LABORATOIRE D'INFORMATIQUE DE NANTES ATLANTIQUE

# Apprendre, c'est savoir oublier

Colin de la Higuera, LINA

7 septembre 2015

UNIVERSITÉ DE NANTES

ECOLE DES MINES DE NANTES

# Remarques préliminaires

- Exposé « différent »

  (académique, selon Jeff)

- La question jusqu'à présent était

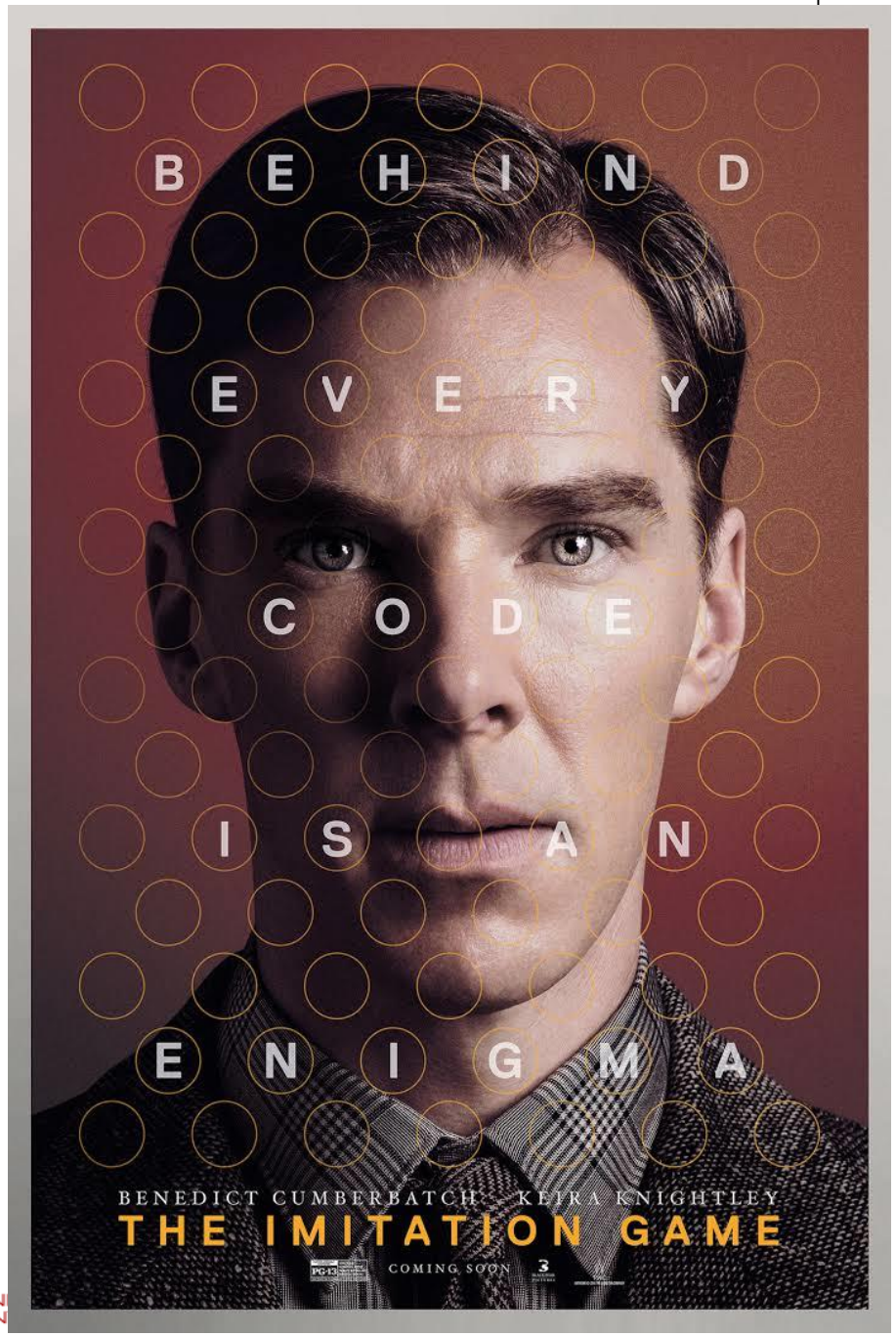
  Comment fait-on ?

- La question du jour est

  Que fait-on ?

# Un peu d'histoire

- Turing, 1912-1954
- La machine de Turing
- Une machine qui apprend

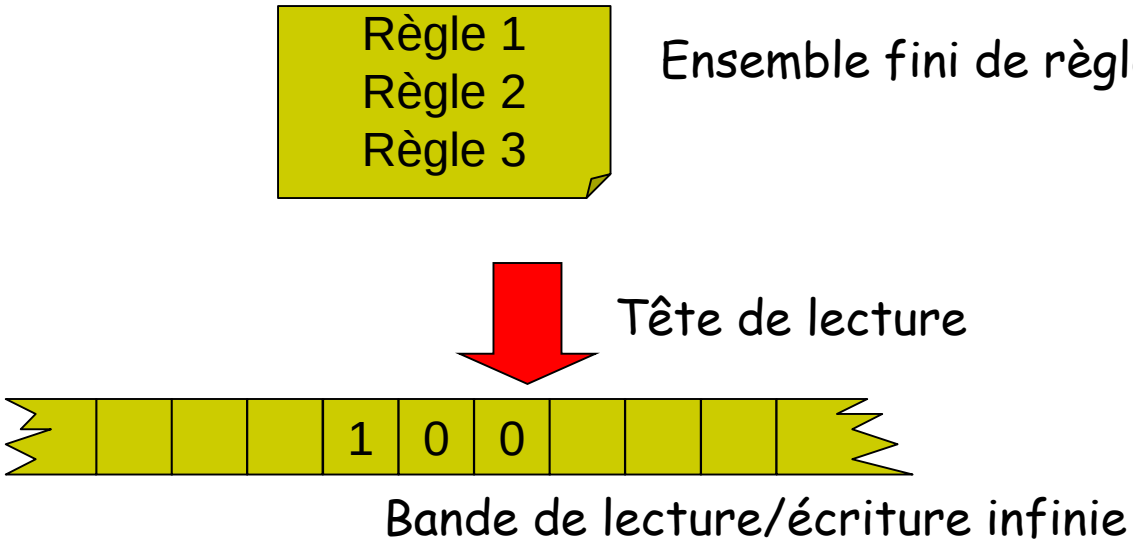


# 2 articles fondamentaux

Alan Turing, *On Computable Numbers with an Application to the Entscheidungsproblem*, *Proc. London Math. Soc.*, 2e série, vol. 42, 1937, p. 230-265

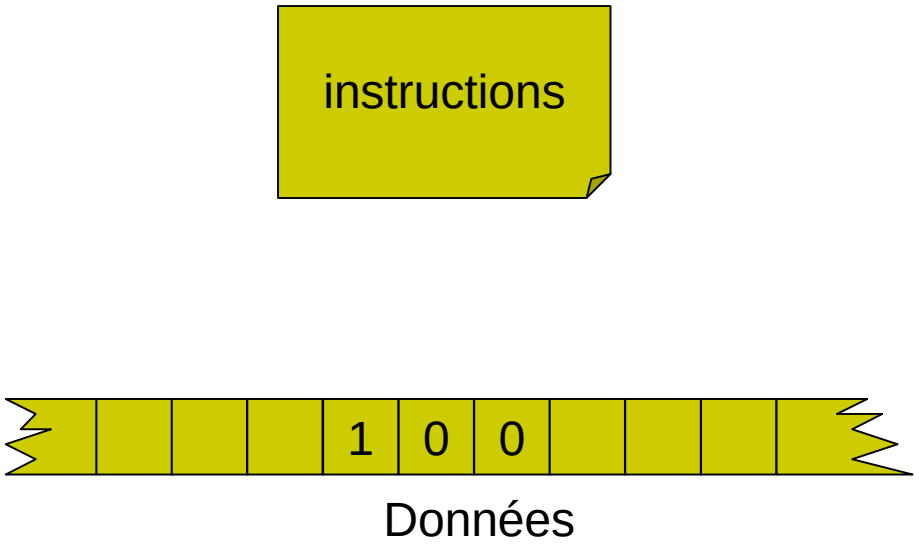
Alan Turing, Computing machinery and intelligence, *Mind*, Oxford University Press, vol. 59, no 236, octobre 1950

# Les objets de la machine de Turing

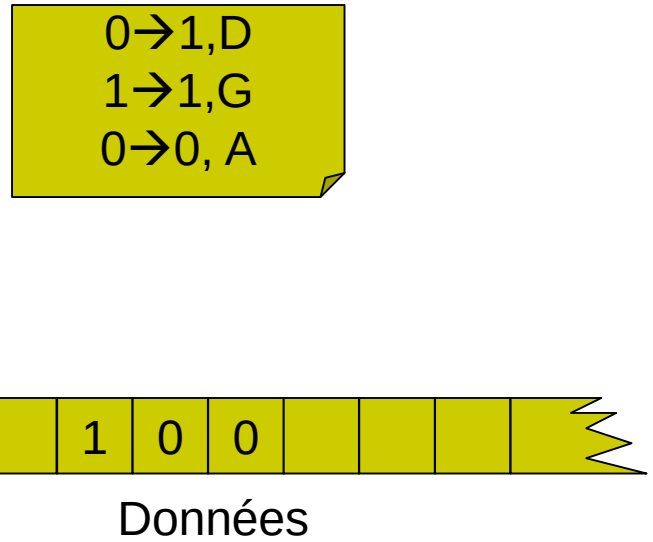



Petite simplification: il y a aussi un ensemble fini d'états

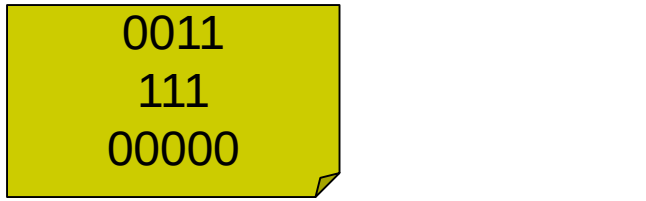
# Que fait une machine de Turing?

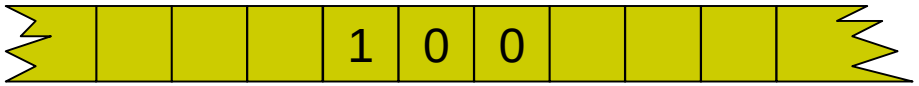


# Que fait une machine de Turing?



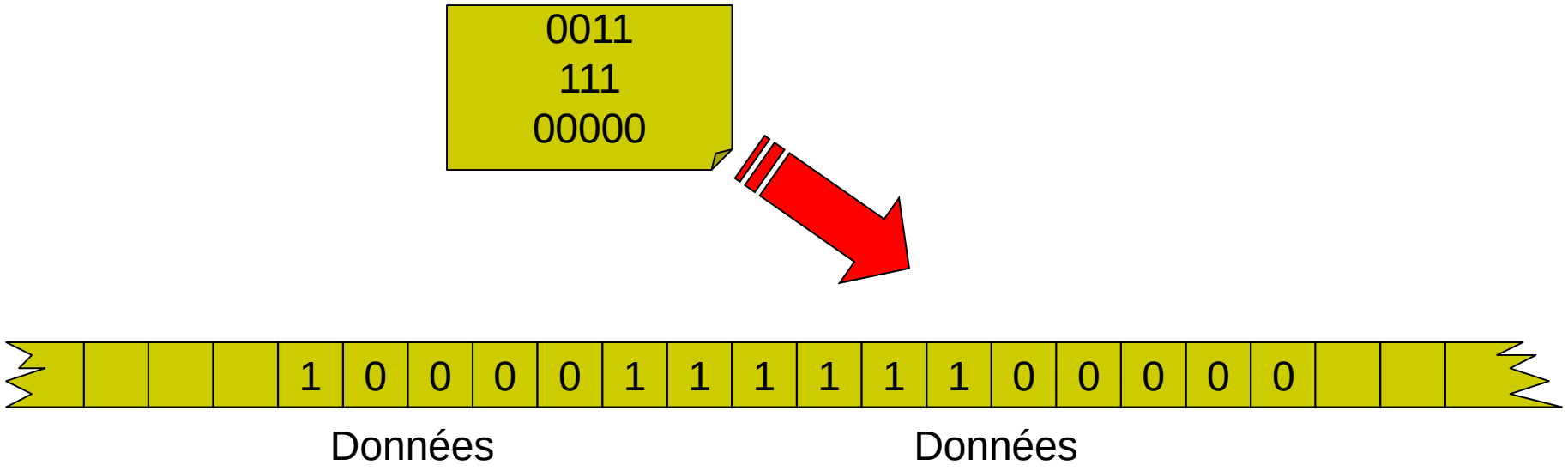
# Que fait une machine de Turing?



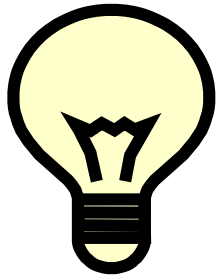




Données

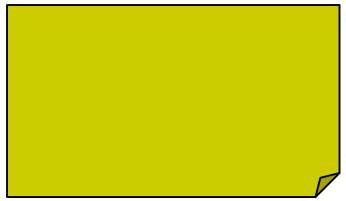
# Que fait une machine de Turing?



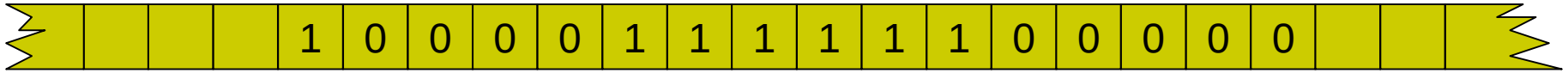
# Que fait une machine de Turing?



- La Machine de Turing universelle



Contient le code permettant de séparer les données du programme et de simuler le comportement du programme sur les données



# Une idée évidente ?

En 1956, Howard Aiken écrivait :

If it should turn out that the basic logics of a machine designed for the numerical solution of differential equations coincide with the logics of a machine intended to make bills for a department store, I would regard this as the **most amazing coincidence** that I have ever encountered.

http://www.turing.org.uk/scrapbook/computer.html

# Et en 1948…

- Si on peut coder n'importe quel programme en tant que donnée et le passer à une machine universelle qui se contente de l'interpréter…
- …pourquoi ne pas passer ce programme à une machine plus compliquée qui va le transformer ?

Alan Turing dans son article de 1950

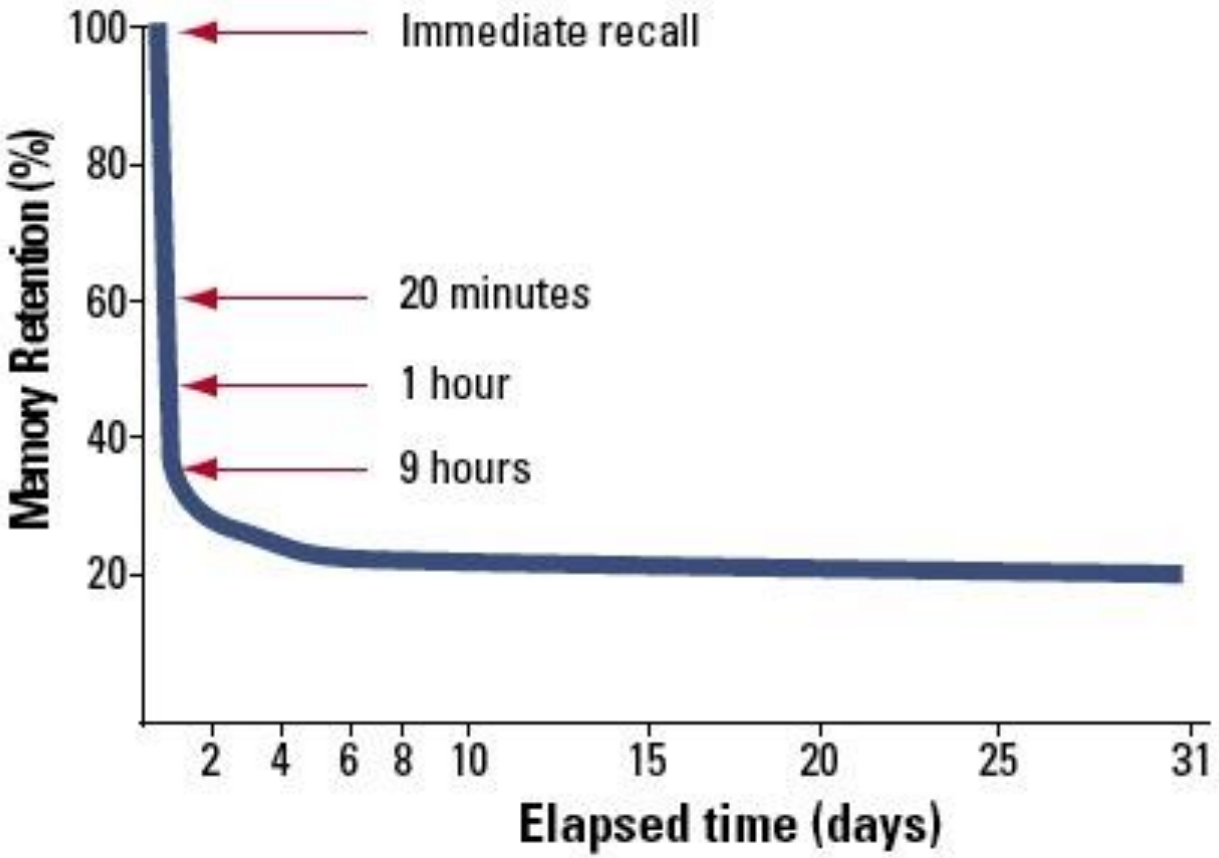
# Alan Turing

Computer machinery and Intelligence 1950

- Instead of producing a programme to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education, one would obtain the adult brain.
- An important feature of a **learning machine** is that its teacher will very often be very largely ignorant of quite what is going on inside, although he may be able to some extent to predict his pupil's behaviour.
- It is probably wise to include a random element in a learning machine *[pour accélérer la recherche et éviter que la machine ne soit déterminée par son passé]*

# LE POINT DE VUE GÉNÉRAL (NON INFORMATIQUE)

**FIGURE 1.**
**The forgetting curve**




The "forgetting curve" was developed by Hermann Ebbinghaus in 1885. Ebbinghaus memorized a series of nonsense syllables and then tested his memory of them at various periods ranging from 20 minutes to 31 days. This simple but landmark research project was the first to demonstrate that there is an exponential loss of memory unless information is reinforced.

Stahl SM, Davis RL, Kim D, et al. *CNS Spectr*. Vol 15, No 8. 2010.

# APPRENDRE
# =
# NE PAS OUBLIER

# Oublier= le résultat d'un mauvais apprentissage

- **Procedural skills: From learning to forgetting**

***Kim, Jong Wook*, Ph.D., THE PENNSYLVANIA STATE UNIVERSITY**

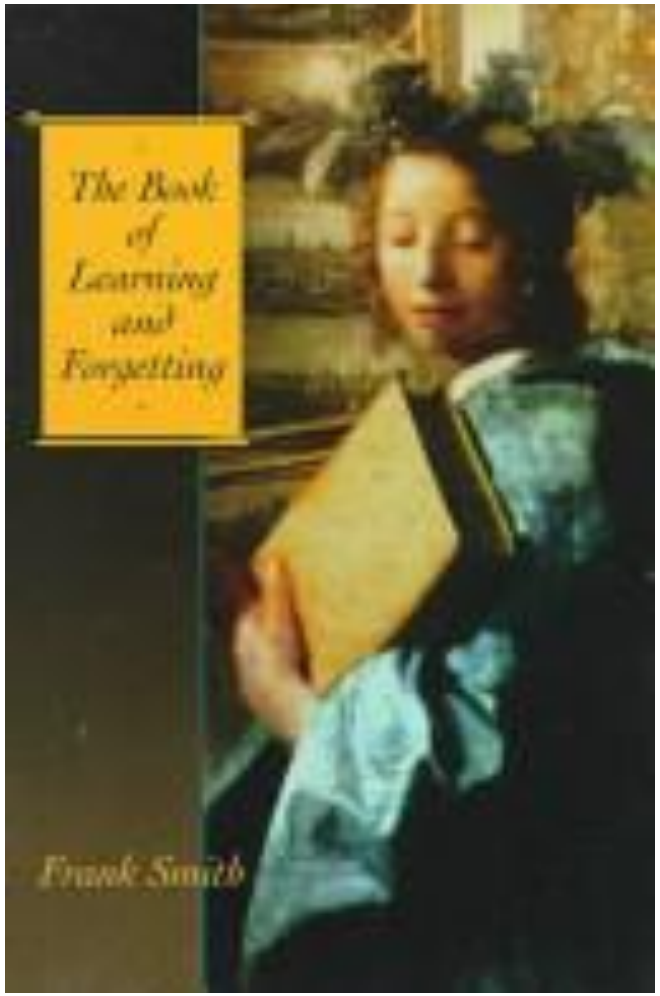
In general, using a forgetting model with a decay function (e.g., an exponential function) requires a known value of the time of the break. The hyperbolic function can represent negative learning when $r$ is less than 0. Based on this property, Nembhard and Uzumeri (2000b) proposed a model of learning and forgetting. They state that a practical forgetting model should be able to capture multiple breaks at irregular intervals. They modified the hyperbolic function of the learning curve by incorporating experiential learning ($R$), called *recency*, as shown in Equation 2.3. $x$ in the denominator indicates each unit of cumulative production, and $t_x - t_0$ indicates the elapsed time for the unit $x$ that is the difference between the time stamps of the completion of the current unit and the start of the first unit ($t_0$). Thus, Equation 2.3 represents the *recency* measure, the ratio of the average elapsed time to the elapsed time of the most recent unit produced.

$$R_x = \frac{\sum_{i=1}^{x}(t - t_0)}{x(t_x - t_0)}$$

**Equation 2.3.**

Frank Smith

Teachers College Press, 1998 -



# http://bjorklab.psych.ucla.edu/index.html

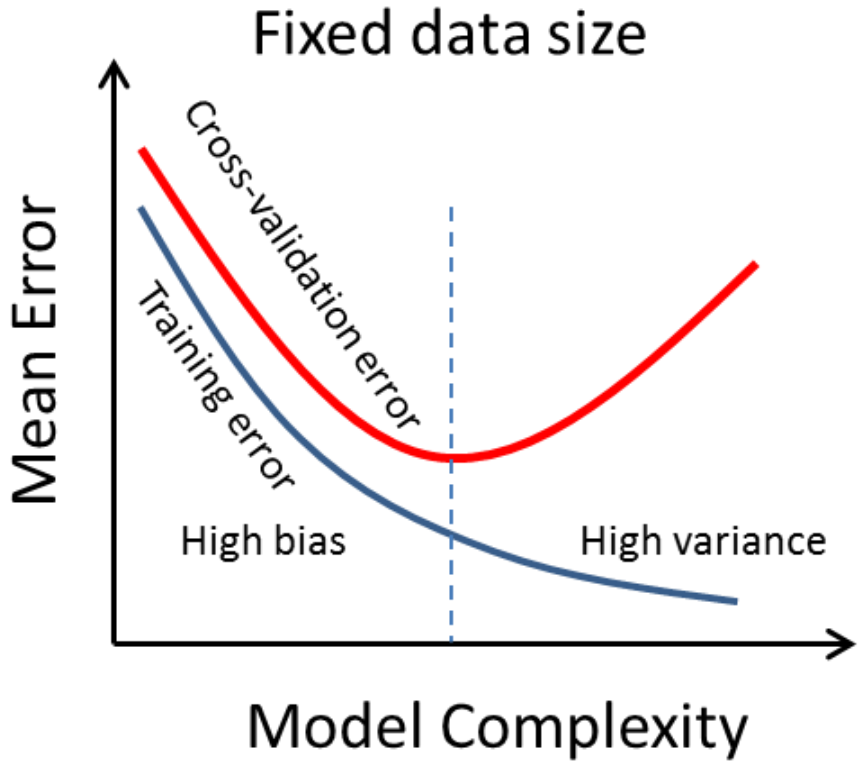
- People often view forgetting as an error in an otherwise functional memory system; that is, forgetting appears to be a nuisance in our daily activities. Yet forgetting is adaptive in many circumstances. For example, if you park your car in the same lot at work each day, you must inhibit the memory of where you parked yesterday (and every day before that!) to find your car today.
- …goal-directed forgetting, that is, situations in which forgetting serves some implicit or explicit personal need (E. L. Bjork, R. A. Bjork, & Anderson, 1998)

# Borges, Ficciones, 1944

- **Funes el memorioso**


# REVENONS AU MACHINE LEARNING

# La courbe qui dérange, celle de la sur-généralisation



# Apprendre=compression (avec perte)

- Argument Ockham
- Théorème des 4 allemands

**Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1987). *Occam's razor*. Information processing letters, 24(6), 377-380.**

Let $C$ and $H$ be concept classes containing target concepts and hypotheses respectively and let sample set $S$ contain $m$ samples each containing $n$ bits. Then, for constants $\alpha \geq 0$ and $0 \leq \beta < 1$, a learning algorithm $L$ is an **$(\alpha,\beta)$-Occam algorithm** for $C$ using $H$ if, given $S$ labeled according to $c$ in $C$, $L$ outputs a hypothesis $h \in H$ such that

- $h$ is consistent with $c$ on $S$ (that is, $h(x)=c(x)\ \forall x \in S$)
- and $size(h) \leq n \cdot size(c)^{\alpha} m^{\beta}$

Such an algorithm $L$ is called an efficient $(\alpha,\beta)$-Occam algorithm if it runs in time polynomial in $n$, $m$ and $size(c)$.

# Et le théorème

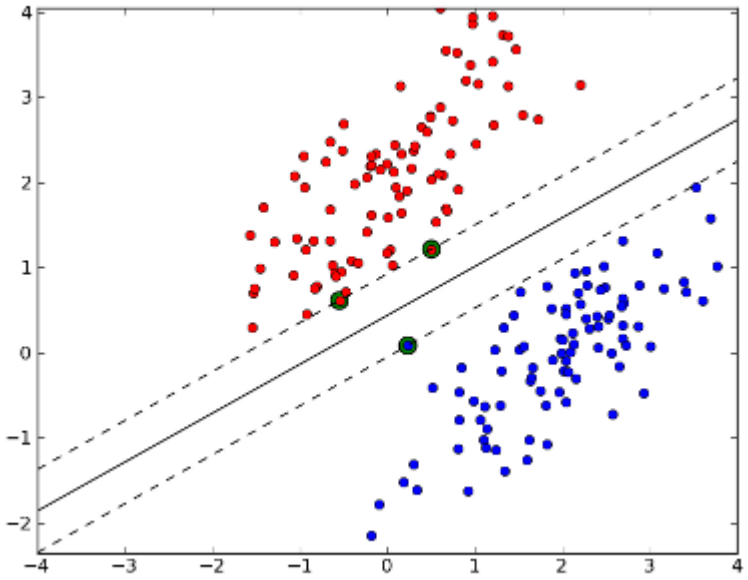
- Any efficient Occam algorithm is also an efficient PAC learning algorithm.

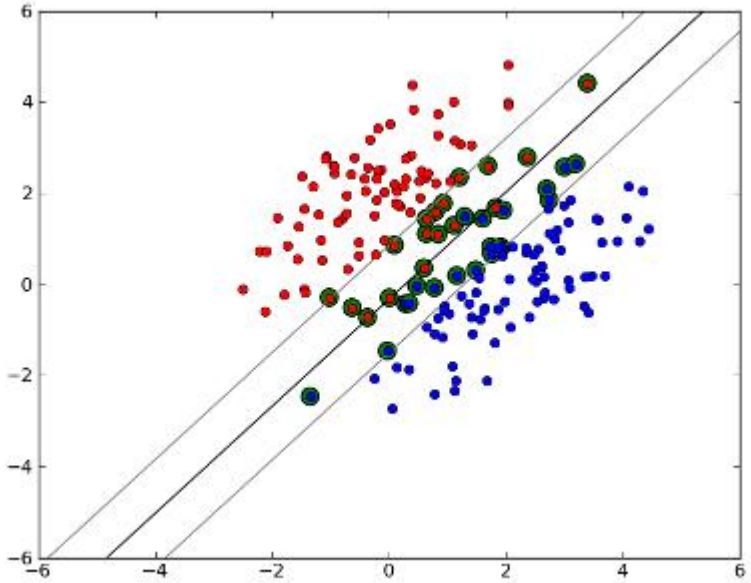
- PAC : Les Valiant, Prix Turing 2011

# APPRENDRE PASSE PAR OUBLIER

# Exemple 1 : le SVM

- Dans un SVM, il s'agit (seulement ?) de trouver les vecteurs de support




http://www.mblondel.org/journal/2010/09/19/support-vector-machines-in-python/

A droite, le cas d'un soft margin où 36 vecteurs sur 120 sont retenus

# Exemple 2 le perceptron

- Dans le perceptron les éléments « évidents » n'interviennent pas
- La version duale du perceptron consiste à trouver les poids des différents exemples évidents est 0

Algorithm 7.2, p.209

## Perceptron training in dual form

**Algorithm** DualPerceptron($D$) – perceptron training in dual form.

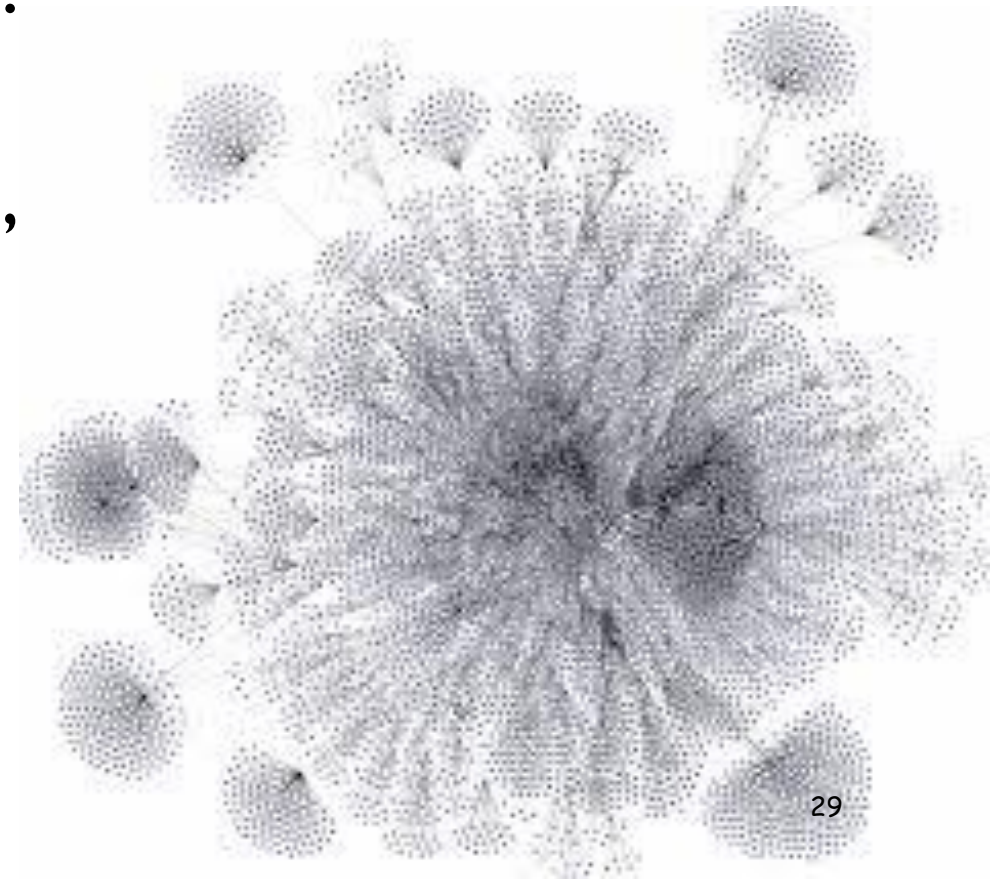
**Input** : labelled training data $D$ in homogeneous coordinates.

**Output** : coefficients $\alpha_i$ defining weight vector $\mathbf{w} = \sum_{i=1}^{|D|} \alpha_i y_i \mathbf{x}_i$.

```
1  α_i ← 0 for 1 ≤ i ≤ |D|;
2  converged←false;
3  while converged = false do
4      converged←true;
5      for i = 1 to |D| do
6          if y_i Σ_{j=1}^{|D|} α_j y_j x_i · x_j ≤ 0 then
7              α_i ← α_i + 1;
8              converged←false;
9          end
10     end
11 end
```

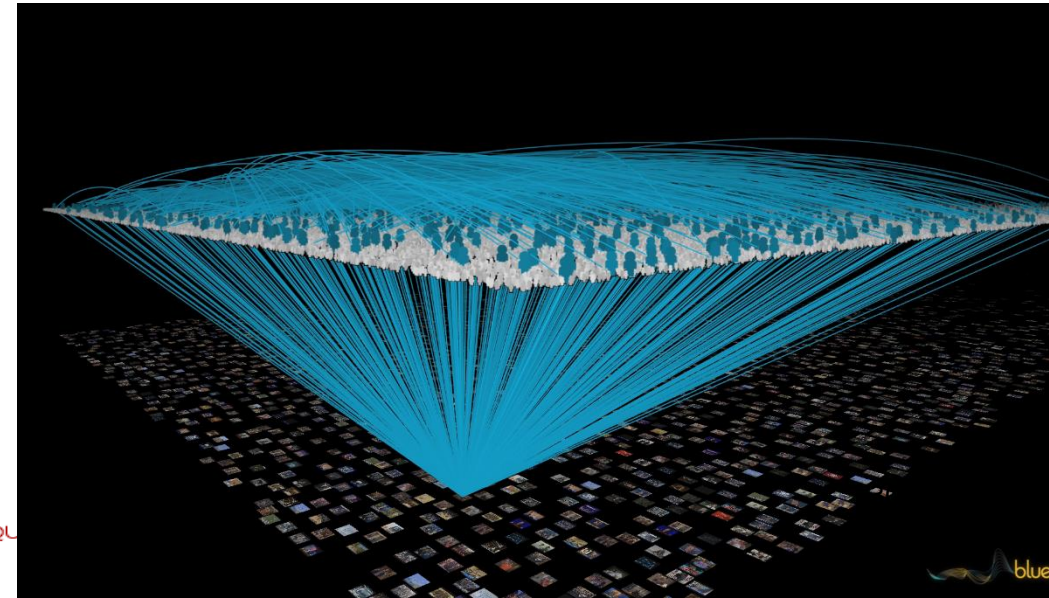
# Exemple 3 : les arbres de décision

- Le processus de sélection des variables s'apparente à répondre à la question : « qu'est-ce que je peux oublier ? »
- Il ne s'agit pas ici d'oublier des données, mais des attributs



# Exemple 4 : les flux de données

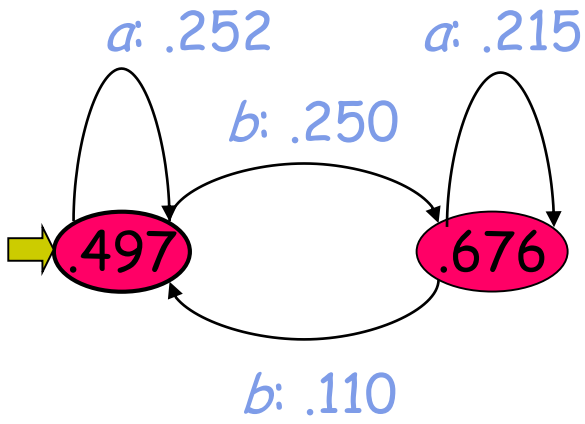
- La question d'apprentissage est la suivante :
  - J'ai vu plein de données, mais pour pouvoir continuer à recevoir de l'information, il faut que je récupère de l'espace
  - Certes, je peux remplacer mes données par une fonction apprise à partir de celle-ci, mais (quel que soit l'algorithme d'apprentissage), je dois aussi garder de l'information sur les données elles-mêmes
  - Donc choisir lesquelles j'oublie



# Exemple 5 : l'inférence grammaticale

- Données : du texte (des chaînes de caractère)
- But : construire un automate ou une grammaire représentant le langage

$S$={λ(490), *a*(128), *b*(170), *aa*(31), *ab*(42), *ba*(38), *bb*(14), *aaa*(8), *aab*(10), *aba*(10), *abb*(4), *baa*(9), *bab*(4), *bba*(3), *bbb*(6), *aaaa*(2), *aaab*(2), *aaba*(3), *aabb*(2), *abaa*(2), *abab*(2), *abba*(2), *abbb*(1), *baaa*(2), *baab*(2), *baba*(1), *babb*(1), *bbaa*(1), *bbab*(1), *bbba*(1), *aaaaa*(1), *aaaab*(1), *aaaba*(1), *aabaa*(1), *aabab*(1), *aabba*(1), *abbaa*(1), *abbab*(1)}



# Exemple 6 : le boosting

- L'algorithme Adaboost, à chaque étape, change les poids des exemples :

8 $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$ ; // confidence for this model

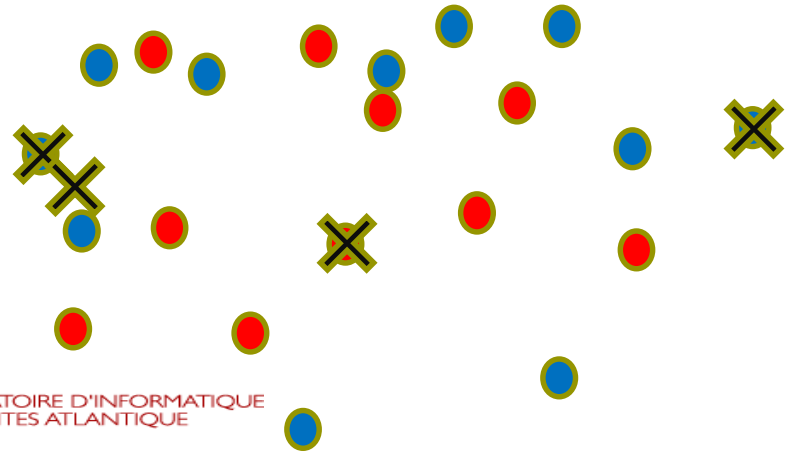
9 $w_{(t+1)i} \leftarrow \frac{w_{ti}}{2\epsilon_t}$ for misclassified instances $x_i \in D$ ; // increase weight

10 $w_{(t+1)j} \leftarrow \frac{w_{tj}}{2(1-\epsilon_t)}$ for correctly classified instances $x_j \in D$ ; // decrease

# TOUT N'EST PAS SI ÉVIDENT…

# Les k plus proches voisins

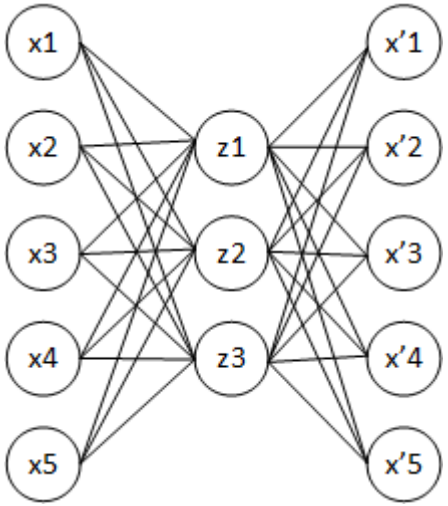
- L'idée de l'algorithme est de garder tous les exemples puisque ce sont eux qui classent !
- Et pourtant
  - Y a-t-il apprentissage ?
  - Un raffinement typique est d'éliminer les exemples inutiles : un exemple est inutile si, en l'enlevant, la classification reste identique



# Le deep et les embeddings…

*Building High-level Features Using Large Scale Unsupervised Learning*

- 10 million 200x200 pixel images
- 1 billion trainable parameters



# Clustering?

- K-means, k-medians ?
- Clustering hiérarchique ?

# Frequent itemsets?

- Ici aussi, on peut avoir plus de résultats que de données.
- Pourtant, le « support » peut conduire à éliminer des items trop peu fréquents

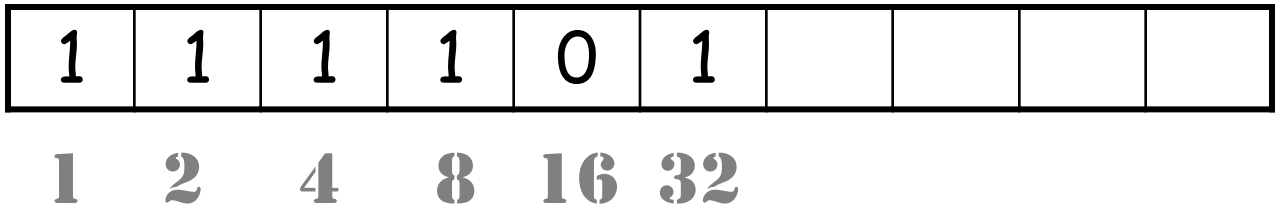
# Conclusion

- Il y a oubli associé à apprentissage dans de nombreux contextes
- L'oubli n'est pas un effet de bord
- Ce n'est pas non plus un moteur
- Questions
  - Y a-t-il des algorithmes qui utilisent l'oubli comme moteur ?
  - Une raison d'oublier (pour les $\psi$) semble être la capacité de mieux apprendre la 2$^{e}$ fois. A-t-on ça en ML ?

# ANNEXE

# Codage binaire inverse

- 47= 4*10+7*1
- 47=1*32+0*16+1*8+1*4+1*2+1*1
- 47=101111$_2$



# Une Machine qui pense
# Programme successeur

Les règles :

Si on lit 1, écrire 0, aller à droite, continuer.

Si on lit 0, écrire 1, ARRET !

Si on lit □, écrire 1, ARRET !

Essayons sur une bande contenant le codage binaire inverse de 47 :

# Une Machine qui pense
# Programme successeur

Si on lit 1, écrire 0, aller à droite, continuer.

Si on lit 0, écrire 1, ARRET !

Si on lit □, écrire 1, ARRET !

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 1 | | | | |

# Une Machine qui pense
# Programme successeur

Si on lit 1, écrire 0, aller à droite, continuer.

Si on lit 0, écrire 1, ARRET !

Si on lit □, écrire 1, ARRET !

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 1 | | | | |

# Une Machine qui pense
# Programme successeur

Si on lit 1, écrire 0, aller à droite, continuer.

Si on lit 0, écrire 1, ARRET !

Si on lit □, écrire 1, ARRET !

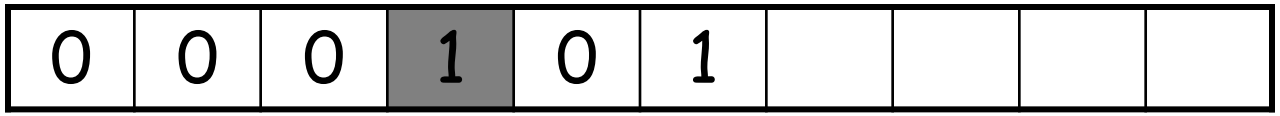
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 1 | | | | |

# Une Machine qui pense
# Programme successeur

Si on lit 1, écrire 0, aller à droite, continuer.

Si on lit 0, écrire 1, ARRET !

Si on lit □, écrire 1, ARRET !

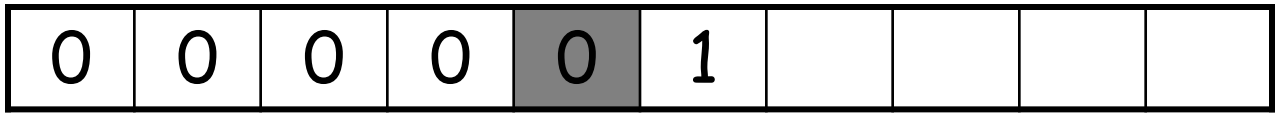


# Une Machine qui pense
# Programme successeur

Si on lit 1, écrire 0, aller à droite, continuer.

Si on lit 0, écrire 1, ARRET !

Si on lit □, écrire 1, ARRET !



# Une Machine qui pense
# Programme successeur

Si on lit 1, écrire 0, aller à droite, continuer.

Si on lit 0, écrire 1, ARRET !

Si on lit □, écrire 1, ARRET !

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | | | | |

Et sur la bande on lit bien (à l'envers) 48

# Une Machine qui pense
# Programme successeur

Donc le résultat du successeur sur 111101 est 000011 qui est la représentation binaire inverse de 48

De même le successeur de 127 devrait être 128

# Une Machine qui pense
# Programme successeur

Si on lit 1, écrire 0, aller à droite, continuer.

Si on lit 0, écrire 1, ARRET !

Si on lit □, écrire 1, ARRET !

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | |

# Une Machine qui pense
# Programme successeur

Si on lit 1, écrire 0, aller à droite, continuer.

Si on lit 0, écrire 1, ARRET !

Si on lit □, écrire 1, ARRET !

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | | | |

# Une Machine qui pense
# Programme successeur

Si on lit 1, écrire 0, aller à droite, continuer.

Si on lit 0, écrire 1, ARRET !

Si on lit □, écrire 1, ARRET !

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | | | |

# Une Machine qui pense
# Programme successeur

Si on lit 1, écrire 0, aller à droite, continuer.

Si on lit 0, écrire 1, ARRET !

Si on lit □, écrire 1, ARRET !

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | | | |

# Une Machine qui pense
# Programme successeur

Si on lit 1, écrire 0, aller à droite, continuer.

Si on lit 0, écrire 1, ARRET !

Si on lit □, écrire 1, ARRET !

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | | | |

# Une Machine qui pense
# Programme successeur

Si on lit 1, écrire 0, aller à droite, continuer.

Si on lit 0, écrire 1, ARRET !

Si on lit □, écrire 1, ARRET !

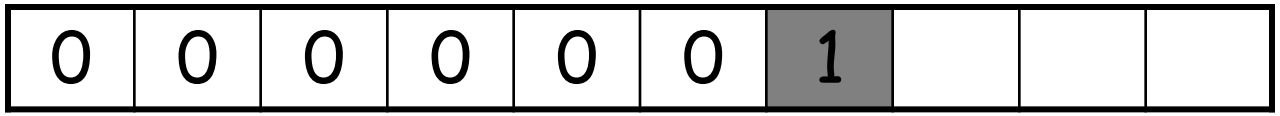
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | | | |

# Une Machine qui pense
# Programme successeur

Si on lit 1, écrire 0, aller à droite, continuer.

Si on lit 0, écrire 1, ARRET !

Si on lit □, écrire 1, ARRET !

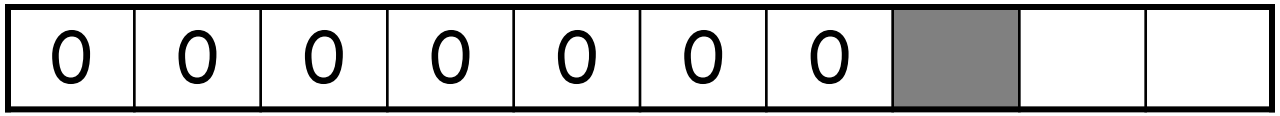


# Une Machine qui pense
# Programme successeur

Si on lit 1, écrire 0, aller à droite, continuer.

Si on lit 0, écrire 1, ARRET !

Si on lit □, écrire 1, ARRET !



# Une Machine qui pense
# Programme successeur

Si on lit 1, écrire 0, aller à droite, continuer.

Si on lit 0, écrire 1, ARRET !

Si on lit □, écrire 1, ARRET !

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | |