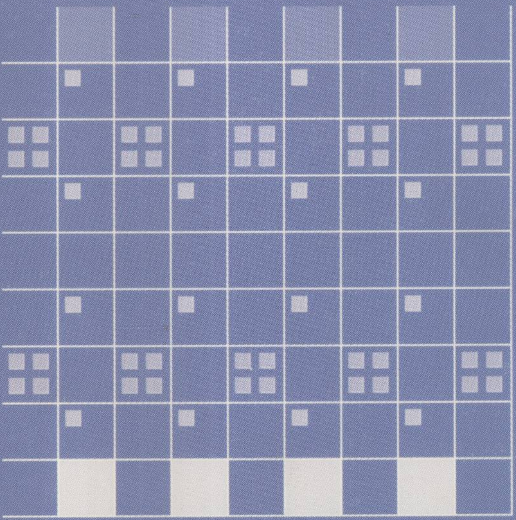
Daniel Kudenko
Dimitar Kazakov
Eduardo Alonso (Eds.)

Hot Topics

LNAI 3394

Adaptive Agents and Multi-Agent Systems II

Adaptation and Multi-Agent Learning



Springer

Daniel Kudenko Dimitar Kazakov
Eduardo Alonso (Eds.)

Adaptive Agents and Multi-Agent Systems II

Adaptation and Multi-Agent Learning

Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editors

Daniel Kudenko
Dimitar Kazakov
University of York
Department of Computer Science
Heslington, York, YO10 5DD, UK
E-mail: {kudenko, kazakov}@cs.york.ac.uk

Eduardo Alonso
City University London
Department of Computing
London, EC1V OHB, UK
E-mail: eduardo@soi.city.ac.uk

Library of Congress Control Number: 2005921645

CR Subject Classification (1998): I.2.11, I.2, D.2, C.2.4, F.3.1, D.3.1, H.5.3, K.4.3

ISSN 0302-9743
ISBN 3-540-25260-6 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springeronline.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11404071 06/3142 5 4 3 2 1 0

Lecture Notes in Artificial Intelligence 3394

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

Preface

Predictions are a delicate matter. The I-told-you-this-was-going-to-happen ones are reliable, but not very helpful, as they only achieve credibility post factum. Similarly uninteresting are those of the shrowded-in-mystery, match-it-all type. Finally, when a respected person has both a vision and courage to state it, the future could prove him right, yet realize his dream with an unexpected twist. A solitary multimillionaire's round trip to an ageing orbital station is far from the crowds of space tourists predicted by A.C. Clark. However, when he said there would be hotels in space by 2001, he was spot on, despite the modest beginning.

We also met the year 2001 magical milestone to the future without being surrounded by either Arthur C. Clark's intelligent computers or their moody cousins of Douglas Adams's cut. However, one of the many small steps in this direction was made when the 1st Symposium on Adaptive Agents and Multi-agent Systems (AAMAS) was organized in that year. In front of you is a collection of selected papers from the 3rd and 4th AAMAS symposia, which persisted in the goals set in 2001, namely, to increase awareness and interest in adaptive agent research, encourage collaboration between machine learning and agent system experts, and give a representative overview of current research in the area of adaptive agents.

Recent years have seen an increasing interest, and the beginning of consolidation of the European research community in the field. Still, there are many major challenges left to tackle. While our understanding of learning agents and multi-agent systems has advanced significantly, most applications are still on simple scaled-down domains, and, in fact, most methods do not scale up to the real world. This, amongst others, is a major obstacle to bring learning agent technologies to commercial applications. Stay tuned for new developments in the – hopefully near – future.

The first book on the subject (Springer LNAI, vol. 2636), largely based on contributions to AAMAS and AAMAS-2, was published in 2002. It is with delight that we present another volume of articles in this emerging multidisciplinary area encompassing computer science, software engineering, biology, as well as the cognitive and social sciences.

Our thanks go to the symposium keynote speakers, Jürgen Schmidhuber and Sorin Solomon, for writing invited papers for this volume, the members of the symposium Program Committee for fast and thorough reviews, AgentLink II & III Networks of Excellence for co-sponsoring the symposium, the Society for the Study of Artificial Intelligence and the Simulation of Behaviour (SSAISB) for providing outstanding help in the organization of this event, and, of course, special thanks to the authors without whose high-quality contributions there would not be a book to begin with.

December 2004 Daniel Kudenko, Dimitar Kazakov, Eduardo Alonso

Organization

Symopsium Chairs

Eduardo Alonso (City University London, UK)
Dimitar Kazakov (University of York, UK)
Daniel Kudenko (University of York, UK)

Program Committee

Frances Brazier, Free University, Amsterdam, Netherlands
Philippe De Wilde, Imperial College, London, UK
Kurt Driessens, Catholic University of Leuven, Belgium
Saso Dzeroski, Jozef Stefan Institute, Ljubljana, Slovenia
Zahia Guessoum, LIP 6, Paris, France
Tom Holvoet, Computer Science Department, Catholic University of Leuven, Belgium
Paul Marrow, BTexact Technologies, UK
Ann Nowé, Free University of Brussels, Belgium
Luis Nunes, University of Porto, Portugal
Eugenio Oliveira, University of Porto, UK
Paolo Petta, Austrian Research Centre for AI, Austria
Enric Plaza, IIIA-CSIC, Spain
Michael Schroeder, City University, London, UK
Kostas Stathis, City University, London, UK
Malcolm Strens, QinetiQ, UK
Marco Wiering, University of Utrecht, Netherlands
Niek Wijngaards, Free University, Amsterdam, Netherlands

Sponsoring Institutions

SSAISB, UK
AgentLink II

Lecture Notes in Artificial Intelligence (LNAI)

Vol. 3452: F. Baader, A. Voronkov (Eds.), Logic Programming, Artificial Intelligence, and Reasoning. XI, 562 pages. 2005.

Vol. 3416: M. Böhlen, J. Gamper, W. Polasek, M.A. Wimmer (Eds.), E-Government: Towards Electronic Democracy. XIII, 311 pages. 2005.

Vol. 3415: P. Davidsson, B. Logan, K. Takadama (Eds.), Multi-Agent and Multi-Agent-Based Simulation. X, 265 pages. 2005.

Vol. 3403: B. Ganter, R. Godin (Eds.), Formal Concept Analysis. XI, 419 pages. 2005.

Vol. 3398: D.-K. Baik (Ed.), Systems Modeling and Simulation: Theory and Applications. XIV, 733 pages. 2005.

Vol. 3397: T.G. Kim (Ed.), Artificial Intelligence and Simulation. XV, 711 pages. 2005.

Vol. 3396: R.M. van Eijk, M.-P. Huget, F. Dignum (Eds.), Agent Communication. X, 261 pages. 2005.

Vol. 3394: D. Kudenko, D. Kazakov, E. Alonso (Eds.), Adaptive Agents and Multi-Agent Systems III. VIII, 313 pages. 2005.

Vol. 3374: D. Weyns, H.V.D. Parunak, F. Michel (Eds.), Environments for Multi-Agent Systems. X, 279 pages. 2005.

Vol. 3369: V.R. Benjamins, P. Casanovas, J. Breuker, A. Gangemi (Eds.), Law and the Semantic Web. XII, 249 pages. 2005.

Vol. 3366: I. Rahwan, P. Moraitis, C. Reed (Eds.), Argumentation in Multi-Agent Systems. XII, 263 pages. 2005.

Vol. 3359: G. Grieser, Y. Tanaka (Eds.), Intuitive Human Interfaces for Organizing and Accessing Intellectual Assets. XIV, 257 pages. 2005.

Vol. 3346: R.H. Bordini, M. Dastani, J. Dix, A.E.F. Seghrouchni (Eds.), Programming Multi-Agent Systems. XIV, 249 pages. 2005.

Vol. 3345: Y. Cai (Ed.), Ambient Intelligence for Scientific Discovery. XII, 311 pages. 2005.

Vol. 3343: C. Freksa, M. Knauff, B. Krieg-Brückner, B. Nebel, T. Barkowsky (Eds.), Spatial Cognition IV. Reasoning, Action, and Interaction. XIII, 519 pages. 2005.

Vol. 3339: G.I. Webb, X. Yu (Eds.), AI 2004: Advances in Artificial Intelligence. XXII, 1272 pages. 2004.

Vol. 3336: D. Karagiannis, U. Reimer (Eds.), Practical Aspects of Knowledge Management. X, 523 pages. 2004.

Vol. 3327: Y. Shi, W. Xu, Z. Chen (Eds.), Data Mining and Knowledge Management. XIII, 263 pages. 2005.

Vol. 3315: C. Lemaître, C.A. Reyes, J.A. González (Eds.), Advances in Artificial Intelligence – IBERAMIA 2004. XX, 987 pages. 2004.

Vol. 3303: J.A. López, E. Benfenati, W. Dubitzky (Eds.), Knowledge Exploration in Life Science Informatics. X, 249 pages. 2004.

Vol. 3276: D. Nardi, M. Riedmiller, C. Sammut, J. Santos-Victor (Eds.), RoboCup 2004: Robot Soccer World Cup VIII. XVIII, 678 pages. 2005.

Vol. 3275: P. Perner (Ed.), Advances in Data Mining. VIII, 173 pages. 2004.

Vol. 3265: R.E. Frederking, K.B. Taylor (Eds.), Machine Translation: From Real Users to Research. XI, 392 pages. 2004.

Vol. 3264: G. Paliouras, Y. Sakakibara (Eds.), Grammatical Inference: Algorithms and Applications. XI, 291 pages. 2004.

Vol. 3259: J. Dix, J. Leite (Eds.), Computational Logic in Multi-Agent Systems. XII, 251 pages. 2004.

Vol. 3257: E. Motta, N.R. Shadbolt, A. Stutt, N. Gibbins (Eds.), Engineering Knowledge in the Age of the Semantic Web. XVII, 517 pages. 2004.

Vol. 3249: B. Buchberger, J.A. Campbell (Eds.), Artificial Intelligence and Symbolic Computation. X, 285 pages. 2004.

Vol. 3248: K.-Y. Su, J. Tsujii, J.-H. Lee, O.Y. Kwong (Eds.), Natural Language Processing – IJCNLP 2004. XVIII, 817 pages. 2005.

Vol. 3245: E. Suzuki, S. Arikawa (Eds.), Discovery Science. XIV, 430 pages. 2004.

Vol. 3244: S. Ben-David, J. Case, A. Maruoka (Eds.), Algorithmic Learning Theory. XIV, 505 pages. 2004.

Vol. 3238: S. Biundo, T. Frühwirth, G. Palm (Eds.), KI 2004: Advances in Artificial Intelligence. XI, 467 pages. 2004.

Vol. 3230: J.L. Vicedo, P. Martínez-Barco, R. Muñoz, M. Saiz Noeda (Eds.), Advances in Natural Language Processing. XII, 488 pages. 2004.

Vol. 3229: J.J. Alferes, J. Leite (Eds.), Logics in Artificial Intelligence. XIV, 744 pages. 2004.

Vol. 3228: M.G. Hinchey, J.L. Rash, W.F. Truszkowski, C.A. Rouff (Eds.), Formal Approaches to Agent-Based Systems. VIII, 290 pages. 2004.

Vol. 3215: M.G.. Negoita, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part III. LVII, 906 pages. 2004.

Vol. 3214: M.G.. Negoita, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part II. LVIII, 1302 pages. 2004.

Vol. 3213: M.G.. Negoita, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part I. LVIII, 1280 pages. 2004.

Vol. 3209: B. Berendt, A. Hotho, D. Mladenic, M. van Someren, M. Spiliopoulou, G. Stumme (Eds.), Web Mining: From Web to Semantic Web. IX, 201 pages. 2004.

Vol. 3206: P. Sojka, I. Kopecek, K. Pala (Eds.), Text, Speech and Dialogue. XIII, 667 pages. 2004.

Vol. 3202: J.-F. Boulicaut, F. Esposito, F. Giannotti, D. Pedreschi (Eds.), Knowledge Discovery in Databases: PKDD 2004. XIX, 560 pages. 2004.

Vol. 3201: J.-F. Boulicaut, F. Esposito, F. Giannotti, D. Pedreschi (Eds.), Machine Learning: ECML 2004. XVIII, 580 pages. 2004.

Vol. 3194: R. Camacho, R. King, A. Srinivasan (Eds.), Inductive Logic Programming. XI, 361 pages. 2004.

Vol. 3192: C. Bussler, D. Fensel (Eds.), Artificial Intelligence: Methodology, Systems, and Applications. XIII, 522 pages. 2004.

Vol. 3191: M. Klusch, S. Ossowski, V. Kashyap, R. Unland (Eds.), Cooperative Information Agents VIII. XI, 303 pages. 2004.

Vol. 3187: G. Lindemann, J. Denzinger, I.J. Timm, R. Unland (Eds.), Multiagent System Technologies. XIII, 341 pages. 2004.

Vol. 3176: O. Bousquet, U. von Luxburg, G. Rätsch (Eds.), Advanced Lectures on Machine Learning. IX, 241 pages. 2004.

Vol. 3171: A.L.C. Bazzan, S. Labidi (Eds.), Advances in Artificial Intelligence – SBIA 2004. XVII, 548 pages. 2004.

Vol. 3159: U. Visser, Intelligent Information Integration for the Semantic Web. XIV, 150 pages. 2004.

Vol. 3157: C. Zhang, H. W. Guesgen, W.K. Yeap (Eds.), PRICAI 2004: Trends in Artificial Intelligence. XX, 1023 pages. 2004.

Vol. 3155: P. Funk, P.A. González Calero (Eds.), Advances in Case-Based Reasoning. XIII, 822 pages. 2004.

Vol. 3139: F. Iida, R. Pfeifer, L. Steels, Y. Kuniyoshi (Eds.), Embodied Artificial Intelligence. IX, 331 pages. 2004.

Vol. 3131: V. Torra, Y. Narukawa (Eds.), Modeling Decisions for Artificial Intelligence. XI, 327 pages. 2004.

Vol. 3127: K.E. Wolff, H.D. Pfeiffer, H.S. Delugach (Eds.), Conceptual Structures at Work. XI, 403 pages. 2004.

Vol. 3123: A. Belz, R. Evans, P. Piwek (Eds.), Natural Language Generation. X, 219 pages. 2004.

Vol. 3120: J. Shawe-Taylor, Y. Singer (Eds.), Learning Theory. X, 648 pages. 2004.

Vol. 3097: D. Basin, M. Rusinowitch (Eds.), Automated Reasoning. XII, 493 pages. 2004.

Vol. 3071: A. Omicini, P. Petta, J. Pitt (Eds.), Engineering Societies in the Agents World. XIII, 409 pages. 2004.

Vol. 3070: L. Rutkowski, J. Siekmann, R. Tadeusiewicz, L.A. Zadeh (Eds.), Artificial Intelligence and Soft Computing - ICAISC 2004. XXV, 1208 pages. 2004.

Vol. 3068: E. André, L. Dybkjær, W. Minker, P. Heisterkamp (Eds.), Affective Dialogue Systems. XII, 324 pages. 2004.

Vol. 3067: M. Dastani, J. Dix, A. El Fallah-Seghrouchni (Eds.), Programming Multi-Agent Systems. X, 221 pages. 2004.

Vol. 3066: S. Tsumoto, R. Słowiński, J. Komorowski, J.W. Grzymała-Busse (Eds.), Rough Sets and Current Trends in Computing. XX, 853 pages. 2004.

Vol. 3065: A. Lomuscio, D. Nute (Eds.), Deontic Logic in Computer Science. X, 275 pages. 2004.

Vol. 3060: A.Y. Tawfik, S.D. Goodwin (Eds.), Advances in Artificial Intelligence. XIII, 582 pages. 2004.

Vol. 3056: H. Dai, R. Srikant, C. Zhang (Eds.), Advances in Knowledge Discovery and Data Mining. XIX, 713 pages. 2004.

Vol. 3055: H. Christiansen, M.-S. Hacid, T. Andreasen, H.L. Larsen (Eds.), Flexible Query Answering Systems. X, 500 pages. 2004.

Vol. 3048: P. Faratin, D.C. Parkes, J.A. Rodríguez-Aguilar, W.E. Walsh (Eds.), Agent-Mediated Electronic Commerce V. XI, 155 pages. 2004.

Vol. 3040: R. Conejo, M. Urretavizcaya, J.-L. Pérez-de-la-Cruz (Eds.), Current Topics in Artificial Intelligence. XIV, 689 pages. 2004.

Vol. 3035: M.A. Wimmer (Ed.), Knowledge Management in Electronic Government. XII, 326 pages. 2004.

Vol. 3034: J. Favela, E. Menasalvas, E. Chávez (Eds.), Advances in Web Intelligence. XIII, 227 pages. 2004.

Vol. 3030: P. Giorgini, B. Henderson-Sellers, M. Winikoff (Eds.), Agent-Oriented Information Systems. XIV, 207 pages. 2004.

Vol. 3029: B. Orchard, C. Yang, M. Ali (Eds.), Innovations in Applied Artificial Intelligence. XXI, 1272 pages. 2004.

Vol. 3025: G.A. Vouros, T. Panayiotopoulos (Eds.), Methods and Applications of Artificial Intelligence. XV, 546 pages. 2004.

Vol. 3020: D. Polani, B. Browning, A. Bonarini, K. Yoshida (Eds.), RoboCup 2003: Robot Soccer World Cup VII. XVI, 767 pages. 2004.

Vol. 3012: K. Kurumatani, S.-H. Chen, A. Ohuchi (Eds.), Multi-Agents for Mass User Support. X, 217 pages. 2004.

Vol. 3010: K.R. Apt, F. Fages, F. Rossi, P. Szeredi, J. Váncza (Eds.), Recent Advances in Constraints. VIII, 285 pages. 2004.

Vol. 2990: J. Leite, A. Omicini, L. Sterling, P. Torroni (Eds.), Declarative Agent Languages and Technologies. XII, 281 pages. 2004.

Vol. 2980: A. Blackwell, K. Marriott, A. Shimojima (Eds.), Diagrammatic Representation and Inference. XV, 448 pages. 2004.

Vol. 2977: G. Di Marzo Serugendo, A. Karageorgos, O.F. Rana, F. Zambonelli (Eds.), Engineering Self-Organising Systems. X, 299 pages. 2004.

Vol. 2972: R. Monroy, G. Arroyo-Figueroa, L.E. Sucar, H. Sossa (Eds.), MICAI 2004: Advances in Artificial Intelligence. XVII, 923 pages. 2004.

Vol. 2969: M. Nickles, M. Rovatsos, G. Weiss (Eds.), Agents and Computational Autonomy. X, 275 pages. 2004.

Vol. 2961: P. Eklund (Ed.), Concept Lattices. IX, 411 pages. 2004.

Vol. 2953: K. Konrad, Model Generation for Natural Language Interpretation and Analysis. XIII, 166 pages. 2004.

Table of Contents

Gödel Machines: Towards a Technical Justification of Consciousness

Jürgen Schmidhuber

IDSIA, Galleria 2, 6928 Manno (Lugano), Switzerland
TU Munich, Boltzmannstr. 3, 85748 Garching, München, Germany
juergen@idsia.ch
http://www.idsia.ch/~juergen

Abstract. The growing literature on consciousness does not provide a formal demonstration of the *usefulness* of consciousness. Here we point out that the recently formulated Gödel machines may provide just such a technical justification. They are the first mathematically rigorous, general, fully self-referential, self-improving, optimally efficient problem solvers, "conscious" or "self-aware" in the sense that their entire behavior is open to introspection, and modifiable. A Gödel machine is a computer that rewrites any part of its own initial code as soon as it finds a proof that the rewrite is *useful,* where the problem-dependent *utility function*, the hardware, and the entire initial code are described by axioms encoded in an initial asymptotically optimal proof searcher which is also part of the initial code. This type of total self-reference is precisely the reason for the Gödel machine's optimality as a general problem solver: any self-rewrite is globally optimal—no local maxima!—since the code first had to prove that it is not useful to continue the proof search for alternative self-rewrites.

1 Introduction and Outline

In recent years the topic of consciousness has gained some credibility as a serious research issue, at least in philosophy and neuroscience, e.g., [8]. However, there is a lack of *technical* justifications of consciousness: so far no one has shown that consciousness is really useful for solving problems, even though problem solving is considered of central importance in philosophy [29].

Our fully self-referential Gödel machine [43, 45] may be viewed as providing just such a technical justification. It is "self-aware" or "conscious" in the sense that the algorithm determining its behavior is completely open to self-inspection, and modifiable in a very general (but computable) way. It can 'step outside of itself' [13] by executing self-changes that are provably good, where the mechanism for generating the proofs also is part of the initial code and thus subject to analysis and change. We will see that this type of total self-reference makes the Gödel machine an *optimal* general problem solver, in the sense of Global Optimality Theorem 1, to be discussed in Section 4.

Outline. Section 2 presents basic concepts of Gödel machines, relations to the most relevant previous work, and limitations. Section 3 presents the essential details of a self-referential axiomatic system of one particular Gödel machine, Section 4 the Global

D. Kudenko et al. (Eds.): Adaptive Agents and MAS II, LNAI 3394, pp. 1–23, 2005.

Optimality Theorem 1, and Section 5 an $O()$-optimal (Theorem 2) initial proof searcher. Section 6 provides examples and additional relations to previous work, and lists answers to several frequently asked questions about Gödel machines. Section 7 wraps up.

2 Basic Overview / Most Relevant Previous Work / Limitations

All traditional algorithms for problem solving are hardwired. Some are designed to improve some limited type of policy through experience [19], but are not part of the modifiable policy, and cannot improve themselves in a theoretically sound way. Humans are needed to create new / better problem solving algorithms and to prove their usefulness under appropriate assumptions.

Here we eliminate the restrictive need for human effort in the most general way possible, leaving all the work including the proof search to a system that can rewrite and improve itself in arbitrary computable ways and in a most efficient fashion. To attack this *"Grand Problem of Artificial Intelligence,"* we introduce a novel class of optimal, fully self-referential [10] general problem solvers called *Gödel machines* [43, 44].[1] They are universal problem solving systems that interact with some (partially observable) environment and can in principle modify themselves without essential limits apart from the limits of computability. Their initial algorithm is not hardwired; it can completely rewrite itself, but only if a proof searcher embedded within the initial algorithm can first prove that the rewrite is useful, given a formalized utility function reflecting computation time and expected future success (e.g., rewards). We will see that self-rewrites due to this approach are actually *globally optimal* (Theorem 1, Section 4), relative to Gödel's well-known fundamental restrictions of provability [10]. These restrictions should not worry us; if there is no proof of some self-rewrite's utility, then humans cannot do much either.

The initial proof searcher is $O()$-optimal (has an optimal order of complexity) in the sense of Theorem 2, Section 5. Unlike hardwired systems such as Hutter's [15, 16] (Section 2) and Levin's [23, 24], however, a Gödel machine can in principle speed up any part of its initial software, including its proof searcher, to meet *arbitrary* formalizable notions of optimality beyond those expressible in the $O()$-notation. Our approach yields the first theoretically sound, fully self-referential, optimal, general problem solvers.

2.1 Set-Up and Formal Goal

Many traditional problems of computer science require just one problem-defining input at the beginning of the problem solving process. For example, the initial input may be a large integer, and the goal may be to factorize it. In what follows, however, we will also consider the *more general case* where the problem solution requires interaction with a dynamic, initially unknown environment that produces a continual stream of inputs and feedback signals, such as in autonomous robot control tasks, where the goal may be

[1] Or *'Goedel machine'*, to avoid the *Umlaut*. But *'Godel machine'* would not be quite correct. Not to be confused with what Penrose calls, in a different context, *'Gödel's putative theorem-proving machine'* [28]!

to maximize expected cumulative future reward [19]. This may require the solution of essentially arbitrary problems (examples in Section 6.1 formulate traditional problems as special cases).

Our hardware (e.g., a universal or space-bounded Turing machine [55] or the abstract model of a personal computer) has a single life which consists of discrete cycles or time steps $t = 1, 2, \ldots$. Its total lifetime T may or may not be known in advance. In what follows, the value of any time-varying variable Q at time t will be denoted by $Q(t)$.

During each cycle our hardware executes an elementary operation which affects its variable state $s \in \mathcal{S} \subset \mathcal{B}^*$ (where B^* is the set of possible bitstrings over the binary alphabet $B = \{0, 1\}$) and possibly also the variable environmental state $Env \in \mathcal{E}$ (here we need not yet specify the problem-dependent set $\mathcal{E}$). There is a hardwired state transition function $F : \mathcal{S} \times \mathcal{E} \rightarrow \mathcal{S}$. For $t > 1$, $s(t) = F(s(t-1), Env(t-1))$ is the state at a point where the hardware operation of cycle $t-1$ is finished, but the one of t has not started yet. $Env(t)$ may depend on past output actions encoded in $s(t-1)$ and is simultaneously updated or (probabilistically) computed by the possibly reactive environment.

In order to talk conveniently about programs and data, we will often attach names to certain string variables encoded as components or substrings of s. Of particular interest are the three variables called *time*, *x*, *y*, and *p*:

1. At time t, variable $time$ holds a unique binary representation of t. We initialize $time(1) =$ '1', the bitstring consisting only of a one. The hardware increments $time$ from one cycle to the next. This requires at most $O(log\ t)$ and on average only $O(1)$ computational steps.
2. Variable *x* holds the inputs form the environment to the Gödel machine. For $t > 1$, $x(t)$ may differ from $x(t-1)$ only if a program running on the Gödel machine has executed a special input-requesting instruction at time $t-1$. Generally speaking, the delays between successive inputs should be sufficiently large so that programs can perform certain elementary computations on an input, such as copying it into internal storage (a reserved part of s) before the next input arrives.
3. Variable *y* holds the outputs of the Gödel machine. $y(t)$ is the output bitstring which may subsequently influence the environment, where $y(1) =$ '0' by default. For example, $y(t)$ could be interpreted as a control signal for an environment-manipulating robot whose actions may have an effect on future inputs.
4. $p(1)$ is the initial software: a program implementing the original (sub-optimal) policy for interacting with the environment, represented as a substring $e(1)$ of $p(1)$, plus the original policy for searching proofs. Details will be discussed below.

At any given time t ($1 \leq t \leq T$) the goal is to maximize future success or *utility*. A typical *"value to go"* utility function is of the form $u(s, Env) : \mathcal{S} \times \mathcal{E} \rightarrow \mathcal{R}$, where $\mathcal{R}$ is the set of real numbers:

$$u(s, Env) = E_\mu \left[\sum_{\tau=time}^{E_\mu(T|s,Env)} r(\tau) \;\middle|\; s, Env \right], \tag{1}$$

where $r(t)$ is a real-valued reward input (encoded within $s(t)$) at time t, $E_\mu(\cdot \mid \cdot)$ denotes the conditional expectation operator with respect to some possibly unknown distribution

μ from a set M of possible distributions (M reflects whatever is known about the possibly probabilistic reactions of the environment), and the above-mentioned $time = time(s)$ is a function of state s which uniquely identifies the current cycle. Note that we take into account the possibility of extending the expected lifespan $E_\mu(T \mid s, Env)$ through appropriate actions.

Alternative formalizable utility functions could favor improvement of *worst case* instead of *expected* future performance, or higher reward intake *per time interval* etc. Clearly, most classic problems of computer science can be formulated in this framework—see examples in Section 6.1.

2.2 Basic Idea of Gödel Machine

Our machine becomes a self-referential [10] *Gödel machine* by loading it with a particular form of machine-dependent, self-modifying code p. The initial code $p(1)$ at time step 1 includes a (typically sub-optimal) problem solving subroutine $e(1)$ for interacting with the environment, such as any traditional reinforcement learning algorithm [19], and a general proof searcher subroutine (Section 5) that systematically makes pairs *(switchprog, proof)* (variable substrings of s) until it finds a *proof* of a target theorem which essentially states: *'the immediate rewrite of* p *through current program* switchprog *on the given machine implies higher utility than leaving* p *as is'*. Then it executes *switchprog*, which may completely rewrite p, including the proof searcher. Section 3 will explain details of the necessary initial axiomatic system $\mathcal{A}$ encoded in $p(1)$. Compare Figure 1.

The **Global Optimality Theorem** (Theorem 1, Section 4) shows this self-improvement strategy is not greedy: since the utility of *'leaving p as is'* implicitly evaluates all possible alternative *switchprogs* which an unmodified p might find later, we obtain a globally optimal self-change—the *current switchprog* represents the best of all possible relevant self-changes, relative to the given resource limitations and initial proof search strategy.

2.3 Proof Techniques and an $O()$-Optimal Initial Proof Searcher

Section 5 will present an $O()$-optimal initialization of the proof searcher, that is, one with an optimal *order* of complexity (Theorem 2). Still, there will remain a lot of room for self-improvement hidden by the $O()$-notation. The searcher uses an online extension of *Universal Search* [23, 24] to systematically test *online proof techniques*, which are proof-generating programs that may read parts of state s (similarly, mathematicians are often more interested in proof techniques than in theorems). To prove target theorems as above, proof techniques may invoke special instructions for generating axioms and applying inference rules to prolong the current *proof* by theorems. Here an axiomatic system $\mathcal{A}$ encoded in $p(1)$ includes axioms describing **(a)** how any instruction invoked by a program running on the given hardware will change the machine's state s (including instruction pointers etc.) from one step to the next (such that proof techniques can reason about the effects of any program including the proof searcher), **(b)** the initial program $p(1)$ itself (Section 3 will show that this is possible without introducing circularity), **(c)** stochastic environmental properties, **(d)** the formal utility function u, e.g., equation (1), which automatically takes into account computational costs of all actions including proof search.

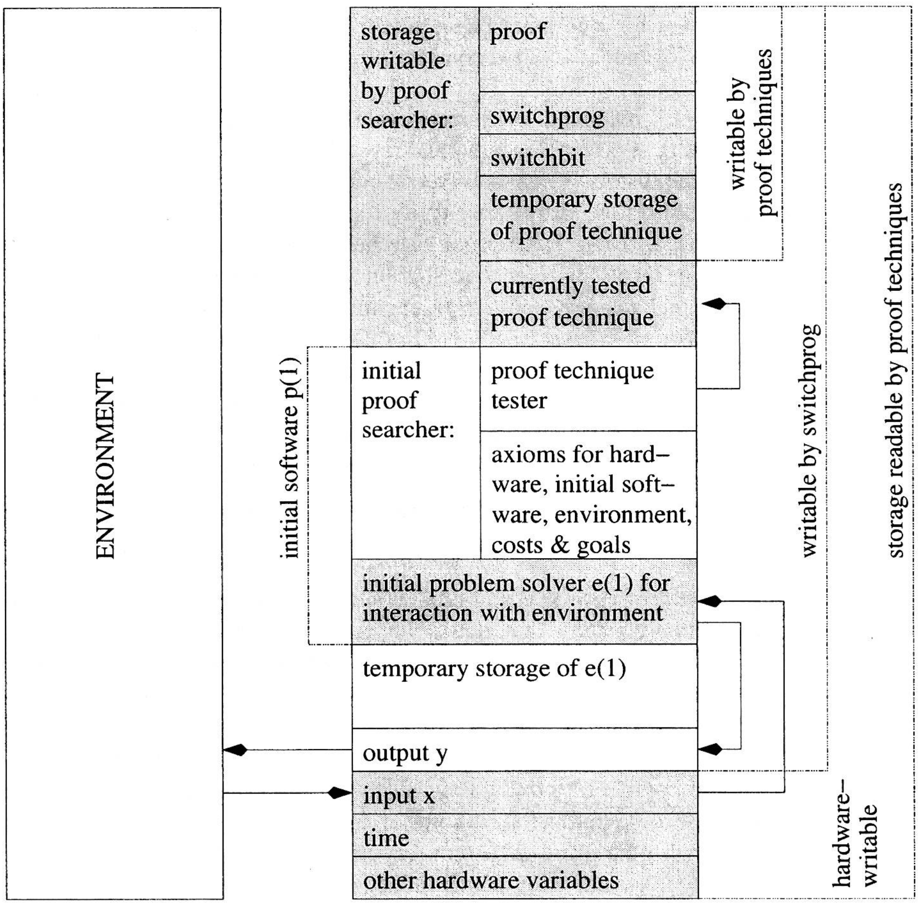


Fig. 1. Storage snapshot of a not yet self-improved example Gödel machine, with the initial software still intact. See text for details

2.4 Relation to Hutter's Previous Work

Here we will briefly review the most closely related previous work, and point out the main novelties of the Gödel machine. More relations to older approaches can be found in Section 6.2.

Hutter's non-self-referential but still $O()$-optimal *'fastest' algorithm for all well-defined problems* HSEARCH [16] uses a *hardwired* brute force proof searcher and ignores the costs of proof search. Assume discrete input/output domains X/Y, a formal problem specification $f : X \rightarrow Y$ (say, a functional description of how integers are decomposed into their prime factors), and a particular $x \in X$ (say, an integer to be factorized). HSEARCH orders all proofs of an appropriate axiomatic system by size to find programs q that for all $z \in X$ provably compute $f(z)$ within time bound $t_q(z)$. Simultaneously it spends most of its time on executing the q with the best currently proven time bound $t_q(x)$. It turns out that HSEARCH is as fast as the *fastest* algorithm that provably computes

$f(z)$ for all $z \in X$, save for a constant factor smaller than $1 + \epsilon$ (arbitrary $\epsilon > 0$) and an f-specific but x-independent additive constant [16]. This constant may be enormous though.

Hutter's AIXI*(t,l)* [15] is related. In discrete cycle $k = 1, 2, 3, \ldots$ of AIXI*(t,l)*'s lifetime, action $y(k)$ results in perception $x(k)$ and reward $r(k)$, where all quantities may depend on the complete history. Using a universal computer such as a Turing machine, AIXI*(t,l)* needs an initial offline setup phase (prior to interaction with the environment) where it uses a *hardwired* brute force proof searcher to examine all proofs of length at most L, filtering out those that identify programs (of maximal size l and maximal runtime t per cycle) which not only could interact with the environment but which for all possible interaction histories also correctly predict a lower bound of their own expected future reward. In cycle k, AIXI*(t,l)* then runs all programs identified in the setup phase (at most 2^l), finds the one with highest self-rating, and executes its corresponding action. The problem-independent setup time (where almost all of the work is done) is $O(L \cdot 2^L)$. The online time per cycle is $O(t \cdot 2^l)$. Both are constant but typically huge.

Advantages and Novelty of the Gödel Machine. There are major differences between the Gödel machine and Hutter's HSEARCH [16] and AIXI*(t,l)* [15], including:

1. The theorem provers of HSEARCH and AIXI*(t,l)* are hardwired, non-self-referential, unmodifiable meta-algorithms that cannot improve themselves. That is, they will always suffer from the same huge constant slowdowns (typically $\gg 10^{1000}$) buried in the $O()$-notation. But there is nothing in principle that prevents our truly self-referential code from proving and exploiting drastic reductions of such constants, in the best possible way that provably constitutes an improvement, if there is any.
2. The demonstration of the $O()$-optimality of HSEARCH and AIXI*(t,l)* depends on a clever allocation of computation time to some of their unmodifiable meta-algorithms. Our Global Optimality Theorem (Theorem 1, Section 4), however, is justified through a quite different type of reasoning which indeed exploits and crucially depends on the fact that there is no unmodifiable software at all, and that the proof searcher itself is readable, modifiable, and can be improved. This is also the reason why its self-improvements can be more than merely $O()$-optimal.
3. HSEARCH uses a "trick" of proving more than is necessary which also disappears in the sometimes quite misleading $O()$-notation: it wastes time on finding programs that provably compute $f(z)$ for all $z \in X$ even when the current $f(x)(x \in X)$ is the only object of interest. A Gödel machine, however, needs to prove only what is relevant to its goal formalized by u. For example, the general u of eq. (1) completely ignores the limited concept of $O()$-optimality, but instead formalizes a stronger type of optimality that does not ignore huge constants just because they are constant.
4. Both the Gödel machine and AIXI*(t,l)* can maximize expected reward (HSEARCH cannot). But the Gödel machine is more flexible as we may plug in *any* type of formalizable utility function (e.g., *worst case* reward), and unlike AIXI*(t,l)* it does not require an enumerable environmental distribution.

Nevertheless, we may use AIXI*(t,l)* or HSEARCH or other less general methods to initialize the substring e of p which is responsible for interaction with the environment. The Gödel machine will replace $e(1)$ as soon as it finds a provably better strategy.

2.5 Limitations of Gödel Machines

The fundamental limitations are closely related to those first identified by Gödel's celebrated paper on self-referential formulae [10]. Any formal system that encompasses arithmetics (or ZFC etc) is either flawed or allows for unprovable but true statements. Hence even a Gödel machine with unlimited computational resources must ignore those self-improvements whose effectiveness it cannot prove, e.g., for lack of sufficiently powerful axioms in $\mathcal{A}$. In particular, one can construct pathological examples of environments and utility functions that make it impossible for the machine to ever prove a target theorem. Compare Blum's speed-up theorem [3, 4] based on certain incomputable predicates. Similarly, a realistic Gödel machine with limited resources cannot profit from self-improvements whose usefulness it cannot prove within its time and space constraints.

Nevertheless, unlike previous methods, it can in principle exploit at least the *provably* good speed-ups of *any* part of its initial software, including those parts responsible for huge (but problem class-independent) slowdowns ignored by the earlier approaches [15, 16].

3 Essential Details of One Representative Gödel Machine

Notation. Unless stated otherwise or obvious, throughout the paper newly introduced variables and functions are assumed to cover the range implicit in the context. $l(q)$ denotes the number of bits in a bitstring q; q_n the n-th bit of q; λ the empty string (where $l(\lambda) = 0$); $q_{m:n} = \lambda$ if $m > n$ and $q_m q_{m+1} \dots q_n$ otherwise (where $q_0 := q_{0:0} := \lambda$).

Theorem proving requires an axiom scheme yielding an enumerable set of axioms of a formal logic system $\mathcal{A}$ whose formulas and theorems are symbol strings over some finite alphabet that may include traditional symbols of logic (such as $\rightarrow, \wedge, =, (,), \forall, \exists, \dots,$ $c_1, c_2, \dots, f_1, f_2, \dots$), probability theory (such as $E(\cdot)$, the expectation operator), arithmetics ($+, -, /, =, \sum, <, \dots$), string manipulation (in particular, symbols for representing any part of state s at any time, such as $s_{7:88}(5555)$). A proof is a sequence of theorems, each either an axiom or inferred from previous theorems by applying one of the inference rules such as *modus ponens* combined with *unification*, e.g., [9].

The remainder of this paper will omit standard knowledge to be found in any proof theory textbook. Instead of listing *all* axioms of a particular $\mathcal{A}$ in a tedious fashion, we will focus on the novel and critical details: how to overcome problems with self-reference and how to deal with the potentially delicate online generation of proofs that talk about and affect the currently running proof generator itself.

3.1 Proof Techniques

Brute force proof searchers (used in Hutter's AIXI*(t,l)* and HSEARCH; see Section 2.4) systematically generate all proofs in order of their sizes. To produce a certain proof, this takes time exponential in proof size. Instead our $O()$-optimal $p(1)$ will produce many proofs with low algorithmic complexity [51, 21, 25] much more quickly. It systematically tests (see Section 5) *proof techniques* written in universal language $\mathcal{L}$ implemented within $p(1)$. For example, $\mathcal{L}$ may be a variant of PROLOG [6] or the universal FORTH[27]-inspired programming language used in recent work on optimal search [46]. A proof