

# EventMinder: A Personal Calendar Assistant That Understands Events

by

Dustin Arthur Smith

Bachelor of Science in Computer Science  
Wake Forest University, 2005

Submitted to the Program in Media Arts and Sciences,  
School of Architecture and Planning,  
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2007

© Massachusetts Institute of Technology 2007. All rights reserved.

Author \_\_\_\_\_  
Program in Media Arts and Sciences  
September, 2007

Certified by \_\_\_\_\_  
Henry Lieberman  
Research Scientist  
Program in Media Arts and Sciences  
Thesis Supervisor

Accepted by \_\_\_\_\_  
Prof. Deb Roy  
Chairperson, Departmental Committee on Graduate Students  
Program in Media Arts and Sciences



# EventMinder: A Personal Calendar Assistant That Understands Events

by

Dustin Arthur Smith

Submitted to the Program in Media Arts and Sciences,  
School of Architecture and Planning,  
on September, 2007, in partial fulfillment of the  
requirements for the degree of  
Master of Science in Media Arts and Sciences

## Abstract

Calendar applications do not understand calendar entries. This limitation prevents them from offering the range of assistant that can be provided by a human personal assistant. Understanding calendar entries is a difficult problem because it involves integrating many types of knowledge: commonsense knowledge, about common events and the the particular instances in the world, and user knowledge about the individual's preferences and goals.

In this thesis, I present two models of event understanding: ROMULUS and JULIUS. ROMULUS addresses the problem of how missing information in a calendar entry can be filled in by having an event structure, goal knowledge, and past examples. This system is able to learn by observing the user, and constrains its inductive hypothesis by using knowledge about common goals specific to the event. Although this model is capable of representing some tasks, its structural assumptions limit the range of events that it can represent.

JULIUS treats the event understanding problem as a plan retrieval problem, and draws from the COMET plan library of 295 everyday plans. These plans are represented as a set of English activity phrases ("buy a cup of coffee"), and so the planning problem is equated to the natural language understanding problem of comprehending events. I show two techniques for retrieving these plans: the first matches plans by their generalized predicate-argument structure, and the second retrieves plans by their goals. Goals are inferred by matching the plans against a database of 662 plans, by computing the conceptual similarity between the goals and components of the plan. From the initial analysis, certain VERBNET argument roles were more useful for retrieving the correct goals than others: INSTRUMENT and STIMULUS, were the two most influential.

Combining the strengths of ROMULUS and JULIUS, I create a prototype of a personal assistant application, EVENTMINDER, that is able to recognize users' goals in order to propose relevant alternatives and provide useful recommendations.

Thesis Supervisor: Henry Lieberman  
Research Scientist  
Program in Media Arts and Sciences

# EventMinder: A Personal Calendar Assistant That Understands Events

Dustin Arthur Smith

The following people served as readers for this thesis:

Thesis Reader \_\_\_\_\_  
Marvin Minsky  
Professor of Media Arts and Sciences  
Professor of Electrical Engineering and Computer Science  
Program in Media Arts and Sciences

Thesis Reader \_\_\_\_\_  
Erik T. Mueller  
Research Staff Member  
IBM Thomas J. Watson Research Center



# Contents

<b>Abstract</b>	<b>3</b>
<b>1 Introduction</b>	<b>11</b>
1.1 An Automated Personal Assistant for Event Management . . . . .	11
1.2 A User Scenario . . . . .	12
1.3 Romulus and Julius: Two approaches to the problem . . . . .	17
<b>2 Romulus</b>	<b>19</b>
2.1 Extracting details from the user's calendar entry . . . . .	20
2.2 Mapping slot values to components of a generic event model . . . . .	21
2.2.1 The problems of representing an event or plan . . . . .	21
2.2.2 A slot-filler object representation of generic events . . . . .	23
2.2.3 Filling in missing slot values . . . . .	25
2.2.4 Application: A Mixed-Initiative User Interface . . . . .	31
2.3 Related Work in User Interfaces . . . . .	31
2.4 Assessing Romulus . . . . .	32
2.4.1 Structural Limitations . . . . .	32
2.4.2 Assertional Limitations . . . . .	32
<b>3 Julius</b>	<b>35</b>
3.1 Framing the problems . . . . .	36
3.1.1 Representing Plans in English . . . . .	37
3.2 Retrieving Plans by Generalized Predicate-Argument Structure . . . . .	38
3.2.1 Background: The Lexical-Semantics of Events . . . . .	38
3.3 Retrieving Plans . . . . .	44
3.3.1 Retrieving Plans . . . . .	44
3.4 Finding alternative plans based on original goals . . . . .	45

<b>4</b>	<b>Inferring Goals from Plans</b>	<b>47</b>
4.1	Indexing and Retrieving Plans by Goals . . . . .	47
4.1.1	Parsing natural language descriptions of plans . . . . .	47
4.1.2	Representing Goals . . . . .	49
4.2	Computing Similarity between Goals and Plans . . . . .	52
4.2.1	Computing semantic similarity with SVD . . . . .	52
<b>5</b>	<b>Evaluation</b>	<b>55</b>
5.1	Creating a Metric . . . . .	55
5.1.1	Collecting a human-annotated gold standard . . . . .	55
5.2	The Results . . . . .	57
<b>6</b>	<b>Conclusion</b>	<b>59</b>
6.1	Joining Romulus and Julius . . . . .	59
6.2	Extending the Calendar . . . . .	60
6.2.1	Recognizing problems with the plan . . . . .	60
6.3	Future Directions . . . . .	61
6.3.1	Integration with a cellphone to “close the loop” . . . . .	61
<b>A</b>	<b>Supplementary Material</b>	<b>65</b>
A.1	The Need for Background Knowledge . . . . .	65
A.2	The Roles of Goals in Problem Solving . . . . .	66
A.3	The problem of underspecification . . . . .	67
<b>B</b>	<b>Accompanying Data</b>	<b>69</b>
B.1	OMCS Plan Set . . . . .	69
B.2	ETS Plan Set . . . . .	72
B.3	Example Goal Annotation . . . . .	73

# List of Figures

1-1	Free text entry into the application. . . . .	13
1-2	Filling in missing information . . . . .	13
1-3	Browsing by plan-level goals. . . . .	14
1-4	Retrieving locations from a goal-driven search . . . . .	14
1-5	Searching plans high-level goals . . . . .	15
1-6	Retrieving plans by searching high-level goals . . . . .	16
1-7	A list of local bars. . . . .	16
2-1	A fixed set of question answered by the event representation . . . . .	24
2-2	The set of slot names in the slot-filler object with their types and example values. . . . .	24
2-3	A dependency graph for knowledge in ROMULUS. . . . .	26
3-1	An example sentence with its semantic roles labeled according to FrameNet’s frame semantics. . . . .	39
3-2	An example of an annotation with FRAMENET. . . . .	41
3-3	An example of diathetical alternation, the “causative/inchoative alternation,” where in this transformation direct object becomes the subject of the transitive verb. . . . .	42
4-1	Representing goals . . . . .	49
5-1	Web-based survey for obtaining a gold corpus. . . . .	56
6-1	A list of events lacking commonsense. . . . .	60



# Chapter 1

## Introduction

### 1.1 An Automated Personal Assistant for Event Management

Consider the differences between a personal assistant who has been hired to manage your schedule and an electronic calendar. A human personal assistant proactively manages your agenda and offers many types of assistance, whereas modern calendar software has been designed to passively collect and display your event entries. The difference is that the human understands the event, and the calendar does not. My goal in this thesis is to build an automated personal assistant that can understand events in order to provide assistance the way a human personal assistant would.

Calendar software is useful because it can tell what a user plans to do at some future times. However, its usefulness is limited because of not including enough details about those events. For example, “lunch with Jim” does not tell us who Jim is nor where the event will take place.

What is required to understand events? A personal assistant brings general **Commonsense Knowledge** to the situation and understands, for example, that lunches take place around noon and typically at restaurants and delis. Also, over time, your assistant learns specific details about your preferences and habits, *i.e.*, that you prefer restaurants near the office during the week and that you dislike Mediterranean food. We’ll refer to this additional information as **User Knowledge**.

Many types of assistance can only be achieved by also understanding *why* the user is doing a certain task. In fact, as we will see in 2.2.3, this goal knowledge can be used to infer missing values from the user’s event. Here is an incomplete list of ways in which knowledge of the user’s goals can be used to solve problems (see Appendix A.2 for others):

**Goal-based categorization.** Suppose you mentioned that you are having lunch with your boss but you did not specify the location. The differences between choosing “restaurants” and not “kitchens” or “offices,” could come from the User Knowledge about the goals you are currently pursuing.

**Prioritizing Plans.** There is a conflict between two plans, which one is more important to you? Again your User Knowledge could help you compare those goals.

**Suggesting Alternative Plans.** If your dinner plan falls through, what are some other similar plans? This also depends on your goals, which in this example could be *satiating hunger, socializing, developing a trusting business relationship, ...*

Where would all of this knowledge come from? Most modern calendar programs have a standardized structure for representing events that is defined by the `iCalendar` standard<sup>1</sup> and include slots for the `ATTENDEES`, `LOCATION`, and `DURATION` of the events (but no `WHY`<sup>2</sup>). However, few users would ever have enough time to include enough specific additional details! If the calendar is to acquire this information to understand the plan, it must do so automatically—using both commonsense world knowledge and examples learned from observing the user.

## 1.2 A User Scenario

EVENTMINDER runs as a web application, and from the initial screen the user has the option of entering a new event or viewing the entire calendar at once. Here is an example interaction with EVENTMINDER, our prototype of a calendar program that understands events and its user’s goals:

---

<sup>1</sup> <http://www.ietf.org/rfc/rfc2445.txt>

<sup>2</sup> It does have a field called `PRIORITY`, which is related to goals.

A potential client, Chris, is visiting you next week, and you add “dinner with Chris next Friday at All Asia” to your calendar program.



Figure 1-1: The user adds a new event to the calendar.

EVENTMINDER parses your calendar entry and extracts key components; for example, it recognizes that *Chris* fills the slot *ATTENDEE* and *All Asia* fills the *DESTINATIONLOCATION* slot.

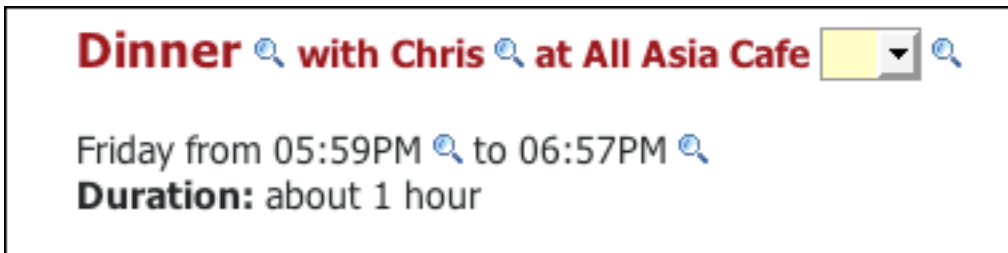


Figure 1-2: EVENTMINDER has filled in the missing information, including the times, and matched “All Asia” to the restaurant in its database.

EVENTMINDER fills in missing information about your plan, guessing a starting time, duration, where you are leaving from. It presents you with a list of nearby restaurants. The interface maintains its interactivity, allowing the user to override the suggestions. Any change will initiate a “cascade,” changing old values and causing new ones to be inferred.

While EVENTMINDER recognizes that dinners take place at restaurants, it represents you a list of restaurants and tries to infer your plan level goals. The user can search by specifying their goals: *e.g.*, “to hear music”, “drink” and “eat asian food.”



Figure 1-3: EVENTMINDER infers which goals are relevant based on the properties of the location you have selected, and allows the user to search for alternatives by specifying goals.

EVENTMINDER finds one location with that matches this unlikely combination of criteria: *Kapow!*

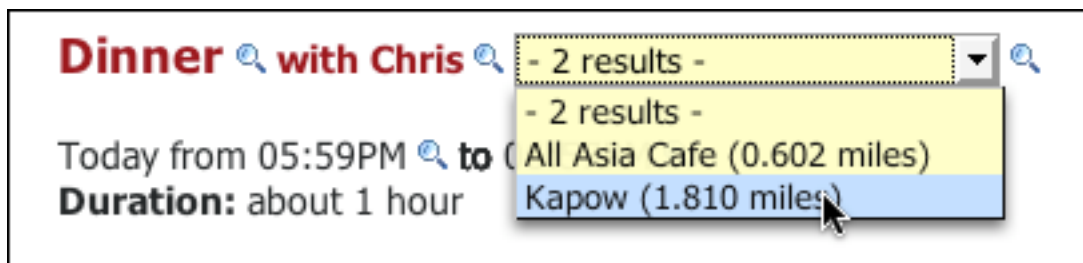


Figure 1-4: The goal-driven query returns a list of qualifying restaurants.

In addition to specifying the goals specific to your current plan (eating dinner), EVENTMINDER can guess the high-level goals that motivated your decision to have lunch in the first place. At the top of figure 1-5, the user selects the goals related to the event, and clicks **Search By Goals**.

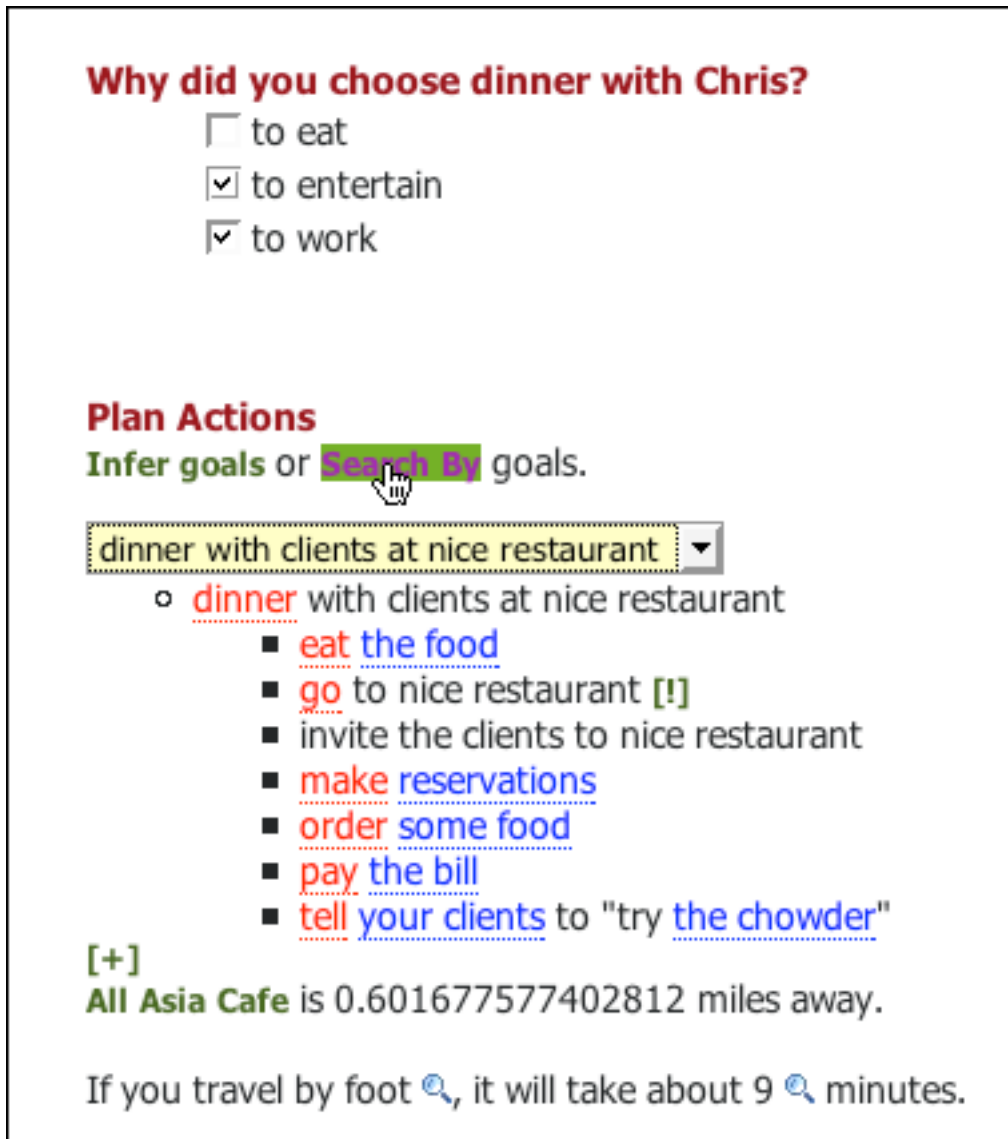


Figure 1-5: EVENTMINDER shows the possible high-level goals behind the plan “dinner with clients at nice restaurant,” the most similar plan to the “dinner with Chris next Friday at All Asia” and allows the user to search for alternative plans.

The result is a list of alternative plans related to the goal that the system has inferred automatically from the plan. The user selects “drinks with client:”

And EVENTMINDER, recognizing that this plan takes place at bars, displays a list of bars sorted by distance to your inferred origin: *The Media Lab*.

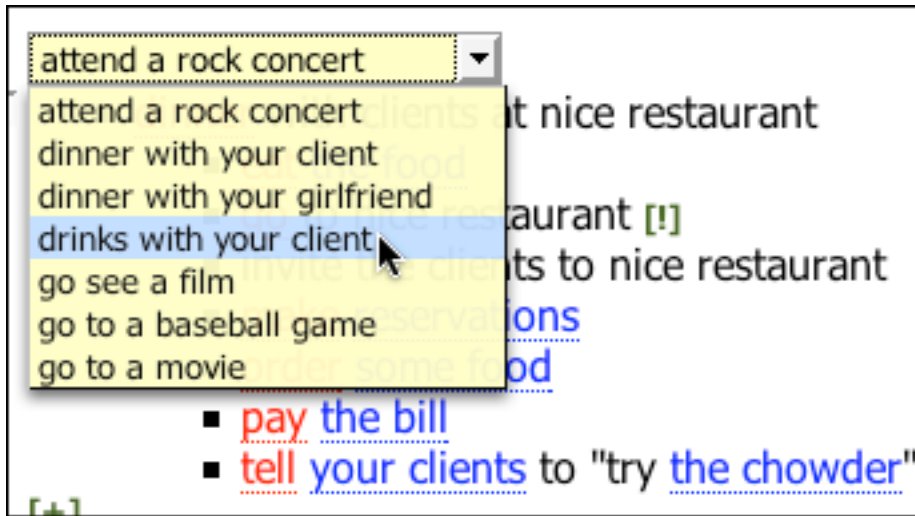


Figure 1-6: Plans that match the user's high-level goal descriptions.

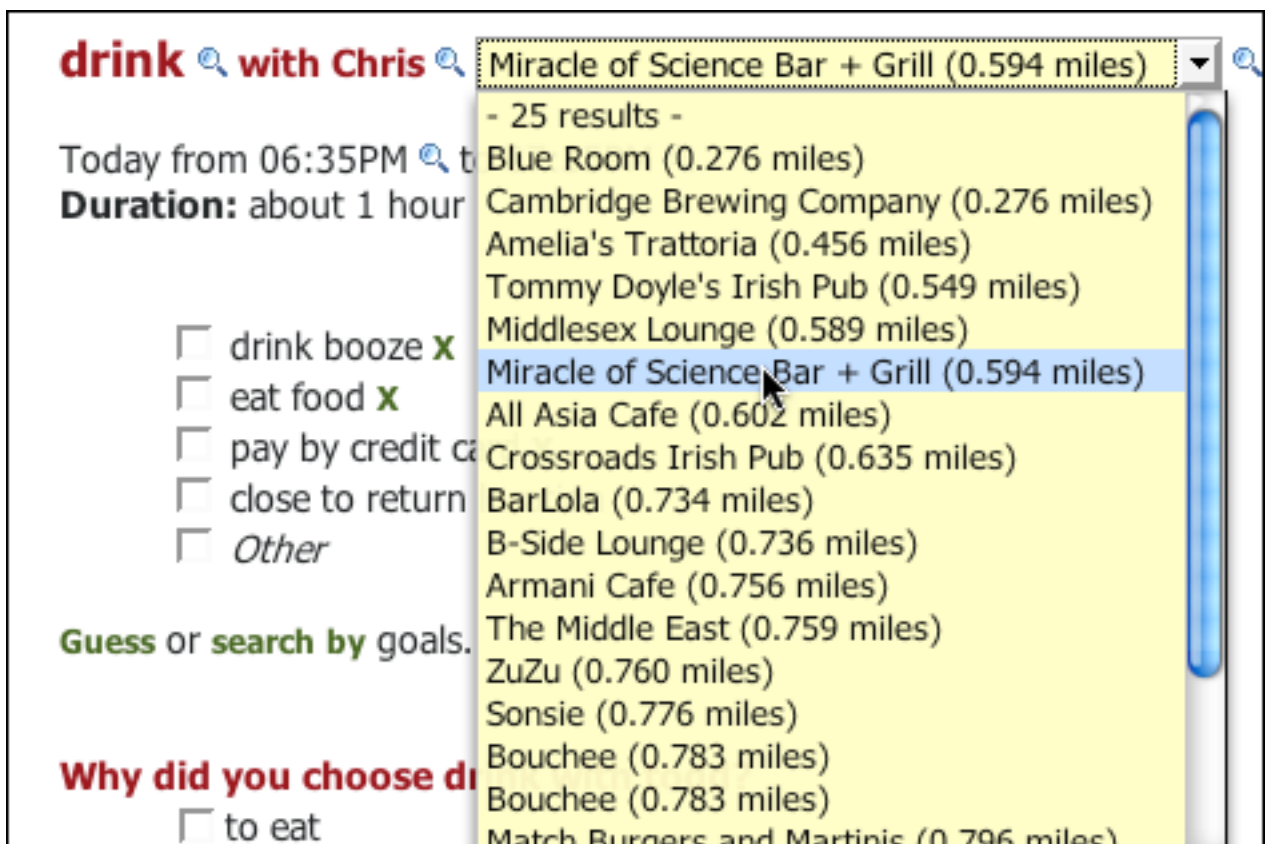


Figure 1-7: EVENTMINDER presents the user with a list of local bars.

### 1.3 Romulus and Julius: Two approaches to the problem

Before EVENTMINDER could help to provide general assistance to the user, there are two main problems that it had to solve:

1. Constructing a rich plan representation from the user's natural language calendar entry,
2. Inferring the goals of the plan.

In chapters 2 and 3, two different approaches lead to two different systems: ROMULUS and JULIUS (two predecessors to modern calendars). The main difference between them is their sources of knowledge; ROMULUS focuses on user knowledge and goal knowledge, while JULIUS focuses on commonsense and goal knowledge. ROMULUS solves problems related to very specific decisions, like finding out exactly which restaurant you will go to for lunch, but is limited to a narrow range of plans such as: dining out, having drinks, seeing concerts, and going to nightclubs. JULIUS on the other hand has a large range of possible events and is extensible by not having fixed assumptions about the plan's structure. The application EVENTMINDER is a combination of both of these models.

The first system, ROMULUS, assumes a generic structure that describes the components of an event and how they are related. The fundamental assumption is that missing values can be inferred from the structure of the event model and completed examples of past events. The system learns, by observation, the user's preferences using inductive learning techniques in order to infer the missing values and learn goal-based representations of categories. Goals are represented as predicates that reference specific features of the target concepts; thus background knowledge about goals is used to constrain the inductive task in accordance with the *explanation-based learning* approach. In other words, the "concept learning" task is transformed into the "goal learning task," and the user is permitted to explore suggestions by goals instead of categories. ROMULUS uses specific knowledge of local restaurants, bars and nightclubs, and thus was limited to the set of events that involved these types of locations.

In chapter 3 the second system, JULIUS, is described. JULIUS's plan knowledge is broader and covers a larger range of common activities. It uses a library of 295 plans represented in English, many of which were collected from volunteers, which

can be easily extended. This background knowledge is used to infer the goals using a corpus-based approach, in order to suggest alternative plans that cover a much wider-range of problems. JULIUS, however, does not have detailed knowledge about specific locations, so it can not solve the same fine grained problems as ROMULUS.

In chapter 4, JULIUS's technique of inferring goals from this natural language plan corpus is explained in detail and evaluated, making inroads for some of the problems using English to represent plans and inferring goals from free text.

Both models are not mutually exclusive. While ROMULUS excels at learning the user's preferences and how they are associated with specific plan-level goals, JULIUS has knowledge about a larger range of goals. In chapter 5, I discuss how these models are combined to produce EVENTMINDER, and then draw general conclusions about this research effort.

## Chapter 2

# Romulus

This chapter addresses the language problem of underspecification in event descriptions, and draws from linguistics and machine learning. The problem arises when important details which are often left out of verbal expressions and is pervasive in human communication, including event descriptions. This chapter shows how events can be represented as slot-filler objects where the structure of the event and background knowledge is used to infer missing values. In addition, this chapter highlights the importance of goals in category learning.

In the next sections, each of the components of ROMULUS is explained along with a more general framing of the problems the component was built to solve. After the user creates a new calendar entry, such as “lunch with Gina”, the system engages and does the following:

1. Extracts slot values from the user’s calendar entry (Section 2.1)
2. Uses the slot values to fill in components of a generic event model (Section 2.3)
3. Infers missing values in the event model using examples from user’s event history (Section 2.4)

ROMULUS uses a generic representation of an event and uses both inductive and deductive inference to fill in missing slot values. The structural assumptions in ROMULUS prohibit the model from representing all types of events, and this problem lead to the more general, case-based approach of JULIUS (Chapter 3).

## 2.1 Extracting details from the user’s calendar entry

The first task is to get as much useful information from the user’s calendar entry as possible. A user may describe a lunch with a client, Larry, in several ways, varying the amount specified detail:

- Lunch
- Lunch with Larry
- Lunch with Larry today at noon
- Lunch with Larry today at MC Squared from 12 to 1

Given the user’s input, the system must extract the components from the calendar entry and recognize that, for example, “MC Squared” is the `LOCATION`.

The solution for this general problem is **semantic role labeling**, which involves extracting the predicate (“have lunch”) and arguments (*e.g.*, “Larry”, “Today”, “noon”) from the sentence and assigning each a semantic role (`ATTENDEE`, `DATE` and `TIME`). The specific labels of the semantic roles come from the underlying lexical-semantic theory, and are usually a function of the predicate.

Although automated semantic role labeling has been recently made possible by advances in corpus-based computational linguistics [13] and the availability of semantic role lexicons [17, 23, 16], the labeling process is too computationally expensive for this step as it would require loading large probability tables into memory at the start of the application. Later, in section 3.1, JULIUS uses semantic role labeling to preprocess a large batch of data offline.

Instead, a rule-based technique is used to parse the calendar entry and makes certain assumptions about the input. It anticipates that the calendar entry begins with an activity phrase and that the rest of the sentences contains a fixed set of other possible arguments (including: `ATTENDEES`, `DESTINATIONLOCATION`, `STARTDATE` and `ENDDATE`), delineated by slot name-specific prepositions (*e.g.*, “with”, “from”, “at”). The dates and times are extracted using a remote call to the “quick add” component of Google’s Calendar API<sup>1</sup> that correctly parses a large variety of date expressions into starting and ending date/times.

---

<sup>1</sup> <http://www.google.com/support/calendar/bin/answer.py?hl=en&answer=36604>

A critic may caution that the text from users' calendar entries will be too cryptic to be understood by a machine. This is indeed a real problem, but I believe it will be irrelevant as the capabilities for offering assistance through automation improve. When the only assistance the calendar provides is a memory aid, an abbreviated description is sufficient; however, when the system provides useful services through understanding the event, the user will likely be inclined elaborate their calendar entries.

## 2.2 Mapping slot values to components of a generic event model

Now that the slot values from the user's event entry have been extracted and labeled, representing those parts in the event model is trivial. The difficult problem is handling **underspecification**, a problem that arises from the fact that people assume shared knowledge and do not specify it in their communications (see A.3 for a detailed example).

The missing knowledge must be filled in; however, before doing so we must recognize which knowledge is missing. Missing knowledge is specified by the event model, which is a slot-filler object that was designed to answer a set of questions about common events. The event model makes assumptions about the structure of the typical event, including: what slot names are used, the values they can take on, and how they are interrelated.

In this section:

- 2.2.1 explains the general problem of representing an event or plan
- 2.2.2 explains the representational assumptions of how ROMULUS represents events
- 2.2.3 explains how missing values are filled in using different sources of knowledge

### 2.2.1 The problems of representing an event or plan

“Questions arise from a point of view—from something that helps to structure what is problematical, what is worth asking, and what constitutes an

answer (or progress). It is not that the view determines reality, only what we accept from reality and how we structure it. I am realist enough to believe that in the long run reality gets its own chance to accept or reject our various views.”—Alan Newell in *Artificial Intelligence and the Concept of Mind* (from [27]).

Suppose you are in some situation  $S$ —*i.e.*, you are in downtown Boston around noon and you have various active goals, such as to satiate hunger, read a Masters’ thesis and go back home. In the classical planning tradition, your goals could be represented as a set of situations  $S'$  in which each goal has been met (also known as the *goal state*) [15]. Accordingly, a valid plan is a sequence of *actions* that can be taken to change the situation from  $S$  to  $S'$ . Thus the plans you construct will contain a sequence of actions that you believe will lead to effects that ultimately meet your goals.

If you have already solved the problem before, the solution is as simple as *retrieving* and *reusing* the old plan. Because the *exact* situation never happens twice, old plans should be generalized before they match new goal and situation states. Consider the following scenario:

*Imagine that you have just learned how to fish and have caught your first catfish off a pier in Galveston, Texas. Now, you are given a chance to fish for salmon on a boat off southern Alaska.*

How much of the catfish-fishing episode can you transfer to the new situation? Clearly there is some generalization involved in human problem solving, but this leads us to specific questions common to case-based reasoning like:

- What components of the plan are fixed and which are not?
- In what other situations is the plan similar enough to be retrieved? Does the plan still match if you are fishing at night or eating a catfish stew? Can the “fishing for catfish” script be re-applied in a different situation, like “soliciting campaign donations” where the assertions differ but some relations remain the same?
- How are plans re-used so that important differences are not abstracted and unessential distinctions can be viewed as the same? How do the mechanisms that transfer CATFISH→SALMON and BOAT→PIER avoid mistakes like BAIT→CAPTAIN?

Cognitive scientists have developed the notion of *schemas* [34], a computationally

androgynous concept that connotes structures of mental events. AI theories have lead to representations called frames [27] and scripts [35], which describe sequential structures that are re-used to solve common problems, by containing *slot names* which are filled by specific types of values or other frames and can have default values. For example, the “eat at restaurant” script would have slot names RESTAURANT and TABLE which would bind the slot names to the specific instances from the current situation.

When should two problems share the same script and when should they be separate plans? There seems to be a key trade-off between the number of schemas/plans/scripts and the complexity and roles of the slot names. Should the “eat at restaurant” script be broken into more specific sub-scripts: “eat at fancy restaurant”, “eat a buffet”, “eat fast food,” or should it be a part of an extremely general script “event” that contains upper-ontological slot names like LOCATION, TIME and ACTIVITY?

In other words, at what level of detail should we represent events? To this, we turn to psychology (the human mind is the best planning software to date) where one theory [46] posits that people maintain the most general level of description sufficient for *the current tasks at hand*; and, when the model ceases predict accurately, one breaks the event/plan representation into a set of smaller units. To concretize this theory: if two plans/events share the same set of questions in the context of a problem being solved, then they should belong to the same script.

Hardly satisfying, this theory replaces the original question with another (that is nearly the same!): what tasks are we trying to solve? With the goal of creating an automated personal assistant for event management, we can define a fixed set of tasks we would like to answer related to this problem.

### 2.2.2 A slot-filler object representation of generic events

In the traditional commonsense reasoning approach [7], the problem of designing a representation is based around the types of questions the application aims to solve. When the event pertains to “eating lunch”, the model should answer many of the questions in figure 2-1.

An alternative to assuming a fixed representation is the incremental approach, where the model gradually expands from a simple representation to answer new ques-

Where am I going to eat? Where have I gone in the past in similar situations? Are there any new restaurants I would like to try? With whom am I going? What is their relationship to me? Why am I going? When should I leave? From where should I leave? How far away is it? How long will it take to get there? Can I take the subway? What will I do afterwards? How long do these events usually take? Where am I going afterwards? How long will it take to get from there to the next place, and how should I travel?

Figure 2-1: The fixed set of questions ROMULUS is designed to answer.

Slot Name	Value Type	Three Examples
ACTIVITY	Nominal	lunch, lunch, dinner
DESTINATIONLOCATION	Location†	Legal Seafood, Home, Stefani’s Pizza
STARTINGLOCATION	Location†	Home, Home, The Media Lab
DISTANCE	Real (miles)	0.2, 0.0, 2.4
TRAVELMETHOD	Nominal	foot, foot, bus
TRAVELTIME	Real (minutes)	6, 0, 18
STARTDATETIME	DateTime	12:30, 2:00, 18:00

Figure 2-2: Example slot names and values from ROMULUS’s event representation. †Location includes both Attribute-Values and Nominals, and for the three example values, only the attribute ‘Name’ is listed.

tions (the way people are capable of doing). With this power comes great responsibility. It requires a sophisticated architecture that is capable of learning from many different types of problems, as there are many ways in which problems could occur. An example of such an architecture is the *Emotion Machine* [29, 38], where *reflective critics* detect general classes of problems [37], such as “missing causal dependencies” and “invalid knowledge structures” and engage specific learning mechanisms to fix them.

Limiting ROMULUS to this set of questions, a slot-filler object was built accordingly. Each of these questions can be answered by looking up slot values, which includes the concepts in figure 2-2.

The event model specifies what slot names are used, the values they can take on, and how they are interrelated. For example, the slot STARTINGLOCATION specifies where you originate, DESTINATIONLOCATION specifies where you are going, and DISTANCE specifies the distance between the two locations. The locations are represented as a set of attribute-value pairs (*Name*=Denny’s; *Latitude*=42.365023; *Longitude*=-71.096969...) and a set of nominal values to specify their properties (“fastfood, open late,...”). DISTANCE is represented as a real number. The structure of the event dic-

tates that `DISTANCE` is function of the latitude and longitude of two locations.

When specifying the values of the slots, values can be represented as nominal values (properties), real numbers, date/times, and feature-lists (set of nominal values). Each data type can take on a certain range of values. Complex values, like `Locations` and `People`, require knowledge of particular instances. For locations, this is achieved by using a database of restaurants, nightclubs and bars in the Boston area from Citysearch<sup>2</sup>, which supplies details about commercial locations.

Although the assumptions that went into building this event model do not extend to all kinds of events (see 2.3), this model has some merits. In particular, it can fill in missing values by using a combination of inductive learning techniques and deductive inference.

### 2.2.3 Filling in missing slot values

Once the user input has been parsed, the next task is to fill in the missing information about the event.

Even when the user has specified the value, it needs to be represented the correct way before it can fill the slot. In the cases of the locations and people, `EVENTMINDER` must handle the lexical-semantic (word to representation) mapping of labels to sub-frames. When the calendar entry includes “MC Squared”, this specifies a corresponding `Location` object (sub-frame) that must be retrieved from background knowledge to fill in the `DESTINATIONLOCATION` slot. This retrieval process depends on the slot name (`DESTINATIONLOCATION` in this case) in order to find the right sub-frame for the label (“MC Squared” could be the name of a person—perhaps a hip-hop musician)<sup>3</sup>.

What happens when the slot value is missing? For example, when the user forgets to mention the starting time of a given “lunch”?

Missing values are inferred by looking at previous completed event objects from the user’s history. Each missing slot value can be presented as an inductive inference

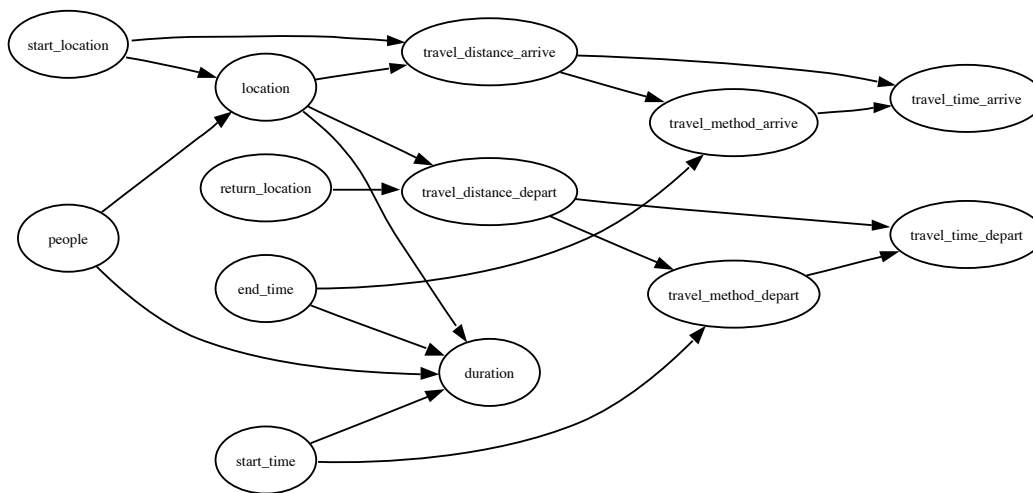
---

<sup>2</sup> <http://boston.citysearch.com>

<sup>3</sup> The types of semantic labels, in turn, depend on the type of slot-filler object and that is specified by the predicate (“have lunch”); for `ROMULUS`, however, only one type of slot-filler object is available: the default event model.

problem: by looking at its dependent slots from the event model and past examples of those values. There are relationships and constraints *between* the various slots of the event model, for example, when the DESTINATIONLOCATION is changed, it affects the DISTANCE, which may influence the TRAVELMETHOD and consequently the TRAVELTIME.

These influential relationships are roughly related to the temporally successive progression of events within the event model (*i.e.*, causation or correlation). These can be represented as *directed acyclic graph*, where an edge from  $a \rightarrow b$  implies that  $a$  “has some influence upon”  $b$ . Knowing one value (type of Location) can help guess the other (type of attendee).



Dining Dependency Graph

Figure 2-3: The dependencies between the slot values of ROMULUS’s event model. For example, the category of people “client” would occupy the value of the **people** node and would be used to infer the category “fancy restaurants” for the **location** node.

How do we find missing values? If the event model contains all of the knowledge dependencies between the instances of knowledge, we can fill in missing values with enough prior examples of specific dining events (and if it is not complete, we can still approximate them). We can break the graph into subgraphs where each bipartite pairing is a separate induction problem. The entire graph can be used to chain inference problems when values are also missing in their dependencies. The input-output pairings are a typical input for machine learning problems: given a series of input-output pairs,  $\langle x_1 \dots x_n, y \rangle_{1 \dots i}$ , approximate the function  $f(\langle x \rangle) \rightarrow y$ . How do we

learn the function  $f(\langle x \rangle) \rightarrow y$ ? That depends on the data type of the values in the function's range:

1. If the target node,  $y$ , is a **real value** (*e.g.*, distance, duration, speed), use **regression** to approximate its value.
2. If the target node is a **nominal value** (*e.g.*, method-of-travel), use **classification** to guess its value.
3. If the target node is a **set of nominal values** (*e.g.*, location, people), infer possible **goals** from the **category descriptions** to suggest alternatives.

Instance-based learning techniques are used for regression and classification, where the value is interpolated by looking at its nearest neighbors from previous examples. Learning locations or people presents a more challenging problem, because they can be represented at different abstraction levels. For example, do you have lunch at “a restaurant”, “a Mexican restaurant” or “Anna’s Taqueria”? Clearly the intermediate level descriptions would be more useful to the user, and that categories should be learned instead of particular instances. How do we move from a set of example instances to a category?

### **Conceptual clustering: A Naïve Approach**

In this model, the target locations (restaurants, bars, nightclubs, and other places specified by the user), are all represented as nominals: a set of attributes. Our goal is to learn a *concept description* that describes the *types* of locations that the user may want to visit. This requires the ability to generalize several instances into a category, which extends to cover more instances that were not in the examples.

**Clustering** is an unsupervised learning technique that can group example observations into a smaller number of groups. (If the category labels are available at the beginning of the task, the learning process is instead a **classification** task, where the goal is to approximate a function that places instances into predefined categories.) A common approach to clustering is to treat each item's attributes as dimensions and the items themselves as points in a multi-dimensional object description space, and then use an inter-object similarity metric, such as Euclidean distance, to uncover boundaries between item regions. A problem with most similarity metrics is that the

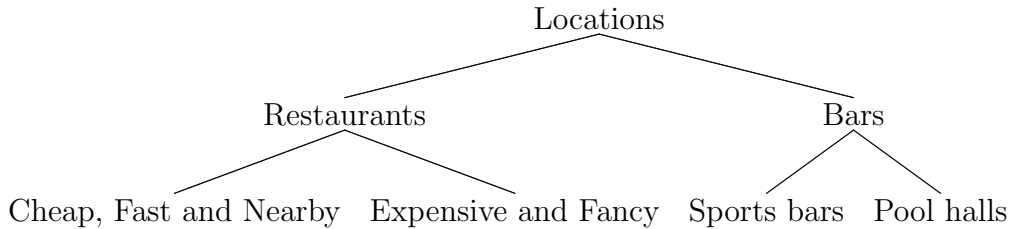
*descriptions* of the clusters are lost in the process, and are left to be retrieved by a post-clustering process or human, whose task is to ascribe meaning to the groups that the clustering process has accumulated. Without any description, the system cannot reflect and communicate about the groups it has learned, and must resort to examples or prototypes to communicate or assimilate new knowledge.

There is an approach to clustering, called **concept learning**, that also aims to group items into categories but also maintains a description of these categories [26]. A category with a description is referred to by this community as a **concept**. Concept learning is said to be an inductive inference technique, because of the symbolic description it extracts from its observations. This is a requirement of our approach, because we want to be able to search for alternative records using declarative statements (*e.g.*, queries in SQL) that describe the presence or absence of whole attributes.

In ROMULUS, for example, suppose you have observations of the last 20 places the user went for lunch. Our concept learning task is to induce a set of categories (types of restaurants) that partitions a set of discrete observations (examples of past restaurants).

To put each location on equal footing, the set of nominal values from each location is transformed into a binary feature vector  $\langle\{0, 1\}^n\rangle$  that has a length  $n$ , the number of unique nominal values in the data set, and where a value of 1 denotes the presence of the attribute and 0 its absence. The concept learning task is to generate a description of general categories to group the examples into. Valid descriptions are  $\{0, 1, ?\}^n$ , where ? means “don’t care” and matches both 1 and 0. Thus the extension of  $\{?\}^n$  is all of the concepts in the data set.

At the start of a clustering problem, one must first define how many clusters to form (or the similarity threshold at which to stop clustering). Alternatively, one could use hierarchical clustering to produce clusters at *all* levels of detail, producing a dendrogram. This is appropriate because the attributes of our locations are also described at varying abstractions. Ideally, we want our concept learning process to produce descriptions like this:



Unfortunately, there were many problems with this approach that make it infeasible with only a few examples. Instead of learning a “Cheap and Fast” category, we would get a description like:

$$\text{Cheap} \wedge \text{Visa} \wedge \text{Discover} \wedge \text{Delivery} \wedge \text{KidFriendly} \wedge (\text{Italian} \vee \text{Sushi})$$

This description draws from many irrelevant features, and would cause the EVENT-MINDER to overlook relevant locations in its recommendation (*i.e.*, by only showing restaurants that accepted Discover cards).

To address this issue, one could naïvely ignore all features that have low **information content**, where information content is defined by the inverse of the times that feature appeared in the data set. This would introduce real-values to our cluster descriptions, which violates the aforementioned commitment to preserving a binary feature vectors as the concept description.

Before we risk perverting this data further, what is really going on with this problem? The problem is that of **feature selection**; in some circumstances the features may be relevant to the category, while in others they are not. If you were planning a romantic dinner, you would not want to be recommended a restaurant on the basis of the fact that they accept a certain credit card. In another situation, such as taking clients out to dinner, that same credit card feature may be salient because you are charging the bill on your company’s card. The problem of selecting the appropriate features changes depends on the active goals of the user and details of the situation.

In other words, just like the adjectives “nearby” and “heavy,” which are always relative to something else, features of slot names should be described relationally because they are only relevant in certain contexts. Some cognitive scientists have advocated similar perspectives [25, 33] and recommended abandoning the experimental

paradigm that led to “slot names in a vacuum” tasks, like classifying and clustering arbitrary items into groups. Listen to Eleanor Rosch, the founder of prototype theory, comment on this problem [33]:

“No matter how abstract and universal a concept may appear to be, that concept actually occurs only in specific, concrete situations. Real situations are information rich complete events... Context effects tend to be studied in psychological research only as negative factors, artifacts that invalidate somebody’s experiment or theory. But it may be that contexts or situations are the unit that categorization research really needs to study.”—Eleanor Rosch, 1999

In order to suggest relevant features, it helps to know the user’s goals. Users cannot be treated as static profiles, because their intentions (the goals they are actively pursuing) change frequently. This problem will arise in rich domains like event modeling where a number of different goals are involved.

### **Explanation-based category learning with goal knowledge**

One step towards relational-features involves learning the goals for selecting the properties rather than the sets of features (categories) themselves. This way, categories can be dynamically constructed by the user selecting goals, which combine sets of properties to produce category descriptions.

A similar idea was proposed in [43] by Stepp and Michalski who suggested that goals could be used to constrain the feature selection in clustering. In order to do so, we need to consider the relationship between properties and goals. To achieve this, we create a mapping from goals to logical combinations of properties, such as:

$$Be\_healthy \rightarrow Seafood \vee Vegetarian \vee Health\ food$$

$$Avoid\_eating\_meat \rightarrow Vegetarian \vee \neg Meat$$

And additional background knowledge specifying the inheritance structures:

$$IsA(Sushi, Seafood)$$

### *IsA(Seafood, Meat)*

This way we could infer the user’s goals from specific restaurants they have visited. Constraints between goals can be used to reduce the hypothesis space in inductive learning (*e.g.*, concept learning). For example, if the user visits both a sushi restaurant and a vegetarian restaurant, the system could deduce that their goal is to *Be\_healthy*, not *Avoid\_eating\_meat*. This would allow the system to suggest other healthy restaurants, including those which serve meat.

This approach would be classified as explanation-based learning [9], a genre of machine learning where the hypothesis space of the inductive task is constrained to concepts that can be deduced from some domain theory of background knowledge.

#### **2.2.4 Application: A Mixed-Initiative User Interface**

Now that we have some of the details specified from the user, and inferred some of the missing values, the user must be able to override these suggestions. This is presented as a mixed-initiative interface [14] where the system offers suggestions and details (when it can) and the user selects from those suggestions or provides their own decisions wherever they want.

### **2.3 Related Work in User Interfaces**

Many calendar based personal assistants have been proposed but few have addressed the full complexity of the issue. The best of these approaches are able to learn by observing the user [30], but still do not comprehend the user’s goals or provide knowledge of particular instances, such as nearby locations. In addition, these learning agents typically rely depend on a fixed model, as ROMULUS did, and thus cannot change their behavior and assistance to the specific task.

EVENTMINDER contributes the ability to browse by *goals* instead of categories, a feature which bolsters the learning process (see section 2.2.3) and has been found to be preferred by users. In a study comparing a goal-based interface for browsing television programs with a traditional history-based recommendation system [36],

viewers preferred the goal-based system, especially when they could browse by goals explicitly (TV shows were presented and categorized by goal). Although this program proved useful at matching users with their goals, there was a larger learning curve over the preference based system, presumably (the experimenters speculated) because it forced people to articulate their goals.

## 2.4 Assessing Romulus

In this section, I describe the contributions and problems of ROMULUS. I use Wood’s useful distinction between *structural* and *assertional* knowledge [45] to describe two categories of problems in ROMULUS. These terms were used to develop a distinction between the internal semantics of a knowledge representation structure (structural) and the particular claims that knowledge was asserting about the world (assertional). With this distinction, the two assertions *Kissed(John, Mary)* and *Action(Kiss, John, Mary)* are recognized as equivalent assertions but different structures.

### 2.4.1 Structural Limitations

One main problem with this model is that its commitment to a single, albeit general, plan limited its applicability to all of the kinds of events a person would typically describe in their calendar. Different kinds of plans implicate different problems and questions, and there does not exist one fixed set of questions for planning tasks.

As proposed earlier, two events should share the same plan if they involve the same set of questions. A problem with ROMULUS is that assumptions were made about a typical event’s structure and which problems related to all events were important to solve. It did not work for all types of plans: “Walk the dog” involves motion but no set destination; “take a vacation” extends many days and contains many sub-events; and “pay electric bill” does not involve a location.

### 2.4.2 Assertional Limitations

A lot of the capabilities of the system came from its ability to learn the user’s preferences. The system met the user halfway by providing an exhaustive list of example

values for locations (assuming their task involved restaurants, bars, or nightclubs) from which to choose, supplying a data point for learning the goals by observation to suggest alternative goal-based categories.

There are pros and cons associated with learning the user’s preferences. The main advantage is that the system is flexible and not committed to the assumptions in the background knowledge. If the user wants to eat lunch at 4:00PM every day, the system would detect and accommodate this behavior. Or, if the user had a new type of event that the system did not know of, it would learn this new class of events. The downsides are that the system is slow to accumulate this knowledge and requires several training examples to produce the appropriate inferences.

With enough examples the learning algorithm should converge on useful slot names; but, this is not feasible from the standpoint of the application. Users may be annoyed with an interface that had to learn slowly from their examples.

Several measures can be taken to speed up the learning process, for example, using background knowledge (such as goals, as presented earlier) to constrain the hypothesis space, and limit the relevant features. In the same fashion, knowledge from other components of the event could be used to cut down features. For example, if you are not bringing children along with you to the event, then it does not matter whether it is children-friendly or not. Additionally, **active learning** [3] can be used to suggest possible "near miss" examples [44] to the user (those near the generalization boundaries) in order to expedite the learning process.

Amassing the appropriate knowledge for this representation is difficult. While slot names were learned from instances (*e.g.*, of particular people, restaurants, tennis courts, etc), obtaining a suitable selection of instance knowledge, and their corresponding goals, was a challenge in its own—particularly when the system ambitiously seeks to be relevant to a wide range of events.

I was unable to derive this goal knowledge automatically from the OpenMind Commonsense (OMCS) corpus [40]. The knowledge I desired would map specific slot names, such as "sushi restaurants," to goals, such as "eat healthy." This is the type of knowledge OMCS typically harbors, but the restaurant category-goal domain was too specific that it could not be found in its over 700,000 statements. I suspect this would be remedied with increased contributions.



## Chapter 3

# Julius

“Experience is the teacher of all things.” —Julius Caesar

While ROMULUS tried to put too much into a single representation, JULIUS constructs its representation out of a case-library of pre-existing plans. This way it is more flexible, allowing different types of plans.

This model presents a new look at planning, where plans are represented in English, and events are denoted by predicate argument structure. Unifying these complementary perspectives on the same problem provides a solution for knowledge acquisition and representing events (with lexical-semantic structure). In this chapter, I address the specific issue of plan retrieval and show two techniques for retrieving appropriate plan from a library of 295 plans: the first matches plans by their generalized predicate-argument structure, and the second retrieves plans by their goals. Goals are inferred by matching the plans against a database of 662 plans, by computing the conceptual similarity between the goals and components from the plan.

JULIUS and offer a solution to ROMULUS’s main problem by extending domain-coverage by using a plan representation that is easy to acquire. The system does the following:

1. Parse the user’s calendar entry and retrieve a similar plan;
2. Infer the goals of that plan;
3. Retrieve alternative plans for the given goals;
4. Recognize and, when possible, execute actions in the plan.

The next chapter (4) describes the goal inference process in more detail.

### 3.1 Framing the problems

The planning problems addressed by JULIUS can be cast within case-based reasoning (CBR) framework. CBR systems involve a cycle of case retrieval, reuse, revision, and retention [24]. The problems faced by EVENTMINDER include **plan retrieval** from both natural language statements and goals, and **plan execution** when the system can recognize opportunities to execute actions.

One of the defining characteristics of CBR is that plans/cases can contain representations and knowledge that is specific to only the current case, so that each case can have its own representations and knowledge. In an unconstrained problem like event planning, it appeared necessary to partition the problem-solving space into separate parameterized plans, each with their own *problem type*, defined by the specific knowledge they involve. The knowledge at the plan-level (*i.e.*, the specific *kinds* of restaurants you go to for lunch) should be tailored to the individual user and learned from observation, while the commonsense mappings between *goals* and general *plans* (*i.e.*, that you eat a meal around noon, possibly at a restaurant) is culturally defined and thus accessible from a shared body of commonsense knowledge. This is reflected by the two complementary architectures ROMULUS and JULIUS.

This is consistent with the notion of hierarchical planning, where plans are constructed at varying granularities, solving the problem by first constructing a vague plan and then gradually filling in specific details. This is not done only for efficiency reasons; in many cases it is necessary to postpone the details until later in the plan's execution. For example, although a plan of dining at a restaurant may involve the event "sit down at your table," this sub-goal should be left vague until you know *which* table you will be sitting at—a piece of information that you will not know until you are at the restaurant [12].

### 3.1.1 Representing Plans in English

The plan library is comprised of a collection of plans in English sentences<sup>1</sup>. The decision to use natural language as the knowledge representation comes from many motivations: they are easy to author and can be collected from volunteers [40, 39, 2]; they are understandable by people; and, perhaps most importantly, the technical challenges presented by using natural language intersect with a number of research communities—providing a large foundation to build upon and a large group to benefit from contributions. This plan representation and its justifications are similar to the commonsense narratives use by Singh [38], which were ultimately to be constructed automatically from English statements.

The plans begin as a collection of short plans consisting of English activity prepositions (*e.g.*, “travel to the airport”, “have lunch with colleagues”), typically with an activity verb that can take an argument, a direct object or indirect object [19]. These were derived from two corpora, OMCS and, ETS, a smaller data set of common event plans that was crafted to complement the specific types of activities found in ROMULUS.

The activity phrases of each plan had two types of interrelationships to specify action orderings and part-of relationships. Analogous to objects having sub-parts arranged in space, plans have sub-plans arranged in time [47]. This is reflected in the assertions by the *PartOf*( $x, y$ ) relationship (where  $PartOf(x, y) \Leftrightarrow HasA(y, x)$ ). If a plan is not a part of any other plan, it is considered a “root plan.” All root plans have been decomposed into flat plan structures by expanding all of its  $HasA(x, *)$  until a flat list of sub-plans has been produced. Combining both OMCS and ETS, results in COMET<sup>2</sup>, a library of 295 plans with an average of 13 steps per plan. Examples from these plan sets can be found in Appendix B.

---

<sup>1</sup> All resources used in this thesis can be obtained from <http://web.media.mit.edu/~dustin/eventminder/>

<sup>2</sup>  $COMET \subseteq OMCS \cup ETS - OMCS \cap ETS$

## 3.2 Retrieving Plans by Generalized Predicate-Argument Structure

The goal here is to retrieve plans from an English assertion. This task is simplified because our plans are already represented in English; however, some generalization is still required.

I use background lexical-semantic knowledge from WORDNET and VERBNET to generalize the predicate-argument structure and retrieve plans. First, I present a survey of lexical-semantic resources and how they may be used to construct representations from English propositions, such as those in each step from each plan in COMET.

### 3.2.1 Background: The Lexical-Semantics of Events

Verbs play a large role in determining the semantic (and possibly syntactic) organization of a sentence; and, because verbs typically describe *changes*, they are associated with the semantics of actions and events<sup>3</sup>. The **predicate** of a sentence includes a verb phrase, that makes some true/false statement about the sentence’s subject and objects, called its **arguments**. So the sentence “John threw the ball to Bill” would have *throw* as the predicate. Similarly to the logical notion of predicate:

*throw(John, ball, Bill)*

**Predicate-argument extraction** involves identifying the predicates and their arguments of a sentence. **Semantic role labeling** is the task of 1) finding the frame for the sentence’s predicates, 2) for each slot type for each predicate, find the corresponding slot values (arguments) in the sentence. The type of frames and slots are specified by the underlying semantic theory. This task has recently been automated through advances in computational linguistics [13] and the availability of lexical-semantic lexicons [17, 23, 11, 16], some of which specify predicate-argument relationships for a lexical-semantic theory. Here, for example, is a sentence that has its slot names annotated according to FRAMENET:

---

<sup>3</sup> Actions  $\subset$  Events. Though similar, an action involves an actor, or doer; an event is a more general class of situational changes.

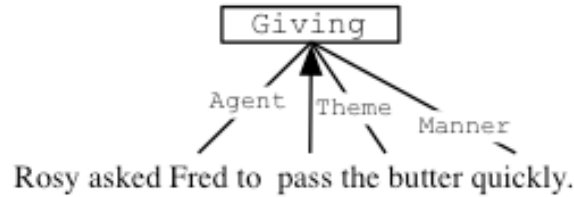


Figure 3-1: Annotating a short sentence with FRAMENET’s slot names. Image taken from Erk and Padó 2007 [10].

Semantic frames and their slot types are typically much more general than their corresponding predicates and arguments; and, there is no consensus on the types of slot names for each language (assuming that similar knowledge structures and word-concept mappings exist between same-language subjects in the first place). Competing theories are rampant, as Gildea et al [13] explain:

“At the specific end of the spectrum are domain-specific...or verb-specific [slot] roles such as EATER and EATEN for the verb *eat*. The opposite end of the spectrum consists of theories with only two “proto-roles” or “macro-roles”: PROTO-AGENT and PROTO-PATIENT (Van Valin 1993; Dowty 1991). In between lie many theories with approximately 10 roles, such as Fillmore’s (1971) list of nine: AGENT, EXPERIENCER, INSTRUMENT, OBJECT, SOURCE, GOAL, LOCATION, TIME, and PATH.”

To the computer scientist, slot names could be thought of as *data types* and the analogous relationship for *predicate-argument* is *frame-slot name*. To stay consistent in terminology, I will stay with the frame/slot terminology used earlier in the thesis. These semantic frames already correspond to events, so we can easily connect *event* to *plan*. Making this association, the question of “at what abstraction level should plans be represented?” from 2.2.2 is the same question as “at what abstraction level should semantic frames be represented?”

There are four English verb lexicons: VERBNET [17], FRAMENET [23], WORDNET [11] and PROPBANK [16]. Each of these has a different approach to verb classification, and thus each will have different ontological commitments [8] to the underlying event representation. Ultimately all resources, except FRAMENET, were used in JULIUS.

## WordNet

Fellbaum and Miller’s WORDNET [11] project is a machine-readable dictionary that categorizes words by part-of-speech and sub-divides them further into *synsets* (short for “synonym sets”), such that words and synsets have a many-to-many mapping. There are several denotational relationships between words (*e.g.*, antonyms) and synsets (*e.g.*, subsumers, a semantic taxonomy that supports conceptual inheritance). Unfortunately, WORDNET has the tendency to partition words into seemingly arbitrary synsets, even when they have similar meanings and may share underlying knowledge structures. For example, WORDNET makes a distinction between the word “head” as the top of something, the leader of an organization (and 30 other senses of the word’s noun form), even though they are conceptual related—problematically giving equivalent treatment to both disjoint *homonyms* and semantically overlapping word senses such as *polynyms* [20].<sup>4</sup> To get a flavor of its complexity, WORDNET has 44 different senses of the verb “give” and 11 senses of “book” nouns. WORDNET does not contain relationships between verbs and arguments; however, it has the largest coverage of the English language with 11,448 verbs as of version 3.0, with each verb having an average of 2.1 senses.

## FrameNet

The FRAMENET project [23] was constructed around Charles Fillmore’s linguistic theory of frame semantics. Frame semantics suggests that predicates and other components of speech reference prototypical frame situation representations. Here is an example sentence that has been annotated by FRAMENET using the SHALMANESER shallow-semantic parsing toolchain [10]:

As of this thesis’ publication, the authors of the FRAMENET project have defined 887 semantic frames, with 4,547 completed lexical units (*e.g.*, words) and twice that counting nearly-completed annotations of lexical units. FRAMENET is a linguistic and cognitive theory at once, forcing the annotators to make distinctions between similarities in meaning versus similarities in syntactic expression.

---

<sup>4</sup> The meaning of lexical units (*e.g.*, words) depend heavily on background knowledge and the context in which they originated. In fact, many lexical units have several distinct meanings: a property known as *homonymy* (*e.g.*, brown bear; to bear), or different but related meanings, a more common property, *polysemy* (*e.g.*, financial bank; I wouldn’t bank on it).

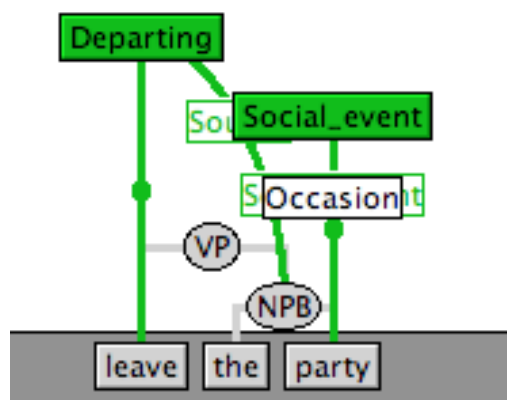


Figure 3-2: The statement “leave the party” after shallow semantic parsing into FRAME-NET’s frame semantics.

### PropBank

PROPBANK [16] is an extension of the one-million word Penn TREEBANK WSJ corpus where its verbs and their arguments have been hand annotated according to Levin’s diathetically alternated verb classes [18]. In 1993, Beth Levin categorized 3,104 verbs according to the ways they can be syntactically expressed. Verbs not only define the predicates that are involved with the sentence, but govern the generation of a valid sentence. Levin’s technique, known as **diathetical alternation**, involves transforming verb-argument examples into “meaning preserving” sentence templates, called *syntactic frames*, to contrast those which result in well-formed sentences from those that do not.

While FRAME-NET conflated syntactic and semantic similarity into the role of the frame, the practice of labeling verbs by their correct syntactic frames can be more clearly defined—at least when it is easy to recognize an ungrammatical sentence (donning the sentence with the conventional asterisk (\*) prefix). For example, the two verbs *hit* and *break*, despite having similar core arguments (AGENT, TARGET and INSTRUMENT) and meanings, can use different syntactic frames:

The fundamental underlying assumption to this approach is that patterns of syntactic expressions reflect the underlying semantic organization, and knowing the syntactic organization will help to understand the semantic. Levin compared the verbs using 79 alternations, which laid the foundation for clustering verbs by their asterisks to produce an arrangement of verb groupings. Levin clustered them into a two-tiered

1. (a) Joan **broke** the mailbox with her car.  
 (b) The mailbox **broke**.
2. (a) Joan **hit** the mailbox with her car.  
 (b) \*The mailbox **hit**.

Figure 3-3: An example of diathetical alternation, the “causative/inchoative alternation,” where in this transformation direct object becomes the subject of the transitive verb.

category structure: first, groups of verbs were formed with the same syntactic frames and then these were put into larger groups by their semantic similarity. For example:

<b>Verbs of change of possession</b>
GIVE VERBS: <i>feed, give, lease, lend, loan, pass, pay, peddle, refund, tender, rent...</i>
CONTRIBUTE VERBS: <i>administer, contribute, disburse, donate, extend, forfeit, proffer...</i>
<b>Verbs of removing</b>
REMOVE VERBS: <i>abstract, cull, delete, discharge, disgorge, dislodge, dismiss, disengage...</i>
BANISH VERBS: <i>banish, deport, evacuate, expel, extradite, recall, remove</i>

Table 3.1: Examples from Levin’s two-tier verb classification [18].

PROPBANK annotates the location of the verb’s arguments in a sentence and each argument’s general type; adapting an annotation scheme where arguments are consistent within each verb sense. This makes the corpus an ideal resource for training and evaluating computational linguistic tools [1]. Arguments are annotated ARG0, ARG1...ARGN for each verb sense, and a variety of adjunct tags (ARGM-TMP specifies the time, ARGM-LOC the location, ARGM-DIR the direction, & cetera) can be used to supplement any predicate. Apart from the adjuncts, the argument labels are inconsistent across verbs: ARG0 is 85% the agent, 7% the experiencer, 2% the theme, and so on [22], as PROPBANK remains neutral to any underlying semantic theory.

## VerbNet

Designed to improve upon WORDNET’s treatment of verb polysemy and some of the problems with Levin’s classes, VERBNET [17] was constructed as a corpus of cross-categorized Levin classes and annotated argument roles.

A problem with the 3,104 verbs that Levin classified is that some appeared to be members of multiple classes, resulting in 4,194 verb-class pairings [6]. This supports

the conclusion that there are separate syntactic and semantic frames at work (confirming an early speculation of Minsky [27]), so that verb-classes and verb-meanings may have a many-to-many relationship. The authors of VERBNET [17] dealt with these overlapping verb classes by introducing **intersective Levin classes** [6] permitting verbs to be cross-categorized (reference many semantic frames), where they can sometimes be disambiguated by the type of syntactic frame they appear within.

Hear Dang et al contrast VERBNET [6] to WORDNET:

“Whereas each WORDNET synset is hierarchically organized according to only one [implicit] aspect [chosen by the annotator], Levin recognizes that verbs in a class may share many different semantic features, without designating one as primary.”

VERBNET associates the arguments with one of 41 thematic roles, such as ACTOR, AGENT, INSTRUMENT, LOCATION, and provides a mapping from verbs to WORDNET synsets, although there are often many synsets for each verb class.

## A Comparison of FrameNet, PropBank and VerbNet

Each of these resources has their strengths and shortcomings. When given the sentence “lunch with Larry”, each of the lexicons provides different descriptions of the slot names for the various types of arguments. This is tricky, because the meaning of noun “lunch” implicitly refers to the predicates *eat(lunch<sub>FOOD</sub>)* or *have(lunch<sub>EVENT</sub>)*. Listed are the core arguments for the corresponding semantic frame for each resource:

**VerbNet** *Lunch* would be classified into verb class **dine-39.5** along with *banquet, breakfast, brunch, dine, feast, graze, luncheon, nosh, picnic, snack, sup*.

1. AGENT Something that is animate [Self,Larry]
2. PATIENT Something that is comestible [Lunch]

**FrameNet** *Lunch* belongs to the **Ingestion** frame along with *breakfast, consume, devour, dine, down, drink, at, feast, feed, gobble, gulp, gulp, guzzle, have, imbibe, ingest, lap, lunch, munch, nibble, nosh, nurse, put away, put back, quaff, sip, sip, slurp, slurp, snack, sup, swig, swig, swill, tuck*.

1. INGESTIBLES: entities being consumed by Ingestor [Lunch]
2. INGESTOR: The person eating, drinking or smoking [Self,Larry]

**PropBank** *Lunch* belongs to the **dine.02** roleset, describing the “dine\_out” predicate.

1. ARG0: An eater or diner [Self,Larry]

PROPBANK was the only resource that recognized that a sense of *lunch* denotes an event; the other two resources considered it to mean the activity of eating (*e.g.*, Larry *lunched* on some Vegemite.). This is not a serious mistake, because one activity of a lunch plan includes the activity of eating, and this could be thought of as a type of meronymy, where an event is described by one of its parts. Another example would be “eat a steak” for the plans “have a dinner at a steakhouse” or “cook and eat a steak for dinner.”

### 3.3 Retrieving Plans

How are plans retrieved to solve problems? This question was evoked earlier in 2.2.1 in the catfish example. In ROMULUS plan retrieval was not a problem: there was only one type of “plan” representation<sup>5</sup>, the slot-filler object, that was ever retrieved. The problems with this approach were evident; namely, the model was not flexible to apply to a wide range of situations.

If both models can coexist, why do they have different goal representations? The categories of goals are at different resolutions of detail: ROMULUS represents plan level goals that could be thought of as sub-goals from within a larger plan (*e.g.*, “save money”, “eat seafood”). JULIUS, on the other hand, represents much more general goals. In a full cognitive architecture, I imagine these would be integrated.

#### 3.3.1 Retrieving Plans

JULIUS uses two ways to retrieve plans: by words and by goals.

---

<sup>5</sup> I hesitate to label it as such, because it did not explicitly represent actions.

The first approach takes the user’s calendar entry, parses the semantic components and retrieves plans that match in those components. This is simplified because the plans are already expressed in natural language, but sometimes the predicates or arguments must be generalized. If a text match of the plan yields results, those plans should be used. However, if for example there are no plans that describe brunch in the corpus, “have brunch” should most similarly match plans for eating lunch or dinner. This can be done by looking up the predicate’s VERBNET role and matching plans which have that argument.

The predicate is sometimes not enough to match a plan. If no plans match the generalized predicate, then plans are retrieved by a search for the arguments. If multiple plans match the generalize predicate, the arguments can be used to filter the possible plans. If both match, we are faced with questions like: Is a lunch with a client more similar to a “dinner with client”, “lunch with friend” plan? Instead of attempting to answer this question, in these cases, I would present the user with a list of options.

The Given an input sentence, this is achieved by the following procedure that generalizes the predicate and arguments and searching, with a back-off technique:

1. Annotating the slot names of the sentence, yielding a verb,  $v$ , and arguments  $A = a_1 \dots a_n$
2. Search for plans that are described by verb  $v$  and arguments  $A$ .
3. If no plans are found, generalize the verb into its VERBNET verb class, and search for plans.
4. If no plans are found and  $|A| \geq 1$ , remove  $a_n$  from  $A$ , and search for plans that are described by verb  $v$  and arguments  $A$ .

This algorithm is first applied to root plans only, and, if no plans are found, this constraint is removed and plans can be retrieved by their parts.

### 3.4 Finding alternative plans based on original goals

In classical planning, it is common to retrieve plans by the sorts of goals they are capable of achieving. However, because our goals are represented in natural language, we must first infer the goals from the natural language statements in order to do this.

Finding alternative plans based on the original plan’s goals is a two-step process. First, goals must be inferred from the original plan. Secondly, a mechanism must retrieve other plans that match those same goals.

What kind of things are goals? Goals are the end states in which the problem has been solved. Goals can differ as to level of detail. For example, a plan to *dine\_out* may include the goals “save money, eat spicy food” and other types of goals that are related to the problem. On the other hand, more general goals like “eat” and “entertain” may be satisfied by other alternative plans.

In order to suggest alternative plans, we need a diverse library of common plans and a way to connect these to the goals. Using the COMET plan dataset, I was able to compute similarities using SVD.

This problem is the central focus of the thesis. The next two chapters (Chapter 4 and 5) are devoted to explaining and evaluating my approach.

## Chapter 4

# Inferring Goals from Plans

The previous chapters focused on the two approaches for inferring goals from plans. The first technique, from ROMULUS, operates on the plan level, and infers goals from example categories that the user has selected. JULIUS uses high-level goals from a large database of plans, using background general commonsense knowledge.

Both take different approaches to representing and inferring the goal knowledge. ROMULUS uses logical representations of goals and uses deduction for inference, while JULIUS takes a corpus-based approach and infers goals using statistical techniques. The main benefit of the deductive approach is accuracy, while the main benefit of the corpus-based technique is that it is easier to extend its coverage.

In this section, I explore the corpus-based approach for goal inference from plans in basic English. In section 4.1, I describe the nature of the data sets, in 4.2, the algorithm, and in 4.4, the evaluation.

### 4.1 Indexing and Retrieving Plans by Goals

#### 4.1.1 Parsing natural language descriptions of plans

Beginning with 295 plans in English, the goal is to match the plans against a large library of goals. The raw plans can be thought of as a set (some are in a sequence) of action-phases or sub-plans, each represented as an English proposition. The proposi-

<i>Raw Plan</i>	<i>Post-Parsed</i>
drink beer with a friend	[drink <sub>eat-39.1</sub> [beer with a friend]]
go out to a bar	[go <sub>escape-51.1-2</sub> [[out(? <sub>,AM-DIR</sub> ) [to a bar( <sub>Location,A4</sub> )]]]
buy a round of drinks	[buy <sub>get-13.5.1</sub> [a round of drinks( <sub>Theme,A1</sub> )]]
talk to you friend	[talk <sub>lecture-37.11-1</sub> [to you friend( <sub>Recipient,A2</sub> )]]
pay for your drink	[pay <sub>give-13.1-1</sub> [for your drink(? <sub>,A3</sub> )]]
get a taxi home	[get <sub>convert-26.6-2</sub> [a taxi home(? <sub>,A1</sub> )]]

Table 4.1: From Propositions to Semantic Roles: Example plan before and after semantic parsing. Predicates are classified into their VERBNET verb classes, and arguments are annotated with both VERBNET and PROPBANK labels (some only have PROPBANK labels).

tions themselves are no good (they are simply strings to the computer), and we need understand the richer event semantics behind each step.

I have automated the first stage of this process by using natural language processing tools to:

1. Identify the syntactic parts-of-speech for each sentence [TreeTagger]
2. Generate a parse tree for each sentence [Charniak Parser]
3. Extract the PROPBANK arguments for each predicate in each sentence [SwiRL, shallow semantic parser]
4. Map each predicate and semantic role to the VERBNET corpus [SemLink]

An example of a plan before and after this process is depicted in figure 4.1.

The result is a set of general predicate-argument structure, which is used to infer the goals from the plan, match the plans against English descriptions, and map each step to the system’s functions and parameters.

Of the 4,344 steps in the plan, there are 3,538 unique propositions (strings). Semantic role labeling reduced the number of predicates to 185 distinct VERBNET frames. Although frames could not be found for 417 (9.6%) of the steps. 8,255 arguments were found (a mean of 1.9 per predicate). Originating from the PROPBANK verb-specific argument labels, in the conversion to VERBNET semantic roles, 870 of the 8,255 (10.5%) were lost their labels. In total, the number of mislabeled (including unlabeled) arguments in the sentence is much greater; 10.5% does not account for mistakes that came from deriving PROPBANK’s predicate-argument structures from the original sentences.

### 4.1.2 Representing Goals

In the context of a calendar application, the goals I sought were at the level of detail immediately above plans. To help you understand this, take the example activity, “talk with clients.” There are a lot of possible associated goals.

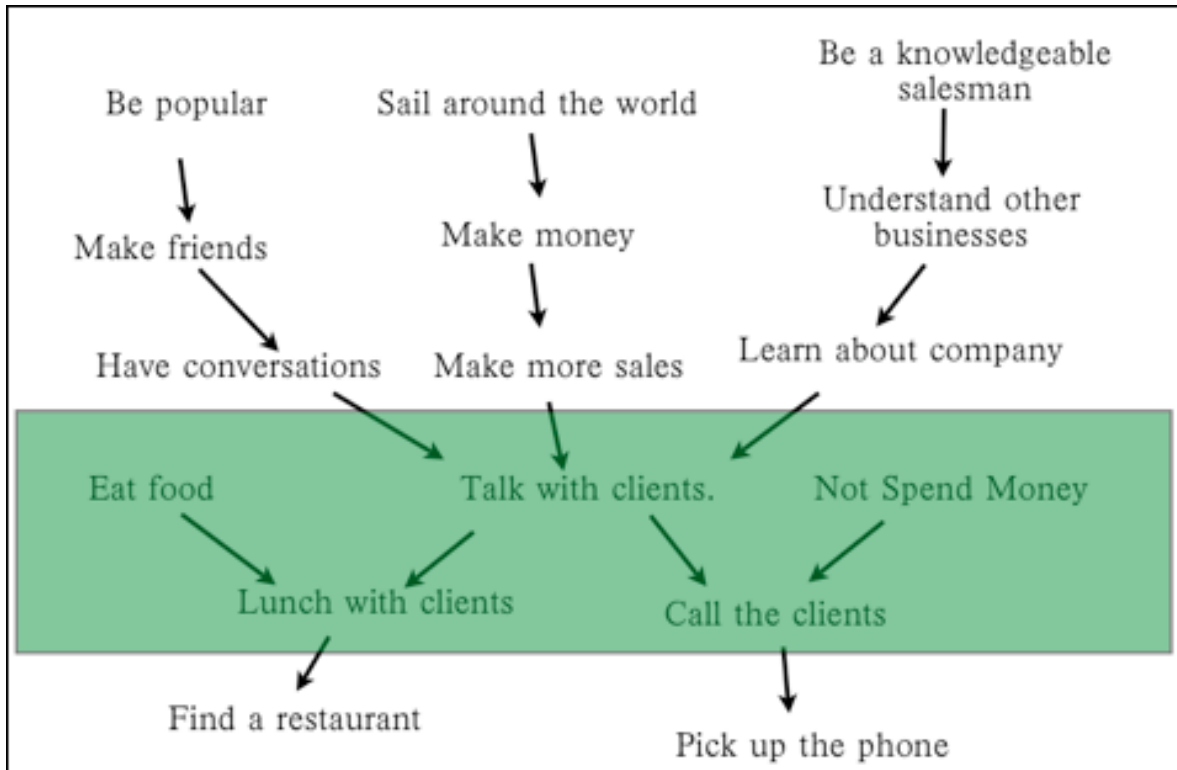


Figure 4-1: Possible goals for “talk with clients.”

The green bar on this graph is the ideal granularity of goals that we want to deal with, where the descriptions tend to be one step more general than the activity itself.

Where do goals like this come from? No available resource was perfect for this desired goal specificity, but two came close enough to be useful: OMCS and 43THINGS. Both repositories came collected goals from volunteers through a web-based interface.

In the following sections, I describe the types of goals these two repositories contain and the steps I took to process these goal descriptions.

<i>Goal Statement</i>	<i>Counts</i>
run a marathon	165
find a lost item	164
maintain good health	162
exercise	152
buy products	151
drive a car	147
go jogging	146
talk to someone	143
learn languages	142
make people laugh	142
send an e-mail	135
read the book	131
kill	128
buy a house	124
kiss	122

Table 4.2: The top 25 goals from OMCS.

### OMCS Goals

Goals from OMCS were extracted by taking the left hand side of the  $MotivationOf(x, y)$  relations in CONCEPTNET. The goals were filtered removing nonsense sentences and goals that were too low-level and contained simple actions (*e.g.*, “pick up a cup”).

### 43Things Goals

“Between the thought and action, between motion and the act, falls the shadow.” —T.S. Eliot in *The Hollow Men*

The goal statements were obtained from the website <http://www.43things.com> which is a website/cultural phenomenon where a community has formed around entering goals into a website. Figure 4.1.2 lists the 15 most common goals from 43THINGS.

Like T.S. Eliot’s shadow, many of the goals on 43THINGS are *daydreams*—fantasy plans stemming from imagining unlimited resources (“live forever,” “sail around the world”), perhaps serving the purpose of preemptive problem solving or generating motivational goals [32]. Another group of goals, perhaps a sub-set of the first type,

<i>Goal Statement</i>	<i>Counts</i>
stop procrastinating	2986
lose weight	2815
write a book	2423
fall in love	2104
be happy	1909
read books	1825
drink water	1787
take pictures	1780
get married	1623
learn spanish	1501
save money	1464
see the northern lights	1444
buy a house	1411
get a tattoo	1387
travel the world	1351

Table 4.3: The top 25 goals from 43THINGS.

involve changing the person’s self-models, like learning new skills (“learn French”) and abandoning old habits (“stop procrastinating”).

Most of the goals in 43THINGS are positive and forward-looking. They describe goals related to the things that people *want* to do, and this is not the same set of goals describing the things that people *actually do*—the things they would enter into their calendar. Consequently, for example, few goals are found to be relevant to the goal “have a meeting with your boss” while many goals were relevant to the plan “have lunch with your friends.”

### **Pre-processing goals**

Goal statements were stemmed using the Porter stemmer and stop-words were removed from the list, then duplicate goals were removed. This, for example, would result in the two goals “make more money” and “make money” being merged. There were 909 unique goals after stemming and stop word removal.

Although the stemming merged most of the goals, there were a lot of redundant goal statements, and so goals were clustered by their SVD similarity (explained in the next section).

The clustering problem requires defining a similarity metric and a threshold or number of groups to form. A good clustering is one that minimizes inter-cluster similarity and maximizes intra-cluster similarity. Truncated singular value decomposition (described later in X) was used to assess between-group similarity, where each goal was mapped into **AnalogySpace** derived from the OMCS corpus. SVD permits each item to be represented as a vector, and thus when goals are merged into the same group, the group’s vector position can be represented as the normalized sum of all its members. One problem is with this is that noise, from outliers in the groups, which accumulates and attracts other unrelated members. For example, after the grouping of goals “get out”...“get up” acquired the goal “get out more”, it caused it to merge with conceptually dissimilar the “get more exercise”...“exercise more” group. To remedy this, I augmented the similarity metric with a size penalty that penalized the groups for getting further from the average size.

$$sim(g_1, g_2) = \vec{g}_1 \vec{g}_2 - \frac{|g_1| + |g_2|}{\frac{2}{n} \sum_{g \in Goals} |g|} * penalty$$

I picked *threshold* = 2.0 the lowest similarity boundary to stop clustering, and *penalty* = 0.05, to maximize the coherent clusters. This resulted in 662 unique groups of goals, some of which were still conceptually identical to others.

## 4.2 Computing Similarity between Goals and Plans

Each plan is represented as a set of annotated semantic components. The task is to compute their pairwise similarity between each goal in the goal corpus, so that the most common goals could be added.

### 4.2.1 Computing semantic similarity with SVD

An important component of associating goals with plans is computing the conceptual similarity between a goal and the components of a plan. A robust and efficient technique for computing conceptual similarity using the principal component analysis technique of truncated singular value decomposition (SVD) was proposed by Rob Speer [42].

This technique is enabled by transforming the representation of the CONCEPT-NET semantic network [21] into a giant matrix  $A_{M \times N}$  that maps the concepts (rows) to their properties (columns). Because concepts originate in the 3-tuple form  $\langle Relation, Concept1, Concept2 \rangle$  they must be decomposed into binary relations:  $\langle Concept1, Relation \rangle$  and  $\langle Relation, Concept2 \rangle$ . For example, the assertion  $\langle CapableOf, bird, fly \rangle$ , would form properties  $p_1 = \langle bird, capableOf \rangle$  and  $p_2 = \langle capableOf, fly \rangle$ . Consequently the row corresponding to concept  $c_1 = bird$  would have a 1 in the column for property  $p_2$ , and  $c_2 = fly$  would have a 1 for  $p_1$ .

The linear algebraic technique of **singular value decomposition** (SVD) is way of reducing the dimensionality of a large matrix so that many operations, such as computing the similarity between two concepts (two rows in  $A$ ), can be performed efficiently. Efficient similarity computation over a large data set presents an ideal application for information retrieval problems, where documents are clustered or retrieved by their similarity. The SVD decomposition for any such matrix is a factorization of the form:

$$SVD(A) = U\Sigma V^T$$

The decomposition of  $A$  yields two unitary matrices  $U$  and  $V^T$ , and a diagonal matrix  $\Sigma$ .  $U_{M \times M}$  is the eigenvectors of  $A^T A$  and  $V_{N \times N}$  is the eigenvectors of  $AA^T$ .

In the smaller matrix, similarity can be computed by taking the row associated with each concept in question (now a vector) and taking their dot product. On the OpenMind Commons data set,  $M = 16,775$  and  $N = 9,414$ , and 50 singular values were found.

An interesting property of decomposition is that the principle components of the matrix, derived from larger patterns in the data, have their own meanings apart from the original 24 relationships that structured the assertions in OMCS. For example, the most significant dimension represents a *good/bad* dichotomy.. An in-depth explanation of this SVD approach and some preliminary analysis of the dimensions can be found in [42].

Another useful aspect of this approach is that similarity can easily be computed between constellations of concepts. This is achieved by adding each concept's vector

and normalizing the product. In JULIUS, this makes it possible to analyze various components of the event and see subsets are most relevant to retrieving goals, an experiment done in the next chapter.

## Chapter 5

# Evaluation

In the previous chapter, I presented a technique for inferring the goals from plans that are represented as basic English sentences. In this chapter, I evaluate five related approaches for goal inference against the data collected from 19 people.

### 5.1 Creating a Metric

In order to use this measure, we need some binary metric to determine whether a goal is relevant to the plan or not. To obtain such a **gold standard**, I turned to people.

#### 5.1.1 Collecting a human-annotated gold standard

Subjects were recruited to participate in a web-based survey with a  $\frac{1}{25}$  chance of earning \$20. The participants were required to input at least 3 goals for 15 plans that were randomly chosen from the corpus. The survey had an auto-completion feature that showed other goals in the database as the user's added the goal; but they were allowed to add their own entries as well (see figure 5-1).

Of the 31 participants, 19 completed the survey and attritional data were removed. Most participants exceeded the 3-goal minimum and in average 4.1 goals were entered per plan, with the most ( $91/19 = 4.8$ ) for the plan “read” and the least (3.4) for “lunch at cheap restaurants with colleagues.” See figure 5.1.

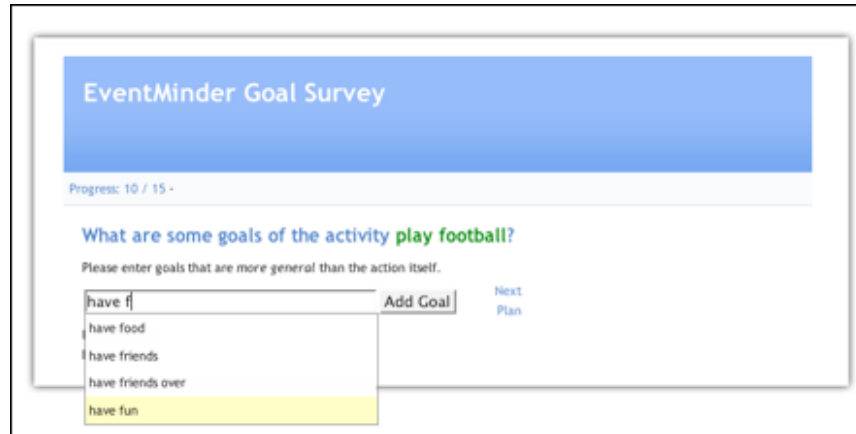


Figure 5-1:

<i>Plan</i>	<i>Total Goals Entered</i>	<i>Identified</i>
read	91	60
fish	87	57
dinner with girlfriend at restaurant	86	51
ski	85	53
sing	84	45
run in a marathon	81	47
ride a bicycle	80	53
play football	78	47
drink beer with your client	76	28
dinner with clients at nice restaurant	74	20
write a term paper	74	44
go to a baseball game	73	42
play poker	72	48
cook a curry	67	35
lunch at cheap restaurant with colleagues	65	37

Table 5.1: Listed are the total number of goals entered by the 19 participants who completed the study.

In this evaluation, I sought to find out which components of a sentence were most useful for retrieving target goals using the SVD analysis to compute between-group similarity. This exploration compared various partitions of the plan to see if lexical-semantic relations could help in goal recognition. Each plan was partitioned in the following ways:

1.  $G_1$  one group for each step of the plan.
2.  $G_2$  one group for each verb.
3.  $G_3$  one group for each VERBNET verb class.

<i>Grouping</i>	<i>Plan Average Positions</i>	<i>In Top 10</i>	<i>Top 10 Average Positions</i>
$G_1$	269.72	2.5	4.6
$G_2$	270.08	2.4	4.5
$G_3$	270.55	2.2	2.2
$G_4$	270.64	2.1	5.0
$G_5$	270.90	1.9	4.3

Table 5.2: Goal retrieval Results

4.  $G_4$  one group for each argument’s VERBNET role.
5.  $G_5$  one group for each argument’s PROPBANK role.

Each plan was divided into groups according to the above structure. Then a ranked similarity list was formed between all groups and each of the unique 662 groups of goals.

**Group Average Position** The average position of each gold goal in the list. The lower the number, the most similar (better).

An aggregate similarity list for each plan was formed by averaging or summing the similarity for each goal between each of the plan’s lists. This produced a single list for each plan. From this list, other a plan level analysis used the metrics:

**Plan Average Positions** The average position of gold goals for all groups.

**Number in Top 10** The average number of gold goals in the top 10 goals per plan:  
 $|retrieved \cap relevant|$

**Top 10 Average Positions** The average positions of the goals *in the top 10 list* per plan.

I did not use the precision/recall metrics because the collected gold goal data was so sparse that it does not completely cover the reasonable goals for each plan. Instead, the “Numer in top 10” yields a similar measure to precision, but where the number of retrieved documents is cut off at 10.

## 5.2 The Results

These data suggest that the  $G_1$  division, at best, is capable up 1/4 of the same results in the top 10. The next analysis looked at the members of the groups that

had particularly strong or weak influences on the correct goal assignment to improve this result.

## Chapter 6

# Conclusion

In this thesis I have contributed:

1. Two models for understanding events in a calendar interface, and an integration of the two in an application.
2. A novel technique for inferring goals automatically from plans, by combining semantic parsing with goal and commonsense knowledge-bases.

In 6.1, I describe the problems of integrating both models and the motivation for doing so. In 6.2, I explain some of the other features of EVENTMINDER's interface. In 6.3, I outline future directions for EVENTMINDER and goal inference.

### 6.1 Joining Romulus and Julius

A reconciliation of both models is easily justified on theoretical grounds, as goals can have different meanings depending on the context of their use.

For example, people would recognize the plan *eating dinner at home* to have the characteristic plan-level goal of "to save money." However, JULIUS would not make such a claim because one of plan's associated steps is "buy groceries," and there are many unrelated plans that do not involve any financial task and are thus (in the system's eyes) more deserving of the goal "to save money." The system, if capable, would justify this conclusion: "if you really wanted to save money, you should play Frisbee, watch the sunset or take a walk."

We can characterize this mistake as a problem of scope. The goal description, “save money,” was conditioned upon the plans that involve eating food. Within the local plan space of eating meals (perhaps dinners), dining at home is indeed more frugal to the alternatives of eating at restaurants, ordering take out, etc. And within this space of cheap meals, there are possible plans “go to a lecture”—containing the possibility of completely free meal, that should not be considered because of other reasons (dependent upon external factors; uncertainty, etc).

## 6.2 Extending the Calendar

Some of the types of assistance provided by EVENTMINDER were mentioned in the scenario but not explored earlier in this thesis.

In 6.2.1, I show how a rule-based system, the *Critic-Selector* model, can be used to recognize problems in plans and react in specific ways by learning or changing the way plans are retrieved. In 6.2.2, I briefly demonstrate of how sentence level predicate-arguments can bind to processes and data, so that EVENTMINDER can execute actions in the plan it is developing.

### 6.2.1 Recognizing problems with the plan

The vernacular connotation of the word *commonsense* means “not making stupid mistakes,” *i.e.*, by using common knowledge. We could imagine that a user of EVENTMINDER would be frustrated if they saw these events on their calendar:

- A dinner that starts at 6:00AM [31]
- A five-hour long lunch
- An meeting that begins before it ends
- Taking a plane to get to lunch

Figure 6-1: A list of events lacking commonsense.

Our system must recognize and, if possible, correct these anomalous events before they provoke the user or cause the system to behave the wrong way.

## The Critic-Selector Model for Debugging Plans

The critic-selector model, proposed by Minsky [29] and implemented by Singh [38] is part of a 6-layer agent-based cognitive architecture that is controlled by rules at each of the six layers. Recognizing patterns in their subordinate layer, *critics* detect problem states and activate context-sensitive *selectors* which manage the activity of agents (they turn on/off resources).

One of the novel and powerful aspects of this model its development of the *reflective* layer (also known as *metacognition* [4, 5]), which detects and reacts to problems in the deliberative sub-layer. The planning problems are dealt with by switching representations and engaging a host of specific learning methods (to achieve, in many cases, “one-shot learning”).

The error correction of EVENTMINDER could be thought of as a series of reflective critics with very simple selectors. Given an example plan  $\mathcal{P}$  and a set of goals  $\mathcal{G}$  inferred to be active:

1. If CRITIC finds a problem in plan  $\mathcal{P}$ , then:
  - (a) SELECTOR finds similar plans to  $\mathcal{P}$  in goal-space  $\mathcal{G}$ , or:
  - (b) SELECTOR proposes anti-goal,  $g$ , and finds similar plans to  $\mathcal{P}$  in goal-space  $\mathcal{G} \wedge \neg g$ .

Like the earlier problem of goal-driven classification, we could change the representation of the plan space depending on the active goals (feature selection).

## 6.3 Future Directions

### 6.3.1 Integration with a cellphone to “close the loop”

A little knowledge goes a long way in a personal assistant—consider a cellphone that knows when you are in a movie theater: one bit of information ( $InMovieTheater(User)$ ) should stifle its ring. Classifying the type of event is difficult because there are a lot of possible things you could be doing at any given time or place.

If a cellphone knew *what* a user was doing at a given time, it could anticipate how they would use the device. If the user were going to meet John at Legal Sea Foods at 7:00PM for dinner, an anticipatory cellphone interface could:

- Alert the user to upcoming events on their schedules.
- Direct the traveling user toward their destination or transportation (*e.g.*, subway stations, garages, etc)
- Predict the people the user will call so those peoples' number can be dialed by pressing a single button.
- Courteously turn down the cellphone's ringer during movies, conferences, nice dinners and the like.
- Facilitate multi-step problem solving, such as using a single command to find the nearest taxi company, send them a pick-up request with your current location and photo, and establish a credit card transaction.

A lot of researchers have recognized the potential of using cellphones as intelligent personal assistants. Cellphones are the computer people don't seem to mind carrying with them at all times. This has led to the research agenda of *pervasive computing*, and many have attempted to create context-aware devices, typically by trying to classify the data collected by the cellphone's sensors to labeled examples of fine-grained activities like "picking something up" and "climbing up stairs." There are many problems with this approach: obtaining training data is difficult (users don't want to annotate their schedules for the program), the learning task is very noisy and leads to poor accuracy, and the physical actions the cellphone attempts to recognize are too fine-grained to map to typical cellphone services.

The idea I propose is simple: use the semantic knowledge when it already exists from the user's calendar! Of course the widely used calendars like Microsoft Outlook and Apple's iCal do not have a lot of rich background knowledge about the events, but they could easily be extended. They currently have optional fields for specifying this information as free-text fields, but they do not fill in missing values nor connect the labels to background web services and address books. **Where** fields should map to GPS coordinates and **Who** labels to people's phone numbers and Bluetooth IDs.

To close the loop, putting EventMinder on a cellphone serves the additional purpose of collecting more accurate information about the user. The cellphone can record *how long* the user spent at a given location and use that to update it's model

about the duration of the user's common events. In other words, this could be used for recognizing planning mistakes and learning by modifying earlier plans.



# Appendix A

## Supplementary Material

### A.1 The Need for Background Knowledge

In the scenario in section 1.2, the decisions made by EVENTMINDER the assistant brought a lot of knowledge to bear. Here is a description of some of the types of knowledge involved:

**Knowledge about the user’s goals.** The user scheduled the event to accommodate some goals; what are they?

**Knowledge about common plans** What types of plans should we consider: from the ambitious and vague (*i.e.*, “entertain guests”) to common minutiae (*i.e.*, “pick up the cellphone”)? How many common events should we represent? How should we represent them and what questions do we need events to answer?

**Knowledge about taxonomic instances.** How do we move from general knowledge, for example that lunches often take place at restaurants, to knowledge of specific nearby restaurants? Is this knowledge stored locally or acquired as needed (*e.g.*, through a web service)?

**Knowledge about ways to fail.** Any intelligent system should anticipate mistakes and should possess **negative expertise** (a term coined by Minsky [28]): ways to detect and correct common mistakes [41, 29]. Without a taxonomy of common errors, the system would have to learn quickly from its mistakes—or detect mistakes before they happen; and, users will be repelled from a system that

does not learn from one or a very few mistakes. A reflective system is one that can identify the type of mistake and then select and engage the appropriate learning mechanism.

**Knowledge about natural language** Lexical-semantic knowledge, or some similarly powerful communication medium, is necessary for converting representations into something that can be understood by people. Particularly when the plans are theirs!

## A.2 The Roles of Goals in Problem Solving

The concept of a goal is useful for thinking about sophisticated problem solving systems. Here are several reasons why. The axes along which these categories were drawn does not distinguish between the system's particular implementation (human/machine) or, in the case of software, its creator's engineering objectives (cognitive architecture/application):

**Planning.** In planning, goals allow the problem to be represented independently from the particular means of achieving a solution. The classical planning formalism is: given a situation and goal state, find a sequence of actions to move between the states. An alternative formulation of goals is as a sequence of changes, veering away from the start state. Goals are useful here for retrieving pre-constructed plans (known as plan retrieval) and plan recognition. Plan selection equally involves the situation and the goal state descriptions, where the task is to select an operator that has preconditions that match the (sub-)situation and effects that result in the (sub-)goal. Plan recognition is the opposite of planning, where the actions are known, and the inference problem is finding the possible plans and goals of those actions.

**Categorization.** In order for plans to be re-used, they must be generalized so they can work in new situations. Plans that are re-used must be tailored to the new situation, a procedure called plan adaptation. This process replaces the specific representations in the original plan with more abstract descriptions that may extend to new instances. Commonly re-used plans, known as scripts or schemas, can be represented as structures that have slots or equivalently, accept parameters. Causal relationships between elements of these scripts specify de-

dependencies between category members or the agent's active goals. For instance, in the context of a transportation problem, a taxi can be seen as a member of either "expensive" or "fast" categories, depending on the agent's current goals.

**Central control.** If all of intelligent behavior is centered around problem solving, what determines the problems that should be solved? Goals. What are the highest-level goals and where do they come from? Of course in animals, many of these many of these come pre-installed by natural selection (e.g., what Dan Dennett calls the 4-fs: fight, flee, feed, and mate), and in humans they can be developed through social relationships (privileged relationship roles, what Marvin Minsky calls Imprimers.).

**Profiling.** Intelligent problem solvers must be able to accomplish multiple goals despite many limitations. For instance, embodied agents must solve problems sequentially and must heed to their bodies demands and those of nature. Simultaneous goals that use the same resources must be scheduled and postponed, and conflicts between goals should be resolved to avoid irony. Computer programs that assist humans in general domains (like event planning) by learning the person's preferences, will need to capture *multiple* caricatures of the user in a variety of situations to deal with their changing active goals, or intentions.

### A.3 The problem of underspecification

People communicate efficiently by assuming shared context and background world-knowledge to compress the verbal messages we impart. Take for example the commonsense assertion: "females have long hair." A lot of details are left implicit:

- *Females:* What kind of females? Zebras, ducks and boats? No, human females. Babies? No, adult human females. At MIT? In Soviet Russia?
- *Hair:* Facial hair? Underarm hair? No, hair on their heads!
- *Long:* How *long* is long? Longer than what? Presumably, the hair on the heads of adult male humans.

One problem with this sentence is that it is under-specified<sup>1</sup>; the sentence assumes a shared-context to parse its intended meaning. An open research problem is to develop a way to automatically expand the context to produce longer statements like: “adult female humans have longer hair on their heads than adult male humans in America on planet Earth in the later 20th century.” Doing this requires parsing the sentences and resolving semantic ambiguities using many sources of background knowledge.

---

<sup>1</sup> This *underspecification problem* is different from the *over-generalization problem*, where assertions are generally true but have a few exceptions (not *all* females have long hair), which suggests that default assumptions must be treated as tentative. The first problem is mostly a communication problem, while the second is a problem of maintaining internal consistency among descriptions of particular instances.

## Appendix B

# Accompanying Data

Listed are a few examples from each data set. COMET is a concatenation of OMCS and ETS. All data can be retrieved from <http://web.media.mit.edu/~dustin/eventminder/>.

### B.1 OMCS Plan Set

Four plans from the OMCS plan set:

```
GO TO A MOVIE
buy a movie ticket
buy popcorn
buy the tickets
call a friend to see if they want to go
decide which movie appeals to you
decide which movie you want to see
eat popcorn
go home afterwards
go to the bathroom
unzip your pants
wash your hands
leave
leave home
```

leave the house  
leave the movie theater  
leave the movie theatre  
leave the theater  
look at the movie listings to see what movies are playing  
select a movie  
walk out of the theater  
watch the credits

#### PLAY SOCCER

celebrate if you won  
congratulate the other team  
flip a coin  
get dressed properly  
go the field and find out the game plan  
jog  
join a team  
kick the ball  
swing your leg backward  
leave the field  
loose your head  
put away the ball  
realize that you should call it football  
shower  
swap shirts  
take off your shoes  
walk home  
walk off the field

#### BUY CHRISTMAS PRESENTS

burn them  
carry them to a car  
cash your Christmas Club Check  
decide what to buy

get in line  
give them to people  
have a budget  
make a budget  
make a Christmas list  
make a list  
pay for them  
pay off your credit cards  
think about what your friends like  
transport them home  
wrap them  
wrap them in wrapping paper and put them under the tree  
wrap the presents

#### SHOP

compare different products  
consume  
decide what purchases you want to make  
drive to the store  
enter a shop  
examine goods available for purchase  
find somewhere to shop  
get money  
go to the mall  
go to the store  
look around  
pay for the goods you are purchasing  
pay for the items  
pay for the things you bought  
pay for your purchases  
paying  
search for item  
take your shopping home  
try on clothes

## B.2 ETS Plan Set

Four plans from the Event Test Set:

DINNER AT YOUR HOME WITH FRIENDS

invite your friends to your house

cook a big meal

eat the food

talk with your friends

LUNCH AT CHEAP RESTAURANT WITH COLLEAGUES

find a cheap nearby restaurant

get directions to the restaurant

walk to the restaurant

order your meal

eat your meal

talk to your colleagues

pay for your meal

return to the office

LUNCH AT CHEAP RESTAURANT WITH COLLEAGUES

find a cheap nearby restaurant

get directions to the restaurant

take the subway to the restaurant

order your meal

eat your meal

talk to your colleagues

pay for your meal

return to the office

MEETING WITH A CLIENT

reserve a room at the time of the meeting

invite clients to the room

go to the room

talk with the clients

DEMO FOR SPONSORS

find out the location of the meeting

find out the time of the meeting

go to the demo

give your presentation

DINNER AT A STEAKHOUSE WITH YOUR FAMILY

find a steakhouse

go to the steakhouse

eat a dinner

return to home

### B.3 Example Goal Annotation

SLEEP

rest

sleep

go to bed

go to sleep

relax

be sleeping the night before

stay in bed

RUN TWENTY-SIX MILES

exercise

you could go for a jog

run a marathon

play sports

go running

playing football  
are competing

#### GO OUTSIDE FOR AN EVENING

you could go for a jog  
go for a walk  
go running  
have clothes to clean  
take a walk  
go for a drive  
go outside for an evening  
go to the laundromat  
walk

#### WATCH A MOVIE

read a newspaper  
you're watching a movie  
read a book  
go to a movie  
see the movie  
go to a play  
watching television  
see your favorite show  
watch a musician perform  
watch TV  
use a television  
enjoy the film  
listen to some music  
use your vcr  
watch a television show  
go to a sporting event  
see a particular program

#### BE INVOLVED IN AN ACCIDENT

kill  
buy a house  
go to work  
know if you're healthy  
have a physical exam

BUY A BEER

wait on-line  
a shop  
buy him a present  
not buy hamburgers

ATTEND A LECTURE

read a newspaper  
read a book  
study your subject  
study  
examine  
visit a museum  
go to school  
learn about a subject

PLEASE YOUR PARENTS

propose to a woman  
a party  
surprise someone  
give gifts  
please your parents

COMFORT A FRIEND

talk

kiss someone  
communicate  
are helpful

#### TAKE FINALS

pass a course  
go to school  
get good grades  
pass the class

#### GO TO AN OPERA

go to a movie  
go to a play  
watch a musician perform  
go to a concert  
go to a sporting event  
see the band  
enjoy the film

#### WIN A BASEBALL GAME

play  
play sports  
run a marathon  
have a game to play  
are competing  
win the baseball game

#### LEARN SOMETHING NEW

learn how  
read a newspaper  
study  
examine

learn about a subject  
study your subject  
learn new things  
learn about science  
find information

#### EAT LUNCH

eat it  
eat lunch  
eat your breakfast  
have something to do during lunch  
eat dinner  
food  
bring home some fish  
not buy hamburgers  
cook dinner  
go to a restaurant  
maintain good health  
make sure you are healthy  
buy fresh fruits and vegetables  
eat an apple

#### WRITE A PROGRAM

remember  
programs  
working  
add each number  
a computer program  
calculate things quickly



# Bibliography

- [1] CARRERAS, X., AND ARQUEZ, L. Introduction to the conll 2005 shared task: Semantic role labeling, 2005. 42
- [2] CHKLOVSKI, T., AND GIL, Y. Improving the design of intelligent acquisition interfaces for collecting world knowledge from web contributors. In *K-CAP '05: Proceedings of the 3rd international conference on Knowledge capture* (New York, NY, USA, 2005), ACM Press, pp. 35–42. 37
- [3] COHN, D., ATLAS, L., AND LADNER, R. Improving generalization with active learning. *Mach. Learn.* 15, 2 (1994), 201–221. 33
- [4] COX, M. T. Metacognition in computation: A selected research review. *Artificial intelligence* 169, 2 (Oct 2005), 104–141. 61
- [5] COX, M. T. Metareasoning, monitoring and self-explanation. *AAAI-07: Proceedings of the First International Workshop on Metareasoning in Agent-based Systems* (Mar 2007), 46–44. 61
- [6] DANG, H. T., KIPPER, K., PALMER, M., AND ROSENZWEIG, J. Investigating regular sense extensions based on intersective Levin classes. In *Proceedings of the Thirty-Sixth Annual Meeting of the Association for Computational Linguistics and Seventeenth International Conference on Computational Linguistics* (San Francisco, California, 1998), C. Boitet and P. Whitelock, Eds., Morgan Kaufmann Publishers, pp. 293–299. 42, 43
- [7] DAVIS, E. *Representations of commonsense knowledge*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990. 23

- [8] DAVIS, R., SHROBE, H., AND SZOLOVITS, P. What is a knowledge representation? *AI Magazine* 14, 1 (May 1993), 17–33. 39
- [9] DEJONG, G., AND MOONEY, R. Explanation-based learning: An alternative view. 31
- [10] ERK, K., AND PADO, S. Shalmaneser - a flexible toolbox for semantic role assignment. In *Proceedings of LREC 2006* (Genoa, Italy, 2006). 39, 40
- [11] FELLBAUM, C., AND MILLER, G., Eds. *WordNet*. The MIT Press, 1998. 38, 39, 40
- [12] GEORGEFF, M. P. Reasoning about plans and actions. In *Exploring artificial intelligence: survey talks from the natl. conf. on AI*, H. E. Shrobe, Ed. Morgan Kaufman, 1988. 36
- [13] GILDEA, D., AND JURAFSKY, D. Automatic labeling of semantic roles. *Association for Computational Linguistics* (Aug 2002). 20, 38, 39
- [14] HORVITZ, E. Principles of mixed-initiative user interfaces. 31
- [15] KAMBHAMPATI, S., AND SRIVASTAVA, B. Universal classical planner: An algorithm for unifying state-space and plan-space planning. In *Proc. of 3rd European Workshop on Planning (EWSP)* (1995). 22
- [16] KINGSBURY, P., AND PALMER, M. From treebank to propbank, 2002. 20, 38, 39, 41
- [17] KIPPER, K., DANG, H. T., AND PALMER, M. Class-based construction of a verb lexicon. In *AAAI/IAAI* (2000), pp. 691–696. 20, 38, 39, 42, 43
- [18] LEVIN, B. *English Verb Classes and Alternations: a preliminary investigation*. University of Chicago Press, Chicago and London, 1993. 41, 42
- [19] LIU, H. Semantic understanding and commonsense reasoning in an adaptive photo agent. Master’s thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2002. 37
- [20] LIU, H. Semantic understanding and commonsense reasoning in an adaptive photo agent. 40

- [21] LIU, H., AND SINGH, P. Conceptnet: A practical commonsense reasoning toolkit. 53
- [22] LOPER, E., YI, S.-T., AND PALMER, M. Combining lexical resources: Mapping between propbank and verbnet. In *Proceedings of the 7th International Workshop on Computational Linguistics* (Tilburg, the Netherlands, 2007). 42
- [23] LOWE, J., BAKER, C., AND FILLMORE, C. A frame-semantic approach to semantic annotation, 1997. 20, 38, 39, 40
- [24] MANTARAS, R. L. D., MCSHERRY, D., BRIDGE, D., LEAKE, D., SMYTH, B., CRAW, S., FALTINGS, B., MAHER, M. L., COX, M. T., FORBUS, K., KEANE, M., AAMODT, A., AND WATSON, I. Retrieval, reuse, revision and retention in case-based reasoning. *The Knowledge Engineering Review*, 1-2 (Nov 2005). 36
- [25] MARKMAN, A. B., AND ROSS, B. H. Category use and category learning. *Psychological bulletin* 129, 4 (Jul 2003), 592–613. 29
- [26] MICHALSKI, R. S., AND STEPP, R. Clustering. *AI Encyclopedia* (Jun 1986). 28
- [27] MINSKY, M. A framework for representing knowledge. In *Readings in Cognitive Science: A Perspective from Psychology and Artificial Intelligence*, A. Collins and E. E. Smith, Eds. Kaufmann, San Mateo, CA, 1988, pp. 156–189. 22, 23, 43
- [28] MINSKY, M. Negative expertise. *International Journal of Expert Systems* 7, 1 (1994), 13–19. 65
- [29] MINSKY, M. *The Emotion Machine*. Simon and Schuster, 2006. 24, 61, 65
- [30] MITCHELL, T. M., CARUANA, R., FREITAG, D., MCDERMOTT, J., AND ZABOWSKI, D. Experience with a learning personal assistant. *Communications of the ACM* 37, 7 (1994), 80–91. 31
- [31] MUELLER, E. T. A calendar with common sense. In *IUI '00: Proceedings of the 5th international conference on Intelligent user interfaces* (New York, NY, USA, 2000), ACM Press, pp. 198–201. 60
- [32] MUELLER, E. T., AND DYER, M. G. Daydreaming in humans and computers. In *Proc. of the 9th IJCAI* (Los Angeles, CA, 1985), pp. 278–280. 50

- [33] ROSCH, E. Reclaiming concepts. *Journal of Consciousness Studies* 6, 11-12 (Sep 1999), 61–77. 29, 30
- [34] RUMELHART, D. E., SMOLENSKY, P., MCCLELLAND, J. L., AND HINTON, G. E. Schemata and sequential thought processes in pdp models. In *Readings in Cognitive Science: A Perspective from Psychology and Artificial Intelligence*, A. Collins and E. E. Smith, Eds. Kaufmann, San Mateo, CA, 1988, pp. 224–249. 22
- [35] SCHANK, R. C., AND ABELSON, R. P. Scripts, plans, goals and understanding. In *Readings in Cognitive Science: A Perspective from Psychology and Artificial Intelligence*, A. Collins and E. E. Smith, Eds. Kaufmann, San Mateo, CA, 1988, pp. 190–223. 23
- [36] SETTEN, M. V., VEENSTRA, M., NIJHOLT, A., AND VAN DIJK, B. Goal-based structuring in recommender systems. *Interacting with Computers* 18, 3 (Aug 2006), 432–456. 31
- [37] SINGH, P. The panalogy architecture for commonsense computing. 24
- [38] SINGH, P. *EM-ONE: An Architecture for Reflective Commonsense Thinking*. PhD thesis, MIT Department of Electrical Engineering and Computer Science, 2005. 24, 37, 61
- [39] SINGH, P., AND BARRY, B. Collecting commonsense experiences. In *K-CAP '03: Proceedings of the 2nd international conference on Knowledge capture* (New York, NY, USA, 2003), ACM Press, pp. 154–161. 37
- [40] 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. *Lecture Notes in Computer Science* 2519 (2002), 1223–1237. 33, 37
- [41] SINGH, P., AND MINSKY, M. An architecture for combining ways to think. 65
- [42] SPEER, R. Learning common sense knowledge from user interaction and principal component analysis. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2007. 52, 53
- [43] STEPP, R. E., AND MICHALSKI, R. S. Conceptual clustering of structured objects: a goal-oriented approach. *Artif. Intell.* 28, 1 (1986), 43–69. 30

- [44] WINSTON, P. H. Learning structural descriptions from examples. In *Readings in Knowledge Representation*, R. J. Brachman and H. J. Levesque, Eds. Kaufmann, Los Altos, CA, 1985, pp. 141–168. 33
- [45] WOODS, W. A. What’s in a link: Foundations for semantic networks. In *Representation And Understanding: Studies in Cognitive Science*, D. G. Bobrow and A. M. Collins, Eds. Academic Press, New York, NY, 1975, pp. 35–82. 32
- [46] ZACKS, J. M., SPEER, N. K., SWALLOW, K. M., BRAVER, T. S., AND REYNOLDS, J. R. Event perception: a mind-brain perspective. *Psychological bulletin* 133, 2 (Mar 2007), 273–93. 23
- [47] ZACKS, J. M., TVERSKY, B., AND IYER, G. Perceiving, remembering, and communicating structure in events. *Journal of experimental psychology General* 130, 1 (Mar 2001), 29–58. 37