De la puissance cachée de certains programmes

Gilles Verley

De la puissance cachée de certains programmes

gilles verley *

** LI, université de tours*

Sections de rattachement : 27

Secteur : tertiaire

RÉSUMÉ. Les programmes dits d'application fournissent des fonctionnalités spécifiques conçues pour répondre aux besoins bien identifiés d'une catégorie d'utilisateurs. Chaque fonctionnalité lorsqu'elle est « appelée » par l'utilisateur devient une action qui concoure au résultat attendu par celui-ci. Dans cet article, nous soutenons pour différentes applications, et montrons pour l'une d'entre elles, qu'il existe une suite invariante d'actions élémentaires de l'application visée qui constitue un cycle complet d'une machine de Turing. Dans le cas d'un traitement de texte, le ruban et le programme au sens de Turing étant simplement représentés par des caractères sur une page, la simple répétition d'une succession toujours identique d'actions élémentaires du traitement de texte (ex. trier, remplacer, mettre en gras …) dans le même ordre va exécuter le programme écrit dans le document et constitué des instructions élémentaires au sens de Turing (déplacer le pointeur de ruban, écrire une valeur à la position courante du ruban, changer l'état courant). Le résultat sera l'état du ruban à la fin du programme. De spécifiques au traitement de texte, les fonctionnalités prennent alors une puissance universelle au sens de Turing. Elles forment un système formel Turing-complet. De même pour les autres applications. Cette manière incongrue d'utiliser ces applications n'est, en général, pas vraie lors de leurs premières versions mais le devient par l'ajout au cours du temps de fonctionnalités apparemment anodines. On en conjecture que les concepteurs d'applications ont tendance inconsciemment à rendre celles-ci complètes et universelles au sens de Turing indépendamment des outils explicites de programmation qu'ils placent volontairement en sus (VBA,visualbasic…).

MOTS-CLÉS : système Turing-complet, machine de Turing, traitement de texte.

1. Introduction

Dans un ordinateur, toute nouvelle fonctionnalité logicielle est construite à partir des fonctionnalités existantes, matérielles ou logicielles. Au final, toute fonctionnalité est une succession automatisée d'opérations matérielles élémentaires qui se déroulent selon

un programme. Le développeur est celui qui fournit à des utilisateurs une fonctionnalité utile sans que ceux-ci n'aient à réaliser le programme eux-mêmes. C'est la justification habituelle de la différence apparemment radicale entre celui qui développe un programme et celui qui l'utilise. Cette différence n'est pas si claire qu'il n'y parait. D'abord, un programme ne fournit pas, en général, une seule fonctionnalité à l'utilisateur mais un ensemble. Ensuite, de nombreux programmes intègrent un véritable langage de programmation qui permet de créer des scripts qui sont interprétables par le programme. Il va sans dire qu'il est bien précisé que cette possibilité est réservée aux « utilisateurs avertis ». Les programmes qui fournissent cette possibilité deviennent alors des automates universels au sens de Turing (des systèmes formels « Turing-complets ») , c'est-à-dire pouvant calculer n'importe quelle fonction calculable pour peu qu'elle ait été développée par le client !

Il en résulte que la différence entre développeur et utilisateur est déjà moins nette. En effet, en tant qu'utilisateur, il me paraît normal qu'en achetant le droit d'utiliser des fonctionnalités qui me sont utiles sans que j'ai besoin de les programmer moi-même, j'achète également la possibilité de créer de nouvelles fonctionnalités à partir de la combinaison rationnelle des fonctionnalités que j'ai achetées. A ma charge de me former pour réaliser cette opération ou de la sous-traiter à un spécialiste. Il existe d'ailleurs une autre manière que les scripts de mettre en œuvre ce même droit, c'est celle qui consiste à obtenir les sources du programme que j'achète en même temps que l'exécutable. Alors, je peux, à ma guise, créer de nouvelles fonctionnalités à partir du code lui-même sous réserves de les compiler. Cela demande plus de compétences que la réalisation de scripts mais la différence n'est pas considérable.

De fonctionnalités élémentaires spécifiques d'une application en enchaînements simples (linéaires) de celles-ci dans des macros-commandes puis à la réalisation de scripts intégrant des structures de contrôle jusqu'à la compilation des sources modifiés, la route qui part de la simple utilisation d'un programme jusqu'au développement complet d'une nouvelle application est pleine d'étapes intermédiaires qui rendent la distinction entre utilisateurs et développeurs bien difficile à tenir d'un point de vue théorique.

Supprimons maintenant par la pensée, les macro-commandes, les langages de scripts ainsi que les possibilités de compilations décrites au paragraphe précédent de tous les programmes dits d'application. La différence entre utilisateurs et développeurs redevient plus claire. Le développeur crée des fonctionnalités en vue d'une utilisation spécifique et de telle manière qu'elles puissent être mises en œuvre simplement par un utilisateur spécialiste de son domaine mais pas de l'informatique. Souvent, ces actions consistent en des traitements sur des données spécifiques à l'application (du texte, des plans, des images, des nombres, du son, etc.). Il peut s'agir de calculer une balance ou un bilan dans un logiciel de comptabilité, de changer la typographie d'un texte dans un traitement de texte, de modifier la luminosité ou le contraste d'une image dans un logiciel de retouche d'images, etc.

Mais, même si on dépouille volontairement les programmes d'applications des possibilités explicites de programmation qui leur ont été parfois ajoutées, nombre de ceux-ci sont devenus des systèmes formels Turing-complets par l'adjonction progressive, au fur et à mesure de leurs versions successives, de fonctionnalités élémentaires apparemment anodines. Pour prendre une analogie souvent utilisée, l'informaticien est comparable à celui qui fabrique (ou répare) les voitures, et l'utilisateur est comparable à l'automobiliste qui conduit la voiture et n'a pas besoin de connaître la mécanique. A aucun moment, il ne viendrait à l'esprit de celui qui conduit une voiture l'idée saugrenue qu'en effectuant une certaine succession d'opérations sur les pédales ou le changement de vitesse, il puisse ainsi avoir construit une deuxième voiture différente de la précédente ou même une formule 1 ou une tondeuse à gazon ! Cela peut faire sourire ! Revenons à nos programmes d'applications. Lorsqu'ils sont vendus avec un langage de scripts, cela revient, dans notre analogie, à vendre une voiture avec tous les outils (fraiseuses, tours, presses, etc.) qui permettraient à l'heureux acheteur d'en reconstruire une autre sous réserves d'avoir la matière première, des plans et de s'y connaître en mécanique. Cela est-il encore possible si on retire cette trousse à outils des programmes d'applications en ne conservant que les fonctionnalités spécifiques de l'application ? Nous montrons en détail que cela est vrai pour des traitements de texte standards. Il en est de même pour la plupart des tableurs ainsi que pour d'autres applications.

Dans une première partie, nous rappellerons ce qu'est un système formel Turing-complet et nous présenterons un petit programme implémentant une fonction calculable. Puis nous montrerons en détail comment un traitement de texte possédant des fonctionnalités standards constitue un système formel Turing-complet. Enfin, nous présenterons sans détailler des résultats similaires et conclurons.

2. Machine de Turing et système formel Turing-complet

Une machine de Turing[1] est un modèle abstrait du fonctionnement des appareils mécaniques de calcul, tel un ordinateur et sa mémoire, créé par Alan Turing en vue de donner une définition précise au concept d'algorithmique ou « procédure mécanique ».

À l'origine, le concept de machine de Turing, inventé avant l'ordinateur, était censé représenter une personne virtuelle exécutant une procédure bien définie, en changeant le contenu des cases d'un tableau infini, en choisissant ce contenu parmi un ensemble fini

[1] Le paragraphe ci-dessous reprend des extraits d'articles de l'encyclopédie en ligne Wikipédia sous licence GNU, donc reproductibles en l'état, eux-mêmes fondés sur les articles fondateurs cités en bibliographie.

de symboles. D'autre part, la personne doit mémoriser un état particulier parmi un ensemble fini d'états. La procédure est formulée en termes d'étapes très simples, du type : « si vous êtes dans l'état 42 et que le symbole contenu sur la case que vous regardez est '0', alors remplacez ce symbole par un '1', passez dans l'état 17, et regardez une case adjacente (droite ou gauche) ».

La mise en œuvre concrète d'une machine de Turing est réalisée avec les éléments suivants :

1. Un « ruban » divisé en cases consécutives. Chaque case contient un symbole parmi un alphabet fini. L'alphabet contient un symbole spécial « blanc » ('0' dans les exemples qui suivent), et un ou plusieurs autres symboles. Le ruban est supposé être de longueur infinie vers la gauche ou vers la droite, en d'autres termes la machine doit toujours avoir assez de longueur de ruban pour son exécution. On considère que les cases non encore écrites du ruban contiennent le symbole « blanc ».

2. Une « tête de lecture/écriture » qui peut lire et écrire les symboles sur le ruban, et se déplacer vers la gauche ou vers la droite du ruban.

3. Un « registre d'état » qui mémorise l'état courant de la machine de Turing. Le nombre d'états possibles est toujours fini, et il existe un état spécial appelé « état de départ » qui est l'état initial de la machine avant son exécution.

4. Une « table d'actions » qui indique à la machine quel symbole écrire, comment déplacer la tête de lecture, et quel est le nouvel état, en fonction du symbole lu sur le ruban et de l'état courant de la machine. Si aucune action n'existe pour une combinaison donnée d'un symbole lu et d'un état courant, la machine s'arrête.

L'adjectif **Turing-complet** s’applique à un système formel ayant le pouvoir des machines de Turing, c'est-à-dire un système dans lequel on peut coder les machines de Turing. Le fait d'être Turing-complet est généralement requis pour un langage de programmation générique. En revanche, ce n'est pas le cas pour certains langages dédiés au traitement de problèmes spécifiques. Nous poursuivons avec un exemple.

La machine de Turing qui suit possède un alphabet {‘0’, ‘1’}, ‘0’ étant le « blanc ». On suppose que le ruban contient une série de ‘1’, et que la tête de lecture/écriture se trouve initialement au-dessus du ‘1’ le plus à gauche. Cette machine a pour effet de doubler le nombre de ‘1’, en intercalant un ‘0’ entre les deux séries. Par exemple, « 111 » devient « 1110111 ».
L’ensemble d’états possibles de la machine est {e1, e2, e3, e4, e5} et l’état initial est e1.
La table d’actions est la suivante :

Ancien état	Symbole lu	Symbole écrit	Mouvement	Nouvel état
e1	**0**		*(Arrêt)*	
	1	0	Droite	e2
e2	**1**	1	Droite	e2
	0	0	Droite	e3
e3	**1**	1	Droite	e3
	0	1	Gauche	e4
e4	**1**	1	Gauche	e4
	0	0	Gauche	e5
e5	**1**	1	Gauche	e5
	0	1	Droite	e1

L’exécution de cette machine pour une série de deux '1' serait (la position de la tête de lecture/écriture sur le ruban est inscrite en caractères gras et rouges) :

Étape	État	Ruban
1	e1	**1**1
2	e2	0**1**
3	e2	01**0**
4	e3	010**0**

Étape	État	Ruban
5	e4	01**0**1
6	e5	0**1**01
7	e5	**0**101
8	e1	**1**101

Étape	État	Ruban
9	e2	100**1**
10	e3	100**1**
11	e3	10010
12	e4	10011

Étape	État	Ruban
13	e4	10011
14	e5	1**0**011
15	e1	11011
		(Arrêt)

Le comportement de cette machine peut être décrit comme une boucle :

Elle démarre son exécution dans l’état e1, remplace le premier 1 par un 0.
Puis elle utilise l’état e2 pour se déplacer vers la droite, en sautant les 1 (un seul dans cet exemple) jusqu'à rencontrer un 0 (ou un blanc), et passer dans l'état e3.
L’état e3 est alors utilisé pour sauter la séquence suivante de 1 (initialement aucun) et remplacer le premier 0 rencontré par un 1.
L'état e4 permet de revenir vers la gauche jusqu’à trouver un 0, et passer dans l’état e5.
L'état e5 permet ensuite à nouveau de se déplacer vers la gauche jusqu’à trouver un 0, écrit au départ par l’état e1.
La machine remplace alors ce 0 par un 1, se déplace d’une case vers la droite et passe à nouveau dans l’état e1 pour une nouvelle itération de la boucle.

Ce processus se répète jusqu’à ce que e1 tombe sur un 0 (c’est le 0 du milieu entre les deux séquences de 1) ; à ce moment, la machine s’arrête.

Traitement de texte Turing-complet

Nous pouvons maintenant dire qu'un traitement de texte constitue un système formel Turing-complet si il existe une suite invariante d'actions élémentaires du traitement de texte considéré (copier, coller, rechercher, remplacer, souligner, etc.) qui constitue un cycle complet d'une machine de Turing, le ruban et le programme quelconque au sens de Turing étant représentés par des caractères sur des pages du traitement de texte.

Le ruban sera représenté par la première ligne de la première page d'un document du traitement de texte. Chaque emplacement de la ligne représente une case du ruban.

L'alphabet sera constitué de deux symboles {0,1}.

La tête de lecture sera représentée par le soulignement du symbole devant lequel elle se trouve. C'est la « position courante » de la tête qui détermine la « valeur courante » lue sur le ruban.

Le programme ou « table d'actions » sera représenté par une série de double-lignes dans la deuxième page. Chaque double-ligne représentant un état de la machine et codant les actions à effectuer lors d'un cycle (écrire une valeur sur le ruban à la « position courante », déplacer ou non la tête, changer « l'état courant ») en fonction du symbole en face de la tête (la valeur courante).

Chaque ligne est formée de 5 symboles « $S_1S_2S_3S_4S_5$ »:

- le premier correspond à l'état de la machine {A-Z} (26 états maximum ici)
- le second correspond à une valeur du ruban {0,1}
- les trois derniers indiquent les actions à effectuer si le premier symbole est identique à celui de l'état courant et le deuxième à la valeur courante du ruban :
    - le troisième indique le symbole à écrire {0,1} sur le ruban à la position courante
    - le quatrième indique le déplacement à effectuer de la tête {+,-,=} (« + » pour la droite, « - » pour la gauche) qui déterminera sa nouvelle position courant et donc la nouvelle valeur courante du ruban
    - le cinquième indique le nouvel état courant {A-Z}

L'état courant est représenté par un symbole {A-Z} stocké à la première ligne de la troisième page.

Soit un traitement de texte possédant les fonctionnalités suivantes :

Atteindre le haut d'une page avec le curseur
Sélectionner un paragraphe où se situe le curseur
Copier dans le presse-papier un paragraphe où se situe le curseur
Coller le paragraphe contenu dans le presse-papier à partir de la position du curseur
Rechercher des expressions régulières et les remplacer par d'autres
Souligner un mot sélectionné

Alors l'enchainement des actions de la liste ci-dessous constitue un cycle de la machine de Turing (l'état courant est en page 3):

Placer la valeur courante en page 4 :
Atteindre le haut de la page 1
Sélectionner et copier la ligne
Atteindre le haut de la page 4 et coller la ligne
Remplacer la ligne complète par la seule valeur soulignée en utilisant une expression régulière[2]
Placer la ligne d'action à exécuter en page 5 :
Atteindre le haut de la page 2
Sélectionner et copier les lignes
Atteindre le haut de la page 3 et coller les lignes
Remplacer les lignes par la seule ligne d'action à exécuter en utilisant une expression régulière
Placer la valeur à écrire à la position courante de la tête :
Atteindre le haut de la page 5
Sélectionner et copier la ligne
Atteindre le haut de la page 6 et coller la ligne
Remplacer la ligne complète par la valeur à écrire
Atteindre le haut de la page 1
Remplacer le caractère souligné par la valeur à écrire en utilisant une expression régulière
Atteindre le haut de la page 6 et effacer la ligne
Changer la position courante
Atteindre le haut de la page 5
Sélectionner et copier la ligne
Atteindre le haut de la page 6 et coller la ligne
Remplacer la ligne complète par la valeur du déplacement de la tête à effectuer
Atteindre le haut de la page 1
Remplacer le caractère souligné par le même entouré de deux « T » et précédé de la valeur du déplacement
Remplacer l'expression +T(.)T(.) par $1T$2T (sous open office writer)
Remplacer l'expression –(.)T(.)T par T$1T$2
Remplacer l'expression =T(.)T par T$1T
Remplacer l'expression T(.)T par $1 en format souligné
Changer l'état courant
Atteindre le haut de la page 5
Remplacer la ligne complète par son dernier caractère

---

[2] Pour le détail des expressions régulières utilisées et la démonstration complète d'un exemple, télécharger le fichier à l'adresse : http://193.52.210.2/CNRIUT2010/verley.odt

La répétition à l'identique de ce cycle d'actions aboutira à la résolution complète de n'importe quel programme « calculable » stocké à la page 2 et agissant sur les données stockées dans la page 1. L'exemple présenté en infra est traité de manière opérationnelle dans un document sous Open Office Writer à l'adresse : http://193.52.210.2/CNRIUT200/verley.odt

La même démonstration a été effectuée par nos soins avec des tableurs standards et des bases de données munies d'interpréteurs de requêtes sql ainsi que d'autres programmes. Nous n'avons pas la place de présenter ces résultats ici. Dans chaque cas, les programmes conçus initialement ne disposaient pas d'un ensemble d'actions élémentaires leur permettant de simuler une machine de Turing, c'est-à-dire d'être Turing-complet. Ainsi, pour les traitements de texte, il est apparemment[3] nécessaire de pouvoir utiliser des expressions régulières comportant des caractères génériques dans la fonction « rechercher/remplacer » , fonctionnalité absente des premières générations de traitement de texte (Sprint, Word 1). Pour les tableurs, il est apparemment nécessaire de disposer de la fonction « coller valeur », fonctionnalité également absente des premières générations de tableur (ex. Multiplan).

3. Conclusion

Les programmes d'application, au cours de leurs versions successives, s'enrichissent de fonctionnalités supplémentaires qui n'étaient pas apparues comme essentielles à l'origine mais qui, à un certain moment de leur histoire, font qu'ils permettent de simuler la machine de Turing et donc d'être « Turing-complet ». On peut penser que cette situation reflète le désir inconscient qu'ont les développeurs de donner à leurs programmes un caractère de complétude et d'universalité au sens théorique de la machine de Turing. Il faudrait vérifier cette hypothèse sur des programmes encore plus exotiques tels que des outils de retouche d'images, de mixage, etc.

4. Bibliographie

Turing A.M., On computable numbers, - Proceedings of the London Mathematical Society, 1937

Deutsch D., Quantum theory, the Church-Turing principle and the universal quantum computer, Proc. R. Soc. Lond. A 400, 97-117, 1985

Terwijn S.A., Éléments de Théorie de la Calculabilité, http://staff.science.uva.nl/~terwijn/publications/syllabus_fr.pdf, Amsterdam, 2008

[3] Apparemment car ce n'est pas ici démontré.