Programmes et données

Introduction :

Le chapitre sur le génie logiciel nous amène à aborder les méthodes de programmation et les bonnes pratiques de développement logiciel. Afin de situer ces activités de conception logicielle, ce premier cours du chapitre s'intéresse à des concepts appartenant à la préhistoire de l'informatique, fondamentale pour la conception et le fonctionnement de nos ordinateurs modernes. Nous étudierons dans un premier temps les machines de Turing, pour nous intéresser dans un deuxième temps à la décidabilité avec le problème de l'arrêt. Nous ferons ensuite le lien avec les notions de programmes et de données dans le contexte des ordinateurs actuels.

1 Machines de Turing

Précisons dans un premier temps le contexte historique dans lequel le mathématicien britannique Alan Turing a mené ses recherches au cours de la première moitié du XX^{e} siècle.

a. Contexte historique

Dans les années 1930, les ordinateurs n'existent pas encore : ils seront inventés en 1948. Pour autant, un certain nombre de concepts et de connaissances existent déjà depuis plus ou moins longtemps :

- les premiers algorithmes sont très anciens puisqu'ils remontent à l'Antiquité ;
- les premières machines à calculer sont apparues au XVII^{e} siècle ;
- le concept de programme informatique est apparu au XIX^{e} siècle, quand Ada Lovelace a proposé un programme applicable à la machine analytique de

Charles Babbage.

Alan Turing travaille sur une question posée par le mathématicien allemand David Hilbert : **qu'est-ce qui est calculable ?**

→ C'est en cherchant à répondre à cette question qu'il va inventer une machine abstraite, laquelle sera rapidement appelée « **machine de Turing** ».

À retenir

> Cherchant à formaliser ce qui caractérise fondamentalement le calcul, Turing réfléchit à une **logique générale qui pourrait s'appliquer à tous les calculs**, et qu'on pourrait faire réaliser par n'importe qui en suivant rigoureusement des procédures composées d'opérations simples et strictement codifiées.

Un humain réalisant ces tâches devrait le faire mécaniquement, en suivant à la lettre les instructions fournies.
L'outil est abstrait, mais le comportement mécanique évoque celui d'une machine, décrite par Turing dans un article scientifique fondateur qu'il publie en 1936. Il imagine sa machine calculante dotée de différents composants, que nous allons à présent nous attacher à détailler.

b. Anatomie d'une machine de Turing

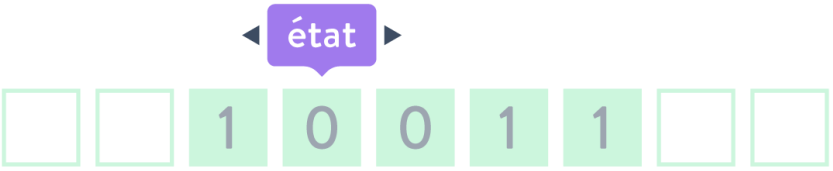
Attention

> Une machine de Turing est un dispositif abstrait.

Elle est composée des éléments suivants :

- un **ruban**, analogue à celui d'une machine à écrire, divisé en cases individuelles sur lesquelles différents caractères d'un alphabet donné peuvent être lus et écrits ;

- un **pointeur mobile** (ou curseur), doté d'un état interne (comparable à un registre mémoire capable de stocker une valeur) et d'une tête de lecture et d'écriture, capable de lire, d'effacer et d'écrire sur le ruban, et aussi de se déplacer par rapport à celui-ci ;



- une **table de transition** qui précise les actions à mener par le pointeur en fonction de son état et de la valeur lue sur le ruban.

→ Ces actions peuvent inclure la lecture, l'écriture, l'effacement et l'écriture d'une valeur à l'endroit où se trouve le pointeur, un changement d'état interne de ce dernier, et son déplacement sur le ruban (d'un cran vers la droite ou vers la gauche).

conditions à réunir		actions à réaliser		
état initial	lecture	écriture	déplacement	nouvel état
e1			>	e2
e2	0	0	>	e2
e2	1	1	>	e2
e2		0	>	arrêt

Astuce

> L'alphabet est fini, de même que le nombre d'états internes que peut prendre le pointeur.
> Le ruban est potentiellement de longueur infinie, et au moins considéré comme suffisamment long pour pouvoir effectuer toutes les étapes nécessaires au calcul effectué.

La table de transition est composée d'une série d'instructions indiquant la marche à suivre en fonction de l'état interne du pointeur (colonne « état initial » du tableau) et de la valeur figurant à la position courante sur le ruban (colonne « lecture » du tableau).

→ En d'autres termes la table de transition constitue le **programme** qui va s'appliquer aux données présentes sur le ruban, et ainsi permettre d'effectuer le calcul désiré en un nombre fini d'opérations.

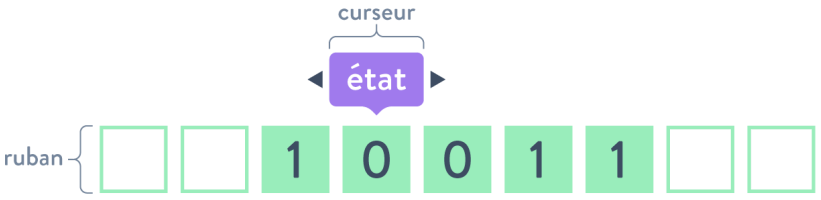


table de transition

état initial	lecture	écriture	déplacement	nouvel état
e1			>	e2
e2	0	0	>	e2
e2	1	1	>	e2
e2		0	>	arrêt

conditions à réunir — actions à réaliser

Rappel

> La machine décrite par Turing est totalement abstraite. Elle peut exister et fonctionner sans aucun support mécanique ou informatique : un humain doté d'un papier, d'un crayon et d'une gomme peut se comporter en machine de Turing.

Étudions un exemple simple de table de transition.

conditions à réunir		actions à réaliser		
état initial	lecture	écriture	déplacement	nouvel état
e1	blanc	blanc	>	e2
e2	0	0	>	e2
e2	1	1	>	e2
e2	blanc	0	>	arrêt

Astuce

L'opération revient à multiplier le nombre par 10 (2 en binaire), c'est-à-dire à ajouter un zéro à la fin du nombre binaire.

Exemple

La table de transition indique les actions à mener, en fonction de l'état interne du pointeur et du contenu de la case courante du ruban. Ces actions consistent en l'écriture d'une valeur (éventuellement identique) sur le ruban, puis en un déplacement ou l'arrêt de la machine.

Le ruban est au départ positionné sur un emplacement blanc, et le nombre binaire à traiter est situé à droite de cet emplacement. Une fois le traitement terminé, la machine s'arrête sur la position blanche située immédiatement après le zéro final.

Appliquons la table de transition au nombre écrit en binaire sur le ruban comme suit : 10011

	conditions à réunir		actions à réaliser		
étape	état initial	lecture	écriture	déplacement	nouvel état
1	e1	blanc	blanc	>	e2
2	e2	1	1	>	e2
3	e2	0	0	>	e2
4	e2	0	0	>	e2
5	e2	1	1	>	e2
6	e2	1	1	>	e2
7	e2	blanc	0	>	arrêt
8	arrêt				

Alan Turing a ensuite généralisé son concept pour le rendre universellement applicable.

c. Machine de Turing universelle

On peut concevoir des machines de Turing pour mener à bien toutes sortes d'opérations spécialisées plus ou moins complexes. Alan Turing avait également imaginé un concept de machine universelle, que l'on désigne sans surprise comme « machine de Turing universelle » ou « machine universelle de Turing ».

À retenir

> Cette machine universelle a pour particularité de pouvoir simuler le fonctionnement de n'importe quelle autre machine de Turing.

Elle prend en entrées :

- la table de transition de la machine considérée ;
- les données d'entrée, initialement présentes sur le ruban, de la machine considérée.

Astuce

> **Remarque :**
>
> Si on transpose ce concept aux ordinateurs modernes, ils ont globalement les possibilités de calcul d'une machine de Turing universelle (à ceci près que leur mémoire, bien que considérable, n'est pas infinie).

La logique de traitement définie par la table de transition indique le modèle des traitements à effectuer. Ces traitements pourraient être indifféremment réalisés par un humain ou par une machine. La logique algorithmique est donc indépendante de tout langage informatique, même si de nos jours les ordinateurs et les langages informatiques sont de précieux alliés.

→ Ils nous permettent d'appliquer cette logique de traitement à d'importants volumes de données.

d Calculabilité et décidabilité

Le modèle proposé par Turing pour évaluer la **calculabilité** est abstrait.

→ En somme, un problème est calculable s'il est traitable par une machine de Turing.

Travaillant sur la même problématique que Turing, le mathématicien américain Alonzo Church invente son propre système formel, appelé **lambda-calcul**, qui servira plus tard de base à des langages informatiques parmi lesquels Lisp, Haskell ou OCaml.

Church et Turing se rejoignent dans leurs recherches sur **les limites de ce qui est calculable et de ce qui ne l'est pas**. Leurs modèles sont différents mais aboutissent aux mêmes conclusions, et permettent de constater une forme d'équivalence, connue sous le nom de **thèse de Church-Turing**.

→ Cette thèse affirme que tout traitement réalisable par des processus mécaniques est également réalisable par une machine deTuring.

Définition

> **Décidabilité :**
>
> La décidabilité algorithmique fait référence à l'existence ou non d'un algorithme capable de fournir une réponse à un problème donné en un temps fini.

À retenir

> La problématique de la décidabilité est équivalente à celle de la calculabilité.

Pour déterminer si tout est ou non calculable, Church et Turing ont chacun recherché et trouvé l'existence d'un problème qu'il ne serait pas possible de résoudre par un algorithme : il s'agit du problème de l’arrêt.

2 Le problème de l'arrêt

Un problème est dit « décidable » s'il existe un algorithme qui permet de le résoudre. *A contrario*, un problème est dit « indécidable » s'il n'existe pas

d'algorithme qui permette de le résoudre. Nous allons étudier le **problème de l'arrêt** avec des machines de Turing.

a. Exposé du problème

La question posée par Turing peut se présenter ainsi : existe-t-il une machine de Turing capable de déterminer si une machine de Turing quelconque finira par s'arrêter ou si au contraire elle continuera de boucler de manière infinie ?
Pour certaines machines, il est facile et rapide d'apporter une réponse. Mais la question porte sur une machine quelconque : autrement dit, un principe général qui pourrait s'appliquer à n'importe quelle machine de Turing.

On ne peut évidemment pas tester tous les programmes imaginables. Mais il suffirait de trouver un seul cas mettant en échec ce principe pour prouver qu'un tel programme ne peut pas exister.

→ Il s'agit d'une démonstration par l'absurde : on suppose que ce problème de l'arrêt est décidable, c'est-à-dire qu'il existe une méthode générale pour savoir si une machine va s'arrêter ou pas sur une entrée donnée.

b. Formalisation avec des machines de Turing

1 Machine A

Appelons « machine A » (pour arrêt) cette machine de Turing qui sait résoudre le problème de l'arrêt, c'est-à-dire qu'elle est capable de déterminer si une machine quelconque finira ou non par s'arrêter.

Cette machine A prend en entrées :

- la table de transition de la machine à évaluer ;
- les entrées à traiter par cette machine.

La machine A doit indiquer en sortie si :

- la machine évaluée finit par s'arrêter ;
- ou si la machine évaluée ne s'arrête pas et boucle à l'infini.

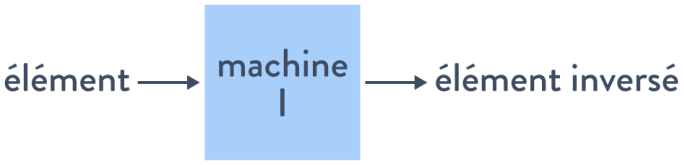
2 Machine C

On crée une deuxième machine de Turing, appelée machine C (pour copie). La machine C est une machine à copier, qui duplique simplement l'entrée reçue et produit deux exemplaires identiques en sortie.

3 Machine I

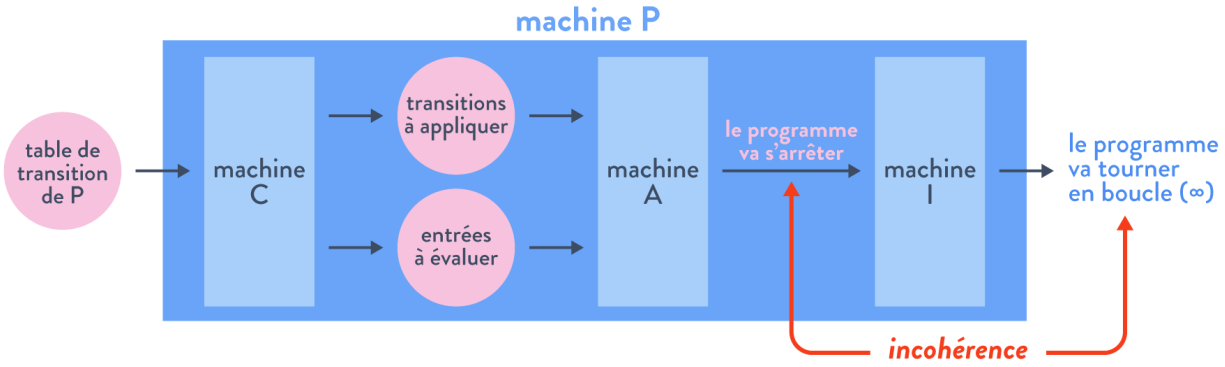
On crée ensuite une troisième machine de Turing, appelée machine I (pour inverseur) qui inverse les résultats reçus en entrée :

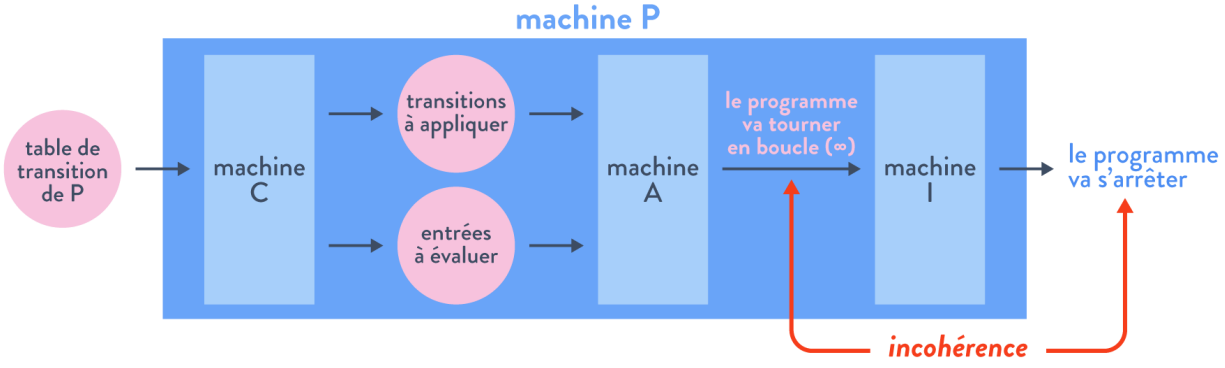
- elle indique « boucle à l'infini » quand elle reçoit l'entrée « finit par s'arrêter » ;
- elle indique « finit par s'arrêter » quand elle reçoit l'entrée « boucle à l'infini ».



4 Machine P

On place en séquence la machine C de copie, la machine A d'arrêt et enfin l'inverseur (machine I). L'ensemble forme une machine P (pour paradoxe).





Exemple

Il est possible d'effectuer la démonstration avec un programme.

Supposons que la fonction `Evaluation` recherchée existe.
`Evaluation (prog,donnees)` renvoie à :

- « se termine » si `prog(donnees)` se termine ;
- ou « ne se termine pas » si `prog(donnees)` ne s'arrête pas.

Considérons maintenant le programme :

```
Test(x)
Si Evaluation (x,x) = "se termine" alors
        boucle infinie
Sinon
        Arret
```

En appliquant `Test` à lui-même `Test(Test)`, on obtient une contradiction : si `Test(Test)` boucle, c'est que `Evaluation(Test,Test)` renvoie « se termine » c'est-à dire si `Test(Test)` se termine.

c. Hypothèses de fonctionnement

Étudions maintenant ce qu'il se passe si on fournit à P sa propre table de transition.

La machine C effectue une copie de son entrée et fournit donc deux exemplaires de la table de transition de P. La machine A reçoit en entrées ces deux exemplaires, l'un étant considéré comme table de transition à appliquer et l'autre comme données à évaluer.

→ Quel résultat peut produire la machine A ?

Supposons que la machine A indique que le programme finira par s'arrêter. Ce résultat est ensuite passé à la machine I qui l'inverse et indique au final que le programme boucle à l'infini. Cela signifierait globalement au niveau de P que fournir la table de transition de P à elle-même la fait boucler à l'infini. Mais la machine A a dit que ce n'était pas le cas, puisqu'elle a indiqué que le programme finirait par s'arrêter. Il y a donc une incohérence.

Supposons *a contrario* que la machine A indique que le programme bouclera à l'infini. Ce résultat est ensuite passé à la machine I qui l'inverse et indique au final que le programme P finit par s'arrêter. Mais la machine A a dit que ce n'était pas le cas, puisqu'elle a indiqué que le programme bouclerait à l'infini. Il y a donc là encore une incohérence.

d. Indécidabilité du problème

Comme le met en évidence la machine P, la machine A produit un résultat incohérent. La mise en évidence de ce paradoxe nous indique qu'il n'existe pas de machine A capable de décider si une machine finira ou non par s'arrêter.

À retenir

> Il n'existe pas d'algorithme qui permette de résoudre ce problème : le problème de l'arrêt est donc indécidable.

Astuce

> On retrouve le même type d'incohérence avec le paradoxe du menteur : un homme affirme qu'il ment. Ment-il ?
>
> - Soit l'homme ment lorsqu'il affirme qu'il ment, ce qui signifie qu'il ne ment pas.
> - Soit l'homme ne ment pas lorsqu'il affirme qu'il ment, ce qui signifie qu'il ment.

L'indécidabilité du problème de l'arrêt a permis à Church et Turing de montrer que tout n'était pas calculable. Même si les travaux de Church et de Turing étaient mathématiques, ils ont eu une grande influence sur la conception ultérieure de machines qu'on appellera ordinateurs.

Turing sera ensuite particulièrement impliqué dans la construction de machines concrètes, notamment au cours de la Seconde Guerre mondiale pour percer les secrets de la machine électromécanique allemande de chiffrement et de déchiffrement Enigma.

Examinons maintenant nos programmes informatiques modernes et les données qu'ils traitent sous l'angle des machines de Turing.

3 Programmes et données

La machine de Turing est à la fois l'ancêtre conceptuel et une représentation tout à fait actuelle de l'ordinateur tel que nous le connaissons aujourd'hui.

Même si sa conception était abstraite, la machine de Turing constituait un modèle à la fois très simple et très puissant par son universalité.

À retenir

> Tout ce qui peut être calculé par une machine de Turing peut l'être par un ordinateur. Inversement, ce qu'une machine de Turing ne peut pas calculer ne peut pas non plus l'être par un ordinateur.

Ainsi, les machines de Turing traitent des tables de transition appliquées à des données, comme nos ordinateurs exécutent des programmes traitant des données.
Précisions ce qu'on entend par programmes et par données numériques.

Définition

> **Programme :**
>
> Au sens informatique, un programme est une implémentation d'un algorithme de traitement de données.

Définition

> **Donnée :**
>
> Une donnée est une information ou un ensemble d'informations pouvant être fournies à un programme, traitées par un programme ou fournies par un programme

Attention

> La distinction entre programmes et données n'est cependant pas aussi nette qu'on pourrait le penser à première vue, dans la mesure où tout programme est aussi une donnée, ainsi que nous allons l'illustrer dans différents contextes.

a. Programmation

L'exemple ci-après montre un extrait de code en langage Python qui définit une fonction déterminant la parité d'un nombre (un entier donné est-il ou n'est-il pas un multiple de deux).

Exemple

> La fonction ci-dessous indique si un nombre est pair ou impair.

```
def est_pair(nombre) :
    """Indique si un nombre est pair.
    Prend en entrée un nombre entier.
    Retourne un booléen (True ou False).
    """
    return nombre % 2 == 0
```

La fonction définit un algorithme (programme) capable d'évaluer des données (nombres) passées en arguments lors des appels de la fonction. Le code définissant cette fonction est lui-même exprimé sous forme de texte interprétable par le langage Python. Ce code constitue des données qui sont traitées par le programme qu'est l'interpréteur Python.

La même logique s'applique au code source d'un langage compilé lorsque ce code est traité par le compilateur afin de produire un programme exécutable.

b. Logiciel

Rappel

> Un logiciel est un programme destiné à s'exécuter sur un système d'exploitation.

Lorsqu'il est téléchargé (par un navigateur, un gestionnaire de fichiers distants ou une instruction en ligne de commande), ce logiciel est transféré sous forme d'octets qui constituent des données de téléchargement, au même titre que le téléchargement d'une image ou d'une vidéo.

Le stockage des logiciels sur la machine locale s'effectue sous forme de données enregistrées et gérées par le système d'exploitation. Lorsqu'il est lancé (exécuté), cet ensemble de données est un programme, qui peut lui-même recevoir, transformer et produire des données.

c. Navigation web

Dans le contexte de la navigation sur le web, une page consultée par un internaute peut avoir été produite de manière dynamique par un script côté serveur, par exemple programmé en PHP. Ce code PHP a donc été interprété localement au niveau du serveur pour produire en résultat un code HTML, lequel est ensuite transmis au navigateur sous forme de données de téléchargement. Ces données sont prises en compte sous forme de code HTML par le navigateur qui interprète à son tour ce code pour en effectuer le rendu graphique et fournir les informations destinées à l'utilisateur.

→ Ces différentes illustrations montrent qu'en fonction du contexte, un même élément peut être soit un programme, soit un ensemble de données, ainsi que nous l'avions abordé de manière plus abstraite avec les machines de Turing.

> **Conclusion :**
>
> Nous avons montré à la fois l'ancienneté et la modernité des machines de Turing comme modèles conceptuels en informatique, et souligné que tout n'était pas nécessairement décidable par un algorithme. Nos ordinateurs modernes étant comparables à des machines de Turing, nous avons ensuite abordé les notions de programmes et de données, constatant qu'en fonction des contextes de traitement, un même élément pouvait être tantôt un programme, tantôt un ensemble de données.