The Dissertation Committee for James Junmin Fan
certifies that this is the approved version of the following dissertation:

Automatic Interpretation of Loosely Encoded Knowledge

Committee:

Bruce Porter, Supervisor

Ken Barker

Risto Miikkulainen

Raymond Mooney

Yolanda Gil

Automatic Interpretation of Loosely Encoded Knowledge

by

James Junmin Fan, B.S., M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2006

Dedicated to my family.

Acknowledgments

I wish to thank all the people who helped me to shape this dissertation, but there is not enough space. I owe them a great deal.

First and foremost, I want to thank my advisor Bruce Porter for his support, wisdom and guidance. Bruce has helped me in every aspect of my graduate study. I have greatly benefited both from his technical expertise and from his role-model as a successful researcher.

I also want to express my gratitude to Ken Barker, who introduced me to the field of computational linguistics and the joy of golf. Ken has been a mentor and friend.

I want to thank the rest of my committee, Raymond Mooney, Risto Miikkulainen and Yolanda Gil. They have reviewed my work, asked good questions, and given me valuable feedback. Their input has given me a different perspective on my work and has given me guidance on future research direction.

I am grateful to Peter Clark for his support on KM, his advice on research and his encouragement on the development of the loose-speak interpreter. I am also grateful to Ken Murray, Vinay Chaudhri and Noah Friedland for their inputs on my research.

I could hardly have completed the thesis without my friends and col-

leagues. I will miss those long discussions on research related topics and not so research related topics. Thanks to: Charlie Benton, Jason Chaw, Bill Jarrold, Lisa Kaczmarczyk, Doo Soon Kim, Paul Navratil, Alan Oursland, Art Souther, Dan Tecuci, Steve Wilder and Peter Yeh.

Gloria Ramirez have been kind, helpful and friendly throughout. Stacy L. Miller has saved me countless hours and piles of paperwork by going out of her way to help me. Thank you.

Last but not least, I want to thank my family. I want to thank my parents for my upbringing, and I want to thank my wife, Chi Duong, for her support and understanding.

Support for this research was provided by SRI as part of DARPA's Rapid Knowledge Formation Project and PAL projects, by Vulcan Inc. as part of Project Halo and by the Boeing Company.

Automatic Interpretation of Loosely Encoded Knowledge

Publication No. _____

James Junmin Fan, Ph.D.
The University of Texas at Austin, 2006

Supervisor: Bruce Porter

Knowledge is critical for a variety of artificial intelligence problems. A key challenge in using knowledge-based systems is how to align one's encoding with the idiosyncrasies in the existing knowledge base. We call such misalignments "loose speak". We found that loose speak occurs frequently in knowledge base interactions with such regularity that it can be interpreted automatically by a machine. We created a loose-speak interpreter based on a unified approach that is capable of interpreting the different forms of loose speak, and we evaluated it through empirical studies in different domains and on different tasks.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Brittleness in knowledge interaction

It has been long recognized that knowledge is critical for solving many artificial intelligence problems. For example, resolving ambiguities during natural language processing often requires that computers have as much knowledge about the world as a human being.

Despite the value of knowledge, using it is not an easy process. In order to query or update the knowledge base, one needs to have an encoding of the query or assertion that is precisely aligned with the existing knowledge of the knowledge base. Any misalignments in the encoding may cause incorrect inference results from the knowledge base.

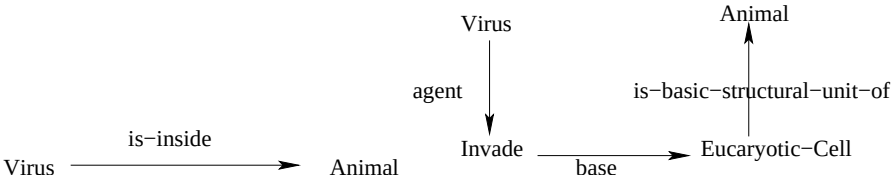
Correcting the misaligned encodings was a core problem in Project Halo [117], in which a knowledge base was built to answer Advanced Placement chemistry questions. After the knowledge base was built, during the evaluation phase, a team of knowledge engineers, programmers proficient in converting natural language specification into the formal language used by the knowledge base, spent two weeks encoding about 150 questions. Aligning the encodings with the knowledge base was a significant part of the effort. Without proper

alignment, few if any of the question encodings would be answered correctly. For example, consider the following simple question:

> Which of the following compounds is insoluble in water?
> (a) $Pb(NO_3)_2$
> (b) Li_2CO_3
> (c) $(NH_4)_3PO_4$
> (d) $Ba(OH)_2$
> (e) $BaSO_4$

The problem with this question is its underspecification. Do the five options (a-e) refer to individual molecules or to chemicals comprised of these molecules? A chemical (which is a set of molecules) has properties, such as quantity, physical state and solubility, that a single molecule does not have. However, in chemistry a chemical formula can be used to denote a single molecule in some scenarios and a set of molecules in others. If the query is incorrectly encoded as asking about the solubility of a single molecule, then no correct answer will be found. In practice, making sure the encodings align with the knowledge base is an important and consuming task.

This problem which is made all the more severe when subject matter experts, who have very little training in knowledge engineering, build knowledge bases directly, as they did in the Rapid Knowledge Formation project [7, 26, 102] whose goal was to enable subject matter experts to enter and modify knowledge directly and easily without the need for specialized training



(a) A simple encoding of animal virus.

(b) A more complicated encoding of animal virus.

Figure 1.1: Two different encodings of "animal virus". One is more impoverished, and serves as introductory material; the other contains more detail, and is used for a more specific domain. When a user encodes "animal virus", he needs to know the detailed semantics of how "animal" is related to "virus" in the knowledge base.

in knowledge engineering, and in the second phase of Project Halo [117], whose goal was "allow domain experts to formulate knowledge with decreasing dependence on knowledge engineers, and to pose questions and problems to the knowledge systems". In this dissertation, we present a solution that addresses the brittleness problem caused by misaligned encodings.

1.2 Goal

Aligning knowledge encoding with a knowledge base is not easy because it requires large amount of intimate knowledge of the idiosyncrasies of the knowledge base. Misalignments between new knowledge and the knowledge base being built are common and unavoidable because knowledge bases, like many engineered systems, are full of idiosyncratic design decisions. For example, an introductory biology knowledge base may encode an "animal virus" simply as a virus inside an animal (as in figure 1.1(a)), while a virology knowl-

edge base may encode "animal virus" in more detail as a virus that invades the cells of an animal (as in figure 1.1(b)). Without intimate knowledge about the knowledge base, one would not know the detailed semantics of "animal virus" in the context of a specific knowledge base, so it would become a source of brittleness in a knowledge base interaction.

In order to address the problem of encoding knowledge that aligns with a knowledge base, we designed and implemented a program, the loose-speak interpreter, that acts in between a knowledge base user[1] and a knowledge base. The program would check for misalignments of input encoding, make suggestions to correct the encoding when misalignments are detected, and then apply the correct encoding to the knowledge base. Figure 1.2 shows the process of knowledge base interaction with the addition of the loose-speak interpreter.

Building such an interpreter is difficult for several reasons. First, there are many ways that a user's encoding might misalign with a knowledge base. Because knowledge encoding misalignments arise from the idiosyncrasies of a knowledge base, and idiosyncrasies are unavoidable and common, it is infeasible to enumerate all possible misalignments, and it is difficult to devise a program that would detect and solve all of these misalignments.

Second, in order to detect a misalignment and resolve it, one needs to understand both the user's intended semantics and the knowledge base's

[1]The knowledge base user might be a person, such as a a subject matter expert or a knowledge engineer, or it might be a computer program, such as a natural language processing system or an agent.

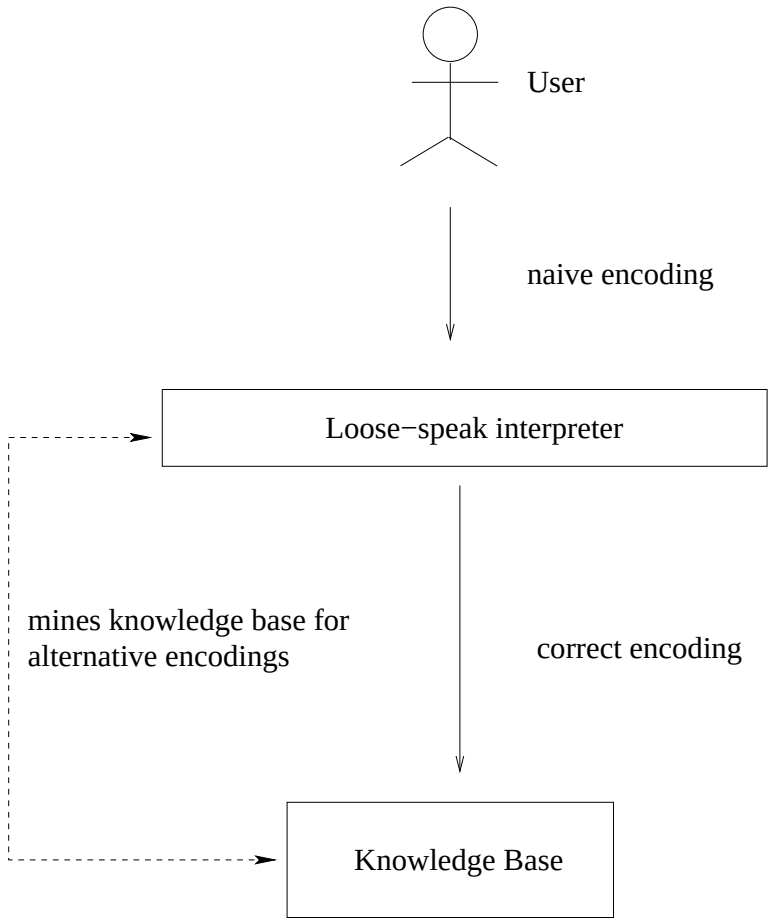


Figure 1.2: The proposed knowledge interaction process. A user, (a subject matter expert, a knowledge engineer, or even a natural language processor unit), encodes knowledge without regard to the idiosyncrasies of the knowledge base (naive encoding). User's encodings are interpreted through a loose-speak interpreter layer into something that still conveys the intended semantics of the user, but also conforms to the idiosyncrasies (correct encodings). The loose-speak interpreter finds interpretations by mining the knowledge base.

idiosyncratic requirements. A user's true intent is not available to the computer; if it were the computer could automatically formulate the intended encoding and the user would not need to encode anything. Knowledge about the knowledge base idiosyncrasies is not available because obtaining such meta knowledge and keeping it up to date is laborious and costly.

Third, the interpreter must perform highly accurately. Because the loose-speak interpreter sits between the user and the knowledge base, its mistakes will cause miscommunication that might make it ineffective. In order to be valuable to the knowledge acquisition process, the interpreter must be highly accurate.

The goal of our project is to create an interpreter that would meet the challenges described above.

1.3 Overview of the solution

Before creating a loose-speak interpreter, we empirically evaluated the frequency of loose speak occurrences and categorized them. Through two studies, we found that loose speak concentrates on a small number of types, such as metonymy, in which an entity or event is referenced by one of its features.

We have found a solution that can interpret all of the common types of loose speak that we found, and it overcomes the three difficulties described above. As illustrated in figure 1.2, the solution detects and interprets most

of the misalignments with high precision by mining the knowledge base for implicit idiosyncratic requirements.

Specifically, the interpreter uses the existing assertions in the knowledge base as examples to infer idiosyncrasies that are not explicitly stated in the knowledge base. Each time an input is received, it is compared with existing assertions in the knowledge base. If an assertion similar to the input is found, then the interpreter determines that the input aligns with the knowledge base. If no similar assertion is found, then the interpreter searches the space of variations of the input to see if an assertion closely related to the input is similar to an existing one. If so, then the transformation from the original input to the closely related assertion is an interpretation for the input. If multiple interpretations are found, then they are ranked and one is selected.

In order to implement the solution, three key pieces of technology are needed. First, we need a decomposition process. Because an input may be arbitrarily complex, it is very unlikely that a similar assertion already exists in the knowledge base. We need to decompose every input into smaller elements to increase the likelihood of finding a prior. For our solution, we chose triples to be the atomic element of an input.

Second, we need a criterion to determine if two triples are similar enough. If we use `equality` as the method to match an input triple with an existing triple, then it will be too restrictive, and the chance of finding a prior is limited. We empirically evaluated a variety of criteria and found that the `super_or_subclass` function works best.

Third, we need a search algorithm. As we search through the space of variations of the input, we need to effectively find the best match in the space. For our solution, we used a plain, breadth-first search.

1.4 Overview of the evaluation

We evaluated the interpreter from three aspects. We first evaluated the usefulness of the interpreter for reducing the brittleness in knowledge base interaction in terms of how well it corrects misaligned naïve encodings. Second, we analyzed the individual contributions of various features of the algorithm. Third, we evaluated the type of knowledge needed to support the interpreter's performance.

1.4.1 How well does the interpreter perform?

The first evaluation we conducted was to evaluate the performance of the interpreter on typical user interactions. We evaluated the interpreter on three tasks and data sets. The first came from Advanced Placement chemistry questions. We asked naïve subjects to encode the questions in a formal query language and to pose them to our chemistry knowledge base. We found that virtually all of the encodings contained occurrences of loose speak, but that our interpreter was able to correct almost all of them.

Our second task involved interpreting noun compounds, which are sequences of nouns, such as "marble statue". Noun compounds are a type of loose speak because the semantic relations among the nouns is left implicit and

must be inferred based on the knowledge base being used. Using three different data sets of noun compounds in three different domains, we found that our interpreter was able to correctly interpret most of the noun compounds.

Our third data set consists of the the data from the second phase of Project Halo. The data were generated by a natural language processing system from simplified English questions across three domains (biology, chemistry and physics). The precision of the loose-speak interpreter on these automatically generated encodings is similar to that on manually generated encodings, and the recall is lower due to natural language processing errors. This suggests that the interpreter can handle human encodings as well as automatically generated correct encodings.

The most impressive result was that our interpreter worked well on all three of these interpretation tasks across different domains.

1.4.2 Inter-input loose speak

We also applied the interpreter to resolving loose speak that extends beyond sentence boundaries. We evaluated its performance on the task of resolving indirect anaphora-type inter-input loose speak. Indirect anaphora is a type of reference that requires background knowledge in order to identify the relation between two utterances. For example, given "... a class ..." and "the student", a reader needs to have access to the background knowledge that students study in a class to associate "the student" correctly with the "class" mentioned earlier. Indirect anaphora is frequently used in discourse [94], and

resolving it is imperative for knowledge base interaction and natural language processing. We applied the loose-speak interpreter to two sets of indirect anaphora examples, and found that it achieved approximately twice the recall of the best previous system without loss of precision.

This is relevant for two reasons. First, it shows that the principle behind the interpreter is not restricted to a single utterance: it applies to loose speak across multiple sentences as well. Second, because the indirect anaphora data set uses a large vocabulary, we can use it to measure the scalability of the interpreter. We applied the interpreter to a large knowledge base derived from WordNet [44], and our test showed that search depth is not the key to finding a good solution. A shallow search is sufficient for loose-speak interpretation. Because the search complexity of a shallow search is considerably smaller, the search algorithm is more scalable.

1.4.3 What type of knowledge is needed?

A common and justifiable critique of knowledge-based systems is that they require a lot of domain specific knowledge, which is difficult to obtain. We conducted an evaluation to determine what type of knowledge is needed to interpret loose speak. If loose-speak interpretation requires detailed knowledge of the knowledge base, then the cost of acquiring this knowledge may overshadow the benefit. If loose speak can be successfully interpreted without much knowledge, then the interpreter will be much more useful in practice. We used the noun compound data sets for this evaluation.

The sensitivity of the interpreter algorithm to each hierarchical level of the knowledge base is measured through a series of ablations. The impact of each ablation is measured in terms of its impact on the interpreter's precision and recall.

The study concludes that the axioms and ontological distinctions important for this task are derived from the top levels of a hierarchical knowledge base; detailed knowledge of specific nouns is less important. This shows that the interpreter only needs domain independent, top ontological levels, which is easy to obtain, and that a large amount of domain specific knowledge is not needed.

Through the evaluations, we found that our unified solution is highly effective for various forms of loose speak, and that it can resolve loose speak using mostly the domain independent, top level ontology.

1.5 Guide to the rest of the chapters

Before we discuss the loose-speak interpreter in detail, we need to define what loose speak is. Chapter 2 defines our terms and states our thesis, which is that although loose speak takes many forms, it can be resolved with a single, algorithmic method.

Although loose speak occurs frequently in different forms, we found that it occurs with regularity. We conducted a study classifying the many different occurrences of misalignments and estimating their frequencies. Chapter 3

presents the results of our study and describes the most common types of loose speak in detail.

The regularity of loose speak occurrences has inspired a unified solution to the common types of loose speak. Chapter 4 presents the loose-speak interpreter algorithm and discusses the three key pieces of technology that the algorithm is made of.

We have evaluated the interpreter on various tasks and data sets in different domains. The details of the three evaluations can be found in chapter 5. The evaluation shows that the loose-speak interpreter is highly effective in producing the correct version of any given input. The knowledge needed for the interpreter to function comes mostly from the top levels of the ontology, and this allows users to utilize the loose-speak interpreter even if the knowledge base contains little or no domain specific knowledge.

We also applied the interpreter to resolving loose speak that extends beyond sentence boundaries. Chapter 6 describes how the interpreter can be applied to resolve indirect anaphora type inter-input loose speak, and we found that the interpreter performed with approximately twice the recall of the best previous system without loss of precision.

Related work is surveyed in chapter 7. Compared with work in related areas, the interpreter is a unified solution to a variety of problems, such as metonymy resolution, noun compound interpretation, and indirect anaphora resolution.

Chapter 8 summaries the dissertation and outlines future work.

Chapter 2

Thesis

2.1 Terms and Definitions

Before we propose a solution to the input misalignment problem, let us define some useful terms. During a knowledge base interaction session, a user encodes either assertions or queries to the knowledge base. If an assertion or a query is encoded without regard for the knowledge base being interacted with, then we call it a *naïve encoding* of the input. If, on the other hand, an encoding not only conveys the meaning of the input, but is also compatible with the knowledge base, then we call it a *correct encoding* of the input.

Loose speak is defined as the knowledge base interaction that results in a discrepancy between a naïve encoding and a correct encoding of the same input. We call the task of resolving this discrepancy – i.e., the task of aligning a knowledge encoding with a knowledge base – *loose-speak interpretation*. The goal of our project is to improve knowledge base interaction by developing ways to automate loose-speak interpretation.

Naïve encodings are often incompatible with knowledge bases, which causes numerous problems. For example, automatic reasoning may fail to pro-

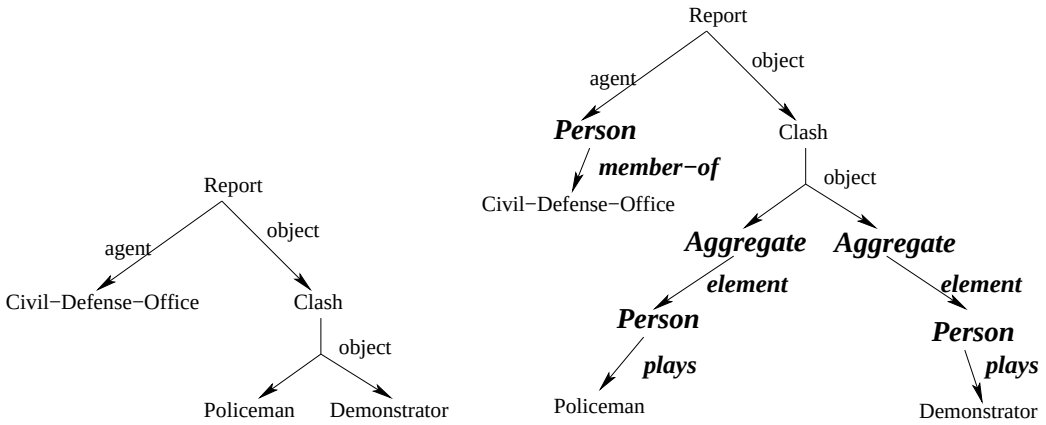
duce correct answers when incompatible knowledge is entered into a knowledge base.

This is not to say that naïve encodings are all bad. In fact typically they are the most straightforward, most faithful and most literal encoding of a piece of knowledge. Fidelity is a basic requirement of knowledge acquisition because without it the encoding may include unintended semantics. Fidelity was so important in Project Halo [117], for example, that it was separately evaluated by a panel of knowledge representation and reasoning researchers.

Here's an example that illustrates both the fidelity of naïve encodings and the problems that they cause. Take the following piece of knowledge:

(1) The civil defense office has reported a clash between policemen and demonstrators.

A naïve encoding may look like figure 2.1(a) in conceptual graph form. Although it appears to be faithful to the original input, whether or not it is loose speak depends on the knowledge base into which it is merged. For example, when merged into our knowledge base, which contains a domain-independent top ontology called the Component Library [7] (see section 2.3 for details), the encoding is indeed incorrect, as it has three problems. First, "civil defense office" is the name of an organization, and its metonymic usage refers to the people who are members of the office. By directly representing civil defense office as the agent of a Report action, it is incompatible with our knowledge base. The second problem is that "policeman" and "demonstrator" are roles played



(a) The naïve encoding. (b) The correct encoding.

Figure 2.1: The naïve and correct encoding of "The civil defense office has reported a clash between policemen and demonstrators." Notice that representation of "policeman", "demonstrator", and "civil defense office" differ in the two encodings because of the role representation idiosyncrasy, aggregate representation, and metonymy.

by people. Our ontology distinguishes a *role* from an ordinary *entity* [42], and only the latter can be the object of certain actions. Third, the naïve encoding represents a clash between *a* policeman and *a* demonstrator, but the input indicates there are more than one of each. The clash took place between a group of policemen and a group of demonstrators, which in our ontology is represented as an *Aggregate*. Figure 2.1(b) shows the correct encoding of the example.

Here's an example from a biology text book [2] used in the Rapid Knowledge Formation project [102]:

(2) In most cells, ... the network of ER tubules extends from the nucleus

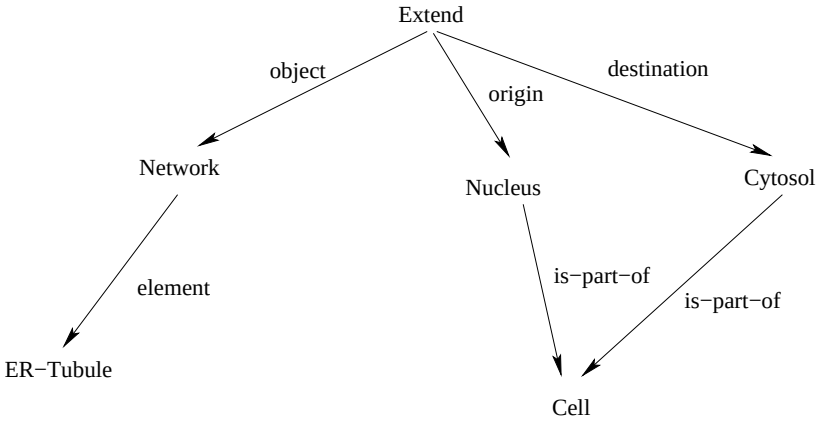
throughout the entire cytosol.

A naïve user may encode the above text as in figure 2.2(a). This encoding is quite faithful to the original input, and it is correct without the context of any knowledge bases. On the other hand, a part of a biology knowledge base may look like figure 2.3: *Cell* has two subclasses, *Eucaryotic-Cell* (a plant or animal cell which has a nucleus) and *Procaryotic-Cell* (a bacteria cell which has no nucleus). The Eucaryotic-Cell has a *Nucleus* part and a *Eucaryotic-Cytoplasm* part. The Eucaryotic-Cytoplasm part contains a *Cytosol* and other parts. If the naïve encoding in figure 2.2(a) is used for the knowledge base in figure 2.3, then two occurrences of loose speak must be resolved. First, because the knowledge base specifies that not all types of Cell have a nucleus part, the generic concept Cell used in the naïve encoding should be specialized into the correct Eucaryotic-Cell type. Second, based on the Eucaryotic-Cell partonomy hierarchy in the knowledge base, a Cytosol should not be a direct part of a Eucaryotic-Cell, but rather it is part of a Eucaryotic-Cytoplasm, which is part of a Eucaryotic-Cell. After interpreting these two occurrences of loose speak, the correct encoding is illustrated in figure 2.2(b).

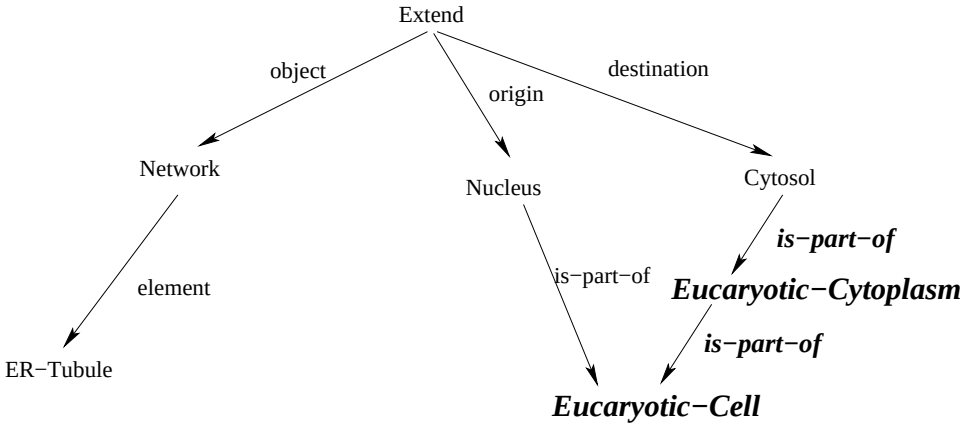
Here's an example from an Associated Press headline:

(3) Kosovo mourns for President Rugova.

In order to translate this into a logic representation, a literal parsing and encoding of the text may look like figure 2.4(a). When merged into the Com-



(a) The naïve encoding.



(b) The correct encoding.

Figure 2.2: The naïve and correct encoding of "In most cells, ... the network of ER tubules extends from the nucleus throughout the entire cytosol". Notice that the generic *Cell* concept is replaced by a more specific *Eucaryotic-Cell* in the correct encoding. Also notice the metonymic usage of the Cytosol part of the Cell.

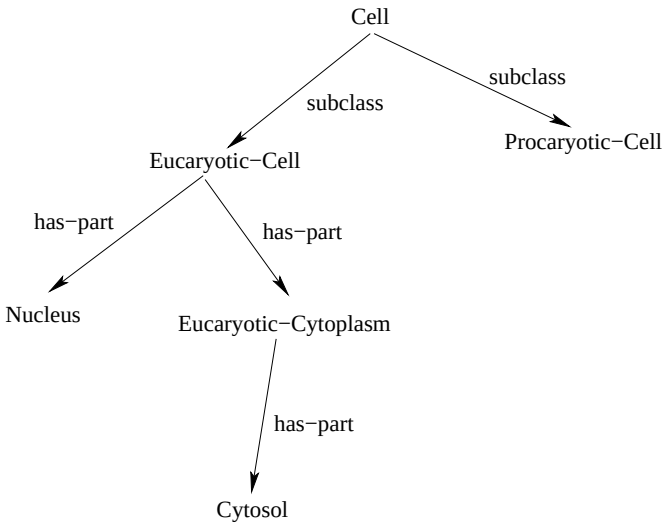
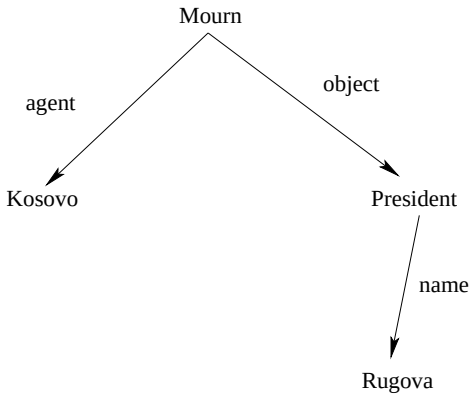
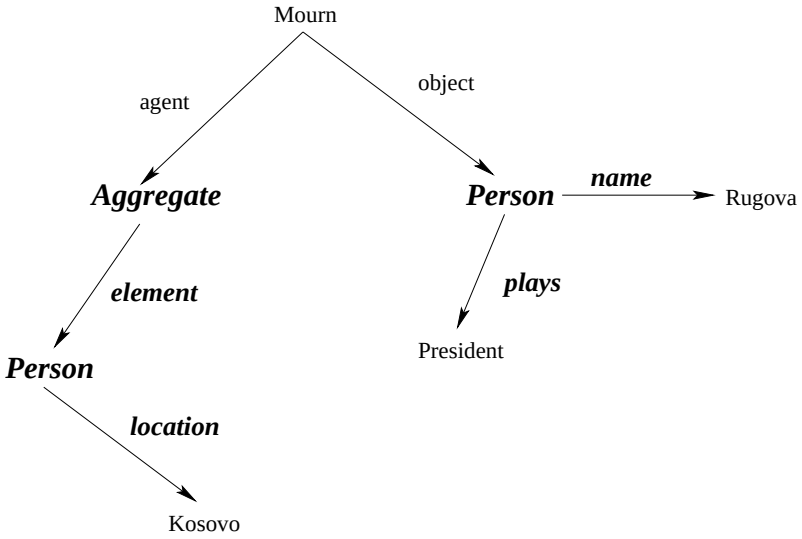


Figure 2.3: A part of a biology knowledge base that specifies the subclasses and parts of a Cell. Cell has two subclasses, Eucaryotic-Cell (a cell with a nucleus) and Procaryotic-Cell (a cell with no nucleus). The Eucaryotic-Cell has a Nucleus part and a Eucaryotic-Cytoplasm part. The Eucaryotic-Cytoplasm part contains a Cytosol and other parts.



(a) The naïve encoding.



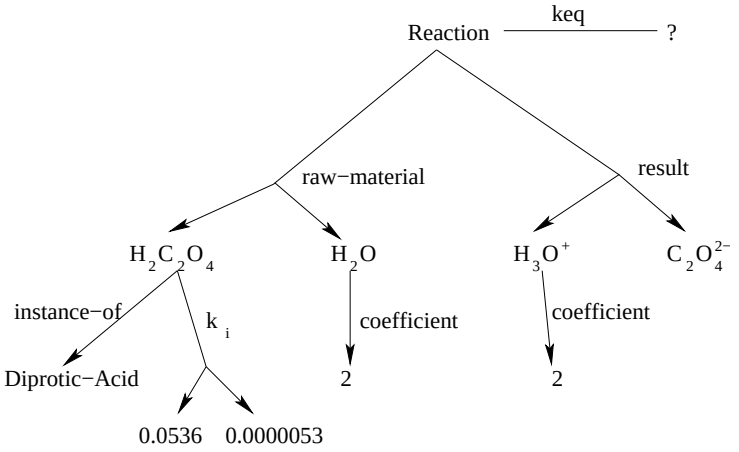
(b) The correct encoding.

Figure 2.4: The naïve and correct encoding of "Kosovo mourns for President Rugova". Notice the metonymic usage of "Kosovo" for "the people of Kosovo". Also, notice that the role President is used in the place of "the person who plays the president role".

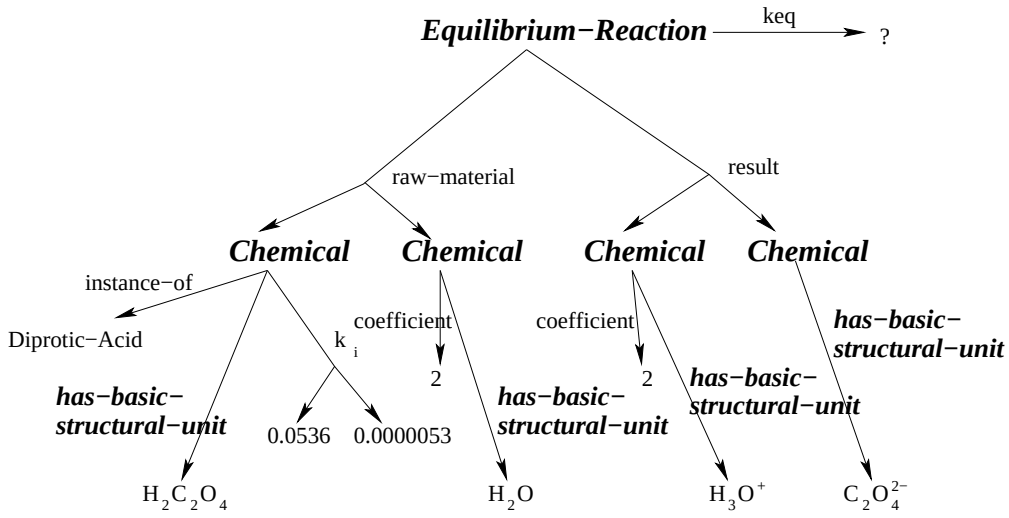
ponent Library, this version contains two problems. First, it does not take into consideration the semantic restriction that countries cannot mourn. Also, as in example (1), the role *President* is used in the place of the *Person* that plays the role. The correct encoding is illustrated in figure 2.2(b).

In addition to assertions added to a knowledge base, a query posed to a knowledge base can also misalign with a knowledge base. Here is an example from Project Halo [117]:

(4) What is the equilibrium constant of the following reaction given $H_2C_2O_4$ is a diprotic acid with $K_1 = 5.36 \times 10^{-2}$ and $K_2 = 4.3 \times 10^{-5}$, $H_2C_2O_4 + 2H_2O \rightarrow 2H_3O^+ + C_2O_4^{-2}$? A naïve encoding of the question may look like the conceptual graph in 2.5(a), in which the question mark indicates the value being queried. This encoding may be very close to the input, but when queried against the knowledge base used in Project Halo, the reasoning mechanism will not find the right answer for two reasons. First, the reaction is not classified correctly. Only equilibrium reactions have equilibrium constants in our knowledge base, so the term "reaction" in the original question is used loosely to mean "equilibrium reaction". Second, in our knowledge base, reactions (both equilibrium and non-equilibrium reactions) occur not among several individual molecules but rather among chemicals composed of those molecules. Chemicals possess properties – such as quantity, or being in a liquid or solid state – that individual molecules do not. To differen-



(a) The naïve encoding.



(b) The correct encoding.

Figure 2.5: The naïve and correct encoding of "What is the equilibrium constant of the following reaction given $H_2C_2O_4$ is a diprotic acid with $K_1 = 5.36 \times 10^{-2}$ and $K_2 = 4.3 \times 10^{-5}$, $H_2C_2O_4 + 2H_2O \rightarrow 2H_3O^+ + C_2O_4^{-2}$?" Notice that the generic *Reaction* concept is replaced by a more specific *Equilibrium-Reaction* in the correct encoding, and that the molecules are replaced by chemicals made of these molecules.

tiate the two, the concept of *Chemical*, whose basic structural unit is a single molecule, is introduced in the ontology. Furthermore, in the knowledge base a reaction is represented as one or more chemicals acting as raw materials, producing other chemicals as the result. Figure 2.5(b) shows the correct encoding of the example, which will enable the reasoning mechanism to produce the right answer.

As these examples illustrate, naïve encodings may be the most faithful encodings of input knowledge, but they may not be useful for automated reasoning because they do not align properly with the knowledge base that the reasoning methods depend on.

2.2 Thesis

Although the misalignments between naïve encodings and correct encodings occur frequently, they occur with such regularity that the various forms of misalignments can be resolved with a single algorithmic method.

The high frequency of loose speak shows the importance of solving such a problem. We believe that knowledge base idiosyncrasies are unavoidable, and misalignments between a naïve encoding and a correct encoding are rampant. Without interpreting loose speak correctly, the knowledge base cannot be utilized effectively, so automatically interpreting loose speak is an important problem.

In addition to frequency, we believe that inconsistent naïve encodings occur in regular patterns. First, they can be categorized into different types based on remedies, and a small number of loose-speak types occur more often than others. Second, they are closely related to their correct encodings. Naïve encodings differ from correct encodings because of ontology misalignments. In other words, naïve encodings and correct encodings are similar except for a few mismatches.

Because of the regularity, loose speak can be interpreted by a machine. We believe all the common types of loose speak can be interpreted using a single unified solution – exploring the close proximity between naïve encodings and correct encodings, and searching the space of encodings similar to the naïve encoding. It is very likely that the correct encoding can be found within the search space.

Loose speak usually can be interpreted using only the knowledge from the knowledge base involved in the interaction; no special "loose-speak interpretation knowledge" is required. If loose speak interpretation required additional knowledge, such as detailed knowledge about the idiosyncrasies in the knowledge base, then any gains in knowledge base interaction by loose speak interpretation systems would be erased by the efforts needed for creating the additional knowledge, and loose speak interpretation would be of little value to knowledge base interaction.

2.3 The Component Library

Loose speak always occurs in the context of a specific knowledge base; therefore, to understand our examples and empirical studies, it is important to understand our knowledge bases. Most of our knowledge bases were built on top of the Component Library. In this section, we take a detailed look at the Component Library and the various design decisions made. More details can be found in [7].

2.3.1 The Component Library design

The Component Library [7] is designed to be used by subject matter experts to author knowledge without having to write axioms. The goal is to create a resource that enables users to build knowledge bases by assembling generic knowledge components selected from a reusable library.

In order to achieve this goal, the Component Library must satisfy the following requirements: *coverage* and *access*.

2.3.1.1 Coverage

There should be enough components to allow users to encode knowledge in a variety of domains. The components in the library should be diverse enough that they can be used to compose all kinds of knowledge. The components in the library should also be specific enough that they provide sufficient support to encode detailed domain knowledge. On the other hand, the components cannot be so specific that the users are restricted or confused by the

subtle differences.

This requirement is met by selecting a small set of general concepts and richly axiomatizing each one. The set is inspired by linguistic sources, such as dictionaries and thesauri. Furthermore, the concepts are represented in such a way that new concepts can be defined through instantiation and assembly [28].

2.3.1.2 Access

In order to allow a subject matter expert to use the Component Library, it has to be accessible. This requirement is met by restricting the number of concepts (to a few thousand) and the number of relations between components (to fewer than one hundred). Such a small library is easy to learn for users with little knowledge engineering background.

2.3.2 The Component Library and its applications

The components are organized into a tree hierarchy whose top level concepts are: *Entities*, *Events*, and *Roles*.

Entities are "things that are". The entity hierarchy is relatively impoverished, but it provides the root concepts, such as TANGIBLE-ENTITY and PHYSICAL-OBJECT, for many of the concepts in specific domains.

Events are "things that happen". Events are further divided into *states* and *actions*. States are temporally stable events; while actions may put objects into states or take objects out of states. For example, the BREAK action puts an object into a BE-BROKEN state, and the REPAIR action takes the

```
(every Move has
  (object       ((a Tangible-Entity)
                 (excluded-values (the origin of Self)
                                  (the destination of Self)
                                  (the away-from of Self)
                                  (the toward of Self)
                                  (the path of Self))))
  (origin       ((must-be-a Spatial-Entity)))
  (destination  ((must-be-a Spatial-Entity)))
  (path         ((must-be-a Spatial-Entity)))
  ...
  (pcs-list     (
                 ; if origin is specified, then the object(s) are at
                 ; that location
                 (if (has-value (the origin of Self))
                     then
                     (forall (the object of Self)
                       (:triple
                         It
                         location
                         (if ((the origin of Self) isa Place)
                          then (the origin of Self)
                          else (the location of (the origin of Self))))))
 ...))

  ...
  ; Post-Condition:
  ;  1) if destination is specified, then the object is at that location
  ;     and inherits _most_ of the spatial properties of the destination
  ;     (the spatial properties not inherited are marked ;; odni
  ;  2) if origin is specified, then the object is NOT at that
  ;     location.
  (add-list     ((if (has-value (the destination of Self))
                     then
                     (forall (the object of Self)
                       (:set
                         (:triple
                            It
                            is-near
                            (the is-near of (the destination of Self)))
                         (:triple
                            It
                            is-above
                            (the is-above of (the destination of Self)))
 ...)))))
  ...
  )
```

Figure 2.6: Some of the axioms for MOVE action. It shows the selectional restriction for the object, origin, destination and the path of MOVE. It also lists the precondition (`pcs-list`) and effects (`post-condition`) of the action.

object out of that state. Each event has a rich set of axioms that describe the selectional restrictions on the participants of the event. In addition to that, actions have axioms about the preconditions and the effects of the actions. Figure 2.6 shows some of the axioms for the MOVE component.

Roles are entities in the context of events. For example, PERSON is an entity, but EMPLOYEE is a role. Someone is an EMPLOYEE only in the context of an EMPLOY event. In the Component Library, roles are linked to instances of entities when they are played by the entities or are the purposes of entities.

2.3.2.1 Language of composition

In order to effectively assemble the components, the composition must be described by clear and precise language. The language of composition in the Component Library is made of two groups: *relations* and *properties*. Relations connect entities, events and roles. They allow the Component Library users to capture the semantics of the composition of multiple components. There are fewer than eighty relations in the Component Library.

Properties link entities, events, and roles with values. For example, *temperature* is the property of an entity whose value can be cardinal (*25 degrees Celsius*) or scalar (*cold relative to drink*).

2.3.2.2 Applications

The Component Library has been used for various domains. In the Rapid Knowledge Formation projects [102], three biologists with no knowledge engineering experience were asked to encode the DNA transcription process using the Component Library. All three of them were able to use the generic components to define biological processes.

The Component Library is also used for Project Halo [117], in which it was used as the top level ontology for the chemistry knowledge base used to answer Advanced Placement chemistry test questions.

The second phase of Project Halo combines the knowledge acquisition challenge of the Rapid Knowledge Formation project with the automated reasoning challenge of the first phase of Project Halo. In addition to the chemistry domain, it includes biology and physics domains in its evaluation, and the Component Library is used as the top level ontology for all three domains.

2.3.3 Comparison with other ontologies

Compared to other ontologies, the Component Library is unique in terms of its coverage, access and semantics. For example, WordNet [44] and Sensus [65] have broad coverage and are easily accessed by the user, but they don't have deep semantics beyond linguistic information. The definitions are in free text which has limited use for computer programs, and the semantic relations are limited. In contrast, the Component Library has more semantics in each of its components, and its coverage is achieved through composition.

VerbNet [90] is a cross-lingual lexicon that concentrates on the semantics of verbs. Compared to the Component Library, it has similar emphasis on semantics, and it has additional cross-lingual knowledge. However, its coverage is limited to verbs.

Framenet [46] is a "lexical resource for English, based on frame semantics and supported by corpus evidence". Compared to the Component Library, its semantics are derived from annotations of text, and it lacks the type of compositions allowed by the Component Library.

The Cyc [70] ontology represents a different approach. It has large coverage through enumeration. This makes it difficult to access. Finding the right concept often takes a long time. Cyc also lacks some of the Component Library axioms, such as the ones that capture the effects of actions [91].

There are also ontologies, such as Ontolingua [45], that trade coverage for semantics. They have rich semantics for restricted domains, and they can be used to reason about problems specific to their domains. On the contrary, the Component Library is not tied to a specific domain, and it is intended to be used to construct domain specific knowledge bases.

2.3.4 The Component Library and loose speak

Despite the differences between the Component Library and other ontologies, there's a key commonality with regards to loose speak: it is the product of many design decisions. The Component Library is shaped by decisions on how to divide the top level ontology, what relations to include in the

composition language, and how the components should be connected through the relations. Knowledge of these idiosyncrasies in the Component Library, or any other ontology, is essential in correctly interacting with the knowledge base. Without it, a user will only end up "speaking loosely" to the knowledge base.

Chapter 3

Types of Loose Speak

The previous chapter explained how loose speak originates from the idiosyncrasies of an artifact, a knowledge base, which raises the concern that occurrences of loose speak are boundless. In this chapter, we present a study that attempts to categorize loose speak. The study shows that although loose speak is very common, it concentrates on a small number of types. We describe the most frequently used types of loose speak in detail.

3.1 Frequency study

We first evaluated the frequency of loose speak occurrences and categorized them. The evaluation was conducted through two studies. The first investigated loose speak frequencies in assertions based on a corpus study. Texts written in natural languages contain a large amount of assertional knowledge, and they are often used as knowledge sources during the process of knowledge acquisition.

We used three corpora in our study. The Brown corpus [66] consists of one million words of American English texts printed in 1961. The texts for the corpus were sampled from 15 different domains. The Alberts corpus

is a college-level cell biology text [2] consisting of 0.45 million words. The MUC 3 and 4 corpus is the data set used in the third and fourth Message Understanding Conference [33, 34]. It consists of 0.56 million words from news stories.

As stated in chapter 2, loose speak occurs in the context of an existing knowledge base ontology, hence the frequencies of different types of loose speak depend on the existing knowledge base ontology. For our study, we used the domain-independent ontology of the Component Library [7]. This ontology provides the typical top-level distinctions between entities, events, and roles, and a relatively small set of relations among these types. Although studies based on other knowledge base ontologies may yield different frequencies, we believe the results of our studies give a qualitative estimate of the distribution of frequencies of different types of loose speak.

Because the corpora are too large to inspect manually and there is no automatic way of identifying loose speak occurrences, the corpus studies were conducted based on random sampling. We randomly sampled approximately 100 sentences[1] from each of the three corpora and examined them manually. A sentence was judged to contain loose speak if its naïve encoding did not match its correct encoding. In order to apply this criterion, we needed to have a knowledge base that included all concepts from the sample so that the correct encoding of each sentence could be constructed. Unfortunately,

[1] Because the samples were generated randomly, a few of the sentences selected – such as "2." – were not actual sentences, and they were discarded.

the Component Library did not cover all the concepts used in the sample sentences. We overcame this problem by using stubs of uncovered concepts to denote their primary word sense and treated these stub-filled encodings as correct. Because these stub concepts were not as restricted as fully encoded concepts, and the correct encoding involving these uncovered concepts was less likely to differ from the naïve encoding, the number of loose speak occurrences might be underestimated in this study.

For example, for the following input from the Brown corpus [66], the encoding with stubs for "Propose" looks like figure 3.1.

(1) Austin, Texas – a Houston teacher, now serving in the Legislature, proposed Thursday a law reducing the time spent learning educational methods.

If the concept *Propose* is implemented with the restriction that the object of Propose must be an action (as in "intend" or "purport"), then the encoding contains an additional loose speak type: causal factor ("propose a law" for "propose to create a law"). However because Propose is a stub, such restriction is not included in the concept, and we cannot conclude that a causal factor type of loose speak has occurred in this example.

The second study investigated loose speak frequencies in queries based on the results from Project Halo [117]. Two sets of Advanced Placement test questions were used during the project, and both of them contain many queries. Again, we used the Component Library as the existing ontology to determine

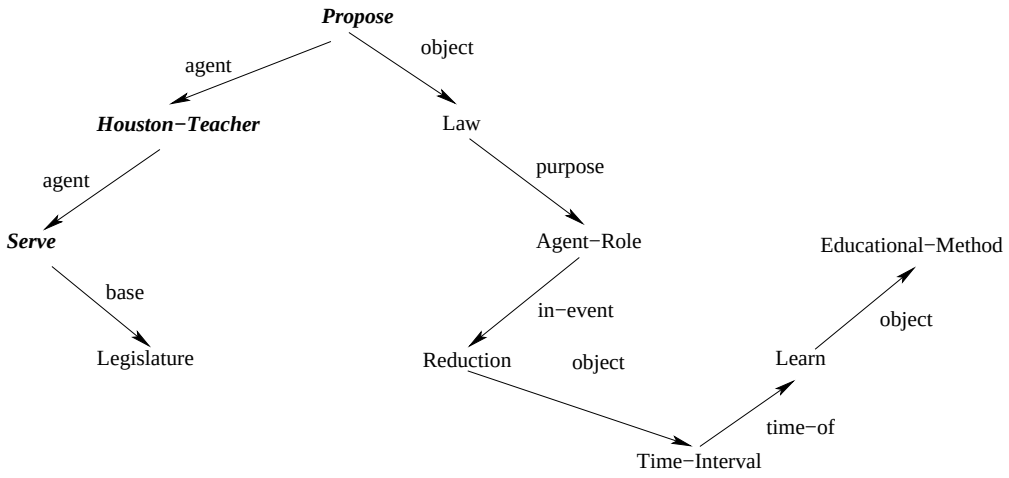


Figure 3.1: An example of encoding with stub concepts. The concepts for "Propose", "Serve", and "Houston-Teacher" are stubs. If the Component Library covered these concepts, there might be semantic restrictions that would make the encoding naïve, resulting in even more occurrences of loose speak.

the loose speak in both sets of questions, and we compared naïve encodings of the queries with the correct encodings. Because all of the concepts in the test questions are defined, no stub concepts were used for this study.

3.1.1 The results

Table 3.1 reports the results of our study. We can draw several conclusions from this study. First, loose speak is ubiquitous. The incidence of loose speak varies from 72% (MUC corpus) to 100% (Halo Final questions). Second, loose speak does concentrate on a small number of types. We found six common types of loose speak, some of which occur as frequently as 87% of the time. Third, the frequencies of the types of loose speak from the corpus

	Brown	Alberts	MUC	Halo Pilot	Halo Final
Sample size	99	96	100	44	102
No loose speak	20.2%	11.5%	28.0%	2.3%	0.0%
Metonymy	7.1%	1.0%	18.0%	25.0%	10.8%
Causal factor	0%	0%	0%	45.5%	23.5%
Aggregate	5.1%	4.2%	2.0%	77.3%	87.3%
Role	14.1%	13.5%	27.0%	9.1%	7.8%
Overly generic	0%	0%	0%	34.1%	40.2%
Noun compounds	50.5%	68.8%	86.0%	22.7%	29.4%

Table 3.1: The frequencies of loose speak occurrences in the three corpus samples and in the questions in Project Halo. The first three columns show the three corpora used; the last two columns show the two sets of questions involved in Project Halo. The second row shows the percentages of sample sentences not containing any type of loose speak. The other rows show the percentages of sample sentences containing different types of loose speak. The sum of the loose speak type frequencies exceeds 100% because each sentence in both the corpus studies and the Halo study may contain more than one loose speak type.

study and the Halo study vary significantly for two reasons. First, the stub concepts used in the corpus studies are not as restrictive as fully encoded concepts. This results in fewer discrepancies between naïve encodings and correct encodings, and hence less loose speak. Second, different domains have different distributions of types of loose speak.

In the following sections, we describe the most frequently used types of loose speak, and analyze one particular type, *role*, in detail to illustrate why knowledge engineers use it, and how it causes misalignments with the knowledge base.

3.2 Metonymy

A metonymy type of loose speak occurs when one refers to an entity or event (the referent) by one of its features or attributes (the metonym). The metonym is related to the referent through a restricted set of relations; a commonly used list of metonymic relations is compiled by [67]. It has the following relations: PART-FOR-WHOLE, PRODUCER-FOR-PRODUCT, OBJECT-FOR-USER, CONTROLLER-FOR-CONTROLLED, INSTITUTION-FOR-PEOPLE-RESPONSIBLE, PLACE-FOR-INSTITUTION, and PLACE-FOR-EVENT. There are more general definitions of metonymy that include other relations, but we define the metonymy type of loose speak to include only the ones compiled by [67].

Figure 3.2 contains an example of metonymy: a property (*conductivity*),

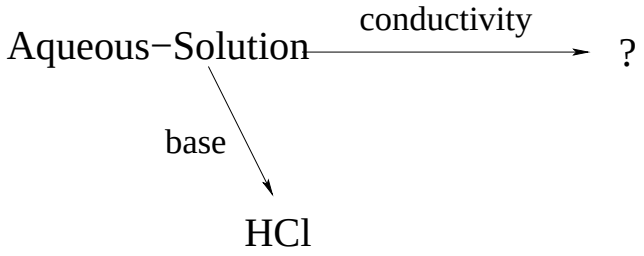


Figure 3.2: An example of loose speak from the Project Halo question set: "Is a water solution of HCl conductive?" The *?* in the figure indicates an query. With respect to the chemistry knowledge base, this encoding contains two occurrences of loose speak. First, it contains aggregate loose speak: HCl is a molecule, and a single molecule does not act as the solute of a solution because it does not have properties such as quantity. The HCl in the encoding should be replaced by a *Chemical* whose basic structural unit is a HCl molecule. Second, it contains metonymy loose speak: the *conductivity* property belongs to solutes (not solutions) because it is the solutes that determine the conductivity of a solution. The query "the conductivity of Aqueous-Solution" should be placed on the base of the Aqueous-Solution instead. Figure 3.3 shows the correct encoding of this query.

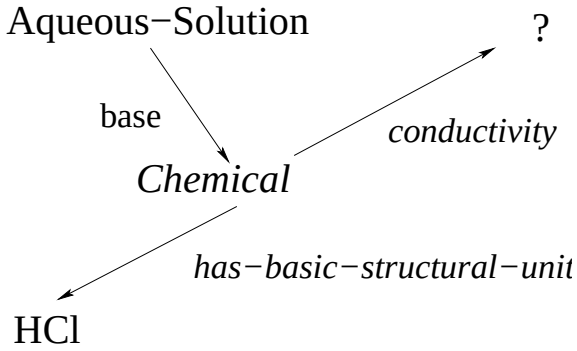


Figure 3.3: The interpretation of the query in figure 3.2 in the context of the chemistry knowledge base built for Project Halo. Notice that HCl is replaced by *Chemical*, and the *conductivity* query has been moved from *Aqueous-Solution* to the *base* of *Aqueous-Solution* to conform to the structure of the knowledge base.

which pertains to a part (solutes), is ascribed to the whole (the aqueous solution).

Metonymy occurs because knowledge engineers implement a part-whole hierarchy that is different from what a user expects. Part-whole hierarchies are a natural way to organize entities for reasoning, and different part-whole hierarchies are possible for the same set of concepts depending on the perspective. For example, a "cylinder" can be considered as a part of an "engine" or as a part of an "automobile". Knowledge engineers must choose one hierarchy which may not be the same as what a user has in mind.

Metonymy resolution could be a heavy burden on a naïve user especially when a large number of parts is involved. For example, a knowledge base of airplanes implements a hierarchy of thousands if not millions of parts of an airplane, and browsing through the parts to ensure that the correct part-whole relation is used is no trivial task.

3.2.1 Related work

Metonymy is one type of loose speak that has been studied extensively in the computational linguistic community because of high frequency of metonmyic expressions in language. Markert and Hanh [80] found metonymic expressions in 15% of the utterances in a German-language corpus of information technology test reports. They found as many as 50% of the 100 randomly drawn occurrences of "*BMW*" were used metonymically.

There are two basic approaches to metonymy interpretation. First,

there are systems that interpret metonymies based on a set of rules. These rules contain explicit knowledge about how to interpret some types of metonymy, and they are matched against inputs to produce interpretations. This approach is easy to implement because no large knowledge base is required, but it can only handle a fixed number of metonymy types.

Weischedel and Sondheimer [119] devised a set of meta-rules to interpret "ill-formed inputs", such as metonymy. Each meta-rule interprets one pattern of ill-formed inputs. For example, the rule "PLACE-FOR-EVENT" is invoked for the input "Pearl Harbor caused us to enter the war". Based on that rule, "Pearl Harbor" is interpreted as "the attack on Pearl Harbor".

Grosz et al. [51] developed TEAM, a natural language interface to databases that handled metonymy through a set of rules. For each different database, domain experts would customize the set of rules through a menu driven acquisition dialogue. For example, in an automobile database, the rule "MANUFACTURER-FOR-CAR" will be acquired and used to interpret cases in which "a Ford" is used to denote "a car made by Ford".

Lytinen et al. [75] used "mapping rules" to interpret metonymies. Unlike the meta-rules of Weishedel and Sondheimer [119], the mapping rules are invoked for any inputs. If a match in the mapping rules is found, then a non-literal interpretation is built in parallel with the literal interpretation of the input. Contextual information is used later to choose which interpretation is the most suitable.

Fass [43] listed five metonymic relations between a pair of word senses – PART-FOR-WHOLE, PROPERTY-FOR-WHOLE, CONTAINER-FOR-CONTENTS, CO-AGENT-FOR-ACTIVITY, and ARTIST-FOR-ART-FORM – and applied them to interpret metonymies. For example, given "Dick plays Beethoven", Fass' system will return a metonymic chain made of ARTIST-FOR-ART-FORM and CONTAINER-FOR-CONTENTS as the interpretation because Beethoven is an ARTIST, whose ART-FORM (music piece) is used in the place of its content (the music contained in the music piece).

The second basic approach to metonymy interpretation involves systems that interpret metonymies based on knowledge base searches. Unlike the rule-based systems, they do not contain explicit knowledge of different types of metonymy, instead they rely on searches in a general-purpose knowledge base. Our interpreter uses this approach. Although this approach has no limit on the types of metonymy it can handle, the requirement of an extensive knowledge base makes it difficult to implement. Our interpreter avoids this cost by operating in a knowledge-acquisition and question-answering setting in which a knowledge base already exists.

Browse [18] developed a metaphor and metonymy interpretation system. In his system, he represented domain knowledge as subject-verb-object triples and called them "prototypes". If the semantic categories of an input do not match with a target prototype, then a search of all prototypes is conducted to find the ones related to the constituents of the input. For example, given the input "Dick plays Beethoven" and target prototype (HUMANS PLAY

MUSICAL-INSTRUMENTS), the semantic category of Beethoven, HUMAN, does not match with the target prototype requirement. A search through all the prototypes reveals a metonymic chain made of the following prototypes (Beethoven IS COMPOSER), (COMPOSERS WRITE MUSIC), and (MUSICAL-INSTRUMENTS EMIT MUSIC). If multiple metonymic chains are found, a measure function is used to evaluate which chain is the most appropriate.

Markert and Hahn [79] implemented a metonymy interpretation system through knowledge base searching. The knowledge base they used contains a taxonomy of concepts C and a set of relations R. Given two concepts, x and y, the system searches for an acyclic path of relations $r_i \in R$ and concepts $c_j \in C$ such that each r_i is a conceptual role of c_{i-1} with $range(r_i) = c_i$ for all $i = (1, ..., n)$ and $c_0 = x \wedge (c_n\ isa\ y \vee y\ isa\ c_n)$.

To overcome the difficulties in creating large knowledge bases for metonymy interpretation, Harabagiu [54] used WordNet as the knowledge base. She used a logic form of the glosses (the textual definitions of synsets), the meronym knowledge, and the hypernym knowledge in WordNet 1.6 to detect metonymy based on type mismatch, and to search for interpretations.

Compared to these systems, our interpreter uses a similar search strategy but a different metonymy detection mechanism. Instead of being invoked when a type mismatch is found, our interpreter looks for an alternative interpretation if no prior knowledge similar to the input exists. The lack of prior knowledge is a more stringent indicator than type mismatches for some cases.

For example, if a car is represented as having an engine part and an engine is represented as having a carburetor part, then the query "the carburetor part of a car" contains metonymy even though it does not violate any type constraints, because both *Engine* and *Carburetor* satisfy the type constraint of the *has-part* relation. However, the interpreter will search for an interpretation for the query because there is no prior knowledge about the "carburetor part of a car", and it will find "the carburetor part of the car's engine" as an interpretation of the original input. The prior knowledge will uncover more metonymies than the type mismatch indicator used by these related systems, and this advantage should be reflected in a higher recall value during the evaluation of the interpreter.

In addition to these two approaches, Hobbs et al. [59, 60] investigated metonymy in an "interpretation as abduction" framework. Markert and Nissim [81] also approached metonymy as a classification problem, and applied machine-learning techniques to it. In order to do so, Markert and Nissim [82] set up a framework for annotating corpora to use (both for training and testing) for evaluating the interpretation of metonymy. Although the machine learning based approach has shown promise, it is not applicable for loose-speak interpretation because the outcome of loose-speak interpretation is not a classification, but rather a path connecting the original naïve input to the correct encoding.

3.3 Causal factor

A causal factor type of loose speak occurs when one refers to a result by one of its causes. For example, a chemistry question might refer implicitly to a chemical reaction by mentioning only the "mix", "combine" or "add" event that causes the reaction, as in the following question from a college chemistry text book [17]:

(2) Question 3.9: When the metallic element sodium is combined with the nonmetallic element bromine, $Br_2(l)$, how can you determine the chemical formula of the product?

The question should be paraphrased into the following:

(3) Question 3.9: When the metallic element sodium combines with the nonmetallic element bromine, $Br_2(l)$, how can you determine the chemical formula of the product *of the reaction caused by the combination*? Here is another example from a physics AP test :

(4) An object is thrown with a horizontal velocity of 20 m/s from a cliff that is 125 m above level ground. If air resistance is negligible, the time that it takes the object to fall to the ground from the cliff is most nearly

(a) 3 s

(b) 5 s

(c) 6 s

(d) 7 s

(e) 8 s In this example, the question asks about the properties of a *Fall* (how long it takes for an object to fall), which is caused by a *Throw* event. The Fall in question was never explicitly stated, and it is up to the user to make it clear in his encoding that the Throw causes Fall, which is being queried. It should be paraphrased into the following:

(5) An object is thrown with a horizontal velocity of 20 m/s from a cliff that is 125 m above level ground *causing the object to fall to the ground from the cliff*. If air resistance is negligible, the time that it takes the object to fall to the ground from the cliff is most nearly

(a) 3 s

(b) 5 s

(c) 6 s

(d) 7 s

(e) 8 s

From a knowledge engineer's perspective, it is more advantageous not to encode the axioms associated with the results (e.g., Reaction or Fall) onto the causes (e.g., Mix, Combine", Add" or Throw") because there are many possible causes. Associating axioms of an action with all of its possible causes is unnecessary and redundant.

The two examples illustrate that the literal encodings may describe the properties of a result by its cause. In order to have a working encoding, one

needs to correctly identify the appropriate concepts beneath the text surface to which the properties belong. Without training, a user can rarely notice such causal factor loose speak.

This loose speak type is similar to INSTITUTION-FOR-PEOPLE-RESPONSIBLE metonymy since both are related to the causes. The metonymy happens when one cause is referred to by another, related one; the causal factor takes place when a result is referred to by a cause. This loose speak type is also similar to PRODUCER-FOR-PRODUCT metonmymy because both are about the misuse of the causes and effects of an action. The metonmymy occurs when the agent of an event is confused with the result of an event; the causal factor arises when the cause of an event is mixed with the result.

3.4 Aggregate

An aggregate is a group of objects. An aggregate type of loose speak occurs when one refers to an aggregate by one of its members or vice versa. For example, in chemistry, a single molecule is different from a chemical, which is a set of molecules. A chemical has properties, such as *state* (solid, liquid, or gaseous) and *solubility*, that a single molecule does not have.

Figure 3.2 contains an example of aggregate: a single molecule (HCl) does not act as the solute of a solution. It should be replaced by a $Chemical$ whose basic structural unit is a HCl molecule.

For a naïve user, the distinction between an object and an aggregate is

often blurred. A chemist rarely notes when a molecule is used or a chemical made of that type of molecule is used.

For a knowledge engineer, such a distinction is crucial. If there is no difference between an object and an aggregate, then the knowledge base contains invalid assertions. For example, if there is no difference between a molecule and a chemical, then the knowledge base may contain an invalid axiom that states a single H_2O molecule is in *liquid* state at room temperature. Aggregate type of loose speak is not limited to the chemistry domain. Example (1) from section 2.1 illustrates an occurrence of aggregate type of loose speak in a different domain.

In order to prevent naïve users from making such mistakes, the loose-speak interpreter needs to know which properties belong to an individual object and which belong to an aggregate, and to properly replace an object specified by the naïve user with an aggregate or vice versa.

Although this loose speak type is similar to PART-FOR-WHOLE metonymy, it differs in terms of the partonomy types to which it applies. The PART-FOR-WHOLE metonymic relation usually applies to heterogeneous sets (e.g., an entity can have different types of parts, such as the various parts of a car), but the aggregate relation usually applies to homogeneous sets (e.g., each individual in the set is a basic unit of the set, such as a single soldier in an army).

3.5 Role

A role type of loose speak occurs when one uses roles interchangeably with entities. Roles are things that exist in the context of events, but unlike entities, they do not exist in isolation of the events. For example, an *Employee* is a role that exists only during the *Employment* event. On the other hand, the *Person*, an Entity, who plays the Employee role, continues with or without Employment. Because of the unique characteristic of roles, they need to be represented differently to ensure correct reasoning. The following sections detail the motivation for treating roles differently than entities and the proper representation of roles from the knowledge engineers' perspective. A naïve user is unlikely to discern the difference between roles and entities, and he is even less likely to follow the peculiarities of the role representation devised by knowledge engineers for better reasoning. Such naïve encodings must be interpreted and corrected.

3.5.1 Motivation for distinguishing roles

Although ontologies typically distinguish *Entities* (things that are) from *Events* (things that happen), it is not obvious how this division admits *Roles* (things that are, but only in the context of things that happen).

The source of the problem lies in the distinction between intrinsic and extrinsic features. **Intrinsic** features , such as *shape* and *size*, describe an entity in isolation. In contrast, **extrinsic** features describe an entity relative to other entities and events. For example, *used to strike nails* is an extrinsic

feature of a hammer because it relates a hammer to nails and striking. Efforts to represent concepts using only intrinsic features have largely failed [104], especially for representing artifacts [9].

Although the distinction between intrinsic and extrinsic features is more spectral than black-and-white, it is important to distinguish the many cases that fall into the uncontroversial extremes because they differ in significant ways. For example, an entity's intrinsic features (such as *age*) may change over time, but they are always applicable to the entity. In contrast, extrinsic features (such as the *salary* of a person) may become completely inapplicable. Moreover, unlike intrinsic features, an entity's extrinsic features may be contradictory, such as the *salary* of a person with multiple jobs. For these reasons, most psychological research on concept representation distinguishes between an entity's extrinsic and intrinsic features [104].

From these distinctions (and others we discuss later) we draw three conclusions. First, the distinction between intrinsic and extrinsic features is important; a knowledge-based system that ignores their differences might draw incorrect inferences. Second, the roles and purposes of an entity are necessarily extrinsic features; *i.e.*, they relate an entity to other entities and events. Finally, roles should be reified in any knowledge representation scheme. The representation of a role consists of those extensional features of an entity that are due to its participation in some event.

3.5.2 The difference between roles and entities

There has been considerable research on roles in data and knowledge modeling, as we summarize below. The research offers two key insights. First, entities and roles are not related taxonomically, at least not in any simple way. "Neither the roles of the real world nor the entities of the real world are a subset of the other" [5]. Guarino [52] offers two criteria for distinguishing roles from entities: (1) a role is "founded" and (2) a role lacks "semantic rigidity". Something is founded if it is defined in terms of relationships to other things. Something is semantically rigid if its existence is tied to its class; that is, if in ceasing to be of kind X, it ceases to be. For example, the concept *Food* is a role because it meets these criteria, as follows:

- *Food* is Founded: The properties of Food, such as *eaten-by* and *nutritional-value*, are extrinsic properties of the entity filling the role of food – they relate that entity to others participating in the *Eating* event, such as the *Eater*, and they are applicable only in that context.

- *Food* lacks Semantic Rigidity: An entity that might fill the role of Food retains its identity (i.e. its primary class membership) outside the context of the role. For example, a grasshopper is Food when eaten by a bird, but when it is no longer considered Food, it is still a grasshopper.

In contrast, these criteria tell us that *Person* is an entity, and not a role, for the following reasons:

- *Person* is not Founded: The properties of a Person, such as *age* and *sex*, are intrinsic features. They are defined independently of other entities and events.

- *Person* has Semantic Rigidity: when a Person ceases to be a Person, she ceases to be.

3.5.3 Roles in use

There would be little value in devising a complicated representation for roles if they did not occur frequently. To gauge how common roles are, we ran a simple experiment using English word lists.

We first extracted from a large online word list [1] nouns that ended in "-ee", "-er", "-or" or "-ist". These endings, as in *employee*, *driver*, *actor,* and *pianist*, are good cues for roles . We pruned this list to only those whose stems are also stems of base verb forms. The result was a list of more than $5,000$ candidate role names. To determine how many of these might actually represent role concepts, we sampled 109 at random. Based on the tests of foundedness and lack of semantic rigidity, 101 of the sampled nouns represented role concepts. Given that there are $74,577$ unique noun entries in the Collins wordlist, this experiment suggests that at least 6% of nouns may represent role concepts (at 95% confidence).

As a second experiment, we checked a list of the most frequently used nouns in the the British National Corpus [62]. 200 of the roughly $3,000$ most

frequent nouns represented role concepts, meaning that role concepts also account for 6% of the most common nouns. (Previous work [112] has established that there is considerable overlap among the more frequent words in different corpora).

The discrepancy between the frequencies of roles in these two studies and the frequencies in table 3.1 is caused by several factors.

First, the frequency described in the suffix study is a lower bound. The suffix filter would miss many potential roles, such as *agent* or *detective*, making the number an underestimate of roles in use.

Second, the frequency listed in table 3.1 shows the actual usage frequency. The 6% frequency from the most frequent nouns study represent the percentage of frequently used nouns that are roles, which is not the frequency of these roles being used.

Third, as mentioned in section 3.1.1, different domains have different preferences for loose speak. In news stories contained in the MUC 3 and MUC 4 data sets, role concepts, such as *spokes-person*, *attacker*, *police*, are more likely to be found, and hence this domain has a higher frequency.

3.5.4 A knowledge representation for roles

Roles are easy to identify yet they are difficult to represent. They are not merely reified names for the participants in events. Rather, roles have their own characteristics which require that they be treated differently than

entities in a knowledge representation scheme. Steimann [110] identified fifteen characteristics of roles, which we have distilled into these four:

1. Roles are created and destroyed dynamically. Because a role represents the extrinsic features of an entity due to its participation in an event, the role is created when the participation begins. If the entity stops participating, the role may cease to exist, and all its properties may no longer hold.

2. A role can be transferred between entities. For example, the role of *Manager* can be transferred from one person to another. Note that many of the role's features are transferred without change, while others must be re-computed in light of the new entity playing the role. For example, if a person earns a 20% bonus for being manager, then the *salary* feature must be recomputed should that role be transferred.

3. An entity may play different roles simultaneously, for example a *Person* may be both an *Employee* and an *Employer*.

4. Entities of unrelated types can play the same role. For example, both a cracker and a grasshopper can play the role of *Food*.

These four characteristics impose requirements on any knowledge representation scheme for roles. The next section assesses past approaches to representing roles in light of these requirements.

3.5.4.1 Previous approaches to representing roles

According to Steimann [110], previous research produced three basic approaches to representing roles. The first approach represents a role as nothing but a label assigned to a participant in an event. For example, the *employer* role labels the agent of an *employ* event. This approach is simple, but it fails to reify roles as distinct from entities (instead combining intrinsic and extrinsic properties into a single representation of an entity), which is problematic, as we discussed in section 3.5.1.

Assuming that these labels can be assigned and retracted dynamically (as entities play roles and later drop them), this approach meets the first requirement ("roles are dynamic") and the second requirement ("roles can be transferred"). The approach does not meet the third requirement ("entity can play multiple roles") because roles are not reified as predicates with arguments. Rather in this approach roles are simple propositions. Consequently the extrinsic features of an entity can clash due to the entity's participation in different events. For example, if a person has two jobs, then the two employee roles will produce two different salary values. If a query about the person's salary is posed, then it is not clear which value should be returned. Finally, this approach meets the fourth requirement ("entities of different types can play the same role"), as there are no constraints on assigning labels to entities.

The second approach, used by Sowa [108] and Uschold [76] , reifies roles and distinguishes them from entities (in that roles represent extrinsic features and entities represent intrinsic ones), then combines the two types of concepts

into a single hierarchy. They can be combined in either of two ways; both are problematic [110]:

1. The roles are subclasses of entities. For example, the role *Employer* would be a subclass of the entity *Person*, as shown in figure 3.4 (a). This becomes problematic when trying to meet the fourth requirement ("entities of different types can play the same role"). To illustrate, consider extending the hierarchy to assert that an *Employer* may be either a *Person* or an *Organization*, as shown in figure 3.4 (b). This taxonomic structure says that every *Employer* is **both** a *Person* and an *Organization* – which is not what we intended. In an effort to represent the disjunction of person and organization, we create a new type, *Legal-Entity*, which subsumes *Person* and *Organization*, as shown in figure 3.4 (c). Because an *Employer* must be a *Legal-Entity*, *Employer* must be a sibling of *Person* and *Organization*. This does not capture our original assertion that an *Employer* is **either** a *Person* or an*Organization*.

2. The roles are superclasses of the entities that play them. For example, *Employer* would subsume *Person*, as shown in figure 3.4 (d). This is clearly wrong because not every person is an employer. Moreover, it fails to meet the first requirement ("roles are dynamic"), unless the subclass relationship between entities and roles is dynamic. To avoid this paradox, some knowledge representation schemes take exactly that approach [13]. (See also *qua-classes* of KL-ONE [15], and the *existence*

subclass of SDM [84], and MERODE [105].) These schemes have the restriction that a role exists if and only if an entity is actually playing that role. This restriction makes it difficult to use role concepts to represent an entity's purpose.

The third approach represents a role as an "adjunct instance" of an entity. An adjunct instance here is a distinct instance of a role class that is coupled with the instance of an entity; the existence of a role instance implies the existence of the entity that plays the role. We adopt this basic approach, as we discuss next.

3.5.4.2 Our approach

We built a representation of roles using the adjunct instance approach to express what an entity is designed to do (its purpose), and what an entity actually does (its role). In our representation, roles are types of independent of entities. An instance of a role is played by an instance of an entity; every instance of a role exists along with an instance of an entity. The role instances are connected with the entity instances through two composition methods described in section 3.5.5. In order to retrieve values of properties that belong to a role, we need to first retrieve the role from the entity with which it is composed, and then we can retrieve the values of properties from the role.

In keeping role concepts separate from entities, the problem arises of where in the taxonomy role concepts belong. In order to avoid the taxonomy

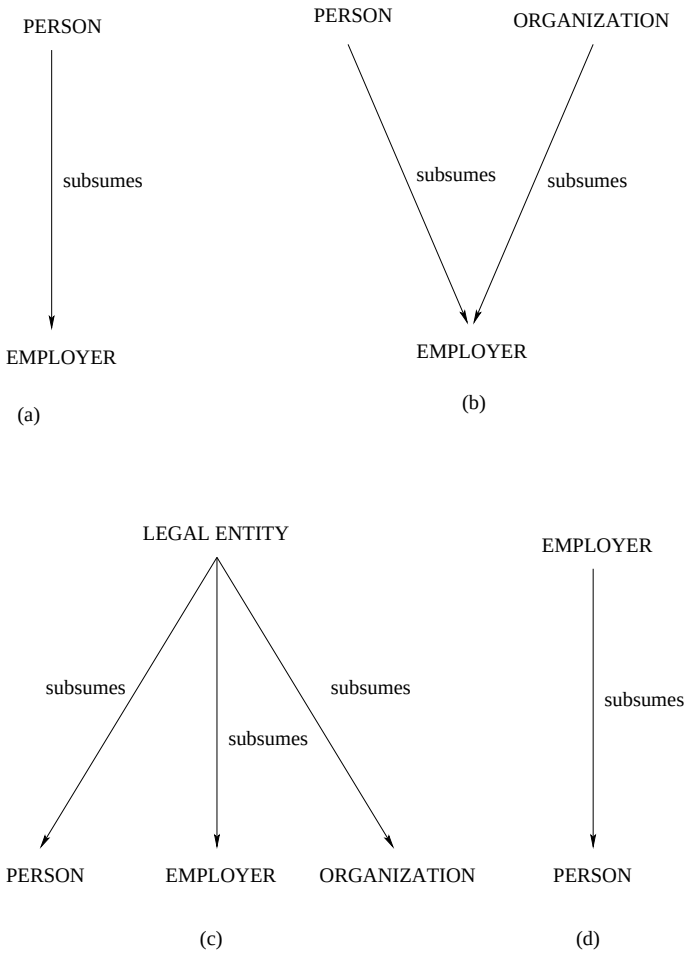


Figure 3.4: Taxonomy paradox. If roles and entities are combined into one hierarchy, none of the hierarchies above fully captures the intended information: an *Employer* can be either a *Person* or an *Organization*.

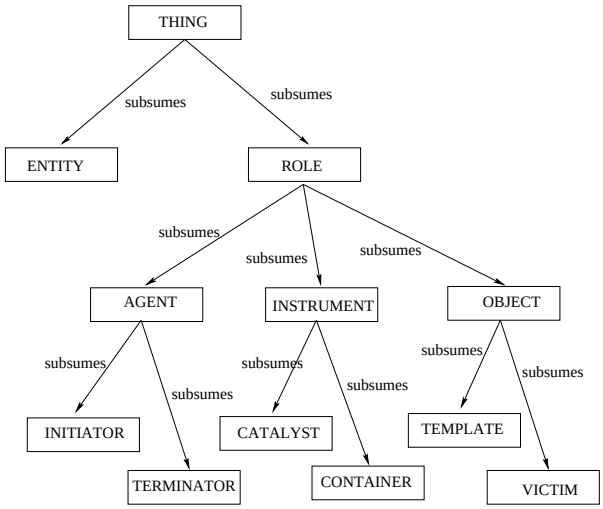


Figure 3.5: A partial listing of our role hierarchy. *Role* is a sibling of the top-level concept *Entity*, and it has several subclasses, such as *Agent*, *Instrument* and *Object*.

paradox described above, we make Role a sibling of the top-level Entity concept (see figure 3.5).

For our project, we are mainly concerned with general role concepts. Examples of such general roles include:

- *Agent*: the role played by an entity performing or responsible for an event. More specific Agent roles include *Initiator*, *Terminator*, *Creator*, and *Interpreter*.

- *Instrument*: the role played by an entity used in some event. More specific Instrument roles include *Container*, *Catalyst*, and *Connector*.

- *Object*: the role played by an entity acted upon in an event. More

specific Object roles include *Template* (Object of a *Copy* event), *Idol*, *Input*, and *Victim*.

3.5.5 Role composition

In our approach roles are types, and instances of roles are played by instances of entities; an instance of a role requires a corresponding entity. The correspondence is established with two relations: *played-by* and *purpose*. When an entity is related to a role with one of these relations, we say they are composed together.

The *played-by* composition represents that an entity is actually participating in an event. (In our knowledge representation scheme, this can be asserted to hold in a temporally bounded state.) For example, when a hammer is participating in a hammering event, the instrument role for the hammering event is *played-by* the hammer. (Equivalently, the hammer *plays* the instrument role for the hammering event.)

The *purpose* composition represents a role that the entity is intended to play, but says nothing about whether it is actually doing so. For example, the purpose of a hammer is to be the instrument of a hammering event, which is true even when the hammer is not participating in any hammering event.

Both *played-by* and *purpose* are many-to-many relations, which means that an entity can play multiple roles and a role can be played by multiple entities. Both relations are fluent, which means that an entity can dynamically acquire and relinquish roles.

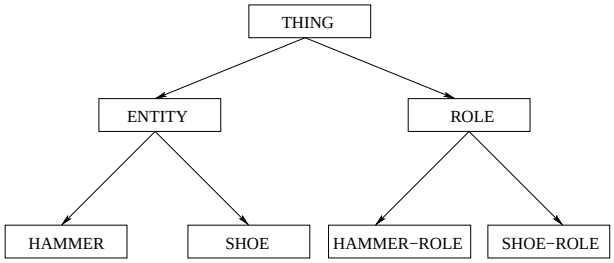


Figure 3.6: The duplication of the entity hierarchy in the role hierarchy caused by the promiscuous reification of the purpose of entities.

It is possible and common for an entity to play a role that is the purpose of another entity. For example, the purpose of a *Hammer* is to be the instrument of a *hammering* event. A *shoe* might also "play the role" of a hammer – or more accurately, play the role that is the purpose of a hammer. In order to avoid this representational gymnastics, we could reify the purpose of a hammer as a *Hammer-Role* so that the shoe plays a Hammer-role. Note that a shoe cannot "play" a hammer because hammer is an entity, not a role.

Although we *could* reify Hammer-role, that would lead to a potential problem. Reifying the purpose of entities promiscuously will result in duplication of the entity hierarchy in the role hierarchy (figure 3.6). The duplication problem is inherent in any representation of purpose. In practice, however, it is not a serious issue because most roles need not be reified. Our criterion is to reify only those roles, such as *Container*, that are likely to be played by many different kinds of entities, not just those entities whose purpose is to play the role.

Non-reified roles are specialized instances of generic roles, and they are

left unnamed. Specialization is accomplished through the addition of properties or constraints on an instance of the generic roles. As an example of composition, the purpose of a hammer might be represented as follows:

$$\forall x\ isa(x, Hammer) \rightarrow$$
$$\exists y, z\ \ isa(y, Instrument) \wedge$$
$$isa(z, Hammering) \wedge purpose(x, y) \wedge in{-}event(y, z)$$

The Skolem variable y is an example of a non-reified role.

By using a combination of *purpose* composition and non-reified role concepts, we can avoid the problem of duplicating the entity hierarchy in the role concept hierarchy. For example, a representation of using my shoe as a hammer would be:

$$\exists p, h\ \ isa(myShoe, Shoe)$$
$$\wedge isa(h, Hammer) \wedge plays(myShoe, p) \wedge purpose(h, p)$$

The Skolem instance p is a non-reified role denoting the purpose of a hammer. It is used to express that $myShoe$ plays that role. That is, $myShoe$ plays the role which is the purpose of a hammer.

3.5.6 Discussion

The distinction between *Entities* (things that are) and *Events* (things that happen) is clear and common in ontologies, but it is decidedly less clear how to handle *Roles* (things that are, but only in the context of things that happen). Although roles are often confused with entities and mixed together in a single hierarchy, we draw from the data modeling literature an operational distinction between them. Using this distinction we determine that roles are frequently used in English text, accounting for more that 6% of the most common nouns. We describe our representation in which roles are reified, and instances of roles are composed with the entities that participate in them. This approach meets all four requirements for role representation, but it also imposes two additional burdens for a user. First, the user needs to distinguish between roles and entities. Second, the user needs to use *plays* relation instead of *isa* relation between an entity and a role. Such additional requirements make it difficult for naïve users' encodings to align with the knowledge base idiosyncrasies.

3.6 Overly generic

Sometimes a generic concept is used naïvely to denote the meaning of a specific concept. For example, Reaction" is used to denote an Equilibrium-Reaction. Or in the context of a discussion on live oak, the specific concept, *Live-Oak*, is often referred to as a more generic concept, such as *Oak* or *Tree*.

This type of loose speak occurs because it is natural to use different

terms that coreference the same concept in human discourse. One frequently used type of coreference is to use the superclass of a concept to denote the class itself. For example, using "the vehicle" to denote a "sedan" mentioned in the context. For a naïve user, it is natural to encode knowledge in a similar fashion.

From the perspective of a knowledge engineer, the distinction between a specific concept and its superclasses is an integral part of the knowledge base. If an overly generic class is used, then properties that belong to the specific class will not be available. For example, the axioms about how to compute *equilibrium constant* are only available to *Equilibrium-Reaction*, not the more generic class *Reaction*. The missing properties may cause the reasoning mechanism of a knowledge base to fail. In order to allow naïve users to naturally encode knowledge and to ensure the correctness of the reasoning process, a loose-speak interpreter must replace the overly generic concept with the correct subclass.

3.6.1 Related work

Indirect anaphora (also known as associative anaphora or bridging) arises when a reference other than direct mention is used. One type of indirect anaphora is based on subsumption relation. For example,

(6) ***Rice*** is a dietary staple of more than half of the world's human population. ***The plant*** grows to 1-1.8 meter tall.

The expression "the plant" relates to the antecedent "rice" through a subsumption link. Such anaphora is an important type of indirect anaphora, and it is resolved by anaphora resolution systems. A more detailed discussion of various approaches to indirect anaphora can be found in section 6.3.

3.7 Noun compound

A noun compound is a sequence of nouns composed of a head noun and one (or more) modifiers. The head noun determines the type of the whole compound (with few exceptions), and the modifiers specialize the type from the head noun.

A noun compound type of loose speak occurs when a user uses a pair of nouns without specifying the semantic relations between them. This type of loose speak is frequently found for several reasons. First, it is natural and efficient to use noun compounds in natural language.

Second, the interpretation of noun compounds is idiosyncratic. For each noun compound, there are dozens of possible semantic relations to connect the head noun with the modifier; an interpretation of the noun compound is correct only in the context of a specific knowledge base. For example, the correct interpretation of *animal virus* in the knowledge based built for the Rapid Knowledge Formation project is "a virus that is the agent of an invade, whose object is a cell that is the basic structural unit of an animal". Finding this interpretation involves a hidden predicate (invade) and multiple semantic relations (agent, object, is-basic-structural-unit-of).

Relation Name	Example
material	marble statue
object-of	troop movement
isa	pine tree
location	sea mammal
product	protein production
part-of	human lung

Table 3.2: Some of the common semantic categories used in noun compound studies in computational linguistics.

Third, some noun compounds' semantic relations are so obvious that it is a cumbersome task for a user to state them explicitly. For example, given *car engine*, it is obvious that the engine is part of a car, but the user is required to state the *part-of* relation explicitly every time the noun compound is encountered.

For a knowledge base, noun compounds are unacceptable because of the ambiguity in how the head noun and the modifier are related. Such ambiguities are not allowed by most reasoning mechanisms.

The loose-speak interpreter faces the daunting task of disambiguating the semantic relations between the head noun and its modifiers no matter how obvious or how obscure the relations may be.

3.7.1 Related work

The computational linguistics community has studied noun compound interpretation extensively [8, 37, 39, 47, 68, 71, 72, 109, 115]. In these studies, noun compound interpretation is treated as a classification problem. The task

is to select a single semantic category for each pair of nouns. The selection is usually made from a list of about 20 semantic categories, such as *part-of, material* and *object-of*. See Table 3.2 for examples.

Our task is more general. Rather than selecting a single semantic category, our task is to find a sequence of semantic relations that links the nouns in a compound. Semantic relations are a list of about fifty thematic roles such as *agent, object, has-part, location*, and so on. For example, given *animal centrosome*, a traditional interpretation may classify this as a location category (an *animal centrosome* is a centrosome inside an animal). A loose-speak interpretation may be composed of a combination of semantic relations, such as: "an *animal centrosome* is a centrosome that is inside a eucaryotic cell, which is the basic unit of an animal".

Furthermore, computational linguists have approached the noun compound interpretation task armed with lots of examples, but little or no knowledge about the constituent nouns. Typical solutions are based on statistical patterns discovered in the corpus of examples. In contrast, we approach the task in the context of deeper knowledge of the constituent nouns – their taxonomic classification, at least – but few examples of noun compounds, let alone a corpus.

3.8 How different types of loose speak are related

The previous sections have shown the most frequently used types of loose speak, and they are quite different in three aspects. First, they differ in

terms of underlying idiosyncrasies. Some, such as the separation of cause and effect, are designed by knowledge engineers for the sake of cleaner conceptualization. Others, such as the representation of roles, are created for sound reasoning. Some idiosyncrasies, such as partonomy (the designation of parts), are created in a more arbitrary manner. None of the idiosyncrasies are obvious to a naïve user, and it is unreasonable to expect that a user's encoding would align with the idiosyncratic requirements.

Second, the types of loose speak differ in terms of why naïve encodings are used. A user may encode naïvely because it is closer to natural language discourse, as is the case for *overly generic* type of loose speak. A user may also encode naïvely for efficiency, as is the case for *noun compound* type of loose speak.

Third, the types of loose speak differ in terms of what type of relations are needed for interpretation. The different relation types range from *has-part*, *causes*, *subclasses*, etc.. The variety in relation types makes it more difficult for an interpreter to find the correct encoding of a naïve user's encodings.

Despite the differences, these types of loose speak share one important feature: proximity. For all of these types, a naïve encoding is only slightly different from the correct encoding. In fact, when viewed independently of any knowledge base idiosyncrasies, the naïve encodings are quite reasonable to convey the intended semantics. When in the context of a knowledge base and its idiosyncrasies, the naïve encoding is only a few modifications away

from the correct encodings. The types of modifications are numerous, but the total number of modifications needed for a given naïve encoding is limited.

Chapter 4

The Algorithm

In the previous chapter we have presented a frequency study, in which we identified frequently used types of loose speak and described them in detail. The various types of loose speak differ in three aspects. First, they differ in terms of the underlying nature of the knowledge base idiosyncrasy. For example, the role representation idiosyncrasy is designed for sound reasoning, and the separation of cause and effect is for cleaner conceptualization. Second, they differ in terms of the motivation for speaking loosely. One may use an overly generic concept because it is closer to natural language discourse, or one may use noun compounds because it is more efficient. Third, they differ in terms of the types of relations needed for their interpretations, which include *has-part*, *causes*, *subclasses*, etc. Despite the differences, we have developed one algorithm that can effectively interpret all of them.

4.1 Inputs and outputs

Before we introduce the algorithm of the loose-speak interpreter, we define its task by its inputs and outputs. The inputs to the interpreter are knowledge structures, represented formally. The input knowledge often comes

from sentences in natural language, such as examples (1), (2), (3), and (4) in section 2.1. A piece of knowledge is translated into a formal language by a naïve user, either a human or a natural language processing system that encodes naïvely, without regards to the idiosyncrasies of a knowledge base. The formal language can be any logic-based symbolic representational language. In our implementation of the loose-speak interpreter, we used KM [95], a frame-based language with clear first-order logic semantics. There are two basic types of KM constructs: assertions and queries. Additionally, KM also provides a variety of representational structures, such as conditional statements, aggregates and universal quantifiers. To avoid issues of KM syntax, we will illustrate our solution using conceptual graphs [108].

The outputs are an interpreted version of the inputs. They are written in the same representational language as the input, and they convey the same intended semantics of the user's encoding. Unlike the inputs, the outputs are created based on the idiosyncrasies of the knowledge base, and they align with the knowledge base. The user can then apply the outputs to the knowledge base without conflicts.

Although our interpreter is developed to take inputs in the KM language, the algorithm does not depend on KM. It applies to other formal languages that have similar constructs for assertions and queries.

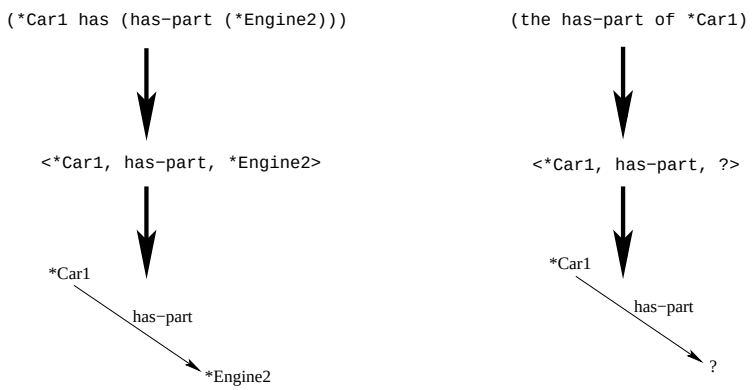


Figure 4.1: Two examples of transforming KM statements into triples, and then into conceptual graph edges. The assertion that `*Car1` has `*Engine2` as its parts is converted into the triple ⟨**Car1, has-part, *Engine2*⟩, which is transformed into a directed edge from **Car1* to **Engine2* with the edge label *has-part*. Similarly, the query about the parts of `*Car1` is translated into the triple ⟨**Car1, has-part, ?*⟩ and directed edge from **Car1* to *?* with the edge label *has-part*.

4.2 Input decomposition

As described in the previous section, the input to the loose-speak interpreter can be an arbitrarily complex structure in a formal language. For example, the input could be the naïve encoding of the context of an Advanced Placement question, such as in example (4) in section 3.3. Such inputs may contain a large number of components made of a wide variety of structures. In order to handle an input of arbitrary length and complexity, we need to decompose it into simpler and more uniform constructs and then to process them one at a time.

For our algorithm, we decomposed an input into a set of binary predi-

cates (triples) in the form of $\langle C_1, r, C_2 \rangle$ where C_1 and C_2 are classes or instances of classes and r is a relation. Queries are decomposed into the form of $\langle C_1, r, ? \rangle$ where ? is the value to be queried. A triple corresponds to a labelled edge in a conceptual graph, in which the edge label represents r, and the two vertices represent C_1 and C_2. Figure 4.1 shows two examples of KM to conceptual graph transformation. The assertion that `*Car1` has `*Engine2` as its part is converted into the triple ⟨*Car1, has-part, *Engine2⟩, which is equivalent to a directed edge from *Car1* to *Engine2* with the edge label *has-part*. Similarly, the query about the parts of `*Car1` is translated into the triple ⟨*Car1, has-part, ?⟩, which is transformed into a directed edge from *Car1* to *?* with the edge label *has-part*. Section 4.2.1 describes how other types of KM structures can be decomposed into the triple format. The decomposition of formal encodings into triples is common; for example it is used in the Resource Description Framework (RDF) specification [10] for the Semantic Web.

4.2.1 The transformation of other structures into triples

In addition to assertions and queries, KM provides a variety of constructs. These constructs are not just limited to KM, as they often appear in other knowledge representation languages. Therefore it is important to show that the input decomposition process works with other constructs as well. In this section, we will illustrate how other types of constructs can be transformed into triples.

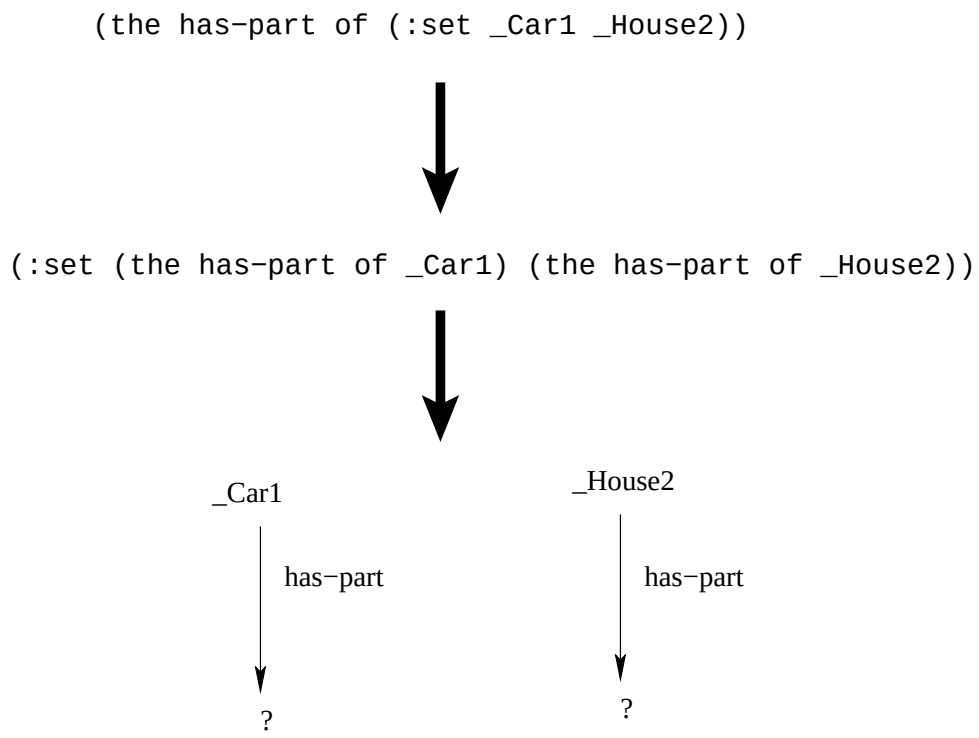


Figure 4.2: The transformation of a set aggregate in KM. In this example, a query expression, `(the has-part of ...)`, applies to a set aggregate, `(:set _Car1 _House2)`, and it can be transformed into a set of query expressions, which in turn can be transformed into a conceptual graph representation.

4.2.1.1 Aggregates

The aggregate construct in KM is a way of representing multiple elements that are referred to collectively, such as a set, a sequence or a pair. An aggregate is the basic representational construct that corresponds to a plural in natural language, and it is common in other representational languages. To transform an aggregate, we can take each element from the aggregate, and use it to form an aggregate of triples. Figure 4.2 gives an example of transforming

a set into triples.

4.2.1.2 `isa`

Another important type of construct is the `isa` operator and the subsumption test operator. The `isa` operator tests to see if an instance belongs to a class, and the subsumption operator tests to see a class subsumes another. Unlike assertions, `isa` returns a true or false answer, and unlike queries, `isa` has all of the elements of the binary predicates available. However, as with assertions, this type of construct can be transformed into triples of $\langle C_1, isa, C_2 \rangle$ or $\langle C_1, subsumes, C_2 \rangle$.

4.2.1.3 Conditionals

Conditional expressions are in the form of `(if` ***expr1*** `then` ***expr2*** `[else` ***expr3***`])`. This type of conditional statement is common among most programming languages. A conditional statement can be transformed into three sets of triples which result from the transformation of ***expr1***, ***expr2***, and ***expr3***. Figure 4.3 shows an example of transforming a conditional statement into three sets of triples.

4.2.1.4 Universal quantification

KM allows universal quantification expressions, which allow users to select from a set of elements and perform operations on them. A typical universal quantification expression has the following form:

```
(if (_X12 isa Boat) then
   (X_12 has (has-part ((a Propeller))))
 else
    (_X12 has (has-part ((a Wheel)))))
```

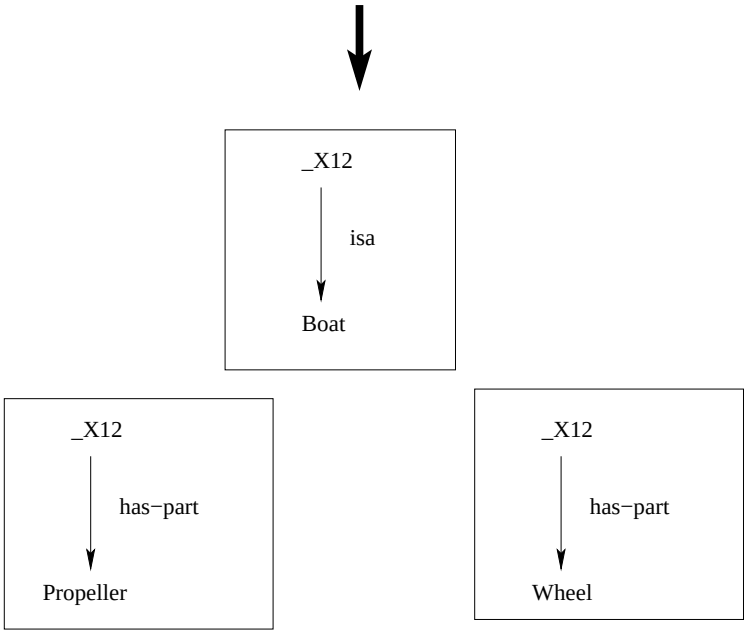


Figure 4.3: The transformation of a conditional statement in KM. In this example, the conditional statement, `(if (_X12 isa Boat) then ...)`, is split into three sets of triples.

```
Given a triple of the form <C1, r, C2>:
   RESULT = nil
   If constraint_violation (<C1, r, C2>) then
      RESULT = repair(<C1, r, C2>)
   else if not(resemblance_test(<C1, r, C2>) )then
            RESULT = repair(<C1, r, C2>)
         end if
   end if
   If RESULT then
      Return RESULT
   else
      Return <C1, r, C2>
   end if
```

Figure 4.4: The search algorithm used to interpret one triple. The algorithm detects loose speak by checking the domain and range constraints and checking to see if the input expression is similar to any existing knowledge. If loose speak is found, then it calls the search routine to search for encodings similar to the input, and it returns the search result if any is found, or the original input if nothing else is found.

(forall *expr1* where *expr2* *expr3*)

It selects all the elements in the set returned by ***expr1*** that satisfy ***expr2*** and applies ***expr3*** to those elements. Similar to conditionals, universal quantification expressions can be transformed into three sets of triples that correspond to ***expr1***, ***expr2*** and ***expr3***.

4.3 Test-and-repair

Before we introduce the order of processing of all the triples from the input decomposition, let's first take a look at how each triple is tested and re-

paired for loose speak. Figure 4.4 shows the algorithm for the `test-and-repair` procedure. The test determines whether the triple contains loose speak. In order to tell for certain whether an input contains loose speak, one needs to compare the naïve encodings with the correct encodings. Unfortunately, the correct encodings are unknown to the interpreter, so the test has to use a heuristic function to detect loose speak. Our interpreter uses the following:

> If the triple violates structural constraints, then it must contain loose speak (because correct encodings are consistent with the structure of the knowledge base).

The only structural constraints used by the interpreter are based on the domain and range of the relation in a triple. The constraint violation reflects the common approach of metonymy detection based on *selectional restriction violations*. In the field of metonymy resolution, a metonymy is usually recognized by a violation of the semantic restrictions that predicates impose on their arguments. Figure **??** shows the algorithm. Specifically, for a triple $\langle C_1, r, C_2 \rangle$, the domain of r must subsume C_1, and the range of r must subsume C_2.

Although the constraint check returns many true positive cases, it also may return many false negatives. Some researchers [43, 80, 89] have pointed out that selectional restriction violation (or constraint check) is not sufficient for all occurrences of metonymy. For our interpreter, the constraint check is complemented by a resemblance check. Figure 4.5 shows the algorithm.

```
Given a triple of the form <C1, r, C2>:
if <C1, r, C2> is a query and there is an answer for the query
   then
      return true
   else
      if exists a triple <C1', r, C2'> in the knowledge base such that
         C1' subsumes or is subsumed by C1 and
         C2' subsumes or is subsumed by C2
      then
         return true
      else
         return false
```

Figure 4.5: The resemblance check for a triple. It is based on the hypothesis that if the input does not resemble any existing knowledge, then it may contain loose speak, because studies have shown that "one frequently repeats similar versions of general theories" [29].

The resemblance check is based on the hypothesis that if the input does not resemble any existing knowledge, then it may contain loose speak, because studies have shown that "one frequently repeats similar versions of general theories" [29]. Even though the knowledge resemblance test may return many false positives, it returns many true negatives as well. The knowledge resemblance test complements the constraint check by ensuring a triple that is judged not to contain loose speak is truly loose speak free.

The resemblance test, applied to a triple, is implemented as follows. If the triple represents a query (i.e., the third element of the triple is "?"), the triple passes the test only if the knowledge base can compute one or more fillers for the tail. (This is the basic question-answering function of a knowledge

```
Given a triple of the form <C1, r, C2>
   RESULT = specialize_head(<C1, r, C2>)
   if not(RESULT) then
       RESULT = specialize_tail(<C1, r, C2>)
   return RESULT
```

Figure 4.6: The `repair` procedure. It repairs a given triple by calling the `specialize_head` and `specialize_tail` procedures.

base.) Otherwise, the triple $\langle C_1, r, C_2 \rangle$ passes the test only if the knowledge base contains a triple $\langle C_1', r, C_2' \rangle$, such that C_1' subsumes or is subsumed by C_1 and C_2' subsumes or is subsumed by C_2.

If loose speak has been detected in a triple, it is repaired by searching the knowledge base for similar triples. The search is conducted through two specialization procedures (see figure 4.6). The specialization steps search the subclasses of C_1 or C_2 to see if they are related through the relation r. Figure 4.7 shows that the `specialize_head` procedure is made of three searches: `search_head`, `search_tail`, and `bi_dir_search`. `Specialize_tail` is identical to `specialize_head` except that it specializes C_2 instead of C_1.

The `search_head` and `search_tail` procedures perform breadth-first searches. Given the triple $\langle C_1, r, C_2 \rangle$, the `search_head` function starts at C_1, traverses all semantic relations, such as "has-part", "agent", and "subclasses", and stops when a suitable instance is found. An instance C_3 is suitable if it is a superclass or a subclass of C_2 and the path from C_1 to C_3 contains the relation r. The successful search path is returned as the interpretation for the input. Figure 4.8 gives an example of `search_head`.

```
Given a triple of the form <C1, r, C2>
   DEPTH = 0
   RESULT = nil
   While DEPTH < MAX-DEPTH and not(RESULT)
      RESULT = search_head(<C1, r, C2>)
      If not(RESULT) then
         RESULT = search_tail(<C1, r, C2>)
         If not(RESULT) then
            RESULT = bi_dir_search(<C1, r, C2>)
         end if
      end if
      C1 = get_subclasses(C1)
   end while
   return RESULT
```

Figure 4.7: The `specialize_head` procedure. It specializes C_1 of the triple $\langle C_1, r, C_r \rangle$ into its subclasses, and calls the `search_head`, `search_tail` and `bi_dir_search` procedures.

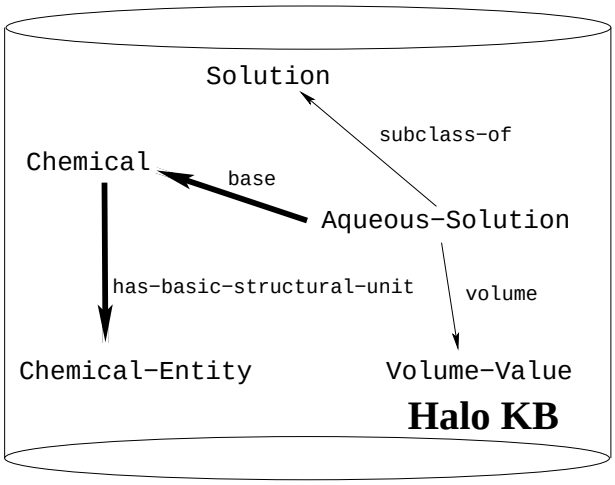


Figure 4.8: The application of the "search-head" function on the triple $\langle Aqueous\text{-}Solution, base, HCl \rangle$. The function is called because this triple does not resemble any existing path in the knowledge base. This occurrence of loose speak is resolved by breadth-first search from *Aqueous-Solution*, along all the semantic relations such as *volume*, *base*, *subclasses*, until a HCl or a superclass or subclass of HCl, is found. In this example, the search stops at depth two, and returns the new triple $\langle Aqueous\text{-}Solution, base, \langle Chemical, has\text{-}basic\text{-}structural\text{-}unit, HCl \rangle \rangle$ because HCl is a subclass of *Chemical-Entity*. The bold lines in the above figure shows how the result is found.

To avoid unintended interpretations, the set of semantic relations traversed by the search does not include the "superclasses" relation. If both "subclasses" and "superclasses" are included in the search path, then any concept can be found from C_1 by climbing up and down the taxonomy, and a large number of spurious interpretations may be returned for the given input.

If multiple suitable instances are found, then the algorithm ranks them and selects the most appropriate one. The ranking scheme is based on the Occam's Razor principle, which states that the simplest answer is usually the correct answer. The complexity is measured by the length of the path from C_1 to a found instance: a shorter path is ranked higher than a longer one. Shorter paths are considered simpler because it is easier to find the suitable instance from C_1. Preferences are given to paths containing essential relations as determined by valency theory [106]. Valency (or valence) refers to the arguments of a verb, and the relation between the verb and its arguments, such as *object* and *agent*, are the essential relations.

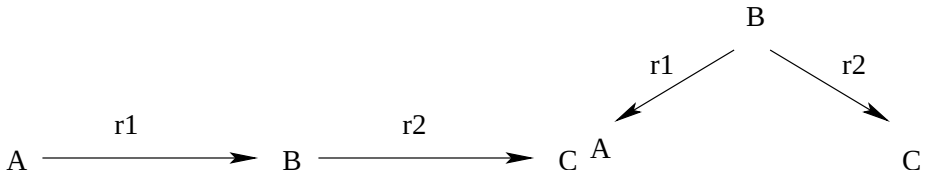
The loose-speak interpreter has two modes of selecting the answer. The automatic mode always chooses the shortest path as the answer; the manual mode presents all the answers sorted by path length and prompts the user to select one.

The `search_tail` function is implemented similarly to a search that starts at C_2. The `bi_dir_search` starts from both C_1 and C_2.

The search is conducted in a breadth-first manner for two reasons. First, because each occurrence of loose speak is a naïve encoding that has a limited number of differences from its correct encoding, a correct interpretation should be closely related to the naïve input. A very deep search will only return encodings that are not closely related to the input. If only a shallow search is needed, then a brute force search such as breadth-first is sufficient. The maximum search depth in our system is set to four. Second, a breadth-first search returns the shallowest interpretation without examining the whole search space. Such breadth-first searches have been shown to be effective in interpreting noun compound type of loose speak, given a top level ontology that contains the fundamental axioms [41].

If no interpretation is found when the depth cutoff is reached, the interpreter returns the original encoding. In addition, if loose speak was detected by a constraint violation, the interpreter returns an error report.

This algorithm applies to all commonly used loose speak types. Metonymy, role, causal factor, aggregate and overly generic types of loose speak are represented as labeled triples, and `specialize_head` and `specialize_tail` are applied if a triple fails either the constraint test or the resemblance test. A noun compound is represented as an unlabeled triple of the form: $\langle ModifierClass, nil, HeadClass \rangle$. Because there are no unlabeled triples in the knowledge base, noun compounds will fail the resemblance test, and consequently trigger `search_head` and `search_tail`.



(a) Linear dependency. (b) Tree dependency.

Figure 4.9: The two types of conflicts resulting from specialization. Linear dependency is when concept B is specialized into a specific class because of relation r_1, and the new class may conflict with another relation r_2. Tree dependency happens when concept B is specialized into a specific class because of relation r_1, and the new class may conflict with another relation r_2.

4.3.1 `Look-Ahead`

The `test-and-repair` procedure works well in most cases, but it does not address a possible conflict with occurrences of loose speak of the "overly generic" type. An overly generic concept is sometimes used to denote a specific concept. For example, given the triple $\langle \textit{Reaction}, \textit{dissociation-constant}, 1.5 \times 10^{-5} \rangle$, Reaction needs to be specialized into Equilibrium-Reaction because the relation "dissociation-constant" only applies to equilibrium reactions. However, a potential conflict may arise as other triples involving the same Reaction concepts may also be specialized.

In general there are two types of conflicts resulting from the specialization procedure. A linear dependency, as in figure 4.9(a), arises when concept B is specialized into a specific class because of relation r_1, and the new class may conflict with another relation r_2. Tree dependency, as in figure 4.9(b), happens when concept A is specialized into a specific class because of relation r_1, and the new class may conflict with another relation r_2. These two types

```
Given a list of possible specializations (C_n)for a class B

For all triples <B, r, A> that contain B,
  if <B, r, A> requires a specialization of B,
  then
    C'_n = possible specialization of B from <B, r, A>
   C_n  = intersection_nof C  and C'
```

Figure 4.10: The `look-ahead` procedure that is called when either C_1 or C_2 is specialized for the triple $\langle C_1, r, C_2 \rangle$ being processed.

differ in terms of the location of the concept that is being specialized. In the case of linear dependency, the specialized concept is the head of one triple and the tail of another; in the case of tree dependency, the specialized concept is the head of both triples. A linear dependency case can be transformed into a tree dependency case by taking the inverse of r_1, thus making B the head of the triple and A the tail.

Despite these differences, both types of conflicts can be resolved by look ahead. Figure 4.10 shows the algorithm for look ahead. When a concept is specialized into a specific class, the look ahead procedure checks whether any other triples in which the concept is involved may need specialization. If there is another triple that needs specialization as well, the procedure makes sure that only candidate classes that are compatible with both triples are kept.

4.4 Input traversal

After an input has been decomposed, the triples will be processed one by one to detect and resolve the loose speak in them. We have discussed the

procedure (`test-and-repair`) that processes a single triple in detail, but we still need to address two issues: first, does every occurrence of loose speak get resolved by traversing the triples; second, what is the best order of triple traversal for resolving all occurrences of loose speak in the triples?

4.4.1 Does every loose speak get resolved through traversal?

If an input is decomposed into triples, and the triples are traversed for loose speak, we need to ensure that every occurrence of loose speak will be detected and repaired by this process. Before we discuss the completeness of the traversal algorithm, let us first take a look at a simpler version of that proposition.

Theorem 4.4.1. *Given an expression in a formal language with one occurrence of loose speak, the loose speak occurrence will be detected and resolved by traversing every triple in the expression once, and applying* ***`test-and-repair`*** *to each triple.*

The theorem states that every traversal of the triples will reduce the number of occurrences of loose speak in the input. In order to prove it, we first need to prove the following lemma.

Lemma 4.4.2. *An occurrence of loose speak spans exactly one triple; it is not distributed across multiple triples.*

Proof. We can prove this lemma by contradiction. Assume there exists an occurrence of loose speak that spans multiple triples. This means no triple

in the encoding contains loose speak, but when taken together loose speak occurs. Because each triple contains no loose speak, then by the definition of loose speak, the naïve encoding of each triple equals the correct encoding of that triple. Thus when combined together, the naïve encoding of all the triples must equal the combined correct encoding, which means the combined encoding contains no loose speak. This contradicts the assumption of the existence of loose speak, and therefore all occurrences of loose speak span over exactly one triple. □

Proof of theorem 4.4.1. Based on lemma 4.4.2, we know an occurrence of loose speak spans only one triple. Given one occurrence of loose speak in an expression, it can be detected and resolved by traversing and applying `test-and-repair` on each triple.

□

Before we prove the completeness of the triple traversal algorithm, we need to prove another lemma:

Lemma 4.4.3. *Each iteration of traversing does not create new loose speak.*

Proof. Each iteration of traversing affects a triple in one of the following ways:

- No change. This obviously does not create new loose speak.
- The triple is extended. This happens if `search_head` or `search_tail` finds an existing path to augment the loose speak in a triple. The new

path will not contain any new loose speak because it is based on prior knowledge, which is assumed to be consistent.

- Either the head class or the tail class of the triple is specialized into a subclass. This will not create new discrepancies because of the following: if the original class does not cause inconsistencies, neither will the subclass, since any new subclass is subsumed by the original class. If the original class causes inconsistencies, the new subclasses will not cause any new inconsistencies in the triple being traversed, since it is produced by the `specialize_head` or `specialize_tail`. Such specialization may cause conflicts with other triples in the expression, but such inconsistencies are resolved through the `look-ahead` routine described in section 4.3.1.

Since none of these effects on a triple will create new loose speak, each iteration of traversing does not create new ones either.

□

Now, we can prove that all occurrences of loose speak will be resolved by traversing the triples.

Theorem 4.4.4. *Given an expression that contains n occurrences of loose speak, all of the occurrences will be resolved by repeatedly traversing through the triples in the input and applying* ***`test-and-repair`*** *on each triple.*

Proof of theorem 4.4.4. From lemma 4.4.3 and theorem 4.4.1, we can conclude that each iteration of the traversal algorithm reduces the number of occurrences of loose speak, therefore all of the loose speak eventually will be resolved by repeatedly traversing the input expression.

□

4.4.2 In what order should the traversal proceed?

Another question we need to address is the order of the traversal. Is there a particular order in which the triples should be processed? If the processing of one triple depends on another, then we need to ensure that they are treated in the correct order. Luckily, theorem 4.4.5 states that triple processings are independent of each other, therefore there is no need for a particular order.

Theorem 4.4.5. *Applying* `test-and-repair` *to one triple does not affect any other triple.*

Proof. As stated in section 4.3, the `test-and-repair` procedure does two types of modification to the input triple $\langle C_1, r, C_2 \rangle$: path extension and concept specialization. The path extension type of modification replaces r in the triple with a series of relations. This does not affect any other triples because the changes take place within the input triple.

The concept specialization type of modification specializes either C_1 or C_2 into more specific classes. This type of modification may potentially affect

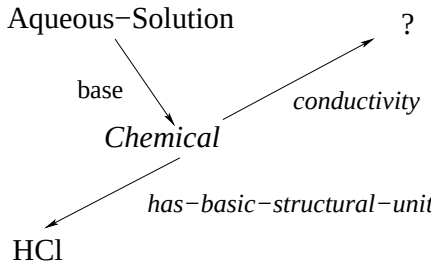


Figure 4.11: The interpretation of the query in figure 3.2 in the context of the chemistry knowledge base built for Project Halo. Notice that *HCl* is replaced by *Chemical*, and the *conductivity* query has been moved from *Aqueous-Solution* to the *base* of *Aqueous-Solution* to conform to the structure of the knowledge base.

other triples that also contain C_1 or C_2. However, the `look-ahead` procedure described in section 4.3.1 resolves this kind of conflict, and it ensures that other triples will not be affected. Hence, applying `test-and-repair` to one triple does not affect any other triple. □

4.4.3 Examples

For the example in figure 3.2, the loose-speak interpreter traverses the query graph to `test-and-repair` each triple. The triple ⟨*Aqueous-Solution*, *base*, *HCl*⟩ is tested first. The domain of *base* is constrained to be a *Solution*, and the range is constrained to be a *Thing*. Both constraints are satisfied, because *Aqueous-Solution* is a subclass of *Solution* and *HCl* is a subclass of *Thing*. However, there is no path in the knowledge base that resembles the triple, so the `search-head` function is called to search from the head, *Aqueous-Solution*. The search returns *Chemical*, whose *has-basic-structural-unit* is

Chemical-Entity (the superclass of HCl). See figure 4.8. Consequently, the triple is interpreted as $\langle$*Aqueous-Solution*, *base*,*Chemical*$\rangle$.

The triple $\langle$*queous-Solution*, *conductivity*, ?$\rangle$ is then tested for loose speak. No constraints are violated, but the query does not yield any result. This prompts a breadth-first search from *Aqueous-Solution*. This search stops at *Chemical*, the *base* of *Aqueous-Solution*, because it has a conductivity value and is a suitable replacement for *Aqueous-Solution* in the original triple.

Figure 4.11 shows the final interpretation of the query graph. Notice that the solute of the *Aqueous-Solution* is replaced by *Chemical*, and the *conductivity* query has been moved from *Aqueous-Solution* to *Chemical*. After these changes, the two types of loose speak – metonymy and aggregate – are resolved in the query graph, and the encoding is correctly aligned with the structure of the knowledge base.

Chapter 5

Evaluation

In this chapter, we present the results of various evaluations of the loose-speak interpreter. There are a number of questions we want to answer through the evaluation. First, we need to find out whether the loose-speak interpreter is useful in knowledge interaction. We've evaluated this by applying the loose-speak interpreter to numerous naïve encodings and measuring its performance. Second, we need to find out the effectiveness of different parameters of the interpreter algorithm, namely the ranking algorithm and the stopping criteria. This will allow us to fine tune the loose-speak interpreter. Third, we need to find out what kind of knowledge is required for the loose-speak interpreter to perform well. If loose-speak interpretation requires additional knowledge, such as detailed knowledge about the input, then any gains in knowledge base interaction by the loose-speak interpreter could be overshadowed by the efforts needed for formulating the additional knowledge. In that case, loose-speak interpretation would be of little value to knowledge base interaction.

5.1 How well does the loose-speak interpreter perform?

To evaluate the effectiveness of the loose-speak interpreter in knowledge base interaction, we conducted three experiments: a user study, a noun compound interpretation study, and a study of the interpretation of loose speak in natural language inputs.

5.1.1 User study

First, we asked several users to encode a set of questions, and compared their naïve encodings with the corresponding output from the loose-speak interpreter.

5.1.1.1 Setup

For this experiment we used a set of fifty multiple choice questions from various mock Advanced Placement chemistry tests. This set was compiled by a chemist (anonymous to us), who also provided the answers for the questions. This data set was distinct from the two sets of questions used in the frequency study in section 3.1, and it was distinct from the training data that we used to refine the interpreter.

We recruited three users, with varying backgrounds in knowledge engineering and chemistry, to pose questions to the chemistry knowledge base built for Project Halo [117]. All three were familiar with KM [95], the knowledge representation language used in the experiment, but none was familiar with the chemistry knowledge base and its idiosyncrasies. Their knowledge engi-

neering experience varied from a few months to many years. Their chemistry backgrounds varied from weak (high school chemistry decades ago) to medium (college chemistry years ago).

The users were given a short (three page) tutorial on encoding questions. It contained two sections: format and vocabulary. The format section explained that question encodings should have a context part, which gave the contextual information of a question, and an output part, which specified the value being sought. The format section illustrated this with three examples. The vocabulary section listed commonly used classes and slots in the chemistry knowledge base and their corresponding chemistry terms. Users were also referred to a web page containing the complete listing of slots and classes; they were allowed to take as much time as necessary to complete the questions. These brief instructions were not a complete tutorial on using the knowledge base, as evidenced by the high rate of loose speak in the users' encodings of questions.

We used metrics of precision and recall to evaluate the interpreter's performances. Precision and recall were defined as follows [61]:

$$Precision = \frac{\#\ of\ correct\ answers\ given\ by\ system}{\#\ of\ answers\ given\ by\ system}$$

$$Recall = \frac{\#\ of\ correct\ answers\ given\ by\ system}{total\ \#\ of\ possible\ correct\ answers}$$

For our experiment, the number of correct answers was the number of question encodings that were interpreted correctly. The number of answers given

by the system was the number of question encodings for which the interpreter detected loose speak and found an interpretation. This includes three types of encodings: first, naïve encodings that are correctly detected and interpreted, second, naïve encodings that are correctly detected but erroneously interpreted, and third, correct encodings that are erroneously detected as containing loose speak. Finally, the number of all possible correct answers was the number of all the question encodings that contained loose speak. An interpretation was determined to be correct if it completely aligned with the chemistry knowledge base and conveyed the intended semantics of the question encoding as judged by us. Precision estimated the likelihood of a correct interpretation when an interpretation was found; recall was a measurement of coverage.

Our experiment did not include noun compound type of loose speak because chemistry is an unrepresentative domain for evaluating noun compound interpretation. Although noun compounds, such as "carbon monoxide" (a molecule containing both an oxide and a carbon), occur frequently in chemistry questions, their interpretations all are of the same format, and they do not require inferring a variety of semantic relations. The effectiveness of the interpreter on noun compounds was evaluated separately in section 5.1.2.

For this experiment, we ran the interpreter in "batch mode". If the interpreter found multiple interpretations for any question, we treated the first interpretation as its sole response.

	KE experience	Chemistry knowledge	Experiment length	Questions encoded	Correct encodings	Precision	Recall
User A	0.5 yr	medium	10.5 hrs	47 (94%)	3 (6.4%)	97.4%	90.5%
User B	17 yrs	weak	9.95 hrs	48 (96%)	4 (8.3%)	100%	90.7%
User C	0.25 yr	medium	10 hrs	44 (88%)	5 (11.4%)	96.8%	87.8%

Table 5.1: The evaluation results of the loose-speak interpreter. The first two columns show users' backgrounds in knowledge engineering and chemistry. The next three columns show users' performance: the time required to finish the experiment; the number of questions encoded; and the number of correct encodings. The last two columns measure the interpreter's performance in terms of precision and recall. As the data have shown, despite different backgrounds, all users were able to encode most of the questions, the encodings had many occurrences of loose speak, and the interpreter was able to correct most encodings, and to do so with high precision.

5.1.1.2 Data analysis and discussion

The experimental results are shown in table 5.1. The first two columns give the users' background in knowledge engineering and in chemistry; the next three columns show how well users performed by recording the time it took them to encode the test questions, the number of questions they were able to encode, and the percentages of correct encodings. The last two columns measure how well the interpreter performed.

Based on the data, we can draw three conclusions:

1. Loose speak is very common – just as we found in our preliminary study (described in chapter 3). Only 9% of the encodings on average contain no loose speak. This underscores the importance of the loose-speak interpreter for building useful knowledge-based question-answering systems.

2. The loose-speak interpreter works well. As shown in Table 5.1, the precision of the interpreter is above 95%, and the recall is around 90%. This shows that the search algorithm we used is suitable for interpreting naïve encodings. In addition we found that 54% of the loose speak occurrences are detected by constraint violations, and the rest by the resemblance check. This suggests both checks are important for detecting loose speak.

3. The loose-speak interpreter significantly assists novice users in posing questions. The results show that there is no clear correlation between the users' knowledge engineering experience and the number of questions they are able to encode. Users with little knowledge engineering experience were able to encode large percentages of questions quickly. This is good news; it suggests that with the burden of loose-speak interpretation handled automatically, users can concentrate on the content of questions instead of knowledge engineering details.

5.1.2 Noun compound interpretation

Because the Project Halo data set is not representative of the noun compound type of loose speak, we evaluated the performance of the interpreter for noun compounds separately. As stated in section 3.7, noun compound is an important type of loose speak. Our algorithm treats a noun compound as a binary predicate with an unspecified relation, and it finds an interpretation just as with other types of loose speak.

5.1.2.1 Setup

To avoid getting results that are skewed to a particular domain or knowledge representation, we used a variety of quite different data sets. The first consists of 224 noun compounds from a college-level cell biology text [2]. The second consists of 294 noun compounds from a small-engine repair manual [4]. The third data set consists of 224 compounds from a Sun Sparcstation manual [23]. The nouns used in these data sets are mapped to the corresponding concepts in knowledge bases on these topics manually.

The interpretation of a noun compound is considered correct if the first path returned by the algorithm provides a *sensible* interpretation, as judged by a human oracle. If multiple paths of the same lengths are returned, then one path is randomly selected as the first path. Here is an example of one such *sensible* interpretation. Given "computer expert", the interpretation "a computer acting as an expert" is correct, even though another interpretation ("an expert on computers") might be the first to come to mind. This criterion is used for two reasons. First, when data sets are extracted from text as pairs of nouns, the associated context is lost, and it is hard to know the "right" interpretation without any context. Second, this criterion is used in previous approaches such as [8, 115]. Using the same criterion makes it easier to compare the results with other reported studies.

The metrics we used to evaluate performance are precision and recall as defined in section 5.1.1.1.

5.1.2.2 Knowledge bases

The knowledge bases used for our experiments are made of a set of concepts and a set of axioms associated with each concept. The concepts are related to one another through subsumption relations to form a hierarchy. The axioms on one concept are assertions of semantic relations between that concept and other concepts. For example *Action* is a concept in our knowledge base that describes things that *happen*. The axioms on Action include things such as "every Action has an object, which is an Entity", where *Entity* is another concept in the knowledge base to describe things that *are*. Axioms are either local or inherited from the axioms of a superclass.

Despite these commonalities, the knowledge bases differ significantly. They differ in terms of how they were built. The knowledge base for the biology text was built using the generic Component Library [7] to answer end-of-the-chapter style questions, as one of the challenge problems for DARPA's Rapid Knowledge Formation project [102]. The knowledge bases for the other two data sets (the small-engine repair manual and the Sparcstation manual) were built "on top of" the knowledge in WordNet [44]. We augmented WordNet with the upper ontology of the generic Component Library plus about ten concepts that are important to each of the two domains (whose partonomies are not complete in WordNet). Through this process, we encoded 416 concepts in about 50 man-hours. The advantage of using WordNet as the foundation for these knowledge bases is two-fold: it includes most of the terms used in the data sets, linked with both taxonomic and partonomic relations, and it

	Data size (N)	Precision	Recall
Biology	224	93.8%	93.8%
Engine	294	85.2%	74.5%
Sparc	224	84.5%	73.2%

Table 5.2: The evaluation results of the loose-speak interpreter on noun compounds. Each data set contains more than 200 pairs of nouns, and the interpreter's precision and recall are around 80% across all three data sets. The performance is similar to that achieved by previous systems.

is widely available and well used. The knowledge bases also differ in terms of content. Other than the shared upper ontology of the generic Component Library, they have few concepts in common.

5.1.2.3 Interpreter performance

Table 5.2 shows the performance of the interpreter for the three data sets in terms of precision and recall. Both precision and recall are around 80% across all three knowledge bases. This is similar to the performance achieved by previous noun compound interpretation systems described in section 3.7.1.

We have also measured the distribution of the path lengths of the top interpretations from the data set. The distribution (figure 5.1) shows the length of a typical interpretation found by the loose-speak interpreter. If the interpretation of a noun compound is made of subsumption relation, for example "an oak tree" is interpreted as "a tree" or "an oak", then path length is zero. If the interpretation of a noun compound is made of more than one relations, for example "animal virus" is interpreted as "a virus that is the

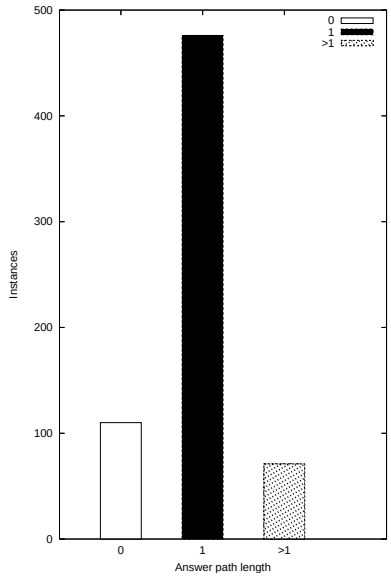


Figure 5.1: The distribution of different path lengths for the top interpretations. If the interpretation of a noun compound is made of subsumption relation, for example "an oak tree" is interpreted as "a tree" or "an oak", then path length is 0. If the interpretation of a noun compound is made of more than one relations, for example "animal virus" is interpreted as "a virus that is the agent of a invade, whose object is a cell that is the basic structural unit of an animal", then path length is greater than 1. The distribution shows two important points. First, it shows that most of the interpretations found are made of a single relation; overwhelming majority of the spreading activation search conducted is very shallow. This suggests that even if the knowledge based being used is very large, the spreading activation can be conducted quickly. Second, it shows that there is a significant number of interpretations that are made of multiple relations. A noun compound classifier will not be able to interpret these noun compounds correctly, and knowledge base searches are needed.

agent of a invade, whose object is a cell that is the basic structural unit of an animal", then path length is greater than one. The distribution shows two important points. First, it shows that most of the interpretations found are made of a single relation; overwhelming majority of the spreading activation search conducted is very shallow. This suggests that even if the knowledge based being used is very large, the spreading activation can be conducted quickly. Second, it shows that there is a significant number of interpretations that are made of multiple relations. A noun compound classifier will not be able to interpret these noun compounds correctly, and knowledge base searches are needed.

5.1.3 Interpretation of natural language input

The user study in section 5.1.1 was based on the encodings of human users. We also evaluated the loose-speak interpreter on encodings produced by a natural language processing program. This was conducted to compare the interpreter's performance on human encodings with performance on encodings generated automatically. The comparison is needed because the loose-speak occurrences in human encodings may be significantly different from those in machine encodings which may result in degradation of the interpreter's performance.

5.1.3.1 Setup

The natural language processing program we used was an interpreter for the Computer Processable-Language (CPL) [27], which interprets sentences written in a simplified version of English. A basic CPL sentence is of the form "subject + verb + complements + adjuncts", where complements are elements needed to complete a sentence and adjuncts are modifiers. The interpreter for the CPL was used in Project Halo phase II to process questions in physics, chemistry, and biology. The questions were written in CPL by domain experts who did not know the structure of the knowledge base that would be used to answer the questions.

The loose-speak interpreter was used in conjunction with the CPL interpreter in three ways. First, the loose-speak interpreter was called whenever a noun compound was encountered in a sentence. Because CPL did not allow noun sequences longer than two, the CPL interpreter did not need to bracket noun compounds [6]. Second, the loose-speak interpreter was called whenever a definite reference was used without a direct antecedent. Because CPL did not allow pronouns, an anaphora was either direct or indirect. If no direct antecedent was available, then the loose-speak interpreter was called to resolve the indirect anaphora. More details on how the interpreter can be used to resolve indirect anaphora can be found in chapter 6. Third, the loose-speak interpreter was called after the CPL interpreter had created an overall formal representation of a sentence to detect and resolve any remaining occurrences of loose speak.

5.1.3.2 Knowledge bases and users

Six knowledge bases were used in this experiment covering three domains: biology, chemistry and physics. Unlike the chemistry knowledge base used in the user study in section 5.1.1, these knowledge bases were not encoded by knowledge engineers; instead they were encoded by six subject matter experts (two per domain). Because subject matter experts have little training in knowledge engineering, the knowledge bases they build may have more gaps and errors.

The test questions used are based on Advanced Placement test questions, and they were posed by five different users. Four of these users were undergraduate students, and one was a high school student. None of them had any experience in knowledge-based systems.

The correctness of the responses by the interpreter was decided by a separate panel of five judges, each responsible for the questions posed by one user. The performance of the interpreter was measured by precision and recall as defined in section 5.1.1. For this experiment, we ran the interpreter in "batch mode" as before. If any question had multiple interpretations, only the first one was chosen.

5.1.3.3 Data analysis and discussion

Table 5.3 shows the results of the experiment. The first two columns show the domain and the users; the next column shows the number of questions

		Num of questions	LS noun compounds		LS indirect anaphora		LS overall	
			Precision	Recall	Precision	Recall	Precision	Recall
Biology	user 1	47	100%	28%	100%	100%	100%	100%
Chemistry	user1	21	71%	14%	100%	100%	83%	33%
	user2	8	100%	12%	0%	n/a	92%	71%
	user3	21	100%	27%	100%	100%	100%	100%
Physics	user 1	41	92%	87%	50%	100%	82%	78%
Overall average		138	92%	38%	69%	100%	86%	68%

Table 5.3: The evaluation results of the loose-speak interpreter on encodings produced by a natural language processing program. The first two columns show the domain and the users; the next column shows the number of questions encoded. The remaining columns show the performance of the loose-speak interpreter using encodings created by the CPL interpreter.

encoded. The remaining columns show the performance of the loose-speak interpreter using encodings created by the CPL interpreter.

In general, the precision of the interpreter at all three tasks is high. On average, the precision for noun compound interpretation task is above 90%, near 70% for indirect anaphora resolution and above 85% for the overall representation's interpretation. The recall of the interpreter is lower. On average, the recall for noun compound interpretation task is 38%, 100% for indirect anaphora resolution and 68% for the overall representation's interpretation[1]. An analysis of the failed cases reveals that there are three main causes for the low recall:

1. Improper word sense disambiguation. One of the first things the CPL

[1]Chemistry user2's data did not include any true positive case or false negative case of indirect anaphora resolution, therefore the recall in that column is not available.

interpreter does is to map a word string into a concept in the knowledge base. In the case of a novel word, the CPL interpreter finds the closest concept or creates a new concept based on the input string. Unfortunately, such mappings are not always perfect. Incorrect word sense disambiguation results in wrong input for the loose-speak interpreter, which fails to find an interpretation.

2. Parsing errors. There are domain specific, idiom-like noun compounds, such as "constant velocity", that should be recognized by the parser and do not require any interpretation. However, the parser fails to recognize them sometimes, and it passes these idiom-like noun compounds to the loose-speak interpreter, which fails interpret them.

3. User errors. During the process of rephrasing a question into simplified English, an user may formulate erroneous inputs to the system. For example, one user rephrased "a screw falls.." into "There is a move. *The move has a fall.*", which is nonsensical.

To quantitatively measure the impact of these three causes, we counted the number of loose speak occurrences that the interpreter failed to interpret due to the causes. We found that out of a total of 112 instances of failed instances, 68 of them are due to the first cause, 13 due to the second cause and 2 due to the third cause. If we exclude the failed cases attributed to these three causes, then the average recall for noun compound interpretation task is 67%, 100% for indirect anaphora resolution and 100% for the overall

representation's interpretation. This is similar to the performance of the loose-speak interpreter in the user study and the noun compound interpretation evaluation described in section 5.1.1 and 5.1.2.

Compared with the results from section 5.1.1, the loose-speak interpreter has demonstrated similar precision with lower recall. The difference is mainly due to the difficulties in natural language processing which result in incorrect input to the loose-speak interpreter. This suggests that the loose-speak interpreter itself can handle human encodings as well as automatically generated correct encodings.

5.2 Why does the interpreter work?

Although the loose-speak interpreter was effective at interpreting the data sets we used in the experiments, we need to find out what contributes to its success.

5.2.1 What type of knowledge is needed?

One common and justifiable critique of knowledge-based systems is that they require detailed domain specific knowledge, which are often difficult to obtain. If loose speak can be successfully interpreted without much knowledge, then the problem is avoided: the small amount of knowledge that is required can be easily encoded. We evaluated the knowledge requirements of the loose-speak interpreter. We believed that the domain independent top levels of an ontology are sufficient for the interpreter, and we measured the contribution

of successive ontology levels to the performance of the loose-speak interpreter by ablating them one at a time. If a level is ablated, and the interpreter's performance is significantly impacted, then the contribution from that level is large, otherwise, the contribution is small.

5.2.1.1 Setup

The challenge in measuring an algorithm's sensitivity to knowledge base content is that the results may vary across domains and across knowledge bases. We attempt to neutralize these factors by replicating our study in three domains with quite different knowledge bases. We used the three noun compound data sets (biology, small-engine repair manual and Sparc workstation owner's manual) and the three associated knowledge bases described in section 5.1.2.

5.2.1.2 Ablation steps

The sensitivity of the algorithm to each level of the ontology is measured through a series of ablations. When a level is ablated, the concepts on that level and all their axioms are deleted from the knowledge base. If these concepts have subclasses, then the subclasses are set to be subsumed by the superclasses of deleted concepts. As a special case, when the $0th$ level (the root level concept) is deleted, it is replaced by a generic concept of "Thing". Because the root level concept is vacuous, deleting it has no affect. As an example, figure 5.2 illustrates the ablation of level 1.

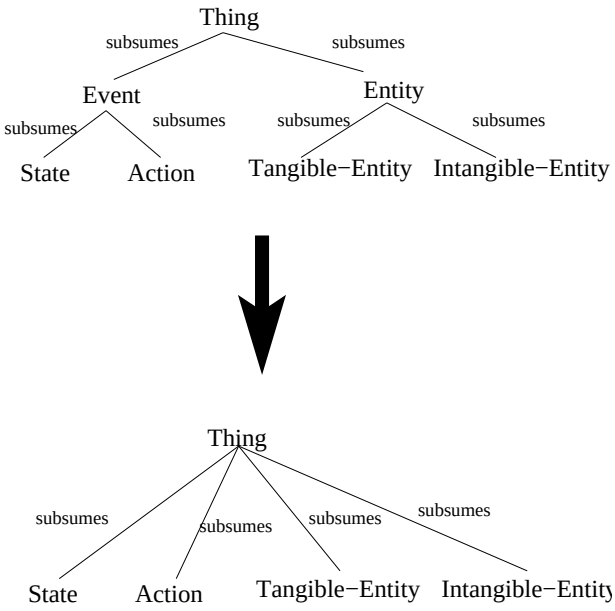


Figure 5.2: The ablation of level 1 on a sample taxonomy. Notice that the concepts "Entity" and "Event" are removed from the taxonomy. Classes on level 2 are promoted to level 1 and they are the direct subclasses of "Thing".

5.2.1.3 Data analysis and discussion

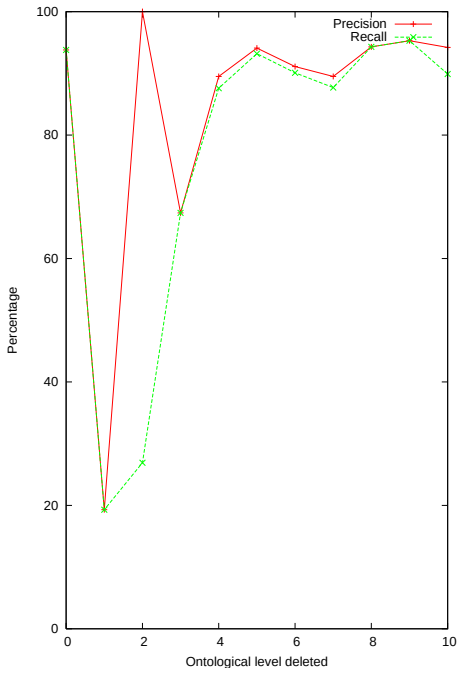
In this section, we analyze the experimental data to determine the contribution of each level of the knowledge bases' ontology to the task of interpreting noun compounds, and to find plausible explanations for the results.

Figure 5.3 shows the effect of ablating different levels of the ontology for the three knowledge bases in terms of precision and recall. Without any ablation, both precision and recall are around 80% across all three knowledge bases.

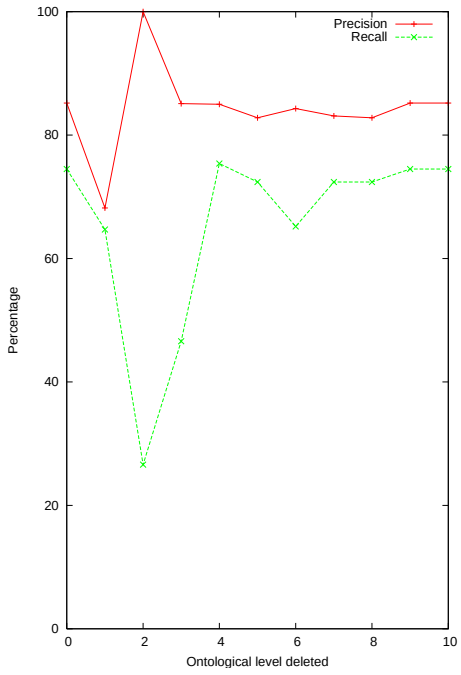
Which levels of the taxonomy are most important? Ablating level one causes a big drop in both precision and recall. Ablating level two introduces a big gap between precision and recall. The gap indicates that the algorithm does not find interpretations for many noun compounds. As lower levels in the ontology are ablated one at a time, the impact diminishes and performance improves to the level of a knowledge base with no ablations.

The contribution of the first two levels of the ontology is observed across all three data sets and knowledge bases. This pattern strongly suggests that the top levels of the ontology are the most important for the noun compound interpretation task.

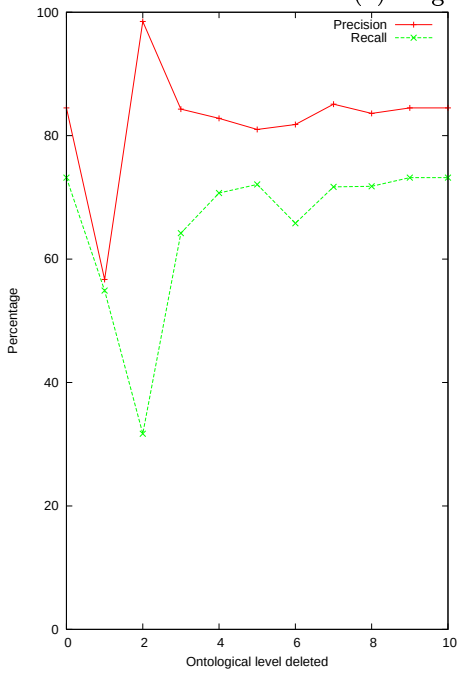
Why are the top levels of the ontology the most important? Perhaps the top levels of the ontology are more important than lower levels because they contain many axioms important to the noun compound interpretation



(a) Biology data set.

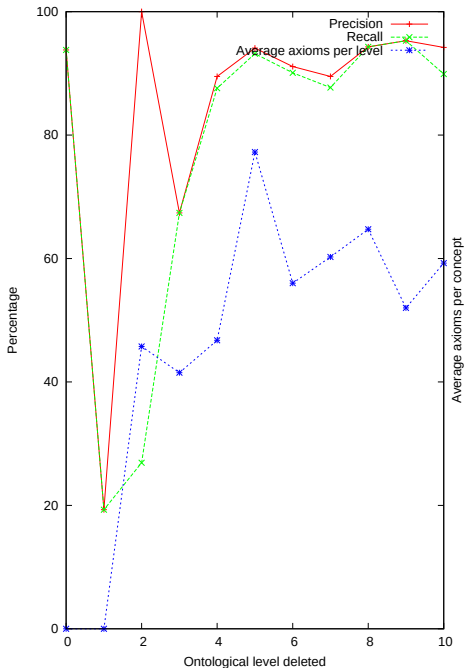


(b) Engine data set.

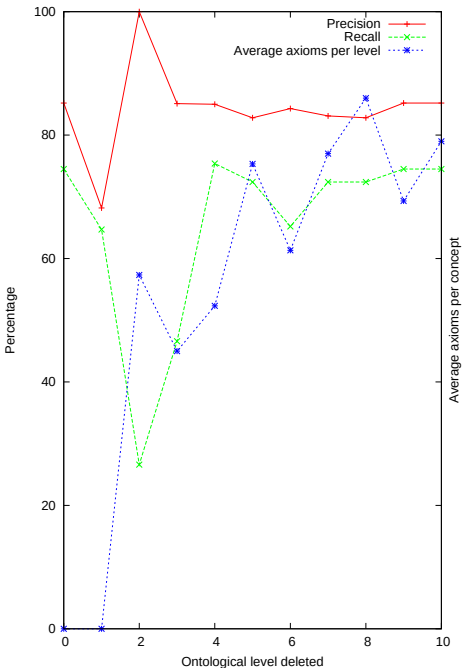


(c) Sparc data set.

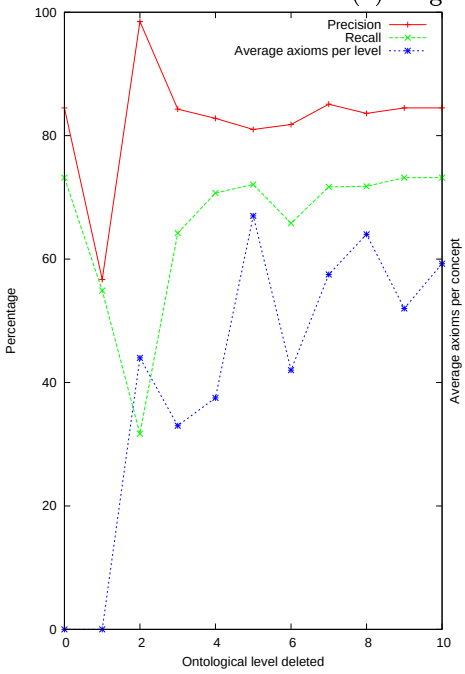
Figure 5.3: The impact on precision and recall of ablating different levels of the ontology for the three data sets.



(a) Biology data set.



(b) Engine data set.



(c) Sparc data set.

Figure 5.4: Comparison of the impact of ablating different levels of the ontology with the average number of axioms on each level.

task. Ablating these levels has the effect of deleting the associated axioms – not only from the concepts at these levels, but also from all concepts below them in the taxonomy (because the axioms are passed by inheritance from superclasses to subclasses).

To test this hypothesis, we counted the number of axioms associated with concepts at each level of the ontology. There are various ways to count axioms – we chose a simple and conservative one. For this study, we considered an axiom about concept C_1 to be an assertion of a semantic relation between instances of C_1 and instances of another concept C_2. For example, one axiom associated with the concept *Car* is: "every car has 4 wheels". We counted only local (non-inherited) axioms, and we counted in minimum fashion, so this axiom is counted once, not four times.

Figures 5.4(a), 5.4(b), and 5.4(c) compare the impact of ablating different ontological levels (in terms of precision and recall), with the average number of axioms per concept on each level. The data show that upper levels of the ontology contribute the most to task performance, yet they contain relatively few axioms; lower levels of the ontology contribute much less, yet they contain relatively more axioms.

We conclude, therefore, that the number of axioms in the top levels of the ontology is not what makes these levels important. Rather, we suggest that they are important for noun compound interpretation for two reasons:

1. Top levels of the ontology include concepts that make important on-

tological distinctions. For example, ablating the level that introduces *Entity* and *Event* blurs the distinction between widely different classes of concepts, which causes the search to stop with erroneous results. Consequently, many more interpretations are returned, and because we only use the first interpretation, the probability of it being correct is reduced.

2. Although they contain relatively few axioms, axioms in top-level concepts are important for the task. The top-level ontology contains the most frequently used axioms, such as the fact that "every Action involves an object that is acted upon". These axioms are used in the search as a step along the way. Deleting these axioms makes it difficult to find an interpretation for many of the noun compounds, thereby causing recall to lag behind precision.

5.2.2 How important is ranking?

One important aspect of the interpreter's algorithm is ranking the interpretations when multiple ones are found. Our algorithm ranks them based on their path length. We evaluated this seemingly simple ranking algorithm using the noun compound data sets. We used two metrics: the average number of answers found and the percentage of correctly ranked instances.

The average number of answers ($\bar{L}$) is defined as the following:

$$\bar{L} = \frac{\sum |A_i|}{n}$$

where A_i is the set of answers found for a noun compound, and n is the number

of noun compounds for which the system finds at least one interpretation. This metric shows the importance of ranking. If, on average, many answers are found, then it is important to rank them appropriately because a randomly chosen answer is less likely to be correct. If, on average, few answers are found, then ranking is less relevant. For the three sets of data, we found that $\bar{L} = 1.9$ and the system finds exactly one answer for 51% of the noun compounds. This suggests that ranking is not a very important problem for the loose-speak interpreter, because, on average, a random ranking algorithm should have at least a 50% success rate.

We also measured the percentage of correctly ranked instances (P), which is defined as the following:

$$P = \frac{|C|}{|D|}$$

where C is the set of correctly interpreted noun compounds and D is the set of noun compounds whose interpretations by the system include the correct one. This measures the effectiveness of the ranking algorithm. If P is high, then it shows that the ranking algorithm is able to accurately identify the correct interpretation. For the noun compound data sets, we found that $P =$ 95%. If a noun compound is considered correctly interpreted when the top three interpretations include the correct interpretation, then P increases to 99.8%. Note that P exceeds the precision of the overall system performance. This is because the precision is hindered by the cases in which none of the interpretations is correct.

The two metrics show that ranking is not a significant problem, and our simple algorithm that ranks interpretations based on their path length is sufficient.

5.2.3 When should the search stop?

Deciding when the search should end is another important part of the algorithm. Our interpreter terminates the search when either a superclass or a subclass of the goal concept is found. There are other variations of the stopping criterion, such as `equality`, which stops the search when the exact goal concept class is found.

5.2.3.1 Setup

We evaluated the impact of four different stopping criteria on the loose-speak interpreter in this study. In addition to `super_or_subclass` and `equality`, we used `superclass` and `subclass`. These are less restrictive than `equality` but more restrictive than `super_or_subclass`. The `superclass` criterion stops the search when the system visits a class whose superclass has already been visited. The `subclass` criterion stops the process when the system visits a class whose subclass has already been visited.

When a more restrictive stopping criterion is used, false positives are expected to decrease, because fewer answers will be found by the system. Consequently precision should increase. When a less restrictive stopping criterion is used, false negatives are expected to decrease, because fewer inputs will have

no interpretation, and recall should increase.

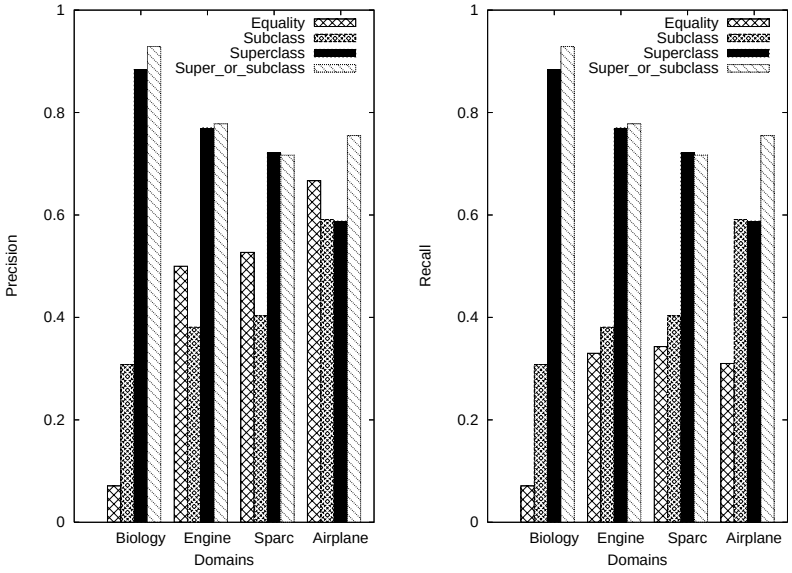
5.2.3.2 Test data

We evaluated the impact of different stopping criteria on the task of noun compound interpretation. Four sets of data drawn from different domains are used to avoid getting results that are skewed to a particular domain, knowledge base, or data set. In addition to the three data sets used in section 5.1.2, we have a fourth data set which consists of 55 noun compounds in the domain of airplane parts. These data are encodings of airplane-related sentences found in various online sources. The knowledge base used to interpret the airplane data is a custom-built knowledge base about airplanes. The taxonomy is 15 levels deep, and it has more than 8,000 concepts. Each concept is directly connected to 70 other concepts on average.

5.2.3.3 Results

Figure 5.5 shows the impact of different stopping criteria. Similar to the previous experiment, recall is improved significantly when `super_or_subclass` is used. The interpreter that used the most restricted stopping criterion, `equality`, always had the worst recall in all four data sets. The interpreter that used the least restricted stopping criterion, `super_or_subclass`, always had the best recall. In addition to the increased recall, the `super_or_subclass` based interpreter also had the best precision.

The results appear counterintuitive at first, because the least restrictive



(a) Precision. (b) Recall.

Figure 5.5: Performance of the noun compound interpreter using four different stopping criteria on four different data sets. The four bars for each domain, from left to right, represent the interpreter's performance in that domain using `equality`, `subclass`, `superclass` and `super_or_subclass` stopping criteria, respectively. The interpreter that used the most common and restricted stopping criterion, `equality`, had the worst recall and second best precision. The interpreter that used the least restricted stopping criterion, `super_or_subclass`, had the best precision and recall.

stopping criterion is based on abductive reasoning, which produces unsound reasoning. It may cause the search to stop prematurely. Such stoppage may induce many false positives and reduce the precision significantly. In practice we found this was rarely the case. In fact, most of the interpretations found by the `super_or_subclass` stopping criterion are true positives, not false positives, so precision was not compromised. In addition, because `super_or_subclass` is a more relaxed stopping criterion, it finds more interpretations, suffers fewer false negatives and has higher recall. This explains the performance difference between various stopping criteria.

5.3 Summary

To summarize, we have evaluated the loose-speak interpreter on various naïve encodings, and we have found it to be highly effective in producing the correct version of any given input. Our evaluation shows that the simple path-length based ranking algorithm for multiple interpretations is sufficient for the test data sets, and that the abductive `super_or_subclass` stopping criterion extends the coverage of the loose-speak interpreter with little cost to the precision.

We also found that the knowledge needed for the interpreter to function comes mostly from the top levels of the ontology, and this allows users to utilize the loose-speak interpreter even if the knowledge base contains little or no domain specific knowledge.

Chapter 6

Inter-input loose speak interpretation

The previous chapters focused on occurrences of loose speak within a single utterance and reported that various common types of loose speak can be resolved by a single mechanism. In this chapter, we study occurrences of loose speak beyond sentential boundaries, and evaluate the performance of the interpreter on correcting them.

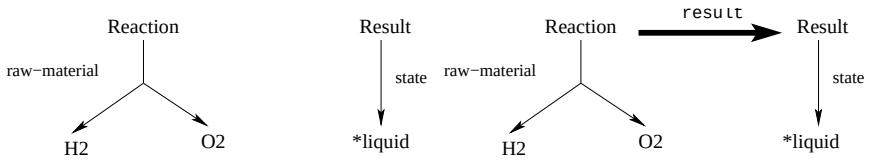
6.1 Inter-input loose speak

Inter-input loose speak occurs when the naïve encoding of inter-input relations misaligns with the correct encoding. For example, take the following two utterances:

(1) a. H2 and O2 react.

b. The result is a liquid.

Figure 6.1(a) shows a naïve encoding of (1-a) and (1-b), which does not have an explicit relation between the two utterances. The correct encoding is shown in figure 6.1(b). The inter-input relation illustrated by the bold edge is added.



(a) Naive encoding. (b) Correct encoding.

Figure 6.1: The naïve and correct encoding of example (1-a) and (1-b). The correct encoding adds an inter-input relation "result" that links the encodings of the two utterances.

Some types of inter-input relations can be identified and encoded without knowledge of the underlying structure of the knowledge base. For example, the direct anaphora between "a tree" and "the tree" in the following example can be identified and encoded by a naïve user.

(2) a. He plants a tree in the backyard.

b. The tree will bloom next year.

In contrast, indirect anaphora are more difficult to resolve because they require domain knowledge. Indirect anaphora arise "when a reference becomes part of the hearer's or reader's knowledge indirectly rather than by direct mention" [85]. Indirect anaphora have been called by a variety of names, including bridge definition, functional anaphora, elided noun phrases, implicit associates and textual ellipsis [32, 53, 79, 80, 116]. The fundamental characteristic of indirect anaphora is the assumption that a listener is able to find the association between the entity being used and the entity being referred to. The object that is being referred to is called the *anchor*, or *antecedent*; the expression that refers to the antecedent is called the *referring expression*; and the asso-

ciation between the referring expression and the anchor is called the *link*. For example, the following sentence contains an instance of indirect anaphora.

(3) When the detective got back to the ***house***, *the door* was unlocked.

The referring expression, "the door", relates to the antecedent, "**the house**", through a whole/part (metonymy) link. It is assumed that readers know enough about the referring expression and the antecedent to infer the link between them.

Indirect anaphora resolution is the task of identifying the element in the text to which the referring expression is related (antecedent) and identifying the relation (link) holding between the referring expression and its antecedent. Indirect anaphora processing is an important area in linguistic study because indirect anaphora are prevalent, and processing them is difficult. It has been empirically shown that indirect anaphora are quite frequent in discourses [94]. Processing indirect anaphora is difficult because indirect anaphora are used when the listener is assumed to have the common sense knowledge required to identify the link, and obtaining such common sense knowledge is difficult for a computer program. Another difficulty in processing indirect anaphora lies within the variety and complexity of possible links between the description and its anchor. A link may be combined of dozens of different relations, and correctly identifying it is not easy.

Link Type	Example
Set/element	a ***class*** ... the *student* ...
Whole/part (metonymy)	a ***room*** ... the *wall* ...
Hypernym/hyponym	an ***oak*** ... the *tree* ...
Event/role	a ***murder*** ... the *killer* ...
Cause/consequence	an ***earthquake*** ... the *debris* ...

Table 6.1: Some frequent types of indirect anaphora.

6.2 Loose-speak interpreter and indirect anaphora

We approach the indirect anaphora resolution problem from a perspective different from previous work. Whereas previous researchers treated indirect anaphora resolution as a unique problem requiring special-purpose methods, we view it as an instance of a more general problem: how to use background knowledge to infer relations among linguistic constituents. We used the loose-speak interpreter as the method to uncover such relations.

Our indirect anaphora resolution system takes in a referring expression and a list of nouns that appear earlier than the referring expression in the same text. The system chooses, as the most likely antecedent, the candidate that is semantically most closely related to the referring expression. A pair of concepts is considered to be semantically related if our loose-speak interpreter can find an interpretation for the pair. Figure 6.2 is an example of how the algorithm works. Given the nouns "house" and "door", the first search begins at Door, then traverses all the semantic relations from Door, such as *has-part* and *is-part-of*. The search stops at Building, which is a superclass of House. The solid bold lines indicate the path found and returned. It describes that

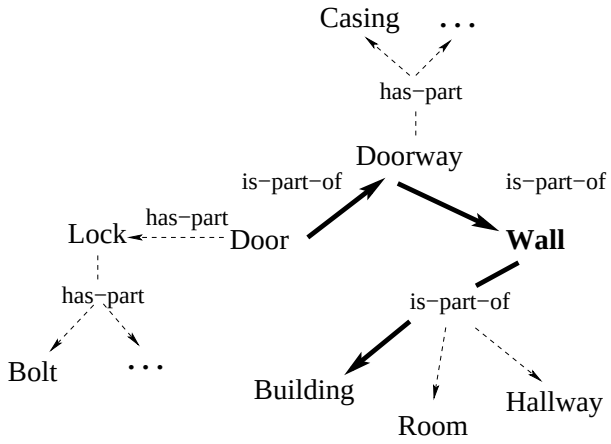


Figure 6.2: An example of how the algorithm works. Given the nouns "house" and "door", the solid bold lines show the path found by breadth-first search starting at Door and ending at Building, which is a superclass of House.

the door "is part of a doorway, which is part of a wall, which is part of a building" (a superclass of House). The second search begins at House, and looks for any superclass or subclass of Door. The sorting function ranks the paths returned from the two searches, and chooses the shortest path to return.

If multiple valid interpretations from different candidate antecedents are found, the same sorting function selects the interpretation whose path length is the shortest.[1]

6.3 Related work

Unlike other types of anaphora, which can often be resolved using syntactic features, the resolution of indirect anaphora requires semantic knowl-

[1]This is typically referred to as lexical distance.

edge of the relationship between the referring expression and the antecedent. Because such knowledge was previously unavailable to computer programs, most of the early studies in indirect anaphora were theoretical [25, 48]. These studies identified a variety of types of indirect anaphora (see table 6.1).

6.3.1 Knowledge-based indirect anaphora resolution

Recently there has been more progress in experimental studies of indirect anaphora. These studies can be divided into knowledge-based systems and corpus-based systems. The knowledge-based systems [53, 80, 116] use knowledge to resolve indirect anaphora. This approach not only returns the antecedent, it also discovers the type of association between each referring expression and its antecedent, which is a piece of information important for other parts of a full natural language processing system. Our system is most similar to knowledge-based systems because it infers relations among candidate antecedents and referring expressions through a series of knowledge base searches.

6.3.1.1 Vieira and Poesio

Poesio et al. [116] used WordNet as the knowledge base. Their system takes in a referring expression and a list of nouns that appear earlier than the referring expression in the same text. The system chooses one noun as the most likely antecedent for the referring expression. It selects the antecedent by first grouping the nouns based on sentence boundary, then using stack-based

theory [103] to sort the candidate associations and select the most promising one. Specifically, the system looks back one sentence at a time and returns a candidate as the antecedent as soon as the candidate satisfies one of the following conditions based on WordNet knowledge:

- The candidate is a synonym of the referring expression, such as *aviator* and *flyer*.
- The candidate is a hypernym (superclass) or hyponym (subclass) of the referring expression, such as *oak* and *tree*.
- The candidate is a coordinate sibling of the referring expression, such as *home* and *house*.
- The candidate has a meronymic (has-part) or holonymic (is-part-of) relation with the referring expression, such as *room* and *wall*.

However, many frequently used types of links, such as event/role or cause/consequence, cannot be discovered by these systems because WordNet does not contain such knowledge.

There are four significant differences between Poesio's system and our loose-speak interpreter based system. First, unlike Poesio's systems [116], which only search direct properties of a class, our interpreter also searches properties inherited from the ancestors of a class. For example, in WordNet, the word *barrack* has a *squad room* part, and one of its ancestor *structure* has

the part *foundation*. Our interpreter searches the *foundation* part as well as the *squad room* part of a *barrack*. This difference originates from the fact that our interpreter was originally developed with a knowledge base of formal representations of actions, entities and modifiers. Such knowledge bases typically use subsumption to form a taxonomic graph, and they use inheritance to infer that properties of a general class apply to subclasses as well. In contrast, WordNet was developed as a lexical reference system, and systems that utilize WordNet typically do not use inheritance.

Second, our interpreter uses a more relaxed stopping criterion (`super_or_subclass`) that stops the search when a superclass or subclass of the goal is found. Previous systems [20, 30, 64, 88, 96, 101, 118] used a more restricted stopping criterion (`equality`), in which the search stopped when a class identical to the goal was found. The more relaxed stopping criterion allows both deductive reasoning and abductive reasoning when deciding if the search should terminate. When a subclass of the goal is found, by deductive reasoning we infer there is a path between the starting concept and the goal concept; any instance of the subclass is an instance of the class itself. For example, given ***a barrack*** *...the room...*, the interpreter will start a search from Barrack and will stop the search at the Squad-Room part of the Barrack, because a Squad-Room is a kind of Room, which is the goal concept.

When a superclass of the goal is found, by abductive reasoning we infer that there may be a path between the starting concept and the goal concept: an instance of the superclass may be an instance of the class itself.

For example, given ***a house...the door...***, the interpreter will conduct a search starting from Door, as illustrated in figure 6.2. The search stops at Building, a superclass of the goal, House. This interpretation is reasonable because if a door is part of a building, then the door (the antecedent) **may** be part of a house (the referring expression).

In contrast, when `equality` is used in this example, the search will not stop at Building, because it is not equal to House. Previous studies [41, 79] have shown that a combination of deductive and abductive reasoning is effective for finding semantic paths between two given concepts.

The third difference with WordNet-based algorithms is that our interpreter conducts deeper searches. The search depth limit of previous systems was set to one for computational efficiency. Our interpreter can use a deeper limit because its more relaxed stopping criterion stops most searches at a shallow depth. It can afford to have deeper searches occasionally.

Finally, our interpreter's sorting and selecting function depends on lexical distance rather than salience[2]. Salience is not used because it utilizes the sequential order of sentences in a discourse; our interpreter is a generic solution to the semantic search problem, whose input is an unordered pair of concepts. If experiments show that salience is key to the interpreter's performance for indirect anaphora resolution, the interpreter can be tailored to

[2]Salience is the contextual distance of a referring expression and its anchor.

indirect anaphora resolution problems by replacing the lexical distance based sorting function with a salience-based sorting function.

6.3.1.2 Markert and Hahn

Markert and Hahn [53, 79, 80] built a knowledge-based indirect anaphora resolution system for specialized domains. Similar to our resolution system, it finds antecedents through a breadth first search, and the search uses the `super_or_subclass` stopping criterion.

The biggest difference between the two systems is connectivity, the connection between two concepts. In Markert's system, two concepts, C and D, are connected if one can find an intermediate concept, $C^{'}$ such that C and $C^{'}$ fit the domain and range constraint of a relation r_1 and $C^{'}$ and D fit the domain and range constraint of a relation r_2. The branching factor in the search is

$$|R_{domain}| \times |F_{range}|$$

, where $R_{domain} = \{r_i \mid r_i \text{ is a relation and the domain of } r_i \text{ includes } C\}$ and $F_{range} = \{C_i \mid C_i \text{ satisfies the range of } r_i \text{ and } r_i \in R_{domain}\}$. The benefit of this type of search is coverage because it includes all possible paths between C and D, but the downside is that it may return many false positives. Markert and Hahn used a post-search filtering process to remove the false positives and they ranked the search results based on a heuristic ranking scheme.

In our system, C and D are connected if every C typically relates to $C^{'}$ through relation r_1, and $C^{'}$ typically relates to D through relation r_2. The

branching factor is the number of relations a concept typically has. In the five knowledge bases we experimented with, the branching factor is around 10. This approach has lower coverage, but fewer false positives, smaller branching factors and it is more scalable.

6.3.2 Corpus-based indirect anaphora

There have been many successful machine learning based-coreference resolution systems, such as [22, 74, 107], however most of them do not resolve indirect anaphora. The ones that do [19, 83] typically use the web as the corpus. Instead of searching through WordNet, they issue a series of web search queries made of the referring expression and each candidate antecedent. These systems use the number of web pages that contain both the referring expression and the candidate antecedent being queried as a measure of the strength of association. If the strength exceeds a threshold, then the systems consider the candidate the true antecedent of the referring expression. Machine learning techniques are used to determine the best threshold. The strength of this approach is that it provides a broad coverage of all types of links, and it has achieved precision and recall that are comparable to the WordNet-based systems. The weakness of this approach is that it does not determine the semantic nature of the relationship between the referring expression and the antecedent.

A more recent study of indirect anaphora has shown that precision for either knowledge-based or corpus-based approach can be significantly improved

with a more sophisticated selection mechanism that learns by combining several features, such as salience (the contextual distance of a referring expression and its anchor) and lexical distance (the semantic distance between a referring expression and its anchor) [93]. Improving recall remains a challenge.

6.4 Evaluation

We first evaluated the performance of our interpreter applied to indirect anaphora by comparing it to our reimplementation of a WordNet-based system from previous studies [116]. Poesio's system was chosen for comparison for two reasons. First, it was one of the best knowledge-based systems. Second, part of the data sets used in its evaluation is available for easy comparison, and the knowledge base it uses (WordNet) is also available.

We used WordNet 2.0 as the knowledge base in our experiments for both systems. In order to apply our interpreter, we had to convert WordNet databases into our knowledge base representation. First, we mapped each WordNet synset into a class concept. Then, we mapped WordNet hyponymy and meronymy relations into *subclass*, *has-part*, *element* and *material* relations accordingly (meronymy relation can be mapped to three relations: has-part, element or material). Because it has been shown that the taxonomy derived from WordNet in this way is not of particularly high quality [87] (sometimes a taxonomic sibling should actually be a descendant or an ancestor), we changed the stopping criterion of the interpreter to be *super_or_subclass_or_sibling*.

We used two data sets for the evaluation. The goal of the evaluation

is to measure the performance of the interpreter for indirect anaphora *resolution*, not *detection*. The first data set is a subset of the Brown corpus containing 32 articles on various topics. This data set was annotated for indirect anaphora and used in a previous study [19]. The annotation marks all referring expressions and their antecedents. We used six articles from the data set for debugging, and the rest for testing. The training data were used to set the threshold for the number of sentences in the window for candidate antecedents. We used a six sentence window for our experiment. In the data set, we excluded instances of direct anaphora to isolate the effects on indirect anaphora. We also excluded named entities because WordNet contains little knowledge about them, and recognizing named entities is not central to our task of indirect anaphora resolution. There are a total of 196 instances of indirect anaphora in the test set.

The second data set consists of 32 Wall Street Journal articles from the Penn Treebank I corpus containing 82 instances of indirect anaphora after removing all named entities and direct anaphora. It was used previously [116], and it was partially annotated. We completed the annotations for our experiments.

Because each word in the data sets may be mapped to multiple WordNet synsets, we needed a mechanism of word sense disambiguation. As a simple approach we took the cross product of all the possible word senses of each referring expression and each candidate antecedent to form pairs of synsets. This yields a set of candidate solutions, and the the sorting and se-

lecting function chooses among them. In the previous example, the referring expression, *door*, has five senses, and one candidate antecedent, *house*, has two senses. The cross product yields ten pairs of synsets (<*door#1, house#1*>, <*door#1, house#2*>, <*door#2, house#1*>, ...). The interpreter then interprets the pairs, and the sorting and selecting function chooses the best among all the solutions found as the interpretation for the candidate, *house*.

The metrics we used to evaluate performance are precision and recall, which are defined as follows [77]:

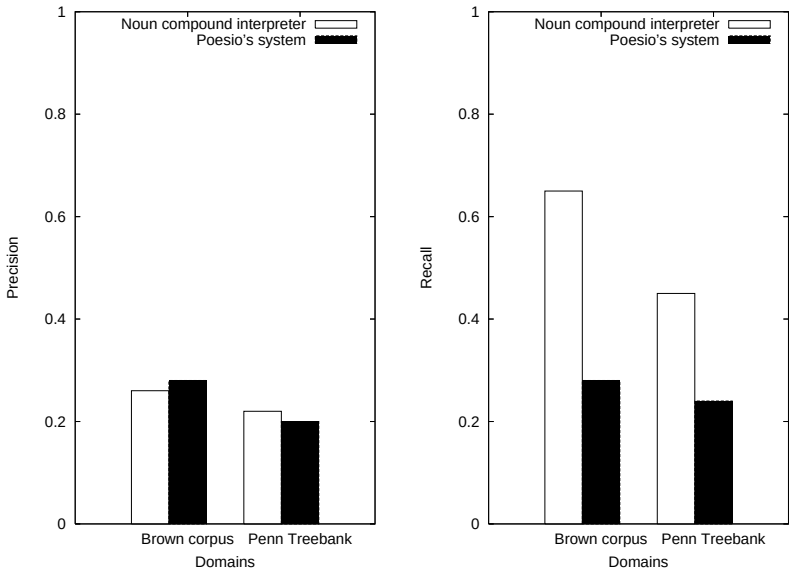
$$Precision = \frac{tp}{tp + fp}$$

$$Recall = \frac{tp}{tp + fn}$$

For our experiment, tp (true positive) is the number of indirect anaphora whose selected antecedents match the annotated ones, fp (false positive) is the number of indirect anaphora whose selected antecedents do not match the annotated ones, and fn (false negative) is the number of indirect anaphora for which the system cannot find an antecedent.

6.5 Results

Figure 6.3 shows the performance of our system and Poesio's system on the two test sets. The left bar is the performance of our interpreter applied to the task of resolving indirect anaphora, and the right bar is the performance



(a) Precision. (b) Recall.

Figure 6.3: Performance comparison of the two indirect anaphora resolution systems on the two data sets. As shown in the data, our interpreter is as precise as Poesio's system, but it approximately doubles the recall.

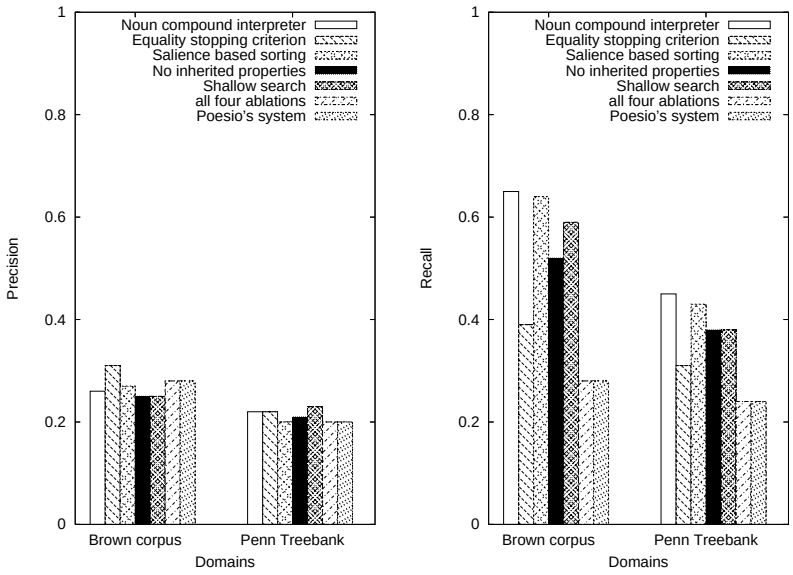
of our implementation of Poesio's system.[3] Our interpreter achieved about the same precision as Poesio's system with approximately twice the recall.

An analysis of the failed cases for both systems revealed that nearly half of them (45% for the Brown corpus data and 49% for the Penn Treebank data) are caused by insufficient knowledge. Because WordNet only provides formal encodings of mereological knowledge (hypernym, hyponym, meronym and holonym), several frequently used type of indirect anaphora, such as Event/role and Cause/consequence, cannot be resolved. For example, given the referring expression *the judge* and candidate antecedent *trial*, WordNet does not contain formally encoded knowledge that would associate "judge" with "trial". In order to improve the performance, other knowledge sources, such as FrameNet [46], the Component Library [7] or formal encodings of WordNet glosses [55], are needed.

6.5.1 Ablation study

Although the analysis of the failed cases reveals that additional knowledge would improve both systems, it does not explain why our system's recall is significantly better than that of Poesio's. As described in section 6.3.1.1, there are four main differences between our interpreter and Poesio's system (use of inherited properties, stopping criterion, search depth bounds and use of lexical distance instead of salience), and some combination of them must account for

[3]We believe our implementation is faithful because on average it performed within 4% of published results [116]. The difference is due to different annotations in the corpora.



(a) Precision. (b) Recall.

Figure 6.4: Performance of our interpreter after a series of ablations. The effect of these ablations on precision is limited, but recall suffered. Of all the ablations, a more restricted stopping criterion has the biggest impact on recall. If all the ablations take place at the same time, then our interpreter behaves essentially like Poesio's system.

the improvement in recall. We analyzed their relative contributions through a series of ablations. Each ablation removes one difference while leaving other aspects of the original interpreter intact. We also removed all four differences at once. This version of the system is functionally identical to Poesio's system, and it should have identical performance as our reimplementation of Poesio's system.

Figure 6.4 shows the results of the ablation study on the two data sets. Replacing lexical distance-based sorting and selecting by salience did not have

a large impact on precision or recall on the two data sets we tested. Shallow searches had a small influence on recall (7 percentage points on the Brown corpus data set and 7 percentage points on the Penn Treebank data set). We have experimented with various search depths, and we found that increasing the search depth beyond two has little effect on performance. Removing inherited properties lowers recall (13 percentage points on the Brown corpus data set and 7 percentage points on the Penn Treebank data set) with little effect on precision. Replacing the stopping criterion has the largest impact on recall. It lowered recall by 26 percentage points on the Brown corpus data set, and 14 percentage points on the Penn Treebank data set. In addition, by removing all four differences, our interpreter performs just like Poesio's system.

The ablation study showed that the biggest increase in recall comes from an abductive stopping criterion, which corroborates the results presented in section 5.2.3.

6.6 Summary

We have found that the interpreter is useful for inter-input loose speak as well. It is able to resolve indirect anaphora at the same level of precision as the state of art indirect anaphora systems, and its recall is significantly higher. The increased coverage is not the result of deeper searches or additional knowledge through inheritance, but it is because of the more relaxed stopping criterion.

Chapter 7

Related Work

7.1 Question answering

There has been a long history of research on question answering systems [69, 120, 121]. These systems typically take in questions formulated in natural language, and they return concise and succinct answers. The earlier ones answer questions based on symbolic representations of knowledge, but they were limited to relatively narrow knowledge domains. This limitation avoids issues, such as loose speak, that would be more important for general systems.

We compare the QUALM [69] system's inferential analysis with the loose-speak interpreter. The QUALM system answers questions about stories; both the input questions and the system's answers are given in natural language. The QUALM system contains five components: conceptual parse, memory internalization, conceptual categorization, inferential analysis and memory search. The first two components, conceptual parse and memory internalization, deal with how to parse and represent the questions. They were developed before QUALM, and they are not specific to question answering. The conceptual categorization classifies a question based on a set of rules.

The inferential analysis rephrases the literal encoding of the question to uncover the unspoken assumptions about the question. The memory search looks for the answer to a given question.

QUALM's inferential analysis component is similar to the loose-speak interpreter: both of them rephrase literal encodings of the input into more precise representation. However they differ significantly in two aspects. First, they emphasize different types of literal encodings. The inferential analysis in QUALM focuses on the literal encodings that are inherent in human cognition, as illustrated by these example queries (part a) and the result of the paraphrase (part b):

(1) a. What haven't I packed?

b. What haven't I packed that I should have packed?

(2) a. Why don't you get Mary a drink?

b. Would you get Mary a drink?

These examples show that the type of interpretations that inferential analysis produces is independent of the underlying knowledge base. In contrast, loose speak occurs in the context of knowledge bases, and its interpretation depends on the idiosyncrasies of the knowledge base.

Second, the QUALM's inferential analysis produces interpretations from rules while the loose-speak interpreter produces paraphrases through mining existing knowledge base. The inferential analysis has a fixed set of rules that

are applied whenever the question encoding meets specific criteria, and they convert the literal encoding into the final interpretation of the question. The loose-speak interpreter's behavior is not restricted to a predetermined set of rules. It decomposes a question encoding into smaller units of triples, and searches the knowledge base for similar triples. This has the advantage of being more flexible, and it may cover a wider variety of idiosyncrasies than rule based systems.

Although the early question answering systems worked well for narrow domains, as the amount of available on-line information increases, the scalability of automated question answering systems becomes more important. Highly scalable systems should allow users to ask questions in natural language, they would search a large corpus and return a succinct answer in real time with sufficient explanation. The Message Understanding Conference [33, 34] and the Text Retrieval Conference [86] have greatly facilitated the development of large scale question answering systems that are capable of answering many questions. The questions that these systems answer are ones that require only retrieval of simple facts. The following is a list of some of the questions used in TREC-9:

(3) When was the Branddenburg Gate in Berlin Built?

(4) What is a meerkat?

(5) How many dogs pull a sled in the Iditarod?

(6) What is the wingspan of a condor?

(7) Who is Barbara Jordan?

The systems [21, 56] combine information retrieval (IR), text summarization, and shallow syntactic and semantic sentence analysis to produce the answers. They use information retrieval to find the relevant passages in the corpus, and they use sentence analysis to parse the question and the retrieved passages. Because of the limited amount of context given in a question and the redundancy in the large corpus, the question answering systems do not have a serious loose speak problem. For example, the FALCON system [56] rarely encounters the problem because it does not have a complete representation of the corpus as its knowledge base, and it does not need to handle loose speak. If the question encoding representation does not match the relevant text representation, for example, the text "Leonov became the first man to walk in space" may contain the answer to the question "who is the first Russian astronaut to walk in space", then FALCON uses an abductive prover to justify using the text as a valid answer.

7.2 Spreading activation

Although the loose-speak interpretation algorithm resembles spreading activation algorithms that were widely popular in the 70s and 80s, there are important distinctions.

In [99], Quillian described how to represent knowledge using semantic

networks. Quillian's network is made of nodes, which represent concepts or instances of concepts, and links, which represent the relations between nodes. There is no separate taxonomy imposed on the concepts, and the links are one-way associations. The knowledge in a semantic network is accessed through spreading activation. Spreading activation (a.k.a. marker passing) is a bi-directional search that stops when the two directional searches intersect.

Since Quillian's original paper, both semantic networks and spreading activation have received considerable attention and refinement in both psychology and artificial intelligence. In the psychology community, Collins [31] proposed a series of refinements to Quillian's theory, such as two-way weighted links, decreasing gradient activation and feature matching based intersection detection. These refinements were added to account for the results of several psychology experiments. In the artificial intelligence community, Fahlman extended semantic network to include taxonomy in spreading activation search [40], and to use large knowledge bases. Spreading activation was used for various natural language processing tasks such as case suggestion and word sense disambiguation [24, 118].

As spreading activation was applied to sizable real-world problems, search depth quickly became the bottleneck. Much of the research gave up "looking for smarts and depended on muscle" [24]. Specialized parallel algorithms and hardware [**?**, 57, 58] were designed to facilitate deep spreading activation searches.

Spreading activation is similar to the loose-speak interpretation algo-

rithm because both of them use bi-directional search that stops when the two searches intersect. However, loose-speak interpretation is not merely the application of spreading activation in knowledge base interaction. They differ in important aspects:

- Key issues. The semantic network representation and the spreading activation search are conceived as a representation and reasoning model, and consequently they are used for a variety of deep inferencing tasks. When solving these deep inferencing problems, the bottleneck becomes how to achieve the bi-directional search efficiently, and hence the rise of specialized parallel computing machines [?, 57, 58]. On the other hand, because loose speak is the slight misalignment between naïve encodings and correct encodings, the problem of interpreting loose speak is not how to search deeper, but rather how to search shallower but finer. In other words, loose speak interpretation calls for looking for smarts instead of depending on muscle.

- Stopping criterion. Because loose speak interpretation calls for shallower but finer searches, it uses a more elaborate stopping criterion. Quillian's spreading activation stops when the two searches have visited the same concept. This stopping criterion is not suitable for loose-speak interpretation because loose-speak interpretation is bound within a shallow depth, and such restricted stopping criterion will result in many false negative results. Instead, a loose-speak interpretation is found when the

two searches are related taxonomically. This more lenient stopping criterion allows more interpretations, both true positive and false positive ones, to be found within a shallow depth constraint.

- Multiple results ranking. Because of the limited search depth, often only a few loose-speak interpretations are found, and ranking them is not an important task for loose-speak interpretation. In contrast, deep spreading activation is more likely to generate multiple results, so ranking is more an issue.

7.3 Underspecification and accommodation

There are two closely related linguistic topics, underspecification and accommodation, that resemble the loose speak problem.

7.3.1 Underspecification

Underspecification [92]is a related topic studied in the computational semantics community. The study of underspecification includes three types:

1. Scope ambiguity, also known as weak structural underspecification, arises when the assignment of universal or existential quantifiers is ambiguous for a given text.

2. Incomplete spoken utterances are often underspecified because they tend to be grammatically incomplete. The human hearer or speech processing system may need to elaborate the input to fully comprehend the inputs.

3. Lexical underspecification is closely related to loose speak. It can be further divided into weak lexical underspecification (such as homonymy) and strong lexical underspecification.

   Here is an example of strong lexical underspecification.

(8) John began the book.

In example (8), the object of the verb *begin* is restricted to be an event, and the noun *book* violates this restriction. A fully specified sentence may look like *John began to read the book* or *John began to write the book*.

Most approaches to studying underspecification are formal. They use meta-variables in a logic representation of the underspecified input, and devise different reasoning mechanism with underspecified inputs. The Quasi-Logic Form (QLF) [3] provides an underspecified treatment of scope as well as certain aspects of referential ambiguity, ellipsis, and focus-background structure. Pustejovsky [97, 98] believed that lexical semantics are generated in a systematic way, and it is best to use complex types for representation so that all aspects of concepts can be captured.

In contrast, we believe that simple types with the assistance of the loose-speak interpreter are sufficient to represent concepts as shown in the evaluation. In example (8), instead of creating a complex type for *book*, the loose-speak interpreter will be able to find all the relevant events, such as *read* and *write*, related to a simple type *book*, and rephrase the input appropriately.

7.3.2 Accommodation

Accommodation [73] is a theory on how to account for the exceptions in common ground theory of presuppositions. According to the common ground theory, there is a set of propositions that the participants in a conversation mutually assume to be taken for granted and not subject to (further) discussion. These propositions describe a set of worlds, the context, upon which speakers agree.

(9) My sister is coming to lunch tomorrow.

However there are instances that do not presuppose the listener's prior knowledge (the common grounds). In example (9), the listener may not know that the speaker has a sister. Accommodation was first introduced by [36] in discourse representation theory as a tool to account for gaps in the discourse.

Accommodation is closely related to indirect anaphora. The accommodation theory takes presuppositions to behave like anaphora, except antecedents are not explicitly stated earlier. The link between indirect anaphora and accommodation is further advanced by [14].

Because of the theoretic nature of accommodation, there is no empirical evaluation that we are aware of, and this makes it difficult to compare the performance of accommodation and the loose-speak interpreter. However, since accommodation is closely related to indirect anaphora, and the loose-speak interpreter has been proven to be effective in resolving indirect anaphora, we

believe it can also be applied to accommodation in similar fashion. Using example (9), the accommodation can be detected because "my sister" is not part of the context, the loose-speak interpreter validates this gap in the discourse by searching a knowledge base of family relations, and it discovers that every sister is a sibling to a person, which is a superclass of the speaker.

7.4 Expectation based knowledge acquisition

It has been long recognized that the knowledge in a knowledge-based system can be utilized to facilitate future knowledge acquisition [35]. One way to take advantage of the existing knowledge is derive expectations that guide knowledge acquisition [11, 12, 38, 50, 63, 78, 111]. Expectations are the predictions of what the user will enter next based on the user's past entries. Expectations have been shown to be effective in knowledge acquisition. In one study [63], the expectation based knowledge acquisition tool provided a 30% savings for knowledge acquisition, and it detected, resolved or even avoided many mistakes early on.

Expectations can come from different sources. Background expectations are derived from the axioms for general concept, and they can be used to assist the acquisition of a more specific concept. Inter-dependent expectations are created among the different factual and procedural components in a knowledge base [12, 38, 63, 111]. By combining these different but related components, a program may detect and resolve the errors in a user's encoding. The following example is given in [12] to illustrate how inter-dependency

expectation can be used for error detection and resolution:

Maximum flight cost = if *Pittsburgh* then 1000.

Notice that the predicate of the *if* statement is not a boolean expression, but rather a location. This type constraint violation signals an error in user's input, and the interpretation routine may rephrase *Pittsburgh* to be *flight destination equals Pittsburgh.*

The inter-dependency expectation based knowledge acquisition is similar to loose-speak interpretation because both of them use existing knowledge to assist a user's future interaction with the knowledge base. They help users by detecting the problems in a user's encoding and providing possible solutions. Their solutions are based on searching the knowledge base for an encoding that is related to the user's original encoding and that conforms to the requirements of the knowledge base.

However, the inter-dependency expectation based knowledge acquisition differs from loose-speak interpretation in two aspects. First they differ in terms of detection. Inter-dependency expectation is used when a syntactic or a semantic error is detected; on the other hand, occurrences of loose speak are entirely semantic. Although the loose-speak interpreter only handles semantic issues, it uses a more active detection method. Loose speak is detected not only by a constraint violation, but also by the failure of the resemblance test (see section 4.3). Our evaluation in section 5.1.1.2 shows that the constraint violation based detection only accounts for 54% of the loose speak occurrences,

and the rest are detected by the `resemblance` test. In addition to our empirical study, other researchers [80] have also shown that constraint violations, such as selectional restriction violations, are not sufficient to detect all occurrences of metonymy; other types of detection methods are needed as well.

Second, they differ in terms of search complexity. Because the inter-dependency expectation can be either syntactic based or semantic based, the consequent search is more free form. It is not as restricted by the triple representation as the ones used in the loose-speak interpreter's `test-and-repair` routine. As a result, there may be more expectations found, and filtering out invalid expectations is an important task. The loose-speak interpreter finds fewer interpretations, so ranking candidate interpretations is not as important.

7.5 Ontology matching

A fundamental operation in knowledge based applications is matching, which takes two encodings and produces a mapping between the elements of the two that correspond semantically. Matching can be used for a variety of applications, such as knowledge integration and analogy interpretation. The need for ontology matching is further fueled by the increasing popularity of the semantic web, in which one agent's ontology often needs to be matched with that of another agent.

There are two approaches to ontology matching [100]. The linguistic based technique matches different elements based on the string similarity of their names or descriptions [16]. The graph based technique matches different

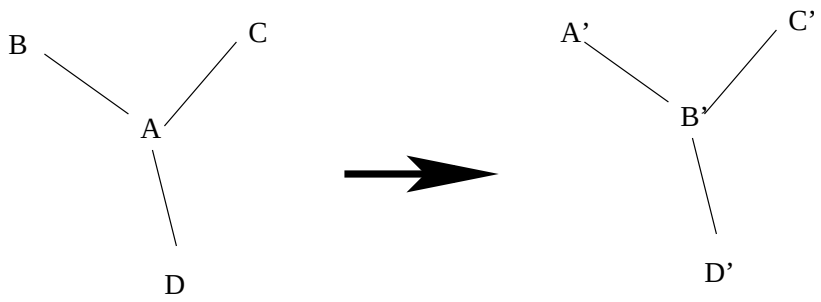


Figure 7.1: An example of an ontology matching algorithm replacing relations in an encoding. The algorithm would map node A to A', B to B', etc., and it would replace the relations in the encoding on the left to make it appear as the one on the right. Notice that none of the relations is elaborated.

elements based on the structure or the semantics of the graph representations of the encodings [49, 122].

One might imagine that ontology matchers can be used to interpret loose speak because the misalignments can be corrected after finding out the mapping from a naïve encoding to that of a correct encoding. Unfortunately, this is not the case for most of the matching algorithms. A typical matching algorithm finds the corresponding element for a given element based on other directly linked elements. In terms of interpretation, such algorithms would only replace a relation in an encoding by another relation, but most of the loose speak types, such as metonymy, involve elaborating relations, not replacing one relation with another. Figure 7.1 shows how such an algorithm would replace the relations in one encoding.

Yeh [122] proposed to include transformations in matching algorithms. A transformation is a rule that would replace one relation in an encoding with a

sequence of relations. Such transformations allow the elaboration of a relation, and can be used to interpret a variety of loose speak occurrences. However, it has two limitations compared to our loose-speak interpreter. First, it cannot address the noun compound type of loose speak because the matcher requires all the semantic relations to be explicit. Second, because each transformation is a rule, the set of transformations needs to be created and maintained manually. The loose-speak interpreter uses spreading-activation search through the the existing knowledge base, and it is not restricted to a predetermined set of idiosyncrasies.

7.6 Semantic-driven natural language processing

An alternative to loose-speak interpretation is semantic-driven natural language processing, which has a much closer coupling of the natural language processing element with the semantic analysis [113]. Unlike the setting in figure 1.2, in which a user, such as a natural language processing unit, produces an encoding independent of the knowledge base, semantic-driven natural language processing integrates the ontology, and uses knowledge for solving problems including semantic role labeling, reference resolution, metonymy resolution and unknown word handling. The solution of these problems utilizes the interpretation of noun compounds [114] as the way to find semantic relations between two concepts, and uses the returned path to assign semantic roles and resolve reference and metonymy.

The advantage of this integrated approach is that it combines both

syntactic features and semantic features of the natural language input, to assist the processing, and because of the additional features, it may generate better results. The disadvantage of this approach is that because of the close coupling, it is difficult to replace a component, such as the parser, with an alternate one.

Chapter 8

Conclusions and future Work

8.1 Summary

As Sir Francis Bacon famously said, "knowledge is power". The importance of knowledge has been long recognized. Knowledge can be used for a variety of artificial intelligence problems, such as natural language processing. However, utilizing this power has not been an easy task.

There are two types of interaction with a knowledge base: querying and updating. In order to correctly perform these operations, one needs to have precise knowledge of how the knowledge base is organized. Any knowledge encoding that does not align with the existing knowledge base's ontology will cause incorrect inference behavior. This problem was significant in Project Halo [117], and a large amount of effort was spent manually aligning encodings with the knowledge base to which they were expected to apply.

In this dissertation, we have focused on the problem of automatically interpreting misaligned inputs to a knowledge base. We have defined *naïve encoding* as an assertion or a query that is encoded without regard for the ontology being used. If, on the other hand, an encoding not only conveys the meaning of the input, but is also compatible with the knowledge base, then we

call it a *correct encoding* of the input. An occurrence of *loose speak* is defined as a misalignment between a naïve encoding and its correct encoding.

The thesis of the dissertation is that misalignments between naïve encodings and correct encodings occur frequently, and moreover with such regularity that the misalignments can be corrected algorithmically. This thesis is manifested in three aspects:

1. The frequency of loose speak occurrence is high.
2. There is regularity in loose speak occurrences.
3. Loose speak can be interpreted by a machine using a single unified solution.

We have conducted empirical studies in various domains on different tasks to address each one of these three aspects. First, we gauged the frequency of loose speak occurrences through two studies. We first analyzed naïve encodings of nearly 300 randomly chosen sentences from three corpora – the Brown corpus [66], Message Understanding Conference corpus [33, 34] and a biology text book [2] – and we found that 80% of the sentence encodings contained at least one occurrence of loose speak. Second, we analyzed 146 naïve encodings of the Advanced Placement chemistry questions used for Project Halo, and we found that 99.3% of the question encodings contained at least one occurrence of loose speak. The results of these studies show that loose speak occurs frequently in knowledge base interactions.

Secondly, the regularity aspect of the thesis is addressed through the analysis of the misalignments found in the two studies. We found that they concentrated on a small number of types. There were six common types of loose speak, some of which occurred as frequently as 87% of the time. Here is the list of these frequently used loose speak types:

- **Metonymy** type of loose speak occurs when one refers to an entity or event (the referent) by one of its features or attributes (the metonym). For example, a user may erroneously encode "elbow" as part of a "body" when the knowledge base has previously identified "elbow" as part of "arm", which is part of the "body". The metonymy type of loose speak includes the commonly used list of metonymic relations compiled by Lakoff and Johnson [67]. It has the following relations: PART-FOR-WHOLE, PRODUCER-FOR-PRODUCT, OBJECT-FOR-USER, CONTROLLER-FOR-CONTROLLED, INSTITUTION-FOR-PEOPLE-RESPONSIBLE, PLACE-FOR-INSTITUTION and PLACE-FOR-EVENT.

- **Causal factor** type of loose speak occurs when one refers to a result by one of its causes. For example, a user may ask questions about the result of mixing two chemicals, even though the result is produced by the reactions caused by the mixing action. It is similar to metonymy since both happen when an attribute is used instead of the thing itself. They differ in terms of the type of relation between the attribute and the thing itself.

- **Aggregate** type of loose speak occurs when one refers to an aggregate by one of its members or vice versa. For example, a single molecule is often used in lieu of a chemical (a set of molecules). Although this loose speak type is similar to PART-FOR-WHOLE metonymy, it differs in terms of the partonomy types to which it applies. The PART-FOR-WHOLE metonymic relation usually applies to heterogeneous sets (e.g. an entity can have different types of parts, such as the various parts of a car), but the aggregate relation usually applies to homogeneous sets (e.g. each individual in the set is a basic unit of the set, such as a single soldier in an army).

- **Overly generic** type of loose speak occurs when a general concept is used in place of a more specific one, such as when "vehicle" is used to refer to a "four-wheel drive truck". This type of loose speak is common in anaphoric references in text.

- **Role** type of loose speak occurs when one refers to an entity instead of the role played by the entity. Roles, such as "spokesman", are things that are in the context of events, such as "speak", and they do not exist in isolation of events like entities, such as "man". Roles, such as "teacher", are often used in the place of the entities, such as "people", that play them.

- **Noun compound** type of loose speak occurs when a user gives a sequence of nouns without specifying the semantic relations between them.

A noun compound, such as "animal virus", is a sequence of nouns composed of a head noun and one (or more) modifiers. The head noun determines the type of the whole compound (with few exceptions), and the modifiers specialize the type of the head noun. The proper semantic relation between the head noun and each modifier can be inferred using a knowledge base, but it would not be easy for a naïve user to encode.

Thirdly, since the results of our analysis show that loose speak occurs frequently and with regularity, this allows us to develop an interpretation algorithm. Instead of developing separate algorithms to interpret each type of loose speak, we developed one algorithm that can accurately interpret all of the frequently used types of loose speak. This interpreter uses existing assertions in the knowledge base as examples to infer idiosyncratic requirements of the underlying knowledge base. The algorithm is made of a `test-and-repair` function that is applied to each triple of an encoding. The `test` part of the `test-and-repair` function detects occurrences of loose speak by checking to see if an input triple violates any knowledge base constraints or if it significantly differs from all existing triples in the knowledge base. The `repair` part finds an interpretation for an occurrence of loose speak through a spreading activation search on the knowledge base with the input as anchor point.

We evaluated the loose-speak interpreter through a variety of studies on different tasks and domains. The user study we conducted involved three users encoding a set of 50 Advanced Placement chemistry questions. We compared their naïve encodings with corresponding output from the loose-speak

interpreter. We found that 96% of the encodings on average contained loose speak, and the loose-speak interpreter corrected these problems with about 95% precision and 90% recall.

Because the chemistry questions are not representative of the noun compound type of loose speak, we also evaluated the interpreter's performance on noun compounds separately. We applied the interpreter to 742 pairs of nouns in three domains: biology, engine repair, and Sparc Workstation. The interpreter performed with about 85% precision and 80% recall.

We also evaluated the effectiveness of the interpreter on encodings generated by a natural language processing system using the data from the second phase of Project Halo. The data were generated by a natural language processing system from simplified English questions across three domains (biology, chemistry and physics). The precision of the loose-speak interpreter on these automatically generated encodings is similar to that on manually generated encodings, and the recall is lower mostly due to natural language processing errors. This suggests that the interpreter can handle human encodings as well as automatically generated correct encodings.

In addition to evaluating the performance of the loose-speak interpreter on single utterances, we tested it on occurrences of loose speak between utterances. Specifically, we evaluated the interpreter's ability to resolve indirect anaphora type of inter-input loose speak. An indirect anaphora arises "when a reference becomes part of the hearer's or reader's knowledge indirectly rather than by direct mention" [85]. For example, given "... a law suit ..." and "the

judge", a reader needs to have the background knowledge of the relationship between "the judge" and the "law suit" mentioned earlier. The fundamental characteristic of indirect anaphora is the assumption that a listener is able to find the association between the entity being used and the entity being referred to. Processing indirect anaphora is difficult because indirect anaphora is used when the listener is assumed to have the common sense knowledge required to identify the link, and obtaining such common sense knowledge is difficult for a computer program. In our evaluation, we used two data sets from two different corpora, which included a total of 278 instances of indirect anaphora, and we used WordNet as the knowledge base. We found that our interpreter performed at the same level of precision as the state-of-the-art indirect anaphora systems, and its recall was significantly higher.

The results of these empirical studies show that not only were we able to utilize the regularity in loose speak occurrences to automatically interpret them effectively, but we were also able to do this using a single unified solution.

Additionally, we addressed a common and justifiable criticism of knowledge-based systems: they require a lot of domain specific knowledge that can be difficult to obtain. We conducted an empirical study of the contribution of the each level of ontology to the success of the interpretation algorithm. The sensitivity of the algorithm to each level of ontology was measured through a series of ablations. The ablation study was conducted on the three noun-compound data sets used earlier. Three separately constructed knowledge bases for the three domains were used to neutralize the variations across domains and across

knowledge bases. The results of the ablation study showed that the top levels of the ontology were most important for the interpretation task – not because they contained more axioms, but rather, because they contained important ontological distinctions and essential axioms.

Previous research has typically addressed a single type of loose speak, in isolation from the rest. For example, prior research has addressed indirect anaphora resolution [94, 116], metonymy resolution [43, 75, 80, 119] or noun compound interpretation [8, 37, 39, 47, 68, 71, 72, 109, 115]. In contrast, our research offers a unified solution to all of these problems. Instead of treating each of these as a separate problem that required its own unique solution, we found one algorithm that solves all of them.

8.2 Future work

8.2.1 Loose-speak interpreter extension

Although we have done most of the loose-speak interpretation work, there are some additional extensions. The current loose-speak interpreter is implemented based on the assumption that the knowledge base being used is perfect. Every assertion in the knowledge base is correct, and no assertion contradicts with another. This assumption works well for carefully crafted knowledge bases, but it does not hold for knowledge bases that are automatically extracted from corpora. One possible extension of the interpreter is detect and resolve loose speak using automatically extracted knowledge bases that contain imperfect knowledge.

8.2.2 Natural language processing application

There are also a variety of possible applications of the loose-speak interpreter. Because the loose-speak interpreter has been successfully applied to three long studied linguistic problems – metonymy, noun compound, and indirect anaphora – there may be other natural language problems, such as discourse analysis, that can utilize the loose-speak interpreter.

8.2.3 Intelligence user interface

The loose-speak interpreter can also be viewed as an intelligent user interface that assists users in interacting with knowledge bases. It may be extended to support other types of intelligent, human-computer interaction.

8.2.4 Knowledge abstraction

In general terms, the loose-speak interpreter provides an abstraction of an underlying knowledge base. It encapsulates the functionality of a knowledge base while sheltering the user from details of the knowledge base structure. Therefore, it facilitates the use of a knowledge base by multiple entities – both human and computer based.

8.2.4.1 Semantic Web

In Semantic Web, different ontologies with different idiosyncrasies are created and distributed across different sites. One ontology may not be familiar with the current idiosyncrasies in another ontology, and when it accesses the

unfamiliar ontology, loose speak may occur. The loose-speak interpreter can be used to assist the communication of different ontologies in Semantic Web.

8.2.4.2 Agent communication

Another example of knowledge encapsulation application is in agent communication. In multi-agent systems, usually the agents share an ontology so that they can communicate effectively. If one agent's copy of the ontology is changed slightly based on a different idiosyncrasy, then all other agents' copies need to be updated accordingly. On the other hand, if every agent has a loose-speak interpreter, then the utterances from the modified agent can be interpreted by other agents based on their copies of the original ontology.

8.3 Closing words

Large knowledge-based systems that cover a variety of domains and reason on diverse topics are much needed, but they are not widely used. One major hurdle in using knowledge-based systems is the difficulty of interacting with them. This dissertation has provided a unified solution that enables an untrained user, either a human or a computer program, to successfully interact with a large knowledge base without knowing the underlying structure of the knowledge base.

Bibliography

[1] *Collins English Dictionary.* William Collins Sons Co. Ltd., 1979.

[2] Bruce Alberts, Dennis Bray, Alexander Johnson, Julian Lewis, Martin Raff, Keith Robert, Peter Walter, and Keith Roberts. *Essential Cell Biology: An Introduction to the Molecular Biology of the Cell.* Garland Publisher, 1998.

[3] H. Alshawi and R. Crouch. Monotonic semantic interpretation. In *Proceedings of the 30th ACL*, pages 32–39, 1992.

[4] Henry F. Atkinson. *Mechanics of Small Engines.* McGraw-Hill, Inc., New York, 1990.

[5] C. Bachman and M. Daya. The role concept in data models. In *Proceedings of the 3rd International Conference on VLDB*, pages 464–476, 1977.

[6] Ken Barker. A trainable bracketer for noun modifiers. In *Proceedings of the Twelfth Canadian Conference on Artificial Intelligence*, Vancouver, 1998.

[7] Ken Barker, B. Porter, and P. Clark. A library of generic concepts for composing knowledge bases. In *Proceedings of First International Conference on Knowledge Capture*, 2001.

[8] Ken Barker and Stan Szpakowicz. Semi-automatic recognition of noun modifier relationships. In *Proceedings of COLING-ACL '98*, Montreal, 1998.

[9] R. Barr and L. Caplan. Category representations and their implications for category structure. *Memory and Cognition*, 15:397–418, 1987.

[10] Tim Berners-Lee. Primer: Getting into RDF & Semantic Web using N3, 2005.

[11] W. Birmingham and G. Klinker. Knowledge acquisition tools with explicit problem-solving methods. *The knowledge engineering review*, 8(1):5–25, 1993.

[12] Jim Blythe. Integrating expectations from different sources to help end users acquire procedural knowledge. In *Proceedings of Seventeenth International Joint Conference on Artificial Intelligence*, 2001.

[13] C. Bock and J. Odell. A more complete model of relations and their implementation: Roles. *Journal of Object-Oriented Programming*, 11:51–54, 1998.

[14] J. Bos, P. P. Buitelaar, and A. M. Mineur. Bridging as coercive accommodation. In Suresh Manandhar et.al., editor, *Computational Logic for Natural Language Processing—Workshop Proceedings*, South Queensferry, Scotland, 1995.

[15] R. Brachman and J. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9:171–216, 1985.

[16] M. W. Bright, A. R. Hurson, and S. H. Pakzad. Automated resolution of semantic heterogeneity in multidatabases. *ACM Transactions on database systems*, 19(2):212–253, 1994.

[17] Theodore L. Brown, H. Eugene LeMay, Bruce E. Bursten, and Julia R. Burdge. *Chemistry: the Central Science.* Pearson Education, Inc., Upper Saddle River, New Jersey, 2003.

[18] Roger A. Browse. Knowledge identification and metaphor. In *Proceedings of the 2nd Biennial Conferene of the Canadian Society for Computational Studies of Intelligence (CSCSI-2)*, pages 48–54, 1978.

[19] Razvan Bunescu. Associative anaphora: a web-based approach. In *Proceedings of the EACL Workshop on the Computational Treatment of Anaphora*, 2003.

[20] R. D. Burke, K. J. Hammond, V. Kulyukin, S. L. Lytinen, N. Tomuro, and S. Schoenberg. Question answering from frequently asked question files: experiences with the FAQ FINDER system. *AI Magazine*, 18(2):57–65, 1997.

[21] Claire Cardie, Vincent Ng, David Pierce, and Chris Buckley. Examining the role of statistical and linguistic knowledge sources in a general-knowledge question-answering system. In *Proceedings of the Sixth Ap-*

plied Natural Language Processing Conference (ANLP-2000), pages 180–187, 2000.

[22] Claire Cardie and Kiri Wagstaff. Noun phrase coreference as clustering. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Very Large Scale Corpora*, 1999.

[23] Loren B. Chan. *SPARCstation 1 installation guide*. Sun Microsystems Inc., 1989.

[24] Eugene Charniak. Passing markers: a theory of contextual influence in language comprehension. *Cognitive Science*, 7:171–190, 1983.

[25] Herbert H. Clark. Bridging. In *Proceedings of the 1975 Workshop on Theoretical Issues in Natural Language Processing*, pages 167–174. Association for Computational Linguistics, 1975.

[26] P. Clark, J. Thompson, K. Barker, B. Porter, V. Chaudhri, A. Rodriguez, J. Thomere, S. Mishra, Y. Gil, P. Hayes, and T. Reichherzer. Knowledge entry as the graphical assembly of components. In *Proceedings of First International Conference on Knowledge Capture*, 2001.

[27] Peter Clark, Phil Harrison, Tom Jenkins, John Thompson, and Rick Wojcik. Acquiring and using world knowledge using a restricted subset of English. In *Proceedings of the 18th International FLAIRS Conference (FLAIRS'05)*, 2005.

[28] Peter Clark and Bruce Porter. Building concept representations from reusable components. In *Proceedings of Fourteenth National Conference on Artificial Intelligence*, pages 369–376. AAAI Press, 1997.

[29] Peter Clark, John Thompson, and Bruce Porter. Knowledge patterns. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Seventh International Conference*, 2000.

[30] P. Cohen and R. Kjeldsen. Information retrieval by constrained spreading activation on semantic networks. *Information processing & management*, 23(4):255–268, 1987.

[31] A. Collins and E. F. Loftus. A spreading-activation theory of semantic processing. *Psychological Review*, 82(6):407–428, 1975.

[32] Deborah A. Dahl. Focusing and reference resolution in pundit. In *Proceedings of Fifth National Conference on Artificial Intelligence*, pages 1083–1088. AAAI Press, 1986.

[33] DARPA. In *Proceedings of the 3rd Message Understanding Conference*. Morgan Kaufmann, 1991.

[34] DARPA. In *Proceedings of the 4th Message Understanding Conference*. Morgan Kaufmann, 1992.

[35] Randall Davis. Interactive transfer of expertise: Acquisition of new inference rules. *Artificial Intelligence*, 12(2):121 – 157, 1979.

[36] Rob A. Van der Sandt. Presupposition projection as anaphora resolution. *Semantics*, 1992.

[37] Pamela A. Downing. On the creation and use of English compounds. *Language*, 53:810 – 842, 1977.

[38] H. Eriksson, Y. Shahar, S. W. Tu, A. R. Puerta, and M. Musen. Task modeling with reusable problem-solving methods. *Artificial Intelligence*, 79:293–326, 1995.

[39] Cecile Fabre. Interpretation of nominal compounds: Combining domain independent and domain-specific information. In *Proceedings of Sixteenth International Conference on Computational Linguistics*, pages 364 – 369, 1996.

[40] S. E. Fahlman. *NETL: A system for representing and using real-world knowledge*. MIT Press, Cambridge, MA, 1979.

[41] James Fan, Ken Barker, and Bruce Porter. The knowledge required to interpret noun compounds. In *Proceedings of Eighteenth International Joint Conference on Artificial Intelligence*, 2003.

[42] James Fan, Ken Barker, Bruce Porter, and Peter Clark. Representing roles and purpose. In *Proceedings of First International Conference on Knowledge Capture*, 2001.

[43] Dan Fass. *Processing Metonymy and Metaphor*. Ablex Publishing, Greenwich, Connecticut, 1997.

[44] Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database*. The MIT Press, Boston, 1998.

[45] R. Fikes and A. Farquhar. Distributed repositories of highly expressive reusable ontologies. *IEEE Intelligent Systems*, pages 73–79, 1999.

[46] Charles J. Fillmore. FrameNet, 2004. http://www.icsi.berkeley.edu/ framenet/.

[47] Timothy W. Finin. *Constraining the Interpretation of Nominal Compounds in a Limited Context*. Lawrence Erlbaum Associates, New Jersey, 1986.

[48] Claire Gardent, Helene Manuelian, and Eric Kow. Which bridges for bridging definite descriptions? In *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora*, 2003.

[49] D. Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7, 1983.

[50] Y. Gil and E. Melz. Explicit representations of problem-solving strategies to support knowledge acquisition. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.

[51] Barbara J. Grosz, Douglas E. Appelt, Paul A. Martin, and Fernando C. N. Pereira. TEAM: An experiment in the design of transportable naturual-language interfaces. *Artificial Intelligence*, 32(2):173–243, 1987.

[52] N. Guarino. Attributes and arbitrary relations. *Data and Knowledge Engineering*, 8, 1992.

[53] Udo Hahn, Katja Markert, and Michael Strube. A conceptual reasoning approach to textual ellipsis. In *Proceedings of 12th. European Conference on Artificial Intelligence*. John Wiley & Sons, Ltd., 1996.

[54] Sanda Harabagiu. Deriving metonymic coercions from WordNet. In *Proceedings of the Workshop on Usage of WordNet in Natural Language Processing Systems, COLING-ACL*, pages 142–148, 1998.

[55] Sanda Harabagiu, George A. Miller, and Dan I. Moldovan. WordNet 2 – a morphologically and semantically enhanced resource. In *Proceedings of ACL-SIGLEX99: Standardizing Lexical Resources*, 1999.

[56] Sanda Harabagiu, Dan Moldovan, Marius Pasca, Rada Mihalcea, Mihai Surdeanu, Razvan Bunescu, Roxana Girju, Vasile Rus, and Paul Morarescu. Falcon: Boosting knowledge for answer engines. In *Proceedings of Text REtrieval Conference (TREC-9)*, 2000.

[57] James A. Hendler. Massively-parallel marker-passing in semantic networks. *Computer Mathematics Applications*, 23:277–291, 1992.

[58] Daniel W. Hillis. *The connection machine.* MIT Press, Cambridge, MA, 1985.

[59] Jerry Hobbs. Syntax and metonymy, 1997.

[60] Jerry R. Hobbs, Mark E. Stickel, Douglas Appelt, and Paul Martin. Interpretation as abduction. Technical Report TR-499, AI Center, SRI International, Menlo Park, California, 1990.

[61] Daniel Jurafsky and James Martin. *Speech and Language Processing: An Introduction to Natural Language Processing Computational Linguistics, and Speech Recognition.* Prentice Hall, New Jersey, 2000.

[62] A. Kilgarriff. BNC database and word frequency lists.

[63] Jihie Kim and Yolanda Gil. Deriving expectations to guide knowledge-base creation. In *Proceedings of Sixteenth National Conference on Artificial Intelligence.* AAAI Press, 1999.

[64] R. Kjeldsen and P. Cohen. The evolution and performance of the GRANT system. *IEEE Expert*, 2(2):73–79, 1987.

[65] K. Knight and S. Luk. Building a large-scale knowledge base for machine translation. In *Proceedings of AAAI '94*, pages 773–778, 1994.

[66] H. Kucera and W. N. Francis. Brown corpus manual. Technical report, Brown University Department of Linguistics, 1979.

[67] George Lakoff and Mark Johnson. *Metaphors We Live By.* University of Chicago Press, Chicago, 1980.

[68] Mark Lauer and Mark Dras. A probabilistic model of compound nouns. In *Proceedings of the 7th Australian Joint Conference on Artificial Intelligence.* World Scientific Press, 1994.

[69] Wendy Lehnert. *The Process of Question Answering.* PhD thesis, Yale University, New Haven, CT, 1977.

[70] Douglas B. Lenat and R. V. Guha. *Building Large Knowledge-Based Systems.* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1989.

[71] Rosemary Leonard. *The Interpretation of English Noun Sequences on the Computer.* Elsevier Science, Amsterdam, 1984.

[72] Judith Levi. *The Syntax and Semantics of Complex Nominals.* Academic Press, New York, 1979.

[73] David Lewis. *Score-keeping in a language game.* Springer, Berlin, 1979.

[74] Xiaoqiang Luo, Abe Ittycheriah, Hongyan Jing, Nanda Kambhatla, and Salim Roukos. A mention-synchronous coreference resolution algorithm based on bell tree. In *Proceedings of ACL 2004*, 2004.

[75] Steven L. Lytinen, Robert R. Burridge, and Jeffrey D. Kirtner. The role of literal meaning in the comprehension of non-literal constructions. *Computational intelligence*, 8(3):416–432, 1992.

[76] S. Moralee M. Uschold, M. King and Y. Zorgios. The enterprise ontology. *The Knowledge Engineering Review*, 13, 1998.

[77] C. D. Manning and H. Schutze. *Foundations of Statistical Natural Language Processing.* MIT Press, Boston, 1999.

[78] S. Marcus and J. McDermott. SALT: A knowledge acquisition language for propose-and-revise systems. *Artificial Intelligence*, 39(1):1–37, 1989.

[79] Katja Markert and Udo Hahn. On the interaction of metonymies and anaphora. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

[80] Katja Markert and Udo Hahn. Understanding metonymies in discourse. *Artificial Intelligence*, 135:145–198, 2002.

[81] Katja Markert and Malvina Nissim. Metonymy resolution as a classification task. In D. Yarowsky, editor, *Proceedings of EMNLP2002*, 2002.

[82] Katja Markert and Malvina Nissim. Towards a corpus annotated for metonymies: the case of location names. In *Proceedings of LREC2002*, pages 1385–1392, 2002.

[83] Katja Markert, Malvina Nissim, and N. Modjeska. Using the web for nominal anaphora resolution. In *Proceedings of the EACL Workshop on the Computational Treatment of Anaphora*, pages 39–46, 2003.

[84] D. McLeod and M. Hammer. Database description with SDM: A semantic database model. *ACM Transactions on Database Systems*, 6(3):351–386, 1981.

[85] Ruslan Mitkov. *Anaphora Resolution.* Longman, London, 2002.

[86] NIST. Text retrieval conference, 2005. http://trec.nist.gov.

[87] Alessandro Oltramari, Aldo Gangemi, Nicola Guarino, and Claudio Masolo. Restructuring WordNet's top-level: The OntoClean approach. In *Proceedings of LREC2002 (OntoLex workshop)*, 2002.

[88] Boyan Onyshkevych and Sergei Nirenburg. A lexicon for knowledge-based MT. *Machine Translation*, 10(1-2):5–57, 1995.

[89] A. Ortony. Some psycholinguistic aspects of metaphor. In R Honeck and R. Hoffman, editors, *Cognition and figurative language*, pages 69–83. Erlbaum Associates, 1980.

[90] Martha Palmer. VerbNet, 2004. http://verbs.colorado.edu/ kipper/verbnet.html.

[91] Aarati Parmar. The representation of actions in KM and Cyc. Technical report, FRG, Stanford, May 2002.

[92] Manfred Pinkal. On semantic underspecification. In H. Bunt and R. Muskens, editors, *Proceedings of the 2nd International Workshop on Computational Semantics*, Tilburg University, The Netherlands, 1999.

[93] Massimo Poesio, Rahul Mehta, Axel Maroudas, and Janet Hitzeman. Learning to resolve bridging references. In *Proceedings of ACL 2004*, 2004.

[94] Massimo Poesio and Renata Vieira. A corpus-based investigation of definite description use. *Computational Linguistics*, 24(2):183–216, 1998.

[95] B. Porter and P. Clark. KM - the knowledge machine: Reference manual. Technical report, University of Texas at Austin, 1998. http://www.cs.utexas.edu/users/

[96] S. E. Preece. *A Spreading Activation Model for Information Retrieval.* PhD thesis, University of Illinois, Urbana-Champaign, IL, 1981.

[97] James Pustejovsky. *Generative Lexicon.* MIT Press, Cambridge, MA, 1995.

[98] James Pustejovsky. The semantics of lexical underspecification. *Folia Linguistica*, 1998.

[99] M. R. Quillian. Semantic memory. In M. L. Minsky, editor, *Semantic Information Processing*, Cambridge, Massachusetts, 1968. MIT Press.

[100] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10:334–350, 2001.

[101] L. Rau. Knowledge organization and access in a conceptual information system. *Information Processing & Management*, 23(4):269–283, 1987.

[102] Robert Schrag, Mike Pool, Vinay Chaudhri, Robert C. Kahlert, Joshua Powers, Paul Cohen, Julie Fitzgerald, and Sunil Mishra. Experimental evaluation of subject matter expert-oriented knowledge base authoring tools, 2001.

[103] Candace L. Sidner. *Towards a Computational Theory of Definite Anaphora Comprehension in English Discourse.* PhD thesis, MIT, Cambridge, MA, 1979.

[104] E. Smith and D. Medin. *Categories and Concepts.* Harvard University Press, Cambridge, MA, 1981.

[105] M. Snoeck and G. Dedene. Specialization and role in object oriented conceptual modeling. *Data and Knowledge Engineering*, pages 171–195, 1996.

[106] H. L. Somers. *Valency and case in computational linguistics.* Edinburgh University Press, 22 George Square, Edinburgh, 1987.

[107] Wee Meng Soon, Hwee Tou Ng, and Daniel Chung Yong Lim. A machine learning approach to coreference resolution of noun phrases. *Computational Linguistics*, 27(4):521–544, 2001.

[108] J. Sowa. *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley Publishing Company, New York, 1984.

[109] Wilco Ter Stal. *Automated Semantic Analysis of Compounds: Problem Identification and Literature Overview.* PhD thesis, University of Twente, The Netherlands, 1996.

[110] F. Steimann. On the representation of roles in object-oriented and conceptual modelling. *Data and Knowledge Engineering*, 35(1):83–106, 2000.

[111] W. Swartout and Y. Gil. EXPECT: Explicit representations for flexible acquisition. In *Proceedings of the Ninth Knowledge-ACquisition for Knowledge-Based Systems Workshop*, 1995.

[112] S. Delisle T. Copeck, K. Barker and S. Szpakowicz. More alike than not–an analysis of word frequencies in four general-purpose text corpora. In *Proceedings of the Fourth Conference of the Pacific Association for Computational Linguistics*, pages 282–287, 1999.

[113] Alicia Tribble. A proposal for knowledge-based labeling of semantic relations in english, 2005.

[114] Alicia Tribble and Scott E. Fahlman. Resolving noun compounds with multi-use domain knowledge. In *Proceedings of FLAIRS-2006*, 2006.

[115] Lucy Vanderwende. Algorithm for automatic interpretation of noun sequences. In *Proceedings of Fifteenth Intional Conference on Computational Linguistics*, pages 782 – 788, 1994.

[116] Renata Vieira and Massimo Poesio. An empirically based system for processing definite descriptions. *Computational Linguistics*, 26(4):539–593, 2000.

[117] Vulcan Inc. Project Halo, 2003. http://projecthalo.com.

[118] David L. Waltz and Jordan B. Pollack. Massively parallel parsing: A strongly interactive model of natural language interpretation. *Cognitive Science*, 9:51–74, 1985.

[119] Ralph M. Weischedel and Norman J. Sondheimer. Meta-rules as a basis for processing ill-formed input. *American Journal of Computational Linguistics*, 9(3-4):161–177, 1983.

[120] Terry Winograd. *Understanding Natural Language*. Academic Press, New York, 1972.

[121] William A. Woods, Ronald N. Kaplan, and Bonnie N. Webber. The lunar sciences natural language information system: Final report. Technical Report BBN Report 2378, Bolt Beranek and Newman Inc., Cambridge, MA, 1972.

[122] Peter Yeh, Bruce Porter, and Ken Barker. Using transformations to improve semantic matching. In *Proceedings of Second International Conference on Knowledge Capture*, 2003.

Index

Vita

James Junmin Fan was born in Shanghai, China on 22 March 1974. He received the Bachelor of Science degree in Computer Sciences from the University of Texas at Austin with highest honor in 1996. After working in the industry for three and half years, he went back to the University of Texas at Austin, from which he received his Master of Sciences in the winter of 2001. He will be working at the IBM T.J. Watson Research Center as a Research Staff Member.

Permanent address: 2400 Drifting Leaf Dr.
Cedar Park, Texas 78613

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.