Knowledge Representation via Verbal Description Generalization: Alternative Programming in Sampletalk Language

Andrew Gleibman

Microspec Technologies
Star Building, Yokneam 20692, Israel
andrew@sampletalk.com

Abstract

In this paper, a concept of alternative programming is defined by analogy with alternative medicine. This is a method of programming where natural data examples (e.g., verbal descriptions of some model objects) are aligned, generalized and composed into an algorithm definition instead of using artificial, previously created formalisms.

We analyze the alternative components of some existing programming methods and describe a specific method, where the usage of the artificial formalisms is reduced to a minimum.

Logic programming language Sampletalk is described as a supporting tool for the method. Advantages and disadvantages of alternative programming are discussed in a Knowledge Representation context.

1. Introduction

A common, technologically oriented medicine is characterized by intensive application of a wide range of technological achievements taken from chemistry, physics, mechanics, electronics etc. Alternative medicine is more concentrated on natural adaptation of the organism to the environment. Both the technological and alternative approaches in medicine have characteristic areas of successful applications and sometimes complement each other. This relation between common and alternative medicine can be analogically translated into a relation between common and alternative programming. The analogy leads to some interesting implications.

First of all, let's use this analogy in order to define what an alternative programming is. Common programming methods today are mostly formalism-oriented. They are based on numerous *artificial* formalisms, taken from formal grammar theory, DB theory, logic, algebra, calculus etc. Alternative programming, as the analogy suggests, assumes much smaller usage of the artificial formalisms. The lack of formalisms is compensated by a more flexible interaction between program elements, which, ideally, contain patterns of natural data and data transformations, *demonstrating* the algorithm.

Like for the medical methods, this partition is not mutually exclusive, and some programming methods can be related to both categories. To make our definition more specific, we relate the alternative methods to an adaptation and hybridization of natural data examples into patterns, suitable for algorithm definition. This can be realized as *an alignment-, a matching-, a generalization- and a composition* of data examples (below we describe these operations in more detail).

Alternative-programming components exist in many existing formalism-oriented programs and programming methods. For example, a formal grammar may contain patterns of natural language data and their transformations. The patterns are aligned and interact with each other when the rules of grammar are applied. Many corpora-related NLP methods can be related to alternative programming. See also Table 1 below.

2. Alternative and Artificial Formal Methods in Logic Programming

Relationship between alternative and artificial formal components in programming is sometimes controlled explicitly. Let's analyze this relationship in some Logic Programming methods and languages.

Consider the simplified scheme of Prolog program given in Fig.1. Assume that some logical atoms, applied in this program, represent generalized patterns of natural data taken from the program environment.

The alternative component here consists of a) a composition of program clauses from the logical atoms, b) a run-time instantiation of the clauses, which is based on a unification of current goals with appropriate clause heads and c) the Resolution principle, which determines the inference. Indeed, logic atom unification can be considered as an alignment and a matching. Prolog variables take values globally in the entire clause; this can be considered as a mutual adaptation of the atoms from the clause goal, clause head and clause body. The results of this interaction are set as new goals by the Resolution rule.

The artificial formal component here consists of a formal syntax of logical atoms, a side effect of built-in Prolog predicates, special Prolog notations for lists, the DCG syntax etc. We can compare the alternative and formal components in different modifications of Prolog. For example, in object-oriented extensions of Prolog the formal component is more sophisticated. In Datalog, where function symbols are not applied, the formal component is simpler.

Because of the formal syntax of logical atoms, presented in the artificial formal component, in Prolog we cannot apply unrestricted texts as goals, clause heads and subgoals (see Fig. 1). Sampletalk language [1, 2, 3] is a modification of Prolog where this limitation is reduced, so the alternative component becomes more significant. Unconstrained texts (which, however, may contain variable symbols and some hierarchical structure) are applied in Sampletalk instead of the logical atoms. In order to make this paper self-contained, we will provide some more details about Sampletalk below.

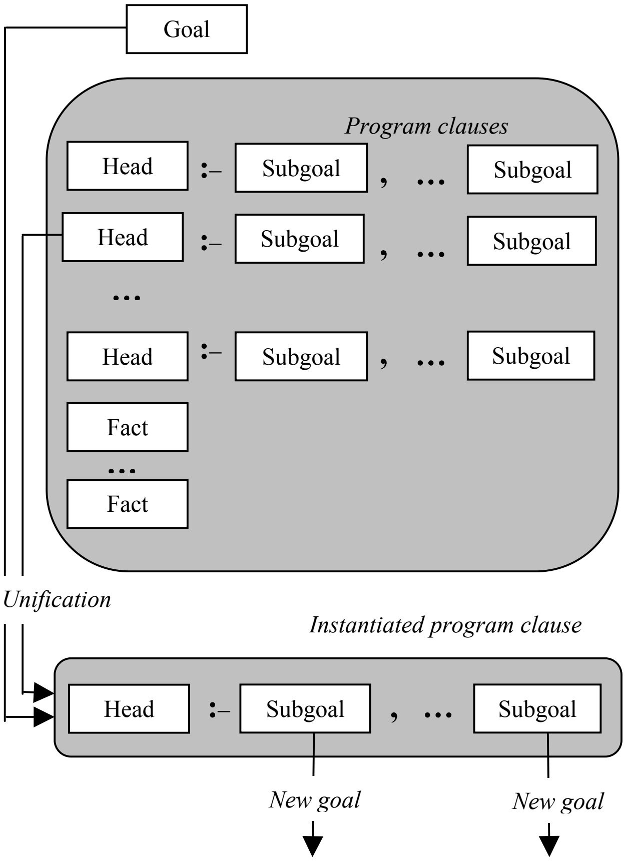

Fig. 1. Instantiation of a clause and generating new goals.
White rectangles represent:
Logical atoms in Prolog,
Logical atoms without functional symbols in Datalog,
Unrestricted texts, containing variable symbols, in Sampletalk

3. Alternative Method of Formalization

Interestingly, alternative components can replace formal components, so the method of alternative programming can be applied for formalization. The Sampletalk theory, given in this section as an example of such alternative formalization, is constructed *exclusively* from generalizations of English phrases that explain morphological analysis, by examples. Without discussion of the linguistic quality of this theory, it should be considered as a demonstration that we can extract algorithms *immediately* from natural data examples, without any other algorithm definition means.

Linguistic terms morphological analysis, verb, noun, past form etc, related to the particular senses applied in this theory, are *defined* just here. The theory produces a morphological analysis of word-forms, which are represented via simpler entries in a dictionary contained in the theory. Consider the following 5 inputs (queries) for the theory:

1) result of morphological analysis of word [b a c k]: this is X.
2) result of morphological analysis of word [b a c k e d]: this is X.
3) result of morphological analysis of word [b a k e d]: this is X.
4) result of morphological analysis of word [q u e r i e d]: this is X.
5) result of morphological analysis of word [q u e r i e s]: this is X.

Variable X stands for the result of the morphological analysis. Application of the theory to the above inputs provides the following results:

1) result of morphological analysis of word [b a c k]: this is canonical verb form "b a c k"’;
result of morphological analysis of word [b a c k]: this is canonical singular form of noun "b a c k";
result of morphological analysis of word [b a c k]: this is adverb "b a c k".
2) result of morphological analysis of word [b a c k e d]: this is past form of verb "b a c k".
3) result of morphological analysis of word [b a k e d]: this is past form of verb "b a k e".
4) result of morphological analysis of word [q u e r i e d]: this is past form of verb "q u e r y".
5) result of morphological analysis of word [q u e r i e s]: this is 2-sg form of verb "q u e r y";
result of morphological analysis of word [q u e r i e s]: this is plural form of noun "q u e r y".

In the Sampletalk language, double dots separate clauses, double commas separate texts in clauses, the neck-symbol ":–" separates clause heads from clause bodies. Variable names, like in Prolog, start from uppercase

Roman letters. Special brackets “[“ and “]” may define a hierarchical structure on texts, which is applied in order to constrain matching of the unified texts (so structured texts with variables are called *samples*). Note that words and phrases “result”, “of”, “morphological analysis” etc. are *not* predefined constructions of the language: they only determine the matching of the corresponding clause heads and goals. About 30 additional Sampletalk program examples can be found in [1] and [2]. In paper [3] Sampletalk is described as a language of First Order Text Calculus (FOTC), which is a generalization of First Order Predicate Calculus (FOPC), where *samples* are applied in logic formulas instead of logical atoms. Below is the theory.

```
result of morphological analysis of word [X i e d]:
this is past form of verb "X y" :–
    word “X y” has canonical morphological tag “verb”..
result of morphological analysis of word [X e d]:
this is past form of verb "X e" :–
    word “X e” has canonical morphological tag “verb”..
result of morphological analysis of word [X e d]:
this is past form of verb "X" :–
    word “X” has canonical morphological tag “verb”..
result of morphological analysis of word [X i e s]:
this is 2-sg form of verb "X y" :–
    word “X y” has canonical morphological tag “verb”..
result of morphological analysis of word [X e s]:
this is 2-sg form of verb "X e" :–
    word “X e” has canonical morphological tag “verb”..
result of morphological analysis of word [X s]:
this is 2-sg form of verb "X" :–
    word “X” has canonical morphological tag “verb”..
result of morphological analysis of word [X i e n]:
this is past-p form of verb "X y" :–
    word “X y” has canonical morphological tag “verb”..
result of morphological analysis of word [X e n]:
this is past-p form of verb "X e" :–
    word “X e” has canonical morphological tag “verb”..
result of morphological analysis of word [X e n]:
this is past-p form of verb "X" :–
    word “X” has canonical morphological tag “verb”..
result of morphological analysis of word [X y i n g]:
this is ing-form of verb "X y" :–
    word “X y” has canonical morphological tag “verb”..
result of morphological analysis of word [X i n g]:
this is ing-form of verb "X e" :–
    word “X e” has canonical morphological tag “verb”..
result of morphological analysis of word [X i n g]:
this is ing-form of verb "X" :–
    word “X” has canonical morphological tag “verb”..
result of morphological analysis of word [X]:
this is canonical verb form "X" :–
    word “X” has canonical morphological tag “verb”..
result of morphological analysis of word [X i e s]:
this is plural form of noun "X y" :–
    word “X y” has canonical morphological tag “singular noun”..
result of morphological analysis of word [X e s]:
this is plural form of noun "X e" :–
    word “X e” has canonical morphological tag “singular noun”..
result of morphological analysis of word [X s]:
this is plural form of noun "X" :–
    word “X” has canonical morphological tag “singular noun”..
result of morphological analysis of word [X]:
this is canonical singular form of noun "X" :–
    word “X” has canonical morphological tag “singular noun”..
result of morphological analysis of word [X]:
this is adverb "X" :–
    word “X” has canonical morphological tag “adverb”..
result of morphological analysis of word [X]:
this is adjective "X" :–
    word “X” has canonical morphological tag “adjective”..

word “b a c k”
    has canonical morphological tag  “singular noun”..
word “b a c k”
    has canonical morphological tag  “adverb”..
word “b a c k”
    has canonical morphological tag  “verb”..
word “b a k e”
    has canonical morphological tag “singular noun”..
word “b a k e”
    has canonical morphological tag  “verb”..
word “b a k e r”
    has canonical morphological tag  “singular noun”..
word “b a k e r y”
    has canonical morphological tag “singular noun”..
word “q u e r y”
    has canonical morphological tag  “singular noun”..
word “q u e r y”
    has canonical morphological tag “verb”..
```

The verbosity of this theory is compensated by extreme affinity of its clauses to simple and easily understandable natural-language phrases and inferences. However large the theory may be the clauses are simple and contain only references to common natural-language terms and senses. Based on this simplicity, we can join many such theories into larger ones, which combine so defined senses.

Note that the theory is simultaneously:

- *a parser* (related to a formal grammar, which we don’t have to build),
- *a program* (which transforms some inputs into some outputs),
- *a generator* (of a class of NL phrases, the grammar of which we also don’t have to build),
- *a knowledge base* (which contains patterns of verbal descriptions, taken from a linguist-expert),
- *an inference system* (based on the Resolution principle),
- *a query answering system* (which understands and produces phrase patterns of a subset of the English language).

The alternative-programming method allows us to easily combine all related formal techniques in a single program entity, without having to bother with the details of those techniques separately or to design interfaces between subsystems.

The functioning of this theory is analogous to the functioning of decision lists. So, we can introduce our understanding of linguistic concepts to a computer in the form of generalized verbal explanations of specific examples, where the process of generalization consists in replacing some text constituents with variables. The theory employs this knowledge by analogy, using a match between text samples, which include the corresponding linguistic terms. This way the program provides an *alternative definition* of the linguistic terms and senses.

The theory does not apply any artificial formal programming components (we do not say this about the Sampletalk compiler, which runs the theory). *For the alternative programmer*, the natural language expressions serve the same role that formal constructions of conventional programming languages serve *for the formal-language programmer.* They help define the data and the actions with this data. Here, however, we can fluently apply as many natural language expressions as we want, provided the corresponding word combinations in the phrase patterns are correctly unified. In comparison with the limited sets of predefined constructs in formal computer languages, here we have a substantially richer natural language with an unlimited set of expressions, which can be *immediately* used for algorithm definition in the alternative framework.

So, instead of adapting our natural thinking to an artificial algorithm description language like C++ or Prolog, we can immediately transform a natural-language description of the algorithm into a working formal definition of the algorithm.

4. Alternative Formalization for Knowledge Management: Logico-Linguistic Inference

Let’s consider a Sampletalk theory as an ontology in the following way:

- Term occurrences in clause heads are considered *concept definitions;*
- Term occurrences in clause bodies are considered suggested *concept applications;*
- *Semantics* of the terms is defined by the suggested concept applications.

Unlike direct concept definitions (based on frames or semantic networks), these definitions are strictly context-oriented and only assume availability of model text examples (generalized and composed into clauses). In the theory example of Section 3 we find definitions and applications of terms and concepts “result”, “of”, “is”, “has”, ‘morphological”, “analysis”, “word”, “verb“ etc. The terms are applied *via analogy*, in similar contexts, and are related to specific senses. Usage of these terms in other word collocations is not defined. The definition of the term “morphological” in this theory is more flexible then the definition of the term “analysis” since the term “morphological” is defined in different contexts and can be related to different senses (see the first and the last clauses of the theory, where this term is applied in different environments). The definition of the term “is” in this theory is still more flexible.

The senses, related to the *variables* X and R in the following Sampletalk theory are also less restricted:

```
where is X? in R :– X is situated in R..   % Inference rules
what is on X? R :– R is situated on X..
red pencil is situated in small bag..       % Facts
book is situated on large table..
notebook is situated on small table..
```

Let’s apply this theory using the following queries:

```
where is red pencil? in R
what is on X? R
```

We will have the following answers:

```
where is red pencil? in small bag
what is on large table? book
what is on small table? notebook
```

In such a way, the generalized natural language descriptions immediately participate in *reasoning* (the first two clauses) and in the *storage of the facts* (the last clauses). Once more, we simultaneously have *a database, an inference engine, a parser* and *a phrase generator* in a single program entity.

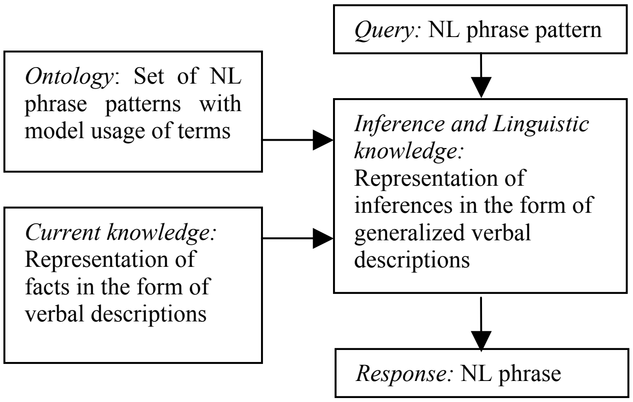
Query: NL phrase pattern
Ontology: Set of NL phrase patterns with model usage of terms
Inference and Linguistic knowledge: Representation of inferences in the form of generalized verbal descriptions
Current knowledge: Representation of facts in the form of verbal descriptions
Response: NL phrase

Fig. 2. Query processing using the alternative Knowledge Representation model. The arrows represent an immediate interaction of phrase patterns, contained in the clauses. Other FOL-based systems require a parser, a phrase generator (vertical arrows), a semantic network and a DB (horizontal arrows).

The application of other, non-alternative First Order Logic (FOL) based knowledge representation methods usually assumes such components to be complex and separate program entities (see Fig. 2).

An analysis of NL phrases, applied in simple question and answer situations, shows that a logic inference, related to the discussed subject, is often toughly connected with the features of the involved words and phrases. Linguistic modeling of a dialogue situation sometimes leads to a more complex model than the discussed subject itself. Let's try to see what the alternative programming method can do for a dialogue modeling.

Consider the following situation, which will represent the discussed subject in our next Sampletalk theory example:

> A book is in an open box. A notebook is in a closed box. The open box is on a red table. The closed box is on the red table. John has approached the red table.

Suppose that a human observes this situation. We may ask various questions like "Can John see the notebook?" or "What is on the red table?". Probably we will get some answers, which may depend on additional facts (like transparency or non-transparency of the boxes), on the observer's logic, and on the observer's language skills.

The Sampletalk theory, given below, also forms the answers according to its own logic and to the natural language phrase patterns, contained in its clauses. Trying to create a flexible dialogue model, in this example we *intentionally* combined a linguistic type of inference with a logic inference, related to the subject. Consider the following input-queries for the theory (the theory is given below).

1) can [john] see [book]?
2) can [john] see [notebook]?
3) can [john] see [open box]?
4) what can [john] see? [X]
5) what is visible on [red table]? [X]
6) what is invisible on [red table]? [X]
7) what is on [red table]? [X]
8) what is in [open box]? [X]
9) what is in [closed box]? [X]

Like in our previous examples, the variable X stands for the fragments of the query results, which are instantiated during the inference. Application of the theory to the above inputs provides the following results:

1) can [john] see [book]? *Yes.*
2) can [john] see [notebook]? *No.*
3) can [john] see [open box]? *Yes.*
4) what can [john] see? [open box]. *Yes*;
   what can [john] see? [closed box]. *Yes*;
   what can [john] see? [book]. *Yes.*
5) what is visible on [red table]? [open box]. *Yes*;
   what is visible on [red table]? [closed box]. *Yes*;
   what is visible on [red table]? [book]. *Yes.*
6) what is invisible on [red table]? [notebook]. *Yes.*
7) what is on [red table]? [open box]. *Yes*;
   what is on [red table]? [closed box]. *Yes*;
   what is on [red table]? [book]. *Yes*;
   what is on [red table]? [notebook]. *Yes.*
8) what is in [open box]? [book]. *Yes.*
9) what is in [closed box]? [notebook]. *Yes.*

The words in Italic, *Yes* and *No*, here indicate a successful or an unsuccessful status of the application of the theory to the query. Some queries have multiple answers.

Below is the theory. It contains a set of logico-linguistic inference rules, followed by the facts in a simple NL form. Note the prefix "~~"in the 5th clause, which in Sampletalk language denotes a logic negation, which is analogous to a negation by failure in Prolog. Comments in Sampletalk start with the %-sign.

```
% Logico-linguistic inference rules:
can [A] see [X]? :– [A] can see [X]..
what can [A] see? [X] :– [A] can see [X]..
what is in [A]? [X] :– [X] is in [A]..
what is visible on [A]? [X] :–  [X] is visible on [A]..
what is invisible on [A]? [X] :– [X] is in [B],, [B] is on [A],,
          ~~[X] is visible on [A]..
what is on [A]? [X] :– [X] is on [A]..
what is on [A]? [X] :– [X] is in [B],, [B] is on [A]..
[A] can see [X] :– [A] is standing near [B],,
          [X] is visible on [B]..
[A] is visible on [B] :– [A] is on [B]..
[A] is visible on [B] :– [A] is in [open X],,
          [open X] is visible on [B]..
[A] is standing near [B] :– [A] has approached [B]..

[book] is in [open box]..                    % Facts
[notebook] is in [closed box]..
[open box] is on [red table]..
[closed box] is on [red table]..
[john] has approached [red table]..
```

This theory example expresses and supports reasoning about objects (a book, a notebook, john etc) and relations between the objects. Among the relations we find:

- Spatial relations: "is on", "is standing near";
- Inclusion relations: "is in";
- Temporal relations and action patterns: "has approached", "is standing near";
- A modal logic relation "can see", concerning a possibility that a person perceives objects;
- Relations concerning the visibility of objects.

We may also consider this reasoning as a reasoning about epistemological relations ("can see" in the sense "is aware of"). The relations are *defined* and *applied* in the alternative-programming way, via matching of the phrase patterns. The roles and types of the objects-operands of the relations are also determined in the alternative way, according to the word positions in the phrase patterns and in compliance with our model of ontology, described at the beginning of this section.

Curiously, we didn't exert any efforts in order to formalize the semantics of these concepts and relations, but still can apply the corresponding senses in the alternative framework! We simply exploit the *natural* semantics of the involved phrase patterns, which is inherited from the original NL phrases.

Flexibility of such representation of the dialogue is supported by the following observation. We can easily develop the logico-linguistic inference for answering a new kind of questions expressed in a NL form. For instance, the user might want to be able to ask what is near John or who is near the red table. In this case we should simply add the following clauses:

```
what is near [A]? [X] :– [A] is standing near [X]..
what is near [A]? [X] :– [A] is standing near [Y],, [X] is on [Y]..
who is near [X]? [A] :– [A] is standing near [X]..
who is near [X]? [A] :– [X] is on [Y],, [A] is standing near [Y]..
```

Due to the default reasoning, already contained in the theory, these new clauses provide more general answers, supporting some additional questions, e.g., a question "who is near [notebook]? [A]". Patterns of unrestricted natural language phrases, which reflect our perception of the situation, are simply composed into logic clauses for modeling the dialogue situations.

The price we pay for such flexibility is a messy theory, where the logic of the discussed subject is intermingled with the logico-linguistic features of the involved words and expressions. Note, however, that even a human, reflecting a situation to be discussed, applies a complex combination of logical and linguistic senses.

The gain of such logico-linguistic formalization is not only the mentioned flexibility for adding a new functionality. Any attempt to formalize a similar reasoning using a predicate-logic language (e.g., Prolog) will immediately show the need to implement:

- A grammar and a parser for the transformation of the input queries into a predicate form;
- A reasoning engine with specially designed predicates for the involved spatial, temporal, epistemological or modal-logic relations;
- Another grammar and a phrase generator for transformation of the inference results into a NL form.

Sampletalk allows us to implement all of this functionality in the simplest possible way: by immediate inclusion of NL phrase patterns into the reasoning engine.

Excessively messy theories may present an evident problem of this approach. In order to avoid excessive sophistication in Sampletalk theories, we usually try to make use of the most frequent, universally recognized NL words and expressions like "is", "is in", "has", "can" etc., applying the more specific expressions like "morphological analysis", "invisible" etc. only for special cases. Our examples show that all these words can *immediately* be used for formalization and maintenance of the meaning of the text in the alternative-programming framework. In a metaphorical sense, the universal words and concepts play the role of nuclear forces for building phrases (atoms) and sentences (molecules). In the Sampletalk context they play a similar role in *samples* and in organizing them into clauses. In another comparative sense (already mentioned at the end of Section 3), these words play a role, similar to the role of key words of the artificial programming languages for using the *artificial* formalisms.

As our examples show, some NL words and expressions, contained in the *samples*, are related to specific semantic relations. In order to implement their semantics in the alternative framework, we don't need any *artificial* formalization. Instead, we strive to inherit the semantics from the *natural* semantics of the existing NL phrases.

5. Implications of the Alternative-Programming Model

The first interesting implication of the alternative method is a possibility to create very large theories without having to define predicates, modules, procedures, arrays, formal grammars and other *artificial* formal constructions. Indeed, a combination of senses via a combination of simple natural-language phrases (however verbose it may be) is more transparent and easily controllable than a combination of artificial formal descriptions, which usually assumes availability of compilers, linkers, formal inference engines and other complex objects, supporting a formal programming. We can extract – manually or automatically – patterns of natural language descriptions, which explain the algorithm, and collect these patterns in the theory. As an example of such a theory, we can develop the morphological analysis theory from Section 3 into a more representative morphological analyzer, which contains a large dictionary of canonical word forms and a large set of morphological transformation rules.

The second implication is a possibility to create formal models, which are unbiased and independent of any artificial formalization known to a human. Alternative formal models, composed *exclusively* from data examples, can provide a new knowledge in domains where the data is not enough understood, e.g., genetic sequences. (We still cannot support this implication by specific examples, but we regard this as a perspective research area. Consider also the discussion in Section 6, speaking of such sequences instead of the verbal descriptions, discussed in that section).

The third implication is an accessibility of the universal programming methods to the problem domain experts (e.g. linguists), who can be reluctant to use sophisticated artificial formalisms or don't have suitable artificial formalisms for their needs. Like the alternative medicine doctors can do without modern pills or an X-ray apparatus, the alternative programmer can create complex algorithms without artificial formal data types or functions. He needs only examples of natural data and data transformations, which *demonstrate* the algorithm (like in our morphology analysis example, where a linguist-expert builds the algorithm immediately from a natural-language description of the algorithm).

The fourth implication is a possibility to apply in our theories a more expressive notation for denoting objects and relations than what is allowed in the formal systems, based on the functional or predicate notations (e.g., FOPC-systems). For example, the disadvantage of the predicate notation

```
hasChild (anna, jacopo)
```

is, that without a detailed reading of the axioms or the informal comments, it might not be clear who is whose child. In Sampletalk we can apply more natural notations, e.g.

```
[jacopo] is child of [anna],
[anna] has child [jacopo],
[anna] is parent of [jacopo],
```

which can be extracted immediately from the domain description texts.

The fifth implication is the availability of natural means for program documentation and program debugging. For example, the morphological analysis theory of Section 3 can be treated as an algorithm documentation, which *works* as the algorithm it describes. In a debug session, the linguist-expert can easily understand the intermediate data of any inference step. As an example of a debug session, consider the 1st input-query for the theory about the open and closed boxes, given in Section 4. The Sampletalk compiler produces the following proof trace, which is typical for logic inference systems except for the unrestricted-text notations, applied instead of the logic predicates:

```
can [john] see [book]?  ?                      Goal
[john] can see [book]  ?                       Proof
[john] is standing near [_]  ?
[john] has approached [red table]  ?  Yes
[john] is standing near [red table]  ?  Yes
[book] is visible on [red table]  ?
[book] is on [red table] ?  Fail
[book] is visible on [red table]  ?
[book] is in [open box]  ?  Yes
[open box] is visible on [red table]  ?
[open box] is on [red table]  ?  Yes
[open box] is visible on [red table]  ?  Yes
[book] is visible on [red table]  ?  Yes
[john] can see [book]  ?  Yes
can [john] see [book]?  ? Yes
```

The *Fail*-marked line helps reveal the deficiency of the "is on" relation in the theory, which can be improved by a slight modification of the theory.

A related implication is an immediate possibility to get an easily understandable verbal explanation of the inference. Below is a shorter version of the above proof, where the resolved subgoals, applied in the inference, are shown in reverse order:

```
    can [john] see [book]?                     Goal
Explanation:
1  [open box] is on [red table],               DB Fact
2  [open box] is visible on [red table],       From 1
3  [book] is in [open box],                    DB Fact
4  [book] is visible on [red table],           From 2,3
5  [john] has approached [red table],          DB Fact
6  [john] is standing near [red table],        From 5
7  [john] can see [book].                      From 6,4
   can [john] see [book]?  Yes                 From 7
```

The relationship between the alternative and the formal components has an influence on the class of problems, which can be treated by the corresponding method. In Table 1 the alternative and the formal components of some existing algorithm creation methods are outlined.

Table 1. Alternative and formal components of some algorithm creation methods

Programming language, formalism, method	*Alternative component*	*Artificial formal component*
Prolog	Atom & term unification, Resolution principle	Syntax and side effect of predicates and functions, proof strategy
Datalog	Atom unification, Resolution principle	Syntax and side effect of predicates, proof strategy
Sampletalk [1, 2, 3]	Text unification, Resolution principle	Proof strategy
Markov algorithm [4, 5]	String alignment, replacement	Rule application strategy
Turing machine	N/A	Commands, machine head movement
C, C++, Assembler	N/A	Syntax & semantics of operators, data types, functions
Snobol [6]	String matching	Syntax & semantics of operators, modules
ILP [7, 8]	Atom & term unification, least general generalization, Resolution principle, Backward resolution	Syntax and side effect of predicates and functions, proof strategy, induction strategy
Genetic programming [9]	Crossover operator	Fitness function
Automatic classification methods	Pattern similarity	Vector space geometry

The alternative component in automatic classification systems (see the bottom row of the table) often includes an immediate incorporation of sample patterns from a learning set into the classification algorithm. Our Sampletalk examples show that we can do this for a general-purpose programming: the generalized patterns of the natural text examples and their interactions define the algorithm.

So far we discussed the advantages of the alternative-programming method. Among the disadvantages we should mention the possibilities of unexpected matching between the data patterns. For instance, when debugging Sampletalk programs, we sometimes meet the wrong inferences, caused by under-generalized or over-generalized phrase patterns. Similar problems of over-fitting and over-generalization exist in most data mining techniques (this can be considered to be additional evidence of the relatedness between data mining and the alternative-programming method).

As the second disadvantage, like alternative medicine, alternative programming may require a special education and talents, unusual among programmers accustomed to the artificial formalisms. The alternative programmer should well understand what kind of knowledge is *demonstrated* in the natural examples, how it can be generalized, which data pattern alignments and interactions are desirable and which are not. We see, however, that the advantages of the alternative methods in such fields as data mining, knowledge engineering and natural language processing, prevail the disadvantages.

6. Discussion: Automatic Generalization of Verbal Descriptions for Practical Applications

As we saw, the generalized verbal descriptions (descriptions where some text constituents are replaced with variables) can serve as a building material for constructing Sampletalk theories. Using simple statistical methods, we can prepare a large set of generalized sentences and phrase patterns from a raw NL corpus. Then we can try to apply this material and some machine learning methods for automatic construction of Sampletalk theories for specific applications.

In a machine learning framework, the problem of generating a Sampletalk theory P can be formulated as follows. Consider the following two sets of pairs of *samples*, which represent the desired and the undesired input-output pairs correspondingly:

$E_+ = \{(s_i, t_i): i = 1, 2, \dots, n\}$ *(positive examples)*
$E_- = \{(s'_j, t'_j): j = 1, 2, \dots, m\}$ *(negative examples)*

A notation Q(s) = t will be used for denoting a theory Q, which produces an output t from an input s. A theory P can be automatically generated using the following objective function:

$f(Q) = (|\{i: Q(s_i) = t_i\}| - |\{j: Q(s'_j) = t'_j\}|) / |Q|,$
$P = \mathrm{argmax}\,(f(Q)),$

where |Q| denotes the total number of *samples*, contained in the clauses of theory Q. The numerator of the objective function characterizes a quality of the theory Q relating to the sets E_+ and E_-. The denominator characterizes a complexity of the theory Q.

We can consider various sources of *samples* for inclusion into the theory Q in the optimization process. For example, we can create such *samples* using a pairwise alignment and generalization of sentences, contained in a natural language corpus.

Consider the following set of sentences (here we avoid capital letters so that they could be used for variables):

a book is in an open box
a notebook is in a closed box
cat is in the cage
a fox is in zoo
john is in his red car
this large table is red

Pairwise alignment of these sentences provides a set of generalizations, which contains the following *samples*:

a [X] is in [D] [Y] box,
[X] is in [D] [Z],
a [X] is in [Z],
[X] is in [Z],
[X] is [A] ,

where the variables X and Z stand for a noun or a noun group, D stands for articles, Y stands for adjectives, and A stands for adjective phrases. The generalization is done by substitution of non-matching text constituents with variable symbols. (Details of our methods for generating such samples from a set of raw text examples are given in [3]). Note that we get such generalized sentence patterns without any previously constructed grammar or morphology models.

The discussable subject, suggested by these examples, is a possibility to construct practical question and answer systems, using material of this kind, extracted from a large raw corpus of sentences. This section can be considered an invitation to carry out experiments in order to confirm or reject this possibility for various problem domains, e.g. for developing a library-related dialogue system about book visibility or availability, or a system for a dialogue about geographical objects.

In view of this discussion, an interesting observation about all our program examples is that we don't need the explicitly defined qualifiers (like formal grammar non-terminals: a noun-group, an adjective phrase etc) for the variables X, R, A, B etc, although the variables may match only phrases of the corresponding types when the theories are applied to correct input. For example, if a simple and correct sentence is aligned with a *sample* "[X] is in [Z]", then we can expect that variables X and Z represent noun groups, although we did not define what a noun group is.

As a curious implication of this observation, we have a certain freedom from the ambiguity problems, related to the usage of formal grammars. In fact, we don't apply formal grammars at all: the alternative methods provide the alternative options for text analysis and text generation.

However, the multiple possibilities for alignment and unification of samples can impose their own ambiguity problems. For example, if the sample “[X] is in [Z]” is aligned with a complex raw sentence, then variables X and Z may represent not only noun groups, but also some non-classifiable sentence fragments. We can approach these problems using the same alternative methods, without application of the artificial mechanisms for resolving ambiguity. For example, when aligning a sentence with the sample “[X] is in [Z]”, we may require that the text constituents for the variables X and Z do not contain still non-structured phrases with words “is”, “in” (and other words, presented in additional patterns of simplest sentences).

Doing a pairwise alignment and generalization for a corpus of sentences, we can expect that the frequency of the shorter generalizations is higher and that the shorter generalizations represent more correct, more general, and more applicable sentence patterns, like those applied in our examples in Section 4.

7. Related Work

Sampletalk Language (formerly known as Sample Language) was introduced in papers [1] and [2]. In [1] and [3] we discuss several algorithms for the induction of Sampletalk theories from positive and negative text examples using ILP methods (see [7, 8]). The Turing-completeness of Sampletalk language is shown in [1] and [3]. The Sampletalk compiler is currently available at http://www.sampletalk.8m.com.

Text unification provides a possibility to alternatively define and operate with senses, related to the aligned text constituents. Unification of various kinds of structures (which have slots or variables for filling-in by the matching constituents) is applied in various fields; the unification of logical atoms is the most prominent example. Unification is applied not only in a Logic Programming framework, as we do, and not only for texts or logical atoms.

Unification grammar (see, e.g., [10], [11]) presents an example where a unification of special objects (feature structures) is applied for constraining the application of formal grammr rules. In this research, however, the unifiable structures play a secondary role in reasoning: the primary role has the formal grammar. In a Logic Programming framework the unification of objects has a more active role, supporting an *alternative definition* and a manipulation of a larger class of senses.

Matching of various structures, containing variables, is applied in Snobol [12], Refal [13] and Planner [14] languages. Sampletalk can be considered an attempt to extend these approaches for working with unrestricted texts, containing slots (variables). Unification is applied also in Carpenter's Attribute Logic Engine [15].

Extraction of decision lists and morphological analysis algorithms from examples using ILP (Inductive Logic Programming) methods is currently an active research area: see, e.g. [16]. ILP suggests substantially developed inductive methods for inference of predicate-logic programs from examples of logical atoms and a background knowledge, which is also expressed in a predicate-logic form. Grammar induction using ILP methods (LLL, Learning Language in Logic) is described in [17]. Sampletalk theories can be considered an alternative to both *predicate-logic-based* and *formal-grammar-based* approaches to text analysis and text generation.

There is a widespread opinion that logical methods will be of growing importance in the field of NLP in the near future. Sampletalk language can be considered a convenient interface between linguistic and logical methods, which may facilitate the combination of ideas from these research fields. Paper [18] presents a large Sampletalk program for machine translation between English and Hebrew, where some ambiguity problems are treated using the alternative methods.

Conclusions

Knowledge representation and management using First Order Logic can be simplified if the generalized verbal descriptions are applied in logic clauses instead of the logical atoms. In the alternative-programming framework, the generalization of verbal descriptions can be done by a pairwise alignment of similar descriptions and substitution of non-matching text constituents with variable symbols.

The Logic Programming language Sampletalk serves to build algorithms *immediately* from such generalizations. Using this technique, we can introduce our understanding of specific concepts (e.g., linguistic, topological, physical, epistemological, time-related concepts) to a computer in the form of generalized verbal explanations of specific examples.

Alternative-programming methods are based on an immediate incorporation of natural environment patterns (such as patterns of natural-language phrases or patterns of objects to be classified) into a theory or an algorithm. Alternative-programming components exist in many existing algorithm creation methods.

Instead of adapting our natural thinking to an artificial formal language, we can transform our natural-language description of the algorithm into a working formal definition of the algorithm.

Alternative-programming methods are suitable for problem domain experts (such as linguists or dialogue system developers) as an alternative to the artificial programming languages. The methods can be applied for creating large formal theories and for an immediate transformation of elements of program documentation into a working algorithm.

Acknowledgement

This work was initiated during my work at the Institute of Theoretical Astronomy of the Russian Academy of Sciences (ITA RAN). I would like to thank Nikolai Kirsanov and Inna Vigant, who were the first Sampletalk programmers (correspondingly at the ITA RAN and at Haifa Technion), for their experiments and suggestions.

References

1. Gleibman, A.H. Synthesis of Text Processing Programs by Examples: The "Sample" Language. Preprint of the Institute of Theoretical Astronomy of the Russian Academy of Sci. No.15, 27 pp., 1991. In Russian.

2. Gleibman, A.H. Sample: New Programming Technology and AI Language. Data processing example abstraction immediately becomes a useful program if text matching is in focus. *ACM SIGSAM Bulletin*, v. 26, No.2, 1992, p.21-29.

3. Gleibman, A.H. First Order Text Calculus. Submitted for publication.

4. Markov, A.A. Theory of Algorithms. *Trudy Matematicheskogo Instituta Imeni V. A. Steklova*, v. 42 (1954), pp.3-374. In Russian.

5. Caracciolo di Forino, A. String processing languages and generalized Markov algorithms. In Symbol manipulation languages and techniques, D. G. Bobrow (Ed.), North-Holland Publ. Co., Amsterdam, The Netherlands, 1968, pp. 191-206.

6. Griswold, Ralph E. The Macro Implementation of SNOBOL4. San Francisco: W.H. Freeman and Company, 1972.

7. Muggleton, S. Inductive Logic Programming, *New Generation Computing*, 8(4), 295-318, 1991.

8. Muggleton, S.H. and De Raedt, L. Inductive Logic Programming: Theory and Methods. *Logic Programming*, 19, 20, p.629-679, 1994.

9. Koza, J.R. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, 1992.

10. Jaeger E., Francez N., Wintner S. Unification Grammars and Off-Line Parsability. *Journal of Logic, Language and Information*. Prepublication Date: 05/06/2004.

11. Jurafsky, D. and Martin, J.H. Speech and Language Processing. An introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Prentice-Hill, 2000.

12. Griswold, Ralph E. The Macro Implementation of SNOBOL4. San Francisco: W.H. Freeman and Company.

13. Turchin, V.F. Programming in Refal. Preprints No. 41, 43, 44, 48, 49 of the Institute for Applied Mathematics of the USSR Academy of Sciences, 1971. In Russian.

14. Hewitt, C. Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot. TR-258, - AI Lab., MIT, Cambridge, Mass.

15. Carpenter, B., and Penn, G. The Attribute Logic Engine. User's Guide. Version 3.2.1. December 2001. SpeechWorks Research, NY; University of Toronto, 1992-2001.

16. Kazakov, D. Achievements and Prospects of Learning Word Morphology with Inductive Logic Programming. In Cussens and Dzeroski (Eds.), Learning Language in Logic, Springer, 2000, p. 89-109.

17. Dzeroski, S., Cussens, J., Manandhar, S. An Introduction to Inductive Logic Programming and Learning Language in Logic. In Cussens and Dzeroski (Eds.), Learning Language in Logic, Springer, 2000, p. 4-35.

18. Vigant, I. Natural Language Processing by Examples. M. Sci. Thesis. Comp. Sci. Dept., Technion, Haifa 1997. (115 pp). In Hebrew, with abstract in English.