DIMACS Technical Report 2001-16
May 2001

Application of default reasoning to semantic processing under question-answering

by

Olga Ourioupina[1] and Boris Galitsky[2]

[1] Dept. Theoretical & Applied Linguistics, Moscow State University, Moscow, Russia
[2] KnowledgeTrail, Inc., 9 Charles Str. Natick MA 01760 USA.
http://dimacs.rutgers.edu/~galitsky
(galitsky@dimacs.rutgers.edu)

The second author gratefully acknowledges the grant NSF STC 91-19999 for the support while visiting DIMACS.

ABSTRACT

We build the natural language question-answering system, where the query representation formula is subject to transformation in accordance to the set of default rules. This transformation is required to disambiguate entities in a vertical domain, where they usually have a principle meaning and a set of foreign ones. Default reasoning is discussed to be applicable to a variety of semantic processing algorithms, including the semantic header approach. Using default rules can be referred as pragmatic machinery, complementary to syntactic and semantic processing of complex queries in a vertical domain. The methodology of building the set of default rules is developed, capable of adding the essential entities or eliminating misleading ones to the representation of input query.

Implementation of operational semantics for default reasoning is suggested to transform the query representation by means of conflicting default rules. Technique of the keyword-based question answering is developed, where default rules link the set of potential queries with the initially coded canonical ones. The task of automatic annotation is then posed as building the set of keywords semantic headers and the set of canonical semantic headers for an answer.

1 Introduction

Although recent years have seen an explosion in the question-answering (Q/A) technology, there is still lack of systems, satisfactorily providing answers in the domains, which are both logically complex and poorly structured. Designers of every Q/A system find the compromise between the full-featured NLP system, oriented for complex domains, and the shallow processing, leveraging performance and automation issues.

In spite of the high number of commercial Q/A systems, functioning in the wide spectrum of domains, it is still unclear what NLP component is essential to achieve a satisfactory answer accuracy, high performance and efficient domain preparation. Let us enumerate the potential components of an abstract Q/A system, keeping in mind that a particular implementation is based only on some of these components:

1) Morphological and syntactic analysis.
2) Semantic analysis, obtaining the most precise query representation.
3) Pragmatic analysis, transforming query representation in accordance to the answer knowledge base.
4) Answer knowledge base.
5) Information extraction.
6) Information retrieval.

The third component, pragmatic analysis, is usually attracts the least attention. Traditionally, it is designed to compensate probable failures or misunderstandings of the syntactic and semantic components. Pragmatic component is usually based on heuristic rules rather than on a solid linguistic or knowledge engineering background and implements verification of compatibility between the formal representation of a query and the knowledge base. Therefore, pragmatic analysis filters out the hypotheses of syntactic and semantic translation of input query, inconsistent with domain representation. This approach is especially viable under the fully-formalized domains (Galitsky, 1999).

In this paper a technique is proposed that allows developers of Q/A systems to use domain-specific pragmatic information. The aim is not to make semantic representations more precise, but to make the given representation fit better to the domain. This is achieved by application of the Default Logic machinery. Default rules can naturally appear in the domain representation itself. Note that the use of default rules for the transformation of translation formula is independent on the degree of knowledge formalization: it helps from fully to poorly formalized domains. However, one needs to distinguish the default rules for query transformation and for domain representation: the former are linked to NL, and the latter are supposed to be language-independent. Though the defaults reasoning technique is applicable to the variety of semantic analysis approaches, the particular set of default rules is domain-specific.

Default reasoning seems to be well suited to represent the semantic rules, which process the ambiguous terms. In a horizontal domain, a term may have a set of meanings such that each of them is important and needs a separate consideration. For a narrow domain, an ambiguous term usually has one common meaning and one infrequent meaning that still has to be formally represented. The system may assume the default meaning of this term unless it is inconsistent to do so. This inconsistency may be determined based on the occurrence of some words in a sentence, or some terms in the translation formula. We refer the reader to (Hirst 1988, Creary & Pollard 1985) for the discussions of disambiguation problem.

In our paper we use semantic headers, as described in (Galitsky, 2000) for query and domain representation. The proposed technique, though, is representation-independent and can be combined with any approach to semantic analysis, achieving certain level of formalization (Sondheimer et al 1984, van Eijck & Moore 1992, Romacker et al 1999, Ciravegna &Lavelli 1999).

2 Semantic Headers machinery

The technique of semantic headers is intended to resolve the problem of converting an abstract textual document into a form, appropriate for answering a question. There are two opposite approaches to this problem. The first one assumes that complete formal representation of any textual document is possible, and the second one assumes that the textual information is too tightly linked to NL and it cannot be satisfactorily represented without it.

The SH technique is an intermediate one in respect to the degree of knowledge formalization. Only the data, which can be explicitly mentioned in a potential query, occurs in semantic headers. The rest of the information, which would be unlikely to occur in a question but can potentially form the relevant answer, does not have to be formalized.

SH technique is based on logical programming, taking advantage of its convenient handling of semantic rules on one hand, and explicit implementation of the domain common-sense reasoning on the other hand. The declarative nature of coding semantic rules, domain knowledge and generalized potential queries introduces logical programming as a reasonable tool (Tarau et.al.1999, Dahl 1999). At the same time, the machinery of text annotation by the set of keywords has been proven to leverage the machine learning technique. Instead of using just the keywords as semantic means to represent the meaning of a short textual document (answer), we either use the logical formula where the keywords serve as atoms (Section 3 and 4) or apply the pragmatic processing to keyword SHs (Section 5). Therefore, SH technique is some way of merging potential results of statistical approach to Q/A with the logical programming way of matching the formal representation of a query with the formal representation of an answer (semantic header of this answer). What we call "statistical" here is indeed the intuition of a knowledge engineer who has manually looked through a manifold of answers to annotate a given one. In legal and financial domains, where the semantics of conversational language can only be ambiguously mapped into the semantics of the legal language, using just the statistical annotation by keyword does not lead to satisfactory results (Galitsky, 2001).

Under the SH technique, the domain coding starts with the set of answers (the content). As an example, we consider an answer from the tax domain:

"*The timing of your divorce could have a significant effect on the amount of federal income tax you will pay this year. If both spouses earn about the same amount of money, getting a divorce before the year ends will save taxes by eliminating the marriage penalty. However, if one spouse earns a significant amount more than the other, waiting until January will save taxes by taking advantage of the married filing jointly status for one last year.*"

What is this paragraph about? It explains how the time of divorce can affect someone's tax liability and describes possible tax saving strategies for people with different income and filing status. Rather than changing the paragraph in order to adjust it to the potential questions answered within it, we consider all the possible questions this paragraph can serve as an answer to. Building the semantic headers of a textual document is based on the posing of a query understanding problem as the recognition of the best pattern (document, answer). For example, if there is a question such that the paragraph above is a more appropriate answer than any other paragraph from the whole domain, then the paragraph above should serve as an answer or at least a part of an answer to this question.

Evidently, knowledge of the semantic model of the whole domain is required to build the set of semantic headers for a given paragraph. This paragraph serves an answer to the following kind of questions

What are the tax issues of divorce?
How can timing your divorce save a lot of federal income tax?
I am recently divorced; how should I file so I do not have a net tax liability?
Can I avoid a marriage penalty?
How to take advantage of the married filing jointly status when I am getting divorced?

Below is the list of semantic headers for the answer above.

divorce(tax(_,_,_),_):-divorceTax.
divorce(tax(_,_,_), time):-divorceTax.
divorce(tax(file(_),liability,_), _):-divorceTax.
penalty(marriage,_):- divorceTax.
divorce(file(joint),_):-divorceTax.

Then the call to *divorceTax* will add the paragraph above to the current answer, which may consist from the multiple pre-prepared ones.
A generic set of semantic headers for an entity *e* and its attributes $a_1, a_2, \ldots$ looks like the following:

$e(A)$:-$var(A)$, $clarify([a_1, a_2, \ldots])$. If the attribute of *e* is unknown, clarification procedure is initiated, suggesting choosing an attribute from the list.
$e(A)$:-$nonvar(A)$, $A = a_1$, $answer(\#)$. The attribute is determined and the system outputs the answer, associated with the entity and its attribute (# is the answer id).
$e(e_1(A))$:-$nonvar(A)$, $A= a_1$, $e_1(A)$.
$e(e_1(A),e_2)$:-$nonvar(A)$, $A\neq a_1$, $e_2(\_)$. Depending on the existence and values of attributes, an embedded expression is reduced to its innermost entity that calls another SH.
$e(A,\#)$. This semantic headers serves as a constraints for the representation of a complex query $e_1(A,\#)$, $e_2(B,\#)$ to deliver just an $answer(\#)$ instead of all pairs for e_1 and e_2. It works in the situation where e_1 and e_2 cannot be mutually substituted into each other.

This template of semantic headers serves as a criteria for completeness, consistency and coverage of the possible spectrum of meanings by a formalized entity. These

We conclude the section by the final definition what semantic headers are. Semantic headers of an answer are the formal generalized representations of potential questions. These representations are built, taking into account the set of other semantically close answers and relevant semantically close questions. Therefore, semantic analysis under the SH technique consists in the formal representation of a question and relating it to a fixed set of answers.

3 Basics of Default Reasoning

3.1 Default Rules

An abstract default logic (as proposed by Reiter 1980) distinguishes between two kinds of knowledge, usual predicate logic formulae (axioms, facts) and "rules of thumb" (defaults). Default theory includes a set of facts, which represent certain, but usually incomplete, information about the world (query, in our case); and a set of defaults that cause plausible but not necessarily true conclusions (because, for example, of ambiguity). Some of these conclusions have to be revised when additional context information becomes available. Consider the traditional example quoted in the literature on nonmonotonic reasoning:

$$\frac{\text{bird(X)}: fly(X)}{\text{fly(X)}}.$$

One reads it as "if X is a bird and it is consistent to assume that *X flies*, then conclude that X flies". As a matching rule default, it reads as follows. "If the query is about the bird, and it is consistent to assume that there can be a word in this query with the meaning of *flying,* then conclude that the query is about flying". If nothing contradictory can be derived from the other words of the query, it is natural to assume that this query is about flying of a bird.

As a traditional default, we obtain the assumption "Usually (typically) birds fly". Given the information that "Tweety is a bird" (in accordance with the history of nonmonotonic reasoning), we may conclude that he flies. But if we learn later that he cannot fly, for example, because he is a penguin, then the default becomes inapplicable.

Nonmonotonic reasoning technique helps us to provide the proper question-answering in this situation. Imagine that we have a set of documents answering questions about birds. Let us imagine there is a general answer, which presents information that *birds fly*, and contains the common remarks. Besides, there is also a specific answer, stating that *there is a bird Tweety, who is a penguin, lives at the South Pole, swims, but does not fly*.

If there is no processing of the translation formula, then the explicit restriction for the semantic representation will look like the following. The possible questions are *Do birds fly?*, *Tell me about birds, Which animals fly?, Do seagulls fly?, What do you know about eagle birds?, Is Tweety a bird?, Does Tweety fly?, Is Tweety a penguin?*; they will cause the same answer that birds fly, unless we mention the *bird Tweety*.

bird(X), not(X=Tweety) → respond(birds fly).

fly(X), not(X=Tweety) → respond(birds fly).

bird(Tweety) → respond(there is a bird Tweety, which is a penguin, lives at the South Pole, swims, but does not fly).

fly(Tweety) → respond(there is a bird Tweety, which is a penguin, lives at the South Pole, swims, but does not fly).

The problem with this approach is that it requires explicit enumeration of constraints in knowledge base. Each exceptional object has to be enumerated in the representation, which includes the properties of normal objects. When default rules come into play to modify the translation, this complication is avoided. The first answer does not have to contain explicit constraints that *X is not Tweety* or some other object; it will be reached if it is consistent to assume that *fly(X)*. The latter is verified using the default rule above and the clause that *Tweety does not fly*.

Here is a formal definition of default rule, as proposed in (Antoniou, 1997).

Main Definition. A default δ has the form $\delta = \frac{\varphi : \psi_1, .., \psi_n}{\chi}$, where $\varphi, \psi_1, .., \psi_n, \chi$ are closed predicate formulae and $n > 0$. The formula φ is called prerequisite, $\psi_1, .., \psi_n$ the justification, and χ the consequent of δ (pre(δ), just(δ) and cons(δ) correspondingly). A default $\delta = \frac{\varphi : \psi_1, .., \psi_n}{\chi}$ is applicable to a deductively closed set of formulae E iff $\varphi \in E$ and $\neg\psi_1 \notin E, .., \neg\psi_n \notin E$.

3.2 Examples of default rules

Defaults can be used to model situations in various domains. Below there are several examples, proposed by Antoniou.

1) Legal reasoning and justice

$$\frac{\text{accused(X): innocent(X)}}{\text{innocent(X)}}.$$

This rule models the presumption of innocence principle: if there is no evidence to the contrary, we assume the accused to be innocent.

2) Databases and Logic Programming

$$\frac{\text{true: not(X)}}{\text{not(X)}}.$$

This rule is the “Closed World Assumption”. According to it, a ground fact is taken to be false if it doesn’t follow from the axioms or cannot be found in the database. For example, if there is no 13.13 train in the schedule, than there is no such train in real world also.

3) Every-day life

$$\frac{\text{go\_to\_work(X): use\_bus(X)}}{\text{use\_bus(X)}}.$$

This rule describes usual morning behavior: if nothing contradicts to it, people go to the work by bus. Following such rule of thumb, I should go to the bus station and, most probably, take a bus. But it can turn out that drivers are on a strike, I am late or I want to go on foot, because the weather is fine. In that cases I definitely know that I cannot (or don’t want to) use the bus and the rule becomes inapplicable.

Let us now imagine that I follow classical logic rule

go_to_work(X)&not(strike)&not(late(X))&not(fine_weather)&…→use_bus(X).

In this case I have to list all the possible obstacles: *strike, hurry, fine_weather,* and so on. Then I have to obtain a lot of information to establish whether all the preconditions are true. Therefore finally I arrive at work at afternoon.

3.3 Operational Semantics

In this Section, informal description of operational semantics for default reasoning is proposed. Formal definitions, theorems and proofs can be found, for example, in (Antoniou, 1997).

The main goal of applying default rules is to make all the possible conclusions from the given set of fact. If we apply only one default, we can simply add its consequent to our knowledge base. The situation becomes more complicated, if we have a set of defaults, because, for example, the rules can have consequents, contradicting each other or a consequent of one rule can contradict to a justification of another one. In order to provide accurate solution we have to introduce the notion of **extensions** — current knowledge bases, satisfying some specific conditions.

Suppose *D* is a set of defaults and *W* - a set of facts (our initial knowledge base). Let Δ be an ordered subset of *D* without multiple occurrences (it is useless to apply default twice, because it would add no information). Denote *In(Δ)* a deductive closure (in terms of classical logic) of $W \cup \{cons(\delta) \mid \delta \in \Delta\}$. Denote also *Out(Δ)* the set $\{\neg\psi \mid \psi \in just(\delta), \delta \in \Delta\}$. We call $\Delta = \{\delta_0, \delta_1, ..\}$ a process iff for every *k* δ_k is applicable to $In(\Delta^k)$, where Δ^k is the initial part of Δ of the length *k*.

Given a process Δ, we can determine whether it's successful and closed. A process Δ is called successful iff $In(\Delta) \cap Out(\Delta) = \varnothing$. A process Δ is called closed if Δ already contains all the defaults from D, applicable to *In(Δ)*.

Now we can define extensions. A set of formulae $E \supset W$ is an extension iff there is some process Δ so that

1) Δ is successful,
2) Δ is closed,
3) *E=In(Δ)*.

Let us consider an example.Suppose *W* is empty and *D* is the set of

$$\delta_1 : \frac{true : not(tax\_fraud(X))}{not(tax\_fraud(X))}, \quad \delta_2 : \frac{true : tax\_fraud(X)}{check(tax\_police, X)}.$$

These rules describe a situation, when people are normally not assumed to commit tax frauds, but if it's consistent to suspect someone in a tax fraud, than tax police looks more thoroughly at his bills.

After we have applied the first rule, we extend our knowledge base by *not(tax_fraud(X))*:

$$In(\{\delta_1\}) = \{not(tax\_fraud(X))\},$$
$$Out(\{\delta_1\}) = \{tax\_fraud(X)\}.$$

The second rule is not applicable to $In(\{\delta_1\})$. Therefore a process $\Delta = \{\delta_1\}$ is closed. It is also successful, so $In(\{\delta_1\}) = \{not(tax\_fraud(X))\}$ is an extension.

Suppose now that we apply δ_2 first:

$$In(\{\delta_2\}) = \{check(tax\_police, X))\},$$
$$Out(\{\delta_2\}) = \{not(tax\_fraud(X))\}.$$

The rule δ_1 is still applicable now, so $\{\delta_2\}$ process is not closed. Let us apply δ_1 to $In(\{\delta_2\})$:

$$In(\{\delta_2,\delta_1\}) = \{check(tax\_police, X)), not(tax\_fraud(X))\},$$
$$Out(\{\delta_2,\delta_1\}) = \{not(tax\_fraud(X)), tax\_fraud(X)\}.$$

Now $In \cap Out \neq \varnothing$, so $\{\delta_2,\delta_1\}$ is not successful and $\{check(tax\_police, X)), not(tax\_fraud(X))\}$ is not an extension. This comes in accordance with our intuitive expectations, because if we accept $\{check(tax\_police, X)),\ not(tax\_fraud(X))\}$ to be a possible knowledge base, we assume that tax police checks thoroughly tax bills of all the people, not the suspicious ones.

The next example shows that there can be multiple extensions for one set of facts and default rules.

$$\delta_1 : \frac{dangerous\_job(X) : insure\_life(X)}{insure\_life(X)},$$
$$\delta_2 : \frac{young(X) : not(insure\_life(X))}{not(insure\_life(X))}.$$

These rules explain that people having dangerous job usually insure their lives and young people normally do not. Suppose now that we want to conclude something about a young man who has a dangerous job: *W={dangerous_job(X), young(X)}*. After the application of each default, the other one becomes inapplicable. So, both $\{\delta_1\}$ and $\{\delta_2\}$ are closed and successful processes. Thus, both *{dangerous_job(X), young(X), insure_life(X)}* and *{dangerous_job(X),young(X), not(insure_life(X))}* are extensions.

4 Pragmatic analysis with the help of default rules

Suppose *S* is a semantic representation of a query. Our intention is to transform *S* to *@S* - another well-formed semantic representation, which fits our narrow domain better. To do this the following algorithm is proposed: using a set of facts like *X is used in S* or *not(X is used in S)* as an initial knowledge base, we can apply default rules (created manually by knowledge engineers) to obtain all the possible extensions. These extensions contain facts about elements of @S (as well as initial statements). After doing that we can use the extensions to build up the @S representation. Note that S is indeed the most accurate and precise representation of the query meaning, taken separately. However, S needs to be transformed to match the domain better. This transformation is intended to eliminate the least important entities not to interfere with the most important ones, as well as to add the implicitly assumed elements.

There are two possible ways to use default systems to modify semantic representations:

- Application of defaults in the fixed order. It can be used when there are no conflicts in the consequents of the default rules.
- Building extensions for conflicting defaults. We employ the operational semantics of default logic in more complex situation, for example, when we have multiple ambiguous terms in a query (Fig.1).

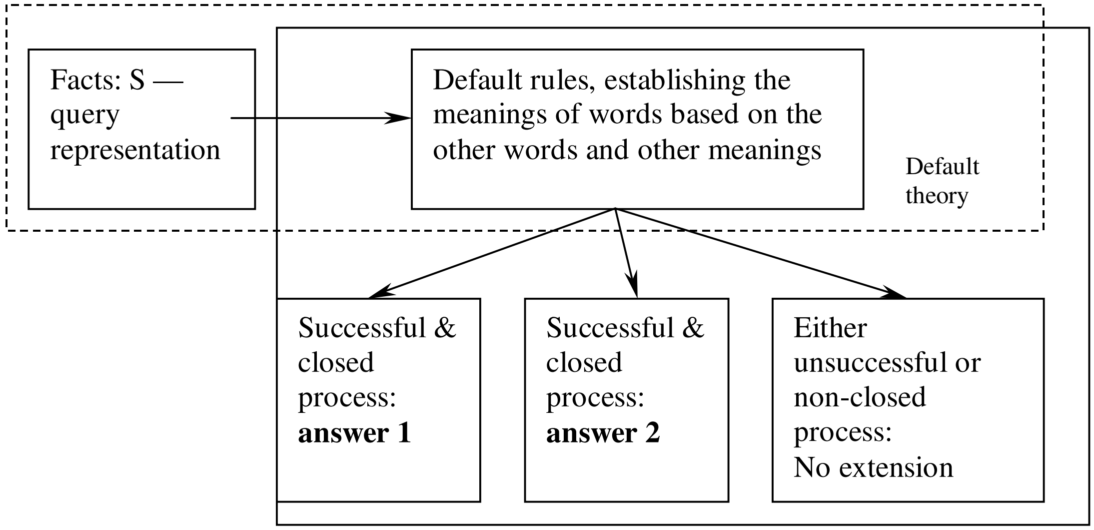

Fig 1. Processing a question as a set of facts in the default theory. For a given domain, default system includes the fixed set of default rules D, determining the meaning of the terms, and the set of facts about the word occurrence, specific to each query. Multiple processes are constructed in real time. If the process is successful and closed, it is associated with an answer via @S. Modified representations are formed by the obtained extensions of the default theory.

4.1 Trivial default rules

If we don't want to modify initial representation at all (S=@S), we can apply trivial defaults to each element of S:

$$\frac{\text{X is used in S: X is used in @S}}{\text{X is used in @S}}.$$

All the facts in our knowledge base are about the element occurrence: it is either used in the representation or not. In our rules we can write *X* instead of *X is used in S* and @*X* instead of *X is used in @S,* not causing any confusion. Sometimes we will speak about entities *X* and *Y* connected semantically or syntactically. In that case we would write *X(Y)*, which means that *X is used in S*, *Y is used in S* and *X and Y are connected by semantic or syntactic link*. In this form trivial rules look like

$$\frac{\text{X: @X}}{\text{@X}}.$$

We can use these defaults not only when we want to have S unmodified. In fact, we should apply trivial rules to every representation. Elements that are not affected by nontrivial defaults are simply moved from S to @S. As far as other elements are concerned, trivial and nontrivial rules can interact with each other, leading to specific extensions.

4.2 Nontrivial default rules.

4.2.1 Processing ambiguous terms

Ambiguous elements correspond to multiple default rules. Consider an example from an insurance domain.

The word *company* is ambiguous: it can refer either to *insurance company* or to *company where the customer works*. In default form it looks like

$$\delta: \quad \frac{\text{company(): @insurance\_company()\&not(@place\_of\_work())}}{\text{@insurance\_company()}},$$

$$\varepsilon: \quad \frac{\text{company(): not(@insurance\_company())\&@place\_of\_work()}}{\text{@place\_of\_work()}}.$$

If we have no other concepts in the query that can help us to make a decision on what *company* is meant, both rules (δ and ε) can lead to an extension. As a result, we have two representations $@S_\delta$ and $@S_\varepsilon$ and two corresponding answers.

But if the query contains some words, incompatible with one of the meanings proposed, then one of the rules does not lead to an extension. For example, if the query is about companies' rating (*What is the best company?*), than it is rather about *insurance_company* than about *place_of_work*. Mind, that this rule of thumb holds for narrow domain on insurance. If a person looks for a job-related domain, the decision should be the opposite one. And in everything-about-the-world domain we cannot create such rules at all. As a default , it looks like the following:

$$\alpha: \quad \frac{\text{company(rating): @insurance\_company()}}{\text{@insurance\_company(rating)}}.$$

Consequent of α contradicts to a justification of ε. That is why there can be no extension, created by means of ε: ε is inapplicable after α, α is still applicable after ε, but it makes the process unsuccessful.

Let us now consider more complicated example from the Individual Tax domain. In the table below some questions concerning *rent* are listed. They refer either to situation when someone rents out a property (rent1), or to situation when someone rents a property for business (rent2). In reality, the first one is more frequent, because *renting for business* has no effect on individual taxes. Nevertheless, both situations may appear in a question, so we need to propose the proper disambiguation here.

Questions about a person who rents out his/her property (landlord) and may have the maintenance expenses	**Questions about a person who rents a property to conduct his/her business (renter) and may consider it as a business expense.**
How to deduct rental expense?	
How to deduct rental expenses from my taxes?	
Can I include my rent expense in the itemized deduction?	
How to deduct rental expenses from my rent income?	How to deduct rent expense from my business income?
How much taxes should I pay on my rental income?	How to calculate my expenses of renting a property for my business?
My business is to rent out my property. How about taxes?	If I need to rent a property from the property management business, what are the tax issues?
How much tax do I pay when I rent out my property?	How to deduct rental expenses from the income of my business?
How can I deduct what I paid for repair from the income of my rental business	
If my business is to rent out my property, how can I deduct the repair expenses?	How to deduct rent from my salary?

We propose the set of defaults to analyze the concept *rent*. The following defaults lead to two extensions, when no specific words are used in the query.

$$\alpha_1 : \frac{rent : @\,rent1, not(@\,rent2)}{@\,rent1}, \qquad \alpha_2 : \frac{rent : @\,rent2, not(@\,rent1)}{@\,rent2}.$$

The second group consists of rules that block α_2 default. They are used when a query contains some concepts, helping us to conclude, that it is actually about renting out a property:

$$\delta_1 : \frac{rent, \_out : @\,rent1}{@\,rent1(\_out)}, \qquad \delta_2 : \frac{income(rent) : @\,rent1}{@\,income(rent1)},$$

$$\delta_3 : \frac{repair(rent) : @\,rent1}{@\,repair(rent1)}, \qquad \delta_4 : \frac{business(rent) : @\,rent1}{@\,business(rent1)}.$$

The third group consists of rules that block α_1 default. They are used when a query contains some concepts, helping us to conclude, that it is actually about renting something for business:

$$\varepsilon_1 : \frac{salary, rent : @\,rent2}{@\,rent2}, \qquad \varepsilon_2 : \frac{rent, business(X), X \neq rent : @\,rent2}{@\,rent2}.$$

Note that some rules $(\varepsilon_2, \delta_2, \delta_4)$ use information not only about words in a query, but also about their semantic and syntactic connections.

Below we describe how some queries can be analyzed with the help of proposed rules.

1) *How to deduct rental expense?*

The initial representation for this query is deduct(expense(rent())). Only α_1 and α_2 defaults are applicable to it:

In($\{\alpha_1\}$) =$\{$deduct,expense,rent,@rent1$\}$,

In($\{\alpha_2\}$) =$\{$deduct,expense,rent,@rent2$\}$.

After the application of α_1, the α_2 rule becomes inapplicable and vice versa. So, both $\{$deduct,expense,rent,@rent1$\}$ and $\{$deduct,expense,rent,@rent2$\}$ are extensions. Therefore, the system will give two answers to the question *How to deduct rental expense?*.

2) *How to deduct my rental expenses from my rental income?*

The initial representation for this query is deduct(expense(rent()),income(rent())). The rules α_1, α_2, and δ_2 are applicable to it. The only successful and closed processes are $\{\alpha_1, \delta_2\}$ and $\{\delta_2, \alpha_1\}$. They lead to the extension $\{$deduct, expense, rent, income(rent), @income(rent1)$\}$. Consider a process tree for this query (Fig.2).

So, if the query contains no information that can help us to make disambiguation, default rules lead to multiple extensions, corresponding to multiple meanings of ambiguous elements. If the query contains some specific concepts, our rules lead to single extension and the system proposes a single answer. We believe that the optimal solution for Question-Answering system is to provide multiple answers in case of multiple extensions (i.e. in case there are no words indicating particular meaning).

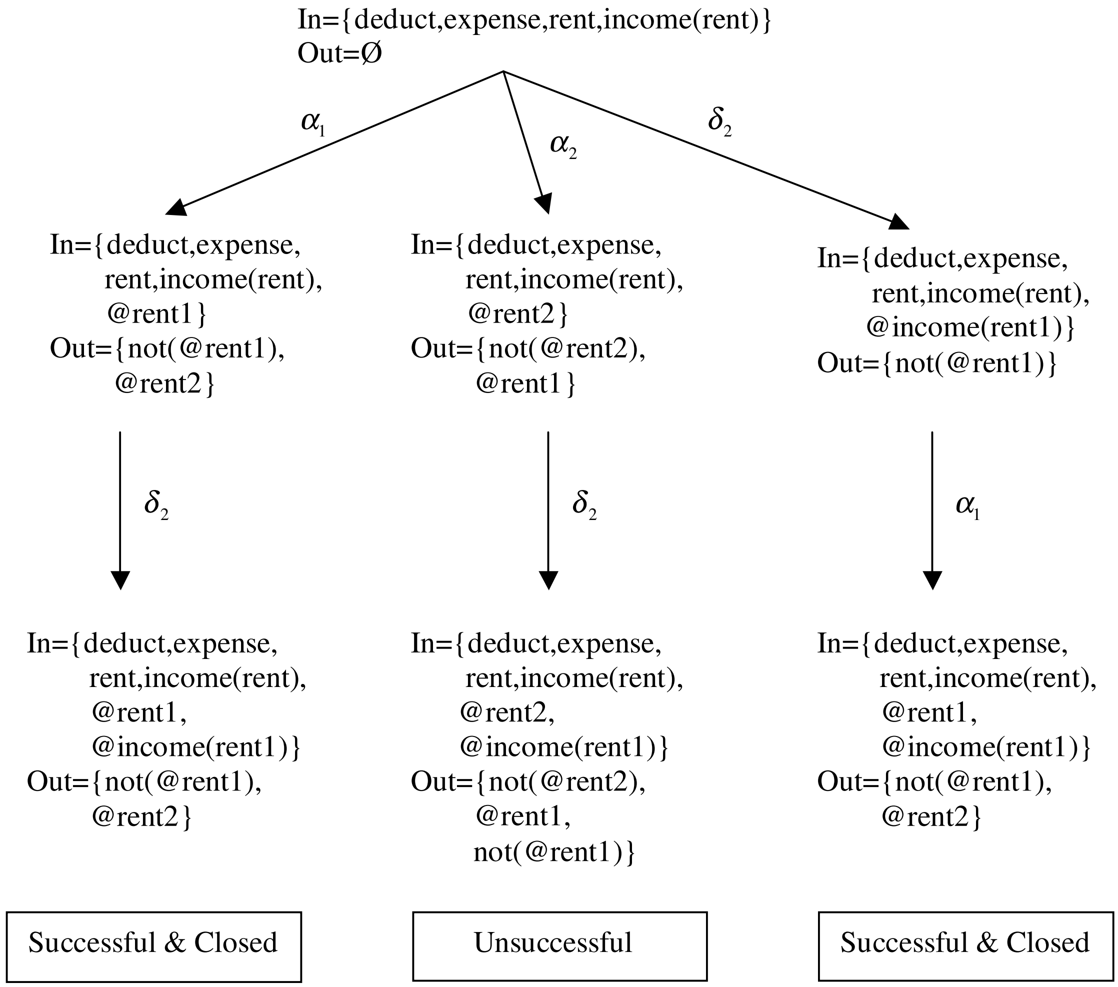

Fig.2 Process tree for "*How to deduct my rental expenses from my rental income?*"

4.2.2 Adding new information

Default rules can help us to add new elements to semantic representation, using the basics of commonsense knowledge. Frequently, it is necessary to insert an entity which links the specific attribute with the more general concept, which occurs in a query. For example, if the query contains word *school*, it is likely to be about education. So *Can I deduct my school expenses?* should be interpreted as deduct(expense(education(school()))). We propose the following default rule:

$$\frac{\text{school(): @education()}}{\text{@education(school())}}.$$

This rule should be accompanied by the clause, presenting situations of justification inconsistency. If we have a query *Can I deduct my donation to a catholic school?,* it is rather about a donation than about education. The following clause provides the proper solution:

@education :- deduct(Attribute),not(Attribute =expense).

This rule expresses the fact that if there are attributes of *deduct* in a query, then the query is likely about these attributes (*donation, construction, moving*, etc.). If *Attribute* is rather general (*expense*), then the clause fails and justification stays consistent.

Let us consider another example, when we add an entity (*donation*) to the query representation to link the specific word (*church*) to a general concept (*deduct*): *Can I deduct what I gave to church?*. We have the similar default rule.

$$\frac{church() : @\, donation()}{@\, donation(church)}.$$

Furthermore, what would happen if we have more than one specific words, potentially connected with the general concept, as in a query *Can I deduct my church school expenses?* Occurrence of multiple attributes may require analysis of conflicting defaults (operational semantics). If the justification failure clause for *@donation* is similar to proposed above, we have two extensions for this query - {…, @donation} and {…,@education}. But if we consider *education* to be more general than *donation*, the justification failure clause looks like

@donation:-deduct(Attribute),not(Attribute=expense),not(Attribute=education),

and the only possible extension is *{…, @donation}.*

Default technique indeed brings in the significant advantages in the processing of poorly structured (NL) knowledge representation. If we do not want to use default technique for our *school* example, then either we are not able to substitute *school()* in the formula at all, or we have to use *deduct(expense(school()))* representation. In the first case we lose important information and obtain a wrong answer. In the second case we have to provide the possibility to substitute *university, college, institute*, and other terms in *expense()* as well. As a result, domain becomes less structured and query processing loses efficiency.

As we have seen, default rules can help us to improve domain hierarchy. It affects not only the performance but also the quality of query processing. Let us imagine that we have no information about *school expenses*. Instead, we know how to treat *educational expenses* in general. If the system cannot connect *school* with *education*, users get wrong answers because the formula *deduct(expense(school()))* is interpreted literary.

4.2.3 Eliminating parts of semantic representation

Semantic representation can contain more or less specific entities. The most general parts can lead to vague answers. For example, the query *How to file time extension for my tax return?* has a representation *tax(return)&extension(file).* It leads to two answers: about *tax returns* in general and about *extension of time for filing*. It's obvious that the first part is superfluous and must be eliminated. In fact, in tax domain we can interpret *extension* only as extension of time to file, and not, for example, as extension of computer file. Therefore, *tax(return)* can be deleted from the formula without loss of information. We first present the naïve stand-alone rule

extension(time): tax

@extension()

This rule can be read as follows: if a query mentions *extension of time* and it is consistent to assume that it is about *tax*, then the query is indeed only about *extension*. Although this rule follows our intuition better, it is not appropriate for interaction with other rules, because it does not actually eliminate *tax* from the current knowledge base. Therefore, we suggest the following rule instead:

$$\delta: \quad \frac{\text{extension(): not(@tax())}}{\text{not(@tax())}}.$$

It is read as follows: if a query contains *extension,* then *tax* should be eliminated.

Mind, that elimination of superfluous elements is connected with the disambiguation problem. If we have no idea about the topic of the query, both meaning (filing time extension and computer file extension) are probable, so we need additional information to apply our analysis to. However, if we provide the question-answering for a narrow domain, only one meaning is expected and the other is exceptional. That's why this additional information becomes superfluous.

To comment on the rule δ, we present the simplified default "without *extension*", which would mean that we can always eliminate *tax* from a query.

$$\frac{\perp : not(@\,tax)}{not(@\,tax)}.$$

However, this rule would misinterpret the queries *What is tax?*, *Can I deduct the tax I paid last year?* and others.

Depending on the order in which rules of eliminating superfluous parts and trivial rules are used, we can obtain several extensions containing more ore less elements of the initial structure, because these two types of defaults make each other inapplicable. If we begin with trivial rules, the representation remains unmodified. Otherwise, some entities are eliminated. The order, in which defaults are applied, can be chosen by knowledge engineers depending on the task specifics.

4.2.4 Nontrivial rules and domain classification graph

Domain structure can be represented as an oriented graph with nodes corresponding to concepts and edges showing their interdependences (possible substitution order). This graph should advisably be close to a tree and, in any case, contain no oriented cycles.

Semantic representation is mapped to a subgraph and defaults can be defined as operations on this subgraph. Rules, introduced in 4.2.1 – 4.2.3 are described below in terms of graphs. Words printed in roman and solid edges show the concepts (and their interdependences), which belong to query representation subgraph. Words printed in italic and dotted edges show the concepts, which do not belong to this subgraph (but still are parts of domain classification graph).

1) Making disambiguation
   These rules are applicable, when query representation contains ambiguous word and therefore corresponds to a subgraph with a node connected with many others. It looks as if two or more nodes were merged. When we apply defaults, we split those nodes in two or more and then try to build up new representations for each node separately. Consider an example for *What is the best company?*.
   The term *place_of_work* was eliminated, because it's not connected with the rest of the query, while the alternative term (*insurance_company*) is linked to *rating*.

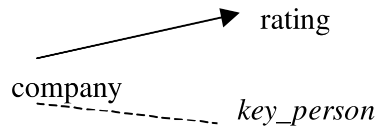

Fig. 3. Transformation of query representation subgraph for *What is the best company?*

2) Adding new information.
   These rules are applicable to queries, corresponding to disconnected subgraphs. They consist in adding an intermediate node and accompanying edges to make a subgraph connected. The reader may consider an example for *How to deduct my school expenses?*.
3) Eliminating superfluous parts.
   These rules are also applicable to disconnected subgraphs. They consist in eliminating parts, placed on high level in the domain hierarchy, if all the other elements are placed under them (and not aside). We may consider an example for *How to file time extension for my tax return?*.

5.0 Using default rules for keyword semantic headers

Usually, using pure keyword analysis for Q/A is considered as rather primitive approach. Representing a query as a Boolean combination of keywords is insufficiently expressive, if there are more than two entities (with attributes) in a query. To obtain high Q/A accuracy under the shallow syntactic processing, it is necessary to apply some pragmatic machinery to sort out improper query translation hypothesis (Grishman 1997, Ng et al 1997, Boros et al 1998) . We use the default logic to build a set of keywords for an answer such that these keywords will be directly matched against the query keywords, providing satisfactory accuracy. In this section, we show that the use of nonmontonic reasoning versus pure Boolean keyword expressions can compensate sophisticated semantic machinery and perform a proper Q/A for queries with multiple entities.

In this section, technique of semantic headers is reduced to one with keywords semantic headers, consisting from the list of entities and their attributes, corresponding to the atoms of the expressions for standard semantic headers. Under the semantic header technique, the question-answering problem is posed as finding the most relevant class (answer) for a query, using the pre-built set of the canonical queries. We reformulate this problem as a *separation* of answers, using the representation of these canonical queries as keyword lists (keyword semantic headers). In contrast to the above presentation of default reasoning for NLP, appropriate for a generic semantic analysis, this section introduces the default reasoning approach, oriented to semantic header technique.

We use the following diagram to position our approach among the other NL systems in respect to the degree of contribution from syntactic, semantic and pragmatic components (Fig.4). The traditional SH system (dotted line), which requires manual coding, is less sensitive to syntactic misunderstanding and requires pragmatic machinery in a lower degree, than the keyword SH–based system (solid line). At the same time, keywords SH technique cannot rely on a sophisticated representation language for query or domain because of the desired automatic annotation features, therefore, more advanced pragmatics (default reasoning) is required.

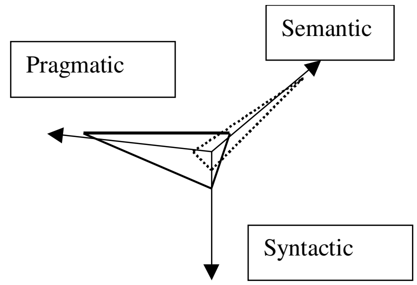

Fig.4: the diagram of feature comparison between the semantic headers and keyword semantic headers approaches.

Below we will present two answers and propose the way to distinguish them. Then we demonstrate how default rules are used to transform an input NL query to a form that can be matched against the keyword semantic header of an answer. Building default rules is followed by their verification, given the set of semantically diverse queries, and construction of the clauses for checking their justifications.

Further on, we show how to perform the automatic annotation of new answers, where the constructed default rules build the keyword semantic headers for these answers, together with the new defaults for them.

5.1 How to distinguish answers using the keyword annotation

As an example, let us have the following two answers that need to be assigned by the distinguishing set of keywords.

*You can get the information about your tax return from the IRS website. The documents are available for download as the files with extension *.pdf. From time to time, the website may experience technical difficulties.*

Everyone is encouraged to file the tax return on time. If you experience financial difficulties, you can file an extension of time for your tax return. Detailed information is available for download at the IRS website.

In terms of the keywords, these paragraphs are almost identical, but the topics they address are quite different. Such a situation takes place frequently, when a Q/A system goes deeper into details within a vertical domain, trying to separate the answers consisting from the same keywords.

We ignore the information about syntactic relations between words in sentences and consider the total list of keywords for these paragraphs. As a result, we see that they are different by domain-invariant words only (*need, encourage, technical*).

{download, tax, return , irs, website, file, extension, time, experience, technical, difficulty},
{encourage, file, tax , return, time, experience, financial, difficulty, file, extension, time, tax, return, information, irs, website}.

It is evident, that though the paragraphs above have the similar keywords, there is the critical difference in their importance for the meanings of these paragraphs. The main question of this section is how to properly annotate these paragraphs, based on the most important keywords and ignoring the least important ones. For the first paragraph, one of the expected annotating sets of keywords (semantic header) is *information, tax, return.* For the second one, we have *file, extension, time, tax, return.*

Pure keyword extraction system would not be able to distinguish these answers. Suggested application of the default reasoning is based on the specific model of how the textual information is distributed through the sentences in a paragraph.

Default rules are intended to link the approximate, deviated meaning, expressed by a *prerequisite*, to the exact meaning, expressed by *consequent*

prerequisite: justifications
----
consequent

where *prerequisite, justifications and consequent* are the lists of keywords. *Prerequisite* includes the list of keywords that may occur in a sentence, *justification* specifies the wider topic (context of this sentence), which is evaluated by processing the current and the other sentences in a paragraph. *Consequent* contains the combination of keywords that is the best representation of meaning of a portion of text (sentence in an answer or a query). Intuitively, an arbitrary way of asking about a topic is expressed (potentially matched) with the prerequisite, and the exact topic is expressed by consequent.

One of the natural way to distinguish the answers is to assign different justifications to the keyword pair *file, extension* , which will serve as a prerequisite to the pair of default rules.

file, extension: download
----
file, download, tax

file, extension : time
----
extension, time, tax, information

Consequences of the first and second rules present the essential idea of the respective answers. These defaults are natural from the prospective of the domain knowledge representation, but they are not helpful to link a possible question with the most relevant canonical one.

5.2 Using defaults to transform a query into a semantic header

To obtain an answer given a question, we would expect to employ the default rules in the following way. A question corresponds to prerequisite and justification, and an answer is associated with the keyword semantic header (list of keywords) of the consequence:

question

SH of answer

How can a question be divided into prerequisite and justification? The prerequisite includes the keywords, unique for the answer, and the justification contains the rest of words. We call the default alien, if there are no common keywords in its prerequisite and consequence.

Let us consider the first sentences of the answers and build the templates of the default rules:

download, information, tax, return , irs, website
----
file, download, tax, information

encourage, file, tax, return, time
----
extension, time, tax
To construct the default rules from their templates, we form the justification from the keywords, not participating in the separation between answers (not occurring in semantic headers). These words are simultaneously frequent and essential for given Q/A domain.

download, information, website, irs: tax, return
----
file, download, tax, information

time, encourage: file, tax, return
----
extension, time, tax

Since the words *tax, return, file* occur in both answers, they do not have to be matched against an input query to provide a specific answer. However, these words are necessary to obtain the evidence of the proper context (subdomain) for the words from prerequisite. Speaking in other way, prerequisite performs the answer separation on the lower (specific) level, and justification does on the higher (general) level.

Learning stage of automatic annotation consists from building the default rules, given the pre-composed semantic headers, associated with an answer. A template of a default rule is built for each semantic header and for each sentence in an answer. A template is convertible into a default rule if there is at least a single common word in prerequisite and in consequent. Evidently, if we allow an overlap just between justification and consequent, it would mean that we allow very general answer to be activated that can cause potential conflict with the other more specific answers. The procedure of conversion a template to a default rules is based on intuition that a potential question can be close to an answer sentence, however, not all the answer sentences need to be represented by semantic headers.

Template of default rule → default rule

answer sentence		*prerequisite (unique words) : justification (assumed words)*
-------------------	→	---
semantic header		*words of the existing semantic header*

How do defaults convert an input query into a semantic header? For a given query, the system looks through semantic headers to match the maximal number of words and through default rules to match the maximal number of words against their prerequisite and verifies the consistency of justification. If the number of matched keywords of a default rule prerequisite is higher than that of a semantic header, the former is chosen. Note that multiple answers can be delivered. We present the algorithm of matching the keywords of input query against semantic headers (SH) and default rules (DR) in accordance to the following expression, where *cardinality* means the number of keywords

$$query \leftrightarrow keywords: \max \{cardinality(keywords): \; keywords \in SH \cup (keywords \in (prerequisite(DR): \; justification(DR) \; holds) \}$$

A *query* is mapped into the set of *keywords* such that this set delivers the highest number of matched words for the totality of all semantic headers and default rules with consistent justifications.

To raise the relevancy of these computations, we can suggest to take into account the number of unmatched keywords in a semantic header and default rule justification as well.

The lesser the number of unmatched keywords, the higher the evidence of relevancy between a query and a keywords semantic header.

To discover that the justifications of default rules are consistent, the system needs to verify the following:

- While processing a query: a clause, expressing the justification inconsistency, fails.
- While processing a sentence within an answer, subject to annotation:
  a) justification words are present in a given or accompanying sentences in an answer;
  b) there are no explicit negations of the justification atoms, obtained by the other default rules that have been applied to the other sentences in an answer;
  c) a clause, expressing the justification inconsistency, fails.

For example, a query *Does IRS encourage taxpayers to timely respond to the audit notice* should not activate the second default rules above. In the *tax* domain, the most common action is filing a tax return that is assumed by default in the rule. If one speaks about such action as *responding to an audit,* it is legitimate but not most common, therefore the justification has to be found inconsistent. We need to add the clause in accordance to the following scheme

justification_inconsistent(tax) :- [tax, Action], Action ≠ file.

It means that the rule is about *encouraging to file tax on time* and not about encouraging to perform some other, less frequent action on time.

The predicate *justification_inconsistent* is called for each keyword in justification. If there is no special clause for a keyword, this predicate fails:

justification_inconsistent(A) :- fail.

Negation may appear in a prerequisite in the following situation. Let us have a pair of answers, one is for an entity *e* and the set of its attributes, and another for this entity *e* and a particular attribute *a*. Then the first answer can be assigned by a default with prerequisite, containing this entity and the justification, containing the negation of this particular attribute. The second default will not have such constraint in its justification:

$$\frac{e,\ldots: \text{not } a, \ldots}{\ldots}, \quad \frac{e,\ldots: \ldots}{\ldots}$$

For example, if we have a general answer about deduction of everything except mortgages, and a specific one about mortgage deduction, the following rules will be in use:

$$\frac{deduct,\ not\ (\ mortgage): tax}{general\_deduct}, \quad \frac{deduct,\ mortgage, \ldots: tax}{deduct,\ mortgage}$$

5.3 Construction of semantic header for new answers, based on built default rules

Using default rules to link a potential query with canonical ones, we approach question-answering and automatic annotation from the uniform positions (Fig.4). An answer that is subject to annotation is considered as a set of sentences, each of which can serve as a template for canonical question to this answer. For each sentence in an answer, we search for the most adequate pre-built default rule in a sense of the best match of this sentence and the consequent. Semantic header is constructed with a new default rule, assigned to the annotated answer. We use the correspondence diagram between the existing and new defaults to show how they are interrelated.

$$\frac{prerequisite:\ justification}{consequent} \rightarrow \frac{answer\ sentence:\ justification}{canonical\ question}$$

Unmatched words in the answer sentence can participate in the formation of semantics as well, if the structure of semantic link between the pair matched-unmatched words in the answer sentence is similar to that of two words in a consequent of the used default rule. We can plot the correspondence diagram, taking into account replacement of *unmatched* by *newTerm*, if *matched* is an entity, and *unmatched* together with *newTerm* are its attributes.

$$\frac{\textit{matched, unmatched, …: justification}}{\textit{matched, unmatched, …}} \rightarrow \frac{\textit{matched, newTerm, … : justification}}{\textit{matched, newTerm, …}}$$

As an example, we construct a semantic header with default rule for the answer, containing the following sentence:

…You can obtain information about filing your tax return by calling IRS 1-800-…

$$\frac{\textit{download, information, website, irs: tax, return}}{\textit{file, download, tax, information}} \rightarrow \frac{\textit{call, information, irs: tax, return}}{\textit{file, call, tax, information}}$$

We replace *download* by *call,* because both *download* and *call* are the attributes (actions) of the entity *information* that is essential for the answer.

Evidently, we need a sufficient number of pre-built default rules to adequately annotate the new answer in the same domain, consisting mostly from the known terms. Therefore, the annotation training (Section 5.1) needs to provide the reasonable coverage of the domain and its terms to prepare it for automatic annotation.

To conclude the section, we outline the semantic features of the suggested technique. Generally speaking, default rule are used to convert a translation formula in case it does not match existing semantic headers. In other words, default rules are designed to cover the space of queries, do not directly represented by semantic headers. As we explained while introducing SH technique, an individual semantic header covers the totality of semantically similar but syntactically different queries. The set of default rules with the consequence that matches a semantic header covers semantically different, but pragmatically similar questions.

Conclusions

Reasoning with exceptions is an inherent feature of natural language. Domain-independent semantic analysis skips the exceptions, and therefore the matching unit becomes the essential one to handle the domain-specific deviations of the concepts.

We implement the machinery of default logic to handle exceptions. Default reasoning is a convenient means of programming the data analysis rules, presented separately for typical and special situations. Typical rules are applied first, and the rules for unusual situations are called only upon request. Thus, default logic serves as a reasonable structure on the set of rules in terms of software coding styles, when a knowledge engineer represents the frequently and infrequently accessed pieces of knowledge separately. As a result, modifications of the latter component (enumeration of the exceptions) do not affect the domain skeleton.

NLP offers unique possibilities to apply the nonmonotonic reasoning. Advanced theoretical investigations have been carried out in order to build the formal background for nonmonotonic intuition and a series of counter-examples. However, the number of practical applications of nonmonotonic reasoning is far below the number and value of the theoretical results.

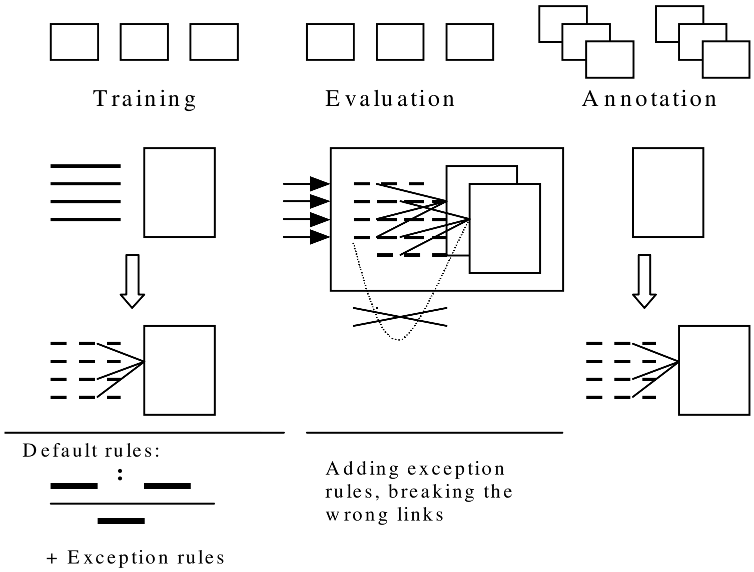

Fig.4: Architecture of the keywords annotation system. Firstly, on the training stage (on the left), the answers with pre-coded queries are automatically assigned with the default rules, based on the sentences in these answers. Clauses for exceptions are encoded manually because they reflect the domain knowledge that is not represented formally for question-answering.

On the second (verification) stage, the association between questions and answers by means of default rules and semantic headers is tested against an extensive set of queries to restructure or remove the wrong links. On the third stage, semantic headers and default rules are built automatically, using existing default rules and the sentences of answers to be annotated.

As we have seen, default rules technique can help to perform pragmatic analysis in processing NL query and therefore improve question-answering quality for narrow domains. Several types of defaults are proposed that can be used in order to process ambiguous concepts, apply commonsense knowledge and eliminate superfluous terms. Default reasoning for the term disambiguation displays some similarity to the popular context-based approach (McCarthy 1993, Buvac 1996).

A disadvantage of the traditional reasoning to represent the semantic and pragmatic rules is that it symmetrically handles frequent and infrequent situations. Default system, instead, always assumes that the typical situation occurs, unless it is not consistent for a variety of reasons to apply this assumption. Considerations of these “reasons” require much more detailed domain analysis. Hence, application of default reasoning helps us to structure the domain representation, firstly considering the main rules and then forming the additional ones.

A reader can evaluate Q/A capabilities of suggested system, for example, in the *tax* domain (see the web sites of HRBlock.com and CBSMarketWathc.com; domains are developed by Anserity.com). However, not all the default reasoning functionality has been implemented commercially.

This study demonstrated that even if an NLP system performs “ideal” syntactic and semantic analyses and is combined with the efficient knowledge representation, coded in sufficiently expressive language, the resultant Q/A system might still be unsuccessful. A pragmatic component inevitably needs to come into play, and the default reasoning seems to be a solid background for the pragmatic framework.

References

1. Antoniou, G., 1997. Nonmonotonic reasoning. MIT Press, Cambridge, London.
2. Galitsky, B. 2001. Semi-structured knowledge representation for the automated financial advisor. *14th Intl Conf on IEA/AIE,* Budapest, Hungary.
3. Galitsky B., 2000. Technique of semantic headers: a manual for knowledge engineers. *DIMACS Tech. Report* #2000-29, Rutgers University.
4. Galitsky, B. 1999 Natural Language Understanding with the Generality Feedback. *DIMACS Tech. Report 99-32*
5. Reiter R., 1980. A logic for default reasoning. *Artificial Intelligence Journal,* N° 13, pp. 81-132.
6. Grishman , R.1997. Information extraction: Techniques and challenges. In M.T. Pazienza, ed. *Information extraction: a multidisciplinary approach to an emerging technology*. Springer-Verlag.
7. Ciravegna, F and Lavelli, A. 1999. Full text parsing using cascades of rules: an information extraction perspective. In *Proc 9th European ACL Conf.,* Bergen Norway.
8. Romacker, M., Markert, K., Hahn, U., 1999 Lean semantic interpretation In *Proc IJCAI-99*, 868-875.
9. van Eijck, J. & Moore, R.C.1992, Semantic rules for English. In *Hiyan Alshawi, ed. The core language engine* 83-115 MIT Press.
10. Creary, L.G, & Pollard C.J.1985 A computational semantics for natural language. In Proc ACM-85 172-179.
11. Sondheimer, N.K., Weischedel R.M., and Bobrow, R.J. 1984. Semantic interpretation using KL-One. In *COLING-84* 101-107.
12. Hirst, G. 1988. Semantic interpretation and ambiguity *AI* 34(2):131-177.
13. Buvac, S. 1996 Quantificational logic of context. In *Proc 13th AAAI Conf* Menlo Park CA.
14. McCarthy J. 1993 Notes on formalizing context Proc IJCAI –93 Morgan-Kaufmann.
15. Ng, H-T, Goh, W-b and Low, K-L. (1997) Feature selection, perceptron learning and a usability case study for text categorization. In Belkin, N.J., Narasimhalu, A.D.,Willett, P., eds. Proc 20th Annual Intl ACM SIGIR Conf on IR, Philadelphia, pp67-73.
16. Boros, E., Kantor, P.B., Lee, J.J., Ng, K.B., Zhao, D.(1998) Application of logical analysis of data to TREC6 routing task *Text Retrieval Conf-6* , NIST Special Publication .