La machine de Turing

Olivier Ridoux

La machine de Turing

Olivier Ridoux – 2020

Motivations

En 1936, Alan Turing publiait l'article intitulé « *On Computable Numbers with an Application to the* Entscheidungsproblem ». Il y répondait par la négative à une question de logique formalisée dans les années 20 mais latente depuis le début du siècle. Il le faisait au moyen d'une construction formelle qu'on appelle maintenant Machine de Turing (MT) et qui est en fait le modèle théorique de presque tout ce qui se fait en matière de calcul depuis plus de 80 ans. L'informatique n'est donc pas que ce domaine où tout change tous les 3 ans. C'est une discipline scientifique dont les lois résistent au temps. Il est donc plus stratégique de s'intéresser à ce qui est aussi stable, plutôt qu'à ce qui change tout le temps.

Le résultat négatif démontré par Turing est que tout n'est pas calculable automatiquement. Dit comme cela c'est une banalité, et l'homme de la rue est aussi persuadé que l'ordinateur ne peut pas tout calculer. Mais là où l'homme de la rue imagine des tâches mal définies et souvent très mal partagées par les humains, comme traduire de la poésie ou poser un diagnostic, Turing démontre que des tâches très formellement définies ne peuvent pas être réalisées automatiquement. Inversement, les tâches mal définies, mais que l'homme de la rue juge impossibles à réaliser automatiquement, sont les cibles favorites des machines qui utilisent les méthodes de l'intelligence artificielle.

Ces fonctions formellement irréalisables automatiquement ne sont pas des inventions tirées par les cheveux, des **fonctions pathologiques** inventées pour l'occasion. Ce sont des fonctions très utiles. L'impossibilité de les automatiser complètement fait qu'il est impossible de prouver automatiquement qu'un programme s'arrête, ou qu'il réalise le service attendu. Pour la même raison, il est impossible de détecter en toute généralité qu'un document contient un virus.

Pour toutes ces raisons, l'article de Turing est un virage dans la compréhension de ces phénomènes, une date dans l'histoire de l'informatique, et un éclairage toujours pertinent sur l'informatique d'aujourd'hui. Dans la suite, les numéros de section font référence à cet article.

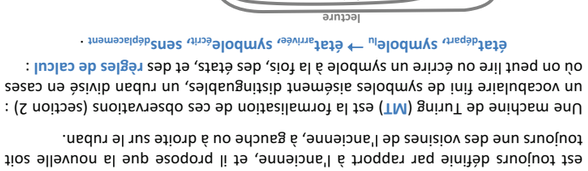
L'intuition de Turing

Dans la section 9, Alan Turing part du calcul arithmétique manuel pour donner une intuition de sa machine. Il fait plusieurs observations :

- L'opérateur qui calcule à la main (Turing dit *computer*) utilise les **symboles** d'un **vocabulaire** fini dont les éléments sont aisément distinguables. Ces symboles sont des lettres, des chiffres, des mots, ..., et pourquoi pas des ♠, ♥, ♦ et ♣. Il y a toujours un symbole, _, ou ***b*** (*blank*) pour exprimer que rien n'est écrit.
- L'opérateur écrit ces symboles sur une grille convenue d'avance, ex. les lignes et les colonnes des opérations arithmétiques. Turing remarque qu'on peut mettre toutes les lignes bout-à-bout pour former une seule ligne, qu'il appelle **ruban** (***taupe***). Et de la même façon qu'une procédure arithmétique suppose toujours d'avoir assez de papier, on supposera que le ruban est toujours assez grand.
- Turing ajoute qu'une procédure de calcul enchaînant différentes actions élémentaires, l'opérateur doit savoir où il en est. Ex. pour une addition, il sait à quelle colonne il en est, quelle ligne, si il y a une retenue, etc. Turing appelle cela l'**état mental** de l'opérateur. Il observe que l'état mental est fragile, et que, interrompu, l'opérateur peut devoir tout recommencer à zéro. Il énonce donc qu'une machine calculante devra expliciter l'état mental, en la représentant par un symbole, ex. 😐, 😳, ou 😡.
- Il observe enfin que l'opérateur ne considère pas sa feuille de calcul globalement, mais aisément position par position. Il observe que la nouvelle position est toujours définie par rapport à l'ancienne, et il propose que la nouvelle soit toujours une des voisines de l'ancienne, à gauche ou à droite sur le ruban.

Une machine de Turing (**MT**) est la formalisation de ces observations (section 2) : un vocabulaire fini de symboles aisément distinguables, un ruban divisé en cases où on peut lire ou écrire un symbole à la fois, des états, et des **règles de calcul** :

état$_{départ}$**, symbole**$_{lu}$ **→ état**$_{arrivée}$**, symbole**$_{écrit}$**, sens**$_{déplacement}$ **.**



3. (section 11) Il n'existe pas de jeu de règles **P** tel que pour toute formule ϕ du calcul des prédicats représentée initialement sur le ruban, $\mathcal{M}_{\mathbf{P}}[\mathbf{P}](\phi) = Vrai$ si ϕ **est démontrable**, **Faux** sinon. Le calcul des prédicats (le *Entscheidungsproblem*) est donc **indécidable**. Turing l'a prouvé en montrant que si une telle machine existait on pourrait s'en servir pour résoudre le Problème de l'arrêt en le modélisant dans la logique du calcul des prédicats, alors qu'il vient d'être prouvé indécidable. Cette méthode de preuve, dite par **réduction**, est aussi typique de l'algorithmique.

Les résultats de Turing

On dit qu'un **jeu de règles R réalise une fonction f** (notons cela $\mathcal{M}[R] = f$), si à chaque fois qu'on représente sur le ruban un **x** pour qui **f(x)** est défini, ces règles réalisent alors un calcul qui s'arrête avec la représentation de **f(x)** sur le ruban. Pour les **x** où **f(x)** n'est pas défini, le calcul **ne s'arrête pas et ne rend donc aucun résultat**.

La notion **d'arrêt** est fondamentale. Elle induit ce creux la notion de **programme qui boucle**, c-à-d. de programme qui rentre dans une séquence d'actions qui ne s'arrête pas, soit qu'elle se répète, soit qu'elle produit des situations toujours nouvelles. Il est vraiment difficile de distinguer les deux ; dans la vie, on ne peut le plus souvent que suspecter qu'un programme boucle parce qu'il prend trop de temps.

Turing démontre les trois résultats suivants (en notation moderne et simplifiée) :

1. (sections 6 et 7) Il existe un jeu de règles **MTU** tel que pour tout jeu de règles **R**, il est vrai que $\mathcal{M}[\mathbf{MTU}](\mathbf{R}, \mathbf{x}) = \mathcal{M}[\mathbf{R}](\mathbf{x})$. Autrement dit, la machine MTU peut simuler tout jeu de règles à condition qu'il soit représenté comme une donnée sur le ruban. On appelle cela une **Machine de Turing Universelle**, et c'est le modèle théorique de tous les ordinateurs. Pour ceux-ci la mémoire est le ruban, le processeur réalise les règles MTU, et leurs programmes stockés en mémoire sont des jeux de règles de calcul **R**. L'existence d'une MTU est fondamentale. Elle exprime que le formalisme des machines de Turing est suffisamment puissant pour se décrire lui-même. Sa réalisation est elle aussi fondamentale. Elle montre qu'il est vain de distinguer programme et donnée lorsqu'ils sont interprétés par une machine de Turing.

2. (section 8) Il n'existe pas de jeu de règles **H** tel que pour tout jeu de règles **R** et valeur **v**, $\mathcal{M}[\mathbf{H}](\mathbf{R}, \mathbf{v}) = Vrai$ si $\mathcal{M}[\mathbf{R}](\mathbf{v})$ **s'arrête**, **Faux** sinon. Le problème qui serait résolu par **H** s'appelle le **Problème de l'arrêt** (*Halting Problem*). C'est le premier dont on ait montré formellement qu'il n'avait pas de solution par une machine de Turing. On dit qu'il est **indécidable**. Turing le prouve en montrant que si un jeu de règles **H** existait, on pourrait en déduire un autre, **D**, tel que $\mathcal{M}[\mathbf{D}](\mathbf{R})$ boucle ssi $\mathcal{M}[\mathbf{R}](\mathbf{R})$ s'arrête. Mais alors on voit que le calcul $\mathcal{M}[\mathbf{D}](\mathbf{D})$ boucle ssi il s'arrête ! Cette méthode de preuve, dite par **diagonalisation**, est typique de l'algorithmique.

La possibilité d'une MTU est tout simplement la possibilité d'une machine programmable, c-à-d. d'un ordinateur moderne. Il faudra attendre **John von Neumann** et ses collègues pour en voir la première réalisation concrète à la fin des années 1940.

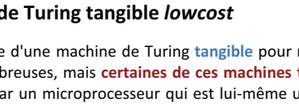
Les premiers calculateurs devaient être reconfigurés avant tout nouveau calcul. Ils disposaient de panneaux de contrôle munis de câbles enfichables et de commutateurs qui permettaient de les configurer pour un calcul particulier. De ce fait, le programme n'était pas dans la machine.

Au contraire, dans une machine de von Neumann, le programme est stocké en mémoire comme de la donnée, et comme le prévoit une MTU. Dans ce cas, l'unité de calcul de la machine ne sait réaliser qu'un jeu de règles figées, ex. **Si tu vois telle instruction fais ceci et cela**, mais la séquence effective des règles exécutées est dictée par le programme stocké en mémoire et qui peut être changé à volonté.

De nos jours, le modèle de machine s'est largement complexifié, en évoluant dans 2 directions (une autre dimension, plus spatiale, est celle de l'augmentation du nombre d'unités de calcul, les **cœurs** – ou *core*) :

1. On utilise des **machines virtuelles** qui sont des programmes dont la fonction est de réaliser les règles de calcul d'une MTU qui exécute les programmes d'un autre langage. Par exemple, les programmes Java sont exécutés par un programme appelé la **JVM** (*Java Virtual Machine*), lui-même stocké en mémoire et exécuté par l'unité centrale de la machine. Et on peut imaginer de programmer en Java la machine virtuelle d'un autre langage de programmation, et ainsi de suite. Cette pratique est largement adoptée pour s'affranchir de certains problèmes de portabilité. Ainsi, on sait aujourd'hui que sur toute machine on trouvera une JVM, ou que dans tout navigateur web on pourra trouver une machine JavaScript.
2. On a aussi pris l'habitude de concevoir les unités de calcul sous la forme de machines plus élémentaires, mais programmables. Dans ce cas, on dit qu'elles sont **micro-programmables**. De cette façon, on ne réalise pas en matériel tous les détails d'une unité de calcul ; on les programme sur une machine plus simple.

On a ainsi un empilement de MTU dans chaque ordinateur, *smartphone*, ...



La machine à calculer présentée par Turing en 1936 n'est pas encore exactement une MT au sens de ce qu'on enseigne aujourd'hui. Le concept sera épuré par des auteurs comme **Stephen Kleene** ou **Martin Davis** dans les années 1950.

De nombreuses variantes seront proposées, souvent pour apporter des facilités, mais aucune n'augmentant strictement la puissance de calcul. On peut jouer sur :

Le nombre d'états : 2 suffisent, si on n'est pas limité en nombre de symboles.

Le nombre de symboles : 2 suffisent, si on n'est pas limité en nombre de états. En pratique, le plus commode est d'avoir beaucoup des uns et des autres !

La nature des symboles et des règles : Turing lui-même utilise des abréviations qui sont essentiellement des **appels de procédure**. On peut utiliser des symboles complexes, ex. des **réels**, avec dans les règles les opérations correspondantes. On peut utiliser des **q-bits**, c-à-d. des bits quantiques. Dans le dernier cas, on peut montrer qu'une machine de Turing quantique pourra toujours être simulée en un temps exponentiel par une machine classique, ce qui réciproquement donne une indication sur les performances espérées des machine quantiques.

La nature de l'ensemble des règles : Au lieu qu'au plus une règle s'applique pour chaque couple état$_{départ}$-symbole$_{lu}$ (machine **déterministe**), on peut imaginer que plusieurs s'appliquent et que la machine sache toujours faire le choix d'une règle qui mène au but (machine **non-déterministe**). Cela peut paraître exorbitant, mais on peut rapprocher cela du principe de moindre action omniprésent dans la nature, et de toute façon ça n'apporte pas de puissance de calcul supplémentaire.

Le nombre de rubans : Un suffit, mais une machine à plusieurs rubans pourra être plus facile à programmer et pourra demander moins d'étapes de calcul.

La forme des rubans : Ils peuvent avoir une extrémité bornée, mais pas les deux. Ils peuvent être adressables pour aller directement en n'importe quelle position (modèle RAM) plutôt que d'y aller de proche en proche. Il est vraiment important que le ruban soit non borné, mais il est abusif de penser qu'il doit être infini puisque aucun calcul qui s'arrête n'utilise un ruban effectivement infini. Inversement, fixer la taille du ruban et ne s'intéresser qu'à ce qu'on peut calculer en utilisant ce ruban fini impose une sévère limitation à ce qu'il est possible de calculer.

Ces variantes peuvent accroître les performance, parfois de façon très importante, ou la concision des programmes, mais en aucun cas une fonction qui n'était pas calculable par une MT classique ne deviendra calculable par une de ces variantes de MT. Qu'être calculable résiste aussi bien à toutes ces variantes fait qu'on dit que c'est un concept **robuste**.

Année 1936, *annus mirabilis* !

En 1936, année de publication de l'article de Turing, Alonzo Church publiait *« An Unsolvable Problem of Elementary Number Theory »* et *« A Note on the* Entscheidungsproblem *»*. Il y expose une formalisation du calcul basée sur une notation de fonction rudimentaire, le λ-calcul (lambda-calcul). Il y montre comment modéliser des calculs complexes, et que le *Entscheidungsproblem* ne peut pas être résolu par le λ-calcul.

La publication de Church suit de peu celle de Turing, et ce dernier montre très vite que sa machine peut simuler le λ-calcul et vice-versa. Les deux formalismes coïncident donc au sens où ils permettent de calculer les mêmes fonctions. Turing puis Church sont vite convaincus qu'il existe une notion intrinsèque de **fonction calculable**, et que c'est cela qui est réalisée par la machine de Turing ou le λ-calcul. On appelle cela la **Thèse de Church-Turing** (***Church-Turing Thesis***, **CTT**).

On dit d'un formalisme qui équivaut à la MT qu'il a la **puissance du calculable**, ou qu'il est **Turing-équivalent**. La plupart des langages de programmation ont la puissance du calculable. En ce sens, dire que tel langage de programmation est plus puissant qu'un autre est le plus souvent ridicule ! il faut vraiment le faire exprès pour concevoir un langage de programmation qui n'ait pas la puissance du calculable. Le calcul relationnel implémenté par **SQL** est une de ces exceptions.

En 1936 toujours, Emil Post publiait *« Finite Combinatory Processes – Formulation 1 »*. Il y décrit encore une autre formalisation du calcul, qu'il envisageait comme le premier maillon d'une chaîne de formalismes toujours plus puissants, c-à-d. où la **Formulation N** peut résoudre des problèmes que ne peut pas résoudre la **Formulation N-1**. Il est vite apparu que la **Formulation 1** était Turing-équivalente, mais Post a d'abord contesté la CTT en espérant qu'une Formulation d'un rang plus élevé ait vraiment une puissance supérieure.

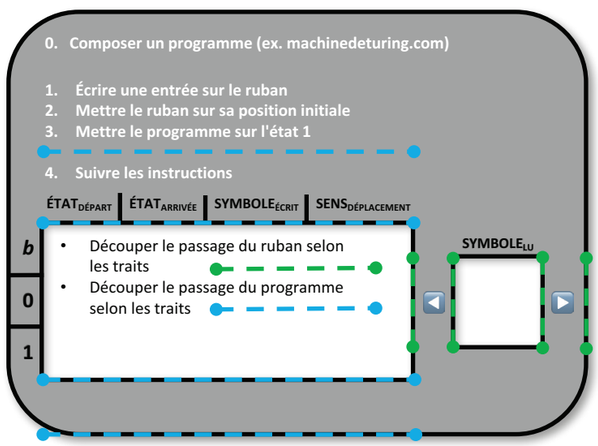
Depuis, énormément d'inventivité a été déployée pour décrire des formalismes de calcul, avec toujours le même résultat ; ils sont au plus équivalents à la MT. La Thèse de Church-Turing s'en trouve donc renforcée, mais elle reste un **énoncé non prouvé**. Church a proposé d'en faire un **axiome**, un peu comme les Lois de la mécanique de Newton ou les Principes de la thermodynamique.

On peut retenir de ces formalismes qu'ils sont souvent extrêmement simples, voire minimalistes. Il faut vraiment peu de chose pour avoir la puissance du calculable. La variété des formalismes proposés est immense. On retiendra à nouveau que le concept de **calculable** est vraiment robuste.

Une machine de Turing tangible *lowcost*

On a parfois envie d'une machine de Turing **tangible** pour mieux la comprendre. Il en existe de nombreuses, mais **certaines de ces machines trichent !** En effet, elles sont contrôlées par un microprocesseur qui est lui-même une MTU. Cela revient à contempler un modèle de machine de Watt où tous les rouages et les bielles sont visibles, mais où la force motrice vient d'un autre moteur. La proposition de **Marc Raynaud** est une des plus réussies des machines de Turing tangibles et honnêtes (**machinedeturing.com**). Cette machine est entièrement réalisée à l'aide de composants dont l'équivalent existaient en 1936 : roues dentées, relais, moteurs électriques, ... Elle reste cependant d'un usage délicat. On propose ici une machine de Turing **tangible** et **rustique**.

Ce ne sera pas une MTU, mais une machine de Turing générique dont les règles sont modifiables. En théorie, il n'existe pas de limite au nombre de symboles qu'une MT peut reconnaître, ni au nombre d'états, ni au nombre de règles. En pratique, il faudra se limiter. Comme pour la machine de Marc Raynaud, on choisit 12 états maximum, numérotés de 1 à 12, et les 3 symboles ***b***, **0**, **1**, donc au plus 36 règles.



Conclusion

La machine de Turing marque une date dans la compréhension de ce que l'on peut calculer automatiquement. Elle est restée un outil très fructueux pour réfléchir formellement aux questions de **calculabilité**, étude de ce qui est calculable ou non, et de **complexité**, étude de combien coûte le calcul.

Paradoxalement, en répondant négativement à la question **« Peut-on tout calculer automatiquement ? »**, mais en le faisant avec précision, la machine de Turing ouvre la porte à la réalisation approchée des fonctions qui ne sont pas calculables automatiquement. Vu comme cela, 80 ans de travaux en algorithmique et les derniers progrès de l'intelligence artificielle ne viennent pas contredire le résultat de Turing. Au contraire, ils le complètent en explorant le paysage des solutions approchées ou empiriques, ou semi-automatiques.

Aussi géniale soit-elle, la machine de Turing ne dit pas tout sur tout. En particulier, elle ne dit pas grand chose des **calculs qui ne terminent pas**. Pourtant, nous sommes entourés de calculs qui ne terminent pas, ou dont nous n'attendons pas la fin, ex. les systèmes d'exploitations, les services du web, ou Internet, et plus généralement tous les systèmes interactifs. Ces systèmes entrelacent calculs et présentations de résultat, alors qu'une machine de Turing ne présente un résultat que lorsque elle s'arrête.

Bibliographie

- *« On Computable Numbers with an Application to the* Entscheidungsproblem *»*, Turing (1936), à lire dans *« The Annotated Turing »*, Petzold (Wiley, 2008). Probablement la meilleure approche pour comprendre en profondeur le texte de Turing.
- *« The Feeling of Power »*, Asimov (1958). Une approche poétique du calcul. Là où Turing extrapole le calcul automatique du calcul manuel, Asimov imagine qu'à une époque où plus personne ne sait compter à la main, un informaticien extrapole du comportement des machines les opérations du calcul manuel. Qu'un humain puisse réaliser les mêmes calculs qu'une machine paraît alors formidable.
- *« Computers LTD, what they really can't do »*, Harel (OUP, 2003). Une présentation très vivante de ce que les calculateurs ne peuvent vraiment pas faire.
- *« The new Turing Omnibus »*, Dewdney (Owl Books, 2001). Quantité de sujets informatiques y sont traités, dont la Machine de Turing évidemment !
- *« Calculateurs, calculs, calculabilité »*, Ridoux et Lesventes (Dunod, 2008). Un cours sur la calculabilité qui remplace la Machine de Turing par le savoir-faire du programmeur.

Mode d'emploi :

- Lire cette page de documentation (histoire, prérequis logiques, opinion), comprendre, apprendre le cas échéant.
- Préparer les plis (voir traits gris rentrants et traits **rouges** saillants au verso de cette page).
- Découper selon le trait rouge entre les deux ● du verso de cette page, puis achever le pliage.
- S'occuper du découpage de la machine de Turing tangible *lowcost* en dernier.

Numérique, digital, binaire, ..., ou symbolique ?

Aujourd'hui, on ne dit pas « l'informatique » mais « le numérique » ! Ce terme fait suite à digital et binaire. Dans tous les cas, l'idée de nombre est mise en avant. De même, on ne programme plus, mais on code, ce qui par voisinage de sens ramène à chiffrer, et encore aux nombres.

Nous pensons que cette mode renforce une perception ésotérique de l'informatique, une forme de numérologie, et qu'elle contribue par là à l'incompréhension de ce qu'est l'informatique,

Ah vous êtes informaticien.ne ! Vous voulez vraiment représentez tout avec des nombres ?
N'est-ce pas réductionniste ? Vous ne pourrez jamais représenter
la vibration des couleurs des tableaux de Turner avec des nombres !

...alors qu'on peut passer une vie d'informaticien.ne sans jamais se soucier de coder quoi que ce soit avec des nombres.

Les travaux de Turing montrent un visage beaucoup plus intéressant ; celui du raisonnement sur les symboles et les lois qui les organisent, selon la tradition très ancienne de recherche d'éléments premiers et de lois de production d'éléments dérivés. C'est ce que font les physiciens et chimistes et leurs ancêtres alchimistes depuis l'antiquité, les linguistes depuis Panini (grammaire du sanskrit, il y a environ 6000 ans), les biologistes, physiologistes et médecins depuis toujours, et bien sûr les mathématiciens. Tous créent des symboles pour donner des noms aux choses, et cherchent des lois pour les organiser. Ce qui est propre à l'informatique est qu'elle a fait de l'invention de symboles une industrie, et qu'elle formalise des calculs en tant que tels, et pas seulement leurs résultats.

Lire Turing est alors très éclairant. Il utilise autant de symboles qu'il veut. Il se passe complètement de système de numération pour définir ses différentes machines ; quand elles doivent compter, c'est avec des bâtons en répétant une marque autant de fois que nécessaire. Et ses premières machines manipulent soit des programmes, soit des formules logiques ! Bien sûr, le prétexte premier de son article est d'énumérer des nombres (*computable numbers*), mais ce n'est pas pour cela qu'on se souvient de son article ! Les origines du binaire, digital, numérique ne sont vraiment pas à aller chercher chez Turing, mais plutôt chez Claude Shannon et dans l'histoire des premiers ordinateurs, mais l'informatique n'est pas plus la science des ordinateurs que l'astronomie n'est celle des télescopes (Michael Fellows et Ian Parberry, 1993).

Alan Turing

Alan Mathison Turing est né en 1912 (Londres) et mort en 1954 (Wilmslow). Paradoxalement, c'est pour ses activités les moins publiques qu'il est le plus connu. Pendant la seconde guerre mondiale, il a participé à l'effort de guerre britannique en contribuant à l'analyse et au décodage des messages que la marine allemande chiffrait à l'aide de la machine *Enigma*. Cet aspect de sa vie était couvert par le secret et n'a été déclassifié que vers les années 1970-80, mais plusieurs hommages lui ont été rendus pour son rôle éminent dans l'accélération de la fin de la guerre. Il est aussi connu pour sa vie privée et en particulier pour son homosexualité, qui était considérée comme un crime à l'époque. Il sera condamné en 1952 à une « castration chimique ». En 1954, il est retrouvé mort chez lui, des suites d'un suicide. En 2009, le gouvernement britannique reconnaîtra la brutalité et l'injustice du traitement imposé à Alan Turing, qui sera gracié à titre posthume en 2013.

Cependant, la facette de sa vie qui nous intéresse le plus ici est sa contribution majeure à la science informatique, et cette facette est complètement publique. Le modèle de calcul qu'il propose en 1936 pour répondre à une question de logique mathématique est toujours pertinent, et les premières conséquences que Turing en tire sont toujours des résultats phares de l'informatique théorique. À partir de la fin des années 1940, il participera aux recherches sur les premiers ordinateurs. Il se déclarera lui-même constamment surpris de ce qu'il est possible de leur faire faire, et cela le conduira à formuler la première idée d'intelligence artificielle. Il aura donc à la fois démontré formellement que des tâches bien formalisées ne sont pas réalisables automatiquement, et fait le pari que des tâches mal formalisées, mais réputées intelligentes, le seront !

① Calcul des propositions (ou logique des propositions)

Le calcul des propositions est une logique d'énoncés élémentaires auxquels on peut attribuer une valeur de vérité, Vrai ou Faux, et qu'on peut articuler en utilisant des connecteurs logiques qui jouent le rôle des conjonctions de coordinations en français pour former des énoncés complexes dont on peut calculer à leur tour la valeur de vérité.

Variables propositionnelles (ou atomes) **:** Ce sont les formules élémentaires. Dans la suite, on les note par des lettres minuscules, ex. **a**, **b**, **c**, mais absolument rien ne l'exige.

Formules propositionnelles : Ce sont des formules, atomiques ou non, qui sont reliées par des connecteurs logiques, généralement ∧, ∨, ¬, ou ⇒.

- **conjonction,** $\phi_1 \wedge \phi_2$ **:** relie deux formules pour en former une troisième qui vaut Vrai ssi les deux premières le sont.
- **disjonction,** $\phi_1 \vee \phi_2$ **:** relie deux formules pour en former une autre qui vaut Vrai ssi au moins une des deux premières l'est.
- **implication,** $\phi_1 \Rightarrow \phi_2$ **:** relie deux formules pour en former une autre qui vaut Faux ssi ϕ_1 vaut Vrai alors que ϕ_2 vaut Faux.
- **négation,** $\neg\phi$ **:** constitue une formule qui vaut Faux ssi ϕ vaut Vrai.

Tautologie et satisfaisabilité : les formules propositionnelles peuvent être vues comme des fonctions des variables qu'elles contiennent : $\{Vrai, Faux\}^n \longrightarrow \{Vrai, Faux\}$ pour une formule à n variables. Même si il existe une infinité de formules à n variables, il n'y a que $2^{(2^n)}$ fonctions différentes, et on peut représenter chacune d'entre elles par une table de vérité. C'est un tableau à 2^n lignes, et n+1 colonnes. Ex. **(aVb)∧c**, **aV¬a**, et **a∧¬a** sont représentées par les tables de vérité ci-contre. Dans leurs n premières colonnes, les lignes représentent toutes les entrées possibles et dans leur n+1-ème colonne, elles représentent ce que vaut la fonction pour chaque entrée.

a	b	c	(aVb)∧c
Vrai	Vrai	Vrai	Vrai
Vrai	Vrai	Faux	Faux
Vrai	Faux	Vrai	Vrai
Vrai	Faux	Faux	Faux
Faux	Vrai	Vrai	Vrai
Faux	Vrai	Faux	Faux
Faux	Faux	Vrai	Faux
Faux	Faux	Faux	Faux

a	aV¬a
Vrai	Vrai
Faux	Vrai

a	a∧¬a
Vrai	Faux
Faux	Faux

On voit que toutes les lignes de la table de **aV¬a** ont la valeur Vrai en position de résultat. Cette formule est une tautologie ; elle vaut Vrai pour toutes ses entrées, c-à-d. pour toutes les interprétations de ses variables. Noter que **aV¬a** est tautologique sans qu'on sache qui de **a** ou de **¬a** est Vrai. Cela peut être jugé problématique et il faudra un autre cadre logique pour traiter cela : la logique intuitionniste. Toutes les lignes de la table de **a∧¬a** ont la valeur Faux. Cette formule est contradictoire ou absurde. Enfin, des lignes de la table de **(aVb)∧c** peuvent avoir Faux mais au moins une a Vrai. Cette formule est satisfaisable. Noter que, pour tout ϕ, ϕ ou $\neg\phi$ est satisfaisable, que ϕ et $\neg\phi$ peuvent l'être toutes deux (ex. **(aVb)∧c**), et que ϕ est tautologique ssi $\neg\phi$ n'est pas satisfaisable (= est contradictoire).

② Calcul des prédicats (ou logique du 1er ordre)

Le calcul des prédicats permet d'exprimer des jugements (ex. **∀X,Y,Z. X>Y∧Y>Z ⇒ X>Z** et **∀X,Y. père(X,Y)∧avare(X) ⇒ prodigue(Y)**), et de décider formellement si ils sont tautologiques (c-à-d. vrais pour toutes les interprétations des symboles) ou non. Le calcul des prédicats n'a commencé à être formalisé qu'à la fin du XIXe siècle, par Gottlob Frege, puis au début du XXe siècle avec Alfred Whitehead et Bertrand Russell, David Hilbert et Wilhelm Ackermann, et Kurt Gödel. Les formules du calcul des prédicats sont formées d'énoncés élémentaires constitués d'un prédicat, c-à-d. un jugement (ex. **être avare**) appliqué à un ou plusieurs sujets, des connecteurs du calcul des propositions (ex. ∧, ∨, ¬, et ⇒) et des quantificateurs ∀ et ∃. On constitue des ensembles de formules qu'on appelle théories. À proprement parler, la tautologie d'une formule s'évalue par rapport à une théorie, qui va par exemple énoncer les propriétés logiques de **<**, **père**, **avare** et **prodigue**. On veut savoir si une formule est une conséquence logique d'une théorie et pas seulement une conséquence d'une certaine interprétation des symboles.

Formules quantifiées : Ce sont des formules dont la valeur de vérité s'évalue par rapport à un ensemble d'objets, un domaine, plutôt que par rapport à un objet. Syntaxiquement, une quantification lie une variable, comme le font $\int \ldots dx$ ou $\partial \ldots / \partial x$. Sémantiquement, les quantifications sont définies comme suit :

- **quantification universelle,** $\forall x \,.\, \phi\,(x)$ **:** constitue une formule qui vaut Vrai ssi $\phi\,(e)$ vaut Vrai pour tout élément e du domaine. Si le domaine est fini, la quantification universelle est juste une conjonction des applications de ϕ à tous les éléments du domaine.
- **quantification existentielle,** $\exists x \,.\, \phi\,(x)$ **:** constitue une formule qui vaut Vrai ssi $\phi\,(e)$ vaut Vrai pour au moins un élément e du domaine. Si le domaine est fini, la quantification existentielle est juste une disjonction des applications de ϕ à tous les éléments du domaine.

Démonstration automatique dans la logique des prédicats : dès le début du XXe siècle, on s'est demandé si il était possible de décider qu'une formule du calcul des prédicats était une tautologie en utilisant une méthode purement algorithmique, c-à-d. systématique, qui finit toujours avec un résultat décisif (c-à-d. oui ou non, mais pas je-sais-pas), et sans nécessité d'un jugement externe. On appelle cela une procédure de décision. Concernant la partie propositionnelle, on sait qu'une telle procédure existe, même si elle est très coûteuse dans le pire des cas (mais alors, on parle de complexité algorithmique et pas de calculabilité). Mais concernant les quantifications, la difficulté réside dans des conditions comme Vrai pour tous les éléments du domaine ou Vrai pour au moins un élément du domaine, et ce pour tous les domaines, y-compris ceux qui ne sont pas finis. Comment explore-t-on un domaine infini en un nombre d'opérations fini ? Comment le faire pour tous les domaines ? Voir le bloc ③ de ce document.

③ Les réductions du *Entscheidungsproblem*

Sitôt posé le problème de la décision du calcul des prédicats (le *Entscheidungsproblem*), on a essayé de le réduire à des problèmes plus simples. Ce programme de recherche avait commencé avant la formalisation par Hilbert de ce problème, car celle-ci ne faisait que formaliser complètement la logique du 1er ordre qui était dans l'air du temps depuis Frege en 1879. Et il a été poursuivi bien après le résultat négatif de Turing, car si celui-ci démontre qu'il n'existe pas de méthode complètement automatique pour décider tout le calcul des prédicats, cela n'empêche pas de rêver d'une méthode qui serait semi-automatique, ou d'une méthode automatique qui déciderait d'une partie seulement du calcul des prédicats, ou d'une méthode automatique qui saurait répondre oui, non ou je-sais-pas (le moins souvent possible).

En 1915-1920, Leopold Löwenheim puis Thoralf Skolem démontrent que si une formule du calcul des prédicats est satisfaisable dans un domaine infini, elle l'est pour toutes les cardinalités de l'infini. En particulier, elle l'est dans un domaine au plus dénombrable, c-à-d. pas plus gros que $\mathbb{N}$.

En 1930, Jacques Herbrand montre qu'une formule du calcul des prédicats est satisfaisable si et seulement si elle l'est dans un domaine dénombrable qui a une certaine forme, facile à énumérer, appelé maintenant son domaine de Herbrand. En conséquence, une formule insatisfaisable dans son domaine de Herbrand est insatisfaisable tout court, et si **¬F** n'est pas satisfaisable dans son domaine de Herbrand alors **F** est une tautologie.

En 1965, Alan Robinson montre une transformation effective des formules du calcul des prédicats en des formules assez proches de celles du calcul des propositions qui permet d'appliquer une procédure automatique qui sait détecter si une formule n'est pas satisfaisable dans le domaine de Herbrand, et donc dans aucun modèle. Cette procédure, appelée résolution, peut ne pas détecter qu'une formule est vraie pour le modèle de Herbrand. On appelle cela un semi-algorithme. Cela illustre qu'un résultat d'infaisabilité, comme celui de Turing, ne doit pas être interprété comme une catastrophe absolue, mais plutôt comme un challenge pour approcher au plus près ce qu'il est impossible de faire.

Le *Entscheidungsproblem*

En 1900, David Hilbert présente au congrès international de mathématiques ses vues sur ce que seront les grands challenges mathématiques du XXe siècle. Il dresse alors une liste de 23 problèmes, qu'on appelle désormais problèmes de Hilbert. Certains ont pu être résolus au cours du XXe siècle au prix d'un travail important, ex. en 1970, la non-décidabilité de la résolution des équations diophantiennes (équations polynomiales en nombre entier, 10e problème de Hilbert) par Youri Matyasevich en s'appuyant sur des résultats de Julia Robinson. D'autres étaient si difficiles qu'ils sont devenus des problèmes du XXIe siècle, et ce sont vus dotés d'un prix de 1 000 000 $ pour qui les résoudrait. D'autres étaient tout simplement mal posés, et n'ont pas été de vrais challenges !

En 1928, Hilbert et Wilhelm Ackermann proposent dans leur Principes de logique mathématique une formalisation du calcul des prédicats qui pourrait servir d'intermédiaire pour le 2nd problème, la non-contradiction des axiomes de l'arithmétique. Se pose alors la question de l'automatisation de la démonstration dans le calcul des prédicats : le *Entscheidungsproblem*. Au début, il allait de soi pour Hilbert que la question n'était pas de savoir si c'était possible, mais par quel moyen : *Wir müssen wissen, Wir werden wissen*.

N'oubliez jamais !

Un modèle est toujours imparfait, parfois utile, et c'est tout ce qu'on peut lui demander (d'après George Box). Les programmes sont des modèles d'une forme de réalité.

L'informatique ne peut travailler que sur la représentation des choses. Ex., la machine de Turing universelle opère sur une représentation des programmes. Mais il faut des capteurs et des actionneurs pour toucher les vraies choses. Par contre, opérer sur des représentations de représentations de … est courant. Ex., 440 est une représentation d'un entier qui peut être la représentation de la mesure d'une fréquence, tout comme 1B8, et La_3 pourrait être une autre représentation de la même fréquence. Il faut toujours se demander quelle représentation a du sens par rapport à ce pourquoi est utilisé le modèle.

Modèles de programme et de ruban à photocopier

ÉTAT$_{DÉPART}$	ÉTAT$_{ARRIVÉE}$	SYMBOLE$_{ÉCRIT}$	SENS$_{DÉPLACEMENT}$
1	règle pour le symbole b		
	règle pour le symbole 0		
	règle pour le symbole 1		
2	règle pour le symbole b		
	règle pour le symbole 0		
	règle pour le symbole 1		
3	règle pour le symbole b		
	règle pour le symbole 0		
	règle pour le symbole 1		
4	règle pour le symbole b		
	règle pour le symbole 0		
	règle pour le symbole 1		
5	règle pour le symbole b		
	règle pour le symbole 0		
	règle pour le symbole 1		
6	règle pour le symbole b		
	règle pour le symbole 0		
	règle pour le symbole 1		
7	règle pour le symbole b		
	règle pour le symbole 0		
	règle pour le symbole 1		
8	règle pour le symbole b		
	règle pour le symbole 0		
	règle pour le symbole 1		
9	règle pour le symbole b		
	règle pour le symbole 0		
	règle pour le symbole 1		
10	règle pour le symbole b		
	règle pour le symbole 0		
	règle pour le symbole 1		
11	règle pour le symbole b		
	règle pour le symbole 0		
	règle pour le symbole 1		
12	règle pour le symbole b		
	règle pour le symbole 0		
	règle pour le symbole 1		