CSC384 Assignment 2 : Question Answering

Due Date: November 11th, 2005

Introduction

In natural language systems, parsing is the act of taking a phrase and understanding the underlying grammar behind it. For example, the phrase, "time flies like an arrow" is the classic example of a sentence that can be parsed in multiple ways:

Time	*flies*	*like*	*an*	*arrow*
N	V	Adv	Art	N
Adj	N	V	Art	N
V	N	Adv	Art	N

Some words, like "an" or "arrow" only have a single meaning in this context, whereas words like "time" can have many meanings, depending on the context of this sentence. However, its meaning may be unambiguous in other sentences, where the parsing of the sentence only assigns it one possible meaning.

In this assignment, you will construct a chart parser in Prolog that will understand and respond to natural English queries about the domain of inventors and inventions. You will construct a parse tree from a knowledge base and grammar of your own invention, whose robustness we will test

Chart Parser Algorithm

Chart parsers work by doing the following:

1. look at each word in a sentence one at a time
2. analyze the word's possible part-of-speech (POS) tag
3. build up as many grammar structures as possible with that POS
    a. if the POS could start a new grammar structure, then add this new structure to the knowledge base, along with the remaining POS tags needed to complete it
    b. if the POS can help complete a previously-created grammar structure, then remove the POS from that structures list of missing roles
    c. if the POS completes a grammar structure completely, then treat that structure like a POS on its own, and go back to Step 3.

Event Knowledge Base

The domain of this assignment is inventions. More specifically, the invention, inventor, date and location for each invention. The knowledge is stored in `event_frame` atoms in the following form:

```
event_frame('thomas edison', 'light bulb', '1879', 'menlo park new jersey').
```

Questions are valid if they ask for information about any of these four fields. For instance, your program should be able to take in questions like, "Where was the light bulb invented?" or, "What did Thomas Edison invent?" and respond with the appropriate answer.

A basic knowledge base of events is provided with this assignment, to help you along.

Design

Objective

To complete the Prolog program that takes in questions from the user and returns the answer to the question.

Starter Code & Expected Components

For this assignment, you are given the following components to work with:

- `events.pl` – A database of tuples that store invention information (inventor, invention, date and location).
- `intrface.pl` – A user interface tool that prompts the user for a question, and records the user's input as a list of strings, representing the words of the question that have entries in the lexicon.
- `semantic.pl` – A semantic analyzer, which extracts the invention information found in the user's question (assuming initial values of `'unknown'` for each, and replacing needed information with the value `'goal'`), storing them in the tuple `parseSemantics(4)`.
- `main.pl` – The main Prolog file, which loads the needed Prolog files, runs the components of the parser, and provides the interactions with the user

Deliverables

Submit the following Prolog files, so that they work with the predicates provided in the sample files, to produce the desired behaviour:

- `synproc.pl` – the file that implements the chart parsing algorithm
- `grammar.pl` – the file that defines the grammar rules for this domain
- `lexicon.pl` – the file that defines the dictionary of words for this domain

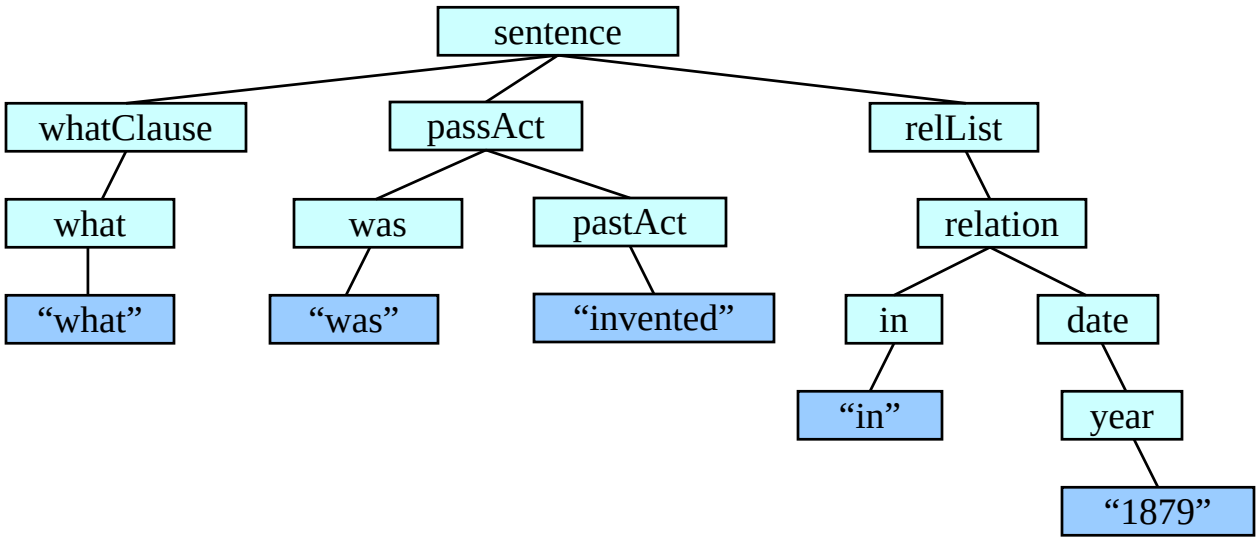
`synproc.pl`

Currently, the main Prolog file expects the following predicates in the `synproc.pl`, the syntax processing file:

- `produceChart(Wordlist)` – put together all the chart entries for the input question, represented as this list of words
- `getParseTree(Role, BeginPos, EndPos, ParseTree)` – given the role to parse (usually sentence) and the beginning and end positions of the section to analyze, bind `ParseTree` to the parse tree that can be constructed from the chart entries for this role and section. For example, the question "*What was invented in 1879?*" might bind `ParseTree` to the following:

```
[sentence, [whatClause, [what, [what]]], [passAct, [was,
[was]], [pastAct, [invented]]], [relList, [relation, [in,
[in]], [date, [year, [1879]]]]]]
```

Or, in graphical form:



`grammar.pl`

The grammar file contains the grammar rules for this parser, and nothing else. These rules should be in the form of `gram_rule(2)` atoms, where the first value is the non-terminal role and the second value is a possible decomposition of that non-terminal.

For example, given the question "*What was invented in 1879?*" from above, a possible high-level grammar rule could be:

```
gram_rule(sentence, [whatClause, passAct, relList]).
```

Terminal roles would not have to be decomposed, as they would match directly with the lexicon instead.

`lexicon.pl`

The lexicon stores all the words and the grammar roles of these words in `kb_rule(2)` atoms, like a dictionary for your parser. For some of the terminal nodes in the above example, you might encounter rules like the following:

```
kb_rule("1879", year).
kb_rule("was", was).
```

Other Prolog Files

You will also have to submit the sample code files, as there will undoubtedly be some changes that you will need to make in order for your implementation to work correctly. Although changes are expected, please do not change the entries in the events knowledge base, since we will need to know some basic events in order to test your parser.

Prolog files are submitted electronically at the following site:

`http://www.utm.utoronto.ca/submit`

In addition to the electronic submission, a short report (max 4 pages, 12pt font, double-spaced) will also be handed in during the class following the due date. The report will outline the design of your grammar and lexicon, design considerations you encountered when making your parser, extra features you may have added, as well as other design details you feel are pertinent for the marker to know when reading your program. Also include some basic analysis of the coverage of your parser, as well as any improvements you might make, given more time.

Mark Breakdown

60% Correctness
10% Style & Documentation
30% Report (Technique & Analysis)

Hints and Tips

- The course web page has useful links to Prolog resources, including a downloadable version of SWI-Prolog and related documentation for it.
- Start this one on paper first, and then move to Prolog. Write out all the questions that you can think of, and figure out how they break down into smaller sentence structures
- It's always good to get started early.
- It's not worth using other people's grammar rules. This assignment has a limited domain, so you should be able to construct a simple grammar on your own. For instance, labels like "noun" aren't nearly as useful here as labels like "actor" or "object". Besides, we recognize a borrowed grammar when we see one ☺
- Creating diagnostic tools for your parser is a useful thing to do for later. In particular, having a display tool for your chart entries will help you figure out whether it's parsing things correctly or not.
- Currently, the main Prolog file is expecting the predicates `produceChart(1)` and `getParseTree(1)` from the syntactic processor, as well as chart parser atoms called `chart(4)`. Try to keep those in mind when thinking about your parser design.
- The following two sections are guides to the expected behaviour of the program, and an API for the semantic processor. Enjoy.

Appendix A: Sample Execution

In order to give you an idea of what kind of behaviour we expect, the following section gives an illustration of the typical operation for the chart parser, given the question:

"*When did Thomas Edison invent the light bulb?*"

Slight ungrammatical or unacceptable inputs will also be shown, to illustrate a few error cases.

Normal Operation

The `consult` command is used on the `main` file. The user can then enter the question about a particular invention, inventor, invention date or invention location.

```
Welcome to SWI-Prolog (Multi-threaded, Version 5.4.7)
Copyright © 1990-2003 University of Amsterdam.

For help, use ?- help(Topic). or ?- apropos(Word).

?- consult(main).
intrface compiled, 0.06 sec, 3,844 bytes.
lexicon compiled, 0.16 sec, 23,620 bytes.
grammar compiled, 0.16 sec, 5,820 bytes.
synproc compiled, 0.22 sec, 9,112 bytes.
semantic compiled, 0.06 sec, 11,064 bytes.
events compiled, 0.05 sec, 4,092 bytes.
main compiled, 0.82 sec, 62,188 bytes.

Yes
?- start.


Welcome to the Natural Language Query System!
Please ask a natural English question about
any inventor or invention, and the system will
try to answer you as well as possible.

Please try to be as grammatical as possible.

Your question: When did Thomas Edison invent the light bulb?
```

Figure 1 : Screen Display for Initial NLQS Operations

The system then takes this input and processes it using the procedures described in the sections above. If the input question is valid, then the system will return a statement indicating the results of the query, including the input data and the goal data, as shown below. The user then has the option to see more details about the query, and if 'Y' is

selected, information about the understood word list, the number of understood sentences, the parse tree and the semantic information is shown.

```
Your question:  When did Thomas Edison invent the light bulb?


thomas edison invented the light bulb in 1879.


Would you like to see details about how your question was analysed? (Y/N)

The recognized word list is:
[when, did, thomas, edison, invent, light, bulb]

The total number of parsed sentences is: 2.
1 sentence used all of the words.

The parse tree for your question is:
     [sentence, [whenClause, [when, [when]]], [didAct, [didSubAct, [did,
[did]], [actor, [actorPart, [thomas]], [actor, [actorPart, [edison]]]],
[action, [invent]]], [object, [objectPart, [light]], object, [objectpart,
[bulb]]]]]]

The following semantic information was obtained:
     Actor    : thomas edison
     Object   : light bulb
     Date     : goal
     Location : unknown

Would you like to ask again? (Y/N)
```

Figure 2 : Screen Display for Query Response

Multiple Event Matches

When a query is made, there is a chance that more than one entry may be retrieved from the knowledge base. In these cases, all retrieved information is displayed, in case some or all of the event details might be of use to the user.

```
Your question:  Who invented the airplane?

There was no single response that matched to your query.
However, the following events contained related information that may
answer your question:

orville wright invented the airplane.

wilbur wright invented the airplane.
```

Figure 1 : Screen Display for Multiple Event Matches

Unacceptable Query Error

In the case where the user gives an input that is either composed of words that the lexicon cannot recognize or is structured ungrammatically, the system will return an error indicating that the user input was unacceptable. The user will have the option of asking again, but will not be able to see the details of the query since the inability to understand the question makes any gathered information useless.

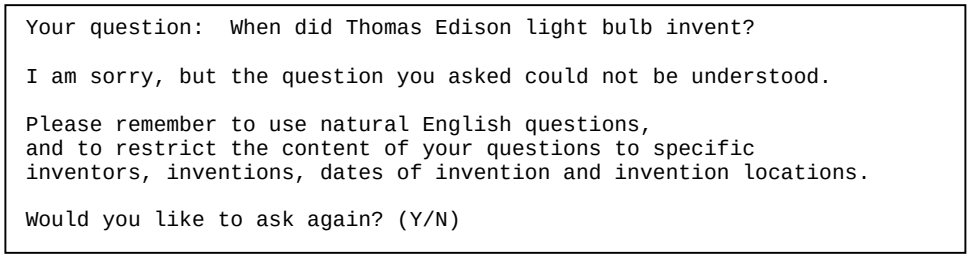


Figure 2 : Screen Display for Unacceptable User Input

Unmatched Query Data Error

Even though the user's input may be grammatically correct, it may not be able to match with an entry in the event knowledge base. This could be because the question contained conflicting information, one of the pieces of information was incomplete or the event knowledge base simply didn't have the event information. In this case the message returned to the user is slightly different, and the user is allowed a chance to view the query details, in case it might help in locating the error.

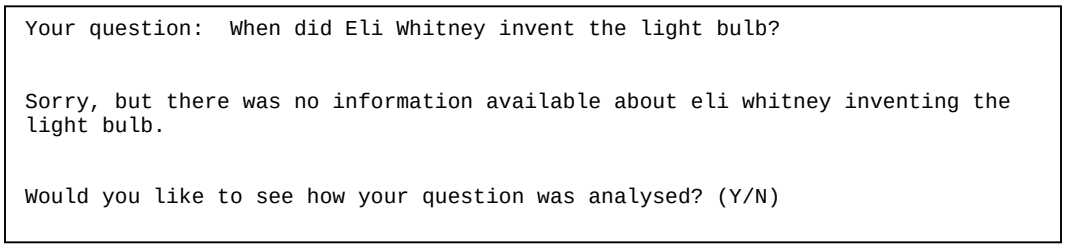


Figure 3 : Screen Display for Unmatched Query Data

Appendix B: API for semantic.pl

`analyseSemantics`	**Input:** The role type to be analyzed, the current parse tree to be analyzed. **Output:** None. **Operation:** Check if the current role type corresponds to the role type of the current parse tree. If there is a match, extract the words contained in this role. Fail otherwise. Then check the field in the semantic data structure that corresponds to this role. If the extracted words are longer than the words in the semantic data structure, replace the field in the semantic data structure with the extracted words.
`extractActor`	**Input:** The parse tree segment to be analyzed. **Output:** The current list of words making up the actor. **Operation:** Traverse the tree segment and extract all of the words from the leaves of the tree. Store all of the words in the actor word list.
`extractObject`	**Input:** The parse tree segment to be analyzed. **Output:** The current list of words making up the object. **Operation:** Traverse the tree segment and extract all of the words from the leaves of the tree. Store all of the words in the object word list.
`extractDate`	**Input:** The parse tree segment to be analyzed. **Output:** The current list of words making up the date. **Operation:** Traverse the tree segment and extract all of the words from the leaves of the tree. Store all of the words in the date word list.
`extractEnv`	**Input:** The parse tree segment to be analyzed. **Output:** The current list of words making up the location. **Operation:** Traverse the tree segment and extract all of the words from the leaves of the tree. Store all of the words in the location word list.
`makeQuery`	**Input:** None. **Output:** None. **Operation:** Retrieve the current semantic structure. Create a query based on the known information that was found in the fields of the semantic structure. Perform this query against the entries in the event knowledge base, and format them in a response to the user.
`createResponse`	**Input:** A list of the different elements of the event entries that matched with the retrieved semantic information. **Output:** None. **Operation:** If no entries were retrieved, display a message to the user indicating that no events corresponded to the specifications. If a single entry was retrieved, display a message indicating what the retrieved event was. If multiple entries were retrieved, print the existence of multiple corresponding events, and use the multiple event display predicate.
`displayResponses`	**Input:** A list of the different elements of the event entries that matched with the retrieved semantic information. **Output:** None. **Operation:** For each element in the lists, display the details of the event, and then loop back to continue with the rest of the list.

Table 1 : Table of Functions Used in Semantic Processor