Le travail d'Emilio à l'usine

I/ Programmation de la machine de Post-Wang

Emilio, lorsqu'il arrive devant l'alignement de cases qui lui sert de lieu de travail, se voit remettre une liste de choses à faire. Ces instructions peuvent être entrées au clavier, ligne par ligne, et choisies parmi les suivantes :

Instructions	Effet
`Prendre un café`	Emilio ne fait rien, mais ensuite déplace son regard à la ligne suivante comme s'il avait fait quelque chose
`aller à gauche`	Emilio va vers la case à gauche de celle qu'il surplombe actuellement
`aller à droite`	Emilio se déplace d'un cran vers la droite
`cocher la case`	Emilio coche la case qu'il surplombe (n'a aucun effet si la case était déjà cochée)
`effacer la case`	Emilio efface la case qu'il surplombe (ou la laisse effacée si elle n'était pas encore cochée)
`aller à la ligne xxx`	Emilio déplace son regard vers la ligne xxx du script
`si coché alors aller à la ligne xxx`	Si la case qui est aux pieds d'Emilio est cochée, Emilio regarde la ligne xxx au lieu de la ligne suivante
`si vierge alors aller à la ligne xxx`	Si Emilio est au-dessus d'une case cochée, il fait comme si de rien n'était (il regarde ensuite l'instruction suivante). Sinon il déplace son regard vers la ligne xxx du script

On constate qu'il n'y a que 8 instructions, et qu'aucune d'entre elles ne signale à Emilio qu'il a fini son travail[1]. On sait donc qu'Emilio a fini lorsqu'il arrive à une ligne vide du programme (ci-dessous, la ligne 6)

Par exemple, il est devant une rangée de cases dont les dernières sont aléatoirement cochées, et est chargé de cocher toutes les cases vierges à partir de celle où il se trouve, vers la droite, jusqu'à ce qu'il arrive à une case déjà cochée, qu'il est au contraire chargé d'effacer avant de s'arrêter.

1 En fait il y a une instruction « stop » mais celle-ci n'étant pas nécessaire, n'a pas été affichée ci-dessus.

Voici le script qui permet à Emilio de faire ceci :

```
1. si coché alors aller à la ligne 5
2. marquer la case
3. aller à droite
4. aller à la ligne 1
5. effacer la case
```

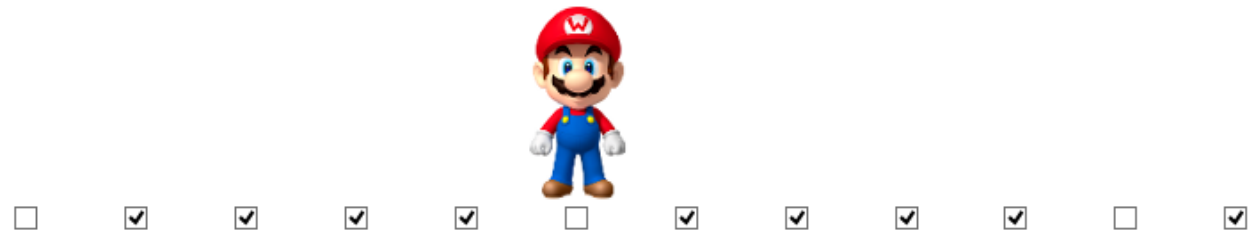
La ligne 1 stipule que dès que la case est cochée, Emilio doit arrêter de la cocher et aller à la ligne 5 du script. Ceci signifie que, ***si Emilio est à la ligne 5, c'est qu'il est arrivé à la première case cochée de son trajet***. Autrement dit, ***tant qu'Emilio lit les lignes 2 à 4, la case devant laquelle il se trouve était initialement vierge***, et ce genre de proposition logique est un invariant de boucle.

La ligne 2 lui intime l'ordre de cocher la case, puisque celle-ci était vierge. La ligne 3 lui demande ensuite de passer à la case suivante. La ligne 4 le fait boucler.

Voici la situation après un passage dans la boucle (Emilio est alors à la ligne 4)

Et voici la situation après le cinquième passage dans la boucle, qui l'a amené à une case déjà cochée, ce qui lui fait lire la ligne 5 :

Puis après lecture de la ligne 5 :



Emilio a alors fini sa mission, et ne fait plus rien si on continue à cliquer désespérément sur « Un pas »…

II/ Addition

Le script suivant effectue une addition de deux nombres écrits en base 1. La numération en base 1 est en fait la plus vieille du monde, elle consiste à représenter un nombre entier par des bâtons alignés, ou en l'occurrence, des cases cochées consécutives. Par exemple, le nombre 4 se représente par 1111 ou par 4 cases cochées entourées par 2 cases non cochées. Et le nombre 3 se représente de façon analogue par 111. Pour en savoir plus sur le traitement algorithmique des nombres entiers en base 1, voir l'annexe à la fin. Le fichier d'addition par machine de Post-Wang est initialement vide de tout nombre :

Avant de lancer l'algorithme d'addition, il faut représenter les nombres à additionner, en cochant les cases, ici pour 4+3, on fait 11110111 :

Du coup l'algorithme d'addition est très simple : Il faut cocher la case vide au milieu, et effacer la case tout à droite.

Une première façon de décrire l'algorithme est la suivante :

- aller jusqu'à la première case vide (lignes 1 à 3 ci-dessous)
- cocher ladite case (ligne 4)
- aller jusqu'à la première case vide (tout à droite ; lignes 5 à 7)
- aller à gauche (donc jusqu'à la dernière case cochée, ligne 8) et l'effacer (ligne 9)

Une traduction dans un langage ne possédant pas le mot « jusqu'à », donc plus proche de Post-Wang, remplacerait « aller jusqu'à la première case vide » par

- si la case est vide alors passer à la suite
- sinon aller à droite
- retourner au départ de la boucle

Ceci est écrit deux fois dans le script ci-dessous, dans les lignes 1 à 3 et dans les lignes 5 à 7 :

```
1. si vierge alors aller à la ligne 4
2. aller à droite
3. aller à la ligne 1
4. marquer la case
5. si vierge alors aller à la ligne 8
6. aller à droite
7. aller à la ligne 5
8. aller à gauche
9. effacer la case
```

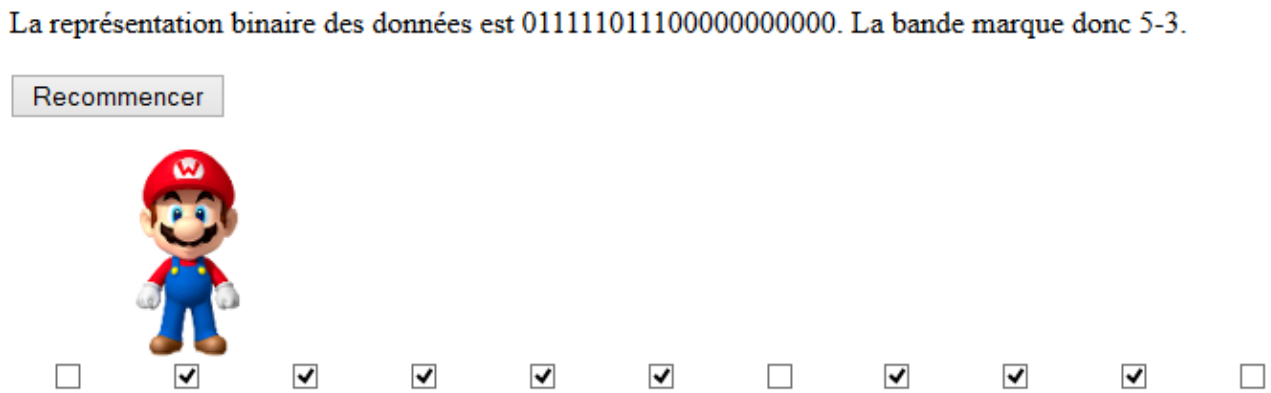
Une fois les cases 11110111 cochées, en lançant le programme d'addition ci-dessus on obtient :

☐ ☑ ☑ ☑ ☑ ☑ ☑ ☑ ☐

En comptant les cases consécutives cochées on en trouve bien 7 ; ainsi ||||+|||=||||||| soit 4+3=7.

III/ Soustraction

Pour soustraire 3 à 5, il est plus simple de le faire en base 1 à nouveau, et là encore on doit entrer les nombres 5 et 3 en cochant des cases successives séparées par une case vierge :



L'algorithme de soustraction est plus compliqué que l'algorithme d'addition parce qu'il faut, à chaque fois qu'on efface une case dans le nombre de gauche, en effacer une aussi dans le nombre de droite, ce qui nécessite une double boucle. L'exécution de l'algorithme affiche d'ailleurs successivement que 5-3=4-2=3-1=2-0=2. Emilio pense d'ailleurs, selon le principe de Peter, devenir un jour comptable…

L'algorithme peut se résumer ainsi :

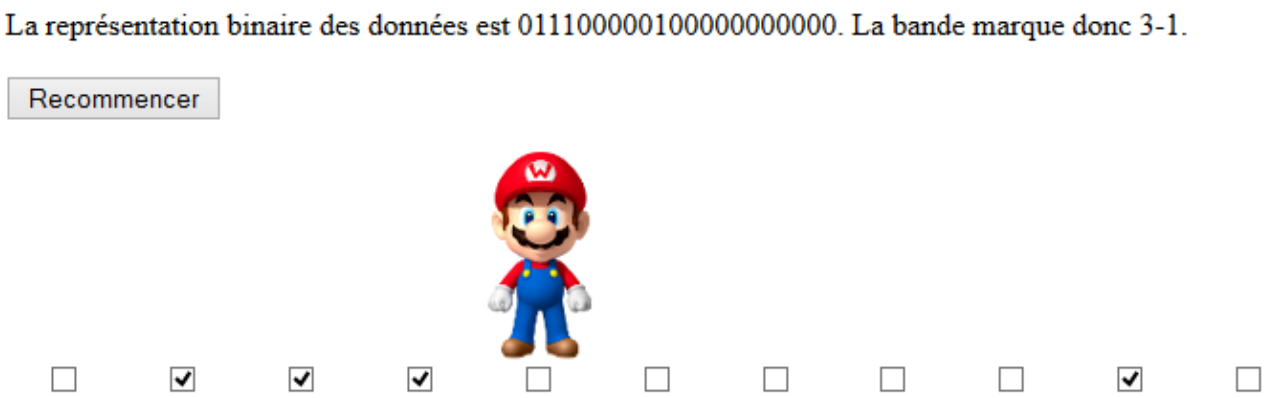
- aller à la première case vierge (lignes 1 à 3) ;
- aller à la première case cochée du nombre de droite (lignes 4 à 6) ;
- effacer cette case (ligne 7)
- retourner vers la gauche, jusqu'à la dernière case cochée du nombre a (celui de gauche ; lignes 8 à 10) ;
- effacer cette case pour compenser l'effaçage précédent (ligne 11) ;
- recommencer tout ça jusqu'à ce qu'il ne reste plus aucune case à effacer à droite (ligne 12).

La ligne 4 est alors le départ de la boucle consistant à enlever des cases à la fois à gauche (nombre a) et à droite (nombre b). La condition de fin n'est maintenant plus le fait qu'Emilio lise des instructions vides, mais le fait qu'il démarre un voyage sans fin vers la droite. Rencontrera-t-il le comptable avant d'arriver à l'infini ?

Voici le programme de soustraction en Post-Wang :

```
1. si vierge alors aller à la ligne 4
2. aller à droite
3. aller à la ligne 1
4. si coché alors aller à la ligne 7
5. aller à droite
6. aller à la ligne 4
7. effacer la case
8. si coché alors aller à la ligne 11
9. aller à gauche
10.     aller à la ligne 8
11.     effacer la case
12.     aller à la ligne 4
```

Voici l'état du lieu de travail d'Emilio après deux passages dans la boucle :



Puis à la fin, lorsqu'Emilio a commencé à partir vers la droite :

Bon voyage jusqu'à l'infini, Emilio !

IV/ Multiplication

La multiplication de nombres unaires est nettement plus compliquée (en Post-Wang) que les addition et soustraction, alors le programme est divisé en trois parties :

1. préparation

L'idée est de recopier b à la suite de c (le produit en cours de construction) chaque fois qu'on efface une des cases de a. Ainsi quand a devient a-1, c devient c+b et quand a est annulé, c est égal à a×b.

Pour repérer les débuts des nombres a, b et c, on doit rajouter une case cochée pour rien à gauche de chacun de ces nombres, ce qui amène a et b à être représentés avec un bâton de trop chacun. Alors le produit sera représenté lui aussi avec un bâton de trop. Cette préparation est confiée au premier programme, qu'Emilio va donc faire avant la multiplication proprement dite.

1.
```
1. aller à droite
2. si vierge alors aller à la ligne 4
3. aller à la ligne 1
4. marquer la case
5. aller à droite
6. effacer la case
7. aller à droite
8. si vierge alors aller à la ligne 10
9. aller à la ligne 7
10.     marquer la case
11.     aller à droite
12.     marquer la case
13.     aller à droite
14.     aller à droite
15.     marquer la case
16.     aller à gauche
```

Pour multiplier 3 par 4, Emilio commence donc par transformer le 11101111 en 1111011111, et la configuration devient alors celle-ci :

☐ ☑ ☑ ☑ ☑ ☐ ☑ ☑ ☑ ☑ ☑ ☐

Pour préparer la suite, Emilio doit cocher une case qui sera l'adresse du nombre c en mémoire :

☐ ☑ ☑ ☑ ☑ ☐ ☑ ☑ ☑ ☑ ☑ ☐ ☑ ☐

2. multiplication

Emilio boucle donc sur a :

- décrémenter a en décochant la case le plus à gauche de a ;
- copier b à la suite de c

Pour copier b à le suite de c, il boucle à nouveau, en effaçant une case à la fois de b, puis en la copiant après c, puis en la restaurant (pour les copies ultérieures).

Voici le script de la multiplication (programme 2) :

```
1. aller à gauche
2. si effacé alors aller à la ligne 5
3. aller à gauche
4. aller à la ligne 2
5. aller à gauche
6. si effacé alors aller à la ligne 9
7. aller à gauche
8. aller à la ligne 6
9. aller à droite
10.     effacer la case
11.     aller à droite
12.     si effacé alors aller à la ligne 40
13.     aller à droite
14.     si effacé alors aller à la ligne 17
15.     aller à droite
16.     aller à la ligne 14
17.     aller à droite
18.     aller à droite
19.     si effacé alors aller à la ligne 1
20.     effacer la case
21.     aller à droite
22.     si effacé alors aller à la ligne 25
23.     aller à droite
24.     aller à la ligne 22
25.     aller à droite
26.     si effacé alors aller à la ligne 29
27.     aller à droite
28.     aller à la ligne 26
29.     marquer la case
30.     aller à gauche
31.     si effacé alors aller à la ligne 34
32.     aller à gauche
33.     aller à la ligne 31
34.     aller à gauche
35.     si effacé alors aller à la ligne 38
36.     aller à gauche
37.     aller à la ligne 35
38.     marquer la case
39.     aller à la ligne 18
```

Pour commencer, Emilio décrémente a :

Emilio lit la ligne 11 qui dit :"aller à droite".

La représentation binaire des données est 00111011111010000000000000.

☐ ☐ ☑ ☑ ☑ ☐ ☑ ☑ ☑ ☑ ☑ ☐ ☑ ☐ ☐

Puis il va copier b après c ; pour cela il boucle sur les cases de b, en commençant par décocher une des cases de b, puis cocher la dernière case de c, puis rétablir la case effacée de b, et passer à la case suivante de b ; ceci se fait dans les lignes 18 à 39.

Lorsque la première case de b (représenté, on le rappelle, par un « 1 » supplémentaire à gauche suivi de 4 « 1 » soit par « 11111 ») est traitée, on voit que b est représenté provisoirement par 10111 et une case a été ajoutée à la fin de c :

Emilio lit la ligne 30 qui dit :"aller à gauche".

La représentation binaire des données est 00111010111011000000000000.

☐ ☐ ☑ ☑ ☑ ☐ ☑ ☐ ☑ ☑ ☑ ☐ ☑ ☑ ☐

Dans la boucle sur b, b est représenté successivement par

- 10111
- 11011
- 11101
- 11110 :

Emilio lit la ligne 30 qui dit :"aller à gauche".

La représentation binaire des données est 00111011110011111100000000.

☐ ☐ ☑ ☑ ☑ ☐ ☑ ☑ ☑ ☑ ☐ ☐ ☑ ☑ ☑ ☑ ☑ ☐

puis retour à 11111 :

Emilio lit la ligne 39 qui dit :"aller à la ligne 18".

La représentation binaire des données est 001110111110111110000000.

à ce moment b a été recopié une fois à droite de c. C'est le premier passage dans la boucle sur a. Après le premier passage on a ceci (a décrémenté) :

Emilio lit la ligne 11 qui dit :"aller à droite".

La représentation binaire des données est 000110111110111110000000.

Après le second passage :

Emilio lit la ligne 11 qui dit :"aller à droite".

La représentation binaire des données est 000010111110111111110000.

Et après le troisième passage, a est devenu vide et on a fini (mais attention, il y a une case cochée supplémentaire à gauche de c) :

Emilio lit la ligne 11 qui dit :"aller à droite".

La représentation binaire des données est 000000111110111111111111.

Emilio a alors fini la multiplication proprement dite, le premier facteur ayant disparu, et les 13 cases cochées à droites (1 comme repère et 12 pour le nombre représenté) représentent le produit.

3. présentation des résultats

Il reste donc à effacer les cases du second facteur et la case superflue du produit, ce que fait le programme suivant :

```
1. aller à droite
2. si vierge alors aller à la ligne 5
3. effacer la case
4. aller à la ligne 1
5. aller à droite
6. effacer la case
7. aller à droite
```

Comme prévu, Emilio pose fièrement auprès de son trophée 12 :

☐ ☑ ☑ ☑ ☑ ☑ ☑ ☑ ☑ ☑ ☑ ☑ ☑

Le fichier aux trois programmes peut être utilisé aussi pour multiplier 3 par 3, par 2 ou 2 par 2 (mais il manque de place en bas pour effectuer des produits plus grands).

On peut démontrer les faits suivants :

1. Toute fonction calculable sur des entiers (division euclidienne, génération de suites comme Collatz ou Fibonacci ou les nombres premiers successifs, résolution d'équations diophantiennes etc) peut être programmée en Post-Wang.
2. Par contre on n'a jamais dit que les programmes seraient simples à écrire (essayer par exemple de rédiger un algorithme d'addition d'octets en binaire) !
3. Il n'existe pas d'algorithme permettant de déterminer d'avance si Emilio arrivera en temps fini à la fin du programme, autrement dit, savoir si une journée de travail d'Emilio durera un temps infini, ne peut être calculé ni par Emilio ni par aucun de ses collègues. Les syndicalistes sont donc furieux et envisagent sérieusement d'entamer une grève dont ils affirment ne pas pouvoir déterminer par algorithme si elle s'arrêtera un jour.

Annexe 1 : La numération en base 1 avec Python

Pour convertir en entier en base 1, on multiplie la chaîne de caractères « 1 » par ce nombre. Python permet cela :

```
a = "1"*5
print a
```

affiche « 11111 » comme attendu. La conversion inverse se fait par la fonction « len », par exemple `len("11111")==5`.

Pour additionner « 11111 » avec « 111 », on effectue juste une concaténation, qui, en Python, se note justement par un « + » :

```
a = "1"*5
b = "1"*3
c = a+b
print c
```

affiche 11111111 qui représente bien 8=5+3.

Pour la soustraction c'est presque aussi simple : On remplace b (qui est nécessairement une sous-chaîne de a) par la chaîne vide :

```
a = "1"*5
b = "1"*3
c = a.replace(b,"")
print c
```

affiche « 11 » et effectivement, 5-3=2.

On remarque que toute tentative de soustraire un grand nombre à un petit nombre donne, non pas un message d'erreur (style « a n'est pas une sous-chaîne de b ») mais renvoie le petit nombre sans modification.

Pour la multiplication, il suffit de remplacer dans a, chaque « 1 » par b tout entier :

```
a = "1"*5
b = "1"*3
c = a.replace("1",b)
print c
```

a pour effet d'afficher « 111111111111111 » qui comprend 15 chiffres « 1 » et effectivement 5×3=15…

La division euclidienne est plus compliquée ; on opère par soustractions successives, en soustrayant b à a jusqu'à ce qu'on ne puisse plus, c'est-à-dire jusqu'à ce que a soit devenu plus petit que b ; alors a est le reste de la division euclidienne. Pour le quotient euclidien, on compte les passages dans la boucle comme le syndicaliste d'Emilio compte les jours en prison[2], en traçant des traits verticaux l'un à côté de l'autre, c'est-à-dire en ajoutant le caractère « 1 » à chaque passage dans la boucle :

```
a = "1"*5
b = "1"*3
c = ""
while len(a)>= len(b):
    c += "1"
    a = a.replace(b,"")
print c, a
```

Au fait, on peut simplement imiter les traits gravés par le syndicaliste emprisonné, en remplaçant les « 1 » par des « | » :

```
a = "|"*5
b = "|"*3
c = a.replace("|",b)
print "{0}*{1}={2}".format(a, b, c)
```

L'exécution de ce script Python affiche `|||||*|||=|||||||||||||||` ce qui est exactement ce que le syndicaliste a aperçu en entrant dans sa cellule, et qui l'a laissé perplexe.

2 Croyant devenir comptable avant Emilio, il a confondu les notations binaire et unaire, écrivant le nombre 7=1111111 dans un compte binaire ce qui l'a « miraculeusement » transformé en 127 qui s'écrit 1111111 en binaire. Ses explications n'ayant pas convaincu le banquier, il a été condamné à 1111111 jours de prison mais on ne sait pas en quelle base...

Annexe 2 : Repères historiques

Le premier mathématicien dont on soit certain qu'il ait élaboré un modèle d'ordinateur théorique (mais incomplet) est Jacques Herbrand, topologiste français, qui a eu l'idée de ramener tout calcul à un calcul sur des suites d'entiers[3], et de définir les fonctions calculables (il disait « récursives ») à partir de fonctions élémentaires[4], par composition et récurrence. La définition proposée par Herbrand pourrait s'écrire récursivement de cette manière :

- une constante est calculable
- la fonction qui, à un entier, ajoute 1, est calculable
- la fonction qui, à un tuple d'entiers, associe un sous-tuple, est calculable
- la composée d'une fonction calculable par une fonction calculable est calculable
- toute fonction définie récursivement à partir de fonctions calculables, est calculable.

Il a écrit une lettre à Gödel en 1931 pour lui demander son avis sur ses travaux, en tant que logicien reconnu, et est décédé peu après dans un accident d'escalade[5]. Mais Gödel passe l'essentiel des années suivantes à Princeton, où il s'est installé définitivement par la suite pour éviter le régime nazi qu'il ne supportait pas. Là, il travaillait au département de logique (on s'en serait douté!) sous la direction d'Alonzo Church. Là, il a affiné et présenté le modèle d'Herbrand, maintenant connu sous le nom de « modèle d'Herbrand-Gödel ». En 1934, Stephen Kleene, alors thésard auprès de Church, a montré à Gödel que le modèle d'Herbrand est incomplet, et qu'il fallait y ajouter la fonction μ, qui, à toute suite d'entiers, associe l'indice de son premier terme nul. Ce qui permis à Gödel de compléter le modèle d'Herbrand-Gödel et de proposer au fil des communications à Princeton, une définition récursive[6] de la calculabilité.

Pendant ce temps, Post lui aussi travaillait dans le département de logique de Princeton, mais apparemment il communiquait peu[7]. Aussi ne peut-on prouver qu'en réalité, il serait le premier à proposer un modèle de calculabilité avec des systèmes de réécriture qu'il disait avoir inventés avant 1930 mais sur lesquels il n'a rien publié avant 1943.

Et Church dans tout ça, qu'étudiait-il ? La théorie des types, une alternative à la théorie des ensembles[8]. Et au sein de celle-ci, le λ-calcul, ramenant à des transformations d'écriture, des problèmes de logique (démonstrations par exemple). Mais en 1935, ses étudiants Kleene et Rosser ont découvert une faille dans la construction de Church. Il a donc développé un λ-calcul non typé pour avoir une définition correcte de la calculabilité. Le moins qu'on puisse dire est que la définition par le λ-calcul n'est pas simple…

En 1936, Church publie un article où, pour la première fois, est donnée une définition mathématique du verbe « calculer ». Grâce en grande partie, à Kleene qui avait déjà aidé Gödel à affiner son modèle. Oui mais coup de tonnerre, la même année, Alan Turing publiait un article sur le même sujet et lui aussi, définissait le verbe « calculer » : Pour Turing, un calcul est ce que peut faire une machine dite « de Turing » qui peut

---

3 Par exemple, calculer π à trois décimales près c'est calculer les entiers 3, 1, 4, 1 et 6

4 La fonction « successeur » x+1 pour les entiers, les projections (comme la fonction qui à un point de coordonnées entières, associe son ordonnée) et les fonctions constantes

5 À l'âge de 23 ans...

6 Avec un changement de vocabulaire : Les fonctions récursives d'Herbrand sont devenues « primitives récursives » et l'adjectif « récursif » désignant désormais la nouvelle définition.

7 Comme Gödel, il est mort des suites d'une dépression : Anorexie pour Gödel et électrocuté par les électrochocs pour Post.

8 Visant à éviter les paradoxes de Russel comme celui de l'ensemble des ensembles qui ne se contiennent pas eux-mêmes

- lire le contenu d'une case (chiffre écrit dans la case) et modifier son état interne en conséquence, ainsi que son comportement ultérieur
- écrire un symbole dans une case
- se déplacer d'une case à celle qui la jouxte à gauche ou à droite

Les actions menées par la machine de Turing dépendant de son état interne (mémoire) et de ce qu'elle lit sur la case courante. Kleene ayant ensuite démontré que les définitions d'Herbrand, de Church et de Turing sont équivalentes, le modèle de Turing a vite eu le plus de succès parmi les trois, en raison de sa simplicité et de son caractère moins abstrait que les autres. Oui mais…

Arrive Post, qui lui aussi en 1936, publie un article dans lequel il révèle qu'il avait pensé avant Turing à une définition analogue, du moins pour les machines de Turing binaires : Il parlait d'un ouvrier allant d'une chambre à la chambre voisine (à gauche où à droite) et y ramassant une carte qu'il peut lire, cocher ou effacer. Effectivement on voit vite l'analogie avec le modèle de Turing, sauf que le modèle de Post était relativement informel. Par exemple, les actions de la machine de Turing se font via des changements d'état interne, alors que l'ouvrier de Post recevait des instructions d'on ne sait où.

Ensuite, l'informatique théorique a pris un tour nouveau lorsqu'on a commencé à construire des ordinateurs, ce qui soudainement donnait un sens à son objet d'étude. Pendant les années 1940, alors que sous les directives de Von Neumann, lui aussi à Princeton, on construisait les premiers ordinateurs américains, Post et Kleene s'attaquaient à des transformations d'écritures[9]. C'est à ce moment que Hao Wang est allé aux USA. En 1954, l'année du décès de Post, Wang a proposé, à des fins de description, une amélioration de la machine de Turing où, au lieu de changements d'états, la machine lit des instructions dans un programme inspiré des langages informatiques comme Fortran : La machine de Wang. Or il se trouve qu'elle ressemble beaucoup à celle de Post, d'où l'idée dans cet article de revenir à l'ouvrier Emilio au lieu du robot de Turing-Wang.

Alain Busser
Lycée Roland-Garros
Le Tampon
ce texte est sous licence Creative Commons CC-by-SA

9 Menant à la création du modèle de calculabilité dit « tag system » par Post, et à l'invention des RegExp par Kleene