Journal of Artificial General Intelligence 3(3) 43-73, 2012
DOI: 10.2478/v10229-011-0021-5

Submitted 2012-07-28
Accepted 2012-12-03

Solving a Problem With or Without a Program

Pei Wang PEI.WANG@TEMPLE.EDU
Temple University
Philadelphia, USA

Editors: Kristinn R. Thórisson, Eric Nivel, Ricardo Sanz

Abstract

To solve a problem, an ordinary computer system executes an existing program. When no such program is available, an AGI system may still be able to solve a concrete problem instance. This paper introduces a new approach to do so in a reasoning system that adapts to its environment and works with insufficient knowledge and resources. The related approaches are compared, and several conceptual issues are analyzed. It is concluded that an AGI system can solve a problem with or without a problem-specific program, and therefore can have human-like creativity and flexibility.

Keywords: problem, solution, program, computation, algorithm, self-programming, knowledge and resources restriction, case-by-case problem-solving

1. Problem, Solution, and Program

This article discusses the approaches for an AI system to solve a problem, especially, what to do if there is no applicable program for the problem. The discussion is going to be carried out at the conceptual level, without addressing engineering details.

Let us start by specifying and analyzing the major notions involved in this discussion: *problem*, *solution*, and *program*. We will see that they are actually more complicated than they seem.

1.1 Problem and solution

From the outside, an AI system, sometimes called an "agent", can be described as interacting with its environment by obtaining a sequence of *percepts* as input, and producing a sequence of *actions* as output (Russell and Norvig, 2010; Wang, 2008). The former can be referred to as the system's "experience", and the latter, its "behavior". Formally, the system's recognizable percepts are often represented as a finite and constant set $\mathbf{P}$, and its executable actions as a finite and constant set $\mathbf{A}$. The experience of the system, from the initial time 0 to the current time t, is a sequence $E = \langle p_0, \cdots, p_t \rangle$, where $p_i \in \mathbf{P}$ for each i from 0 to t. Similarly, the behavior of the system is sequence $B = \langle a_0, \cdots, a_t \rangle$, where $a_i \in \mathbf{A}$.

Though this description looks natural (even trivial), there are still points that should not be missed. First, describing a system in this way is not the same as describing the system as a Turing Machine (Hopcroft and Ullman, 1979), where both the input and output of the system are strings of symbols. Second, very often the same system can be described at different levels and scopes of description, and in each of them, the sets of percepts and actions may be different. Each meaningful description defines a "virtual machine" out of

the actual system, and therefore corresponds to a certain level of description (Hofstadter, 1979). Mixing different virtual machines together in a discussion may cause confusions and misconceptions (Wang, 2007). In the following a system under discussion is always a virtual machine with fixed, though unspecified, **P** and **A**.

When talking about the system's problem-solving capability, it is natural to take a subsequence of experience as a "problem", and a subsequence of behavior as its "solution". Formally, a problem is a subsequence of E, $S = \langle p_{i_S}, \cdots, p_{j_S} \rangle$ where $0 \leq i_S \leq j_S \leq t$, and the system's solution is a subsequence of B, $R = \langle a_{i_R}, \cdots, a_{j_R} \rangle$ where $i_S \leq i_R \leq j_R \leq t$. Therefore, the solution always starts after the beginning of the corresponding problem (though may be before or after its ending), and multiple problem-solving processes may interweave in the system's experience and behavior.

Defined in this way, the *problem–solution* relation is similar to the *stimulus–response* relation in psychology, and the system's life-long history consists of many problem-solving events.

1.2 Algorithm and program

The previous definition of "problem" and "solution" seems natural, and is consistent with the ordinary usage of the words. However, it is not how "problem solving" has been treated in mathematics and computer science. In the classical sense of problem solving in mathematics, two additional requirements are needed:

1. The *problem–solution* relation must be predetermined. For a given problem $S \in \mathbf{P}^*$, there is a predetermined solution $R \in \mathbf{A}^*$, or a set of possible solutions, independent of *when* the problem appears in the system's experience (i.e., i_S). Therefore, the system serves as a *function* that maps the problems into the corresponding solutions.

2. The above "problem" S is considered as a "problem instance" that belongs to a "problem class" C of a certain type, and all the instances of the same class should be solved by the same method, which is formally specified as an *algorithm* that produces the solution (instance) for each problem (instance) in C.

For a given mathematical problem specified as a function, it is *computable* if and only if an algorithm can be found for the function. It is in this sense that "problem solving" is studied in mathematics, such as in the Church-Turing thesis (Davis, 1958). When people explore the possibility of going beyond the capacity of the Turing Machine, what they have in mind are the more complicated functions (Bringsjord and Arkoudas, 2004), rather than problem-solving processes that cannot be treated as "functions" at all.

In computer science, a *program* is the implementation of an algorithm in a computer programming language, as a fixed sequence of executable statements. Therefore, a problem is solved if a (correct) program can be composed for it. It is in this sense that "problem solving" is studied in computer science (Cormen et al., 2001). One of the most influential expression of this "computation–algorithm–implementation" procedure for problem-solving is contributed by Marr (1982).

Though this "problem solving" process is well-defined, and plays a central role in mathematics, computer science, as well as in many other domains, in AGI we have reasons to explore other approaches (Wang, 2004b, 2009a). Traditionally all the problems are solved

by the human beings, and computers just carries out the solution by following the given programs. In AGI, we expect a computer *to solve a problem by itself*, i.e., without a given program or algorithm for the problem.

This expectation comes from the observation that the human mind works in this way (Boden, 1991). The mind is "creative" in the sense that it can work out a solution for a novel problem that it never heard of previously; it is "flexible" in the sense that it can solve the same problem in different ways according to the context where the problem appears. Both of these properties deny the existence of a program followed in the problem-solving process. Since to many people AI is the attempt to give computers creativity and flexibility, it is natural to demand an AGI system to solve problems for which no program is given in advance.

1.3 To solve a problem without a program

Some people argue that a computer cannot solve a problem without following a program, since all activities of a computer are controlled by programs (Lucas, 1961; Dreyfus, 1979; Penrose, 1989). This argument comes out of a misconception (Wang, 2007): in this context "without a program" does not mean "without *any* program", but "without a *given* program *for that problem*" — after all, even the human mind cannot be said as working "without any program" in all levels and scales, given the various forms of regularity it displays, though such regularity is not shown as a rigid processing procedure for every problem it encounters.

If an AI system is given a problem, but within the system there is no existing program for that problem, there are still at least three approaches that *may* solve the problem:

1. Use a general-purpose program to process the problem. For example, to heuristically or exhaustively try all combinations of actions until the problem is solved. In this situation, the system follows a program all the time, though the program is not specially designed for the given problem.

2. Use some meta-level programs to build a program for the problem, then use the program to solve the problem. In this situation, the system does not have a program for the problem at the beginning, but has it at the end of the problem-solving process.

3. Use some meta-level programs to directly process the problem instance in a case-by-case and context-sensitive manner. Since such a process is usually unpredictable and unrepeatable, the system may not have a program for the problem even after it is solved.

It must be stressed that none of these approaches guarantees to solve every problem, and even when a problem is solved in the second or the third way, the *solution* may not have the properties demanded in mathematics and computer science, since it is often tentative and fallible. However, the same is true for the human mind when "solving" novel problems, so it is not a valid reason to reject these approaches for AGI.

Now let us briefly check the three approaches one by one.

A well-known example of the first approach is General Problem Solver (Newell and Simon, 1963). Though it works in some domain-specific problems, it cannot be used in AGI systems, given the excessive demand on computational resources of the approach. It is not

an issue that can be resolved by the progress of hardware — no matter how fast computers become, we cannot expect it to solve every problem by brute-force search in the solution space.

The second approach is basically what "machine learning" means in this context. In a learning system, there is typically a "learning element" and a "performance element" (Russell and Norvig, 2010). At the very beginning, if the system has no program for a given problem, it will try to build one first, and there are several ways that have been explored:

Inductive logic programming: The system is given sample input–output pairs, and tries to generalize from them to get a procedure that can produce the same results (Muggleton, 1991).

Reinforcement learning: The system uses reward-or-punishment signals to evaluate the effect of each action, then to select the best policy on when-to-do-what, so as to maximize the expected reward (Sutton and Barto, 1998).

Genetic programming: The system uses trial-and-error to select the best program among a population of randomly generated ones, through a simulated evolution process over many generations (Koza, 1992).

Each of these approaches has its own applicable situations, though also has some limitations, such as in its demands on training data and computational expense, which will be discussed in more detail later.

The third approach is arguably most similar to what happens in human problem-solving: when we "solve a problem", usually we directly deal with a problem *instance*, not a problem *class*, except under explicit demand to do so, such as in mathematics and computer science. The system does not have two parts, responsible for program-learning and problem-solving, respectively, nor are the two processes clearly separated. Even after the problem is solved, the system still does not necessarily obtain a program for similar problems in the future. Such a case-by-case working mode can be implemented in AGI systems (Wang, 2009a), and will be discussed with more details later.

For a given job, when multiple approaches each has its advantages and disadvantages, the best solution is often provided by the combination of them. In the following, such a combination is introduced.

2. A Unified Approach

In this section a new approach of problem-solving in AGI is introduced. This approach allows a system to use an existing program to solve a problem whenever such a program is applicable. When no such program exists for a novel problem, the system may solve the problem instance in a case-by-case manner, and in the process generate new programs to be used on similar problems in the future.

This approach is experimented in the AGI project NARS (Wang, 1995, 2006, 2013). In NARS, problem solving involves all components of the system, rather than is carried out by part of it. Since NARS is a complicated system, it cannot be fully described in a single journal article (which is probably the case for every AGI system). To deal with

this dilemma, this article proposes a *conceptual* solution to the problem, while leaving the technical details of NARS to other publications.[1]

2.1 Theoretical considerations

NARS is based on the theory that "intelligence" is *the ability for a system to adapt to its environment while working with insufficient knowledge and resources*. Concretely, it means that the system has the following properties:

Finite. The system has a finite capacity in information processing, such as the number of processors and the size of storage space.

Real-time. New tasks can show up at any moment, and each with a time requirement attached, such as a specific deadline or a level of urgency.

Open. The system can accept a new task with any content, as far as it is in the format or pattern that can be recognized by the system.

These constraints mean that the system cannot fulfill every task perfectly — for any non-trivial task, there usually is relevant knowledge that the system does not know, and known possibilities that the system cannot afford the time–space expense to explore. On the other hand, it is clearly not the case that in such a situation all responses are equally valid. It can be argued that this is exactly the environment where human beings display their *intelligence*, which is not omniscient and omnipotent, but a "highly developed form of mental adaptation" (Piaget, 1960), or a form of *bounded rationality* (Simon, 1957). Such a system is *adaptive* and *rational* in the sense that its actions are the best choices it can find, among the known alternatives, using the available resources.

NARS is a formal model built according to a *principle of relative rationality* (Wang, 2011). The model takes the form of a reasoning system, with formal grammar rules and inference rules, a semantic theory providing explanation and justification, and is implementable in existing computer systems. The reasoning system framework is preferred over the alternatives (such as the conceptual framework of a *computational* system or that of a *dynamic* system), because it provides a combination of justifiable steps (specified by the inference rules) and flexible ways to link the steps into complex processes (by the control mechanism), as well as has the extensibility to cover various types of knowledge and activity. These properties will become more clear later in the discussion.

To achieve AI using logic and reasoning is not a new idea at all (Hayes, 1977; McCarthy, 1988; Nilsson, 1991). However, the traditional theories dominating the study of reasoning systems, namely mathematical logic (in language and the inference rules), probability theory (in uncertainty processing), and the theory of computation (in computer implementation) are all based on certain assumptions of *sufficient* knowledge and resources, and therefore is improper for NARS. For instance, a logic-based model cannot represent beliefs with graded evidential support, and a probabilistic model demands a (consistent) probability distribution over the belief space (Pearl, 1988), which is not always feasible. Finally, algorithmic problem-solving in these systems do not allow content-dependent real-time responses. Some of these problems have been discussed in detail in Wang (2004a,b, 2009b).

1. Many publications on NARS can be accessed at `http://www.cis.temple.edu/~pwang/papers.html`

Designed under the opposite assumption, NARS ends up very different from the traditional reasoning systems in several major design decisions (Wang, 2006, 2013). In the following, the main aspects of the system are introduced one by one, with the focus on the relation between program and problem-solving.

2.2 Formal language and semantics

Like other computerized reasoning systems, NARS uses a formal language for both internal representation and external communication with other (human or computer) systems. The language is dubbed *Narsese*, and the latest version of its grammar rules are defined in Wang (2013), while the earlier versions can be founded in Wang (2006), as well as at the project website[2]. Given the nature of this article, only part of the language is introduced.

One special property that distinguishes Narsese from the other formal languages used in reasoning systems is that it is a *term-oriented* language. In this aspect, it belongs to the *term logic* school started by Aristotle (1882), rather than the *predicate logic* school started by Frege (1999). The basic component of Narsese is *term*, an identifier that names a concept in the system. An atomic term is just a sequence of characters coming from a given alphabet. In this article, the English alphabet is used, plus digits and a few special symbols, such as hyphen ('-').

A Narsese *statement* represents a conceptual relation, by relating a few terms to each other. The basic form of statement is an *inheritance statement* "$S \rightarrow P$", where S is the *subject term*, P the *predicate term*, and "$\rightarrow$" the *inheritance copula*, which is a reflexive and transitive relation between two terms. Intuitively, it states that S is a specialization of P, and P is a generalization of S. For example, "Water is liquid" can be represented as "$water \rightarrow liquid$" in Narsese. A variant of *inheritance* is the *similarity copula*, "$\leftrightarrow$", which is reflexive, transitive, and symmetric. For example, "A boat is a ship" can be represented as "$boat \leftrightarrow ship$". In general, statements indicate the *substitutability* among terms, which provides the basis of all reasoning activities in NARS.

To express complicated content in a term-oriented language, Narsese utilizes *compound term* of various types, each of which is formed from some component terms by a *term connector* that serves a predetermined function. Examples:

Sets. A term can be a set formed by given instances or properties. For example, *extensional set* "$\{Pacific, Atlantic, Indian, Antarctic, Arctic\}$" is a compound term that can be used in statement "$ocean \leftrightarrow \{Pacific, Atlantic, Indian, Antarctic, Arctic\}$" to enumerate the oceans; *intensional set* "$[red, round]$" is a compound term that can be used in statement "$apple \rightarrow [red, round]$" to say "Apples are red and round".

Intersections and differences. A compound term can be formed from the common (or different) instances (or properties) of other terms. For example, "$bird \cap [black]$" represents "black bird", while "$bird - [black]$" represents "non-black bird".

Products and images. Statements about conceptual relations that are not copulas can be equivalently rewritten as inheritance statements. For example, "Water dissolves salt" can be equivalently represented as inheritance statements "$(water \times salt) \rightarrow dissolve$",

2. `http://code.google.com/p/open-nars/wiki/InputOutputFormat`

"$water \rightarrow (dissolve\ /\ \diamond\ salt)$", and "$salt \rightarrow (dissolve\ /\ water\ \diamond)$", with different terms as subject and predicate. The '$\diamond$' symbol is a place holder indicating the position of the subject (or predicate) in the relation. Intuitively speaking, "$(water \times salt)$" is "the relation between water and salt", $(dissolve\ /\ \diamond\ salt)$ is "something that dissolves salt", and $(dissolve\ /\ water\ \diamond)$ is "something that water dissolves".[3]

The above term connectors share the same intuitive meaning with the corresponding set operators in set theory, though their exact definitions differ from those in set theory — a term is not necessarily a set, and nor is it defined pure extensionally by its instances. For formal definitions of the above compound terms, see Wang (2006, 2013).

Narsese allows a *statement* to be used as a (compound) term to form "higher-order statement", i.e., statement-on-statement. For example, "John knows that the Earth is round" can be represented as "$\{John \times (\{Earth\} \rightarrow [round])\} \rightarrow know$", where "$know$" is a relation between a cognitive system "$John$" and a statement "$\{Earth\} \rightarrow [round]$". As explained above, the same content can be represented by statements "$\{John\} \rightarrow (know\ /\ \diamond\ \{\{Earth\} \rightarrow [round]\})$", and "$\{\{Earth\} \rightarrow [round]\} \rightarrow (know\ /\ \{John\}\ \diamond)$".

Just like compound terms can be composed out of existing terms, *compound statements* can be composed from simpler statements, using statement connectors *conjunction* ('$\wedge$'), *disjunction* ('$\vee$'), and *negation* ('$\neg$'). Furthermore, two "higher-order copulas" are defined between statements: the *implication copula* ('$\Rightarrow$') intuitively stands for "if-then", and the *equivalence copula* ('$\Leftrightarrow$') for "if-and-only-if". They are partially isomorphic to the *inheritance copula* and the *similarity copula*, respectively. Again, for their formal definitions, as well as their similarity and difference with the corresponding connectives in propositional calculus, see Wang (2006, 2013).

Now we can visualize a set of Narsese sentence as a conceptual graph, with terms as nodes, and the four copulas as links. The copulas *inheritance* and *similarity* indicate the substitutability between *terms* in *meaning*, and the copulas *implication* and *equivalence* indicate the substitutability between *statements* in *truth-value*.

Since NARS is based on the assumption of insufficient knowledge and resources, in it the semantic notions *meaning* and *truth-value* are specified according to an "experience-grounded semantics" (Wang, 2005). As mentioned in the beginning of the article, in this discussion the "experience" of the system is its input history. The *meaning* of a term is defined by its role in the system's experience, and the *truth-value* of a statement is the evidential support the statement gets from the experience.

Concretely speaking, the semantic theory first defines the notion of "evidence" for a statement, then defines the corresponding truth-value. The *truth-value* of a statement has two factors in it: a *frequency* value in [0, 1] indicating the proportion of positive evidence among all available evidence at the current time, and a *confidence* value in (0, 1) indicating the proportion of current evidence among all available evidence at a future moment, after a constant amount of evidence comes. The details of the semantics can be found in Wang (2005, 2006, 2009b, 2013), and will not be repeated here. For the current discussion, it is enough to know that no statement is "absolutely true" in NARS. Instead, every statement is

3. In this article, *image terms* in Narsese are written in an *infix* format that is slightly different from the format used in the previous publications on NARS, though there is no conceptual change.

true to a degree that corresponds to available evidence, and may be revised by new evidence from time to time.

Similarly, the meaning of a term is defined by the links in the conceptual graph built from the system's experience that connect it with the other terms. As the coming of new experience, the meaning of a term also changes from time to time, depending on what the system knows about it.

2.3 Inference rules and steps

As a *term logic*, a typical inference rule in NARS is *syllogistic*, and it takes two premises to derive a conclusion. The two premises contain a common term, and the conclusion is between the other two terms. For example, when the copula involved is *inheritance*, there are three basic syllogistic inference rules:

deduction	**induction**	**abduction**
$M \rightarrow P \ \langle t_1 \rangle$	$M \rightarrow P \ \langle t_1 \rangle$	$P \rightarrow M \ \langle t_1 \rangle$
$S \rightarrow M \ \langle t_2 \rangle$	$M \rightarrow S \ \langle t_2 \rangle$	$S \rightarrow M \ \langle t_2 \rangle$
$S \rightarrow P \ \langle F_{ded} \rangle$	$S \rightarrow P \ \langle F_{ind} \rangle$	$S \rightarrow P \ \langle F_{abd} \rangle$

These three rules are named using the terminology introduced by Peirce (1931), though the exact definitions of the rules are not the same as those of Peirce. In NARS, the *deduction* rule extends the transitivity of the *inheritance* copula from binary to many-valued, while the *induction rule* and the *abduction* rule can be seen as "reversed deduction", since either of the two can be obtained from the *deduction* rule by exchanging the conclusion with a premise (the first premise for induction, and the second for abduction), then renaming the terms.

Each inference rule has an associated truth-function that calculates the truth-value of the conclusion from the truth-values of the premises (i.e., t_1 and t_2). The details of those functions are discussed in detail in Wang (2006, 2013). For the current discussion, it is enough to know that the *deduction* rule is "strong", in that the *confidence* value of its conclusion is relatively high (it can approach 1), while the other two rules are "weak", in that they only produce conclusions with low *confidence* values (less than 0.5), or we can say that they only produce "hypotheses" or "educated guesses". If the truth-values are omitted and the rule is applied among binary statements, the strong rules are still valid, while the weak rules are not.

Presented in this way, the above inference rules may look like a "rule-based methods for uncertain reasoning" that has been judged as wrong by mainstream AI (Russell and Norvig, 2010). NARS indeed shares some intuition with those methods, such as the use of "local" and "truth-functional" rules, but there is a fundamental difference in semantics — a truth-value in NARS does not indicate the extent to which the statement is in agreement with a (binary or statistical) "fact" in the world or in a model, but with the available evidence. In particular, in an inference rule the truth-value of the conclusion only measures the evidence provided by the premises of the rule, according to the experience-grounded semantics.

When the conclusions about the same statement are derived from different evidential basis, NARS uses a *revision rule* to calculate the amount of accumulated evidence, so as

to determine the truth-value of the conclusion that is based on the pooled evidence. This mechanism allows the weak beliefs to become stronger, as well as weighing of conflicting evidence. NARS can handle inconsistent beliefs, and that is a major advantage it has over approaches based on probability theory (Pearl, 1988), since to maintain the consistency (either in the logical sense or in the probabilistic sense) among the beliefs of an AGI system is not practically affordable (due to its resource demand), no matter how much it is desired to have consistency. This is a major reason for NARS to use "local" inference rules to handle uncertainty, rather than the "global" rules required by probability theory, such as Bayesian conditioning.

When the copula involved is *implication*, there are also three basic syllogistic rules:

deduction	**induction**	**abduction**
$M \Rightarrow P \ \langle t_1 \rangle$	$M \Rightarrow P \ \langle t_1 \rangle$	$P \Rightarrow M \ \langle t_1 \rangle$
$S \Rightarrow M \ \langle t_2 \rangle$	$M \Rightarrow S \ \langle t_2 \rangle$	$S \Rightarrow M \ \langle t_2 \rangle$
$S \Rightarrow P \ \langle F_{ded} \rangle$	$S \Rightarrow P \ \langle F_{ind} \rangle$	$S \Rightarrow P \ \langle F_{abd} \rangle$

This group of rules is isomorphic to the previous group, in the sense that the truth-functions are the same, though the meaning of the sentences are different, due to the use of different copulas.

In the rules on implication statements, when one term (M in induction, and S in the other two) is taken to mean "everything that the system knows" and is implicitly represented, we get a third group of rules that is a variant of the second group:

deduction	**induction**	**abduction**
$M \Rightarrow P \ \langle t_1 \rangle$	$P \ \langle t_1 \rangle$	$P \Rightarrow M \ \langle t_1 \rangle$
$M \ \langle t_2 \rangle$	$S \ \langle t_2 \rangle$	$M \ \langle t_2 \rangle$
$P \ \langle F_{ded} \rangle$	$S \Rightarrow P \ \langle F_{ind} \rangle$	$P \ \langle F_{abd} \rangle$

This last group is closer to how these three types of inference rules are specified in the current AI research, in the framework of propositional logic (Flach and Kakas, 2000).

Even though each rule is defined differently, the above syllogistic rules show a common principle: in its conclusion, each rule summarizes the information in the premises, and the conclusion usually contains compound terms that are not in the premises.

Beside the rules mentioned above, there are other inference rules in NARS to process the other copulas or term connectors, which are described in Wang (2006, 2013) and other publications.

2.4 Reasoning-learning processes

Now we can enrich our visualization of the beliefs of NARS: it roughly corresponds to a (mixed, multi, and hyper) graph that has four types of links that are weighted by the truth-values attached to the links. Both the topological structure and the weights of the graph constantly change by the coming of new experience, as well as by the inference activity of the system.

As a reasoning system, the running of NARS consists of repeated working cycle. In each cycle, the system works on a node in the conceptual graph, and uses the term naming

the node as the shared term of the premises. With two statements containing that term as premises, the applicable rules are invoked to derive one or multiple conclusions. Derived conclusions are added into the conceptual graph, leading to the creation of new links, and even new nodes. Therefore, even if there is no input during this cycle, the system's reasoning activity still changes the content of the memory, by changing the meaning of some terms and the truth-value of some statements. When a derived conclusion corresponds to an existing concept or belief, it will be merged into it, and therefore contribute to its meaning and/or truth-value, otherwise it will create a novel concept or belief in the system. In general, all the compound terms (including statements) defined in Narsese can be generated by the system itself, usually in more than one way.

Consequently, in NARS "reasoning" and "learning" are two aspects of the same process. As a reasoning system implementing a formal logic, every inference step in NARS is justified according to the experience-grounded semantics, which requires the conclusion to be based on the evidence provided by the premises. This is also the case for the composition and decomposition of compound terms, which are used to organize the system's knowledge more efficiently. Therefore, every activity in the system, including learning, is *reasoning* regulated by semantically justified formal rules. On the other hand, since NARS is designed to be adaptive, and to work with insufficient knowledge and resources, the conclusions are summaries of the system's experience, not theorems derived from a constant set of axioms (Actually "NARS" is the acronym of "Non-Axiomatic Reasoning System"). Therefore, the reasoning process has long-term impacts on the system's beliefs, by adding new terms and statements and modifying the current ones, and therefore is *learning*. These two words merely focus on different scopes of the process: "reasoning" provides a *micro* perspective on single steps, while "learning" provides a *macro* perspective on long-term effects.

The inference step to be carried out in each moment is fully determined by the two selected links in the selected node. Since NARS is assumed to have insufficient processing time, it cannot carried out all relevant inference for every task to be accomplished. Instead, the system dynamically allocates its resources among the tasks, according to their relative urgency and importance to the system. While processing a task, the system cannot consider all related beliefs, neither. Similarly, it gives the more useful and relevant beliefs higher chance to be applied.

To implement this strategy, the system maintains a priority distribution among the nodes in the conceptual graph, as well as among the links in each node. In each cycle, a node is selected probabilistically according to the priority values, then two links are selected in the node in the same way. The priority distributions are adjusted according to the immediate feedback collected after each inference step, so as to achieve a high overall efficiency in task processing.

In general, the priority of a data item (a concept, a task, or a belief) depends on three major factors:

- its *intrinsic quality*, such as the simplicity and clarity of a concept, or the confidence of a judgment in a task or belief;

- its *performance history*, that is, whether the item has been useful to the system in the past for task-processing;

- its *immediate relevance*, that is, whether it is directly related to the current situation, with respect to the current tasks.

Roughly speaking, the first two factors correspond to the "long-term memory" studied in psychology, while the last to the "short-term memory" or "working memory", though their difference in NARS is relative, not absolute — unlike many cognitive architectures (Newell, 1990; Franklin, 2007), NARS does not use separate memory models for different types of knowledge, but unifies them together, while still acknowledges their difference.

In this article, it is neither possible nor necessary to describe the details of the measurements involved above. It suffice to see that the system *selectively* uses its beliefs to process its tasks.

Since NARS is designed to use a constant amount of storage space, the number of nodes and the number of links within each node have upper bounds. When the memory (or a node) is "full", the node (or link) with the lowest priority value is removed. This policy, plus the "decay" that happens to all priority values, forms a "forgetting" mechanism. Consequently, the conceptual graph not only constantly gets new nodes and links from the environment and the reasoning process, but also loses old nodes and links.

In a "life-cycle" of the system, in principle the conceptual graph, or the system's memory, starts empty, though for practical applications it may start with preloaded content. During a life-cycle, the content of the memory changes constantly, as the result of new experience and reasoning activity, and the internal states of the system never repeat. Therefore, if a task is given to the system at different moments of a life-cycle, it may be treated more or less differently. This is where the "case-by-case problem-solving" feature comes from (Wang, 2009a).[4]

2.5 Procedural inference

The NARS described so far is basically a "question-answering" system — it answers questions according to available knowledge, under the restriction of available resources. To extend it into a "problem-solving" system specified at the beginning of the article, we need to give the system the ability to express and process *goals* (as "problems") and *actions* (as "solutions").

To do this in the framework of a reasoning system, the approach used in NARS is similar to logic programming (Kowalski, 1979; Lloyd, 1987; Muggleton, 1991). The basic idea is to give some sentences a "procedural interpretation", so that the statement "There is a relation R among arguments a, b, and c" becomes "To build a relation R among arguments a, b, and c". What differs NARS from ordinary logic programming is the logic involved and the inference control strategy.

As a preparation for *procedural knowledge*, In NARS, *episodic knowledge* is represented as *events*, which are statements with time-dependent truth-values. The temporal information of an event is mainly represented *relatively* with respect to another event, so the system can indicate that an event E happens *before* event F as "(E, F)", or the two happen *at the same time* as "$(E; F)$".

4. "Case-by-case" problem-solving is different from "case-based" problem-solving. In the former there is no predetermined algorithm for each case (i.e., occurrence) of a problem; in the latter a problem instance is treated as similar to one of the existing cases, which provides a predetermined algorithm.

An *operation* is a special type of event that the system can make it happen, usually by the running of a piece of software or hardware. Therefore, the "actions" mentioned at the beginning of the paper are represented in NARS as operations. Like other terms, an operation can be either *compound* (consisting of simpler operations) or *atomic* (irreducible to simpler operations).

An atomic operation has the form of "$op(a_1 \dots a_n)$", which is logically equivalent to an inheritance statement "$(a_1 \times \dots \times a_n) \rightarrow op$". When executed, a command *op* is called with the argument list "$(a_1 \dots a_n)$". The actual execution of the operation is usually not the responsibility of NARS, but its hosting system or an outside device.

As a general-purpose system, NARS uses a "hands plus tools" model for its operations. There is a small number of built-in operations, "the hands", that is directly recognized by the inference rules and executed within NARS, mainly to manage the system's internal activities. On the other hand, there is a "sensorimotor interface" that allows the registration of "plug-in" operations, "the tools", which drive the software and hardware of the host system in which NARS is embedded, as well as the outside systems and devices connected to NARS. In this way, NARS can serve as a *mind* within different (robotic or virtual) *bodies*, or an "intelligent operating system" managing various application programs and devices.

From the existing operations, various compound operations can be formed. The basic operation connectors are extensions of some connectors defined on statements or events:

Sequential: If A, B are operations, (A, B) is an operation that does A, then B.

Parallel: If A, B are operations, $(A; B)$ is an operation that does A and B together.

Conditional: If A is a statement and B is an operation, $(A \Rightarrow B)$ is an operation that does B when A is true.

Using these connectors, more complicated operations can be built step by step. In particular, repetitive executions (loops) can be achieved by recursion, as in logic programming. Therefore, Narsese can be considered as a simple programming language, with (compound) operations as "programs". Such a program can either be provided from the outside (as plug-in atomic operations or compound operations defined by input sentences), or derived by the inference rules from the given programs.

As mentioned previously, the meaning of a term in NARS is defined by its experienced relations with other terms. This is also the case for operations, For example, in the memory of a legged robot, the meaning of the operator "*walk*" consists of the following aspects:

- Its position in the operator hierarchy, as revealed by *inheritance* and *similarity* statements between it and other operators, like "$hike \rightarrow walk$", "$walk \rightarrow move$", and "$walk \leftrightarrow run$", with truth-values indicating their evidential support.

- Its conditions and consequences, as revealed by *implication* and *equivalence* statements between the operation and other events. For instance, for the robot to walk to location L, it needs to be in walkable distance, and the consequence will be that the robot being at L. In Narsese, this knowledge can be written as something like

$$(\#L \rightarrow [walkable]) \Rightarrow (walk\text{-}to(\#L) \Rightarrow (\#L \rightarrow [at]))$$

with a truth-value. The above statement is equivalent to

$$((\#L \rightarrow [walkable]) \wedge walk\text{-}to(\#L)) \Rightarrow (\#L \rightarrow [at])$$

In both statements, "$\#L$" is a variable term used as an argument.[5]

- A connection to the walking mechanism of the robot, which implements the operation.

It is important to remember that the meaning of "*walk*" (and its variant *walk-to*) includes all of the above aspects, and the first two change according to the system's experience. Even the last one may change, if the system's "body" changes, as a result of enhancement or damage. This is a major difference between NARS and other AI systems, where operations usually have fixed meaning (Russell and Norvig, 2010).

Also due to insufficient knowledge and resources, NARS usually neither knows all the conditions and effects of an operation, nor can it take all the known ones into consideration every time an operation is considered. Instead, only part of the conditions and/or effects of an operation is expressed in each belief about it. This treatment represents a very different attitude to the Frame Problem (McCarthy and Hayes, 1969) and the related issues: instead of attempting to fully specify all the preconditions and consequences of every operation, the system is designed to be tolerant to the incompleteness and uncertainty of the specifications (Xu and Wang, 2012).

A *goal* is a special type of event that the system attempts to achieve, like in logic programming. All the initial goals in NARS come from its designer or instructor, and the system produces derived goals from them according to the system's beliefs. Again, there are some important differences when the goals in NARS are compared with those in other approaches:

- A goal, represented by a *statement*, is a *partial* description of the environment, which is different from a *state*, which is normally a *complete* description of the environment.

- Since no statement can be absolutely true, no goal can be *completely* achieved, but only partially or approximately achieved, as indicated by the truth-value of the correspond statement. That is, if S is a statement serving as the content of a goal, it means the system is willing to execute some operation to make both the *frequency* and *confidence* of S to be as close to 1 as possible, though the *confidence* value cannot reach 1.

- Since NARS is an open system that may encounter unanticipated tasks and situations, there is neither request nor constraint on the consistency among its goals. Since goal derivation in NARS is based on the system's existing beliefs, a derived "descendant" goal may turn out to compete or even to conflict with its "ancestor" goals. Even the goals imposed to the system by its designers and instructors may contain conflicts and contradictions.

To handle the conflicts and competitions among goals, each goal has a *desire-value* attached, which is derived from *truth-value* — the desire-value of a goal G is defined to be the truth-value of statement "$G \Rightarrow D$", where D is a "virtual statement" indicating an idealized "desired state", whose content is not explicitly specified (Wang, 2006, 2013).

5. The exact representation is much more complicated than the representation used in this example. For more details, see Wang (2013).

In summary, there are three types of sentence in Narsese:

Judgment, a statement with a truth-value, punctuated by a period ('.').

Goal, a statement with a desire-value, punctuated by an exclamation mark ('!').

Question, a statement to be evaluated for its truth-value or desire-value, punctuated by a question mark ('?').

In this article, the punctuation mark of a sentence is often omitted when its type is explicitly indicated in the context.

In NARS, the most common form of *procedural knowledge* is an implication statement "$S \Rightarrow P$" where one of the two terms is an operation. Such a belief can be generated in several ways. It can be "implanted" into the system as an "instinct" when the operation is registered or defined. It can also be acquired directly from the system's experience after the operation comes into existence, or derived from other knowledge about the operation. As a reasoning system, NARS derives most of its knowledge (or beliefs) by itself.

Since events and operations are (special) statements, all the previously defined inference rules can be applied on them, and therefore reveal the preconditions and consequences of operations. For example:

- From knowledge "Event M is a sufficient condition of operation P" ($M \Rightarrow P$) and "Event S implies event M" ($S \Rightarrow M$), the *deduction* rule concludes that "Event S is a sufficient condition of operation P" ($S \Rightarrow P$).

- From knowledge "Event P is a sufficient condition of operation M" ($P \Rightarrow M$) and "Event S is a sufficient condition of operation M" ($S \Rightarrow M$), the *abduction* rule proposes an explanation "Event S implies event P" ($S \Rightarrow P$), though with a low confidence.[6]

- From knowledge "Operation S is followed by event P" (S, P), the *induction* rule generalizes the case into "Operation S has event P as a consequence" ($S \Rightarrow P$), also with a low confidence.

Similarly, the rules can be used *backwards* to derive goals. For example, if the system believes an implication statement "$S \Rightarrow P$", and P is a goal that is actively pursued by the system, then by backward inference using the deduction rule, S becomes a candidate goal, since its realization and the previous belief derive the realization of P by deduction.

Though similar inference happens in many AI systems, NARS is different here. Under the assumption of insufficient knowledge and resources, in the system usually there are multiple goals coexist at any given moment, and these goals may be inconsistent in what they described as desired, as well as compete with each other for the system's resources. Consequently, to let S become a goal just because of its relation with P may cause undesired by-products to other coexisting goals.

NARS uses a *decision-making* rule to deal with this situation. The previous "backward deduction" does not directly produce S as a goal, but increases its *desire-value*, and that may trigger the decision-making process on it. For a candidate goal S with desire-value d

6. Obviously, there is also a conclusion "$P \Rightarrow S$", probably with a different truth-value.

and plausibility p, it is accepted as a goal if $(d - 0.5)p$ is larger than a (positive) threshold value, which means that there is more evidence to like it than that to dislike it, and the system believes there is a way for the goal to be achieved. The plausibility of an event S is defined to be the truth-value of statement "$\#op \Rightarrow S$", where "$\#op$" is a variable term representing an anonymous operation.

Intuitively, the *desirability* and *plausibility* in NARS is similar to the *utility* and *probability* in traditional decision theory (Jeffrey, 1965), respectively. However, in NARS it is not assumed that the system knows all the paths to reach a given goal, nor that its knowledge about each known path is accurate (even as a probability). Since the goals constantly change and do not necessarily converge to a limit, NARS is not guided by a fixed fitness function, utility measurement, or reward signal, neither. Instead, the desirability and plausibility of each alternative need to be evaluated by the system itself through reasoning, according to its experience and using its available resources. This is why NARS does not take the decision-theoretic approach, which is popular in the study of planning, decision making, and reinforcement learning (Hutter, 2005; Pollock, 2006).

Beside desirability and plausibility, another factor considered when comparing alternatives is their simplicity. Due to insufficient resources, NARS prefers simple concepts, beliefs, and operations. The importance of "Occam's razor" in learning is widely acknowledged in AGI (Baum, 2004; Hutter, 2005), though in NARS it is a natural implication of the assumption of insufficient knowledge and resources, rather than an independent postulation by itself. Furthermore, simpler behaviors are not taken as necessarily "more likely to be correct", as assumed in Solomonoff induction (Solomonoff, 1964). In NARS, the desirability, plausibility, and simplicity of an event are evaluated separately, though the final selection among alternative operations is usually a trade-off among the three and some other factors.

3. An Example

NARS is an on-going AGI project. Aspects of this project has been described in two books (Wang, 2006, 2013) and a large number of journal articles and conference papers. Several prototypes of the system have been developed, and the current version is an open-source project, with documentations and working examples that can be tested in a demonstration applet.[7]

The mechanism needed for self-programming, as described in the previous section at the conceptual level, has been partially implemented — all the related inference rules have been coded in the demonstration applet, though the control mechanism is a preliminary version that can only handle simple problems. Though there are still remaining engineering issues to be resolved before this technique can be tested on practical problems, we can nevertheless use simplified examples to provide a proof of concept, which is exactly what this article intends to do.

To make the description of NARS more concrete, an example is used here to show the properties of the system. Since many aspects of the system are not addressed in this article, the process described in the following is not what exactly happens in NARS, but a simplified

7. At `http://code.google.com/p/open-nars/`

version that only keeps the major features.[8] In particular, the following simplifications are made:

- A problem-solving process is described as a sequence of inference steps, and in each step proper premises are provided. In this way, the choice of premises will not be discussed here, and irrelevant activities in the system are omitted.
- The truth-values of the premises and conclusions are omitted, though as described previously, every statement is true to a degree. In the following, truth-values are only mentioned qualitatively when necessary.
- The temporal indicators are omitted in the formal language, and consequently this article will not address topics like tense, difference between pre-conditions and post-conditions, and so on.

As a general-purpose reasoning system, NARS does not have a determined solution to any practical problem. Instead, the system's solution to a problem depends on the system's available knowledge and resources, and all domain knowledge comes from its experience, i.e., input history. In the following, we are going to describe several variations of a simple problem-solving process in NARS, with different given knowledge and problems.

3.1 Base-case scenario

In this example NARS is assumed to control a robot, with a set of Narsese operators corresponding to the commands of the robot. To make the example easy to understand, only high-level commands are used, though in principle any command of a robot or other system can be handled in this way, including those dealing with sensorimotor details.

The initial goal given to the robot is to open a certain door identified as $door1$. In Narsese, it is represented as

$$(\{door1\} \rightarrow [opened])! \tag{1}$$

Literally, the goal requests the system to do something so that "$door1$" will have the property "*opened*". The line number at the end specifies the location of the sentence in the system's experience, and will be used to identify the sentence in the following discussion.

As explained previously, the meaning of each term is defined by its experienced relations with other terms. For example, the name "$door1$" does not tell the system that it is a door. To specify that, a judgment needs to be obtained by the system, such as

$$(\{door1\} \rightarrow door). \tag{2}$$

For the robot to be able to achieve the goal, it must be able to take some operation, such as $push$, as well as some knowledge about its conditions and consequences, like

$$(((\#x \rightarrow [at]) \wedge (\#x \rightarrow door) \wedge push(\#x)) \Rightarrow (\#x \rightarrow [opened])). \tag{3}$$

8. For the actual input/output of similar examples produced by a demonstration version of NARS, visit the above project website. For the details of the involved grammar rules and inference rules, see Wang (2006, 2013).

This statement says that if the robot is at a door and push it, the door will be opened. Of course, it is not always the case, and its success frequency, as will as the reliability of the knowledge, are recorded in the truth-value of (3).

To satisfy the condition of the above knowledge, the robot needs another operation, such as the previously mentioned "*walk-to*", as well as the related knowledge

$$(((\#x \rightarrow [walkable]) \wedge walk\text{-}to(\#x)) \Rightarrow (\#x \rightarrow [at])). \tag{4}$$

Finally, the robot needs to know that "*door*1" has the property of being "[*walkable*]", that is, within walking distance, which is represented as

$$(\{door1\} \rightarrow [walkable]). \tag{5}$$

3.2 CBC problem-solving

When the system is given sentence (1) as a problem to be solved, and there is no existing program (i.e., an operation sequence) with this goal as its consequence, NARS will handle this problem in a Case-By-Case (CBC) manner (Wang, 2009a), which means to achieve the goal step by step, using the available knowledge and resources.

When sentences (1) and (3) are selected as premises, a version of the *deduction* rule will be used backwards to derive a new goal

$$((\{door1\} \rightarrow [at]) \wedge (\{door1\} \rightarrow door) \wedge push(\{door1\}))! \tag{6}$$

In the process the variable "$\#x$" in (3) is substituted by the constant term "$\{door1\}$". A decomposition rule will derive a child goal from (6):

$$(\{door1\} \rightarrow [at])! \tag{7}$$

Similarly, from (7) and (4), with the variable "$\#x$" substituted by "$\{door1\}$", a new goal is derived:

$$((\{door1\} \rightarrow [walkable]) \wedge walk\text{-}to(\{door1\}))! \tag{8}$$

When (8) and (5) are selected as premises, a simpler goal is derived

$$walk\text{-}to(\{door1\})! \tag{9}$$

Since goal (9) contains an atomic operation, it can be directly achieved by executing the operation.

At this moment, NARS will send a "*walk*" command to the robot controller, with "*door*1" as an argument indicating the destination of the walking. Also, a confirmation judgment is produced that satisfies the goal:

$$walk\text{-}to(\{door1\}). \tag{10}$$

From (10), (5), and (4), the system will generate an expectation by deduction

$$(\{door1\} \rightarrow [at]). \tag{11}$$

Assuming the operation is executed successfully, and consequently the robot moves to the door, then (11) will be confirmed by the sensory organs of the robot, so as to get a high confidence value. It satisfies (7), and consequently reduces the priority value of the latter.

With (11) and (2) established, the last goal is derived from (6):

$$push(\{door1\})! \tag{12}$$

This goal is achieved by executing the corresponding operation, like the situation of (9). After that, the initial goal (1) will be achieved in a way that is parallel to the achieving of (7). Now the problem is solved, with the solution consisting of the executed operation sequence "$(walk\text{-}to(\{door1\}), push(\{door1\}))$" that was sent from NARS to the robot controller.[9]

This problem-solving process is referred to as CBC, because the solution is not specified by any single problem-specific program, but by the cooperation of many programs. The cooperation is context-sensitive, in the sense that a similar problem, even another occurrence of the same problem, may be processed differently when it appears at a different moment.

3.3 Planning

If the initial goal is not to actually open the door, but to find a way to do so, then it is represented in NARS as the following:

$$(?op \Rightarrow (\{door1\} \rightarrow [opened]))? \tag{13}$$

where "$?op$" is a variable term to be instantiated, by an operation in this case (let us assume that can be somehow specified). In English, it is like the question "How to open that door?".

From the given knowledge (4) and (5), by deduction (with substitution) the system can derive

$$(walk\text{-}to(\{door1\}) \Rightarrow (\{door1\} \rightarrow [at])). \tag{14}$$

From (3), (2), and (14), by deduction the system can get

$$((walk\text{-}to(\{door1\}), push(\{door1\})) \Rightarrow (\{door1\} \rightarrow [opened])). \tag{15}$$

Now judgment (15) provides an answer to question (13), by suggesting a compound operation "$(walk\text{-}to(\{door1\}), push(\{door1\}))$" to instantiate the variable "$?op$". The compound operation can be considered as a "plan" or a "program" formed from atomic actions. After that, if goal (1) comes, from it and (15) the following goal will be derived:

$$(walk\text{-}to(\{door1\}), push(\{door1\}))! \tag{16}$$

This goal can be directly achieved by executing the compound operation.

Though this solution is the same as provided by the above CBC approach, these two approaches solve the problem in different "modes". In the CBC mode, the robot interacts with the environment while solving the problem, and takes actions as soon as they are

9. Among other details, the above description omitted the temporal inference involved in this example. The system must know that certain conditions must be satisfied *before* an action can be taken, certain consequences are established *after* an action is finished, and order among actions often matters. The details of temporal inference in NARS can be found in Wang (2013).

expected to achieve certain goals. On the contrary, in the planning mode the robot deliberates first, and takes actions only after a complete plan is established. Each of the two has its advantages and disadvantages, and is suitable for different knowledge-resource situations. If the system has the knowledge and resources demanded by planning, then the planning mode is better, given its execution efficiency. However, if the relevant knowledge is unreliable, planning may become a waste of time. When the number of possibilities makes complete planning impossible, the system may be forced to take tentative actions that aim at some intermediate goals.

Plans may have different levels of generality. For the current example, it is possible to keep the conditions in procedural knowledge (3) and (4) when combining them into knowledge about a compound operation:

$$(((\#x \rightarrow ([walkable] \cap door)) \wedge (walk\text{-}to(\#x), push(\#x))) \Rightarrow (\#x \rightarrow [opened])). \quad (17)$$

This belief provides a plan to open any nearby door, though its frequency value may be lower than that of (15), since it is more likely to run into counter examples — this will be explained immediately in the following. In such a situation, the system may keep both beliefs, and uses (15) on $door1$, and (17) on other doors.

3.4 Learning

Previously, the problem is solved mostly by deduction, which is reliable and efficient, though not always applicable. For example, what if the system does not know (3)?

Different from most other planning systems, NARS has the ability of learn procedural knowledge from observations, using "weak" inference rules like *induction*. As mentioned previously, from given premises S and P, the induction rule can derive "$S \Rightarrow P$", with a confidence value indicating the amount of evidence, which is at most one unit from a single observation.

In the current example, if the robot incidentally pushes a door, and the door opens, then from this experience by induction the system gets

$$(push(\#x) \Rightarrow (\#x \rightarrow [opened])). \quad (18)$$

Obviously, many such generalizations will turn out to be wrong. Later, each time a robot pushes something and that thing fails to become "opened", it will be a piece of negative evidence for (18), and decrease its frequency (while increase its confidence). In this way, NARS attempts to generalize its observations, and adjusts the truth-values of these hypotheses incrementally according to new evidence. In the system, the difference between "hypotheses" and "knowledge" (or "fact") is relative, mainly depends on the confidence value of a judgment.

From the success cases of (18) when applied to doors, the system can produces another more successful hypothesis by using induction again:

$$((\#x \rightarrow door) \Rightarrow (push(\#x) \Rightarrow (\#x \rightarrow [opened]))). \quad (19)$$

which, as mentioned before, can be equivalently rewritten as

$$(((\#x \rightarrow door) \wedge push(\#x)) \Rightarrow (\#x \rightarrow [opened])). \quad (20)$$

Compared to (18), this judgment will get a higher frequency, though it still will fail in various situations. Adding another condition on the object being "reachable", (20) will become (3). In the same way, the system can add more conditions, such as the object of pushing is not locked, not blocked, not too heavy, etc., and consequently better predict the consequences of the operations. However, the system can never *fully* specify the precondition for an operation to cause a certain effect.

Furthermore, more complicated conditions will increase the complexity of the program, so as to decrease the priority of the knowledge. Therefore, in the long run, the high-priority knowledge about an operation will be the judgments that have a balanced truth-value and complexity, so as to be both simple and reliable enough. For example, we can expect (3) to have a higher priority than (18) and (20), because it is much more reliable, while not complicated too much. At the same time, (3) also has a higher priority than a similar judgment with an additional condition that the door is not locked, since the increase in frequency may not worth the higher complexity, if doors are usually not locked in that environment.

In principle, all knowledge about the conditions and consequences of an operation can be learned from experience in this way, which is similar to Pavlovian conditioning, in which an animal predicts the future according to its past experience. This mechanism can also realize *imitation*, where a system's behavior is produced from knowledge obtained from the actions of other systems, with their observed preconditions and consequences.

A belief in NARS can be acquired in different ways. For example, the same implication relation can be derived by induction (from observed correlation between two statements) and by deduction (from assumed causal relations). When both happen, the revision rule will combine the conclusions derived from distinct evidence, to get a summarized conclusion with a higher confidence value.

4. Comparison and Discussion

Now let us see what makes the approach described previously novel and different.

4.1 NARS vs. other approaches

The problem solving process in NARS shares some ideas with several existing AI techniques, such as *planning* (Albus, 1991; Bratman, Israel, and Pollack, 1988; Fikes and Nilsson, 1971; Newell and Simon, 1963), *production system* (Anderson, 1983; Laird, Newell, and Rosenbloom, 1987), *inductive logic programming* (Muggleton, 1991), *genetic programming* (Koza, 1992), *behavior-based robotics* (Arkin, 1998; Brooks, 1991), and *reinforcement learning* (Kaelbling, Littman, and Moore, 1996; Sutton and Barto, 1998). At the same time, NARS is not based on any of them, due to the system's commitment to *adaptation with insufficient knowledge and resources*, which is not assumed in the same sense by any of the other approaches.

Therefore, what distinguishes the NARS-style problem-solving from the other AI/AGI techniques are not *performance* (in terms of domain, accuracy, speed, and so on), but the *restriction* under which the system works. In the following, a few key topics are discussed.

Problem, solution, and program, revisited

NARS treats each input Narsese sentence as a *task* to be processed. As mentioned previously, there are three types of them:

Judgment. To process it means to use it to solve the pending questions and goals, to revise the relevant beliefs, and to derive new judgments.

Goal. To process it means to execute some operations to change the environment or the system itself, so as to get a judgment that best satisfies what the goal demands. In this process, other goals may be generated.

Question. To process it means to get a judgment that is the best answer to the question among the candidates the system has found. In this process, other questions may be generated.

As explained previously, since NARS has insufficient knowledge and resources, few tasks can be processed to its "logical end", but will only be processed using part of the system's knowledge, so the obtained solution depends on the currently available knowledge and resources, which usually change from moment to moment.

In a broad sense, each task is a problem, and its processing is the solution. In a narrow sense, however, only a goal is considered as a problem, and its solution is the sequence of operations executed by the system to achieve it. Both senses of "problem-solving" are consistent with the general definition introduced at the beginning of the article. This discussion focuses on the narrow sense of problem-solving, though most conclusions apply to the general sense, too.

In NARS, the notion of "program" also has two senses. In the ordinary sense, NARS as a software consists of nothing but programs, written in a programming language, such as Java and Prolog (for the currently maintained implementations). On the other hand, to an outsider who describes the system's *externally observable* experience and behavior, a program is a (usually compound) operation. To avoid confusion, in the following they are referred to as "meta-level program" and "object-level program", respectively, since the former composes and executes the latter.

With respect to the solving of the problems mentioned above, the directly relevant programs are whose at the object-level. For a given problem, in NARS it can be solved in three ways:

With a preexisting program: If the goal directly calls an existing (atomic or compound) operation (like in the sentences (9), (12), and (16) of the previous example), then the execution of the operation may solve the problem.

With a "by-demand" program: When the goal comes to the system, there is no ready-made operation for it. However, during the processing of the goal, a compound operation is formed for it (or for a problem class it belongs), then the execution of the operation may solve the problem, as discussed in Section 3.3. After that, the compound operation will become part of the system's knowledge, and can be used (as an existing program) for future problems.

Without a program: The goal is achieved by executing a sequence of operations, though there is no compound operation in the system corresponding to this sequence, either before or after the problem is solved, as discussed in Section 3.2. If the goal reappears, the system will solve it again according to the available knowledge and resources at the moment, and the solution may be different.

In NARS, a goal may be processed in more than one way at the same time. Even when there is a program for a goal, the system still can explore alternative paths to achieve it. The "reactions" in reactive systems (Brooks, 1991) is represented in NARS as operations directly triggered by specific input stimulus, so they are expressed using the same format as plan-based actions (Fikes and Nilsson, 1971). In this way, what is taken to be opposite approaches in robotics, reactive vs. deliberative (Murphy, 2000), are unified in NARS. The system has some knowledge in the "stimulus-response" from, which allows certain behavior to be directly triggered by the corresponding percepts. However, the majority of the procedural knowledge is indirectly related to percepts, and the related beliefs link operations and goals to concepts and beliefs that summarize the system's experience in various ways. It is this type of knowledge that allows the system's behavior to depend not only on the current situation, but also on the system's long-term goals and past history.

The above definitions of "problem", "solution", and "program", as well as the relation among them, are clearly different from those accepted in traditional computer science, as described in Section 1. These differences have profound implications. For example, in NARS, the class of "solvable problems" is not the same as the class of "computable functions" as studied by Church, Turing, and many others (Hopcroft and Ullman, 1979), since the problem–solution relationship is not necessarily fixed as a function. It does not mean that the traditional notions are wrong, but that "problem solving" in mathematics and computer science is different from the same phrase when used in everyday life and artificial intelligence. It is rational to assume sufficient knowledge and resources in the former situation, but not so in the latter (Wang, 2011).

Hands plus tools

An important design decision in NARS is to allow an arbitrary software or hardware device to be called within NARS through a sensorimotor interface, where all controllable actions of the device are registered in NARS as operations implemented outside the system.

The advantage of this approach is that it keeps the intelligent core simple and general, and at the same time allows the whole system to be extended in various ways. Since the object-level programs can use the plug-in operations as atomic units, their capability is not limited by the built-in operations of NARS.

The price to pay for this approach is that the "out-of-box" NARS, without external tools and domain knowledge, has little built-in practical problem-solving capability. Whether this is an issue depends on the objective of the research. NARS is designed according to the belief that "intelligence" does not measure a system's problem-solving ability, but how fast this ability increases over time, so it is fine for the system to have weak innate problem-solving ability, as long as it has strong learning ability.

For AGI research, this approach has special significance, because of its general-purpose nature. The "hands plus tools" model allows a general-purpose reasoning system (a "mind")

to be equipped with certain (built-in) domain-independent operations (its "hands"), as well as (plug-in) special-purpose operations (its "tools"). The knowledge and skills related to the operations are partly given, but mostly learned by the system from its own experience. For such a system, it is neither necessary nor possible for its designer to restrict the situations the system may run into and the problems it may encounter. Since NARS assumes much less on what the system knows and how much time and space it can afford, it serves AGI research better than techniques that can only be applied under certain strong assumptions on available knowledge and resources.

Operation choice

At a given moment, the set of atomic operation is constant, so the system's behavior fully depends on *which* operation is chosen for execution.

First, let us see two extreme situations. In traditional program-based problem-solving, a problem triggers the execution of a program, and the order of operations in the program is predetermined. Therefore, the system has perfect information on which operation should be chosen at every moment. On the contrary, if there is no program for the problem, and the system has no relevant information on the operations, any choice is as valid as random and arbitrary choices.

AI problems are typically between these extremes. The system usually has some information about which operation is more suitable than the others, though this information is fallible. Consequently, many AI techniques use various evaluation functions to numerically compare the alternatives, such as heuristic function, fitness function, expected reward, etc. (Russell and Norvig, 2010). NARS does something similar. When a goal G may be achieved in multiple approaches represented by beliefs "$(C_i \wedge O_i) \Rightarrow G$" (where $1 \leq i \leq n$, and O_i is an operation and C_i its condition), respectively, the major factors influencing the choice of an approach include:

- The truth-value of the belief, which indicating how *promising* this approach is.
- The priority-value of the belief, which indicating how *relevant* this approach is.
- The complexity of the belief, which indicating how *expensive* this approach is.

Even though eventually all these factors must be combined to choose an operation for execution, they are defined and maintained separately, so as to support different usages of the operations. With multiple goals at the same time, in NARS there is no overall fitness or reward value associated with an operation. Instead, it is the *beliefs* linking an operations to various conditions and consequences that get evaluated, so that different operations can be chosen for different goals, in different contexts.

Furthermore, as a reasoning system, NARS can infer the conditions and effects of a novel compound operation according to the knowledge about its components, while some other techniques, such as genetic programming, must actually execute an operation to get knowledge about it, which is usually more expensive and dangerous. An intelligent system should be able to recognize some (though not all) improper operations without experimenting them in the environment.

Scalability

Scalability is a major challenge to all AI systems, especially to AGI systems. Being general-purpose, it is unrealistic for an AGI system to consider all combinations of all operations that may be relevant for a problem.

Traditionally, scalability is considered as an issue about computational complexity, and its solution should be algorithms with low complexity order (Littman, Goldsmith, and Mundhenk, 1998). Under the assumption of insufficient resources, in NARS even an algorithm that requires a constant time may ask too much in a certain situation. As explained previously, NARS solves this type of problem in a CBC mode, without following a program. In this situation, it does not make sense to ask how much time the system needs to solve the problem, since the system will take as much time as affordable or allowed, and find the best solution within this constraint. In this aspect NARS is similar to an "anytime algorithm" (Dean and Boddy, 1988), though there is no single algorithm responsible for the problem-solving process.

Working in this way, NARS no longer has the predictability and repeatability of the conventional computational systems with respect to a given problem, but at the same time, it has the creativity, flexibility, and adaptability that are expected from intelligent systems. Like it or not, this type of behavior is an inevitable consequence of "adaptation with insufficient knowledge and resources".

Not following a predetermined algorithm allows the problem-solving process in NARS to be adaptive, which lead to higher efficiency and scalability. As by-products of CBC problem-solving, compound operations are composed so that the same process can be described using fewer steps. The compound operations are kept in the system (with various priority) and used just like the atomic operations, so that planning becomes hierarchical and incremental. Though *hierarchical planning* is a natural idea, and has been used in AI in various forms (Russell and Norvig, 2010), it is usually formalized as a certain algorithm, which is different from what NARS does, where it is carried out as reasoning.

Learning as reasoning

NARS clearly puts a lot of weight in learning, though its treatment of learning is different from the common "machine learning" approaches (Mitchell, 1997; Russell and Norvig, 2010), where a learning process is specified using an algorithm.

In NARS, learning refers to all the changes in memory when the system runs, which takes various forms:

- New sentences come either as input or as derived conclusions, so the meaning of existing terms is changed accordingly.
- The truth-value of certain judgments are revised according to new evidence.
- New compound terms (including operations) are composed from existing ones, to summarize the system's knowledge.
- Priority distributions among terms and sentences are adjusted according to the changes in the environment and the system.

- Terms and sentences with low priority values are removed to make room for the new ones.

Each of these activities either follows certain inference rule, or is part of the inference control mechanism. Therefore, in NARS *learning* is carried out by *reasoning*, if we divide the process into steps and check their justification. On the other hand, if we study the long-term effects of the *reasoning* process, it is more natural to describe it as *learning*. If the process is about the achieving of a goal, it can be considered as *problem solving*; if in the process compound operations are formed, then it may be called *planning*. These different names do not correspond to separate processes in the system, but different aspects of the same process. There is no separate "learning element" and "performance element", as in many AI systems (Russell and Norvig, 2010).

By treating learning as reasoning, NARS is similar to the "Inference Theory of Learning" proposed in Michalski (1993), though the two approaches have very different language, semantics, inference rules, and control mechanism. They are compared in Wang (2000).

Unlike genetic programming (Koza, 1992), in NARS there is no pure-random factor in the generation of new behaviors. Even so, the behavior of NARS is still not predictable from the given problem alone, because it is *context-sensitive*, in the sense that the system's solution to a problem depends not only on the design of the system and the problem to be solved, but also on the *time* when the problem is encountered by the system, because the internal states of the system never repeat during a life-cycle, and in different states, the problem is handled differently.

In summary, in NARS learning is case-by-case, real-time, open-ended, life-long, and multi-strategy, rather than realized by a program following a "learning algorithm".

4.2 Conceptual issues

Beside the technical issues, there are important conceptual issues involved in this discussion. In the following, several anticipated objections to the NARS-style problem-solving are analyzed.

"*NARS does not really solve the problems.*" As described before, the solutions provided by NARS may be rejected by future observation or further consideration, and they do not have the repeatability demanded by the traditional theories, so are not acceptable as "solutions" in them (*How can a "solution" start with "I guess"?*). However, if a system *has to* deal with problems for which it has insufficient knowledge and resources, by definition it cannot provide "solutions" in the traditional sense. Actually, this is the type of *solution* produced by the human mind for many practical problems. These solutions do not meet the requirements of *absolute rationality* as specified in theories like mathematical logic or probability theory, but can be justified as showing *bounded rationality* (Simon, 1957) or *relative rationality* (Wang, 2011).

"*There are still programs in NARS.*" Of course, NARS (when implemented) is nothing but a group of programs plus some knowledge bases. However, unlike conventional computer systems, its problem-solving capability is not limited by the existing programs. In the discussions on creativity and flexibility, "without a program" is always with respect to a given problem, explicitly or implicitly, and the "creative

solutions" are not coming from nowhere (Boden, 1991). The existence of programs in NARS, both in the object-level and the meta-level, does not prevent the system from being creative (since it can handle novel problems) and flexible (since it can adjust its behaviors according to the context). On the other hand, we should not expect an AGI system without any program — even if the meta-level programs (source code) can be modified by the system itself, there will still be meta-meta-level program to regulate the process (Hofstadter, 1979). There have been some attempts to explain intelligence as a group of programs, at various levels and scopes (Minsky, 1985; Baum, 2004; Thórisson and Helgasson, 2012). The lack of creativity and flexibility in conventional computers is not caused by the use of programs, but by the use of programs *directly at the level of problem-solving*.

"*NARS is still a Turing Machine.*" The creativity and flexibility in problem-solving of NARS come from its adaptability and context-sensitivity, rather than from some pure randomness. If the whole-life experience and behavior of the system are taken to be its input and output, respectively, then NARS can still be analyzed as a Turing Machine (TM), and is fully deterministic and predictable (Wang, 1995, 2006). However, this result cannot be used to deny the system's creativity. "Being a TM at *a certain level* of description" does not mean "being a TM at *all levels* of description", since even some activities in an ordinary computer may go beyond the scope of "computing in a TM" (Kugel, 1986). Many claims of "Computer is fundamentally different from the human mind" are based on such misconceptions, where the computer and the mind are described at different levels or scales, and therefore show different properties (Wang, 2007). These claims are invalid, because even the human mind shows the same differences when described at these levels or scales.

"*NARS will be out of control.*" While most people are still worrying about the lack of creativity in computers, there are people worrying about the opposite possibility, that is, some future computers may become so creative and autonomous that their behaviors become completely unpredictable and uncontrollable, which will lead to a so-called "singularity" (Kurzweil, 2006). Though we do need to be careful about the social impacts of AGI research, such a future is not a necessary consequence of the technique proposed here. As explained previously, though NARS can acquire new problem-solving skills, they can be understood in principle, according to the system's design and experience. Due to the complexity of the involved factors, it is not always plausible to accurately control or predict the system's behaviors in practical situations. Again, such an AGI will be like a human being on this aspect. It is not always plausible to accurately control or predict one's behavior, either by an observer or by oneself, but to consider a normal human as "completely unpredictable and uncontrollable" is clearly an exaggeration. Like the other techniques, AGI will present a new challenge to the human beings while serving us. Systems with creativity will probably never be either "fully under control" or "completely out of control", but somewhere in between.

In summary, with respect to its problem-solving power, AGI systems like NARS will have similar capability and limitation as the human mind.

4.3 Conclusion

The problem-solving approach described in this article can be summarized as the following: when facing a problem, if the system has an applicable program, it usually just uses it to solve the problem. Otherwise it reasons about the problem, using its available knowledge and resources, and finds the best solution possible. In the meanwhile, it tries to build programs for future similar problems.

Nivel and Thórisson (2009) argue that an AGI system needs some form of self-programming ability. NARS can "self-program", in the sense that

- It can form compound operations out of simpler operations. Then they will be used just like other operations.
- It can learn the conditions and consequences of each operation, so as to decide when to use it by itself.
- It can perform certain *mental operations* that changes its own reasoning routine.

The first two aspects have been discussed previously, and the last one is introduced in Wang (2013).

Though there are still engineering issues to be resolved, the experiments of NARS provides a proof of concept. At least we can argue that the dominating paradigm in problem-solving, which is based on the theory of computability and computational complexity, is not the only possibility for AI/AGI systems. There are possible approaches that have been ignored, not for technical reasons, but for conceptual reasons.

Compared to program-dependent problem-solving, Case-By-Case (CBC) problem-solving and NARS-style self-programming may provide the system with creativity, adaptability, flexibility, and scalability, at the price of repeatability. It is an approach of problem-solving that is different from the traditional way, but still valid for its target situations.

Since NARS differs from the traditional theories and other AI techniques in the fundamental assumption on the working environment, restrictions, and objectives of the system, technically speaking it is designed for problems not addressed by the other theories and techniques, rather than competing with them. This environment–objective assumption is of special interest to AGI, first because it is closer to the reality of human intelligence, as well as to the situations where we hope AI systems to work. It can be argued that if a system has sufficient knowledge and resources (with respect to the problems it needs to solve), it can just execute the problem-specific algorithms, or do exhaustive search. It needs "intelligence" only when it has no applicable and affordable program for a problem.

Acknowledgment

A preliminary version of this article was presented in the AGI-11 Workshop on Self-Programming. The revision benefits from the discussions with the participants of the workshop, especially Kris Thórisson and Eric Nivel. The author thanks the anonymous reviewers for their helpful comments, and Simon Li for proofreading the manuscript.

References

Albus, J. S. 1991. Outline for a Theory of Intelligence. *IEEE Transactions on Systems, Man, and Cybernetics* 21(3):473–509.

Anderson, J. R. 1983. *The Architecture of Cognition*. Cambridge, Massachusetts: Harvard University Press.

Aristotle. 1882. *The Organon, or, Logical treatises of Aristotle*. London: George Bell. Translated by O. F. Owen.

Arkin, R. C. 1998. *Behavior-Based Robotics*. Cambridge, Massachusetts: MIT Press.

Baum, E. B. 2004. *What is Thought?* Cambridge, Massachusetts: MIT Press.

Boden, M. A. 1991. *The Creative Mind*. New York: BasicBooks.

Bratman, M. E.; Israel, D. J.; and Pollack, M. E. 1988. Plans and resource-bounded practical reasoning. *Computational Intelligence* 4(4):349–355.

Bringsjord, S., and Arkoudas, K. 2004. The modal argument for hypercomputing minds. *Theoretical Computer Science* 317:167–190.

Brooks, R. A. 1991. Intelligence without representation. *Artificial Intelligence* 47:139–159.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2001. *Introduction to Algorithms*. MIT Press, McGraw-Hill Book Company, 2nd edition.

Davis, M. 1958. *Computability and Unsolvability*. New York: Mcgraw-Hill.

Dean, T., and Boddy, M. 1988. An analysis of time-dependent planning. In *Proceedings of AAAI-88*, 49–54.

Dreyfus, H. L. 1979. *What Computers Can't Do: Revised Edition*. New York: Harper and Row.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3-4):189–208.

Flach, P. A., and Kakas, A. C. 2000. Abductive and inductive reasoning: background and issues. In Flach, P. A., and Kakas, A. C., eds., *Abduction and Induction: Essays on their Relation and Integration*. Dordrecht: Kluwer Academic Publishers. 1–27.

Franklin, S. 2007. A foundational architecture for artificial general intelligence. In Goertzel, B., and Wang, P., eds., *Advance of Artificial General Intelligence*. Amsterdam: IOS Press. 36–54.

Frege, G. 1999. Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought. In van Heijenoort, J., ed., *Frege and Gödel: Two Fundamental Texts in Mathematical Logic*. Lincoln, Nebraska: iUniverse. 1–82. Originally published in 1879.

Hayes, P. J. 1977. In defense of logic. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 559–565.

Hofstadter, D. R. 1979. *Gödel, Escher, Bach: an Eternal Golden Braid*. New York: Basic Books.

Hopcroft, J. E., and Ullman, J. D. 1979. *Introduction to Automata Theory, Language, and Computation*. Reading, Massachusetts: Addison-Wesley.

Hutter, M. 2005. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Berlin: Springer.

Jeffrey, R. C. 1965. *The Logic of Decision*. New York: McGraw-Hill.

Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research* 4:237–285.

Kowalski, R. 1979. *Logic for Problem Solving*. New York: North Holland.

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

Kugel, P. 1986. Thinking may be more than computing. *Cognition* 22:137–198.

Kurzweil, R. 2006. *The Singularity Is Near: When Humans Transcend Biology*. New York: Penguin Books.

Laird, J. E.; Newell, A.; and Rosenbloom, P. S. 1987. Soar: an architecture for general intelligence. *Artificial Intelligence* 33:1–64.

Littman, M. L.; Goldsmith, J.; and Mundhenk, M. 1998. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research* 9:1–36.

Lloyd, J. W. 1987. *Foundations of Logic Programming*. New York: Springer-Verlag.

Lucas, J. R. 1961. Minds, machines and Gödel. *Philosophy* XXXVI:112–127.

Marr, D. 1982. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. San Francisco: W. H. Freeman & Co.

McCarthy, J., and Hayes, P. J. 1969. Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., and Michie, D., eds., *Machine Intelligence 4*. Edinburgh: Edinburgh University Press. 463–502.

McCarthy, J. 1988. Mathematical logic in artificial intelligence. *Dædalus* 117(1):297–311.

Michalski, R. S. 1993. Inference theory of learning as a conceptual basis for multistrategy learning. *Machine Learning* 11:111–151.

Minsky, M. 1985. *The Society of Mind*. New York: Simon and Schuster.

Mitchell, T. M. 1997. *Machine Learning*. New York: McGraw-Hill.

Muggleton, S. 1991. Inductive logic programming. *New Generation Computing* 8(4):295–318.

Murphy, R. R. 2000. *An Introduction to AI Robotics*. Cambridge, Massachusetts: MIT Press.

Newell, A., and Simon, H. A. 1963. GPS, a program that simulates human thought. In Feigenbaum, E. A., and Feldman, J., eds., *Computers and Thought*. McGraw-Hill, New York. 279–293.

Newell, A. 1990. *Unified Theories of Cognition*. Cambridge, Massachusetts: Harvard University Press.

Nilsson, N. J. 1991. Logic and artificial intelligence. *Artificial Intelligence* 47:31–56.

Nivel, E., and Thórisson, K. 2009. Self-Programming: Operationalizing Autonomy. In *Proceedings of the Second Conference on Artificial General Intelligence*, 150–155.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems*. San Mateo, California: Morgan Kaufmann Publishers.

Peirce, C. S. 1931. *Collected Papers of Charles Sanders Peirce*, volume 2. Cambridge, Massachusetts: Harvard University Press.

Penrose, R. 1989. *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics*. Oxford University Press.

Piaget, J. 1960. *The Psychology of Intelligence*. Paterson, New Jersey: Littlefield, Adams & Co.

Pollock, J. L. 2006. Against Optimality: The Logical Foundations of Decision-Theoretic Planning. *Computational Intelligence* 22(1):1–25.

Russell, S., and Norvig, P. 2010. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, New Jersey: Prentice Hall, 3rd edition.

Simon, H. A. 1957. *Models of Man: Social and Rational*. New York: John Wiley.

Solomonoff, R. J. 1964. A formal theory of inductive inference. Part I and II. *Information and Control* 7(1-2):1–22,224–254.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts: MIT Press.

Thórisson, K. R., and Helgasson, H. P. 2012. Cognitive Architectures and Autonomy: A Comparative Review. *Journal of Artificial General Intelligence* 3(2):1–30.

Wang, P. 1995. *Non-Axiomatic Reasoning System: Exploring the Essence of Intelligence*. Ph.D. Dissertation, Indiana University.

Wang, P. 2000. The logic of learning. In *Working Notes of the AAAI workshop on New Research Problems for Machine Learning*, 37–40.

Wang, P. 2004a. The limitation of Bayesianism. *Artificial Intelligence* 158(1):97–106.

Wang, P. 2004b. Problem solving with insufficient resources. *International Journal of Uncertainty, Fuzziness and Knowledge-based Systems* 12(5):673–700.

Wang, P. 2005. Experience-grounded semantics: a theory for intelligent systems. *Cognitive Systems Research* 6(4):282–302.

Wang, P. 2006. *Rigid Flexibility: The Logic of Intelligence*. Dordrecht: Springer.

Wang, P. 2007. Three fundamental misconceptions of artificial intelligence. *Journal of Experimental & Theoretical Artificial Intelligence* 19(3):249–268.

Wang, P. 2008. What do you mean by 'AI'. In *Proceedings of the First Conference on Artificial General Intelligence*, 362–373.

Wang, P. 2009a. Case-by-case problem solving. In *Proceedings of the Second Conference on Artificial General Intelligence*, 180–185.

Wang, P. 2009b. Formalization of Evidence: A Comparative Study. *Journal of Artificial General Intelligence* 1:25–53.

Wang, P. 2011. The Assumptions on Knowledge and Resources in Models of Rationality. *International Journal of Machine Consciousness* 3(1):193–218.

Wang, P. 2013. *Non-Axiomatic Logic: A Model of Intelligent Reasoning*. Singapore: World Scientific. (in press).

Xu, Y., and Wang, P. 2012. The frame problem, the relevance problem, and a package solution to both. *Synthese*.