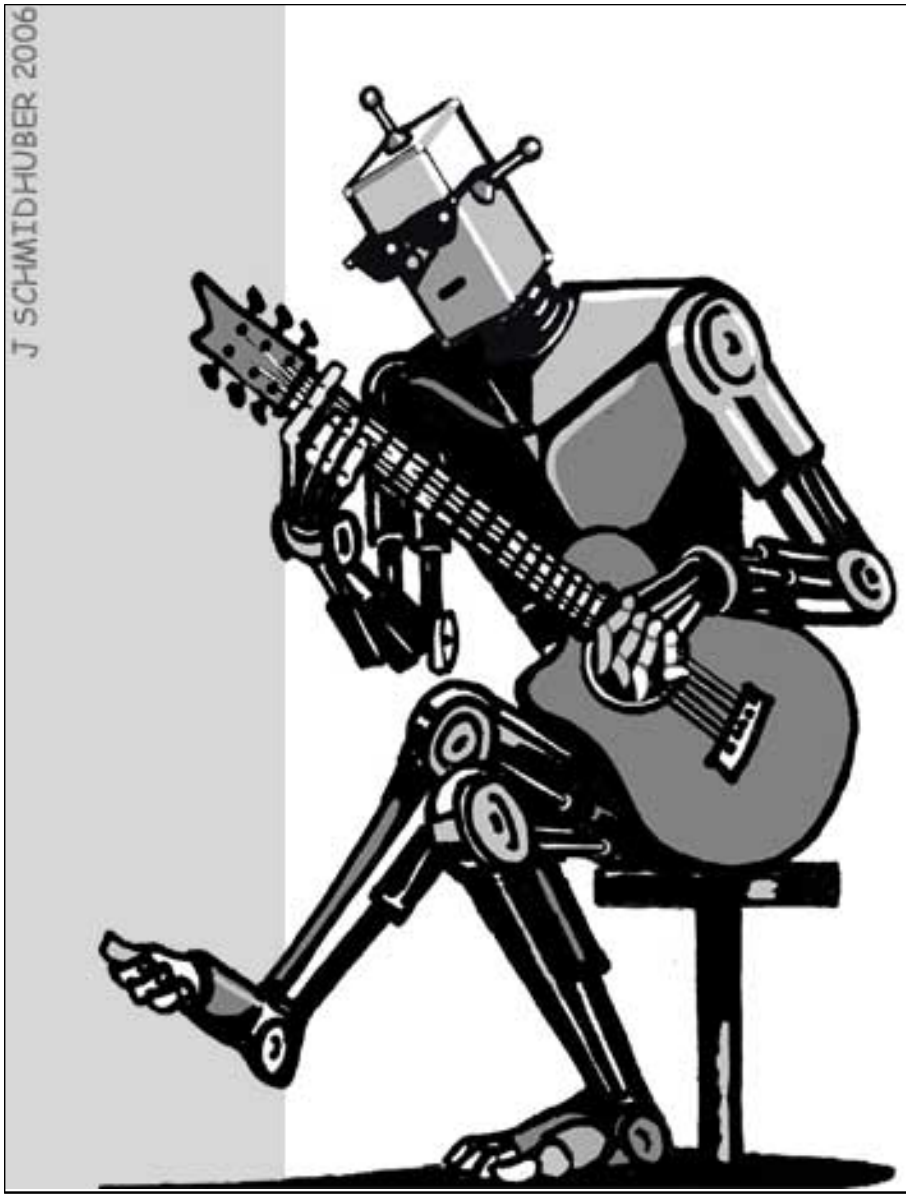


Recursive Self-Improvement

Jürgen Schmidhuber
The Swiss AI Lab IDSIA
Univ. Lugano & SUPSI
http://www.idsia.ch/~juergen

NNAISENSE

Jürgen Schmidhuber

You_again Shmidhoobuh

J. Good (1965): informal remarks on an "intelligence explosion" through RSI "super-intelligences”

Next: four concrete algorithms for RSI: 1987, 93, 97, 2003…

of meta-levels is shown next.

2.2. **Meta-evolution.**

Meta-evolution is a non-de
rithms making use of a few
(programs). On the domain le
that are useful in the domain.
elements of the programming
domain primitives in an alg
prove their adequacy by holdi
within.

On the level above the domai
by itself. This means that op
again represented as plans tha
ming language and plan mani

Because plan primitives are ab
grams are represented as plan
the level of constructing plan

And so on.

Of course this proceeding
enough, so that compositions
(Turing equivalence). One m
are able to define some kind o
to set markers, to compare ele
dependent on such tests, to ins
are necessary for well known
plan into the other one or to
should be trivial, however, the
arbitrarily complex.

To start from scratch it is nec
tactically correct plans for th
default means to intermix el
primitives at haphazard or by
constrained by the syntax of t
$n > 1$ is essentially the same w
included in that mixing process

Here is the top level loop of
language that should be self-ex

To do **meta-evolution** :

1. Set n = 1.

2. Forever do :

2.1. Call S(n) the set of nth-order-plans and set S(n) = {}.

2.2. While | S(n) | < maxpoolsize(n) do :
2.2.1. Create a new nth-order-plan by default,
give it a new name P.
2.2.2. Set S(n) = S(n) ∪ {P}.
2.2.3. **Test_and_criticize P.**

2.3. Set n = n+1.

As long as the pool of a certain level is not complete, it is enlarged. If a pool is filled, the pool corresponding to the level above is created. Pools of lower levels are changed by members of higher levels in a way that is hidden in the procedure **test_and_criticize** to which the main work is delegated.

To **test_and_criticize** a plan **P** out of S(**n**) :

1. If n=1
then

1.1. Transmit P to the domain critic who executes P in the environment and assigns a worthmeasure to it.

else
1.2. While no termination criterium is reached do :

1.2.1. Select probabilistically some plans from S(n-1) and generate a new candidate P' by applying P to the selected plans.

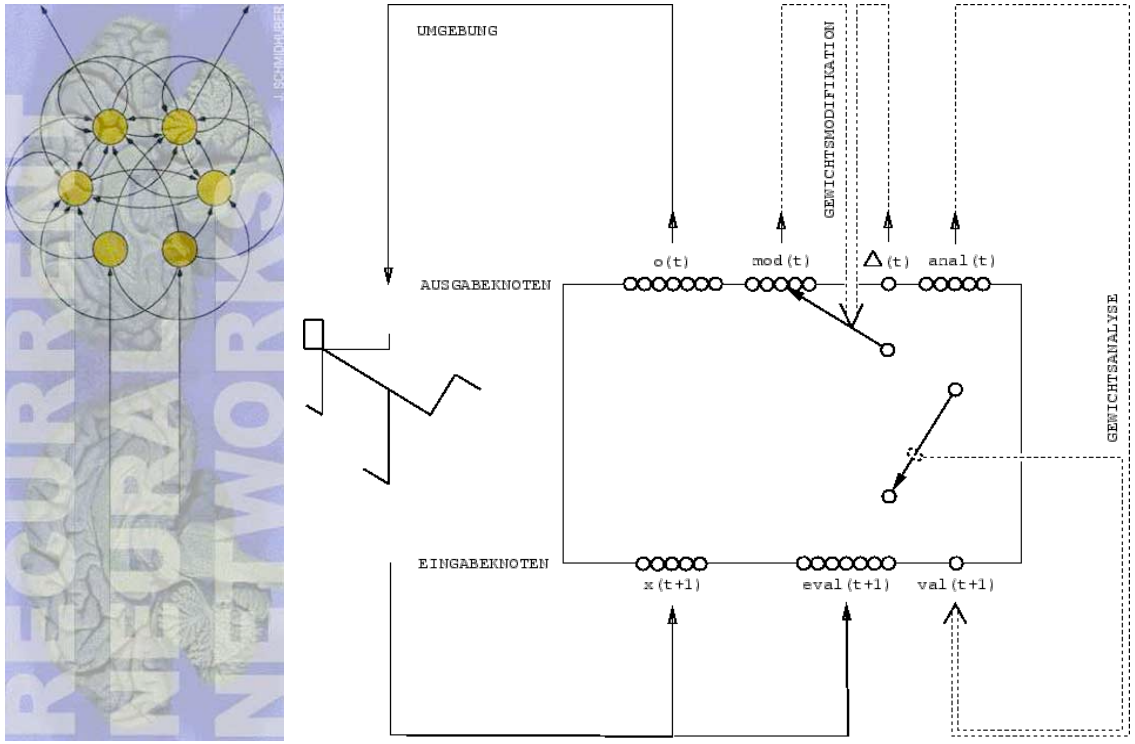
1.2.2. **Test_and_criticize P'**, treating it like a member out of S(**n-1**).

1.2.3. update the current worthmeasure of P by using information about changes of performance gained by comparing the worthmeasure of P' and its ancestors.

2. Decide whether P displaces another member of S(n).

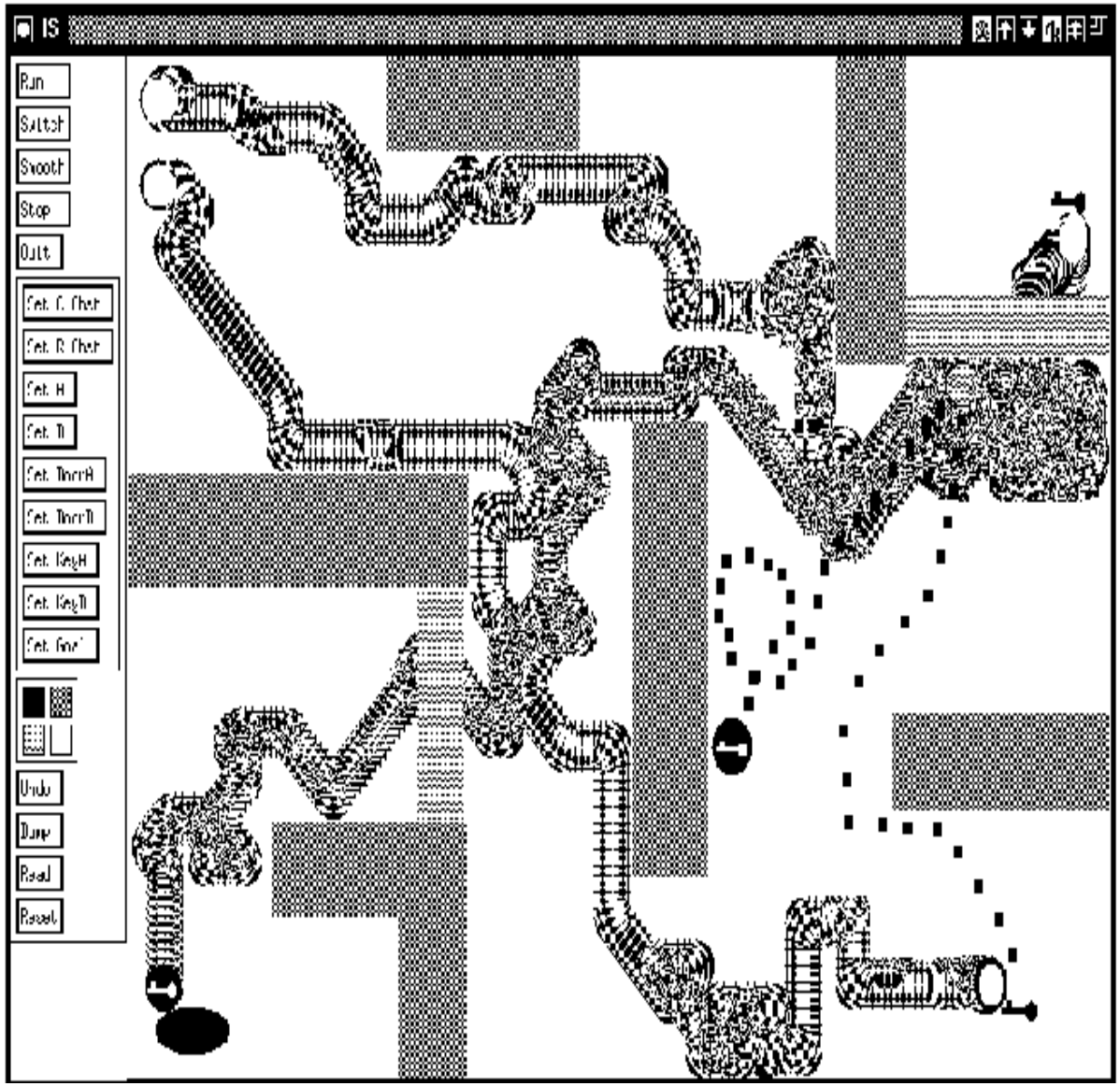
Test_and_criticize gains worthmeasures for the meta-plans it considers by applying itself recursively to the plans of lower levels generated by the meta-plans.

1987 - First RSI: Genetic Programming (Cramer, 1985) recursively applied to itself, to obtain Meta-GP and Meta-meta-GP etc:
J. Schmidhuber. Evolutionary principles in self-referential learning. On learning how to learn: The meta-meta-... hook. Diploma thesis, TU Munich, 1987

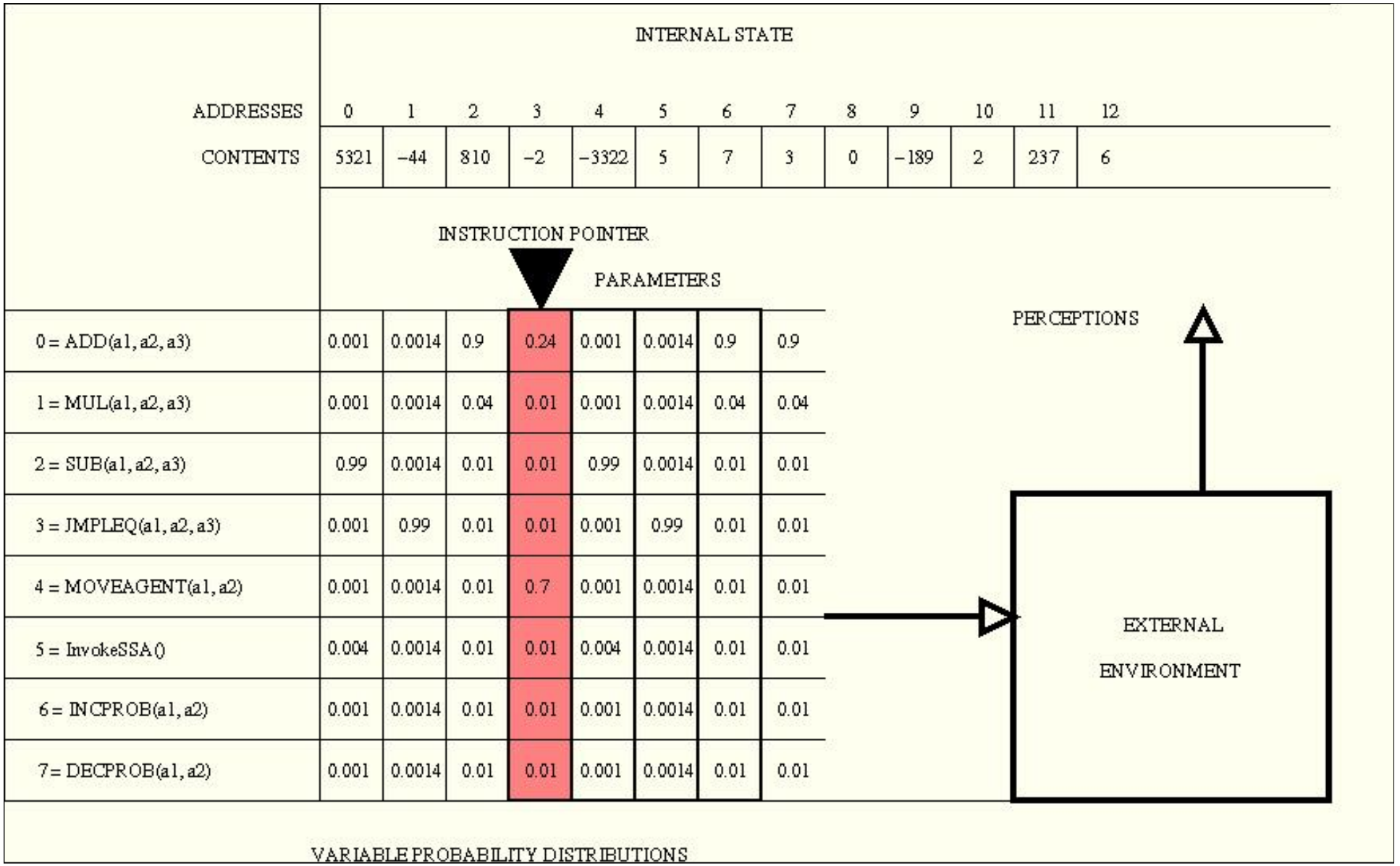


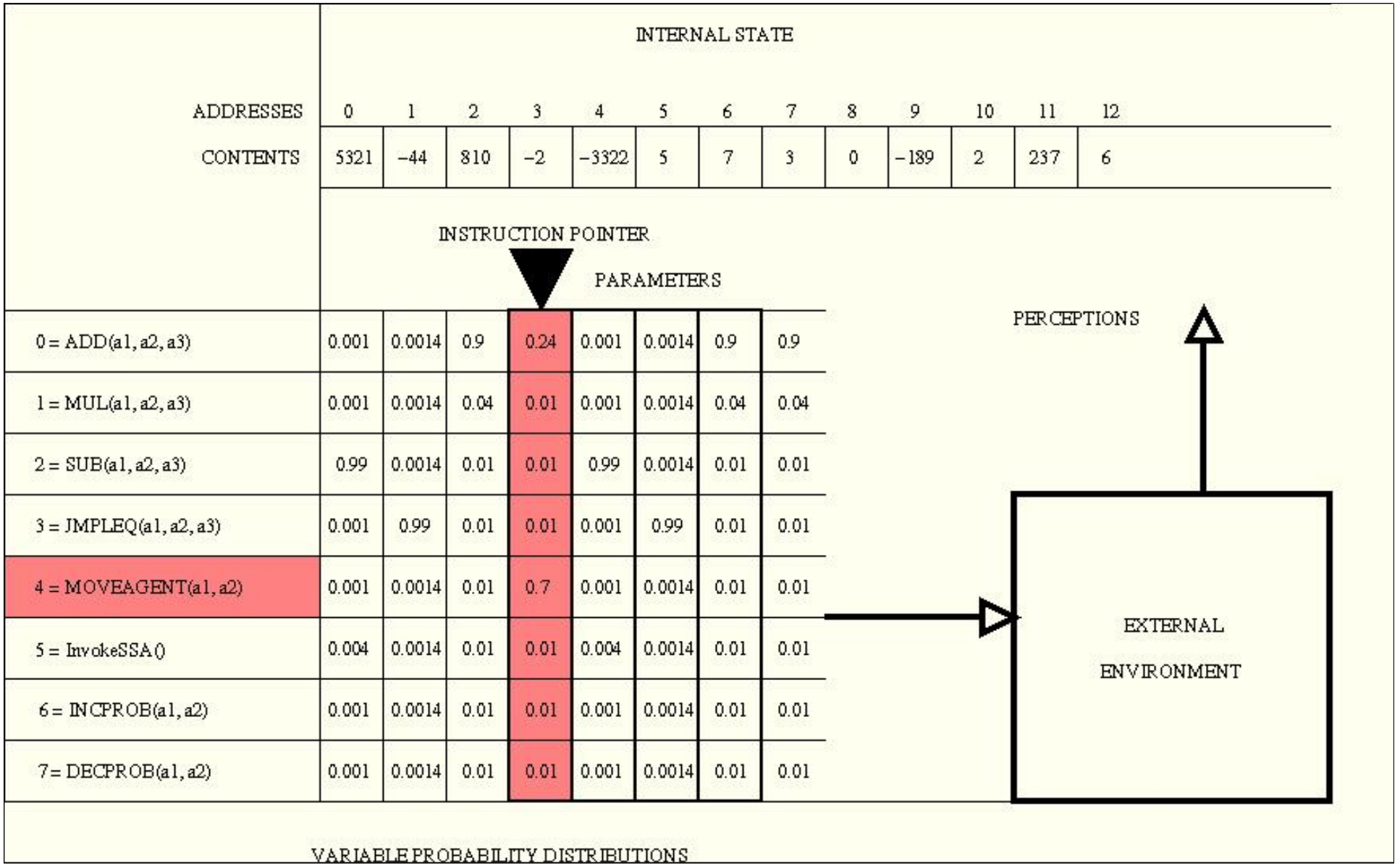
1993: Gradient-based meta-RNNs that can learn to run their own weight change algorithm:
J. Schmidhuber. A self-referential weight matrix. ICANN 1993

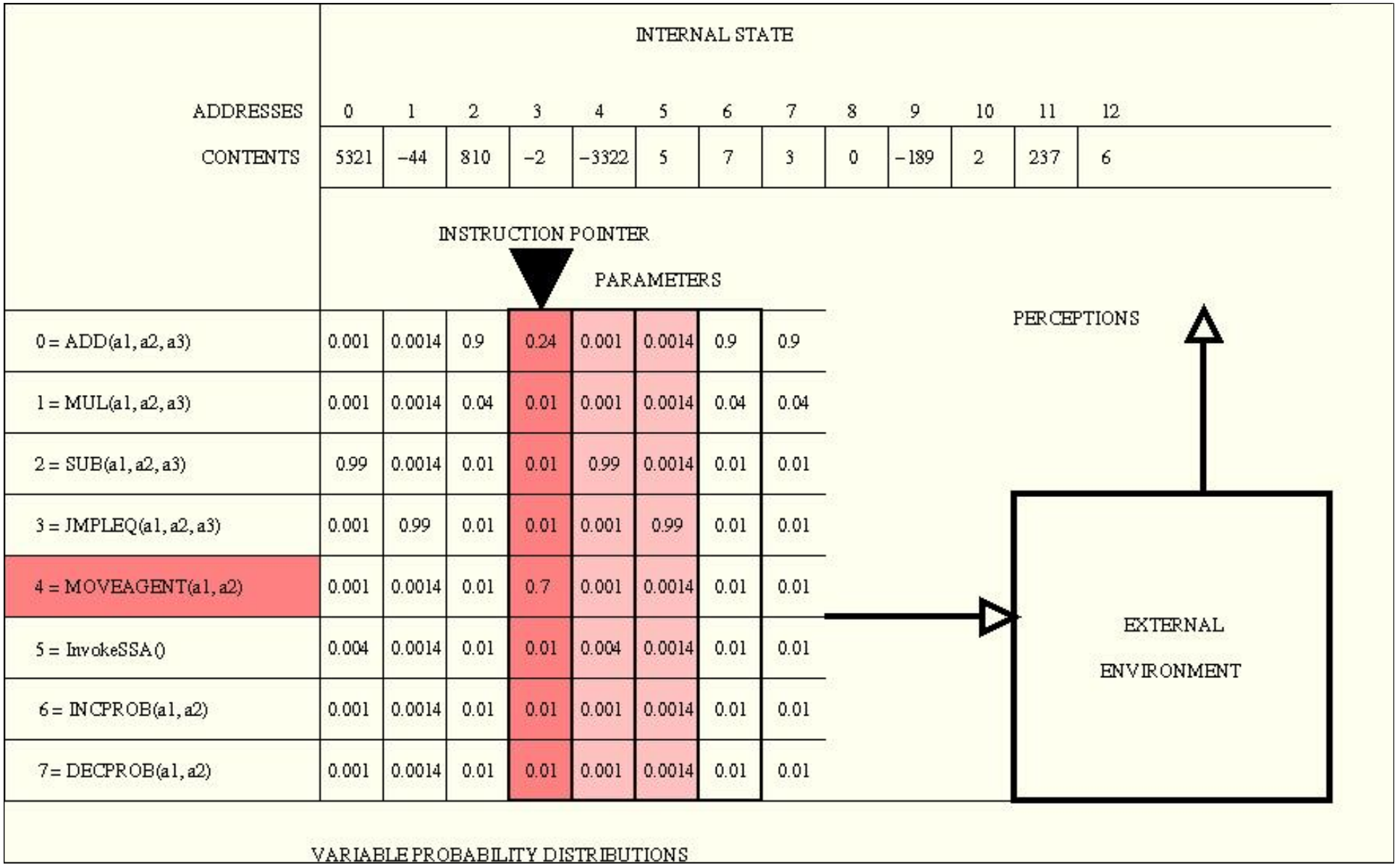
This was before LSTM. In 2001, however, Hochreiter taught a meta-LSTM to learn a learning algorithm for quadratic functions that was faster than backprop

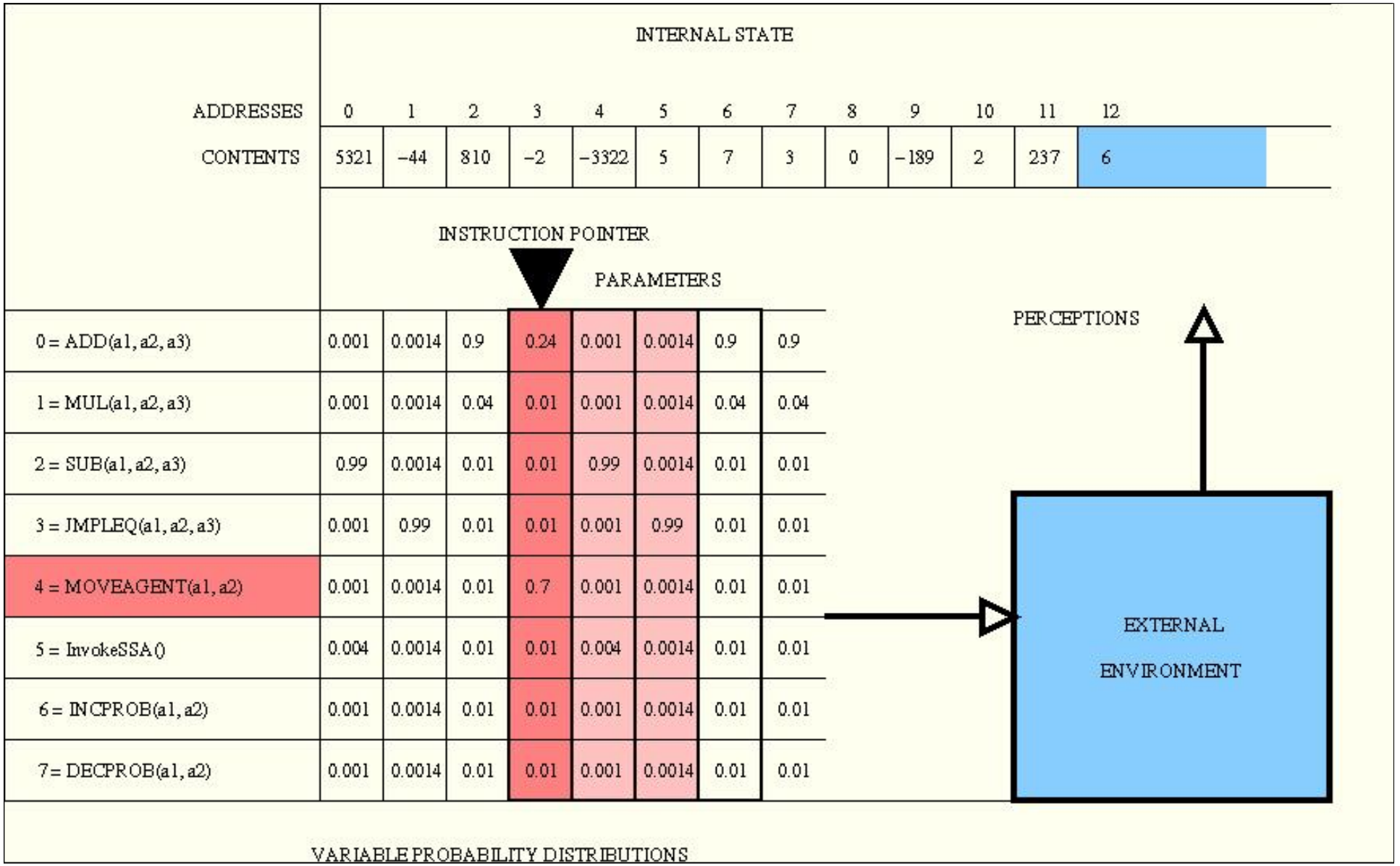


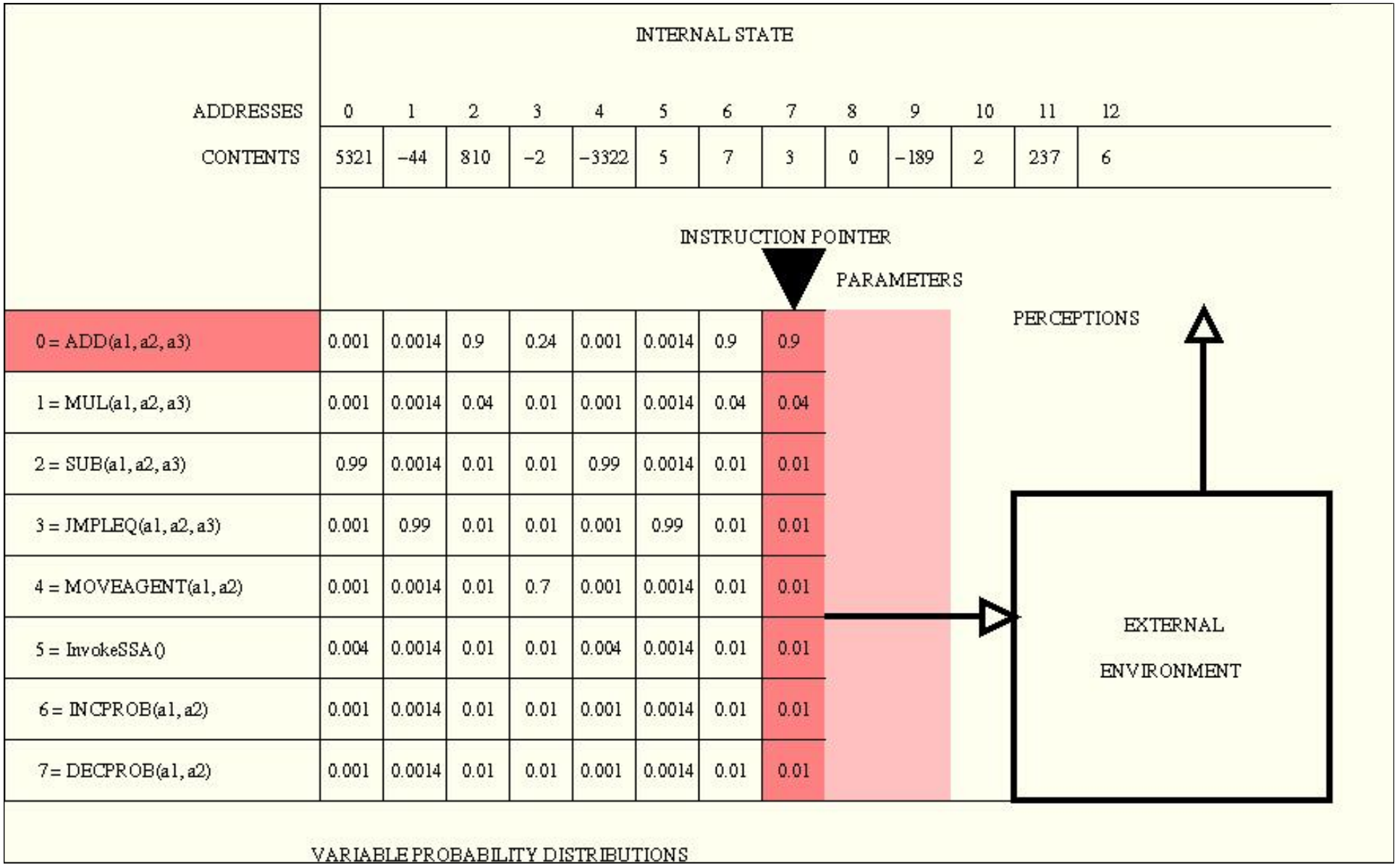
1997: Lifelong meta-learning with self-modifying policies: 2 agents, 2 doors, 2 keys. 1st southeast wins 5, the other 3. Through recursive self-modifications only: from 300,000 steps per trial down to 5,000.

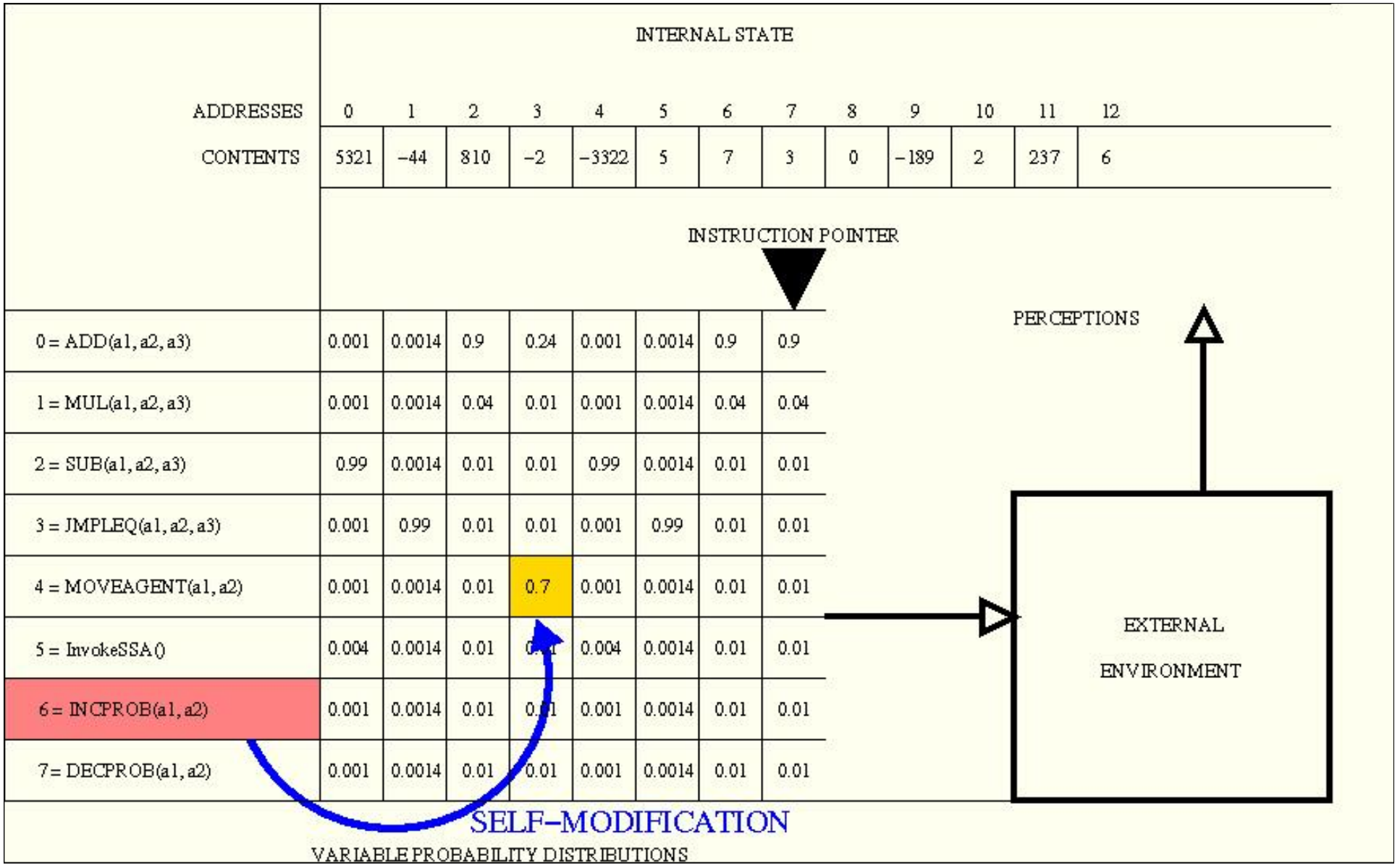

INTERNAL STATE
ADDRESSES
0 1 2 3 4 5 6 7 8 9 10 11 12
CONTENTS
5321 −44 810 −2 −3322 5 7 3 0 −189 2 237 6
INSTRUCTION POINTER
PARAMETERS
0 = ADD(a1, a2, a3) 0.001 0.0014 0.9 0.24 0.001 0.0014 0.9 0.9
1 = MUL(a1, a2, a3) 0.001 0.0014 0.04 0.01 0.001 0.0014 0.04 0.04
2 = SUB(a1, a2, a3) 0.99 0.0014 0.01 0.01 0.99 0.0014 0.01 0.01
3 = JMPLEQ(a1, a2, a3) 0.001 0.99 0.01 0.01 0.001 0.99 0.01 0.01
4 = MOVEAGENT(a1, a2) 0.001 0.0014 0.01 0.7 0.001 0.0014 0.01 0.01
5 = InvokeSSA() 0.004 0.0014 0.01 0.01 0.004 0.0014 0.01 0.01
6 = INCPROB(a1, a2) 0.001 0.0014 0.01 0.01 0.001 0.0014 0.01 0.01
7 = DECPROB(a1, a2) 0.001 0.0014 0.01 0.01 0.001 0.0014 0.01 0.01
VARIABLE PROBABILITY DISTRIBUTIONS
PERCEPTIONS
EXTERNAL
ENVIRONMENT


INTERNAL STATE
ADDRESSES
0 1 2 3 4 5 6 7 8 9 10 11 12
CONTENTS
5321 −44 810 −2 −3322 5 7 3 0 −189 2 237 6
INSTRUCTION POINTER
PARAMETERS
0 = ADD(a1, a2, a3) 0.001 0.0014 0.9 0.24 0.001 0.0014 0.9 0.9
1 = MUL(a1, a2, a3) 0.001 0.0014 0.04 0.01 0.001 0.0014 0.04 0.04
2 = SUB(a1, a2, a3) 0.99 0.0014 0.01 0.01 0.99 0.0014 0.01 0.01
3 = JMPLEQ(a1, a2, a3) 0.001 0.99 0.01 0.01 0.001 0.99 0.01 0.01
4 = MOVEAGENT(a1, a2) 0.001 0.0014 0.01 0.7 0.001 0.0014 0.01 0.01
5 = InvokeSSA() 0.004 0.0014 0.01 0.01 0.004 0.0014 0.01 0.01
6 = INCPROB(a1, a2) 0.001 0.0014 0.01 0.01 0.001 0.0014 0.01 0.01
7 = DECPROB(a1, a2) 0.001 0.0014 0.01 0.01 0.001 0.0014 0.01 0.01
VARIABLE PROBABILITY DISTRIBUTIONS
PERCEPTIONS
EXTERNAL
ENVIRONMENT


INTERNAL STATE
ADDRESSES
0 1 2 3 4 5 6 7 8 9 10 11 12
CONTENTS
5321 −44 810 −2 −3322 5 7 3 0 −189 2 237 6
INSTRUCTION POINTER
PARAMETERS
0 = ADD(a1, a2, a3) 0.001 0.0014 0.9 0.24 0.001 0.0014 0.9 0.9
1 = MUL(a1, a2, a3) 0.001 0.0014 0.04 0.01 0.001 0.0014 0.04 0.04
2 = SUB(a1, a2, a3) 0.99 0.0014 0.01 0.01 0.99 0.0014 0.01 0.01
3 = JMPLEQ(a1, a2, a3) 0.001 0.99 0.01 0.01 0.001 0.99 0.01 0.01
4 = MOVEAGENT(a1, a2) 0.001 0.0014 0.01 0.7 0.001 0.0014 0.01 0.01
5 = InvokeSSA() 0.004 0.0014 0.01 0.01 0.004 0.0014 0.01 0.01
6 = INCPROB(a1, a2) 0.001 0.0014 0.01 0.01 0.001 0.0014 0.01 0.01
7 = DECPROB(a1, a2) 0.001 0.0014 0.01 0.01 0.001 0.0014 0.01 0.01
VARIABLE PROBABILITY DISTRIBUTIONS
PERCEPTIONS
EXTERNAL
ENVIRONMENT


INTERNAL STATE
ADDRESSES
0 1 2 3 4 5 6 7 8 9 10 11 12
CONTENTS
5321 −44 810 −2 −3322 5 7 3 0 −189 2 237 6
INSTRUCTION POINTER
PARAMETERS
PERCEPTIONS
0 = ADD(a1, a2, a3) 0.001 0.0014 0.9 0.24 0.001 0.0014 0.9 0.9
1 = MUL(a1, a2, a3) 0.001 0.0014 0.04 0.01 0.001 0.0014 0.04 0.04
2 = SUB(a1, a2, a3) 0.99 0.0014 0.01 0.01 0.99 0.0014 0.01 0.01
3 = JMPLEQ(a1, a2, a3) 0.001 0.99 0.01 0.01 0.001 0.99 0.01 0.01
4 = MOVEAGENT(a1, a2) 0.001 0.0014 0.01 0.7 0.001 0.0014 0.01 0.01
5 = InvokeSSA() 0.004 0.0014 0.01 0.01 0.004 0.0014 0.01 0.01
6 = INCPROB(a1, a2) 0.001 0.0014 0.01 0.01 0.001 0.0014 0.01 0.01
7 = DECPROB(a1, a2) 0.001 0.0014 0.01 0.01 0.001 0.0014 0.01 0.01
EXTERNAL
ENVIRONMENT
VARIABLE PROBABILITY DISTRIBUTIONS


INTERNAL STATE
ADDRESSES
0 1 2 3 4 5 6 7 8 9 10 11 12
CONTENTS
5321 −44 810 −2 −3322 5 7 3 0 −189 2 237 6
INSTRUCTION POINTER
PARAMETERS
PERCEPTIONS
0 = ADD(a1, a2, a3) 0.001 0.0014 0.9 0.24 0.001 0.0014 0.9 0.9
1 = MUL(a1, a2, a3) 0.001 0.0014 0.04 0.01 0.001 0.0014 0.04 0.04
2 = SUB(a1, a2, a3) 0.99 0.0014 0.01 0.01 0.99 0.0014 0.01 0.01
3 = JMPLEQ(a1, a2, a3) 0.001 0.99 0.01 0.01 0.001 0.99 0.01 0.01
4 = MOVEAGENT(a1, a2) 0.001 0.0014 0.01 0.7 0.001 0.0014 0.01 0.01
5 = InvokeSSA() 0.004 0.0014 0.01 0.01 0.004 0.0014 0.01 0.01
6 = INCPROB(a1, a2) 0.001 0.0014 0.01 0.01 0.001 0.0014 0.01 0.01
7 = DECPROB(a1, a2) 0.001 0.0014 0.01 0.01 0.001 0.0014 0.01 0.01
EXTERNAL
ENVIRONMENT
VARIABLE PROBABILITY DISTRIBUTIONS

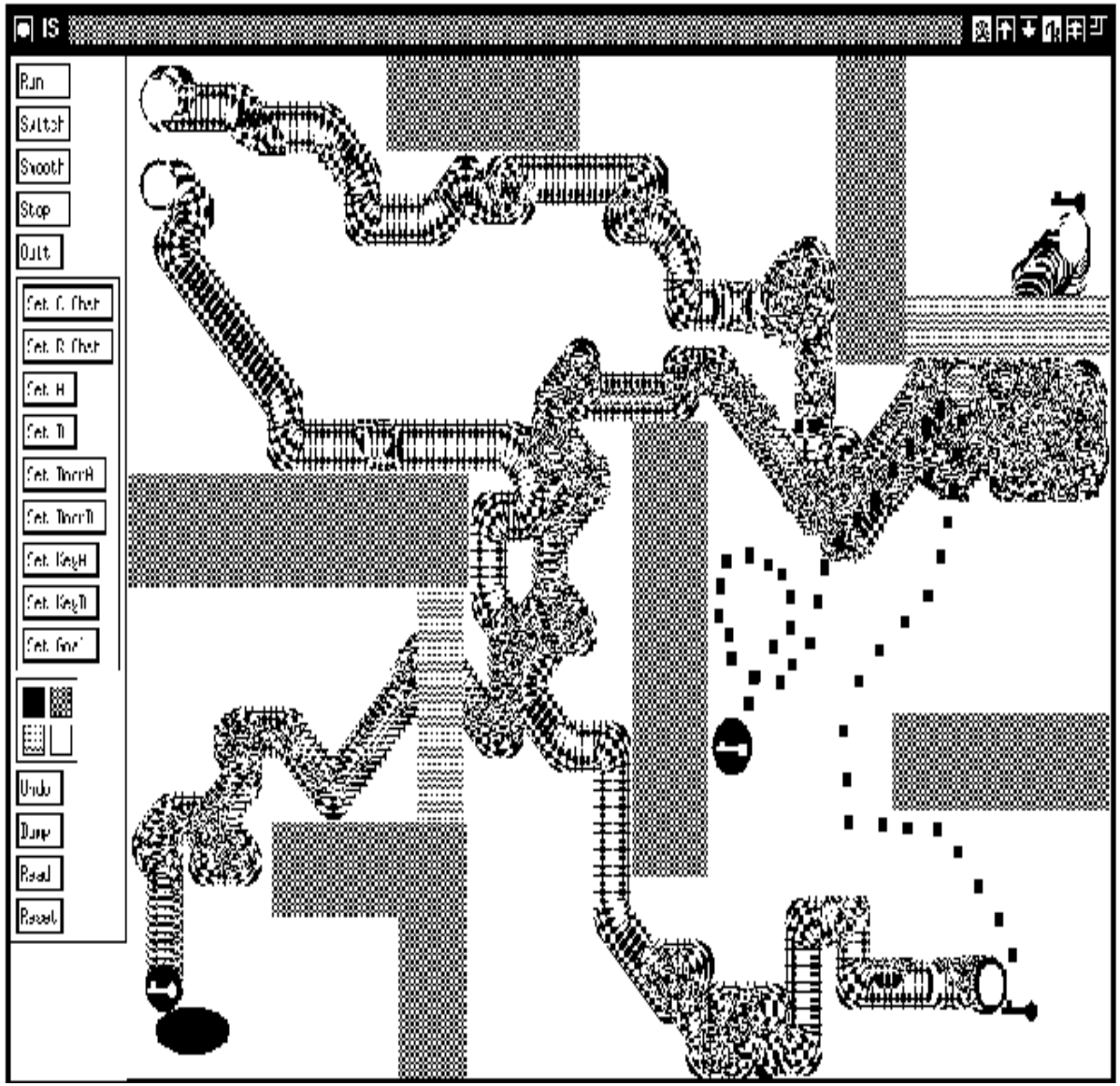

INTERNAL STATE
ADDRESSES 0 1 2 3 4 5 6 7 8 9 10 11 12
CONTENTS 5321 −44 810 −2 −3322 5 7 3 0 −189 2 237 6
INSTRUCTION POINTER
0 = ADD(a1, a2, a3) 0.001 0.0014 0.9 0.24 0.001 0.0014 0.9 0.9
1 = MUL(a1, a2, a3) 0.001 0.0014 0.04 0.01 0.001 0.0014 0.04 0.04
2 = SUB(a1, a2, a3) 0.99 0.0014 0.01 0.01 0.99 0.0014 0.01 0.01
3 = JMPLEQ(a1, a2, a3) 0.001 0.99 0.01 0.01 0.001 0.99 0.01 0.01
4 = MOVEAGENT(a1, a2) 0.001 0.0014 0.01 0.7 0.001 0.0014 0.01 0.01
5 = InvokeSSA() 0.004 0.0014 0.01 0.01 0.004 0.0014 0.01 0.01
6 = INCPROB(a1, a2) 0.001 0.0014 0.01 0.01 0.001 0.0014 0.01 0.01
7 = DECPROB(a1, a2) 0.001 0.0014 0.01 0.01 0.001 0.0014 0.01 0.01
PERCEPTIONS
EXTERNAL
ENVIRONMENT
SELF−MODIFICATION
VARIABLE PROBABILITY DISTRIBUTIONS

Success-story algorithm for self-modifying code

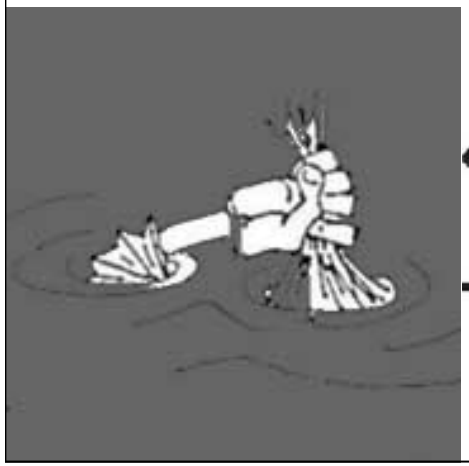
E.g., Schmidhuber, Zhao, Wiering: MLJ 28:105-130, 1997

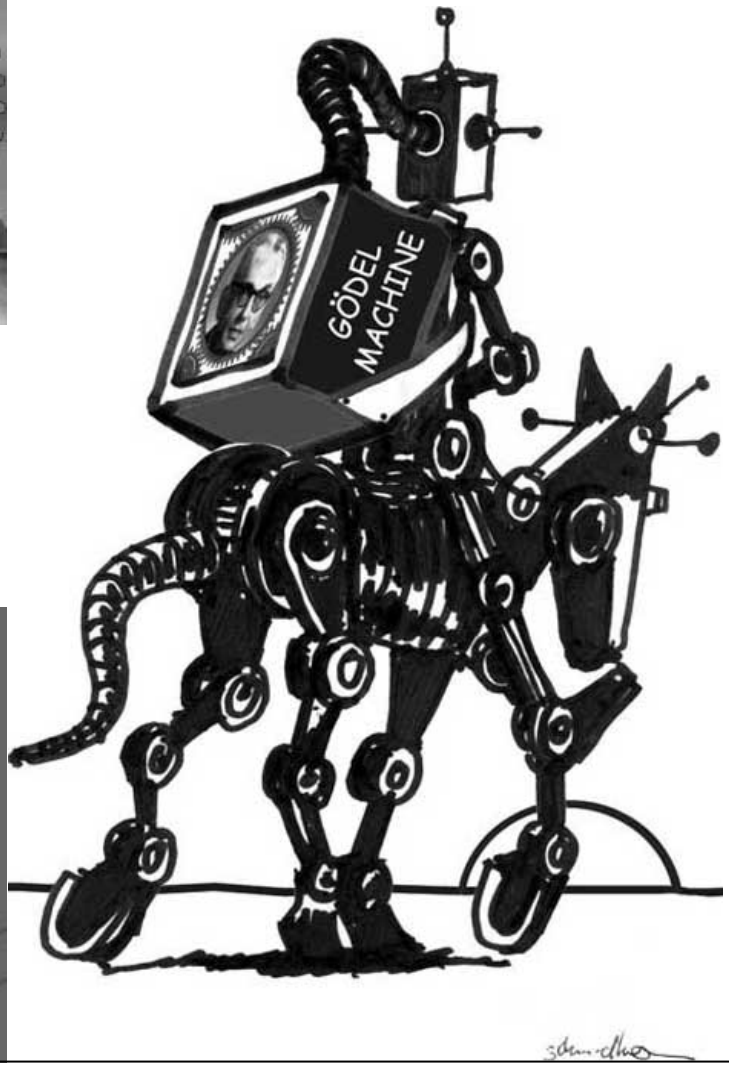
R(t): Reward until time t. Stack of past check points $v_1 v_2 v_3$ … with self-mods in between. SSA undoes selfmods after v_i that are not followed by long-term reward acceleration up until t (now):

$R(t)/t <$
$[R(t)-R(v_1)] / (t-v1) <$
$[R(t)-R(v_2)] / (t-v_2) <$…



1997: Lifelong meta-learning with self-modifying policies: 2 agents, 2 doors, 2 keys. 1st southeast wins 5, the other 3. Through recursive self-modifications only: from 300,000 steps per trial down to 5,000.



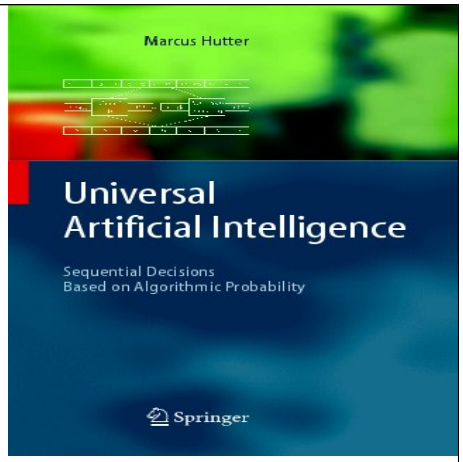


Gödel Machine (2003): agent-controlling program that speaks about itself, ready to rewrite itself in arbitrary fashion once it has found a proof that the rewrite is *useful*, given a user-defined utility function

Theoretically optimal self-improver!

Initialize GM by asymptotically fastest method for all well-defined problems

Marcus Hutter
IDSIA, 2002, on
Schmidhuber's SNF
Universal AI grant



Given $f:X \rightarrow Y$ and $x \in X$, search proofs to find program q that provably computes f(z) for all $z \in X$ within time bound $t_q(z)$; spend most time on f(x)-computing q with best current bound

$n^3+10^{10000}=n^3+O(1)$

As fast as fastest f-computer, save for factor $1+\varepsilon$ and f-specific const. independent of x!
