

From Speech to Knowledge

Verónica Dahl

Simon Fraser University,
Burnaby B.C. V5A 1S6, Canada,
veronica@cs.sfu.ca,
<http://www.cs.sfu.ca/people/Faculty/Dahl>

Abstract. In human communication, assumptions play a central role. Linguists and logicians have uncovered their many facets. Much of AI work is also concerned with the study of assumptions in one way or another. Work on intuitionistic and linear logic has provided formally characterized embodiments of assumptions which have been influential on logic programming (e.g. [9,22,14]).

In this article we examine some uses of assumptive logic programming for speech-driven database creation and consultation, for speech driven robot control, and for web access through language.

This type of research can help relieve health problems related to the present typing/screen model of computer use. It can also partially address the need to integrate voice recognition, voice synthesis, and AI, along the route towards making computers into true extensions of our human abilities- extensions that adapt to our biology, rather than requiring our bodies to adapt.

1 Introduction

More than twenty years have elapsed since the first efforts towards declarative programming catapulted computing sciences from the old number-crunching paradigm into a new era paradigm of inferential engines. An era in which we no longer measure efficiency in terms of calculations per second, but in terms of inferences per second- a fantastic qualitative leap.

Logic programming, at the heart of this revolution in Computing Sciences, has been responsible for many beautiful incarnations, particularly in Artificial Intelligence, of the idea of programming through logic. Notable among them, natural language processing applications have blossomed around the axis of parsing-as-deduction (an expression coined by Fernando Pereira), ever since Alain Colmerauer developed the first logic grammar formalism, *Metamorphosis Grammars* [10].

These applications mostly span either language-to-language translation (e.g. French to English), with some meaning representation formalism mediating between the source language and the target language, or language-to-query translation, e.g. for using human languages as database front ends.

The latter kind of applications exploit a natural closeness between the logic programming style of queries and human language questions. Consider for instance representing the query "Find the names of employees who work for First Bank Corporation" as a Datalog or as a Prolog query:

```
query(X):- works(X,'First Bank Corporation').
```

versus its SQL equivalent

```
select employee_name
from works
where company-name= "First Bank Corporation"
```

or its QUEL equivalent

```
range of t is works
retrieve (t.person-name)
where t.company-name= "First Bank Corporation"
```

Traditional database query languages are in fact closer to computer programs than to human language questions: notions irrelevant to the question itself need to be explicitly represented, such as the range of a tuple variable, or operations such as selecting. In contrast, logic programming based queries are almost readable by people with little background of either logic programming or database theory.

While written text has long been used for database consultation (e.g. [15]), its use for representing knowledge itself lags behind. Database updates through language have been studied (e.g.[17]), but database creation through language has not, to the best of our knowledge, been attempted yet.

This is partly because creating knowledge bases through human language presents more difficulties than thus consulting it or updating it, and partly because the availability of reasonably efficient speech analysis and synthesis software is relatively new. Typing in the human language sentences necessary to create a knowledge base is probably as time consuming, and perhaps more error-prone, as typing in the information itself, coded in one of the current knowledge representation formalisms. With a good speech analyzer, however, dictating natural language sentences that represent a given corpus of knowledge becomes a much more attractive task.

The field of speech analysis and recognition has in fact been slowly reaching maturity, to the point in which very effective speech software is now available at relatively low cost. For instance, products such as Microsoft speech agent or Dragon Co.'s Naturally Speaking software can recognize a person's speech modalities after about a half hour of training. That person can then dictate into

a microphone, including punctuation marks, as s/he would dictate to a secretary, and see the written form or his or her utterances appear on the screen, being placed into a text file or being used as commands to the computer. Speech editing facilities are of course available (e.g. for correcting mistakes made by the speaker or by the speech software).

Impressive as it is, speech software has not yet been properly put together with artificial intelligence. Virtual Personalities Inc.'s verbal robots (verbots) come about the closest, but they mimic Weizenbaum's Eliza style of "understanding", abounding in default responses such as "What does that suggest to you?", "I see", etc.

Yet AI technology is also mature enough for many of its applications to be profitably augmented with speech capabilities. Also, hardware technology is swiftly moving towards networks of wireless, portable, small personal computers that have little to envy our previous "big" computers in terms of power.

The pieces of the puzzle are laid down to now attempt more human like communication with computers- computers in a wide sense, including robots, virtual worlds, and the Internet. Industry is already identifying the need to integrate "voice recognition software so that the computer can listen to you, voice synthesis so it can talk back to you, and AI programs to guess what you really want" (Newsweek, March 1998: interview to Bill Gates).

In this article we examine some applications of logic programming that in our view, should be explored along the route towards making computers into true extensions of our human abilities- extensions that adapt to our biology, rather than requiring our bodies to adapt.

Our presentation style is intuitive rather than formal, since we assume little previous knowledge of logic grammars, language processing, or logic programming. Technical details can be found in the references given. Readers familiar with logic programming can skim through section 2, noticing only our notation conventions.

Section 2 describes our logic programming tools, and at the same time shows step-by-step the construction of a (simplistic but adequate for exemplifying purposes) first prototype grammar for gleaning knowledge from natural language sentences, based on assumptive logic programming. The resulting data bases may include general rules as well as facts. Section 3 proposes a more detailed approach to the creation of knowledge bases (this section partially overlaps with [16]), and introduces a new type of assumption reasoning for dealing with discourse.

Section 4 discusses other types of knowledge bases that can be driven through speech: concept-based retrieval, robot control, generating animations and controlling virtual worlds. Finally, we present our concluding remarks. A sample session from a more encompassing database creation prototype than the one developed in 2 is shown in Appendices I and II. Appendix III shows a sample interaction with a Spanish consultable virtual world.

2 A First Prototype for Creating and Consulting Knowledge Bases

2.1 Definite Clause Grammars

The Basic Formalism Imagine rewrite grammar rules that can include variables or functional symbols as arguments, so that rewriting involves unification. What you have is called metamorphosis grammars [10], or DCGs [8]. Through them, you can for instance declare a noun phrase to be constituted by a name, or by a quantifier, an adjective, and a noun (we take notational liberties for consistency throughout this paper. “:-” stands for “rewrite into”.):

```
noun_phrase:- name.
noun_phrase:- quant, adj, noun.
```

More usefully, we can augment the grammar symbols with arguments which automate the construction of meaning representation:

```
noun_phrase(X,true) :- name(X).
noun_phrase(X,(A,N)):- quant, adj(X,A), noun(X,N).
```

Variables are capitalized. The first rule, for instance, commands the rewrite of any symbol matching *noun_phrase(X,true)* into *name(X)* (where X has been matched in the same way). Variables names are, as in Prolog, local to the clause (rule) in which they appear (since they are implicitly universally quantified within it).

The second argument of “noun_phrase” constructs the noun phrase’s “meaning”. In the case of a complex noun phrase, this meaning is composed by the meanings of the adjective (A), and the noun (N) (both of which will involve X, as we shall next see). In the case of a simple noun phrase (a name), its meaning is “true”- a primitive Prolog predicate which is always satisfied.

Words are preceded by “#”, and “or” can be represented as “;” (thus the quantifier rule below shows six alternative quantifying words). Proper names must be written in lower case. We can make up rewrite rules for the remaining grammar symbols as well, e.g.:

```
name(rahel):- #rahel.
name(estha):- #estha.

quant:- #the; #a; #an; #some; #all; #every.

adj(X,wise(X)):- #wise.
adj(X,true).

noun(X,owl(X)):- #owl.
```

Notice that adjectives are optional- if not present, the trivial representation "true" is generated. Here are some noun phrases and their representations as obtained by the above grammar:

```
the wise owl      (wise(X),owl(X))
an owl            (true,owl(X))
rahel               rahel
```

Querying Logic Databases From representations such as the above we can, for instance, directly query a Prolog database about the subject domain (e.g. owls), and obtain the answers automatically.

For instance, to ask for a wise owl, we would write the representation of that noun phrase as a Prolog query:

```
?- wise(X),owl(X).
```

Prolog will respond with $X=ow\text{sa}$ with respect to a database in which $ow\text{sa}$ has been defined to be wise and to be an owl, e.g.:

```
wise(ow\sa).
owl(ow\sa).
```

Initializing Knowledge Bases We could also *create* a database of knowledge from such representations. Let us first add verb phrases, and compose sentences from noun phrases and verb phrases, e.g. through the rules:

```
verb(X,Y,saw(X,Y)):- #saw.
verb(X,Y,likes(X,Y)):- #likes.

verb_phrase(X,Head,VP):- verb(X,Y,Head),noun_phrase(Y,VP).

sentence((Head,NP,VP)):- noun_phrase(X,NP),verb_phrase(X,Head,VP).
```

Anonymous variables (i.e., variables which only appear once in a rule and thus do not need a specific name) are noted "_".

To analyze a string from a given start symbol, we simply query the primitive predicate `analyze(Symbol)`, and follow the prompts, e.g.:

```
?- analyze(sentence(S)).
```

```
Enter the string to be analyzed, followed by a return: estha saw
the wise owl
```

```
S=saw(estha,_x16074),wise(_x16074),owl(_x16074)
```

Notice that in the sentence's representation S we have all the elements needed to create a Prolog rule representing to a data base the knowledge that Estha saw the wise owl. We merely need to rearrange its components in Prolog rule fashion:

```
saw(estha,X):- wise(X),owl(X).
```

where ":-" now is read as "if": if X is wise and an owl, then estha saw it.

Assumption Grammars for Relating Long-distant Constituents Now suppose you want to add a relative clause to the complex noun phrase rule:

```
noun_phrase(X,(A,N,R)):- quant, adj(X,A), noun(X,N), #that,
    relative(X,R).
```

For noun phrases in which it is the subject that is missing (to be identified with the relative's antecedent), the relative clause reduces to a verb phrase, so we could simply write instead:

```
noun_phrase(X,(A,N,H,VP)):- quant, adj(X,A), noun(X,N), #that,
    verb_phrase(X,H,VP).
```

This rule makes X (the representation of the relative's antecedent) the subject of the relative clause's verb phrase. We can now test, for instance, the sentence:

```
estha saw the owl that likes rahel
```

from which we obtain:

```
X=saw(estha,_x19101),true,owl(_x19101),likes(_x19101,rahel),true
```

However, extrapolating this technique to relatives from which another noun phrase than the subject is missing (e.g., "the owl that rahel saw", or "the owl that rahel gave a scolding to") would necessitate passing the antecedent X , which needs to be identified with the missing noun phrase, all the way to the place where the noun phrase is missing. This is inconvenient because it imposes the addition of X into symbols that have no direct business with it, since they merely act as transfer entities for X .

It would be convenient to have a way of hypothesizing a potential relative's antecedent as such in a more global fashion, and then using it wherever it is required (i.e., where a noun phrase is expected and cannot be found).

This is exactly what we can do with assumption grammars, in which a hypothesis- a (linear) assumption ¹- is noted as a Prolog predicate preceded by

¹ Linear assumptions [9] can be consumed only once, whereas intuitionistic assumptions can be consumed any number of times.

”+”, and its use (consumption) is noted ”-”. An assumption is available during the continuation of the present computation, and vanishes upon backtracking.

So, for instance, we can define a relative as a sentence with a missing noun phrase preceded by a relative pronoun, through the grammar rule:

```
relative(X,R):- #that, +missing_np(X), sent(R).
```

The appropriate antecedent (i.e. the variable to be bound with X- remember that variables are implicitly quantified within each rule, so variables of the same name in different rules are a priori unrelated) can then be transmitted to the relative clause through the noun phrase rule:

```
noun_phrase(X,(A,N,R)):- quant, adj(X,A), noun(X,N),
    relative(X,R).
```

and the missing noun phrase is associated with this antecedent through consumption at the point in which it is shown missing (i.e., as the last noun phrase rule, to be tried after all others have failed):

```
noun_phrase(X,true) :- -missing_np(X).
```

Finally, we allow for noun phrases with no relative clauses:

```
relative(_,true).
```

A sentence such as ”the owl that estha saw likes rahel” now yields the representation:

```
S=likes(_x19102,rahel),(true,owl(_x19102),saw(estha,_x19102),true,
true),true
```

The spurious ”true” predicates disappear upon writing these results into a file.

Again, from such a representation we can then construct the Prolog database definition:

```
likes(A,rahel):- owl(A),saw(estha,A).
```

3 A More Detailed Approach to Knowledge Base Creation

Going beyond our first prototype for knowledge base creation and consultation involves decisions such as what language subset is going to be covered, how exactly is it going to be represented, etc.

In this section we propose one possible approach, and we argue that a simple higher level extension of LP, timeless assumptions, greatly facilitates the task of going from discourse to databases. More research is needed to provide a thorough proof of concept.

3.1 Underlying Conventions

Let us examine what kinds of English descriptions we shall admit, and what kinds of representations for database relations should be extracted automatically from them.

The Natural Language Subset Our subset of language consists of sentences in the active voice, where relation words (nouns, verbs and adjectives) correspond to database predicates, and their complements to arguments of these predicates. For instance, "John reads Ivanhoe to Mary" generates the Prolog assertion `reads(john,ivanhoe,mary)`.

Vocabulary reasonably common to all databases belongs to the static part of our system (e.g. articles, prepositions, common verbs such as "to be", etc.), and vocabulary specific to each application (e.g. proper names, nouns, verbs proper of the application, etc.) is entered at creation time, also through spoken language, with menu-driven help from the system.

In the interest of ease of prototyping, we shall first only use universal quantifiers (as in the clausal form of logic), whether explicit or implicit, and only the restrictive type of relative clauses (i.e., those of the form "(All) ... that...", where the properties described by the relative clause restrict the range of the variable introduced by the quantifier "all". Such relatives, as we have seen, translate into additional predicates in the clause's body, as in:

`People like cats that purr.`

for which a possible translation is

`like(P,C):- person(P), cat(C), purrs(C).`

It is easy to include alternative lexical definitions in the language processing module of our system, so that all words for a given concept, say "people" and "persons", translate into a single database relation name (say, "people"). Thus we can allow the flexibility of synonyms together with the programming convenience of having only one constant for each individual- no need for equality axioms and their related processing overhead.

Semantic Types It is useful to have information about semantic types. For instance, we may have informed the database that people like animals, and may not have explicitly said that people like cats that purr. But if we knew that cats are animals, we could easily infer that people like cats and that they like cats that purr, given the appropriate query.

If we wanted to reject semantically anomalous input, much of this could be done through simply checking type compatibility between, say, the expected argument of a predicate, and its actual argument. For instance, if "imagine" requires a human subject, and appears in a sentence with a non-human subject, the user can be alerted of the anomaly and appropriate action can be taken. Variables can be typed when introduced by a natural language quantifier, and constants can be typed in the lexicon, when defining proper names. The notation used to represent a semantic type can reflect the relevant set inclusion relationships in the type hierarchy.

Much effort has been devoted to efficient encodings of type hierarchies. A recent survey and breakthrough results for logic programming are presented in [12]. These results can be transferred to a menu-driven module of our system which will question the user about a topmost class in the database's domain, its subsets, etc., and accordingly construct the class hierarchy encoded in such a way that set inclusion relationships are decidable with little more than unification.

Set-orientation Our logic programming representation of a database should be set-oriented. That is, rather than having n clauses for representing a unary property that n individuals satisfy, we can have one clause for the entire set (e.g. `biblical([adam,eve])`). Database primitives for handling relations on sets are provided.

Intensionally as well as extensionally represented sets should be allowed, in two ways. First, having semantic types associated to each variable and constant makes it possible to give intensional replies rather than calculating an extensionally represented set. For instance, we can reply "all birds" to the question of which birds fly if the database expresses that all entities of type bird fly, rather than checking the property of flying on each individual bird (assuming we even were to list all birds individually). We can even account for exceptions, e.g. by replying: "All birds except penguins". The user can always choose to request an extensional reply if the first, intensional answer is not enough.

Secondly, we have found it useful in some cases to represent sets of objects as a type and an associated cardinality (e.g., in a query such as "How many cars are in stock?", we do not really want to have a name for each of the cars, being only interested in the numbers of (indistinguishable) entities of type car.

Events In order to correctly relate information about the same event given in different sentences of a discourse, we translate n -ary relations into sets of binary ones linked by an event number. For instance, if we input the sentence:

"John gave Rover to Mary."

instead of generating the ternary relation:

```
gave(john,rover,mary)
```

the analyzer can generate a representation that keeps track of the event, or information number within the discourse, through the three assertions²:

```
gave(1,who,john).
gave(1,what,rover).
gave(1,to,mary).
```

This method roughly corresponds to event logics, dating as far back as 1979 [11]. It provides us with a simple way to be flexible as to the number of arguments in a relation. Notice that the event number is not necessarily the same as the sentence number. If our next sentence is

```
"This happened in 1998"
```

then the system needs to recognize the event described in the previous sentence as the one referred to by "this", and add the following clause to the database:

```
gave(1,when,1998).
```

In order for our natural language analyzer to be able to effect such translations, it needs information about semantic types of each argument of a relation. This can be given in the lexicon, specifically in the definitions of nouns, verbs and adjectives, since these words typically correspond to predicates in a database. Lexical definitions for these can either be input once and for all for a given domain, or elicited from the user in menu-driven fashion. The latter option has the attraction of providing extensibility to the system, which can then admit new words into its vocabulary.

3.2 From Sentences to Discourse- Timeless Assumptions

While it is certain that we must restrict the range of natural language accepted by our speech applications, we should seek naturalness and allow not only isolated sentences or queries, but the flexibility of discourse and paraphrase as well.

Determining which entities co-specify (i.e., refer to the same individual) is one of the most important problems in understanding discourse. For instance, if a user enters the information: "John Smith works in the toy department. It does not meet fire safety standards. His country's laws are not enough to protect him", the system translating this input into a knowledge base needs to identify

² In fact, semantic types can also be generated at language analysis stage, but we overlook them here for simplicity.

"it" with the toy department mentioned, and "his" and "him" as relating to John Smith.

A thorough treatment of co-specification is a complex and widely studied issue, involving not only syntactic but also semantic, pragmatic and contextual notions (consider for instance "Don't step on it", referring to a snake just seen, or "John kicked Sam on Monday and it hurt", where "it" refers to an abstract proposition rather than a concrete individual).

A discussion of possible treatments of co-specification is beyond the scope of this article, but we shall introduce a methodology -timeless assumptions- for facilitating the detection of co-specification. This basic technique can be adapted to incorporate different criteria for co-specification determination.

Timeless assumptions allow us to consume assumptions after they are made (as before), but also, when a program requires them to be consumed at a point in which they have not yet been made, they will be assumed to be "waiting" to be consumed, until they are actually made (the Prolog cut, noted "!", is a control predicate preventing the remaining clauses from being considered upon backtrack):

```
% Assumption:
```

```
% the assumption being made was expected by a previous consumption
=X:- wait(X), !.
% if there is no previous expectation of X, assume it linearly
=X:- +X.
```

```
% Consumption:
```

```
% uses an assumption, and deletes it if linear
=-X:- -X, !.
% if the assumption has not yet been made,
% adds its expectation as an assumption
=-X:- +wait(X).
```

With these definitions it no longer matters whether an assumption is first made and then consumed, or first "consumed" (i.e., put in a waiting list until when it is actually made) and then made.

We can use timeless assumptions for instance to build a logic programmed database directly from:

```
The blue car stopped. Two people came out of it.
```

```
or:
```

```
Two people came out of it after the blue car stopped.
```

To cover both cases, we could timelessly assume an object of description D for each noun phrase represented by a variable X and with features F that appears in the discourse:

$=\text{object}(X, F, D)$.

When encountering a pronoun with matching features F' , in a sentence that further describes the referred object as D' , replace the timeless assumption by one which adds D' to what is known of X . With some notational license:

$=\text{object}(X, F, D \& D')$.

Once the discourse ends, we can firm the assumption into a regular database clause.

Of course, there will be cases of unresolvable ambiguity, in which even the most informed co-specification resolution criteria will fail (as it will in human communication). And introducing the complexities of discourse into a system that depends on it fully for gathering information its is no small a challenge. But the availability of a time-independent assumption mechanism can greatly help. Timeless assumptions have also proved useful for treating other linguistic phenomena such as coordination [14]

4 Further Applications Amenable to Speech

As we hope to have shown in previous sections, the database field is one of the main candidates for speech interactions through logic, being sufficiently studied, and given that it also includes logic-based incarnations such as Datalog.

Other applications, while less studied, are emerging as good candidates too. For instance, the new need for processing massive amounts of web data which are mostly expressed in natural language opens up several interesting possibilities. In this section we briefly discuss current research which sets the stage for speech interactions to be also incorporated into them.

4.1 Speech Driven Robot Control

Controlling Mobile Mini-robots Using speech to control robots is also an interesting application of natural language processing through logic: since robots' worlds are fairly restricted, commands tend to be relatively simple and fairly devoid of the ambiguity that plagues other natural language applications.

Preliminary work has been done jointly with Marie-Claude Thomas and Andrew Fall to control a simple world of mini robots through natural language commands [4,5]. These robots move in an enclosed room, avoiding obstacles through sensors. They know about time units, they can detect features such as room temperature and humidity, move to sources of light, avoid obstacles, and recharge themselves through sources of light positioned within the room.

Natural language commands translate into a specially developed formal logic system. This high degree of formalization is currently believed necessary in

robotics, and allows us in particular great economy: the syntax of our logic system is the representation language for commands, and its semantics is the high level execution specification, resulting in several calls to Prolog and C routines.

Natural language commands are of the form exemplified below:

- Go to the nearest source of light in ten minutes.
- Go to point P taking care that the temperature remains between X and Y degrees.
- Let robot A pass.
- Give X to robot B.
- Stop as soon as the humidity exceeds H.

Imperative sentences with an implicit subject, which are rare in other applications, are common here. Complements are typically slots to be filled in mostly by constants.

Our approach combines the two current main approaches in robotics: a high-level deductive engine generates an overall plan, while dynamically consulting low level, distributed robotics programs which interface dynamically with the robot's actions and related information.

Obviously, adding a speech component to such systems would enhance them greatly. Mobile computers and wireless networks, as well as already available speech software which can be adapted, will help towards the goal of transporting these specialized robots to wherever they are needed, while communicating with them in more human terms.

Controlling Virtual, Visual World Robots Another interesting family of Internet applications of speech is that of robots that operate in virtual, visual worlds. For instance, Internet-based VRML animations have been generated through English-controlled partial order planners [3]. The next step is for such systems to accept speech rather than written input.

This research, done in collaboration with Andrea Schiel and Paul Tarau, presents a proof-of-concept Internet-based agent programmed in BinProlog which receives a natural language description of a robot's goal in a blocks world, and generates the VRML animation of a sequence of actions by which the robot achieves the stated goal. It uses a partial order planner.

The interaction storyboard is as follows: The user types in an NL request via an HTML form, stating a desirable final state to be achieved. This is sent over the Internet to a BinProlog based CGI script working as a client connected to the multi-user World-State server. The NL statement is used to generate an expression of the goal as a conjunction of post-conditions to be achieved. The goal is passed to the planner module, also part of the CGI script. The plan materializes as a VRML animation, ending in a visual representation of the final state to be sent back as the result of the CGI script, as well as in an update of the WorldState database.

Our analyzer, based on Assumption Grammars, can deal with multisentential input and with anaphora (e.g. relating pronouns to antecedents in previous sentences). Pronoun resolution involves intuitionistic rather than linear assumptions (since an antecedent can be referred to by a pronoun more than once).

The results we prototyped for the blocks world can be transposed to other domains, by using domain-tailored planners and adapting the NL grammar to those specific domains. A more interesting extension is the development of a single agent to produce VRML animations from NL goals with respect to different applications of planning. This might be achieved by isolating the domain-specific knowledge into an application-oriented ontology; adapting a partial order planner to consult this ontology modularly; and likewise having the grammar examine the hierarchy of concepts in order to make sense of domain-oriented words.

4.2 Web Access Through Language

The intersection between logic programming and the Internet is a very new but rapidly growing field. Recent logic-based web applications have been presented [18,20], and a special issue of the Journal of Logic Programming on this subject is currently under preparation.

Among the exciting new applications that these interactions are making possible, concept-based retrieval and virtual worlds stand out as particularly promising for long-distance interactions, distributed work, and interactive teaching through the web [6]. Endowing these budding applications with access through language, and in particular speech, would greatly enhance their capabilities. In particular, multilingual access to virtual worlds over the Internet would help remove geographic and language barriers to cooperation. Preliminary work in this direction is [2].

Speech Driven Virtual World Communication This research implements a Bin-Prolog based virtual world running under Netscape and Explorer, called LogiMOO, with a multilingual and extensible natural language front end [13], which will also be endowed with a speech as well as a visual component. It allows (written at present) communication between distant users in real time, hiding the complexities of the distributed communication model through the usual metaphors: places (starting from a default lobby), ports, ability to *move* or *teleport* from one place to another, a *wizard* resident on the server, *ownership* of objects, the ability to *transfer* ownership and a built-in notifier agent watching for messages as a background thread.

LogiMOO makes use of linear and intuitionistic assumption techniques, and is based on a set of embeddable logic programming components which interoperate with standard Web tools. Immediate evaluation of world knowledge by the parser yields representations which minimize the unknowns, allowing us to deal with advanced natural language constructs like anaphora and relativization efficiently. We take advantage of the simplicity of our controlled language to provide as well an easy adaptation to other natural languages than English, with English-like representations as a universal interlingua.

The peculiar features of the world to be consulted- a virtual world- induced novel parsing features which are interesting in themselves: flexible handling of dynamic knowledge, immediate evaluation of noun phrase representations, allowing us to be economic with representation itself, inference of some basic syntactic categories from the context, a treatment of nouns as proper nouns, easy extensibility within the same language as well as into other natural languages.

Speech Driven Concept Based Retrieval As anyone knows who has tried to query the Internet through the existing search engines, there is a glaring need for intelligent access to that fantastic but frustratingly mechanical repository of world knowledge that the Internet has become. From this perspective alone, logic should play a major role, given that deduction is obviously needed to make enough sense of a query as to minimize noise (the number of irrelevant documents obtained for a given query to a search engine) and silence (the number of failures to find any relevant documents that do exist).

For instance, within a forestry domain of interest, we can use a taxonomy of forestry-related concepts, which allows the search engine to specialize or generalize given concepts (e.g. going from "water" to "lakes", or vice versa), and to use the contextual information provided by forestry domains in order to avoid nonsensical answers. Search engines that base their search on keywords rather than semantics, in contrast, have been known to respond for instance to a query for documents related to "clear cuts near water" with "Complete poetical works from William Wordsworth", among a list of other equally wrong associations.

Concept-based search engines are starting to appear, but to our knowledge there are none yet that go much beyond a shallow use of keywords and concept classifications. Full meaning extraction and comparison, however, can only be done once the subtleties of natural language semantics are taken into account. A challenging task, but again, one for which logic programming is particularly suited. Assumption Grammars have been proposed, in conjunction with other techniques from Artificial Intelligence and Databases (concept hierarchies, multi-layered databases and intelligent agents) for intelligently searching information pertaining to a specific industry on the web [7].

5 Concluding Remarks

Computers have become so ubiquitous, that it is high time to develop alternative computer work modes than the present typing/screen based model. We are the first generation with exponents that have spent twenty or thirty years working in front of a computer terminal, and the ill effects are all too visible around us: tendonitis; eye, neck and back strain; Carpal Tunnel syndrome...

Speech-driven knowledge base creation and consultation, just as speech driven robot control or programming, and web access through language, could bring relief from such problems. They can also partially address the present need to integrate voice recognition software, voice synthesis, and AI programs.

In this article we have proposed some emerging approaches towards such ends. However, putting all the pieces of the puzzle together will require careful crafting. Within the logic-based database field, some recent developments could prove most valuable towards this objective, like the uses of Inductive Logic Programming to automate the construction of natural language interfaces for database queries [19].

Logic grammar formalisms have been developed moreover with linguistic ease of expression in mind. In helping linguists write executable grammars in terms that are not too removed from their own, we might by the way be able to tap on linguistic expertise that might be most valuable for our own language processing applications.

The availability of powerful while mobile computers and applications (e.g. [21]) also adds unprecedented potential to speech interfacing software, for instance for business people or academics who often travel, and who could therefore make the most of such results if they were portable. Finally, our experience with the circumscribed domain of database creation could prove useful as a first step towards an even more daring application, that of programming through natural language.

Efforts are under way for a EU/NorthAmerica Compulog Network of Centres of Excellence to launch a cooperation around NL projects using logic programming, presently coordinated jointly by Dr. Pearce and the author. With the present article we hope to stimulate further interaction along these lines, both in a geographic sense and across areas.

6 Appendix I: Sample Database Creation and Consultation Session

In the following sample creation session, user input is prompted by ">"

```
?- go.
```

```
Enter information for the database one sentence at a time
When you are done, say stop
```

```
> anne is a person
```

```
I am adding the clause:
  person(anne)
```

```
> garfield is a cat
```

I am adding the clause:
cat(garfield)

> garfield purrs

I am adding the clause:
purrs(garfield)

> earth is the mother of anne

I am adding the clause:
mother_of(earth,anne)

> anne is canadian

I am adding the clause: canadian(anne)

> people like cats that purr

I am adding the clause:
likes(_x38299,_x38336):-
 (person(_x38299),true),cat(_x38336),purrs(_x38336),true

> the mother of anne likes cats that purr

I am adding the clause:
likes(_x39082,_x39147):-
 (mother_of(_x39082,anne),true,true),cat(_x39147),
 purrs(_x39147),true

> eve and adam like paradise

I am adding the clause:
likes([eve,adam],paradise)

> peter is angry with rover

I am adding the clause:

```
angry_with(peter,rover)
```

```
> the mother of anne gives rover to peter
```

```
I am adding the clause:
```

```
gives(_x37230,rover,peter):-  
  (mother_of(_x37230,anne),true,true),true
```

```
> anne is intelligent
```

```
I am adding the clause:
```

```
intelligent(anne)
```

```
> a person that likes cats that purr is intelligent
```

```
I am adding the clause:
```

```
intelligent(_x38073):-  
  person(_x38073),likes(_x38073,_x38193),cat(_x38193),  
  purrs(_x38193),true
```

```
> stop
```

```
You can consult the database by typing "answer."
```

```
yes
```

```
?- answer.
```

```
> who is intelligent
```

```
Representation of the query:
```

```
question(_x2599,(intelligent(_x2599),true))
```

```
Answer: anne
```

```
> who likes cats that purr
```

```
Representation of the query:
```

```
question(_x2889,(likes(_x2889,_x2948),cat(_x2948),purrs(_x2948),  
true))
```

```
Answer: anne
```

```
Answer: earth
```

> who is angry with rover

Representation of the query:

```
question(_x3169,(angry_with(_x3169,rover),true))
```

Answer: peter

> who gives rover to peter

Representation of the query:

```
question(_x3459,(gives(_x3459,rover,peter),true))
```

Answer: earth

> earth gives rover to who

Representation of the query:

```
question(_x3749,(gives(earth,rover,_x3749),true))
```

Answer: peter

> earth gives who to peter

Representation of the query:

```
question(_x4039,(gives(earth,_x4039,peter),true))
```

Answer: rover

> who likes paradise

Representation of the query:

```
question(_x4257,(likes(_x4257,paradise),true))
```

Answer: [eve,adam]

> who likes garfield

Representation of the query:

```
question(_x4643,(likes(_x4643,garfield),true))
```

Answer: anne

Answer: earth

```
> stop  
Goodbye
```

N.B. The mother of anne (earth) is not identified as one of the answers on who is intelligent because She has not been declared to be a person.

7 Appendix II- The Database Created Through the Session in Appendix I

```
% dyn_compiled: person/1:  
person(anne).
```

```
% dyn_compiled: cat/1:  
cat(garfield).
```

```
% dyn_compiled: purrs/1:  
purrs(garfield).
```

```
% dyn_compiled: mother_of/2:  
mother_of(earth,anne).
```

```
% dyn_compiled: canadian/1:  
canadian(anne).
```

```
% likes/2:  
likes(A,B) :-  
    person(A),  
    true,  
    cat(B),  
    purrs(B).  
likes(A,B) :-  
    mother_of(A,anne),  
    true,  
    true,
```

```
        cat(B),
        purrs(B).
likes([eve,adam],paradise).
```

```
% angry_with/2:
angry_with(peter,rover).
```

```
% gives/3:
gives(A,rover,peter) :-
    mother_of(A,anne),
    true,
    true.
```

```
% intelligent/1:
intelligent(anne).
intelligent(A) :-
    person(A),
    likes(A,B),
    cat(B),
    purrs(B).
```

8 Appendix III- Sample Interaction with a Spanish Consultable Virtual World Through LogiMOO

This Spanish session illustrates among other things flexibility re. nouns., within a controlled English coverage. Since the virtual world admits the crafting of new objects, and we do not want to restrict the user to crafting only those objects whose vocabulary is known, the system assumes from context that an unknown word is a noun, and proceeds to translate it as itself. Thus we get a definite "Spanglish" flavour in the internal representations obtained, but for the purposes of reacting to the Spanish commands, this does not matter much. Ultimately, a bilingual dictionary consultation on line should generate the translation, perhaps in consultation with the user as to different possible translations. Other shortcuts are also taken (e.g. guest room translates into the single identifier `guest_room`, clitic pronouns are in an unnatural position, etc).

The Spanish interactions shown translate into:

I am Paul. Dig a guest room. Go there. Dig a kitchen. Go to the hall. Look. I am the wizard. Where am I? Dig the bedroom. Go there. Dig a kitchen, open a port to the south of the kitchen, go there, open a port to the north of the bedroom. Go there. Build a portrait. Give it to the wizard. Look. I am Diana. Build a car. Where is the car? Build a Gnu. Who has it? Where is the Gnu? Where am I? Give the wizard the Gnu that I built. Who has it?

```
test_data("Yo soy Paul.").
test_data("Cave una habitacion_huespedes.
  Vaya alli. Cave una cocina.").

test_data("Vaya al vestibulo. Mire.").

test_data("Yo soy el brujo.
  Donde estoy yo?").

test_data("Cave el dormitorio. Vaya alli.
  Cave una cocina, abra una puerta al sur de
  la cocina, vaya alli, abra una puerta
  al norte del dormitorio. Vaya alli.
  Construya un cuadro. Dese lo al brujo.
  Mire.").

test_data("Yo soy Diana. Construya un
  automovil. Donde esta el automovil?").

test_data("Construya un Gnu. Quien tiene
  lo? Donde esta el Gnu? Donde estoy yo?").

test_data("Dele al brujo el Gnu que yo
  construi. Quien tiene lo?").

/* TRACE:

==BEGIN COMMAND RESULTS==
TEST: Yo soy Paul.
WORDS: [yo,soy,paul,.]
SENTENCES: [yo,soy,paul]

==BEGIN COMMAND RESULTS==
login as: paul with password: none
your home is at http://199.60.3.56/~veronica

SUCCEEDING(iam(paul))

==END COMMAND RESULTS==

TEST: Cave una habitacion_huespedes.
  Vaya alli. Cave una cocina.
```

```
WORDS: [cave,una,habitacion_huespedes,.,
vaya,alli,.,cave,una,cocina,.]
SENTENCES: [cave,una,habitacion_huespedes]
[vaya,alli] [cave,una,cocina]

==BEGIN COMMAND RESULTS==
SUCCEEDING(dig(habitacion_huespedes))
you are in the habitacion_huespedes
SUCCEEDING(go(habitacion_huespedes))
SUCCEEDING(dig(cocina))

==END COMMAND RESULTS==

TEST: Vaya al vestibulo. Mire.
WORDS: [vaya,al,vestibulo,.,mire,.]
SENTENCES: [vaya,al,vestibulo] [mire]

==BEGIN COMMAND RESULTS==
you are in the lobby
SUCCEEDING(go(lobby))
user(veronica,none,'http://...').
user(paul,none,'http://...').
login(paul).
online(veronica).
online(paul).
place(lobby).
place(habitacion_huespedes).
place(cocina).
contains(lobby,veronica).
contains(lobby,paul).
SUCCEEDING(look)

==END COMMAND RESULTS==

TEST: Yo soy el brujo. Donde estoy yo?
WORDS: [yo,soy,el,brujo,.,donde,estoy,yo,?]
SENTENCES: [yo,soy,el,brujo]
[donde,estoy,yo]

==BEGIN COMMAND RESULTS==
login as: wizard with password: none
your home is at http://199.60.3.56/~veronica

SUCCEEDING(iam(wizard))
you are in the lobby
SUCCEEDING(whereami)

==END COMMAND RESULTS==

TEST: Cave el dormitorio. Vaya alli. Cave
```

una cocina, abra una puerta alsur de la cocina, vaya alli, abra una puerta alnorte del dormitorio. Vaya alli. Construya un cuadro. Dese lo al brujo. Mire.

WORDS: [cave,el,dormitorio,,vaya,alli,,cave,una,cocina,(,),abra,una,puerta,alsur,de,la,cocina,(,),vaya,alli,(,),abra,una,puerta,alnorte,del,dormitorio,,vaya,alli,,construya,un,cuadro,,dese,lo,al,brujo,,mire,.]

SENTENCES: [cave,el,dormitorio] [vaya,alli] [cave,una,cocina] [abra,una,puerta,alsur,de,la,cocina] [vaya,alli] [abra,una,puerta,alnorte,del,dormitorio] [vaya,alli] [construya,un,cuadro] [dese,lo,al,brujo] [mire]

==BEGIN COMMAND RESULTS==

SUCCEEDING(dig(room))
 you are in the room
 SUCCEEDING(go(room))
 SUCCEEDING(dig(cocina))
 SUCCEEDING(open_port(south,cocina))
 you are in the cocina
 SUCCEEDING(go(cocina))
 SUCCEEDING(open_port(north,room))
 you are in the room
 SUCCEEDING(go(room))
 SUCCEEDING(craft(cuadro))
 logimoo:<wizard># 'wizard:I give you cuadro'
 SUCCEEDING(give(wizard,cuadro))
 user(veronica,none,'http://...').
 user(paul,none,'http://...').
 user(wizard,none,'http://...').
 login(wizard).
 online(veronica).
 online(paul).
 online(wizard).
 place(lobby).
 place(habitacion_huespedes).
 place(cocina).
 place(room).
 contains(lobby,veronica).
 contains(lobby,paul).
 contains(room,wizard).
 contains(room,cuadro).
 port(room,south,cocina).
 port(cocina,north,room).
 has(wizard,cuadro).
 crafted(wizard,cuadro).

SUCCEEDING(look)

==END COMMAND RESULTS==

TEST: Yo soy Diana. Construya un automovil.

Donde esta el automovil?

WORDS: [yo,soy,diana,.,construya,un,
automovil,.,donde,esta,el,automovil,?]

SENTENCES: [yo,soy,diana]
[construya,un,automovil]
[donde,esta,el,automovil]

==BEGIN COMMAND RESULTS==

login as: diana with password: none
your home is at http://199.60.3.56/~veronica

SUCCEEDING(iam(diana))
SUCCEEDING(craft(automovil))
automovil is in lobby
SUCCEEDING(where(automovil))

==END COMMAND RESULTS==

TEST: Construya un Gnu. Quien tiene lo?

Donde esta el Gnu? Donde estoy yo?

WORDS: [construya,un,gnu,.,quien,tiene,
lo,?,donde,esta,el,gnu,?,donde,estoy,yo,?]

SENTENCES: [construya,un,gnu]
[quien,tiene,lo] [donde,esta,el,gnu]
[donde,estoy,yo]

==BEGIN COMMAND RESULTS==

SUCCEEDING(craft(gnu))
diana has gnu
SUCCEEDING(who(has,gnu))
gnu is in lobby
SUCCEEDING(where(gnu))
you are in the lobby
SUCCEEDING(whereami)

==END COMMAND RESULTS==

TEST: Dele al brujo el Gnu que yo construi.

Quien tiene lo?

WORDS: [dele,al,brujo,el,gnu,que,yo,
construi,.,quien,tiene,lo,?]

SENTENCES: [dele,al,brujo,el,gnu,que,yo,
construi] [quien,tiene,lo]

==BEGIN COMMAND RESULTS==

```

logimoo:<diana># 'wizard:I give you gnu'
SUCCEEDING(give(wizard,gnu))
wizard has gnu
SUCCEEDING(who(has,gnu))

==END COMMAND RESULTS==

SUCCEEDING(test)

==END COMMAND RESULTS==
*/

```

References

1. V. Dahl, P. Tarau, P. Accuosto, S. Rochefort, and M. Scurtescu. Assumption Grammars for Knowledge Based Systems. *Informatica* 22(4), pages 435–444, 1998.
2. V. Dahl, P. Tarau, S. Rochefort, and M. Scurtescu. A Spanish Interface to LogiMoo- towards multilingual virtual worlds. *Informatica*, volume 2, june 1999. [62](#)
3. A. Schiel, V. Dahl and P. Tarau. *Generating Internet Based VRML Animations through Natural Language Controlled Partial Order Planners* Technical Report, Simon Fraser University, 1998. [61](#)
4. V. Dahl, A. Fall and M.C. Thomas. Driving Robots through natural language. *Proceedings 1995 International Conference on Systems, Man and Cybernetics*, pages 1904–1908, july 1995. [60](#)
5. M. C. Thomas, V. Dahl and A. Fall. Logic Planning in robotics. *Proceedings 1995 International Conference on Systems, Man and Cybernetics*, pages 2951–2955, july 1995. [60](#)
6. S. Rochefort, V. Dahl and P. Tarau. A Virtual Environment for Collaborative Learning. *Proc. World Multiconference on Systemics, Cybernetics and Informatics (SCI'98) and 4th International Conference on Information Systems Analysis and Synthesis (ISAS'98)*, Orlando, Florida, 1998. [62](#)
7. O.R. Zaiane, A. Fall, S. Rochefort, V. Dahl and P. Tarau. On-Line Resource Discovery using Natural Language. *Proc. RIAO'97, Computer-Assisted Searching on the Internet*, pp. 336–355, McGill University, Montreal, June 1997. [63](#)
8. F.C.N. Pereira and D.H.D. Warren. Definite Clause Grammars for Language Analysis - A survey of the formalism and a Comparison with Transition Networks. *Artificial Intelligence*, vol. 13, pages 231–278, 1980. [52](#)
9. R. Pareschi and D. Miller. *Extending definite clause grammars with scoping constructs*, pages 373–389. Warren, David H. D. and Szeredi, P. (eds.) International Conference in Logic Programming, MIT Press, 1990. [49](#), [54](#)
10. A. Colmerauer. Metamorphosis Grammars. *Lecture Notes in Computer Science* 63, pages 133–189, Springer-Verlag, 1978. [49](#), [52](#)
11. R.A.K. Kowalski. *Logic for Problem Solving*. North-Holland, 1979. [58](#)
12. A. Fall. The foundations of taxonomic encoding. *Computational Intelligence*, 14(4):1–45, 1998. [57](#)
13. P. Tarau, K. De Boschere, V. Dahl, and S. Rochefort. LogiMOO: an Extensible Multi-User Virtual World with Natural Language Control *Journal of Logic Programming*, 38(3), pages 331–353, 1999. [62](#)

14. V. Dahl, P. Tarau and R. Li. Assumption Grammars for Processing Natural Language. *Proceedings International Conference on Logic Programming'97*, pages 256–270, 1997. 49, 60
15. V. Dahl. Logical Design of Deductive, Natural Language Consultable Data Bases. *Proc. V Int. Conf. on Very Large Databases, Rio de Janeiro*, pages 24–31, 1979. 50
16. V. Dahl. The logic of language. In K. Apt, V. Marek and D.S. Warren (eds.), *The Logic Programming Paradigm: A 25 year perspective*, pages 429–451, Springer-Verlag, 1999. 51
17. J. Davison. A Natural Language Interface for Performing Database Updates. *ICDE*, pages 69–76, 1984. 50
18. P. Tarau, K. De Bosschere and M. Hermenegildo (eds.). *Proc. of the 2nd International Workshop on Logic Programming Tools for Internet Applications, ICLP'97*, 1997. 62
19. J.M. Zelle and R. J. Mooney. Learning to Parse Database Queries using Inductive Logic Programming. *Proc. Thirteenth National Conference on Artificial Intelligence*, pages 1050–1055, 1996. 64
20. S.W. Loke. *Adding Logic Programming Behaviour to the World Wide Web* PhD Thesis, Univ. of Melbourne, Australia, 1998. 62
21. K.A. Bharat and L. Cardelli. Migratory applications. *Proc. 8th Annual ACM Symposium on User Interface Software and Technology*, 1995. 64
22. D.A. Miller and G Nadathur. Some uses of higher -order logic in computational linguistics. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, pages 247–255, 1986. 49