Having been introduced to a myriad of approaches that are used for natural language processing, we were to embark on the task of understanding a short paragraph titled *Dashed Hopes*. (For purposes of being self-contained, the story follows at the end of this section). As apparent, *Dashed Hopes* is a rather simple, toy story that heavily contains sequential actions in the basic form of simple and compound sentences. Moreover, the story is entirely in the form of active voice, which serves as another convenience for making our job slightly easier. Nevertheless, as we started this project, we had only briefly been exposed to the many elements that we may need to include within our developed software. Specifically, we knew that it would be ideal to include semantic and episodic memory, usage of scripts, goals, reasoning, unification, forward-chaining, a way of generating conceptual dependencies (CD), and English generation.

We were made aware of the benefits of using various types of programming languages. For example, LISP would be convenient for its natural way of handling unification, and PROLOG for its representation of facts. Yet, we clearly wanted to make our job more challenging by choosing an object-oriented language–JAVA–for developing our system. Our main motivation behind this is that we (1) have much more experience using it than any other language; (2) we felt that there was much more to the project than unification and representation of facts, and that treating everything as an object with attributes would be useful.

Once we decided on using JAVA, we decided on a rough outline of our system's design. It was clear that we needed some way to transform these initial words into slightly more meaningful groups of text–a lexicon. So, we were to create a **LexiconAnalyzer** that would serve as the pre-processor of our story, and would perform other tasks that are outlined in Section 2.

Next, with these slightly meaningful lexicons, we could then start to analyze the format and obtain meaning of these lexicons by fitting them to various patterns and rules. Moreover, we could further resolve implied details of the story by matching our formed sentences against *scripts*–a succinct item-list of events that generally occur for a given scenario such as dating or dining. Collectively, these steps would be performed by our **ConceptualAnalyzer (CA)**. As the ConceptualAnalyzer makes these inferences and unifies, it will produce ConceptualDependencies (CD) diagrams, as detailed in Section 4.

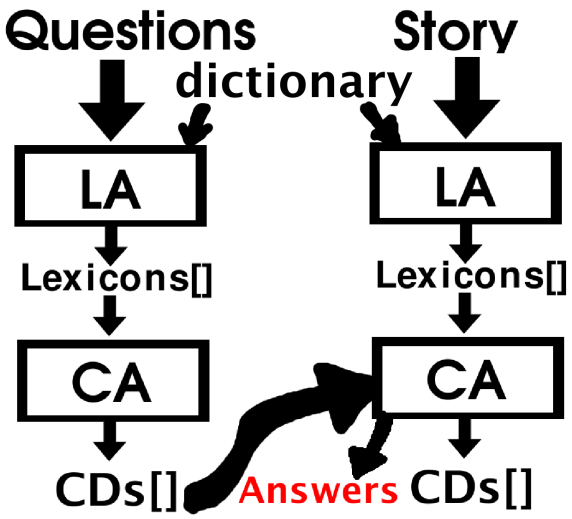


Figure 1: Design of System

Once we have modeled the story' sentences, we can now handle questions. Similar to how we read the story, the LexiconAnalyzer will serve as a pre-processing step for the questions. It convert the questions into lexicons and will appropriately pass them to the CA for analyzing. The CA will construct CDs (lets call them QCDs), just as it did with the original story sentences. When one wishes, one can then obtain answers to the questions by passing the QCDs to the original CA that constructed the story's sentences.

For a simplified diagram of our system's design, see Figure 1.

Dashed Hopes

Bob was famished. His fridge was empty so he got his car. Leones was crowded so the hostess asked him to share a table. Alice immediately introduced herself. He thought she was very attractive and they hit it off. When the waitress brought the menus she ordered the salad and he the steak. He asked her to come back to his place afterwards and she agreed. Bob was very hopeful. After the food came Bob said McCain should be president because government health care in England is bad. Alice got angry. She said he was too senile to be president. Bob felt bad. Bob's hopes were dashed.

As we started to think about the responsibilities of the Lexicon Analyzer (LA), the first thing that became apparent is that some words of the story are meaningless individually but have meaning as a group of words. Moreover, words often make sense only together as a clause or gerund phrase. Although we weren't sure if we would let the LA take responsibility of resolving such phrases, we realized that at least smaller examples or words should be handled. Specifically, pronouns such as "he," "his,", "herself" clearly refer to an actor (or actress), yet our system would originally have no way of knowing to whom the pronoun is referring. So, this is the first task of the LA, but how should the system know how to do this? Clearly, a masculine pronoun refers to a male; likewise, a feminine pronoun for a female. So, this sparked our idea to create a *dictionary*. The dictionary would need to have an entry for all pronouns, and the definition would merely state if it's a male or female. For example, a snippet of our dictionary follows:

```
he (pronoun) = male
she (pronoun) = female
him (pronoun) = male
his (pronoun) = male
hers (pronoun) = female
herself (pronoun) = female
himself (pronoun) = male
```

Whenever our LA sees one of these pronouns, it can attempt to replace it with an appropriate actor. Yet, if the pronoun is unclear and could be ambigious, the system *does not* attempt to resolve it, for fear of corrupting the story.

Second, and along the same lines, there are two colloquial phrases within the story that would be pretty much impossible to semantically interpret. So, we re-define this phrase within our dictionary so that it makes more sense to our system:

```
hit it off (colloquial verb) = encountered fun
got his car (colloquial verb) = drove his car
```

We felt that it was okay to make these substitutions because doing so did not really aid our system in any inference step; rather, it just clarified the slang English.

Third, and most tricky, was the need to transform dependent clauses into independent clauses. There were a few motivations for this. Namely, take the following sentence for example: *"When the waitress brought the menus she ordered the salad and he the steak."* Ignoring the fact that there is a missing comma, the sentence has two main concepts that are represented: (1) The waitress brought the menus; (2) She ordered the salad and he the steak. Additionally, the first concept enabled the second concept to occur. With regard to CDs, we would need to represent both of these clauses. One can then realize that every CD essentially represents one independent clause. Moreover, every dependent clause has an *implied* independent clause. We wish to resolve these clauses by having our LA transform the entire story into a collection of independent clauses. With this, it could then pass these clauses to the CA so that the CA would have nice, clean input with which to work.

The question then arose, how do we tell our system what is a dependent clause and what is an independent clause? We conformed to the grammar rules of English and defined an independent clause to have a *subject* and a *verb*. (Note: we define it such that an actor, object, or place could potentially be the subject). And, if a sentence starts with a subordinating conjunction (i.e. after, before, wherever, even though, while, as long as, until, because, etc), we are at the start of a dependent clause. Our sytem then removes the subordinating conjunction and converts the rest of the clause to an indenpendent clause. For our system to correctly know the parts of speech, we merely listed the part of speech for each word in our dictionary.

Similarly, for compound sentences, the *coordinating conjunction* indicates causality of the clauses. For example, when we have two independent clauses joined by "so," we should know that that the first clause causes the second to be true. This information could be useful for our CA to appropriately construct the CDs. This made us keenly aware of the flexibility of the word "and." Specifically, "and" is not always used as a coordinating conjunction; rather, it can also be used as a delimeter for a list of items. When it is used this way, the subject and verb are often implied. As is the case in other clauses too. So, our system also kept track of the last subject and verb. When our system had to separate a dependent clause and make it into an independent clause, yet it lacked a subject or a verb, it knew how to include them. For example, in our above sentence of "... she ordered the salad and he the steak," our system would terminate the clause when it sees the word "and." The words "he the steak" would then become its own independent clause, but only after we inject "ordered" as the verb.

In summary, our LexicalAnalyzer's tasks could be outlined as follows:

- read in the dictionary file
- replace all pronouns with their antecedents, when there's no ambiguity
- replace colloquialisms with the dictionary's translation
- transform dependent clauses into independent clauses
- transform all implied verbs and subjects into explicit ones

Figure 2: Example of a CD

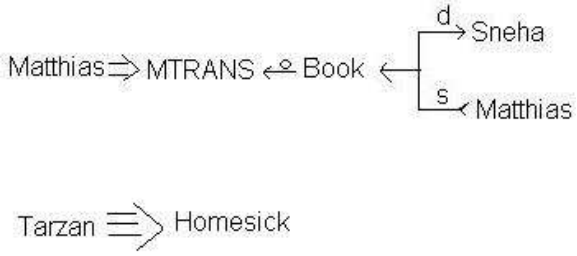


Figure 3: Two forms of a CDAtom

When analyzing a story, having a way to represent the semantic meaning of the story is useful. Conceptual Dependency (CD) diagrams give us a canonical representation of the semantic meaning of sentences. CD diagrams combine basic elements in a "molecular-structure" that any language can be converted to. Figure 2 is an example CD that represents Matthias throwing a ball at Sneha: One of the criticisms of CD diagrams is that there can be many ways to represent the meaning depending on how much detail is desired. Note that we could have described the ball being thrown with Matthias grasping the ball with his hand, flexing his elbow to pick it up, winding up for the throw, etc. Therefore, there is still some flexibility in the representation of meaning of a sentence.

We use a subset of the normal CD representation for our project. Since we were using JAVA, we broke up the representation into a set of 2 major classes: (1) CDAtom and (2) CD.

A CDAtom is a simplified CD that can have either of the two forms pictured in Figure 3. The first part of Figure 3 (the one with a double-arrow) represents the form for an action. The second part of the figure (the triple arrow), represents a state change. The primary form is the double-arrow (DA) form, which has five other fields:

- PP: person or object doing the object
- ACT: the action that is being performed
- OBJ: the object on which the ACT is being performed
- PPS: the person or place that is the originator of the ACT
- PPD: the person or place that is the destination or target of the ACT

Figure 4: A compound CD

Note that the OBJ field can either be a single String or an entirely new CDAtom. This allows compound CDs, which greatly extends our ability to represent semantic meaning. Figure 4 is an example of a CDAtom with a non-singular object.

the CD class represents the Conceptual Dependency of a single clause. It contains an ArrayList of CDAtoms, and can also be causally linked with other CDs. When our program generates the CDs representing the story, they are stored in an ArrayList as well. This is a version of episodic memory, where the events that occur are all stored to one area.

The Conceptual Analyzer (CA) does the work of converting the clauses generated from the Lexical Analyzer (LA) into the CD form. The basic idea we followed was keying off the verb in the clause to form the structure or skeleton of the CD. Then, once the general structure had been created, we tried to match parts of the clause to the proper fields in the CD. This was aided by hints that each verb contained, which guided the CA in figuring out what words in the clause should go where. A variety of hints were employed that would hint to the CA to look for an actor, object, or place. Since the lexicon had tagged all words with their part-of-speech, we can do a search through the clause and try to fit the proper clause in the right field. For example, if we try to generate the CD for "The hostess asked Bob to share a table," we would first get the clause that breaks up each word and identifies it by its type (article, actor, verb, object, etc.). Then, the CA starts from the beginning of the clause and finds the first verb which is "asked." Next, the CA looks up this verb in a side "semantic" memory that contains the hints for how this CD should look and what to look for. In this case it comes in the form of:

```
asked : !PP0,DA,MTRANS,!CDA
```

This is a comma-separated list of the values of the fields. The ones with the "!" represent hints to the CA, suggesting that it should find an actor for the PP0 slot and another CD to fill the CDA slot. The hostess is the first actor, so she is consequently put in the PP0 slot. The CDA slot uses the next verb which is "share." This in turn has its own structure to generate and attach. Note that a source and destination were unspecified, for it is the Script's responsibility to help fill these in later.

If we just relied on creating CDs from the clauses, then we would not be able to fully fill out the CDs. This is because vital, *implied* information is left out in the sentences. In normal stories, these details are not mentioned because they are too trivial and common, or it is obvious what the sentence is

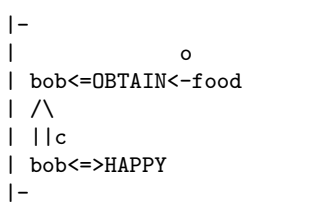
referring to. The elements that are obvious to humans can be modeled and enumerated by scripts. Scripts give a way of organizing all the information we see. It is a set of steps that normally occur as part of an action. By referring to our scripts, we can deduce missing information in a story. As such, it is another form of semantic memory that we utilize to create our CDs.

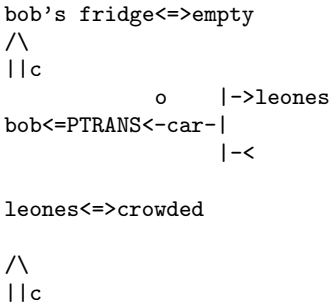
We implemented scripts so that we could fully resolve the ambiguous pronoun references such as "he" or "she." Moreover, it also helps us create more specific CDs by filling in more missing information. The general idea with scripts was to have some "variables" that each script contained that defined what was participating in the script. Therefore, in our Dating script there were two actors involved that needed to be resolved. Steps in the script were represented as CDs. A match and unification function was used that would try to find the CD that matched the closest and fill in the blanks that it could find.

Specifically, in our dating script, the two actors X and Y must be resolved. The first step in dating is introduction. The matcher sees that Alice is introducing herself and consequently binds "Alice" to one of the variables. Then, a later step matches Bob asking her to his place. This then binds Bob to the remaining variable and the script can now easily match the other steps of the dating process and resolve references to "he" or "she."

We also implemented a Restaurant script that details the normal steps of going to a restaurant. By using this, we can fill in CDs that had missing information that are so obvious in restaurant situations. Such as when a "waitress brought the menus," that normally means she brings the menus to the table. Although this is not stated in the clause and not implied by the verb, it is obvious to anyone who has gone to a restaurant. Thus, we see the power of scripts to put in common-sense knowledge to our CA.

The following CDs were generated by our CA after running our scripts.





As depicted in Figure 1, our approach to answering questions is similar to how we interpret the original story. First, the questions are fed into the Lexical Analyzer (LA). The LA sees that they are questions due to their ending in with a "?" It then looks at the beginning words of the sentence to determine if there should be any precendence as to the order that we try to resolve the answer. For example, if the sentence starts with "why did," we flag the clause to first attempt the upstream causality link. So, let's say that one of our original sentences read "John shot a cop, so he went to jail." Let's say that our question posed was "Why did John go to jail." Often times, it would be useful to first look at the clause that preceeded the matched sentence, "John went

to jail." Of course, inferences could be implied or happen as a result of a much earlier sentence. Regardless, the LA strips the questioning elements from the question and makes it into an independent clause. So, "Why did John go to jail" would be the resulting clause that is sent to the Conceptual Analyzer (CA). The CA transforms all of the clauses into CDs. For clarity purposes, let's call these resulting CDs the QCDs (question CDs). With this, we can then pass the QCDs to the answerQuestions() method of the original CA instantiation. Keep in mind that this original CA object contains a stored copy of all the story's sentences' CDs. So, the QCDs can be matched with the original CDs from the story. Depending on the type of question, we can know if we should return certain missing words that were matched, or if we should return a different sentence based on its causality. Regardless, we do not generate new sentences. Rather, our system finds sentences that fits or answers the questions with the needed information.

To provide a complete example, let's look at the original story. The dining script helped create the implied clause "Bob drove to Leones." So, the original CA contians a CD that represents "Bob drove to Leones." Let's call this CD_a. The question "How did Bob get to Leones?" is transformed to a QCD representing "Bob get to Leones." We then pass this QCD to the original CA, and it finds that the QCD and CD_a match in their representation. Yet, the missing word is "drive." The system then reports that "Bob drove to Leones."

Our system was able to correctly answer four questions:

1. How did Bob get to Leones?

   Bob drove

2. Why does Bob believe McCain should be president?

   Bob said McCain should be president because

   government health care in England is bad

3. Why did the hostess ask Bob to sit with Alice?

   Leones was crowded

4. How did Bob know Alice's name?

   Alice immediately introduced Alice

Chris Tanner and Leslie Choong shared equal responsibilities on brainstorming and agreeing on the overall design. Chris was the main force behind the Lexicon Analyzer and came up with the ideas of what it would do and how it would work. Similarly, Leslie was the main force behind the Concept Analyzer and construction of the CDs. Leslie implemented the usage of the scripts. Chris started the Question Answering part, but only together did we come up with how to make it all work–Leslie's expertise with the CAs was required. Chris badly wanted the CDs to be represented via ASCII art, and Leslie gave Piccaso a run for his money. Chris took the main responsibility of producing the paper using LaTex with figures, but we each wrote roughly equal amounts. **Our work is publicly viewable at http://cs.ucla.edu/ ctanner/cs263a/**

The project allowed us to see that NLP truly is a highly complex field of AI. In class, we are reminded that in order to have perfect NLP, all of the other problems of AI must be solved–problems such as planning, reasoning, search, vision, etc. We truly understand and agree with this statement now, for it is apparent that in order to create a system to thoroughly understand our very simple story, one would need to make usag of reasoning, planning, goals, scripts, unification, forward-chaining, backtracking, and more. Plus, if more complex questions were asked, we would need to add even more to our system.

We found it challenging to decide on a way to implement everything, and we were particularly careful not to provide our system with much help. We tried to make our system fully rely on matching and unifying in order to represent the sentences and questions. We were hesitant to construct our CDs in the same manner that we learned about in class, but it ended up being beneficial and was robust enough for us to work with it the way we wanted. Our first instincts were to make use of syntactical characteristics, such as including part-of-speech tagging, in order to enable our system to have something more clear to work with. It was satisfying to later learn that this was a common approach that others use. The usage of scripts served us well, so we were glad to learn about them in class.

Overall, this project was quite intimidating at first, yet, we were satisfied with our end results. In fact, as it all came together at the end, it was almost disappointing that we didn't have more time to try to answer more questions–especially since the hardwork of constructing the foundation had already been done. Nevertheless, it was a good experience, a very worthwhile project.