# A Common-Sense Reasoning System for Mechanical Enginnering

by

Aaron M. Sokoloski

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Bachelor of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2005

Author ..........................................................
Department of Mechanical Engineering
May 6, 2005

Certified by ..........................................................
David Wallace
Associate Professor
Thesis Supervisor

Accepted by ..........................................................
Ernest G. Cravalho
Chairman of the Undergraduate Thesis Committee

# A Common-Sense Reasoning System for Mechanical Enginnering

by

Aaron M. Sokoloski

Submitted to the Department of Mechanical Engineering
on May 6, 2005, in partial fulfillment of the
requirements for the degree of
Bachelor of Science in Mechanical Engineering

## Abstract

This thesis describes the Mechnet system, which was created to test an implementation of analogy-based reasoning about mechanical engineering, focusing on tools and machines that would be found in a shop. Mechnet uses common-sense data about how these things are used and what they are made of, and attempts to judge which objects are similar to each other. The goal was to make "sensible" analogies about things, in that they make sense to a human who has experience in this area.

Results show some reasonable analogies for many tools and machines entered, but non-sensical analogies in several instances. The problems which cause these non-sensical analogies are discussed and analyzed. Recommendations are made for improvement and further research into other kinds of common-sense reasoning.

Thesis Supervisor: David Wallace
Title: Associate Professor

# Acknowledgments

Thank you, Push Singh and Ian Eslick, for helping me think of this project. It was UROPing at the Media Lab that started it all.

Thank you to my thesis advisor, David Wallace, for allowing me to basically do this entire project on my own, and for guiding me toward a clean finished Mechnet.

Thank you, Yang, for your support and for being constantly excited about my work.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Making computers more intelligent is a problem that has inspired many people. The power of computers has increased by several orders of magnitude since they were first created, and work in simulation, 3-D modelling, and other math-intensive programs has kept pace with computing power. However, the perceived intelligence of computers has not, because they lack what humans would term "common sense"[1].

This project aimed to apply reasoning based on analogies, as well as older simple node-traversal reasoning, to a common-sense database of knowledge about tools and machines. The goal was for a program, Mechnet, to make inferences about tools and machines that "make sense" to a human engineer.

For example, one might consider a drill press and a hand drill to be similar tools. Similar names nonwithstanding, the two tools have a number of components in common, and are functionally similar, which is what leads a human to consider them similar. Ideally, if Mechnet contains reasonably accurate data, it should come to the same conclusion. Also, it should not consider similar things which humans do not: a drill and a hammer, for example.

The structure of this thesis is as follows: first, some prior work in Artificial Intelligence is considered. The system structure is described, followed by the data structures used, the user interface, and the implementation of analogy-based reasoning. Results from the finalized system are considered, and the unsatisfactory results are discussed and analyzed. Finally, some future research and improvements are recommended.

# Chapter 2

# Background

## 2.1 Prior Work

### 2.1.1 Rule-Based Logic

The two kinds of reasoning employed in this project have very different histories. Rule-based logic is used, but not in the form in which it normally appears–it is implicit in the functions which traverse links, which are described in section 5.1. Most rule-based logic follows the form "If condition A is true, then condition B is true". All the normal rules of logic follow, for example, if condition B is false, that implies that condition A is false.

Rules can be very useful for representing expert knowledge about very specific domains. They can be used to solve problems that people consider to be difficult: for example, symbolic differentiation (which today is so common that even some pocket calculators have this ability). Wolfram Research's "Mathematica" is a very powerful mathematics program mainly because of its rule-based manipulations of expressions [2].

Another example is for language processing. In "A trainable rule based algorithm for word segmentation", David Palmer uses a rule based system to automatically determine word boundaries in Chinese (and other Asian languages) where they are not made evident by spaces as in English [3].

However, rules are not as useful for solving problems that people consider easy, paradoxically. A strictly rule based system is not good at answering open-ended questions, like “What could I get my baby sister for christmas?”, or for a domain specific example, “How can I bore a hole in a shaft?”

Problems without a definite, easily defined answer are difficult to solve with rule based systems. For example, the question “What could I get my baby sister for Christmas” can have answers like a rattle, baby booties, a stocking, a reindeer doll, etc. If one were trying to answer this with a rule based system, there would need to exist rules like “if [occasion = birthday] and [person is child] then [possible gifts = (rattle, booties, stocking, reindeer doll)]”. The amount of rules needed to address all the little things we just know would be huge. The system breaks cannot produce an answer to any question without specification, for example, “What should I get my baby sister for her birthday?”.

There are ways to generalize, for example, using “holiday” instead of “Christmas” in the rule, but often one wants to retain the flavor that comes with specifying the particular holiday. Rule based systems can be very focused on a particular area, but they are fragile in that the valid problem set is relatively small, especially when compared to the problems a human encounters in everyday life.

### 2.1.2 Common Sense

Questions such as “Where might I buy milk?”, “What can I use to attach wallpaper”, etc. fall more into the realm of what people term “common sense”. Common sense, unfortunately, is a huge domain of knowledge that is built up cumulatively over many years. It could be defined as the knowledge that is common to the vast majority of people in any given culture [1]. This knowledge is mostly automatically learned just by living. Comparing the amount of time spent gathering common sense (nearly every waking hour since birth) to the amount of time necessary to learn, for example, basic calculus (a few hours per day for a semester, to estimate), makes the difference in magnitude of the knowledge involved becomes evident.

Compared to most types of AI systems, relatively few have attempted the task of converting common sense into a computer system. The first and most famous such project is Cyc (http://www.cyc.com/), which was started in 1984 by Douglas Lenat, and only recently has it fully developed. Cyc is actually a rule-based system that has been made more adaptible by separating its rules into “microtheories”, which allow for local consistency, but do not fail on global inconsistencies. Cyc’s rules are completely hand-entered using an assertion language with LISP-like syntax[4].

Another more recent project is the MIT Media Lab’s Open Mind Common Sense project (http://commonsense.media.mit.edu/). This project is also an attempt to give computers common sense, but with a different approach. OMCS is aimed at using the Internet’s ability to gather the work of large numbers of people. The data in this project is gathered when people enter it at the Open Mind website.

On the OMCS website, there are many different forms in which data can be entered. Originally the data was entered as freeform natural sentences and recurring forms parsed out into structured data. Now, most of the data is entered with a structure in mind, for example, the user inputs the effect of a given action, or where something is typically found. [1]

### 2.1.3 Analogy-Based Reasoning

Making analogies about things is something that humans do very well. In fact, Douglas Hofstader, author of ”Gdel, Escher, Bach”, proposes that analogies form the very basis of human thinking, and that analogies, the main way we represent concepts in the mind, are built and evolved over time from early childhood. He gives examples such as how people can easily refer to the circle of bare ground around a tree after a snow as a ”snow shadow”, or how it is much easier for people to understand strange fonts in their mother language than in a second tongue. [5]

When people make analogies, they are essentially creating new knowledge from old knowledge. To create the term “snow shadow” (or understand it without having heard it before) requires understanding the essence of “shadow” rather than just the simple definition. A shadow is commonly known as the absence of light that occurs when an opaque object blocks a light source. “Light”, however, can be replaced with something else that also travels in straight lines, like rain or snow. Humans do not learn this analogous definition of shadow in school however, or perhaps even at all, but Hofstader argues that we do it constantly and automatically.

Using analogies in software, however, is a relatively new technique. It is very useful for extending computation further into the realm of human abilities. In "Semantic Classification of Model Services in an Internet-Based Distributed Modeling Environment", Cao, Senin, and Wallace describe their use of similarity matching to align the interfaces of various models of physical systems in mechanical engineering. Two models with similar (aka analogous) interfaces are matched and combined to form a more generalized interface, a job that in the past has required human input. In this way models with small differences in their interfaces can still be matched and plugged together automatically, which is central to the Distributed Object-based Modeling Environment (DOME) project at MIT [6, 7]

Though this type of analogy is not nearly as advanced as the ones humans are able to make, it is a step in that direction. Making analogies requires looking at the relationships between data rather than just the data themselves. The semantic classification done by Cao, Senin, and Wallace derives higher-level information from many aspects of the parameters of model interfaces and how they are connected. This is the same kind of information a human might use in deciding whether two interfaces are compatible, even in other fields. For example, a pilot learning to fly a new airplane would most likely be able to apply his learned "interfacing" style with the interface of new plane, assuming they were relatively similar. Programming computers to learn new ways of interacting with each other, and especially with humans, makes them more human.

## 2.2 System Structure

The goal of this project was to create a system (Mechnet) where the user can enter simple facts relevant to fabricating something in a shop, and look up information about what a tool can be used for, or how a material can be machined. In addition,

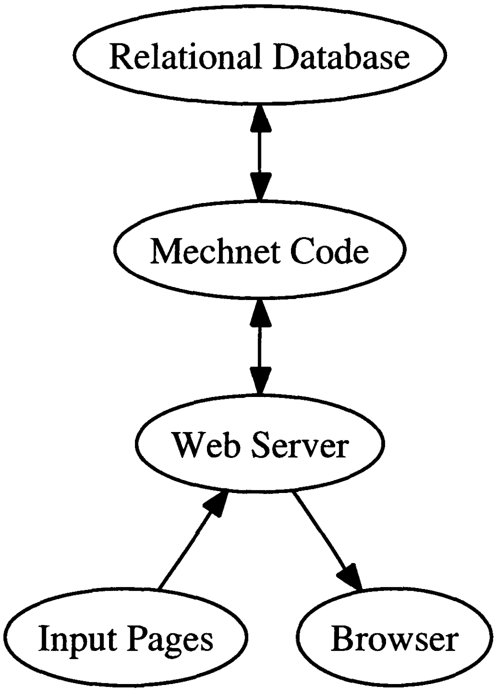


Figure 2-1: High-level data flow within the Mechnet system

the system should be able to do more than just return previously entered data, but also be able to make analogies about tools and materials and machines. This data might be useful for finding substitute tools that perform the functions of a mill, for example, or methods of fastening sheet metal. Finally, these analogies should "make sense", which is a rather subjective goal, but does apply to a common-sense system.

The Mechnet system was designed as a simple centralized webserver through which the user can enter new data and retrieve information about existing data. Practically all computation is done on the server, including everything that is relevant to software reasoning. The data flow diagram in figure 2-1 illustrates the general system structure.

## 2.3 Software Tools

### 2.3.1 Languages

Common LISP was used for the majority of the programming involved in making Mechnet. It is a generally suitable language for artificial intelligence related applications, as well as having open source webserver and database software available. The implementation of Common LISP used was CMUCL (public domain, developed at Carnegie Mellon University)[8].

Javascript and the XMLHttpRequest object were used to do client-side browser scripting for the input suggestion boxes (See Section 4.2.2).

### 2.3.2 Software Packages

MySQL was used for the relational database. It is released under the GPL [9].

The database interface library used was CLSQL, which provided an easily extensible object-oriented LISP interface to the MySQL database. It is released under the LISP Lesser Gnu Public License (LLGPL) [10].

The AllegroServe webserver was used for the entire website. It is also released under the LLGPL[allegroserve].

### 2.3.3 Other

The regular expressions used for the pluralization of most English nouns were adapted from the online book "Dive into Python" [11]. The expressions to change plural nouns to singular were derived from examples provided in this book.

# Chapter 3

# Data Representation

The structure of the database goes a long way in helping or hindering the processing that is performed on these data. The most versatile form of data that could be used to represent common-sense plain English is also very difficult to process and do anything useful with.

> "...natural language, the symbol system that is easiest for humans to learn and use, is hardest for a computer to master"[12]

While natural language communication with computers is a very important goal in the AI field, it is beyond the scope of this project.

At the other extreme, the structure of concepts and their properties can be specified in great detail, using a specially-designed language. A good example of this approach is the Cyc project. Entering facts into the Cyc database directly requires learning the S-expression based CycL language, which resembles Lisp code. Ways to map other ontologies into CYC are also being developed, such as Medical Subject Headings (MeSH), Wordnet, Unified Modeling Language (UML) and others [13].

While this kind of language allows the data entered to be very easily processed and used, it is inappropriate for a database where all users will be encouraged to enter facts.

A compromise between natural language and a fully structured artificial language was chosen. Pre-formed sentences were used for data entry (with blanks to be filled in) which describe a specific relationship between the phrases entered. The sentences used and the relationships they encode are described in the following sections.

## 3.1 Structure

In order to minimize both the complexity of the data representation and the complexity of the program that accesses it, a simple, minimally structured format was used. The database contains two types of objects and three types of links that relate them. The objects categories are “*Thing*” and “*Action*”, while the links are “*Ability*”, “*Hypernym*” and “*Composition*”.

The database program used was MySQL, an open source relational database server. The *Thing* and *Action* objects were implemented as tables containing an integer primary key and a “name” field, which simply hold a noun or verb phrase, respectively. The *Thing* object also contained an additional field called “lemmatized”, which consisted of the name field with pluralization and articles (a, an) removed. This field is a reference field, used to compare newly entered *Thing*s with those already existing in the database in order to avoid duplication. *Thing*s referenced in new link entries are lemmatized and the database is checked for matches before creating a new *Thing*.

All the link tables are effectively many-to-many join tables. They all contain the ids of two *Thing*s, with the *Ability* link additionally containing the id of an *Action*. This allows, for example, a *Thing* to have many Parts, as well as allowing a *Thing* to be a Part of many *Thing*s (see Fig 3-1).

### 3.1.1 Data Type *Thing*

A *Thing* is any kind of object. In the context of this project, a drill bit is a *Thing*, aluminum is a *Thing*, and a flat surface is a *Thing*. The *Thing* category is intentionally broad, including things which can be measured by number of units or other quantities, for example, heat. It can also include modifiers such as “flat” or “stiff” along with the noun that forms the main concept of the *Thing*.

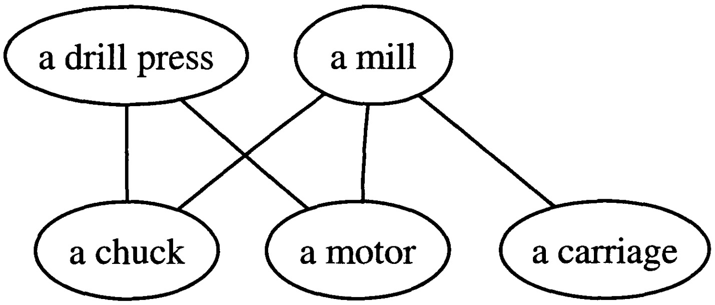


Figure 3-1: Possible many-to-many relationship of Things to their Parts

In this way, the wideness of the *Thing* category captures the ability of humans to think of abstract concepts, like thinking of the missing material that constitutes a *hole* as a thing rather than the absense of a thing. This makes it more natural to think about the concepts expressed in the Mechnet database, without stopping to think about what a *hole* really is.

### 3.1.2 Data Type *Action*

If *Things* are nouns, *Actions* are verbs. An *Action* is anything that can be done to a *Thing*. For example, one can *make holes in* a *Thing*, or *heat* a *Thing*, or *attach* a *Thing* (to something else). In many cases, prepositions or adverbs are included in an action, such as *make holes in* as mentioned above, *remove burrs from*, or *quickly attach*.

Sometimes, as in the case of to *attach* something, the verb is intended to take an indirect object. In these cases the indirect object is ignored. Considering it would introduce additional complexity but was not predicted to increase the ability to represent concepts by very much, thus giving very little benefit to the reasoning capabilities of the system. Many such small refinements and additions in the data structure were considered and rejected as not being essential to the main purpose of the project.

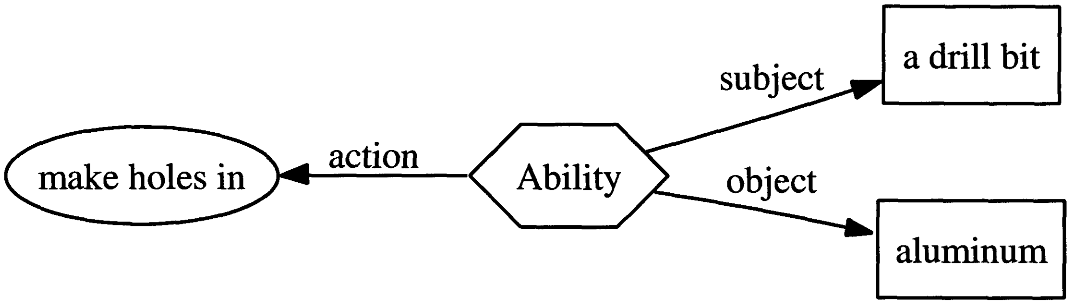


Figure 3-2: Database representation of *Ability* link

### 3.1.3 Link Type *Ability*

An *Ability* is simply a link that connects an *Action* to a *Thing* that can perform it, and a *Thing* that it can be performed upon. The sentence format that represents this is "You can use [*Thing*] to [*Action*][*Thing*]". For example, "You can use **a drill bit** to **make holes in aluminum**". In this example of an *Ability* link, the *Thing* called *a drill bit* is referred to as the Subject and *aluminum* is the Object. Figure 3-2 shows how an *Ability* link expresses the sentence above.

This link is intended to represent a large portion of the common-sense knowledge about how tools and machines are used. For example, a student might wish to know how to make large holes in wood, when most conventional drill bits tend to be less than 1" in diameter. The *Ability* link between *a forstner bit make large holes in* and *wood* would represent a solution to this student's problem. An important part of mechanical engineering hands-on experience is knowing when to use the most appropriate tool or machine to acheive a particular result.

### 3.1.4 Link Type *Hypernym*

A *Hypernym* is basically a categorical link, which expresses an "is a" relationship. The sentence that represents this is simply "[*Thing*] is / are [*Thing*]" (this allows for plurals as well as singular noun phrases to be entered). Examples include *a forstner bit* is *a drill bit*, *aluminum* is *a metal*, and *a flat surface* is *a surface*. The more general *Thing* is referred to as the Hypernym, and the more specific instance, the Hyponym.

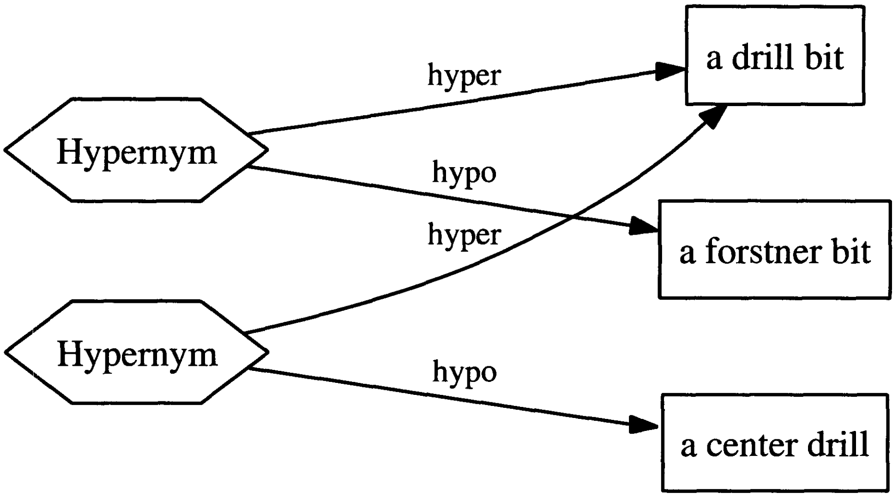


Figure 3-3: Database representation of *Hypernym* link

3-3 shows a typical *Hypernym* link.

The *Hypernym* link serves two closely related purposes. One is the representation of instances of a class, for example, *a forstner bit* is a specific kind of drill bit. The other is to represent specificity, as in *a flat surface* is *a surface*, with the added constraint that it must be flat. Only the first is truly a hypernym by definition, however, it is convenient to group both types of link under the same name.

These two types of *Hypernyms* are very similar and can easily be confused, however, it should be noted that the specific vs. general *Hypernym* can sometimes be automatically inferred from the lexical relationship between the two *Things*. In the example of *a flat surface*, it can be automatically detected that *flat* is an adjective constraining *a surface* and the link created automatically. However, this can go wrong, for example, with *a masonry nail*, which is not *a nail* that is *masonry*. Simply entering these links by hand where necessary was judged to be more appropriate, as this project derives little benefit from separating *Things* into nouns and adjective modifiers.

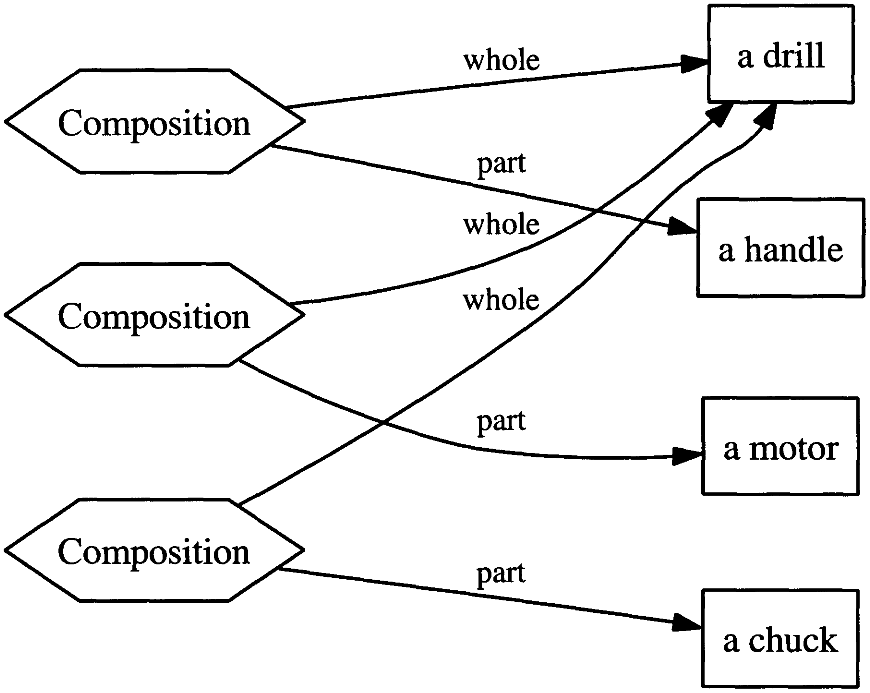


Figure 3-4: Database representation of *Composition* link

### 3.1.5 Link Type *Composition*

The third and final link, *Composition*, was added to represent how *Things* can be viewed as a collection of parts. The *Composition* link is described by the sentence "[*Thing*] has / contains [*Thing*]". For example, *a drill* has *a chuck* and *a handle* (these would be stored as two links). The *Thing* known as *a drill* is referred to as the Whole in this type of link, and it contains *a chuck* and *a handle* as Parts (Fig 3-4).

The composition link helps to represent common knowledge about parts of tools and machines. For example, knowledge about how to use a vise applies not only to bench vises, but to a vise on a horizontal bandsaw or a mill carriage.

Thus, the composition link is very useful in judging the similarity between *Things*. *Things* that have overlapping lists of parts are likely to be similar. For example, *a reciprocating saw* contains *a motor* and *a handle*, like *a drill*, and thusly is likely to be similar (which is true in that both are hand-held power tools).

## 3.2 Memory Cache

Both for programmatic ease and efficiency purposes, a cache of the data in the database was created using CLSQL objects, which are an extension of the normal Common Lisp Object System (CLOS). The cache represents the links between objects using LISP's basic list datatype. For example, an object's abilities are stored as a list of lists, the first element of each referring to a CLOS object representing an *Action*, and the second to the *Thing* which is the object (in the grammatical sense) of the action. This field is called "ability-subject", "ability" being the link name, and "subject" meaning the slot in the link occupied by this object (see Fig. 3-5). In this naming scheme, an object's hyponyms are stored in the "hypernym-hyper" slot, which is perhaps unintuitive but consistent. This form of cache allows for fast link traversal and retrieval of linked nodes.

Obviously, the size of the cache is limited by the available memory. As every *Thing* and *Action* is stored in memory only once (as in the MySQL database), the caching scheme is relatively efficient. As the database grows however, and sections of memory are necessarily paged to disk, the efficiency gains may be offset by the slowness of virtual memory. See 7.1.1 for a possible solution to this problem.

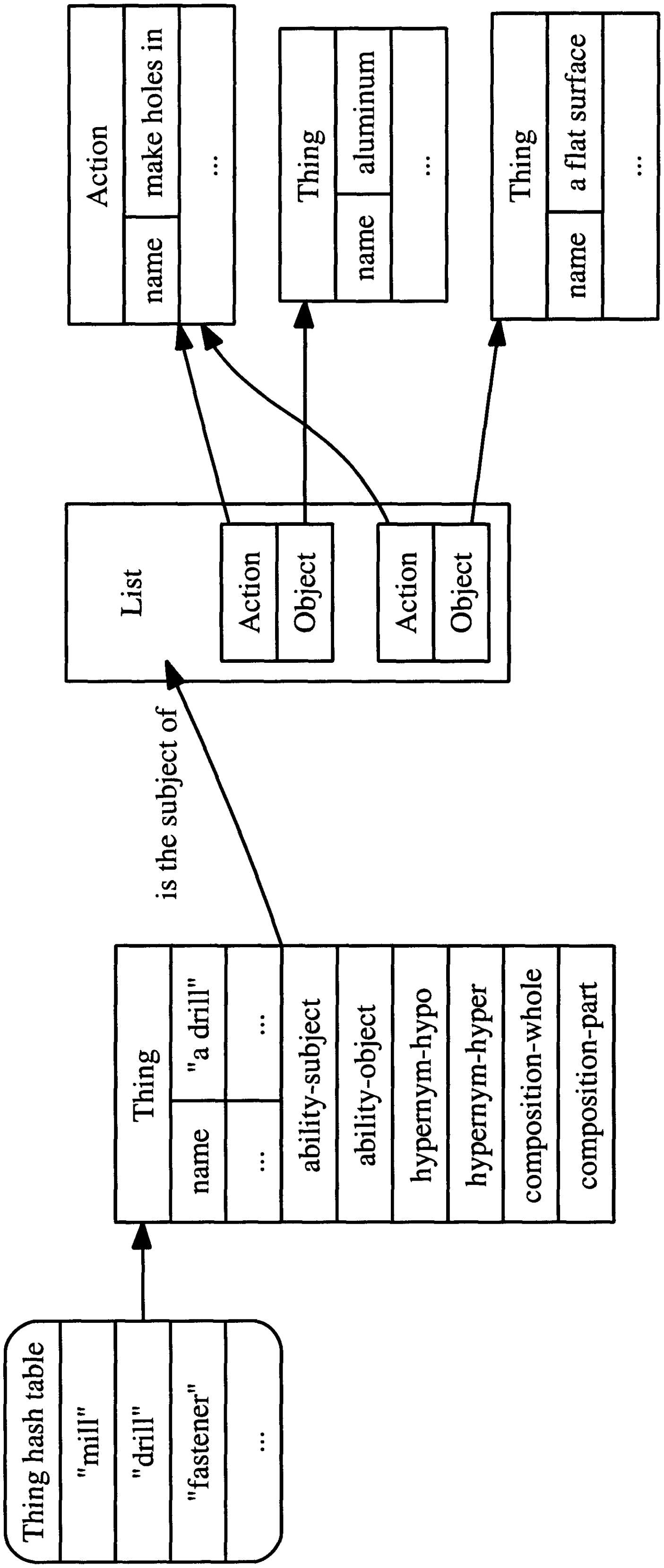


Figure 3-5: Object caching scheme

# Chapter 4

# User Interface Design: Mechnet Website

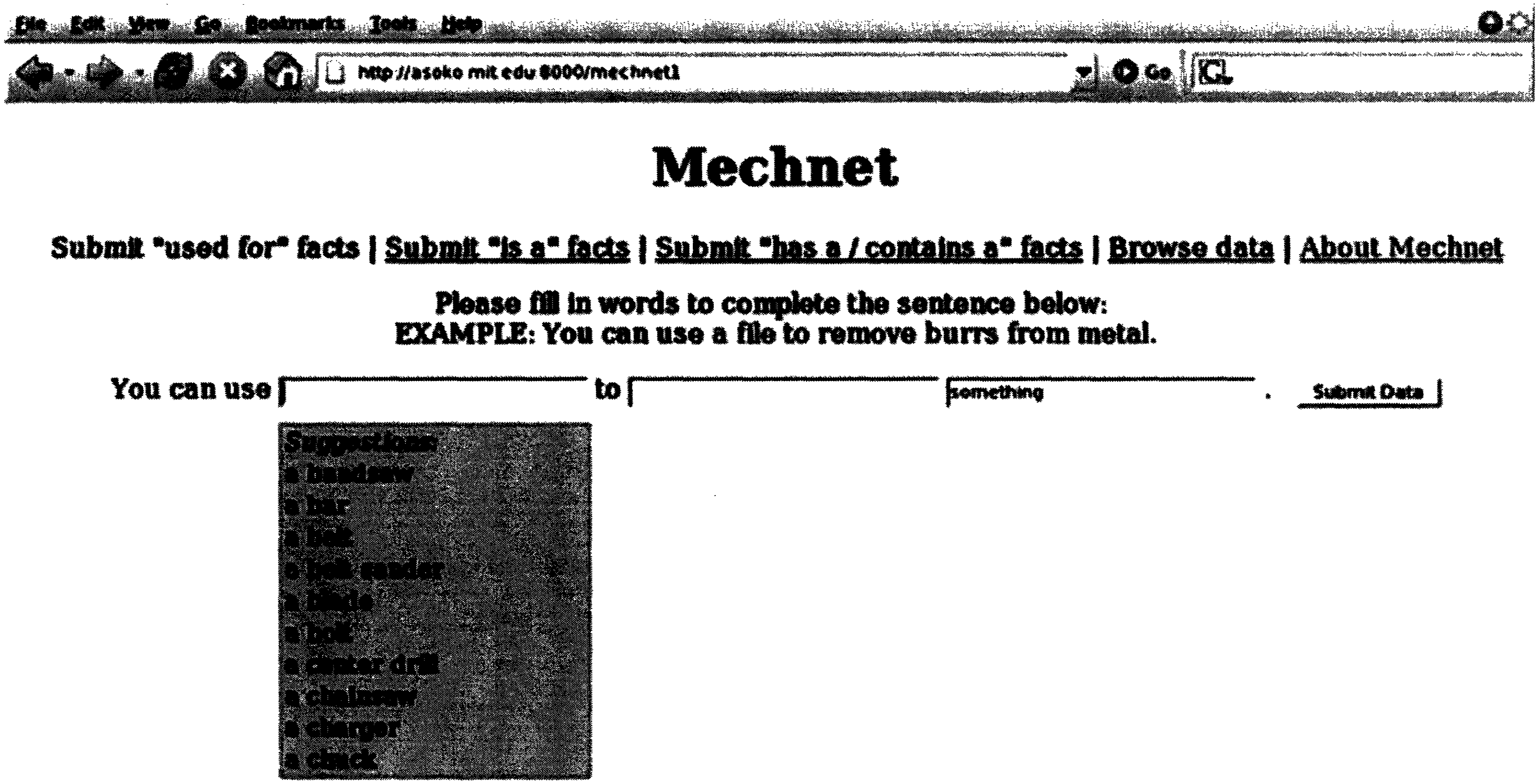
The website consists of four main pages, one for the entry of each of the three link types, and another page which is a browser for all the data in the database, along with different types of inferred data. Figures 4-1 through 4-3 are screenshots of the display of the three data entry pages.

On these pages the sentence form for data is clearly visible. These pages were kept as simple as possible, leaving room for possible improvement of the interface in the future. See Section 4.2.2 for an explanation of the "Suggestions:" box underneath some of the input boxes.

## 4.1 Mechnet Browser

The browser page display is the main focus of this project. It shows all the data that has been entered for a particular *Thing*, as well as information that was inferred by link traversal or analogy. The browser pages are generated dynamically , so changes in the database are reflected when the page is reloaded. All of the algorithms used to generate the output described in this section is are discussed in section 5.2.

The page shown in figure 4-4 is the browser display for the *Thing* with name *a bandsaw* (stored as lowercase in the database and capitalized as needed). The first



File Edit View Go Bookmarks Tools Help
http://asoko.mit.edu:8000/mechnet1
Go

# Mechnet

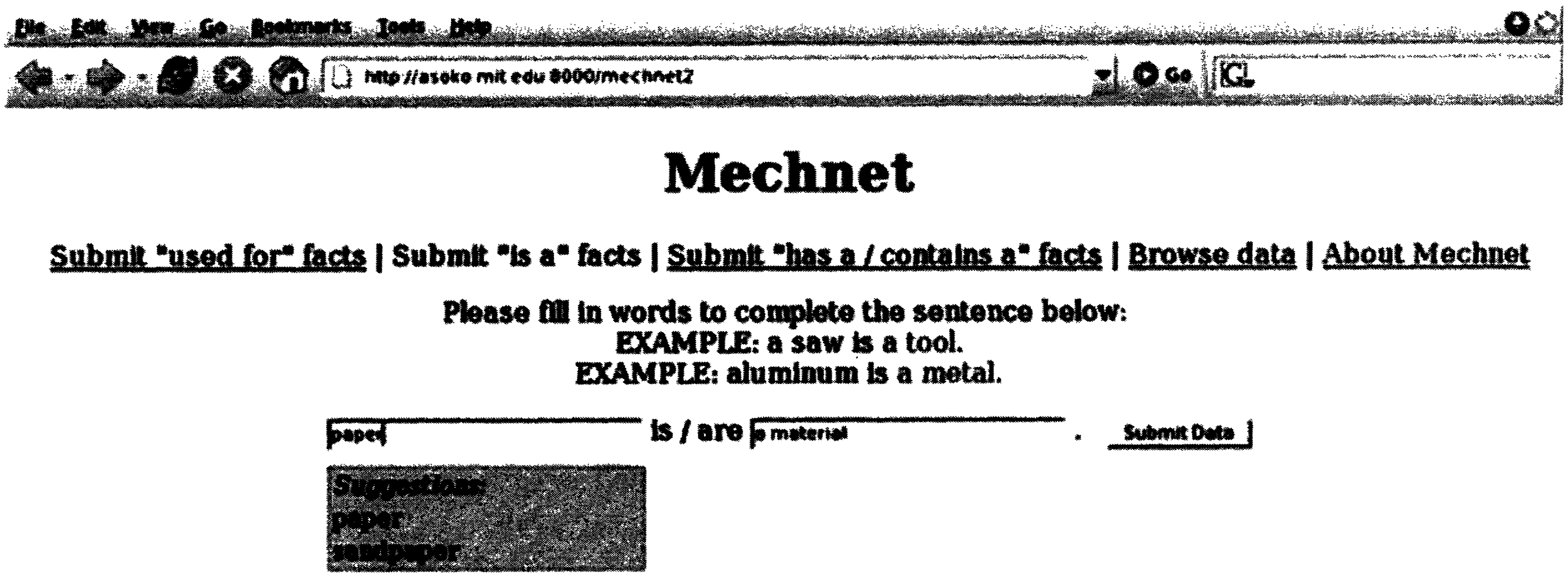
Submit "used for" facts | Submit "is a" facts | Submit "has a / contains a" facts | Browse data | About Mechnet

Please fill in words to complete the sentence below:
EXAMPLE: You can use a file to remove burrs from metal.

You can use ________ to ________ something ________ . Submit Data

Suggestions:
a bandsaw
a bar
a belt
a belt sander
a blade
a bolt
a center drill
a chainsaw
a charger
a chuck

Figure 4-1: Screenshot of the *Ability* entry page


File Edit View Go Bookmarks Tools Help
http://asoke.mit.edu:8000/mechnet2
Go

**Mechnet**

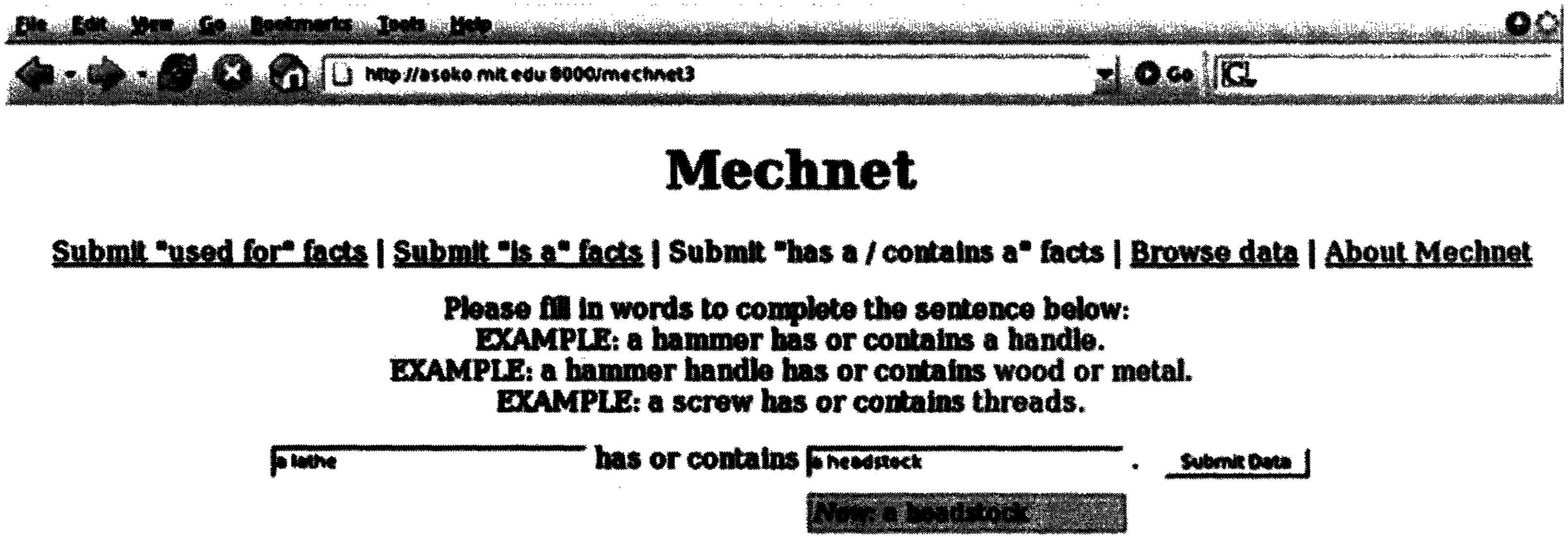
Submit "used for" facts | Submit "is a" facts | Submit "has a / contains a" facts | Browse data | About Mechnet

Please fill in words to complete the sentence below:
EXAMPLE: a saw is a tool.
EXAMPLE: aluminum is a metal.

paper is / are a material .

Suggestions:
paper
sandpaper

Figure 4-2: Screenshot of the *Hypernym* entry page


File Edit View Go Bookmarks Tools Help
http://asoko.mit.edu:8000/mechnet3
Go

Mechnet

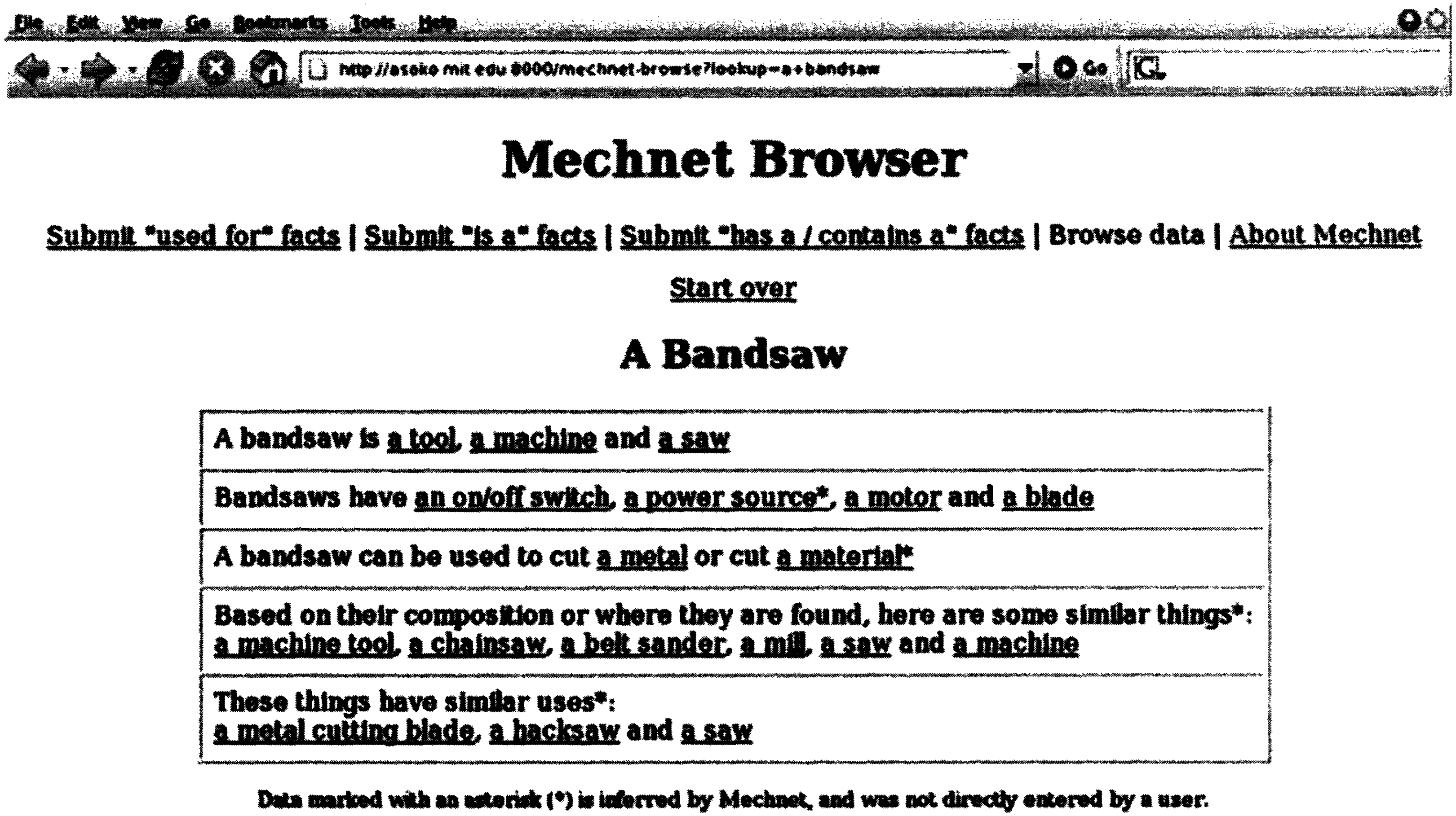
Submit "used for" facts | Submit "is a" facts | Submit "has a / contains a" facts | Browse data | About Mechnet

Please fill in words to complete the sentence below:
EXAMPLE: a hammer has or contains a handle.
EXAMPLE: a hammer handle has or contains wood or metal.
EXAMPLE: a screw has or contains threads.

a lathe has or contains a headstock . Submit Data

New: a headstock

Figure 4-3: Screenshot of the *Composition* entry page


File Edit View Go Bookmarks Tools Help
http://asoko.mit.edu:8000/mechnet-browse?lookup=a+bandsaw
Go

# Mechnet Browser

Submit "used for" facts | Submit "is a" facts | Submit "has a / contains a" facts | Browse data | About Mechnet

Start over

## A Bandsaw

A bandsaw is a tool, a machine and a saw

Bandsaws have an on/off switch, a power source*, a motor and a blade

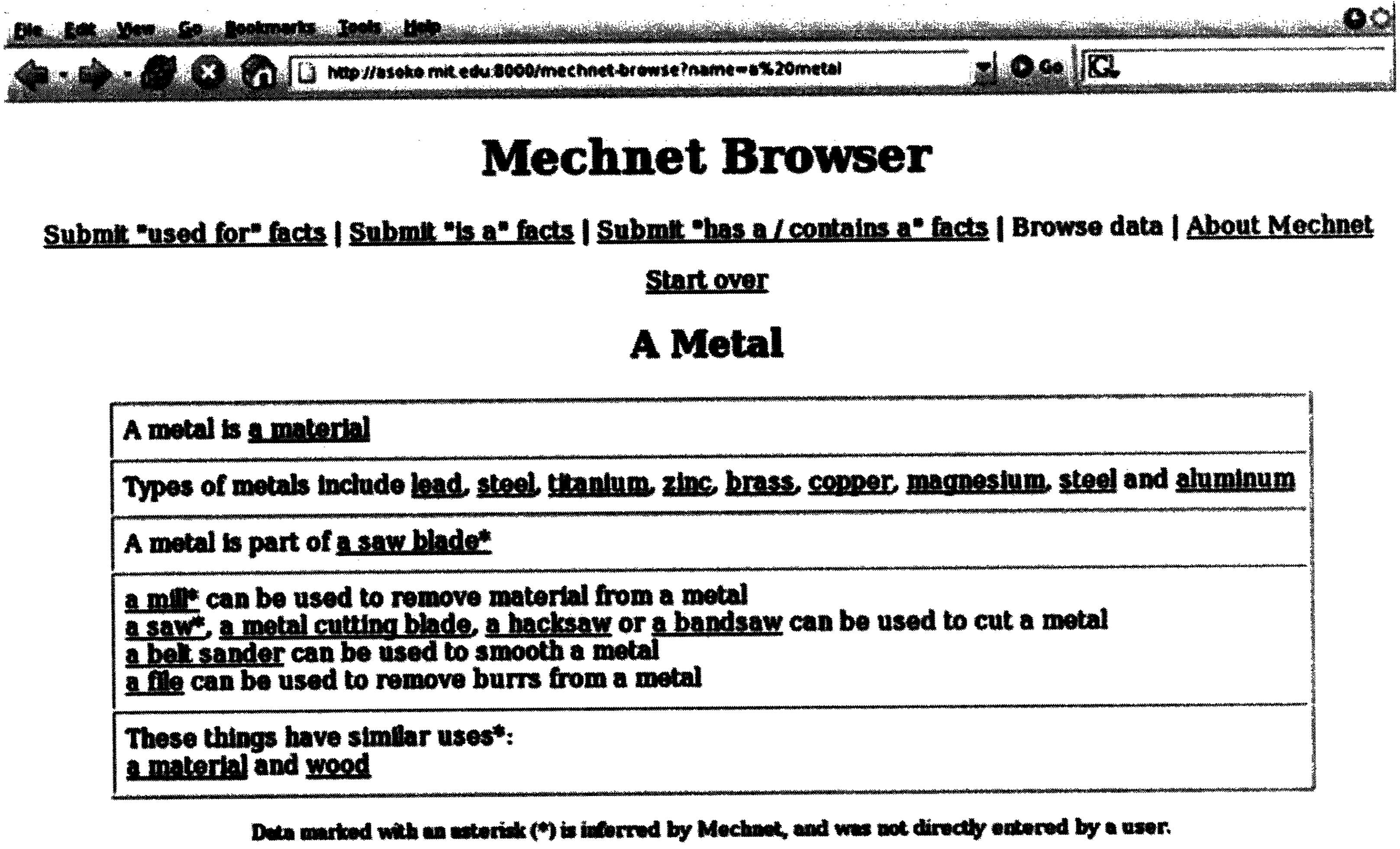
A bandsaw can be used to cut a metal or cut a material*

Based on their composition or where they are found, here are some similar things*:
a machine tool, a chainsaw, a belt sander, a mill, a saw and a machine

These things have similar uses*:
a metal cutting blade, a hacksaw and a saw

Data marked with an asterisk (*) is inferred by Mechnet, and was not directly entered by a user.

Figure 4-4: Screenshot of the Mechnet Browser for *a bandsaw*


File Edit View Go Bookmarks Tools Help
http://asoke.mit.edu:8000/mechnet-browse?name=a%20metal
Go

# Mechnet Browser

Submit "used for" facts | Submit "is a" facts | Submit "has a / contains a" facts | Browse data | About Mechnet

Start over

## A Metal

A metal is a material

Types of metals include lead, steel, titanium, zinc, brass, copper, magnesium, steel and aluminum

A metal is part of a saw blade*

a mill* can be used to remove material from a metal
a saw*, a metal cutting blade, a hacksaw or a bandsaw can be used to cut a metal
a belt sander can be used to smooth a metal
a file can be used to remove burrs from a metal

These things have similar uses*:
a material and wood

Data marked with an asterisk (*) is inferred by Mechnet, and was not directly entered by a user.

Figure 4-5: Screenshot of the Mechnet Browser for *a metal*

section on this page contains “A bandsaw is **a tool**, **a machine** and a **a saw**”. The bolded text represents *Things*, and each *Thing* name is a link to its own browser page. Since none of these *Things* have asterisks next to their names, this means that they all have direct links to “a bansaw” with no *Things* in between. In other words, these three data were entered directly at some point.

The next section displays “Bandsaws have an **on/off switch**, a **power source***, **a motor** and **a blade**”. Only *a power source** has an asterisk. This indicates that “A bandsaw has a **power source**” is something that Mechnet inferred, because a hypernym of *a bandsaw*, *a machine*, has a power source.

After that is shown “A bandsaw can be used to cut **a metal** or cut **a material***”. Again, the ability to cut *a material** is derived from one of the hypernyms of *a bandsaw*, in this case, *a saw.*

The next section is more complicated. Here, Mechnet is showing some *Things* that are similar to *a bandsaw* because of what parts they have, or what they are a part of. *A bandsaw*, obviously, is more likely to have Parts than to be a Part of something, and in fact these analogies are all made on the basis of having similar Parts, rather than being Parts of similar *Things*.

> Based on their composition or where they are found, here are some similar things*: **a machine tool**, **a drill**, **a chainsaw**, **a belt sander**, **a mill**, **a saw** and **a machine**.

Mechnet is showing *Things* that have similar *Composition* links to *a bandsaw.*

The following section is similar:

> These things have similar uses*: **a metal cutting blade**, **a hacksaw** and **a saw**.

In this case, Mechnet is comparing the *Ability* links of the *Things* shown, rather than the *Composition* links. Again, *a bandsaw* is more often the Subject of an action than an Object (such as *a metal*, which is cut). However, a *Thing* can also be found to be analogous to another *Thing* on the basis of having similar actions performed upon it.

The Browser display for *a bandsaw* does not show every display possibility, but the page for *a metal* illustrates the ones that it lacks (See Fig 4-5). The first new section contains:

> Types of metals include **lead**, **steel**, **titanium**, **zinc**, **brass**, **copper**, **magnesium**, **steel** and **aluminum**.

These are the *Things* that have *Hypernym* links to *a metal.* All are direct links, since none have asterisks.

The next section displays "A metal is part of **a saw blade***". This *Composition* link is derived from one of the links of *a metal*'s hyponyms. Clicking on **a saw blade*** displays its browser page which asserts that "Saw blades have **steel**", thus making the particular hyponym evident.

The next section shows the Abilities which have *a metal* as an Object.

- a mill* can be used to remove material from a metal
- a saw*, a metal cutting blade, a hacksaw or a bandsaw can be used to cut a metal
- a belt sander can be used to smooth a metal
- a file can be used to remove burrs from a metal

These Abilities are grouped by the type of *Action* being performed. *A metal*, unlike *a bandsaw*, is more often the reciepient of an action.

With all these sections, the Browser shows a fairly detailed description of the relationship of a *Thing* to the other *Things* in the database, both explicitly defined relationships, implicit ones, and inferred analogies.

## 4.2 Specific Features

There were some features implemented with ease of use and cosmetics in mind. These are intended to make using Mechnet natural in order to encourage rapid entry of data and well-formed entries.

Table 4.1: Example of lemmatized *Thing* names

| **Name** | **Lemmatized** |
|---|---|
| a hammer handle | hammer handle |
| threads | thread |
| a blade | blade |
| a saw | saw |
| a nut | nut |

### 4.2.1 Lemmatization

As every new *Thing* is added to the database, its name is lemmatized - any indefinite articles ("a" or "an") are stripped off, and it is converted into a singular noun phrase if it is plural [11], and finally this lemmatized version is stored in the *Thing*'s "lemmatized" field in the database. Table 4.2.1 shows various *Things* and how their names are lemmatized.

Having a lemmatized version of all *Thing* names simplifies pretty-printing for the Browser, as pluralization can easily be added back in, for example, in the phrase "Types of **metals** include ...". The biggest benefit, however, is that *Thing* names can be compared based on the lemmatized database version in plain SQL, so it is simple and efficient to determine if a *Thing* already exists in the database before adding it.

### 4.2.2 Automatic Suggestions

In order to help users know what sorts of *Things* are already present in the database, when a user starts typing in an input box, database *Things* (or *Actions*, depending on the field) that match their text are displayed in real time below the box. This feature is derived from the idea behind Google Suggest (Beta) [14].

The entries in the suggestion boxes are simply plain text matches to the Name field of *Things* or *Actions* in the database. They are updated on every javascript "onKeyUp" event for the text box.

# Chapter 5

# Reasoning Implementation

In the sections below, it is often useful to show the output of functions with sample data at the LISP Read-Eval-Print loop. In this case, lines beginning with ``MECHNET>'' are expressions typed at the prompt, and lines beginning with ``=>'' show the return value of whatever expression was typed.

Also, the **get-thing** function used in the examples below simply looks up a *Thing* in the cache, or, failing that, the database.

## 5.1 Helper Functions

The two most basic helper functions used in the analogy generation, are **get-all-hypernyms** and **get-all-hyponyms**. These have symmetric functionality simply by recursively traversing *Hypernym* links up and down respectively, returning them all in a list. By default, this list contains CONS pairs (aka dotted lists) of each *Thing* and the symbol 'DIRECT for direct links, and 'INDIRECT for indirect links. For example,

```
MECHNET> (get-all-hypernyms (get-thing "a lathe"))
=> ((#<T=153 a machine tool> . direct) (#<T=54 a machine> . indirect))
```

This shows that *a machine tool* is a directly entered hypernym of *a lathe*. *A machine* is a hypernym of *a machine tool*, and thus is an indirect hypernym of *a*

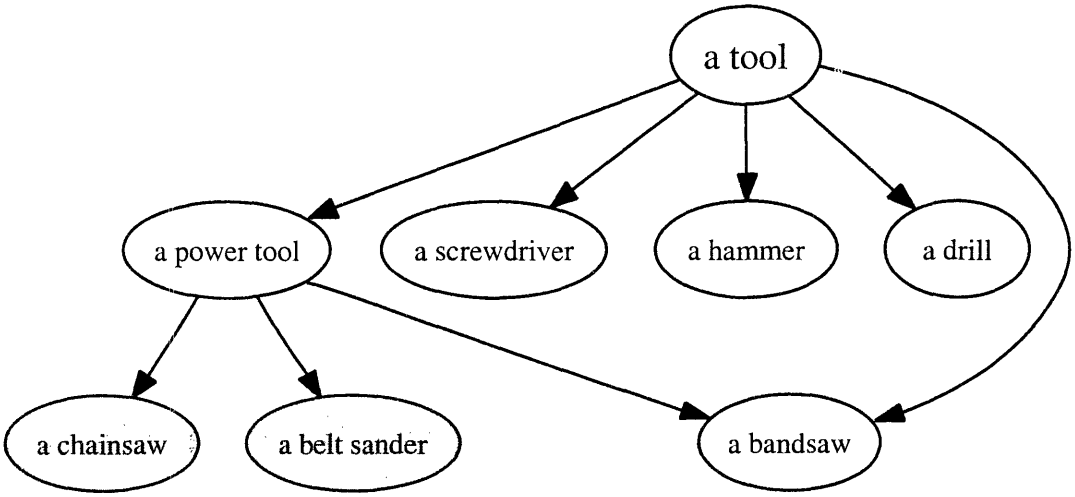


Figure 5-1: Graph of *Hyponym* tree for *a tool* (some Hyponyms omitted for clarity)

*lathe.* If a true value is passed for the keyword :plain in the call, a flat list of *Things* is returned, with no direct/indirect information.

```
MECHNET> (get-all-hyponyms (get-thing "a tool"))
=> ((#<T=151 a power tool> . direct) (#<T=31 a punch> . direct)
    (#<T=89 pliers> . direct) (#<T=77 a wrench> . direct)
    (#<T=83 cable cutters> . direct) (#<T=34 a bandsaw> . direct)
    (#<T=66 a screwdriver> . direct) (#<T=64 a hammer> . direct)
    (#<T=27 a drill> . direct) (#<T=117 a chainsaw> . indirect)
    (#<T=30 a belt sander> . indirect))
```

In this case all the Hyponyms have direct links to *a tool* except *a chainsaw* and *a belt sander*, which have indirect links via *a power tool* (though this is not evident from the return value). *A bandsaw* also has a link through *a power tool.* However, since it also has direct link to *a tool*, this link takes preference (see Fig. 5-1. Indirectly linked hyponyms are in grey).

There are several more helper functions, and these all use **get-all-hypernyms** and **get-all-hyponyms**. These functions are useful, in that they replace the kind of reasoning that might be found in a rule based system: mainly, various kinds of transitivity. For example, if *Thing A* is a *Hypernym* of *Thing B* and *Thing B* is a

Table 5.1: Second-tier helper functions

| **Function** | **Direct** | **Indirect** |
|---|---|---|
| (**get-all-parts** X) | Parts of X | Parts of (get-all-hypernyms X) |
| (**get-all-containers** X) | *Things* of which X is a Part | *Things* of which (get-all-hyponyms X) are Parts |
| (**get-all-abilities** X) | *Ability* links whose Subject is X | *Ability* links whose Subjects are (get-all-hypernyms X) |
| (**get-all-actors** X) | *Ability* links whose Object is X | *Ability* links whose Objects are (get-all-hypernyms X) |

*Hypernym* of *Thing C*, then *Thing A* is a *Hypernym* of *Thing C*. Table 5.1 shows these functions, and how they use **get-all-hypernyms** and **get-all-hyponyms** in retrieving *Things*.

## 5.2 Analogies

There are two kinds of analogies performed, as were discussed in previous sections: *Ability* analogies and *Composition* analogies. In either case, the general procedure is as follows, starting with a *Thing*:

1. Find all the links of the appropriate type which connect to this *Thing*, using the helper functions described above.
2. Follow them to the other end (Subject <-> Object for *Ability*, Whole <-> Part for *Composition*)
3. For each *Thing* found in the last step, follow its links in the opposite direction. For *Abilities*, only consider links which involve the same *Action*.
4. Rank the matches by frequency

Link traversal happens in both directions for each type of link, using the helper functions described in table 5.1. So, in step 2 of this procedure, *Ability* links are followed in both Subject -> Object and Object -> Subject directions, and *Composition*

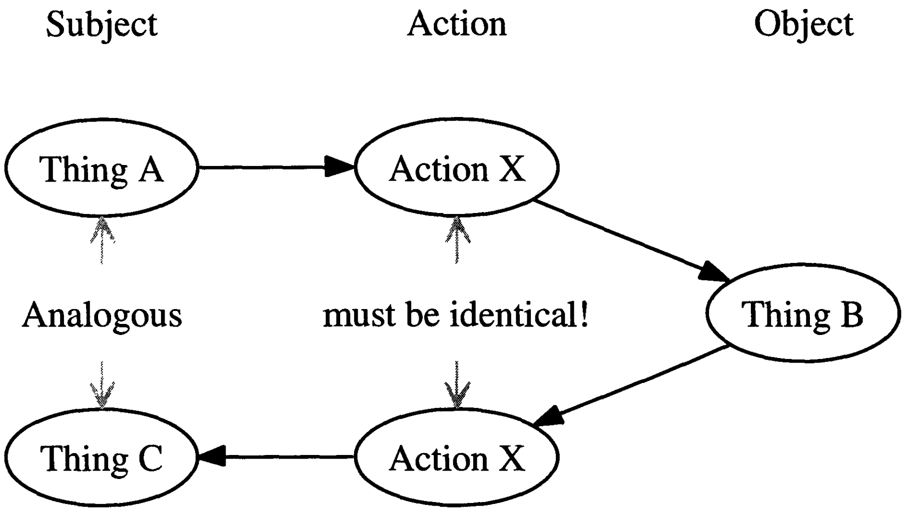


Figure 5-2: Link traversal for a single match of *Ability*-based analogy

links are followed in both Whole -> Part and Part -> Whole directions (See sections 6.1.1 and 6.1.2 for possible problems with this approach). Figure 5-2 illustrates how an analogous *Thing* is found starting out in the Subject -> Object direction. Note that the *Ability* link is stylized as a Subject-Action-Object chain for simplicity.

The *Composition* analogy is simpler, as there is no *Action* link to consider (Fig 5-3). As in the *Ability* diagram, the actual *Composition* link is not shown. This closely resembles how the links are followed using the data cache, but not the database structure.

These diagrams show how a single *Thing* is found to be analogous to the original *Thing*. For each analogy, this process is repeated many times to create the list of results.

When viewed from a high level, the analogy engine is actually very simple, as it consists of basic link traversal with some filtering of the results. However, it can be extended to encompass more complex data. The method of link traversal itself is not essential to achieve the desired result of finding analogies for a particular data structure. However, it does work well for the data struture used in Mechnet, because the algorithm naturally focuses on only those links that might be relevant to the result.

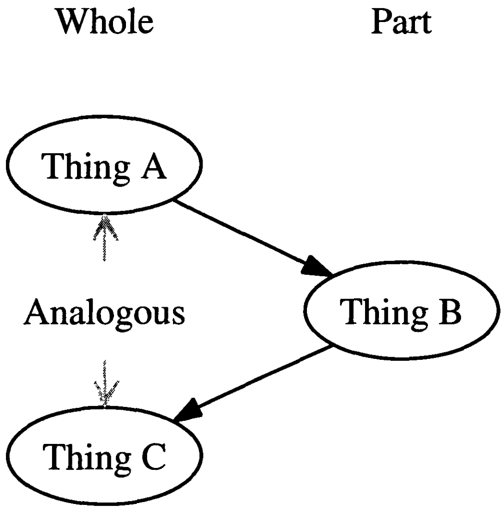


Figure 5-3: Link traversal for a single match of *Composition*-based analogy

# Chapter 6

# Results & Discussion

## 6.1 Selected Data

The data shown below are an example selected from the larger available data set. Unfortunately, the nature of the data is such that any analysis must take place on a case-by-case basis. Since time did not permit a quantitative study of the accuracy (or "sense") of the analogies by humans, the data shown in tables 6.1.1 and 6.1.2 is given instead as a general sampling.

Table 6.1: Analogies based on *Ability*

| *Thing* | **Similar *Thing*s by Use/*Ability*** |
|---|---|
| **a belt sander** | **sandpaper** |
| **a chainsaw** | **a blade, circular saw, a metal cutting blade, a hacksaw blade, a cutting tool** and **a saw** |
| **wood** | **a material, aluminum, steel** and **a metal** |
| **a collet** | **an end mill, a cutting tool** and **a chuck** |
| **a reamer** | **a blade, circular saw, a metal cutting blade, a hacksaw blade, a chainsaw, a drill bit, a center drill, an edge finder** and **a cutting tool** |
| **a motor** | **a wrench** and **a lathe** |

### 6.1.1 *Ability* analogies

In table 6.1.1, the results which seem to make the most sense are the analogies for *a belt sander* and *wood.* Obviously a belt sander and sandpaper have similar functions, to sand something. The analogies for wood (aluminum, steel, and metal) can in many places replace wood as a building material. Listing *A material* is unnecessary, as wood is a material. It is listed because *Things* such as *a mill* can be used to cut both *wood* and *a material.*

As for *a chainsaw*, its analogies are can all be used to cut something, and so make sense superficially. However, any kind of *cutting tool* operates on a much smaller scale than a chainsaw, and so this analogy seems rather silly, as one would basically never be able to substitute one for the other.

The last three *Things*' analogies make the least sense. *A collet* is in no way similar to *an end mill.* This analogy was generated from the dubious data that *a mill* can be used to turn *a collet*, as well as to turn *an end mill.* While technically true, this does not make a collet similar to an end mill. It would seem that while being the Object of common Abilities implies similarity for *Things* like metal and wood, this is not generally true. *A reamer* suffers the same problem: both it and *an edge finder* can be held by a chuck. Also, the same scale problem occurs with *a chainsaw*, where the two *Things* can both be used to cut, but not in the same way.

Finally, Mechnet thinks *a motor* is similar to *a wrench* and *a lathe.* Both *a motor* and *a wrench* can exert torque on something, and both *a motor* and *a lathe* can be used to turn something. These are superficial similarities, which are essentially outweighed in the human mind by differences. It is possible that with much more data in the database, these similarities would be outweighed analogies to other devices, for example, *a solenoid* (which currently does not exist).

### 6.1.2 *Composition* analogies

For the *Composition*-based analogies, there are successes as well as problems, similar to the *Ability*-based analogies. For an example of a sensible analogy, Table 6.1.2 shows

Table 6.2: Analogies based on *Composition*

| *Thing* | **Similar** *Things* **by** *Composition* |
|---|---|
| **a bandsaw** | **a machine tool**, **a drill**, **a chainsaw**, **a belt sander**, **a mill**, **a saw** and **a machine** |
| **a hacksaw** | **a drill** and **a hammer** |
| **a motor** | **a chuck**, **a handle**, **an on/off switch**, **a blade**, **a belt** and **a power source** |
| **a magnet** | **windings** |
| **a chuck** | **a motor** and **a handle** |

*a bandsaw* and *a belt sander* being similar. These machines have a similar form, each containing a motor, rotating belt, etc. This is also true to a lesser extent with *a drill*, *a mill* and other analogous *Things*.

Next, *a drill* and *a hammer* are proposed as analogies for *a hacksaw*. The only similarity they really share is that the all have a *a handle*. The problem, like with the aforementioned *Ability* analogies of *a motor*, is that there is only one common link between the *Things* in question. This kind of problem is definitely dependent on the database size, and how well-connected the *Things* in it are.

The final three *Things*, *a motor*, *a magnet*, and *a chuck*, are also not very similar to their respective analogies. From these examples, it becomes evident that using an object's *Parts* to judge similarity is more reliable than using its location, or *Things* of which it is a *Part*. The same is true of *Ability* Subjects vs. Objects, in that *Things* which can do common *Actions* are more similar than *Things* which can be acted upon in the same way. For example, one can hit a nail or a crystal wine glass with a hammer, but this does not make the two similar.

# Chapter 7

# Conclusion

Overall, the system performed as expected: the analogies based on both the use and composition of *Things* generally "made sense", in that it was relatively easy to understand the "thought process" of Mechnet. There were also several cases where the analogies did not make sense. It is these cases that are most important, because they are what need to be avoided if the system is to be improved.

One problem is that for both *Ability* and *Composition* analogies, when the links were followed in one direction the analogies made sense, but going in the other direction produced dubious results. For *Ability* analogies, comparing objects that can do certain things was useful, but not objects that can have similar *Actions* performed upon them by the same *Thing* (with some exceptions: metal and other materials, for instance). For *Composition* analogies, basing the analogy on Parts yielded good results, but not comparing *Things* that are a Part of the same *Thing*.

A peculiarity of this problem is that it does not occur when one is dealing with materials like *wood* or *metal* being the object of *Abilities.* This is logical, however, because the fact that these materials are all grouped under what we refer to as "materials" is what makes them similar. The percieved similarity in this case is not related to their being able to be acted upon in the same way. After all, one can use a drill to make holes in almost any household item; specific materials are not special in this regard. What they have in common is that they can be used to construct things (though there was no data relating to this *Ability*).

In sum, finding analogies should be based only on following links in the direction that worked well for either type: Subjects for *Abilities*, and Parts for *Compositions*. Even with the relatively small data set, it is evident what types worked and didn't, and the algorithms would work better if they were modified to reflect this.

## 7.1 Future Recommendations

The Mechnet system was kept simple and small in order to test the analogy algorithms. Accordingly, there are many ways in which it could be made more scalable, expanded to represent more knowledge, and generally improved.

### 7.1.1 Cache

The memory cache of *Thing*s presents a scalability problem. Implementing a partial cache would most likely not be very beneficial, due to the random-access nature of the analogy algorithms.

Instead of using a cache, the cached link-retrieval functions could be replaced with large SQL joins, but this adds complexity and maintenance difficulty. The ideal solution would be LISP macros that create the joins automatically from LISP code. This has been achieved for some Mechnet functions that are somewhat simpler than the larger link traversal functions, and should be feasible for them as well.

### 7.1.2 Data Quality Improvement

If Mechnet is to be open to many users, there are some improvements that might increase the ratio of good data to noise. One would be to allow users to give individual data (links) an accuracy score, which would then be propogated through the system and used to evaluate the accuracy of analogies.

Having users register before entering data allows user input to be tracked, so that it is easy to isolate sources of consistently good / bad data.

### 7.1.3 "Processes"

In the current Mechnet system, there is no way to represent a process. If an *Ability* represents "what" a *Thing* can do, a *Process* would define "how" it can do it. A process would define the steps one takes to fulfill an *Action*, but even more, it would define what the start and end conditions are. A pseudo-code example of a *Process* is:

- Start: flat surface
- Drill hole
- Ream hole
- Deburr hole
- End: surface containing clean hole

This type of data structure can be used to define processes that have many more steps than are shown here. These *Process*es could even be used to explain something like how to operate a mill or lathe, by defining the steps required to accomplish a specific task. Most humans know what they want for a result; the problem usually lies in how to get there. Defining the start and end properties for a *Process* could allow the system to discern the difference between the two states. Then different processes can be compared, and this shows what the individual steps do. For example, one could leave out "Ream hole" and "Deburr hole" in the example above, and change the end state to "surface containing rough hole". The system can then reason that somehow reaming and deburring change a rough hole into a clean one.

This type of reasoning comes from measuring what Marvin Minsky, in "Society of Mind" calls "differences between differences." He proposes that "The ability to consider differences between differences is important because it lies at the heart of our abilities to solve new problems"[15].

Implementing *Process*es would probably be more complicated than the reasoning described in this paper, but it could be one more step toward human-like intelligence in machines.

# Appendix A

# Source Code

The source code for Mechnet is available for download from the following URL: `http://cadlab.mit.edu/~wallace/mechnet`. Installation instructions are available there.

# Appendix B

# Database in Human-Readable Format

Table B.1: *Ability* Data

| | **Subject** | | **Action** | **Object** |
|---|---|---|---|---|
| You can use | a bandsaw | to | cut | a metal |
| You can use | a bar | to | attach | a blade |
| You can use | a belt | to | turn | a pulley |
| You can use | a belt sander | to | smooth | wood |
| You can use | a belt sander | to | smooth | a metal |
| You can use | a blade | to | cut | something |
| You can use | a bolt | to | fasten | something |
| You can use | a center drill | to | make a guide hole in | something |
| You can use | a chainsaw | to | cut | something |
| You can use | a chainsaw | to | cut | wood |
| You can use | a chainsaw | to | have a really good time or | something |
| You can use | a charger | to | charge | batteries |
| You can use | a chuck | to | hold | a drill bit |
| You can use | a chuck | to | hold | a reamer |

Continued on next page

**Table B.1 – continued from previous page**

| | Subject | | Action | Object |
|---|---|---|---|---|
| You can use | a chuck | to | hold | a center drill |
| You can use | a chuck | to | hold | an edge finder |
| You can use | a collet | to | hold | an end mill |
| You can use | a collet | to | hold | a fly cutter |
| You can use | a collet | to | hold | a drill bit |
| You can use | a cutting tool | to | cut | something |
| You can use | a cutting tool | to | remove material from | something |
| You can use | a drill | to | make holes in | something |
| You can use | a drill | to | make holes in | a flat surface |
| You can use | a drill bit | to | make holes in | aluminum |
| You can use | a drill press | to | make holes in | something |
| You can use | a fastener | to | attach | something |
| You can use | a file | to | remove burrs from | a metal |
| You can use | a fly cutter | to | face | a flat surface |
| You can use | a forstner bit | to | make a large hole in | wood |
| You can use | a funnel | to | filter | a liquid |
| You can use | a grinder | to | grind | something |
| You can use | a hacksaw | to | cut | a metal |
| You can use | a hacksaw blade | to | cut | something |
| You can use | a hammer | to | hit | something |
| You can use | a hammer | to | hammer | a nail |
| You can use | a laser cutting machine | to | cut | titanium |
| You can use | a lathe | to | turn down | a shaft |
| You can use | a lathe | to | face | something |
| You can use | a lathe | to | spin | something |
| You can use | a lathe | to | cut | a shaft |
| You can use | a lathe | to | turn | a shaft |

Continued on next page

**Table B.1 – continued from previous page**

| | Subject | | Action | Object |
|---|---|---|---|---|
| You can use | a lathe | to | hold | a cutting tool |
| You can use | a lathe | to | face | the end of a shaft |
| You can use | a lathe | to | turn down | a curved surface |
| You can use | a masonry nail | to | attach | brick |
| You can use | a masonry nail | to | attach | masonry |
| You can use | a metal cutting blade | to | cut | something |
| You can use | a metal cutting blade | to | cut | a metal |
| You can use | a mill | to | turn | an end mill |
| You can use | a mill | to | make holes in | something |
| You can use | a mill | to | hold | a collet |
| You can use | a mill | to | hold | a chuck |
| You can use | a mill | to | turn | a collet |
| You can use | a mill | to | turn | a chuck |
| You can use | a mill | to | mill | wood |
| You can use | a mill | to | mill | aluminum |
| You can use | a mill | to | mill | steel |
| You can use | a mill | to | hold | a cutting tool |
| You can use | a mill | to | remove material from | a material |
| You can use | a motor | to | turn | something |
| You can use | a motor | to | apply torque to | something |
| You can use | a motor | to | spin | something |
| You can use | a motor | to | power | something |
| You can use | a pipe clamp | to | clamp | something |
| You can use | a polisher | to | polish | something |
| You can use | a pump | to | pressurize | a fluid |
| You can use | a punch | to | make holes in | sheet metal |
| You can use | a reamer | to | clean up | a hole |
| You can use | a saw | to | cut | a material |

Continued on next page

**Table B.1 – continued from previous page**

| | Subject | | Action | Object |
|---|---|---|---|---|
| You can use | a screw | to | fasten | something |
| You can use | a screw | to | fasten | wood |
| You can use | a screwdriver | to | fasten | a screw |
| You can use | a shear | to | cut | sheet metal |
| You can use | a spray gun | to | spray | something |
| You can use | a staple | to | hold down | a wire |
| You can use | a table | to | support | something |
| You can use | a tack | to | attach | carpet |
| You can use | a tack | to | attach | a fabric |
| You can use | a transistor | to | amplify | current |
| You can use | a vise | to | grip | something |
| You can use | a waterjet cutting machine | to | cut | titanium |
| You can use | a wire | to | provide power to | something |
| You can use | a wire | to | hang | something |
| You can use | a wire | to | fasten | something |
| You can use | a wrench | to | apply torque to | something |
| You can use | a wrench | to | turn | a nut |
| You can use | an awl | to | plane | something |
| You can use | an end mill | to | mill | something |
| You can use | an end mill | to | make a slot in | something |
| You can use | an iron bar | to | smash | something |
| You can use | band clamp | to | clamp | something |
| You can use | c-clamps | to | clamp | something |
| You can use | cable cutters | to | cut | cables |
| You can use | calipers | to | measure | something |
| You can use | circular saw | to | cut | something |
| You can use | engraver | to | make grooves in | something |
| You can use | goggles | to | protect | your eyes |

Continued on next page

**Table B.1 – continued from previous page**

| | Subject | | Action | Object |
|---|---|---|---|---|
| You can use | pliers | to | grip | something |
| You can use | sandpaper | to | smooth | wood |
| You can use | scissors | to | cut | paper |
| You can use | scissors | to | cut | soft materials |
| You can use | scissors | to | cut | a wire |

Table B.2: *Hypernym* Data

| Hyponym | | Hypernym |
|---|---|---|
| a bandsaw | is | a saw |
| a bandsaw | is | a machine |
| a bandsaw | is | a tool |
| a belt | is | a tension-transmitting device |
| a belt sander | is | a power tool |
| a bolt | is | a fastener |
| a center drill | is | a drill bit |
| a chainsaw | is | a saw |
| a chainsaw | is | a power tool |
| a curved surface | is | surfaces |
| a drill | is | a tool |
| a drill press | is | a machine |
| a fly cutter | is | a cutting tool |
| a forstner bit | is | a drill bit |
| a hacksaw blade | is | a blade |
| a hammer | is | a tool |
| a hammer handle | is | a handle |
| a lathe | is | a machine tool |
| a machine tool | is | a machine |
| a masonry nail | is | a nail |

Continued on next page

Table B.2 – continued from previous page

| Hyponym | | Hypernym |
|---|---|---|
| a metal | is | a material |
| a mill | is | a machine |
| a mill | is | a machine tool |
| a nail | is | a fastener |
| a pipe clamp | is | a clamp |
| a power tool | is | a tool |
| a pulley | is | a simple machine |
| a pump | is | a machine |
| a punch | is | a tool |
| a reamer | is | a cutting tool |
| a rivet | is | a fastener |
| a screw | is | a fastener |
| a screwdriver | is | a tool |
| a shaft | is | a power-transmitting structure |
| a staple | is | a fastener |
| a table | is | a piece of furniture |
| a tack | is | a nail |
| a wrench | is | a tool |
| aluminum | is | a metal |
| an iron bar | is | a bar |
| band clamp | is | a clamp |
| batteries | are | a power source |
| brass | is | a metal |
| cable cutters | are | a tool |
| calipers | are | a measuring device |
| car | is | a vehicle |
| copper | is | a metal |
| electricity | is | a form of power |
| goggles | are | a protective device |

Continued on next page

**Table B.2 – continued from previous page**

| **Hyponym** | | **Hypernym** |
|---|---|---|
| heat | is | a form of energy |
| kinetic energy | is | a form of energy |
| lead | is | a metal |
| magnesium | is | a metal |
| pliers | are | a tool |
| sandpaper | is | rough |
| soft materials | are | a material |
| steel | is | a metal |
| steel | is | a metal |
| threads | are | a simple machine |
| titanium | is | a metal |
| wood | is | a material |
| wood or metal | is | a material |
| zinc | is | a metal |

Table B.3: *Composition* Data

| **Whole** | | **Part** |
|---|---|---|
| a bandsaw | has or contains | a blade |
| a bandsaw | has or contains | a motor |
| a bandsaw | has or contains | an on/off switch |
| a belt sander | has or contains | a motor |
| a belt sander | has or contains | a belt |
| a bolt | has or contains | threads |
| a bolt | has or contains | a head |
| a chainsaw | has or contains | a motor |
| a drill | has or contains | a chuck |
| a drill | has or contains | a motor |
| a drill | has or contains | a handle |

Continued on next page

**Table B.3 – continued from previous page**

| Whole | | Part |
|---|---|---|
| a hacksaw | has or contains | a handle |
| a hammer | has or contains | a handle |
| a hammer handle | has or contains | wood or metal |
| a machine | has or contains | a motor |
| a machine | has or contains | a power source |
| a machine tool | has or contains | an on/off switch |
| a mill | has or contains | a belt |
| a mill | has or contains | a motor |
| a motor | has or contains | windings |
| a motor | has or contains | a magnet |
| a nail | has or contains | a head |
| a saw | has or contains | a blade |
| a saw blade | has or contains | steel |
| a screw | has or contains | threads |
| a screw | has or contains | a head |
| a threaded rod | has or contains | threads |

# Bibliography

[1] Singh, P., Lin, T., Mueller, E. T., Lim, G., Perkins, T., and Zhu, W. L. "Open mind common sense: Knowledge acquisition from the general public". *In Robert Meersman, Zahir Tari (Eds.), Lecture Notes in Computer Science: Vol. 2519. On the Move to Meaningful Internet Systems 2002: DOA/CoopIS/ODBASE 2002 (pp. 1223-1237)*, Aug. See also `http://commonsense.media.mit.edu/cgi-bin/search.cgi` (Unknown date), 5/4/2005.

[2] Gunnarsson, J., McKelvey, T., and Helmersson, A., 1999. Matrix algebra package for mathematica. Research paper, Linkoeping University, S-581 83 Linkoeping, Sweden, Department of Electrical Engineering, S-581 83 Linkoeping, Sweden, Mar.

[3] Palmer, D. D., 1997. "A trainable rule-based algorithm for word segmentation". In Proc. 35th Annual ACL, ACL, Association for Computational Linguistics, pp. 321–328.

[4] Cycorp, 2002. *Ontological Engineer's Handbook Version 0.7.* Cycorp, Inc., Suite 100, 3721 Executive Center Drive, Austin, TX 78731, Aug.

[5] Hofstadter, D., 2005. Analogy as the core of cognition. Talk at Bartos Theatre, MIT Media Lab (E15): Tuesday, April 12, 2005, 2:00 PM EST, Apr.

[6] Cao, Q., Senin, N., and Wallace, D. R., 2005. Semantic classification of model services in an internet-based distributed modeling environment. Research paper,

Under review by the 2005 ASME Design Engineering Technical Conferences, May.

[7] DOME (Distributed Object-based Modeling Environment), research project at MIT CADLAB (2003). See `http://cadlab.mit.edu/research-dome/`, 5/2/2005.

[8] MacLachlan, R., 1991. CMU Common Lisp user's manual. Research paper CMU-CS-91-108, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, Feb. This is a revised version of Technical Report CMU-CS-87-156. See also `http://www.cons.org/cmucl/`, 2/5/2005.

[9] MySql AB. MySQL Database Server Version 14.7 Distrib 4.1.10a, for pc-linux-gnu (i686) (Unknown date). See `http://www.mysql.com/`. 5/2/2005.

[10] Rosenberg, K. M. CLSQL - A multi-platform SQL interface for Common Lisp, 8/3/2004. See `http://clsql.b9.com`. 5/2/2005.

[11] Pilgrim, M., 2004. *Dive Into Python.* (Free Online Text), May. See `www.diveintopython.org`, 5/2/2005. Section 17.6 contains pluralization regular expressions for English nouns.

[12] Microsoft. Natural Language Processing Research Group Overview (Unknown date). See `http://research.microsoft.com/nlp/`. 5/2/2005.

[13] Reed, S. L., and Lenat, D. B., 2002. "Mapping ontologies into cyc.". In Proc. AAAI 2002 Conference Workshop on Ontologies For The Semantic Web, Cycorp, Inc., AAAI. See also `http://www.cyc.com/cyc/technology/pubs`.

[14] Google. Google Suggest Beta (Unknown date) - Real time display of search query suggestions. See `http://www.google.com/webhp?complete=1&hl=en`. 5/4/2005.

[15] Minsky, M., 1986. *The Society of Mind.* Simon & Schuster, New York, section 23.1, p. 238.