Expressivité d'un langage de programmation

Le langage de programmation défini dans l'introduction n'est pas très puissant : il ne permet pas, par exemple, d'empiler les gobelets de deux piles quelconques en une seule sans connaître *a priori* le nombre de gobelets par pile, ce qui correspond en quelque sorte à additionner le nombre de gobelets des deux piles. Autrement dit, il n'est pas possible d'écrire un programme qui, partant de deux piles comprenant chacune un nombre quelconque de gobelets, permette de construire une seule pile contenant la totalité des gobelets. En effet, pour cela, il faudrait d'une part pouvoir déterminer s'il y a ou non des gobelets présents sous la pince, et d'autre part être capable de « répéter » une séquence d'instructions (par exemple prendre un gobelet sur la première pile, le déposer sur la seconde, et revenir se positionner au-dessus de la première pile). Ces deux fonctionnalités ne sont pas réalisables par le langage simple proposé pour contrôler la pince.

Cet exemple illustre le fait qu'il est important de comprendre quelles sortes de programmes un langage donné permet d'écrire. C'est ce qui est appelé *expressivité* d'un langage de programmation. Tous les langages de programmation ne permettent pas de réaliser n'importe quelle tâche, car ils ne possèdent pas tous la même expressivité.

Paradoxalement, cette question a été étudiée sur un plan théorique par des mathématiciens (précurseurs et inventeurs de l'informatique) plusieurs années avant l'invention du premier ordinateur. Dans les années 1930, les ordinateurs n'existaient pas, mais Alan Turing (le père de l'informatique, cf. Figure **??**) s'est posé la question de savoir s'il était possible de concevoir un cadre mathématique qui permette de décrire tous les calculs réalisables par un procédé mécanique (nous dirions par une machine). Il a démontré que c'était bien le cas en inventant en 1936 la notion de « *machine de Turing* ». Il s'agit d'un objet mathématique qui abstrait le fonctionnement des ordinateurs. Ces « machines » permettent de décrire n'importe quel algorithme à partir d'un jeu minimum d'instructions élémentaires. Les résultats d'Alan Turing et les découvertes en micro-électronique ont permis à John Von Neumann en 1945 de concevoir un des premiers ordinateurs, appelé *EDVAC*. Ces travaux ont servi de base pour l'architecture de la plupart des ordinateurs modernes.

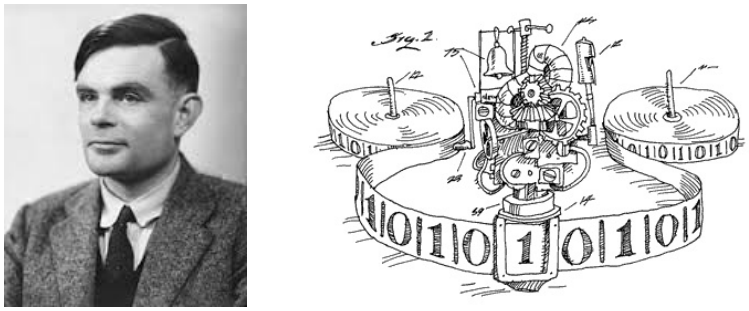

FIGURE 1 – Alan Turing et une illustration d'une « machine de Turing ».

En référence aux machines de Turing, un langage de programmation est dit « *Turing complet* » s'il permet de décrire n'importe quel algorithme. Les langages de programmation Turing complets sont tous aussi universels et puissants les uns que les autres. Ils diffèrent par les machines cibles, leur lisibilité, et leur efficacité (par exemple en ce qui concerne le temps d'exécution ou l'utilisation de la mémoire). Sauf quelques cas particuliers, les langages de programmation utilisés de nos jours sont tous Turing complets. Très grossièrement, on peut considérer qu'un langage autorisant les quatre opérations élémentaires sur les entiers (addition, soustraction, multiplication et division), et contenant en plus des instructions de comparaisons (`si ... alors ... sinon ...`) et de répétitions est Turing-complet. Au contraire, le langage utilisé pour contrôler la pince robotisée n'est pas Turing-complet, car comme nous l'avons vu plus haut, il ne permet pas de faire une addition.