knowledge from human contributors. Technical report, MIT Artificial Intelligence Laboratory, February 2003.

[11] A. B. Markman D. Gentner. Structure mapping in analogy and similarity. *American Psychologist*, 52(1):45–56, 1997.

[12] F. C. Keil Edited by R. A. Wilson. *The MIT Encyclopedia of the Cognitive Sciences (MITECS)*. MIT Press, 2003.

[13] P. Winston et al. Four powerful pieces. Online resource, at `http://www.ai.mit.edu/projects/genesis/frames.html`.

[14] K. J. Holyoak and K. Koh. The pragmatics of analogical transfer. In G.H. Bower (Ed.), The psychology of learning and motivation: Advances in research and theory, pp. 59-87, New York: Academic Press, 1985.

[15] P. Brezillon J. Agabra, I. Alvarez. Contextual knowledge based system: A study and design in enology. In *Proceedings of the International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT-97)*, pages 351–362. Federal University of Rio de Janeiro, February 1997.

[16] R. Jackendoff. *Semantics of Spatial Expressions*. 1983. in Semantics and Cognition.

[17] S. Kirby. Language evolution without natural selection: From vocabulary to syntax in a population of learners. Technical report, Language Evolution and Computation Research Unit, University of Edinburgh, 1998.

[18] J. Kolodner. *Case-based reasoning*. Morgan Kaufmann, San Mateo, California, 1993.

[19] R. D. Greenblatt L. M. Vaina. The use of thread memory in amnesic aphasia and concept learning. AI working paper, MIT Artificial Intelligence Laboratory, 1979.

[20] L. G. Valiant L. Pitt. Computational limitations on learning from examples. *Journal of the ACM*, 35:965–984, 1988.

[21] G. Lakoff. *Women, fire, and dangerous things*. Univ. of Chicago Press, 1987. Chapter One, The Importance of Categorization.

[22] D. Lenat. The dimensions of context-space. online documentation of CyCorp, October 1998.

[23] A. Ram M. Devaney. Dynamically adjusting concepts to accomodate changing contexts. In *Proc. ICML-96 Workshop on Learning in Context-Sensitive Domains*, July 1996.

[24] M. Minsky. A framework for representing knowledge. 1985. In P. H. Winston, ed., The Psychology of Computer Vision, pages 211-277, McGraw-Hill, New York.

[25] M. Minsky. *The Society of Mind*. Simon and Schuster, Inc., first touchstone edition edition, 1988.

[26] P. Brezillon N. M. Bigolin. An experience using context in translation from systems requirements to conceptual model. In *Proc. International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT-97)*, pages 319–330. Federal University of Rio de Janeiro, February 1997.

[27] L. R. Novick. Representational transfer in problem solving. *Psychological Science*, (1):128–132, 1990.

[28] P. Szolovits R. Davis, H. Shrobe. What is a knowledge representation? *AI Magazine*, 14(1):17–33, 1993.

[29] D. L. McGuinness R. J. Brachman. Knowledge representation, connectionism, and conceptual retrieval. In *Proceedings of the 11th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 161–174, 1988.

[30] D. B. Lenat R. V. Guha. Enabling agents to work together. *Communications of the ACM*, 37(7), July 1994.

[31] A. Whiten R. W. Byrne. *Machiavellian Intelligence: Social Expertise and the Evolution of Intellect in Monkeys, Apes and Humans*. Oxford: Oxford University Press, 1988.

[32] E. Rosch. Cognitive representations of semantic categories. *Journal of Experimental Psychology: General*, (194):192–233, 1975.

[33] E. Rosch. *Principles of Categorization*. New Jersey: Lawrence Erlbaum, e. rosch and b. b. lloyd edition, 1978.

[34] M. Kubat S. Matwin. The role of context in concept learning. Invited talk at the ICML-96 Workshop on Learning in Context-Sensitive Domains, July 1996.

[35] E. Sali S. Ullman, M. Vidal-Naquet. Visual features of intermediate complexity and their use in classification. *Nature*, 5(7), July 2002.

[36] M. Sala. On the importance of context to improve knowledge discovery. In *Proc. International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT-97)*, pages 331–342. Federal University of Rio de Janeiro, February 1997.

[37] R. N. Shepard. Multidimensional scaling, tree-fitting, and clustering. *Science*, 210(4468):390–398, October 1980.

[38] R. N. Shepard. Toward a universal law of generalization for psycholofical science. *Science*, September 1987.

[39] D. B. Lenat R.V. Guha K. Pittman D. Pratt M. Shepherd. Cyc: Toward programs with common sense. *Communications of the ACM*, 33(8), August 1990.

[40] P. Singh. The open mind common sense project. Article published on KurzweilAI.net, January 2002.

[41] J. B. Tenenbaum. *Rules and Similarity in Concept Learning*, pages 56–65. Advances in Neural Information Processing Systems. MIT Press, Cambridge, Massachusetts, s. a. solla, t. k. leen, k.-r. muller edition, 2000.

[42] P. Turney. The identification of context-sensitive features: A formal definition of context for concept learning. In *Proc. ICML-96 Workshop on Learning in Context-Sensitive Domain*, July 1996.

[43] P. Turney. The management of context-sensitive features: A review of strategies. In *Proc. ICML-96 Workshop on Learning in Context-Sensitive Domains*, July 1996.

[44] A. Tversky. Features of similarity. *Psychological Review*, 84(4), July 1977.

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

[46] P. Winston. 20 ways to contribute to the bridge project. Online resource, `http://www.ai.mit.edu/projects/genesis/projects2002.html`.

[47] P. H. Winston. Learning and reasoning by analogy. *Communications of the ACM*, (23):689–703, 1980.

[48] P. H. Winston. *Artificial Intelligence*. Addison-Wesley, third edition, 1993.