Barnyard Politics: A Decision Rationale Representation for the Analysis of Simple Political Situations

by

Arian Shahdadi

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

Master of Engineering in Computer Science and Engineering

at the Massachusetts Institute of Technology

August 9, 2003

Author ....
Department of Electrical Engineering and Computer Science
August 9, 2003

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

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

Barnyard Politics: A Decision Rationale Representation for the Analysis of Simple Political Situations

by

Arian Shahdadi

Submitted to the Department of Electrical Engineering and Computer Science
on August 9, 2003, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

How can a computational system understand decisions in the domain of politics? In order to build computational systems that understand decisions in the abstract political space, we must first understand human decision-making and how human beings, in turn, are able to understand ideas in abstract realms such as politics. The work of Jintae Lee attempted to address the problem of understanding decision rationales in a non-domain specific way. His work falls short, however, when applied to decision problems in politics. I present a new representation, the Augmented Decision Rationale Language (ADRL) that attempts to address the shortcomings of Lee's work in this regard. ADRL expands Lee's vocabulary of relations to include forms of causation such as enablement and gating. ADRL also refines the relations and primitives of Lee's representation and focuses primarily on States, Actions and Goals as the basic units of decisions. Finally, ADRL grounds itself in spatial understanding using the Lexical Conceptual Semantics of Jackendoff, in contrast to the DRL, which ignores the text associated with a decision rationale. An implementation of a subset of this representation is displayed, along with a matcher that is able to create analogies between two decision scenarios cast in the representation. The matcher is presented an existence proof that this representation can be used to readily structure decision scenarios and make analogies between them.

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

Acknowledgments

Thanks to: My parents, Patrick Winston, Paul Gray and my many instructors and mentors at MIT.

This research was supported in part by the National Science Foundation, Award Number IIS-0218861.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the new global political climate, fear and uncertainty abound. The old ways of analyzing and understanding political situations have proved fallible at best, as terrorism and recent wars demonstrate. A significant point of failure in current methods of political analysis is the human analyst. With the sheer volume and complexity of the information that analysts must process, it is no wonder that many disasters are not identified and averted. For this reason, we seek to create computationally oriented applications to help analysts better understand and utilize information.

One application in particular that could be of great importance is called a *blunder stopper*. A blunder stopper is an application that can automatically analyze a decision scenario and tell a user if, based on past precedents, the current decision-making process may lead to unexpected and possibly disastrous outcomes. Thus, the question becomes, how can we best make analogies between past precedents and current situations?

In order to create an application like a blunder stopper, we must first understand how human beings structure decisions in the political space. To this end, we must have a computational representation that captures the essential aspects of political decisions. The representation must also facilitate the construction of analogies between decision scenarios cast in its language.

This work takes the first steps toward the vision by introducing a representation for decision rationale in the political domain. The representation expands and refines the work of Jintae Lee. [6] Lee's work introduces the DRL (Decision Rationale Language), a language meant for use in collaboration tools. His language takes eight basic primitives and describes the possible relationships between them in an arbitrary decision situation. Take the following short example:

```
Jane kicked to the ball.
The ball moved to the tree.
```

In Lee's conception, this situation would be represented by two objects with a relation between them. The sentences associated with these objects are the so-called *textual content*. Lee's work does not concern itself with the textual content of the decision. Rather, the only important aspects of a decision scenario in Lee's representation are the classes of objects in the system and the types of relations between them. This is, in part, because Lee wanted to create a system that did not rely on domain-specific knowledge.

I believe, however, that the textual content of the decision is crucial to its understanding. In particular, it would be very difficult for a computational system to discern similarities and differences between decision situations without some knowledge inherent to the specifics of the situation. This does not mean that we must rely entirely on domain specific representations. Rather, we can use a non-domain specific representation that enables understanding of specific domains. I will argue that Lee's representation is insufficient to enable meaningful analogies between political decision scenarios without the help of an augmenting representation for the textual content of the decision. To enable these analogies, I have refined Lee's representation in the following ways:

1. I have clarified and narrowed Lee's set of primitives to five basic types: Scenarios, Elements, States, Actions and Goals.

2. I have removed relations, such as those dealing with alternatives to a decision and questions about a decision, that are not crucial to understanding past precedents.

3. I have expanded Lee's relation vocabulary to encompass more diverse relations, especially focusing on kinds of causation, such as enablement and gating.

4. I have augmented the decision rationale representation to include a semantic representation of the textual content of the decision rationale.

The new relations and my refinement of Lee's primitives are inspired by the work of Robert Abelson. [1] Abelson sought to create a computational system that would understand political concepts in texts. His work clearly outlined many different types of causation in political systems and also delineated the need for three primitive types representing the current state of the world, the possible actions that can be taken in the world and the overall purpose that is being achieved.

I selected the Lexical Conceptual Semantics (LCS) representation of Jackendoff for the semantics of sentences in a decision rationale. Jackendoff's work represents notions both concrete and abstract in terms of places and paths in space. Using Jackendoff's work as a basis, I have developed LCS representations for 42 different sentences useful in describing simple political situations. These situations represent politics in a so-called "barnyard" of agents, objects and actions. I do not attempt to address complex political situations or vocabulary, although I am confident that they can be represented in the same way with very little additional representational apparatus.

In order to show that the representation is useful and to take the first steps toward creating a blunder stopper, I have implemented a subset of the representation and developed a matcher that can match decision scenarios in the new representation. The matcher application allows users to enter decision scenarios in an efficient manner as a series of sentences and numerical references to relations between them. The application will then match them up based on their spatial semantic representations and relations. If a match occurs, the returned match represents the base commonalities between the two scenarios and shows the relations that are common and missing among scenario elements. A functioning matcher represents a valid first step toward a blunder stopper, because such an application would need a way to match up scenarios to determine possible blunders.

The remainder of this document will present the details of my work. Chapter 2 discusses the decision rationale representation I developed and contrasts it to Lee's work. Chapter 3 presents the structures I developed for representing sentences in the barnyard world and briefly touches on the implementation of these structures via the Bridge system. Chapter 4 presents the details of the matcher, specifically the process of converting scenarios represented in the ADRL into a graph structure more suitable for matching and how the matcher operates on these structures. Chapter 5 presents avenues for future work. Chapter 6 is a discussion of the salient contributions of this thesis. Appendix A is a reference for the decision rationale language presented in this thesis. Appendix B is a reference for the LCS structures developed to support this work. Appendix C gives a brief overview of the Bridge system and how it was used to develop the matcher. Finally, Appendix D gives an overview and discussion of some of the related works used as a foundation for this thesis.

Chapter 2

A Representation for Political Decision Rationales

Jinate Lee's work on decision rationale is inspired by the pioneering work of Toulmin, who sought to describe the primitives necessary for structuring an argument. Lee's Decision Rationale Language (DRL) addresses this problem from the standpoint of extreme generality, where no domain-specific knowledge is necessary for describing and understanding a decision. Where Lee falls short, however, is in his insistence that the textual content of a decision has no merit in terms of describing its structure. I will show that, in fact, Lee's representation is insufficient for creating meaningful analogies between decision scenarios in the political domain. I will present my refinements of Lee's work, which include changes to his set of primitives and his relation vocabulary, as well as the addition of augmenting representations for the specific sentences describing the various aspects of a decision rationale.

2.1 Some Motivating Examples

In order to show the differences between Lee's work and my own, and specifically, to show why my changes to his representation were necessary, it is important to look at some examples. The problem that we are trying to solve is one of analogy: how can we represent scenarios such that a computational system can readily make analogies between similar, but not identical, decision rationales? What sort of representation for the scenarios is necessary for this task?

Let us address a simple example of a decision scenario in the barnyard world. Billy is a young boy who buys a toy at the local store. Now that Billy has this toy, he wishes to put it in a secure place. Billy is in possession of a box, but this box doesn't have a lock on it. Therefore, the box is insecure. Now, Billy wants to keep this toy that he has just purchased. At the same time, the local bully, Joel, knows that Billy has this toy and wants it for himself. Billy, realizing this, is faced with several options. Billy can keep the toy on himself at all times, perhaps by putting it in his pocket. Billy could also put the toy in the box. If he is feeling apprehensive, Billy could very well give the toy up to Joel. Billy could also go down to the local "bank" (his parents) and ask them to store the toy for him.

Our basic example is a simple story. How are we to represent this story computationally? The first step would be to decompose the story into several smaller and distinct sentences that reveal the relations between aspects of this decision. One such decomposition is as follows:

```
Billy bought the toy.
Billy owns the toy.
The box does not possess a lock.
The box is insecure.
Billy wants to keep the toy.
Joel wants to own the toy.
Billy puts the toy in his pocket.
Billy gives the toy to Joel.
Billy puts the toy into the box.
Billy sends the toy to the bank.
```

Let us say that this is our "basic" example. What sorts of scenarios could we try to compare this to, such that the comparison will inform us about the power of the representations we are looking at?

Consider another example. In the far future, the Earth is fighting a war with Mars. The Earth has purchased a special hypership from alien arms dealers. The Earth, for obvious reasons, would like to keep this ship and store it in a secure location. One location that has been considered is a space-port orbiting the moon. The space-port, however, has a very light set of military defenses and is not very secure. The Martians have taken note of the Earth's purchase and would desperately like to procure it for themselves. The Earth's government is faced with faced with several options. It could send the hypership to the space-port. The Earth could also keep the hypership in one of it's countries, like Canada. The Earth also has access to a battle-platform orbiting Venus, which could both store and defend the hypership. Finally, if they were so inclined, Earth's governors could accede and give the hypership to Mars willingly, perhaps as a peace offering.

As any human reader can notice, this decision story is analogous to the previous story about Billy. This is easier to notice if the story is decomposed as follows:

```
The Earth bought the hypership.
The Earth owns the hypership.
The space-port does not possess a military.
The space-port is insecure.
The Earth wants to keep the hypership.
Mars wants to own the hypership.
The Earth puts the hypership in Canada.
The Earth gives the hypership to Mars.
The Earth puts the hypership into the space-port.
The Earth sends the hypership to the battle-platform.
```

Let us consider an example where there is no similarity. The following sentences describe the new story:

```
1.  Billy bought the toy.
2.  Billy owns the toy.
3.  Joe has the fish.
4.  Joe ate the fish.
5.  Billy wants to keep the toy.
```

```
6.  Joe wants to own the car.
7.  Billy puts the toy in his pocket.
8.  Joe buys the car.
9.  Joe's mother buys the car for Joe.
10. Billy sends the toy to the bank.
```

One can see that this story seems structurally similar to our basic example. Note, however, that the two stories are quite different. In particular, a good representation should elucidate this difference so that it can be exploited by applications trying to find or invalidate matches between decision rationales.

We can look at the story about Billy in another way as well. What if we simplified the story? Say that we reduce the set of sentences in the basic example to the following:

```
Billy bought the toy.
Billy owns the toy.
The box is insecure.
Billy wants to keep the toy.
Billy puts the toy in his pocket.
Billy puts the toy into the box.
```

To the human reader, the essential details of the story have not changed. The main difference is that some of the less relevant details have been removed. A good representation should capture the essence of the story so that a proper matcher will be able to tell that this version is more or less the same as our basic example.

Using these examples, we will examine Lee's DRL and my own augmented version of the DRL to determine the areas where each representation succeeds and falls short.

2.2 DRL: Lee's Decision Rationale Language

Lee created the DRL as a way to capture active decision-making processes. His language is well suited for collaborative purposes and the implementation that he demonstrates is of a collaboration tool for multiple users. The following sections present an overview of the DRL, beginning with the primitives and continuing on with the relations. The final section attempts to represent some of our example scenarios with the DRL.

2.2.1 The Primitives of the DRL

Figure 2-1 shows a diagram of the DRL vocabulary, including primitives and relations. As we can see from the figure, DRL has eight primitives, all descending from a single base primitive, DRL object. The primitives are: Alternative, Goal, Claim, Question, Group, Viewpoint, Procedure and Status. Also, a Decision Problem is a special class that represents an overall decision scenario, and is a type of Goal. Decided is a special type of Status that marks decisions that have already been made.

In order to understand how we can apply these primitives in structuring a decision, we must first discuss what they represent. A *Claim* is the basic unit of the DRL. Claims represent ideas noted by the people making the decision. Claims have an associated truth value as well, called the *plausibility* measure. If this value is high, they can be considered facts. If low, they can be taken as assumptions, hypotheses, or any number of other types

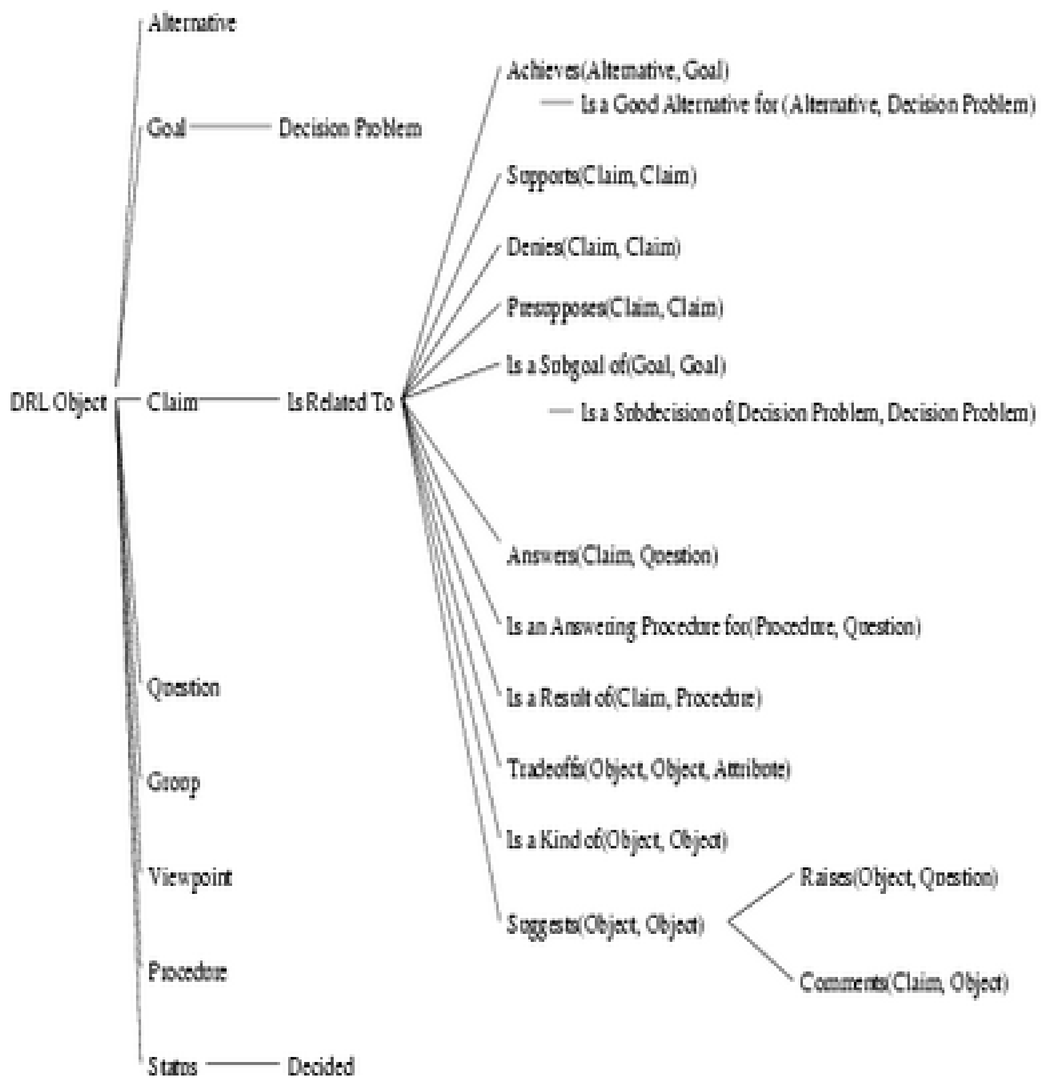

DRL Object
Alternative
Goal
Decision Problem
Claim
Is Related To
Achieves(Alternative, Goal)
Is a Good Alternative for (Alternative, Decision Problem)
Supports(Claim, Claim)
Denies(Claim, Claim)
Presupposes(Claim, Claim)
Is a Subgoal of(Goal, Goal)
Is a Subdecision of(Decision Problem, Decision Problem)
Answers(Claim, Question)
Is an Answering Procedure for(Procedure, Question)
Is a Result of(Claim, Procedure)
Tradeoffs(Object, Object, Attribute)
Is a Kind of(Object, Object)
Suggests(Object, Object)
Raises(Object, Question)
Comments(Claim, Object)
Question
Group
Viewpoint
Procedure
Status
Decided

Figure 2-1: The full vocabulary of the DRL, including primitives and relations. This diagram is adapted from an identical diagram in Lee's thesis, pp. 43.

of information as specified by the user. The threshold for these values is also set by the user. Claims also have a value called *degree*, which represents the importance of the claim in the overall scenario. This is a subjective valuation that can change as a decision situation evolves. A Claim itself is a very general category, and so analyses of decisions using Claims must rely on the extra information contained in the dynamic plausibility and degree valuations.

A *Goal* is a state that the decision-makers wish to achieve. Essentially, the decision problem involves the users setting up a series of Claims and deciding what actions need to be taken to achieve a set of Goals. Goals can be sub-goals of other goals, which means explicitly that the sub-goal must be accomplished in order for the overall goal to be accomplished.

An *Alternative* is a possible action that can be taken in the decision. Alternatives represent the steps that can be taken to accomplish a goal, and can encompass many sentences depending on the level of granularity chosen by the users. Alternatives have several attributes, including *status* (an indication of whether or not the alternative was selected) and *evaluation* (a value that describes the evaluation of this alternative in relation to the goals it is meant to achieve).

The other five primitive values are meant to support collaborative decision-making. The *Question* primitive stores a question that a user has for other members of the decision-making team, and relates this question to other objects in the decision. The *Group* primitive is a container that allows grouping of arbitrary classes into a larger set that is related in some way specified by the user. A *Viewpoint* contains pointers to a set of DRL objects that represent one opinion about the current decision. A *Procedure* defines a set of steps used to perform a task in the rationale. Procedure objects can hold both executable code and informal descriptions. Finally, the *Status* object describes the current status of the decision scenario.

2.2.2 Relations of the DRL

Although the primitives of the DRL are primarily focused on group collaboration in decision-making, the set of relations DRL encompasses are more general and can support a broader array of computations on decision rationales. They can be broken down into four categories:

1. Relations among Claims.
    - Supports(Claim, Claim)
    - Denies(Claim, Claim)
    - Presupposes(Claim, Claim)
2. Relations between Claims and other objects.
    - Answers(Claim, Question)
    - Is a Result of(Claim, Procedure)
3. Relations involving Goals.
    - Achieves(Alternative, Goal)
    - Is a Sub-Goal of(Goal, Goal)
4. Relations useful for collaborative analysis.

- Is an Answering Procedure for(Procedure, Question)
- Tradeoffs(Object, Object, Attribute)
- Is a Kind of(Object, Object)
- Suggests(Object, Object)

The first set of relations deals with the ways that Claims can connect to other Claims. *Supports* and *Denies* are two opposing relation types that describe how one Claim can impact the truth value of another. For example, if I had one Claim that stated “Pat is a woman” and another that stated “Pat is a man,” the two Claims would deny each other. *Presupposes* deals with a relationship where one Claim needs to be true in order for a subsequent Claim to also be true. For example, the Claim “Bobby can run” presupposes something like “Bobby has legs.” Note that there are no relations dealing with causation among Claims, and only a very limited vocabulary for Support and Denial.

The second set of relations involving Claims is also limited. Because Claims are central to the expression of a decision rationale, one would expect that they interact with the other classes (especially Goals and Alternatives) in many ways. That is not the case, however. *Answers* describes how a stated Claim can answer a Question that has been posted to the scenario by a user. *Is a Result of* describes how a stated Procedure object can lead to stated Claim. This relation is our first hint of causal descriptions in Lee’s representation. Note, in particular, that this only describes causation between a Procedure and a Claim. When trying to describe a larger scenario with many past states and current actions, a much richer vocabulary of causation is needed, as we will see.

The third set of relations is small, but powerful. In particular, the *Achieves* relation is very important, because it allows the user to specify what particular Alternatives will lead to accomplishing a Goal. This knowledge is important for analyzing a decision and represents the only other form of causation evident in Lee’s vocabulary. The *Sub-Goal* relation presents a formal way of specifying sub-goals. The relation also invites a question: why are there not other similar “sub” relations for some or all of the other primitives? The *Presupposes* relation is close, but one can imagine situations where certain Alternatives enable others to occur in a manner similar to how a sub-goal must be completed to enable the completion of an overarching goal.

The final set of relations deals primarily with ways to express relations among elements of the representation meant explicitly for collaboration. *Is an Answering Procedure for* describes how a Procedure object contains the information necessary for answering a Question posed by a user of the system. *Tradeoffs* gives the tradeoffs between two objects (presumably choices in the system) as a list of attribute differences. *Is a Kind of* allows users to note similarities between objects in the system even when the basic vocabulary cannot express these similarities. Finally, the *Suggests* relation is a way for users to annotate decisions. Note that in figure 2-1, this relation is a superclass of *Comments*, which is the formal way of entering uninterpreted user comments into the decision.

2.2.3 Representing Our Examples in the DRL

Now that we have Lee’s representation at hand, how are we to judge its usefulness? I will focus on attempting to represent our previous examples in Lee’s framework. I will show that Lee’s vocabulary is too limited to fully express the scenarios in a meaningful way. This becomes even more apparent when we consider that the DRL does not take the textual content of the examples into account.

Look once again at our basic example, this time with the sentences numbered for easy reference.

```
1.  Billy bought the toy.
2.  Billy owns the toy.
3.  The box does not possess a lock.
4.  The box is insecure.
5.  Billy wants to keep the toy.
6.  Joel wants to own the toy.
7.  Billy puts the toy in his pocket.
8.  Billy gives the toy to Joel.
9.  Billy puts the toy into the box.
10. Billy sends the toy to the bank.
```

Our first step is to cast the sentences in terms of Lee's primitives. Sentences 1 through 4 can all be set up as Claim objects, because they make a statement about the world as it is at the time of the decision. Sentences 5 and 6 are clearly Goals, although we have an interesting problem here, because the goals depend on the point of view of the user. If the user is Joel, then our focus would be on accomplishing 6. If the user is Billy, then the focus would be on accomplishing 5. In the DRL, there is no explicit way to describe Goals from the perspective of individual actors, because it is always assumed that all stated Goals are equally important to all users of the scenario. Finally, we have four possible Alternatives in sentences 7 through 10.

Now, let us specify the relations between the objects of this decision, because those are the key to representing decisions in the DRL. Figure 2-2 is a diagram of a plausible set of relations for this situation. Sentence 2 presupposes sentence 1 and sentence 4 presupposes sentence 3. These are the obvious causal relations among the Claims. Sentences 7 and 10 seem to achieve Billy's goal, in the same way that sentences 8 and 9 seem to accomplish Joel's. Thus, we have our DRL representation of this situation.

Consider our second example, with the planetary dispute between Earth and Mars. Again, the following are the sentences, annotated with numbers:

```
1.  The Earth bought the hypership.
2.  The Earth owns the hypership.
3.  The space-port does not possess a military.
4.  The space-port is insecure.
5.  The Earth wants to keep the hypership.
6.  Mars wants to own the hypership.
7.  The Earth puts the hypership in Canada.
8.  The Earth gives the hypership to Mars.
9.  The Earth puts the hypership into the space-port.
10. The Earth sends the hypership to the battle-platform.
```

This story is structurally similar to our basic example and also well aligned. Numbers 1-4 are Claims, numbers 5 and 6 are Goals and numbers 7-10 are Alternatives. As we can see, the relational structure is also the same; the graph in figure 2-2 could just as easily apply to this scenario as our basic example. In this way, DRL is adequate for representing the two scenarios and finding the analogy. There is, however, a significant problem here. Notice that the structures for the two *presuppose* relations and the two *achieve* relations

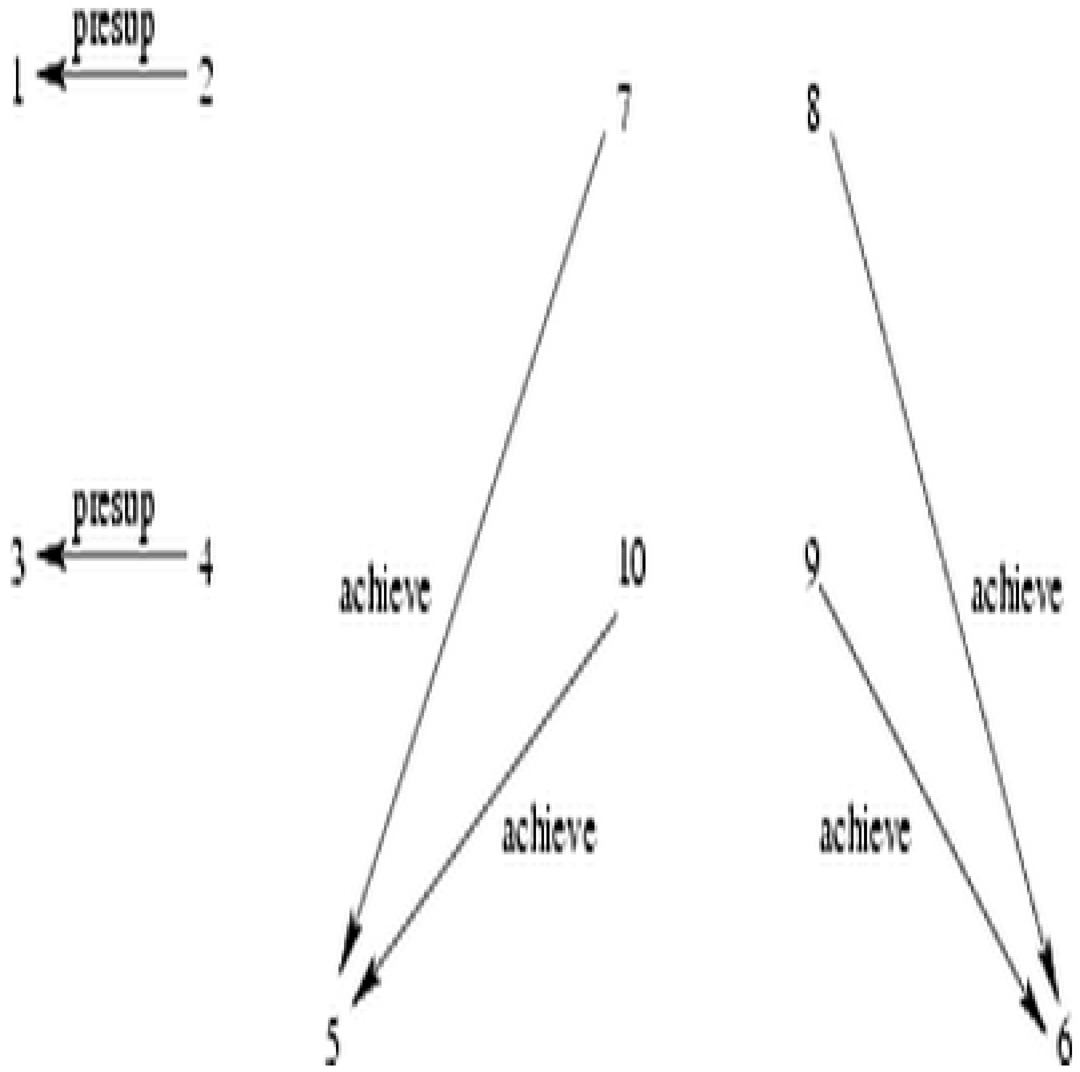


Figure 2-2: The DRL expression of the basic example, referenced by number: 1-4 are Claims, 5 and 6 are Goals, 7-10 are Alternatives.

are identical in each graph. How are we to know the correspondence between the different decisions? Because we have no representation for the semantic content of the decisions, we can never be sure of the exact alignment between the two scenarios, nor can we even be sure of alignment issues within a given scenario itself! The alignment facilitated by the DRL is too coarse to be useful in most situations. Furthermore, because there is no representation for the sentences, we have no way of knowing the specifics of the analogy we just made. For example, let's say we would like to know that the Earth and Billy fulfill similar roles in the two stories. DRL does not allow us to extract this information and in fact gives us no way of representing it.

Another problem is that the relation vocabulary is not rich enough to describe all of the inter-structure relations that might help to make up for the lack of an internal sentence representation. Take another of our examples:

```
1.  Billy bought the toy.
2.  Billy owns the toy.
3.  Joe has the fish.
4.  Joe ate the fish.
5.  Billy wants to keep the toy.
6.  Joe wants to own the car.
7.  Billy puts the toy in his pocket.
8.  Joe buys the car.
9.  Joe's mother buys the car for Joe.
10. Billy sends the toy to the bank.
```

This situation is quite different from our basic example, in that we have two distinct and non-intersecting decision problems here. Note, however, that in terms of the DRL, this situation looks exactly the same as our previous two examples. Sentences 1-4 are Claims, 5 and 6 are Goals and 7-10 are Alternatives. The relations, as specified in figure 2-2 all hold, but this scenario should not match our example. One way of alleviating this would be to set up special relations between those objects that have the same agents, but again, DRL neither has these relations nor does it have the representational apparatus to support automatic discovery of them.

Finally, consider the stripped down version of the original example:

```
1. Billy bought the toy.
2. Billy owns the toy.
3. The box is insecure.
4. Billy wants to keep the toy.
5. Billy puts the toy in his pocket.
6. Billy puts the toy into the box.
```

How would we represent this in the DRL? Sentences 1-3 are Claims, sentence 4 is a Goal and sentences 5 and 6 are Alternatives. We can see that this stripped down version of our original scenario is merely a subset of the original set of objects and relations. A simple comparison using DRL decisions would note this, but again, would analogize this simple situation to any number of similar structures that do not match it at all. Again, we face this problem because the DRL neither gives us no way of knowing the structures within the objects, nor does it have any additional relations in the language to compensate for this.

2.3 ADRL: A New Language for Decision Rationales

The decision rationale language presented in this section addresses the shortcomings of the DRL. Specifically:

- The language restricts the set of primitives to those necessary for analyzing and analogizing decision scenarios, and focuses primarily on five main primitives: Scenarios, Elements, States, Actions and Goals.
- The language expands the relations between primitives, not only allowing for a broader range of relational expressions similar to those of Lee, but also adding many relations dealing with types of causation, like enablement.
- The language explicitly represents the semantic content of the sentences associated with the decision scenario and uses this representation in making analogies and defining relations.

The next sections explore the primitives and relations of the language, as well as discussing how we can use it to represent our previous examples.

2.3.1 Primitives of the ADRL

The primitives of the language are very similar to those of the DRL. They are, however, more inspired by the work of Abelson than of Lee, because the new set of primitives does away with much of the apparatus Lee put in place to support collaborative decision-making in favor of a more descriptive language for describing the interrelations between the aspects of the decision. Figure 2-3 shows the primitives of the ADRL.

Element is the base class of the important ADRL primitives. An Element encapsulates a sentence or phrase that is a part of the decision scenario. An Element by itself has little meaning except that it can participate in a variety of relations that do not depend any specific Element sub-type.

Scenario is a class that encapsulates a single decision rationale situation. It is merely a list of the sentences in the situation stored in their respective Element sub-types, along with a record of the Relations between the Elements. Scenario objects are the basic unit of analogies: that is, analogies are made only between whole Scenario objects.

Goal is a representation of a desired state of the world. A Goal is something that the user wishes to achieve. We can also model the Goals of other actors in the situation using the *Actor-Goal* sub-type. The set of Goals and Actor-Goals defines the important desirable (or undesirable) possible outcomes of the Scenario.

Action represents a possible action in the current Scenario. An Action, when taken, will result some State or Goal. The possible actions that the user can take are modeled by Action objects, while *Actor-Actions* allow a user to model those Actions that other actors in the Scenario can take.

State represents a piece of information about the current state of the world. A State is presumed to be true or at least assumed to be true for the purposes of analyzing the decision. State has two sub-types, *Claim* and *Result*. A Claim is a piece of uncertain state information. Unlike in the DRL, Claims have no numerical values associated with them. Rather, a Claim is presumed to be uncertain enough that it should be used with some care, but this uncertainty is not quantified. This, to some extent, makes decomposing

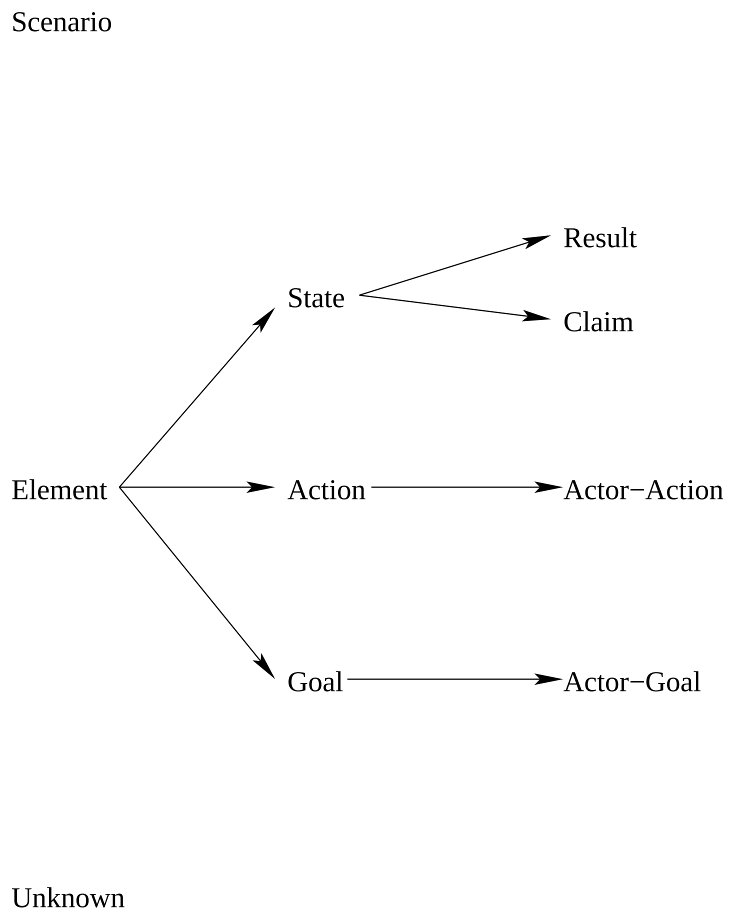


Figure 2-3: The full set of primitives in the ADRL.

decision situations easier and also leaves the handling of uncertainty up to the user. A Result represents a State that exists due to a previous Action. For example, let us say that we have the following small decision scenario:

```
GOAL: John wants the ball.
ACTION-1: John gets the ball.
ACTION-2: Jody gets the ball.
```

If the Action "John gets the ball" is taken, a possible Result is "John has the ball." Result objects are attached to their related Scenarios and Actions, allowing a computational system to trace back the causes of certain events. This information also allows for a system to predict possible outcomes after analogies have been made between two Scenarios.

Unknown is a special class that represents a piece of information unavailable to the current analysis. It is a marker of sorts for something important that is not yet quantifiable but may have some bearing on the current decision. Unknowns are generally not to be considered when performing computations using the ADRL, but this is an implementation choice at the very least.

2.3.2 Relations of the ADRL

The set of relations in the ADRL is much richer than those of the DRL in terms of describing the dynamic and static relationships among the elements of a decision rationale. Figure 2-4 is a diagram of the relation hierarchy of the ADRL. The relations in the diagram are separated into categories based primarily on the type of objects on which they operate on (this does not hold for causal and temporal relations, as these operate on all Element types).

Causal relations are those that describe different types of causation effects that can take place between two Elements. There are three main types of causation that the representation embodies: typical causation ("The boy kicked the rock" *causes* "The boy hurt his foot"), enablement ("Joe opened the door" *enables* "Jody walked through the door") and gating. Gating is somewhat more complex than the other types of causation. As described by Abelson, gating occurs when a state allows a distinct action to lead to a certain outcome. Take the following example:

```
The robbers rob the bank.
The getaway driver waits outside the bank with the getaway car.
The robbers exit the bank.
The robbers escape the bank quickly.
```

In this case, the driver waiting outside is gating the possibility of the robbers making a quick exit from the bank. Gating situations are rather complex and so the arguments are slightly different. A gating situation is expressed as an Element gating the interaction between two other Elements. Note in the relation diagram that the causal relationships also have symmetric opposites: caused by, enabled by and gated by. There is no strict requirement that both of the symmetric relations be in place in a scenario decomposition, but this is often a good sanity check.

State relations are between State classes. They describe the structural relations between States and give ways of relating one State to the veracity of another. *Strengthens* describes how one state increases the importance of, or our belief in, another. It means that the source of the relation applies a net positive influence on the target. On the other hand,

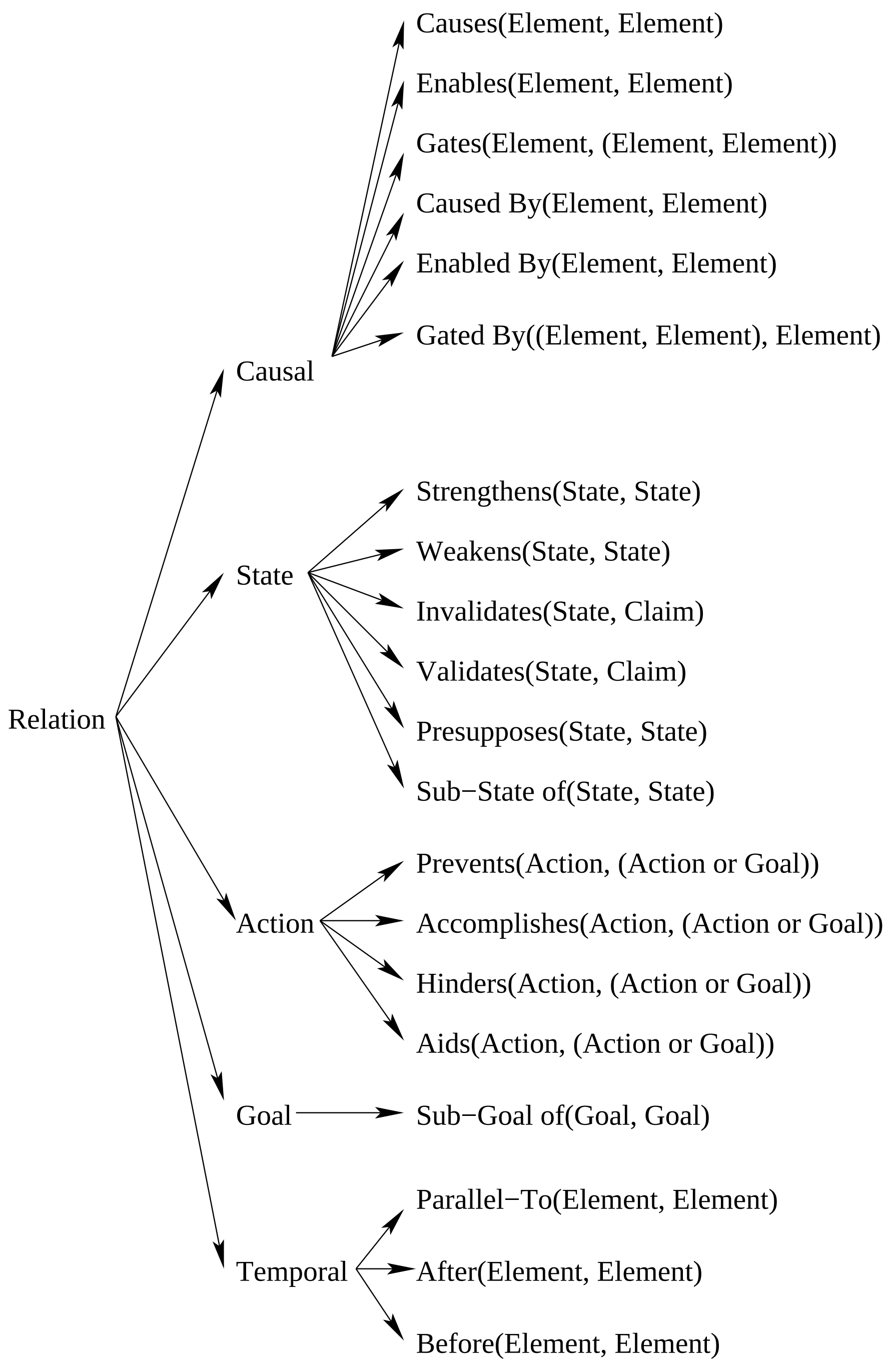

Relation
Causal
Causes(Element, Element)
Enables(Element, Element)
Gates(Element, (Element, Element))
Caused By(Element, Element)
Enabled By(Element, Element)
Gated By((Element, Element), Element)
State
Strengthens(State, State)
Weakens(State, State)
Invalidates(State, Claim)
Validates(State, Claim)
Presupposes(State, State)
Sub−State of(State, State)
Action
Prevents(Action, (Action or Goal))
Accomplishes(Action, (Action or Goal))
Hinders(Action, (Action or Goal))
Aids(Action, (Action or Goal))
Goal
Sub−Goal of(Goal, Goal)
Temporal
Parallel−To(Element, Element)
After(Element, Element)
Before(Element, Element)

Figure 2-4: The full set of relations in the ADRL.

weakens describes the opposite condition, where a state decreases the importance of, or our belief in, another. Weakening represents a net negative relationship between two states. *Validates* and *invalidates* have a similar positive and negative relation to Claims, except that these relations are more clearcut. Validation means that a State has provided us with the information needed to verify the target Claim. Invalidation, again, is the opposite, where a State has provided the information necessary to declare a Claim to be false. *Presupposes* is a special causal relation between States, where one State relies on another in order to be true. This is nearly identical to the DRL version of the relation. Finally *sub-state of* allows a user to group States with subordinate relations. These groups can be combined informally into "mega-states", which encapsulate a set of information about a given topic. For example, if we had the State "Dr. Evil is evil" as an overall mega-state, the various pieces of evidence supporting this assertion would be sub-states of the primary assertion.

Action relations are between Actions and other Elements. They describe the effect that a source Action can have these Elements. *Prevents* describes how an Action can prevent a Goal or another Action from occurring. For example, the Action "John closes the door" would prevent the Action "Joe walks through the door" or the Goal "Anna wants to bring the apples through the door." *Accomplishes* is the opposite, telling of how an Action explicitly leads to a Goal state or achieves the same end as another Action. For example, "John opens the door" would accomplish the same Goal as the Action "Bill opens the door." *Hinders* is a less severe form of prevents, where an Action can make another Action or Goal difficult to perform or accomplish. For example, the action "John locks the door" would make the Action "Joe opens the door" and any accompanying Goals more difficult to achieve. *Aids* is the opposite, describing how an Action can facilitate the accomplishment of another Action or Goal.

Because Goals are our end states and have not yet been achieved, there are very few ways to describe the dynamics between them. One relation among goals is *sub-goal of*, which describes how certain sets of Goals can be grouped. Just as in the subordinate relation among States, sub-goal of allows Goals to be grouped into larger "mega-goals" that can only be accomplished after the subordinates. The relation between subordinates and the mega-goal also need not be necessary (that is, one may not need to accomplish all of the sub-goals in order to accomplish the mega-goal), but this is a choice that is up to the user of the representation.

Temporal relations describe constraints on the temporal sequencing between Elements. *Parallel-to* tells the system that two elements have occurred or must occur in parallel. *After* describes how the source Element occurs after the target and *before* describes how the source Element occurs before the target. This sort of temporal sequencing is often unnecessary for describing decision rationales, but the existence of these relations allows for an added degree of flexibility in structuring the rationale decomposition.

2.3.3 Representing Our Examples in the ADRL

Now that we have presented the ADRL, we must show that it can succeed where Lee's DRL did not. Once again, let us examine our basic example:

```
1.  Billy bought the toy.
2.  Billy owns the toy.
3.  The box does not possess a lock.
4.  The box is insecure.
```

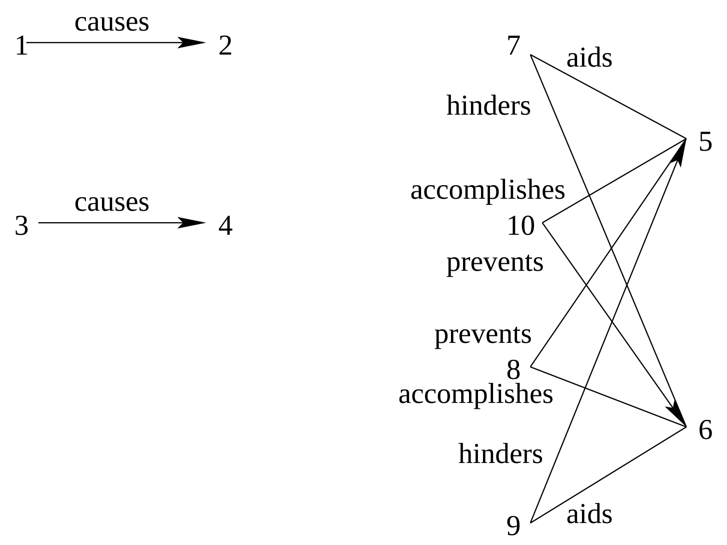


Figure 2-5: The ADRL expression of the basic example, referenced by number: 1-4 are States, 5 and 6 are Goals, 7-10 are Actions.

```
5.  Billy wants to keep the toy.
6.  Joel wants to own the toy.
7.  Billy puts the toy in his pocket.
8.  Billy gives the toy to Joel.
9.  Billy puts the toy into the box.
10. Billy sends the toy to the bank.
```

We can decompose this into States, Actions and Goals in the same way as we did for the DRL. Sentences 1-4 are States, sentences 5 and 6 are Goals and sentences 7-10 are Actions. If we wished to model the situation with a bit more complexity, we could take the perspective of Billy and say that 6 is an actor-goal for Joel, whereas before we could not do such modeling with the DRL.

Figure 2-5 shows one possible ADRL relational decomposition of the basic example. Besides the causal relationships, the main ones to take note of are the hinders, aids, prevents and accomplishes relationships between the possible Actions and the Goals. Note that many of the relations here are flexible depending on interpretation. For example, we could consider that Billy putting the toy into his pocket is secure enough that it would accomplish his goal rather than merely aid it. Rather than a weakness, this ambiguity is a strength of the representation, because it allows multiple perspectives on the same situation. One could imagine a system where analogies can only be made to Scenarios authored by a certain user who's point of view is somehow desirable.

Let us look once again at another one of our examples:

```
1.  The Earth bought the hypership.
2.  The Earth owns the hypership.
3.  The space-port does not possess a military.
4.  The space-port is insecure.
5.  The Earth wants to keep the hypership.
6.  Mars wants to own the hypership.
```

```
7.  The Earth puts the hypership in Canada.
8.  The Earth gives the hypership to Mars.
9.  The Earth puts the hypership into the space-port.
10. The Earth sends the hypership to the battle-platform.
```

Because this example is so similar to our basic example, the decomposition could be the same as the one shown in figure 2-5. This would present us with an interesting problem. How is this different from the coarseness problem that we had with the DRL? It seems that analogies between different scenarios that should not match would still be made even with this representation. One key addition with the ADRL, however, is that we are now considering the textual content. Specifically, we have a structured semantic representation that will allow us to compare structures between this Scenario and our basic example. It is the presence of this information that facilitates our ability to make analogies, not the primitive and relational structure of the language. To demonstrate this, let's look once again at the example that foiled the DRL:

```
1.  Billy bought the toy.
2.  Billy owns the toy.
3.  Joe has the fish.
4.  Joe ate the fish.
5.  Billy wants to keep the toy.
6.  Joe wants to own the car.
7.  Billy puts the toy in his pocket.
8.  Joe buys the car.
9.  Joe's mother buys the car for Joe.
10. Billy sends the toy to the bank.
```

In this case, the decomposition could, once again, look the same as in figure 2-5. The way we can avoid a false match, however, is to note that structurally the two Scenarios are not at all the same. The roles that Billy and Joe play in this example do not exactly fit the roles that Billy and Joel play in the basic example. A good semantic representation of the sentences, along with a matcher that recognized the structural differences, would be able to tell these Scenarios apart.

Now, look again at our abbreviated version of the basic example:

```
1. Billy bought the toy.
2. Billy owns the toy.
3. The box is insecure.
4. Billy wants to keep the toy.
5. Billy puts the toy in his pocket.
6. Billy puts the toy into the box.
```

Previously, the DRL couldn't precisely match this with the first story because the relational structure was different. If we consider the internal structure of the textual content, however, we can make analogies between the agents of the two stories, and their Goals and Actions. These analogies would allow a computational system to note that the two stories are fundamentally similar and signal a match.

The lesson here is that, even with the great improvements we have made to the DRL, the significant change is the addition of the information we receive from the semantic content of the text associated with the Scenario. In the next chapter we will examine the representation that I selected for this information and see how it facilitates meaningful analogies between Scenarios.

2.4 A Limited Implementation of ADRL

I have implemented a subset of the ADRL for future use in a blunder-stopping application. The subset does not deal with uncertain information, and therefore does not implement Claims or any of the relations involving them. In the implementation, Scenario objects can hold any number of State, Action or Goal objects. Each of these objects, in turn, has pointers to the other State, Action or Goal objects to which it is related based on four different categories: (causes), (caused_by), (strengthens), (weakens). I have not fully implemented the relation taxonomy of the ADRL for the sake of simplicity. One can notice that (strengthens) and (weakens) simply represent positive and negative effects respectively. Because there is no constraint in the implementation on the objects that can be targeted by the relations, the current implementation is actually more general than that prescribed by the language itself. Essentially, any Element can have a positive or negative effect on any other Element, and these effects are not made specific. A part of the implementation is an XML file format that stores Scenario information in a way easily accessible to both computer and human readers.

Below is a possible decomposition of the basic example of this chapter in terms of the language subset that I have implemented.

```
1  - State:  Billy bought the toy. causes:2
2  - State:  Billy owns the toy. caused_by:1
3  - State:  The box does not possess a lock. causes:4
4  - State:  The box is insecure. caused_by:3
5  - Goal:   Billy wants to keep the toy. weakens:6
6  - Goal:   Joel wants to own the toy. weakens:5
7  - Action: Billy puts the toy in his pocket.
8  - Action: Billy gives the toy to Joel.
9  - Action: Billy puts the toy into the box.
10 - Action: Billy sends the toy to the bank.
```

Chapter 3

Augmenting a Decision Rationale Representation with a Spatial Semantics

The key to increasing the power of the representation we have introduced is giving meaning to the textual content associated with a decision scenario. To this end, I chose to apply the work of Jackendoff. [5] Jackendoff's Lexical Conceptual Semantics (LCS) representation brings ideas in language down to a basic vocabulary of places, paths and events. His primitives are THINGs, PLACEs, PATHs, PATHELEMENTs, EVENTs (such as GO, STAY, CAUSE and LET) and STATEs (such as BE, ORIENT and EXTEND).

A number of studies provide experimental support for the assertion that abstract notions ground out in spatial understanding, particularly the work of Boroditsky. [2] For this reason, using a spatial semantics to represent the political concepts needed for the work of this thesis seemed a good choice. The next section will present salient examples of the forty-two Jackendoff frames I developed for representing sentences in the barnyard politics domain. These frames are adequate for representing many basic political concepts and focus primarily on ideas of possession, control and causation.

3.1 Representing Political Ideas with Jackendoff Frames

3.1.1 State and Causation

The first basic class of sentences that we need are those that represent state information, in the form of sentences such as "The bull is black" and "The bull is an animal." In the former case, we are characterizing the bull itself, and it is "at" the place "black" in some kind of characterization space. In the latter case, we are classifying the bull as an animal. These relations are both fairly simple. This representation was developed with the help of Patrick Winston as a part of the Bridge project. Below is a diagram of the Jackendoff frame for the sentence "The bull is black."

```
Sentence: ''The bull is black.''

| characterization
|
|    |
|    | Thing animal bull
|
|    |
|    | Thing color black
|
| Thing verb is
```

The next important class of sentences deals with ideas of causation. These causal structures are basic to representing sentences as Jackendoff frames, as many other ideas can be thought of as types of causation. This frame is central to the work of Jackendoff, and this particular conception of it was developed by Patrick Winston as a part of the Bridge project. Below is the Jackendoff frame diagram for the sentence "Billy kicked the ball to the tree." This sentence represents the case where Billy has caused the ball to go along a path to the tree by kicking it.

```
Sentence: ‘‘Billy kicked the ball to the tree.’’

| cause
|
|    |
|    | Thing agent Billy
|
|    | go
|    |   |
|    |   | Thing ball
|    |
|    |   | to
|    |   |    |
|    |   |    | Thing tree
|    |   |
|    |
|
| Thing verb kick
```

3.1.2 Possession and Control

With these two structures as a foundation, we can now attempt to represent sentences dealing with more abstract notions. Some of the most important ideas in political situations are those of possession and control. If an agent (not limited to a human being) possesses an object, then that object is at their location. Structurally, this involves a STATE rather than an EVENT, using a BE primitive. An example of possession is the sentence "Billy owns the ring." The following is a diagram of the form of this structure as a Jackendoff frame.

```
Sentence: ‘‘Billy owns the ring.’’

| be
|
|   |
|   | Thing ring
|
|   | at
|   |     |
|   |     | Thing agent Billy
|   |
|
| Thing verb own
```

Control is expressed in exactly the same way, except now the object or agent that is being controlled is at the controller in some sort of control space. One can think of this as the thing being controlled being somehow within the "sphere of influence" of the controlling agent. An example of control is "The US controls the capitol city." The following is a diagram of this structure.

```
Sentence: ‘‘The US controls the capitol city.’’

| be
|
|   |
|   |
|   | Thing ‘‘capitol city’’
|
|   | at
|   |     |
|   |     | Thing agent Billy
|   |
|
| Thing verb control
```

Possession and control also have causative forms, where, for example, an agent can receive an object or give an object. In these situations, the structures change to reflect the movement of said object along a trajectory toward the agent receiving it, either in concrete space (with possession) or abstract space (with control). The following is a diagram of the sentence "Billy caused Suzy to control Jim."

```
Sentence: ‘‘Billy caused Suzy to control Jim.’’

| cause
|
|   |
|   |
|   | Thing agent Billy
|
|   | be
|   |
|   | |
|   | |
|   | | Thing agent Jim
|   |
|   | | at
|   | |     |
|   | |     | Thing agent Suzy
|   | |
|   |
|   | Thing verb control
|
| Thing verb cause
```

3.1.3 Desire

The idea of desire is expressed similarly to the cause-possession relationship. If an agent wants an object, then he wants that object to move along a trajectory from its current location to his current location. This can be expressed by the same structure as the agent receiving the object, except that the verb is WANTS rather than RECEIVES. It is interesting that wanting something and actually receiving it are structurally similar, but given our use of these concepts in language and our conception of how they work in physical space, this is not an unexpected result. The following is a frame diagram of the sentence "Billy wants the ball."

```
Sentence: ‘‘Billy wants the ball.’’

| cause
|
|   |
|   |
|   | Thing agent Billy
|
|
|   | go
|   |     |
|   |     |
|   |     | Thing ball
|   |
|   |     | to
|   |     |     |
|   |     |     | Thing agent Billy
|   |     |
|   |
|
| Thing verb want
```

3.1.4 Encouragement and Discouragement

The ideas of encouragement and discouragement are crucial to concepts in decision analysis. Take, for example, the sentence "Billy encourages Jim." Implicit in this sentence is an action, so it can be expressed as "Billy encourages Bobby (to) (action)." The relation for discourage can be expressed similarly. Given this way of looking at the sentence, the structures for encouragement and discouragement are clearly causal relations, where one agent causes another to either perform an action or not perform an action. The actual level of causation (whether they supported the action, or actually made the actor follow through and complete the action, for example) depends on the specific verb used in the sentence.

3.1.5 Transitions of State

Transitions of state are also important in expressing a variety of situations. This encompasses ideas like creation, destruction, and transformation. Structurally, these can be interpreted via causation. Take, for example, the sentence "Billy made a castle." In this case, Billy caused a sand castle to appear in some unspecified location. Destruction can be viewed similarly. Take, as an example, "Billy destroyed the castle." Here, Billy caused the castle to go from a state of existence (or cohesiveness) to a state of non-existence (or rubble). The specifics of the representation are open to some interpretation and should generally depend on previous observed examples of a similar type. The following is a diagram of the structure for "Billy destroyed the castle."

```
Sentence: ''Billy destroyed the castle.''

| cause
|
|    |
|    | Thing agent Billy
|
|    | go
|    |   |
|    |   | Thing castle
|    |
|    |   | to
|    |   |    |
|    |   |    | Thing nothing
|    |
|
| Thing verb destroy
```

Transformation issues are handled slightly differently. Take the sentence "Iraq became a democracy from a dictatorship." This can be represented as Iraq moving from a dictatorship to a democracy in an abstract political space. It is thus a simple GO type of relation. If a cause for this change was expressed in the sentence, we could easily change the representation to deal with this and make it a causative GO, as in the cases handled for creation and destruction. The following is a diagram of the structure for "Iraq became a democracy from a dictatorship."

```
Sentence: ''Iraq became a democracy from a dictatorship.''

| go
|   |
|   | Thing Iraq
```

```
|
|   | from
|   |    |
|   |    | Thing dictatorship
|
|   | to
|   |    |
|   |    | Thing democracy
|
```

3.1.6 Coercion and Permittance

The structures for sentences that deal with ideas of coercion and permittance are closely related to those structures that deal with issues of control. Take, as an example, the sentence "Ruby forced Lucas to run." This is represented as a causative GO, where the agent Ruby caused the agent Lucas to perform the action of running. If we take the sentence "Ruby coerced Lucas to run" it would be represented similarly. Both sentences have an implicit part that may or may not be represented depending on the circumstance, which deals with exactly how the coercion took place: "Ruby forced Lucas to run (by) (action)." This is a more complex causative situation, where a completed action or series of actions leads to an agent performing another action. The representation for this, however, is still similar to the other forms of causative GO that we have seen so far. The following is a diagram of the structure for the sentence "Sue forced Jim to dance," which illustrates this.

```
Sentence: ‘‘Sue forced Jim to dance.’’

| cause
|
|    |
|    | Thing agent Sue
|
|    | go
|    |   |
|    |   | Thing agent Jim
|    |
|    |   | to
|    |   |    |
|    |   |    | Thing verb dance
|    |   |
|    |
|
| Thing verb force
```

Appendix B presents all 42 of the structures devised to represent the political dynamics of the barnyard politics world. Armed with these structures, we are now able to parse the sentences in our previous examples into a spatial representation to enable the creation of analogies between Scenarios.

3.2 An Implementation of Jackendoff Frames Using the Bridge System

My implementation of the ADRL relies on an implementation of Jackendoff frames in the Bridge system, developed under the supervision of Professor Patrick Winston by researchers in the MIT Artificial Intelligence laboratory. Bridge implements a limited form the Jackendoff frames I developed via the BridgeSpeak parser. This parser provided a convenient way to convert sentences into frame objects that were readily used for computations within Scenarios. Appendix C presents a detailed overview of the features of Bridge used by my implementation. This information is crucial to understanding the implementation of the matcher presented in Chapter 4.

Chapter 4

A Matcher for Facilitating Analogies Between Scenarios

In order to develop a matcher based on my ADRL representation, I needed a good way of structuring scenarios before the matching process. I developed an intermediate graph representation for Scenarios that brings together the internals of Jackendoff frames and the relations between Elements. This representation was motivated by two engineering goals. First, I wanted to simplify the matcher by removing some of the recursive aspects of Jackendoff frames and bringing them out into a flat graph representation. Second, I wanted to have a uniform representation for matching that incorporated relations within Jackendoff frames and relations between Elements as equally important links in a graph structure. In the subsequent sections I will present my graph representation as well as the graph conversion algorithm. I will then present the implementation details of the matcher.

4.1 Converting Jackendoff Frames to Flat-Graph Structures

The first step in creating a graph structures from our scenario representation is converting each Jackendoff frame into what can be called a "flat-graph". A flat-graph is a specialized graph representation of a Jackendoff frame. By linking flat-graphs together, we can make fully formed graphs of our scenarios in order to use them with the matcher. Each element within a flat-graph is a Thing and each edge has a label that is also a Thing (usually a Relation). The elements within a flat-graph are references to the actual objects used in the Jackendoff frame from which it was generated.

One of the difficulties in this domain is the problem of recursion. Since we would like to flatten the recursive structures of a Jackendoff frame, we need a way to bring them out and delineate them. As such, I have devised a special way to represent the constituent parts of a frame in a flat-graph. Figure 4-1 shows what the structure for the sentence "Billy ran to the ball" looks like as a flat-graph.

There are several things to notice here. First of all, notice the "START" and "END" nodes. These exist to demarcate the start and end of one particular sentence (one trajectory space). All edges that have no special significance are labeled with "connected", and we can see this from the connection between the "START" node and the "ladderStart" node. Since each sentence can have several phrases, we demarcate these with the "ladderStart" and "ladderEnd" nodes. Subsequent phrases are connected from the previous "ladderEnd" to the next "ladderStart". Notice figure 4-2, of the flat-graph for the sentence "Billy ran to

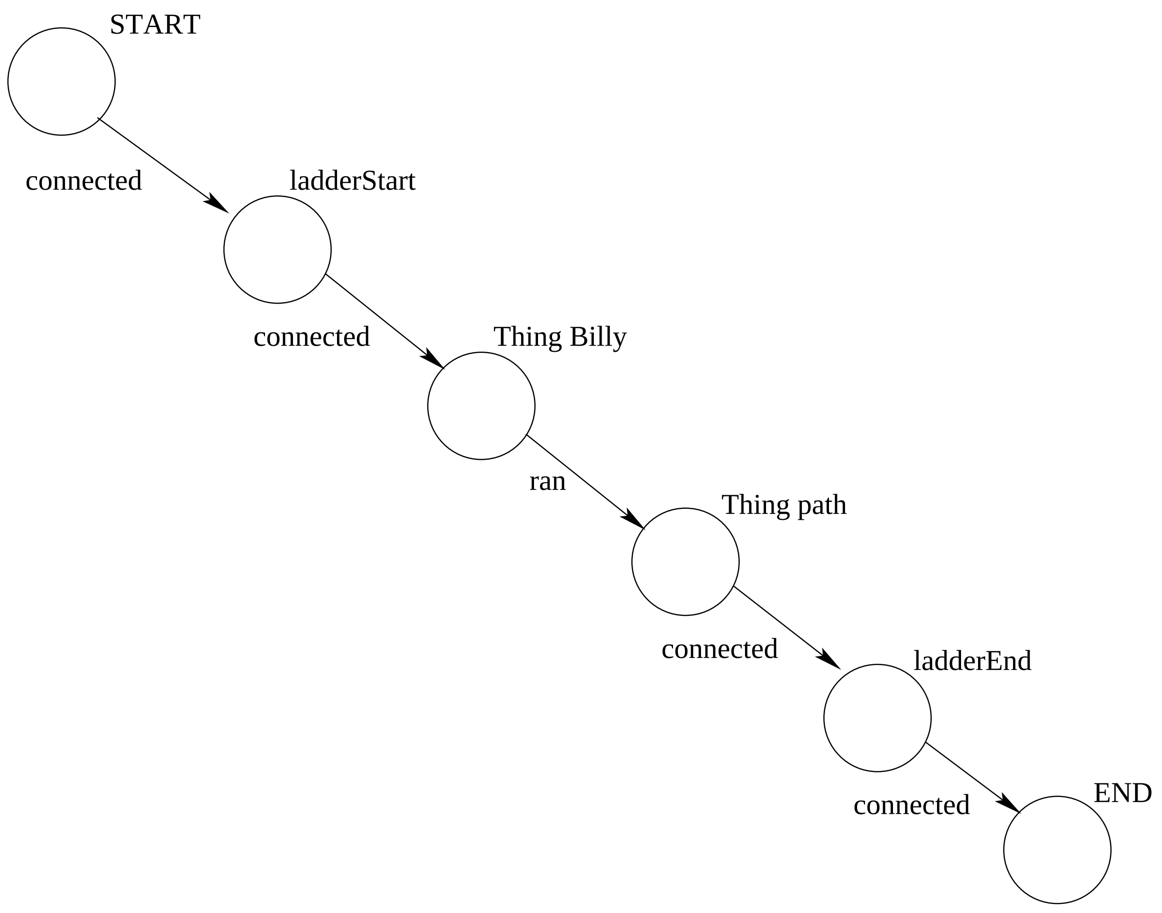


Figure 4-1: The flat-graph for “Billy ran to the ball.”

the tree because Billy ate the pie." Here, we can see two ladders, connected by a "caused" relation. Note also that the ladders are in the opposite order from that presented in the sentence, since the sentence presents the phrases in the wrong temporal order. This is the same order generated by the BridgeSpeak parser when it generates the Jackendoff frames for this sentence. When there is no causal link between sentences (such as in the phrase "Billy walked and then Billy ran") the ladder nodes are connected by a "connected" relation.

Note also the nondescript "path" element in both graphs. This represents a compound Thing object containing the specific information for the path involved in the given sentence. Therefore, the "path" in the first graph would be a path object representing "to the tree."

4.2 Combining Flat-Graphs into Full Scenario Graphs

Now that we have a graph representation for the individual structures, how do we create larger graphs to encompass our scenarios? We can look at this as the problem of combining many flat-graphs into one larger graph structure. We must also take into account the interrelations between sentences (and thus flat-graphs) that the ADRL enables users to describe.

One crucial problem with combining the graphs is the shared memory architecture used by the BridgeSpeak parser. The shared memory means that subsequent parses containing the same words will return references to the same objects. This is a problem for the graph conversion process because the flat-graphs will then contain references to the same object, even though we mean to have distinct objects. When we try to combine these graphs in a simple manner (by adding the nodes and edges of each graph to a common graph) we will be faced with the problem of being unable to determine the exact structures used to create the graph. In other words, the process will not be reversible. We would like the conversion to be reversible since we would hope to recover a full scenario from a graph once we have finished matching.

Given this problem, I devised a solution that calls for the cloning of nodes that are pointing to the same object. These clones, in turn, are connected by an edge marked "SAME" so that they are identifiable as coming from the same object. This modification allows each structure to be reversed back into a Jackendoff frame. It also allows for fairly quick retrieval of all "SAME" nodes, so that the structure of identical nodes can be compared between graphs relatively easily. This will prove important to the matching process. Figure 4-3 shows two flat-graphs with connected "SAME" nodes.

Another issue with combining the graphs is identifying the flat-graphs for later retrieval. We can do this easily by tagging each sentence in a scenario with a unique identification number and adding this number onto the start and end nodes of the graph. Using the keywords "START" and "END", along with the unique ID, each flat-graph can be retrieved from either its start or end node. In addition to this, we also mark the START and END nodes with a tag that identifies the type of element this flat-graph represents, either State, Action, Goal or Element.

Once the structures are combined, links must be made between the flat-graphs to show how they are related. Figure 4-4 shows the different types of relations between flat-graphs and how flat-graphs connect to each other.

Flat-graphs are connected from the END of one to the START of another, as we can see. The possible relations are "CONNECTED", "CAUSES", "CAUSED_BY", "WEAKENS" and "STRENGTHENS". These relations come directly from the scenario representation we

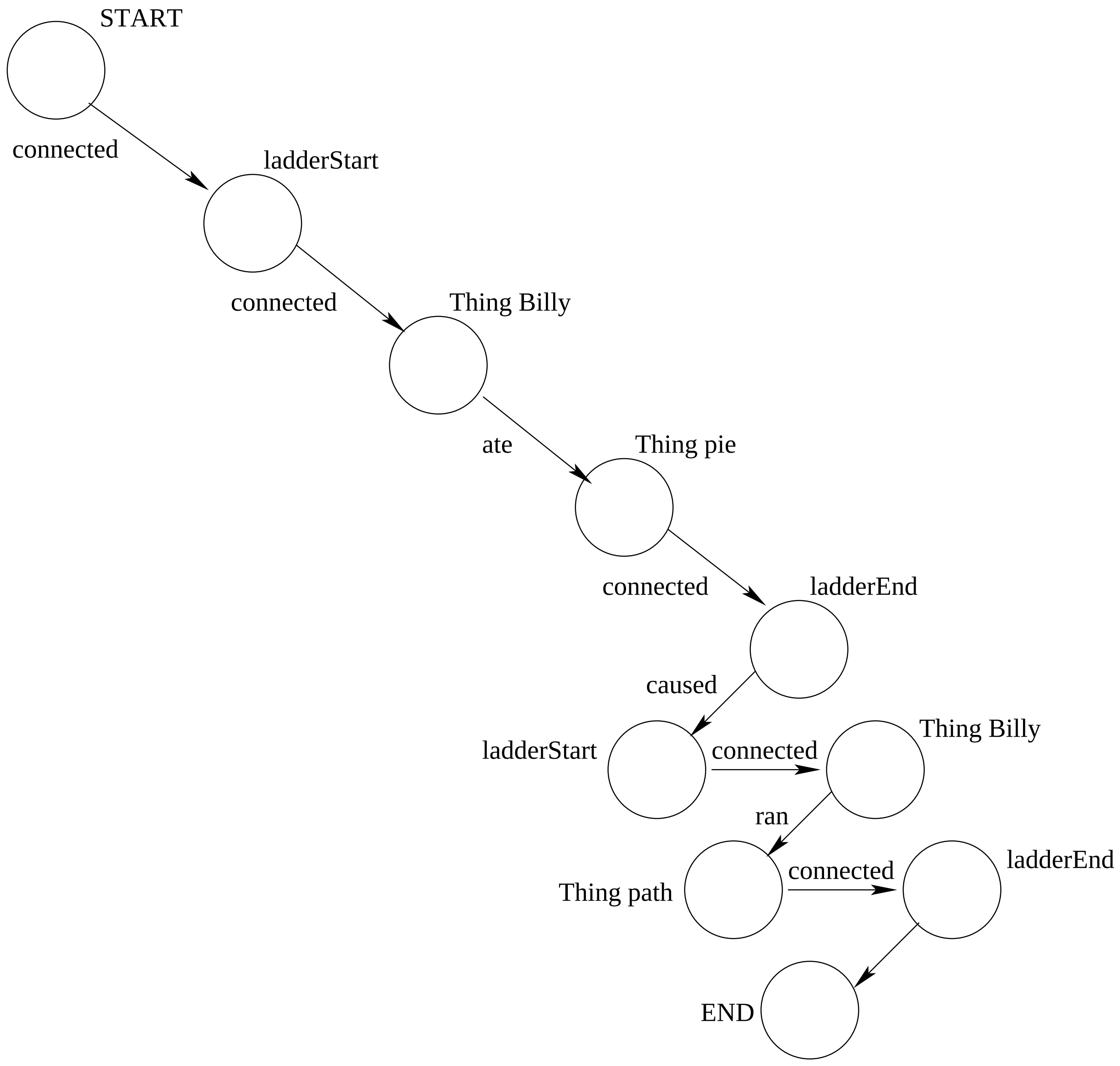


Figure 4-2: The flat-graph for "Billy ran to the tree because Billy ate the pie."

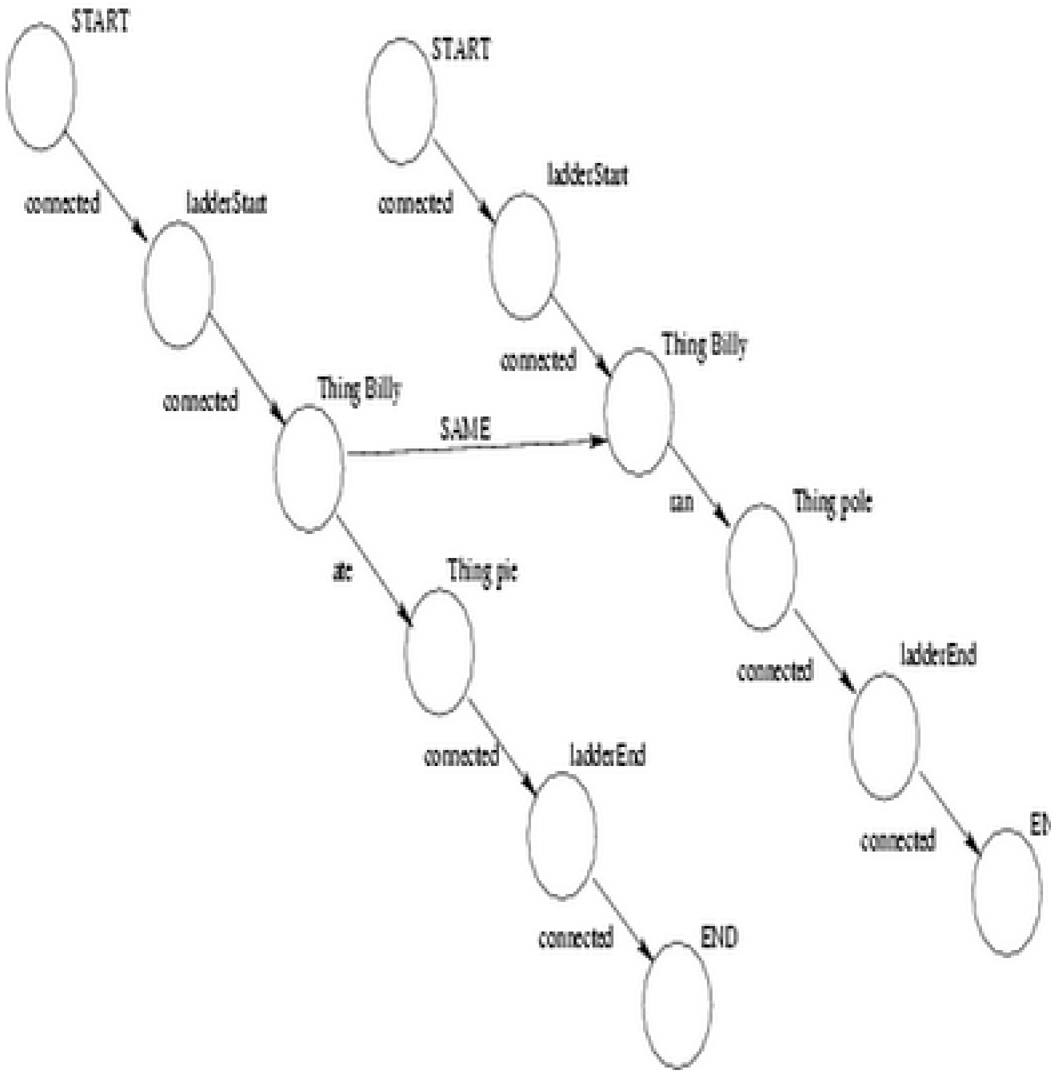


Figure 4-3: The flat-graphs for “Billy ate the pie” and “Billy ran to the pole” connected by a “SAME” relation.

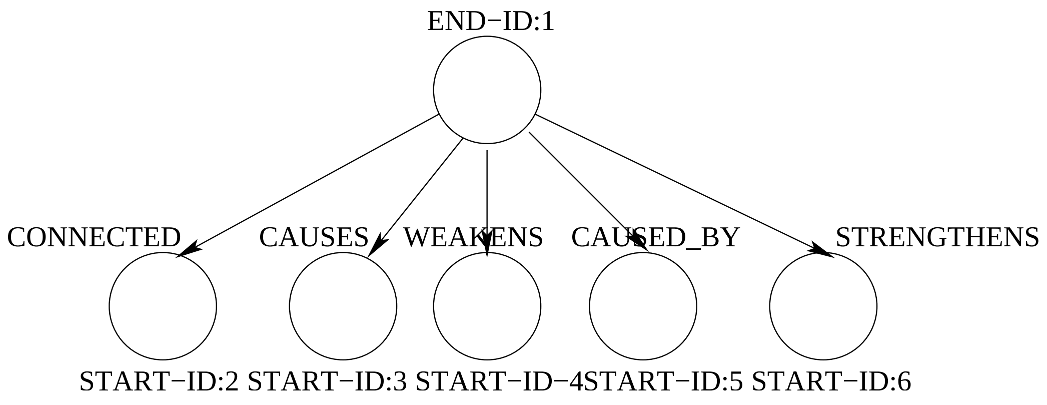


Figure 4-4: How flat-graphs connect to each other and the possible relations between them.

discussed earlier, save for "CONNECTED". "CONNECTED" is a special relation connecting two structures that have been entered in sequence. Since the explicit order of sentences in a story is often important, the "CONNECTED" relation allows us to specify this in the graph. It also allows for a traversal of the entire graph starting from the very first START node, meaning that every flat-graph can somehow be reached from every other flat-graph.

The final step in the conversion process is to create a "GRAPH-START" node and connect this to the START node of the first flat-graph of the scenario (which in turn corresponds to the first Jackendoff frame and thus the first sentence entered by the user). Similarly we make a "GRAPH-END" node and connect this to the END node of the last flat-graph of the scenario.

Now that we have this conversion process, we can take the scenario structure for our basic example from Chapter 2 and convert it to a fully formed graph. The graph for this scenario is rather large (74 nodes and 195 edges) so figure 4-5 shows only the first two sentences after the conversion process.

4.3 Performance Analysis of the Graph Conversion Algorithm

The graph conversion process runs in $o(n)$ time, where n is the number of objects in the structure being converted. This is easy to see, because as the algorithm iterates through the structure, it adds one node or edge to the graph. Depending on the number of SAME nodes and inter-structure relations, extra work will have to be done, but each of these factors is asymptotically less than the number of objects in all of the structures combined. To be more specific, if we say that the number of SAME nodes is m and the number of outside relations is r, then the total performance characteristic of the conversion algorithm is $o(n+m+r)$, where m and r are asymptotically less than n. From this, we can also note that the number of nodes and edges in the graph is also $o(n)$.

4.4 Thing Matchers

Given that Thing objects and their various subclasses are the substrate of our representation, we need a robust set of matchers for Things before we can build a scenario matcher. This means matchers that handle simple Things, Derivatives, Relations and Sequences. We would also like matchers that discern exact matches as well as *soft* matches. What is a soft match? Things are typed by the threads they contain. We definite a soft match as a match where the two Things share something common in the thread hierarchies of their prime threads. Note also that all matches should return an object. An exact match returns a copy of the Things matched while a soft match returns a Thing representing the "least" in common among the two Things being matched. Take the following example.

```
Thing -> Animal -> Agent -> BillyBob
Thing -> Animal -> Kanga
```

If we were to match the Thing "BillyBob" with the Thing "Kanga," an exact matcher should return no match. A soft matcher, however, should return the a Thing object representing the highest common class on the prime thread. In this case, that would be "Animal." Why do we need a soft matcher? Soft matches allow for very fast one-shot generalizations

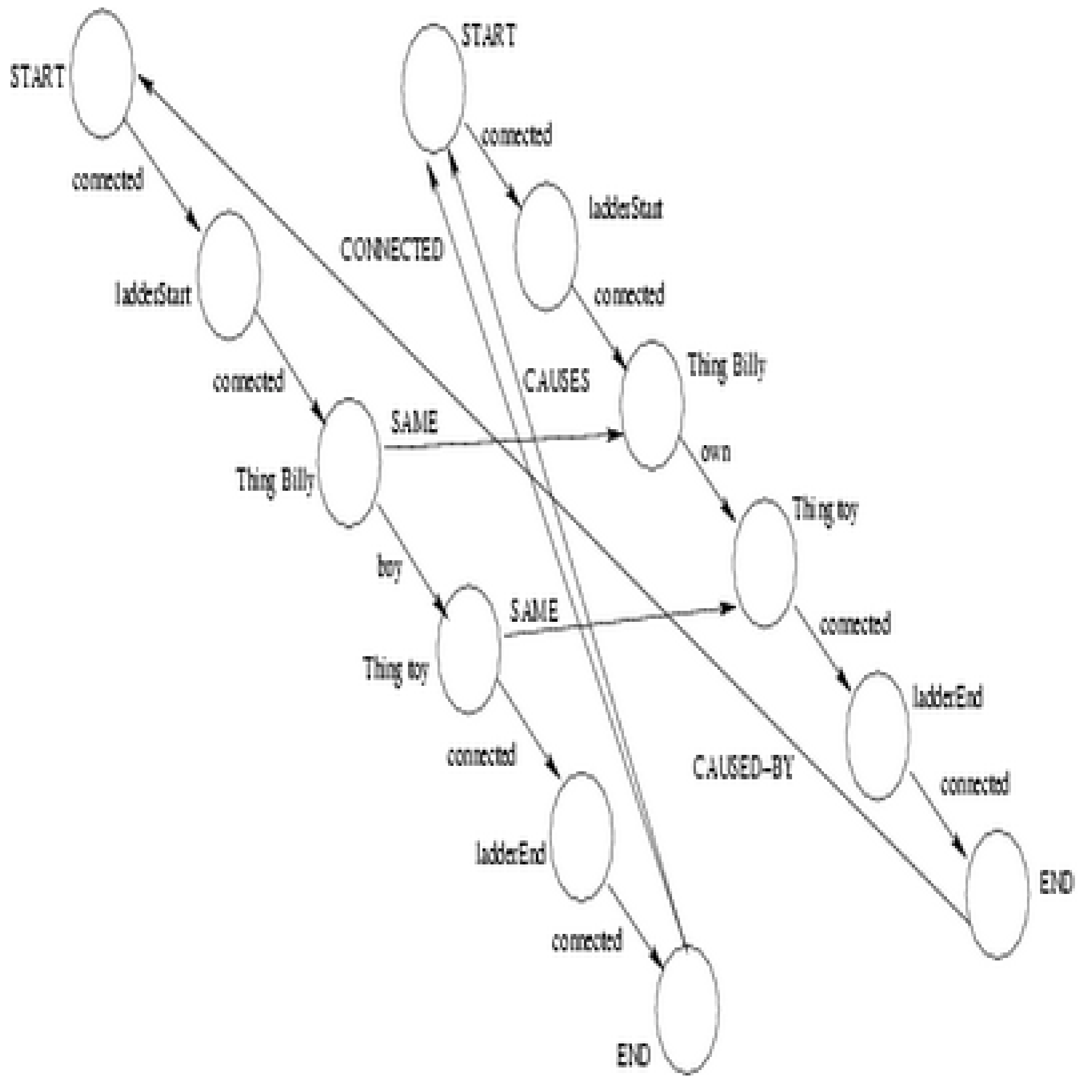


Figure 4-5: The flat-graph for the first two sentences of our basic example scenario: “Billy bought the toy” and “Billy owns the toy.”

and also reveal the salient similarities between objects. Although soft matches are keyed on the prime threads, once a match has been found on the prime thread, the soft matching process also propagates to the other threads in the bundle. Take the following example.

```
Bundle for Thing BillyBob:
Prime: Thing -> Animal -> Agent -> BillyBob
Color: pink
Description: tall

Bundle for Thing Kanga:
Prime: Thing -> Animal -> Kanga
Color: brown
Description: tall
```

The prime threads would match as before, giving us the knowledge that the common class is Animal. The color threads, however, don't match at all. In this case, we drop this thread entirely from the match, since it has proved to be irrelevant. Therefore, the common thing returned would be the following:

```
Bundle for matched Thing Animal:
Prime: Thing -> Animal
Description: tall
```

The different classes of Thing are handled in similar ways. For soft matches on Derivative, the subjects are soft matched in addition to the threads attached to the actual Derivative. For Relations, both the subject and object are soft matched. In Sequences, the soft matcher attempts to soft match each element of the Sequence in order, truncating the match Sequence at the last place that a soft match was possible.

Although the soft matchers are a powerful tool, they are too blunt. Since most things share a common base category on their threads (usually Thing), many matches return positive, but trivial, results. Because of this I also created several matchers that perform a mixture of soft and exact matching behaviors. For example, I created a Relation matcher that matches the actual Relation exactly, but performs soft matches on the subject and object. This matcher in particular proved to be very useful in the scenario matching process.

4.5 The Alignment Issue and Matching Flat-Graphs

Looking at our scenario structure, one assumption that seems reasonable is that the sentences are aligned, to some extent, by the order in which they are presented. It is quite possible, however, that the user would enter sentences in a haphazard way or in a way that would thwart match attempts if the assumption of alignment was made. Therefore, another way to deal with the issue must be found.

The most logical approach is to begin by dealing with the two scenarios we are trying to match differently. Let us call the scenario we are trying to match against the "precedent" and the scenario we are matching it to the "situation." We can define the precedent as *the minimal set of sentences we need to match in order to declare success.* By this I mean that every Element in the precedent must be matched in order to declare a full positive match. If there is ever a case where we believe the situation should match and it doesn't because of this assumption, we can merely treat the situation as the precedent and visa versa.

Given this assumption, we can now take the approach of trying to find matches for each Element in the precedent individually; we can take each flat-graph in the precedent and try to find all flat-graphs in the situation that could possibly match it. In order to accomplish this, we can utilize the Thing matchers that we developed to match up each node in the precedent flat-graph to a node in the situation flat-graph (taking into account START, ladderStart, ladderEnd and END nodes).

The matching process works as follows. The set of START nodes in the precedent is identified, and for each of these start nodes a list is created. This list holds all of the nodes and edges between START and END for this flat-graph, in order. Thus, if we had (node)Billy connects to (node)tree with (edge)run, the list would contain (Billy), (run) and (tree) in that order. The list also contains ladderStart and ladderEnd nodes as they appear, since a structure can have arbitrarily many phrases. A similar process takes place with the flat-graphs of the precedent.

With these lists, matching now becomes a process of comparing the elements in the lists and determining if they match. This comparison is facilitated by the Thing matchers. If simple Thing objects are encountered, they are "soft" matched. If edges that are Relations are encountered, however, they are exact matched by type and soft matched by subject and object. Why is this? All of the edges in the graph turn out to be Relations, since the graph conversion process turns Relations into edges (in most cases, unless a Relation is the target of another overarching Relation that is already an edge). In the Bridge system, the Relations generated by the BridgeSpeak parser have very sparse hierarchical information. Verbs such as run, walk, talk, push, kill and attack all share the common base type of "event". Therefore, if soft matched, most Relations would end up matching, creating many trivial matches. If a mismatch occurs at any point in the list matching process, the process ends and attempts to find a match for the precedent list with another list in the situation that has not yet been explored. Note also that the set of lists the matcher looks at is not the entire set of lists in the situation. The matcher will only examine flat-graphs in the situation that meet two criteria. First, they must not have been selected as a best match for another structure in the precedent. Second, they must represent a scenario element that is of the same class as the one the matcher is attempting to find a match for. Thus, States cannot match Goals or Actions and so on.

One final issue with matching the lists is dealing with differences in length. In keeping with the principle that the precedent represents the minimal set of items that needs to be matched, a list in the situation could match even if it is not exactly the same length as the precedent list. Take the following example.

```
Precedent: ladderStart -> Billy -> ran -> tree -> ladderEnd
Situation: ladderStart -> Billy -> ran -> tree -> ladderEnd -> caused ->
ladderStart -> Bobby -> flew -> path (to the pole) -> ladderEnd
```

In this example, the lists would match, although the common match returned would only be the following.

```
Match: ladderStart -> Billy -> ran -> tree -> ladderEnd
```

Clearly, in this matching process, it is quite likely that many different lists will end up matching. Therefore, it is crucial that we have an effective algorithm for choosing the best match among this set of matches.

4.6 Finding the Best Match

Once we have a set of possible matches, we can select the best match. How do we define the best match? We must set some criteria for this evaluation before it can be made. There are three main principles that we would like a "best" match to embody.

- Matches should be as specific as possible, that is, have elements as close in type to those of the precedent and situation as possible.
- Matches should be as close in length to the length of the precedent flat-graph structure as possible.
- The "SAME" structure of the match should be as close to that of the precedent as possible.

As an illustration, consider the following example, condensed for clarity.

```
Precedent List:
Billy -> ran -> pole

Other lists with ``Billy'' in the precedent:
Billy -> ate -> pie
Joe -> hit -> Billy

Situation Lists:
Joel -> ran -> tree
Joel -> ate -> food
John -> ran -> pole
John -> hit -> Joel

Possible matches:
Joel -> ran -> tree
John -> ran -> pole
```

In this case, which match should we select? The target "pole" is clearly more specific a match than "tree" so on the surface, one might wish to select (John → ran → pole). When we look, however, at the "SAME" structures (implied here by the nodes having the same name) we see that, in fact, (Joel → ran → tree) is a much better match, because the Joel SAME nodes in the situation are involved in structures that are similar to those that the Billy SAME nodes are involved with in the precedent.

How are we to weigh these three criteria of length, specificity and structural alignment? I chose to make aligned structures the key measure. Next in importance is length and finally specificity. The algorithm works as follows. For every possible match, the SAME structures are compared in a shallow manner. That is, the number of SAME nodes is discovered and the number of relations that these SAME nodes have in common with the SAME nodes of the precedent item are also recorded. By default, the first match becomes the best match, and these values are stored. The length is also stored, as well as a numerical measure of similarity between the elements of the lists, where each hierarchy difference increments a "difference count" by one. These number are calculated for each possible match in the situation, and weighted by importance through logic (the SAME value will decide unless they are equal, in which case the length decides unless they are equal in which case the similarity measure decides).

After iterating through this algorithm, a best match is selected and returned. This best match represents the best possible soft match between the current flat-graph of the precedent and a flat-graph in the situation. Once the match is returned, it is added to a return graph which contains the actual match between the two graphs. Also, the Elements in the situation and precedent that have been matched are marked with "MATCHED" on their description threads, so that they will not be matched again. The overall algorithm continues until either all flat-graphs of the precedent are matched, or one Element is found such that there are no possible matches for that Element. In the latter case, the algorithm will return a null value. In the former, it will return the common match graph between the two scenarios.

4.7 Performance Analysis of the Matcher

We will show that the matcher has good performance characteristics given its scope. Let us consider n, the number of nodes in the precedent graph and m, the number of nodes in the situation graph. Initially, we examine each node in the precedent $o(1)$ times in order to build the lists for each flat-graph. This gives us an initial $o(n)$ computations. Next, we build the same lists for the situation, which gives us $o(m)$ units of computation.

With the lists built, we must now find all possible matches for the lists in the precedent. What this means in terms of the number of lists, in the worst case, is that every list in the precedent will get compared to every list in the situation. Thus, in terms of the number of nodes in both graphs, we can say that the total amount of computation involved in finding all possible matches for each list in the precedent is $o(nm)$. There are constant factors involved for specific cases, but in the worst case, this will be the performance of the algorithm.

Finally, we must find the best match. In the worst case, each of the lists in the precedent will match all of the lists in the situation. Therefore, we must examine all lists in the situation for each list in the precedent once again. In terms of nodes, this once again gives us $o(nm)$ computations performed. We must also build the return graph in the worst case. This involves another $o(n)$ computations, since the size of the return graph is at worst the size of the precedent graph.

In total we have $o(n) + o(m) + o(nm) + o(nm) + o(n)$ computations, which reduces to $2(o(nm) + o(n)) + o(m)$. Considering that $o(nm)$ is the asymptotically largest term in this expression, we can say that the matcher runs in $o(nm)$, which is polynomial in the number of nodes in the two graphs.

4.8 A Simple Application Using the Matcher

In order to demonstrate the ADRL representation and the capabilities of the matcher, I have built a simple graphical application using Java Swing. The application allows a user to enter a precedent scenario and a situation scenario and then match them up. Scenarios can also be saved to files and loaded from previously saved files. Figure 4-6 is a screen shot of this application.

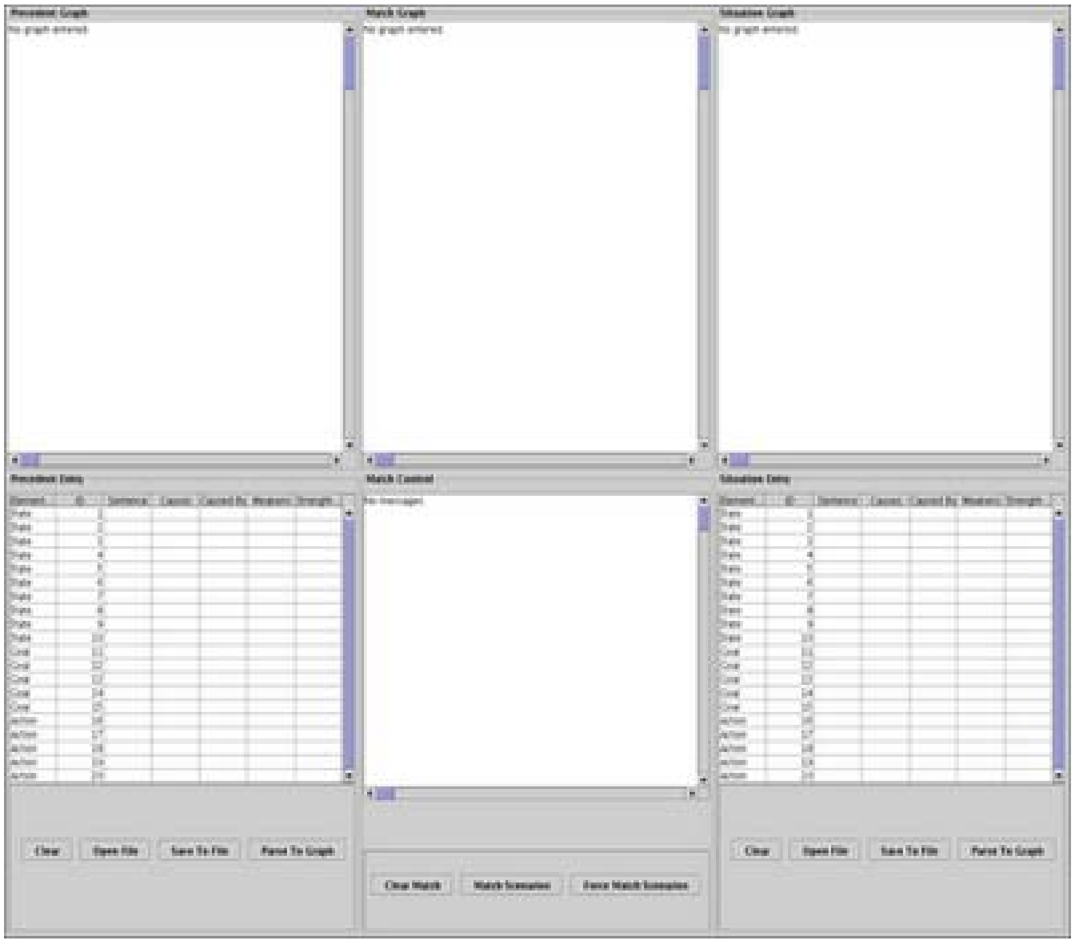


Figure 4-6: The blank matcher application. Precedents are entered and displayed on the left and Situations are entered and displayed on the right.

Users are able to enter sentences representing a decision scenario into each of the two windows. They can select the type of Element for the sentence, as well as assigning the sentence an identification number. Finally, they are able to enter a list of numbers that represent the inter-structure links in the scenario, in each of the columns “Causes”, “Caused-By”, “Weakens” and “Strengthens”. This is a rather simple but effective way of allowing a user to explicitly specify these relationships. Note also that these categories are identical to those in the limited implementation of the decision rationale language.

In the top windows, the application displays the graphs corresponding to each scenario and the graph of the match, if any. One note to make here is that the graph output is text rather than graphical. This omission is due to testing constraints, since the graphs are rather large and could not be displayed in any meaningful way without making them difficult to use as a debugging tool. The graph implementation is based on the Java HashMap class and so could very easily be plugged into a graphical output environment if such a change is required.

4.9 A Tabular Mechanism for Relating Components of a Decision Rationale

One significant aspect of the application created to test the matcher is the user interface, as can be seen in figure 4-6. With this interface, users are able to enter the components of a rationale in a fast and non-graphical way, using entries in a table with columns for each of the possible relations between the sentences. The entries are comma delimited lists of identification numbers, where each number refers to a sentence in the rationale (presumably a row in the current table).

Why is this input mechanism interesting? Although many users would prefer a visual method of scenario input, analysts in the social sciences are often presented with a great amount of textual and not visual information. It seems more natural, in some cases, to be able to specify a decision rationale as a series of sentences connected by relations where the visual layout of the scenario itself is unimportant. The interface I have created allows a user to merely enter the sentences of a story, in any particular sequence, and relate them quickly via their assigned numerical ID. This input method has the further advantage of being very flexible for large stories, since the ID number allows users to keep track of individual components of a rationale very easily. I believe that a fast textual decomposition such as this will prove to be more efficient for many analysts than the graphical environment presented in Lee’s work.

The interface is not, however, perfect. In particular, each type of relation requires a column in the table. Given that the full specification for the ADRL has twenty relations, a full implementation of the ADRL would require an input table of twenty columns. There is no obvious solution to this problem, although the degree to which this would hamper the usability of the interface is unclear and would depend on the tolerances of the user. These are issues that should be considered by those who wish to implement the ADRL and create an interface method for it. This input method also does not handle relations that do not fit the “source to target” modality very well (such as the gating relation). One solution is to use multiple delimiters in the table data. For example, if a relation required multiple arguments in its subject or object, one could wrap the arguments in braces and delimit those with commas. Although such fixes obscure the power of the interface, they are a part of the design trade-offs that implementers of the language should consider.

Chapter 5

Future Work

This work represents only a small step toward the larger goal of not only understanding human decision-making, but also of creating intelligent software to help humans in making decisions. I believe there are many ways this work can be improved upon and extended, given some effort.

5.1 Possible Applications of the Matcher and Current Implementation

The table 5.1 lists the different ways that the matcher and currently implemented ADRL representation can be used. Specifically, the table lists the kind of questions a computational system should be able to answer given two scenarios and after going through the matching process.

This list is far from comprehensive, but gives a flavor of the possible applications of the matcher. The currently implemented set of functions represents the capability of the matcher to discern the structural differences between scenarios. All of this information is obtained as a by-product of simply going through the matching process. For example, I have implemented a series of functions which takes two matching scenarios and reports the inter-structure links that they have in common. This is a simple matter of determining if the links in one structure exist in the other after they have been matched and aligned.

The other questions on the table represent more complicated applications of the matcher. They are extended applications that would use the matcher, but would also require infrastructure to store and manage a database of previous scenarios. Such a knowledge base is the key to building applications like a blunder stopper, which would have to rely on previous experience in order to determine if a possible blunder is being presented to the system by the user. For example, take the question "What is the best action to take in the current scenario based on past experience?" In order to answer this, we would need a structured knowledge base of previous scenarios and their outcomes. Using the matcher, we could find the set of scenarios which partially or completely matches the current scenario and present the user with the possible outcomes based on the most probable matches. Note also, however, that the matcher is arguably the most complicated part of this chain. Now that I have shown a matcher is possible using this representation, more complex applications based on it are much easier to conceptualize.

Possible Questions Answerable by a Decision Rationale Analogy System
*What are the elements of a specific type in each scenario? (ex. Who are the agents?)
*What are the possibly matching structures?
*What are the causal relationships?
*What are the missing inter-structure links?
*What are the common inter-structure links?
*What are the common subjects and objects?
*What are the missing structures, if any?
*What are the (states, actions, goals) in this situation?
Given this element, what could it lead to?
What are the causes of this situation?
What are the conditions in this situation supporting this (state, action, goal)?
What are the conditions in this situation hindering this (state, action, goal)?
What are the different scenarios that could match this scenario?
What is the best action to take in the current scenario based on past experience?
What actions in situations similar to this scenario have led to blunders?

Table 5.1: The types of questions answerable by systems able to make analogies between decision rationales. * indicates that my system currently implements functions that can provide these answers.

5.2 Dealing with Inter-Structure Relations: A Learning Opportunity

One aspect of the scenario graphs that is not taken into account during the matching process is the presence of inter-structure edges. As we have seen, these edges specify the explicit relations between elements of the scenario. The presence (or lack thereof) of these edges presents a unique opportunity for applications that make use of the matcher.

In the original conception of the matcher, the edges between flat-graphs were crucial to discovering a match. It is quite plausible, however, that many situations would arise where a precedent or situation was missing one or more of these links where they should occur. The cues for learning about these links are contained in the possible structural matches that may occur.

Specifically, the edges can be used to "teach" the system about the interrelations between certain types of structures. Adding causal links to a matching scenario and then matching it against a stored precedent can allow that precedent to incorporate the new links as a way of "learning" about causation. Currently, the matcher identifies missing links and notifies the user, but functions exist to incorporate missing links into a matched graph. This sort of fast learning relies in large part on the system trusting a "teacher" to give it correct information, but even so, it does represent the first steps toward a program that can learn about the scenarios with which it is working.

In the current version, the matcher implements a series of functions that discovers discrepancies between the inter-structure relations of the precedent and the situation. These functions are used to reconcile the discrepancies and return the common graph with a set of inter-structure relations that represents the common set between the two scenarios. These functions can, in future work, be used to implement the learning procedure I have described here.

5.3 Improvements and Other Future Work

With regard to improvements that can be made to this work, there are several noteworthy areas to consider. At this point the system is not able to handle very many English sentences. The current parsing mechanism can only deal with a very limited vocabulary of words and sentence structures. In order for an application of this type to be usable by an intelligence analyst, it must be able to understand a wide range of expressions. Although we have taken the first steps toward that in this work, more needs to be done. Still focusing on the average user, many usability improvements could be made to the user interface. Currently it is not what I would call "pretty" nor is it easy to use and understand. Streamlining the interface and the scenario construction process are desirable goals for future work in this area.

Another important area for further work is the matcher. One significant improvement would be a good way to reconcile the use of general templates with the use of specific scenarios in the matching process. Because the system must go through all known scenarios to find a match, and because this database could grow to be quite large, the whole process could be rather inefficient without some clever solutions involving generalized match templates. A fast and robust solution to this problem would make the matcher infinitely more usable. Another area of improvement would be in the matching process itself. Although I consider the matcher a good first step, the similarity mechanism is far from perfect. I envision a more bi-directional matcher that attempts to align the matches in both directions.

Although this can be simulated currently by running the matcher sequentially in opposite directions, a more structured and biologically motivated mechanism for this is desirable.

The ADRL representation is a powerful one, and could be used in applications for information analysis. Currently the system does not deal with uncertain information, nor does it implement my entire language of decision rationale. Dealing with uncertainty is a crucial issue, because most information that analysts deal with is not known to be entirely factual. The ability to mark information with degrees of belief and have this factor into the matching and blunder-stopping mechanisms should prove to be very rewarding. Implementing my full language of decision rationale would make a new system more complex, but would also allow users a better and more correct vocabulary for describing their scenarios. I also believe that future systems should take this work out of the "barnyard" and begin to focus on real world political and military situations. Only by dealing with larger and more realistic examples will we be able to understand the decision-making process enough to create an effective tool.

Chapter 6

Contributions

I have made several contributions with this work.

- I have presented a refinement of the decision rationale problem. My refinement focuses on the question of how we can describe decision rationales such that computer systems will be able to make analogies between decision scenarios.
- I have introduced a new language of decision rationale, based on the works of Jintae Lee and Robert Abelson called the Augmented Decision Rationale Language. This language refines Lee's work, using Abelson as an inspiration, in the following ways.
  - ADRL narrows Lee's set of primitives to five basic types based on those introduced by Abelson: Scenarios, Elements, States, Actions and Goals.
  - ADRL expands upon the relation vocabulary of the DRL to encompass more diverse relations, especially focusing on kinds of causation, such as enablement and gating, that are explored in Abelson's work.
  - ADRL augments DRL by including a semantic representation of the textual content of the decision rationale.
- I have developed a series of semantic structures, based primarily on the work of Jackendoff, for the representation of forty-two sentences typical of political situations that one would encounter in a "barnyard" political domain. These represent considerable coverage of concepts relevant to political thought.
- I have presented an implementation of a subset of the representation and developed a matcher for scenarios in this representation. This matcher stands as an existence proof that the ADRL representation can be implemented and that there are no obvious practical or theoretical barriers to making use of the representation in real-world systems.

It is my hope that this work becomes a stepping stone for future explorations in combining artificial intelligence and political science, for there are many open problems in this inter-disciplinary domain. I believe that both disciplines can learn from one another and usher in a new age of intelligent applications for automated decision and information analysis.

Appendix A

A Reference for the Augmented Decision Rationale Language (ADRL)

```
Primitives:


Element: The common base class of all primitives in the language, save
Scenarios. Elements hold a semantic representation of the sentence they
were created with.

Scenario: A group of Element objects and the relations between them.

Goal: Something that the decision-maker wants to accomplish, a desire.
Examples:
My country wants to acquire 5000 barrels of oil.
I want to stop John from leaving the company.

Actor-Goal: The goal of another actor in the scenario. The
decision-maker can model these and they can be matched up not only with
other Actor-Goals but also with the Goals of the decision-maker.
Examples:
The United States wants to stop terrorism.
Iraq wants to conquer the Middle East.

State: The current state is represented by a set of facts establishing
the current conditions in the given world. Each fact is a state of the
world. This is in contrast to the desired state, which is a set of Goals
(and Results if the decision has already been made).
Examples:
Iraq is a nation.
There are 50,000 Iraqi troops around Baghdad.

Claim: A claim is a piece of information with a prescribed veracity used
```

in making a judgment. Claims are not known to be completely true, else they would be considered States. Claims are a subclass of State.
Examples:
Iraq has weapons of mass destruction. (assumption or uncertain information)
Saddam Hussein is dead. (assumption based on taken actions)

Action: Some step that can be taken in order to make some tangible change to the state of the current ‘‘world’’.
Examples:
Attack Iraq.
Assassinate Osama Bin Laden.
Dissolve the United Nations.

Actor-Action: Some step than can be taken by another agent to make some tangible change to the state of the ‘‘world’’.
Examples:
Saddam can attack our tanks.
The Iraqi military can surrender.

Result: The result of an action in a decision process. Results can be present both if the decision has already been completed and the result is known, or if the decision-maker is presuming a result in order to construct a larger Scenario.
Example:
Given a goal of stopping terrorism and an action of attacking Iraq, an unintended result would be alienating the United Nations.

Unknown: A piece of information that the decision-maker does not know, but is implicitly important to the current decision. Unknowns exist for marking purposes, but play no role in any computations.
Example:
The number of troops currently in the Iraqi army.
The weapons of mass destruction that Saddam Hussein possesses.

Relations:

Causal Relations:

Causes: A given Element can have a causal relationship with any other element.
Caused-By: A given Element can be caused by any other element.

Example:
Rumsfeld going to Iraq (State 1) caused the Iraqi people to riot (State 2).

Enables: A given Element can enable another Element to occur.

Enabled-By: A given Element can be enabled by another Element.

Example:
John opening the door (Action 1) allows Billy to walk through the door (Action 2).

Gates: A given Element can allow another Element to lead to a third Element.
Gated-By: A set of two Elements with a sequential relationship can be allowed to occur because of a third Element.

Example:
The driver waiting outside the bank (State 1) gates the event that the robbers leave the bank (Action 1) and make a quick getaway (State 2).

Relations between State objects:

Strengthens: A given State can strengthen another State. The strengthens relation can also hinge on the truth of the supporting State, if that State is a Claim.
Example:
If Iraq as a nuclear weapon (State 1) then Iraq has weapons of mass destruction (State 2).

Weakens: A given State can weaken another State. This relation again hinges the truth of the weakening State, if that State is a Claim.
Example:
If Iraq is a poor country with no weapons (State 1), then it does not have weapons of mass destruction (negated State 2).

Invalidates: A given Claim can completely invalidate another Claim. Again, this can also hinge on the truth of the invalidating Claim.
Example:
Iraq has no weapons (Claim 1) therefore Iraq does not have weapons of mass destruction (invalid Claim 2).

Validates: A given Claim can completely validate another Claim. If both are considered to be true, they can be recast as States. Again, this can also hinge on the truth of the invalidating Claim.
Example:
Iraq has Sarin Gas (Claim 1) therefore Iraq has weapons of mass destruction (valid Claim 2).

Presupposes: A State can assume another State as a part of its expression.
Example:

Iraq is a terrorist nation (State 1) only if Iraq has weapons of
mass destruction (Claim 1).

Sub-State-of: This makes the relation among states and sub-states (those
goals that are subordinately related to the higher goal) explicit.
Example:
Iraq having weapons of mass destruction (Sub-State 1) is evidence that Iraq
supports terrorism (State 2).

Relations between Actions and other Objects:

Hinders: An action can make another action or a goal more difficult to
achieve.
Example:
Irradiating Iraq (Action 1) would make it difficult to invade Iraq with
few casualties (Goal 1).

Aids: An action can aid another action or a goal, rather than causing it
to be accomplished outright.
Example:
Providing money to rebels (Action 1) in Iraq would help an effort to
invade Iraq (Action 2).
Providing money to the Iraqi poor (Action 1) would partially help to
destabilize Saddam's regime (Goal 1).

Prevents: An action can prevent another action, or a goal from being
accomplished. By prevent I mean outright stop the action or goal, as
opposed to hinder.
Example:
An attack on Iraq (Action 1) would prevent Iraq from using weapons of
mass destruction (Actor-goal 1).

Accomplishes: An action can cause a goal to be accomplished/completed or
can cause another action to be completed.
Example:
Attacking Iraq (Action 1) would cause the US to find weapons of mass
destruction (Goal 1).
Attacking Iraq (Action 1) would cause the US to move troops into
Iraq (Action 2).

Relations between Goals and other objects:

Sub-Goal-of: This makes the relation among goals and sub-goals (those
goals that are subordinately related to the higher goal) explicit.
Example:
Destroying Iraq (Goal 1) is a part of winning the war on terrorism

(Goal 2).

Temporal Relations:

Parallel-to: Elements can be constrained to take place in parallel.
Example:
The US must attack Iraq (Goal 1) while at the same time appeasing the United Nations (Goal 2).

Before: Elements can be constrained such that one Element takes place before another.
Example:
The US must move troops to the Middle East (State 1) before it attacks Iraq (Action 1).

After: Elements can be constrained such that one Element takes place after another.
Example:
Iraq must be rebuilt (Goal 1) after the war against Saddam Hussein is won (Goal 2).

Appendix B

Jackendoff Structures for Political Concepts

```
1. Sentence: ‘‘Billy has the ring.’’

| be
|
|  |
|  |
|  | Thing ring
|
|  | at
|  |    |
|  |    | Thing agent Billy
|  |
|
| Thing verb has


2. Sentence: ‘‘Billy does not have the ring.’’

| be
|
|  |
|  |
|  | Thing ring
|
|
|  | not
|  |     | at
|  |     |    |
|  |     |    | Thing agent Billy
|  |     |
|  |
|
| Thing verb has


3. Sentence: ``Billy wants the ring.''

| cause
|
|  |
|  |
|  | Thing agent Billy
|
|
|  | go
|  |    |
|  |    |
|  |    | Thing ring
|  |
|  |    | to
|  |    |    |
|  |    |    | Thing agent Billy
|  |    |
|  |
|
| Thing verb wants


4. Sentence: ``Billy has nothing.''

| not
|  | be
|  |
|  | |
|  | |
|  | | Thing anything
|  |
|  |
|  | | at
|  | |    |
|  | |    | Thing agent Billy
|  | |
|  |
|

5. Sentence: ``Billy kept the ring.''

| cause
|
|  |
|  |
|  | Thing agent Billy
|
|
|  | stay
|  |
|  |    |
|  |    | Thing ring
|  |
|  |    | at
|  |    |    |
|  |    |    | Thing agent Billy
|  |
|
| Thing verb keep
```

6. Sentence: ‘‘Billy received the ring.’’

```
| cause
|
|  |
|  |
|  | Thing agent Unknown
|
|
|  | go
|  |
|  |    |
|  |    | Thing ring
|  |
|  |    | to
|  |    |    |
|  |    |    | Thing agent Billy
|  |
|
| Thing verb receive
```

7. Sentence: ‘‘Billy lost the ring.’’

```
| cause
|
|  |
|  |
|  | Thing agent Unknown
|
|
|  | go
|  |
|  |    |
|  |    | Thing ring
|  |
|  |    | from
|  |    |     |
|  |    |     | Thing agent Billy
|  |
|
| Thing verb lose
```

```
8. Sentence: ‘‘Billy obtained the ring.’’

| cause
|
|  |
|  |
|  | Thing agent Billy
|
|
|  | go
|  |
|  |    |
|  |    | Thing ring
|  |
|  |    | to
|  |    |    |
|  |    |    | Thing agent Billy
|  |
|
| Thing verb obtain
```

```
9. Sentence: ‘‘Billy accepted the ring.’’

| cause
|
|  |
|  |
|  | Thing agent Unknown
|
|  | go
|  |
|  |    |
|  |    | Thing ring
|  |
|  |    | to
|  |    |    |
|  |    |    | Thing agent Billy
|  |
|
| Thing verb
```

OR (needs a LET primitive)

```
| let
|
|  |
|  |
|  | Thing agent Billy
|
|
|  | go
|  |
|  |    |
|  |    | Thing ring
|  |
|  |    | to
|  |    |    |
|  |    |    | Thing agent Billy
|  |
|
```

10. Sentence: ‘‘Billy relinquished the ring.’’

Same as ‘‘Billy lost the ring.’’

OR

```
| let
|
|  |
|  |
|  | Thing agent Billy
|
|
|  | go
|  |
|  |    |
|  |    | Thing ring
|  |
|  |    | to
|  |    |    |
|  |    |    | Thing agent Billy
|  |
|
```

11. Sentence: ‘‘Billy bought the ring for 100 dollars.’’

```
| cause
|
|  |
|  | Thing agent Billy
|
|  | go
|  |
|  |    |
|  |    | Thing ring
|  |
|  |    | from
|  |    |    |
|  |    |    | Thing agent Unknown
|  |
|  |    | to
|  |    |    |
|  |    |    | Thing agent Billy
|  |
|
|  | go
|  |    |
|  |    | Thing dollars 100
|  |
|  |    | from
|  |    |     | Thing agent Billy
|  |
|  |    | to
|  |    |     | Thing agent Unknown
|  |
|
| Thing verb buy
```

12. Sentence: ‘‘Billy does not want the ring.’’

```
| cause
|
|  |
|  |
|  | Thing agent Billy
```

```
|
|
|   | go
|   |   |
|   |   |
|   |   | Thing ring
|   |
|   |   | from
|   |   |   |
|   |   |   | Thing agent Billy
|   |   |
|   |
|
| Thing verb wants
```

13. Sentence: ‘‘Billy controls the playground.’’

```
| be
|
|   |
|   | Thing playground
|
|   | at
|   |  |
|   |  | Thing agent Billy
|   |
|
| Thing verb control
```

14. Sentence: ‘‘Billy wants to control Diana.’’

```
| cause
|
|   |
|   |
|   | Thing agent Billy
|
|
|   | go
|   |  |
|   |  |
|   |  | Thing agent Diana
|   |
|   |  | to
|   |  |   |
|   |  |   | Thing agent Billy (controlspace)
|   |   |
|   |
|
| Thing verb wants
```

15. Sentence: ‘‘Billy wants Diana to come to him.’’

```
| cause
|
|   |
|   |
|   | Thing agent Billy
|
|   | go
|   |    |
|   |    |
|   |    | Thing agent Diana
|   |
|   |    | to
|   |    |    |
|   |    |    | Thing agent Billy (realspace)
|   |    |
|   |
|
| Thing verb wants
```

16. Sentence: ‘‘Wait for Bob to get the book.’’

```
| stay
|
|   |
|   |
|   | Thing agent
|
|   | at
|   |    |
|   |    | place
|
|   | until
|   |    | go
|   |    |    |
|   |    |    | Thing Agent Bob
|   |    |
|   |    |    | to
|   |    |    |    | Thing book
|   |    |    |
|  |
|
```

17. Sentence: ‘‘Do not get the book.’’

```
| not
|
|  | go
|  |  |
|  |  | Thing agent Unknown
|  |
|  |  | to
|  |  |  |
|  |  |  | Thing book
|  |
|  |
|
| Thing verb do
```

18. Sentence: ‘‘Bob tried to get the book.’’

```
| go
|
|  |
|  | Thing agent Bob
|
|  | to
|  |  |
|  |  |
|  |  | Thing book
|  |
|
| Thing verb try
```

19. Sentence: ‘‘Bob walks with Gina.’’

```
| go
|
|  |
|  | Thing agent Bob
|
|  | with
|  |   |
|  |   |
|  |   | Thing agent Gina
|  |
|
|  | to
|  |   |
|  |   |
|  |   | Thing place
|  |
|
| Thing verb walk
```

20. Sentence: ‘‘Billy supports Bobby <to> <action>.’’

```
| cause
|
|  |
|  | Thing agent Billy
|
|  | <action>
|  |
|  |  |
|  |  | Thing agent Bobby
|  |
|  |  |
|  |  |
|  |  | Thing Unknown
|  |
|
| Thing verb support
```

21. Sentence: ‘‘Billy hinders Bobby <from> <action>.’’

```
| cause
|
|  |
|  | Thing agent Billy
|
|
|  | not
|  |
|  |  | <action>
|  |  |
|  |  |  |
|  |  |  | Thing agent Bobby
|  |  |
|  |  |  |
|  |  |  |
|  |  |  | Thing Unknown
|  |  |
|  |
|  | Thing verb do
|
| Thing verb hinder
```

```
22. Sentence: ``The top of the hill is difficult to control from the bottom
of the hill.''
Also: ``It is difficult for <agent> to control <top> from <bottom>.''

| be
|
|  | go
|  |   | be
|  |   |   |
|  |   |   | Thing agent
|  |   |
|  |   |   | at
|  |   |   |   | bottom
|  |   |   |   |       | Thing hill
|  |
|  |   | to
|  |   |   | top
|  |   |   |    | Thing hill
|  |   |
|  |
|
|
|  | at
|  |   |
|  |   |
|  |   | Thing characterization difficult
|  |
|
```

```
23. Sentence: ``Billy let Jim attack Bob.''

| let
|
|  |
|  | Thing agent Billy
|
|  | go
|  |   |
|  |   | Thing agent Jim
|  |
|  |   | to
|  |   |   |
|  |   |   | Thing agent Bob
|  |   |
|  |
|
```

24. Sentence: ‘‘Billy removed the knife from the box.’’

```
| cause
|
|  |
|  | Thing agent Billy
|
|  | go
|  |
|  |    |
|  |    | Thing agent Billy
|  |
|  |    | to
|  |    |    |
|  |    |    | Thing box
|
|  | go
|  |    |
|  |    | Thing knife
|  |
|  |    | from
|  |    |     | Thing box
|  |
|  |    | to
|  |    |     | Thing agent Billy
|  |
|
| Thing verb remove
```

25. Sentence: ‘‘Billy built a castle in the sand.’’

```
| cause
|
|  |
|  | Thing agent Billy
|
|  | go
|  |  |
|  |  | Thing castle
|  |
|  |  | to
|  |  |   | in
|  |  |   |   | sand
|  |
|  | Thing verb create
|
| Thing verb build
```

26. Sentence: ‘‘Joe destroyed the castle.’’

```
| cause
|
|  |
|  | Thing agent Joe
|
|  | go
|  |  |
|  |  | Thing castle
|  |
|  |  | to
|  |  |   |
|  |  |   | Thing nothing
|  |
|
| Thing verb destroy
```

27. Sentence: ‘‘Billy needs the book by Tuesday.’’

```
| cause
|
|  | go
|  |  |
|  |  | Thing book
|  |
|  |  | to
|  |  |   |
|  |  |   | Thing agent Billy
|
|  | at
|  |   | by
|  |   |   |
|  |   |   | Thing day Tuesday
|  |
|
| Thing verb need


28. Sentence: ``Iraq conquered Kuwait.''

| go
|
|  |
|  | Thing Kuwait
|
|  | to
|  |    |
|  |    | Thing Iraq
|


29. Sentence: ``Iraq became a democracy from a dictatorship.''

| go
|
|  |
|  | Thing Iraq
|
|  | from
|  |    |
|  |    | Thing dictatorship
|
|  | to
|  |    |
|  |    | Thing democracy
|


30. Sentence: ``Conditions in Iraq range from bad to worse.''

| go
|
|  | be
|  |  |
|  |  | Thing conditions
|  |
|  |  | at
|  |  |    | Iraq
|
|  | from
|  |    |
|  |    | Thing bad
|
|  | to
|  |   |
|  |   | Thing worse
|
```

31. Sentence: ‘‘The book will be destroyed on Monday <by> <agent>.’’

```
| cause
|
|  |
|  | Thing agent
|
|  | go
|  |  |
|  |  | Thing book
|  |
|  |  | to
|  |  |   |
|  |  |   | Thing nothing
|  |
|  |  | at-by
|  |  |
|  |  |   |
|  |  |   | Thing day Monday
|  |  |
|  |
|
| Thing verb destroy
```

32. Sentence: ‘‘The bombs are in Iraq.’’

```
| be
|
|  |
|  | Thing bombs
|
|  | in
|  |  |
|  |  | Thing Iraq
|  |
|
```

33. Sentence: ‘‘The bombs are under the table in Iraq.’’

```
| be
```

```
|
|  |
|  | Thing bombs
|
|  | under
|  |
|  |  | be
|  |  |    |
|  |  |    | Thing table
|  |  |
|  |  |    | at
|  |  |    |    | Thing Iraq
|  |  |
|
```

34. Sentence: ‘‘The missiles are aimed at Iraq.’’

```
| cause
|
|  |
|  | Thing agent
|
|  | stay
|  |  |
|  |  | Thing missles
|  |
|  |  | at
|  |  |    |
|  |  |    | iraq
|  |  |
|  |
|
| Thing verb aim
```

35. Sentence: ‘‘The missiles are fired at Iraq.’’

```
| cause
|
|   |
|   | Thing agent
|
|   | go
|   |  |
|   |  | Thing missles
|   |
|   |  | from
|   |  |   |
|   |  |   | iraq
|   |  |
|   |
|
| Thing verb fire
```

```
36. Sentence: ``Sue forced Jim to sing.''

| cause
|
|   |
|   | Thing agent Sue
|
|   | go
|   |  |
|   |  | Thing agent Jim
|   |
|   |  | to
|   |  |   |
|   |  |   | sing
|   |  |
|   |
|
| Thing verb force
```

```
37. Sentence: ``Sue kept Jim from singing.''

| cause
|
|   |
|   | Thing agent Sue
|
|   | stay
|   |  |
|   |  | Thing agent Jim
|   |
|   |  | at | not
|   |  |    |    |
|   |  |    |    | sing
|   |  |
|   |
|
| Thing verb keep
```

```
38. Sentence: ``Sue allowed Jim to sing.''

| let
|
|   |
|   | Thing agent Sue
|
|   | go
|   |  |
|   |  | Thing agent Jim
|   |
|   |  | to
|   |  |   |
|   |  |   | sing
|   |  |
|   |
|
| Thing verb allow


39. Sentence: ``Sue released Jim from singing.''

| let
|
|   |
|   | Thing agent Sue
|
|   | go
|   |  |
|   |  | Thing agent Jim
|   |
|   |  | from
|   |  |   |
|   |  |   | sing
|   |  |
|   |
|
| Thing verb release


40. Sentence: ``Sue exempted Joe from singing.''

| let
|
|   |
|   | Thing agent Sue
|
|   | stay
|   |  |
|   |  | Thing agent Jim
|   |
|   |  | at | not
|   |  |    |    |
|   |  |    |    | sing
|   |  |
|   |
|
| Thing verb exempt
```

41. Sentence: ‘‘The bull is black.’’

```
| characterization
|
|   |
|   | Thing animal bull
|
|   |
|   | Thing color black
|
| Thing verb is
```

42. Sentence: ‘‘Billy kicked the ball to the tree.’’

```
| cause
|
|   |
|   | Thing agent Billy
|
|   | go
|   |  |
|   |  | Thing ball
|   |
|   |  | to
|   |  |   |
|   |  |   | Thing tree
|   |  |
|   |
|
| Thing verb kick
```

Appendix C

An Overview of Relevant Aspects of the Bridge System

The Bridge System implements its own version of Jackendoff structures, furthermore called simplified Jackendoff frames. Each object in a sentence is represented as a Thing frame, and these Thing frames are used to build objects that resemble Jackendoff's conception of LCS.

C.1 The Thing Framework

Every object in the Bridge system is a Thing. A Thing encompasses many ideas, but is a specific implementation of Minsky's frames concept. [8] Thing objects have various slots that can be filled to identify them as any number of elements basic to the functioning of the system, such as simple objects and Jackendoff frames.

In its basic form, a Thing is merely a set of boolean values and a bundle of threads, inspired by Greenblatt and Vaina's thread idea. [12] A thread in the Bridge system is a linked list of types. There is no built-in constraint on the use of a thread, nor is there any mechanism to ensure that hierarchical information flows from most specific to least specific as in Greenblatt and Vaina's work. In general, however, this is how they are used. It is up to the programmer to ensure such constraints if they are necessary. Threads are held in bundles, which are ordered lists that contain only threads. Each bundle has a "primed" thread that is meant to represent the most salient information about the object to which this bundle is attached. For example, if we had a Boy object, and two threads (BOY → PERSON → CHILD) and (BOY → HUMAN → ANIMAL) we could imagine that the first thread would be primed, since the information on that thread seems the most relevant to the concept of Boy. Note, however, that threads are generally not used in this manner. Rather, threads are most often used to store specific properties. For example, in the Boy bundle, there could be a "color" thread, a "size" thread and a "sex" thread, all of which would hold the relevant information about the Boy. This allows for relatively easy indexing into a thread bundle in order to discover object properties. The primed Thread in these instances is usually the one that describes the object in the most general terms, which in this example would be (OBJECT → LIVING OBJECT → BOY). This is clearly a somewhat loose implementation of the very specific idea in Greenblatt and Vaina's work, but the flexibility in this representation allows for broader and more flexible applications.

There are several types of Things in addition to those that represent simple objects.

These types are Derivatives, Relations and Sequences. Derivatives represent an object derived from another object via a preposition. To this end, a Derivative has a special slot for a subject and the actual derivative function itself. An example of a Derivative comes from the sentence fragment “to the table”. The Derivative here would be “to” and the subject would be “table”. A Relation represents a relationship between two Things (which could potentially be any subclass of Thing). A Relation is a kind of Derivative, and thus has slots for a subject and the actual relation itself. Relations also have a slot for the object of the relation. An example of a Relation comes from the sentence “The boy ran to the table.” In this example, the subject is a Thing representing “boy”, the relation is the verb “ran” and the object is the derivative “to the table”. The final type of Thing is a Sequence. A Sequence is formalism for a sequence of Things that are related in some way by their ordering. As such, a Sequence is merely an ordered list of Thing objects.

C.2 Jackendoff Frames in the Bridge System

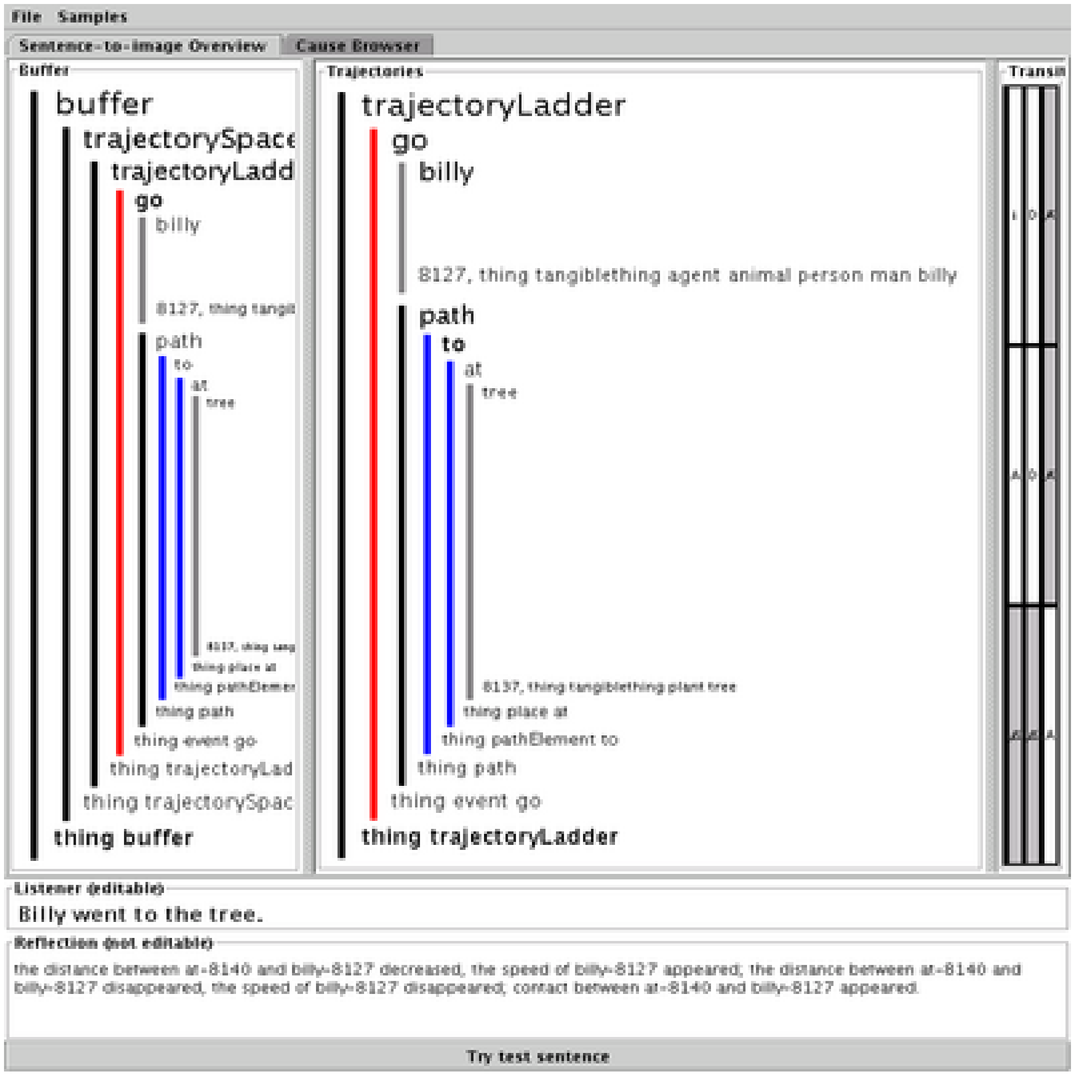
Things are also the basis for a loose implementation of Jackendoff’s LCS representation. The form of Jackendoff structure used in the Bridge project is somewhat simplified from the specifications outlined by Jackendoff in his work. Within the system, Jackendoff frames are a part of a “trajectory space” representation of the world. A trajectory space encompasses one whole sentence or phrase and shows how that phrase fits into the idea that almost everything in the world can be described in terms of places and paths. As such, the main primitives of the different things in a trajectory space representation of a sentence are states, events, places, objects and paths.

Each of the trajectory space primitives is created from the main Thing classes. Each new sentence invokes the creation of a new Jackendoff frame. Each frame is a Sequence that holds one or more “trajectory ladders”. A trajectory ladder represents one entirely parsable phrase, and is also a Sequence. An example of a sentence with multiple phrases is “The bird flew to the top of the tree and then the bird flew to the rock.” The use of “and then” demarcates a new phrase and thus a new trajectory ladder. The elements in the Sequence for a trajectory ladder depend on the structure of the sentence. Parts of the phrase that deal with verbs are held in Relations. For example, in the previous case, the first trajectory ladder would hold the Relation “bird” (subject) “flew” (relation verb) “to the top of the tree” (object). Any phrases representing a path are contained within a Sequence object, to differentiate them from other primitives. Depending on the phrase structure, the primitives can be recursively nested to give the most accurate representation of the phrase.

C.3 The BridgeSpeak Parser

The specifics of creating Jackendoff frames are handled within the BridgeSpeak parser. BridgeSpeak takes a sentence in English and creates a Sequence object representing a Jackendoff trajectorySpace structure. The parser reads in a known vocabulary of verbs and nouns separated into certain categories and parses based on this knowledge. Depending on the words used in a certain phrase, the parser will create different structures to represent the meaning of a sentence. The following figure shows a user interface for the BridgeSpeak parser and displays a graphic of a typical sentence parse.

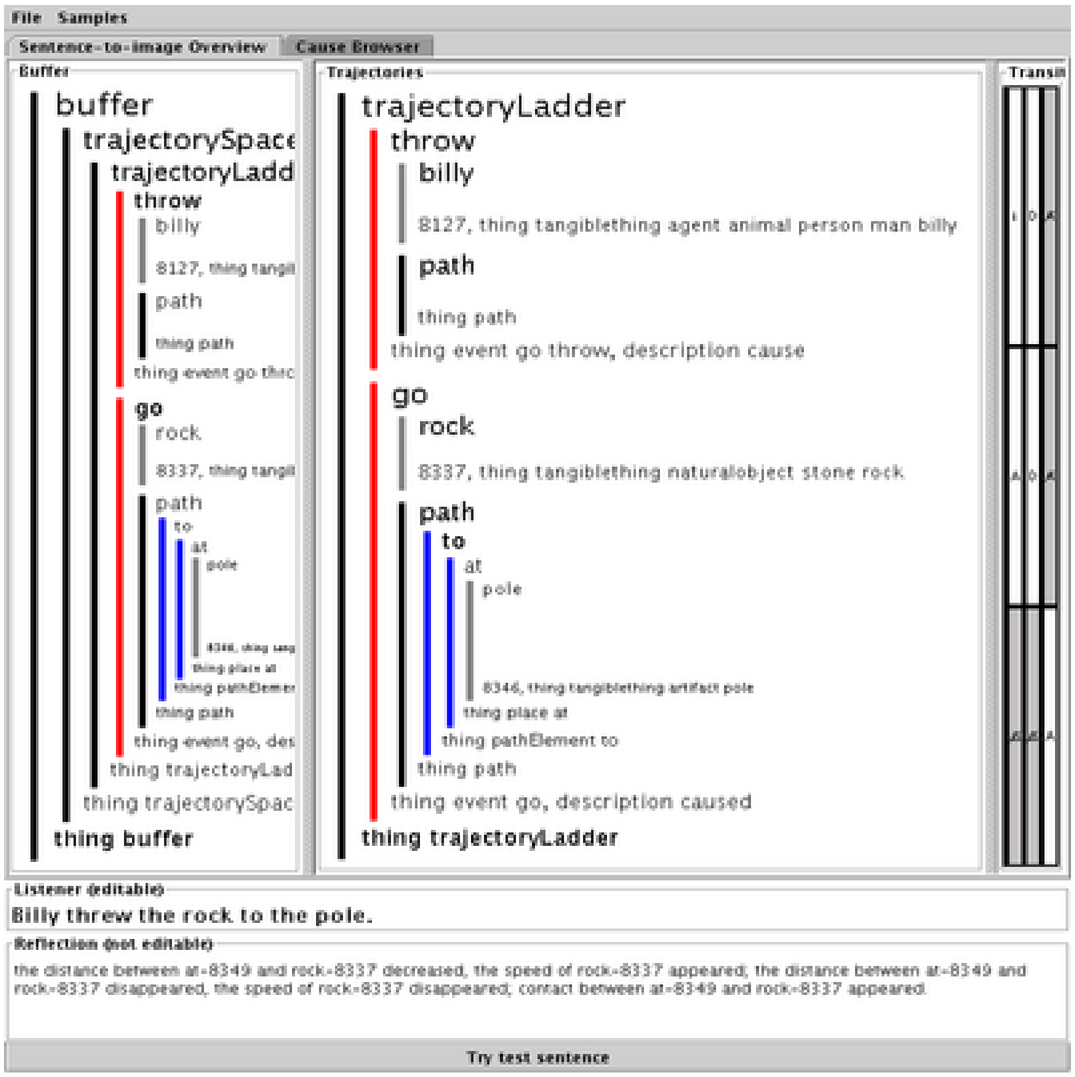
The parser handles many different kinds of phrases. Simple phrases involving a movement along a path, such as “Billy went to the tree” are represented as GO structures, where



Billy went to the tree.

the distance between at-8140 and billy-8127 decreased, the speed of billy-8127 appeared; the distance between at-8140 and billy-8127 disappeared, the speed of billy-8127 disappeared; contact between at-8140 and billy-8127 appeared.

Figure C-1: A typical BridgeSpeak parse and the corresponding Jackendoff frame. Red bars (such as go) are Relations, blue bars (such as to and at) are Derivatives, gray bars (such as billy and tree) are Things and black bars (such as trajectoryLadder and path) are Sequences. Notice the recursive structures of the representation. Notice also the additional information displayed about each object. This information is an output of the content on the prime thread for each object as well as the information on the description thread.



trajectoryLadder
throw
billy
8127, thing tangiblething agent animal person man billy
path
thing path
thing event go throw, description cause
go
rock
8337, thing tangiblething naturalobject stone rock
path
to
at
pole
8346, thing tangiblething artifact pole
thing place at
thing pathElement to
thing path
thing event go, description caused
thing trajectoryLadder

Billy threw the rock to the pole.

the distance between at-8349 and rock-8337 decreased, the speed of rock-8337 appeared; the distance between at-8349 and rock-8337 disappeared, the speed of rock-8337 disappeared; contact between at-8349 and rock-8337 appeared.

Figure C-2: A BridgeSpeak parse of a causal relation.

an object moves along a path toward another object. These are constructed in specific ways by the parser. What the parser also does is associate a thread bundle with each constituent part of the structure. The primed threads of these bundles indicate the type of Sequence, Derivative, Relation or Thing that the object represents. For example, a GO relation would have a type of "go" on its primed thread. This typing information is retrieved from a lexicon loaded into the parser at runtime.

The parser is able to generate specific structures based on the type of verb it encounters in a sentence. One salient example of this is verb phrases that denote causation. Consider the phrase "Billy threw the rock to the pole." In this case, "threw" is a CAUSE-GO verb, which makes "the rock" GO on a PATH to "the pole". Knowing that "throw" is a CAUSE-GO verb allows the parser to construct a special representation for this situation. Note also that both GO and CAUSE-GO situations are EVENTS, as in Jackendoff's original framework.

One final and important note should be made about the memory system of the parser. BridgeSpeak keeps all of the objects it creates in a common "Thing memory." This memory is used to store and change objects from previous parses. For example, if we were to ask BridgeSpeak to parse the sentence “Bob flew to the tree,” the parser would create Thing objects for Bob and the tree (as well as the other parts of the Jackendoff frame). If we subsequently had BridgeSpeak parse “Bob ran to the tree and walked to the pole”, the “Bob” and “tree” objects that BridgeSpeak returned would be the same as the previous objects, since they were stored in the Thing memory and retrieved for the new parse. The only way to generate new objects is to either introduce a completely new word or use the prefix “a” when describing something. For example, if my second sentence was instead “A Bob ran to a tree,” the objects returned would be new instances of “Bob” and “tree”. The workings of this memory structure prove to be very important in creating actual applications relying on the parser, as we saw in the discussion of the graph conversion process and the matcher.

Appendix D

Foundations

The basic foundations for analyzing decision rationale problems are found in the work of Jintae Lee. In his 1992 paper *A Comparative Analysis of Design Rationale Representations*, Lee presents a framework for evaluating representations of the rationale involved in the process of designing some artifact, whether it be software or a piece of machinery. [7] Lee wishes to understand the mental processes behind human decision-making by studying the rationale behind design decisions.

A design rationale encapsulates the reasons, methodology, specifications and alternatives behind the decision to select a certain design for an artifact. Lee's work evaluates representations for design rationale based on five different models that isolate critical issues in different levels of granularity, each being more specific than the next. The first model with which a representation can be evaluated works in the argument space, which encompasses all arguments relevant to the design of an artifact. Next is an alternative space, where all alternatives to the current design are made clear and relations among them are specified. An evaluation space is found underlying the alternative space. The evaluation space holds evaluation measures used to evaluate arguments and the relations among these measures. Further beneath the evaluation space is the criteria space, which, in addition to including all of the previous spaces, also makes explicit the criteria used in making an evaluation and the relationships among these criteria. The final model deals with design rationales in terms of an issue space, where individual issues are the main functional units. Each issue contains the alternatives, evaluations and criteria for the evaluations used in discussing the issue. In this model the argument space also includes meta-arguments about the issues and their relations.

The five models that Lee presents are evaluated based on their ability to facilitate the answering of questions. For example, the model based on a simple argument space can answer a question such as "What can we learn from past design cases?" It cannot, however, handle comparisons among arguments nor can it encompass the reasons behind the selection of certain choices in the design process. Lee concludes that his most specific model, that which deals directly with issues while covering the related alternatives, evaluations and evaluation criteria for each issue, is the most powerful in this sense. The rest of his paper goes on to evaluate several design rational representations based on this assertion. Lee concludes that his representation, the Decision Rationale Language (DRL) is the best suited for the design rationale problem, although much of this is self-fulfilling since his representation was designed to perform well given his own evaluation criteria.

Lee's other major work, *A Decision Rationale Management System: Capturing, Reusing* *and Managing the Reasons for Decisions*, is the original inspiration for this thesis. [6] In this work his overall vision, that of understanding the decision-making process, has not changed. His intermediate goal, however, is not to create a program to demonstrate this understanding, but rather to use an understanding of decision rationale in the creation of a collaboration tool for decision-making tasks. For this reason, much of the machinery in his DRL representation is concerned with capturing opinions and the reasons for these opinions. The representation is designed specifically to cater to multiple viewpoints, allowing many users to submit input with regard to a decision before it is made. Many of the primitives of the language, such as VIEWPOINT (used to represent a personal view on an argument) and QUESTION (used to raise a question for another user to answer) reflect this bias. The manifestation of the representation is an application called SIBYL with which multiple users can enter and view the various parts of a decision rationale asynchronously.

Of all the prior work on decision modeling, Lee's work is a major influence on my own for several reasons. His is the primary (to this date) computational approach to decision rationales. His primitives are more or less applicable to any situation and serve to capture much of the relevant information in a decision rationale. Lee also created software to demonstrate the feasibility of his representation, which indicates that at the very least it presents a traceable implementation. Lee is also one of the first researchers to articulate the problem of decision rationale from a computational perspective, and thus presents an excellent starting point for future research.

There are many problems fitting Lee's work into the context of the decision rationale problems covered in this thesis. Lee's work was primarily focused on collaboration and did not address the ability to systematically analyze and learn from past decisions. Lee rather believed that storing the decision rationale in a structured way would facilitate analysis by human hands in the future. Although this is certainly true, I believe it is possible to find structural similarities between current and past decisions in order to identify possible points of failure. In this regard, Lee's representation is also missing some key information. Originally, I believed that there was possibly some structural similarity that would become apparent by fitting two similar decisions into Lee's framework. This does not hold true, however, since any one decision has a nearly infinite number of representations in Lee's framework due to the representational focus on collaboration rather than specific decision analysis. Therefore, this led me to the conclusion that structural similarities among decisions must occur on a level of granularity below that of the basic decision structures, down to the actual sentences and data used to construct the decision. The so-called "words in the boxes" are not important in Lee's conception of decision-rationale. These words, however, are key to any meaningful analysis of a specific decision for the purposes of automatic verification and comparison. In many ways, this is why Lee's work is so easy for human users to exploit, since Lee is counting on humans to do the work of comparison and analysis rather than machines.

Lee's work alone, however, was not enough to fully elucidate exactly which components of the text associated with a decision are important. A major inspiration in this regard is the work of Abelson on political reasoning systems, most clearly explained in *The Structure of Belief Systems*. [1] Abelson's broader vision is to understand human reasoning by creating computer software that can reason about the world. In this particular work, Abelson focuses on creating a system that can reason about political ideology (such as the ideology of a "Cold Warrior") and demonstrate this reasoning by answering questions about political situations. This sort of goal was especially important when his paper was written because of the Cold War and the desire of many political analysts for new tools to understand complex geopolitical situations.

In order to achieve his vision, Abelson created an “Ideology Machine” able to deal with 500 nouns and noun phrases and 100 verbs and verb phrases. The vocabulary of this software is generally tailored for political ideologies (such as neutral and liberal) and actions taken in political contexts (such as subversion, physical attack and material support). The application focused on the task of answering six questions: (“Is E credible?”), (“If and when E, what will happen?”), (“If and when E, what should A do?”), (“When E, what should A have done?”), (“How come E? What caused E or what is E meant to accomplish?”) and (“Please comment on E.”).

At the time of writing, Abelson had not created an application that could accomplish his given task. His is overly vague on the details of what he was actually able to accomplish, but he goes into great detail on a model for future work in the area. This model is based on Schank’s Conceptual Dependency representation, a structured approach at a conceptual semantics for automated reasoning systems. The basic units of Abelson’s model are “elements”. Elements are things such as actors and basic actions. When certain elements are brought together using the Conceptual Dependency model, they form “atoms”. There are three kinds of atoms in Abelson’s system: Action, Purpose and State. Atoms can be linked together to form “molecules”, which signify an action taken by an actor to achieve a purpose. Further above this are “plans”, chains of atoms of length three or more in network configurations that signify a complex series of actions and purposes leading to a final outcome. “Themes” are molecules or plans connected together to represent the interaction between two different actors. Finally, a “script” is a sequence of themes between two actors, representing the evolution in their relationship over time given certain circumstances.

Much of Abelson’s representation is too rigid and mechanistic to be truly useful in a wide variety of circumstances. He does, however, contribute three crucial ideas with his work.

- The critical distinction he makes between actions, states and goals. These distinctions are important because they represent what I consider to be the fundamental components of any decision: the current state of the world, the goals of the decision and the actions possible to accomplish the goals. A goal can be seen as a possible future state of the world that the decision is aimed at reaching.

- His exploration of causation. Abelson outlines many different kinds of causation, such as explicit causation and what he calls “gating”, where one series of elements can allow an action or set of actions to occur. He gives a thorough enumeration of these types in terms his basic representation, which although not generally applicable, does ground out his work.

- The idea of using a basic conceptual semantics to underpin his representation. This acknowledges the need for some basic level of understanding in order to perform more complex reasoning on larger structures. I do not, however, agree with the actual conceptual semantics used. Although they are useful, they do not reflect any plausible level of understanding for human intelligence and instead rely on somewhat arbitrary distinctions for primitives such as “PROPEL” (for propelling an object), “PTRANS” (for a physical transfer) and “MTRANS” (for a mental transfer).

The work of Jackendoff addresses the inadequacies of the conceptual semantics used by Abelson. His vision is to understand human language and its role in human cognition. In his *Semantics and Cognition*, Jackendoff outlines his Lexical Conceptual Semantics (LCS) model for linguistic understanding. [5] A key component of this model is a semantics of spatial expressions, which represents common spatial relationships in language. Jackendoff's representation sees the world in terms of events and states. Involved in these events and states are places and paths. Both sets of distinctions are important. The difference between events and states is similar to the distinction between actions and states noted in Abelson's work, and thus allows Jackendoff's representation to fit well into a similar framework. The distinction between places and paths points out a fundamental aspect of human cognition, that much of what we understand in both the realms of space and abstract thought can be described in terms of objects moving along trajectories.

Jackendoff provides explicit evidence that his representation is useful in situations beyond those of simple spatial relations. This is evident with spatial expressions such as "I threw the ball into the hoop." Jackendoff also stresses that many abstract notions can be described in this spatial framework. For example, "Jane gave Billy the ball" evokes a notion of a ball moving along a path from Jane to Billy. Jackendoff's representation makes explicit the notion that the human understanding of abstract notions grounds out in the understanding of spatial relationships. Furthermore, the primitives of this representation (such as EVENT, STATE, PLACE, PATH and THING) are sufficiently basic as to "feel right", and are certainly more plausible than those of Schank and Abelson's representation.

Both Abelson and Jackendoff fail to adequately describe the objects in their representations. This leaves room for another representation to handle objects, their properties and their classifications. An interesting approach to this problem is the thread memory model of Greenblatt and Vaina, as presented in *The Use of Thread Memory in Amnesia Aphasia and Concept Learning.* [12] Greenblatt and Vaina seek to understand human memory by building a semantic memory system. In their model, a thread is a loop-free chain of concepts, representing a notion from most-specific type to least-specific type. Threads are keyed on their most-specific type.

The representation of memory using threads is interesting, in that it has a notion of hierarchy, but does not branch. Rather, multiple threads are used to show that a certain concept belongs in different hierarchy chains. For example, note the following two chains:

```
GIRL -> PERSON -> FEMALE -> GIRL
GIRL -> CHILD -> GIRL
```

Both chains describe a girl, but each chain is represented independently. This level of redundancy is built in to account for the possibility of multiple points of view in the memory system. A local change to a concept will not have a global effect in this model. The thread memory model describes an object using a group of threads. Object properties are contained on threads marked by the property name. For example, to find the color of a box, one would look at the thread (BOX COLOR-OF) → BROWN. These groups of threads allow for a robust and detailed description of an object via its various properties and its classification. Note also that the model does not have any stereotypes. A stereotype can be described as a "typical member" of a certain group. For example, when you think of a bird, the image of a bird that comes to mind immediately can be thought of as the stereotype of a bird that you have stored in your memory. The lack of stereotypes makes the representation less fragile (since the stereotypes are not there and thus cannot be lost) and also able to handle deviations from the norm by the addition of more threads rather than the creation of an entirely new object as an exception to the stereotype.

A key idea of this thesis is that abstract ideas find their representational grounding in spatial understanding. Although this has long been a suspicion of many researchers, only recently has work in cognitive science provided evidence for this assertion. The most compelling of this evidence comes from the work of Boroditsky. She seeks to understand human intelligence by performing experiments to demonstrate the role of language and spatial understanding on cognition. In her paper *Metaphoric structuring: understanding time through spatial metaphors* [2], Boroditsky demonstrates that the human understanding of time grounds out in an understanding of space. Her work is based on the work of Lakoff, who believes that all abstract notions are understood through metaphors with a small set of experiential knowledge, spatial knowledge being foremost in this set.

Boroditsky focuses her efforts on providing evidence for the Metaphorical Structuring View of abstract reasoning. The general model states that all abstract notions ground out in schemas associated with experiential concepts. The metaphor itself provides structure to an abstract domain by analogy to one that is well understood. There are two versions of this model. The "weak" version states that our initial understanding of an abstract notion always returns to more concrete experiential knowledge, unless the abstract notion is well-known and often used. In that case, specific relations in the abstract domain are "compiled" for easy use later on, and thus the reliance on the base representation is no longer necessary. In contrast, the "strong" view asserts that abstract relations always return to their experientially grounded roots.

Boroditsky attempts to support the "weak" hypothesis by attacking time as a realm that relies heavily on spatial understanding. Linguistic phrases such as "move the meeting forward" and "push the time back" directly demonstrate her assertion basic assertion in terms of spatial grounding. She provides rigorous examinations in the form of psychological experiments conducted on college students with the purpose of determining whether spatial schemas are at all related to time, whether spatial schemas can be used to understand time, and whether these schemas are always necessary to understand time. She conducted three experiments; one involved a paper questionnaire in which spatial situations were used as primers for temporal questions, one involved a series of paper questionnaires in which temporal and spatial primes were intermixed to determine if spatial understanding is always used in thinking about time, and the final experiment involved a rather lengthy series of questions and timing analyses to determine if the effect of time priming space was noticeable versus space priming time and time priming time. The results of these three experiments indicate that spatial understanding is in fact crucial to our understanding of time, but that without priming, we tend to think about time in its own terms; that is, there is some sort of compiled representation for time being used so that the spatial grounding is not always necessary.

The body of this evidence indicates that the "weak" Metaphorical Structuring View is a further step towards a model of the human understanding of abstract notions. This work is a crucial inspiration for the work in this thesis. It provides evidence that the approach taken here, that of grounding out an understanding of crucial abstract political ideas in spatial understanding, is biologically correct.

One of the key problems examined in this work is the general problem of matching. Matching is perhaps the most crucial problem in building a computational model of decision understanding, since there needs to be a structured way to match current situations with past experience. The matching model presented in this work is philosophically influenced by the works of Ullman [11] and Gentner [4]. The main contribution of Ullman is his streams and counter-streams model. Streams and counter-streams is an approach to object recognition which uses a bi-directional search. A target image comes in from the sensory inputs, while possible matches are explored from memory until a match is found connecting the target image to an object in memory. The model is highly flexible and accounts for such things as visual hallucinations (when matches are improperly primed and a false match is passed on as correct to the visual recognition apparatus). Although the model is specific to vision, the idea of using bidirectional search via a set of manipulations and transformations on a source and a target is compelling, and was the primary inspiration for the bidirectional matcher used in this work.

Gentner's work (written in conjunction with Markman) focuses on the use of structure maps in making analogies. Specifically, their work shows that analogy and similarity rely on matching that works by aligning the structures between a source and a target. The alignment must obey four principles: the alignment must show parallel connectivity (matching relations must have matching arguments), it must show one-to-one correspondence (any one element in one representation in the source must match with only one representation in the target), it must show relational focus (the relations but not the objects need to be in common) and it must demonstrate systematicity (the tendency to match based on higher-order connections among relations). Guided by these principles, Gentner and Markman produced the Structure Mapping Engine, an advanced matcher that takes two textual situations and completes a structural analogy between them. The structural analogy is created based on a series of transformations working in one direction to form a connected structure encompassing both situations.

An original goal of this work was to analyze specific military blunders in order to find exploitable regularities in these failures. Cohen and Gooch have done an excellent job of analyzing the causes of military failures in *Military Misfortunes: The Anatomy of Failure in War*. [3] Through a thorough analysis of several conspicuous military failures, Cohen and Gooch come to the conclusion that many of the failures in military movements and decision-making are due to structural issues. Specifically, the organizational structures of the military and its decision-making apparatus in each of the cases they examined failed in some way, causing imperfect information and flawed decision-making processes. For example, in the Egyptian attack on Israel in 1973, the Israeli military failed to anticipate and react to the invasion. The failure in this case was twofold: the Israeli military was grounded in thinking that gave them an imperfect model of Egyptian goals, and the Israeli military organizational structure was such that early information about the attack failed to reach authorities capable of acting on it. This study points out the issues with trying to stop blunders in complex military situations, particularly that with imperfect models and information, stopping blunders is often not possible. Furthermore, many blunders cannot be traced to one point of failure, and thus a singular decision-making application would have a very hard time preventing highly complex decision-making errors. It is these observations that caused the focus of this work to shift from complex political and military decisions to more simple cases in the barnyard, so as to take the first steps toward eventual applications for real world and real time use.

Bibliography

[1] Robert P. Abelson. *The Structure of Belief Systems*, chapter 7, pages 287–339. Freeman, San Francisco, California, 1973.

[2] Lera Boroditsky. Metamorphic structuring: Understanding time through spatial metaphors. *Cognition*, 75(1):1–28, 2000.

[3] Eliot A. Cohen and John Gooch. *Military Misfortunes: The Anatomy of Failure in War.* Vintage Books, New York, New York, 1991.

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

[5] Ray Jackendoff. *Semantics and Cognition*, volume 8 of *Current Studies in Linguistics Series.* MIT Press, Cambridge, Massachusetts, 1983.

[6] Jintae Lee. A decision rationale management system: Capturing, reusing and managing the reasons for decisions, 1992.

[7] Jintae Lee and Kum-Yew Lai. Comparative analysis of design rationale representations.

[8] Marvin Minsky. A framework for representing knowledge. AI Memo 206, MIT Artificial Intelligence Laboratory, 1974.

[9] Srini Narayanan. Moving right along: A computational model of metaphoric reasoning about events. In *Proceedings of the National Conference on Artificial Intelligence (AAAI '99)*, pages 121–128, Orlando, Florida, 1999. AAAI Press.

[10] Alan Turing. Computing machinery and intelligence. *Mind*, (59):433–460, 1950.

[11] Shimon Ullman. *Sequence Seeking and Counter Streams: A Model for Information Flow in the Visual Cortex*, chapter 10, pages 317–358. MIT Press, Cambridge, Massachusetts, 1996.

[12] Lucia M. Vaina and Richard D. Greenblatt. The use of thread memory in amnesic aphasia and concept learning. AI Working Paper 195, MIT Artificial Intelligence Laboratory, 1979.

[13] Patrick Winston. Bridge project memo. Artificial intelligence laboratory memorandum, MIT Artificial Intelligence Laboratory, 2003.