AMOEBA, PLANARIA and DREAMING MACHINES

Bruno MARCHAL
I.R.I.D.I.A. Université Libre de Bruxelles
50 av. F. Roosevelt. CP194/6. B-1050 Brussels, Belgium

Abstract : A general definition of *personal identity* is given for mechanical entities. It is shown how to use it to build self-reproducing machines, self-regenerating organized collection of machines, and more general self-referential nets. We study different classes of propositions that such machines are able to communicate (correctly prove, correctly infer) about themselves in the limit., together with a linked class of rememorable, and correctly known, but not as such communicable, experiences.

Key words : Machine, diagonalization, Self-Reproduction, Self-referencial nets, Inductive Inference Modality, Identity, Arithmetic[1].

1. HISTORY

Descartes's idea was that animals and human's body were machines in contrast with the immaterial human soul. Nevertheless he fail in his obligatory attempt to prove that some machine was able to reproduce itself like an animal or like our own body. The first explicit self-reproducing man-made automata has been conceived by Von Neumann some centuries later. Basically Von Neumann's construction embody a diagonalization similar to those occuring throughout the work bearing on the foundation of mathematics like the one of Cantor, Russel, Gödel, Kleene and others. (see Davis, 10) Moreover an important theorem of Kleene, which will be proved in the next paragraphe, contain all what is necessary to build a self-reproducing -or more generally a self-transforming-machine relatively to an universal environnement. Myhill has given a deep account on that result and has illustrate the strength of Kleene's theorem in building an evolutionnary machine (see 19, 27).
Kleene's theorem has been generalized by Case (see 8) for very general collection of machines including, for exemple, the offsprings of Myhill 's evolutionnary machine.
Diagonalisation, and especially his double (intensional) use has been fruitful in the theory of inductive inference (Putnam, Gold, Case & Smith), and in proof theory (Boolos, Solovay), see 28,16,9,4,32.

2. GÖDEL NUMBERINGS

Let ω be a name for the set of natural numbers :

$$\omega = \{0, 1, 2, ...\}$$

Church, Turing (and others, see Davis) has feeled the need to define precisely what is meant by *intuitively* -or *mechanicaly*- computable function from ω to ω, i.e. having number as argument and number as value. At that time there was a general feeling that such a general definition could not existed. The reason is the following one : in case such a definition could exist, we would dispose of a finitely descriptible language L, in which we can code all the computable function, and we would be able to produce an enumeration of all the computable functions by putting these codes in some order :

$$\mathbf{P}_L = \{\phi_0, \phi_1, \phi_2, \phi_3, ...\},$$

ϕ_i, is the function (extension) computed by the ieme code -or simply *i*- (intension).

[1]The following text presents research results of the Belgian National incentive-program for fundamental research in artificial intelligence initiated by the Belgian State, Prime Minister's Office, Science Policy Programming. The scientific responsibility is assumed by the author.

but then we would be able to compute the diagonal function[1] :

$\Delta_L = \lambda x\ \phi_x(x)+1$ (first diagonalisation)

which, although it is intuitively computable, differ from each ϕ_i. Indeed, in that case Δ_L would belong to $\mathbf{P}_L$, and there would exist a k such that $\Delta_L = \phi_k$. But this would entail :

$\phi_k(k) = \phi_k(k)+1$ (second diagonalisation)

It seems that any language L cannot describe how to compute Δ_L

This apparent refutation will not work if we allow L (and we allow) to describe *partial* -i.e *not defined everywhere* - function. Or, in term of machines, if we allow L to describe some machines which does not always stop on all the inputs. In that case the second diagonalisation only imply that ϕ_k is not defined on k.

Definition $\mathbf{R}_L$ is the subset of $\mathbf{P}_L$ wchich contain all total computable function descriptible in L.

If $\mathbf{P}_L$ contains all the intuitively computable function, then a rereading of the proof above show only that there is no L-descriptible machine capable of distinguishing the (code of) function belonging to $\mathbf{R}_L$
Now it happens that any attempt to capture formally the set of intuitive computable functions -partial functions include- has given rise to the same set of functions, i.e. to the same subset $\mathbf{P}$ of ω^ω.: $\mathbf{P}_{algol} = \mathbf{P}_{lisp} = \mathbf{P}_{prolog} = \mathbf{P}_{c++} = \mathbf{P}_{game\text{-}of\text{-}life} = \mathbf{P}_{n\text{-}body\ problem}$.(Moore 25)

This is the basic empirical motivation for Church's thesis : $\mathbf{P}_{your\text{-}prefered\text{-}programming\text{-}language}$ = the set of intuitively computable function = $\mathbf{P}$. There is a corresponding thesis for $\mathbf{R}$. In particular these functions are the one which are computable with a computer, and tranlatable in (any) programming language[2].. Exemple : The following, algorithmicaly generated, list of Lisp programs give rise to a precise enumeration (with repetition) of $\mathbf{P}$:

```
1  (lambda (x) x)
2  (lambda (x) 'equal)
3  (lambda (x) 'car)
4  (lambda (x) 'cdr)
5  (lambda (x) (car x))
6  (lambda (x) (cdr x))
7  (lambda (x) 'x)
8  (lambda (x) (k x))
9  (lambda (x) (null x))
10 (lambda (x) 'quote)
11 (lambda (x) 'lambda)
12 (lambda (x) 'k)
13 (lambda (x) 'cons)
14 (lambda (x) 'cond)
15 (lambda (x) 'null)
16 (lambda (x) '(equal))
17 (lambda (x) '(car))
18 (lambda (x) '(cdr))
19 (lambda (x) '(x))
20 (lambda (x) '(quote))
21 (lambda (x) '(lambda))
22 (lambda (x) '(k))
23 (lambda (x) '(cons))
24 (lambda (x) '(cond))
25 (lambda (x) '(null))
26 (lambda (x) (equal x x))
27 (lambda (x) (cons x x))
```
etc.

In $\mathbf{P}$ we can interpret ϕ_i as the function from ω in ω, computed by the $i^{ème}$ lambda expression given in the list above, where ω is seen as a set of code for finite expressions. $\mathbf{P}$ is called a Gödel numbering (Rogers see 30) and it enjoy two remarquable properties. First :

$$\exists u\ \forall i \forall x \quad \phi_u(i,x) = \phi_i(x) \qquad (1)$$

This mean that there is a machine *u* able to compute any ϕ_i, given his description *i*, on his domain. *u* correspond to the code (we identify number and code) of an universal machine : like computers, interpreters... *i* plays the role of the program for that computer.

---

[1] λx means that x is the variable, for exemple $\lambda x\ x$ is the identity function which send each number on itself.

[2] With the notable exception of some intuitionnist attempt to capture reasonnably big enough enumerable set of total computable functions (platonistically include in R).

Definition : we call such an *u* an universal environment. Second :

$$\exists s \ \forall i \forall x \forall \mathbf{y} \ \ \phi_i(x,\mathbf{y}) = \phi\phi_s(i,x)(\mathbf{y}). \qquad (2)$$

s is a metaprogram which parametrise a function of n+1 variables in a function of n variables by fixing one of the variable. For exemple:

$$\phi_s((\lambda x \lambda y \ x+y), 2) = \lambda y \ 2+y.$$

Function of several variables are implicitely represented in **P** by function of one variable, that variable being interpreted as the (code of) a list (possibly empty) of variables. Here is s given in LISP :

```
(def 's '(lambda (p d)
  (list (quote lambda)
      (enlevedebut (length d)
            (argu p))
      (subst-list-sauf-quote (mapquote d)
                  (npremier (length d)(argu p))
                  (body p)))))
```

s is essentially the substitution function introduced by Gödel in his famous paper on incompleteness. *Subst-list-sauf-quote* is a generalised substitution function which handle our identification between number and program (or machine), *enlevedebut* put away the "length <input>" first element of the input list. We are ready to give Kleene 's theorem :

$$\forall t \ \ \exists e \ \ \forall y \quad \phi_e(y) = \phi_t(e,y)$$

In english : for any computable transformation coded by *t*, there is a machine coded by *e* which on any input y (written in bold for remembering that it could be a list , possibly empty, of inputs) compute the transformation t, parametrized by y, on itself.

Proof. $\lambda x \lambda y.\phi_t(\phi_s(x,x),y)$ (first diagonalisation) is an intuitively computable function. So, by Church's thesis, there is a ϕ_r in **P** wich compute it.

$$\phi_r(x,y) = \phi_t(\phi_s(x,x), y).$$

Using automated parametrization (2) :

$$\phi_r(x,y) = \phi\phi_{s(r,x)}(y).$$

With x = r (second diagonalisation) we get :

$$\phi_t(\phi_s(r,r), y) = \phi\phi_{s(r,r)}(y).$$

We are done with $e = \phi_s(r,r)$. QED.

To avoid the use of Church's thesis it suffices to do the same work in a particular formal system.

Here is a description of the diagonalisation written in LISP :

```
(def 'diag '(lambda (f)
  (list (quote lambda)
      (argu f)
       (subst-sauf-quote
       (list (quote s)
            (car (argu f))
            (list (quote list)
                 (car (argu f))))
       (car (argu f))
       (body f)
      ))
))
```

The double diagonalisation is captured, in an uniform way, with the following program :

```
(def 'k '(lambda (f)
  (s (diag f) (list (diag f)))))
```

So k on any code of a transformation t will build the machine which apply t on itself.
Definition. k is Kleene's metaprogram.

We are ready to give our definition of *personal identity* : a machine M has personal identity with respect to an (intensionally conceived) function f relatively to an universal environment *u* if M = (k f) in *u*. Personal identity, so defined, is a relative and intensional concept.

3. THE AMOEBA

A self-reproducing machine is a machine which has personal identity with respect

to the identity function. Such a machine apply identity (I = λx x) on itself :

$$\phi_e() = I(e) = e$$

exemple : if the universal environment is LISP then I = (lambda (x) x) :

```
(def 'I '(lambda (x) x) )
```

Then, to get the self-reproducing machine -the amoeba- in LISP, it suffice to apply Kleene metaprogram k on I :

```
(k I)
```

```
(lambda nil
  (s '(lambda (x) (s x (list x)))
    (list '(lambda (x) (s x (list x))))))
```

This program apply on no input give itself :

```
((k I))
```

```
(lambda nil
  (s '(lambda (x) (s x (list x)))
    (list '(lambda (x) (s x (list x))))))
```

4. THE PLANARIA

Here is the generalisation (Case 74) : For all machine *t*, there exist a machine *e* such that

$$\phi_{\phi_{\cdots\phi_e(x_1)\cdots}(x_n)}(a) = \phi_t(e,x_1,...,x_n,a)$$

Proof : (by induction)

For n=1, it is Kleene theorem. Using the parametrisation theorem we find a *g* (given by an application of the metaprogram *s*) such that :

$$\phi_{\phi_g(x,x_1...x_n)}(a) = \phi_t(x,x_1,...,x_n,a) \quad (*)$$

take $g = \phi_s(t,x,x_1,...,x_n)$.
Suppose the theorem true for n-1, then we can apply it to $\phi_g(x,x_1,...,x_n)$ with x_n playing the role of u. So there is a *e* such that :

$$\phi_{\phi_{\cdots\phi_e(x_1)\cdots}(x_{n-1})}(x_n) = t(e,x_1,...,x_n)$$

Substituting in (*) gives the result. QED.

Such a e defines what we shall call a self-referential net.

With that generalisation we are able to buid a *planaria* -a little flatworm with an amazing power of self-regeneration (Buschsbaum, see 6). I show the simplest one : it is a list of two cells (C1,C2) such that on a certain flag FLAG, each cell reconstitutes the entire planaria (C1,C2).

$$C1(FLAG) = (C1,C2)$$
$$C2(FLAG) = (C1,C2)$$

I give different (extensional) functions for each cells :

$$C1 = \lambda x.x+1$$
$$C2 = \lambda x.x+2$$

Construction : we know by the generalized recursion theorem (with n=2) :

$$\forall t \exists e \quad \phi_{\phi_e(\mathbf{y})}(\mathbf{z}) = \phi_t(e,\mathbf{y},\mathbf{z}) \quad (**)$$

ϕ_e will be the generator of C1 and C2 : $\phi_e(1) = C1$, $\phi_e(2) = C2$, so we want :

$$\phi_{\phi_e(1)}(\mathbf{z}) = \text{if } z=FLAG \text{ then op } (\phi_e(1),\phi_e(2)) \text{ else } z+1$$

$$\phi_{\phi_e(2)}(\mathbf{z}) = \text{if } z=FLAG \text{ then op } (\phi_e(1),\phi_e(2)) \text{ else } z+2$$

e will be find by applying (**) on the following *recursion matrix* :

$$t(x,1,FLAG) = (\phi_x(1),\phi_x(2))$$
$$t(x,2,FLAG) = (\phi_x(1),\phi_x(2))$$
$$t(x,1,z) = z+1, \text{ if } z \neq FLAG$$
$$t(x,2,z) = z+2, \text{ if } z \neq FLAG$$
$$t(x,y,z) = \text{'erreur if } y > 2.$$

that is to say, by applying k and the parametrization function following the proof of the generalized recursion theorem. I do it in the LISP, (see the semantics below), in which

$$\phi_{\phi_e(y)}(z)$$

is represented by ((e y) z), i.e meta-programming = parenthesized at the left. I am using "m" instead of "t", because "t" means *true* in LISP.

```
(def 'm  '(lambda (x y z)
      (cond
        ((and (equal y 1) (equal z 'FLAG))
          (list (x 1) '***** (x 2) ))
        ((and (equal y 2) (equal z 'FLAG))
          (list (x 1) '+++++ (x 2) ))
        ((equal y 1) (+ z 1))
        ((equal y 2) (+ z 2))
        (t '(ERREUR il n'y a que 2 cellules))
       ))
    )
```

then we define g following the proof of the generalisation of Kleene's theorem :

```
(def 'g '(lambda (x y) (s m (list x y))) )
```

it suffices now to use Kleene's k to get the planaria :

```
(def 'p (k g))
```

and the second cell generates the entire planaria when applied on the flag FLAG :

```
((p 2) 'FLAG)
```

```
((lambda (z)
   (cond ((and (equal '1 1) (equal z
'FLAG))
          (list ('(lambda (y)
                    (s m
                       (list (s '(lambda (x
y) (s m (list (s x (list x)) y)))
                             (list '(lambda
(x y)
                                      (s m (list
(s x (list x)) y)))))
                             y)))
                1)
                '*****
                ('(lambda (y)
                   (s m
                      (list (s '(lambda (x
y) (s m (list (s x (list x)) y)))
                            (list '(lambda
(x y)
                                     (s m (list
(s x (list x)) y)))))
                            y)))
                2)))
         ((and (equal '1 2) (equal z
'FLAG))
          (list ('(lambda (y)
                    (s m
                       (list (s '(lambda (x
y) (s m (list (s x (list x)) y)))
                             (list '(lambda
(x y)
                                      (s m (list
(s x (list x)) y)))))
                             y)))
                1)
                '+++++
                ('(lambda (y)
                   (s m
                      (list (s '(lambda (x
y) (s m (list (s x (list x)) y)))
                            (list '(lambda
(x y)
                                     (s m (list
(s x (list x)) y)))))
                            y)))
                2)))
         ((equal '1 1) (+ z 1))
         ((equal '1 2) (+ z 2))
         (t '(erreur il n 'y a que 2
cellules))))
  +++++
  (lambda (z)
    (cond ((and (equal '2 1) (equal z
'FLAG))
           (list ('(lambda (y)
                     (s m
                        (list (s '(lambda (x
y) (s m (list (s x (list x)) y)))
                              (list '(lambda
(x y)
                                       (s m (list
(s x (list x)) y)))))
                              y)))
                 1)
                 '*****
                 ('(lambda (y)
                    (s m
                       (list (s '(lambda (x
y) (s m (list (s x (list x)) y)))
                             (list '(lambda
(x y)
                                      (s m (list
(s x (list x)) y)))))
                             y)))
                 2)))
          ((and (equal '2 2) (equal z
'FLAG))
           (list ('(lambda (y)
                     (s m
                        (list (s '(lambda (x
y) (s m (list (s x (list x)) y)))
                              (list '(lambda
(x y)
                                       (s m (list
(s x (list x)) y)))))
                              y)))
                 1)
                 '+++++
                 ('(lambda (y)
                    (s m
                       (list (s '(lambda (x
y) (s m (list (s x (list x)) y)))
```

```
                                (list '(lambda
(x y)
                                        (s m (list
(s x (list x)) y)))))
                                y)))
                        2)))
                ((equal '2 1) (+ z 1))
                ((equal '2 2) (+ z 2))
        (t '(erreur il n 'y a que 2
cellules)))))
```

and each cells works :

((p 1) 3)
4

((p 2) 3)
5

Some handling of λ-expression permits to build less redondant cells.
This planaria *p* has personal identity -with respect to a simple self-regenerating ability- relatively to LISP.
Note that in a self-regenerating net each cell is potentially an egg.

5. MYHILL'S MACHINES

Remark. All what has been prove so far can be formalized in a sufficiently rich formal theory. The interest of that fact does not lie in the quest of some rigorous presentation, but because it means that we are able to construct a machine with the ability of proving these facts. In particular we are interested in what a machine is able to prove concerning its own capability of proving propositions. We must distinguish between M proves a simple proposition p (we shall write $M \vdash p$, or simply $\vdash p$) and M proves the rather sophisticated proposition "M proves p". (which we should write $M \vdash \square_M \ulcorner p \urcorner$). with "$\square_M$"representing M's provability ability in M's language and $\ulcorner p \urcorner$ representing a representation of p in M's language, but we shall simply write $\square p$).

Definition. A machine M is said to be self-referentially correct. in an universal environment u if M has personal identity with respect to provability relatively to u. (for a related definition see also Smullyan)

If that machine is able to prove correctly some elementary facts concerning substitution, then we can find a sentence *g* such that the M will prove :

$$g \leftrightarrow \neg \square g$$

This is a particular case of the diagonalisation schema : we can replace $\square$ by any formula with one free variable and find a corresponding g. The diagonalisation lemma is essentially the formalisation of Kleene's recursion theorem. Now if M does not prove any false statements -i.e if M is consistent- then g is true and M is not able to prove g. (this is basically Gödel's first incompleteness theorem). This proof is *constructive* (algorithmic) so that we can add to M's power the algorithmic ability to synthetize this very sentence. In that case we obtain a new machine M^2 with more powerfull provability capacities. This construction can be iterated in the constructive transfinite (see Feferman 12).

With Case's extension of Kleene's theorem we can build a self-referential sequence of more and more powerfull machines.with respect to provability.
Suppose x is the code of a machine M which has a recognizable part which is a theorem prover, then M belongs to the domain of the two following function : 1) $\lambda x T(x)$, T genere the set of theorems of the proving system of x. 2) $\lambda x RL(x)$, RL transforme[1] x into a equivalent machine except that he add a statement,true but not provable, concerning x in -let us say- the set of axioms of x.
We want build a sequence of machines : m_0, m_1, m_2, m_3..., such that each m_i, on a certain flag -PROVE- gives $T(m_i)$, and on a flag -NEXT- gives m_{i+1}, where m_{i+1} is $RL(m_i)$. m_0(PROVE) generate the theoreme of an elementary first order axiomatisation of Peano Arithmetic (PA).

---

[1] RL for *Refutation* of mechanist philosophy given by *Lucas*.Such a refutation is by itsef mecanisable. The refutation of mechanism and the mechanical refutation of that refutation has been already done by Post. and Webb has write a book on the subject.

As in the planaria, the role of m_i, are played by the $\phi_e(i)$:

$\phi_{\phi_e(i)}(z)$ = if i=0 & z = PROVE op T(PA), else
if z=NEXT then output ($\phi_e(i+1)$), else
if z= PROVE then output T(RL($\phi_e(i-1)$)).

As above *e* will be given by applying *k* on a *s* variant of the recursion matrix :

t(x,i,z) = if i=0 & z = PROVE op T(PA), else
if z=NEXT then output $\phi_x(i+1)$, else
if z=PROVE output T(RL($\phi_x(i-1)$)). (3)

z being any list, we can made any m_i universal :

if z=(UNIV x y) then output $\phi_x(y)$.

6. INDUCTIVE INFERENCE

The first who has used diagonalisation in the field of inductive inference is Putnam.
Definition. (Gold) An Inference Inductive Machine (IIM) is a machine which receives successively as inputs *all* (in the limit) couples <input-output> (in any order), of a function f and which successively outputs programs called hypotheses. We will say that f is presented to M. The IIM converges if it outputs finally always the same program. The IIM M correctly identifies f and we write $f \in EX(M)$ if M converges to a program which computes f. Note that any ϕ_i is trivially identifiable : ϕ_i is always identified by the idiotic $\lambda x\ i$, so that the interesting concept is the identification -by *one* IIM- of a *subset* of **R**

Definitions
1) $EX = \{S \subset: \mathbf{R}\ \exists M\ S \subset EX(M)\}$.
Putnam has gived a more strict criterion which can be shown equivalent to the *Popperianity* of the IIM :
2) Case & Smith call an IIM Popperian if all the hypotheses produced are total functions. (So the generated hypotheses are all refutable in Popper sens).
3) $PEX = \{S : \exists M$ Popperian $S \subset EX(M)\}$.

Theorem (Putnam, see 28) : $\mathbf{R} \notin$ PEX.
Putman 's proof is based on the intuitive feeling that any given inductive machine α can be made wrong. Indeed let m be the code of α, then the following computable function defeat any PEX machine :

$\phi_e(0) = 0$
$\phi_e(x+1) = \phi_{\phi_m(\phi_e(: x))}(x+1)$.

where $\phi_e(: x)$ is :

$\{(0,\phi_e(0)), (1,\phi_e(1)), \ldots (x, \phi_e(x))\}$

QED. The proof is constructive, so that it is easy to add evolving inductive inference capabilities to the self-referential nets.

Theorem (Case et Smith) : $PEX \subsetneq EX$
Case and Smith has shown that the set of functions computed by self-reproducing machines (on the flag 0) :

$$S = \{f : \phi_{f(0)} = f\}$$

belongs to EX, but not to PEX. Indeed, the function :

$\phi_e(0) = e$; compare with Putnam's function
$\phi_e(x+1) = \phi_{\phi_m(\phi_e(: x))}(x+1)$.

is trivialy identified by the IIM β which output a on {(0,a), …}.

Here is a translation of the proof in LISP:

```
(def 'EX-BUT-NOT-PEX '(lambda (iim-pex-var)
 (k
  (list 'lambda (list 'y 'x)
      (list 'cond
          (list (list 'equal 'x 0) 'y)
          (list 't (list (list '+ 1
                       (list  (list iim-pex-var
                                (list  'ulis 'y
                                     (list '- 'x 1)))
                          'x))
                    )
              )
          )
      )
  )
```

) ; ulis (i,n) = {(o,ϕ_i(o), …(n,ϕ_i(n))}
)) ; k is Kleene's metaprogram. QED.

It is easy to show that any algorithmatically generable subset of **R** belongs to EX (Gold, see 16), but, like PEX, **R** itself does not.

We can enlarge the set of identifiable functions by a non-constructive weakening of the identification criteria.

Definitions. (Case and Smith, see 9)

1) f $=^{0v1}$ g means that f is equal to g excepting, *may be*, on a single input. Put in another way, it means that {x : f(x)=g(x)} has cardinality less *or* equal to 1.

2) M EX^1-identify f if, when we present f to M, it converges on a program computing a g such that f $=^{0v1}$ g. We write f $\in$ EX^1(M).

3) EX^1 = {S : $\exists$M S $\subset$ EX^1(M)}.

Theorem. (Case and Smith) EX $\subsetneq$ EX^1
More precisely they have shown that the set :

$$S^{0v1} = \{f : \phi_{f(0)} =^{0v1} f\}$$

does not belong to EX, (see) but the IIM α (see above) witness that it belongs to EX^1.

The "*may be*" is ineluctable : let us prove that such an enlargement cannot be done constructively (algorithmicaly) (Chen)[1] :

Proof. Suppose that there is a computable function ϕ_j such that for any inductive inference machine M : 1) $\phi_{\phi_j(M)}$ is total computable and 2) $\phi_{\phi_j(M)}$ does not belong to EX(M). In that case the following IIM

$$m = \lambda x\ (k\ j)$$

i.e. the machine which on any inputs always output ϕ_j apply on itself, is such that $\phi_{\phi_j(m)}$ belongs to EX(m), and this is in contradiction with 2).QED.

Non union and branching nets.
Suppose ϕ_k is presented to an IIM γ, and that ϕ_k(a) = b. Let h be the last hypothesis of γ on ϕ_k.In the case ϕ_k(x) is equal to ϕ_h(x) for every x excepting a : i.e. ϕ_h(a) = c, and c $\neq$ b,we say that m is precisely incorrect on a.
Definition $EX^{=1}$ = {S : $\exists$M S $\subset$ EX(M) and M is precisely incorrect on exactly one number}. Such an error can always be refuted and corrected in the limit. So we have :

$$EX^{=1} = EX$$

Non-union theorem (Blum & Blum, see 3) : A $\in$EX & B$\in$EX does not imply A$\cup$B$\in$EX.

Proof. Let S_1 be the set of functions which are computed by machines which reproduce themselves in a precisely incorrect way.on the flag 0. S_1 = {f : $\phi_{f(0)}$ $=^{=1}$ f}. Let S = {f : $\phi_{f(0)}$ = f}, α witness that S_1 belongs to $EX^{=1}$ = EX, and α witness also that S belongs to EX. So both S_1 and S belongs to EX, but the union $S_1 \cup S = S^{0v1}$ = does not belong to EX.(preceding theorem). That proves the non-union theorem of Blum & Blum.: union of members of EX does not necessarily belongs to EX. The "v" in 0v1 is a necessarily not constructive "v". QED. We have a typical situation where

$\vdash$ (p v q) does'nt imply $\vdash$ p *or* $\vdash$ q

If we want take advantage of such non constructive enlargements We must allow branchings in the net, by adding, for exemple, in (3) above the line :

else if z=INDUCE op IND(ϕ_x(j)) i $\geq$j). (4)

where INDUCE is a new flag, and IND is what I call a semi-realizer, that is essentially a program which, given some

[1] Communication to Case and Smith.(see 9).

specification, built a collection of programs -capable of running in parallele- one of which meets the specification although there is no possible algorithmic choice of that one.

7. DREAMING MACHINES

In his little book *DREAMING* Malcom opposed Descartes's view[1] on dreams with his own, young-Wittgensteinien, verificationist conception.
By true judgment he means verifiable one, and he argues that true judgment are communicable. In particular he argues against the meaningfullness of Descartes's doubt between dream and reality.(see 7, 11).
In this setting Malcom's conception can be compare with Brouwer's : where true (mathematical) judgment are those intuitively provable. Malcom defend a kind of restricted solipsism with respect of the dreaming person.
To know if one is dreaming or not is a problem of personal knowledge
There are sufficiently affinities between the concept of truth and the concept of proof which invite an attempt for defining knowledge with them. The personal aspect beeing (hopefully) captured by (double) diagonalisation (or personal identity).
We agree with a common opinion that knowledge's, representation in modal logic is given by S4 :

AXIOMS :	⊡p->p
	⊡p->⊡⊡p
	⊡(p->q)->⊡p->⊡q
RULES	p and p->q entails q
	p entails ⊡p

8. FROM G TO G*

Let fix M be the initial machine m_0 of our self-referential nets.

First attempt : ⊡p is , p ?
Do we have , p->p? (for any p beeing a representation of a sentence in M's language). The provability of M is PA's provability. If we believe in the soundness of PA we certainly believe that , p->p is true.
Let us look at the rule p entails , p.
We have , p->p.If the rule is valid, we should have , (, p->p).
But Löb shows that ⊢ , p->p entails ⊢ , p, and it has beenshow that the proof is formalisable in PA, so we have

, (, p->p)-> , p (Löb's formula, see 21)

And thus, if we have , (, p->p) for any p, we obtain in particular , (, FALSE->FALSE), then from Löb's formula we get , FALSE,and with , p->p,we get FALSE.
We cannot have both the axiom , p->pand the rule "p entails ⊡p" for , .
(see Montague and Thomason for similar result).

Second attempt : what is knownable is what is inferable in the limit. Here truth is seen as a limit platonistic concept.
In this case we follow the sequence m_i.
I need Solovay's theorem (Solovay) :

the following modal system G:

AXIOMS :	, p-> , , p
	, (p->q)-> , p-> , q
	, (, p->p)-> , p
RULES	p and p->q entails q
	p entails , p

is a sound, decidable, and complete formalisation of provability about provability of PA, (PA's description of his own provability's ability) and this one, known as G* :

AXIOMS :	, p->p
	All the theorems of G
RULES	p and p->q entails q

is a sound, decidable, and complete formalisation of truth about provability by PA.

Now even an idiotic m_0 can be used as a mechanist counter-example contra-malcom, for he would distinguished communicable knowledge from non-

[1] cf also Caillois 1956.(see 7)

communicable one (the decidable complementary of G in G*). If either by RL, or by inductive inference he reflect an undecidable sentence then he gives birth to a new machines for which G and may be G* can still be sound and on some ways, even complete.

With G we loss ,p->p.With G* we loss "p entails ,p".In the platonistic realm the machine can infer non communicable truth, and even communicate them with some precautions, using an "?" for exemple. But how to reconcile ,p->p and "p entails ,p", i.e. how to isolate immediate, actual and solipsist-like terrestrial knowledge ?

9. ARITHMETICAL NECESSITY

An (ad hoc ?) attempt : personal identity, as defined here, is really identity based on the shape -the form(al)-, which permit to the machine to manifest itself relatively to an universal environment. This is certainly a counter-intuitive view of personal identity.(see for paradoxes). Is it possible to capture the *intuitive* or *absolute* identity" ? We have identify finitely convincing proposition whith platonistically correct proof by a machine in the net. Is there a corresponding concept for *intuitive* or *informal* proof in the literature ? The use of Kleene-like realizability for the inner building of the (local) past (defined as self-organized collection of rememorable personal experiences -these beeing with the universal ability build in the recursion matrix- suggest the use of intuitionistic concept of proof. Thank to Gödel's presomptions followed by the proof of McKinsey and Tarski, (see 14, 26, 13)this lead naturaly toward S4, but we loose connection with the Platonistic G(*)s, and, I have a feeling of ad-hocness about it .

third attempt : ⊡p is ,p&p

We know that ,p and ,p&p are platonisticaly (i.e. G*) equivalent, but not provably so (i.e. G) as we can deduce from Löb's theorem, moreover ⊡p will obey S4. But more can be said : the following system, where a formula due to Grzegorczyk (see 18) is added :

AXIOMS :	,p->p
	,p->,,p
	,(p->q)->,p->,q
	,(,(p->,p)->p)->p
RULES	p and p->q entails q
	p entails ,p

is a sound, decidable, and complete formalisation of the provable true-and-provable sentence in PA, thus in m_0 (so that it works in the actual, we are not obliged to wait for the limit).That system is known as S4Grz.
Astonishingly enough, it is also a sound, decidable, and complete formalisation of the true true-and-provable sentence in PA. This follows from independant work of Boolos and Goldblatt (see 4, 5, 17). So there is, after all, a self-referentially based reconciliation of truth and provability. Using Grzegorczyk's extension of Gödel's link between S4 and Intuitionistic logic, Goldblatt describes an arithmetical interpretation of IL in PA, and proves a similar double completeness result.

Here is a summary of the works of Solovay, Boolos and Goldblatt :

Provable by M		True for M	
G	->	G*	LN
IL	->	IL	NP
S4Grz	->	S4Grz	NP

LN = Loss of necessitation, NP = necessitation preserved, and necessitation is the common name for the rule p entails ,p.

11. APPLICATIONS

We have seen that Idiotic self-inductive-inference machine <G,G*>, i.e.machines which belongs to a self-referential nets where G, G* has been put in the recursion matrix at the start -so that correct self-inference are *instinctive*-, are

doubtfully reasonable as models for the mind, but can nevertheless be useful for building counter-exemples to some arguments in philosophy of Mind.
Other exemples : Kripke-like argument against the identity thesis.(see 20). Refutation of Baker arguments against functionalism (see 2). In both case their arguments remains valid within a purely extensional view of identity and function(alism). It is a confusion between

p& , p and , p

or at a more primitive level, it is a confusion between :

necessarily $\phi_e()$ = e. and necessarily e=e.

The basic philosophical motivation for the present approach are described in the papers 22, 23, 24).
Other result can be extract from the present analysis. If we accept the darwinian-like idea of survival of the *sound*-machine: then the self-referential nets produce, at term (in the limit with the theoretical computer science concept of limit), machines describable with the help of G, G*, S4Grz, or extensions. More can be said : communication of *non-communicable* statements enhance not only the scope of possible dialogues between machines and environment, but permit to speed-up computations relatively to that environment. This speed-up phenomena has been also presumed by Gödel (see 1, 15, 31). Mackay has made an analysis of relative *freedom* -of a machine relatively to an other- in term of relative computational efficiency. That analysis, transposed in our setting, would entail emergence of *freedom* (relatively to universal environment) in the limit (see 33).

Bibliography

1. ARBIB M.A., ***Brains, Machines, and Mathematics***, Springer-Verlag, 1987. (1er ed. McGraw-Hill, 1964).
2. BAKER L. R. *A Farewell to Functionalism* Philosophical Studies, 48, 1985. Reprinted in Silvers (ed) 1989.
3. BLUM L. & BLUM M. *Toward a Mathematical Theory of Inductive Inference.* Information and Control 28, 1975.pp. 125-155.
4. BOOLOS G., *Provability, Truth, and Modal Logic*, Journal of Philosophical Logic, 9, pp. 1-7, 1980.
5. BOOLOS G., *On Systems of Modal Logic with Provability Interpretations*, Theoria, 46, 1, pp. 7-18, 1980.(b).
6. BUSCHSBAUM R., ***Animals without Backbones : 1***, Penguin Books, 1938. (Third Edition : with BUSCHSBAUM M., PEARSE J. and PEARSE V., The University of Chicago Press, Chicago and London, 1987).
7. CAILLOIS R. ***L'incertitude qui vient des rêves***. Gallimard, 1956.
8. CASE J., *Periodicity in Generations of Automata*, Mathematical Systems Theory. Vol. 8, n° 1. Springer Verlag, NY, pp. 15-32, 1974.
9. CASE J. & SMITH C. *Comparison of Identification Criteria for Machine Inductive Inference.* In Theoretical Computer Science 25, 1983.pp 193-220.
10. DAVIS M. (ed.). ***The Undecidable***. Raven Press, Hewlett, New York, 1965.
11. DESCARTES R. ***Oeuvres et Lettres***. Bibliothèque de la Pléiade. Gallimard, 1953.
12. FEFERMAN S, *Transfinite Recursive Progressions of Axiomatic Theories*, Journal of Symbolic Logic, Vol 27, N° 3, 1962, pp. 259-316.
13. FITTING M. C., ***Intuitionistic Logic, Model Theory and Forcing***, North-Holland Publishing Company, Amsterdam 1969.
14. GODEL K., *Eine Interpretation des Intuitionistischen Aussagenkalküls*, Ergebnisse eines Mathematischen Kolloquiums, Vol 4, pp. 39-40, 1933.
15. GÖDEL K. *On the Length of Proofs*. In Davis [1965]
16. GOLD E.M. *Language Identification in the Limit*. Information & Control 10, 1967.pp. 447-474.
17. GOLDBLATT R., *Arithmetical Necessity, Provability and Intuitionistic Logic*, Theoria, Vol 44, pp. 38-46, 1978.
18. GRZEGORCZYK A., *Some relational systems and the associated topological spaces*, Fundamenta Mathematicae, LX pp. 223-231, 1967.
19. KLEENE S.C. ***Introduction to Metamathematics***. P. Van Nortrand Comp. Inc., 1952.
20. KRIPKE S. ***La logique des noms propres***. Les éditions de minuit, 1982,

(traduit de l'américain : ***Naming and Necessity***, 1972)

21.LOB M. H., *Solution of a Problem of Leon Henkin*, Journal of Symbolic Logic, 20, pp. 115-118, 1955.

22.MARCHAL B., *Informatique théorique et philosophie de l'esprit.* Actes du 3ème colloque international de l'ARC, Toulouse 1988.

23.MARCHAL B., *Des Fondements Théoriques pour l'Intelligence Artificielle et la Philosophie de l'Esprit*, Revue Internationale de Philosophie, 1, n° 172, pp 104-117, 1990.

24.MARCHAL B., *Mechanism and Personal Identity*, WOCFAI 91.

25.MOORE C. M., *Unpredictability and Undecidability in Dynamical Systems*, Physical Review Letters, V. 64, N° 20, pp.2354-2357, 1980.

26.McKINSEY J. C. C. & TARSKI A., *Some Theorems about the Sentential Calculi of Lewis and Heyting*, Journal of Symbolic Logic, 13, pp. 1-15, 1948.

27.MYHILL J., *Abstract Theory of Self-Reproduction*, in Views on General Systems Theory, M.D. Mesarovic, ed. Wiley NY, 1964, pp 106-118.

28.PUTNAM H. *Probability and confirmation* The Voice of America, Forum Philosophy of Science, 10 (U.S. Information agency, 1963). Reprinted in ***Mathematics, Matter, and Method***. Cambridge University Press.Cambridge 1975.

29.ROGERS H. ***Theory of Recursive Functions and Effective Computability.*** McGraw-Hill, 1967. (2ed, MIT Press, Cambridge, Massachusetts 1987)

30.ROGERS H., *Gödel Numbering of the Partial Recursive Functions*, Journal of Symbolic Logic, 23, pp. 331-341, 1958.

31.ROYER J.S., *Two Recursion Theoretic Characterizations of Proof Speed-ups*, The Journal of Symbolic Logic, 54, N° 2, 1989.

32.SOLOVAY R., *Provability Interpretations of Modal Logic*, Israel Journal of Mathematics 25, 1976, pp. 287-304.

33.WATKINS J.W.N. *Freedom and Predictability: an Amendment to Mackay*. Brit. J. Phil. Sci. 22 (1971), 263-274.