Learning Natural Language Interfaces over Expressive Meaning Representation Languages

Johan Granberg **<johang@cs.umu.se>**,

30th September 2010
Master's Thesis in Computing Science, 30 credits
Supervisor at CS-UmU: Michael Minock
Examiner: Fredrik Georgsson

Umeå University
Department of Computing Science
SE-901 87 Umeå
Sweden

Abstract

This thesis focuses on learning natural language interfaces using synchronous grammars, λ-calculus and statistical modeling of parse probabilities. A major focus of the thesis has been to replicate Mooney and Wong's λ-WASP [17] algorithm and implement it inside the C-PHRASE [12] Natural Language Interface (NLI) system. By doing this we can use C-PHRASE's more expressive and transportable *meaning representation language* (MRL), rather than the PROLOG-based MRL Mooney and Wong used.

Our system, the C-PHRASE LEARNER, relaxes some constraints in λ-WASP to allow use of more flexible MRL grammars. We also reformulate the algorithm in terms of operations on trees to clarify and simplify the approach. We test the C-PHRASE LEARNER over the US geography corpus GEOQUERY and produce precision and recall results slightly below those achieved by λ-WASP. This was expected as we have fewer domain restrictions due to our more expressive and portable MRL grammar.

Our work on the C-PHRASE LEARNER system has also revealed some promising avenues of future research including, among others, alternative statistical alignment strategies, integrating linguistic theories into our learning algorithm and ways to improve named entity recognition. C-PHRASE LEARNER is presented as open source to the community to allow anyone to expand upon this work.

Contents

Chapter 1

Introduction

A long standing usability problem with computers is the need for humans to express complex commands and queries in some kind of formal input language. This 'language' can be visual (iconic) as in point-and-click graphical user interfaces or textual as in command line interfaces. In many cases it would be simpler if humans could communicate with computers using their native tongue, such as English or Swedish. This is what automated natural language understanding promises. Parsing natural languages into logical formulas with clear semantic meaning seems to be a necessary condition for communicating with computer systems using natural languages. Hopefully this will make computers easier to use, especially for people with less technical training.

This work focuses on the subproblem of restricted domain natural language interfaces to databases. The actual example we have worked with is a database over US geography, although our work and implementation should generalize to learning interfaces for any system that understands database query syntax. For example a robot, with memory encoded as a database, that could be commanded by changing values in an objectives table can easily be envisioned.

Previous experience in the 1970's and 1980's [3] has shown that building Natural Language Interfaces (NLIs) to databases by hand is a hard, time consuming task and the resulting systems are brittle. This has motivated many researchers look at learning NLIs from training corpora [6, 10, 12, 18, 20, 21]. A corpus for learning NLIs to databases typically is a list of sentences in a natural language paired with a formal meaning representation language (MRL) expression. Figure 1.1 shows three examples from our translation of GEOQUERY[1], a set of united states geography questions asked by students at the University of Texas [20].

The first part of each line is, as can be seen, a natural language string. This is followed by a # symbol that acts as divider. The rest of the line is an MRL

[1] `http://code.google.com/p/c-phrase/source/browse/trunk/demo/geo/corpora/mrl-corpus.crp`

```
how big is new mexico # ((X AREA) STATE (= X NAME "New Mexico"))
what state has the smallest population # (X STATE (:BOTTOM 1 POPULATION X))
what state has the smallest area # (X STATE (:BOTTOM 1 AREA X))
```

Figure 1.1: Example corpus

expression in the C-PHRASE MRL syntax. See Appendix C for a BNF of C-PHRASE query syntax. See Appendix E for more examples from the GEOQUERY corpus.

The amount of effort spent in building a good training corpus cumulatively adds to the resulting quality of a learned system, as a good learning algorithm will improve slightly with each additional correct training example. An attempt to render the difference is seen in Figure 1.2. While the rate of improvement can not be compared there is more of a choppy improvement in the hand-built case. This choppiness can be seen as the developer having to find new domain-dependent tricks to boost system quality by some amount at a time. When using learning on the other hand the quality increases steadily and the effort that has to be put in is to create good training data. That is a task that a much less specialised administrator can be responsible for.

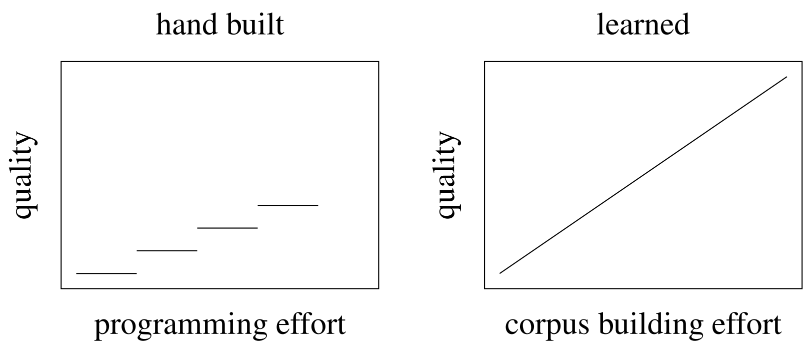


Figure 1.2: Quality effort characteristics of building NLIs and learning them

To port learning algorithms to different domains by only changing the corpus, a complex and general purpose meaning representation language must be used. Simpler meaning representation languages using variable-free functional style [10] appears too weak for this.

This thesis explores learning of complex MRLs that are domain-independent. The domain is encoded only in the corpus and the associated database schema.

1.1 Thesis Contributions

This thesis contributes a domain-independent way to learn semantic grammars from corpora using only the database schema and an NL/MRL training set as inputs. It also provides an implementation of this as a LISP package in the freely available C-PHRASE system [14]. Additionally this system distributes jobs to SSH-accessible hosts to parallelize learning. Finally this thesis clarifies some restrictions in earlier work, mainly by Mooney and Wong that was not obvious from their papers.

1.2 Organization of this Thesis

Chapter 2 covers the background of the field, including techniques and approaches such as grammar types, ways to construct natural language interfaces, and how different groups have approached learning them. Chapter 3 continues by describing our way to tackle the problem, describing in detail the algorithm we are using. This involves synchronous context free grammars, statistical modeling and computing statistical word alignments. Although our work is very closely related to λ-WASP, we introduce several new angles and refinements to the basic algorithm. Results over the GEOQUERY corpus are presented in Chapter 4 and we also explain our metrics. In Chapter 5 the work and possible continuations of it are discussed. Chapter 6 concludes.

Chapter 2

Background

This chapter provides background on, Context free and Synchronous Context free grammars, with and without lambda calculus extensions as well as natural language interface to databases and the learning of semantic grammars.

2.1 Grammar Formalisms

This thesis uses some well known results on grammars from formal linguistics and computer science. Mainly context free grammars and extensions thereof.

2.1.1 Context Free Grammars

Discovered by Noam Chomsky in the 1950's, context free grammars are a formal system used to generate and parse strings in context free languages. Context-free grammars are interesting because they can generate and efficiently parse formal languages and subsets of natural ones.

Definition 1

- Let the Context Free Grammar G be a tuple (V, Σ, R, S) where,
- V is a set of nonterminal symbols
- Σ is a set of terminal symbols
- R is a finite set of rewrite rules, taking a nonterminal symbol and replacing it with a string of any number of symbols from $V \cup \Sigma$
- S is the start symbol

To generate a string using a context free grammar write down the start symbol. As long as there are nonterminals in the string, select a rule that can rewrite some nonterminal in the string, replace the nonterminal with the right hand side of the rule.

Note that usually there are multiple choices of rules in each step. The choices affect which string is generated.

- Nonterminals are $S \quad NP \quad N \quad VP \quad V$
- Terminal symbols are 'Philip', 'Mary', 'candy', 'lawyers', 'likes' and 'hates'.
- Startsymbol is S
- Rules are
  $S \to NP \cdot VP$
  $NP \to N$
  $VP \to V \cdot NP$
  $N \to Philip$
  $N \to Mary$
  $N \to candy$
  $N \to lawyers$
  $V \to likes$
  $V \to hates$

Figure 2.1: A small context free grammar

Consider the simple example grammar in Figure 2.1. This grammar can generate "Philip likes candy" by selecting first the first rule, here we have no alternatives, then the second rule, the third rule, the second rule again followed by the fourth, sixth and eight rule. The steps are shown below.

$$S \Rightarrow NP \cdot VP \Rightarrow N \cdot VP \Rightarrow N \cdot V \cdot NP$$
$$\Rightarrow N \cdot V \cdot N \Rightarrow Philip \cdot V \cdot N \Rightarrow Philip \cdot likes \cdot N \Rightarrow Philip \cdot likes \cdot candy$$

where $\Rightarrow$ stands for a derivation step. The notation $\overset{*}{\Rightarrow}$ stands for an arbitrary number of derivation steps so the above can be written as

$$S \overset{*}{\Rightarrow} Philip \cdot likes \cdot candy$$

This could equally well have generated "Mary hates lawyers" or choosing rules involving those words. Note that in step six we only replaced one N even when we had several nonterminals N.

Parsing a string in a context free language involves finding out which rules were used to generate it. Additionally which rule expanded which inner node needs to be found. This results in a tree where the interior nodes are the expanded nonterminals. The root is the start symbol. In general there can be multiple potential expansions that generate the same string, resulting in ambiguity. If that is not the case for any string generated by the grammar the language is *unambiguous*.

2.1.2 Synchronous Context Free Grammars

A synchronous context free grammar is a grammar that works over pairs of languages, in this case a natural language (English) and a meaning representation language.

Definition 2

- Let an SCFG be a tuple $(\Sigma_{NL}, \Sigma_{MR}, N, S, P)$ where,
- Σ_{NL} is a set of words or terminals in the natural language,
- Σ_{MR} is a set of terminals in the meaning representation language,
- N is a set of nonterminal symbols,
- $S \in N$ is the start symbol and
- P is a set of productions, $P \rightarrow \langle \alpha, \beta \rangle$, following these constraints,
  - The left hand side is a nonterminal in N,
  - The right hand side is a pair of strings, $\langle \alpha, \beta \rangle$, the first string, α, is over $\Sigma_{NL} \cup N$, $\alpha \in (\Sigma_{NL} \cup N)^*$ and the second string, β, is over $\Sigma_{MR} \cup N$, $\beta \in (\Sigma_{MR} \cup N)^*$.
  - If the production produces a nonterminal in one of the strings it must produce a corresponding[a] identical nonterminal in the other one.

To generate a string pair with an SCFG start with a tuple (S, S), while there are nonterminals left in the strings, select a rule matching a nonterminal, take the first part of the rule's right hand side and replace the nonterminal in the first string with that, then take the second part of the rules right hand side and replace the corresponding nonterminal with that.

[a]A good way to keep track of the correspondence is to associate each new pair of nonterminals with a monotonically increasing index, increased every time a rule produces a nonterminal

If we take the grammar from the ordinary context free grammar example and expand it to become a SCFG we the example in figure 2.2.

This grammar can yield the following derivation.

$$S_1\ ,\ S_1 \Rightarrow NP_2 \cdot VP_3\ ,\ VP_3;NP_2) \Rightarrow$$
$$\Rightarrow NP_2 \cdot V_4 \cdot NP_5\ ,\ V_4(NP_5;NP_2) \Rightarrow N_6 \cdot V_4 \cdot NP_5\ ,\ V_4(NP_5;N_6) \Rightarrow$$
$$\Rightarrow N_6 \cdot V_4 \cdot N_7\ ,\ V_4(N_7;N_6) \Rightarrow N_6 \cdot hates \cdot N_7\ ,\ \mathsf{hates}(N_7;N_6) \Rightarrow$$
$$\Rightarrow Mary \cdot hates \cdot N_7\ ,\ \mathsf{hates}(N_7;\mathsf{Mary}) \Rightarrow Mary \cdot hates \cdot lawyers\ ,\ \mathsf{hates}(\mathsf{lawyers};\mathsf{Mary})$$

or more concisely

$$S_1, S_1 \overset{*}{\Rightarrow} Mary \cdot hates \cdot lawyers\ ,\ \mathsf{hates}(\mathsf{lawyers};\mathsf{Mary})$$

- $N = \{S, NP, N, VP, V\}$
- $\Sigma_{NL} = \{$*'Philip', 'Mary', 'candy', 'lawyers', 'likes', 'hates'*$\}$
- $\Sigma_{MR} = \Sigma_{NL} \cup \{ \; (\; \cdot \;) \; \cdot \; ; \; \}$ [a]
- Startsymbol is S
- Rules are

$S \rightarrow \langle NP_1 \cdot VP_2 \; , \; VP_2;NP_1)\rangle$
$NP \rightarrow \langle N \; , \; N\rangle$
$VP \rightarrow \langle V_1 \cdot NP_2 \; , \; V_1(NP_2\rangle$
$N \rightarrow \langle Philip$, Philip$\rangle$
$N \rightarrow \langle Mary$, Mary$\rangle$
$N \rightarrow \langle candy$, candy$\rangle$
$N \rightarrow \langle lawyers$, lawyers$\rangle$
$V \rightarrow \langle likes$, likes$\rangle$
$V \rightarrow \langle hates$, hates$\rangle$

[a]That there is an overlap between symbols in the NL and MR is just a coincidence

Figure 2.2: Example SCFG

Depending on the definition of the MRL language the second part of the above might be understandable for the system as stating the fact that lawyers are hated by Mary.

Note the use of indices to keep track of nonterminal correspondence. Also note how the words Mary and lawyers are in a slightly unnatural order. This is due to the fact that the NL grammar is fixed and the SCFG does not have enough expressive power to reorder them. The last problem is fixed in λ-SCFG grammars.

SCFG parsing starts by obtaining parse trees for the NL by using the non-terminals and NL parts of the rules as an NL grammar. These parse trees keeps track of which SCFG productions where used to parse these trees. The MRL part of the rules that was used to parse these trees are then applied to the start symbol generating a second set of trees. Combining the trees in these sets we produce the same trees as applying the SCFG productions to the tuple (S, S) would have generated. This shows that a SCFG can translate both ways, something that can be useful. Note that when the grammar is non-ambiguous only a single tree pair will be generated.

2.1.3 λ-SCFGs

λ-SCFGs [18] are modified SCFGs where the MRL part of the productions are λ-expressions (as in λ-calculus). When generating the MRL translation from the NL tree we therefore have to evaluate λ-expressions.

Aside λ-calculus is a computational model able to express any computation. It is equivalent in computational power to turing-machines. The basis of λ-calculus is functions and variables, used in the same way as in most math. Functions takes parameters and replace internal variables with the values of the parameters. Parameters can be either functions or variables. The fact that functions can be parameters allows for copying sub-expressions. This is necessary to express infinite computations. More practically, function copying is used to apply the same function over all elements in lists and other repetitions[a].

[a] A creative rendering of λ-calculus is AlligatorEggs [16], a game that looks aimed at school children and can be found at `http://worrydream.com/AlligatorEggs/`. Even those understanding λ-calculus might want to take a look.

Definition 3 Formally λ-SCFG is obtained from SCFG extended so that the second half of the right hand side, β is replaced with $\lambda x_1 \ldots \lambda x_n \beta$, where the λ's indicate variable binding as per normal λ-calculus.

- The string β can, in addition to nonterminals and terminals, contain bound variables.
- All nonterminals in β must be followed by zero or more parameters. The nonterminal can only be expanded by a rule whose MRL part takes that many formal parameters.
- Nonterminals in the MRL part are represented as lambda expressions and the subtrees are given as parameters.

This extension is powerful as λ-calculus is Turing-complete arbitrary computations can be done. This work uses it to bind MRL variables to a "declaration".

In the example in Figure 2.3 on the following page we only have one option in each step but in a real scenario several options are often present. These options are then expanded in parallel and the ones that can not be continued are discarded yielding one or several resulting parses.

Example usage using rules extracted by our system

- Nonterminals and terminals are implicit
- Start symbol is $QUERY$
- Rules are

  $QUERY \rightarrow \langle gap^3 \cdot cities \cdot COND_1 \ , \ \lambda gap^3.\lambda COND_1.(X\ city\ COND_1\ X)\rangle$

  $COND_* \rightarrow \langle in \cdot COND_2 \ , \ \lambda COND_2.\lambda X.(:\ exists\ (Y)\ (state\ Y)\ (=\ X\ state\ Y\ name)\ COND_2\ Y)\rangle$

  $COND_* \rightarrow \langle arizona \ , \ \lambda X.(=\ X\ name\ ''arizona'')\rangle$

  $gap^3 \rightarrow \langle give \cdot me \cdot the \ , \ \rangle$

To parse “give me the cities in arizona” the following steps may be taken

1. the string “give me the” matches the last rule and produce the MRL string ‘nothing’
2. the string ‘cities’ matches the first rule producing the MRL string
   $\lambda gap : 3.\lambda COND_1.(X\ city\ COND_1\ X)$
3. the MRL ‘nothing’ from 1 will be passed to the MRL in 2 resulting in
   $\lambda COND_1.(X\ city\ COND_1\ X)$
4. the string ‘in’ matches the second rule resulting in the MRL string
   $\lambda COND_2.\lambda X.(:\ exists\ (Y)\ (state\ Y)\ (=\ X\ state\ Y\ name)\ COND_2\ Y)$
5. the string ‘arizona’ matches the MRL string
   $\lambda X.(=\ X\ name\ ''arizona'')$
6. the MRL from 5 is passed to the result from 4 resulting in
   $\lambda X.(:\ exists\ (Y)\ (state\ Y)\ (=\ X\ state\ Y\ name)\ \lambda X.(=\ X\ name\ ''arizona'')Y)$
7. resolving the lambda function simplifies this to
   $\lambda X.(:\ exists\ (Y)\ (state\ Y)\ (=\ X\ state\ Y\ name)\ (=\ Y\ name\ ''arizona''))$
8. the MRL from 7 is passed to the result from 3 resulting in
   $(X\ city\ \lambda X.(:\ exists\ (Y)\ (state\ Y)\ (=\ X\ state\ Y\ name)\ (=\ Y\ name\ ''arizona''))\ X)$
9. resolving the lambda function simplifies this to
   $(X\ city\ (:\ exists\ (Y)\ (state\ Y)\ (=\ X\ state\ Y\ name)\ (=\ Y\ name\ ''arizona'')))$
10. there is nothing more to read and no lambdas to apply, therefore this is our resulting MRL

Figure 2.3: Step by step semantic parsing using an λ-SCFG

2.2 Natural Language Interfaces to Databases

It is important to understand that achieving acceptable results over all of English is not currently feasible. This makes reduction of the problem size with domain restrictions very important.

To use the restricted domain property the domain must be communicated to the user in an clear way [3]. If the user does not understand the limitations of the NLI system the user will both ask questions that the NLI can not handle and avoid asking questions the user believes the NLI can not handle, neither which is very good. It is important for the NLI designer to understand this and ensure that human users understand such limitations and possibilities. If the NLI is instead expanded into related areas, the coverage of the NLI very quickly goes from being restricted to being general by recursively adding in more of English as user expectations increases.

As noted above this work focuses on natural language interfaces to databases. This is an important subdomain of NLIs for several reasons, the foremost being able to easily connect a natural language interfaces to the vast number of existing databases. Another reason is that databases are a well understood way to structure information and any system that can use a database as its memory can make use of techniques and advances in NLIs over databases[1].

The word ‘database’ is often used synonymously with ‘store of data’. For example text files, documents, PROLOG systems or XML files. This work uses a more strict view of what a database is, namely we use database as a shorthand for *relational database*. By assuming relational databases we can use the database constraints such as the primary and foreign keys. We also inherit all the other advantages of relational databases.

[1]Henceforth in this thesis when we refer to NLIs we are referring to NLIs over databases. The reasoning behind this is that the whole term is a bit vague and while there exists natural language interfaces to things that are not databases in most cases the same techniques can be used.

Aside A relational database can be seen as a set of tables. Each table has a number of attributes corresponding to table columns, rows in the table are tuples that have a value for each attribute. Tables are constrained by primary and foreign keys. A primary key is a unique value for some attribute in a table that uniquely identifies a tuple in the relation. A foreign key is an attribute in some table whose values are a subset of the values in the primary key column in some other table, this creates a relation between the data in the involved tables. This enables creation of links between arbitrary data, for example many-to-many relationships can be described with a table with two attributes that are both foreign keys to the table they link. As none of them are a primary key in that table their is no need for them to be unique and can as such be repeated to create links to the same data items several times.

Example schema for a database we might use

Tables	Attributes
STATE	NAME AREA POPULATION
CITY	NAME POPULATION
BORDER	STATE1 STATE2

2.2.1 Basic Approaches to NLIs

There is a long history of attempts to building natural language interfaces to databases. Since the beginning, several approaches have been tried and many systems have been built with varying degree of success.

Some early work used a *pattern matching approach* to build up natural language interfaces. The way they worked was that there was a stock of patterns that said, for example, if a city name follows a country name the response should be whether the city is in the country. By having a large number of such rules the builder of such an interface could make the system give reasonable answers to questions within its domain [2]. This approach is impractical and requires large, expensive to maintain, rule bases. Also the systems can not cover syntactically complex cases for example “give me two cities of the west that are not in california”.

Syntax based systems use a parse tree of the natural language to incorporate of understanding of sentence structure in the NLI. The first step for a syntax based system is to construct a parse tree of nouns, verbs and similar grammatical constructs. For every node in the parse tree there is then a mapping to a query language expression. These kind of systems usually use domain specific access methods as it is hard to make the mappings work to general languages like SQL [2]. An example of a syntax based system is LUNAR [19].

The *semantic grammar* approach is similar to the syntactic approach in that it uses parse trees of the natural language to decompose the input sentences. The key difference is that in a semantic grammar the inner nodes of the parse tree do not correspond to actual word classes. Instead they correspond to structure in the formal query language. This is used to encode semantic domain specific information into the rules of the language. The main weakness of semantic grammars is that they are time consuming to construct and lack portability between different domains as the rules correspond to concepts in the domain. This problem makes it quite prohibitive in terms of work to construct semantic grammars by hand. Regardless, several semantic grammar systems have been built. An early semantic grammar system was LADDER [8].

The idea behind the *transportable approach* is to translate the user's query into an expression in *logical form* expressing domain independent concepts about the world. This is independent of database structure and ensures that the work on the translation into this high level form can be reused when switching databases. This database independence enables the use of wide coverage parsers built by professional linguists to be used with any database. Additionally it allows for systems using multiple database simultaneously. This internal logical form must then be mapped into an actual database query by the system. Examples following this approach are CLE [1] and TEAM [7].

Although the transportable approach is promising there are profound difficulties with acquiring 'mapping' knowledge. There has been more practical success in using semantic grammar, and that is what this thesis will focus on.

2.3 Learning Semantic Grammars

Traditionally semantic grammars are hand built. This is time consuming, error prone, and motivates work on learning them instead. This could enable deployments of NLIs into new areas that can not afford hand built systems.

An early work in this area is Miller's work on the system HUM [11] which uses a syntax based approach with some domain specific productions. A problem for HUM is that it requires hard to acquire augmented parse trees. Mooney and Ge [6] presented an improved of this work with the system SCISSOR, which is able to handle for example deeper nesting of MR expressions. This share HUM's weakness in that it require annotated parse trees. By removing the syntactic annotations from the parse trees a simpler corpus can be used. Kate [10] describes the algorithm SILT that does this. More recent work by Mooney has produced algorithms like KRISP, WASP and λ-WASP algorithms [17]. The later using using SCFGs over the Geo880 database of US geography, which is the same as in our work. Zettlemoyer and Collins [21] treats the parse trees as hidden variables and in this way works around this limitation. Later work by Zettlemoyer and Collins [22] uses probabilistic Combinatorial Categorical Grammars, CCGs, that they call PCCG to handle speech related artifacts and flexible word order.

Several of these algorithms are using simple functional MRLs that works by mathematically constraining sets. For example WASP uses FunQL by Kate [10]. These languages have less expressive power that the lambda calculus based MRLs that are used in for example λ-WASP and this work. To handle those limitations meta predicates are often defined, though this makes the language highly domain dependant, something that the learning of semantic grammars was supposed to avoid in the first place.

Chapter 3

The C-Phrase Learner Approach

Our learning approach is based on Mooney and Wong's work [18] on λ-WASP [17]. The high level steps of the algorithm are the following, given an NL/MRL training-set:

1. Acquire word alignments from NL words to MRL productions in the training set
2. Extract rules from the training set based on the alignments
3. Use a statistical model to weight the rules to favor correct analyses

Figure 3.1 on the next page shows the architecture of the implemented system.

3.1 MRL Grammar Generation

We have used the C-PHRASE system to implement our learning system. C-PHRASE has an expressive MRL, see Appendix C for the MRL's BNF. The BNF grammar can not be used directly in our learning approach. The main problem is that many productions do not express verbalized meaning but rather build up structure. The published results about λ-WASP indicate that all MRL productions must express some meaning that is reflected in natural language.

This problem is solved by generating a new grammar using the database. The idea is to look at the primary and foreign keys of the database and from that generate productions using a template mechanism. By defining productions in this way we construct all productions that are needed to express common queries over the database. For an example generated grammar see Appendix D.

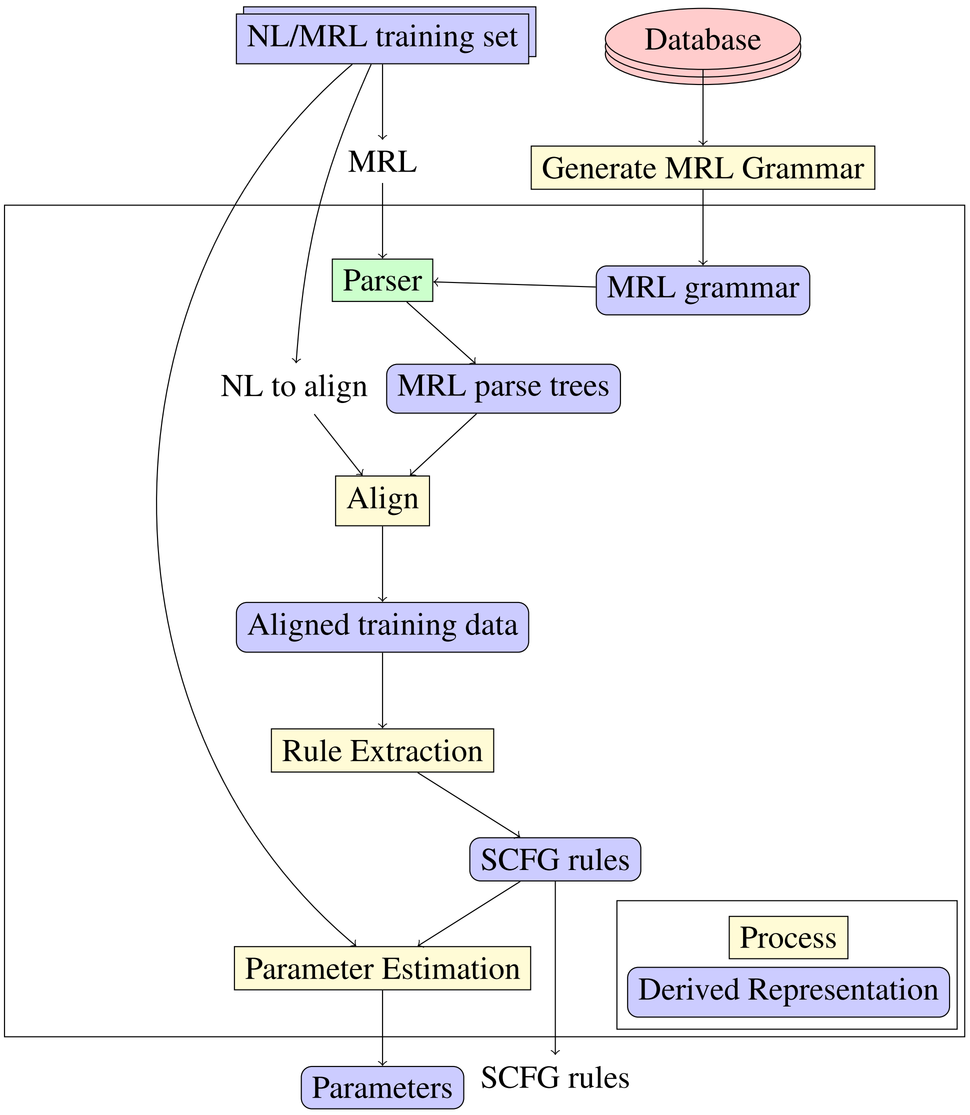


Figure 3.1: Architecture of the C-PHRASE learn systems

3.2 Calculating Alignments

In general a *word alignment* is a many-to-many relation describing which words in a source language correspond to which words in a target language. In this case the source language is English and the target language is production sequences of the C-PHRASE MRL. Our learning algorithm imposes restrictions that remove alignments that are not many-to-one. This makes every English word align with one or zero MRL-productions, and the alignments where an MRL-productions is not aligning to any words are discarded[1].

Figure 3.2 shows a visual example of word alignments.

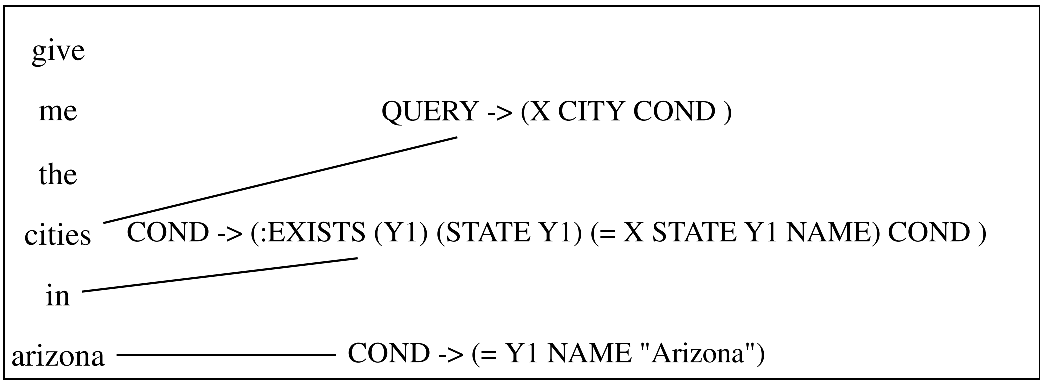

give
me
the
cities
in
arizona
QUERY -> (X CITY COND)
COND -> (:EXISTS (Y1) (STATE Y1) (= X STATE Y1 NAME) COND)
COND -> (= Y1 NAME "Arizona")

Figure 3.2: NL words in English aligning with MRL productions

Aligned MRL grammar productions are put into a linear form by a pre-order traversal of the parse tree adding productions as they first appear. The production numbers of these productions are then used as input to the alignment process.

3.2.1 Representation of Alignments as Trees

The pre-order traversal gives which, if any, parse tree node each word aligns with. We use this informations and the MRL parse tree to construct a larger tree where the inner nodes have additional children that are the aligning NL substrings. A node aligns with a NL substring that consists of two types of terminals. The first type are the NL words the MRL production directly align with. The second type are the nonterminals replacing the substrings that the node's children align with. Nonterminals are indexed to keep track of NL MRL correspondence.

The example in Figure 3.3 on the next page illustrates this. The terminals coming from the NL part of the training set are separated from the ones coming from the MRL part with a # sign.

[1] The exception to this last rule are lists where we have relaxed this restriction. Lists are represented as a list MRL part and an empty MRL part that terminates the lists.

Define aligning substring by their first and last aligning word or nonterminal. If an aligning substring contains a word or nonterminal that aligns with a node outside the current subtree, rule extraction fails and the alignment is discarded.

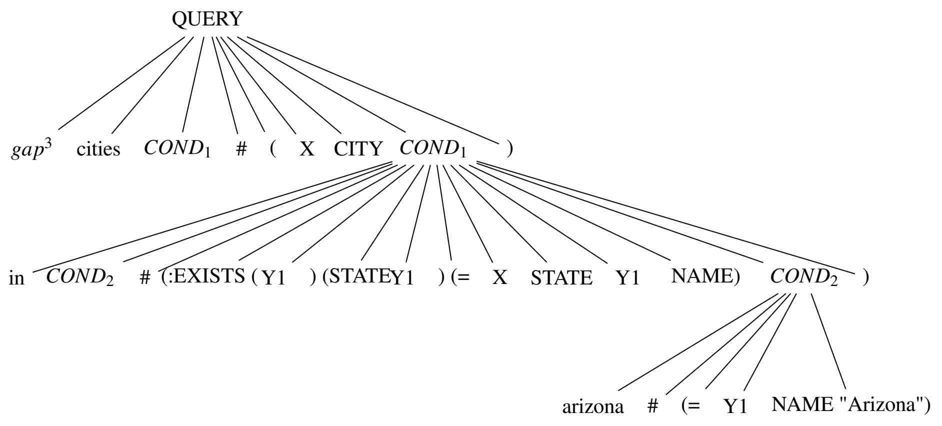

QUERY
gap3 cities COND1 # (X CITY COND1)
in COND2 # (:EXISTS (Y1) (STATE Y1) (= X STATE Y1 NAME) COND2)
arizona # (= Y1 NAME "Arizona")

Figure 3.3: Tree for the query “what are the cities in arizona”

3.2.2 Word Gaps

As can be seen in the tree of Figure 3.3, a special terminal gap^3 replaces the substring “what are the”. This is done under the condition that the NL words are not aligning with any MRL productions. Where they are attached to is determined as follows: if the word is inside a substring aligning with another production it is attached to that production, otherwise it is attached to the root node.

Intuitively, gaps are words and substrings that do not add any semantic content to the sentence. This can be because it is assumed from the context. For example “what are the” is a question prefix and our grammar assumes the user is asking questions. There are also commonly used words that only links things together, for example, “is” and “and”.

3.2.3 GIZA++

To obtain alignments the software package GIZA++ [15] is used. GIZA is a statistical machine translation tool that was extended by Franz Josef Och to create GIZA++. GIZA++ is run with the following command

```
GIZA++ -nbestalignments K -o dict -S nl.vcb -T mr.vcb -C corpus.snt
```

where **K** is substituted for a numerical constant describing how many potential alignments per sentence are sought, **dict** is the name of our output directory,

nl.vcb and **mr.vcb** are vocabulary files as described in GIZA++ documentation for the NL and MRL respectively, and **corpus.snt** is the training set encoded in the numerical format GIZA++ expects as input.

3.3 Rule Extraction

Rule extraction consumes the tree produced by the alignment process bottom-up. This can be seen as a type of bottom up tree transduction. For example the rules in Figure 2.3 on page 9 are extracted from the tree in Figure 3.2.1 on the preceding page.

Rule Extraction

Visit all the nodes in the tree bottom up, each node is visited only once and all child nodes must be visited before the parents.

- If the tree consists of only one symbol the process is finished
- If the node currently being visited is a leaf node leave it and continue
- Otherwise, call the nonterminal forming the root of this subtree N, call the list of children coming from the NL α, and call the list of children coming from the MRL β. Find the set of variables that are present both inside and outside the subtree N, these are the bound variables X. As a side effect save a rule on the form $N^k \rightarrow \langle \alpha, \lambda x_1 ... \lambda x_k \beta' \rangle$ where $x_1 ... x_k$ are the elements of X. β' is β with each nonterminal A replaced by $A x_{A_1} ... x_{A_j}$ where $x_{A_1} ... x_{A_j}$ are the variables bound by the subtree of A. For example if *Cond* binds x and y, *Cond* will be replaced by *Cond x y*. This is how lambda parameters are passed to subtrees. Save these rules and replace the subtree rooted in N with N^k and continue with the next node.

Rule Extraction in Pseudo Code

```
**ruleExtract**($tree$)
    $third$(**ruleExtractInternal**($tree, \{\}$))

**ruleExtractInternal**($tree, outerVariables$)
    $vars \leftarrow$ foreach $t$ in $childrenOf(tree)$ if $variable(t)$ collect $t$
    $variable(tree) \Rightarrow \langle tree, \{tree\}, \{\}\rangle$
    $leaf(tree) \Rightarrow \langle tree, \{\}, \{\}\rangle$
    $node(N, children) \Rightarrow$
        let
            $childSymbols, childVariables, rules \leftarrow$ foreach $c$ in $children$
                collect **ruleExtractInternal**($c, outerVariables \cup vars$)
            $innerVariables \leftarrow childVariables \cup vars$
            $\alpha, \beta \leftarrow$ split $childSymbols$ at #
            $boundVariables \leftarrow innerVariables \cap outerVariables$
            $k \leftarrow numberOf(boundVariables)$
        in
            $\langle N^k, boundVariables, rules \cup$
                $N^k \rightarrow \langle \alpha,$ (foreach $b$ in $boundVariables$ collect $\lambda b$)$\rangle$
                    (replace in $\beta$ using
                        $nonterminal(A) \Rightarrow$
                            ($A$ $childVariables$[indexOf $A$ in $childSymbols$])
                        $T \Rightarrow T$)
            $\rangle$
        end
```

3.4 Statistical Weighting

To train the statistical model, two steps are preformed. First a set of feature functions based on the training set are computed, then a vector θ of parameters is estimated.

3.4.1 Feature Functions

Feature functions are functions from derivations to real numbers. The set of feature functions is computed from the training set and is made up of the following types.

- A feature function for each rule in the SCFG describing how many times that rule is used in the derivation
- A feature function for each word in the training set counting how many times that word was in a word gap in the derivation
- A feature function counting how many words in the derivation were in word gaps

When all feature functions are applied to a derivation, a vector of numeric[2] values is generated. It is important to always use the same order when applying feature functions to preserve the order in which the vector represents specific feature functions.

[2]In our current set of features only integers are used

3.4.2 Parameter Estimation

The goal of parameter estimation is to find θ, where θ is a vector of length equal to the number of feature functions. In our case the length of these feature vectors is usually about 1200.

θ can later be used to find the probability of a derivation being the correct translation, given the input string. That is:

$$P(d|s) = \frac{e^{\theta \bar{f}}}{\sum\limits_{\bar{f}' \in F'} e^{\theta \bar{f}'}} \tag{3.1}$$

Where d is the derivation, s is the input string, $\bar{f}$ is the feature vector of d, F is the set of feature vectors for all derivations of s. To find the correct value for θ maximize the following expression over the sentences in the training set. Where NL_{corpus} is the NL part of the training set, $d_{correct}$ is the derivation for the correct translation of n.

$$\sum_{n \in NL_{corpus}} P(d_{correct}|n) \tag{3.2}$$

That is, compute the value of the following equation

$$\arg\max_{\theta} \left(\sum_{n \in NL_{corpus}} \frac{e^{\theta \bar{f}_{correct}}}{\sum\limits_{\bar{f}' \in F'} e^{\theta \bar{f}'}} \right) \tag{3.3}$$

To do this some parameter estimation technique, for example hill climbing[3], should be used.

[3]Zettlemoyer and Collins [22] use gradient ascent. We plan to do that as well in future work

Hill Climbing

Hill climbing is a heuristic search process that iteratively tries to achieve a higher score. When no higher score is possible we are at a local maxima and the process terminates. Hopefully the local maxima is also the global maxima.

Express the parameters to be estimated as a vector θ and give it an initial value. It can be random or all 1's or any other combination. We are using all 1's in this work.

Let Θ be the set containing each dimension of $\theta \pm\Delta$ where Δ is a small step size.
Repeat calculating the score[a] for each $\theta' \in \Theta$ and let $\theta = \max(\theta')$
until no $\theta' > \theta$ exists.

In pseudo code

```
loop
    $\Theta \leftarrow$ foreach $i$ from 0 below $lengthOf(\theta)$ collect
            let
                $t^+ \leftarrow \theta$
                $t^- \leftarrow \theta$
            in
                $t^+[i] += \Delta$
                $t^-[i] -= \Delta$
                yield $t^+$ and $t^-$
            end
    $Scores \leftarrow$ foreach $t$ in $\Theta$ do $score(t)$
    $\theta' \leftarrow maxScoring(Scores, \Theta)$
    if $\theta \geq \theta'$
        break
    $\theta \leftarrow \theta'$
end
```

[a]In our case computing the formula described in equation 3.2

The above is the basic hill climbing algorithm and there are improvements that can be made. This work uses a performance optimization that works as follows. When an improving direction has been found as, keep moving in that direction as long as it is improving, without looking around for other improving directions.

3.5 Using the Learned System

The output from the training is the SCFG created in the rule extraction step and the weights and features from the statistical weighting step. To use the system for translating English to MRL, first parse the English sentence with the SCFG. This will most likely generate multiple parse trees due to ambiguity in the SCFG grammar. These are then all scored by applying the feature functions on them, taking the dot product with the weight vector θ and selecting the largest resulting score as the top scoring analysis. Figure 3.4 shows the data flow in such a system.

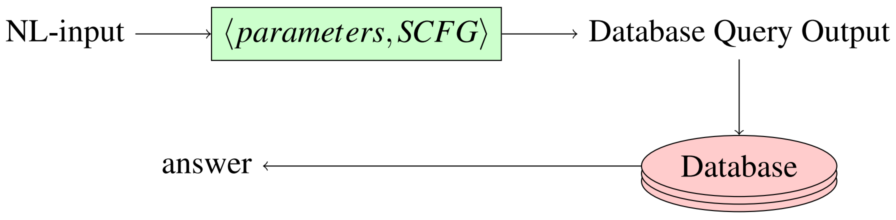


Figure 3.4: Data flow for query in learned system

Chapter 4

Results

When we test our system we select two disjoint subsets of the corpus, one *training set* and one *test set*. The training set is given to our algorithm which outputs a system that is capable of translating from the natural language into the MRL. This finished system is then tasked with analyzing the test set, the result of this is measured via the metrics *willingness*, *precision* and *recall* as follows[1].

$$willingness = \frac{\#correct + \#incorrect}{\#total}$$

$$precision = \frac{\#correct}{\#correct + \#incorrect}$$

$$recall = \frac{\#correct}{\#total}$$

Willingness is how likely the system is going to answer us at all. Precision describes how likely the system is to answer correctly if it answers. Recall is the product of the two and shows us how likely we are to get a correct answer. It is important to note that while it is desirable to have good values for all of them, a bad value for precision is worse as this can lead to incorrect answers that may not be detected. Basically the system will be lying.

To improve the statistical confidence of our results we are doing *n-fold cross validation*. This means running the system n times and taking the mean as the result. Therefor 5-fold cross validation runs the system 5 times. Note that this is per data point so if the system is run for 100, 150 and 200 items it means n times per problem size.

[1]*#Correct* refers to the correct answers the system gives, *#Incorrect* refers to the incorrect answers the system gives — Note that a refusal to answer is neither incorrect nor correct, *#Total* refers to all questions the system is asked

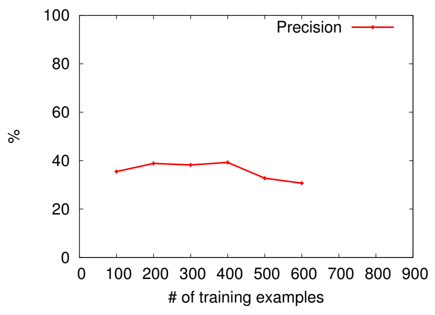
The main corpus we are running our tests on is our recoding of the GEOQUERY corpus used by Mooney and Woong, for example in Wong's thesis[17]. Examples of our recoding of GEOQUERY can be found in Appendix E.

4.1 Baseline Results

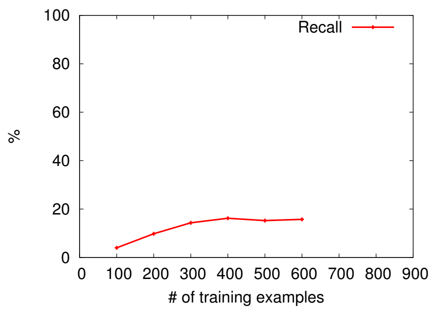
When training on our entire translation of the GEOQUERY corpus with 100 items test set and 5-fold cross validation.

Table 4.1: Willingness, Precision, and Recall for the full corpus

set size	willingness	precision	recall
100	0.1150	0.3549	0.0400
200	0.2640	0.3885	0.0980
300	0.3733	0.3823	0.1433
400	0.4120	0.3926	0.1620
500	0.4700	0.3274	0.1525
600	0.5125	0.3071	0.1575



(a) Graph of the precision



(b) Graph of the recall

Figure 4.1: Graphs of the result from the full corpus

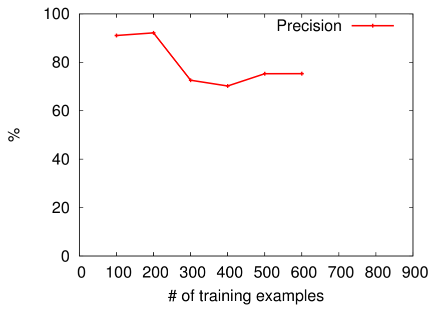
Due to slow execution of our system in the face of high ambiguity 5 out of 30 runs failed to finish in a timely fashion. As a result the cross validation is thus 3-fold for the 300 size and 4-fold for the 500 and 600 sizes.

4.2 Extended Algorithm

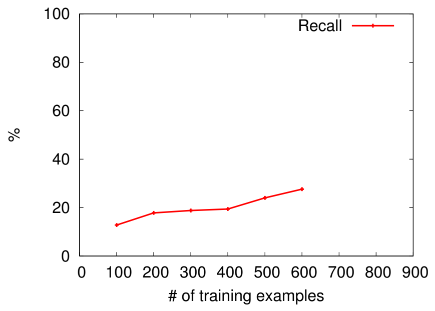
When using some small tweaks to the algorithm described bellow the following results are achieved. 100 items for comparison and 5-fold cross validation.

Table 4.2: Willingness, Precision, and Recall for the full corpus using extended algorithm

set size	willingness	precision	recall
100	0.1400	0.9105	0.1280
200	0.1920	0.9216	0.1780
300	0.2500	0.7259	0.1880
400	0.2780	0.7022	0.1940
500	0.3200	0.7528	0.2400
600	0.3660	0.7530	0.2760



(a) Graph of the precision



(b) Graph of the recall

Figure 4.2: Graphs of the result from the full corpus using extended algorithm

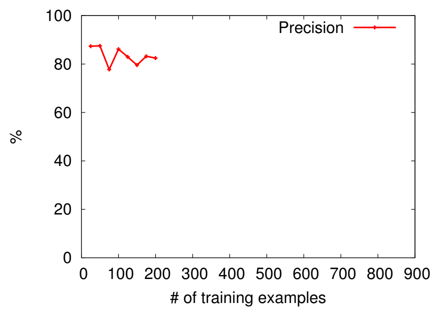
The tweaks consists of automatically wrapping each item in the MRL-part of the corpus with a rule that does nothing before sending it to GIZA, this causes GIZA to reduce gap sizes for the implied questions in the MRL that are explicit in the NL. We are also named using a initial version of entity recognition as described in subsection 5.2.4.

4.3 Clean Corpus

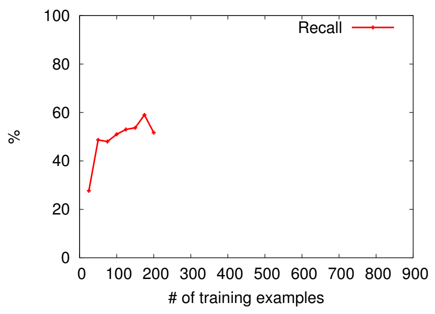
When training and testing on a clean corpus that has been carefully verified, the following results are gathered. Results from running with 30 items to compare the training to and 10-fold cross validation.

Table 4.3: Willingness, Precision, and Recall for a clean corpus

set size	willingness	precision	recall
100	0.5933	0.8615	0.5100
150	0.6800	0.7957	0.5367
200	0.6267	0.8248	0.5167



(a) Graph of the precision



(b) Graph of the recall

Figure 4.3: Graphs of the result from the clean corpus

4.4 Initial Study on Ambiguity

We have during the work seen indications that the ambiguity of the corpus is strongly affecting the experimental results.

In Table 4.4 are the results for 10-fold cross validation on a small corpus of 230, where 280 are used for training and 50 for testing. In the Bad corpus file 4 sentences where changed to an alternative meaning and thus translation results in ambiguities.

Table 4.4: Willingness, Precision, and Recall for a clean and an ambiguous corpus

	willingness	precision	recall
Bad data	0.6980	0.7800	0.5460
Good data	0.7000	0.8406	0.5880

As can be seen, the drop in precision is quite noticeable, 6%, after only 4 bad input sentences.

Chapter 5

Discussion

This thesis set out to replicate the results of λ-WASP and did so within reasonable error limits. We did this replication without communicating with the community to see how well the results could be replicated using only the published material as our guide. It can be noted that our results are slightly below those achieved by Mooney and Wong [18] this possibly means that there are some small tweaks that can be applied to boost our results or that we missed something obvious. When we in the future discuss this work with the community we hope to identify those tweaks, missed knowledge or the reason why our result was lower in the absence of tweaks. For example our more general MRL may be the reason.

Doing this replication has provided us with a testbed for further experimentation. This testbed, will enable experimentation with different aspects of NLI learning algorithms. However, there are some limitations which are described below. We also describe future avenues of research such as experiments to run and application domains that we can apply natural language understanding technologies to.

5.1 Difficulties

There are some minor problems with the C-PHRASE LEARNER that have not yet been fixed because they either are too complex or because of bugs in peripheral software.

5.1.1 Alignment Software

As our learning approach depends on getting word alignments to use in semantic analysis we keep having problems with our alignment software (GIZA++). The main problem has been that GIZA is returning a mix of too conservative and nonsensical alignments. Even in cases where all occurrences of a phrase are matched by some rule, GIZA chooses to align with only some words of the phrase. A possible solution is to use some other alignment algorithm such as Dice coefficients [5].

5.1.2 Ambiguity in Training Sets

As can be seen in Table 4.4 on page 27, the algorithm is sensitive to ambiguous input. This ambiguity can be things like the word 'largest'. If talking about cities it is usually, but not always, referring to *population*, but it can also be *area*. For states it is the other way around. This makes the word *largest* both context dependant and ambiguous, which then confounds the learning algorithm. This of course is a limitation and in a possible deployment scenario the administrator would have to be careful with corpus construction to avoid inputting bad training examples. The fact that the administrator has to take this kind of care precludes the use of raw user input as training data. A robust algorithm on the other hand could potentially do continuous training, if by no other method then by expanding the corpus with new data continuously and running the training on the current snapshot where the last run of the training finished. This way incrementality can be achieved even if the algorithm is not designed to support it.

5.1.3 Implementation Stability

While we would like our learning system to be rock solid and never crash, currently this is not the case. The stability issues are not mainly in our code although an occasional bug pops up there from time to time. The main stability issues stems from hard to track down bugs in the underlying CLISP implementation. The main symptoms are the following two. Sometimes when under some memory pressure the system spuriously reports a stack overflow and resets. If we increase stack size with compiler flags this limit increases slightly but then results in an unstable system and a segmentation fault. The second symptom is that in some cases, about 1 in 50, CLISP just closes pipes to external programs. Also this only happens under load and one theory is that CLISP receives an advisory signal or short read, becomes confused and errors out. If this happens the CLISP system is in an unknown and unstable state and has to be restarted. We are in the process of identifying exact error conditions to send to the CLISP developers.

5.2 Future System Improvements

During the work on this thesis some additional ideas for experiments were sparked. They could improve different aspects of the algorithms like quality of results and algorithmic complexity.

5.2.1 Exploration of Strategy Space

Even when the basic algorithm is fixed there are a lot of techniques that can optionally be used. If a technique is working well is then best demonstrated by showing performance with and without that technique by enabling and disabling the technique with the other techniques fixed to some setting. If the results then improve if the technique is on it can empirically be considered beneficial, in the context of the other enabled techniques, it may not be beneficial in some other context due to dependencies between techniques (i.e. one technique boosts or hinders another). To ease this kind of testing the C-PHRASE LEARNER should grow a system that allows for pluggable components for techniques so that adding or removing techniques becomes simple.

If the basic task is considered to be

1. pre-process corpus
2. compute alignments
3. extract grammar
4. filter wrong derivation
5. rank remaining derivations
6. test willingness, precision and recall

then for each step but the first and the last there should be a possibility to select a set of techniques to study.

5.2.2 Implementation of Gradient Descent

As was mentioned briefly in Section 3.4.2, there have been different ways to do parameter estimation. While the literature is a bit unclear on the different ways parameter estimation is done, the work by Zettlemoyer and Collins in [21] looks promising and would probably improve on our current hill climbing approach.

5.2.3 Integration of X-bar [9]

An idea that seems important but we have not yet had time to test is to integrate linguistic theory rules into the learned grammar. If this was done in a smart way the intuition is that it would improve the training results by making the parts of the language that has to be learned smaller. Possibly it could also give the effect that when these rules are used in parsing they will divide up the sentences into smaller parts that then have a lower probability of being wrong. This could be done by enabling the system to make use of the rules that are stored in the file `nlu/english/x-bar.lisp` in the C-PHRASE folder. The enabling part could be done piecewise by bringing in some rules at a time.

5.2.4 Improving Named Entity Recognition

Our technique for rule extraction does not know which words are named entities such as cities and states. This forces the corpus to contain all values for such entities. Otherwise the system will not be able to understand them. If the system is augmented to recognize when some part of the MRL is an attribute value, and it is aligning with the same attribute value on the NL side, then the system can generalize so that that value can be replaced by any value for that attribute in the relation. Doing this enables the system to understand for example which city is intended in a query even when the city was not in the training data.

For example the following rule

$$COND \rightarrow \langle arizona\ ,\ \lambda X.(=\ X\ name\ "arizona")\rangle$$

consists of a selection of the attribute value “arizona” from the database. The NL part of the rule is also “arizona” and if the system recognizes this it can generate a rule for all rows in the table that contain the value “arizona”. This would give us rules for all states even if only one was in the training data.

5.2.5 Utilizing Schema Meta-Information

Database schemas may have names for attributes and relations that are called the NL language word for that concept. If that is the case a kind of entity recognition similar to what is done in subsection 5.2.4 can be done to generalize queries on specific attributes and relations to speed up learning.

For example in

$$QUERY \rightarrow \langle every\ city\ in\ ,\ ((X\ CITY)\ (CONDITION^1\ X))\rangle$$

the word ‘city’ can be recognised in both the NL and MRL and therefor this rule can be generalized so that other attributes can be matched other than city

For example

$$QUERY \rightarrow \langle every\ river\ in\ ,\ ((X\ RIVER)\ (CONDITION^1\ X))\rangle$$

This can also be built so that it unifies the words to common case before matching, this would cause ‘cities’ and ‘CITY’ to match.

5.2.6 Using Dictionaries

For natural languages there are structure that can be extracted from sources such as dictionaries and schoolbook grammars. For example recognising common form or words that are synonyms.

5.2.7 Combining Domains

Consider the domains of Netbooks and Mobile Phones. Both of these domains contain information about availability, price, processor architecture, memory, network access, etc. They are also quite close to each other. If we have to small corpora, one for each domain, it might be possible to combine them.

One way would be to just concatenate the corpora, another to learn each domain separately then from the rules from each domain generate a huge training set. If learning on this huge training set gives good enough results in both of the domains and also on questions comparing the domains then the hope is to use this technique to combine information from small domains into larger domains, large domains can be seen as a compromise to wide coverage semantic parsing.

5.2.8 Ambiguity Aware Type-System

As natural languages are inherently ambiguous there will be sentences fragments that are ambiguous, some sentence fragments can be disambiguated from the surrounding context of the sentence however. If we recognise that the following query can return either the population or area

$$size\ of\ Washington,\ D.C.$$

in some contexts the most sensible thing to do is to return the most common or answer both questions. However if the whole sentence is

$$area\ of\ District\ of\ Columbia\ if\ the\ size\ of\ Washington,\ D.C.\ are\ removed$$

here the surrounding context makes it clear that area is intended.

If a type system was constructed that was able to handle ambiguous types such as $(AREA\ or\ POPULATION)$ then the use of those types in the surrounding context could be used to select the correct meaning.

5.2.9 Implementation of Node-merging

Currently the schema is used to generate a grammar that can be reduced to a subset of the MRL language that C-PHRASE uses. The reason the original MRL is not used is because in that MRL many productions have no correspondence to the natural language and just exists to build up the syntax. If node merging were to be implemented, productions could be marked as either alignable or non-alignable. This might enable the algorithm to do this work for us, automatically merging nodes to suitable size for use in the SCFG. This will not guarantee the same split that now exists but it is unclear which is better. Even if using the C-PHRASE MRL turns out to be unworkable, we might see some benefit from being able to utilize the small number of productions we now throw out because of bad alignments[1].

5.2.10 Expressing Intention in the MRL

While our MRL is quite expressive it does not have a construct that denotes that a query is a question and this thus have to be implied. While working over GEO-QUERY corpus this does not limit our ability to form correct MRL expressions for all the sentences but it cause us to have rather large gaps in the start of sentences for phrases like “what are the”. This can lead to cases that generates parses where words that bears semantic content is treated as a gap. Even if this is not the only parse it adds to the ambiguity and strains the statistical model.

The core limitation this exposes is that our MRL is only handling *semantics* and not *pragmatics*. Semantics here means what the information content that is transmitted, while pragmatics is the intended action the semantics is supposed to invoke. For example a question is expected to yield an answer, this is pragmatics, while whats asked about is semantics. Language constructs exists that form pragmatics for things like, orders, assertions and questions.

We hope to extend our learning system to be able to handle less constrained MRLs that are able to assert facts to the system. Although there may be problems with how a user would have to structure them to conform to the underlying database it is still worthwhile to be able to express for example assertions of facts. Interaction with the system could then be extended to a more complicated dialogue system to utilize this. A possibility raised by this is changing the database so that it supports not knowing the value of some attributes in relations.

[1]The case solved by node merging is one of two cases of bad alignments that causes training sentences to be rejected

5.2.11 Best First Parsing

We observed that the extracted SCFG can become very large, producing a huge number of parses. These are then weighted by our statistical model and the top one is selected. If it was possible to integrate a statistical model into a parser in a way that: Does a search for possible parses, finds the most probable first, or at least know when the most probable is found and find it early. And avoids exploring less probable regions of parse space by using the statistical model to assert that no high probability parse can be found in that region of the space of probable parses. That could speed up parsing. It also enables creating a cutoff that only finds the most probable parses. If there is any way to do this with our current statistical model, or some other, that would be worth exploring. It seems that Zettlemoyer and Collins [22] did something similar to this but it is not clear if their algorithm was probabilistic or if it guaranteed that only incorrect results are discarded. Their approach involves using beam search in the parsing to discard partial parses of low probability to keep the number of simultaneous potential parses low.

5.2.12 Tree Alignment

Reading the method description of this thesis closely reveals that the algorithm operates mostly on trees with one exception, the alignment. If we were to have a way to obtain alignment between the MRL parse tree and a syntax tree of the NL instead of the NL string, it could, apart from possibly saving us some work, also give us a cleaner division of which training data belongs to which production and thus might speed up learning. One idea is to start with a wide coverage parser (such as C&C [4]) and then do some statistical alignment between subtrees in the MRL parse tree and the syntax tree from the wide coverage parsers. If a reasonable many-to-one relation from NL syntax tree to MRL syntax tree can be found this might be interesting to explore.

5.3 Future Experiments

The C-PHRASE LEARNER system of this thesis will be able to serve as a testbed for several additional experiments.

5.3.1 Test with Different Corpora

Our results show the performance of the system over the GEOQUERY corpus, we have not been able to test how well it ports to other databases and corpora. A future task cold be to test the system using the RESTURANTS and JOBS corpora that Mooney has used in some work. It would also be interesting to create a couple of corpora with different complexity and test over them. Different types could include for example set constructs "people that are students or teachers at some school" or could include numerical attributes and comparisons "do the 10 top earning executives earn more together than the 1000 persons in middle management". To build these corpora one could start by building a database then a simple NLI to put on top, for example using C-PHRASE NLI construction tools. Releasing users onto the interface will then produce queries to collect. This would be especially simple if the system was either of practical use; where it answers question about for example schedules, lecturers and prerequisites; or if their was some incentive for the users for example a small monetary payment or search for a price Amazon's mechanical turk is a professional way to achieve this.

We already have begun some initial work on one corpus, a corpus that restricts users to the question domain of bus schedules in Umeå, work has been done to collect questions from people that could be potential users. Another idea is to build a corpus that ask questions about sports scores [13]. Here there could be information about different sports, people practicing them, which competitions where part of which tournaments, and so on. Given the number of people with an interest in sports this might be of practical use to build an NLI that works in this domain. Another idea would be a corpus over the domain of research publications. It would not be surprising if many researchers would be very happy to be able to get answers to questions of where things were originally defined and what is referring to what. Although here it is probably building the database that is the hard part as the data is scattered around quite a bit. A possible source of data is the is the Freebase (`www.freebase.com`) database. Freebase is a database containing content that is free to use under a Creative Common license and covers a wide range of topics.

5.3.2 Investigation of how Hard Different MRLs are to Learn

While exploring differences among C-PHRASE MRL and the MRL used in the original λ-WASP it occurred to me that there must be some properties in the MRLs that makes the difficulty of learning them different. I can see two things that would be able to explain these differences that could be explored. First, is the obvious case that the MRL should not contain unneeded complications such as being hard to parse or being ambiguous. Second, does the number of kinds[2] of MRL productions affect learning? Will a minimizing number of MRL production kinds needed to express the problem domain fully and unambiguously make learning easier?

It would be interesting to explore if there are any correlations between how easily an MRL can be learned automatically and how easy it is for users to learn it. A large correlation could be explained by both users and machines benefiting from clear domain restrictions.

[2]A 'kind' of MRL productions are here productions that are equal, apart from which relations, values and attributes they range over. This can also be expressed as having identical structure

Chapter 6

Conclusions

We have implemented a sub-system named C-PHRASE LEARNER in the freely available C-PHRASE [12] system that replicates Mooney and Wong's work on λ-WASP. The main point of our work is that we use a general purpose MRL over which we generate a more specific MRL-grammar using the database schema. Thus we have shown that portability between different databases can be achieved. We have also reformulated λ-WASP's [18] rule extraction algorithm as recursive operations on trees rather than string operations.

Although our results are slightly lower than what Mooney and Wong [18] achieved over their domain specific MRL, our replication study was done in isolation and we have not yet fully identified all the causes of the marginally lower results. We have identified two critical factors that degrade results. One is ambiguity in the training data. The second is the MRL-format. Simple experiments show that much better alignments are acquired using a slightly modified MRL. Finally we identify a collection of strategies that can be used to further refine C-PHRASE LEARNER.

We present a translation of the GEOQUERY corpus into C-PHRASE MRL, found in the file:

`http://code.google.com/p/c-phrase/demo/geo/corpora/mrl-corpus.crp`

Also available at `http://code.google.com/p/c-phrase/` is our CLISP compatible parallelization system used to parallelize learning. This is freely available along with the rest of C-PHRASE.

In total we conclude that learning NLIs over expressive MRLs currently has many open problems and interesting avenues for future work. We look forward presenting our isolated replication study to the community and to conduct more experiments in this promising area.

Chapter 7

Acknowledgements

Work on this thesis has sometimes been stressful culminating in a mad dash that seemed endless. Fortunately people around me have been nice and supportive and I would therefore like to thank primarily my thesis supervisor Michael Minock for continuous support and feedback during this work. He took time to explain many things to me, even things that afterwards felt basic. I would also like to thank the rest of the NFL research group for interesting discussions and for letting me practice presenting this material to them, having someone to explain to helped clarify it to me as well. I also extend a thank you to the support group at the Computer Science Department at Umeå University for having patience with me with broken software and my 'abuse' of lab equipment, and to Ola Ågren for taking the time to help me with LATEX and the arcane art of typography. Finally a thank you to friends and family for not forgetting me even when I disappear all too deep into work.

Bibliography

[1] H. Alshawi, editor. *The Core Language Engine*. MIT Press, Cambridge, Mass., and London, England, 1992.

[2] I. Androutsopoulos, G.D. Ritchie, and P. Thanisch. Natural language interfaces to databases–an introduction. *Natural Language Engineering*, 1(1):29–81, 1995.

[3] Ann Copestake and Karen Sparck Jones. Natural language interfaces to databases. *The Natural Language Review*, 5(4):225–249, 1990.

[4] James Curran, Stephen Clark, and Johan Bos. Linguistically motivated large-scale nlp with c&c and boxer. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 33–36, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[5] L.R. Dice. Measures of the amount of ecological association between species. *Journal of Ecology*, 26:297–302, 1945.

[6] Ruifang Ge and Raymond Mooney. A statistical semantic parser that integrates syntax and semantics. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 9–16, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[7] Barbara J. Grosz, Douglas E. Appelt, Paul A. Martin, and Fernando C. N. Pereira. Team: An experiment in the design of transportable natural-language interfaces. *AI*, 32(2):173–243, 1987.

[8] Gary G. Hendrix, Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. Developing a natural language interface to complex data. *ACM Trans. Database Syst.*, 3(2):105–147, 1978.

[9] Ray Jackendoff. *X-bar-Syntax: A Study of Phrase Structure, Linguistic Inquiry Monograph 2.* MIT Press, 1977.

[10] Rohit J. Kate and Raymond Mooney. Using string-kernels for learning semantic parsers. In *Proceedings of COLING/ACL-2006*, pages 913–920, 2006.

[11] Scott Miller, Robert Bobrow, Robert Ingria, and Richard Schwartz. Hidden understanding models of natural language. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 25–32, Morristown, NJ, USA, 1994. Association for Computational Linguistics.

[12] M. Minock. C-Phrase: A system for building robust natural language interfaces to databases. *Journal of Data and Knowledge Engineering*, 2009.

[13] Michael Minock. Where are the 'killer applications' of restricted domain question answering? In *Proceedings of Knowledge and Reasoning for Answering Questions (KRAQ)*, pages 98–101, Edinburgh, Scotland, 2005.

[14] Michael Minock. A STEP towards realizing Codd's vision of rendezvous with the casual user. In *33rd International Conference on Very Large Data Bases (VLDB)*, Vienna, Austria, 2007. Demonstration session.

[15] Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51, 2003.

[16] Bret Victor. Alligator eggs! a puzzle game, March 2010. `http://worrydream.com/AlligatorEggs/`.

[17] Yuk Wah Wong. *Learning for semantic parsing and natural language generation using statistical machine translation techniques*. PhD thesis, Austin, TX, USA, 2007. Adviser-Mooney, Raymond J.

[18] Yuk Wah Wong and Raymond J. Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. *ACL-07*, pages 960–967, 2007.

[19] William A. Woods, Ronald M. Kaplan, and Bonnie Lynn Nash-Webber. The lunar sciences natural language information system. Technical Report TR 2378, Bolt, Beranek and Newman, Inc., June 1972.

[20] John M. Zelle and Raymond J. Mooney. Learning to parse database queries using inductive logic programming. In *In Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1050–1055, 1996.

[21] Luke S. Zettlemoyer and Michael Collins. Online learning of relaxed ccg grammars for parsing to logical form. In *Proceedings of the 2007 Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2007.

[22] Luke S. Zettlemoyer and Michael Collins. Learning context-dependent mappings from sentences to logical form. In *ACL-IJCNLP '09: Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2*, pages 976–984, Morristown, NJ, USA, 2009. Association for Computational Linguistics.

Appendix A

User's Guide to the C-PHRASE Learner

The learning system described in this thesis is implemented as a part of the C-PHRASE distribution, available at `http://code.google.com/p/c-phrase/`. It is implemented in a LISP package called *:learn* and uses other parts of C-PHRASE to parse and compare queries among other things.

Regardless of how the system is started there are a couple of steps that are done to run the training. The corpus must be read from a file, suitable parts of it selected to run training on, training has to be run and finally some kind of result has to be returned. Depending on the way the system is started this can be statistics about precision and recall or a system that is capable of mapping natural language input to MRL expressions.

A.1 Running Learning from within C-PHRASE

Depending on ones goal there are three main ways to run the system from within C-PHRASE.

If one wants to answer natural language input queries the function `cpl-train` is most suitable to use. Parameters are a string representing the filename of the corpus to use, a MRL grammars of the type returned by the function `generate-mrl` and an integer describing how many possible alignments to maximally use in the training. The first return value is a function that translates NL strings into MRL. The other return values are meant for internal use.

Example call

```
(setf *trained* (first (cpl-train "corpus.txt" (generate-mrl) 5)))
```

If one wants to test the system against unseen data the function

```
test-cpl-learning
```

is used. This function is called with the arguments

```
corpus-filename training-size test-size [*random-generator-state*]
```

where the first takes a filename as a string the second and third integers describing how many sentences to train on and how many to test on to produce willingness, precision, and recall measures. The last parameter is optional and is used to seed the random number generator.

Example call

```
(test-cpl-learning "corpus.crp" 100 50 :state (make-random-state t))
```

If one wants to test how closely the system memoizes the inputs and correctly sort out ambiguities in known input, the function `validate-cpl-learning` is used. It works like the previous function but here only a single size parameter is supplied and that amount of sentences from the corpus are selected once and then used both as training and test set.

A.2 Running Learning from Scripts

To run the system from the command line the following `cpl-experiment` script exists.

It takes the following 12 parameters (in order):

- C-PHRASE configuration containing among other things the database schema, for example `geo.cph`
- the corpus file
- an integer signifying the size of the test set
- an integer signifying the size of the training set
- an integer signifying the number of alignments to use, .
- a boolean (t or nil) that enables rule-merging
- a boolean (t or nil) that enables the 'request' MRL modeling strategy
- a boolean (t or nil) that enables peeking, this is never enabled in our results unless clearly stated and is intended for debugging
- minimum probability for considered alignments
- max derivations to consider
- a boolean (t or nil)
- a comma separated list of words that might not align

Running this script will produce a directory with some files, among them a one line csv-file containing precision recall and willingness.

The usage of the program looks as follows

cpl-experiment CPH-FILE CORPUS-FILE TEST-SIZE TRAINING-SIZE
K-ALIGNMENTS MERGE-ON-OR-OF REQUEST-ON-OR-OF
PEEKING-ON-OR-OF ALIGN-CUTOF MAX-DERIVATIONS
NAMED-ENTITY-RECOGNISION ALIGN-STOPWORDS

Example call

```
cpl-experiment geo.cph mrl-corpus.crp 100 500 5 nil t nil 0.05 16 t "that,is"
```

Appendix B

Parallel Job Distribution Protocol

To speed up computation a parallel computation system has been employed to implement this software. The program written to do this is called `slavedriver`[1] and is distributed along with the system as a C source file. The slavedriver program is not tied to this system but defines a general purpose protocol for distribution of jobs to state-full nodes over SSH.

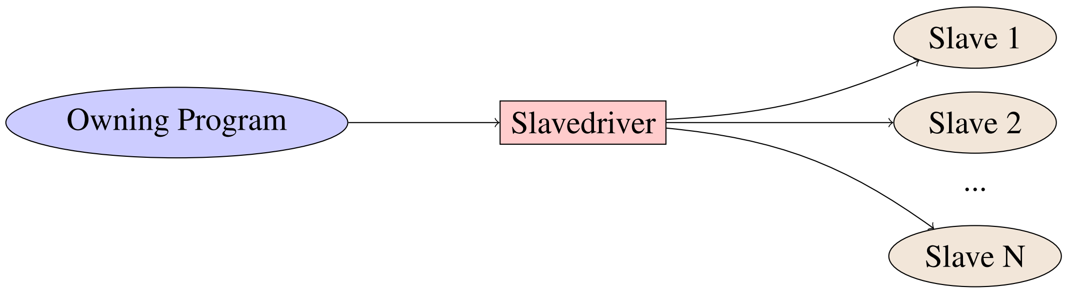


Figure B.1: Architecture of the slavedriver program

B.1 Program Compilation and Startup

The slavedriver program is intended to be used by other applications. To use it commands are sent to its standard input and replies read from its standard output. Slavedriver starts the remote nodes and sends commands on their standard input and reads back their replies on their standard output. The programs are supposed to be closed down when the standard input of the slavedriver process is closed.

[1]The master slave terminology is used to stay consistent with other network protocols

The slavedriver program is implemented in a file slavedriver.c, found in the src directory in the C-PHRASE distribution. To compile slavedriver type

```
make bin/slavedriver
```

in the C-PHRASE directory.

The binary can then be run by typing:

```
slavedriver remote-program
```

Running it manually is very cumbersome and it is assumed that this will be run from a host program that has jobs to distribute.

B.2 Job Dispatch Protocol

To dispatch jobs and state, and to receive the results of jobs the following messages and replies are used.

The notation works as follows, text in `typewriter` font should be interpreted as those letters in ASCII unless they are enclosed in square brackets in which case the square brackets will contain a textual description what they must be replaced with. All numbers are represented as base 10 written out as ASCII. As per convention in many languages, strings of the form \n stand for the newline character with ASCII value 10.

The protocol guarantees that broadcast jobs are transmitted to all hosts in the order they are received by the slavedriver system. Additionally no ordinary job that is issued after a broadcast job may be executed earlier than that broadcast job on any node. The order replies to ordinary jobs are received are the same order in which they were submitted.

If a remote node crashes the slavedriver program tries to resubmit any job it was working on to some other worker node.

- `b[number of bytes in the job string]\n[job string]`
  the job string is transmitted to all remote workers. Nothing is returned

- `j[number of bytes in the job string]\n[job string]`
  the job string is transmitted to one worker. The return message can be fetched with the g command

- `g\n`
  this prints to stdout the answer to the oldest message submitted, blocking if necessary

- `c\n`
  waits for all jobs to complete, discards their replies and prints nothing
- `a[hostname as a string of max 1023 charachters]\n`
  adds this host as a new compute host, any jobs started after this can be run on this node
- `r[hostname as a string of max 1023 charachters]\n`
  removes all children with this name, NOTE this is not the inverse of the a operation this removes ALL children added on the node n not one of them so that a and r would be symmetric. Children are removed once their current task is finished.

B.3 Worker Program Protocol

Commands sent from the slavedriver program to the remote hosts follow no determined syntax, that is up to the user of slavedriver. The remote programs for the learn system expects new line terminated strings as input. For every message received by the remote program it is expected to produce a reply, the reply can be anything but must follow the following syntax, same notation as in the last section.

```
RESULT:FROM:HOST:TO:MANAGER [number of bytes in response]\n[response string]
```

This is to be printed to standard output. All other messages printed to standard output and all messages printed to standard error are treated as debug messages. Slavedriver will print them to standard error on the controlling host[2].

[2]The reprinting of standard-error messages is broken due to buggy dependencies but printing of standard-out error-messages works

Appendix C

The C-PHRASE Meaning Representation Language (MRL)

The C-PHRASE MRL that has been used in this work is an extension of tuple calculus using a LISP syntax. Tuple calculus is well defined and of the same expressive power as first order logic. This makes it slightly less expressive than pragmatic database languages such as SQL. Therefor we need an extended tuple calculus to use with the C-PHRASE MRL for expressing thing like counting the number of tuples in an expression, to express superlatives such as top and bottom or to sum the values of some attribute in a relation. By adding extensions like this, basically a small set of arbitrary functions from tuples to tuples, we can increase the expressive power to that of SQL while having a relatively regular and simple language to work with. This then gives us the power of SQL without its syntax.

The C-PHRASE MRL has the following syntax in BNF notation

```
Query -> Tuple-Query | Projection-Query |
         Treated-Tuple-Query | Treated-Projection-Query

Tuple-Query -> (Var Relation Condition*)

Treated-Tuple-Query -> ((TupleFunction Var) Relation Condition*)

Projection-Query -> ((Var Attribute+) Relation  Condition*)

Treated-Projection-Query ->((AttrFunction (Var Attribute+)) Relation Condition*)

TupleFunction -> :truth | :count

AttrFunction -> :sum | :avg

Condition -> Range-Condition | Simple-Condition | Set-Condition |
             Join-Condition | Component-Condition | Cardinality-Condition |
             Tuple-Inequality-Condition | Superlative-Condition

Range-Condition -> (Relation  Var)

Simple-Condition -> (Operator  Var  Attribute  Value)

Set-Condition -> (SetOperator  Var  Attribute  Set-Value)

Join-Condition -> (Operator  Var  Attribute  Var  Attribute)

Component-Condition -> (Quantifier  (Var+)  Condition* )

Cardinality-Condition -> (Card  IntValue  Var  Condition*)

Tuple-Inequality-Condition -> (<>   Var  Var)

Superlative-Condition -> (Sup  IntValue  Attribute  Var)

Quantifier ->  :exists | :not-exists

Card -> :at-least | :at-most | :exactly | :max | :min | :the-most | :the-least

Sup -> :top | :bottom

Operator ->  = | > | < | >= | <= | <> | :like | :not-like

Set-Operator ->  :in  |  :not-in

Value -> NUMBER | STRING | PARAMETER | Date

SetValue -> (NUMBER*)  |  (STRING*)  |  Parameter

Number ->  INT  |  REAL

String -> "STRING"

IntValue -> INT  | PARAMETER

Date -> "YYYY-MM-DD"

Parameter -> $INDENTIFIER

Var -> INDENTIFIER
```

Appendix D

Generated MRL from the GEOQUERY Schema

As explained in Chapter 3.1 on page 14 we generate our MRL from the database schema to make our system portable between databases. Here we show a MRL generated for the database in 2.2, over 5 attribute values per non numerical attribute. The start symbol for the grammar is QUERY.

The examples uses the following schema:

Relations	Attributes
STATE	<u>NAME</u> AREA POPULATION
CITY	<u>NAME</u> POPULATION
BORDER	STATE1 STATE2

Productions over the STATE relation

```
(QUERY -> "(" 1? "STATE" CONDLIST ")" )
(QUERY -> "((:count" 1? ") STATE" CONDLIST ")" )
(QUERY -> "((:truth" 1? ") STATE" CONDLIST ")" )
(CONDITION -> "(:TOP 1 AREA" 1? ")" )
(CONDITION -> "(:BOTTOM 1 AREA" 1? ")" )
(CONDITION -> "(>" 1? "AREA 200000)" )
(CONDITION -> "(:not-exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE" 1? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE" 1? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (STATE" 1? ")" "(=" 0? "NAME" 1? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (STATE" 1? ")" "(=" 0? "NAME" 1? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (STATE" 1? ")" "(=" 0? "POPULATION" 1? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (STATE" 1? ")" "(=" 0? "POPULATION" 1? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE1" 1? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE1" 1? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE2" 1? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE2" 1? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE" 1? "CAPITAL" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE" 1? "CAPITAL" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (STATE" 1? ")" "(=" 0? "NAME" 1? "CAPITAL" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (STATE" 1? ")" "(=" 0? "NAME" 1? "CAPITAL" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (STATE" 1? ")" "(=" 0? "POPULATION" 1? "CAPITAL" ")" CONDLIST ")" )
```

```
(CONDITION -> "(:exists (" 1? ") (STATE" 1? ")" "(=" 0? "POPULATION" 1? "CAPITAL" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE1" 1? "CAPITAL" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE1" 1? "CAPITAL" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE2" 1? "CAPITAL" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE2" 1? "CAPITAL" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE" 1? "POPULATION" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE" 1? "POPULATION" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (STATE" 1? ")" "(=" 0? "NAME" 1? "POPULATION" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (STATE" 1? ")" "(=" 0? "NAME" 1? "POPULATION" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (STATE" 1? ")" "(=" 0? "POPULATION" 1? "POPULATION" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (STATE" 1? ")" "(=" 0? "POPULATION" 1? "POPULATION" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE1" 1? "POPULATION" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE1" 1? "POPULATION" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE2" 1? "POPULATION" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE2" 1? "POPULATION" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE" 1? "AREA" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE" 1? "AREA" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (STATE" 1? ")" "(=" 0? "NAME" 1? "AREA" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (STATE" 1? ")" "(=" 0? "NAME" 1? "AREA" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (STATE" 1? ")" "(=" 0? "POPULATION" 1? "AREA" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (STATE" 1? ")" "(=" 0? "POPULATION" 1? "AREA" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE1" 1? "AREA" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE1" 1? "AREA" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE2" 1? "AREA" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (STATE" 1? ")" "(=" 0? "STATE2" 1? "AREA" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? 2? ") (BORDER" 1? ") (STATE" 2? ")"
                           "(=" 0? "NAME" 1? "STATE2" ")" "(=" 1? "STATE1" 2? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? 2? ") (BORDER" 1? ") (STATE" 2? ")"
                       "(=" 0? "NAME" 1? "STATE2" ")" "(=" 1? "STATE1" 2? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? 2? ") (BORDER" 1? ") (STATE" 2? ")"
                           "(=" 0? "NAME" 1? "STATE1" ")" "(=" 1? "STATE2" 2? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? 2? ") (BORDER" 1? ") (STATE" 2? ")"
                       "(=" 0? "NAME" 1? "STATE1" ")" "(=" 1? "STATE2" 2? "NAME" ")" CONDLIST ")" )
(QUERY -> "((:sum (" 1? "NAME)) STATE" CONDLIST ")" )
(QUERY -> "((:avg (" 1? "NAME)) STATE" CONDLIST ")" )
(QUERY -> "((" 1? "NAME) STATE" CONDLIST ")" )
(CONDITION -> "(=" 1? "NAME \"Wisconsin\")" )
(CONDITION -> "(=" 1? "NAME \"West Virginia\")" )
(CONDITION -> "(=" 1? "NAME \"Washington\")" )
(CONDITION -> "(=" 1? "NAME \"Virginia\")" )
(QUERY -> "((:sum (" 1? "CAPITAL)) STATE" CONDLIST ")" )
(QUERY -> "((:avg (" 1? "CAPITAL)) STATE" CONDLIST ")" )
(QUERY -> "((" 1? "CAPITAL) STATE" CONDLIST ")" )
(CONDITION -> "(=" 1? "CAPITAL \"Washington\")" )
(CONDITION -> "(=" 1? "CAPITAL \"Trenton\")" )
(CONDITION -> "(=" 1? "CAPITAL \"Topeka\")" )
(CONDITION -> "(=" 1? "CAPITAL \"Tallahassee\")" )
(CONDITION -> "(=" 1? "CAPITAL \"St. Paul\")" )
(QUERY -> "((:sum (" 1? "POPULATION)) STATE" CONDLIST ")" )
(QUERY -> "((:avg (" 1? "POPULATION)) STATE" CONDLIST ")" )
(QUERY -> "((" 1? "POPULATION) STATE" CONDLIST ")" )
(QUERY -> "((:sum (" 1? "AREA)) STATE" CONDLIST ")" )
(QUERY -> "((:avg (" 1? "AREA)) STATE" CONDLIST ")" )
(QUERY -> "((" 1? "AREA) STATE" CONDLIST ")" )
```

Productions over the CITY relation

```
(QUERY -> "(" 1? "CITY" CONDLIST ")" )
(QUERY -> "((:count" 1? ") CITY" CONDLIST ")" )
(QUERY -> "((:truth" 1? ") CITY" CONDLIST ")" )
(CONDITION -> "(:TOP 1 POPULATION" 1? ")" )
(CONDITION -> "(:BOTTOM 1 POPULATION" 1? ")" )
(CONDITION -> "(>" 1? "POPULATION 200000)" )
(CONDITION -> "(:not-exists (" 1? ") (CITY" 1? ")" "(=" 0? "NAME" 1? "STATE" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (CITY" 1? ")" "(=" 0? "NAME" 1? "STATE" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (CITY" 1? ")" "(=" 0? "CAPITAL" 1? "STATE" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (CITY" 1? ")" "(=" 0? "CAPITAL" 1? "STATE" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (CITY" 1? ")" "(=" 0? "POPULATION" 1? "STATE" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (CITY" 1? ")" "(=" 0? "POPULATION" 1? "STATE" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (CITY" 1? ")" "(=" 0? "AREA" 1? "STATE" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (CITY" 1? ")" "(=" 0? "AREA" 1? "STATE" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (CITY" 1? ")" "(=" 0? "NAME" 1? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (CITY" 1? ")" "(=" 0? "NAME" 1? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (CITY" 1? ")" "(=" 0? "CAPITAL" 1? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (CITY" 1? ")" "(=" 0? "CAPITAL" 1? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (CITY" 1? ")" "(=" 0? "POPULATION" 1? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (CITY" 1? ")" "(=" 0? "POPULATION" 1? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (CITY" 1? ")" "(=" 0? "AREA" 1? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (CITY" 1? ")" "(=" 0? "AREA" 1? "NAME" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (CITY" 1? ")" "(=" 0? "NAME" 1? "POPULATION" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (CITY" 1? ")" "(=" 0? "NAME" 1? "POPULATION" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (CITY" 1? ")" "(=" 0? "CAPITAL" 1? "POPULATION" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (CITY" 1? ")" "(=" 0? "CAPITAL" 1? "POPULATION" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (CITY" 1? ")" "(=" 0? "POPULATION" 1? "POPULATION" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (CITY" 1? ")" "(=" 0? "POPULATION" 1? "POPULATION" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (CITY" 1? ")" "(=" 0? "AREA" 1? "POPULATION" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (CITY" 1? ")" "(=" 0? "AREA" 1? "POPULATION" ")" CONDLIST ")" )
(QUERY -> "((:sum (" 1? "STATE)) CITY" CONDLIST ")" )
(QUERY -> "((:avg (" 1? "STATE)) CITY" CONDLIST ")" )
(QUERY -> "((" 1? "STATE) CITY" CONDLIST ")" )
(CONDITION -> "(=" 1? "STATE \"Wyoming\")" )
(CONDITION -> "(=" 1? "STATE \"Wisconsin\")" )
(CONDITION -> "(=" 1? "STATE \"West Virginia\")" )
(CONDITION -> "(=" 1? "STATE \"Washington\")" )
(CONDITION -> "(=" 1? "STATE \"Virginia\")" )
(QUERY -> "((:sum (" 1? "NAME)) CITY" CONDLIST ")" )
(QUERY -> "((:avg (" 1? "NAME)) CITY" CONDLIST ")" )
(QUERY -> "((" 1? "NAME) CITY" CONDLIST ")" )
(CONDITION -> "(=" 1? "NAME \"Youngstown\")" )
(CONDITION -> "(=" 1? "NAME \"Yonkers\")" )
(CONDITION -> "(=" 1? "NAME \"Wyoming\")" )
(CONDITION -> "(=" 1? "NAME \"Worcester\")" )
(CONDITION -> "(=" 1? "NAME \"Woodbridge\")" )
(QUERY -> "((:sum (" 1? "POPULATION)) CITY" CONDLIST ")" )
(QUERY -> "((:avg (" 1? "POPULATION)) CITY" CONDLIST ")" )
(QUERY -> "((" 1? "POPULATION) CITY" CONDLIST ")" )
```

Productions over the BORDER relation

```
(QUERY -> "(" 1? "BORDER" CONDLIST ")" )
(QUERY -> "((:count" 1? ") BORDER" CONDLIST ")" )
(QUERY -> "((:truth" 1? ") BORDER" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (BORDER" 1? ")" "(=" 0? "NAME" 1? "STATE1" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (BORDER" 1? ")" "(=" 0? "NAME" 1? "STATE1" ")" CONDLIST ")" )
```

```
(CONDITION -> "(:not-exists (" 1? ") (BORDER" 1? ")" "(=" 0? "CAPITAL" 1? "STATE1" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (BORDER" 1? ")" "(=" 0? "CAPITAL" 1? "STATE1" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (BORDER" 1? ")" "(=" 0? "POPULATION" 1? "STATE1" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (BORDER" 1? ")" "(=" 0? "POPULATION" 1? "STATE1" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (BORDER" 1? ")" "(=" 0? "AREA" 1? "STATE1" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (BORDER" 1? ")" "(=" 0? "AREA" 1? "STATE1" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (BORDER" 1? ")" "(=" 0? "NAME" 1? "STATE2" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (BORDER" 1? ")" "(=" 0? "NAME" 1? "STATE2" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (BORDER" 1? ")" "(=" 0? "CAPITAL" 1? "STATE2" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (BORDER" 1? ")" "(=" 0? "CAPITAL" 1? "STATE2" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (BORDER" 1? ")" "(=" 0? "POPULATION" 1? "STATE2" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (BORDER" 1? ")" "(=" 0? "POPULATION" 1? "STATE2" ")" CONDLIST ")" )
(CONDITION -> "(:not-exists (" 1? ") (BORDER" 1? ")" "(=" 0? "AREA" 1? "STATE2" ")" CONDLIST ")" )
(CONDITION -> "(:exists (" 1? ") (BORDER" 1? ")" "(=" 0? "AREA" 1? "STATE2" ")" CONDLIST ")" )
(QUERY -> "((:sum (" 1? "STATE1)) BORDER" CONDLIST ")" )
(QUERY -> "((:avg (" 1? "STATE1)) BORDER" CONDLIST ")" )
(QUERY -> "((" 1? "STATE1) BORDER" CONDLIST ")" )
(CONDITION -> "(=" 1? "STATE1 \"Wyoming\")" )
(CONDITION -> "(=" 1? "STATE1 \"Wisconsin\")" )
(CONDITION -> "(=" 1? "STATE1 \"West Virginia\")" )
(CONDITION -> "(=" 1? "STATE1 \"Washington\")" )
(CONDITION -> "(=" 1? "STATE1 \"Virginia\")" )
(QUERY -> "((:sum (" 1? "STATE2)) BORDER" CONDLIST ")" )
(QUERY -> "((:avg (" 1? "STATE2)) BORDER" CONDLIST ")" )
(QUERY -> "((" 1? "STATE2) BORDER" CONDLIST ")" )
(CONDITION -> "(=" 1? "STATE2 \"Wyoming\")" )
(CONDITION -> "(=" 1? "STATE2 \"Wisconsin\")" )
(CONDITION -> "(=" 1? "STATE2 \"West Virginia\")" )
(CONDITION -> "(=" 1? "STATE2 \"Washington\")" )
(CONDITION -> "(=" 1? "STATE2 \"Virginia\")" )
```

Special productions

These productions are always present. The number of variable rules should get adjusted to the number of variables in the training set.

```
(0? -> "X")
(1? -> "X")
(2? -> "X")
(0? -> "Y1")
(1? -> "Y1")
(2? -> "Y1")
(0? -> "Y2")
(1? -> "Y2")
(2? -> "Y2")
(0? -> "Y3")
(1? -> "Y3")
(2? -> "Y3")
(0? -> "Y4")
(1? -> "Y4")
(2? -> "Y4")
(0? -> "Y5")
(1? -> "Y5")
(2? -> "Y5")
(0? -> "Y6")
(1? -> "Y6")
(2? -> "Y6")
(CONDLIST -> NIL)
(CONDLIST -> CONDITION CONDLIST)
```

Appendix E

A subset of GEOQUERY coded in C-PHRASE's MRL

give me the cities in virginia

```
(X CITY (:EXISTS (Y1) (STATE Y1) (= X STATE Y1 NAME) (= Y1 NAME "Virginia")))
```

what are the high points of states surrounding mississippi

```
(X HIGH (:EXISTS (Y1) (STATE Y1) (= X STATE Y1 NAME)
(:EXISTS (Y2 Y3) (BORDER Y2) (STATE Y3) (= Y1 NAME Y2 STATE1)
(= Y2 STATE2 Y3 NAME) (= Y3 NAME "Mississippi"))))
```

name the rivers in arkansas

```
(X RIVER (:EXISTS (Y1 Y2) (RIVERFLOWSTHROUGH Y1) (STATE Y2)
(= X NAME Y1 RIVER) (= Y1 STATE Y2 NAME) (= Y2 NAME "Arkansas")))
```

can you tell me the capital of texas

```
((X CAPITAL) STATE (= X NAME "Texas"))
```

could you tell me what is the highest point in the state of oregon

```
(X HIGH (:TOP 1 ELEVATION X) (:EXISTS (Y1) (STATE Y1) (= X STATE Y1 NAME)
(= Y1 NAME "Oregon")))
```

give me all the states of usa

```
(X STATE)
```

give me the cities in usa

```
(X CITY)
```

give me the lakes in california

```
(X LAKE (:EXISTS (Y1 Y2) (LAKEINSTATE Y1) (STATE Y2) (= X NAME Y1 LAKE)
(= Y1 STATE Y2 NAME) (= Y2 NAME "California")))
```

give me the largest state

```
(X STATE (:TOP 1 AREA X))
```

give me the longest river that passes through the us

```
(X RIVER (:TOP 1 LENGTH X))
```

give me the number of rivers in california

```
((:COUNT X) RIVER (:EXISTS (Y1 Y2) (RIVERFLOWSTHROUGH Y1) (STATE Y2)
(= X NAME Y1 RIVER) (= Y1 STATE Y2 NAME) (= Y2 NAME "California")))
```

give me the states that border utah

```
(X STATE (:EXISTS (Y1 Y2) (BORDER Y1) (STATE Y2) (= X NAME Y1 STATE1)
(= Y1 STATE2 Y2 NAME) (= Y2 NAME "Utah")))
```

how big is alaska

```
((X AREA) STATE (= X NAME "Alaska"))
```

what state has the smallest population

```
(X STATE (:BOTTOM 1 POPULATION X))
```

what state has the smallest area

```
(X STATE (:BOTTOM 1 AREA X))
```

what state has the smallest population density

```
(X STATE (:BOTTOM 1 DENSITY X))
```

how big is north dakota

```
((X AREA) STATE (= X NAME "North Dakota"))
```

how high are the highest points of all the states

```
((X ELEVATION) HIGH)
```

how high is guadalupe peak

```
((X ELEVATION) HIGH (= X POINT "Guadalupe Peak"))
```

how high is the highest point in america

```
((X ELEVATION) HIGH (:TOP 1 ELEVATION X))
```

how high is the highest point in the largest state

```
((X ELEVATION) HIGH (:TOP 1 ELEVATION X) (:EXISTS (Y1) (STATE Y1)
(= X STATE Y1 NAME) (:TOP 1 AREA Y1)))
```

how large is the largest city in alaska

```
((X POPULATION) CITY (:TOP 1 POPULATION X) (:EXISTS (Y1) (STATE Y1)
(= X STATE Y1 NAME) (= Y1 NAME "Alaska")))
```

how long is rio grande

```
((X LENGTH) RIVER (= X NAME "Rio Grande"))
```

how long is the longest river in california

```
((X LENGTH) RIVER (:TOP 1 LENGTH X) (:EXISTS (Y1 Y2) (RIVERFLOWSTHROUGH Y1)
(STATE Y2) (= X NAME Y1 RIVER) (= Y1 STATE Y2 NAME) (= Y2 NAME "California")))
```

how long is the longest river in the usa

```
((X LENGTH) RIVER (:TOP 1 LENGTH X))
```

what is the most populous city

```
 (X CITY (:TOP 1 POPULATION X))
```

how many big cities are in pennsylvania

```
((:COUNT X) CITY (> X POPULATION 200000) (:EXISTS (Y1) (STATE Y1)
(= X STATE Y1 NAME) (= Y1 NAME "Pennsylvania")))
```

how many cities are in louisiana

```
((:COUNT X) CITY (:EXISTS (Y1) (STATE Y1) (= X STATE Y1 NAME)
(= Y1 NAME "Louisiana")))
```

how many cities named austin are there in the usa

```
((:COUNT X) CITY (= X NAME "Austin"))
```

how many citizens does the biggest city have in the usa

```
((X POPULATION) CITY (:TOP 1 POPULATION X))
```

how many colorado rivers are there

```
((:COUNT X) RIVER (= X NAME "Colorado"))
```

how many inhabitants does montgomery have

```
((X POPULATION) CITY (= X NAME "Montgomery"))
```

how many people are in the state of nevada

```
((X POPULATION) STATE (= X NAME "Nevada"))
```

how many people live in spokane washington

```
((X POPULATION) CITY (= X NAME "Spokane") (= X STATE "Washington"))
```

how many people live in the biggest city in new york state

```
((X POPULATION) CITY (:TOP 1 POPULATION X) (:EXISTS (Y1) (STATE Y1)
(= X STATE Y1 NAME) (= Y1 NAME "New York")))
```

how many people live in the smallest state bordering wyoming

```
((X POPULATION) STATE (:BOTTOM 1 AREA X) (:EXISTS (Y1 Y2) (BORDER Y1) (STATE Y2)
(= X NAME Y1 STATE1) (= Y1 STATE2 Y2 NAME) (= Y2 NAME "Wyoming")))
```

how many people live in the state with the largest population density

```
((X POPULATION) STATE (:TOP 1 DENSITY X))
```

how many people live in the united states

```
((X POPULATION) STATE)
```

how many people live in washington

```
((X POPULATION) STATE (= X NAME "Washington"))
```

how many people live in washington dc

```
((X POPULATION) STATE (= X NAME "District of Columbia"))
```

how many rivers are found in colorado

```
((:COUNT X) RIVER (:EXISTS (Y1 Y2) (RIVERFLOWSTHROUGH Y1) (STATE Y2)
(= X NAME Y1 RIVER) (= Y1 STATE Y2 NAME) (= Y2 NAME "Colorado")))
```

how many rivers are in the state with the highest point

```
((:COUNT X) RIVER (:EXISTS (Y1 Y2) (RIVERFLOWSTHROUGH Y1) (STATE Y2)
(= X NAME Y1 RIVER) (= Y1 STATE Y2 NAME) (:EXISTS (Y3) (HIGH Y3)
(= Y2 NAME Y3 STATE) (:TOP 1 ELEVATION Y3))))
```

how many rivers are in the state with the largest population

```
((:COUNT X) RIVER (:EXISTS (Y1 Y2) (RIVERFLOWSTHROUGH Y1) (STATE Y2)
(= X NAME Y1 RIVER) (= Y1 STATE Y2 NAME) (:TOP 1 POPULATION Y2)))
```

how many square kilometers in the us

```
((:SUM (X AREA)) STATE)
```

how many states are in the united states

```
((:COUNT X) STATE)
```

how many states are next to major rivers

```
((:COUNT X) STATE (:EXISTS (Y1 Y2) (RIVERFLOWSTHROUGH Y1) (RIVER Y2)
(= X NAME Y1 STATE) (= Y1 RIVER Y2 NAME)))
```

how many states border colorado and border new mexico

```
((:COUNT X) STATE (:EXISTS (Y1 Y2) (BORDER Y1) (STATE Y2) (= X NAME Y1 STATE1)
(= Y1 STATE2 Y2 NAME) (= Y2 NAME "Colorado")) (:EXISTS (Y3 Y4) (BORDER Y3)
(STATE Y4) (= X NAME Y3 STATE1) (= Y3 STATE2 Y4 NAME)
(= Y4 NAME "New Mexico")))
```

how many states border on the state whose capital is boston

```
((:COUNT X) STATE (:EXISTS (Y1 Y2) (BORDER Y1) (STATE Y2) (= X NAME Y1 STATE1)
(= Y1 STATE2 Y2 NAME)) (= X CAPITAL "Boston"))
```

how many states border the largest state

```
((:COUNT X) STATE (:EXISTS (Y1 Y2) (BORDER Y1) (STATE Y2) (= X NAME Y1 STATE1)
(= Y1 STATE2 Y2 NAME) (:TOP 1 AREA Y2)))
```

how many states does iowa border

```
((:COUNT X) STATE (:EXISTS (Y1 Y2) (BORDER Y1) (STATE Y2) (= X NAME Y1 STATE1)
(= Y1 STATE2 Y2 NAME) (= Y2 NAME "Iowa")))
```

how many states does the missouri river run through

```
((:COUNT X) STATE (:EXISTS (Y1 Y2) (RIVERFLOWSTHROUGH Y1) (RIVER Y2)
(= X NAME Y1 STATE) (= Y1 RIVER Y2 NAME) (= Y2 NAME "Missouri")))
```

how many states have cites named austin

```
((:COUNT X) STATE (:EXISTS (Y1) (CITY Y1) (= X NAME Y1 STATE)
(= Y1 NAME "Austin")))
```

how many states in the us does the shortest river run through

```
((:COUNT X) STATE (:EXISTS (Y1 Y2) (RIVERFLOWSTHROUGH Y1) (RIVER Y2)
(= X NAME Y1 STATE) (= Y1 RIVER Y2 NAME) (:BOTTOM 1 LENGTH Y2)))
```

what are the capital cities of the states which border texas

```
((X CAPITAL) STATE (:EXISTS (Y1) (CITY Y1) (= X NAME Y1 STATE))
(:EXISTS (Y2 Y3) (BORDER Y2) (STATE Y3) (= X NAME Y2 STATE1)
(= Y2 STATE2 Y3 NAME) (= Y3 NAME "Texas")))
```

what are the largest cities in the states that border the largest state

```
(X CITY (:TOP 1 POPULATION X) (:EXISTS (Y1) (STATE Y1) (= X STATE Y1 NAME)
(:EXISTS (Y2 Y3) (BORDER Y2) (STATE Y3) (= Y1 NAME Y2 STATE1)
(= Y2 STATE2 Y3 NAME) (:TOP 1 AREA Y3))))
```

what are the populations of the major cities of texas

```
((X POPULATION) STATE (:EXISTS (Y1) (CITY Y1) (= X NAME Y1 STATE)
(> Y1 POPULATION 200000)) (= X NAME "Texas"))
```

what capital is the largest in the us

```
(X CITY (:EXISTS (Y1) (STATE Y1) (= X NAME Y1 CAPITAL)) (:TOP 1 POPULATION X))
```

what cities are located in pennsylvania

```
(X CITY (:EXISTS (Y1) (STATE Y1) (= X STATE Y1 NAME) (= Y1 NAME "Pennsylvania")))
```

what is the capital of states that have cities named durham

```
((X CAPITAL) STATE (:EXISTS (Y1) (CITY Y1) (= X NAME Y1 STATE) (= Y1 NAME "Durham")))
```

what is the capital of the state that borders the state that borders texas

```
((X CAPITAL) STATE (:EXISTS (Y1 Y2) (BORDER Y1) (STATE Y2) (= X NAME Y1 STATE1)
(= Y1 STATE2 Y2 NAME)) (:EXISTS (Y3 Y4) (BORDER Y3) (STATE Y4)
(= X NAME Y3 STATE1) (= Y3 STATE2 Y4 NAME) (= Y4 NAME "Texas")))
```

what is the largest state that borders the state with the lowest point in the usa

```
(X STATE (:TOP 1 AREA X) (:EXISTS (Y1 Y2) (BORDER Y1) (STATE Y2)
(= X NAME Y1 STATE1) (= Y1 STATE2 Y2 NAME)) (:EXISTS (Y3) (LOW Y3)
(= X NAME Y3 STATE)))
```

what is the least populous state

```
(X STATE (:BOTTOM 1 POPULATION X))
```

how many rivers do not traverse the state with the capital albany

```
((:COUNT X) RIVER (:NOT-EXISTS (Y1 Y2) (RIVERFLOWSTHROUGH Y1) (STATE Y2)
(= X NAME Y1 RIVER) (= Y1 STATE Y2 NAME) (= Y2 CAPITAL "Albany")))
```

what states border states that border states that border florida

```
(X STATE (:EXISTS (Y1 Y2) (BORDER Y1) (STATE Y2) (= X NAME Y1 STATE1)
(= Y1 STATE2 Y2 NAME)) (:EXISTS (Y3 Y4) (BORDER Y3) (STATE Y4)
(= X NAME Y3 STATE1) (= Y3 STATE2 Y4 NAME) (:EXISTS (Y5 Y6) (BORDER Y5)
(STATE Y6) (= Y4 NAME Y5 STATE1) (= Y5 STATE2 Y6 NAME)
(= Y6 NAME "Florida"))))
```

where is san jose

```
((X STATE) CITY (= X NAME "San jose"))
```

what is the elevation of death valley

```
((X ELEVATION) LOW (= X POINT "Death Valley"))
```

what is the combined area of all 50 states

```
((:SUM (X AREA)) STATE)
```

what is the combined population of all 50 states

```
((:SUM (X POPULATION)) STATE)
```

which state has the sparsest population density

```
(X STATE (:BOTTOM 1 DENSITY X))
```

what is the total population of the states that border texas

```
((:SUM (X POPULATION)) STATE (:EXISTS (Y1 Y2) (BORDER Y1) (STATE Y2)
(= X NAME Y1 STATE1) (= Y1 STATE2 Y2 NAME) (= Y2 NAME "Texas")))
```