Langages de programmation

1 Introduction

Une manière de diriger une pince robotisée déplaçant des gobelets est de lui donner une suite d'*instructions* élémentaires (déplacer la pince à droite $\rightarrow$, ou à gauche $\leftarrow$, saisir un gobelet placé sous la pince $\uparrow$ et poser un gobelet $\downarrow$). Chaque instruction élémentaire correspond à un mouvement de la pince. Dans ce contexte, un *programme* est une suite finie d'instructions élémentaires. Le programme suivant : $\uparrow \rightarrow \rightarrow \rightarrow \rightarrow \downarrow \leftarrow \leftarrow \leftarrow \leftarrow \uparrow \rightarrow \rightarrow \downarrow \leftarrow \leftarrow \uparrow \rightarrow \rightarrow \rightarrow \downarrow$ permet de construire une pyramide de 3 gobelets à partir de 3 gobelets empilés et de la pince positionnée au-dessus de cet empilement, comme indiqué sur la Figure 1.

FIGURE 1 – À gauche position initiale des gobelets, à droite position finale.

Pour appréhender la nature des langages de programmation, il est essentiel d'avoir à l'esprit la dissymétrie fondamentale entre les deux protagonistes, le programmeur et la machine. En effet, ce programme a été écrit par un humain dans le but de faire construire la pyramide au robot, alors que le robot accomplit simplement les actions indiquées sans viser un quelconque objectif. Le robot ne *sait* pas qu'il est en train de construire une pyramide de gobelets.

D'une façon générale, l'intention du programmeur est de commander une machine (ordinateur, robot, téléphone, console de jeu, ...) afin qu'elle accomplisse certaines tâches qu'il a définies. Pour sa part, la machine se contente d'exécuter à la lettre les ordres rédigés par le programmeur. Les langages utilisés par les humains pour écrire des programmes qu'exécutent les machines sont appelés des *langages de programmation*. Il existe de nombreux langages de programmation correspondant aux différentes machines, mais aussi permettant de rédiger des programmes de différentes manières.

2 Principes des langages de programmation

Chaque machine possède ses propres composants et mécanismes qui lui permettent de réaliser diverses opérations : dans le cas d'un ordinateur, il s'agit de modifier son état interne ; pour un robot, il s'agit par exemple de se déplacer. Pour faire exécuter des tâches complexes à une machine, le programmeur dispose d'un jeu d'instructions décrites à l'aide d'un langage de programmation adapté à cette machine. Un langage de programmation contient d'une part des *symboles* correspondant aux instructions et d'autre part des symboles particuliers permettant de combiner différentes instructions. Par exemple pour la pince robotisée, les quatre flèches constituent les symboles représentant les instructions élémentaires du langage de programmation et la juxtaposition de deux symboles sur la même ligne représente l'enchaînement de l'exécution de deux instructions. Ces instructions élémentaires et ces symboles de combinaison constituent le *vocabulaire* utilisé pour écrire des programmes. L'écriture des programmes doit respecter des règles de *syntaxe* très précises qui permettent uniquement certaines combinaisons d'instructions et assurent une communication non ambigüe avec la machine. Par exemple, le programme $\rightarrow + \leftarrow$ n'est pas valide car il contient le symbole $+$ qui n'appartient pas au vocabulaire autorisé.

$$\begin{array}{ccc} & \uparrow & \\ \uparrow & \rightarrow & \leftarrow \\ & \uparrow & \end{array}$$

FIGURE 2 – Programme non syntaxiquement correct.

Par ailleurs, le programme de la figure 2, bien qu'il ne contienne que des symboles autorisés, n'est pas syntaxiquement correct car les symboles ne sont pas écrits en ligne.

Pour que l'exécution d'un programme produise le résultat souhaité par le programmeur, il est important de connaître le résultat de l'exécution par la machine de chacune des instructions utilisées. On parle alors de *sémantique* des instructions et des langages de programmation. Avec la connaissance de la sémantique des instructions, le programmeur peut anticiper les actions des machines lorsqu'il conçoit ses programmes. Par exemple, si la sémantique des opérations des flèches de la pince robotisée était différente de celle présentée dans l'introduction alors l'exécution du programme proposé ne produirait probablement pas une pyramide de trois gobelets.

Ainsi, les langages de programmation sont très rigoureusement codifiés, ce qui les différencie des langues naturelles comme le Français. Cependant, ce ne sont pas les seuls langages de ce type que nous utilisons. Par exemple, l'écriture des partitions de musique est, elle aussi, régie par des règles bien précises :

- les symboles comprennent les notes, les clefs, les silences, la portée, etc.
- la syntaxe correspond aux conventions d'écriture des partitions : les notes occupent des emplacements bien définis sur la portée, les clefs sont mises en début de partition, etc.
- la sémantique correspond aux sons associés à la partition.

Nombres

```
0.2  .8  4.  1.e10  1.0e-7        Nombres à virgule
True; False                       Constantes booléennes
import math                       Librairie mathématique
```

Listes

```
s=[1, "bla", [1+2J, 1.4], 4]      Création de listes
s[2][0]                           Accès à un élément d'une liste
s=range(1000)                     Liste d'entiers de 0 à 999
len(s)                            Taille d'une liste
s.append(x)                       Ajouter l'élément x à la fin de la liste s
```

Instructions de base

```
if x = 3 :   sum = sum + x        Instruction conditionnelle
elif x = 4 : sum = sum + 2 * x
else: sum = sum - x
while ( cpt > 0) : cpt = cpt - 1  Boucle
print "hello world"               Affichage à l'écran
input()                           Saisie d'une valeur
def f(x): return x + 1            Définition d'une fonction
```

FIGURE 3 – Exemple d'une partie des instructions du langage de programmation Python portant sur les nombres, listes et instructions de base.

Les langages de programmation sont composés d'un nombre fini d'instructions. Chacun possède sa propre syntaxe, et sa propre sémantique. Par exemple la Figure 3 contient une partie des instructions du langage Python et de la sémantique associée. Ensuite, la Figure 4 présente un exemple d'une tâche décrite en langage naturel (ici en français) et un programme en langage Python qui réalise cette tâche. Afin d'écrire ce petit programme en Python, il est nécessaire de connaître les effets des différentes instructions : par exemple l'instruction `input` correspond à la saisie d'un nombre au clavier et l'instruction `print` à l'affichage d'un message à l'écran.

```
Prendre un entier
Le multiplier par 2
Ajouter 7 au résultat
Afficher le nombre obtenu
```

```
x = input("Entrer un nombre : ");
resultat = 2 * x + 7;
print(resultat);
```

FIGURE 4 – Exemple d'une tâche décrite en Français à gauche et un programme Python réalisant cette tâche à droite.

Après avoir décrit ce qu'est un langage de programmation du point de vue du programmeur mais aussi au niveau de la machine, il est important de comprendre que le lien entre le programmeur et la machine est réalisé par une étape logicielle automatisée appellée « *compilation* ».

3 Compilation

Une fois un programme écrit, les instructions qu'il contient doivent être exécutées par la machine choisie. Chaque machine possède sa propre architecture (composants, mémoire, processeurs, périphériques, etc ...). Ces différents éléments électroniques possédent en général deux états stables, activé ou chargé et désactivé ou déchargé. Ceci impose une représentation utilisant des 0 et des 1, qui est dite « *binaire* ». Chacune de ces architectures est régie par son propre langage appelé « *langage machine* » ou « *assembleur* ». Par exemple le programme de la Figure 5, écrit dans le langage assembleur du processeur Intel 80386, calcule le minumum de trois entiers. Expliquer le fonctionnement d'un tel programme dépasse largement le cadre de cette fiche.

Cependant, *in fine*, la machine n'exécute que des opérations binaires sur des composants électroniques. Il faut donc produire un fichier exécutable, aussi appelé « *fichier binaire* », à partir du programme en assembleur pour que la machine puisse accomplir les instructions du programme.

Ce fichier binaire est généré par un « traducteur » (lui-même un programme), appelé *compilateur*, entre le langage de programmation et le langage

```
min:
    pushl   %ebp
    movl    %esp, %ebp
    movl    8(%ebp), %edx
    movl    12(%ebp), %ecx
    movl    16(%ebp), %eax
    cmpl    %edx, %ecx
    leave
    cmovbe  %ecx, %edx
    cmpl    %eax, %edx
    cmovbe  %edx, %eax
    ret
```

FIGURE 5 – Un exemple de programme assembleur

propre de la machine constitué d'instructions en binaire. Plus généralement, un compilateur est un programme spécifique qui transforme un programme écrit dans un langage de programmation (le langage source) en un autre (le langage cible). Au cours de ce processus, le compilateur vérifie la syntaxe et effectue souvent des optimisations.

Les compilateurs et les langages machines offrent un moyen plus simple que le binaire pour contrôler une machine. Cependant, ces langages restent fastideux à utiliser. Ainsi sont nés les langages dit de *haut niveau* comme le C, le Java ou le Python. Ces langages, plus compréhensibles par l'humain, permettent une écriture plus facile des programmes. Lors de la production d'un fichier binaire, il y a finalement en quelque sorte deux compilations successives, une du langage de haut niveau vers le langage machine et une du langage machine vers le binaire.

La Figure 6 montre les principales étapes de la programmation, partant de la conception d'un algorithme par un être humain jusqu'à la réalisation d'actions par une machine. Chaque langage de programmation est traduit vers les langages assembleurs d'un certain nombre de machines cibles à l'aide de compilateurs spécifiques. Par exemple, il n'exsite pas encore de compilateur pour le langage Prolog vers les langages machines des consoles de jeu.

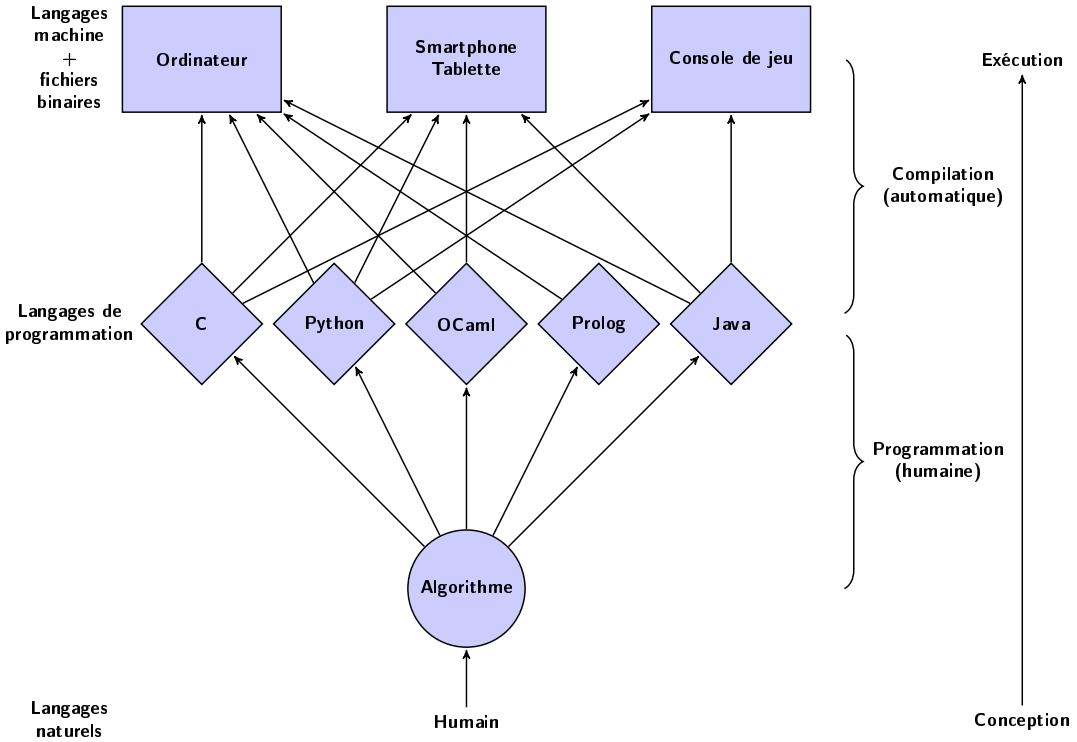

FIGURE 6 – Étapes de la programmation : de la conception à l'exécution.

Selon les langages de programmation l'étape de compilation peut s'effectuer à la volée pendant l'exécution (on parle alors d'interpréteur au lieu de compilateur) ou bien préalablement. Par exemple le C est un langage compilé tandis que le langage Prolog possède un interpréteur, et le langage Python peut être interprété ou compilé selon le choix de l'utilisateur.

4 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 7) 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.

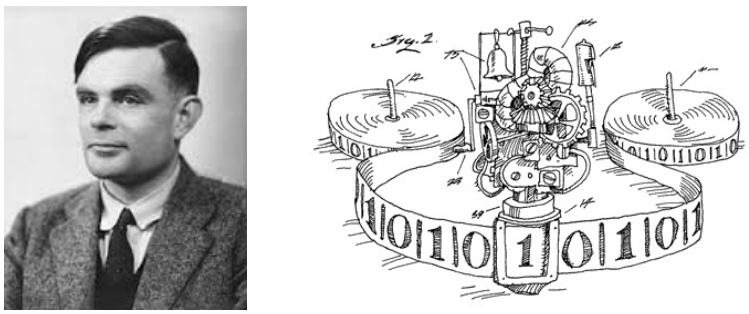

FIGURE 7 – Alan Turing et une illustration d'une « machine de Turing ».

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.

5 Paradigmes de programmation

Il existe des milliers de langages de programmation pour commander des machines (ordinateurs, téléphones, robots ...). Ils sont souvent classés par catégories, correspondant à des principes de fonctionnement différents : ces catégories sont appelées des *paradigmes de programmation*. Chaque paradigme suggère sa propre façon de concevoir les programmes. Pour illustrer cela, un exemple calculant le minimum d'une liste de nombres est donné ci-dessous dans les trois principaux paradigmes de programmation. Ces trois programmes reposent sur le même algorithme, qui consiste à parcourir la liste du début à la fin en mémorisant la plus petite valeur rencontrée au fur et à mesure.

Les langages impératifs : ils sont fondés sur une exécution étape par étape, on parle d'exécution « *séquentielle* ». Le programmeur décrit l'ordre selon lequel sont exécutées les instructions. Il a aussi la possibilité d'accéder directement à la mémoire contenant les données, ce qui permet d'écrire des programmes dont l'exécution est très rapide. Ce paradigme est le plus courant et historiquement le plus ancien, puisqu'il a été inventé conjointement à la création des premiers ordinateurs. Par exemple, les langages de programmation C, Pascal, Python, Fortran et COBOL font partie du paradigme impératif.

```
longueur=len(s)
if longueur == 0 : print ("La liste est vide")
else :
    mini = s[0]
    i = 1
    while i < longeuur :
        if s[i] < mini :  mini = s[i]
        i = i+1
    print ("Le minimum est : ", mini)
```

Exemple de programme Python qui calcule le minimum de la liste s.

Le programme Python commence par calculer dans la variable `longueur` le nombre d'éléments de la liste `s` grâce à la fonction `len(s)`. Ensuite le programme teste si la liste est vide en comparant sa longueur avec 0. Si c'est le cas, il affiche le message `La liste est vide`. Dans le cas contraire, il affecte à la variable `mini` la première valeur de la liste `s[0]` et initialise le compteur `i` à 1. Ensuite, tant que la valeur du compteur `i` est plus petite

que la longueur de `s`, le programme compare la valeur de l'élément d'indice `i` de la liste `s` avec la valeur de `mini`. Si `s[i]` est plus petit que `mini` alors la variable `mini` prend pour nouvelle valeur `s[i]`. Ensuite le compteur `i` est augmenté de 1. Une fois la boucle terminée, le programme affiche la valeur de la variable `mini` qui est bien le minimum de la liste.

Les langages fonctionnels : ils sont fondés sur l'évaluation de fonctions, dont le résultat est à son tour utilisé pour évaluer d'autres fonctions. Leur proximité avec le formalisme mathématique est un atout majeur de ces langages. En effet, cette caractéristique permet de prouver plus facilement que dans le paradigme impératif que les programmes réalisent ce pour quoi ils ont été conçus. Cette propriété est très souhaitable pour simplifier le travail des programmeurs. De plus, des objets mathématiques comme les listes ou les fonctions sont omniprésents dans ces langages, ce qui permet d'écrire des programmes plus concis (environ dix fois plus courts qu'en C). Les langages de programmation tels que Haskell, Ocaml et Lisp font partie des langages de programmation les plus connus dans le paradigme fonctionnel. Ces langages sont de plus en plus utilisés car ils permettent d'écrire des programmes certifiés sans bug, puisque prouvables.

```
let rec minimum s = match s with
|[]   -> failwith "La liste est vide"
|[x]  -> x
|e::r -> let mini = minimum r in
         if mini > e then e
                     else mini
```

Exemple de programme OCaml qui calcule le minimum de la liste s.

La première ligne de ce programme OCaml définit une fonction récursive appellée `minimum` qui prend en paramètre une liste `s`. En fonction du contenu de cette liste trois situations se présentent :

- Si la liste est vide `[]` (seconde ligne) alors le programme affiche le message `La liste est vide`.
- Si la liste `s` vaut `[x]` (troisième ligne), elle contient un seul élément `x`. Dans se cas la fonction renvoie `x` qui est bien le minimum de la liste.
- Enfin si la liste `s` vaut `e::r` (quatrième ligne), elle est composée d'un élément `e` et de la liste du reste des éléments de `s` notée `r`. Dans ce cas, la fonction `minimum` est appelée récursivement sur la liste `r` afin de calculer le minimum de cette liste. Cette valeur est alors notée `mini`. Il suffit finalement de comparer la valeur de `e` avec `mini` pour trouver le minimum de la liste et donc de renvoyer soit `e` soit `mini` en fonction des cas.

La programmation logique : elle est fondée sur la logique formelle définie au début du XXème siècle pour traiter des problèmes posés par les fondements des mathématiques. Le langage emblématique de cette catégorie est Prolog. Le programmeur définit un ensemble de faits élémentaires et de règles de logique leur associant des conséquences. Il permet d'écrire des programmes très concis et élégants car proches de la description mathématique de l'algorithme à l'origine du programme. L'usage de ces langages reste toutefois confidentiel.

```
minimum([X], X).
minimum([E|Fin], Mini) :- minimum(Fin, MinFin),
                                  Mini is min(E, MinFin)
```

Exemple de programme Prolog qui calcule le minimum d'une liste.

La première ligne de ce programme Prolog traite du cas où la liste contient un seul élément. Dans ce cas, le minimum de cette liste `[X]` est `X`. La suite du programme décrit le cas d'une liste contenant strictement plus d'un élément. Cette liste `[E|Fin]` est alors constituée d'un premier élément noté `E` et du reste de la liste noté `Fin`. Le programme calcule, de manière récursive, le minimum de la fin de la liste qui est stocké dans `MinFin`. Finalement, le minimum de la liste initiale, noté `Mini` est le minimum du premier élément `E` et de `MinFin`. Par exemple, l'exécution de la commande `minimum ([2,1,3],M)` calcule le minimum de la liste `[2,1,3]` à savoir `1` et stocke cet élément dans `M`.

Autres langages : Il existe d'autres catégories de langages que les trois paradigmes ci-dessus. Citons par exemple les langages événementiels : par opposition à la programmation séquentielle (impérative), dans ce cas, l'exécution est fondée sur la gestion d'événements (comme un clic de souris). La librairie swing en Java ou encore Node.js en JavaScript sont des langages événementiels. Il existe aussi des langages spécialisés, adaptés à des utilisations particulières : les langages orientés objet (Java), les langages pour pages Web (PHP, JavaScript), pour les bases de données (SQL) ou encore pour la pédagogie (LOGO, Scratch) et même des langages de programmation très exotiques [1].

L'objectif de ces exemples est uniquement d'illustrer quelques différences entre les trois paradigmes. Ainsi, on peut constater que le programme en Prolog est le plus concis, et que les syntaxes diffèrent radicalement entre les différents langages. Dans tous les cas, on note que la connaissance de chaque langage serait cruciale pour comprendre en détail le fonctionnement de ces différents programmes, pourtant issus du même algorithme.

1. Voir par exemple `http://en.wikipedia.org/wiki/BrainFuck`.

6 Conclusion

Traduire un algorithme sous la forme d'un programme écrit dans un langage de programmation permet en quelque sorte de lui donner vie ; car le programme pourra être exécuté par une machine, après compilation vers le langage de celle-ci. Il est donc indispensable d'aborder la notion de langage de programmation lorsque l'on s'intéresse à l'informatique.

Bien que la plupart des langages de programmation modernes soient Turing complets et donc en théorie équivalents, le choix d'un langage de programmation est quelque chose d'important. En particulier, le choix du paradigme de programmation est crucial, car il est plus naturel d'écrire certains programmes dans certains paradigmes, comme l'illustre l'exemple sur le calcul du minimum d'une liste donné précédemment : l'écriture en Prolog est bien plus concise qu'avec les langages appartenant aux autres paradigmes. Par ailleurs, le choix du langage conditionne aussi la manière de penser du programmeur en lui offrant de nombreuses possibilités de structure de données et en lui imposant des contraintes de performances. Il faut donc choisir le langage le plus adapté pour écrire un programme en fonction des objectifs à réaliser.

Comme on l'a vu, un langage de programmation doit être utilisable par un être humain mais aussi compatible avec un ensemble de machines qui exécutent les programmes. Il permet de faire le lien entre la conception humaine des tâches permettant d'aboutir à un résultat visé et leurs réalisations concrètes par les machines. Par conséquent, un langage de programmation doit être régi par des règles syntaxiques rigides levant toute ambiguïté pour la machine, mais aussi être suffisamment proche de la sémantique compréhensible par des humains. Toutes ces contraintes font que la conception d'un langage de programmation n'est pas facile, et cela mobilise encore de nos jours de nombreuses équipes de recherche et développement dans le monde afin de répondre aux nouveaux défis engendrés par les progrès des nouvelles technologies. Par exemple, il a fallu proposer ces dernières années de nouveaux langages de programmation pour les ordinateurs multi-coeurs (programmation parallèle), pour les flottes de robots autonomes (programmation distribuée comme dans le projet SYMBRION), ou encore pour les smartphones ou les drones (programmation embarquée).

Les concepteurs font des efforts considérables pour rendre les langages de plus en plus faciles à utiliser par des humains, tout en ayant de plus en plus de machines sur lesquelles les programmes seront exécutés.