DIMACS Technical Report 2002-42
October 2002

Using inductive reasoning and reasoning about dynamic domains for automatic processing of claims

by

Boris Galitsky[1] and Dmitry Vinogradov[2]

[1] KnowledgeTrail, Inc., 9 Charles Str. Natick MA 01760 USA.
(galitsky@dimacs.rutgers.edu)
[2] VINITI RAS Usievicha 20, Moscow 125190 Russia
(vin@viniti.ru)

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

ABSTRACT

We report on the novel approach to modeling a dynamic domain with limited knowledge. A domain may include participating agents so that we are uncertain about motivations and decision-making principles of some of these agents. Our reasoning setting for such domain includes the deductive and inductive components. The former component is based on situation calculus and describes the behavior of agents with complete information. The latter, machine learning-based inductive component (with the elements of abductive and analogous reasoning) involves the previous experience with the agent, whose actions are uncertain to the system.

Suggested reasoning machinery is applied to the problem of processing the claims of unsatisfied customers. The task is to predict the future actions of a participating agent (the company that has upset the customer) to determine the required course of actions to settle down the claim. We believe our framework reflects the general situation of reasoning in dynamic domains in the conditions of uncertainty, merging analytical and analogy-based reasoning.

Introduction

In the recent decade, reasoning about action has become a discipline, promising from the standpoint of applications. A series of formalisms, developed into the programming environment have been suggested for robotics applications. Particularly, the logic programming environment for dynamic domains GOLOG, suggested by (Levesque et al 1997) has been extended by multiple authors for a variety of fields (Lakemeyer 1999, Grosskreutz & Lakemeyer 2001, Souchansky 2001). Involvement of sensory information in building the plan of multiagent interaction has significantly increased the applicability of GOLOG. However, GOLOG is still not well suited to handle the multiagent scenarios with lack of information concerning the actions of opponent agents, when it is impossible to sense them.

At the same time, the methodology of obtaining a formal description for a set of facts in the form of inductive reasoning in the wide sense has found a series of applications, including bio-chemistry, protein engineering, drug design, natural language processing, finite element mesh design, satellite diagnosis, text classification, medicine, games, planning, software engineering, software agents, information retrieval, ecology, traffic analysis and network management (De Raedt 1999). The most appealing applications have three following important features:

1) The hypotheses are generated by a general purposes language that belongs to the family of logic programming languages;
2) The results are judged by human expert;
3) New scientific insights in the application domain are obtained.

In this paper, we target domains that cannot be solved with classical attribute value learning system. A knowledge discovery system that is based on inductive logic programming or similar approaches (Muggleton 1992) is insufficient, taken alone, because it is incapable to perform necessary reasoning about actions in accordance to heuristics available from domain experts. Neglecting this knowledge would dramatically decrease the extents of possible predictions. Also, a generic knowledge discovery system is not oriented to handle dynamic kind of data. Therefore, we intend to merge the reasoning about action-based and learning based systems to form the environment to handle dynamic domains with incomplete information.

We choose the GOLOG (Levesque et al 1997) and JSM (Finn 1999) environments for reasoning about action and inductive machine learning respectively because of their flexibility and powerfulness. Using above approaches to illustrate our methodology, we keep in mind that our architecture of merging deductive and inductive components is independent of the choice of particular formalism and better models real-world domains than these approaches, taken separately.

A generic environment for reasoning about actions is not well suited for handling essentially-incomplete data, where neither totality of procedures, nor actions preconditions nor successor state constraints are available. Evidently, situation calculus by itself does not have a predictive power and needs to be augmented by a learning system capable of operating in the dynamic language. Abstraction of reasoning about action in the way of GOLOG assumes that action preconditions, successor state expressions and expressions for complex actions are known.

Incomplete knowledge about the world is reflected as an expression for nondeterministic choice of order to perform actions, nondeterministic choice of argument value, and nondeterministic repetition. These settings are adequate for the selected robotics application, where the designer uses a particular approximation of external world. In a general setting, an agent that performs reasoning about actions is expected to learn from the situations, where the actual course of actions has been forced by the environment to deviate from the obtained plan, using the current world model.

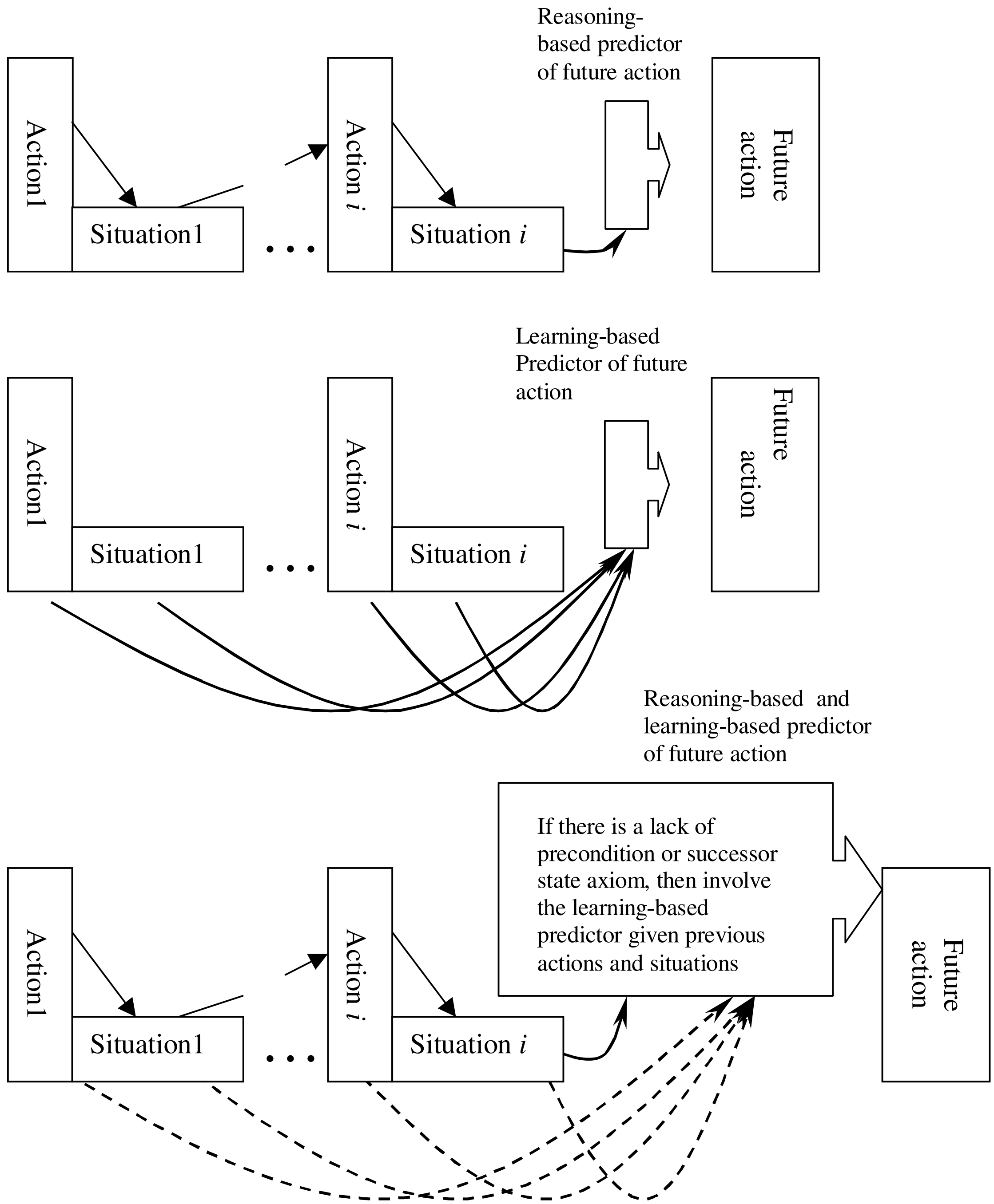

Fig.1: Illustration for the merging reasoning about action-based and learning-based machinery for performing the recognition of actions in dynamic domain. On the top: reasoning about action, in the middle: machine learning, on the bottom: merged approaches.

Let we have a task of predicting the action that follows a given situation (achieved after a series of known actions) in the deterministic environment. The latter means that the agents perform actions based on the fixed set of rules. There are two basic kinds of methodologies for deriving the future action or set of possible actions (Fig.1):

1) By means of reasoning about actions. Its framework specifies a set of available basic and combined actions with conditions, given a current situation, described via a set of fluents. These fluents are in turn have additional constraints and obey certain conditions, given a set of previous actions. Action possibilities, pre-conditions and successor state axioms are formulated *manually*, analyzing the past experience. This methodology can fully solve the problem if full formal prerequisites for reasoning about action are available.
2) By means of learning the future action from the set of examples. Given a set of examples with the sequence of actions and fluents in each, a prediction engine generates the hypotheses how they are linked to the future action. Resultant hypotheses are then applied to predict the future engine. Such kind of learning requires enumeration of actions and fluents; learning itself is performed automatically. This methodology is worth to apply in a stand-alone manner if neither explicit possibilities for actions nor their pre-conditions are available.

The intention of current study is to demonstrate that above methodologies are complementary to each other. The following facts contribute to such observation:

1) Almost any prediction task, particularly in deterministic approach, is some combination of manually obtained heuristics and automatically extracted features, which characterize an object of interest;
2) If to try to predict all actions using learning, the problem complexity dramatically increases and, therefore, the accuracy of solution under possible approximation drops.
3) On the other hand, if to try to explicitly present pre- and post-action conditions for the deductive settings, we run against a frame problem that may need a unique solution for a specific situation (Shanahan 1999). Besides, some other difficulties are associated with the search of inference (plan).
4) Considering a sequence of actions in a dynamic domain, the longer this sequence is, the more inductive reasoning comes into play relatively to the deductive one.

Note that above considerations are valid when the choice of action does not occur in a pure mental world, i.e. a world where the situations are described in terms of belief, knowledge and intension (Galitsky 2001). Choosing an action that is to be performed in a given mental state occurs in accordance to quite different laws (Galitsky 2002), unlike the ones for a physical state we are talking in this report.

Inductive machine learning as a logic program

In this paper we deploy both the GOLOG and JSM environments in the way of logic program. We will briefly comment on GOLOG in the following Section because it has been thoroughly presented in the literature. As to JSM (Finn 1999), we build it as a logic program in this paper, following the formal frameworks of (Anshakov et al. 1989) and (Vinogradov 1999).

JSM environment is set by features, objects and targets. Within a first-order language, objects are individ constants, features and targets are terms which include these constants. For a target (feature), there are four groups of objects with respect to the evidence they provide for this target:

- Positive;
- Negative;
- No influence;
- Unknown.

Clearly, an inference in such environment can be represented as one in a respective four-valued logic. Prediction in JSM setting is based on verification whether the objects from last group satisfy

the target feature or not. Prediction machinery is based on building the *example separation* hypothesis $target(X)\text{:-}\ features_1(X, \ldots), \ldots, feature_n(X, \ldots),$
where $target \in targets,\ features_1, \ldots, feature_n \in features,\ X$ ranges over objects,
targets is the set of all targets, *features* is the set of all features.

Desired separation is based on the *similarity* of objects in terms of features they satisfy. Usually, such similarity is domain-dependent. However, building the general framework of inductive-based prediction, we use the anti-unification of formulas that express the totality of features of the given and other objects (our futures do not have to be unary predicates).

Our starting example of JSM settings for unary predicate is as follows (from now on we use the conventional PROLOG notations for variables and constants):

```
features([a,b,c,d,e]).
objects([o1, o2, o3, o4, o5, o6, o7]).
targets([v1]).
a(o1). b(o1). c(o1).
a(o2). b(o2). c(o2). e(o2).
a(o3). d(o3).
a(o4). c(o4). d(o4).
a(o5). b(o5). e(o5).
a(o6). d(o6).
a(o7). b(o7). c(o7). e(o7).
v1(o1). v1(o2). v1(o5).
unknown(v1(o7)).unknown(v1(o6)).
```

Starting with the positive and negative examples *jPos(X, v)* and *jNeg(X, v)* for the target *v*, we form the totality of examples' intersections (positive ones *U* that satisfy *iPos(U,v)* and negative ones *U* that satisfy *iNeg(U,v)*):

iPos(U, v):- jPos(X1, v), jPos(X2, v), X1\=X2, similar(X1, X2, U), U\=[].
iPos(U, v):- iPos(U1, v), jPos(X1, v), similar(X1, U1, U), U\=[].

iNeg(U, v):- jNeg(X1, v), jNeg(X2, v), X1\=X2, similar(X1, X2, U), U\=[].
iNeg(U, v):- iNeg(U1, v), jNeg(X1, v), similar(X1, U1, U), U\=[].

Above are the recursive definitions of the hypothesis. As the logic program clauses that actually form the totality of intersection of examples, we derive the following:

```
iPos(U, V):- iPos(U, V, _).
iPos(U, V, Accums):- jPos(X1, V), jPos(X2, V), X1\=X2, similar(X1, X2, U),
                     Accums=[X1, X2], U\=[].
iPos(U, V, AccumsX1):-  iPos(U1, V, Accums), !,   jPos(X1, V),
                      not member(X1, Accums),    similar(X1, U1, U), U\=[],
                      append(Accums, [X1], AccumsX1).

iNeg(U, V):- iNeg(U, V, _).
iNeg(U, V, Accums):- jNeg(X1, V), jNeg(X2, V), X1\=X2, similar(X1, X2, U),
                     Accums=[X1, X2], U\=[].
iNeg(U, V, AccumsX1):-  iNeg(U1, V, Accums), !, jNeg(X1, V),
                     not member(X1, Accums),   similar(X1, U1, U), U\=[],
                     append(Accums, [X1], AccumsX1).
```

To obtain the actual positive and negative hypotheses from their proimages, we filter out the hypotheses that are satisfied by both positive and negative examples:

```
j0Hyp(U, V):- iPos(U, V), iNeg(U, V).
```

```
jPosHyp(U, V):-iPos(U, V), not j0Hyp(U, V).
jNegHyp(U, V):-iNeg(U, V), not j0Hyp(U, V).
```

The following clauses deliver the background (enumeration of objects that cause) for positive, negative and inconsistent hypotheses:

```
ePos(X, V):- jPos(X, V), jPosHyp(U, V), similar(X, U, U).
eNeg(X, V):- jNeg(X, V), jNegHyp(U, V), similar(X, U, U).
j01(X, V):-jT0(X,V), jPosHyp(U1, V),  jNegHyp(U2, V), similar(X, U1, U1),
similar(X, U2, U2).
```

Finally, we approach the clauses for prediction. For the objects with unknown target, the system predicts that they either satisfy that target, does not satisfy the target, or that the fact of satisfaction is inconsistent with the input facts:

```
jPos1(X,V):- jT0(X, V), jPosHyp(U, V), similar(X, U,U), not j01(X, V).
jNeg1(X,V):- jT0(X, V), jNegHyp(U, V), similar(X, U,U), not j01(X, V).
jT1(X,V):- jT0(X,V), not jPos1(X, V), not jNeg1(X, V), not j01(X, V).
```

Indeed, the first clause above will serve as an entry point to predict (to choose) an action from the explicit list of available actions that can be obtained for a current state given the precondition axioms. Also, if there is no positively predicted action, we chose one with inconsistent prediction (assuming it is better than nothing in the case when an action has to be selected and committed).

For example, for the knowledge base above, we have the following results:

```
Intersections for positive, negative and unassigned
hypotheses[[a(_E0C),b(_E1C),c(_E2C)],[a(_DDC),b(
_DEC)],[a(_D9C),b(_DAC),e(_DBC)]]
[[a(_E84),d(_E94)]]
[[a(o7),b(o7),c(o7),e(o7)],[a(o6),d(o6)]]
Positive, negative and contradiction hypotheses:
[[a(_FDC),b(_FEC),c(_FFC)],[a(_
FAC),b(_FBC)],[a(_F6C),b(_F7C),e(_F8C)]] , [[a(_1054),d(_1064)]] , []
Background for positive, negative and inconsistent hypotheses:
[[a(o1),b(o1),c(o
1)],[a(o2),b(o2),c(o2),e(o2)],[a(o5),b(o5),e(o5)]] ,
[[a(o3),d(o3)],[a(o4),c(o4)
,d(o4)]] , []
Prediction for positive, negative results and inconsistent prediction:
([[a(o7),b(o7),c(o7),e(o7)]] , [[a(o6),d(o6)]] , [])
```

Merging deductive and inductive reasoning about action

Based on the motivations, which were presented in the Introduction, we have the following methodology to predict an action of an agent in the environment where we do not have complete information on this agent. If we are unable to derive an action of this agent given the preconditions for his actions and successor state axioms to sufficiently characterize his current state, learning-based prediction needs to come into play. Instead of just taking into account the current state as reasoning about action would do, learning based prediction takes into account the totality of previous actions and states. It is required because there is the lack of knowledge which previous actions and situations affect the current choice of action.

Situation calculus is formulated in a first-order language with certain second-order features (McCarthy & Hayes 1969). A possible world history that is a result of sequence of *actions* is called *situation*. *do(a,s)* denotes the successor situation to *s* after action *a* is applied. For example, *do(complain(Customer), do(harm(Company),S_0))* is a situation expressing the world history that is based on the sequence of actions { *complain(Customer), harm(Company)*}, where *Customer* and *Company* are variables (with explicit meanings). We refer the reader to (Levesque et al 1997) for the further details on the implementation of situation calculus. Also, situations involve the *fluents,* whose values vary from situation to situation and denote them by predicates with last argument ranging over the situations, for example,

upset(Customer, do(harm(Company),S_0))

Actions have *preconditions* – the constraints on the action possibility:

poss(complain(Customer), s) ≡ upset(Customer, s).

Effect axioms (post-conditions) describe the effect of a given action on the fluents:

poss(complain(Customer), s) & responsive(Company,) ⊃ settle_down(Customer, do(complain(Customer), s)).

Effect axioms express the causal links between the domain entities.

As we see, the methodology of situation calculus is building a sequence of actions given their pre- and post-conditions. To choose an action, we verify the preconditions dependent of fluents. After an action is performed, it affects the fluents, which in turn determines the consecutive action, and so forth.

The *frame problem* (Shanahan 1997) comes into play to reduce the number of effect axioms that do not change (the common sense law of inertia). Successor state axiom resolves the frame problem (Reiter 1991):

$poss(a,s) \supset [\, f(\hat{y}, do(a,s)) \equiv \gamma_f^{+}(\hat{y}, a,s) \lor (f(\hat{y},s) \,\&\, \neg\gamma_f^{-}(\hat{y}, a,s)) \,],$

where $\gamma_f^{+}(\hat{y}, a,s)$ ($\gamma_f^{+}(\hat{y}, a,s)$) is a formula describing under what conditions doing action *a* in situation *s* makes fluent *f* become true (false, respectively) in the successor situation *do(a,s)*.

GOLOG extends the situation calculus with complex actions, involving, in particular, *if-then* and *while* constructions. Macros *do(δ, s, s')* denotes the fact that situation s' is a terminating situation of an execution of complex action *δ* starting in situation *s*. If $a_1,\ldots, a_n$ are actions, then

- [a_1:…: a_n] is a deterministic sequence of actions;
- [a_1#…# a_n] is a non-deterministic sequence of actions;
- *ifCond(p)* is checking a condition expressed by *p*;
- *star(a)*, nondeterministic repetition;
- *if*(*p*, a_1 , a_2), if-then-else conditional;
- *while*(*p*, a_1 , a_2), iteration.

Fig.2 depicts the problem of finding a plan as a theorem-proving in situation calculus:

Axioms $\models (\exists\, \delta, s)\ Do(\delta, S_0, s)\ \&\ Goal(s)$, where plan *Goal*(*s*) is synthesized as a side effect.

We suggest the reader to consult (Levesque 1997) for mode details, and proceed to the GOLOG interpreter.

We add the last line to suggest an alternative choice of action by means of learning by example, if the other options for the following action are exhausted.

```
do(A1 : A2,S,S1) :- do(A1,S,S2), do(A2,S2,S1).
do(ifCond(P),S,S) :- holds(P,S).
do(A1 # A2,S,S1) :- do(A1,S,S1) ; do(A2,S,S1).
do(if(P,A1,A2),S,S1) :- do((call(P) : A1) # (call( not P) : A2),S,S1).
do(star(A),S,S1) :- S1 = S ; do(A : star(A),S,S1).
do(while(P,A),S,S1):- do(star(call(P) : A) : call( not P),S,S1).
do(pi(V,A),S,S1) :- sub(V,_,A,A1), do(A1,S,S1).
do(A,S,S1) :- proc(A,A1), do(A1,S,S1).
do(A,S,do(A,S)) :- primitive_action(A), poss(A,S).
do(A, S, do(A, S)):- predict_action_by_learning(A, S).
```

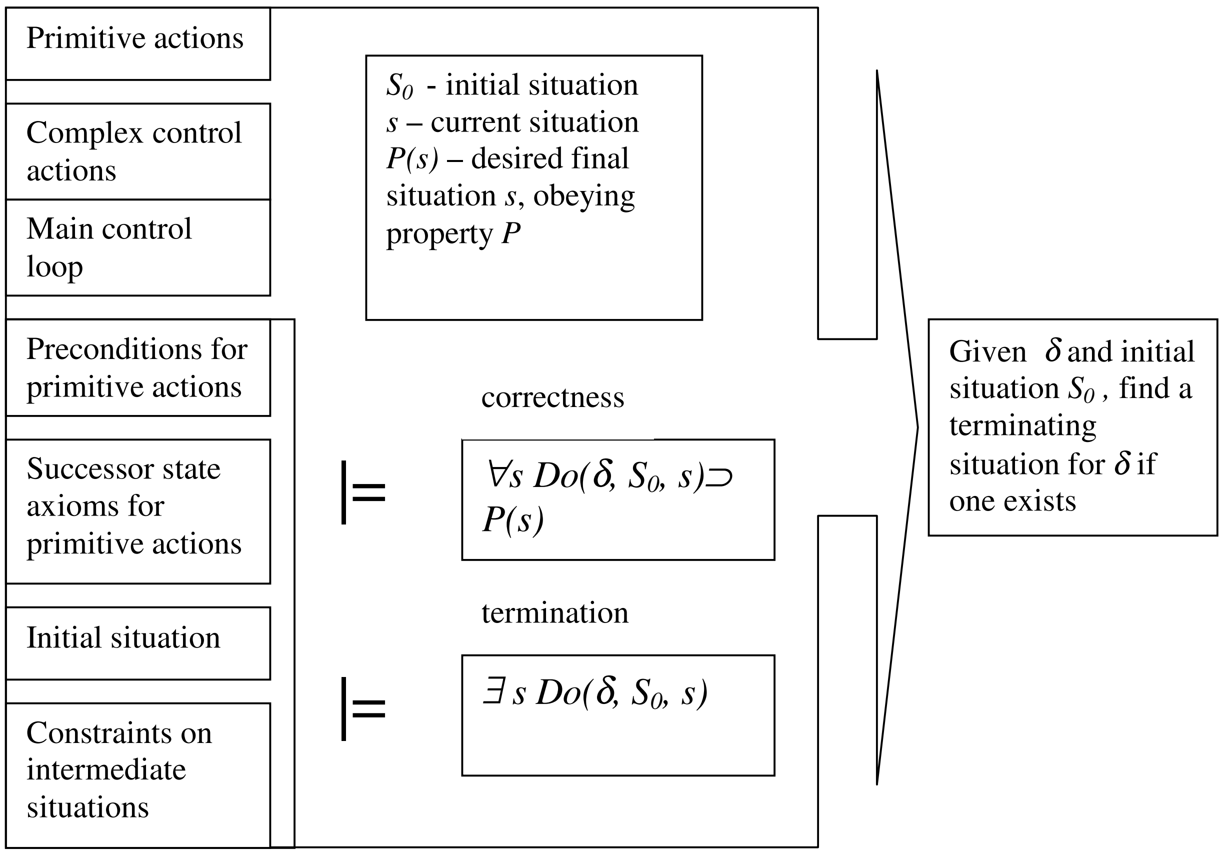

Fig.2: Methodology for deriving a plan in the settings of situation calculus.

Modeling the claims of unsatisfied customers

To suggest an extensive illustration of reasoning about actions in the conditions of uncertainty, we consider a real-life problem of automation of customers' claim. This problem requires a logical component that is capable of modeling the process of interaction between a customer and a company. This logic component is expected to provide a predictive power: given initial claim circumstances, it suggests a set of procedures that might likely lead to customer satisfaction by a company or a third party.

Our problem domain is based on the experience of a series of consumer advocacy companies that try to help the customers, unsatisfied by particular product or service. We base our model on the publicly available database of claims (for example, planetfeedback.com, my3cents.com, etc.). In our considerations, we skip the natural language component that would extract actions, fluents and their parameters; we refer to quite extensive literature on information extraction or to the particular approach of the author, based on logic programming (Galitsky 2001). The task of the written claim processing can be formulated as relating a claim to a class that requires certain set of reactions (direct contacting a producer or retailer, clearing the situation to a consumer, bringing a case to a court, addressing to particular consumer advocacy firm etc.). Performing such kind of task allows us to automate the claim processing, significantly accelerating the results and reducing the cost of operations for a consumer advocacy firm (Fig.3).

Clearly, application of statistical machine learning would skip too many details to adequately relate a natural language claim to a class. If determination of such class deploys a keyword-based information extraction, peculiarities of both natural language representation and dispute process are ignored. The reader should take into account that frequently similar claims with slightly distinguishing mental states of agents may belong to totally different classes of reaction in accordance to the willingness of a company to satisfy its customer (so that he or she drops his/her claim).

Hence, we reveal a set of actions and fluents for each claim. The claim scenario includes two main agents: a customer and a company (for simplicity, we consider multiple representatives of a company as a conventional agent). The behavioral patterns for these two agents are quite different with respect to uncertainty: a customer is assumed to be with us (the consumer advocacy company), and her intentions and motivations are clear. Conversely, we can observe the actions of a company, but we can only build beliefs on its intentions and motivations: we are uncertain about the causal relations between the company's actions and its strategy.

Our current approach to modeling the claims is based on reasoning about actions and respective data mining; here we do not model mental states of involved agents in details, using a special machinery of reasoning about mental states, developed elsewhere (Wooldridge 2000, Galitsky 2002). As we said in the introduction, we perform this step of approximation (ignoring the peculiarities of mental world) to focus on merging reasoning about action with deductive data mining. Also, in this domain we did not use default reasoning to describe the typical and abnormal situations.

Processing a database of claims, we can reveal the typical action patterns for both customers and companies. What we have discovered (and may be natural for the reader) is that the family of strategies for a given company is constraint, as well as that of the totality of customers (the claim database available allows one a statistically significant analysis of a variety of claims for a given company).

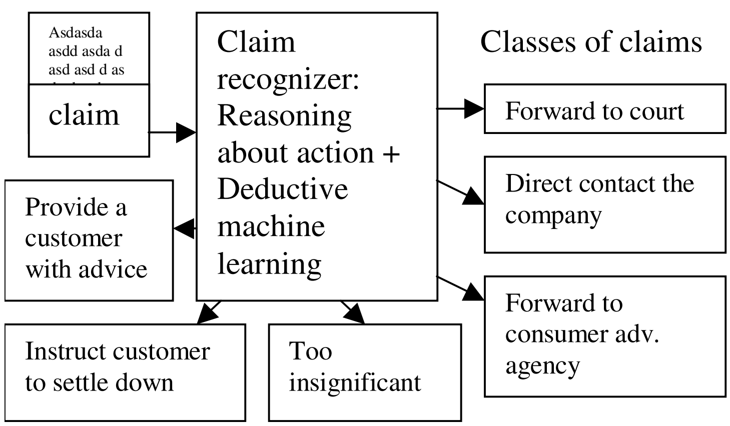

Fig.3: The problem domain: recognizing the class of a claim.

We proceed to the sample set of actions for a claim:

wrongDoC – initial action of a company that caused the claim. Usually, customers complain when they believe something went really wrong and their interests are affected. Company's actions are denoted with the identifier with last character 'C' to distinguish from actions of the customer and his representatives.

askWhy – the first thing a customer does is asking the customer service "Why did it happen? What got wrong?", believing that the company is ready to help.

customerServiceIgnoreAskWhyC – a typical step in the development of relationships between a customer and a company is ignoring the customer's request by the company.
askHowToFix – at this step the customer proceeds from the questions "Why/how did it happen" to "How to achieve the normal/satisfaction situation?"
explainWronglyC – the company gives a wrong advice that does not lead to above situation. It is questionable whether the company's representatives are aware of it.
followWrongAdviceNoResult – the customer believes that company's advice is adequate, follows it and discovers that it is not the case.
ClaimBadCustomerService – customer starts to complain not only about *wrongDoC* but about customer service as well.
FindUnreasonableCauseForCustComplainC – at this point, the company blames the customer that he is non-cooperative and states that there are no reasons to complain.
complainToOtherEstablishments , *askFriend, askLawer, askConsumerAdvocCompany* – the customer understand that she can achieve nothing being one-by-one with the company and looks for help, including a friend, an attorney or a consumer advocacy company.
settleDown, agreeToFixC, agreeToCompensateC, convinceToBeActingAsRequiredC – the customer is settled down by a promise or actual fulfillment of the required actions by the company that has found one way or another to satisfy this customer, or has convinced him that nothing went really wrong.

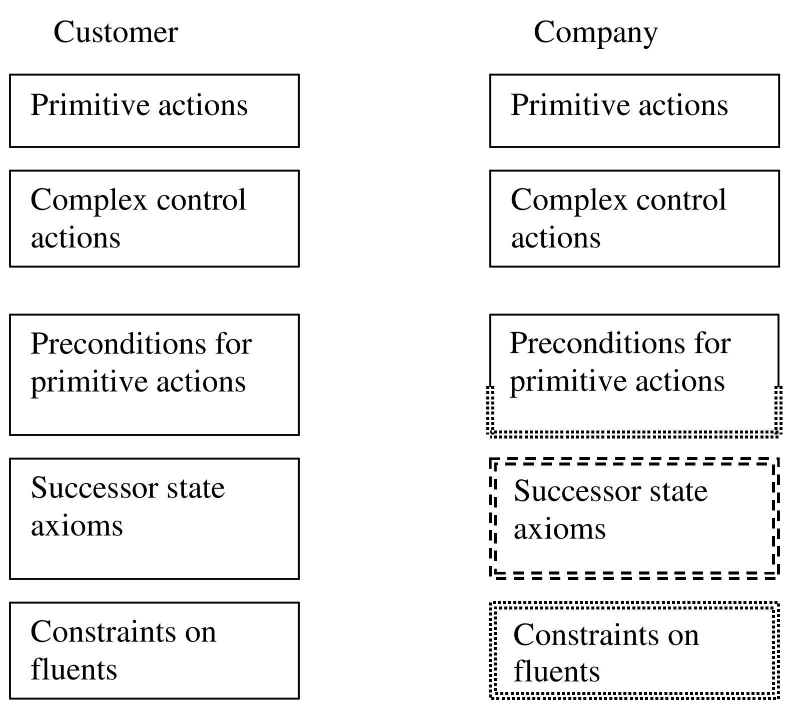

Fig.4: The knowledge base components of a claim: on the left: customer's component, on the right: company's component. Dotted boxes denote incomplete information.

In their claims, the customers explicitly mention their actions and most of the fluents expressing their mental states. Ambiguity in representation of actions for the customers may be caused by the errors in natural language processing and inadequate writing by the customer.

As to the company, we obtain a description of its actions and incomplete plot of its fluents in accordance to the customer's viewpoint. This data is insufficient to perform the prediction of the company's planning given only current scenario. On the other hand, using pure predictive machinery without actually modeling the company's sequence of action in the deductive way does not allow obtaining the latter in sufficient details to relate a given claim to a class.

The knowledge base components of a formalized claim are depicted at Fig.4. The customer component includes complete knowledge of the precondition for primitive actions, successor state axioms and constraints on fluents. The customer's component is ready for modeling its sequence of actions. However, the company's component is incomplete: we don't fully know preconditions for primitive actions; also, our knowledge on successor state axioms and constraints on fluents is incomplete.

Note that we cannot conduct learning for our system in a usual sense because no company would ever disclose its customer support policy to eliminate uncertainty in our representation for the company's fluents. We form the learning dataset using the past claims, where we may get information about the claim results and manually assign the class of reaction. At the same time we would never get information on the company's side.

Indeed, recognition of a claim is determining the following action of an opponent. A usual claim is a description of a process of interaction between two conflicting agents:

> My account was debited from a positive balance to a negative balance for "inactivity fees" which I was not fully informed. When I opened the account, I wanted to leave the money to earn interest. (Guess my surprise when I logged on to check my account and saw that instead of the positive balance I was looking forward to seeing, I saw a negative balance!) Now I cannot close the account until I pay NetBank for "inactivity fees". While I was in the process of closing the account, my account continued to be debited for the "inactivity fees" . This charge came 7 days after I was informed of my negative balance. The last straw was that I called to cancel my account, and the customer service representative asked "why". This oblivious and appalling customer service made me laugh at how NetBank can still be in business…

Id	Essence of service failure	Operation that confused a provider	Essence of customer service failure	Claim
1	Payment delay	Credit card payment	None	The problem concerns the billing or payment at
10	Reject promised service	Sign-up bonus	Cheating and avoiding expected service	The problem concerns the overall experience a
11	Unable to follow the agreement with customer and properly notify him/her	Account inactivity	None	My account was debited from a positive balance to a negative balance for "inactivity fees" which I was not fully informed. When I
12	Unable to correct amount of deposit		Cheating/disinforming/ignoring customer	I'm writing concerning an absolutely rediculous
14	Unable to verify existing of account for deposit	Transfer to unexisting account	Unwillingness/inability to understand client's	On May 24th I ordered goods and services for $270.- (Transaction #
15	Unable to handle lost credit card situation	Lost credit card	Responding with wrong information	They've stolen money from me! I've heard several months of BS and lies, and they still
16	Delays	Online bill payments	Cheating	NetBank delivered my on-line bill payments
26	Unable to handle a client with lost login	Transactions with no login information	Unwillingness/inabilityy to help/advice	I have a PayPal Account, this account was created roughly 1.5yrs ago. I created it strictly

Fig.5: A fragment of claim database.

Below we present a domain code for *claims* that includes enumeration of primitive and complex actions, and their pre- and post-conditions:

```
proc( complainToOtherEstablishments, ( askFriend #  askLawyer #
                              askConsumerAdvocCompany   ) ).
proc( settleDown, agreeToFixC # agreeToCompensateC # convinceToBeActingAsRequiredC ).
proc( mainComplaintScenario, [
                         [wrongDoC, askWhy, customerServiceIgnoreAskWhyC, askHowToFix,
if(long_wait, askHowToFix, askWhy), explainWronglyC, followWrongAdviceNoResult,
claimBadCustomerService,
findUnreasonableCauseForCustComplainC, askFriend, askLawyer, askConsumerAdvocCompany,
agreeToFixC, agreeToCompensateC, convinceToBeActingAsRequiredC]).
  ]).
```

```
poss(actionToBePredicted(A),S).
poss(UnsatisfAction,S):- holds(unsatisfied,S), member(UnsatisfAction,
[askWhy, askHowToFix, claimBadCustomerService ]).
poss(followWrongAdviceNoResult, S) :-  holds(disinformed, S).
poss(CompanyAction,S):- nonvar(CompanyAction), not member(CompanyAction,
[askWhy, askHowToFix,followWrongAdviceNoResult, claimBadCustomerService                        ]).

%% successor state axioms for primitive actions
%% a customer is unsatisfied if a Company did something wrong or this customer was
%% unsatisfied at the previous step
holds(unsatisfied, do(E,S) ):- E = wrongDoC;
                          E = customerServiceIgnoreWhy;
                          E = explainWrong;   E = findUnreasonableCauseForCustComplain;
              ( holds(unsatisfied,S), not member( E, [agreeToFix,
                      agreeToCompensate, convinceToBeActingAsRequired])
                                                                    ).
holds(disinformed, do(E, S) ):- E = explainWronglyC.
holds(company_untrusted, do(E, S) ):- holds(disinformed,S),  E = followWrongAdviceNoResult.

%%Everithing starts with a company wrongdoing (from the customer's perspective)
holds(wrongDoC, s0).

%% for JSM
unaryParam. %%% handles only unary features
%%reducedNegParam. %%% handles reduced formation of negative hypotheses (those who have at least two
atoms in common with a positive hypothesis)
%%%jsmGenParam.

% features are the set of all primitive actions and fixed
% objects are determined by available dataset and fixed
% targets are all actions that are pre-asumed to be possible at this step
% we may assume available actions are by counter-agent

features(Fs):-findall(F, primitive_action(F), Fs).
objects([o1, o2, o3, o4, o5, o6, o7, o8, o0]).
%targets([companyIsRight, consumerAdv,  tooHardToHelp]).

companyIsRight(o1).
askWhy(o1).
askHowToFix(o1).
explainWronglyC(o1).
claimBadCustomerService(o1).
findUnreasonableCauseForCustComplainC(o1).
askLawyer(o1).
agreeToFixC(o1).
agreeToCompensateC(o1).
convinceToBeActingAsRequiredC(o1).

consumerAdv(o2).
askWhy(o2).
customerServiceIgnoreAskWhyC(o2).
askHowToFix(o2).
explainWronglyC(o2).
followWrongAdviceNoResult(o2).
claimBadCustomerService(o2).
findUnreasonableCauseForCustComplainC(o2).
askFriend(o2).
```

Now we proceed to the protocol of the plan building

```
primitive_action(wrongDoC) , poss(wrongDoC,s0)
ok_primitive_action(wrongDoC) , poss(wrongDoC,s0)
list_conjunct(do(askWhy,do(wrongDoC,s0),_2AC),do([customerServiceIgnoreAskWhyC,askHowToFix,actionToBePredicted(_11C)],_2AC,_CC))
proc(askWhy,_2E8) , do(_2E8,do(wrongDoC,s0),_2AC)
primitive_action(askWhy) , poss(askWhy,do(wrongDoC,s0))
ok_primitive_action(askWhy) , poss(askWhy,do(wrongDoC,s0))
list_conjunct(do(customerServiceIgnoreAskWhyC,do(askWhy,do(wrongDoC,s0)),_438),do([askHowToFix,actionToBePredicted(_11C)],_438,_CC))
proc(customerServiceIgnoreAskWhyC,_474) , do(_474,do(askWhy,do(wrongDoC,s0)),_438)
primitive_action(customerServiceIgnoreAskWhyC) ,
poss(customerServiceIgnoreAskWhyC,do(askWhy,do(wrongDoC,s0)))
ok_primitive_action(customerServiceIgnoreAskWhyC) ,
poss(customerServiceIgnoreAskWhyC,do(askWhy,do(wrongDoC,s0)))
list_conjunct(do(askHowToFix,do(customerServiceIgnoreAskWhyC,do(askWhy,do(wrongDoC,s0))),_5C4),do(actionToBePredicted(_11C),_5C4,_CC))
proc(askHowToFix,_600) , do(_600,do(customerServiceIgnoreAskWhyC,do(askWhy,do(wrongDoC,s0))),_5C4)
primitive_action(askHowToFix) ,
poss(askHowToFix,do(customerServiceIgnoreAskWhyC,do(askWhy,do(wrongDoC,s0))))
ok_primitive_action(askHowToFix) ,
poss(askHowToFix,do(customerServiceIgnoreAskWhyC,do(askWhy,do(wrongDoC,s0))))
proc(actionToBePredicted(_11C),_79C) ,
do(_79C,do(askHowToFix,do(customerServiceIgnoreAskWhyC,do(askWhy,do(wrongDoC,s0)))),_CC)

  %% forming the environment for prediction: finding the target features.
getObjects([o1,o2,o3,o4,o5,o6,o7,o8,o0])
getFeatures([wrongDoC,askWhy,customerServiceIgnoreAskWhyC,askHowToFix,if(long_wait,askHowToFix,askWhy),explainWronglyC,followWrongAdviceNoResult,claimBadCustomerService,findUnreasonableCauseForCustComplainC,askFriend,askLawyer,askConsumerAdvocCompany,agreeToFixC,agreeToCompensateC,convinceToBeActingAsRequiredC,actionToBePredicted(_C74)])
```

going_to_predict_future_action_after([askHowToFix,customerServiceIgnoreAskWhyC,askWhy,wrongDoC])

===== target ===== explainWronglyC

Positive, negative and unassigned examples

```
[[askWhy(_1B54),askHowToFix(_1B74),explainWronglyC(_1B98),claimBadCustomerService(_1BC0),findUnreasonableCauseForCustComplainC(_1BF0),agreeToCompensateC(_1C30)],[explainWronglyC(_1A88),claimBadCustomerService(_1AB0),findUnreasonableCauseForCustComplainC(_1AE0),askLawyer(_1B20)],…]
[[askWhy(_1F3C),customerServiceIgnoreAskWhyC(_1F5C),askHowToFix(_1F90),findUnreasonableCauseForCustComplainC(_1FB4)],[askWhy(_1E40),customerServiceIgnoreAskWhyC(_1E60),askHowToFix(_1E94),followWrongAdviceNoResult(_1EB8),findUnreasonableCauseForCustComplainC(_1EEC)],[askWhy(_1D54),customerServiceIgnoreAskWhyC(_1D74),askHowToFix(_1DA8),findUnreasonableCauseForCustComplainC(_1DCC),agreeToFixC(_1E0C)]]
[[wrongDoC(o0)]]
```

Positive, negative and contradiction intersections:

```
[[askWhy(_26C4),askHowToFix(_26E4),explainWronglyC(_2708),claimBadCustomerService(_2730),findUnreasonableCauseForCustComplainC(_2760),agreeToCompensateC(_27A0)],[explainWronglyC(_25F8),claimBadCustomerService(_2620),findUnreasonableCauseForCustComplainC(_2650),askLawyer(_2690)],…] ,
[[askWhy(_2AAC),customerServiceIgnoreAskWhyC(_2ACC),askHowToFix(_2B00),findUnreasonableCauseForCustComplainC(_2B24)],[askWhy(_29B0),customerServiceIgnoreAskWhyC(_29D0),askHowToFix(_2A04),followWrongAdviceNoResult(_2A28),findUnreasonableCauseForCustComplainC(_2A5C)],…] , []
```

Background for positive, negative and inconsistent hypotheses:

```
[[askWhy(o1),askHowToFix(o1),explainWronglyC(o1),claimBadCustomerService(o1),findUnreasonableCauseForCustComplainC(o1),askLawyer(o1),agreeToFixC(o1),agreeToCompensateC(o1),convinceToBeActingAsRequiredC(o1)],…] ,
```

[[askWhy(o3),customerServiceIgnoreAskWhyC(o3),askHowToFix(o3),followWrongAdviceNoResult(o3),findUnreasonableCauseForCustComplainC(o3),askConsumerAdvocCompany(o3),agreeToCompensateC(o3)],…] , []
Prediction for positive, negative results and inconsistent prediction: ([wrongDoC(o0)], [] , []) , explainWronglyC – predicted!

[(([] , [] , [[wrongDoC(o0)]]) , explainWronglyC),(([] , [] , [[wrongDoC(o0)]]) , findUnreasonableCauseForCustComplainC)]
posActions = []
futureActions = [explainWronglyC,findUnreasonableCauseForCustComplainC]
predict_action_by_learning(explainWronglyC,do(askHowToFix,do(customerServiceIgnoreAskWhyC,do(askWhy,do(wrongDoC,s0)))))
ok_list_conjunction(do(askHowToFix,do(customerServiceIgnoreAskWhyC,do(askWhy,do(wrongDoC,s0))),do(askHowToFix,do(customerServiceIgnoreAskWhyC,do(askWhy,do(wrongDoC,s0))))),do(actionToBePredicted(explainWronglyC),do(askHowToFix,do(customerServiceIgnoreAskWhyC,do(askWhy,do(wrongDoC,s0))))),do(actionToBePredicted(explainWronglyC),do(askHowToFix,do(customerServiceIgnoreAskWhyC,do(askWhy,do(wrongDoC,s0)))))))
ok_list_conjunction(do(customerServiceIgnoreAskWhyC,do(askWhy,do(wrongDoC,s0)),do(customerServiceIgnoreAskWhyC,do(askWhy,do(wrongDoC,s0)))),do([askHowToFix,actionToBePredicted(explainWronglyC)],do(customerServiceIgnoreAskWhyC,do(askWhy,do(wrongDoC,s0))),do(actionToBePredicted(explainWronglyC),do(askHowToFix,do(customerServiceIgnoreAskWhyC,do(askWhy,do(wrongDoC,s0)))))))
ok_list_conjunction(do(askWhy,do(wrongDoC,s0),do(askWhy,do(wrongDoC,s0))),do([customerServiceIgnoreAskWhyC,askHowToFix,actionToBePredicted(explainWronglyC)],do(askWhy,do(wrongDoC,s0)),do(actionToBePredicted(explainWronglyC),do(askHowToFix,do(customerServiceIgnoreAskWhyC,do(askWhy,do(wrongDoC,s0)))))))
ok_list_conjunction(do(wrongDoC,s0,do(wrongDoC,s0)),do([askWhy,customerServiceIgnoreAskWhyC,askHowToFix,actionToBePredicted(explainWronglyC)],do(wrongDoC,s0),do(actionToBePredicted(explainWronglyC),do(askHowToFix,do(customerServiceIgnoreAskWhyC,do(askWhy,do(wrongDoC,s0)))))))
ok(proc(mainComplaintScen1,[wrongDoC,askWhy,customerServiceIgnoreAskWhyC,askHowToFix,actionToBePredicted(explainWronglyC)]),do([wrongDoC,askWhy,customerServiceIgnoreAskWhyC,askHowToFix,actionToBePredicted(explainWronglyC)],s0,do(actionToBePredicted(explainWronglyC),do(askHowToFix,do(customerServiceIgnoreAskWhyC,do(askWhy,do(wrongDoC,s0)))))))
%% below is the resultant plan
do(actionToBePredicted(explainWronglyC),do(askHowToFix,do(customerServiceIgnoreAskWhyC, do(askWhy,do(wrongDoC,s0)))))

We also present the algorithm for finding anti-unification:

```
antiUnify(T1, T2, Tv):- var_const( (T1,T2), (T1c,T2c)),
                              antiUnifyConst(T1c,             T2c,             T),
%writeln(antiUnifyConst(T1c, T2c, T)),
                              var_const(Tv, T).

antiUnifyConst(Term1, Term2,Term):-
     Term1=..[Pred0|Args1],   len(Args1, LA),
     Term2=..[Pred0|Args2],   len(Args2, LA),
    findall( Var, (  member(N, [0,1,2,3,4,5,6,7 ]),
             [! sublist(N, 1, Args1, [VarN1]),
    sublist(N, 1, Args2, [VarN2]),
    string_term(Nstr, N), VarN1=..[Name|_], string_term(Tstr, Name),
   concat(['z', Nstr,Tstr], ZNstr), atom_string(ZN, ZNstr) !],
                ifthenelse( not (VarN1=VarN2),
                    ifthenelse( ( VarN1=..[Pred,_|_], VarN2=..[Pred,_|_]),
        ifthenelse( antiUnifyConst(VarN1, VarN2, VarN12),
              ( antiUnifyConst(VarN1, VarN2, VarN12),Var=VarN12  ),
                     Var=ZNstr) ),
                     Var=ZNstr),
                     Var=VarN1)
             ), Args),        Term=..[Pred0|Args].
```

Conclusions

In this report, we merged deductive reasoning about action with the logic and combinatorial predictive machinery that implements the inductive, abductive and analogical kind of reasoning. Resultant formalism was found adequate in terms of modeling the conflicting agents under the automated processing of claims for consumer advocacy companies. Our model includes about ten consecutive actions, including deterministic and non-deterministic ones. For a given claim that mentions the initial sequence of actions, the system provides the prediction of the consecutive actions. As a result, the claim is assigned with the class of reactions. Our initial testing showed that the proper decisions are carried out in more than 80% of cases (20 testing claims, coming from 12 learning examples). We believe the method verification indeed requires more thorough evaluation, combined with information extraction capabilities. Also, the future study will focus on the improvement of the performance of our PROLOG implementation and on the NLP support.

References

1. De Giacomo, G. and Levesque, H. 1999. Progression and regression using sensors. In Proc. IJCAI – 99.
2. Levesque, H.J., Reiter, R., Lesperance, Y., Lin, F., Scherl, R.B., (1997) GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming* v31, 59-84.
3. Shoham, Y. 1988. Reasoning about Change: Time and Causation from the standpoint of Artificial Intelligence. MIT Press.
4. Reiter, R. 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, ed. *AI and Math. theory of computation.* p.359-380. Academic Press, San Diego 1991.
5. Wooldridge, M. 2000. Reasoning about Rational Agents. The MIT Press Cambridge MA London England.
6. De Raedt, L. (1999) A perspective on Inductive Logic Programming. In Apt, K.R, Marek, V.W., Truszczynski, M., Warren, D. Eds. *The Logic Programming Paradigm* Springer.
7. Lakemeyer, G. (1999) On sensing in GOLOG.in Levesque, H.J., Pirri, F., eds. *Logical foundations for cognitive agents*, Springer.
8. Grosskreutz & Lakemeyer, G. (2001) On-line execution of cc-GOLOG plans. in *IJCAI*-01.
9. Souchansky, M. (2001) On-line Decision-Theoretic GOLOG Interpreter. *IJCAI*-01.
10. Muggleton,S., 1992 ed. Inductive Logic Programming. Academic Press.
11. Finn, V.K. (1999) On the synthesis of cognitive procedures and the problem of induction NTI Series 2 N1-2 8-45.
12. Anshakov, O.M., Finn, V.K., Skvortsov, D.P.(1989) On axiomatization of many-valued logics associated with formalization of plausible reasoning, *Studia Logica v42 N4 423-447.*
13. Vinogradov, D.V. (1999) Logic programs for quasi-axiomatic theories NTI Series 2 N1-2 61-64.
14. McCarthy, J., Hayes, P., (1969) Some philosophical problems from the standpoint of artificial intelligence. In Melzer, B., Michie, D., eds. *Machine Intelligence 4*, 463-502. Edinburgh.
15. Shanahan, M. (1997) Solving the frame problem. MIT Press.
16. Reiter, R. (1991) The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, ed. *AI and Math theory of computation.* p.359-380. Academic Press, San Diego 1991.

17. Galitsky, B. (2001) Semi-structured knowledge representation for the automated financial advisor. In Monostori et al, eds. *Engineering of Intelligent Systems* LNAI 2070: 14th IEA/AIE Conference, p.874-879.
18. Wooldridge, M. (2000) Reasoning about rational agents MIT Press. Cambrdge, MA, London, England.