

PRAGMATIST: A RATIONAL AGENT AS PROGRAM DEBUGGER

VISIT HIRANKITTI

Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang
Chalongkrung Road, Ladkrabang, Bangkok 10520
Tel: 326-9969 Email: visit@ce.kmitl.ac.th

Abstract: *We propose that Pragmatist, a logical program debugger developed in [1] is a rational agent. Like a scientist or precisely a rational agent, Pragmatist applies scientific method for diagnosing and potentially correcting errors in its belief, which is a logic program supplied by the programmer, in order to attain an error-free belief with respect to an intended model. Pragmatist is formulated entirely in metalogic using logic programming techniques and is computable.*

In this paper we explain the metalogical framework of Pragmatist and its rationality. Additionally, we propose some extension on the Pragmatist.

Keywords: *Artificial Intelligence, Software Engineering*

1. INTRODUCTION

The *fundamental requirement* of a rational agent is an essential demand for *truth*—the correctness and completeness of that system with respect to some reference which the system purports to model. This reference is called the ‘intended model’ and comprises exactly those answers that the agent is required to derive.

An intended model can be viewed as a ‘world’. A rational agent must adapt to the given ‘world’ in order to survive in it. It must therefore be able to improve itself by correcting the *errors* it generates. Survival in the given ‘world’ depends on the agent being able to learn and to correct its *misbehaviour* in order to improve its degree of consistency with the ‘world’. If it cannot correct its errors then it will be *eliminated* (or falsified) by ‘natural selection’ in the sense of Darwinian Theory of Evolution.

An agent can be formulated, however abstractly, as an inference system comprising ‘assumptions’ and an ‘inference system’. Assumptions serve to specify the problem whilst the inference system manipulates the assumptions to derive conclusions, or ‘answers’. Such answers are supposed truths about the problem, in essence, supposed solutions to it. In order that these answers shall indeed be truths, complying with and covering the intended model, the inference system needs to be *sound* and *complete* with respect to that model.

If a wrong answer is derived from the agent, that wrong answer will be caused by errors within at least one of the system’s two components—i.e. within the assumptions or within the inference system, or both. If the inference system *is* sound and complete, the correctness of the answers depends only upon that of the assumptions. The role of the inference system then becomes *deduction*, that is, the sound explication of answers from the assumptions supplied as premises.

In an ideal state of affairs, an agent would solve any given problem perfectly straightaway, making no mistakes. More realistically, we seek a rational agent whose conclusions may at first be wrong because it has relied upon assumptions that are themselves wrong. By a process of *trial and error*, however, it can *learn* from its mistakes—i.e. from its *experience*—so as not to repeat those same mistakes in the future. As errors are detected, so they will be both remembered and *corrected*. Such a process amounts to **scientific method**.

We will assume that when an error is detected its cause must lie within the input assumptions which may be called a ‘program’. Our focus in this paper is upon ‘program debugging’, primarily upon its tasks of detecting and diagnosing errors in the assumptions. We will not consider to any great extent the third main debugging task of error correction. We will explain a

general framework for applying scientific method to debugging. The general framework forms a debugger called 'Pragmatist'. We argue that Pragmatist is a rational agent and we also propose some extension on it.

2. METALOGIC

An agent is defined abstractly as an entity which possesses belief and can reason about its belief. Logically, agent's belief and its reasoning about belief are assumptions and inferences respectively. Therefore, we may say that an agent consists of assumptions, or we may call 'belief' or 'program', and inferences.

An agent must employ some kind of metalogic when it reasons about its belief. This is because metalogic treats assumptions and inferences of the object level as objects which can be manipulated and reasoned about at the meta level.

Using metalogic, at the meta level the agent can reason about correctness and/or completeness of its assumptions; it can reason about soundness and/or completeness of its inferences; it can reason about how to control inferences, i.e. to decide which inferences should be adopted so that the agent can achieve its goals; finally it can reason about economy, i.e. to decide how it can make use of resources for reasoning, for example time, storage, cost, etc., in a reasonable way.

The Pragmatist framework employs metalogic of the first-order logic to perform reasoning about its belief and inferences at the meta level. In addition, it employs logic programming techniques to formulate that metalogic in order to make it computable and behave as an agent.

In what follows the framework of Pragmatist is formulated entirely in terms of metalogic.

3. BELIEF, TRUTH, AND ERRORS

Pragmatist maintains belief in the forms of meta-level logic programs. Each object-level logic program is a normal program [2] which is a set of normal clauses each having the form

$$A \leftarrow \text{literals}$$

where **literals** (called the *body* of the clause) is a conjunction of zero or more literals each of which has the form **B** or **not B** where **B** is an atom. Each clause is universally closed by implicit outermost quantifiers. Here, **not** denotes negation-as-failure as characterized in [3].

In a practical sense, a normal clause can be represented by a Prolog clause. Therefore, for the implementation issue a belief set held by Pragmatist is represented in terms of a Prolog program.

An object normal clause is treated at the meta level as a term in which object-level connectives are treated as *function symbols* at the meta level. Object-level elements are passed through the meta level via the Reflection Principle naming mechanism [4]. For example, an object-level normal clause $A \rightarrow B$ will be treated as a name ' $A \rightarrow B$ ' where the connective \rightarrow is treated as a function symbol ' \rightarrow ' at the meta level.

3.1 METALOGICAL REPRESENTATION OF BELIEF

Since an agent can use metalogic to reason about correctness and/or completeness of an object-level assumption, we can classify assumptions or belief set into three categories.

The first category is assumptions that have been tested fully and they are known to be both correct and complete with respects to an intended model. These correct and complete assumptions are treated as '*truth*' or '*facts*'. Suppose clause $A \rightarrow B$ is a fact, we express it at the meta level by predicate $fact(T, A \rightarrow B)$ and it is read as ' $A \rightarrow B$ is a fact belonging to theory T ,' the theory T in turn belongs to an agent. Sometimes, for convenience we may assume that the agent itself is such a theory.

The second category is assumptions that have already been tested and they are known to be incorrect with respects to an intended model. Thus, these assumptions are treated as '*errors*' which are the *negation of facts*, or *negation of truth*. Suppose a clause $A \rightarrow B$ is an incorrect assumption,

we express it at the meta level by predicate *incorrect_ass*($T, A \neg B$) and it is read as ‘ $A \neg B$ is an incorrect assumption belonging to theory T .’

The third category is assumptions that have not been tested or have not been tested fully, and consequently they are not known, with certainty, to be facts or errors. So, they are called ‘*hypotheses*’. A hypothesis $A \neg B$, which is a normal clause, is expressed at the meta level by predicate *hyp*($T, A \neg B$) which is read as ‘ $A \neg B$ is a hypothesis belonging to theory T .’

Usually an agent invents an assumption as a hypothesis and then test it to establish whether it is a fact or an error. If it is a fact, it will be applied without any doubt to solve problems in the future. If the assumption is still a hypothesis, there will be some risk of error emerging when it is applied to solve problems. However, if the assumption is found to be an error, it will not be applied for solving any problems in the future but be corrected, and consequently this corrected error may become a hypothesis or a fact later.

In the context of software development a fact here corresponds to a correct and complete subroutine which is later taken to be a library program to possibly be widely applicable in many applications. An incorrect assumption is taken to be an incorrect subroutine or a bug. Thus, hypotheses are any subroutines in a program which have not been tested fully and we still have some doubt about their correctness and completeness. Apparently, a program consists mainly of hypotheses. Therefore, many essential steps in a software process are aimed for discovering errors and correcting them so that truth can be discovered.

3.2 THE PROGRAM AGENT

We have said earlier that belief is possessed by Pragmatist, we said that in a rather loose manner. To be more precise, the belief is possessed by another agent called ‘Program agent’ and this Program agent is *a part of* the Pragmatist agent. Thus, this may be taken to mean that the belief belongs to the Pragmatist as well, or in other words, the belief is shared by both program agent and the Pragmatist.

According to the Pragmatist framework, the Program agent can only deduce an answer from its belief but cannot perform other kind of reasoning. Consequently, it is not regarded as a rational agent but a naive agent. For example, it cannot reason about correctness and/or completeness of its belief, but this will be performed instead by the Pragmatist. Therefore, the Pragmatist will perform several kinds of reasoning on the Program agent’s belief as its own belief; these kinds of reasoning bring rationality to the Pragmatist. For example, Pragmatist can test the Program agent’s belief and diagnoses errors in that belief. So, the Program agent plays only the deductive role to obtain an answer from its belief for the Pragmatist.

4. META-REASONING OF THE PROGRAM AGENT

An agent possesses not only assumptions but also inferences. There are three kinds of inferences employed by Pragmatist. They are abduction, deduction, and induction. In this section we describe a deductive inference employed by Pragmatist to derive an answer from its belief. Particularly, this deductive inference is performed via the Program agent. That is to say, when a query posed as a problem (or a goal) to solve by Pragmatist, Pragmatist will employ the Program agent to deduce an answer for this query from its belief.

The deductive inference adopted by the Program agent is in the form of a meta-interpreter. We named this meta-interpreter the ‘Pr-rP meta-interpreter’ [1] and we will call it ‘Pr-rP’ for short. The Pr-rP is a meta-level formulation of provability and unprovability; thus, it is an extension of the Vanilla meta-interpreter [5]; Pr-rP is logically derived from Clark’s completion of the provability.

4.1 PR-RP META-INTERPRETER

Pr-rP simulates SLDNF resolution at the object level. Practically speaking, Pr-rP can be understood as a metalogical formulation of a Prolog interpreter. That means that, it can be used to interpret a Prolog program. However, Pr-rP is more capable than any ordinary logic program interpreter, because it can record the assumptions adopted while performing query evaluation in

order to obtain an answer for a query posed to it. In other words, to deduce an answer for a query posed to Pr-rP, the interpreter will adopt some assumptions from Program agent's belief and record those assumptions. So, we can say that Pr-rP has a *consciousness* in the sense that it *knows* every deductive step it performs in order to deduce the answer and also *knows* which assumptions are adopted for deducing the answer which is the deductive conclusion. The set of assumptions Pr-rP adopts to deduce an answer is called a 'reason'. This means that, for any answer deduced by Pr-rP, Pr-rP will give a reason for deriving this answer.

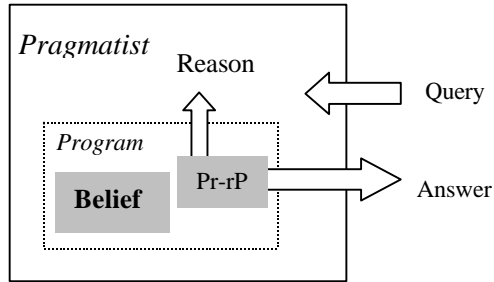


FIGURE 1. COMMUNICATION BETWEEN PRAGMATIST AND THE PROGRAM AGENT

The reason for obtaining an answer is essential for a rational agent like the Pragmatist. This is because a rational agent will request an explanation (reason) for any concluded answer how this answer can be derived. When an answer deduced by Pr-rP is found to be wrong, there must be errors in either the reason or the Pr-rP itself. Since Pr-rP is used in the way that ensures it is sound and complete, the errors (or bugs) then must be only in the reason. Therefore, Pragmatist will analyse the reason to identify errors or wrong assumptions in it.

Pr-rP is the deductive inference possessed by the Program agent. To solve a query the Pragmatist employs the Program agent to deduce an answer (for the query) from its belief using Pr-rP. Then Pragmatist will test the deduced answer for its correctness; if the answer is wrong, Pragmatist will diagnose the reason to see why the answer is wrong and try to identify errors in that reason. Error diagnosis by Pragmatist will be described shortly.

Pr-rP is formulated by predicate $demo_reason(Theory, Answer, Yes/No, Reason)$. This predicate is a provability and unprovability predicate which is an extension of $demo(Theory, Answer)$, a provability predicate. $demo_reason(T, A, Yes/No, R)$ is read as 'answer A is proved (when Yes/No is *yes*), or unproved (when Yes/No is *no*), from theory T by adopting reason R .'

The assumptions adopted in reason are three kinds. They are facts, hypotheses, and completeness assumptions. Facts and hypotheses are the assumptions which have been already described earlier; they are a part of Pragmatist's belief. Facts and hypotheses are predicates $fact(T, A \rightarrow B)$ and $hyp(T, A \rightarrow B)$ respectively. A completeness assumption is a *meta-level assumption to specify the completeness of assumption set*, i.e. the completeness of a set of hypotheses and/or facts. The completeness assumption is expressed by predicate $comp(T, A)$ which is read as 'a set of all object-level hypotheses and/or facts in theory T that *define* an object-level predicate A is (assumed to be) complete.' All these three assumptions are adopted by Pr-rP in a reason to deduce an answer for a query. Completeness assumptions can be thought of as a part of Pragmatist's belief, but they are rather different from the other three kinds of assumptions, i.e. facts, hypotheses, and errors, in that they are meta-level assumptions about completeness of a belief set consisting of facts and/or hypotheses.

$demo_reason(T, true, y, [])$.

$demo_reason(T, and(A, B), y, R) :-$
 $demo_reason(T, A, y, AR),$
 $demo_reason(T, B, y, BR),$

```

append(AR,BR,R).

demo_reason(T,A,y,[Hyp_or_Fact(T,A¬B) |R]) :-
hyp_or_fact(Hyp_or_Fact(T,A¬B)),
demo_reason(T,B,y,R).

demo_reason(T,not A,y,R) :-
isatom(A),
demo_reason(T,A,n,R).

demo_reason(T,and(A,B),n,R) :-
demo_reason(T,A,n,R).

demo_reason(T,and(A,B),n,R) :-
demo_reason(T,B,n,R).

demo_reason(T,A,n,[comp(T,A)|BRs]) :-
isatom(A),
comp(T,A),
findall(B,hypothesis_or_fact(T,A¬B),Bs),
all_fail(T,Bs,BRs).

demo_reason(T,not A,n,[R]) :-
demo_reason(T,A,y,R).

all_fail(T,[],[]).
all_fail(T,[B|Bs],R) :-
demo_reason(T,B,n,BR),
all_fail(T,Bs,BRs),
append(BR,BRs,R).

hyp_or_fact(hyp(T,A¬B)) :- hyp(T,A¬B).

hyp_or_fact(fact(T,A¬B)) :- fact(T,A¬B).

hypothesis_or_fact(T,A¬B) :- hyp(T,A¬B).

hypothesis_or_fact(T,A¬B) :- fact(T,A¬B).

```

FIGURE 2. THE EXTENDED PR-RP META-INTERPRETER

The extension we propose in this paper for Pr-rP are two kinds. Firstly, the reason returned from Pr-rP is now *flatten* whereas the one obtained from the previous version in [1] is *structured*. This extension requires a modification of the Pr-rP. The result simplifies the way Pragmatist reasons with the reason. Secondly, we add facts as another kind of assumptions to be adopted together with hypotheses by Pr-rP. This allows Pragmatist to reason with facts as well hypotheses. Note that these two extensions can still be handled naturally by the framework of Pragmatist presented in [1] due to its generality. The extended version of Pr-rP is illustrated by **Figure 2**.

We have already presented the extended Pr-rP. Now we shall explain how the Program agent employs this Pr-rP to deduce an answer from its belief. Suppose its belief is as shown in **Figure 3**; this belief is named theory '1'. At first, the belief is supplied to Pr-rP by the Program agent. After that the agent is ready to answer a query. For example, if we want to know Can *a* be deduced from its belief?, we have to pose query *?demo_reason(1,a,YN,R)* to the Program agent. This

agent then poses the query to Pr-rP and Pr-rP will return an answer $?demo_reason(I, a, y, [hyp(I, a \neg (not\ b)\ and\ c), comp(I, b), comp(I, d), comp(I, e), fact(I, c \neg\ true)])$; this means ‘ a is provable ($YN = y$) from theory I and the reason for this answer is $[hyp(I, a \neg (not\ b)\ and\ c), comp(I, b), comp(I, d), comp(I, e), fact(I, c \neg\ true)]$.’

$hyp(I, a \neg (not\ b)\ and\ c).$
 $hyp(I, b \neg\ d).$
 $hyp(I, b \neg\ e).$
 $fact(I, c \neg\ true).$

 $isatom(a).$
 $isatom(b).$
 $isatom(c).$
 $isatom(d).$
 $isatom(e).$

FIGURE 3. AN EXAMPLE BELIEF OF THE PROGRAM AGENT

The reason returned from Pr-rP means that to deduce $demo_reason(I, a, y, R)$, Pr-rP adopts assumptions $hyp(I, a \neg (not\ b)\ and\ c)$, $comp(I, b)$, $comp(I, d)$, $comp(I, e)$, and $fact(I, c \neg\ true)$. The answer and reason returned from Pr-rP will be given to the Program agent and the Pragmatist respectively.

4.2 EXPECTATION

When we consider on an agent’s side, we will find out the following. Because an agent always aims to answer a query (solve a problem) successfully, or in other words to give a correct answer, it therefore *expects* its answer to always hold *true in the external world*. Thus, the answer $demo_reason(T, A, YN, R)$ returned from Pr-rP should be understood as an *expectation* of the Program agent. That is, the Program agent will expect the deduced answer A to be a correct one, so will the Pragmatist who takes this answer from the Program agent.

Clearly a deduced answer from Pr-rP could be wrong with respect to an intended model which is a set of all answers intended to be deduced from the agent’s belief if the belief is to be correct and complete with respect to the external world. In other words, an intended model is the whole set of answers that the agent intends to model. If the agent, whose deduced answer is found to be wrong, cannot learn from this wrongness, it should not be regarded as a ‘rational agent’. The Program agent is such an agent, in that, when its deduced answer is found to be wrong, it cannot reason about this evidence. This is because it can only perform deduction to obtain an answer but cannot perform any inference apart from the deduction. So, we shall call the Program agent a ‘naïve agent’. This is different from Pragmatist. For Pragmatist, when it finds that the deduced answer from the Program agent is wrong, it will infer that there must be at least one error in the reason adopted to deduce the wrong answer.

In summary, Pragmatist takes an intended answer from the intended model to be the observation for the deduced answer obtained from Program agent. If the observation indicates that the deduced answer is wrong, Pragmatist will diagnose the reason adopted for deducing that wrong answer to find errors in it. So, Pragmatist is *rational* in the sense that it acquires the observation from the external world, i.e. the intended model, to justify the correctness and/or completeness of its belief.

To explain this in the context of software development, here the Program agent’s belief corresponds to a computer program being developed by a programmer, and the intended model is the set of all answers that the programmer intends his/her program to compute (if the program is to be correct and complete). So, here Pragmatist behaves as a programmer, in that, it regards an answer derived by the Program agent as being fallible with respect to the intended model and ready to diagnose errors in the reason adopted to derived the answer if the answer is found to be wrong

(with respect to the intended model). The external world we referred to earlier is precisely the intended model.

If we make Pragmatist a robot, then its belief will be the program it applies to solve a problem when relevant and an intended answer will be a fact it observes from its environment which is its external world. If Pragmatist gives an answer to a problem which contradicts an observed fact or the intended answer, Pragmatist will know that the answer is wrong and there must be some errors in the program it applies to give the answer.

We may also compare Pragmatist to a scientist, a scientist's belief is his/her scientific theory which is used to explain Nature—the scientist's external world. An answer predicted (deduced) from the theory could be wrong. He/she has to gather experimental facts from Nature to test the answer and verify theory. Here Nature is understood as the intended model since it provides observations to the theory.

Essentially, Pragmatist not only behaves like a programmer, it also behaves like a scientist; what are common in all the three agents is their rationality, i.e. the logic they apply to seek truth in different domains. Precisely the logic they all apply for seeking truth is *scientific method*. Pragmatist adopts the logic of scientific method for testing its belief, diagnosing correcting errors in its belief in order to attain truth—which is the state where its belief is all correct and complete with respect to the intended model [1]. Next we consider the Pragmatist's external world.

5. EXTERNAL WORLD

We treat the external world of Pragmatist as an agent called the 'oracle'. The oracle possesses an intended model and will provide intended answers to Pragmatist whenever being queried by it. To seek truth, Pragmatist requires observations in terms of intended answers from the oracle in order to test its belief, diagnose, and correct errors in its belief. So, the communication between Pragmatist and its external world (the intended model) is the communication between the agent and the oracle. We will explain how they communicate with one another in the next section. Scientific method can then be understood as logic of communication between rational agents (if there is more than one rational agents) and communication between rational agents and Nature.

We formulate the oracle using metalogic. Since the oracle is treated as an agent, an intended model is therefore the belief of the oracle and it is expressed into three logical forms. The three forms vary according to their degrees of expressiveness. The most expressive form is a (program) specification, the less expressive form is a (program) property, and the least expressive form is in terms of intended answers. A specification is expressed at the meta-level by predicate *specification*(Specs, Spec: $A@S$) where A is a computed answer and S is a specification for it. A property is expressed by predicate *property*(Props, Prop: $A@F$) where P is a property of computed answer A . Intended answers are obtained by querying the user which is another agent (a human agent) who can provide the answers. The communication between the oracle and the user is done by predicate *demo*(user, A, yes) which is read as 'the user infers (demonstrates) an intended answer is A .' The metalogical formulation of the oracle is illustrated by **Figure 4**.

$$\begin{aligned}
 &demo(oracle, not A, y) :- demo(oracle, A, n) \\
 &demo(oracle, not A, n) :- demo(oracle, A, y) \\
 &demo(oracle, A, YN) :- one((demo(specification, A, YN) ; \\
 &\quad\quad\quad demo(property, A, YN) ; \\
 &\quad\quad\quad demo(intmodel, A, YN))) \\
 &demo(specification, A, YN) :- specification(Specs, Spec: $A\leftrightarrow S$), \\
 &\quad\quad\quad demo(Spec: axioms, S, YN) \\
 &demo(property, A, n) :- property(Props, Prop: $A@F$), \\
 &\quad\quad\quad demo(Prop: axioms, F, n) \\
 &demo(intmodel, A, y) :- demo(user, A, y) \\
 &demo(intmodel, A, n) :- not demo(user, A, y), \\
 &\quad\quad\quad comp(user, A) \\
 &demo(intmodel, A, dn) :- not demo(user, A, y), \\
 &\quad\quad\quad not comp(user, A)
 \end{aligned}$$

This conclusion means that if an answer A deduced from reason **Reason** of Pragmatist is not the intended answer, the answer A is falsified by the intended answer deduced from the oracle and so is the reason **Reason** of the answer. In other words, this falsification will happen when we have:

$$demo(program, A, y) \text{ and } demo(oracle, A, n) \quad (\text{falsification})$$

The falsification of **Reason** implies that there must be at least one wrong assumption (or error) in **Reason**. After Pragmatist knows that **Reason** is falsified, it will start diagnosing **Reason** to identify wrong assumptions in it. The identified wrong assumptions are *bugs* to be removed from its belief.

7. BELIEF REVISION AS DEBUGGING

Pragmatist applies the three stages of scientific method, i.e. abduction, deduction, and induction, for diagnosing errors in the falsified reason [1]. That is to say, Pragmatist applies abduction to infer diagnoses or potential errors. Then it applies deduction and induction to test these diagnoses to identify the correct diagnoses; the testing step requires the oracle to supply new observations in terms of intended answers so that some of these observations may falsify and eliminate some diagnoses. By persisting in this process of diagnosis testing, it will come to the point where a diagnosis is verified to be a correct diagnosis which is precisely the error or bug identified by Pragmatist.

This error needs to be corrected, but our Pragmatist at this stage is not capable of doing complicated error correction. This will require more research on this direction. Let us suppose that if Pragmatist could correct the identified error in its belief, this would make its belief evolve in the way of eliminating the error and this would, in the long run, lead Pragmatist to the truth which is fully correct and complete belief because other errors would be identified and corrected in the belief.

Our proposal in this paper to allow facts as well as hypotheses to be adopted in a reason by Pr-rP can simplify the diagnosis process performed by Pragmatist. This is because when a reason, that consists of facts, hypotheses, and possibly completeness assumptions, is falsified only the hypotheses and the completeness assumptions are potential errors or diagnoses, but not the facts; this is because facts have already been known to be correct and complete with respect to an intended model. Therefore, Pragmatist can take facts out of the diagnosis candidate set.

For example, consider our previous example in 4.1, the reason [$hyp(I, a \neg (not\ b) \text{ and } c)$, $comp(I, b)$, $comp(I, d)$, $comp(I, e)$, and $fact(I, c \neg true)$] returned from Pr-rP is adopted for deriving answer $demo(program, a, y)$. Suppose the oracle derives $demo(oracle, a, n)$, which means that $demo(program, a, y)$ is a wrong answer. That is, both answer $demo(oracle, a, n)$ from the oracle and answer $demo(program, a, y)$ from the Program agent violate the correctness constraint. Therefore, the reason is falsified, Pragmatist infers the negation of the reason, i.e.

$$\neg[hyp(I, a \neg (not\ b) \text{ and } c), comp(I, b), comp(I, d), comp(I, e), fact(I, c \neg true)]$$

which is equivalent to

$$\neg hyp(I, a \neg (not\ b) \text{ and } c) \text{ or } \neg comp(I, b) \text{ or } \neg comp(I, d) \text{ or } \neg comp(I, e) \text{ or } \neg fact(I, c \neg true).$$

The last disjunct $\neg fact(I, c \neg true)$ can be eliminated from this disjunction, since it is always false. As Pragmatist derives all diagnoses from this disjunction, $fact(I, c \neg true)$ will not appear in any candidate diagnosis.

8. ECONOMIC RATIONALITY

Not only Pragmatist has logical rationality, it also takes economy into account in the context of scientific method as well. We have applied information theory to improve the efficiency of diagnosis testing by Pragmatist [1]. So, Pragmatist is also rational in the sense of conducting

scientific method using minimal resources, such as time. The whole framework of Pragmatist is summarized by **Figure 6**.

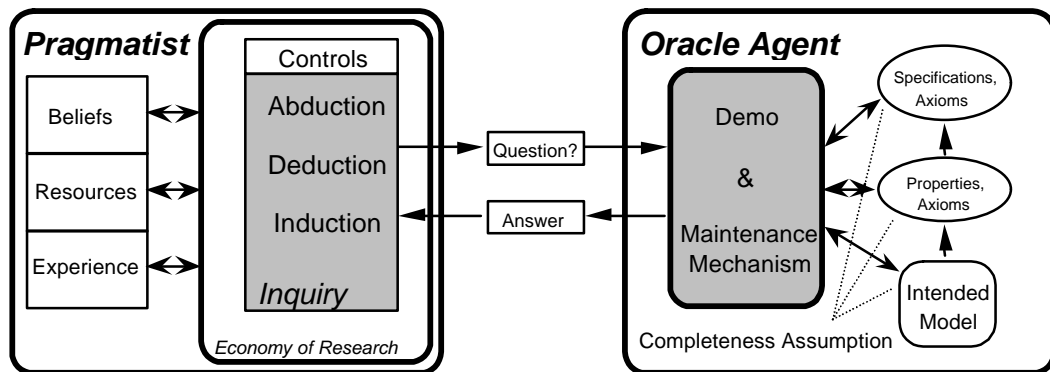


FIGURE 6. COMMUNICATION BETWEEN PRAGMATIST AND THE ORACLE

9. PHILOSOPHICAL DISCUSSION

The rationality of Pragmatist has its root in scientific method. Therefore, Pragmatist can be thought of as a metalogical formulation of scientific method as proposed by Hirankitti [1998]. We make this formulation computable; this benefit is mainly due to the advantage of metalogic.

As scientific method is the core of philosophy of science, it is interesting to compare the Four Noble Truth of the Buddhist philosophy to the scientific method. They are both logic for problem solving. There are many similarities between the two logics.

10. CONCLUSION

We have presented the framework of Pragmatist agent and argue that it is a rational agent, in that, it takes observations to test its belief and applies scientific method for diagnosing and correcting errors in its belief. It also takes economy into account in the context of scientific method to improve efficiency of the method.

We have also proposed some improvement on the agent to allow its Pr-rP to take account of facts and produce a flatten reason. This improvement will ease in some way the rational steps to be taken later by the agent.

REFERENCE

- [1] V. Hirankitti, *Applying Scientific Method to Program Debugging*, Ph.D. Thesis, Imperial College (1998).
- [2] J. W. Lloyd, *Foundation of Logic Programming* (Second Edition), Springer-Verlag (1987).
- [3] K. L. Clark, Negation as Failure, in *Logic and Database* (H. Gallaire and J. Minker Eds.), Plenum, New York, pp. 293-322 (1978).
- [4] R. W. Weyhrauch, A Prolegomena to a Theory of Mechanical Formal Reasoning, *Artificial Intelligence* **13** (1980).
- [5] R. A. Kowalski and J.S. Kim, A Metalogic Programming Approach to Multi-Agent Knowledge and Belief, in *Artificial Intelligence and Mathematical Theory of Computation* (V. Lifschitz, ed.), Academic Press (1991).
- [6] V. Hirankitti and C. J. Hogger, A Generalized Query Minimisation for Program Debugging, *Proc. of 1st International Workshop on Automated and Algorithmic Debugging, AADEBUG'93*, Springer-Verlag (1993).
- [7] C. S. Peirce, *Philosophical Writings of Peirce*, (ed.) Buchler, Dover Publication (1955).
- [8] K. R. Popper, *Logic of Scientific Discovery*, Hutchinson Education (1959).
- [9] E. Y. Shapiro, *Algorithmic Program Debugging*, The MIT Press (1983).

[10]Th. Stcherbatsky, *Buddhist Logic Vol. I*, Dover Publication, Inc. (1962).

[11]W. F. Clocksin and C. S. Mellish, *Programming in Prolog (3rd Edition)*, Springer-Verlag (1987).

APPENDIX

```
/* A Prolog Implementation of the Improved Pr-rP Interpreter */

:- op(100,xfx,<-).
:- op(90,xfy,and).

demo_reason(T,true,y,[]).

demo_reason(T,and(A,B),y,R) :-
    demo_reason(T,A,y,AR),
    demo_reason(T,B,y,BR),
    append(AR,BR,R).

demo_reason(T,A,y,[Hyp_or_Fact(T,A<-B)|R]) :-
    hyp_or_fact(Hyp_or_Fact(T,A<-B)),
    demo_reason(T,B,y,R).

demo_reason(T,not A,y,R) :- isatom(A),
    demo_reason(T,A,n,R).

demo_reason(T,and(A,B),n,R) :- demo_reason(T,A,n,R).

demo_reason(T,and(A,B),n,R) :- demo_reason(T,B,n,R).

demo_reason(T,A,n,[comp(T,A)|BRs]) :-
    isatom(A),
    comp(T,A),
    findall(B,hypothesis_or_fact(T,A<-
B),Bs),
    all_fail(T,Bs,BRs).

demo_reason(T,not A,n,[R]) :- demo_reason(T,A,y,R).

all_fail(T,[],[]).
all_fail(T,[B|Bs],R) :- demo_reason(T,B,n,BR),
    all_fail(T,Bs,BRs),
    append(BR,BRs,R).

hyp_or_fact(hyp(T,A<-B)) :- hyp(T,A<-B).

hyp_or_fact(fact(T,A<-B)) :- fact(T,A<-B).

hypothesis_or_fact(T,A<-B) :- hyp(T,A<-B).

hypothesis_or_fact(T,A<-B) :- fact(T,A<-B).

comp(X,Y).

%*****%
%   example programs   %
%*****%

hyp(1,a<-(not b) and c).
```

```

hyp(1,b<-d).
hyp(1,b<-e).
fact(1,c<-true).
isatom(a).
isatom(b).
isatom(c).
isatom(d).
isatom(e).

hyp(2,a<-b and c).
hyp(2,b<-d).
hyp(2,c<-true).
hyp(2,d<-true).

hyp(3,member(X,[H|T])<-member(X,T)).
isatom(member(_,_)).

hyp(3,set_diff([],L,[])<-true).
hyp(3,set_diff([X|Y],L,[X|Z])<-(not member(X,L)) and set_diff
(Y,L,Z)).
hyp(3,set_diff([X|Y],L,Z)<-member(X,L) and set_diff(Y,L,Z)).
isatom(set_diff(_,_,_)).

hyp(4,temp(Summer_day,high)<- (not raining(Summer_day))).
hyp(4,temp(Summer_day,low)<-raining(Summer_day)).
isatom(temp(_,_)).
isatom(raining(_)).

hyp(5,a<-b).
hyp(5,a<-c and (not d)).
hyp(5,c<-g).
hyp(5,c<-h).
hyp(5,g<-true).
hyp(5,d<-e and (not f)).
hyp(5,e<-true).
hyp(5,f<-m).
hyp(5,f<-n).
hyp(5,n<-true).

isatom(f).
isatom(g).
isatom(h).
isatom(i).
isatom(m).
isatom(n).

hyp(6,a<-b and e).
hyp(6,b<-c).
hyp(6,c<-(not d)).
hyp(6,e<-f).
hyp(6,f<-true).
hyp(6,d<-(not i)).
hyp(6,d<-g).
hyp(6,g<-h).
hyp(6,i<-true).

```