Behavioral Self-Programming by Reasoning

Pei Wang

Temple University, Philadelphia, PA 19122, USA
http://www.cis.temple.edu/~pwang/

Abstract. Behavioral self-programming is the ability for a system to process a task without following a predetermined algorithm. This paper introduces a new approach to do so in a reasoning system that is adaptive to its environment and works with insufficient knowledge and resources. This approach is compared with other existing approaches, and its special nature is discussed.

1 Models of self-programming

From outside, an AI system can be described as interacting with its environment by obtaining a sequence of percept as input, and producing a sequence of action as output [22, 27]. The former is usually referred to as the system's *experience*, and the latter, its *behavior*. When the focus of the discussion is on the system's problem-solving capability, a subsequence of experience can be taken as a "problem", and a subsequence of behavior as its "solution". In this way, the system's life-long history can be seen as a sequence of problem-solving activities, which may overlap with each other in time.

Formally, the system's recognizable percepts can be 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 $a_i \in \mathbf{A}$. A problem, or *stimulus*, 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, or *response*, is a subsequence of B, $R = \langle a_{i_R}, \cdots, a_{j_R} \rangle$ where $0 \leq i_R \leq j_R \leq t$.

In computer science, the above S-R relation is typically specified as a "computation" or "function", that is, a (usually deterministic) mapping from S (the input) to R (the output), or $R = F(S)$. This process is experience-independent, in the sense that the same S will produce the same R, no matter when S appears in E. In AI, however, we are interested in adaptive and creative systems, where the occurrences of the same stimulus may trigger different responses, and usually the latter responses are "better" than the earlier ones, since the system learns from its experience. It is also desired for the system to make a proper response for stimulus anticipated by neither the system itself nor its designer.

In this paper, the above task is called "behavioral self-programming", because the behavior $\langle a_{i_R}, \cdots, a_{j_R} \rangle$ is programmed from actions by *the system itself* at

run-time, rather than by its designer in advance. The notion of *behavioral self-programming* is broad enough to include several existing AI techniques, such as *planning* [1, 5, 8, 20], *production system* [2, 14], *inductive logic programming* [18], *genetic programming* [13], *behavior-based robotics* [3, 6], and *reinforcement learning* [11, 24].[1] These techniques are based on different assumptions about the system and its environment, and consequently, they are different in the following major aspects of the self-programming process:

Knowledge representation: how to represent the system's *knowledge* about percepts and actions, that is, the relations among them with respect to the system's goals.

Knowledge generation: how to generate new knowledge from existing knowledge and new experience, so to avoid explicit programming for each problem by the designer.

Action selection: how to select a behavior among candidates when it is too time-consuming and dangerous to actually test each of them.

In the following, a new approach, "behavioral self-programming by reasoning", is introduced, and compared with the above techniques in these aspects. This approach is used in NARS, a general-purpose intelligent reasoning system [25, 26]. Since NARS has been introduced and discussed in many previous publications (most of which are available at the author's website), this paper only addresses the aspects of the system that are directly related to the behavioral self-programming process.

2 Knowledge representation

NARS uses a *term-oriented* language for knowledge representation [26]. A *term* names a concept within the system, and in its simplest form, a term is just an atomic identifier. For complicated knowledge, a *compound term* can be formed by a logical connector from simpler terms, where the connector comes from a constant set, and the component terms come from the existing terms in the system.

A *statement* is a special type of compound term that specifies a relation from one component to another. 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*. Intuitively, it states that S is a specialization of P, and P is a generalization of S. Many statements on other relations can be equivalently rewritten as inheritance statements. For example, "There is a relation R among A, B, and C" can be represented as "$(\times, A, B, C) \rightarrow R$", where the subject term is a compound, representing a relation among the three terms in the given order, and the statement says that the relation is a special case of R.

[1] Since this "self-programming" happens at the *behavioral* level of the system, it is not necessarily achieved by "modifying the system's own source-code", which would happen at the *implementational* level the system — though that is possible, it is often not the most reliable, feasible, and efficient approach for this task.

Beside the *inheritance copula*, the *similarity copula* "↔" is used for symmetric inheritance relation between two terms, and the *implication copula* "⇒" and the *equivalence copula* "⇔" are used between two statements for "if-then" and "if-and-only-if", respectively. All other relations among terms are translated into these four types of copula.

Given the assumption of insufficient knowledge, in NARS every empirical statement is true to a degree. The *truth-value* of a statement has two factors in it: a *frequency* in [0, 1] indicating the proportion of positive evidence among all available evidence at the current time, and a *confidence* in (0, 1) indicating the proportion of current evidence among all available evidence at a future moment, after a constant amount of evidence comes.

In NARS, episodic knowledge is represented as *events*, which are statements with time-dependent truth-values. The temporal information of an event is represented relatively with respect to another event, so the system can indicate that event E happens *before* event F, or the two happen *at the same time*. An *operation* is a special type of event that the system can make it happen, usually by the running of a software or hardware. Therefore, the "actions" mentioned at the beginning of the paper are all represented in NARS as operations. A *goal* is another special type of event that the system desires to happen. Each goal has a *desire-value* attached (which is a variant of *truth-value* mentioned above) to indicate the extent to which the system wants the statement to become true.

To be 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 mainly used to manage the system's internal activities. The system's direct interaction with its environment is carried out by a set of (optional) plug-in operations, "the tools", that drives the software and hardware of the host system in which NARS is embedded. In this way, NARS can serve as a *mind* within different *bodies*, or an intelligent operating system managing various application programs and devices.

From the existing built-in and plug-in operations, compound operations of various type can be formed. The basic compounding structures include *sequential* and *parallel* execution (using temporal orders among operations), *conditional* execution (using the *implication* copula), and *repetitive* execution (using the *implication* copula in recursion). In this way, a group of statements can serve as a program, by given certain terms a procedural interpretation, as in logic programming [12, 16].

In NARS the preconditions and effects of an (atomic or compound) operation is represented by implication relations between it and other statements, rather than as "states" in a model of the world [1]. Due to insufficient knowledge and resources, the system usually neither knows all the preconditions and effects of an operation, nor can it take all the known ones into consideration when making each decision. Instead, only part of the conditions and/or effects of an operation is considered and even known.

The "reactions" in reactive systems [6] can be represented in NARS as an implication statement "$S \Rightarrow R$", so that whenever the system gets stimulus S,

it takes action R as response. In more complicated situations, the S-R relation is conditional, which can be represented in NARS as "$C \Rightarrow (S \Rightarrow R)$", or equivalently, "$(C \wedge R) \Rightarrow S$", where C is the condition. The same statement can also represent the knowledge that an operation S can has consequence R under condition C. Since in these statements C, S, and R can all be compound terms with internal structures, this type of knowledge can represent complicated plans to reach a complex goal [8]. In this way, what is taken to be opposite approaches in robotics, reactive vs. deliberative [19], are unified in NARS.

Because statements (including events, operations, and goals as special cases) can be related to each other by the copulas (*inheritance*, *similarity*, *implication*, and *equivalence*), in NARS the knowledge about percepts and actions is represented *hierarchically* [1], rather than all reduced to direct relations from percepts to actions (as in many planning and reinforcement learning systems). For example, the plan to reach a certain goal can be represented as consisting of three major steps, then each step is further specified as a subgoal that can be reached by smaller steps, and so on, until the level of executable operations.

Similar ideas can be found in logic programs and production systems, though the language of NARS is less restrictive. In a logic programming language like Prolog, each "rule" must be a Horn clause, where the "body" of the rule implies the "head", which cannot have complicated internal structure. In a production system, a "rule" only specifies the necessary condition for an action to be taken. In NARS, on the contrary, a statement can represent other types of knowledge about an action.

In decision-theoretic planning [4, 21] and reinforcement learning [11, 9], the uncertainty in action consequences is usually represented as probability values. This approach is not taken because in NARS the uncertainty in beliefs cannot be assumed to be a (coherent and stationary) probability distribution [29].

In summary, in NARS declarative, episodic, and procedural knowledge are represented and stored in a uniform format, as logical relations among statements, where the truth-value of the statements is determined by available evidence according to the same semantic theory. The formal grammar of the language can be found in [26], as well as in on-line documents in the author's website.

3 Knowledge generation

In NARS, the simplest 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 when the operation is registered, as an "instinct" of the system. It can also be acquired directly from the system's experience after the operation comes into existence. As a reasoning system, NARS derives most of its beliefs by itself, following a predetermined set of inference rules.

As a *term logic*, a typical inference rule in NARS is *syllogistic*, that takes two premises to derive a conclusion. The two premises contain a common term,

and the conclusion is between the other two terms. When the copula involved is *implication*, there are three basic inference rules:

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

When one term (M in induction, and S in the other two) is taken to mean "whatever the system knows" and is implicitly represented, we get a group of variants of the above rules, which is closer to how these three types of inference are specified in the current AI research:

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

In each rule, t_1 and t_2 are the truth-values of the two premises, respectively. The truth-value of the conclusion is calculated by the corresponding truth-value function from truth-values of the premises. Though the truth-values in NARS intuitively correspond to statistical information about the evidential support of statements, they do not necessarily form a consistent probability distribution. Instead, in each inference step, the truth-value calculation only considers "local information" provided by the premises. This treatment is a direct consequence of the *insufficient knowledge and resources* assumption [29].

The details of the truth-value functions have been discussed in [26] and many other publications. Here we will only mention one property of them: in NARS some inference rules are "strong", and some others are "weak". The confidence of conclusions produced by the former takes 1.0 as upper bound, while that by the latter has a much lower upper bound (0.5, in the current implementation); 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. In the above table, deduction is a strong rule, and the other two are weak. Here their difference is in the *confidence* of the conclusions, rather than their *frequency*.

When applied to procedural knowledge, the above inference rules reveal the condition and consequence of operations. For example, from knowledge "M is a sufficient precondition of action P" ($M \Rightarrow P$) and "S implies M" ($S \Rightarrow M$), the deduction rule generates "S is a sufficient precondition of action P" ($S \Rightarrow P$). From the observation that S is followed by P, the induction rule generalizes the case into "S is a sufficient precondition of action P", though with a low confidence. From knowledge "M is an implication of event S" ($S \Rightarrow M$) and "M is a consequence of action P" ($P \Rightarrow M$), the abduction rule proposes an explanation "S is a sufficient precondition of action P" ($S \Rightarrow P$), also with a low confidence.

When conclusions about the same statement are derived from different evidence, the revision rule in NARS merges the evidence, and produces a more confident conclusion. This is also how a wrong belief gets corrected. Since the confidence of a statement never reaches its upper bound (1.0), all beliefs of the system are revisable, including those that are implanted and input directly. Like in statistical learning, in NARS a stable knowledge is usually not the result of a single inference step, but depends on a large numbers of steps. Though sometimes a certain behavior can be traced back to a single belief or concept, more often it comes from the cooperation of a large number of concepts and beliefs.

Beside the rules mentioned above, there are many other inference rules in NARS, which are described in [26] and other publications. The above rules show the principle followed by all of them: in its conclusion, each rule summarizes the information in the premises, and the conclusion usually contains compound terms not in the premises. When a derived conclusion corresponds to an existing concept or belief, it will be merged into it, so as to contribute to its meaning and/or truth-value, otherwise it will be a novel concept or belief in the whole system. In general, all the expressible compound terms (including the statements) 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, which is also responsible for the generation of all the behaviors of the system. As a reasoning system following a formal logic, every inference step is justified according to the semantics of the system, which requires the conclusion to be based on the evidence provided by the premises. Therefore, unlike genetic programming [13], in NARS there is no pure-random factor in the generation of new behaviors. 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".

NARS is tolerant of the uncertainty, incompleteness, and inconsistence in knowledge. With the coming of new experience and the changing of the context, the system adjusts its concepts and beliefs, which can be considered as various types of learning. Similarly, various types of ways for the behaviors to be organized, such as reaction, conditioning, clustering, abstracting, planning, etc., are all covered as special cases of the same reasoning/learning process, rather than as following separate "learning algorithms" [17].

4 Action selection

At every moment, the actual behavior of NARS is produced by a selected subset of its procedural knowledge (on the preconditions, effects, resemblance, composition, etc., of operations), and the selection is usually a combination of reactive responses and goal-directed deliberations and decisions.

As mentioned previously, 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 long-term goals of the system.

As mentioned previously, in NARS a "goal" is an event that the system desires to happen, which is represented as a *statement* (a *partial* description of the environment), rather than a *state* (a *complete* description of the environment). All the initial goals in NARS come from its designer or tutor, and the system produces derived goals from them according to the system's beliefs.

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, S becomes a candidate goal (since its realization provides a means to realize P), just like in many AI systems. However, what makes NARS different here is that 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 competitive with each other for the system's resources.

To deal with this situation, in NARS each goal has a "desire-value" attached to indicate the system's preference, relative to other goals. Desire-value is defined as a variation of truth-value. The desire-values of the initial goals are assigned by the designer or tutor, and that of the derived goals are determined by the truth-value functions when they are derived. When a goal gets different desire-values from different sources, they are merged together by the revision rule, just like the truth-values. For the above example, if S is recognized as leading to a highly undesired consequence Q, it will not be taken as a way to achieve P.

When NARS makes decisions on what goals to actually pursue, it considers both the *desirability* and the *plausibility* of each alternative, as in traditional decision theory [10]. 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 certain. Instead, the desirability and the plausibility of each alternative need to be evaluated by the system itself through reasoning, according to its experience and restricted by its available resources. Since the goals constantly change, NARS is not guided by a fixed fitness function, utility measurement, or reward signal. Instead, the evaluation of desirability and plausibility is context dependent, and the result does not necessarily converge to a limit.

Beside desirability and plausibility, another factor in the selection process is the simplicity of the candidates. Due to insufficient resources, NARS prefers simple concepts and beliefs. The importance of an "Occum's Razor" in learning is widely acknowledged, 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 [23]. In NARS, the desirability, plausibility, and simplicity of an event are evaluated separately,

though the final selection among alternative actions is usually a trade-off among the three and some other factors.

As a reasoning system, the learning process in NARS is very different from that governed by various "learning algorithms", in terms of its resource consumption. Since each inference step only considers the evidence in the premises, it can be completed in a short constant time. A learning or problem-solving process is usually carried out by a sequence of inference steps, but the selections of the steps and the termination of the sequence are decisions made by the system in run time, according to many factors.

For example, when the system looks for a path to achieve a goal, it will not go through all known candidates, then pick the best one according to certain criteria — since the goal may need to be achieved via derived goals, such an exhaustive search usually leads to a combinatorial explosion. Instead, the system selects candidates probabilistically, with the chance of each candidate determined by its current priority value, which summarizes factors like quality (simplicity, confidence, etc.), usefulness (past performance), relevance (to the most recent experience), and so on.

Consequently, though there is no pure-random factor, the behavior of NARS is still *nondeterministic* and *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. This "case by case" working mode [28] can handle novel cases for which previous experience is not directly applicable, as well as to carry out one-shot learning.

Therefore, how much time a problem-solving process takes is not a constant, but varies from case to case, like in an anytime algorithm [7]. The response of NARS to the combinatorial explosion problem is not to find an algorithm with a polynomial growth order [15], but to get rid of the notion of "problem-specific algorithm" from the start. For each problem (instance), the system simply deals with it with the knowledge that happens to be recalled at the moment, and explores as many possibilities as the current resource supply allows, rather than continues until the solution satisfies a predetermined standard.

Working in this way, NARS no longer has the predictability and repeatability of the conventional computational systems, but at the same time, it gains the autonomy, flexibility, and adaptability we expected from intelligent systems. Like it or not, this type of behavior is an inevitable consequence of "adaptation with insufficient knowledge and resources", the working definition of intelligence on which NARS is based.

5 Conclusion

Under the length restriction, this paper cannot describe the technical details of the behavioral self-programming in NARS. Even so, the above conceptual-level description should be enough to show that this approach is different from the others in its major features. The basic idea is to organize operations in an

inheritance hierarchy (using *inheritance* ans *similarity* relations), and to specify each operation with (partial) conditions and consequences (using *implication* ans *equivalence* relations). When representing these relations, various types of uncertainty (randomness, fuzziness, ignorance, inconsistency, etc.) are allowed. The system uses the available knowledge to derive new knowledge, as well as to select responses to the current and recent stimulus. Since the behaviors of the system is experience-dependent and context-sensitive, the system "programs itself", in the sense that for a given problem, the system's solution is not fully determined by a given algorithm.

Though NARS is similar to the other behavioral self-programming approaches here or there, as a whole it is not based on any of them. The differences mainly come from the system's commitment to *adaptation with insufficient knowledge and resources*, which is not assumed in exactly the same sense by any of the other approaches. Since NARS differs from the other theories in the assumptions on the working environment, restrictions, and objectives of the system, technically speaking it is designed for a problem not addressed by the others.

This environment-objective assumption is of special interest to AI, 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 algorithm to follow in the problem-solving process.[2]

For AGI research, the NARS approach has special significance, because of its general-purpose nature. The "hands plus tools" model introduced previously 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.

References

1. Albus, J.S.: Outline for a theory of intelligence. IEEE Transactions on Systems, Man, and Cybernetics 21(3), 473–509 (1991)
2. Anderson, J.R.: The Architecture of Cognition. Harvard University Press, Cambridge, Massachusetts (1983)
3. Arkin, R.C.: Behavior-Based Robotics. MIT Press, Cambridge, Massachusetts (1998)

[2] Of course, NARS still follows algorithms in its internal activities, and can be seen as a Turing Machine in its whole life cycle, but that is at a different level of description. See [26] for a detailed discussion on this topic.

4. Blythe, J.: Decision-theoretic planning. AI Magazine 20(2), 37–54 (1999)
5. Bratman, M.E., Israel, D.J., Pollack, M.E.: Plans and resource-bounded practical reasoning. Computational Intelligence 4(4), 349–355 (1988)
6. Brooks, R.A.: Intelligence without representation. Artificial Intelligence 47, 139–159 (1991)
7. Dean, T., Boddy, M.: An analysis of time-dependent planning. In: Proceedings of AAAI-88. pp. 49–54 (1988)
8. Fikes, R.E., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2(3-4), 189–208 (1971)
9. Hutter, M.: Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability. Springer, Berlin (2005)
10. Jeffrey, R.C.: The Logic of Decision. McGraw-Hill, New York (1965)
11. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: a survey. Journal of Artificial Intelligence Research 4, 237–285 (1996)
12. Kowalski, R.: Logic for Problem Solving. North Holland, New York (1979)
13. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, Massachusetts (1992)
14. Laird, J.E., Newell, A., Rosenbloom, P.S.: Soar: an architecture for general intelligence. Artificial Intelligence 33, 1–64 (1987)
15. Littman, M.L., Goldsmith, J., Mundhenk, M.: The computational complexity of probabilistic planning. Journal of Artificial Intelligence Research 9, 1–36 (1998)
16. Lloyd, J.W.: Foundations of Logic Programming. Springer-Verlag, New York (1987)
17. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
18. Muggleton, S.: Inductive logic programming. New Generation Computing 8(4), 295–318 (1991)
19. Murphy, R.R.: An Introduction to AI Robotics. MIT Press, Cambridge, Massachusetts (2000)
20. Newell, A., Simon, H.A.: GPS, a program that simulates human thought. In: Feigenbaum, E.A., Feldman, J. (eds.) Computers and Thought, pp. 279–293. McGraw-Hill, New York (1963)
21. Pollock, J.L.: Against optimality: The logical foundations of decision-theoretic planning. Computational Intelligence 22(1), 1–25 (2006)
22. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, Upper Saddle River, New Jersey, 2nd edn. (2002)
23. Solomonoff, R.J.: A formal theory of inductive inference. Part I and II. Information and Control 7(1-2), 1–22,224–254 (1964)
24. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, Massachusetts (1998)
25. Wang, P.: Non-Axiomatic Reasoning System: Exploring the Essence of Intelligence. Ph.D. thesis, Indiana University (1995)
26. Wang, P.: Rigid Flexibility: The Logic of Intelligence. Springer, Dordrecht (2006)
27. Wang, P.: What do you mean by 'AI'. In: Proceedings of the First Conference on Artificial General Intelligence. pp. 362–373 (2008)
28. Wang, P.: Case-by-case problem solving. In: Proceedings of the Second Conference on Artificial General Intelligence. pp. 180–185 (2009)
29. Wang, P.: Formalization of evidence: A comparative study. Journal of Artificial General Intelligence 1, 25–53 (2009)