

UNIVERSITÉ FRANÇOIS RABELAIS

DÉPARTEMENT INFORMATIQUE DE BLOIS

MASTER 1 - SIAD

Vérité & Démonstration

De l'ambivalence de la notion vérité et l'implication des machines abstraites
dans le processus de prouvabilité

Auteur :

Clément MOREAU

Encadrants :

Jean-Yves ANTOINE

Evelyne MOREAU

Évaluateurs :


Nicolas LABROCHE

Dominique LI

*
* *

21 juillet 2017

1 Entre recherche de formalisme et vérité

 n 1879, la *Begriffsschrift* de Frege donne le premier exemple de langage logique permettant l'écriture de raisonnements formalisés. Cette entreprise était, à l'initial, éloignée de toute considération relative aux machines. Cet instrument devait uniquement servir à libérer la pensée formelle (i.e. mathématique) du joug des langues naturelles qui opèrent des imperfections, singularités grammaticales mais aussi des usages non formels comme le sous-entendu ou le présupposé contraire à la logique.

À ce titre, l'idéographie construite par Frege se révèle comme étant le microscope du logicien, un artifice transparent à travers lequel on peut mener une observation. En outre, l'idéologie fréگیenne mais aussi celle de Russell et Whitehead à la rédaction des *Principia Mathematica* est une approche syntaxique de la logique permettant de vérifier l'exactitude d'un raisonnement, sans se soucier de la véracité des propositions, le but étant la production de théorèmes à l'aide des axiomes fournis et des règles d'inférences établies. Cependant, l'objectif ne réside pas [encore] dans la mécanisation des procédures déductives ou dans l'interprétation de constructions formelles en terme de machine. Pour qu'une correspondance profonde et féconde puisse, dès lors, être élaborée entre constructions logiques et machines, il fallait qu'il s'impose l'idée d'une l'orientation des raisonnements mathématiques vers des procédures *finies* et donc vers une exigence d'effectivité et la mise en place d'un langage fait de symboles et de règles d'inférence pouvant faire l'objet d'une étude logico-mathématique extérieure et servant de fondations pour l'élaboration de théories logiques.

Ces points furent réalisés partiellement par le programme de Hilbert qui avait pour idée d'édifier une formalisation globale des mathématiques par des moyens finis, mécaniquement vérifiables. S'ensuivent alors un ensemble de conséquences dont l'importance n'apparaît, pour certaines d'entre elles, qu'après l'ar-

ticle de Gödel publié en 1931 ou même après la solution de l'*Entscheidungsproblem* par Church et Turing en 1936, mais qui figurent des transformations ultérieures de la logique et de la notion de machine dans cette discipline. Ainsi nous sommes menés à un des points nodaux de notre réflexion : *la logique ne traite pas seulement de la vérité, du raisonnement et du langage mais aussi du calcul, des algorithmes et des machines*¹. Dès lors on peut se demander de quelles performances les machines logico-informatiques sont-elles capables ? Et quelles vérités ces machines peuvent-elles nous apporter ?

La question de la véracité d'une théorie logique tient dans l'approche sémantique ou théorie des modèles. On s'intéresse alors à la propagation de la vérité en se basant uniquement sur la forme des raisonnements. Le but de cette approche est alors de déterminer des modèles, soient des interprétations rendant une formule vraie. Ainsi, une théorie possédant au moins un modèle est dite satisfaisable. Si l'approche sémantique et syntaxique semblent *a priori* disjointes, il n'en est rien, car la valeur de vérité ne dépend que de la forme du raisonnement. En outre, ces approches partagent aussi les notions de consistance et de complétude qui questionnent les liens entre démonstration et vérité d'une théorie. On examinera alors les propriétés de consistance et de complétude qui sont fondamentales pour l'informatique et les mathématiques modernes ; propriétés qui ont également été étudiées par Gödel dans sa thèse de doctorat et qui mettent en jeu les théorèmes de complétude et incomplétude.

La complétude pose le problème de savoir si un énoncé est universellement valide, c'est-à-dire vrai pour toute interprétation de la théorie. Or, cette exigence mène à la recherche d'une procédure effective de validation de ce problème, on est alors amené à caractériser la notion de « procédure effective ». Cette caractérisation sera effectuée par Turing lors de son étude du problème de la décision où il remarque que tout calcul arithmétique usuel est l'application d'une procédure effective, bien que la réciproque ne soit pas vraie. Il

¹On confondra ici volontairement la notion de machine en tant que construction formelle (i.e. dispositif abstrait) et entité matérielle (i.e. ordinateur)

donne ainsi une représentation codifiée de ce qu'est un calcul ; cette représentation prend la forme d'une machine et la procédure effective celle d'une succession d'opérations opérée par la machine sur un alphabet de symboles et un ruban semi-infini de travail. Chaque machine définit donc, par son tableau de commandes et son alphabet, une fonction. À l'inverse, étant donné une fonction, on pourra se demander s'il existe une machine de Turing capable de la calculer. On dira qu'une fonction est Turing-calculable s'il existe une machine de Turing associée capable de représenter cette fonction. Cependant, la solution de l'*Entscheidungsproblem* suppose que toute procédure effective, quelle qu'elle soit, puisse être effectuée par une machine de Turing ou encore, en terme de fonction, que toute fonction pour laquelle il existe une procédure effective soit Turing-calculable. En outre, en esquissant, dans ses grandes lignes, la solution du problème de la décision et en formalisant la notion de calcul, on s'apercevra comment une notion abstraite de machine vient s'insérer dans la démonstration d'un théorème logique.

2 De la différence entre prouvabilité et vérité

Le problème des rapports entre démonstration et vérité se pose dans le programme de Hilbert tel que l'on puisse concevoir un système dans lequel les propositions démontrables soient uniformellement valides. Le théorème de complétude montre que cette exigence peut effectivement être satisfaite. Pour les vérités arithmétiques, on souhaite de même construire un système cohérent et complet : tout énoncé arithmétique serait alors démontrable ou réfutable, par la démonstration de sa négation. Dans cette perspective, la vérité des propositions arithmétiques ne serait pas garantie par une fonction d'interprétation mais par une démonstration de cohérence. Or, on présuppose alors une équivalence entre vérité et proposition formellement démontrable ; équivalence qui fut réfutée et prouvée par Gödel dans ses théorèmes d'incomplétude et qui établissent une distinction (mais aussi correspondance) entre le vrai et le démontrable.

Premier théorème d'incomplétude. —

Si \mathcal{T} est une théorie du premier ordre cohérente, récursivement axiomatisable et contenant l'arithmétique de Robinson, alors \mathcal{T} est incomplète, en ce sens qu'il existe une formule close G dans le langage de \mathcal{T} telle qu'aucune des formules G et $\neg G$ n'est conséquence des axiomes de \mathcal{T} .

Second théorème d'incomplétude. —

Si \mathcal{T} est une théorie du premier ordre cohérente, récursivement axiomatisable et contenant l'arithmétique de Peano, alors la formule $\text{Cons}\mathcal{T}$ (qui dans le langage de \mathcal{T} exprime la cohérence de la théorie \mathcal{T}) n'est pas une conséquence des axiomes de \mathcal{T} .

2.1 Le premier théorème d'incomplétude

L'interprétation des théorèmes d'incomplétude concerne avant tout la philosophie des mathématiques mais elle concerne également l'informatique et son lien avec la logique sur la manipulation des structures finies - des termes, démonstrations, énoncés pour la logique et des structures de données pour l'informatique.

On se propose en annexe de définir plus précisément et approfondir quelques concepts à partir d'un examen des moyens démonstratifs utilisés dans la preuve des théorèmes d'incomplétude, le but n'étant pas de dresser un cadre formel mais davantage de comprendre les engrenages généraux et le mécanisme interne.

Le premier théorème fait intervenir une méthode de diagonalisation, assez proche de la méthode utilisée par Cantor dans sa démonstration de la non-dénombrabilité des nombres réels, et une formule semblable à celle du paradoxe du menteur.

On considère une formule close G dans une théorie \mathcal{T}^2 telle que :

²On ne s'intéresse ici qu'aux théories du premier ordre construites sur le langage de l'arithmétique. Dans ce cadre, une théorie \mathcal{T} est essentiellement un ensemble de formules closes, qu'on appelle les axiomes de \mathcal{T} . Une théorie \mathcal{T} permet de dé-

$$G \equiv \neg \exists x \text{Dem}(x, \ulcorner G \urcorner)$$

En substance, la formule G nous dit : “Je ne suis pas démontrable”. On retrouve ici l’idée du paradoxe du menteur.

La formule G ne conduit pas [directement] à une contradiction, car elle ne parle pas de *vérité* mais de *prouvabilité*. On évite ainsi la contradiction et donc l’inconsistance de \mathcal{T} . Reste à prouver que ni G ni $\neg G$ ne sont démontrables, ce que fait Gödel par l’absurde en montrant que le cas échéant, l’hypothèse de ω -consistance serait contredite. Dès lors, tout système récursif qui est suffisamment puissant, comme l’arithmétique de Peano, ne peut pas être à la fois cohérent et syntaxiquement complet³.

2.2 Le deuxième théorème d’incomplétude

Au niveau de la portée symbolique du théorème et de ses implications épistémologiques, si la formule G en elle-même n’est pas digne d’intérêt car elle est la résultante d’une construction très artificielle, son existence est en revanche très instructive d’un point de vue à la fois philosophique et technique.

Sur le plan philosophique, elle nous enseigne que même un système formel défini de la manière la plus rigoureuse est capable d’engendrer du bruit, c’est à dire des objets (i.e. des formules) dont le contenu métamathématique dépasse le cadre donné par le système (d’où leur indécidabilité dans le système donné). Sur le plan technique, l’énoncé exprimant la $(\omega-)$ consistance de \mathcal{T} peut-être traduite par une formule de la théorie \mathcal{T} elle-même que l’on notera

duire un certain nombre de théorèmes par voie de conséquences logiques. Formellement, on appelle une démonstration de la formule Φ dans la théorie \mathcal{T} toute dérivation D dont la conclusion est de la forme $\Gamma \vdash \Phi$, où Γ est un ensemble (fini) d’axiomes (ou théorèmes précédemment démontrés) de \mathcal{T} . Dès lors, Φ est un théorème de \mathcal{T} , ce que l’on note $\mathcal{T} \models \Phi$. On dit que la théorie \mathcal{T} est cohérente si $\neg \exists \Phi \in \mathcal{T} | \mathcal{T} \models \Phi \wedge \mathcal{T} \models \neg \Phi$, soit qu’il n’existe pas de formule où celle-ci et sa négation soient toutes deux conséquence de la théorie. Dans le cas contraire, \mathcal{T} est dite incohérente.

³La complétude syntaxique est décrite en annexe par le biais des relations représentables dans \mathcal{T} . On peut également définir la complétude syntaxique d’une théorie \mathcal{T} telle que $\forall \Phi \in \mathcal{T} | \mathcal{T} \models \Phi \vee \mathcal{T} \models \neg \Phi$.

Cons. De plus Gödel remarque que les moyens de démonstration utilisés pour la preuve du premier théorème d’incomplétude ne dépassent pas ceux dont dispose l’arithmétique élémentaire (celle de Peano). Il est donc possible de formaliser dans \mathcal{T}_x (théorie résultante de l’ajout de l’ensemble récursif d’axiomes x) la preuve de la formule

$$\text{Cons} \rightarrow \neg \exists x \text{Dem}(x, \ulcorner G \urcorner)$$

qui est l’expression d’une partie du théorème d’incomplétude. Par conséquent, si Cons est démontrable, $\neg \exists x \text{Dem}(x, \ulcorner G \urcorner)$ le serait aussi, contrairement à ce qu’énonce le premier théorème. Dès lors, le second théorème d’incomplétude énonce que si \mathcal{T}_x est $(\omega-)$ consistant, alors la formule Cons qui affirme cette propriété de \mathcal{T}_x est indémontrable dans ce système.

2.3 L’idée d’une automatisisation du processus démonstratif

Que résulte-t-il de ces deux théorèmes et de leur démonstration ? La première est la mise en vedette de l’idée de codage dont on connaît aujourd’hui la fortune en informatique et en théorie de la calculabilité : les fonctions de transition des machines de Turing, les entiers et les fonctions numériques sont codés par des λ -termes, les données sont représentées symboliquement, etc... Ainsi, Gödel marque du même coup les limites de toute tentative pour mécaniser le raisonnement par la voie de systèmes formels ou d’un moyen équivalent car le codage et la réduction de certaines questions logiques traditionnelles à un calcul arithmétique révèlent les limites inhérentes à toute conception semblable au calcul logique. Et, de ce point de vue, la preuve des théorèmes d’incomplétude se présente comme la première étape d’une étude sur les capacités et incapacités des machines menée dans la théorie de la démonstration.

3 De la décidabilité algorithmique des problèmes

La notion de machine informatique peut donc être observée d’un point de vue abstrait et purement logique. Les machines de Turing prolongent cette idée d’étude des procédures effectives en vue de la résolution du problème de la décision.

Comme établit précédemment, Turing construit un modèle de calculateur abstrait sensé cristalliser, par le biais d’un codage et la notion de procédure effective, la notion de calcul au sens arithmétique du terme mais la portée de cette entreprise dépasse très largement son cadre initial. Turing met au point ici une méthode de caractérisation de l’ensemble des fonctions calculables, avec ce concept, on a réussi pour la première fois à donner une définition absolue d’une notion épistémologique intéressante (i.e. qui ne dépende pas du formalisme choisie)⁴.

3.1 Formalisation de l’idée calcul et de procédure effective

Mais comment exprimer formellement que la machine de Turing M est une procédure de calcul pour la fonction numérique F à p places⁵ ?

Soit Σ , l’alphabet de M , i un codage des entiers dans l’ensemble Σ^* des mots finis sur Σ et j un codage de l’ensemble des p -uplet d’entiers dans Σ^* ⁶. Soit $\vec{k} = (k_1, \dots, k_p)$, un p -uplet d’entiers quelconques, et $j(\vec{k})$ son code. On note $M(j(\vec{k}))$ le résultat de la procédure M appliquée à $j(\vec{k})$ si ce résultat existe (ce que l’on notera $M(j(\vec{k})) \downarrow$), et l’on écrit $M(j(\vec{k})) \uparrow$ si la procédure de la machine M appliquée à $j(\vec{k})$ ne produit aucun résultat ou si le résultat $M(j(\vec{k}))$ n’est le code d’aucun entier par le codage i . On dit que M est une procédure de calcul pour F si et seulement si :

1. Si $F(\vec{k})$ est défini, alors $M(j(\vec{k})) \downarrow$ et $M(j(\vec{k})) = i(F(\vec{k}))$;
2. Si $F(\vec{k})$ n’est pas définie, alors $M(j(\vec{k})) \uparrow$.

F est alors Turing-calculable.

Cependant, comme nous l’avons vu précédemment, cette construction ne permet pas *a priori* de savoir si toute fonction calculable, c’est-à-dire toute fonction pour laquelle il existe une procédure déterminée par des moyens finis permettant de calculer sa valeur en chacun des points de son domaine, est Turing-calculable.

Ce qui reste informel dans une telle définition, c’est la nature des “moyens finis” susceptibles d’être utilisés, et c’est précisément ce qui est rendu explicite et formellement exprimable par les machines de Turing de par leur construction.

La thèse avancée par Church et Turing en 1936 est que ces moyens extrêmement restreints suffisent à couvrir toutes les procédures effectives possibles (i.e tout problème de calcul fondé sur une procédure algorithmique peut être résolu par une machine de Turing). Comme le souligne l’article de 1936, la thèse de Church-Turing n’est pas susceptible d’être mathématiquement démontrée, puisqu’elle revient à caractériser une notion informelle, celle de procédure effective, par une construction formelle, celle des machines de Turing. On détaillera les points de vue défendus en annexe.

Néanmoins, ce formalisme des machines de Turing permet d’appréhender la notion de *problème* qui était, jusqu’ici éludée et considérer comme implicite par le biais d’une machine.

3.2 L’idée d’une formalisation de la notion de problème et de décidabilité

Un problème est spécifié par l’ensemble de ses *instances* et une *requête*.

Les requêtes que l’on considère ici n’admettent comme réponse que *oui* ou *non* : une requête détermine l’ensemble des instances pour lesquelles la réponse est positive. De façon abstraite, ce sont des

⁴Kurt Gödel, *Remarks before the Princeton bicentennial conference on problems in mathematics*, 1946.

⁵Si la fonction numérique F associe un entier à tout k -uplet d’entiers $k \in \mathbb{N}$, on dit que F est une fonction numérique totale. Si F n’est pas définie pour chaque k -uplet, on dira que F est partielle. On considérera en général F partielle.

⁶On précise que ce codage est similaire à celui de Gödel, à savoir ici qu’il s’agit d’une injection de $\Sigma^* \rightarrow \mathbb{N}$ et respecte les mêmes conditions d’effectivité que l’on retrouvera en annexe.

problèmes de décision, de la forme $x \in A?$, pour une partie de A de l'ensemble des instances.

Ainsi, un problème de décision est dit *décidable* s'il existe un algorithme (i.e. une machine de Turing) qui se termine en un nombre fini d'étapes et qui répond par oui ou non à la requête du problème. Si un tel algorithme n'existe pas, le problème est dit indécidable (i.e non récursif).

On distingue de manière claire un pont se construire entre théorie logique et la notion de décidabilité. On peut alors se demander s'il existe une procédure effective capable de résoudre le problème d'instance \mathcal{T} et Φ , respectivement une théorie axiomatique et une formule de \mathcal{T} , et de requête " $\mathcal{T} \models \Phi$ ".

La démonstration d'une formule étant une procédure finie, la décidabilité logique⁷ d'une théorie implique nécessairement sa décidabilité algorithmique. Dès lors, si \mathcal{T} est syntaxiquement complète alors elle est décidable. On peut en effet imaginer opérer une recherche exhaustive à l'aide d'un algorithme qui répondrait *oui* à requête "Cet énoncé est-il prouvable?" et non dans le cas contraire. Par contre, une théorie syntaxiquement incomplète n'assure pas la décidabilité algorithmique (ni logique par contraposée), on ne peut, dès lors, ni être sûr d'obtenir en un temps fini la réponse à la requête précédente ni, par voie de conséquence, conclure.

3.3 Le problème de l'arrêt comme prélude à la solution de l'*Entscheidungsproblem*

À la lumière de ces résultats de décidabilité, Turing interroge d'un point de vue métathéorique les capacités de ses machines. On voudrait savoir, par exemple, s'il existe un moyen effectif de résoudre le problème de l'arrêt, c'est-à-dire s'il existe un moyen effectif de déterminer pour une machine de Turing M et un mot m si M entrera en état final après un nombre fini d'étapes de calcul pour m . Par la thèse de Turing, la question revient à déterminer un algorithme capable de savoir si M s'arrête.

⁷La décidabilité logique d'une théorie \mathcal{T} se déduit de la complétude syntaxique de \mathcal{T} .

On donne une démonstration de l'indécidabilité du problème de l'arrêt en annexe.

Pour résoudre l'*Entscheidungsproblem*, il suffit alors de montrer que sa solution permettrait aussi de résoudre le problème de l'arrêt. Ce qui suppose la possibilité d'une mise en relation des machines de Turing et des systèmes formels de la logique, puisque le problème de la décision porte sur ces problèmes. Plus précisément, cela suppose que l'on puisse décrire le fonctionnement des machines par une formule logique ce que l'on montre aisément par la construction d'un langage construit sur la logique des prédicats du 1er ordre et doté des prédicats adéquats. Par exemple, on peut proposer le prédicat binaire R_{σ_i} et que l'on interprète $R_{\sigma_i}(x, y)$ par la fonction propositionnelle suivante : "Dans la configuration x de la machine M , le symbole inscrit sur le case y du ruban est σ_i ". Ainsi, en introduisant un nombre suffisant de prédicats semblables dont chacun permet de décrire une caractéristique de M , on peut représenter dans son entièreté les machines de Turing dans la logique des prédicats du 1er ordre et par conséquent la proposition suivante : "Il existe une étape du calcul où la machine M entre dans l'état final", par une formule existentielle

$$\exists x A_M(x)$$

où $A_M(x)$ est une formule qui dépend de la machine M , et dans laquelle la variable x est libre. Or, en construisant une formule logique adéquate, Turing montre que si l'*Entscheidungsproblem* admet une solution, il existe un algorithme permettant de déterminer, pour toute machine, si la formule $\exists x A_M(x)$ est un théorème, et donc de résoudre le problème de l'arrêt. Or, puisque le problème de l'arrêt est insoluble l'*Entscheidungsproblem* n'admet pas de solution.

4 Conclusion : Vérité et démonstration

Ainsi, à la question de la cohérence logique vient s'ajouter celle d'une cohérence algorithmique. Plus généralement, les propriétés que l'on cherche à satisfaire en concevant un système logique concernant au-

tant les machines, les algorithmes ou les calculs que la vérité, le raisonnement ou langage.

Bien que les pages qui précèdent montre éminemment en quel sens la logique prend pour objet les machines dans un but de recherche de la vérité ou dans la production de processus de démonstration, elles sont loin de dresser un tableau exhaustif des relations qui existent.

Un examen plus complet s'attarderait sur la méthode de résolution de Robinson, utilisée pour la démonstration automatique dans des langages tels que Prolog, et les travaux de Gentzen concernant le calcul des séquents. Il expliciterait les motivations logiques et informatiques de la logique linéaire de Girard ou la logique temporelle très utilisé dans la vérification formelle de programme ou dans les systèmes de modèles checking. Il considérerait aussi le fondement logique de l'interrogation des bases de données et ses liens avec la logique des prédicats. Il mettrait en vedette la correspondance de Curry-Howard et ses liens avec la théorie de la démonstration. Il dévoilerait l'importance fondamentale des structures récursives dans les systèmes étudiés ainsi que certains formalismes permettant leur étude comme le λ -calcul.

Au-delà des théories de la calculabilité et de la démonstration, il montrerait qu'il existe une étude *sémantique* des programmes et des théories et montrerait, plus en détails, les conséquences, par exemple, de l'indécidabilité du problème de l'arrêt comme le théorème de Rice.

Cette liste est-elle même non exhaustive mais elle donne quelques indications supplémentaires sur la manière dont les analyses précédentes pourraient être prolongées. Ensemble, elles permettent de comprendre plus clairement en quel sens la logique est liée à l'informatique et pourquoi elle ne traite pas seulement de la vérité, du langage, raisonnement ou des paradoxes mais aussi des machines, des programmes, et du calcul en un sens plus discret.

Annexe

Éléments sur les théorèmes d'incomplétude

Le premier théorème d'incomplétude de Gödel repose sur l'observation que toutes les structures syntaxiques de l'arithmétique de Peano — les termes, les formules et les démonstrations — sont des structures de données finies qui peuvent être représentées par des entiers naturels à l'aide d'un codage approprié. L'informatique nous a habitué au fait que toutes les données que nous manipulons sur nos ordinateurs (les textes, les programmes, les bases de données, et même les images et les sons) sont représentées dans les entrailles de la machine par des suites finies de 0 et de 1. L'intérêt d'une telle numérisation des structures syntaxiques de l'arithmétique (i.e. des termes, des formules et des démonstrations) réside dans le fait qu'elle nous permet d'exprimer les propriétés formelles de ces structures (une fois qu'on les a représentées par des entiers naturels) dans le langage même de l'arithmétique, et de raisonner sur ces structures dans l'arithmétique formelle. La démonstration du théorème de Gödel procède donc d'une véritable mise en abyme, dans laquelle on construit des formules métamathématiques qui parlent des formules mathématique et des démonstrations métamathématiques qui établissent les propriétés des démonstrations mathématiques.

Pour se faire, Gödel établit un codage numérique et systématique à toute expression formelle appelé *codage de Gödel*. Ce codage est effectif au sens suivant :

1. Il existe une procédure effective pour calculer en un nombre fini d'étapes le nombre de Gödel pour toute expression formelle ou suite d'expressions formelles.
2. Il existe une procédure effective permettant de déterminer en un nombre fini d'étapes si un nombre entier quelconque donné est associé au nombre de Gödel d'une expression formelle (ou suite d'expressions formelles). En ce sens, il s'agit également d'une fonction récursive, et c'est la raison pour laquelle on doit se restreindre aux théo-

ries récursives, c'est-à-dire aux théories \mathcal{T} telles que l'ensemble des codes des axiomes de \mathcal{T} est récursif.

Ce codage se représente formellement par une fonction injective \mathcal{G} telle que :

$$\mathcal{G} : \begin{cases} \mathcal{T} & \rightarrow \mathbb{N} \\ e & \mapsto \ulcorner e \urcorner \end{cases}$$

On ne parle plus alors de l'ensemble de formules de la théorie \mathcal{T} mais de l'ensemble des entiers tels que le nombre de Gödel associé est un élément de \mathcal{T} . On notera $\text{Form}(x)$ le prédicat « x est le nombre de Gödel d'une formule de \mathcal{T} ». On considère ainsi les prédicats suivants :

- * $\text{Form}(x) \equiv$ « x est le nombre de Gödel d'une formule de \mathcal{T} »
- * $\text{Neg}(x, y) \equiv$ « x est le code de la négation d'une formule représentée par le code y »
- * $\text{Dem}(x, y) \equiv$ « x est le code d'une démonstration dans \mathcal{T} de la formule représentée par le code y »
- * $\text{Th}(x) \equiv$ « x est le code d'une formule démontrable de \mathcal{T} »

De la sorte, on obtient toutes les formules qui permettent d'exprimer les opérations désirées. La théorie \mathcal{T} choisie par Gödel permet de formaliser l'arithmétique élémentaire. Si $k_1, \dots, k_n \in \mathbb{N}$, on représentera par $\mathbf{k}_1, \dots, \mathbf{k}_n$ la représentation formelle de ces entiers dans \mathcal{T} . On cherche désormais à exprimer par des formules de la théorie des relations métamathématiques numériques comme $\text{Form}(x)$ qui portent sur la théorie. À cette fin, on donne la définition suivante : une *relation* numérique n -aire R est représentable dans la théorie \mathcal{T} par une formule $A[x_1, \dots, x_n]$ (où les x_i sont des variables libres de A) lorsque l'on a :

1. Si $R(k_1, \dots, k_n)$ alors $\vdash A[\mathbf{k}_1, \dots, \mathbf{k}_n]$

C'est à dire que si les entiers k_1, \dots, k_n vérifient la relation R , alors la formule $A[\mathbf{k}_1, \dots, \mathbf{k}_n]$ est démontrable dans \mathcal{T} .

2. Si $\text{non-}R(k_1, \dots, k_n)$, alors $\vdash \neg A[\mathbf{k}_1, \dots, \mathbf{k}_n]$

C'est à dire que si les entiers k_1, \dots, k_n ne vérifient pas la relation R , alors la formule $\neg A[\mathbf{k}_1, \dots, \mathbf{k}_n]$ est démontrable dans \mathcal{T} .

De manière similaire, une fonction numérique f (on considère ici f unaire mais on pourrait généraliser à n arguments) est représentable dans la théorie \mathcal{T} lorsqu'il existe une formule à deux variables libres, $A[x, y]$ qui vérifie la condition suivante : $\forall n, m \in \mathbb{N}$, on a $f(m) = n$ si et seulement si la formule $\forall y (A[\mathbf{m}, y] \leftrightarrow y = \mathbf{n})$ est démontrable⁸ dans la théorie \mathcal{T} .

Pour prouver qu'une fonction ou une relation est représentable dans \mathcal{T} , Gödel utilise une méthode indirecte. Il définit d'abord la classe des fonctions numériques primitives récursives et la notion de relations primitives récursives et prouve qu'elles sont toutes deux représentables dans \mathcal{T} . On donne leur définition respective :

* Soit $p \in \mathbb{N}$ et soient p entiers $k_1, \dots, k_p \in \mathbb{N}$. $\forall i, 1 \leq i \leq p$. On notera le p -uplet $(k_1, \dots, k_p) = \vec{k}$

Les fonctions primitives récursives sont construites par induction par assemblage des trois fonctions suivantes :

- La fonction identiquement *nulle* \mathcal{Z} telle que
$$\mathcal{Z} : \begin{cases} \mathbb{N}^p & \rightarrow \mathbb{N} \\ \vec{k} & \mapsto 0 \end{cases}.$$
- La fonction *successeur* Succ telle que $\forall n \in \mathbb{N}, \text{Succ}(n) = n + 1$.
- Les fonctions de *projection* π_i sur le i -ème argument telle que $\pi_i : \begin{cases} \mathbb{N}^p & \rightarrow \mathbb{N} \\ \vec{k} & \mapsto k_i \end{cases}.$

⁸On trouve de manière équivalente dans certains ouvrages la formulation suivante : $f(m) = n$ si et seulement si les formules $A[\mathbf{m}, \mathbf{n}]$ et $\exists! A[\mathbf{m}, y]$ sont l'une et l'autre démontrables dans \mathcal{T} . Dans les deux cas, on veut exprimer que pour un entier donné, il existe un unique entier n tel que $A[\mathbf{m}, \mathbf{n}]$.

Et par itération des constructions suivantes, la composition et la récurrence :

- Soit $p \in \mathbb{N}$ et soient $p + 2$ entiers k, r, k_1, \dots, k_p . Soient p fonctions f_1, \dots, f_p où $f_i : \mathbb{N}^k \rightarrow \mathbb{N}^{k_i}$ et soit g , une fonction de $\mathbb{N}^{\sum_{i=1}^p k_i} \rightarrow \mathbb{N}^r$. La composée $g(f_1, \dots, f_p)$ est la fonction de $\mathbb{N}^k \rightarrow \mathbb{N}^r$ qui à chaque k -uplet $n = (n_1, \dots, n_k)$ associe le r -uplet $g(f_1(n), \dots, f_p(n))$.
- Soient $k, r \in \mathbb{N}$. Soit $f : \mathbb{N}^k \rightarrow \mathbb{N}^r$ et soit $g : \mathbb{N}^{k+r+1} \rightarrow \mathbb{N}^r$. La fonction $h = \text{Rec}(f, g)$ est la fonction de $\mathbb{N}^{k+1} \rightarrow \mathbb{N}^r$ définie récursivement telle $\forall n \in \mathbb{N}$ et $\forall m \in \mathbb{N}^k$:

$$\begin{aligned} \cdot & h(0, m) = f(m), \\ \cdot & h(n+1, m) = g(n, h(n, m), m) \end{aligned}$$

* Une relation $R(x)$ est primitive récursive si sa fonction caractéristique χ_R définie telle que :

$$\chi_R(x) = \begin{cases} 1 & \text{si } R(x) \\ 0 & \text{si } \neg R(x) \end{cases}$$

est elle même primitive récursive.

Ainsi, pour démontrer qu'une relation numérique qui exprime une propriété métamathématique (par exemple $\text{Form}(x)$) est représentable dans \mathcal{T} , il suffit de montrer qu'elle est primitive récursive.

Idéalisme et constructivisme autour de la notion d'effectivité

L'argumentation du point de vue constructif

La notion d'effectivité est plutôt une notion première (au même titre que la notion d'entier naturel). De manière intuitive, cette notion se ramène à celle d'une construction, ainsi on considère un résultat comme effectif s'il peut-être construit à partir d'un ensemble fini de données. Mais la notion de construction est aussi sujette à discussions. Certaines opérations

sont identifiables clairement comme construction, par exemple la dérivation logique mais pour d'autres la définition ne permet pas toujours de trancher. Et il y a le domaine inconnu des mathématiques à venir, sur lequel tout pari est hasardeux. Ainsi, l'effectivité ne peut pas être définie *stricto sensu*. D'autre part, la définition des suites mécaniquement calculables fait appel à la notion d'effectivité de manière incontournable, au moins lorsque l'on se place d'un point de vue constructif. En effet, dans la définition d'une fonction mécaniquement calculable, il y a une utilisation inévitable de la notion d'effectivité intuitive (non définie). Cet utilisation de la notion d'effectivité ne peut pas être contournée. Lorsque l'on dit que pour toute entrée $m \in \Sigma^*$, la Machine de Turing M donne un résultat, on dit que la machine aboutira "effectivement" à l'état final après un certain nombre d'étapes élémentaires de calcul. Autrement dit, sans le concept de base (non défini) d'effectivité, on ne peut comprendre de manière vraiment rationnelle la notion de fonction mécaniquement calculable.

L'argumentation du point de vue idéaliste

Le point de vue idéaliste fait intervenir certaines notion que l'on retrouve dans la philosophie platonicienne. L'ensemble des entiers naturels est un infini en acte au moins de manière idéale, c'est-à-dire au moins dans un monde des idées qui nous dépasse. Chaque fois que l'on démontre un théorème au sujet des entiers naturels, on met en puissance une vérité qui existait déjà de toute éternité, dans le monde des idées pures. Ainsi, même s'il n'y a pas de procédé mécanique qui permette de savoir si une Machine de Turing M donnée calcule bien une fonction de Σ^* dans \mathbb{N} , il n'en reste pas moins vrai que la machine M calcule une fonction de Σ^* dans \mathbb{N} ou bien échoue à calculer au moins une fois la valeur de la fonction. En conséquence la notion de fonction effectivement calculable (au sens intuitif) est bien capturée par la notion (absolue) de fonction mécaniquement calculable au sens de Turing.

Démonstration de l'indécidabilité du problème de l'arrêt

On suppose alors que le problème de l'arrêt soit décidé par une machine de Turing M^* . On construit une machine de Turing M'' qui fonctionne telle que :

- * M' prend en entrée un mot $\langle M \rangle$ codant une machine de Turing M .
- * M' appelle la machine de Turing M' sur la paire $\langle \langle M \rangle, \langle M \rangle \rangle$, c'est-à-dire sur l'ensemble constitué du codage de la machine M , et du mot m correspondant aussi à ce codage).
- * Si la machine de Turing M^* :
 - Accepte m , M' refuse.
 - Refuse m , M' accepte.

Par construction M' termine sur toute entrée.

On raisonne par l'absurde. On applique la machine de Turing M' sur le mot $\langle M' \rangle$, c'est-à-dire le mot codant la machine de Turing M' .

- * Si M' accepte $\langle M' \rangle$, alors cela signe, par définition du problème de l'arrêt et de M^* , que M^* accepte $\langle \langle M' \rangle, \langle M' \rangle \rangle$. Mais si M^* accepte ce mot, M' est construit pour accepter son entrée $\langle M' \rangle$. Il résulte une contraction.

Cette preuve de la non-existence d'un algorithme qui résout le problème de l'arrêt est très important dans les mathématiques. En effet, si un tel algorithme existait, on pourrait, par exemple résoudre facilement la conjecture de Goldbach ; il suffirait de vérifier pour tout nombre pair N s'il est somme de deux nombres premiers p et q .

Goldbach() :

```

N ← 4
while ∃p, q ∈ P | p + q = N
  N ← N + 2
return N

```

Si l'algorithme pouvait savoir si **Goldbach()** se termine, on pourrait savoir si la conjecture est vraie

ou fausse. Ici, si elle se termine, c'est qu'il existe un nombre qui met en défaut la conjecture et si `Goldbach()` ne se termine jamais, la conjecture serait démontrée et deviendrait un théorème.

Références

- [1] Jean-Yves Antoine, Jeanne Villaneau, *Logique pour l'informatique*. (2017)
- [2] Olivier Carton, *Langages formels, calculabilité et complexité*, Vuibert. (2008)
- [3] Alonzo Church. *A note on the Entscheidungsproblem - Journal of Symbolic Logic*. (1936)
- [4] Gilles Dowek, *Les démonstrations et les algorithmes*, cours de l'Ecole Polytechnique. (2008)
- [5] Pascal Engel, *La Norme du vrai*, 3ème édition, Gallimard. (1989)
- [6] Gerhard Gentzen, *Recherches sur la deduction logique*, PUF. (1955)
- [7] Jean-Yves Girard, *Le point aveugle : Vers la Perfection*, Hermes. (2007)
- [8] Jean-Yves Girard, *Le point aveugle : Vers l'Imperfection*, Hermes. (2007)
- [9] Kurt Gödel, *Sur les propositions formellement indécidables des Principia Mathematica et des systèmes apparentés*. (1931)
- [10] Dexter Kozen, *Automata and computability*, Springer Verlag. (1997)
- [11] Roger Lallemand, *Logique, réduction, résolution, études et recherches en informatique* Masson. (1990)
- [12] Henri Lombardi, *Les nombres réels calculables selon Alan Turing*. (2011)
- [13] Alexandre Miquel, *Les théorèmes d'incomplétude de Gödel*, ENS Lyon (2003)
- [14] E. Nagel, J. R. Newman, K. Gödel et J.-Y. Girard. *Le théorème de Gödel*, Éditions du Seuil (coll. Points), ISBN : 2-02-032778-3. 1989
- [15] Alan Turing, *Sur les nombres calculables suivie d'une application au problème de la décision*. (1936)
- [16] Pierre Wagner, *La machine en logique*, PUF. (1998)