Project Title: Construction Tools for Predicable English

April 2005

By Jeremy Smith

Supervisor: Roger England

1 Abstract

This dissertation describes a new kind of editor/parser/postparser combination, which allows humans to enter text structured in a way the computer can understand in terms of first-order logic or more specifically, Prolog predicates, after which a query can be made on the database, phrased in this manner. Because the project is experimental, it can only deal with a limited subset of English. It has been designed from the bottom-up to allow for a steady increase in complexity without damaging the earlier functionality. Based on the results from this project and the research done, it is suggested that this approach works quite well on limited domains, such as the test case of textual house descriptions in a database of Estate Agents. Also to be described will be other application domains where this could be used, known problems with the system, the design and implementation of a suitable editor, and the design and implementation of the postparser.

2 Acknowledgements

This project could not have been done without the support of the University of Huddersfield's support facilities, the project supervisor Roger England who came up with suggestions and ideas, knowledge of the subject, and who patiently read the draft copy of this document, and the author's friends and family.

3 TABLE OF CONTENTS

4 Introduction

4.1 Context

Here we are in the 21st century, with computers that still can't answer the kind of questions a 5-year old human has been able to for thousands of years. People have been struggling for decades to make computers intelligent, and one of the signs of intelligence is that of thought, as the old saying goes, "I think, therefore I am", which could perhaps be considered as the ability to answer questions.

What if a computer was able to learn enough information to be able to answer many questions?

One of the first things humans learn to do is to read and talk. This implies that a basis of intelligence is having enough information to reason with.

Most human knowledge is contained within the printed or written word. There is a huge information deficit between the amount of information that is scannable or typeable but computer-unreadable, and that which people working for projects such as Wordnet, can reasonably expect to structure into logical databases such as Prolog.

The reason for using Prolog for this project is that it is already established and can backtrack and perform complex searches with a database.

What if the computer could work out the structure all this written data itself? The resulting knowledge base would exceed, by factors of ten, all existing computer-readable structured data in the world.

The aim of this Textbase project is to allow computers to indeed parse text into a form where queries can be made, as in a database. This 'textbase' has the advantage of being readable by any human person. Unlike the syntax-strict nature of an SQL database, a textbase can be stored on a webpage, embedded within a story, spoken over a phone or stored within a legal document.

4.2 Problem

The original idea was to be able to read simple text, such as a children's book, and parse it into structured data. The problem was that the domain required would have been enormous, as children's books can cover a lot of topics. So, a use for the system was found that involved something with a limited domain, and house descriptions seemed to fit that need. If a larger domain was required, the existing system could be built upon incrementally to fit with it.

The problem used in this project is that of an Estate Agent's hundreds of houses in their database, which are composed of unstructured textual data. This needs

organising so that searches can be made for houses with specific features. e.g., 3 bedrooms and a garage or a bathroom on the ground floor. This information could be entered into a structured relational database, but the users may not be experts in doing this. A solution would be a text editor which allows only structured English text to be entered.

4.3 Clients

The clients would be Estate Agent's, whose business depends on matching their purchasers' requirements to the houses being sold by the vendors.

4.4 Users

The users would ultimately be the people using the Estate Agent's website. Hopefully, the staff would not require much training to use the software, which would allow them to enter house descriptions in a structured form.

4.5 Similar Applications

4.5.1 MIT's Metafor

This program allows people to write a computer program in natural language.

Although it is an interesting premise, the problem is that computer programs are much more complicated than a database of facts. With facts, it is possible to do what has been done with the text processor, TinyWP, the client interface to the system, which is to let the user edit their facts in realtime, with the results displayed below in a logical form. This would not be very easy while writing a non-static piece of software – if anything was wrong with the resulting code, there is the remote possibility of destructive failure.

It says in the Metafor paper that it does allow the user to see the results of what they are typing, but again, the program could be dangerous, and the user might not notice a destructive error, because of something they missed in the Metafor-generated source code.

One interesting thing mentioned in the Metafor paper is ConceptNet, which is a common-sense database on the Internet. Textbase exists just for these kinds of databases, so a little bit of research has been done into ConceptNet to see how

4.5.2 Attempto Controlled English

This has the same idea of Textbase, which is that of restricting the language so much that the computer can understand it. It is used for a similar purpose as Metafor; that is,

to write formal specifications of computer programs. The output from the process goes into the Discourse Representation Structure or DRS, which is a structured language similar to first-order logic.

This appears to be very similar to what Textbase does, but it was only during the final stages of development that this author came across it. A list of the negative aspects of ACE:

*ACE has a very limited vocabulary, unlike the Link Parser used in this project

*ACE is not publicly available as a download

It is gratifying to know that others are working in the same area of controlled language inputs.

5 Background and Literature Survey

5.1 Parser

A parser had to be found which would fulfill the following requirements:

1. Written in a non-Java language to allow integration
2. An open license to allow integration, that does not involve the GPL or any other open-source license
3. Suited the task.

By fortune, the first parser found, the Link Parser, fit all these requirements.

When I went to research a number of other parsers (Interactive Online CL Demos, 2004), I found that although the majority of them suited the task, they did it no better than the Link Parser and without exception, the licenses were all closed source, with a fixed cost to purchase these programs.

Although paying a fixed cost is fine for a commercial product, if it is closed-source as well, then integration is impossible.

For purposes of demonstration, I have put the following sentence into each online parser demo, with comments on how useful I consider its output to be. This sentence has been chosen as it is parsed successfully by CTPE and has the kind of predicate structure that is used in a Prolog domain.

> The house is blue and brown, contains a boat and is in Sweden.

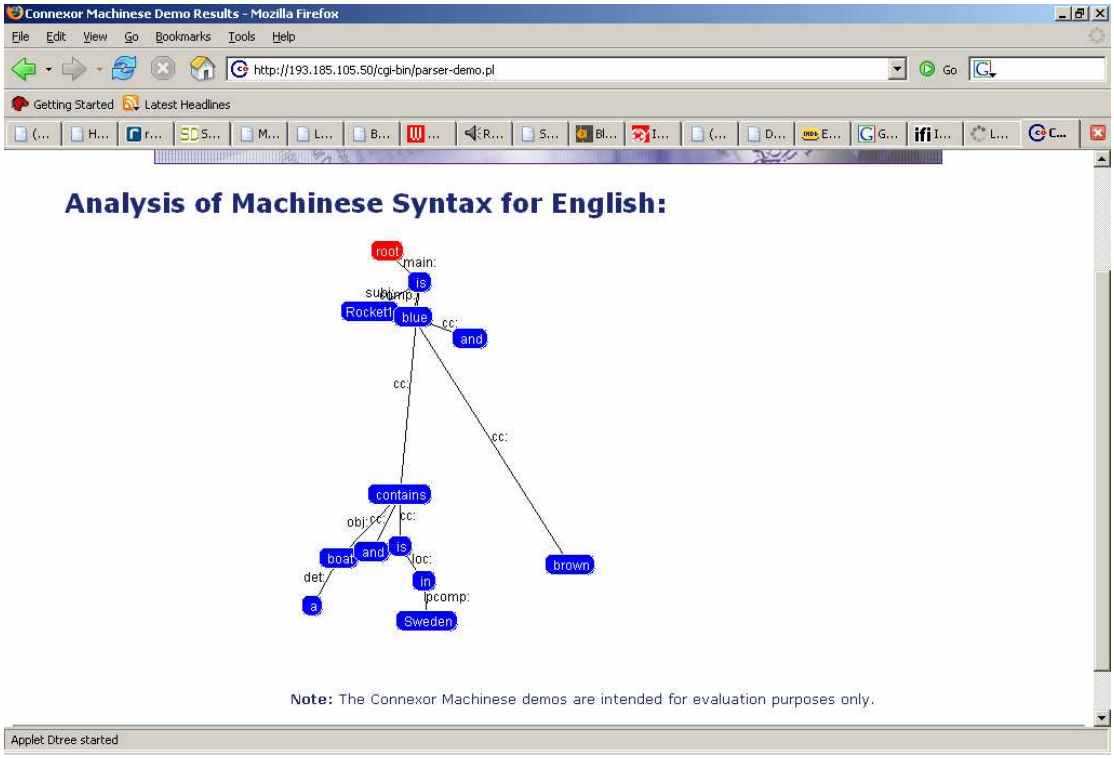
5.1.1 ENGCG. English Constraint Grammar Parser (Lingsoft, Helsinki)

```
"<*house>"
        "house" <*> N NOM SG  @SUBJ
"<is>"
        "be" <SV> <SVC/N> <SVC/A> V PRES SG3 VFIN  @+FMAINV
"<blue>"
        "blue" A ABS  @PCOMPL-S
"<and>"
        "and" CC @CC
"<brown>"
        "brown" <Nominal> A ABS  @PCOMPL-S
"<$\,>"
"<contains>"
        "contain" <SVO> V PRES SG3 VFIN @+FMAINV
"<a>"
        "a" <Indef> DET CENTRAL ART SG @DN>
```

```
"<boat>"
        "boat" N NOM SG  @OBJ
"<and>"
        "and" CC @CC
"<is>"
        "be" <SV> <SVC/N> <SVC/A> V PRES SG3 VFIN  @+FMAINV
"<in>"
        "in" PREP  @ADVL
"<*sweden>"
        "sweden" <*> <Proper> N NOM SG  @<P
"<$.>"
```

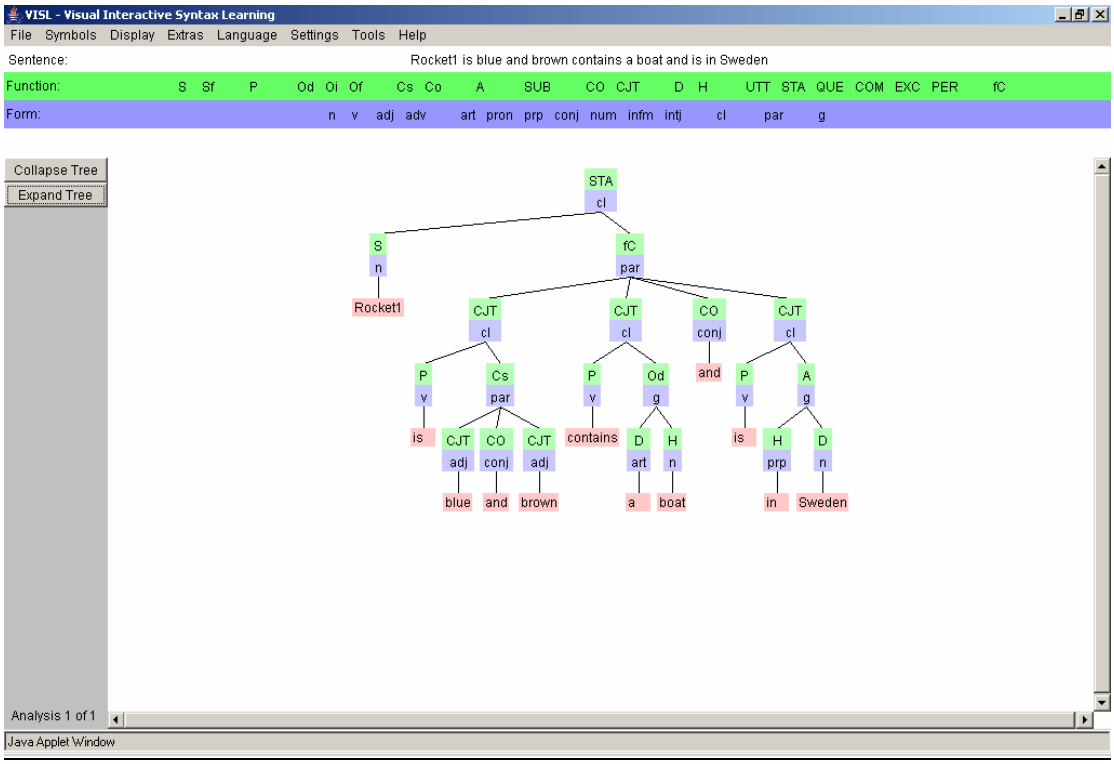
5.1.2 EngLite Parser. Functional Dependency Grammar Parser (Connexor, Helsinki)

This parser has an output which is hard to understand, with a tree of 'balls' that can be pulled around the screen, giving the impression of an old puzzle-based computer game.



5.1.3 English Grammar Page. Constraint Grammar Parser, Interactive Grammar Learning, Corpus Search (Odense University)

http://visl.sdu.dk/visl/en/parsing/automatic/trees.php



5.1.4 IPS 1.0. An interactive parsing system (GB-based Parser) (LATL, University of Geneva)

http://www.latl.unige.ch/

First, this website was in French and the language was unknown. The demo didn't seem to do anything.

5.1.5 Natural Language Parser Demo (Alpo Lind, Finland)

http://www.teemapoint.net/nlpdemo/servlet/ParserServlet

There is also a Wordnet parse and a 'TR' option, but the meaning of this could not be found.

Parses found: 1

```
                                <SENTENCE>
                                <CENTER>-|
                             <ASSERTION>   |
                             <SUBJECT>-|   |
                              <NSTG>   |   |
                             <LNR>     |   |
                           <NVAR>      |   |
House   ..................... <*N>     |   |
                               <VERB>-|   |
                              <VVAR>   |   |
```

is ..... <*TV> | | v: be
<OBJECT>-| |
<OBJECTBE> | |
<OBJBE> | |
<ASTG> | |
<LAR> | |
<AVAR>-| | |
blue ..... <*ADJ> | | | a: blue
<ANDSTG>-| | |
and ..... AND-| | |
<Q-CONJ>-| | |
<AVAR> | |
brown ..... <*ADJ> | | a: brown
<COMMASTG>-| |
, ... ,-| |
<Q-CONJ>-| |
<VERB>-| |
<VVAR> | |
contains ..... <*TV> | | v: contain
<OBJECT>-| |
<NSTGO> | |
<NSTG> | |
<LNR> | |
<LN>-| | |
<TPOS> | | |
<LTR> | | |
a ..... <*T> | | |
<NVAR>-| | |

boat ..... <*N> | | n: boat
<ANDSTG>-| |
and ..... AND-| |
<Q-CONJ>-| |
<VERB>-| |
<VVAR> | |
is ..... <*TV> | | v: be
<OBJECT>-| |
<OBJECTBE> |
<OBJBE> |
<PN> |
in ..... <*P>-| |
<NSTGO>-| |
<NSTG> |
<LNR> |
<NVAR> |
Sweden ..... <*N> | n: sweden
<ENDMARK>-|
. ...

5.1.6 NP Chunking Demo

(Erik Tjong Kim Sang, University of Antwerp, Belgium)
http://staff.science.uva.nl/~erikt/research/chunkdemo.html\

This demo is now inoperative.

5.1.7 Memory-Based Shallow Parser Demo (Tagging, Chunking, Subject-Object Detection) (Center for Language Studies, Tilburg University)

http://ilk.kub.nl/cgi-bin/chunkdemo/demo.pl

This demo is now inoperative.

5.2 Text Editor

Research was done to find an existing text editor which could be hooked up to a parser.

The editor had to be cross-platform to allow use on Windows and Unix, graphical to make it easy to use, have a license that allowed for both commercial use and modification and be written in C to allow the integration of a parser.

Unfortunately, a search on Freshmeat.net, which is a database of commercial and non-commercial software, revealed that no such editor was listed.

The editors that were capable and graphically-oriented were either closed-source, which didn't allow for integration, or released under a license such as the Gnu Public License which would require the source code for all the modifications to be released.

Fortunately, there existed a pre-written editor developed the year before, for another project, which has no license problems, being written by the author, and fulfills all the other requirements. The first version of the pre-written editor, TinyWP, was not reliable enough, so a complete rewrite was done, in C++ and based around a pixel-based cursor.

5.3 The Relation of Language to Thought and Meaning

The next stop on the research trail was to look at some classic texts on language itself, to find out how far people had got with figuring out how language's structure relates to the actual meaning, and what that meaning is.

The first essay in Noam Chomsky's book, "Language And Mind" (Chomsky, 1972), was about the history of this very subject.

Although being very old, and not often cited, this seemed to fit the topic perfectly.

Chomsky seemed to be saying that there were 2 classical approaches to the understanding of language:

- One was that everything you needed to know was semantically embedded in the text itself
- The other was that you could only decode the meaning of the text once it had been processed by the mind.

The author does not have an opinion on which is correct, but has learned that if you keep the language simple enough, there can be no ambiguity that requires extra decoding.

Thus, a simpler language fulfills both meanings.

5.4 The Storage Format

The storage format had to be researched, to find the one that was fast but had the ability to store Prolog predicates. The decision was made to use KIF, the Knowledge Interchange Format, which is a textual way of representing any kind of logic in a compatible manner.

My research in this field was first to read the description of it in the book Knowledge Representation by John Sowa (2000), followed by the journal article The KIF Of Death (Ginsberg, 1991), which was a document not written by the authors of KIF, but rather a couple of the NLP field's researchers.

In essence, they felt that KIF was not ready to be standardised, and should be described simply as an ongoing research effort.

Further, they suggested a more flexible format which could say "I use these extensions" and then the client reading the document and finding a sentence specifying that extension could either load those extensions, or in the absence of them, work around the absence of them by deciding to either ignore the sentence, or do something else close to the spirit of that extension.

This helps, because it is possible to use KIF as a format which can be made flexible enough, although for the purposes of this project it seems best to stick with first-order logic.

However, KIF has not been used in the prototype as a text-based file of Prolog predicates was found to be efficient and easy to use.

5.5 Efficient Prolog database system

Finally, some research was done on ways of storing Prolog data in a form that can be quickly processed, yet not depend on having the entire file loaded into memory.

Unfortunately, nothing was found except for a paper on integrating Sicstus Prolog with SQL database servers.

A quick test showed that 20,000 dummy predicates could be loaded into Sicstus, and searched, in under 2 seconds. The reason for such a large sample is that a typical estate agent's website may have thousands of houses for sale across the nation.

5.6 Client Survey

Finally, a visit was paid to a local Estate Agents, Whitegates. The staff were very helpful and gave of their time freely, to help where they could.

The current computer system was shown to the author. Their system allows the searching of a database by area, or owner. A user could type in the district or area and a long list of houses would come up. Then the user can click on the house and get details of the price, and search by price and the number of bedrooms. Ultimately, if the user wanted any more information, they had to read the brochure description.

Because of the visit, the author's suspicions were confirmed that the Estate Agents' system only allowed a few details to be searched, with the brochure description as the final arbiter of information.

Although saying there is a market for this project could be considered spurious, certainly it shows that if the brochure descriptions were parseable into first-order logic, searches on finer details could then be made, and the client research confirms this.

6 Design/Investigation and Research

6.1 Editor Design

6.1.1 Planned features

** indicates the feature has not been implemented, a + means it has been partially implemented*

- Any paragraphs prefixed with '+PRED:' are fed into the predicaliser, generating lines after that paragraph, with the text '+GENPRED:'. This allows a user to insert textual predicates anywhere in a document, and to disable them by deleting the '+PRED:' from the start of the line.
- +Infinite undo/redo which is storable on disk, and can be viewed as a tree. The tree facility makes a new branch when you undo and type without redo'ing
- Auto-updating global word count
- Word count of text before/after the current cursor position
- *Simple formatting, such as bold, italic, larger and smaller characters, different fonts
- *Insertion of pictures
- *The ability to compose a document from many sources
- *Little black lines below each line, like notepaper
- *A choice of cursors - circles, squares, underline, overline, crosshair
- +Mouse control; click to move the cursor to that point, select text to copy or paste, move text, select text and drag in to create a transclusion
- *Wheelmouse support, and horizontal/vertical scrollbars
- *+/- on the left to expand or shrink - ie, outlining
- *Macro language
- *Unicode support
- *Find and replace which can match a standard regular-expression, get the things in the brackets, and replace them with whatever you want (this feature lets you specify the bracketed match as $1, where $ is $D)
- +Auto-save as-you-type.
- *If you press ctrl+page down, the font size shrinks as it pages down
- +Python integration
- *If the document has HTML tags, the formatting tags are displayed, but the formatting is applied too (so you can delete the tag and the formatting goes away).

6.1.2 Architecture

By using a C++ vector instead of an array of strings, we remove all the problems associated with implementing linked lists.

vector<Line>Document;

For now a Line is a C++ string.

//A Line class contains:

We need a string formatter, which returns a vector of positions into the string.

```
vector<long>Formatter(string Line, int Wrap)
{
    vector<long>retval;
    return retval;
}
```

The screen starts at a particular character offset into a line. It does not start at a virtual line, because if the screen is resized, we want to draw from the same character position onwards so as not to confuse the user.

```
long ScreenLine;//Offset into Document
long ScreenCharOffset;//Offset into Document[ScreenLine]
```

6.1.3 Document object

This contains:

```
class Document
{
public:
    //Accepts a string of length 1 for Unicode usage
    virtual long GetGlyphWidth(string Char)
    {
        return 1;
    }

    //Accepts a single ASCII char for normal usage
    virtual long GetGlyphWidth(char Char)
    {
        return 1;
    }

    //Height of a character
    long GlyphHeight;
    long GetGlyphHeight()
    {
        return GlyphHeight;
    }

    void SetGlyphHeight(long argGlyphHeight)
```

```
    {
        GlyphHeight = argGlyphHeight;
    }

    long GetLineWidthInPixels(string Line)
    {
        long retval = 0;
        long c;
        for (c = 0; c < Line.length() c++)
        {
            retval += GetGlyphWidth(Line[c]);
        }
        return retval;
    }

    //Width of the screen
    long ScreenWidthPixels;
    //Height of the screen in lines
    long ScreenHeightLines;

    long CursorScreenLine;
    long CursorPosInPixels;
    vector<string>DocumentLines;

    //Goes along
    vector<long>ScreenFormatter(string Line)
    {
        vector<long>retval;
        return retval;
    }

/*  vector<Format>GetFormatting(string Line)
    {

    }*/

    //Draw the current screen from the top (DrawFrom), to the bottom
(DrawFrom+ScreenHeightLines), with the right-hand boundary being
ScreenWidthPixels. The cursor position is returned, and is in characters offset from
the top and left.
    vector<string>DrawScreen(long &CursorX, long &CursorY)
    {
        vector<string>retval;
        long c,d;

        for (c = 0; (c < DocumentLines.size()) && (retval.size() <
ScreenHeightLines); c++)
        {
            vector<long>ThisLine = ScreenFormatter(DocumentLines[c]);
```

```
            for (d = 0; (d < ThisLine.size()) && (retval.size() < ScreenHeightLines);
d++)
            {
                string Copy.append(DocumentLines[c],


            }
        }
    }

}
```

6.1.4 Cursor movement

The cursor is just a reference to a position into the screen. It is no more than MaxLines in the Y direction, and no more than MaxXPixels in the X direction, which is calculated by feeding the cursor's line (CursorScreenLine, the distance from the first line of the document, as explained below), not the offset but the string itself, into the GetWidthPixels(string Line) function.

```
long CursorScreenLine;//Offset into Document
long CursorPosInPixels;//Constant, convertable to an offset into
Document[ScreenLine]
```

Moving the cursor right or left will move along each pixel until the ScreenCharOffset increases by 1.

Right:If the cursor goes past the end of the virtual line, it will move to 0 pixels a line further down. Special case, bottom-right of the screen - move the cursor to 0 and ScrollDown().

Left:If the cursor goes to negative pixels (hits 0), it will move to MaxXPixels on the line before. Special case, top-left of the screen - move the cursor to MaxXPixels on the line before, and ScrollUp().

Ctrl+right/left (word skip) can be implemented with a macro that stops at white space.

Down:Cursor down will increase the CursorScreenLine without changing the CursorPosInPixels, and when it hits the bottom of the screen (MaxLines), the screen will ScrollDown().

Up:Cursor Up will decrease the CursorScreenLine without changing CursorPosInPixels. If it hits the top (0), the screen will ScrollUp().

6.1.5 Undo/Redo design

As the user types, the data typed is stored in a string buffer. Whenever they do a non-input action, such as to move the cursor, or backspace, the buffer is stored in memory, along with details of which line and position the buffer came from.

As the user deletes text, a string buffer stores the data deleted. When they do a non-delete action, such as moving the cursor or typing, a record is kept of which line the deleted buffer came from, and where.

Now, the undo record to be stored should state the line in the document where the typing or deletion took place. This is:

*Physical line (Line)
*Virtual line (LineLine)
*Offset into line that the text was deleted or inserted

We also need to keep undo records of these events:

*Line join (backspace, delete)
*Line add (Enter)

6.2 Predicate Generator Discovery and Design - Converting English text into Prolog Predicates

6.2.1 Post-Parser Design - Converting english text into Prolog predicates

6.2.1.1 Introduction

Here is an explanation of the reasoning used for this project in parsing different sentences into predicates. This process goes through the problem step-by-step. The reason for this process being explained in this document is to record the way in which the project author came to a solution. It could be helpful for people attempting the same thing.

6.2.1.2 How CTPE Works and How the Elements Slot Together

It's very easy in such a long document, to overlook the basic design of the software described. CTPE has a simple 3-pass system:

- The user types in a sentence in the editor
- The sentence is fed into the Link Parser, which turns it into a parse tree
- The parse tree is processed by the CTPE engine, and converted into textual Prolog predicates

As noted above, any paragraphs prefixed with '+PRED:' are fed into the predicaliser, generating lines after that paragraph, with the text '+GENPRED:'. This is a very portable idea, meaning the generated predicates can be saved to disk, but then deleted when the preceding '+PRED:' is itself deleted. There is no data that needs to be stored other than what is storable as plaintext.

The editor can be generic – any editor will suffice, such as Emacs; although the author doesn't use it, Emacs is sufficiently flexible to allow for this kind of usage.

The Link Parser is not generic, but it is released under a very flexible license. The author does not claim any authorship of this parser.

The author does, however, claim authorship of the CTPE engine, the backend software, and its design is not based on any existing program.

6.2.1.3 Analysis of a House Description/Sentence

In our chosen domain of house descriptions, there is a need to parse sentences such as:

"The house is near to Streatham Common and near to the High Street."

```
(S (NP The house)
   (VP is
       (ADJP (ADJP near
                   (PP to
                       (NP Streatham Common)))
             and
             (ADJP near
                   (PP to
                       (NP the High Street)))))
   .)
```

The adjective 'near' in the sentence has a PP subtree, which contains the words "to" and the noun-phrase "Streatham Common".

We would expect to see these predicates generated from this sentence:

nearTo(house, StreathamCommon)
nearTo(house, HighStreet)

Or we could need to parse this sentence:

"This fine house in Leeds has a bathroom."

```
(S (NP (NP This fine house)
       (PP in
           (NP Leeds)))
   (VP has
       (NP a bathroom))
   .)
```

And desired predicates could be:

in(FineHouse,Leeds)
has(FineHouse,Bathroom)

Each PP or VP is a predicate. An ADJP is a predicate which can contain other predicates.

Good grammar is partly about keeping words that can help reduce ambiguity. To demonstrate how taking out these words can confuse the parser, we take out the 'to' of 'near to' and say this:

The house is near Streatham Common and is near to the High Street.

We get this:

```
(VP (VP is
        (PP near
            (NP Streatham Common)))
```

As we can see, ‘near’ has gone from being an adjective to being a preposition. The problem is, if we have 'nearTo' as a predicate, then it won't be the same as 'near' in this example.

Or instead, this rule:

Rule 1:Ignore any words like 'to' when constructing predicate titles.

There is no other word we could put after 'near' that would make any sense other than 'to'.

6.2.1.4 The Use of Actions/Verbs

Here are the next 2 sentences. We are using the example of a verb happening to an object here. We could use houses, but houses don’t really do anything.

The ‘rocket world’ is a particular predicate domain, chosen as an example because the rocket world is a series of predicates that come with the Gipo software, and the author came across it and thought it would be interesting to be able to express such a domain in English.

The rocket moves from London to Aberdeen.
Move the rocket from London to Aberdeen.

```
(S (NP The rocket)
   (VP moves
       (PP from
           (NP London))
       (PP to
           (NP Aberdeen)))
   .)

(S (VP Move
```

```
        (NP the rocket)
        (PP from
            (NP London))
        (PP to
            (NP Aberdeen)))
    .)
```

The first is descriptive, the second is a command to someone.

What is a predicate? Is it a description or an order? Well, there are 2 types of predicate in Prolog: one is a fact, the other is a function or action.

Here, a NP precedes the VP (on the same indent), so that turns it from an action into a description. The second starts with a VP (move), so it is an action. It states what the action is (move), what the target is (the rocket), and has 2 arguments, from London and to Aberdeen. If we ignore the prepositions ‘to’ and ‘form’, we can say:

move(rocket,London,Aberdeen).

6.2.1.5 Conditional Sentences

Now, for conditionals. Unfortunately,

If the rocket is in Aberdeen, move it to London.

is not parseable by this parser into a complete linkage. The best it could come up with is:

```
(S If
   (S (NP the rocket)
      (VP is
          (PP in
              (NP London ,
                  (PP move it to
                      (NP Aberdeen)
                      .))))))
```

We can say this though:

Is the rocket in Aberdeen?

```
(S Is
   (NP the rocket)
   (PP in
       (NP Aberdeen))
   ?)
```

Another, "The rocket must be in London":

```
(S (NP The rocket)
   (VP must
       (VP be
           (PP in
               (NP London))))
   .)
```

The lack of conditionality would be a problem if we normally put all the predicates in the same block. However, there are 'pre-conditions', so we could just say:

A precondition is that the rocket is in London.
A precondition is that the cargo is in the rocket.
Another precondition is that the cargo is full.

This doesn't parse because it mentions preconditions as an adverb, which CTPE can’t handle yet, so we'll have to say it like this:

Precondition:

The rocket is in London.
The cargo is in the rocket.
The cargo is full.

This parses thus:

```
(S (NP The rocket)
   (VP is
       (PP in
           (NP London)))
   .)

(S (NP The cargo)
   (VP is
       (PP in
           (NP the rocket)))
   .)

(S (NP The cargo)
   (VP is
       (ADJP full))
   .)
```

And now we know the preconditions, which are:

in(rocket,London)
in(rocket,cargo)
full(cargo)

Notice how we ignore “is”, as it is a descriptive verb.

```
(S (NP The rocket)
   (VP 's
       (PP in
           (NP London)))
   .)
```

That seems to work well.

It is very likely that someone could represent the rocket world in fully parseable english, if someone so wished.

6.2.1.6 Rewriting and Deconstructing an Entire House Description

Moving on from that introductory example, now the task is to predicalise an estate agent's house description. First it's going to be rewritten into a simpler form of English. The aim of this task is not to parse any sentence, but to be able to accurately represent the data stored in the original house description. Thus, rewriting it is acceptable.

"A three bedroom back to back property, this offers an ideal opportunity for an investor or first time buyer. The accommodation briefly comprises of Lounge, Kitchen, Cellar, to the first floor there is a Master Bedroom and Bathroom and a further two bedrooms to the second floor. To the front aspect is an enclosed courtyard style garden. Close to Leeds City Centre. An early inspection is a must!"

This is rewritten as these sentences:

This property has 3 bedrooms and is back-to-back.
It would be an opportunity for an investor or first-time buyer.
This property has a lounge, a kitchen and a cellar on the first-floor.
This property also has a master bedroom, a bathroom, and 2 bedrooms on the second-floor.

```
(S (NP This property)
   (VP (VP has
           (NP 3 bedrooms))
       and
       (VP is
           (ADJP back-to-back)))
   .)

(S (NP It)
   (VP would
       (VP be
           (NP an opportunity)
           (PP for
               (NP an investor or first-time buyer))))
   .)
```

If we change it to "'a' first-time buyer", it looks like this, which is better parsed:

```
                (NP (NP an investor)
                    or
                    (NP a first-time buyer)))))
```

Here's the next sentence:

```
(S (NP This property)
   (VP has
       (NP (NP a lounge)
           ,
           (NP a kitchen)
           and
           (NP a cellar))
       (PP on
           (NP the first-floor)))
   .)
```

This is all good. All these nouns are on the first floor and are what the house has.

```
(S (NP This property)
   (VP (ADVP also)
       has
       (NP (NP a master bedroom)
           ,
           (NP a bathroom)
           , and
           (NP 2 bedrooms))
       (PP on
           (NP the second-floor)))
   .)
```

This structure is exactly the same as the previous, except for 'also', which we can safely ignore.

Here is another block of sentences to parse, again from an actual house description on our website:

> There is a garden at the front of the house.
> The house is close to Leeds' City Centre.
> The buyer should inspect the house early.

```
(S (NP There)
   (VP is
       (NP a garden)
       (PP at
           (NP (NP the front)
               (PP of
                   (NP the house)))))
   .)
```

This is a bit complicated but we would like to parse it into at_front(house,garden)

```
(S (NP The house)
   (VP is
       (ADJP close
             (PP to
                 (NP (NP Leeds ')
                     City Centre))))
   .)
```

close(house,Leeds'CityCentre)

```
(S (NP The buyer)
   (VP should
       (VP inspect
           (NP the house)
           (ADVP early)))
   .)
```

inspect(buyer,house,early).

The buyer is the first noun-phrase, and so is the subject of the sentence.
should is the verb and thus is the predicate title.
The verb phrase contains another verb phrase, how can that be? Should isn't really a verb, it's a preposition. eg.:

should(buyer,inspect(house,early))

6.2.1.7 The First Set of Rules

It seems that we might be able to formalise the rules from the analysis that has been done.

- Rule 1: Ignore any words like 'to' when constructing predicate titles.
- Rule 2: Each PP or VP is a predicate. An ADJP is a predicate which can contain other predicates.

Here's how we convert a parsed sentence into Prolog predicates. The parts marked with a * are parts that flout one of the above rules.

First, we need a description of what a Prolog predicate is. It is based on first-order logic. A sample predicate is “near(massachusetts,newyork). This means that Massachusetts is near New York. The nouns are uncapitalised because in Prolog, that indicates the use of a variable to be resolved.

- “near” is the ‘name’ of the predicate.
- “massachusetts” is the ‘first part’ of the predicate.
- “newyork” is the ‘second part’ of the predicate.

Any comments in the following testing of the rules are surrounded by square braces and underlined, thus: "[This is a comment]". The data below each rule shows where the rule would take effect.

Rule 3: There is always a subject "NP" after the first "S". This is the first part of the predicate.

```
      (S (NP This property)
      (S (NP The house)
*     (S (NP There)
      (S (NP It)
      (S (NP The buyer)
```

Rule 4: An S followed by followed by 'NP' is always followed by a Verb Phrase in all the cases in the above example. Let's not worry about ADVP or PP for now. Take note of the verb, this is the name of the predicate.

```
       (VP (VP has
       (VP would
       (VP (ADVP also)
*      (VP is
       (VP should
```

Rule 5: A Verb Phrase has a completing statement. If the argument to a verb phrase is an NP, this is the second part of the predicate. If not, ignore the first NP and use the second.

```
      (NP 3 bedrooms))
      (NP (NP a lounge)
      (NP (NP a master bedroom)
      (NP a garden)
      (NP the house)

*     (VP should [actually a preposition]
*       (VP inspect
*       (NP the house)
```

This deals with simple predicates. More rules will be needed for nested statements.

Rule 6)If there are multiple NP's within an NP, create a separate predicate for each one contained:

```
(S (NP This property)
   (VP has
       (NP (NP a lounge)
           ,
           (NP a kitchen)
           and
           (NP a cellar))
```

```
        (PP on
            (NP the first-floor)))
    .)
```

First, 'This property' is the first part of the predicate. The verb is 'has', then we come to a NP. This contains nothing but NP's, so each one is a separate predicate.

Next, we come to the PP (preposition). This says that the lounge, kitchen and cellar are on the first-floor. Because it follows the NP at the same 'indent' (viewed as monospaced ASCII) , it obviously applies to the NP, and thus, all the ones within it.

on(lounge,firstfloor)
on(kitchen,firstfloor)
on(cellar,firstfloor)
has(property,lounge)
has(property,kitchen)
has(property,cellar)

Another example:

'is' is a stop word, and a VP followed by a PP will eliminate the first part of the VP ('is'). Thus, the verb is 'in' and parse as normal.

```
(S (NP The house)
   (VP is
       (PP in
           (NP London)))
   .)
```

in(therocket,london)

Yet another example:

```
(S (NP This property)
   (VP (VP has
           (NP 3 bedrooms))
       and
       (VP is
           (ADJP back-to-back)))
   .)
```

First part: This property

The VP contains 2 VP's, separated by 'and' (though the and itself is irrelevant; we just know we have 2 sub-VP's)

has(thisproperty,3bedrooms)
is(thisproperty,backtoback)

6.2.1.8 Rewriting and Deconstructing another Entire House Description

“Spacious three bedroom cottage style semi-detached house situated in the popular city of Leeds which is famed for its castle and magnificent grounds.”

Here are the sentences, rewritten in a plain manner:

1.This house has 3 bedrooms, is spacious, cottage-style and semi-detached.
2.It is in the city of Leeds.
3.Leeds is famed for the castle and magnificent grounds.

6.2.1.8.1 Sentence 1:

```
(S (NP Leeds)
   (VP is
       (ADJP famed
             (PP for
                 (NP (NP the castle)
                     and
                     (NP magnificent grounds)))))
   .)
```

This sentence turns out to be hard to parse with our rules:

The first part: Leeds
VP contains an ADJP. As VP is 'is', we use 'famed' as the predicate title.
The parse tree goes too deep for our rules to cope with.

We are going to have to ignore that sentence, unless we can rewrite it further.

6.2.1.8.2 Sentence 2:

```
(S (NP It)
   (VP is
       (PP in
           (NP (NP the city)
               (PP of
                   (NP Leeds)))))
   .)
```

This sentence is too complicated for our rules. This is because instead of stating that Leeds is a city, it says that ‘It’ (the house) is in a city, which is of Leeds. Until we can come up with better rules, it is imperative that we rewrite the sentence to be more specific:

It is in Leeds.

Leeds is a city.

The rule here is that

Rule 7: If the predicate-generator finds a sentence which fits no rules, and thus nothing that matches the rules can be generated, then the user will receive no results and must rewrite it until they do. As more rules are added, that task will need to be done less frequently.

```
(S (NP It)
   (VP is
       (PP in
           (NP Leeds)))
   .)

(S (NP Leeds)
   (VP is
       (NP a city))
   .)
```

First.

First part: It
VP: is, cancelled out by in
Second part: Leeds
in(It,Leeds)

This predicate has ‘It’ as the first part. For the purposes of this example, we are going to assume that we previously defined ‘It’ as “This property”. The reason we have to assume this, is that the predicate generator in the CTPE software only deals with one sentence at a time, and does not have the scope to be able to store the last noun referred to, for use as ‘It’. In fact, a user of CTPE is presently advised to not use ‘It’, but they should include the whole noun again.

First part: Leeds
VP: is, nothing to cancel it out so it's a verb
Second part: A city
is(Leeds,City)

Finally

6.2.1.9 Rewriting and Deconstructing yet another Entire House Description

The following is surprisingly simple and fits the rules.

“This house has 3 bedrooms and is spacious, cottage-style and semi-detached.”

```
(S (NP This house)
```

```
(VP (VP has
        (NP 3 bedrooms))
    and
    (VP is
        (ADJP (ADJP spacious)
              ,
              (ADJP cottage-style)
              and
              (ADJP semi-detached))))
.)
```

First part: This house
VP contains 2 VP's: “has” and “is” so 2 predicates are created
1st predicate:
VP:has
Second part: 3 bedrooms

Result 1: has(thishouse,3 bedrooms)

2nd predicate:
VP:is, nothing to cancel it out
ADJP contains 3 ADJP's so there are 3 predicates here.

Result 2:is(thishouse,spacious)
Result 3:is(thishouse,cottage-style)
Result 4:is(thishouse,semi-detached)

Final result:

has(thishouse,3 bedrooms)
is(thishouse,spacious)
is(thishouse,cottage-style)
is(thishouse,semi-detached)
in(It,Leeds)

If we can detect that the previous noun subject was thishouse, we can identify the target of “It”:
in(thishouse,Leeds)
is(Leeds,City)

6.2.1.10 Preliminary Investigation into constructing the Prolog predicates from a Parse Tree

The aim of the project was a system to turn estate agents' listings into machine-readable text. Therefore, I chose a house description from the Fish4 Homes website and I converted it into the simplest kind of English possible without affecting the meaning.

Here is the original description of a house in Leeds:

"A three bedroom back to back property, this offers an ideal opportunity for an investor or first time buyer. The accommodation briefly comprises of Lounge, Kitchen, Cellar, to the first floor there is a Master Bedroom and Bathroom and a further two bedrooms to the second floor. To the front aspect is an enclosed courtyard style garden. Close to Leeds City Centre. An early inspection is a must!"

I converted it into these simple sentences:

- This property has 3 bedrooms and is back-to-back.
- It would be an opportunity for an investor or first-time buyer.
- This property has a lounge, a kitchen and a cellar on the first-floor.
- This property also has a master bedroom, a bathroom, and 2 bedrooms on the second-floor.

Then I converted each sentence into a parse tree (using my editor) and analysed the results.

```
 (S (NP This property)
   (VP (VP has
           (NP 3 bedrooms))
       and
       (VP is
           (ADJP back-to-back)))
   .)
```

Analysis: This starts with a Noun Phrase, and has a VP which has 2 sub-VP's. There is a subject Noun Phrase (This Property) on which is operated a series of Verb Phrases. This is a simple pattern to spot:

First, there is just one noun-phrase, so that is the 'first part' of the predicate (ie, has(This_property,3_bedrooms).

Rule 8: If an NP is followed by one VP, which consists solely of children which are also VP's, then pass the first part of the predicate, and the pointer to the VP node, to a function which parses the verb phrases and returns a list of textual predicates. For each VP within the root VP, it gets the title (the first part of the VP), and the second part of the predicate (the phrase that comes after the first part of the VP).

This pattern, once spotted, is easy to implement in code, and is done so in CTPE.

Here is the next sentence:

```
(S (NP It)
   (VP would
       (VP be
           (NP an opportunity)
           (PP for
               (NP an investor or first-time buyer))))
   .)
```

'and' and 'or' are useful for splitting the phrase up so that the parser can cope with it. I'll come to this later.

Rule 9: Any 'or' or 'and' must be prefixed with 'a', 'an', 'the', etc to help the Link Parser split up the sentence properly into multiple noun-phrases.

First off, the NP is 'It'. This refers to the sale of the house, really, but in logical terms probably refers to the previous NP, "This Property". ‘Would’ is treated as a Verb, which contains an NP and a PP. The NP is what usually comes after the VP. What about the PP? Well, the verb 'an opportunity' could be followed thus:

*For
*In
*With
*After
*etc

It's a preposition. It's easy to parse this, but how do we represent this in Prolog? For now, treating it like a verb, we'll just take the PP's name ("for") and put it as the predicate title:

opportunity(ThisProperty,for(investor))
opportunity(ThisProperty,for(first_time_buyer))

The reason for this is that a predicate can be a fact, and a fact with a verb followed by a noun, or a preposition followed by a noun, takes the same form as a predicate.

The rule is:

Rule 10: If a noun-phrase is followed by a preposition (PP) and prefixed by a verb phrase (VP), create a sub-predicate with the title of the second part of the outer predicate being a predicate with the title of the preposition.

```
(S (NP This property)
   (VP has
       (NP (NP a lounge)
           ,
           (NP a kitchen)
           and
           (NP a cellar))
       (PP on
           (NP the first-floor)))
   .)
```

This should work with our current rules, the ones defined as rules 1- 4 above. We'll assume in the following predicates that there is only one lounge, kitchen or cellar in the system.

has(ThisProperty,lounge)

has(ThisProperty,kitchen)
has(ThisProperty,cellar)
on(lounge,first-floor)
on(kitchen,first-floor)
on(cellar,first-floor)

```
(S (NP This property)
   (VP (ADVP also)
       has
       (NP (NP a master bedroom)
           ,
           (NP a bathroom)
           , and
           (NP 2 bedrooms))
       (PP on
           (NP the second-floor)))
   .)
```

Exactly the same as the previous, except for 'also'.

There is a garden at the front of the house.

```
(S (NP There)
   (VP is
       (NP a garden)
       (PP at
           (NP (NP the front)
               (PP of
                   (NP the house)))))
   .)
```

This is a bit complicated and we would like to parse it into at_front(house,garden). I'm leaving this complicated structure until later. [fix this]

The house is close to Leeds' City Centre.

```
(S (NP The house)
   (VP is
       (ADJP close
             (PP to
                 (NP (NP Leeds ')
                     City Centre))))
   .)
```

close(house,Leeds'CityCentre)

```
(S (NP The buyer)
   (VP should
       (VP inspect
           (NP the house)
```

```
            (ADVP early)))
    .)
```

should_inspect(buyer,house,early).

6.2.2 How I started to implement this in C++

1. I first get a pointer to the root of the constituent tree, this is stored in the variable ‘Node *ptr’, where the ‘’ is a C pointer.

2. We expect an S, followed by an NP. This is the most basic structure used in a fact.

3. Get all the words in the first NP, this is the first part of the predicate (the subject).

4. After the NP, all we can deal with after this is Verb Phrases, so I describe that here.

5. The first words in this VP are the predicate's title if the VP has only a PP as a child, or no children.

6. If the VP has a child that is a VP, the pointer variable ptr is set to the VP's child and the algorithm will follow the tree via ptr->next, calling a recursive function on the Noun Phrases in the Verb Phrases, until there are no more VP phrases. First, it is a good idea to check the entire chain from the ptr root is VP's, so that we don't come across other link types. The recursive function takes the node, the 'first part' of the predicate (which is the first NP, directly after the S), and the predicate title, which is ptr->child->label.

Currently the project is restricted to being able to cope with the following structures. Further developments are possible (see section []):

- A case to deal with a single NP/VP
- A case to deal with a single NP and a VP with multiple VP children
- A case to deal with a chain of NP's (but no verbs)
- A case to deal with a single NP, and a VP with children, each of which has a string of NP's or ADJP's inside it

What is needed is a more robust scanning algorithm:

- It can deal with a string of NP's, which are the subjects of the VP or multiple VP's below.
- It can deal with Adjectives and Prepositions

6.2.3 Throwing information away and the need for Prolog

The main problem with converting english into predicates is that information can get lost. English contains the exact meaning, such as "The house is very close to Leeds' City Centre", but in Prolog it could end up as "close(house,Leeds'CityCentre)". There is one solution, which is to tag the words.

"close(very(house,Leeds'CityCentre,NP,NP),PP)

If we do this too much, it makes the use of Prolog irrelevant. In that case, we might as well just store a database of constituent trees.

6.2.4 Handling Questions and Queries

6.2.4.1 Introduction

Most questions start with the following words [type 1]:

- Where
- When
- Who
- Why
- How
- Which

Here's some examples of type 1 questions:

- Where is the cat?
- How does it work?
- Why does it do that?
- Which house do you mean?
- When did that happen?
- Who threw that?

Then there are other types of question [type 2]

- In which house was Bob born?
- Near which town is there a statue of a famous writer?

These type 2 questions have a preposition, followed by the above type of query.

Parsing these questions is fairly easy - the problem is how to represent them as Prolog predicates.

Let's start with the simplest.

6.2.4.2 “Where is the cat?”: How do questions work?

In Prolog, the data would be stored as:

is_in(cat,Tipperary)

So the search query would have to be

is_in(cat,X)

The problem is that "is_in" could be a number of other predicates; "is_near", "underneath", "is_inside". The best solution seems to be to take all those prepositions (near, underneath, inside) and shrink them into one of the question words above, ie, "where".

where(cat,Tipperary)

Perhaps

where(cat,Tipperary,in)

Similiar predicates:

Query: How does it work?
Prolog: work(X,it)

Query: Where is the cat?
Prolog: where(X,cat)

The problem here is that someone or something "is" a preposition, but distinguishing between prepositions requires a database that puts prepositions into the following category:

- where(Jim,X): shop
- what(Jim,X): shopkeeper
- who(Jim,X): X contains various facts about Jim, such as where he is, what he does
- why(Jim,became a shopkeeper): Jim likes shopkeeping.
- how(Jim,become a shopkeeper):inheritance

The easiest prepositional phrases to deal with are where and what. Who is not a simple question - neither are why or how. Why and how require a reason (a sentence), while who is not a one-predicate answer. How also requires a whole sentence.

However, a house description only contains facts, not reasons. Nobody gives a reason for a house having a backyard, it just has one.

- This house is blue because the owner doesn't like pink

Instead we'd see:

- This house is blue

Or

- This house is situated in Colorado

A possible solution to this is, for every predicate, we make 2 versions:

- situated(This_house,Colorado)
- is(This_house,situated_in(Colorado))

- blue(This_house)
- is(This_house,blue)

Then the user can search with "is" to find all the properties of it.

Query: is(This_house,X):
Result: blue; situated_in(Colorado)

However, this makes the type of query irrelevant, if "Where", "What" and "When" each come up with all the words "London", "Cat" and "Tuesday".

One solution is to give the user the choice of selecting what type of word a preposition is.

Where prepositions:

- near
- underneath
- on top of
- beside
- outside
- inside
- behind
- in front
- north/south/east/west of
- between (2-ary)
- in proximity to

6.2.4.3 How the program is going to ‘answer’ questions

6.2.4.3.1 Disclaimer

It should be noted that the CTPE’s backend engine is very poor at answering questions, at present. It can only answer the most basic questions. However, the theory below, and the rules it comes up with, could be implemented in C++ the same way in that the CTPE can currently handle multiple facts and combinations of

statements. The theory behind turning questions into predicates works, but the code doesn't.

6.2.4.3.2 How CTPE will answer questions

Here is how the CTPE is going to create questions that Prolog can answer, given an existing Prolog database which may have been created with CTPE originally.

Here are possible queries that an estate agent may have to answer:

1. How many bathrooms are there?
2. Does the house have central heating?
3. Is the house near Castleford?

Sentence 1:

```
(S How many bathrooms
   (VP are
       (PP there))
   ?)
```

has(house,bedrooms(X))

Sentence 2:

```
(S Does
   (NP the house)
   (VP have
       (NP central heating))
   ?)
```

have(house,central_heating)

Sentence 3:

```
(S Is
   (NP the house)
   (PP near
       (NP Castleford))
   ?)
```

near(the_house,Castleford)

However, these questions are regarding a specific house. Here are some general queries:

1. Where is a house with 3 bedrooms?
2. Is there a house with central heating?
3. I need a house near Castleford

Sentence 1:

```
(S Where
```

```
    (VP is
        (NP a house)
        (PP with
            (NP 3 bedrooms)))
    ?)
```

Let's say that someone entered this data previously, using CTPE or another application:

"This house has 3 bedrooms"

```
(S (NP This house)
   (VP has
       (NP 3 bedrooms)))
```

The structure of query 1 is very similar to the structure of the data that has been entered. This means that they will both parse to approximately the same Prolog predicates. If the question is parsed into these predicates, with the 'X' signifying the 'which' to be resolved:

has(X,3_bedrooms)

, then it will find the predicates entered in the "This house has 3 bedrooms" sentence above:

has(this_house,3_bedrooms)

The question will be answered from the Prolog command-line thus:

X = this_house?

Sentence 2:

```
(S Is there
   (NP a house)
   (PP with
       (NP central heating))
   ?)
```

CTPE parsed this sentence as:
is(X,there)
X(with,central_heating)

Sentence 3:

```
(S (NP I)
   (VP need
       (NP a house)
       (PP near
           (NP Castleford))))
```

[CTPE was unable to parse this sentence]

Sentences 2 and 3 have approximately the same structure.

6.2.4.4 Basic set of questions the program can answer

As seen before, if we start off with the least-ambiguous structures, that is a good starting point.

First, a single verb and a single noun-phrase.

> I want a house with central heating.

Multiple noun phrases

> I want a house with central heating and a shower

No relevant question seems to have a verb phrase after the first verb. IE,

> I want a house with central heating that is semi-detached

> Is there a house that is blue?

```
(S Is there
   (NP (NP a house)
       (SBAR (WHNP that)
             (S (VP is
                    (ADJP blue)))))
   ?) [See below]
```

As a side-note, from the Link Parser documentation on SBAR's [http://www.link.cs.cmu.edu/link/ph-explanation.html]:

> "SBARs are generated in several cases: Embedded clauses with "that", relative clauses with a pronoun, dependent clauses with a conjunction, and indirect questions. Example: "He said [SBAR that he was coming]."

And a WHNP is:

> "A one-word constituent to contain relative pronouns: "The dog [WHNP who] chased me was big"."

Now for verification questions. These range from simple fact checking to verification that someone is doing something. The 'Jill' in these examples is from the Jack and Jill story.

> Is Jill annoyed?

```
(S Is
   (NP Jill)
   (ADJP annoyed)
```

```
   ?)
```

Is Jill annoyed and angry?

```
(S Is
   (NP Jill)
   (ADJP (ADJP annoyed)
         and
         (ADJP angry))
   ?)
```

Is the house blue?

```
(S Is
   (NP the house)
   (ADJP blue)
   ?)
```

Very simple. Let's have a rule that:

Rule 11: S/Is followed by NP/ADJP is a question, where the generated predicate is: is(the_house,blue)

These confirmation questions, asking if a fact is true, should be straightforward to parse. The next step is to convert this question to a predicate and see if the answer is true in Prolog, given the working dataset.

This leads onto the next possibility, another type of question in the English language, one involving a range of objects fitting a certain category.

```
(S Is
   (NP any house)
   (ADJP green)
   ?)
```

This would match as:

is(Anyhouse,green)

This tells us if there exists a house that is green, but also finds all green houses. We're using the same structure as before, but the NP is prefixed with 'any'.

```
(S Is there
   (NP (NP a house)
       (SBAR (WHNP that)
             (S (VP is
                    (ADJP (ADJP green)
                          and
                          (ADJP comfortable))))))
   ?)
```

This is more complex. Before we look at how it could be dealt with, let's consider the predicates that should result:

is(X,a_house),
is(X,green)
is(X,comfortable)

This a simple series of predicates.

The problem with parsing the above parse tree structure is the 'SBAR' and 'WHNP' word types which are getting in the way. If we remove the offending word, 'that', then the sentence is reduced to:

```
(S Is there
   (NP a
       (ADJP green and comfortable)
       house)   ?)
```

Which still doesn't post-parse with the current post-parser, but this can be left for future versions. For now, all that needs to be understood is that the SBAR and WHNP are something that we can safely ignore, because the word 'that' is superfluous.

6.2.4.5 The kind of questions we can answer with the rules we have

I want a house near Leeds

```
(S (NP I)
   (VP want
       (NP a house)
       (PP near
           (NP Leeds))))
```

I want a house near Leeds that is blue

```
(S (NP I)
   (VP want
       (NP (NP (NP a house)
               (PP near
                   (NP Leeds)))
           (SBAR (WHNP that)
                 (S (VP is
                        (ADJP blue)))))))
```

The documentation says that SBAR appears before 'that' and is a 'that' phrase. So in other words, as the SBAR comes under the NP "a house near Leeds", this means that the house near Leeds is whatever is in the SBAR.

```
           (SBAR (WHNP that)
```

```
                    (S (VP is
                           (ADJP blue)))))))
```

Inside the SBAR, we get WHNP. We ignore this and go on to the sentence below it (is blue).This S sentence parses (using the rules) into a Prolog query, thus:

is(the_house,blue).

So, the user wants a house near Leeds that also has the property that it is blue. We represent that thus:

near(X,leeds),is(X,blue)

That's fairly easy.

As a footnote, WHNP is a constituent (ie, a word) to contain a relative pronoun (which, who, etc). The fact that the Link parser can identify them is quite promising.

Moving on, let's try another query:

```
Linkage 1, cost vector = (UNUSED=0 DIS=0 AND=0 LEN=15)
(S (VP Tell
       (NP me)
       (PP about
           (NP a house))
       (PP with
           (NP 3 bedrooms))))
```

This starts with a verb, and is thus a command, such as "Dig the garden before 4'O clock. The user is telling us to do something for 'me'. As all we can presently do is answer questions, we could assume it's a question, but instead of that, we see there is an "about" preposition, so that really determines that this is a question. What is the question about? Well, the subject is a house. Oddly, the 'with' is below the PP not the NP, which is odd. If we find an alternative linkage using the parser's Application Programming Interface or 'API' (the parser's functions that we can call directly), we get:

```
Linkage 2, cost vector = (UNUSED=0 DIS=1 AND=0 LEN=11
(S (VP Tell
       (NP me)
       (PP about
           (NP (NP a house)
               (PP with
                   (NP 3 bedrooms))))))
```

That's better. This has a lower distance than the previous linkage.

Rule 12: Always choose the linkage with the lowest distance, the "LEN" property.

This makes it easy now.

A preposition 'about' means this is a question
The subject is a house, and the subphrase is “with 3 bedrooms”
This generates the following Prolog query (the ‘,’ signifies a conjunction, or ‘and’:

has(X,3 bedrooms), is(X,house)

The user is asking for an item that has the property of having 3 bedrooms. The preposition 'with' ensures that its subject is a property of its parent.

The problem is how to determine which alternative linkage we need. There are 2 options:

*Find the linkage with the least number of constituents in the parse tree (the ‘LEN’ property which isn’t presently shown in the CTPE editor) – this is likely to be the least ambiguous. The reason for this is unknown – all that can be observed, is that the linkage with the lowest cost is always the least ambiguous.

*The user could insert a comma after ‘a house’ if they find the results unsatisfactory.

6.2.4.6 Possible query phrases

Here I'm going to try and exhaust every possible query the user could ask about finding a house with 3 bedrooms, then the same but with a porch as well.

Tell me which houses have 3 bedrooms

```
(S (VP Tell
       (NP me)
       (SBAR (WHNP which houses)
             (S (VP have
                    (NP 3 bedrooms))))))
```

Tell me about a house with 3 bedrooms

```
(S (VP Tell
       (NP me)
       (PP about
           (NP a house))
       (PP with
           (NP 3 bedrooms))))
```

I want a house with 3 bedrooms

```
(S (NP I)
   (VP want
       (NP a house)
       (PP with
           (NP 3 bedrooms))))
```

Give me a house that has 3 bedrooms

```
(S (VP Give
       (NP me)
       (NP (NP a house)
           (SBAR (WHNP that)
                 (S (VP has
                        (NP 3 bedrooms)))))))
```

Which houses have 3 bedrooms

```
(S Which houses
   (VP have
       (NP 3 bedrooms)))
```

Where is there a house with 3 bedrooms

```
(S Where is there
   (NP a house)
   (PP with
       (NP 3 bedrooms)))
```

What houses have 3 bedrooms

```
(S What houses
   (VP have
       (NP 3 bedrooms)))
```

Is there a house with 3 bedrooms

```
(S Is there
   (NP a house)
   (PP with
       (NP 3 bedrooms)))
```

Do you have a house with 3 bedrooms

```
(S Do
   (NP you)
   (VP have
       (NP a house)
       (PP with
           (NP 3 bedrooms))))
```

There's one common thread here. "a house with 3 bedrooms" always parses as NP/PP/NP.

The other pattern is SBAR/WHNP/S/VP/NP for "which houses have 3 bedrooms". SBAR/WHNP is explained above; here, instead of 'that', the phrase is 'which houses'.

```
       (NP a house)
       (PP with
           (NP 3 bedrooms))))

       (SBAR (WHNP which houses)
             (S (VP have
                    (NP 3 bedrooms))))))
```

This means that there are 2 main query types:

*NP: One with a subject, preposition and target property (we could replace 'with' with 'near' and it would be the same structure, so the preposition changes)
*SBAR: One which identifies a subject, and then specifies in a new sentence, what the preposition is (have) and the target property (3 bedrooms).

The part before the query should always contain one of these

What
Is
Do
Where
Which
About
That
Does

(and so on). Any of these identifies the sentence as being, not a statement (such as 'This house has 3 bedrooms'), but a query, 'Which house has 3 bedrooms'.

6.2.4.7 Algorithm to check if it's an NP/PP/NP-based question, or an NP/VP/NP-based question

First see if there are any words after the 'S' and before the 'NP'. If there are, then they precede the noun phrase and thus it is probably a question.

The first NP is the type (or name of a single item) of thing being asked about. IE, a house, a person or a topic. After this is the preposition or verb; this is the predicate title. The last thing is the query itself.

```
(S Does
   (NP this house)
   (VP have
       (NP 3 bedrooms)))
```

‘Does’ indicates it's a question
this house: thing being asked about
VP have: have(this_house,...)
NP 3 bedrooms: have(this_house,3 bedrooms)

```
(S Do
   (NP you)
   (VP have
       (NP a house)
       (PP with
           (NP 3 bedrooms))))
```

‘Do’ indicates that it's a question

you: thing being asked about
VP have
NP a house with 3 bedrooms: have(you,have(this_house,3 bedrooms))

The above is a complicated sentence.

```
 (S (NP I)
   (VP want
       (NP a house)
       (PP with
           (NP 3 bedrooms))))
```

This isn't a question, it's a statement.

```
(S Which houses
   (VP have
       (NP 3 bedrooms)))
```

Which houses: it's a question
?: thing being asked about
VP have
NP 3 bedrooms
have(X,3 bedrooms)

Here, houses is not coming out as a noun properly. Let's say that if we get an S with words after it, but a VP after that, then the remaining words of the S are the noun. This is messy, but it works for the time being.

6.2.4.8 Testing the Algorithm

```
(S Where
   (VP is
       (NP a house)
       (PP with
           (NP (NP a porch)
               and
               (NP a bathroom))))
   ?)
```

S has a word directly after it, ‘Where’, which triggers a query.
NP is "a house" – this is the thing being asked about
Followed by Preposition "with" – this is the predicate title
Then we get the query subjects which are "a porch" and "a bathroom"

is(X,house), with(X,porch), with(X,bathroom)

That sentence parsed well using this algorithm. Let's try another.

```
(S Is there
```

```
   (NP a house)
   (PP (PP with
           (NP 6 bathrooms)
           and
           (NP 20 bedrooms))
       and
       (PP near
           (NP Leeds))))
```

S has words after, "Is there".
NP is "a house" - thing being asked about
Followed by prepositions, with and near.
Then we get the subqueryies "6 bathrooms, 20 bedrooms" and "leeds"

is(X,house), with(X, 6 bathrooms), with(X,20 bedrooms), near(X,Leeds)

Seems straightforward.

Is there a house with a green cooker in the kitchen?

```
(S Is there
   (NP a house)
   (PP with
       (NP a green cooker))
   (PP in
       (NP the kitchen))
   ?)
```

We're going for minimum linkages here, and that cooker isn't in the kitchen (it's the house that's in the kitchen) so, for this purposes of this example, we re-parse the sentence using the online parser, with ‘Show All Linkages’ ticked, then pick the one with the least LEN:

```
(S Is there
   (NP (NP a house)
       (PP with
           (NP (NP a green cooker)
               (PP in
                   (NP the kitchen)))))
   ?)
```

S has words after, "Is there"
NP is 'a house', the thing to look for
Followed by preposition, 'with'
The preposition's target is the green cooker, and a sub-condition of that is of it being in the kitchen.

is(X, house), with(X,Green_cooker), in(Green_cooker,kitchen) and (though not strictly parseable):in(kitchen,X)

One more thing about the number of linkages of "the green cooker in the kitchen". The reason the parser misunderstood that the house was in the kitchen, is that it was hard to tell from the way the sentence was typed. So, with a comma after ‘cooker',

```
(S Is there
   (NP a house)
   (PP with
       (NP a green cooker ,
           (PP in
               (NP the kitchen)
               ?))))
```

Then we follow the usual rules.

S has words after, "Is there"
NP is "a house", the thing to look for
Followed by a preposition, with
The target is the green cooker, and that cooker is/should-be in the kitchen.

is(X,house), with(X,Green_cooker), in(Green_cooker,kitchen)

6.2.4.9 Telling a question from a statement

The statement:

> There is a cat in the closet

Can be a question or not. If we check that there is a '?' at the end, that solves everything. The question mark is generated to S->next.

```
(S (S (NP There)
      (VP is
          (NP a cat)
          (PP in
              (NP the closet))))
   ?)
```

However... this isn't as bad as it looks. Question or not, it would parse to the same thing:

> is_in(cat,closet)

We can deduce the following from this:

- If there is no "?" and it's a statement, this is a fact predicate to be put in a file of facts

- If there is a "?" and it's a question, then it is a query to Prolog to verify that that predicate exists (ie, as a condition or just a statement on the command-line).

The "?" merely tells us which Prolog mode to use when entering this predicate. The ‘?’ is for a condition, and the absence of one is for a fact.

To conclude, we will use the presence of a question mark to distinguish between facts and questions.

Rule 13: A question mark at the end, signifies a question, distinguishing it from a statement

7 Implementing the Prototype

7.1 Development

Development consisted of a few well-defined stages.

- Rewrite and debug the TinyWP editor in C++ for use in CTPE

This initial task took about 4 days, and various features have been added, and bugs fixed, since then, over perhaps 14 days of development in total.

- Integrate the editor and the parser in C

This took probably 4 days in total. It was remarkably simple. All I had to do was initialise the Link parser, pass it a sentence, tell it to parse, then grab the return pointer to the parse tree.

- Write a parse verifier to verify the parser's output is valid, and a simplifier

I decided early on that this wasn't realistic, last term when I wrote the development report. The reason it's not realistic is because there is no 'right' or 'wrong' way to predicalise a sentence. However, the program constructs predicates according to the rules in section [].

One way to verify a sentence, perhaps the best way, would be for the post-parser to mark each word that it has put into a predicate, and at the end of post-parsing, the user would be informed of the words that went unused. This seems like a nice solution.

- Write a Prolog program to use the parsed output data as predicates

I decided to test it with simple queries – an actual 'program' seems excessive.

- Test the parsed data's accuracy

This would require formal methods, and they take an order of magnitude longer than having no accuracy checking. Thus, there was no time to implement them in this project. A project such as this could be used to compile an online shopping list from someone's notepad, in which case the damage done if inaccurate is the wrong item being purchased. But for use on a space mission or nuclear submarine, formal methods would be crucial. However, the author does not have the resources of NASA or the Navy.

7.2 Testing

7.2.1 Editor

I created a test script/file which does the following:

- Backspacing from the start of a paragraph onto the last line of a multi-line paragraph
- Backspacing from the start of a paragraph onto the first line of a single-line paragraph
- Inserting a new line with the Enter key, at the middle of a single-line paragraph
- Inserting a new line with the Enter key, at the middle of a multi-line paragraph
- Inserting a piece of text onto a blank line
- Typing until the text wraps
- Backspacing until the cursor reaches the start of the line, and unwraps to the line above
- Undo'ing a string of text insertions and deletions, which are composed of the above editing tests. The tests are done, then the Undo key is held down until the end of the stack, and then Redo is held down until the end of the Redo stack, after which it is undone again. Text insertions are done halfway through the Undo stack. If the document is the same at the end of this, then the Undo and Redo work fine.

This script is in the appendix.

If it performs all these tests without crashing, then it is tested and working.

As for memory leaks, there are none as the program uses stack-allocated objects and doesn't touch the heap, though the C++ Vectors and String created on the stack may use the heap (in their constructors), but this memory is deallocated automatically with the Vector and String destructors.

7.2.2 The Parser

The parser is an interesting case for testing. As I am using the Carnegie-Mellon University's Link Parser, I am relying on a parser that has already been tested extensively. This parser operates like this: It tries to parse a sentence, and if it fails, no parse tree is generated. If no tree is generated, a NULL pointer is used. The parser never 'crashes' or fails otherwise.

Because there are only 2 states, one being that of a sentence that is parseable, and one that isn't, we only need 2 cases to deal with the parser, to avoid it crashing. One case is that the sentence to be fed into the parser is not empty (otherwise the link parser crashes), the other is not to pass a NULL pointer to the post-parser.

7.2.3 The Post-Parser

There are only a few sentence forms that can be dealt with, with the current system. There are a combination of pronoun phrases, verb phrases, and noun phrases, and it can kind of deal with adjective phrases, but they aren't recommended.

7.2.3.1 Testing for stability and robustness:

I have put in checks for NULL pointers in the parse tree wherever possible. I have put in checks from empty sentences.

The testing itself will be done with a test text-file which will crash-test the post-parser to the limit. This text-file is displayed here:

7.2.3.1.1 Text file

- Which house has a kitchen?
- Does this house have a kitchen?
- What is on the first floor, at templeviewterrace?
- What rooms are in templeviewterrace?
- Rocket1 contains cargo1 and cargo3 and cargo4 and is blue and explosive
- Jill isn't annoyed
- Jack is not hurt and is annoyed
- Rocket2 is not in London or in Texas
- Jack fell down and broke his crown

Not all of these phrases will be convertable into predicates.

7.2.3.1.2 First, here are the parse trees:

```
+PRED:Which house has a kitchen?
+GENPRED:      (S Which house
+GENPRED:         (VP has
+GENPRED:            (NP a kitchen))
+GENPRED:         ?)

+PRED:Does this house have a kitchen?
+GENPRED:      (S Does
+GENPRED:         (NP this house)
+GENPRED:         (VP have
```

```
+GENPRED:            (NP a kitchen))
+GENPRED:        ?)

+PRED:What is on the first floor, at templeviewterrace?
+GENPRED:      (S What
+GENPRED:        (S (VP is
+GENPRED:              (PP on
+GENPRED:                 (NP the first floor ,
+GENPRED:                    (PP at
+GENPRED:                       (NP templeviewterrace)
+GENPRED:                       ?))))))

+PRED:What rooms are in templeviewterrace?
+GENPRED:      (S What rooms
+GENPRED:        (VP are
+GENPRED:           (PP in
+GENPRED:              (NP templeviewterrace)))
+GENPRED:        ?)

+PRED:Rocket1 contains cargo1 and cargo3 and cargo4 and is blue and explosive
+GENPRED:      (S (NP Rocket1)
+GENPRED:        (VP (VP contains
+GENPRED:              (NP (NP cargo1)
+GENPRED:                 and
+GENPRED:                 (NP cargo3)
+GENPRED:                 and
+GENPRED:                 (NP cargo4)))
+GENPRED:           and
+GENPRED:           (VP is
+GENPRED:              (ADJP (ADJP blue)
+GENPRED:                    and
+GENPRED:                    (ADJP explosive)))))

+PRED:Jill isn't annoyed
+GENPRED:      (S (NP Jill)
+GENPRED:        (VP isn't
+GENPRED:           (ADJP annoyed)))

+PRED:Jack is not hurt and is annoyed
+GENPRED:      (S (NP Jack)
+GENPRED:        (VP (VP is not
+GENPRED:              (VP hurt))
+GENPRED:           and
+GENPRED:           (VP is
+GENPRED:              (ADJP annoyed))))

+PRED:Rocket2 is not in London or in Texas
+GENPRED:      (S (NP Rocket2)
+GENPRED:        (VP is not
+GENPRED:           (PP (PP in
```

```
+GENPRED:                (NP London))
+GENPRED:             or
+GENPRED:             (PP in
+GENPRED:                (NP Texas)))))

+PRED:Jack fell down and broke his crown
+GENPRED:     (S (NP Jack)
+GENPRED:       (VP (VP fell
+GENPRED:             (PRT down))
+GENPRED:          and
+GENPRED:          (VP broke
+GENPRED:             (NP his crown))))
```

7.2.3.1.3 Now for the predicates

```
+PRED:Which house has a kitchen?
+GENPRED:     ***Is a question:
+GENPRED:     is(X,house)
+GENPRED:     has(x,a_kitchen).

+PRED:Does this house have a kitchen?
+GENPRED:     ***Is a question:
+GENPRED:     is(X,this_house)
+GENPRED:     have(x,a_kitchen).

+PRED:What is on the first floor, at templeviewterrace?
+GENPRED:     ***Is a question:
+GENPRED:     is(X,)

+PRED:What rooms are in templeviewterrace?
+GENPRED:     ***Is a question:
+GENPRED:     is(X,rooms)
+GENPRED:     in(X,templeviewterrace)
+GENPRED:     are(x,templeviewterrace).

+PRED:Rocket1 contains cargo1 and cargo3 and cargo4 and is blue and explosive
+GENPRED:     contains(rocket1,cargo1).
+GENPRED:     contains(rocket1,cargo3).
+GENPRED:     contains(rocket1,cargo4).
+GENPRED:     is(rocket1,blue).
+GENPRED:     is(rocket1,explosive).

+PRED:Jill isn't annoyed
+GENPRED:     isn't(jill,annoyed).

+PRED:Jack is not hurt and is annoyed
+GENPRED:     is_not(jack,hurt).
+GENPRED:     is(jack,annoyed).
```

+PRED:Rocket2 is not in London or in Texas
+GENPRED: in(Rocket2,London)
+GENPRED: in(Rocket2,Texas)
+GENPRED: is_not(rocket2,).

+PRED:Jack fell down and broke his crown
+GENPRED: fell(jack,down).
+GENPRED: broke(jack,his_crown).

+PRED:This house has 3 bedrooms, is spacious, cottage-style and semi-detached.
[Nothing generated]

+PRED:It is in the city of Leeds.
+GENPRED: in(It,the_city)
+GENPRED: in(It,of)
+GENPRED: is(it,).

+PRED:Leeds is famed for the castle and magnificent grounds.
+GENPRED: is(leeds,famed)

7.3 Debugging

The debug sessions were not documented. The programmer used the single-step and Variable-Watch functionality of the development environment, and where necessary, print statements were inserted to dump information to the console, such as which lines of code were executed, and the state of variables. All print statements were removed immediately after the bug was fixed.

8 Evaluation of Project

8.1 Evaluation of Prototype

8.1.1 Good points

8.1.1.1 The Editor

I consider the editor to be quite a success. Although it is definitely overkill for the task in hand, which is to let people enter a few sentences at a time and immediately see the output as predicates or parse trees, it fulfills all the requirements of the desired kind of editor.

I added Save/Load file requesters. The File Load requester is launched on the loading of the program, while Save is triggered with F2.

8.1.1.2 The Parser

As this is just a small application with just a few rules, it is perhaps amazing that it works at all, even if it's just on the most basic sentences and questions.

The problem with some Artificial Intelligence applications has been that simple examples are simple, while human-level complexity is an order of magnitude more difficult. CTPE could be seen as the former, but allowing the human user to simplify the text as they write it, hopefully gives CTPE the ability to bridge the gap between simplicity and human-level complexity.

8.1.2 Bad Points

8.1.2.1 The Parser

One bad point is that the parser seems too slow even on modern machines! The impact of this could be lessened by only parsing the currently-edited sentence and storing the parsed results of the others in memory. This is not done in the present version.

8.1.2.2 The Editor

The editor is slightly unstable, and Undo/Redo has not been finalised. There is no Search/Replace or even Find, and the window cannot be resized due to problems with re-positioning the cursor.

8.1.2.3 Prolog

I found out rather late that if a series of predicates with the same title are generated, Prolog tends to form links between them. Thus, when a search query is entered, it starts finding irrelevant results. This is unfortunate when I'd expected to be able to search for things like "is(house,X)" and "is(cooker,Y)", on these predicates:

is(house,green)
is(cooker,green)

I don't know what the solution to this is.

8.2 Analysis of Project

8.2.1 Usefulness of the parsed output and Prolog queries

So how useful is the program for the target clientele, in particular, the customer of an Estate Agents', who is looking for an ideal home?

Part of the original design was that if the program could handle converting text into predicates, then it wouldn't be too hard to do the same with queries. However, this was slightly ambitious and there wasn't time to fully implement this. It *does* handle queries, just not the kind of queries the average client would come up with.

Some sample output from the predicaliser:

+PRED:The view from this house is tranquil.
+GENPRED: is(the_view,tranquil).

+PRED:This house is near to Streatham Common and close to the High Road.
+GENPRED: is(this_house,near).
+GENPRED: is(this_house,close).

+PRED:This flat is spacious and affordable, has 2 bedrooms, is on the ground floor, and should be looked at.
+GENPRED: is(this_flat,spacious).
+GENPRED: is(this_flat,affordable).
+GENPRED: has(this_flat,2_bedrooms).
+GENPRED: on(This_flat,the_ground_floor)
+GENPRED: is(this_flat,the_ground_floor).
+GENPRED: should(this_flat,be).

+PRED:The property has a 15' lounge come diner, a modern bathroom suite, communal grounds and off-street parking .

+PRED:The house has 2 bedrooms and a lounge.
+GENPRED: has(the_house,2_bedrooms).
+GENPRED: has(the_house,a_lounge).

+PRED:The house is situated in a nice area.
+GENPRED: is(the_house,situated).

Here are a couple of queries which, if entered into Prolog, with a house-description Textbase loaded, will find the houses exactly as the user has requested:

Query: Which house has a kitchen?
Prolog form: is(X,house),has(X,a_kitchen).

Query: Does this house have a cellar?
Prolog form:is(X,this_house),have(X,a_cellar).

Query: Which house has a kitchen and a driveway?
Prolog form:is(X,this_house),has(X,a_kitchen), has(X,a_driveway)

Here are some queries, entered into Prolog in predicate form, which find some houses by their common features:

The actual predicates are in appendix 13.3.

Now come the queries, based on this data:

%Which house has a kitchen?
| ?- has(X,a_kitchen).

X = a241dewsburyroad ? ;
X = templeviewterrace ? ;
X = glensdaleterrace ? ;
X = branderdrive ? ;
X = austhorperoad ? ;

%What is on the first floor at Temple View Terrace?
| ?- on(at(X,templeviewterrace),the_first_floor).
X = a_bedroom ? ;
X = a_bathroom ? ;

%In what houses, and on what floor, is there a bathroom?
| ?- on(at(a_bathroom,X),Y).

X = templeviewterrace,
Y = the_first_floor ? ;

X = branderdrive,
Y = the_first_floor ? ;

X = austhorperoad,

```
Y = the_first_floor ? ;

%What houses are on any floor at Temple View Terrace?
| ?- on(at(X,templeviewterrace),Y).

X = a_bedroom,
Y = the_first_floor ? ;

X = a_bathroom,
Y = the_first_floor ? ;

%Which house has a cellar?
| ?- has(X,a_cellar).

X = glensdaleterrace ? ;
X = austhorperoad ? ;
```

I think this shows quite clearly the usefulness of the system.

9 Conclusion

It has been the ambition of the author to implement this system for maybe 7 years. It started with statistical analysis of the frequency of English words in a series of webpages and similar corpus, using Wordnet to tag the words. This project went nowhere, but the spark of inspiration remained.

It was very satisfying to get the help, inspiration and resources to bring this ambition nearer. The help came from the Project Supervisor, the inspiration from the Supervisor and the author, and the resources of the Internet and the University Library were combined with the luck of finding a decent parser.

It is also satisfying when the original theory is thought up, which in this case was turning text into a computer-readable form, and seeing it slowly come to fruition.

The project is rather rudimentary at present, barely able to answer questions and parse some basic sentences, but with more work it could perhaps become useful to the world at large, and perhaps be marketable, and thus fulfil the ambition of the author.

10 Bibliography

Chomsky, N. (1972) *Language and Mind.* New York, Harcourt Brace

Sowa, John F. (2000) *Knowledge Representation: Logical, Philosophical and Computational Foundations*. Pacific Grove CA, Brooks/Cole

Ginsburg, M. (1991) *Knowledge Interchange Format: The KIF of Death.* AI Magazine

Temperley et al (2004) *Link Grammar*: http://www.link.cs.cmu.edu/link/index.html

Metafor (2005)
http://web.media.mit.edu/~hugo/publications/papers/IUI2005-metafor.pdf

Attempto Controlled English (ACE) (2004)
http://www.ifi.unizh.ch/attempto/

Interactive Online CL Demos (2004)
http://www.ifi.unizh.ch/CL/InteractiveCLtools/index.php

ENGCG. English Constraint Grammar Parser
(1995)
http://www.lingsoft.fi/cgi-bin/engcg

EngLite Parser. Functional Dependency Grammar Parser
(2004)
http://www.connexor.com/demos.html

Freshmeat, a repository of software http://www.freshmeat.net/

11 APPENDICES

11.1 User Manual for the CTPE editor

11.1.1 Introduction

The CTPE editor has been written for users such as yourself, who wish to enter computer-readable data in a textual format that people can also understand.

It consists of an editor which is linked to a parser, and then the output from the parser is fed into a post-parser.

A sentence like this:

> The house has 2 bedrooms and a lounge.

Would look like this after parsing in CTPE:

> has(the_house,2_bedrooms).
> has(the_house,a_lounge).

11.1.2 Using the Editor

The editor works like any other modern graphical text editor.

When you start up, the program will ask you to select a file. For now, this file must be in the same folder as the program. This is because if you select another folder, it is unable to find the font file. Select a file from the list given, without entering a folder, and click ‘Open’.

If you hit ‘Cancel’, the program will start with a blank document.

Once loaded, you move around the document with the cursor keys and can use backspace to delete text before the cursor (the Delete key does not work as yet). You can also type in text to be inserted at the cursor position.

Any line starting “+PRED:” will be parsed, and the output displayed on the next few lines. Do not start a sentence with the phrase “+GENPRED:” as it will be erased.

To toggle the parser output between the “parse tree” and the “predicate” output, press F1.

To save the file, press F2.

To copy some text to the clipboard, press F4 to set a start point, then move the cursor to the end of the text (using the cursor keys) and when done, hit Ctrl+C.

To paste text into the editor, from the Windows clipboard, hit Ctrl+V.

11.1.3 The Title Bar

This shows you:

- The line of the document the cursor is on. Unfortunately, this is not the paragraph the cursor is on, but the editor line.
- The total number of editor lines in the document
- The Column (from far-left column 1) the cursor is on
- The number of words in the document before the cursor position
- The total number of words in the document
- The total number of bytes in the file

11.1.4 Exiting the Editor

To exit, hit Escape or use the 'X' in the top-right corner of the window. Please be sure to save the file first before exiting, as the program will not ask you to save, before exiting.

11.2 Sample Prolog Queries

```
SICStus 3.8.5 (x86-win32-nt-4): Wed Oct 25 16:04:12  2000
Licensed to hud.ac.uk
| ?- ['c:/linuxsafe/uniwork/testpreds_.txt'].
{consulting c:/linuxsafe/uniwork/testpreds_.txt...}
{Warning: in lines 0-1: (discontiguous)/1 - not redefined}
{Warning: in lines 1-9: is/2 - not redefined}
{Warning: clauses for user:has/2 are not together}
{Warning: clauses for user:on/2 are not together}
{consulted c:/linuxsafe/uniwork/testpreds_.txt in module user, 20 msec 6432 bytes}

yes
| ?- has(X,a_kitchen).

X = a241dewsburyroad ? ;

X = templeviewterrace ? ;

X = glensdaleterrace ? ;

X = branderdrive ? ;

X = austhorperoad ? ;

no
| ?- on(at(X,templeviewterrace),the_first_floor).

X = a_bedroom ? ;

X = a_bathroom ? ;
```

```
no
| ?- ;
    on(at(a_bathroom,X),Y).
{EXISTENCE ERROR: ?-: procedure user:(?-)/0 does not exist}
| ?- on(at(a_bathroom,X),Y).

X = templeviewterrace,
Y = the_first_floor ? ;

X = branderdrive,
Y = the_first_floor ? ;

X = austhorperoad,
Y = the_first_floor ? ;

no
| ?- on(at(X,templeviewterrace),Y).

X = a_bedroom,
Y = the_first_floor ? ;

X = a_bathroom,
Y = the_first_floor ? ;

no
| ?- has(X,a_cellar).

X = glensdaleterrace ? ;

X = austhorperoad ? ;

no
| ?-
```

11.3 Sample Generated Predicates

```
%['c:/linuxsafe/uniwork/testpreds.txt'].

%Problem: How do we say "This bedroom is on the first floor" and specify the
property the bedroom is in?

%is_on(at(a_further_bedroom,241dewsburyroad),second_floor)

%241, Dewsbury Road, Leeds, West Yorkshire, LS11 5HZ

%contiguous(has/2).

myhas(A,B):-
        has(A,B),
        \+ has(B,A).
```

```
has(_241dewsburyroad,been).
has(_241dewsburyroad,an_open-plan_lounge).
has(_241dewsburyroad,a_sleeping_area).
has(_241dewsburyroad,a_shower_room).
has(_241dewsburyroad,a_study_area).
has(_241dewsburyroad,a_kitchen).

has(templeviewterrace,a_bedroom).
has(templeviewterrace,a_bathroom).
has(templeviewterrace,a_wc).
has(templeviewterrace,_2_bedrooms).
has(templeviewterrace,a_lounge).
has(templeviewterrace,a_kitchen).
has(templeviewterrace,a_basement_cellar).

has(glensdaleterrace,a4_bedrooms).
has(glensdaleterrace,double_glazing).
has(glensdaleterrace,a_lounge).
has(glensdaleterrace,a_kitchen).
has(glensdaleterrace,a4_bedrooms).
has(glensdaleterrace,a_bathroom).
has(glensdaleterrace,a_cellar).

has(branderdrive,an_entrance_lobby).
has(branderdrive,a_lounge).
has(branderdrive,a_kitchen).
has(branderdrive,_3_bedrooms).
has(branderdrive,a_bathroom).
has(branderdrive,a_wc).

has(austhorperoad,_3_bedrooms).
has(austhorperoad,a_lounge).
has(austhorperoad,a_kitchen).
has(austhorperoad,a_cellar).
has(austhorperoad,a_master_bedroom).
has(austhorperoad,a_bathroom).
has(austhorperoad,_2_bedrooms).

on(at(a_bedroom,templeviewterrace),the_first_floor).
on(at(a_bathroom,templeviewterrace),the_first_floor).
on(at(a_wcmtempleviewterrace),the_first_floor).
on(at(_3_bedrooms,branderdrive),the_first_floor).
on(at(a_bathroom,branderdrive),the_first_floor).
on(at(a_wc,branderdrive),the_first_floor).
on(at(a_lounge,austhorperoad),the_ground_floor).
on(at(a_kitchen,austhorperoad),the_ground_floor).
on(at(a_cellar,austhorperoad),the_ground_floor).
on(at(a_master_bedroom,austhorperoad),the_first_floor).
on(at(a_bathroom,austhorperoad),the_first_floor).
```

on(at(_2_bedrooms,austhorperoad),the_first_floor).
on(at(a_further_bedroom,templeviewterrace),the_second_floor).

myis(_241dewsburyroad,open-plan).
myis(templeviewterrace,back-to-back).
myis(glensdaleterrace,back-to-back).
myis(it,ideal).
myis(branderdrive,semi-detached).
myis(austhorperoad,back-to-back).

would(it,be).
would(it,be).

%Temple View Terrace, Leeds, West Yorkshire LS9 1AA [map]

%Glensdale Terrace, Leeds, West Yorkshire LS9 9DB [map]

near(glensdaleterrace,local_amenities).
near(glensdaleterrace,shops).
near(glensdaleterrace,schools).
recommend(we,early_viewing).

%Brander Drive, Leeds, West Yorkshire LS9 1AA [map]

needs(branderdrive,modernisation).

%22, Austhorpe Road, Leeds, West Yorkshire, LS15 8DX

11.4 Keyboard Macro for Testing the Editor

This macro would operate on a test file, which follows.

Test.txt
[

Blank line above and below this sentence

Sentence that runs on and on across multiple screen lines.

Blank line above

Sentence that fits on one line.
]

As the macro facility is not implemented, it is not possible to record the Test actions that would operate on this file. Therefore, only the first action has been noted, so it can be seen how the macro testing functionality would work. The cursor starts at the start of Test.txt before the test macro is run.

- Backspacing from the start of a paragraph onto the last line of a multi-line paragraph

  ^CURSORDOWN>
  ^CURSORDOWN>
  ^CURSORDOWN>
  ^CURSORDOWN>
  ^CURSORDOWN>
  ^CURSORDOWN>
  ^CURSORDOWN>
  ^CURSORDOWN>
  ^BSP>

BSP is ‘backspace’. The remaining test actions are left to be done in the future.

- Backspacing from the start of a paragraph onto the first line of a single-line paragraph

- Inserting a new line with the Enter key, at the middle of a single-line paragraph

- Inserting a new line with the Enter key, at the middle of a multi-line paragraph

- Inserting a piece of text onto a blank line

- Typing until the text wraps

- Backspacing until the cursor reaches the start of the line, and unwraps to the line above

- Undo'ing a string of text insertions and deletions, which are composed of the above editing tests. The tests are done, then the Undo key is held down until the end of the stack, and then Redo is held down until the end of the Redo stack, after which it is undone again. Text insertions are done halfway through the Undo stack. If the document is the same at the end of this, then the Undo and Redo work fine.

11.5 Rules comprising the CTPE System

- Rule 1: Ignore any words like 'to' when constructing predicate titles.
- Rule 2: Each PP or VP is a predicate. An ADJP is a predicate which can contain other predicates.
- Rule 3: There is always a subject "NP" after the first "S". This is the first part of the predicate.
- Rule 4: An S followed by followed by 'NP' is always followed by a Verb Phrase in all the cases in the original examples. Let's not worry about ADVP or PP for now. Take note of the verb, this is the name of the predicate.
- Rule 5: A Verb Phrase has a completing statement. If the argument to a verb phrase is an NP, this is the second part of the predicate. If not, ignore the first NP and use the second.
- Rule 6: If there are multiple NP's within an NP, create a separate predicate for each one contained.
- Rule 7: If the predicate-generator finds a sentence which fits no rules, and thus nothing that matches the rules can be generated, then the user will receive no results and must rewrite it until they do. As more rules are added, that task will need to be done less frequently.
- Rule 8: If an NP is followed by one VP, which consists solely of children which are also VP's, then pass the first part of the predicate, and the pointer to the VP node, to a function which parses the verb phrases and returns a list of textual predicates. For each VP within the root VP, it gets the title (the first part of the VP), and the second part of the predicate (the phrase that comes after the first part of the VP).
- Rule 9: Any 'or' or 'and' must be prefixed with 'a', 'an', 'the', etc to help the Link Parser split up the sentence properly into multiple noun-phrases. EG:
- `"(NP an investor or [a] first-time buyer))))"`
- Rule 10: If a noun-phrase is followed by a preposition (PP) and prefixed by a verb phrase (VP), create a sub-predicate with the title of the second part of the outer predicate being a predicate with the title of the preposition.
- Rule 11: A sentence node S followed by NP/ADJP is a question, where the generated predicate is: is(the_house,blue)
- Rule 12: Always choose the linkage with the lowest distance, the distance being the "LEN" property of the linkage, in the Link Parser.

- Rule 13: A question mark at the end, signifies a question, distinguishing it from a statement

11.6 Sequence Diagram for CTPE

11.7 Source code for the Link module in the CTPE