DIMACS[1] Technical Report 98-28

June 1998

A formal scenario and metalanguage support means to reason about it

by

Boris Galitsky
Department of Mathematics, Rutgers University, New Brunswick, NJ 08903
galitsky@dimacs.rutgers.edu

[1]DIMACS is a partnership of Rutgers University, Prinston University, AT&T Labs, Bell-core, Bell Labs and NEC Research Institute.

DIMACS is an NSF Science and technology Center, funded under contract STC-91-19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

We build the reasoning tool of metalanguage support (MS) with the embedded mechanism of the invention of predicates and metapredicates for the comprehension and synthesis tasks. This tool, based on metalanguage representation with weakened soundness, gains the inference flexibility to represent the formal scenarios (for example, a limited class of "logical" anecdotes, derived from NL). We introduce the concept of formal scenario as an alternative to the traditional axiomatic method (logical program) for the practical applications and develop specific MS means to perform the reasoning within a formal scenario. The anecdote scenario is an appealing object of study in the logical programming because it reflects the top level of human intellectual activity on one hand and is rather compact on the other.

We present such applications as scenario comprehension and generation, genetic algorithm of MS formula modifications, NL aspects of scenarios and modeling of the psychological disorder with corrupted pretending and believing concepts (autism). Such MS capabilities for scenario representation as reasoning about action, change and belief, temporal and spatial reasoning, are illustrated.

MS is based on the rules of metalanguage transfer (inconsistency shift, variability shift and assumption enforcement), which connect the object level with the metalevel and introduce a number of ways to construct the scenario explanation predicates.

Introduction

In this study we introduce the concept of **formal scenario**, a specific reduction of the traditional axiomatic method towards the reasoning framework of higher flexibility for the practical AI applications.

Traditional axiomatic method starts from the basic minimal number of facts (axioms), accepted as the results of the extended reasoning experience in a particular domain. Then the domain-independent inference rule machinery is applied to yield the infinite number of theorems. Non-standard logical systems possess either specific inference rules (for example, intuitionistic, three - valued) or specific axioms (modal, temporal, etc.). These systems obey the traditional axiomatic method and are parameterized by the properties of the set of derived theorems (validity, soundness, completeness, etc.).

The practical reasoning typically occurs in the restricted problem domains, and the number of facts is rather limited. An applied reasoning system is rather intended to deductively link existing facts with each other and reduce them to a canonical representation, than to characterize the properties of the set of possibly derived facts (theorems). We suggest the formal scenario as a structural unit of reasoning about a problem domain. This unit contains the agents, their properties and knowledge about this problem domain, which is necessary and sufficient to reason about the interactions of these agents.

We build the metalanguage deductive means (metalanguage support, MS), capable of construction of the set of **explanation** predicates and metapredicates to construct a scenario (a specific logical program) and to link its components. A necessity to build the explanation predicates arises in the comprehension tasks (natural language (NL) scenarios [12], scene analysis [8], semantic analysis for speech recognition, etc.) and mathematical reasoning[6,7,16], when a logical program, containing detection rules, is modified to match a new problem domain[1]. Scenarios are intended to be the structural units of the knowledge representation for a multi-agent system, distributed in space, time and associated with the specific action and reasoning machinery of the agents.

The MS learning capability is a useful addition the first-order learning of concepts (for instance, learning from interpretation, entailment, satisfiability [2,3,4]). We call the **metalanguage learning** the process of deriving metapredicates (or metarules), which describe the specifics of object-level rule modification, occurring in a reasoning session. We fix the language and the set of rules (clauses) for a particular PD (this is an initial universe in model-theoretic viewpoint). Then, we set another problem domain (new universe) for this set of clauses and modify them to obtain the best description (in terms of minimum modifications and maximum subset of the new universe, satisfying the modified clauses). We state that the metarules, reflecting the clause modifications, are much more important information about PD than the resultant clauses (compare with [9,10,11]).

We highlight the following five cases when the learning in metalanguage is helpful for a reasoning system:

1. Higher expressive means (in respect to an object-level language) are required to present concepts for a PD.
2. Learning procedure is necessary to be implemented in more general terms than the

description of a particular PD.
3. Learning results should be stored in a compact and canonical way, which allows transfer from one PD to another.
4. Metalanguage implementation of learning is designed to be invariant in respect to the local inconsistencies (deviation of meanings of some concepts, which do not affect the whole system).
5. Metalanguage implementation of learning is capable of ignoring the infrequent occurrences and/or meanings of some object-level concepts (probabilistic filtering).

If a reasoning system is built to function in the fourth and fifth cases, we consider this system as being **consistent to low-level noise.**

We present a series of examples where the explanation predicate construction is a key task for both comprehension and synthesis problems. To construct a formal anecdote (the special kind of scenario), we need to derive a metapredicate, which is capable to explain how the set of clauses for "traditional" behavior of the anecdote agents turns into the "emergent" anecdote result ("punch line"). The mechanism of deriving such linking metapredicate, applied to the anecdote skeleton, leads us to the posing of the task of invention of a set of object-level clauses for the scenario (anecdote, joke) generation.

This is the computational study in the sense that there are no formal proofs, but suggested axioms and metalanguage inference rules are employed in the strongly distinguishing PDs and are verified to be adequate to solve the posed tasks. The anecdotes, referred as logical by human experts, are recognized; the synthesized formal scenarios are considered as anecdotes by these experts; an artificial ant significantly increases the complexity of his behavior; a problem domain becomes extendible by means of natural language; the human autism disorder is effectively represented. To achieve this, we represent each of these domains as formal scenarios instead of analyzing the properties of infinite number of theorems, deducible from the axioms for these domains.

The presentation could be considered as a guide of metapredicate construction under scenario formalizations in various PDs. Meta-programming technique is illustrated throughout the text for applied (semantic, having NL interpretation) and pure syntactic (formula transformation) metapredicates.

We start with the simple example of the common-sense reasoning for the multi-agent scenario in Section 1.1. After we present an anecdote example in 2.1, we introduce the anecdote model (2.2) with the help of a three-level language system. It includes the predicates, describing the agents' properties, the metapredicates, describing the agent's states and actions, and the meta metapredicates, describing the links between the logical programs (inference objects) of the anecdote components. The predicate invention occurs at each level in accordance to the specific mechanism. The presentation of metalanguage support (MS) reasoning, its motivations and corollaries in Section 3 are followed by the posing of the anecdote (scenario) synthesis task and the example of synthesized scenarios for the Internet advertising (4.2).

Genetic algorithms, applied to MS formulas (Section 5), illustrate the formation of the behavior rules, adjusted for a particular problem domain. Section 6 is devoted to the scenario models of understanding NL inquiries and definitions for extendible problem domain. The last application of the study is the autism scenarios, designed to represent specific believing and pretending phenomenology (Section 7).

A reader can achieve three levels of familiarity reading this paper. The superficial level contains the series of examples of the formal scenarios and MS reasoning about them (Sections 1,2,7). The deeper level would include comprehension of all applications of the formal scenario (Sections 4,5,6 in addition) and introduction to the metalanguage inference rules (Section 3). More involved reader could consider the technical details of metalanguage programming, presented in Sections 2,3 and 5.

The clauses are presented in MS extension of the standard PROLOG. We use the `Courier` font to present the clauses, interpretable in our MS PROLOG implementation and *Italic* to present more general clauses, ignoring some details. Variables are capitalized; in the clauses, syntactic metapredicates are shifted right relatively to the applied ones.

1 A trivial example of the common-sense reasoning

We start our presentation with the multi agent scenario. The agents are capable of reasoning, based on the limited knowledge about the world and each other, and making decision concerning their behavior.

The Wolf and the Bush

A small girl is walking through a forest to visit her grandmother, and she passes a bush behind which a Wolf is hiding, planning to pounce out and eat her. Just as she gets close, however, the Wolf hears the singing of the woodcutters as they start work nearby, so decides to stay hidden and not pounce on the little girl after all. The problem is to explain why the Wolf decided to stay behind the bush.

This problem (scenario) is taken from
http://www.dcs.qmw.ac.auk/conferences/CS98/problems (the set of common-sense reasoning problems).

We divide the logical program, capable of explanation of such scenario, into the following components:

K : Adequate knowledge about the world.

H : The scenario agents, their properties, intentions and initial states.

R : The final decision of the main agent (wolf in this scenario) .

Knowledge about the world *K*, necessary for the presentation of this scenario, introduces the mutual dependence between the concepts `eat, noise, near, hear, see` and `kill`. Note that not all of these concepts are explicitly mentioned in the scenario text. Agents have to know the causal links between `noise, hear, near, see` to be able to interact within a scenario, and this knowledge is assumed to be available to them.

We present the mutual definitions of these concepts via *dynamic clauses*. They are introduced to express the causal relationships between the predicates irrespectively to the *instantiation state* (the subset of variables for a clause, currently instantiated by constants). In other words, instantiation state tells us which arguments of the clause predicates are known so far (see more details in Section 6).

Upon a call to any predicate of a dynamic clause, it is converted into a (regular) clause, having its head unknown. Its body consists from all the predicates, which are assumed to be currently known. The same dynamic clause can be converted into the different clauses during the same session of a logical program.

(***K***)

If `Who stays behind a bush,` and it is a wolf, if does not eat `Whom`

```
stay_behind_bush(Who)→ wolf(Who), not eat(Who, Whom).
```

For example, if `Who` is known and `Whom` is not known, this dynamic clause could be converted into the regular clause

```
eat(Who, Whom):- not (wolf(Who), stay_behind_bush(Who))
```

It means that the `wolf eats Whom` (unknown) in case he does not `stay behind bush.`
If `Whom` is known, the dynamic clause
If `Whom` is subject to eating, `Who` creates a `noise`

```
eat(Who,Whom) → noise(Whom).
```

`Who` has to be `near` its victim to `eat` it

```
eat(Who, Whom) → near(Who, Whom).
```

Noise of `Whom` is something that could be heard by some agent `Who`

```
noise(Whom), agent(Who) → hear(Who, Whom).
```

To hear `Whom, Who` has to be `near Whom`

```
hear(Who, Whom) → near(Who, Whom).
```

`Who` sees those who near her or close to those, who near her:

```
near(Who, Whom) → see(Who, Whom).
near(Who, Whom), near(WhomElse, Whom) → see(Who,WhomElse).
```

If a `woodcutter sees` a `wolf`, he `kills` it.

```
woodcutter(Who),wolf(Whom),see(Who, Whom) → kill(Who, Whom).
```

(***H***)
We present the agents

```
agent(girl). agent(wolf). agent(woodcutter).
woodcutter(woodcutter). wolf(wolf).
```

Then the wolf's intentions are enumerated. The wolf has two choices: to stay behind a bush or to eat the girl. We assume that the wolf want either of these choices. (It wanted to stay behind a bush at least at the previous step of decision making, that is why he is currently there).

```
want(wolf, stay_behind_bush(wolf)).
want(wolf, eat(wolf, girl)).
```

Besides, the wolf wants not to be killed. This is his passive intention: it cannot achieve it directly. The wolf will make a choice of intention such that it would not imply something unwanted, given his world knowledge *K* and unlimited reasoning capabilities of our agent model.

```
want(wolf, not kill(Smb, wolf)).
```

We approach the clause, which generates ***R***. It is implemented as a decision-making procedure, which searches for the unwanted consequences of each action. It runs through all the possible actions of an `Agent` and gives the `PossibleActions` list. Before each action is processed, we eliminate all the previous assumptions for this `Agent`: `clean_assumptions(Agent).` Then we check whether the wanted condition is active or passive (expressed by the negation, in particular). To do this, we attempt to identify it with the negation of a variable term `NegPossibleAction.`
The next step is to make assumption that the action has actually taken place. We iterate through all the `UnwantedEvent` and call them to check if at list one of them occurs. If it is, the action is rejected from `PossibleActions.`

```
whatToDo(Agent,PossibleActions ):-
    findall( PossibleAction, ( clean_assumptions(Agent),
            want(Agent, PossibleAction),
            not (PossibleAction = (not NegPossibleAction)),
```

```
            assume( PossibleAction ),
            not  (want(Agent, not UnwantedEvent),
                 UnwantedEvent )
             ),
         PossibleActions).
```

Finally, we present the clauses, which operate with assumptions (initiate and reject them, correspondingly)

```
assume( PossibleAction ):-
 ifthenelse( (PossibleAction = (not NegPossibleAction)),
 assert((NegPossibleAction:-fail)),
 assert((PossibleAction)) ).

clean_assumptions(Agent):-not  ( want(Agent,
   PossibleAction),
 ifthenelse( (PossibleAction = (not NegPossibleAction)),
 retract((NegPossibleAction :- fail)),
 retract((PossibleAction))), fail ).
```

***R*:** `[stay_behind_bush(wolf)]` could be generated by the `PROLOG` program, available at http://dimacs.rutgers.edu/~galitsky/MS/wolf.ari

This logical program uses the syntactic metapredicates `whatToDo`, `assume` and `clean_assumptions` to model agents decision making and applied metapredicate `want(Agent, Formula)` to express the agents intentions. First order logic is sufficient to represent world knowledge *K* and the agents reasoning.

2. Anecdote comprehension problem

2.1 Metalanguage support means

We revise the applicability of an axiomatic method, traditional in respect to the property of soundness, to comprehension problems requiring the construction of the explanation metapredicate. A class of textual anecdotes is considered, which intuitively can be referred to as "logical". In our approach, we analyze the anecdotes from the **pure logical** viewpoint taken separately from the natural language (NL) specifics and without any attention to the mechanism of human anecdote comprehension. A language without metapredicates is not expressive enough to support the spectrum of meanings for the predicates of agents' "mental" states and actions (modalities) inherent to an anecdote. Traditional reasoning, involving metapredicates for temporal, spatial and deductive relationships (for example, [5,6,9,13]) is believed to be insufficient for anecdote representation. The anecdote end (agent reaction) is inconsistent to the preceding anecdote part within both traditional first-order and multilevel axiomatic. Combinations of the enumerated approaches could be an alternative to out approach, but the computational implementation seems to be rather hard in spite of all the advantages of the traditional axiomatic method.

We introduce the anecdote model by means of metalanguage support, which **gains** the **flexibility** of inference construction by **sacrifice** of the **soundness** (transition to the **relative** soundness). Not only both the arbitrary and variable metapredicates are allowed to be involved in the inference, but the whole axiomatic system should be reestablished to extend the deducibility relation for the closed formal systems (scenarios).

Our class of **logical** anecdotes allows free translation from one natural language into another and requires each concept to be defined within the bounds of the other anecdote concepts. We approach the anecdotes as if they were originally presented in a formal language. In terms of language conversion, the anecdote comprehension problem (AC) is posed as the transition from a representation in a reduced formal language, which has to be accompanied by NL comments (in the object level) to the MS language with sufficient expressiveness means. At some point, we start to ignore the fact that NL is the native anecdote language and consider the properties of the formalized ”anecdote model” system.

Our **anecdote model** is **tested** in respect to its ability to verify whether a particular formal system is an anecdote or not. To assure proper testing environment, we take the text fragment, which sounds like an anecdote to the group of examinees. Then, each person is given the series of various formal language presentation of that anecdote, and chooses the most appropriate one. The presentation that was chosen by the majority of examinees is considered as a proper anecdote translation and can serve as a test for AC program. This way we eliminate NL component of the AC problem. The program checks whether a formalized system match the anecdote criteria. When the program results start to agree with the anecdote estimates of the majority of examinees, the model is accepted and is subject to further investigation.

AC is considered as one of the most challenging task in the series of MS reasoning tools for scheduling, scene analysis, semantics for natural language and speech processing [1,8]. Being objectively complex and simultaneously closed and self-contained, AC can serve as an applicability test for the wide variety of reasoning tools for the flexible reasoning. We believe that AC poses high level requirements to the reasoning power, language expressiveness and predicate invention in comparison with the other AI problems and the logic for foundation of mathematics.

2.2 An anecdote and its MS representation

We present the following anecdote and its simplest formal representation to make the reader familiar with MS extension of a logical programming language. The predicates' semantic is assumed to be clear given the predicate names.

A man comes home from work and gets into bed, expecting his wife to be there. Suddenly there is a sneeze from underneath the bed.
The husband calls out: "Fido, is that you?"
"Yes, it's me!" - comes a happy response, as the man feels his hand licked.

The anecdote understanding is specific for each individual; there is a wide variety of possible formal representations of this scenario. We suggest the scenario representation and explanation that seem to be the simplest, ignoring the majority of details, making the anecdote sound realistic.

We assume, that in a “lover” anecdote, the husband wants to find the lover. We simplify the scenario, not mentioning the wife, the bed, the husband's process of perceiving the sneeze, etc. Instead, we directly approach the state when the husband heard a sneeze, and he knows that it could be either from his dog or from a lover.

(H.1) know(husband,(sneeze(Who),(Who=lover;Who=dog))).

The husband wants to know who have sneezed:

(H.2) want(husband, know(husband, sneeze(Who))).

and asks about it:

ask(husband, Who, sneeze(Who)).
The lover does not want husband to know who have sneezed:
(H.3) want(lover, not know(husband, sneeze(lover)))
and responds (behaves) as a dog:
(R) respond(lover, husband, sneeze(dog)).
Here we significantly simplified the lover's reaction *R*. Instead of presenting what the lover said and that he licked the husband's hand, we just express that the lover responded that the sneeze was performed by the dog.

To derive the last statement (lover's reaction *R* to that have happened (*H.1-H.3*)), one needs the following general knowledge about the world:
If *Smb* asks *SmbTo* if *SmbTo* is involved in an *Action*, and *SmbTo* wants *Smb* to think that this action was performed by *SmbElse,* then *SmbTo* usually let *Smb* know that *SmbElse* did this *Action:*
(KU.1) respond(Smb, SmbTo, Action(SmbFake)):-
ask(SmbTo, Smb, Action(Smb)),
want(Smb, not know(SmbTo, Action(Smb))),
know(Smb, know(SmbTo, (Action(Who),
(Who=Smb; Who=SmbElse)))),
SmbFake=SmbElse.

Dogs are not able to speak (*respond*):
(KU.2) know(SmbTo, (Smb=dog, not respond(Smb, SmbTo, Smth))).
We do not present here some common-sense knowledge about *ask/respond* and *know* (see Section 7.1).

Sometimes, when *Smb* wants badly to perform an *Action* with (over) *SmbTo* (here, in particular, to tell a lie), he does it ignoring his *know*ledge that everybody *know*s that this *Action* is impossible*:*
(KS) ***insist_ignoring_contradictions(****Smb, Action****):-***
want (Smb, Action(Smb, SmbTo)),
know(Smb, know(SmbTo, not Action(Smb, SmbTo))),
Action(Smb,SmbTo)).
This metapredicate is required to build the inference *KU.1, KU.2, KS, H.1, H.2, H.3 :- R* (the anecdote explanation). We can "dissipate" this formal anecdote into a "common-sense reasonable" formal system, which requires just the "usual" knowledge about the world *KU*, if we either set $H.3_{dissip}$ or R_{dissip} in the following way:
($H.3_{dissip}$) *want(friend, know(husband, sneeze(dog)))* (it was a friend, but not a lover, who wants to help the husband to find a dog).
(R_{dissip}) *respond(lover, husband, sneeze(lover))* (the lover confessed) .

The semantics of the (canonic) metapredicates of the mental state and action (modal metapredicate) are defined in accordance to the scheme: *name(Actor, [Addressee], Action_or_Item), name*$\in${*want, find, know, ask, respond, suggest, inform*}. The second argument *Addressee* of these canonical modal metapredicates is optional, and the last argument ranges over the formula syntactic properties. An anecdote requires at least two modalities, *want* and *know,* but typically contains multiple modalities. Each modality is characterized by the specific reasoning mechanism, so the logic with the operators for multiple modalities, functioning within the object-level, seems to be insufficiently expressive. We refer to Section 7 for the axioms for various modalities.

It could be shown for MS, that a modal concept that seems to require more than these three arguments could be derived from the canonic modal metapredicate by means of concretizing**.** It includes the conjunction with the other modal metapredicates, which link the additional variables to the canonical ones (see examples in Section 3.3). Conversely, the predicates like *sneeze* just express the relationship between the universe elements and their properties.

What do we have in addition to the previous scenario (Section 1)? The lover reaction R is emergent, in contrast to the wolf decision, which seems reasonable from the usual knowledge viewpoint. Though it becomes possible to explain lover's actual reaction R relatively to his dissipated reaction R_{dissip}, when one involves the *KS* knowledge for explanation.

In the wolf scenario we just need the single modal metapredicate *want* to express the wolf intentions and *whatToDo* to express the mechanism of intention competition. In the lover anecdote, we need the metapredicate ***insist_ignoring_contradictions*** in addition (*whatToDo* is skipped for briefness), to be able to explain the particular agent's choice R when the other choices have higher likelihood. *KS* allows us to explain the behavior in the inconsistent conditions, when the agents do not act, as the usual world knowledge would expect. To be able to construct that explanation, we need MS and two kinds of knowledge: what usually happens (*KU)* and what happens sometimes (*KS*), when the agents are forced to behave in the specific conditions.

2.3 Anecdote components

In this section we informally describe the components of the conventional formal anecdote (the partial case of the formal scenario). The formal scenario motivations are drawn from the knowledge representation prospective: We introduce the anecdote model, which is intended to embrace the common properties of reasoning inherent to anecdotes. To make the term *model* sound more formally, we add another metalevel, whose (meta-metapredicates range over the logical programs (anecdote components). Then, the task of linking the anecdote components and building the components, obeying certain properties, can be formulated as the **invention of a meta- metapredicate.**

The formal anecdote model contains the three following components: world **knowledge**, what **happened** with the anecdote agents, and the final **reaction** of one of them, the punch line.

*K***: (Knowledge)** about the world, necessary (from a human agent viewpoint) to understand the anecdote. This component contains the definitions of the anecdote concepts through each other. An important result concerning the knowledge representation is that AC is invariant in respect to the meaning of each concept taken separately and is determined by the structure of the mutual dependence of these concepts.

The main idea of K representation is that it is sufficient for scenario to **mutually define** the concepts through each other rather than supply the actual definition of the concepts. Scenario reasoning does not require the division of the set of all concepts into the basic (indefinable, set by axioms) and derived (obtained via expression through the basic). This is the key difference between the scenario reasoning and traditional axiomatic method, which starts with the introduction of basic concepts. Besides, the human agents indeed aware of what *know* and *want* means, but the scenario agents just informed that "if *Smb knows* this..., than *SmbElse wants* that ...". Hence the term "definition" has the special (relative) sense for the scenario reasoning.

In the course of the scenario inference new clauses (assumptions) could be asserted, which introduce the additional dependence between the concepts, defined via each other. Hence the meanings of the predicates within a scenario are specified during the reasoning procedure.

K is divided into *KU* (**usual**, commonsense knowledge) and *KS* (known to some anecdote agents to be true **sometimes**, if some special facts occur). We pose the following requirements for *KU* to work as an anecdote model component. These requirements are compatible with the definition of formal scenario (Section 2.4) but is not implied by it.

1. As many concepts as necessary can be used, but all the links (mutual definitions) between the concepts must be as **general** as possible and adjusted for the particular anecdote in as little degree as possible. Formally, for each *KU* statement there is no constant *c* which could be replaced by a variable *V* of the proper type leading to the successful run of this definition, instantiating *V* by $c' \neq c$. The pair of syntactic metapredicates ***generalize / concretize*** could be applied to a clause to eliminate/introduce some formula constraints. *generalize* metapredicate, for example, converts the first-order (object-level) predicate constants into the second-order predicate variables for agent actions with addition of the quantifiers over these variables. Application of ***attenuate*** metapredicate sometimes raises the formula generality (See Section 6 for more examples).
2. *KU* obeys the properties of **deductive closeness**: for each pair of predicates there is a way (by means of adding the *H* fact assumption) to obtain the definition of one of them, dependent on the other (in terms of the logical programming, a call of the first predicate implies a call of the second one). In other words, everything could depend on everything in the course of the scenario reasoning.
3. The **separability** property holds for each pair of the predicates: for any H (or any (Herbrand) universe) there exists their definitions which would not be modified into identical by the
4. *KU* is **compact**: there is no such predicate that its definition requires infinite loop of the clause run for any H (or universe). Observing (2), we resume that each predicate could be expressed via the other (with the help of MS, which can reduce possible loops using the metapredicate ***attenuate***, see details in 3.2).

KU contains the properties of participating agents, which can perform action, provide other agents with information and be the subject of other agent's information.

KS (inconsistency loci) contains the **metapredicates,** which describe the exceptions of *KU* for the particular universe. The boundary between *KS* and *KU* is drawn as only the other anecdote components are fixed. *KS* (in addition to *KU*) are necessary to explain the unexpected end of an anecdote, linking the *H* and *R* components. The object-level representation of these metapredicates would be incompatible with *KU.* This is one of the key reasons the anecdote model requires MS to explicitly **transfer exceptions into metalanguage** and to declare the inconsistent predicate (a predicate, whose elimination would remove the contradiction in a formal system) **variable** (unknown). It's worth mentioning that *KS* and *KU* do not necessarily obey the above properties of generality, deductive closeness and separability.

There is an important question concerning the building of *KU* and *KS* for a particular anecdote. There are two approaches with increasing levels of complexity:

1) Explicit enumeration of *KU* predicates for a particular anecdote, achieving the above *KU* conditions and adding *KS* metapredicates prior to the actual anecdote representation.

2) Explicit enumeration of *KU* and *KS* predicates for all possible anecdotes for given problem domain. AC, then, independently finds the appropriate concepts, builds the anecdote inference and checks the *KU* conditions. This approach poses the same requirements to the variety of the concept meanings as the NL representation formalisms. The inference search dimension is significantly increased because each possible link of the anecdote components requires the multiple iteration through all the available concepts in *KU* and some concepts of *KS*. The task of the adequate concept choice of AC can be considered as a traditional rule-based understanding system in AI.

H: **(Happen)** This is the anecdote body, which contains all the information about the anecdote agents, natural sequence of the agents' actions and the explanation of why these actions did occur in that sequence. From the logical viewpoint, each sentence in *H* neither fully follow from the previous ones and *K* nor directly contradict *K*. If the agents know *K* they would not behave differently or something else (not *H*) would happen.

R: **(Reaction)** The anecdote reaction, which is assumed to be a humor locus.

MS inference is required to explain the "emergent" properties of an anecdote, $KU,H\rightarrow R$. *KS* metalanguage predicate (for example, ***insist_ignoring_contradictions*** in 2.1) is necessary to link the "unusual" *R* to the "typical" *H*: $KS,KU,H\rightarrow R$ That metapredicate plays the key role in drawing the basic idea of a scenario and could serve as its identification.

The anecdote model **meta metapredicates** range over the following inference objects. We need to build them to verify if a formal system obeys the anecdote criteria (Section 2.4).

Direct dissipation

Declaring *R* unknown, we derive the most natural anecdote reaction R_{dissip} having *K* and *H*. Building this inference, we loose all the humor and originality features and convert the anecdote into a trivial fully formalized situation. We try to obtain the complete set of statements, which describe that usually takes place when *H* happens in a world, living in accordance to *K*. If some statements of *H* are not compatible with *K*, it is necessary to involve MS to complete *K* with all the necessary assumptions.

Inverse dissipation

Declaring *H* unknown, we reconstruct what could happen (H_{dissip}) if the most natural *R* was actual *R*. We try to "restore" all the details of *H* and make explanation for all components of *R* and derive their order having *K*. Like direct dissipation, obtained H_{dissip} eliminates all the humor features and converts the anecdote in the fully formalized representation. Usually direct and inverse dissipation requires MS rather for knowledge representation than for actual reasoning.

Real anecdote

To derive the actual R having actual H and K we need to employ the whole expressiveness power of MS. We reevaluate K to be fully compatible with H, and then verify if K is adequate to infer R. The real anecdote reasoning takes into account the closeness feature of an anecdote in respect to H: each term should be in use for the inference, explaining R.

2.4 The formal scenario, anecdote and joke

We traditionally approach the concept of logical program. The only addition is the set of syntactic and applied metapredicates. We assume A term is assumed to be a (meta) variable, a (formula) constant or a complex term of the form $f(t_1,...,t_n)$, where $t_1,...,t_n$ are terms and f is a functional symbol (syntactic metapredicate with finite arity n>0. We distinguish predicates and metapredicates, variables for agents and objects and meta-variables (ranging over formulas). An atom is a formula of the language that is of the form $p(t_1,...,t_n)$ where p is a predicate or metapredicate symbol of a finite arity n>0 and $t_1,...,t_n$ are terms (arguments of the atom). A literal is an atom or its negation.

A rule (clause, definition) r is the formula of the language of the form

H :- B_1, ..., B_k , k>0,

H is atom (head) and B_i are literals (body) of the rule. We call a **fact** a rule with no body. We distinguish dynamic and static (traditional) rules (Section 1.1 and 6 with more details). When the head is called, the body's literals are consequently subject to call.

A term, atom, literal rule is ground (instantiated) if there is no variables. A logical program is a finite set of rules. Given a logic program, the Herbrand universe U is the set of all possible ground terms recursively constructed by using constants and functional symbols occurring in the logical program.

Scenario is a logical program, consisting of 3 components:

1) *KU* (contains the clauses) and *KS* (contains the clauses of metapredicate definitions).
2) *H,* a sequence of (ground) facts.
3) *R,* a single fact.

R is a metapredicate of mental state and action, *modal(Who, Whom, What),* such that:

1) The call to *modal(agent1, agent2, What)* succeeds and *What* is determined as a fact.
2) This call involves the calls to all scenario rules and facts.

The H component obeys the condition of being monotonic relatively to the instantiation states *is* (Section 1 and 6 with more examples). is: $p(X_1,\ldots, X_n) \rightarrow \{b_1,\ldots, b_n\}$, b_n is true if X_n is instantiated and false if it is not. $\{b_1,\ldots, b_n\} > \{c_1,\ldots, c_n\}$ if for $i= 1,n$ if c_i is true than b_i is true. We introduce the order > on the family of all instantiation states for a predicate p. Then for all $i<j$ the following holds for H to be a scenario component. We convert H_i and H_j into the terms var(H_i) and var(H_j) with all variable arguments such that any two variables of var(H_i) and var(H_j) are different if the proimage constants are different in H_i and H_j . If

$is(\ var(H_i\ ):\ K,\ H_1,\ldots,\ H_{i-1}\ ,\ H_j\ \text{:-}\ var(H_i\ )\ )\ >\ is(\ var(H_i\ )\ :\ K,\ H_1,\ldots,\ H_{i-1}\ \ \text{:-}\ var(H_i\ )\ ),$

then $is(\ var(H_j\ ):\ K,\ H_1,\ldots,\ H_{i-1}\ ,\ H_i\ \text{:-}\ var(H_j\ )\ )\ >\ is(\ var(H_j\ )\ :\ K,\ H_1,\ldots,\ H_{i-1}\ \ \text{:-}\ var(H_j\ )\ ).$

Informally, each next step of H is rather implied by K and the previous steps than that of the previous steps.

We outline the following properties of scenarios, anecdotes and jokes. (Their relations are shown at Fig 2.4.) They are either follow from the definitions of scenario, anecdote and joke or were obtained computationally.

1) Scenario reasoning is monotonic relatively to Hi. For ant fact F, if
   $K, H_1, \ldots, H_{n-1} :\text{-} F$ than $K, H_1, \ldots, H_{n-1} :\text{-} F$, $i>1$.
2) Anecdote reasoning is non-monotonic relatively to *KS* (for anecdotes). Addition of *KS* leads to deriving R versus ¬R:
   $KU, H_1, \ldots, H_{n-1} :\text{-} \neg R$, *but* $KU, H_1, \ldots, H_{n-1}, KS :\text{-} R$.
3) Anecdotes allow direct and inverse dissipation.
4) Jokes posses the mechanism to express the predicate's "degree" and to perform semantic amplification (Section 2.7).
5) Anecdote knowledge representation of allows posing the experimentally verified joke criteria.
6) The formal simplification procedure can be applied to a scenario. If there exists a predicate or metapredicate, which could be expressed via the others within a scenario, it could be eliminated such that the resultant logical program would be a scenario.

The scenario reasoning requires formalization of spatial, temporal, action and knowledge relationships together with the capabilities of adaptation to a particular PD. Using traditional axiomatic method, merging the axioms, reflecting particular aspects of human intellectual activity, is the complicated task, not always subject to accomplishment.

There is an algorithm of merging any pair of scenarios with arbitrary components. The merging result is easy to characterize because one has to determine the common reaction and perform the common direct and inverse dissipation. We leave the issues of splitting and merging the scenarios outside the scope of this study.

2.5 Representation of scenarios by the sequence of graphs

To draw the scenario scheme not based on the concept of the logic program, we introduce the sequence of graphs $\{G_i\}$ with the fixed set of nodes for agents and objects of their action. The agent and object constants, agent and object variables correspond to the set of graph nodes N. Meta-variables and constant formula arguments correspond to subgraphs (meta-nodes) of G_i. Predicates and metapredicates are the graph paths (sequences of directed edges), which go through the nodes and meta-nodes of their arguments (nodes for the agent and object arguments, meta-nodes for the meta-variable arguments.

Each clause and fact has its own graph in a sequence. Only those nodes are connected by edges, which occur in corresponding clause or fact. We will not distinguish the names of arguments and the names of corresponding nodes further on.

The call of a predicate $p(X_1,\ldots, X_n)$ (or metapredicate), which is a clause head from the graph G_i , is defined as the set of mapping from G_i to $G_{k1}, \ldots G_{km}$, $k,m>0$, for each predicate q_m from the body of the clause

$p(X_1,\ldots, X_n)$:- $q_1(Y_{11},\ldots, Y_{1n}), \ldots, q_m(X_{m1},\ldots, X_{mn})$.

The map between G_i and G_k by call of the predicate p is arranged as argument-by-argument correspondence $X_1 \rightarrow Y_1,\ldots, X_n \rightarrow Y_n$. The mapping between meta-nodes is established, involving the metapredicate semantics. (Note, that the graph construction is invariant relatively the semantics of the object-level predicates.)

Scenarios K - H - R components
Mental state and action metapredicate, which initiates the run of all clauses from K and all facts from H

Anecdotes agents, who *want* and *know*
$K = KU \cup KS$: *link metapredicates*
Non-monotonic derivation of R

Jokes emergent R
R is inconsistent with KU, H in the absence of KS
R_{dissip}\= R, H_{dissip}\=H

Figure 2.4. The *Jokes* $\subseteq$ *Anecdotes* $\subseteq$ *Scenarios* inclusion diagram. Anecdotes are multi-agent scenarios, when the agents have knowledge and possibly mutually inconsistent intentions. Jokes are anecdotes with the emergent reaction, incompatible with the usual knowledge; dissipation is non-trivial and leads to an anecdote.

For each m, G_{km} contains the path for predicate q_m with the nodes $Y_{m1},\ldots, Y_{mn}$. If G_{km} corresponds to a fact, variable Gi nodes $\{X_{m1},\ldots, X_{mn}\}$ change their names to the constant nodes of $\{Y_{m1},\ldots, Y_{mn}\}$. If G_{km} corresponds to a clause, variable Gi nodes $\{X_{m1},\ldots, X_{mn}\}$ change their names to the updated nodes from$\{Y_{m1},\ldots, Y_{mn}\}$ after the body predicates are called. Finally, the nodes $\{X_1,\ldots, X_n\}$ will change in accordance to the arguments of the body predicates.

Building the family of mappings for recursively called predicates is a traditional task in building the interpreters for declarative programming. The situation becomes non-trivial when we introduce arbitrary (syntactic and applied) metapredicates. The values of meta-arguments (and the metapredicate satisfactions) become dependent on the order the mappings are established for the given called predicate. To simplify the presentation, we eliminate the graph for syntactic metapredicate clauses from our considerations.

Hence, the result of a call to a predicate for the sequence of graphs is establishing the mappings between the graphs, which contain the set of predicates, the called predicate depends on. The necessary condition for a **sequence of graphs to be a scenario** is the existence of a graph with modal metapredicate and the call to it, which initiates the family of mappings such that each other graph is mapped into.

Posing the sufficient conditions for a sequence of graphs to be a scenario, anecdote or joke seems to be rather hard. Graph representation meets some difficulties operating with syntactic metapredicates, when the actual formula transformation could not be ignored.

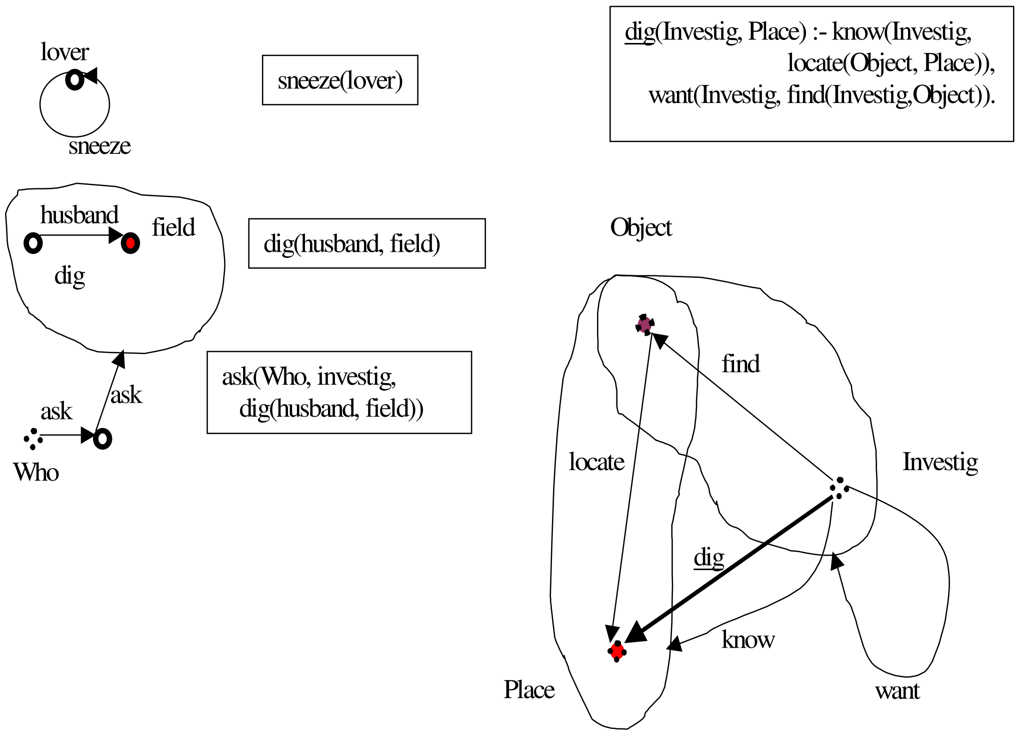


Figure 2.5.1 Representation of clauses by the graph with colored nodes and edges. The predicates are represented by the path, consisting of the directed edges of the same type. The arguments of the object-level predicates are represented by circles (dashed circles for variables). A meta-variable is a subgraph; a metapredicate is a fixed color graph path through the nodes for *Agents* (white) and *Objects* (gray) and through subgraphs for the meta-variables. Facts are shown on the left, and the rule - on the right: the head predicate is underlined. For the dynamic clauses, the search of mappings occurs for all paths in a graph: which path is playing the head role of a clause.

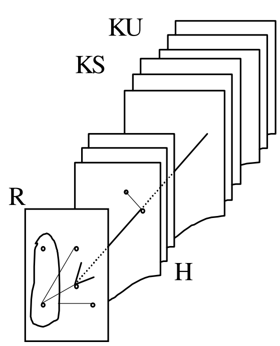


Figure 2.5.2 Representation of the scenario by the sequence of plane graphs. We have sequentially (from back to front) graphs for *KU, KS, H* and *R.* If sequence of graph is a scenario, than the call to *R* initiates mapping into all graphs from this sequence.

2.6 The joke criteria

It is a trivial task to determine weather a scenario is an anecdote, having the inclusion diagram Figure 2.4. More sophisticated means are required to recognize whether an anecdote is a joke.

Computational experiments showed that it is always possible to build the following three inferences: actual K,H :- R, forward dissipation K,H:- R_{dissip} and inverse dissipation K,R:-H_{dissip}. If $R= R_{dissip}$ or $H= H_{dissip}$, the formal system is rejected to be related to a joke. Otherwise we proceed to the anecdote recognition based on the comparison of these three inferences. We suggest the following criteria, when the **values** for the **dissipated inferences** are **lower** than the value for the actual one.

1. Inference complexity, which includes the degree of metapredicate embedding.
2. Usage of *KS* versus *KU*.
3. Abrupt drops of criteria 1-2 for an anecdote and smooth decrease of these parameters for an arbitrary scenario under formal simplification (elimination of some predicates by means of expression via the rest of ones, Section 2.4).

The lower confidence level is passed when the certain criterion for the actual inference exceeds ones for the dissipated inferences. The higher confidence level requires the maximum criteria for R versus some inference for R', which would still be actual relatively to R_{dissip} and H_{dissip}. The recognition algorithm does not try to synthesize R'. We use the lower confidence level because we are rather concerned not to miss an anecdote than to accept a formally poor one.

To present an example of the joke criteria verification, we consider the scenario from Sections 1 and 2.1. The wolf scenario is clearly not a joke, because the dissipation operation does not change it: $R= R_{dissip}$ and $H= H_{dissip}$. It is not an anecdote, because there is no linking metapredicates, just the metapredicate of usual decision-making *whatToDo.*

To create an anecdote from the wolf scenario, we must add the metapredicate, capable of explanation of "something unusual". For example, we could take the metapredicate ***insist_ignoring_contradictions****(Smb, Action)* to explain, that the wolf was so hungry that he decided to eat the girl in spite of woodcutters ability to kill him. Except that, the wolf needs the basis of two modalities: knowledge and intention. The former modality should be added explicitly to comprise the anecdote. *Know* was skipped in the original wolf scenario because the approximation, that its agents have the same world knowledge, was adequate to reason about this scenario. To derive a scenario, both ***insist_ignoring_contradictions*** and *know,* specific for the wolf, should form *KS*.

Hence, we have `eat(wolf, girl)` as a reaction R, which should be obtained if we call `want(wolf, What)`. R_{dissip} will be the wolf's choice in the original scenario, and H_{dissip} could be something like wolf's knowledge that the `woodcutters` do not hear the `girl`.
The resultant scenario would be an anecdote, but not a joke, because the complexity check (1) fails. The requirements 1)-3) are satisfied for scenarios 2.2 and 2.8, therefore they are formal jokes.

2.7 Atomic humor predicates

Our model considers the pure logical properties of the anecdotes. These properties are based on the definitions of these predicates through each other, without taking into account the meaning of the particular predicate, taken separately. Is it true, that all the predicates, participating in an anecdote, are uniform in the sense that they affect the total impression only by being a part of the whole definition system and not by their own semantics? It is not true for the majority of jokes, when there is a few predicates with the special meanings we do not include in these definitions. We call these predicates **atomic humor** (for instance, incompetent, bad judgement, bad luck, drunkard, etc.). Each of them causes the series of associations we do not include in *K* to obey the closeness condition.

We are interested in how the properties of atomic humor predicates are combined with pure logical properties to form a scenario, which is rated higher by a human agent than if this scenario would possess either set of properties. The interaction between atomic humor and pure logical sets of properties works in the way of **exaggeration** of the former by means of the latter. The inference which ties *H* and *R* together amplifies atomic humor predicates in the following sense: a human agent is convinced by the scenario explanation inference that the **degree** an atomic humor predicate is true is higher than it was at the very beginning of *H*.

To formalize the meaning of atomic scenario predicate's "degree", we introduce the partial order relationship on the set of all clauses $\{d_i\}$ containing this predicate p which participate in the scenario inference. The degree of the definition d_i containing p is higher than that of d_j if ***attenuate***$(d^p{}_i, d^p{}_j)$ holds ($\boldsymbol{d^p{}_i}$ and $\boldsymbol{d^p{}_j}$ are the formal expressions of p via d_i and d_j correspondingly after application of the inconsistency shift rule from Section 3.2).

The additional amplification condition is that the inference of ***attenuate***$(d^p{}_i, d^p{}_j)$ is necessarily performed either by an anecdote agent or a human agent. In other words, this inference is not just possible for a particular anecdote, but should have been actually built in the process of understanding of this anecdote.

The possible variability of the atomic humor predicates is much lower than that of the mental state and action ones. So the scenario composition (Section 4) starts from the choice of the 1-3 atomic humor predicates which form the skeleton. It introduces the **amplification** limitations for *K* and *H*, narrowing the search of their possible combinations. One can imagine, for example, that something happens (in *H*) with an agent, possessing the quality, expressed by an atomic scenario predicate *p*. The other agent learns in *R* that the third agent's action occurred because the first agent possesses that quality in even higher degree (that we knew in the beginning of *H*).

2.8 A complete example of an anecdote

Once during a World War I a soldier receives a letter from his wife saying that it's a time to dig up a potato field, but nobody can help.

The husband writes the letter back " Do not dig up the field. There is my gun hidden there"

In a week, the wife responds: "The whole field is dug up by the investigator. I started to plant potatoes".

The problem domain includes the knowledge that letters were subjected to review by police and that the weapon ownership was prohibited.

This is the definition, how an investigator `investig` can obtain knowledge of `Matter` reading the `Letter` from `Smb` to `SmbTo` `(inform(Smb, SmbTo, Letter))`. This

Letter mentions the Object, which is the subject of investigator's search; the investigator wants (himself or somebody else) to find this Object: want(investig, find(Who, Object)). The investigator extracts from the Letter all information Matter about this Object. Syntactically, Matter consists from all the terms from Letter which contain Object. This definition means that the investigator gets all information about his Object of interest (including the other agents) from any Letter and agents Smb and SmbTo.

(KU.1)
```
know(investig, Matter):- agent(Smb), agent(SmbTo),
            Smb\= investig,SmbTo\=investig,
        inform(Smb, SmbTo, Letter),
        want(investig, find(Who, Object)),
            %The part of the clause below performs the search of the
            % terms Matter in Letter, containing the argument Object
            expand(Letter, Letters), var_const(Letters, Lettersc),
                          member(Object, Lettersc),
               member(Matter, Lettersc), Matter \= Object,
                          Matter=..TermArgs, var_const(TermArgs,
TermArgsc),
                          member(Object, TermArgsc).
```

Below is another illustration of the dynamic clause. An agent, who wants to find an Object and knows that this Object is located in a Place, digs this Place. The first clause, which expresses the digging condition, is the following:

(KU.2)
```
dig(Investig, Place) :-
  know(Investig, locate(Object, Place)),
  want(Investig, find(Who, Object)).
```

On the other hand, if we need to reason about an agent knowledge, we conclude, that she knows the Object location locate(Object, Place) from the fact that she wanted to find this Object and dug the Place:

```
 know(Invest, locate(Object, Place)) :-
                want(Investig, find(Investig, Object)),
                                             dig(Who, Place).
```

The dynamic clause will look like

```
know(Who, locate(Object, Place)),
    want(Who, find(Who, Object))  -> dig(Who, Place) .
```

This clause is required for inverse dissipation. Below we conventionally use the verb tenses in the definition comments, not expressing the temporal relationships explicitly.

An agent Smb responds to another agent Addressee to his request inform(Addressee, Smb, Matter) that nobody can dig a Place, if Smb failed his plan concerning Whom to dig this Place:

(KU.3)
```
respond(Smb, Addressee, not(agent(Who),dig(Who, Place))):-
          agent(Smb), not plan(Smb, dig(Whom, Place)),
          inform(Addressee, Smb, Matter),
             %below is check that Matter was about dig
            expand(Matter, MatterE),
            member(dig(Agent, Place), MatterE).
```

This definition is required for direct dissipation.

The following knowledge of the concept ask is necessary for inverse dissipation. We present the definition for the situation, when an agent is asking another agent to act conversely to his initial will. After an agent Who received from Whom the Letter with

Whom's Intention, Who asks Whom to perform an Action = not Intention, because of the Reason. The Reason is something that the competing agent Investig does not know, but Whom has to know to logically link Reason and Action. The latter procedure resumes the clause.

(KU.4)
```
ask(Who, Whom, (Reason, Action) ) :-
       inform(Whom, Who, Letter),
        clause_list(Letter, Letters),
        member( (want(Whom, Intention)), Letters),
         (Action= (not Intention)),
       want(Who, not know(Investig, Reason )),
        % find linking clause
       clause(hDissip, H), assert(H),
     inform(Investig, Who, Info),
       clause(know(Invest, Reason), RBody),
       clause_list(RBody, RBodys),
       member(Info, RBodys), member(Intention, RBodys) ,
       Intention \= Info.
```

We proceed to the definition of an agent's planning of performing an action (here - dig) by another agent Who or himself. The easiest way is when Who (has already) performed digging (ore ready to do it):

(KU.5)
```
plan(Smb, dig(Who, Place)) :- agent(Who),
                              dig(Who, Place).
```

Here we ignore the verbs' tenses to simplify the scenario consideration, eliminating the temporal reasoning.

Another possibility is when the other agent Who wants to dig, and Smb himself knows that he will not(cannot) dig:

```
plan(Smb, dig(Who, Place)) :-
               want(Who, dig( Who, Place)), Who \= Smb
                 know(Smb, not dig( Smb, Place)).
```

The third option is sometimes possible. If neither himself nor the other agents done this action and do not want (cannot) perform it, Smb can introduce some false information to make the other agent Who doing that action (cheat on this agent).

(KS.1)
```
plan(Smb, dig(Who, Place)) :-   agent(Who),
              Who\=Smb, want(Who, not dig( Who, Place)),
              not want(Smb, not (dig(Who, Place)))
               cheat(Smb, Who, Fake, dig(Who, Place)).
```

We introduce the concept of cheating, when an agent Who, achieves the action WhoWill to be performed by Whom by means of informing Whom that Fake holds. Who should want WhoWill, besides, it should not be true that Who want Whom not to perform WhoWill (PROLOG peculiarity of the negation as failure) and Whom should not want to do WhoWill . Besides, Who should know that Fake

(KS.2)
```
cheat(Who, Whom, Fake, WhoWill):- ( WhoWill =
                  dig(Whom, Place)),
           want(Who, dig(Whom, Place)),
           not want(Who, not dig(Whom, Place)),
           want(Whom, not (dig(Whom, Place))),
           know(Who, not Fake),
                %we process the clause, which links Fake and WhoWill
              clause(dig(Whom, Place), Body),
              clause_list(Body, Bodys),
              member(know(Whom, Fake), Bodys),
```

```
    %the part below verifies that the cheat was successful for Who,
    %because Whom accepts Fake and performs WhoWill
   clause(reaction, R), assert(R),
           know(Whom, Fake),
           WhoWill.
```

The "pure semantic" definition of cheat is presented in (3.3).

We note that only the investigator's digging condition actually needs specification what action (`dig`) to perform. To obey the generality of knowledge representation, the other definitions should contain the quantifier "for all `Action` " over the second order variable for `Action`. We write `dig` instead of `Action` in our definitions to simplify the syntax, though higher-order PROLOG extension is available for this example.

The *H* component enumerates the agents:

`agent(husband). agent(investig). agent(wife).`

The investigator wants to `find` a `gun`. He does not `want` to `dig` a `Place.`

(H.1) `want(investig, find(investig, gun)).`

(H.2) `want(investig, not dig(investig, Place)).`

The `wife` writes to `husband` that she `wants` somebody to `dig` her `field` and nobody (can) do it.

(H.3) `inform(wife, husband, ( want(wife, dig(Smb, field)), not dig(Smb, field) )).`

The `husband wants` somebody to `dig` this `field`, he `knows` he cannot do it himself and he does not `want` his `wife` to do it:

(H.4) `want(husband, dig(Smb, field) ).`

(H.5) `know(husband, not (dig(husband, field))).`

(H.6) `want(husband, not (dig(wife, field))).`

The husband knows that there is no gun in the field (eliminated under inverse dissipation):

(H.7) `know(husband, not locate(gun, field)).`

The anecdote reaction is the husband's decision how to `dig` the `field:`

`?-plan(husband, dig(Who, field)).`

He chooses the following:

(R) `inform(husband, wife, ( locate(gun, field), not dig(wife, field) )),`

to inform the wife that the gun is located in the field and that the wife should not dig it.

The usual `husband's respond` would be the impossibility to have anybody to `dig` the `field`

(R_{dissip}) `respond(husband, wife, not ( agent(Who), dig(Who, field))).`

For the inverse dissipation, we deduce what would naturally imply from the fact that the `husband asks` his `wife` not to `dig` the `field.` The `husband` does not `want` his `wife` (or anybody else) to `dig` the field because he does not `want` the `investigator` to `know` the `gun location.`

(H_{dissip}.4) `want(husband, not know(investig, locate(gun, field) )).`

We conclude, that the `investig` should have `informed` the `husband` about his interest to the `gun.`

(H_{dissip}.8) `inform(investig, husband,`
`want(investig, find(investig, gun))).`

This scenario is available as a PROLOG demo, where the main calling predicate is the husband's planning. We present three separate files for real scenario, direct and inverse dissipation at http://dimacs.rutgers.edu/~galitsky/MS/dig.

Location of the atomic humor predicate in this scenario is non-trivial. We consider the (political) property of being an investigator, reading the private letters, as an example of the non-logical humor locus.

We resume this section with the inconsistency shift demonstration. We consider the world of `husband`'s intention, `want(husband, Smth)`. In particular, `want(husband, dig(Smb, field))`. Originally, in this world the following was not satisfied: `want(investig, not dig(investig, field))`. Besides, `know(investig, locate(gun, Place))`. We apply the inconsistency shift rule to the investig mental state, enforcing it's compatibility with `husband`'s mental state. Let us consider the clauses for *KU.1* and *KU.2*. The `husband` applies the inconsistency shift (telling a lie, cheating) to the `investig` to turn
`want(investig, not dig(investig, field)), know(investig, locate(gun, Place))` into `want(investig, dig(investig, field)), know(investig, locate(gun, field))`.
The latter is consistent with `husband`'s intentions. The input formula argument for the inconsistency shift metapredicate is the second line of *(KU.1)* clause, and the output formula arguments are the third and the fourth argument of `cheat` *(KU.2)*.

3 Basic MS

3.1 Metapredicates and relative soundness

To build ML, the language for MS, we add arbitrary metapredicates and metalanguage inference rules to the traditional first-order language. The **arguments** of metapredicates range over **well-formed formulas**. Both constant and variable (semantically unknown, defined as a formula syntactic transformation) metapredicates are required for the formal anecdote representation.

The interpretation function maps the ML metapredicate into relations between the objects (universe elements) and the **syntactic relation between its formula arguments**. We do not force any constraints for the metapredicates, except that the interpretation function does not necessarily map each well-formed formula. If a derived formula cannot be interpreted, more inference steps could be required to achieve the ability for it to be interpreted. We distinguish applied metapredicates (interpretable in the problem domain), in contrast to pure (syntactic) metapredicates, which transform a formula or express its syntactic feature like *var(Term)* from the standard PROLOG.

Traditional axiomatic serves the purposes of finding the proofs to theorems given fixed axioms. Scenario oriented formal system should be able to yield explanations of R having H and K with mutual contradictions and lack of data, redefining predicates from K (axioms) and making K closed. Derived K could be incompatible with original K, but it describes more (Herbrand) universe U elements in the following sense. For each subset $U_{orig} \subseteq U$ and such formula φ_{orig} from K original that it is satisfied under substitution of all elements from U_{orig}, such formula $\varphi_{derived}$ from K derived and the set $U_{derived} \subseteq U$ exists that $U_{orig} \subseteq U_{derived}$ and $\varphi_{derived}$ is satisfied under substitution of all elements from

$U_{derived}$. We call this property **relative soundness**: to make the system sound, we have to add new (Herbrand) universe elements to the system which satisfy metapredicates together with the lower order formulas whose validity we intend to achieve (see assumption enforcement rule below).

MS allows **automatic derivation** of the concepts **upon request**. An unknown metapredicate first appears as a syntactic transformation of the given formulas in the target one. Then its variables could be instantiated by the universe elements, when this metapredicate becomes converted into applied one as only its definition through the other predicates has been derived. More details are presented in Section 3.2.

3.2 The inconsistency shift rule of metalanguage transfer and the syntax metapredicate attenuate

The natural way to derive a new clause, which matches the current universe, is to take original one and to apply a **syntactic transformation**, achieving the satisfaction by this universe. No modification is required if the original clause is satisfied with this universe, so the derived and original clauses are the same. The difference between the arguments of the original clause and the derived clause is expressed via metapredicate representation of this transformation.

In contrast to general validity in algebra and logic, MS can turn any formula φ into a valid one if there exists such metapredicate Ψ, that it can be satisfied by φ and a valid formula φ_{Satisf}, which we call **inconsistency shift** of φ:

$$\frac{\varphi}{\varphi_{Satisf},\ \Psi(\varphi, \varphi_{Satisf})}$$

Ψ can be defined, for example, as a transformation, which gives the strongest φ_{Satisf} in the sense of minimum number of the universe elements (constants) to satisfy it. We build the metapredicate

attenuate*(OriginalFormula, SatisfiedFormula),* which constructs *the SatisfiedFormula* applying minimum possible transformations to make it run successfully, given *OriginalFormula*.

Formally, ***attenuate***$(\varphi, \varphi_{Atten})$ holds, if φ_{Atten} is the valid formula, which is the strongest in the following sense. There is no such formula φ_{Atten}‘ that its collection of fitting subsets is included into the collection of fitting U subsets for φ_{Atten}, $U\varphi_{Atten'} \subset U\varphi_{Atten}$. A fitting subset of a formula is a universe subset such that there exists the satisfying instantiation of this formula with the elements of this subset, when each element occurs at least once. A formula yields a collection of fitting subsets at the universe.

The following list of possible syntactic transformations for ***attenuate,*** turning φ into φ_{Atten}, is sufficient to reach the relative soundness and requirement for knowledge representation *KU*, enumerated in Section 2.3. The clauses are presented for predicate ***attenuateStep***, which is the internal implementation of the recursive search step for ***attenuate***:

1) If the direct clause modification affects its satisfaction "too strongly", attenuation could descend to the definition of some term of the currently attenuating clause.

Subdefinition shift turns a_i from original clause to a_{iAtten} , which is defined by means of the attenuated clause for a_i *:*

attenuateStep *(*$a_1,\ldots a_i ,\ldots a_n$*, Result):- Result =* $a_1,\ldots a_{iAtten},\ldots a_n$*,*
attenuateStep *(*$b_1,\ldots b_k,\ldots b_m$*, SubDefResult),*
a_{iAtten} *:- SubDefResult.*

2) The **negation shift** is required to strengthen the negation of a term as an oppose to its attenuation:

attenuateStep*((…,not p,…),Result) :-* ***attenuateStep*** *(*$p_{Strength}$*, p),* ***attenuateStep*** *((…,*$p_{Strength,}$*,…),Result).*

3) We schematically draw the **disconnection enforcement** rule

attenuateStep*((…,*p_n *(…,X,…), …,* p_m *(…,X,…), …) ,*
(sub(X1/X (…,p_n *(…,X,…), …)) , sub(X2/X* p_m *(…,X,…), …)).*

A clause becomes weaker, if we eliminate the constraint for the pair of predicates p_n, p_m by means of breaking the occurrence of their common argument X into X_1 from p_n towards left and X into X_2 from p_m towards right*).* The transformation could be applied if there is at least one universe element $u\in U$ such that p_n *(…,u,…)&*p_m *(…,u,…)* holds. *sub(*X_1*/X* φ) denotes the substitution of the variable X_1 instead of X in the formula φ.

We present the meta clauses for the recursive search of ***Attenuate*** in details. First, attenuation of a clause, satisfied in current universe is itself:

attenuate*(*φ *,* φ_{Satisf}*, ListOfFittingFor*φ_{Satisf}*) :-* φ*,* $\varphi_{Satisf}=\varphi$*.*

To set the search of the least attenuation result for a formula, we accumulate the list of collections of fitting areas for various transformations in *BestListsSoFar*. We add the third argument *ListOfFittingFor*φ_{Satisf} to the internal version of ***attenuate*** to keep the collection of fitting areas for the current $\varphi_{Satisf.}$

attenuate*(*φ_{Step}*,* φ_{Satisf}*):-* ***attenuate****(*φ_{Step}*,* φ_{Satisf}*,[ ]).*
attenuate*(*φ *,* φ_{Satisf}*, ListOfFittingFor*φ_{Satisf}*) :-*
accumulate(BestListsSoFar),
% it is not superset of any member of accumulated list
not(member(MemberBestListsSoFar, BestListsSoFar),
subset(MemberBestListsSoFar, ListOfFittingSubsetResult),
% if *ListOfFittingFor*φ_{Satisf} is a subset of any member of accumulated list, it replaces
%it, otherwise it is added to *BestListsSoFar:*
ifthen ((not (member(MemberBestListsSoFar, BestListsSoFar),
subset(ListOfFittingSubsetResult, MemberBestListsSoFar),
replace(ListOfFittingSubsetResult, MemberBestListsSoFar,
BestListsSoFar)
), add(ListOfFittingSubsetResult, BestListsSoFar)
), update(BestListsSoFar).

We use the internal metapredicate ***attenuateStep*** to search through all possible transformations, leading to the totality of φ_{Satisf} , closed in respect to inclusion of the subsets of fitting areas:

attenuate*(*φ *,* φ_{Satisf}*, ListOfFittingFor*φ_{Satisf}*):-*
attenuateStep*(*φ *,* φ_{Step}*, ListOfFittingFor*φ_{Satisf}*Step),*

attenuate$(\varphi_{Step},\ \varphi_{Satisf}, ListOfFittingFor\varphi_{Satisf}Step)$.

One of the main capabilities of MS is to consider metapredicate ***attenuate***$(\varphi_{Step},\ \varphi_{Satisf})$ unknown object-level predicate and express it via the known metapredicates. This is the base of MS algorithm for invention of a new first-order predicate, adjusted for a new PD by means of formula transformation metapredicates.

We define the equality relation on the pair ***attenuate***$(\psi, \psi_{Satisf}\ )$, ***attenuate***$(\varphi\ ,\ \varphi_{Satisf})$, expressing satisfied ***attenuate*** via ***attenuateStep*** in a non-recursive way. Representing formula argument as a variable, we check the condition (***attenuate***$(\psi, \psi_{Satisf}\ )=$ ***attenuate***$(\varphi\ , \varphi_{Satisf})$). If it holds, we obtain the **system of equations** for unknown object level predicate, represented by the variable metapredicate ***attenuate***.

3.3 The rules of variability shift and assumption enforcement

The rule of **variability shift** transfers the syntactic property of being a variable from the formula variables (metavariables) to the first order variables under the transition from metalanguage to language - object. The rule introduces a new way to interpret the implemented metapredicate $\boldsymbol{\Psi}$ as an unknown predicate P_Ψ of the language-object with the variables, which uniquely satisfy the formulas $\varphi_1(X_1,\ldots)$and $\varphi_2(X_2,\ \ldots)$ as arguments of this metapredicate. This rule is useful when we can apply variability shift to more than one occurrences of $\boldsymbol{\Psi}$ and then substitute P_Ψ expressed from one result formula into another:

$$\frac{\boldsymbol{\Psi}\ (\varphi_1(X_1,\ldots),\ \varphi_2(X_2,\ \ldots))}{P_\Psi\,(X_1,X_2)}$$

Variability shift could be considered as an inverse to the inconsistency shift, but the former typically could not be applied immediately after the latter unless we have explicit evidence of the uniqueness of $\varphi_1(X_1,\ldots)$ and $\varphi_2(X_2,\ldots)$. Basically, this is an explicit way to yield a new concept P_Ψ in the conditions of uncertainty of the semantics of $\varphi_1(X_1,\ldots)$ and $\varphi_2(X_2,\ \ldots)$.

Knowledge of specificity of modal metapredicates allows us to adjust the inconsistency and variability shift rules to the needs of anecdote reasoning. The necessity to obtain $\varphi(Modal_1(Who,\ Whom,\ \phi\ ))$ makes us to enforce the assumption $Modal_2(Who,\ Whom,\ \phi\ )\rightarrow Modal_1(Who,Whom,\phi\ )$. We conventionally add it to KU and continue the inference. If this assumption is not employed by any clause ε in the course of inference, it is rejected together with the formula $\varphi(Modal_1(Who,\ Whom,\ \phi\ ))$.

Hence we have the following rule (the clause ε is implied by the assumption):

$$\frac{\varphi(\ Modal_2(Who,\ Whom,\ \phi\ )),\ Modal_2(Who,\ Whom,\ \phi\ )\rightarrow Modal_1(Who,\ Whom,\ \phi\ ){:}\text{-}\ \varepsilon}{\varphi(\ Modal_2(Who,\ Whom,\ \phi\ )),\ Modal_2(Who,\ Whom,\ \phi\ )\rightarrow Modal_1(Who,\ Whom,\ \phi\ )}$$

The **assumption enforcement** rule is intended for the reasoning in the environment of high ambiguity caused by the predicates of mental states and actions (modal). They derive the closed system: each of these predicates can be defined via the others in multiple ways.

This property provides the ability to combine the various meanings of the pair of modal predicates to build an explanation for
particular anecdote. For instance, the fixed set of definitions of the knowledge properties ([17]) is not appropriate to represent an arbitrary scenario.

The following definition introduces the series of meanings, which are expressed by the word "cheat". The meanings "cheat by action", "implicit", "unsuccessful", "without intention" could by derived under substitution into the metapredicate ***Attenuate*****,** choosing alternative definitions for the modal predicates *know, want, say, believe***.**

cheat(Who, Whom, SayFake):-
know(Who, Real), want(Who, believe(Whom, SayFake)), Real=/=SayFake,
say(Who, SayFake),believe(Who, trust(Whom, SayFake)),
believe(Who, hear(Whom,SayFake), believe(Whom, SayFake)).

We note the difference between our approach and the axiomatization of knowledge and belief (for example, [17]). Any fixed set of axioms and their (possible world) model can not handle AC, where the meaning of knowledge and belief varies even within a single anecdote. Furthermore, the concepts "to know a feature of objects" versus "to know a fact" explicitly need MS to be defined (see Section 4.2).

In general, the formula proof could be split into the proofs of modal and non-modal components. To match a term with its target (instance), we first check the correspondence of the non-modal predicates. In case of success we can achieve the formula validity applying assumption enforcement**.** It would mean that the modal predicates from this term become redefined to match the corresponding modal predicates from the target term. These modified modal predicates should participate in the definition of something independent (clause ε).

4. Scenario synthesis

4.1 Generation algorithm

Below is the order of the formal scenario synthesis (from scratch):

1) Choosing the problem domain and *KU*; verification of *KU* properties (Section 2.3). Choosing the atomic humor predicates.
2) Generation of *H*, checking consistency with *KU*. Achieving some degree of freedom from *KU*.
3) Application of the full *KU* to *H* and derivation of R_{dissip}
4) Generation of an arbitrary H_{dissip} using the similar criteria as *H*. H_{dissip} should be symmetric to *H* relatively *KU* in the sense of the set of modal metapredicates.
5) Application of the full *KU* to H_{dissip} and derivation of the statements with the same non-modal predicates as in R_{dissip} .
6) Choosing the specific *KS,* which ties together *R* and *H*.
7) Verification that the resultant system obeys anecdote criteria.

$$
\begin{array}{ccccc}
(1) & & (2) & & (3) \\
KU & & H & \rightarrow & R_{dissip} \\
 & & \downarrow (7) & & \\
H_{dissip} & \leftarrow & R & & KS \\
(4) & & (5) & & (6)
\end{array}
$$

The search is based on the standard backtracking mechanism in logical programming, similar to a search in the object-level . In case of failure of the step *k,* the search is repetitively run for the step *k-1, k=2…7.*

4.2 A synthesized scenario

We use the above algorithm for the design of the Scenario Synthesis system, which is aimed to automate the creation of Internet advertisement scenarios, to raise their attractiveness, make them more natural and remarkable and increase the composition performance. The scenario is composed in accordance with the presented synthesis order. It could be automatically multiplied for various products such that each product could be identified based on the unique logical frame. The automation of the scenario synthesis is based on the multiple reinvention of meanings of the linking metapredicate (in this case, of ***explain_by_example***).

The story below helps to present the Internet advertisement of P200 computer with CD-ROM. The product typically includes the picture and the list of parameters (see http://www.firstauction.com for example of implementation).

We start with the commonsense definition of the concept *"how to explain_by_example, what feature could a particular object possess". Who* explains *Whom* what *Feature* could possibly *Object* possess, using the *FeatureList* as the set of examples. The following has to hold to give *Who* an ability to explain *Whom* the *Feature* of the *Object: Who knows* that *Feature,* and *Whom* does not *know,* what could be a *Feature* of the *Object.* Then *Who informs* (*says, shows, writes*, etc.) *Whom* about the example *FeatureList.*

(KU.1) ***explain_by_example****(Who, Whom, Feature(Object, FeatureList)) :-*
know(Who, Feature),
not know(Whom, Feature(Object, FeatureList)), know(Who, Object),
inform(Who, Whom, FeatureList).

It seems rather hard to fit two following concepts in the single definition: one explains some **facts** about objects and one explains the concept of having **a feature** for a given object. The suggested definition for the latter concept is not subject to further generalization.

Besides *KU.1,* the anecdote agents possesses the basic concepts such as *want, ask, respond, know.*

Some agents are familiar with the definition for *parameters* (to be able explain to another agent*).*

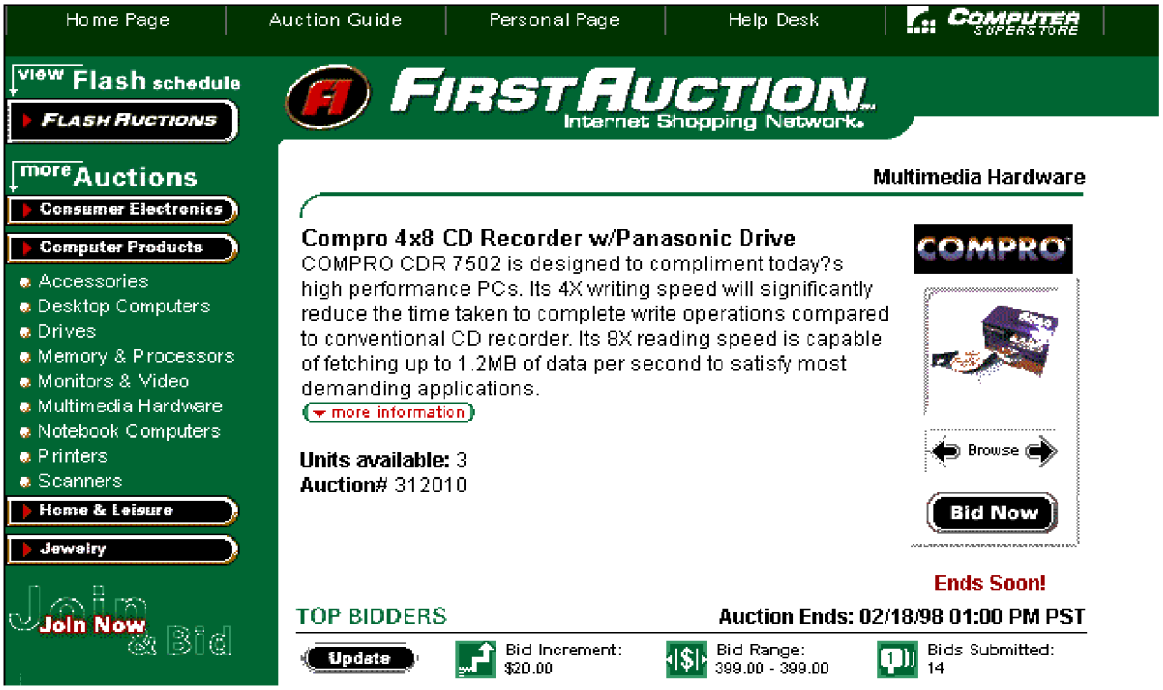
We construct the *H* component of our scenario for the agents *bill* and *doug*:
(H.1) want(bill, (find(bill , Cd_rom), insert(bill, Cd_rom, computer))).
not know(bill, find(bill, Cd_rom)).

(H.2) ask(bill, doug, advice(doug, bill, find(bill, Cd_rom))).
(H.3) ask(doug, bill, (Feature(computer, FeatureList), insert(bill, cd_rom, computer)))).
(H.4) not know(bill, Feature (computer, FeatureList)).
(H.5) ***explain_by_example****(doug, bill,*
parameters(computerDoug, [freq(200),ram(64), hd(2000)]) .
(H.6) respond(bill, doug, patameters(computer, [freq(133),ram(16), hd(1000)]))).
Now it't time to stop and start working with the punch line:
Bill just buys the CD-ROM after he understand the Doug's explanations:
(R_{dissip}) insert(bill, cd_rom, computer).

Doug wanted Bill to have more practice in buying CD-ROMs and suggested buying it for his own computer first. The second-order (predicate) variable *Feature* from the definition *KU.1* is instantiated by the predicate name *parameters.*
($H.5_{dissip}$) ***explain_by_example****(doug, bill,*
parameters(computerDoug, [freq(200),ram(64), hd(2000)]) .
ask (doug, bill, insert(bill, Cd_rom, computerDoug)),
want(doug, learn(bill, parameters(computerDoug, [freq(200),ram(64), hd(2000)]))).
We finally approach the construction of the linking metapredicate:
(KS) ***prefer_participate_in_better****(Smb,Smth,SmthBetter):- want(Smb,Smth), better(Smth,SmthBetter), suggest(Smb, SmthBetter).*
This metapredicate participates in the rule of inconsistency shift, deriving *suggest(bill, insert(bill, cd_rom, computerDoug))* from *want(bill, insert(bill, cd_rom, computer))*
with the help of *better(computerDoug, computer*). This is an example of inconsistency shift rule, applied to
a non-syntactic metapredicate ***prefer_participate_in_better,*** (in contrast to ***attenuate***).
(*R*) *suggest(bill, doug, insert(bill, cd_rom, computerDoug))).*
Finally, we presented the manually composed textual representation. A reader has probably imagined something similar.

Bill wanted to get a CD-ROM for his computer. He called his friend, Doug and asked him which one he would get. Doug wanted to know what Bill's system parameters were and Bill said,
"What do you mean by "system parameters?"
Doug said, "Well, that's the performance and memory. For example, I've got P200 with 64 Megs of RAM and a 2 Gig hard drive".
Bill said, "Oh, I understand. Let's see here... I have a 133-Megahertz processor with 16 Megs of RAM... Actually, how about I buy a CD-ROM for your machine..."

The input of a scenario synthesizer contains the description of the problem domain: formalized world knowledge, required to link *H.1...H.6* with each other and with *R*.



Bill wanted to get a CD-ROM for his computer. He called his friend, Doug and asked him which one he would get. Doug wanted to know what Bill's system parameters were and Bill said, "What do you mean by "system parameters?" Doug said, "Well, that's the performance and memory. For example, I've got P200 with 64 Megs of RAM and a 2 Gig hard drive".Bill said, "Oh, I understand. Let's see here... I have a 133 Megahertz processor with 16 Megs of RAM... Actually, how about we buy a CD-ROM for your machine..."

The prototype of the anecdote generation system, developed in Arity Prolog for PC, created a series of scenarios for advertising of computer products for Internet Shopping Network,Inc.

4.3 Interactive Scenario Synthesizer

Traditional Internet advertisement is based on the text, pictorial and sound presentation components. The semantics of an advertisement, aimed to convince a potential buyer to purchase an item by means of a successful and impressive example, is usually the results of a manual composition. The **Scenario Synthesizer** (SS) is developed to automate the creation of scenarios for the human composer. It is designed to raise the appeal of an advertisement, to make it more natural and remarkable and to increase the composition variety and performance. The behavioral specifics of the agents in a short advertisement story allow to control the advertisement impression, and to target it to the particular audience. If a product database contains a few thousand items, and the range of products significantly varies in time, manual composition or handling of such stories becomes impossible. Each product or service item must be associated with a unique story, which is assumed to help a buyer, browsing an Internet shopping site, to identify this product.

Such a story could be an example of a successful and satisfied buyer, who has come through some difficulties, interference with the other story agents and finally took advantages of the features, inherent to the particular product. The story, which is intended to make an advertisement more attractive and easy to recall, must be short, reasonably logical, call the series of associations, and end with an emergent positive result. We suggest considering such kinds of stories as **logical anecdotes** (scenarios), and to formulate the task of synthesis of a manifold of such stories as the **scenario generation problem.**

SS task is tightly connected with the specific model of uncertainty, because it delivers an imaginative freedom to the audience. The stories with the same level of strictness and determinism as that is accepted in mathematics ("strict scenarios") would not be attractive and the number of possibly generated scenarios would be rather limited. Sophisticated human reasoning in the high uncertainty environment, required to perceive a story, assists in memorizing of its basic points.

SS design is based on the model of a logical anecdote (scenario). Analysis of the logical structure of a formalized scenario allows us to derive the mechanism of synthesis of a formal scenario, that satisfies the **anecdote criteria**. Knowledge of the key and variable logical components of a formal scenario gives the ability to generate the **spectrum of scenarios** having the fixed set of participating agents and their properties.

4.4 A session with SS

SS could be viewed as something similar to code generation software. Ad designer fills the input fields such as the advertising targets, scenario agents with their properties and interactions, etc. The SS yields the formal backbone scenario. Ad designer tries to understand it and likely has to change some accents. It is typically necessary to modify some agent properties to achieve that which is usually impossible to express explicitly. After a few such iterations the ad designer finally comes to conclude whether the input data can yield satisfactory scenario. In case of success SS is finally run with the option "to generate the series of scenarios". This option gives the most advantage of using SS, because a single scenario could be generated even faster without SS, but it definitely wins generating a spectrum of scenarios given the same input data.

Since the task of the scenario generation is rather hard, in spite of the significant SS assistance, an ad designer should possess developed logical and analytical skills together with fluency in basic programming concepts (structural, OO, logical). The practice of the SS prototype testing showed that it is significantly harder for an ad designer to come up with an original scenario than to translate a formalized one into NL, having the training period of a few days. So the task of NL synthesis seems less important than the scenario synthesis.

The tool introduces significant help in the design of the text advertisements in the following conditions:

1. Limited and strictly circumscribed problem domain.
2. Necessity to create a number (thousands) of distinguishing scenarios for the similar subjects (products).
3. The automated procedure of access to an ad library.

Besides mentioned in the introduction, SS possesses the following advantages in respect to manual ad composition:

1. Scenario storage and reusability for other problem domains.
2. Easy transfer from one to another problem domain having the similar set of agents.
3. Fast object (product) search. The formal scenarios are rather appropriate for the product search given the implicit criteria than the NL advertisements.

4.5 Another example of a synthesized scenario

Bill recently opened a new insurance office. He tried to call his friend Doug for help in finding a computer for the office, but only found Doug's secretary there, Sally. "By the way, would you know something about computers for a small office. My trouble is, I know nothing about them." Sally's friend, Pat, was to start working for Doug in a week. Sally knew Pat was crazy about computer games with color graphics, so she suggested to Bill for him to buy one of the MMX computers with a ZIP drive so that Pat can play games after hours and switch ZIP discs.

Doug returned Bill's call shortly after that and recommended more emphasis on the modem and sound card so that he could save money on long-distance and do business remotely. Doug was skeptical about the ZIP drive because he thought Bill does not need that much hardware on his machine.

So, Bill decided to make another call to his new employee, Pat, at home. As she picked up the phone, Bill heard strange noises as if Pat were flying a small airplane. "What's happening!" he yelled out and then realized it was just a flight simulator game. "Wait...wait!" - he thought, how about if people call my insurance office and hear really great crashing noises. That's what I need my office computer for!

Bill immediately hung up the phone and ordered a full-multimedia computer system.

The basic idea of this scenario could be briefly drawn in the following way:
(KU) …
(KS) If a person has two advises and asks a third person which advise to follow, some emergent information could affect the decision …
(H.1) want(bill, advise(Who, bill, buy(bill, Computer))).
(H.2) know(sally, play(pat, mmx&zip) → advise(sally, bill, buy(bill, mmx&zip)).
(H.3) advise(doug, bill, not buy(bill, mmx&zip)).
(R) hear(bill, crash_noise) → know(bill, buy(bill, mmx&zip)).

5 MS in genetic algorithms

MS delivers to genetic algorithms (GA) an ability to use the higher expressive language in the search. The search space becomes more sophisticated, but the optimal point gives the higher score than that of the simpler search space, based, for example, on all the possible disjunctive normal forms in the basis of the fixed set of terms.

For the simplest example, if we want to improve the system of rules, consisting from addition and multiplication, it would possibly help to add the exponent operation such that it could be adequate for a particular problem domain. The improvement step of higher generality (and, possibly, efficiency) would be to add some heuristic constraints to how the expressions with exponents can be reasonably combined.

MS provides rather expressive language for building the rules. This language includes the formalized heuristics as syntactic metapredicates. The rules themselves could contain the object-level predicates, second order predicates which range over the explicit set of the search space features and applied metapredicates.

We show that the scenario framework is adequate to represent an environment for GA learning. We base our presentation on the "artificial ant problem" [18-20], a traditional problem domain to illustrate the various approaches to GA. An ant happens to find itself in a new environment given the simplest possible actions it can perform.

To survive in this environment, the ant has to build the new complex actions, achieving better results for this environment. To move faster in a new environment, the ant must form the appropriate motion frameworks from the primitives like move one step forward, backward, right, left. For example, in a 3D environment the ant would create the concept "jump" from the primitive concepts "move towards the object", "move up", "move above the object" and "move down".

Why would we need MS to combine the enumerated concepts? We need the mechanism to "glue" the rules for each concept to obtain the more complex rule with higher performance. MS is necessary to control the instantiation states (see Sections 1.1 and 6 for more details and examples) of the predicates from these rules such that they agree. Besides, attenuation of the resultant rule is typically required.

We choose the simplest environment of a 2D maze.

(*K*) The ant knows 4 basic one-cell moves and can observe the neighbor cells at each position.

(*H*) The ant runs through a particular maze.

(*R*) The ant creates the move patterns for this maze and designs the high-performance rules.

The reaction call to this scenario will be

want(ant, (find(ant, Path), go(ant, AnotherPath), Path < AnotherPath)).

want(ant, (find(ant, Path), go(ant, AnotherPath), Path < AnotherPath)).

The dissipated scenarios (see 2.3) are ones with the same rules (R_{dissip}) and with the maze, which contains only specific move patterns from *R* (H_{dissip}).

GA system is based on the additional inference rule. We introduce the metapredicate ***score****(OrderedSetOfRules, ScoreValue)*, which calculates the perfomance criteria, *ScoreValue*, given the *OrderedSetOfRules*. At each state the ant tries to satisfy the rule conditions in that order and follow the first rule which succeeds.

We derive the following MS inference rule from the inconsistency shift for GA:

$$\frac{\textit{OrderedSetOfRules},\ \mathbf{score}(\textit{OrderedSetOfRules, ScoreValue}),\ \mathbf{score}(\textit{BetterRules, BetterScoreValue}),\ \textit{BetterScoreValue} > \textit{ScoreValue}}{\textit{BetterRules}}$$

Now we consider GA as a methodology to obtain *R* by means of the inference search, based on this MS inference rule. GA provides the heuristics to build the reasonable cover of the search space for *BetterRules.*

The reaction call for the scenario will now be *want(ant, (find(ant, BestRules))).*

In our example the score is based on the number of moves necessary for the ant to reach the target cell from the fixed initial position. The maze is represented by a square matrix (0 – passages, 1 – walls), see Figure 5. From any cell the ant can move to any of 4 neighboring cells (diagonal cells excluded) if it is not a wall. It has initially the ordered 4-tuple of rules (one rule for each direction) and chooses the first one it could satisfy. The ant is capable of keeping the limited number of rules (less than 8). It knows where the target cell is, where its current position and is capable of memorizing of all the visited cells and seeing the neighbor cells.

The ant accumulates the experience of moving for a particular maze and builds the system of rules, adjusted for this maze. To minimize the number of moves to the target cell, the ant derives the rules, which allow the farther move (evidently, checking that there is no wall in the intermediate cells). On one hand, the longer moves decrease the number of steps, but on the other hand some moves could be missed.

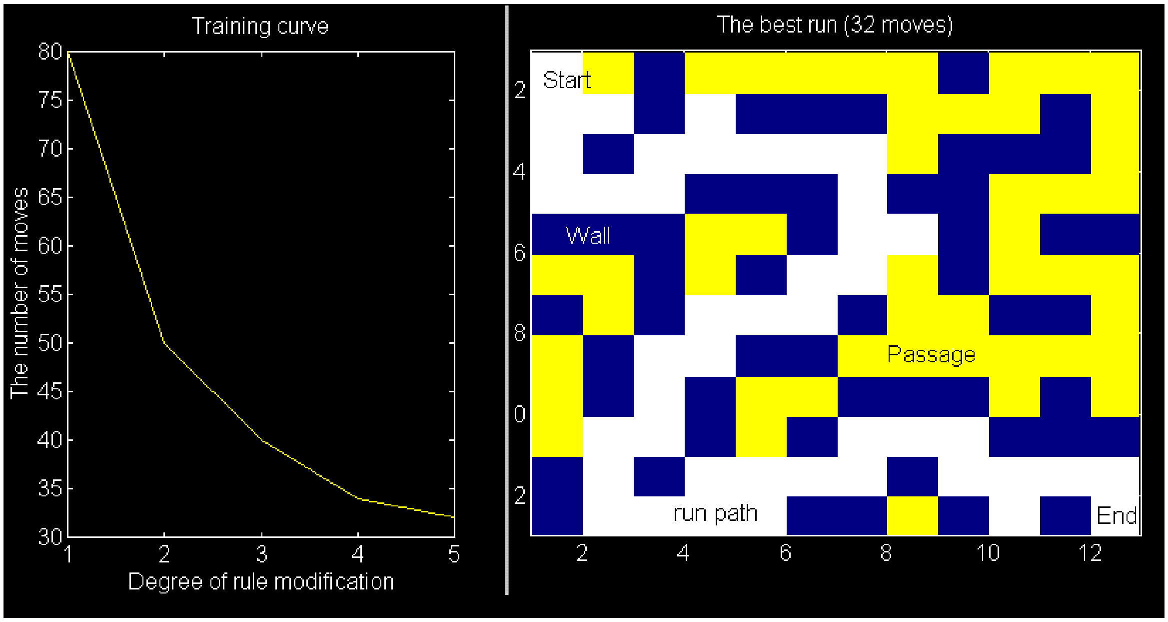


Figure 5. Left: the training curve. The number of moves for the current rule modification degree is shown. Initial number of moves (for the basic move rules) is 80. Resultant number of moves for the rules in Section 5.1 is 32. Right: the training maze; walls are black and the passages are gray (the actual path is white). The ant is optimizing its path from the upper left to the bottom right corner.

A trivial approach would be to build the basis of rules of various complexity (move lengths) and then to perform the exhaustive search to find the optimal combination from this basis. That combination would include the simple rules for "local" moves not to miss the better passage, and the complex rules, which provide the "faster" moving. But the building of this basis, and, what is worth, implementation of this exhaustive search is rather hard computationally.

The approach we suggest gradually increases the rules' complexity, replacing the local rules, non-efficient for particular maze, by the rules with the longer move. Note that we present just the skeleton of the clauses, sufficient to understand the declarative algorithm. The full program is available at http://dimacs.rutgers.edu/~galitsky/MS/maze.ari .

We proceed to the move definitions. First, we check whether we have reached the target cell with coordinates `(A,B)`

```
reach(X,Y,Step,Path):- Step=1, init(A, B),
                       A=:=X, B=:=Y. %A,B - initial coordinates
reach(X,Y,Step,Path):- (X >=0) , (Y>=0) , (9>=X), (9>=Y), % within maze
              not step(X, Y), % hasn't been here before
              not wall(X, Y), % this isn't a wall cell
              assert(step(X, Y)),     %memorize where has been
              nextReach(X,Y,  Step1, Path), %where to go
              inc(Step1, Step).
```

The predicate `reach(X,Y,Step,Path)` denotes that the ant could be in the cell `(X,Y)` in

`Step` moves from the beginning after she visited the cells `Path= [(X1,Y1),…, (Xstep,Ystep)].`

The rules of the move choice are expressed `nextReach(X,Y,Step,Path)` expresses that the ant could be in the cell `(X,Y)` in `Step` moves from the beginning after she visited the cells from the list `Path`.

Now we present an initial list of rules (just two of them). Informally, these rules use information about the mutual location of the ant and cheese, and the previous move history to make decision about the next move.

```
([(nextReach(X,Y,  Step1, Path):- init(Xinit, Yinit),
                   sign(X1, Xinit,X), XX1 is X+X-X1,
                   append(Path, [(XX1,Y)], Path1),
                   reach( XX1, Y,Step1, Path1)
),(nextReach(X,Y, Step1, Path):-
                   init(Xinit, Yinit),
                   last(Path, (Xl,Yl)),
                   X1 is 2*X-Xl;  Y1 is 2*Y-Yl;
                   append(Path, [(X1,Y1)], Path5),
        reach( X1, Y1, Step1,Path5)
  )
]).
```

The former rule uses the predicate `sign` to determine the direction of current move:

```
sign(Unit,To,Number):-ifthen(var(Number),
             ifthen( ( To-Number<1), Unit is Number-1),
             ifthen( (To-Number>0), Unit is Number+1).
```

We presentation the metapredicate `train` of the procedure which creates the best rule.

```
train(Rules, NewRules):-
            abolish(nextReach/4), rules(Rules),% clear the rules.
    findall( (Comb,Step,NewRule), (
                 %search through all possibly derived rules and their combinations
                   %with the existing system of  rules
          member(R1, Rules), member(R2, Rules), R1 \== R2,
          [! mergeRules((R1, R2),  NewRule),
                 %choose any pair of rules and merge them
               append([NewRule], Rules, NewRules),
               subcombination([1,2,3,4       %,… initial rule order
                       ], BasicComb),
                      %derive an order on the set of current and newly generated rules
      append([0], BasicComb, Comb),%new rule has number 0
               [!abolish(nextReach/4),       %  eliminate current rules
                  init,
                  sublist_assertz(NewRules,Comb) !] !],
                                          %  asserts certain rules from the list  Comb
          reach( 1, 1, Step,[]),
          findall(S, step(S,_), Ss), len(Ss, Step1)
                %calculate the number of steps to the food, having the current
                %ordered set of old rules and certain position of the newly
                %generated rule in this set
         ),
         (CombsSteps)   % accumulated triple (list of rule indexes, number of
                        % steps, new rule)
      ),
 findall(M, member( (_,M,_), (CombsSteps) ), ToSorts),
         [!
           quicksort(ToSorts, Sorteds),
                %sort the list of ordered set of rules from minimum number of
                %steps to maximum steps
              sublist(0,1,Sorteds,[Max]),
```

```
            member((BestSeqs,Max,NRule), (CombsSteps)) !],
            abolish(nextReach/4),
            append([NRule], Rules, NewRules),       %rule refreshing
            sublist_assertz(NewRules, BestSeqs).
```

The metapredicate **mergeRules**((R1, R2), NewRule) combines the rules R1 and R2 to extend the move. First, it eliminates the call of reach from the clause for R1. Instead, the output of R1 (variables, instantiated during the call of R1) are subject to evaluation by the following rule, R2, which finally calls reach with determined values for X,Y and Path.

```
mergeRules((R1, R2),  NewRule):-
     [! var_var(R1, R1vv),   % we need to get a copy of an uninstantiated
                             %formula before we subject it to modification.
     var_const(R1vv, R1c), R1c = (H1c :- B1c), %start with R1
   clause_list(B1c, B1cs), %convert the clause body into a list
   cl_list_search( [reach(X,Y,Step,Path )],B1cs, Off, Outs),
     cl_delete( reach(X,Y,Step,Path ), B1cs, B1csd),
         %delete reference to reach/4 from the left part
     clause_list(Out,Outs),        % we get the output variables of R1
     var_const(R2, R2c),                    %then we process R2
     R2c = (H2c :- B2c),
     clause_list(B2c, B2cs), H2c =.. [_|In_s],
     clause_list(In,In_s), % we get the output variables of R2
   append(B1csd, [ (Out) = (In) ], B1CsdOutIn),
         %we add the term, mapping output of R1 into input of R2
   append(B1CsdOutIn, B2cs, Bodysc),   % main split R1 and R2
     clause_list(Bodyc, Bodysc),
     HB = (H1c :- (Bodyc))!], var_const( NewRule, HB ) .
```

In our notations, predicates transformations of the type "variable ↔ constant", "clause body ↔list" and "clause ↔its head and body" are shown with extended margins. Comments are at the left or below the expressions.

We focus on some peculiarities of the meta-programming technique concerning the operations with the **formula with non-instantiated arguments**. There is a necessity of making a copy of an expression with variables for modification (the metapredicate **var_var**(R1, R1vv)). If we have a variable term T and want to modify it, we need to obtain its copy Tv with different names of variables. Then Tv is converted to Tvc by replacing variables by constants such that one-to-one correspondence of equal variables and equal constants is observed. Tvc itself is subject to modification into Tvc^m, followed by conversion of constants into variables $Tvc^m v$. The latter is inverse to Tv → Tvc in a sense that it maps equal constants to the equal(proimage) variables. Finally, we get T and its modified result $Tvc^m v$ with the different naming for variables. Because of these procedures we avoid interference (unwanted instantiation), calling original and modified clauses T and $Tvc^m v$.

5.1 Resultant rules

We present the resultant rules, which improved the number of moves from 80 to 32. These 7 rules are derived from the initial 4 rules. We have one letter coding for variables because the rule output was performed in the internal standard of Arity Prolog Compiler (Arity Corp.).

```
nextReach(A,B,C,D) :-init(E,F), sign(G,E,A),H is A + A - G,
```

```
    append(D,[(H , B)],I), nw(H,B),
    (H , B , C , I) = (J , K , L , M),
    init(N,O),sign(P,O,K), append(M,[(J , P)],Q), nw(J,P),
    (J , P , L , Q) = (R , S , T , U),
    init(V,W), sign(X,W,S),append(U,[(R , X)],Y), nw(R,X),
    (R , X , T , Y) = (Z , AA , AB , AC),
    init(AD,AE), sign(AF,AD,Z), append(AC,[(AF , AA)],AG),
    nw(AF,AA),reach(AF,AA,AB,AG).   % extended zigzag
nextReach(A,B,C,D) :- init(E,F),sign(G,F,B),
    append(D,[(A , G)],H), nw(A,G),
    (A , G , C , H) = (I , J , K , L),
    init(M,N), sign(O,M,I),  append(L,[(O , J)],P), nw(O,J),
    reach(O,J,K,P).                 % longer move towards the target
nextReach(A,B,C,D) :-init(E,F), sign(G,E,A), H is A + A - G,
    append(D,[(H , B)],I), nw(H,B),
    (H , B , C , I) = (J , K , L , M),
    init(N,O),sign(P,O,K), append(M,[(J , P)],Q),   nw(J,P),
    reach(J,P,L,Q).             %longer move towards the target(diagonal tendency)
nextReach(A,B,C,D) :- init(E,F), sign(G,F,B),
    append(D,[(A , G)],H), nw(A,G), reach(A,G,C,H).
                                    % original rule for move towards the target
nextReach(A,B,C,D) :- init(E,F), sign(G,E,A),
    H is A + A - G,   append(D,[(H , B)],I),
    nw(H,B),reach(H,B,C,I).         % original rule for move towards the target
                                    % (vertical tendency)
nextReach(A,B,C,D) :- init(E,F),sign(G,F,B), H is B + B - G,
    append(D,[(A , H)],I), nw(A,H), reach(A,H,C,I).
                                    % original rule for move towards the target
                                    % (horizontal  tendency)
nextReach(A,B,C,D) :- init(E,F),sign(G,E,A),
    append(D,[(G , B)],H),nw(G,B),  reach(G,B,C,H).
                                    % original rule for move towards the target,
                                    % mirrored relatively the diagonal
```

We highlight the following observation for the resultant rule system:

1. The rules are sorted in the order of decreasing complexity:
   - The first rule is the most complex.
   - The last few rules are the initial rules.
2. The ant rejected the initial rule of inertia and did not derive any complex rule based on inertia move (preferred direction for the current move is the direction of the previous move).
3. The simultaneous merge of *n* rules could be represented by *n*-1 consecutive merges of the rule *k* with the result of merge of the rules *k-1, k=2,...,n.*
4. The procedure of rule modification is invariant in respect to problem domain (in particular, the specifics of rules). While deriving the new rules, we **control** the semantics by means of specifying the map between the output variables of the left rule and input variables of the right rule.
5. During the procedure of construction of the rules for the complex moves the ant independently formed the concepts of "zig-zag move", "inertia", "U-turn","island", "figure eight" "exploration move". (The last concept is a move, motivated to collect some useful information about the maze, not directly towards the food. This information could be used in the future moves in the same run.)

We conclude, that MS implementation of GA search is a strong tool for deterministic learning. MS language possesses sufficient expressive means to have the metalanguage formula cross-over and selection appropriate operation to cover the formula search space for a limited problem domain.

6 Understanding NL inquires and definitions with feedback to problem domain

The key features of NL understanding system we intend to achieve is the **flexibility** degree in respect to the **different PD** [12]. The current study investigates the NL system, which can be adjusted to the particular PD by a customer, without the help of the system designer. It is evident, that the more degree NL understanding is based on the advanced reasoning rather than on the detailed morphological and syntactical analysis, the stronger flexibility features in respect to PD we expect to obtain.

The understanding process is divided into two stages: **translation** into formal representation (reduced traditional) and its **adaptation** towards the maximum compatibility with the PD representation. We skip the traditional morphological and syntax analysis (MSA) and focus on the semantic one, requiring rather expressive formal language and reasoning capabilities. We describe the construction of all the possible translations given the MSA results and present the reasoning mechanism, which leads to obtaining the most likelihood translation (compare, for example, with [11]).

The formal scenario concept is helpful to pose the understanding problem in the extendible domain:

(K) Knowledge part of the scenario is exactly what is called PD knowledge.

(H) The set of possible translations of the sentence (ordered in accordance to the generality criteria, see details in 6.2 below).

(R) The best translation.

To develop the concept of a logical program to be appropriate for the inquiry representation, we introduce the concept of instantiation set of a predicate in a logical program (continuing Section 2.4). An argument of a predicate can be instantiated (a variable has been turned into a constant) or non-instantiated (variable). We call the *instantiation state* of *n*-ary predicate a binary function, which maps each of the predicate argument into the set *{instantiated, non-instantiated}*. There is a subset of the set of all 2^n instantiation sets when this predicate always fails (independently on the constants, substituted to its non-instantiated arguments). We call this subset the *failure instantiated set* of a predicate in a logical program. Besides, we introduce the set of all *types* of subject constant. Each argument of each predicate is assigned by the type from the set of all types $\{\tau_s\}$.

6.1 Problem domain: control of the chemical process

We control the process, involving apparatus, reactors, heat exchangers, reservoirs, pumps, etc., which are characterized by the reagent flows, temperatures, pressures, etc. The definitions of the apparatuses and flows of the process are presented in the form of clauses at http://dimacs.rutgers.edu/~galitsky/MS/t.ari

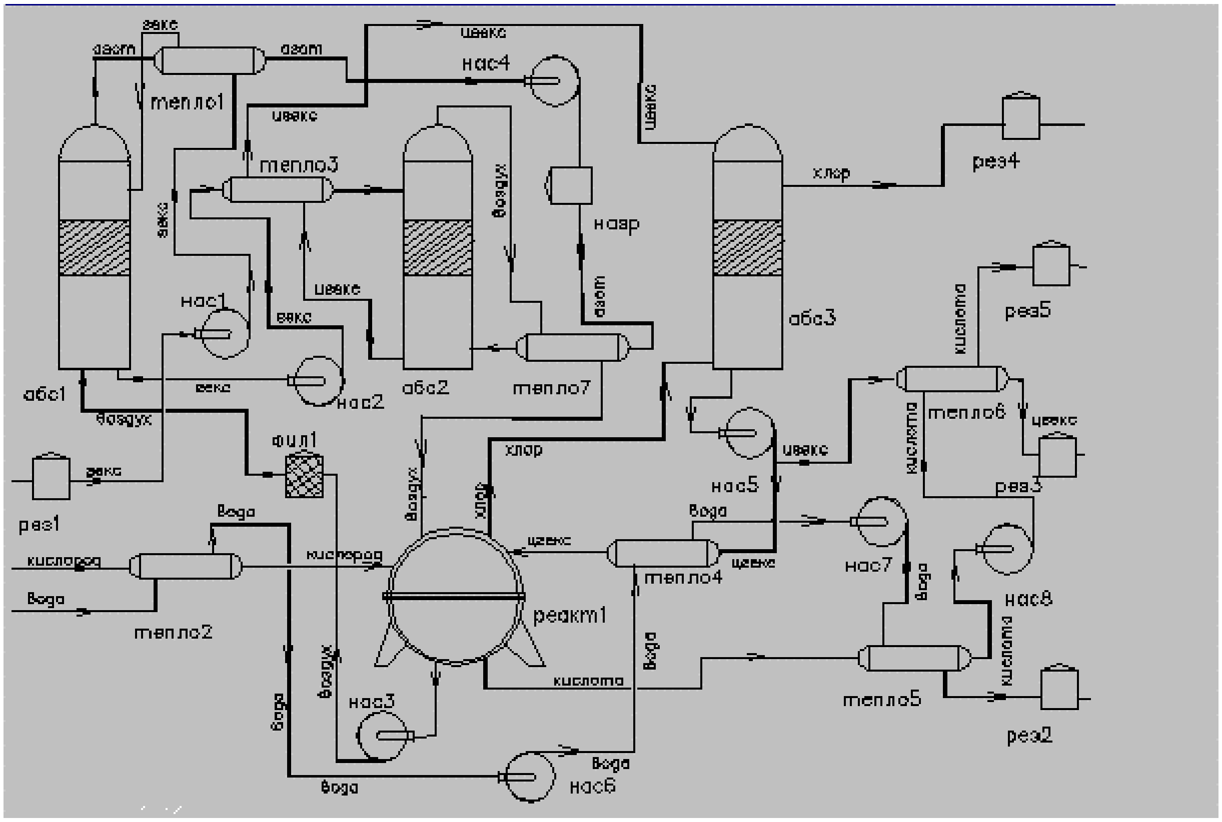


Figure 6.1. The scheme of the control process. The apparatuses and flows are subscribed in Russian, the language this system was originally developed in.

6.2 Translation from NL to ML

We represent the input inquiry as an ordered set of words *W*. MSA gives us the following mappings from the subsets of *W* and Cartesian products of *W* into the symbols and syntactic relations of our language ML (introduced in Section 3.1):

(1) Features $\subseteq$ *M* $\rightarrow$ set of predicate constants $\{p_k\}\in P$.

(2) Objects $\subseteq$ *M* $\rightarrow$ set of individ constants $\{a_s\}\in A$.

(3) Syntax links between Features $W\times W \rightarrow$ set of equalities $(X_{su} = X_{tv})$ for the pairs of arguments X_{su}, X_{tv} of the predicates p_s , $p_t \in P\times P$ and set of expressions $\boldsymbol{p_s}(\ldots, p_t,\ldots)$ if $\boldsymbol{p_s}$ is a metapredicate.

(4) Syntax links between Features and Objects $W\times W \rightarrow$ set of equalities $(X_{su} = a_s)$ for $p_s \in P$ and $a_s \in A$.

k,s,u,t,v are the indexes($\in$ Z). In addition, we have the following (“inquiry oriented”) mappings:

(5) “and”, “or”, “no”, “neither, etc. $\rightarrow$ set of propositional connectives.

(6) “which”, “how”, “what”, “when”, “where”, etc. $\rightarrow$ the key (accent) non-instantiated arguments of the predicates from *P*. We call it the inquiry *target.*

We give the examples of (1) – (4). Note that the word combinations are first mapped into the pairs of words and than to ML:

“follow” $\rightarrow$ *flow(Reagent, ApparatusFrom, ApparatusTo);*

"absorber2" $\rightarrow$ *absorber2;*
"flow to cool" $\rightarrow$ "flow","cool" $\rightarrow$ *(Reagent= ReagentCool), flow(Reagent, AppFrom, AppTo), cool(ReagentCool, AppCool, TempChange);*
"cool by 30 degrees" $\rightarrow$ *(TempChange=30), cool(ReagentCool, AppCool, TempChange).*
Below we present the procedure of deriving the series of formula translations $\{T_j\}$, $W \rightarrow T$ in the language ML. The steps of this procedure, yielding the branching of the translations (fulfilled or not) are <u>underlined</u>. That way we cover the possible semantic ambiguity of the inquiry.

1) We build the translation to start with as a formal conjunction of predicates from (1): $p_1(X_{11}, \ldots, X_{1n}) \& \ldots \& p_k(X_{1k}, \ldots, X_{kn})$. Predicates initially occur in the same order as the corresponding words from *W* occur in the inquiry.
2) If p_s and p_t are connected by the connective "or", then <u>p_t is moved behind p_s. Disjunction is inserted between p_s and p_t</u>. If there is a negation, found by MSA to be related to the word - proimage of p_q, then <u>*not* is inserted in the translation before p_s</u>.
3) If p_q is found to point to the question locus or accent of the inquiry, then <u>p_q is moved to the rightmost position</u> in the translation. One of the variables of p_q is considered to be the inquiry *target*. For example, "What pressure" $\rightarrow$ *pressure(Value,…), Value* is the inquiry target.
4) If there is some object constant a_s of the type τ_s such that there is no predicate p_k in the translation T_j having an argument of this type, than the term $(X_{su} = a_s)$, followed by the new *predicate* $p_s(X_{s1}, \ldots, X_{su}, \ldots, X_{sn})$, is <u>added</u> to the translation, X_{su} is of the type τ_s.
5) If p_s and p_t are found by MSA to be dependent, and they are of the same type, then there exists at least one variable X_{su} from p_s and X_{tv} from p_t such that $X_{su} = X_{tv}$. This equality should be inserted before p_s or p_t, whichever occurs earlier in T_j. We obtain the representation

   $\{p_1(X_{11}, \ldots, X_{1n}) \& \ldots \& (X_{su} = X_{tv}) \& p_s(X_{s1}, \ldots, X_{su}, \ldots, X_{sn}) \& \ldots \& p_t(X_{t1}, \ldots, X_{tv}, \ldots, X_{tn}) \& p_k(X_{1k}, \ldots, X_{kn})\}$

   If p_s and p_t are dependent and $\boldsymbol{p_s}$ is a metapredicate, then p_t (and its preceding equality terms) <u>are assigned to be equal</u> to X_{su}. The translation looks like

   $\{p_1(X_{11}, \ldots, X_{1n}) \& \ldots \& (X_{su} = \boldsymbol{p_t}(X_{t1}, \ldots, X_{tn})) \& p_s(X_{s1}, \ldots, X_{su}, \ldots, X_{sn}) \& \ldots \& p_t(X_{t1}, \ldots, X_{tn}) \& p_k(X_{1k}, \ldots, X_{kn})\}$

   There is no type verification for the substitution of a formula into a metapredicate
6) If p_s and a_t are found by MSA to be dependent, and they are of the same type, then there exists at least one variable X_{su} from p_s such that $X_{su} = a_t$. This equality should be inserted before p_s, where X_{su} first appears in the translation. We derive

   $\{p_1(X_{11}, \ldots, X_{1n}) \& \ldots \& (X_{su} = a_t) \& \ldots \& (X_{su} = X_{tv}) \& p_s(X_{s1}, \ldots, X_{su}, \ldots, X_{sn}) \& \ldots \& p_t(X_{t1}, \ldots, X_{tv}, \ldots, X_{tn}) \& p_k(X_{1k}, \ldots, X_{kn})\}$

   The complicated situation appears in case of antisymmetric predicates. We introduce the following convention to their instantiation states: the first antisymmetric argument (possibly) gets the value from the left predicate, the second argument derives the value and (possibly)passes it to the right. There is strong semantic correlation: these arguments are correspondingly *passive* and *active*. We use prepositions from *W* to determine which link to form: with first or second antisymmetric argument. For example, “follow from absorber3” → *follow(Reagent, absorber3, ToApparatus),* “follow to heat_exchanger1” → *follow(Reagent,FromApparatus, heat_exchanger1),* “influence on pump2” → *influence(Reagent, Apparatus, pump2).*

7) Each predicate in the translation is verified not to be in the *failure instantiation state*. It could be changed by changing the order of predicates in the translation. Any MSA result, leading to the *failure instantiation state* of a predicate in the translation, should be disregarded.

This is the end of the translation procedure. In addition to requirements of 5) and 6), we demand the satisfiability of given stand-alone predicate with the linked or substituted argument, if other arguments are non-instantiated, or non-satisfiability of this predicate with all non-instantiated variables.

Participation of the metapredicates significantly increases the language expressiveness and precision of the optimal translation search. We can use a simple combinatorial estimate of the language expressiveness as the ratio between the number of “meanings” (properly constructed formulas) and the number of constants. This ratio is increasing when we introduce the metapredicates in the translation.

We check the resultant translations on the compatibility with the domain representation (the logical program of PD). The linear order is introduced on the set of all translations based on the compatibility *degree* with the problem domain. First, each variable is assigned its weight for the score calculation: leftmost variable has the weight of 1/2, and the rightmost variable (the inquiry focus) weight is 1. If the call to a translation succeeds, its score is calculated as the weight sum of the number of constants, satisfying each variable. If the call to a translation fails, than it is subject to metapredicate ***attenuate***, and the score is calculated for the attenuated formula as a negative weighted sum. Basically, suggested score calculation algorithm is a way to represent the *generality* of an inquiry, perceived by its particular translation.

The computational experiments showed that *the best inquiry understanding corresponds to the translation with the positive score, approaching zero.*

Motivations: we take advantage of the fact that typically the inquiry authors assume to obtain rather limited set of data. If it is not the case, the understanding system switches to the interactive mode. We base the rating of inquiry translation on the comparison of the generated data set with the data set, highlighted by the user in case he knows all the facts in the problem domain. That way we eliminate the inherent NL ambiguity.

Suggested approach is fruitful for the *interactive inquiry* system. As only the system makes decision about the optimality of the particular translation, it outputs the resultant data together with the comments concerning the perceived level of generality. A user can then explicitly raise or drop that level, reformulating the question in more general or concrete terms. Finally, the system translates the reformulated inquiry and finds the intersection of the translation sets for original and reformulated inquiries in the positive vicinity of zero score. The inquiry understanding practice showed the *fast convergence to the proper translation* (3 or less iterations) under the interactive inquiring.

This approach achieves the **high flexibility and independence on PD**, which could be significantly extended by a user. The translation procedure transfers the complexity focus from the linguistics to meta-reasoning, reducing the task of understanding the definitions to the analysis of the defining part as an inquiry.

6.2 Examples of inquiries and definitions

We present the NL sentences(underlined) and their translation into ML .

How is the temperature of reagents in the reaction vessel positively influenced if the acid flows to the absorber and is warmed by less then 50 degrees

question:-
Reagent=acid,
warm(Reagent, AppWarm, TempChange),
TempChange<50,
AppWarm=AppTo,
flow(ReagentFlow, AppFrom, AppTo), absorber(AppTo),
ReagentFlow=Reagent,
ReagentInflTo=ReagentFlow,
AppInflTo=AppFrom
influence(ReagentInflTo, AppInflTo, ReagentOn, AppOn),
reaction_vessel (AppOn),
ReagentOn=ReagentTemp,
AppOn=AppTemp,
temperature(ReagentTemp,AppTemp,T).

We have the correlation between variables as the syntactic arguments of the implicit meta predicates, which attenuated the strongest initial translation with the constraints *AppInflTo=AppTo,* … It is worth mentioning, that the meta predicate with arguments, for example, (*ReagentFlow=Reagent),(ReagentInflTo=ReagentFlow)* as syntactical units could be interpreted as the language-object predicate, introducing particular limitations to the scheme architecture. This is the illustration of meta-logical transition of a meta predicate to the level of language-object.

The definition of the concept "to engage a pump" A pump, from which a reagent flows to absorber2, is **engaged**, if the results of the reaction must be cooled.

engage(Reagent,App) :- pump(App),
AppFrom=App,
flow(Reagent, AppFrom, AppTo),
AppTo=absorber2,
resolve(ReagentIn, ReagentOut, AppReact),
ReagentIn=Reagent,
ReagentCool=ReagentOut,
cool(ReagentCool, AppCool, TempChange),
AppReact\=AppCool.

We translated the right part of the definition as the inquiry : "What reagent can flow from a pump to absorber 2, if the results of the reaction are cooled"

Not only *App*, but also *Reagent* is transferred to the newly constructed predicate. The introduction of *ReagentOut* would make the predicate ***engage*** not general enough to embrace all the situations, where it is necessary to start a pump in a particular environment.

We note that there are two ways to interpret this definition. The first way is to introduce the concept of engaging by the example of the pump with absorber and the special process. The first way changes only the terms of the PD description (conservative extension of PD). It is necessary to reveal the difference between the definition specifics when we say "It is necessary to engage a pump , when..." instead of saying "to engage a pump means, that ..." . In other words, awareness what is the reason, and what is the result ("pump is engaged, because …") is necessary both for proper translation of the right part and for the transfer of variables. The second possible interpretation assumes adding of the new data in PD by means of the new concept. In that case the new predicate does not bring in new constraints to its variables. The example above shows, how a new control procedure can be introduced, which is called "engage a pump".

Now we are able to process an inquiry with newly defined concept:

How many degrees warmer would the acid get in those heat exchangers, which are affected by the **engaged** pump

question:-
  pump(Apt),
  engage(Reagent,Apt),
  ReagentInflTo=Reagent,
  AppInflTo=Apt,
  influence(ReagentInflTo, AppInflTo, ReagentOn, AppOn),
  AppWarm=AppOn,
  ReagentWarm=ReagentOn,
  warm(ReagentWarm, AppWarm, TempChange).
  TempChange - ?

We finally present the inquiry with metapredicates *want* and *know*.

For what apparatus does a controller want to know the water temperature does not influence on the reaction vessel?

want(controller, Intention),
Intention= know(controller, ControllerKnowledge),
ControllerKnowledge = (
  Reagent = water,
  temperature(Reagent, Apparatus, T),
  Reagent = ReagentInflTo, Apparatus = AppInflTo,
  influence(ReagentInflTo, AppInflTo, ReagentOn, AppOn),
  reaction_vessel (AppOn)).
  Apparatus - ?

7 The autism scenarios

In this section we model the autism - relatively rare multifactorial disorder that affects between 3 and 5 out of every 10,000 school-aged children. The disorder is characterized by impairment in social interaction and communication coupled with repetitive and stereotyped patterns of behavior. We focus on some intellectual peculiarities of 2-3 years-old children, which is tightly connected with logical AI and could be significant for both autism diagnosis and training.

It was recently discovered [23-25], that the autism children cannot handle properly such concepts as pretending and belief, having the similar capabilities with the control children in the other aspects of the intellectual activity (reasoning about modality, space, time, reasoning complexity, etc.) These studies raised our interest to assign the autism disorder to the specific limitations of reasoning about belief.

The autism is traditionally analyzed in terms of the behavior specifics of an autism agent interacting with the other agents. In this case the autism can be characterized as a property of hard communication with the agents having the same world knowledge and reasoning machine (excluding the specific belief-pretending reasoning machine). We pose the main autism theorem for the multi-agent scenario: how the lack of proper believing and pretending for the autistic agents implies the general miscommunication problems for the agents. This is the deductive link of the microscopic / macroscopic type: how some deviations of the agent properties affect the collective behavior of the group of agents.

We highlight the following 5 steps towards the building of the logical model of the autism and its applications.

1) Building the belief and pretending model of the autism
    a) The logical program with autistic/control switch, which understands the set of scenarios and comprehends them (responds) like an autistic/control subject.
    b) Building the stand-alone logical program of pretending.
    c) Highlighting the most adequate representation of belief.
2) Increasing the estimation accuracy of the reasoning locus of the autism by interactive generation of the more specific experimental scenarios (What kind of beliefs? Which way of pretending?).
3) Development of the autism revealing system. It will assist psychologists, physicians and parents in setting the proper testing environment to detect the autistic reasoning properties as early as possible.
4) Design of the autism training system, which implicitly forms the adequate concept of believing and the concept of pretending in the autistic children.
5) Explanation of the miscommunication phenomena, having the formalization of autistic and normal agents.

This study is focused on the tasks 1a) and 1b). Distinguishing properties of the autism scenarios is that the K scenario component of autistic agents has reduced, "straight-forward" definitions for pretending and believing. We show the adequacy of the MS scenario approach to present the series of psychological experiments and to yield the explanation of the children responses in these responses having the autism/control "switch" in the scenario *K*.

7.1 Axioms for knowledge

To extend the concept of knowledge for scenario representation, we consider the following meanings:

1) An agent knows that some *Object* possesses some *Feature*.
2) An agent knows some *object* (can identify it , distinguish from other objects).
3) An agent knows what a concept *Feature* means.

We highlight that the second argument of *know* is a meta-variable. The semantics of *know* depends on the syntactic properties of the formulas this meta-variable ranges over. Below are three cases of definition for *know(Agent, Formula)*. It is clear, that no one of them could be expressed by the definitions for know (there are three different knowledge modalities)

1) *know(Agent, Feature(Object)) :- const(Feature), const(Object) :- …*
   This is the standard knowledge for the fact (S5).

   know(Agent, Feature(Object)) :- var(Feature), var(Object), fail if an *Agent know*s neither *Feature*, an *Object* could possess, nor *Object*, which could satisfy a *Feature*, then the statement says nothing. The statement that there exists a *Feature* and an *Object* such that *Feature(Object)* holds, is assumed to be non-informative.

2) *know(Agent, Object) :- know(Agent, Feature(Object)),*
   const(Feature), var(Object),
   $\forall$ AnotherObject $\exists$ AnotherFeature
   know(Agent, (AnotherFeature (AnotherObject),
   Feature \= AnotherFeature,
   Object \= AnotherObject,
   not AnotherFeature(Object))).

   For any *Object*, an *Agent know*s it if he can identify it in the following way: this *Agent* should *know* that there exists a *Feature* such that *Feature(Object)* holds, and for any *AnotherObject* there exists its *AnotherFeature* such that it is satisfied by the *AnotherObject* and is not satisfied by the *Object.*
   There is another way to identify an *Object* by a unique *Feature*:
   know(Agent, Object) :- know(Agent, Feature(Object)),
   const(Feature), var(Object),
   $\forall$ AnotherObject $\exists$ AnotherFeature
   know(Agent, not Feature(AnotherObject)).

3) *know(Agent, Feature) :- know(Agent, Feature(Object)),*
   var(Feature), const(Object),
   know(Agent, exist AnotherObject Feature(AnotherObject)).

   For any *Feature*, an *Agent know*s this *Feature* when he *know*s, that *Feature(object)* holds, and this *Agent knows anotherObject* which satisfies *Feature.*

Positive, negative introspection and Barkan property are satisfied by all three types of knowledge. The knowledge axiom is not applicable to the cases of knowledge of a concept or an object.

To be more precise in presenting these definitions, we note that it is a clause scheme rather than a clause. The latter clause, for example, is derived from the clause scheme by means of instantiating *Feature $\rightarrow$ feature* and *AnotherFeature$\rightarrow$ anotherFeature*. Then all the rest of variables are instantiated in the traditional way to obtain the fact, that the particular agent knows the particular object. In contrast to the first order interpretation (the standard logical program), the latter definition requires two steps of the variable instantiation and could be interpreted by a meta-reasoning system.

Now we present the definitions how the agents can get these three kinds of knowledge with the help of the other agents.

1) To make an *Agent know* a *Fact*, another *Kagent inform*s that it holds (naïve)

know(Agent, Fact) :- not know(Agent, Fact), inform(Kagent, Agent, Fact).

In terms of reasoning about action, it looks better with the metapredicate *cause*

know(Agent, Fact) :- cause (not know(Agent, Fact), inform(Kagent, Agent, Fact)).

Being more specific, we have to explicitly use the modality *want(Agent, Formula)*

know(Agent, Fact) :- know(Kagent, not know(Agent, Fact)),
want(Kagent, know(Agent, Fact)),
inform(Kagent, Agent, Fact)).

2) To make an *Agent* identify an *Object*, another *Kagent* has to *inform* it about some distinguishing *Feature* of this *Object* and to convince him that *AnotherFeature* does not work.

know(Agent, Object) :- know(Kagent, not know(Agent, Object)),
want(Kagent, know(Agent, Object)),
inform(Kagent, Agent, Feature(Object)),
inform(Kagent, Agent,
(∀ AnotherObject there exists AnotherFeature
AnotherFeature (AnotherObject),
Feature \= AnotherFeature,
Object \= AnotherObject,
not AnotherFeature(Object)
)).

The only way to make an *Agent know* a *Feature* is to explain it by example

know(Agent, Feature) :- know(Kagent, not know(Agent, Feature)),
want(Kagent, know(Agent, Feature)),
inform(Kagent, Agent, Feature(Object)),
inform(Kagent, Agent,
∃AnotherObject Feature(AnotherObject),
Object \= AnotherObject
).

Once we have the definition of what it means **not to know the concept** versus not to know the fact, we are able to present the following scenario This was the basis of the artificial scenario, composed for Internet advertisement in Section 4.

Bill's Dilemma

There was a guy named Bill who was in quite the dilemma. His wife had disappeared and he was desperately looking for her.

He came upon a policeman and explained his situation. The policeman said, "So, can you describe your wife?"

Bill said, "I don't understand - what do you mean describe her?"

The policeman said, "You know, her features and what she looks like. For example, my wife has long blonde hair, blue eyes and she's thin."

Bill said, "Well, my wife has dark hair, dark eyes and is rather large." With a pause, Bill said, "Actually, how about we look for your wife?"

We have the following examples for various knowledge modalities from Section 4:
Bill does not *know*, which computer to *buy* (he knows that he wants to buy a computer, but does not know which one

know(bill, buy(bill, UnknownComputer)),
not know(bill, UnknownComputer) .

Further, Bill is not familiar with a property of a computer he wants to buy

know(bill, (computer(UnknownComputer),
Feature(UnknownComputer))),
not know(bill, Feature).

Finally, Bill does not know if Doug knows which computer to buy

know(bill, (know(doug, buy(bill, UnknownComputer)),
not know(doug, UnknownComputer))).

7.2 Axioms for pretending and believing

We first consider the simple axiom system of pretending as a modal operator, which is based on the traditional concept of knowledge, presented elsewhere [17]. Based on the modal operator of knowledge, one is unable to express the series of concepts like "good will" versus "deceptive" pretending. We need the knowledge metapredicate, used in the examples above and presented in details 7.2 to give the explicit definitions of these types of pretending in addition to the other types of modalities.

We denote by P_iF the agent *i*'s pretend that the fact F holds. We do not use special symbols here to express that the agent *i* pretends for another agent *j* and rather show it indirectly.

1) General definition: an agent *i* pretends to the agent *j* that the fact *F* holds if he knows that *j* will understand the pretending: a) *i* knows that *j* knows that *i* pretends, b) *i* knows that *F* does not hold and that *j* knows that *F* does not hold, c) *i* assumes this pretend will be accepted.

   $K_iK_j\, P_i\, F \;\;\&\;\; K_i\, not\, F \;\&\; K_i\, K_j\;\, not\, F \;\;\rightarrow\;\; P_i\, F$.

2) The pretend addressee either accepts the pretend (pretends that he knows the fact) or reject it (not know it).

   $P_i\, F\;\; \text{->}\; P_j\, K_j\, F\;\; v\;\; not\; K_j F.$

3) If an agent *i* pretends that F_1 holds for agent *j* and pretend that F_2 holds for agent *m*, and *j* can inform *m* about some fact *G* (in particular, it could be that *G=M1)*, than *i* has to keep pretending that the conjunction of F_1 and *F* holds

   $K_j\, P_i\, F_1 \;\;\&\;\; K_m\, P_{\,i}\, F_2 \;\;\&\;\; (K_j\, G \rightarrow\;\; K_m G) \rightarrow\;\; F{=}F_1 \,\&\, F_{\,2} \;\&\;\; P_i\, F$.

4) If an agent *i* pretends that F_1 holds and agent *j* pretends that F_2 holds, and this pretending has been accepted by both of them, than both of them aware that neither F_1 nor F_2 holds.

   $P_i\, F_1 \,\&\, P_j\, F_2 \,\&\, P_j\, K_j\, F_1 \;\;\&\, P_i\, K_i\, F_2 \;\rightarrow\; K_i\, not\, (F_1 \,\&\, F_2) \;\;\&\, K_j\, not\, (F_1 \,\&\, F_2).$

5) An agent can pretend only about his own knowledge

$P_i\, K_j\, F \rightarrow i = j$.

To make the pretending axioms adequate for the autism representation, we need the (modal) concept *inform* (*say, write,* etc.). Then the general definition will look like

1') *pretend(Who, Fact)* $\rightarrow$ *inform(Who, Whom, Fact),*
know(Who, know(Whom, pretend(Who, Fact))),
know(Who, not F), know(Who, know(Whom, not Fact)).

2') *pretend(Who, Fact)* $\rightarrow$ *inform(Who, Whom, Fact),*
(pretend(Whom, know(Whom, Fact); know(Whom, Fact)).

3'*) inform(Agent, Agent1, F1), pretend(Agent, F1),*
inform(Agent, Agent2, F12), pretend(Agent, F2),
(inform(Agent1, Agent2, F2);inform(Agent2, Agent1, F1)) $\rightarrow$
pretend(Agent, F1&F2).

6) The definition of pretending, preserving consistency: If *Who* has started to pretend that *SmthBefore* had hold and he knows, that *SmthBefore* implies *Smth*, *Who* has to continue pretend that *Smth* holds.

pretend(Who, Whom, Smth):- pretend(Who, Whom, SmthBefore),
know(Who, (know(Whom, SmthBefore$\rightarrow$ *Smth)))).*

MS definitions of pretending introduce multiple meanings by means of *know* and *inform*, which is essential to express the autism pretending.

Now we approach the definitions for *believe*

believe(Who, Whom, Matter):- not know(Who, Matter),
not know(Who, not Matter),
inform(WhoKnows, Who, Matter),
know(Who, know(Whom, Matter)).

This is the dynamic clause for the *change* of belief:

trust(Agent, AnySource); not believe(Agent, not Fact)),
inform(AnySource, Agent, Fact) $\rightarrow$
change(Agent, believe(Agent, WrongFact), believe(Agent, Fact)),
believe(Agent, Fact)).

Here we link the general concept (metapredicate) *change* in its general sense (to *change* an object, a feature or a belief) with the metapredicates *trust* and *believe*.

We need to define the companion concept *trust*. There is (was) no such *Matter* that *Who believed Whom* that it holds and then discovered that it does not hold:

trust(Who, Whom) :- not (inform(Whom, Who, Matter),
believe(Who, Whom, Matter),
know(Who, not Matter)).

Important hierarchical features of belief are considered in [21]. The fact, that an agent *a* prefers a belief *B1* to belief *B2* at time *t*, is expresses by means of the special predicate *pref(a, B1, B2, t)* such that certain features of an antisymmetric binary relation hold.

Extending the MS style, we consider *prefer* as a regular metapredicate
∀Fact trust(Agent, Agent1,Fact), not trust(Agent, Agent1,Fact)→
believe(Agent, prefer(Agent, believe(Agent1, Fact1),
believe(Agent2, Fact2))).

In contrast to [21], we explicitly specify that the *Agent* changes an old belief into a new one in case it is about the same object.
prefer(Agent, believe(Agent, Feature1(Object)), believe(Agent, Feature2(Object))).

We proceed to the definition of the **"good will"** pretending: *Who* wants *Whom* to perform an *Action* and *Who know*s that if *Whom know*s that *Who* possesses (a property) *Matter* than *Whom* performs that *Action* when *Who informs Whom* about *Matter.*
pretend(Who, Whom, Matter(Who)):-
want(Who, inform(Who, Whom, Matter(Who))),
know(Who, not Matter(Who)),
know(Who, (know(Whom, not Matter(Who))),
inform(Who, Whom, Matter(Who)).
This type of pretending is required for the autism modeling.
Intentional pretending turns into deceiving when *Whom* does not *want* this *Action* and *Who knows* that.
pretend(Who, Whom, Matter(Who)):- want(Who, action(Whom, Action)),
know(Who, not Matter(Who)),
know(Who, not want(Whom, action(Whom, Action))),
know(Who, (know(Whom, Matter(Who))-> action(Whom, Action))),
inform(Who, Whom, Matter(Who)).
Compare this definition of "cheating by means of pretending" with *Cheat* definition from Sections 2.8 and 3.3. One needs to distinguish pretending and deceiving.
We show, what it means to pretend to perceive (understand) somebody else's pretending
perceive(Whom, pretend(Who, Whom, Matter)):-
pretend(Who, Whom, Matter),
inform(Who, Whom, Matter(Who)),
know(Whom, not Matter(Who)),
know(Whom, want(Who, action(Whom, Action))).
What it means to believe to somebody's pretending (not to understand it)
believe(Whom, pretend(Who, Whom, Matter)):- pretend(Who, Whom, Matter),
inform(Who, Whom, Matter(Who)),
believe(Whom, not Matter(Who)),
not know(Whom, want(Who, action(Whom, Action))).
In purpose of simplification we skip the expression that *Action* occurs in *Matter*. The required syntactic metapredicate was introduced in Section 2.8.

7.3 Autism revealing scenarios

Below are the texts of scenarios from [23] and their MS representations. These scenarios contain the description of what happened with some agents and questions to the autistic/control agent. We consider this questions as "reaction stimulation" and do not present them explicitly in the formal scenario.

The child is encouraged to “fill” two toy cups with “juice” or “tea” or whatever the child designated the pretend contents of the bottle to be. The experimenter then says, “Watch this!”, picks up one of the cups, turns it upside down, shakes it for a second, then replaces it alongside the other cup. The child is then asked to point at the “full cup” and at the “empty cup”. (Both cups are, of course, really empty throughout.)

The knowledge of the concept *turn_upside_down:*

(KU) know(Kid, (turn_upside_down (Exper, Cup) $\rightarrow$ *not full(Cup))).*

(H1) not full(cup1), not full(cup2).
(H2) pretend(kid, _, full(cup1), full(cup2)).
(H3) turn_upside_down(exper, cup1)).
(R) pretend(kid, _, not full(cup1), full(cup2)).

The child was introduced to a doll character, Billy, and three pieces of toy furniture: a bed, a dressing table and toy-box. A story was enacted for the child in which Billy had a ball. Billy put his ball on the dressing-table then went downstairs for breakfast. While Billy was away, his mother came into his room, picked up the ball and put it in the toy-box. The child was then asked four questions: control question 1, "Where did Billy leave his ball?"; known question, "Does Billy know where his ball is?"; think question, "Where does Billy think his ball is?" . We used test question rather than a prediction of behavior question to be closer to asking directly about a mental representation. Finally, control question 2, "Where is the ball really?"

($KU_{control}$) (trust(Agent, AnySource); not believe(Agent, not Fact)),
inform(AnySource, Agent, Fact) $\rightarrow$
change(Agent, believe(Agent, WrongFact), believe(Agent, Fact)),
believe(Agent, Fact)).
($KU_{autistic}$) believe(Agent,Fact) $\rightarrow$ *believe(AnotherAgent, Fact).*
(KU) see(Agent, Fact) $\rightarrow$ *inform(AnySource, Agent, Fact).*
(H.1) put(billy, ball, table)
(H.2) see(agent, put(mother, ball, box)

(R?) believe(agent, believe(billy, locate(ball, Where))).
($R_{control}$) believe(agent, believe(billy, locate(ball, table))).
($R_{autistic}$) believe(agent, believe(billy, locate(ball, box))).

The autism phenomenon seems to be important to choose the formalism, adequate for the human intelligence. The model of the brain and its specific reasoning, and, in particular, the model of the autistic brain is hardly desirable to build. But the difference between these two models fits into the limited formalism of just two concepts, believing and pretending. So this restricted component of brain activity could be subject to the logical modeling. The experiments of the agent scenario understanding show that various autistic children experience the lack of (formally) the same meanings of believing and pretending.

Conclusions

For comparison, we mention two metalanguage reasoning systems [14, 15], which could serve as an appropriate basis for our problems after addition of the metalanguage transfer rules, representations for *KU, KS* and other features, described in this paper.

To draw the comparison with the other meta-reasoning approaches, we highlight the distinguishing features of our formalism, illustrated by means of the series of scenarios and presented with the help of the example for ***attenuate***, the key metapredicate of MS:

1. A formal scenario with MS could serve an alternative for the traditional axiomatic methods for the practical applications, gaining flexibility and computational efficiency.
2. Predicate invention is an important mechanism, inherited to MS, which adjusts a scenario-like axiomatic to a new problem domain.
3. MS possesses sufficient expressiveness to represent the scenarios with common sense, spatial, temporal reasoning, reasoning about action, knowledge and intentions. MS provides the capabilities for the metapredicates, inherited to these kinds of reasoning, to be invented for a particular scenario.
4. Formal scenarios have embedded mechanism for representation of multiple meanings by means of MS. It makes them adequate to model the human intellectual activity and to operate with NL scenarios with the natural ambiguity.
5. MS learning is an important feature in modeling the human understanding.

We introduce the meta-programming technique for prototyping the commercial application, required advanced reasoning. The PROLOG implementation of rather flexible, but inefficient meta-reasoning is intended to draw the boundary of a new problem domain, build the specific concepts and reduce the real-time inference search. The prototyping results yield the adequate object (and data) model specification, subject to developing in accordance to the traditional programming technologies.

MS was successfully implemented in the constraint satisfaction problems for the advertisement scheduling, delivery and content generation. The scheduling of broadcasting commercials, developed for Marketron, Inc. (http://www.marketron.com), possesses such features as ability to adjust to the environment of a particular station, ability to handle uncertainty and contradictions in the requirements under incremental scheduling and adaptation to the personnel with various intentions for client satisfaction. The system of the Internet advertisement delivery control and targeting, designed for NetGravity, Inc. (http://www.netgravity.com) combines MS algorithms with the dynamic programming.

The prototype of advertisement scenario synthesizer was originally created for the Internet auction project of Internet Shopping Network, Inc. in addition to implementation of the auction scheduling and "intelligent auction buddy" (http://www.firstauction.com). The language of high expressiveness and flexible deductive means is necessary to capture the "behavior" of a customer, browsing an Internet site, in respect to the interest to various pages. MS system builds an explanation, *why* did the customer switch from page to page in a particular order, and generates the order of advertisement presentation for this customer for future delivery. The features of a customer behavior are important for the task of fraud detection, which could significantly decrease the losses of the Internet retailers. The manifold of MS implementations, presented above, implies its applicability in the various fields of the flexible reasoning software.

Acknowledgements

The author is grateful to Profs. D.Usikov, I.Muchnik, I.Gelfand and V.Fain for fruitful discussions and to Internet Shopping Network, Inc. for financial support.

References

1. Galitsky, B.A. The work of logical processor with metalanguage accompaniment under the scene analysis *Image Analysis and Pattern Rec*., 1995, v5, N 2, 245-263.
2. De Raedt, L. Logical settings for concept-learning AI v.95, 187-201, 1997.
3. Angluin, D., Frazier, M. and Pitt, L., Learning conjunction of horn clauses, *Machine Learning* 9 (1992) 147-162.
4. Frazier, M. and Pitt, L., Learning from entailment, in *Proceedings 9^th^ International Conferemce on Machine Learning*, Amherst, MS (1993).
5. Hill, P.M., Lloyd, J.W. Analysis of metaprograms, *in Abramson & Rogers (eds) Metaprogramming in Logic Programming*, MIT Press, pp23-52, 1989.
6. Giunchiglia F.and Traverso P A system for Multi-Level Mathematical Reasoning, *AI in mathematics* Johnson, J.H., McKee, S. and Vella, A. (eds) 1994.
7. Bundy A. and Welham, B. Using meta-level inference for selective application of multiple rewrite rules in algebraic manipulations, AI 16 (2) , pp189-212, 1981 .
8. Galitsky, B.A. Metalanguage accompaniment tool for detection rules, run by a neural network *Proc. J.Conf.Informational Sci, NC,* pp. 370-373, 1995.
9. Weyhrauch, R.W. Prolegomena to a theory of mechanized formal reasoning *AI* 13(1), 1980, 133-176.
10. Giunchiglia F. and Traverso P., Reflective reasoning with and between a declarative metatheory and implementation code., in: *Proc. IJCAI-91, Sydney, Australia,* pp.111-117, 1991.
11. Miikkulainen R. Subsymbolic Natural Language processing *The MIT Press*, Cambridge, MS;London, England, 1994.
12. Galitsky, B.A. Understanding of definitions under NL interaction *proc EastWest AI Conf-1993*, pp.234-240.
13. Sripada, S.M. A metalogic programming approach to reasoning about time in knowledge bases *in Proc. IJCAI* 1993.
14. Heintze, N., Michaylov, S.,Stuckey, P.,Yap, R.H.C. Meta-programming in CLP(R) *J. Logic Programming* 221-259, 1997.
15. Cicekli I Abstract metaprolog engine *J Logic Programming* 169-199 1998.
16. Muggleton,S., ed. Inductive Logic Programming. Academic Press, 1992.
17. Fagin, R., Halpern, J.Y.,Moses, Y.,Vardi, M.Y. Reasoning about knowledge *The MIT Press*, Cambridge, MA, London, England, 1995.
18. Koza, J., Bennett III, Forrest H, Keane, Martin A., and Andre, David. 1997b. Evolution of a time-optimal fly-to controller circuit using genetic programming. In Koza, J, Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max, Iba, Hitoshi, and Riolo, Rick L. (editors). 1997. Genetic Programming 1997: Proceedings of the Second Annual Conference, July 13ȡ16, 1997, Stanford University. San Francisco, CA: Morgan Kaufmann. 207 ȡ 212.

19. Koza, J. 1997. Future work and practical applications of genetic programming. In Baeck, T., Fogel, D. B., and Michalewicz, Z. (eds.) Handbook of Evolutionary Computation. Bristol, UK: Institute of Physics Publishing and New York: Oxford University Press. Pages H1:1-6.
20. Koza, J. Genetic Programming: On the Programming of Computers by Natural Selection. MIT Press, Cambridge, MA, USA, 1992.
21. Bell, J. and Huang, Z. Dynamic belief hierarchies *Common-Sense 98* R.Miller and M.Shanahan (eds.) 1998.
22. Shoham, Y. Reasoning about change *MIT Press*, Cambridge MA 1986.
23. Leslie, A. and Thaiss, L. Domain specificity in conceptual development: Neuropsychological evidence from autism. *Cognition,* 43, 225-251 1992.
24. Perner, J. Understanding the representational mind. *MIT Press* Cambridge MA 1991.
25. Zaitchik, D. When representations conflict with reality: The preschooler's problem with false beliefs and "false" photograph. *Cognition*, 35, 41-68 1990.