	Algorithmique	**NSI T^{ale}**
	Décidabilité et calculabilité	Cours

*

1. Introduction

Il y a maintenant des dizaines de siècles que les mathématiciens décrivent et utilisent des ***méthodes de calcul*** permettant de résoudre leurs problèmes. Mais jusqu'à une date relativement récente (**1934**) ils ne savaient pas de façon précise ce qu'est une ***méthode de calcul***.
Aujourd'hui les mathématiciens possèdent une définition précise de ce qu'on doit appeler ***méthode de calcul***, ils savent exactement quels sont les problèmes qu'elles peuvent traiter, et mieux encore, ils savent que certains problèmes ne sont pas "***traitables***"
En effet, et c'est là l'un des aspects les plus extraordinaires de ces travaux, ils permettent d'établir des résultats négatifs, c'est-à-dire de la forme :

- pour tel problème non seulement aucune méthode n'est actuellement connue, mais aucune ne le sera jamais.

Ces problèmes pour lesquels on démontre qu'il n'existe aucune méthode de calcul adaptée s'appellent des **problèmes indécidables**.

Les premiers résultats d'indécidabilité des années 1930 semblaient artificiels, très vite on s'est aperçu que des problèmes assez simples entraient dans la classe des **problèmes indécidables**.
En particulier en informatique, de nombreuses questions naturelles qui se posent aux programmeurs se révèlent après étude correspondre à des ***problèmes indécidables***.
Lorsqu'on démontre qu'un problème P est ***indécidable***, on en déduit :

- qu'on doit renoncer à le résoudre tel quel, et donc que,
- qu'il faut trouver une version simplifiée de P, puis, soit réussir à la résoudre, soit à nouveau établir qu'elle est encore ***indécidable*** et donc la simplifier encore, etc.

2. Petit historique

La notion informelle d'**algorithme** est extrêmement ancienne et les procédés que nous avons appris à l'école primaire pour additionner deux nombres écrits en base dix ou pour les multiplier sont des algorithmes.
Le mot même d'algorithme provient du nom d'un **mathématicien arabe du IXe siècle**, **Al Khwarizmi** .
De même que Monsieur Jourdain faisait de la prose sans le savoir dans *Le Bourgeois Gentilhomme* de Molière, les mathématiciens recherchaient et élaboraient des algorithmes bien avant que le concept soit clairement identifié et défini.
En fait, au début du siècle les mathématiciens ne soupçonnaient pas qu'on pourrait arriver à préciser vraiment cette notion, ni encore moins, qu'on pourrait démontrer pour certains problèmes qu'aucun algorithme n'existe.
Le travail d'identification et de formulation de la notion d'algorithme fut effectué en plusieurs étapes entre 1931 et 1936 par les mathématiciens Alonzo Church, Stephen Cole Kleene, Alan Turing et Kurt Gödel.

Ils introduisirent plusieurs classes différentes de fonctions dont ils montrèrent ensuite qu'elles coïncidaient, et qu'ils reconnurent alors comme la classe des fonctions calculables.
Une fonction est calculable s'il existe une façon finie de la décrire qui permette effectivement d'en calculer toutes les valeurs. La définition précise de la notion de fonction calculable fixe en même temps celle d'algorithme, et on peut donc dire qu'en 1936 la formulation exacte de la notion d'algorithme était acquise.
De toutes les définitions finalement équivalentes de la notion d'algorithme formulées dans les années 1930 et depuis, celle donnée par **Turing en 1936** est la plus pratique pour les théoriciens, et on l'utilise encore aujourd'hui.
C'est sur elle que nous allons nous appuyer en TP pour définir la notion d'algorithme et montrer que certains problèmes précis n'ont pas d'algorithmes.

Chaque année des dizaines de résultats de décidabilité ou d'indécidabilité sont établis dans de nombreux domaines des mathématiques et on peut être certain, tant les questions ouvertes restent nombreuses, que ce mouvement n'est pas prêt de cesser.
Ici nous nous intéresserons uniquement au problème de la possibilité d'un algorithme ou de son impossibilité, et non pas au problème de l'**efficacité** qui lui aussi est à l'origine d'un domaine de travail et de réflexion des plus vivants aujourd'hui.

3. Définitions

Programme en tant que donnée
Un compilateur, un débugueur, un interpréteur sont des programmes qui prennent en paramètres d'autres programmes. C'est aussi le cas d'un logiciel de téléchargement, ou d'un système d'exploitation qui va faire tourner d'autres programmes. Ainsi, tout programme peut être perçu comme une donnée pour un autre programme.

Calculabilité et décidabilité, deux notions proches
Étudier la calculabilité d'une fonction ou la décidabilité d'un problème, c'est chercher si cette fonction peut être calculée, ou si ce problème peut être résolu, à l'aide d'un algorithme.

Problème de décision
Un problème de décision est un problème qui prend une entrée (une donnée) et renvoie en sortie **oui** ou **non**.
Un **problème de décision** est dit **décidable** **s'il existe un algorithme** qui, **pour chaque entrée**, réponde par oui ou par non à la question posée par le problème. **S'il n'existe pas un tel algorithme, le problème est dit indécidable.**

4. Quelques problèmes décidables élémentaires

Problème A.

```
Soient m et n deux entiers donnés > 1.
m est-il un multiple de n ?
```

On sait qu'il est vrai que :

```
12 est un multiple de 2,
```

et qu'il est faux que :

```
16 est un multiple de 5.
```

On sait même bien plus que cela, on sait comment s'y prendre pour déterminer pour tout n et tout m quand est vrai que

```
"m est un multiple de n"
```

et quand est faux que :

```
"m est un multiple de n".
```

Il suffit en effet de :

```
- faire la division de m par n,
- regarder le reste obtenu, r,
- si r = 0 alors il est VRAI que "m est un multiple de n"
- sinon il est FAUX que "n est un multiple de m"
```

Ce procédé général et systématique de calcul, constitue un algorithme (informel).
C'est un procédé absolument sûr, qui fonctionne en un temps fini pour tous les jeux de données possibles et conduit toujours à la bonne réponse :

il montre que le problème A est décidable.

Remarquons bien que pour être précis quand on parle de ***problème décidable*** ou ***indécidable*** il faut indiquer ce que sont les **paramètres** du problème.
Dans notre exemple n et m sont deux entiers > 1 . En fait on ne cherche pas à résoudre un problème unique, on cherche à résoudre une classe infinie de problèmes, ici tous les problèmes :

```
2 est-il un multiple de 2 ?
3 est-il un multiple de 2 ?
... ... ... ...
10 est-il un multiple de 3 ?
... ... ... ...
1050501 est-il un multiple de 101 ?
... ... ... ...
```

Autres exemples de problèmes décidables

Problème des nombres premiers.

```
Soit n un entier donné > 1
n est-il un nombre premier ?
```

Problème des racines entières d'un polynôme.

```
Soit un polynôme P à coefficients entiers, à une variable,
(par exemple X3-X2+X-1)
existe-t-il une valeur entière de X qui annule P ?
```

Problème des décimales de PI

```
Soit n un entier > 0.
Le nième chiffre de Pi écrit en base 10 est-il 7 ?
```

5. Un problème indécidable de géométrie élémentaire.

Problème B

```
Soient F1 F2 ... Fn une liste de formes polygonales.
Peut-on paver le plan, sans recouvrement ni espace vide avec des
exemplaires de F1 F2 ... Fn ?
```

Il a été démontré que ce problème est **indécidable** :

Il n'existe aucun algorithme permettant par un calcul fini à partir des données (la liste des formes géométriques) d'établir si OUI ou NON il est possible de paver le plan avec des exemplaires des formes géométriques considérées.

Dire que le problème est **indécidable** est plus fort que dire que l'on ne sait pas résoudre le problème, ce qui marquerait simplement notre ignorance.
L'**indécidabilité c'est l'impossibilité absolue et définitivement démontrée de résoudre par un procédé général de calcul un problème donné.**

Il est important de noter que le fait qu'un problème soit indécidable ne signifie pas que chaque cas particulier du problème est impossible à résoudre.
On sait très bien par exemple que le plan peut être recouvert avec des hexagones réguliers.
On sait très bien aussi que le plan ne peut pas être recouvert avec des pentagones réguliers (ce n'est pas difficile à établir).
Dire que le problème est indécidable signifie qu'il est impossible de trouver une méthode traitant tous les cas systématiquement.

Le problème peut sembler trop général, et finalement on se dit que s'il est indécidable c'est qu'il y a trop de listes de formes géométriques. C'est ce qui conduit à considérer des sous-problèmes du problème B.
Ces sous-problèmes peuvent être indécidables eux-aussi ou au contraire être décidables.

Problème B'

```
Soient F1 F2 ... Fn une liste de formes géométriques composées chacune de carrés juxtaposés.
Peut-on paver le plan, sans recouvrement ni espace vide avec des exemplaires de F1 F2 ... Fn ?
```

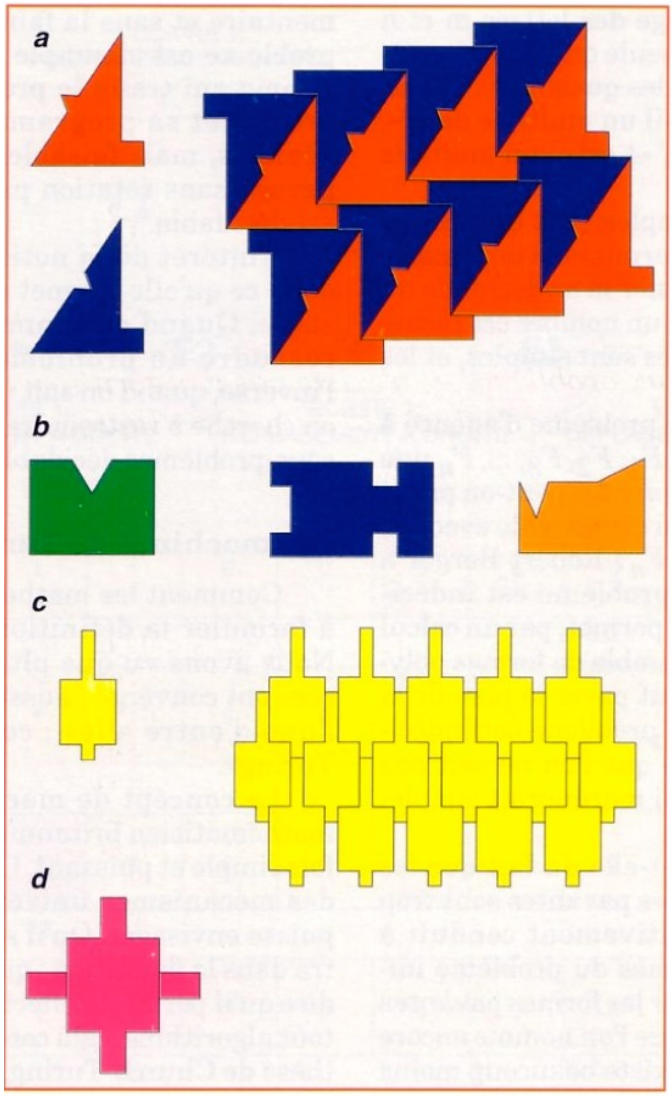
Il y a beaucoup moins de "listes de formes géométriques composées chacune de carrés juxtaposés" que de "listes de formes géométriques", le problème B' est donc plus simple que le problème B.
Il se pourrait donc que B', puisse être résolu par un algorithme. Pourtant ce n'est pas le cas (la démonstration que le problème B était indécidable établit en fait que B' est indécidable).

Lorsqu'on connaît un algorithme pour résoudre un problème :

- il est alors naturel de chercher à le généraliser (i.e. être plus tolérant dans les jeux de données acceptés).

A l'inverse quand on sait qu'un problème est indécidable,

- on cherche alors à en trouver des sous-problèmes décidables.



CERTAINS PROBLÈMES DE PAVAGE DU PLAN par des polygones sont indécidables. Quand on se donne les deux polygones du haut de la figure, on peut paver le plan. Dans ce cas précis, on trouve facilement un pavage. La méthode qui permet de résoudre le problème dans ce cas est-elle généralisable? R. Berger a démontré en 1966 que non : le problème est indécidable. Déjà, pour les trois formes du milieu de la figure, il faut un peu d'inventivité pour démontrer qu'un pavage du plan sans recouvrement ni espace vide est impossible. L'indécidabilité du problème du pavage signifie que, pour traiter de nouvelles situations, le mathématicien sera inévitablement amené à inventer de nouvelles méthodes de raisonnement : jamais aucun procédé général mécanique ne réussira à englober tous les cas possibles. En revanche, pour un polyomino, composé de carrés adjacents, on sait que la question du pavage est décidable : il existe un algorithme qui, lorsqu'on lui donne un polyomino, indique correctement s'il est possible de paver le plan en l'utilisant sans le faire tourner. La frontière entre le décidable et l'indécidable passe entre cette forme simplifiée du problème du pavage et la version générale. L'indécidabilité du problème des pavages du plan est liée à l'existence de pavés qui ne peuvent recouvrir le plan que non périodiquement. Un pavage non périodique d'un nouveau type a été découvert en 1994 par Charles Radin, de l'Université du Texas : contrairement à tous les pavages connus jusqu'à présent, le pavage de Radin oblige les pavés à effectuer des rotations selon une infinité d'angles différents.

6. Calculabilité - Problème de l'arrêt en Python.

On dit qu'une fonction se termine si elle renvoie une valeur ou si elle lève une exception (par exemple `ZeroDivisionError` si une division par zéro est tentée).
Une fonction peut se terminer ou continuer à calculer à l'infini.
Ainsi, la fonction `ma_fonction` (voir ci-dessous), pour un entier `n`, se termine pour `n` inférieur ou égal à 10 (elle renvoie `None`) et ne se termine pas pour `n` strictement plus grand que 10. En outre, cette fonction se termine pour une chaîne de caractères `n` (en levant l'exception `TypeError` à cause de l'opération +).

```
def ma_fonction (n):
  while n != 10 :
    n = n + 1
```

On pourrait penser qu'il serait utile d'avoir en Python une fonction arrêt, qui se termine sur toutes les entrées possibles, et telle que `arrêt(f, x)` renvoie `True` si le calcul de `f(x)` se termine et `False` sinon. Une telle fonction, si elle était ajoutée au langage, permettrait d'éviter les boucles infinies en faisant un simple test.

Si on voulait programmer une telle fonction, il serait naturel que la fonction `arrêt` accède au code de la fonction `f` (ce qui est possible grâce au module dis), et analyse ce code, par quelque moyen compliqué, pour en déduire si oui ou non le calcul de `f(x)` va un jour terminer.

La question "`Est-il possible de programmer une telle fonction`" est connue sous le nom de "**problème de l'arrêt**".
Nous allons montrer qu'une telle fonction ne peut pas exister par un raisonnement par l'absurde.
Supposons que quelqu'un ait réussi à programmer cette fonction arrêt. Nous pouvons alors l'utiliser pour programmer la fonction `strange` suivante :

```
def strange(f, x):
  if arret(f, x):
    while True:
      pass
```

Cette fonction teste si le calcul de `f(x)` se termine, si c'est le cas, elle rentre dans une boucle infinie dans laquelle elle ne fait rien. Si ce n'est pas le cas, elle ne fait rien et se termine.
Autrement dit, `strange(f, x)` se termine si et seulement si `f(x)` ne se termine pas.

La précédente fonction nous permet d'en définir une nouvelle, la fonction `paradox`.

```
def paradox(f):
  strange(f, f)
```

Par construction, le calcul de `paradox(f)` termine si et seulement si le calcul de `f(f)` ne se termine pas. Appelons (1) cette propriété.

Maintenant, demandons-nous si le calcul de `paradox(paradox)` se termine. Pour cela, dans la propriété (1), remplaçons `f` par `paradox`. La propriété devient : "`le calcul de paradox(paradox) se termine si et seulement si le calcul de paradox(paradox) ne se termine pas`".

Cette dernière propriété est évidemment contradictoire, fausse.

Nous pouvons donc conclure notre raisonnement par l'absurde et affirmer qu'il est impossible de programmer la fonction `arrêt`.

Remarque. Cela reste impossible même si on se limite à ce que les arguments de `arrêt` soient une fonction `f` des entiers dans les entiers et un entier `x`. Pourquoi ? Simplement parce que n'importe quelle valeur de n'importe quel type sera représenté en machine par une suite d'octet, soit à peu près un entier naturel.

Sources : http://www2.lifl.fr/~delahaye/HECI/Calcul.pdf
https://qkzk.xyz/docs/nsi/cours_terminale/prog/calculabilite/

Compléments :
https://www.youtube.com/watch?v=92WHN-pAFCs
https://interstices.info/alan-turing-du-calculable-a-lindecidable/
https://youtu.be/13O1qhX4Bqo