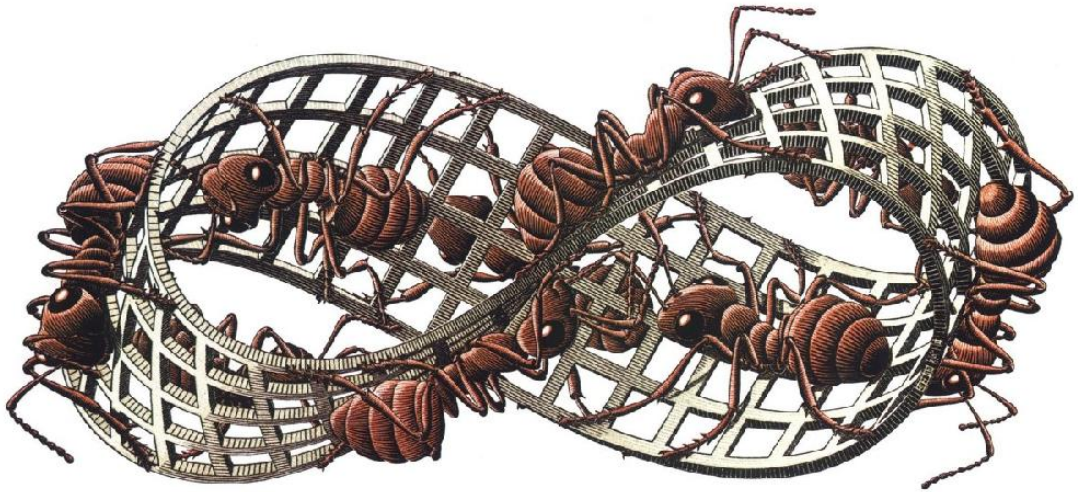
Mathématiques pour l'informatique

Jean-Pierre Messager (jp@xiasma.fr)

21 janvier 2021 – version 1.0b



Utilisation commerciale interdite sans autorisation

Télécharger ce cours à jour et y contribuer

Il est disponible sur un dépot git

- Accès public à ce cours : https://framagit.org/jpython/math
- Si vous créez un compte sur https://framagit.org/
- Transmettez-moi votre identifiant
- Je vous accorde le status *reporter* sur ce projet
  - Ouverture de tickets
- D'autres projets : https://framagit.org/jpython/meta
- Des mises à jours sont régulièrement disponibles
- Chaque changement important de version est accompagné d'un *tag*

Mathématiques pour l'informatique

Plan du cours

Mathématiques pour l'informatique

« Matematyczny ład świata to nasza modlitwa do piramidy chaosu. »

« L'ordre mathématique du monde est notre prière à la pyramide du chaos. »

Stanislas Lem (1921–2006)

pour Kate et Kelly

Présentation

Ce que nous allons évoquer ici est le résultat de plusieurs millénaires de pratiques et de réflexions mathématiques, scientifiques, esthétiques et philosophiques. En particulier à l'articulation du XIXe et du XXe siècle la *crise des fondements* a amené à formuler l'ensemble des mathématiques en termes d'ensembles et à mettre à plat la logique formelle.

Ces travaux sont intrinsinquement liés aux réflexions sur la *calculabilité* qui allaient amener à la naissance de l'informatique moderne : matériels comme logiciels.

D'autres champs des mathématiques sont évoqués du fait de leur importance en programmation : probabilités, algèbre linéaire.

Beaucoup de propositions sont affirmées *sans démonstration*. Elles existent bien entendu. De même des termes sont introduits sans définitions (*corps*, *anneau*, ...) ; elles se trouvent aisément sur Wikipedia.

Des pans entiers sont, hélas, négligés car ils nécessiteraient, chacun, un cours entier : géométries, calcul différentiel et intégral, analyse numérique, topologie, calcul formel, arithmétique, cryptographie, théorie des codes, théorie des jeux, statistiques, etc.

La présentation s'efforce d'éviter tout excès de formalisme, la théorie axiomatique *ZFC*, par exemple, n'est pas présentée. Le lecteur pourra s'y référer s'il cherche un peu plus de *rigueur*.

L'histoire ne s'est pas arrêtée là. La logique intuitionniste a donné naissance à toute une branche des mathématiques et à une façon de programmer qui permet de *prouver* qu'un programme n'a pas de *bugs* (système *CoQ* de l'INRIA par exemple).

Nous n'avons, ici, guère plus d'ambition que de nous prendre au jeu et d'éveiller notre curiosité. Chaque domaine n'est, au fond, qu'à peine *effleuré*. Il reste bien d'autres fruits à découvrir.

Note typographique

Ce document est rédigé en *Markdown*, format texte assez simple complété par le système de composition LaTeX, basé sur TeX de Donald Knuth, (particulièrement adapté à la composition de textes mathématiques mais pas seulement) ce qui permet à *Beamer* LaTeX (via *pandoc*) de construire le document final. Je ne saurais trop vous en recommander l'usage, en particulier LaTeX.

$$\underbrace{\ln\left(\frac{\pi}{4}\right)}_{\simeq -0{,}241564} < \overbrace{\sin\left(\frac{4}{\pi}\right)}^{\simeq 0.9560555}$$

Image de couverture : Maurits Cornelis Escher – *Möbius Strip II* (1963)

Qu'est-ce qu'une démonstration mathématique ?

Une démonstration est une successions d'énoncés qui, partant d'hypothèses et de vérités supposées (axiomes) établit une conclusion supposée valide : propriété d'un objet abstrait, existence d'un certain objet, inexistence ou impossibilité d'un autre.

Une démonstration vise d'abord à *convaincre* de la validité d'un théorème ; se convaincre déjà soi-même d'une intuition, ou l'invalider, puis de convaincre un *public* donné, invité à approuver ou infirmer la validité d'une *preuve* espérée.

La nature des arguments utilisé, leur niveau de détail, dépendent du public visé. Dans l'enseignement une *démonstration* est demandée aussi pour une autre raison : *convaincre* le lecteur que l'on a bien assimilé une technique de preuve ou un cadre conceptuel.

Démonstrations et automates

Alors, il ne sera plus besoin entre deux philosophes de discussions plus longues qu'entre deux mathématiciens, puisqu'il suffira qu'ils saisissent leur plume, qu'ils s'asseyent à leur table de calcul (en faisant appel, s'ils le souhaitent, à un ami) et qu'ils se disent l'un à l'autre : « Calculons ! »

— Gottfried Wilhelm Leibniz,
Nova methodus pro maximis et minimis in Acta Eruditorum, 1684

Si les *règles* de déduction utilisées pour passer d'une proposition à une autre dans une preuve sont suffisemment précises et mécaniques on peut envisager d'en *automatiser* la rédaction et la validation (« *pensée aveugle* » de Leibniz) voire de l'implémenter dans une *machine*.

En ce sens une preuve est une sorte de *programme* informatique. Cette intuition est formalisée rigoureusement par l'*isomorphisme de Curry-Howard*.

Exemple : $\sqrt{2}$ est irrationnel

Supposons qu'il existe p, q, entiers premiers entre eux (pas de facteurs communs) tels que :

$$\sqrt{2} = \frac{p}{q}$$

$$\Rightarrow p^2 = 2q^2$$

donc p^2 est pair, et donc p aussi : $p = 2k$, mais alors :

$$p^2 = (2k)^2 = 4k^2 = 2q^2$$

$$\Rightarrow q^2 = 2k^2$$

Donc q^2 est pair, donc q aussi, tout comme p.

Mais, par hypothèse, q et p n'ont pas de facteur commun...
L'hypothèse de départ est donc *fausse* : $\sqrt{2}$ est *irrationnel*.

Exemple : infinité des nombres premiers

Soient, pour $n \in \mathbb{N}^*$, $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, $p_4 = 7$, $\ldots$, p_n les n premiers nombres premiers consécutifs. Considérons le nombre :

$$N = p_1 \times p_2 \times p_3 \times p_4 \times \ldots \times p_n + 1$$

Pour tout n le reste de la division euclidienne de N par p_n est égal à 1.
N n'a donc aucun facteur premier dans la liste $(p_i)_{1 \leqslant i \leqslant n}$.
Il a donc un facteur premier *plus grand* (il n'y a pas de nombre premiers plus petits que p_n).

Pour toute liste des n premiers nombres premiers, on peut en trouver un *autre*, plus *grand*. Ils sont donc en quantité infinie, inépuisables.

Exemple : existence non-constructive

Question : existe-t-il a et b irrationnels tels que a^b soit rationnel ?

Considérons la proposition $(*)$: $\sqrt{2}^{\sqrt{2}}$ est irrationnel.
De deux choses *l'une* : ou bien elle est *vraie*, ou bien elle est *fausse*.

Si elle est *vraie*, alors la réponse à la question est *oui* : il suffit de prendre $a = b = \sqrt{2}$.

Si elle est *fausse*, alors on peut prendre $a = \sqrt{2}^{\sqrt{2}}$ et $b = \sqrt{2}$.
a est irrationnel (puisque $(*)$ est *fausse*) et b aussi, mais alors :

$$a^b = \left(\sqrt{2}^{\sqrt{2}}\right)^{\sqrt{2}} = \sqrt{2}^{\sqrt{2}\times\sqrt{2}} = \sqrt{2}^2 = 2$$

2 étant rationnel, la réponse à la question est aussi *oui*.

Donc la réponse est *oui*. Euh ?

Paradoxes

Il arrive que notre intuition soit prise en défaut...

Il existe des courbes du plan qui sont *continues* et n'ont de *tangentes* quasiment *null part i.e.* des *fonctions* de $\mathbb{R}$ dans $\mathbb{R}$ partout *continues* et *presque partout* (sauf en des points isolés) non-dérivables.

On peut découper une sphère en un nombre *fini* de morceaux et réarranger les morceaux pour fabriquer *deux* sphères de même taille (paradoxe de Banach-Tarski).

Il existe des parties de la droite réelle qui n'ont pas de *longueur* (même pas 0 ou $+\infty$) mais on ne peut en définir aucune.

Les ensembles *infinis* sont, d'une certaine façon, ausi *grands* que certaines de leurs *parties strictes*.

Paradoxe... ou contradiction ?

Si l'*ensemble de tous les ensembles* existe, on peut consider une de ses parties : l'ensemble U des ensembles qui *n'appartiennent pas à eux-mêmes*.

Est-ce que $U \in U$?

Si *oui*, alors $U \notin U$ (par définition de U) et donc *non*.
Si *non*, alors $U \in U$ (par définition de U) et donc *oui*.

Il *n*'existe donc *pas* d'ensemble de tous les ensembles !

La barbe !

Dans un village peut-on dire que le barbier ne rase que les gens qui ne se rasent pas eux-mêmes ?

Pourquoi la théorie des ensembles ?

- Les mathématiques comprennent la géométrie, l'arithmétique, l'algèbre, l'analyse, etc.
- À la fin du XIXe siècle la discipline apparaît très fragmentée même si les méthodes de démonstration sont communes
- Des paradoxes sont signalés ici et là...
- Les intuitions premières sont parfois fausses

- Recherche d'un ensemble de notions *simples* unificatrices pour encoder tout ça
- Georg Cantor, lors de recherches en analyse (théorie de Fourier), propose les ensembles
- D'autres propositions arriveront au XXe siècle : *fonctions* (lambda-calcul de Church), *symboles* (logique combinatoire de Schönfinkel), formalisme logique (Russell, Whitehead, Hilbert), *flèches* (théorie des catégories)

Ensembles

Définition informelle

Un ensemble est une collection d'objets distincts

On note $\in$ la relation *« est un élément de »*

Un ensemble est totalement défini par ses éléments :

Si $x \in E \Rightarrow x \in F$ **et** si $x \in F \Rightarrow x \in E$, alors $E = F$, en d'autres termes :

$$E = F \iff (x \in E \iff x \in F)$$

Inclusion entre ensembles

Un ensemble E est inclus dans un ensemble F ssi tous les éléments de E sont éléments de F.

$$E \subset F \iff (x \in E \Rightarrow x \in F)$$

Ensemble vide, union, intersection

L'ensemble vide

L'ensemble vide, noté $\{\}$ ou $\varnothing$ est l'unique ensemble qui n'a aucun élément : $x \in \varnothing$ est toujours *faux*.

Union

L'union de deux ensemble est l'ensemble des éléments qui appartiennent à l'un **ou** à l'autre.

$$x \in E \cup F \iff x \in E \text{ ou } x \in F$$

Intersection

L'intersection de deux ensembles est l'ensemble des éléments qui appartiennent à l'un **et** à l'autre.

$$x \in E \cap F \iff x \in E \text{ et } x \in F$$

Exercices

Prouver qu'il n'existe qu'un seul ensemble vide.

Prouver que si $E \subset F$ alors $E \cup F = F$ et $E \cap F = E$.

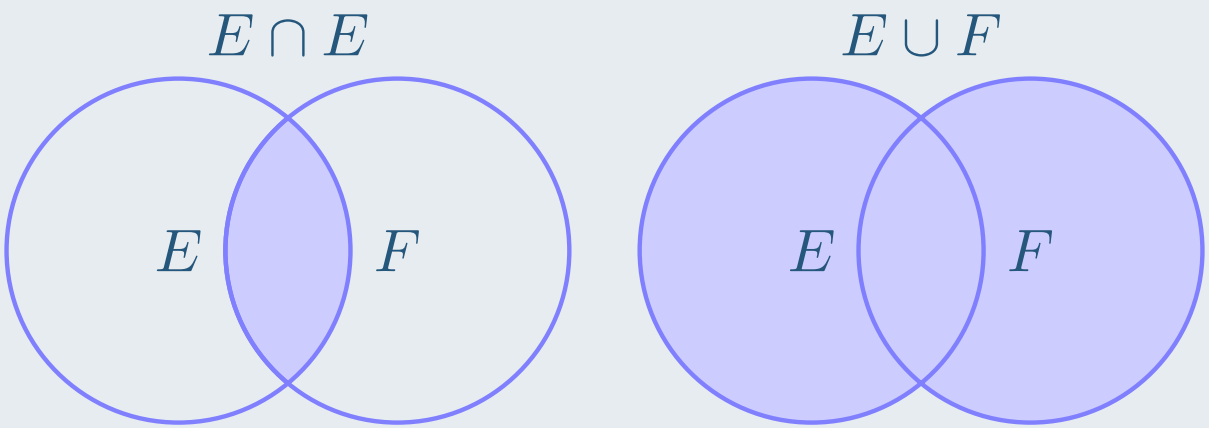
Prouver que si $E \cup F = F$ alors $E \subset F$ et que si $E \cap F = E$ alors $E \subset F$.

Prouver que : $E \cup (F \cap G) = (E \cup F) \cap (E \cup G)$
et que $E \cap (F \cup G) = (E \cap F) \cup (E \cap G)$

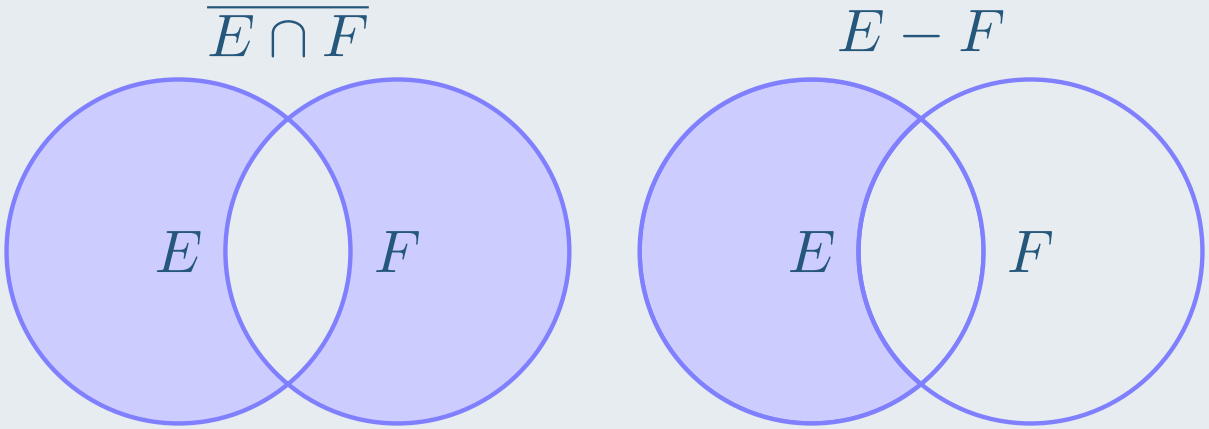
Soit U tel que $E \subset U$ et $F \subset U$. On note :
$\overline{E} = U \backslash E = \{x \in U : x \notin E\}$.
Prouver que $\overline{E \cup F} = \overline{E} \cap \overline{F}$ et $\overline{E \cap F} = \overline{E} \cup \overline{F}$

Diagrammes de Venn

Intersection et union



Différence symétrique et soustraction



Produit cartésien et relations

Couples d'éléments

On note (x, y) l'ensemble $\{\{x\}, \{x, y\}\}$ (couple de Kuratowski).
On peut montrer que $(x, y) = (x', y') \iff (x = x' \text{ et } y = y')$.

Produit cartésien

On note $E \times F = \{(x, y) : x \in E \text{ et } y \in F\}$ le *produit cartésien* de E et F.

Relations

Une relation $\mathcal{R}$ entre éléments se définit comme sous-ensemble d'un produit cartésien : pour $x \in E$ et $y \in F$, $\mathcal{R} \subset E \times F$.

$$x \mathcal{R} y \iff (x, y) \in \mathcal{R}$$

On dit parfois que le sous-ensemble de $E \times F$ est le *graphe* de la relation (en fait *c'est* $\mathcal{R}$). On parle de relation sur E si $E = F$.

Relations particulières

Relations d'équivalence ($E = F$)

- Réflexive : $x\mathcal{R}x$
- Symétrique : $x\mathcal{R}y \Rightarrow y\mathcal{R}x$
- Transitive : $x\mathcal{R}y$ et $y\mathcal{R}z \Rightarrow x\mathcal{R}z$

Relations d'ordre ($E = F$)

- Réflexive : $x\mathcal{R}x$
- Antisymétrique : $x\mathcal{R}y$ et $y\mathcal{R}x \Rightarrow x = y$
- Transitive : $x\mathcal{R}y$ et $y\mathcal{R}z \Rightarrow x\mathcal{R}z$

Fonctions

Une relation $\mathcal{R} \subset E \times F$ est une fonction ssi pour tout $x \in E$ il existe *un et un seul* couple $(x, y) \in \mathcal{R}$.
On note alors : $y = f(x) \iff x\mathcal{R}y$

Note historique

Historiquement, in France (suivant le groupe de mathématiciens Bourbaki) on admet souvent une définition légèrement différente :

- Une *fonction* est une relation telle que, pour x donné, il existe *au plus* un y tel que $x\mathcal{R}y$
- S'il en existe *un et un seul* on parle d'*application*
- Il peut donc alors exister $x \in E$ tel que $y = f(x)$ n'existe pas
- Si l'on *restreint* f au *domaine de définition* $\mathcal{D}_f \subset E$ de f on obtient une *application*

La pratique internationale usuelle parle de fonction uniquement si il existe *un et un seul* y tel que $x\mathcal{R}y$.
Nous suivrons cet usage.

Exercices

Prouvez que $E \times F = \varnothing \iff E = \varnothing$ ou $F = \varnothing$.

Soit $\mathcal{R}$ une relation d'ordre sur E. Comment décrire la relation d'*ordre* $\mathcal{R}'$ *« inverse »* de $\mathcal{R}$?

Soit $\mathcal{R}$ une relation d'équivalence sur E.
Pour $x \in E$ on note $\overline{x} = \{y : x\mathcal{R}y\}$.
Montrez que $x\mathcal{R}y \iff \overline{x} = \overline{y}$.
On dit que $A, B, C, \ldots$ forment une *partition* de E ssi ils sont disjoints deux à deux ($A \cap B = A \cap C = B \cap C \ldots = \varnothing$) et recouvrent E ($A \cup B \cup C \ldots = E$)
Montrez que les ensembles $\overline{x}$ forment une *partition* de E.
Note : on appelle $\overline{x}$ la *classe d'équivalence* de x.

Propriétés des fonctions

Si $f \subset E \times F$ est une fonction, on dit aussi que f est une fonction de E *vers* F. Si $y = f(x)$ on dit que y est l'*image* de x par f et que x est un *antécédent* de y.

Surjectivité

Une fonction surjective est telle que tout élément de F a *au moins* un antécédent dans E.

Injectivité

Une fonction injective est telle qu'un élément de F a *au plus* un antécédent dans E : $f(x) = f(y) \Rightarrow x = y$

Bijectivité

Une fonction bijective est injective et bijective.

Construction des nombres naturels

Giuseppe Peano (1889)

Il existe un ensemble $\mathbb{N}$ (et un élément 0 et une fonction S de $\mathbb{N}$ dans $\mathbb{N}$) tels que (en notant Sn pour $S(n)$) :

- $0 \in \mathbb{N}$
- Pour tout $n \in \mathbb{N}$, il existe $Sn \in \mathbb{N}$ (successeur de n)
- Il n'y a aucun élément n de $\mathbb{N}$ tel que $Sn = 0$
- Si $Sn = Sm$ alors $n = m$ (S est *injective*)
- Si $E \subset \mathbb{N}$, $0 \in E$ et $(n \in E \Rightarrow Sn \in E)$ alors $E = \mathbb{N}$

Intuitivement : On a zéro, le successeur est le nombre suivant, 0 n'est le successeur d'aucun nombre, deux nombres différents ont toujours des successeurs différents et... le principe de récurrence.

Opérations

Définitions

- On peut définir l'addition :
  - *(a0)* $n + 0 = n$
  - *(a1)* $x + Sy = S(x + y)$
- ... et la multiplication :
  - *(m0)* $n \times 0 = 0$
  - *(m1)* $x \times Sy = (x \times y) + x$

$S0 + S0$
$= S(S0 + 0)$: application règle *(a1)*
$= S(S0) = SS0$: application règle *(a0)*

C'est une démonstration de :

$$1 + 1 = 2$$

Étonnant, non ?

Exercices

On note $1 = S0$, $2 = SS0$, $3 = SSS0$, $4 = SSSS0$,
$5 = SSSSS0$, $6 = SSSSSS0$
Prouvez (en indiquant les règles appliquées à chaque étape) que :

$$2 + 2 = 4$$

$$2 \times 3 = 6$$

Prouvez les propriétés suivantes de $+$ et $\times$:

- Commutativité : $a + b = b + a$, $a \times b = b \times a$
- Associativité : $(a + b) + c = a + (b + c)$ et $(a \times b) \times c = a \times (b \times c)$
- Distributivité de $\times$ sur $+$: $a \times (b + c) = (a \times b) + (a \times c)$

Définissez une relation d'ordre « naturelle » sur $\mathbb{N}$ sous forme de règles similaires à celles de l'addition et de la multiplication.

Entiers de Von Neumann

Un *modèle* satisfaisant les axiomes de Peano

- $0 = \varnothing$
- Pour un ensemble x, $S(x) = x \cup \{x\}$
- $\mathbb{N}$ est l'intersection de tous les ensembles E contenant 0 et clos pour S $(x \in E \Rightarrow Sx \in E)$

Exercices

- Ecrivez les nombres de 0 à 4 sous forme d'ensembles selon ce modèle
- Implémentez les entiers de Von Neumann sous forme d'une classe en Python.

Entiers de Zermelo

Un autre modèle satisfaisant les axiomes de Peano

- $0 = \varnothing$
- $S(x) = \{x\}$

Exercices

- Écrivez les nombres de 0 à 4 sous forme d'ensembles selon ce modèle
- Implémentez les entiers de Zermelo sous forme d'une classe en Python.

Retour sur les classes d'équivalences

On a vu que si $\mathcal{R}$ est une relation d'équivalence sur un ensemble E l'ensemble des classes d'équivalence $\overline{x} = \{y \in E : x\mathcal{R}y\}$ forment une *partition* de E.

Un élément d'une *classe d'équivalence* est appelé un *représentant* de cette classe.

On appelle l'ensemble des classes d'équivalence pour la relation $\mathcal{R}$ l'ensemble *quotient* de E par $\mathcal{R}$, noté $E/\mathcal{R}$

Exemples

- Classes d'équivalence de droites parallèles du plan : *directions*
- Classes d'équivalence de bi-points équipollents (*i.e.* (A, B) et (C, D) forment un parallèlogramme $ABDC$) : *vecteurs*

Opérations et classes d'équivalences

Compatibilité

Étant donnée une relation d'équivalence $\sim$.
Si une opération *binaire* interne $*$ sur E (c'est-à dire une fonction de $E \times E$ dans E) vérifie :

$$\forall (x, y) \in E \times E \quad \forall x' \in \overline{x} \ \ \forall y' \in \overline{y} : x' * y' \in \overline{x * y}$$

On dit que la relation $*$ est *compatible* avec $\sim$.

Autrement dit : si on applique $*$ à deux représentants quelconques de leurs classes d'équivalence respectives, le résultat est toujours dans la même classe d'équivalence.

L'opération $*$ permet donc de définir naturellement une opération (qu'on notera de la même façon) sur $E/\sim$.

Construction des nombres relatifs

Classes d'équivalence de couples d'entiers naturels

$\mathbb{Z} = \mathbb{N} \times \mathbb{N}/\sim$ (ensemble quotient des couples d'entiers naturels par la relation d'équivalence $\sim$) où $\sim$ est définie par :

$$(a, b) \sim (c, d) \iff a + d = b + c$$

$+$ et $\times$ étant définis ainsi :

$$(a, b) + (c, d) = (a + c, b + d)$$

$$(a, b) \times (c, d) = (a \times c + b \times d, a \times d + b \times c)$$

Ces définitions sont compatibles avec la relation $\sim$.

$n \in \mathbb{N}$ est identifié à la classe d'équivalence de $(n, 0)$.
Les classes d'équivalence qui contiennent $(0, n)$ sont notées $-n$.

Construction des nombres rationnels

Classes d'équivalence de couples d'entiers relatifs

$\mathbb{Q} = \mathbb{Z} \times \mathbb{Z}^* / \sim$ (ensemble quotient des couples d'entiers relatifs par la relation d'équivalence $\sim$) où $\sim$ est définie par :

$$(a, b) \sim (c, d) \iff a \times d = b \times c$$

$+$ et $\times$ étant définis ainsi :

$$(a, b) + (c, d) = (a \times d + b \times c, b \times d)$$

$$(a, b) \times (c, d) = (a \times c, b \times d)$$

Ces définitions sont compatibles avec la relation $\sim$.

$n \in \mathbb{Z}$ est identifié à la classe d'équivalence de $(n, 1)$.
Les classes d'équivalence qui contiennent (n, m) sont notées $\frac{n}{m}$

Construction des nombres réels

Couples d'intervalles particuliers de $\mathbb{Q}$

$(A, B) \in \mathcal{P}(\mathbb{Q}) \times \mathcal{P}(\mathbb{Q})$ tels que :

1. $A \neq \varnothing$, $B \neq \varnothing$
2. $A \cup B = \mathbb{Q}$
3. $\forall q \in A, \forall r \in B : q < r$

N.B. Ces trois premières conditions impliquent que la connaissance de A détermine celle de B (et réciproquement) : $B = \mathbb{Q} \backslash A$ et $A = \mathbb{Q} \backslash B$

4. Si B à une *borne inférieure* dans $\mathbb{Q}$, elle appartient à B

La borne inférieure d'un ensemble B est le plus *grand* des *minorants* (éléments plus petits que tout élément de B), si il existe.

Ce sont des *coupures de Dedekind* (1858) qui définissent ainsi l'ensemble $\mathbb{R}$ des nombres *réels*.

Exemples de coupures de Dedekind

Un nombre rationnel r est identifié à :

$$A =]-\infty, r[\quad \text{et donc } B = [r, +\infty[$$

$$\sqrt{7} \text{ est } A = \left\{ \frac{p}{q} \in \mathbb{Q} : p^2 < 7q^2 \right\}$$

$$\sqrt[3]{2} \text{ est } A = \left\{ \frac{p}{q} \in \mathbb{Q} : p^3 < 2q^3 \right\}$$

Une autre construction, équivalente, à partir des suites de nombres rationnels, existe : classes d'équivalence de *suites de Cauchy*.
Intuitivement : un nombre réel est la classe des suites de rationnels qui « convergent » vers lui.

Ce n'est pas une définition circulaire car le *critère de Cauchy* permet de définir une sorte de convergence sans qu'il n'existe nécessairement une valeur limite.

Construction des nombres complexes

Cette fois on part de l'anneau des *polynômes* à une variable et à coefficients dans $\mathbb{R}$:

$$\mathbb{R}[X] = \left\{ a_n X^n + a_{n-1} X^{n-1} + \ldots + a_1 X + a_0 : \ a_i \in \mathbb{R} \right\}$$

La relation d'équivalence $\sim$ sur $\mathbb{R}[X]$ est définie par :

$$P \sim Q \iff P = Q \quad \left[X^2 + 1 \right]$$

i.e. le reste de la division euclidienne de P et Q par $X^2 + 1$ est le même, ou encore $P - Q$ est un multiple du polynôme $X^2 + 1$.

Voilà donc le *corps* des nombres complexes

$$\mathbb{C} = \mathbb{R}[X]/\sim = \mathbb{R}[X]/\left(X^2 + 1 \right)$$

Quelques nombres complexes

Un nombre réel a est identifié à la classe d'équivalence du polynôme constant a.

i est la classe d'équivalence du polynôme X. Tout nombre complexe peut s'écrire $a + ib$ où $(a, b) \in \mathbb{R}^2$.

$$i = \overline{X} = \left\{ P \in \mathbb{R}[X] : P = X \ \ [X^2 + 1] \right\}$$

$$X^2 = -1 \ \ \left[X^2 + 1\right] \Rightarrow -1 \in \overline{X^2} \Rightarrow i^2 = -1$$

Une telle construction, due au jeune Évariste Galois (1811–1832), fonctionne avec n'importe quel polynôme irréductible sur un corps quelconque. On peut, par exemple, en partant du polynôme (dans $\mathbb{Q}[X]$) $X^2 - 2$ construire $\mathbb{Q}[\sqrt{2}] = \{a + b\sqrt{2} : (a, b) \in \mathbb{Q}^2\}$.

Et en informatique ?

L'électronique étant généralement *binaire* (ce n'est pas la seule possibilité) les nombres entiers sont généralement représentés dans cette base, et s'ils sont signés (peuvent être négatifs) plusieurs approches sont possibles :

- Stocker le signe à part (rare)
- Complément à 1
- Complément à 2 (le plus courant)

On ne représente pas *entièrement* $\mathbb{N}$! En complément à 2, par exemple, on est limité à l'intervalle $[-2^{63}, 2^{63} - 1]$ (en 64 bits).

Les nombres décimaux sont stockés selon le même principe avec *mantisse* (chiffres binaires « après la virgule ») et *exposant*.

Pour les traitements comptables et financiers on préfèrera le *Décimal codé binaire* (DCB en français, BCD en anglais).

Quelques surprises. . .

En Python, mais pas seulement. . . Attention à *float* !

```
>>> 1/10 + 2/10 == 3/10
False
>>> from decimal import Decimal
>>> Decimal(0.1) + Decimal(0.2) == Decimal(0.3)
False
>>> Decimal('0.1') + Decimal('0.2') == Decimal('0.3')
True
>>> from fractions import Fraction
>>> Fraction(1,10) + Fraction(2,10) == Fraction(3,10)
True
```

En Python les entiers n'ont pas de limitation

```
>>> print(2**63, 2**63 + 1)
9223372036854775808 9223372036854775809
```

D'autres surprises...

PHP

```php
<?php
print(2**62); print("\n"); print(2**63);
?>
```

```
4611686018427387904
9.2233720368548E+18
```

JavaScript

```javascript
console.log(9223372036854775806)
```

```
9223372036854776000
```

Parlons un peu à la machine (presque) directement...

```c
#include<stdio.h>
#include<stdlib.h>

int main() {
  int     n1 = 2147483647;          /* 2**31 - 1 */
  int64_t n2 = 9223372036854775807; /* 2**63 - 1 */
  printf("%+d\t%+lld\n", n1, n2);
  printf("%+d\t%+lld\n", n1 + 1, n2 + 1);
  exit(0);
}
```

```
$ make && ./maxi
cc -o maxi maxi.c
+2147483647     +9223372036854775807
-2147483648     -9223372036854775808
```

Autres classes de nombres

D'autres types de nombres exotiques ont été définis par des mathématicien·e·s et peuvent avoir des applications en physique ou informatique.

- Quaternions (comme les complexes mais avec quatre composantes réelles). Il sont utilisés en représentation de l'espace 3D (robotique, images de synthèse, jeux vidéos)
- Octonions

- Nombres *surréels* de Conway, utiles en *théorie des jeux*
- Infinitésimaux en calcul différentiel et intégral (analyse *non-standard*)

- Nombres *p-adiques* : nombre de chiffres infini en base p, p étant un nombre premier.

Nombres cardinaux

Définition

Soient deux ensembles E et F, on dit qu'ils ont le même cardinal ssi il existe une *fonction bijective* de E dans F.
On note : $\mathbf{card}(E) = \mathbf{card}(F)$ ou encore $|E| = |F|$

Examples

- Le cardinal d'un ensemble *fini* s'identifie à un entier naturel
- Tout ensemble a le même cardinal que lui-même ($f = Id$)
- L'ensemble des nombres entiers pairs a le même cardinal que celui de tous les nombres entiers
- De même que les ensembles des nombres impairs, carrés, premiers, ...
- L'ensemble des rationnels a le même cardinal que celui de l'ensemble des nombres entiers : $|\mathbb{N}| = |\mathbb{Q}|$ mais celui de l'ensemble des réels est « plus grand » : $|\mathbb{N}| < |\mathbb{R}|$
- Deux segments de droite ont le même cardinal

Ordinaux

Bon ordre

Un ensemble ordonné $(E, \leqslant)$ est dit *bien ordonné* si tout sous-ensemble de E admet un plus petit élément.

Deux ensembles *bien ordonnés* $(E, \leqslant)$ et $(F, \leqslant')$ ont le même *ordinal* si il existe une fonction bijective f de E dans F préservant les relations d'ordre sur E et F :

$$x \leqslant y \iff f(x) \leqslant' f(y)$$

Un *nombre ordinal* peut se concevoir comme la classe d'équivalence d'ensembles partageant un même ordinal.

N.B. Cette définition n'est pas tout à fait bien fondée, les classes en question étant plus grandes que des ensembles dans la plupart des schémas d'axiomes. Voir la construction de Von Neumann.

Ensembles finis et infinis

Il a longtemps été tenu comme problématique de considérer des collections infinies comme un tout : du fait qu'elles peuvent être mises en correspondance (bijection) avec une partie stricte d'elles-mêmes.

C'est la définition *positive* des ensembles infinis depuis la fin du XIXe siècle.

- Entiers naturels / nombres pairs : $n \to 2n$
- Segment de droite / moitié du segment : $x \to x/2$
- Demi-droite / segment : $x \to \frac{1}{x}$
- Points d'un cercle, d'un disque, ...

Les ensembles *finis* sont ceux où une telle bijection entre E et un sous-ensemble strict F de E ($F \subset E, F \neq E$) n'existe pas.
Leurs cardinaux s'identifient à un entier naturel, leurs ordinaux aussi.

Ensembles dénombrables

Un ensemble infini est *dénombrable* s'il peut être mis en bijection avec $\mathbb{N}$. On note son *cardinal* $\aleph_0$.
Il n'existe pas d'ensemble infini *plus petit* que $\mathbb{N}$ *i.e.* tout ensemble infini contient un sous-ensemble dénombrable.
Attention : dans certains contextes *dénombrable* inclut aussi les ensemble *finis*.

Exemples

- Nombres pairs, impairs, premiers, carrées, relatifs, ...
- Paires de nombres entiers (a, b)
- n-uplets de nombres entiers $(a, b, c, ...)$
- Nombres rationnels
- Mots (en théorie des langages) construits sur un alphabet fini
- Programmes d'une machine de Turing
- Programmes dans un langage donné
- Théorèmes d'une théorie formelle

Parties d'un ensemble

Ensemble des sous-ensembles

On note $\mathcal{P}(E)$ l'ensemble des sous-ensembles de E.
Appelé aussi *ensemble des parties* de E (qui contient, entre autres, $\varnothing$ et E lui-même).

Fonction indicatrice

Pour $F \subset E$ (*i.e.* $F \in \mathcal{P}(E)$), on note $\mathbb{1}_F$ la fonction de E vers $\{0,1\}$ définie par :

$$\mathbb{1}_F(x) = \begin{cases} 1 & \text{si } x \in F \\ 0 & \text{si } x \notin F \end{cases}$$

Inversement si f est une fonction de E vers $\{0,1\}$, elle est la fonction indicatrice de l'ensemble :

$$\{x \in E : f(x) = 1\} \in \mathcal{P}(E)$$

Argument diagonal

$\mathbb{R}$ n'est pas *dénombrable*

On montre qu'une *partie* de $\mathbb{R}$, l'intervalle $[0, 1[$ n'est pas dénombrable (Cantor).
S'il l'était on pourrait constituer une suite de tous les réels de $[0, 1[$ sous forme décimale :

$$
\begin{array}{llllllll}
x_1 = 0, & \mathbf{d_{11}} & d_{12} & d_{13} & d_{14} & d_{15} & d_{16} & d_{17} \ \ldots \\
x_2 = 0, & d_{21} & \mathbf{d_{22}} & d_{23} & d_{24} & d_{25} & d_{26} & d_{27} \ \ldots \\
x_3 = 0, & d_{31} & d_{32} & \mathbf{d_{33}} & d_{34} & d_{35} & d_{36} & d_{37} \ \ldots \\
x_4 = 0, & d_{41} & d_{42} & d_{43} & \mathbf{d_{44}} & d_{45} & d_{46} & d_{47} \ \ldots \\
x_5 = 0, & d_{51} & d_{52} & d_{53} & d_{54} & \mathbf{d_{55}} & d_{56} & d_{57} \ \ldots \\
x_6 = 0, & d_{61} & d_{62} & d_{63} & d_{64} & d_{65} & \mathbf{d_{66}} & d_{67} \ \ldots \\
x_7 = 0, & d_{71} & d_{72} & d_{73} & d_{74} & d_{75} & d_{76} & \mathbf{d_{77}} \ \ldots \\
\vdots & & & & & & & \ddots
\end{array}
$$

Argument diagonal

À partir de la *diagonale* des *chiffres* placés à la n^{e} décimale de x_n.
On peut contruire un réel, *antidiagonal*, $r \in [0, 1[$:

$$r = 0, \delta_1 \delta_2 \delta_3 \dots \delta_n \dots$$

$$\text{où } \delta_n = \begin{cases} 5 & \text{si } d_{nn} \neq 5 \\ 0 & \text{si } d_{nn} = 5 \end{cases} \quad \text{donc } \delta_n \neq d_{nn}$$

Pour tout $n \geqslant 1, r \neq x_n$
Car r et x_n diffèrent, au moins, sur leur n^{e} décimale.
r n'est donc pas dans la liste, une telle liste n'existe donc pas.

Un argument similaire permet de prouver que $|\mathcal{P}(E)| > |E|$ ou encore qu'il n'existe pas de programme capable de préduire l'arrêt de tout autre programme en connaissant son entrée.

Exercices

Combien un ensemble *fini* de taille n a-t-il de parties ?

Montrer que pour tout ensemble (fini ou infini), il n'existe pas de bijection entre lui et l'ensemble de ses parties.
En conclure que $|\mathcal{P}(E)| > |E|$.

Conclusion

Il existe donc des ensembles infinis strictement plus grand que $\mathbb{N}$, *i.e. non-dénombrables* (cardinaux $> \aleph_0$):

- $\mathcal{P}(\mathbb{N})$: cardinal $\aleph_1$
- $\mathbb{R}$, $\mathbb{R}^n$

Et, au-delà, des ensembles infinis de *« plus en plus grands »* :

- $\mathcal{P}(\mathcal{P}(\mathbb{N})), \mathcal{P}(\mathcal{P}(\mathcal{P}(\mathbb{N}))), \ldots$
- Dont les cardinaux sont notés $\aleph_2, \aleph_3, \ldots$
- *L'infini* (ScienceÉtonnante) : https://www.youtube.com/watch?v=1YrbUBSo4Os

The infinite we shall do right away. The finite may take a little longer.

— Stanislaw Ulam

La nature de l'infini est telle que des pensées finies ne le sauraient comprendre.

— Descartes, *Principes*

Suites

Une suite d'éléments d'un ensemble E est une fonction de $\mathbb{N}$ dans E définie à partir d'un certain rang (généralement 0 ou 1).

Limite ensembliste

Si les éléments d'une suite sont des ensembles on peut définir les limites respectivement *inférieure* et *supérieure* :

$$\underline{\lim}\ E_n = \bigcup_{n\geqslant 0} \bigcap_{k\geqslant n} E_k$$

$$\overline{\lim}\ E_n = \bigcap_{n\geqslant 0} \bigcup_{k\geqslant n} E_k$$

Si $\underline{\lim}\ E_n = \overline{\lim}\ E_n$ alors on dit que la suite converge au sens de la théorie des ensembles et, par définition :

$$\lim E_n = \underline{\lim}\ E_n = \overline{\lim}\ E_n$$

Limite ensembliste

Intuitivement, la limite au sens de la théorie des ensembles concerne ce dont parle la théorie : appartenance ou non.

Un élément est membre de la *limite ensembliste* (si elle existe) dès lors qu'il est *acquis* à un certain rang de la suite et conservé ensuite par *tous* les termes suivants de la suite.

Cette limite peut donc facilement se retrouver *vide*.
Si un élément est régulièrement *acquis*, puis *perdu*, puis encore *acquis* ensuite, et ainsi de suite, il sera membre de la limite supérieure mais *pas* de la limite inférieure.
La fonction indicatrice va alors *osciller* entre 0 et 1 pour cet élément.

Les calculs de *complexité algorithmique* peuvent nécessiter des raisonnements portant sur la limite ensembliste.

Limite ensembliste

Théorème : Si la limite de la suite des fonctions indicatrices existe, la fonction limite obtenue est la fonction indicatrice de l'ensemble limite au sens précédent.
Les fonctions indicatrices ne peuvent prendre que deux valeurs en chaque point (0 ou 1), donc *si* une telle limite existe c'est que, *en chaque point*, les valeurs prises en un point ne changent plus à partir d'un certain terme (cf. *On Ducks and Bathtubs*, http://bsb.me.uk/dd-wealth.pdf, par Ben Bacarisse).

Exercice

Soit $F_n = \{0, ..., n\} = \{k \in \mathbb{N} : k \leqslant n\}$.
Déterminer, si elle existe, la limite ensembliste de la suite (F_n).

- Vous pouvez la déterminer à partir des formules des limites supérieure et inférieure.
- Vous pouvez aussi la déterminer en considérant la limite de la suite des fonctions indicatrices des ensembles F_n.

Solution de l'exercice

Première version (limites supérieure et inférieure)

Pour tout $n \in \mathbb{N}$:

$$\bigcap_{k \geqslant n} F_k = \{0, ..., n\} \cap \{0, ..., n+1\} \cap \ldots = \{0, ..., n\} = F_n \text{ donc :}$$

$$\underline{\lim}\, F_n = \bigcup_{n \geqslant 0} \overbrace{\bigcap_{k \geqslant n} F_k}^{F_n} = \bigcup_{n \geqslant 0} F_n = \{0\} \cup \{0, 1\} \cup \{0, 1, 2\} \cup \ldots = \mathbb{N}$$

Pour tout $n \in \mathbb{N}$: $\bigcup_{k \geqslant n} F_k = \{0, ..., n\} \cup \{0, ..., n+1\} \cup \ldots = \mathbb{N}$ donc :

$$\overline{\lim}\, F_n = \bigcap_{n \geqslant 0} \overbrace{\bigcup_{k \geqslant n} F_k}^{\mathbb{N}} = \bigcap_{n \geqslant 0} \mathbb{N} = \mathbb{N} \cap \mathbb{N} \cap \ldots = \mathbb{N}$$

Les limites supérieure et inférieure étant égales à $\mathbb{N}$, la suite (F_n) a une limite qui est $\mathbb{N}$.

Solution de l'exercice

Seconde version (fonctions indicatrices)

Soit $\mathbb{1}_{F_n}$ la fonction indicatrice de l'ensemble F_n

$$\mathbb{1}_{F_n}(k) = \begin{cases} 1 & \text{si } k \leqslant n \\ 0 & \text{si } k > n \end{cases}$$

Donc, pour tout $k \in \mathbb{N}$:

$$\mathbb{1}_{F_n}(k) = 1 \text{ si } n \geqslant k$$

$$\text{donc } \forall k \in \mathbb{N} : \lim_{n \to +\infty} \mathbb{1}_{F_n}(k) = 1$$

la suite de fonctions $(\mathbb{1}_{F_n})$ converge donc (au sens des fonctions) vers la fonction constante : $k \to 1$, qui est la fonction indicatrice $\mathbb{1}_{\mathbb{N}}$ de $\mathbb{N}$.
La suite (F_n) a donc comme limite $\mathbb{N}$.

Arithmétique modulaire

Ou *arithmétique de l'horloge* : on réalise les calculs et on ne garde que le reste de la division euclidienne par un certain entier n (calculs « *modulo* n »), l'ensemble de travail est *fini*. On constate que l'addition et la multiplication fonctionnent presque comme d'habitude.

Par exemple : $3 \times 4 + 7 = 1 \ [6]$, mais $3 \times 2 = 0 \ [6]$ (diviseurs de zéro !)

Si p est un nombre premier, alors il n'y a pas de diviseurs de zéro. $\mathbb{Z}/p\mathbb{Z}$ est alors un *corps* : même propriétés algébriques essentielles que $\mathbb{R}$ et $\mathbb{C}$. Sinon c'est seulement un *anneau*.

On peut cependant construire des corps finis à p^n éléments si p est premier.

Applications des calculs modulaires

Dès qu'un système de comptage *boucle*, les calculs modulaires sont utiles : *temps*, *dates*, *rotations*, etc.

Dès que l'on travaille sur des nombres stockés en espace fixe en mémoire on travaille de cette façon.

Cryptographie

- La plupart des algorithmes de chiffrement, symétriques ou non, de calcul de *sommes de contrôle* les utilisent
- Dans SSL : RSA, Diffie-Hellman
- Sommes MD5, SHA, etc.

Dénombrement

La combinatoire est l'art de compter combien il existe de structures construites à partir d'ensembles *finis*.

- Nombre de parties d'un ensemble à n éléments : 2^n
- Nombre de *permutations* de n objets, factorielle de n : $n! = 1 \times 2 \times ... \times n$ et $0! = 1$
- Nombre de *combinaisons* de p objets parmi n :

$$\mathrm{C}_n^p = \binom{n}{p} = \frac{n!}{p!(n-p)!}$$

- Nombre d'*arrangements* de p objets parmi n :

$$\mathrm{A}_n^p = P(n,p) = n \times (n-1) \times (n-2) \times (n-p+1)$$

$$\mathrm{A}_n^p = p!\binom{n}{p} = \frac{n!}{(n-p)!}$$

Formule du binôme

Newton

$$(a+b)^2 = a^2 + 2ab + b^2$$

$$(a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$$

$$(a+b)^4 = a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4$$

$$\dots$$

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k$$

Facile à retrouver !

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Triangle de Pascal

$$\binom{0}{0} = \mathbf{1}$$

$$\binom{1}{0} = \mathbf{1} \qquad \binom{1}{1} = \mathbf{1}$$

$$\binom{2}{0} = \mathbf{1} \qquad \binom{2}{1} = \mathbf{2} \qquad \binom{2}{2} = \mathbf{1}$$

$$\binom{3}{0} = \mathbf{1} \qquad \binom{3}{1} = \mathbf{3} \qquad \binom{3}{2} = \mathbf{3} \qquad \binom{3}{3} = \mathbf{1}$$

$$\binom{4}{0} = \mathbf{1} \qquad \binom{4}{1} = \mathbf{4} \qquad \binom{4}{2} = \mathbf{6} \qquad \binom{4}{3} = \mathbf{4} \qquad \binom{4}{4} = \mathbf{1}$$

En Python

Le module *itertools* permet de générer des permutations, des combinaisons, sans et avec remise, et génère des itérables.

```
>>> from itertools import (product, permutations,
        combinations, combinations_with_replacement,
        groupby)
>>> list(combinations( ['spam', 'ham', 'egg'], 2))
[('spam', 'ham'), ('spam', 'egg'), ('ham', 'egg')]
```

Voir aussi le module *sympy.combinatorics*

Exercice

Écrire une fonction qui renvoie les *arrangements* de p objets parmi une liste. Indice : combinez combinations et permutations, pensez à utiliser éventuellement *chain*.

Pour juste les compter

Calculez efficacement les nombres de combinaisons, d'arrangements, de permutations, etc. ? Dans le module *math* : *comb* et *factorial*. Voir aussi le module *scipy.special.comb*.

```
from math import factorial, comb
def k_perm(n, p):
    '''Calcule le nombre d'arrangements
    de p objets parmi n.'''
    return factorial(p)*comb(n,p)
```

Probabilités

Axiomatisation de Kolmogorov (1933)

Soit un ensemble Ω appelé *univers*. Une probabilité sur Ω c'est une fonction P de $\mathcal{P}(\Omega)$ vers $[0,1]$ [a]. Qui vérifie :

- $P(\Omega) = 1$ (la probabilité de l'univers vaut 1)
- Si $A_1, A_2, ...$ sont disjoints deux à deux (pour tout i, j, si $i \neq j$ alors $A_i \cap A_j = \varnothing$) alors (additivité) :

$$P\left(\bigcup_{i \geqslant 1} A_i\right) = \sum_{i \geqslant 1} P(A_i)$$

[a] Plus rigoureusement d'une *tribu* sur Ω vers $[0,1]$.

Un évènement est un sous-ensemble de Ω, deux évènements A et B sont dits indépendants ssi :

$$P(A \cap B) = P(A) \times P(B)$$

Probabilités conditionnelles

Si A et B sont deux évènements, B de probabilité non-nulle, la *probabilité conditionnelle de A sachant B* est :

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Si A et B sont indépendants $P(A|B) = P(A)$.

Théorème de Bayes

Si $P(A)$ et $P(B)$ sont non-nulles :

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Variables aléatoires

Une variable aléatoire est une fonction X est une fonction *mesurable* de Ω vers un *espace mesurable* (généralement $\mathbb{R}$).

On peut définir une probabilité P_X à partir de X (X^{-1} est la fonction inverse de X) : $P_X(A) = P(X^{-1}(B)) = P(X \in B)$
Dans le cas d'une variable X *non-discrète* ($X(\Omega)$ ni fini, ni dénombrable), si $P(a \leqslant X \leqslant b) = \int_a^b f(x)dx$ on appelle f sa *fonction de densité*.

L'espérance de X est :

- $E[X] = \sum_i x_i P(X = x_i)$ si X est *discrète*
- $E[X] = \int_{-\infty}^{+\infty} x f(x) dx$ si X est *continue*

On peut voir l'*espérance* comme une moyenne en statistique, on peut aussi définir l'écart-type, la variance, etc.

Probabilités discrètes et combinatoire

Probabilité uniforme

Si Ω est fini, si on définit P ainsi, pour $x \in \Omega$:

$$P(\{x\}) = \frac{1}{|\Omega|}$$

Ça suffit pour définir P sur tout $\mathcal{P}(\Omega)$, pour $A \in \mathcal{P}(\Omega)$:

$$P(A) = \frac{|A|}{|\Omega|}$$

Exemple

On lance deux dés : $\Omega = \{1, 2, 3, 4, 5, 6\} \times \{1, 2, 3, 4, 5, 6\}$.
A : « La somme des tirages est paire »

$$A = \{(1, 1), (1, 3), ..., (2, 5), ...\}$$

Comment choisir Ω et P ?

On lance deux pièces de monnaie. On se demande quelle est la probabilité que les deux pièces tombent sur deux faces différentes. . .

Comment peut-on représenter une *épreuve* (un tirage) ? On peut en imaginer deux selon que les distingue les pièces ou pas.

$$\{\ (F,F),(F,P),(P,F),(P,P)\ \}$$
$$\{\ \{F\},\{F,P\},\{P\}\ \}$$

Si la probabilité est uniforme sur ces deux univers, les résultats sont-ils les mêmes ? Lequel est le bon ?

Ce n'est pas une question mathématique, seules l'intuition et l'expérience peuvent nous fournir la réponse.

Lambda-calcul

Créé par Alonzo Church (1903–1995), basé sur les notions, *syntaxiques*, de *fonction* et d'*application*.

Fonction identité : $\lambda x.x$
Fonction qui renvoie une fonction constante : $\lambda x.(\lambda y.x)$

Règles de substitution et de réduction

Substitution de variables dans un terme :

$$(\lambda x.(\lambda y.x))[x = u] \text{ donne } (\lambda u.(\lambda y.u))$$

Réduction :

$$(\lambda x.xx)(\lambda y.y) \text{ donne } (\lambda y.y)(\lambda y.y)$$

Programmation et lambda-calcul

C'est un langage de programmation !

Vrai $= \lambda ab.a$
Faux $= \lambda ab.b$
if-then-else $= \lambda buv.(buv)$

On peut construire les nombres entiers !

$0 = \lambda fx.x$
$1 = \lambda fx.fx$
$2 = \lambda fx.f(fx)$
$3 = \lambda fx.f(f(f(x)))$
$Sn = (\lambda nfx.f(nfx)).n$

Et les opérations $+$ et $\times$!

$n + p = (\lambda npfx.nf(pfx)).np$
$n \times p = (\lambda npf.n(pf)).np$

Lambda-calcul dans les langages informatiques

Langages fonctionnels

- LISP et Scheme sont construits directement à partir du λ-calcul :

```
(define sum-squares
            (lambda (x y)
               (+ (* x x) (* y y))))
(sum-squares 7 10) ; 149
```

- CaML est construit sur le λ-calcul *typé*
- Clojure, Haskell, ...

Concepts fonctionnels

- *lambdas* de Python
- Fonctions anonymes de JavaScript

Machines de Turing

Alan Turing (1912–1954) définit une machine théorique

- Ruban de longueur infinie, découpé en cases
- Alphabet : symboles présents ou non sur une case (symbole *blanc*), on peut se ramener à 0/1/blanc (binaire) ou I/blanc (unaire)
- Tête de lecture : placée sur une case, peut la lire et y écrire

Programme

Liste d'états : I, II, III, ...

Chaque état décrit une action :

Selon ce que contient la case...

- Modifier la case (gomme et crayon)
- Changer d'état ou non ou s'arrêter (HALT)
- Déplacer ou non d'une case à gauche ou à droite

Examples en live sur : https://turingmachine.io/

Résultats mathématiques

Machine *universelle*

Tout ce qui est intuitivement calculable est calculable avec une machine de Turing. Il existe une machine de Turing *universelle* :

- Un programme pour une machine de Turing peut être représenté par un nombre. Il suffit d'utiliser des conventions pour décrire états, actions, etc.
- Il existe (quel que soit le codage) une Machine de Turing qui prend en entrée sur le ruban:
  - Le codage d'une machine de Turing
  - L'entrée supposée

et qui va l'exécuter (la *simuler*) sur l'entrée

Indécidabilité de l'*arrêt*

Il n'existe pas de machine de Turing prenant en entrée une machine de Turing et son entrée et qui prédit si cette dernière s'arrêtera ou pas.

La bonne et la mauvaise nouvelle

La mauvaise

Si on peut fabriquer un ordinateur, on ne pourra jamais en faire un autre qui fait quelque chose de plus que les autres...

- On pourra le faire plus vite
- On pourra le faire moins cher
- On pourra le faire avec plus de couleurs
- mais rien de plus (et rien de moins)

La bonne

Universalité

En programmation ça s'exprime par la propriété d'être *Turing complete* pour un langage : tous les langages de programmation permettent de faire strictement la même chose (il suffit de pouvoir y programmer une machine de Turing universelle)

- Pascal, C, Python, JS, ... sont *Turing complete*
- SQL (de base, standard ISO) ne l'est pas
- CSS (HTML 5) est Turing complete !

D'une machine abstraite à une machine concrète

Alan Turing n'a pas seulement inventé une machine théorique. Il a, pendant la 2nd guerre mondiale, participé à la construction d'une machine aidant au décryptage des messages ennemis, à Bletchley Park (UK), dont l'architecture a inspiré les premiers ordinateurs électroniques.

Calcul propositionnel

On a des *propositions* élémentaires $p, q, ...$ qui peuvent être vraies ou fausses et des opérateurs *non*, *ou*, *et*, *implique*, ... notés $\neg, \vee, \wedge, \rightarrow$ .... Une formule logique est, par exemple :

$$(p \vee q) \wedge (\neg p \rightarrow q)$$

Les règles de transformations sont similaires à celle de la théorie des ensembles (*et* correspond à *intersection*, *ou* à *réunion*, *non* à *complémentaire*, *implique* est *inclus dans*).

$$\neg(p \vee q) \equiv (\neg p \wedge \neg q)$$

Utile à connaître pour simplifier, déboguer un test *if*, *while*, ...

Table de vérité

On peut définir un opérateur par sa *table de vérité* et calculer celle de n'importe quelle expression logique.

p	q	$p \wedge q$	$p \vee q$	$p \to q$	$p \equiv q$
V	V	V	V	V	V
V	F	F	V	F	F
F	V	F	V	V	F
F	F	F	F	V	V

p	q	$\neg p$	$p \to q$	$q \vee \neg p$	$(p \to q) \equiv (q \vee \neg p)$
V	V	F	V	V	V
V	F	F	F	F	V
F	V	V	V	V	V
F	F	V	V	V	V

Déduction naturelle

Manipulations *syntaxiques*

Règles permettant de *déduire* des formules à partir d'hypothèses.

À gauche : élimination de l'implication (*modus ponens*).

$$\dfrac{A \quad A \to B}{B} \qquad \dfrac{A \wedge B}{A} \qquad \dfrac{A}{A \vee C}$$

À droite :

- élimination de la conjonction
- introduction de la disjonction.

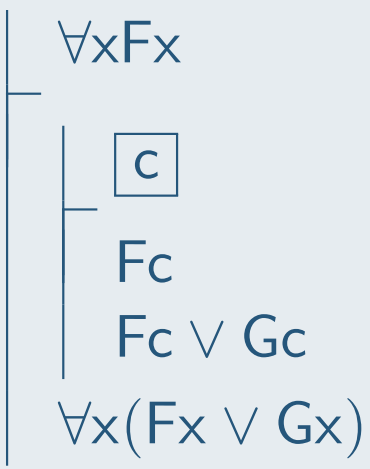
Logique des prédicats

Quantificateurs et prédicats

Introduction de quantificateurs : *il existe* ($\exists$), *pour tout* ($\forall$) et de *prédicats* ($P(a)$ *i.e.* P est vrai pour a)

$$\forall x\ P(x) \iff \neg\exists x\ \neg P(x)$$

- Pas de tables de vérité
- Des règles de déduction supplémentaires



Théorèmes de Gödel

Problème posé par David Hilbert (1862–1943) :

Peut-on prouver la consistance de l'arithmétique ? En d'autres termes, peut-on démontrer que les axiomes de l'arithmétique ne sont pas contradictoires ?

Kurt Gödel (1906–1978) démontre plusieurs théorèmes :

Un théorème de *complétude*, qui démontre que si une proposition est *indécidable* (ni démontrable, ni réfutable) dans une théorie alors il existe deux modèles de cette théorie, l'un où elle est vraie et l'autre où elle est fausse.

Théorèmes d'incomplétude

- Il existe, dans toute théorie formelle exprimant l'arithmétique, des énoncés *indécidables*
- Il est *impossible* de démontrer la non-contradiction de l'arithmétique

Raisonner sur un programme

Vous n'aimez pas les *bugs* ?

Il n'est pas vain de vouloir prouver qu'un programme est *correct* !

Invariants de boucles

Dans toute boucle on devrait pouvoir déterminer une propriété *invariante*, toujours *vraie*.
Si elle reste vraie en *sortie* de boucle on a alors prouvé que notre code est correct.

Exemple

Dans l'algorithme de *tri par insertion*, on peut *démontrer* que le tableau jusqu'à un certain index est trié et que la condition de sortie implique que l'index a atteint la fin du tableau.
D'autres algorithmes de tri auront d'autres invariants et conditions mais une conclusion finale identique : le tableau est trié.

Logique intuitionniste

Constructionnisme

Issue des réflexions du mathématicien Luitzen Egbertus Jan Brouwer (1881–1966) sur la notion de *preuve constructive*.

Des propositions tautologiques (toujours vraies) en logique classique ne le sont plus nécessairement :

$$p \vee \neg p \text{ (tiers exclu)}$$

$$\neg\neg p \to p$$

(par contre $p \to \neg\neg p$ est vraie)

Se généralise à la logique des prédicats.

Programmes et preuves

Un théorème de logique intuitionniste impliquant le λ-calcul typé.

Isomorhisme de Curry–Howard

Tout programme est une *preuve* logique, ce qui est *prouvé* est le *type* du programme.

$$P(x : A) \to B \text{ est une preuve de } A \Rightarrow B$$

Intuitivement : le programme P transforme une *preuve* de A en une *preuve* de B.

C'est la base du développement du système de preuve de programmes *CoQ*.
Il existe un compilateur C prouvé correct par *CoQ*, ainsi que des systèmes de commande critiques (aéronautique).
CoQ peut aussi prouver des théorèmes mathématiques.

Logique·s

Pour aller plus loin

Logiques pour l'intelligence artificielle, P. Gribomon :
https://people.montefiore.uliege.be/lens/logic/notes/LogProp.pdf

Algèbre linéaire

On rencontre extrêmement souvent des ensembles munis d'une addition et d'une multiplication par un *« scalaire »* (typiquement un nombre réel, plus généralement un élément d'un *corps* K comme $\mathbb{R}$, $\mathbb{C}$ ou un corps fini) tels que :

$$\forall u, v \in E, \forall \lambda \in K : u + \lambda v \in E$$

Et sur ces ensembles dits *vectoriels*, des fonctions intéressantes (transformations) sont *linéaires* :

$$\forall u, v \in E, \forall \lambda \in K : f(u + \lambda v) = f(u) + \lambda f(v)$$

Espaces de vecteurs du plan, de l'espace, de fonctions particulières (continues par exemple), de solutions d'une équation différentielle, de suites, de polynômes, de séries, ...

Bases, dimension, coordonnées

Une famille d'éléments (*vecteurs*) est *libre* ou *linéairement indépendante* ssi *aucun* d'entre eux ne peux être exprimé comme combinaison linéaire des autres. Autrement dit :

$$\lambda_1 v_1 + \lambda_2 v_2 + ... + \lambda_n v_n = 0 \Rightarrow \lambda_1 = \lambda_2 = ... = \lambda_n = 0$$

Bases

Il existe des familles libres (*bases*) telles que tout vecteur en soit une combinaison linéaire.
Théorème : toutes les bases ont la même cardinalité, c'est la *dimension* (finie ou infinie) de l'espace vectoriel.

Coordonnées

Les coefficients λ_i d'un vecteur v dans une base donnée sont appelés *coordonnées* du vecteur v.

Matrices

En dimension finie toute transformation linéaire d'un espace E vers un espace F, de dimensions n et m, munies de bases, peut s'exprimer par un *tableau* de scalaires avec une règle de multiplication spécifique.

Ce sont des *matrices*.

Leur ensemble forme un *anneau* $\mathcal{M}_{mn}(K)$. Il y a des *diviseurs de zéro*.

En dimension 3 vers 2

$$A \cdot v = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} a_{11}v_1 + a_{12}v_2 + a_{13}v_3 \\ a_{21}v_1 + a_{22}v_2 + a_{23}v_3 \end{pmatrix}$$

Matrices et géométrie

La multiplication d'une matrice $n \times m$ par une matrice $m \times n$ se fait d'une façon similaire.

Transformations linéaires

En géométrie une transformation dans le plan où l'espace s'exprime, étant donnée une base, généralement *orthonormée*, par une matrice *carrée* $n \times n$. 2×2 dans le plan, 3×3 dans l'espace.

Matrices de passage

Obtenir les coordonnée d'un vecteur dans une base à partir de celle dans une autre base s'exprime aussi par une matrice $(v = P \cdot u)$.
Ces matrices sont *inversibles* : $\exists P^{-1} \in \mathcal{M}_n(K) : P \cdot P^{-1} = Id$.

Une transformation T, ayant comme matrice A dans une base $\mathcal{E}$, a comme matrice dans une base $\mathcal{E}'$: $P^{-1} \cdot A \cdot P$ où P est la matrice *de passage* entre les deux bases.

Applications : rotations

Python ?

Python propose tout ce qu'il faut pour faire de l'algèbre linéaire en toutes dimensions (*numpy* et *scipy*), manipuler des transformations (*pytransform3d*), afficher le résultat (*matplotlib*, *OpenGL*).

Rotations du plan

Creating a rotation matrix in NumPy :
https://scipython.com/book/chapter-6-numpy/examples/creating-a-rotation-matrix-in-numpy/

Dans l'espaaaace

En 3D avec matrices et quaternions :
https://rock-learning.github.io/pytransform3d/rotations.html

Complexité algorithmique

On peut évaluer en ordre de grandeur le nombre d'opérations élémentaires (lecture mémoire, écriture, calcul de base) que réalise un programme comme fonction de la taille de son entrée.

Si on attend un nombre en paramètre, la taille est le nombre de chiffres (ou de bits) du nombre : $\log(n)$.
Si c'est une collection c'est le nombre d'éléments de la collection.

La complexité est le comportement asymptotique du temps de calcul, notation dite « grand O de Landau » :

$$f(x) \in O(g(x)) \iff \exists k, C \ \ \forall x > k : |f(x)| \leqslant C|g(x)|$$

La constante k (seuil) sert à ignorer les « petites » tailles de données, quand l'initialisation du programme l'emporte sur les calculs, et C (facteur) à ne pas dépendre de la vitesse de la machine utilisée.

Complexité

Complexités courantes

- $O(\log(n))$: logarithmique
- $O(n)$: linéaire, $O(n^2)$: quadratique
- $O(n\log(n))$, $O(n\log(\log(n)))$
- Constant : $O(1)$, polynomial : $O(n^p)$
- Exponentiel : $O(\exp(n))$, pire sur-exponentiel $O(\exp(\exp(x)))$

Beaucoup d'algorithmes sont polynomiaux ou en $O(n\log(n))$, la plupart des problèmes d'optimisation sont exponentiels.

Cas d'école : étudiez la complexité des algorithmes de tri. Examinez les algorithmes choisis par différents langages (Python, JS, Java) ou bases de données.

On peut aussi calculer la complexité en terme d'espace mémoire consommé.

Réduction de complexité

Avec un peu de jugeotte on peut diminuer la complexité en temps ou mémoire d'un calcul.

Méthode de Ruffini-Horner

$$P(x_0) = a_n x_0^n + a_{n-1} x_0^{n-1} + \ldots + a_0$$

Une évaluation directe nécessite $n(n+1)/2$ multiplications et n additions, complexité $O(n^2)$. Si on réécrit l'expression ainsi :

$$P(x_0) = ((\ldots((a_n x_0 + a_{n-1})x_0 + a_{n-2})x_0 + \ldots)x_0 + a_1)x_0 + a_0$$

Plus que n multiplications, la complexité est en $O(n)$!

Cerises sur le gâteau : on peut démontrer que c'est le nombre minimal d'opérations nécessaires et on évite, en plus, d'avoir des valeurs intermédiaires trop grandes qui pourraient dépasser la capacité du type de nombres (entiers, flottants) utilisés.

P et NP

Il existe une classe de problème dit *NP* où il est « facile » (polynomial) de vérifier qu'une solution donnée convient mais présumé « dur » (non-polynomial) d'en trouver une.
Parmi eux, les « plus durs » ont pu êtres prouvé *équivalents* : on peut transformer une résolution de l'un d'entre eux en un autre de façon « facile ». Ce sont les problèmes *NP-complets*.
On ne sait pas si $P = NP$ ou si $P \neq NP$.

Autrement dit il se pourrait qu'on puisse trouver une solution polynomiale pour résoudre tous ces problèmes !

Un des *problèmes du millénaire* de l'institut Clay

Un million de dollar à la clef !
Le consensus est que $P \neq NP$, mais on n'en a pas la preuve.

Complexité algorithmique

Pour aller plus loin

Complexité des algorithmes et notation grand O
Geneviève Savard, Université du Québec :

https://cours.etsmtl.ca/SEG/GSavard/mat210/Documents/grandO.pdf

Théorie des graphes

Un graphe est la donnée d'un ensemble de *nœuds* reliés par des *arrêtes*.
Les arrêtes peuvent être orientées ou non, et munies d'un *poid*.

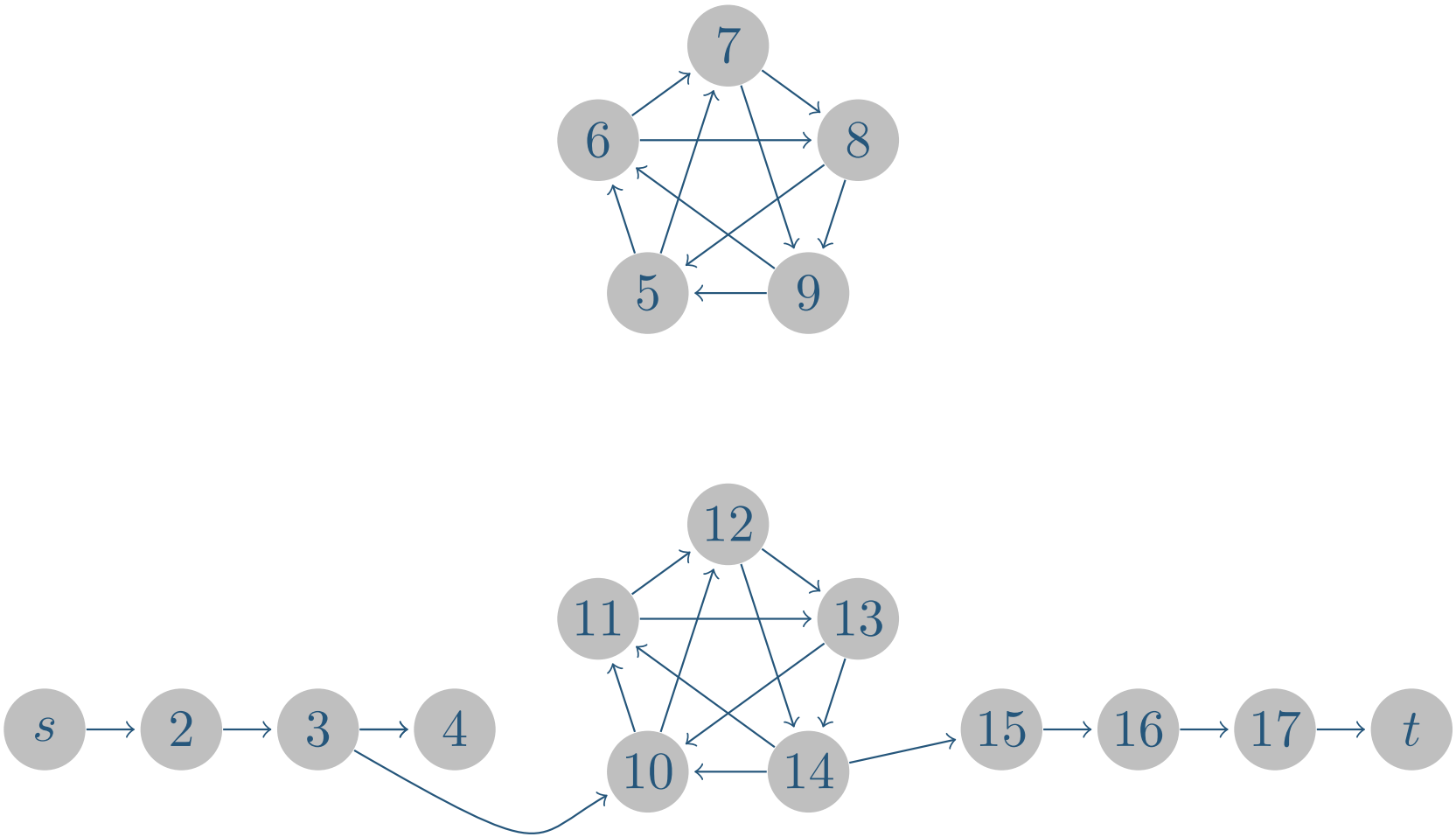
Structure de données courante

- Liens de transports entre villes
- Réseaux informatiques
- Circuits électriques, électroniques, hydrauliques, ...
- Relations d'amitié dans un groupe
- Systèmes de fichiers hiérarchiques
- Automates, machines de Turing, expressions régulières

Représentables de plusieurs façons

- Ensembles des nœuds et des arrêtes (couples de nœuds)
- Matrices d'adjacence
- Un des domaines les plus riches de l'*algorithmique*

Représentation graphique



Quelques logiciels pour faire des mathématiques

Python (avec *numpy*, *scipy*, *pandas*, *sympy*, *matplotlib*, *jupyter*, etc.) sait faire beaucoup de choses mais pas tout !

Impossible, bien sûr, d'être exhaustif

Logiciels *privateurs* (*propriétaires*) :

- *MATLAB*
- *Mathematica*, *Maple*

Logiciels libres

- Calcul numérique : *GNU Octave*, *Scilab*, *Genius*
- Calcul formel : *Maxima*, *Axiom*, *SAGE*
- Arithmétique : *Pari GP*
- Géométrie : *Geogebra*

Langages

R (statistiques), *Julia*, *Scala*, *CaML*, *Haskell*, *Scheme*, *LISP*, ...

Pour conclure. . .

La philosophie reste une discipline menacée dans les classes terminales, et les mathématiques un opérateur ennuyeux de sélection sociale. Eh bien moi, je propose la dernière année de maternelle pour les deux : les gamins de cinq ans sauront assurément faire bon usage de la métaphysique de l'infini comme de la théorie des ensembles.

Alain Badiou avec Georges Haéri, *Éloge des mathématiques*, 2015

Références

The Art of Computer Programming, Donald Knuth, 1969-2020 Addison-Wesley

L'informatique à la lumière de quelques textes de Leibniz, Laurent Bloch : https://www.epi.asso.fr/revue/articles/a1610b.htm

Infinity and the Mind: The Science and Philosophy of the Infinite, Rudy Rucker, 1982

Éléments de mathématiques – Théorie des ensembles, N. Bourbaki, 1970

Théorie des ensembles, Jean-Louis Krivine, 2007

Gödel, Escher, Bach : Les Brins d'une Guirlande Éternelle, Douglas Hofstadter, 1979

On Ducks and Bathtubs, Ben Bacarisse : http://bsb.me.uk/dd-wealth.pdf

Références

Page Oueb du logicien Jean-Yves Girard :
https://girard.perso.math.cnrs.fr/Accueil.html

Page Web du logicien Jean-Louis Krivine :
https://www.irif.fr/~krivine/

Infinite Reflections, Peter Suber (configuration SSL du serveur Web invalide, mais sans risque !) :
https://legacy.earlham.edu/~peters/writing/infinity.htm

Diagonalisation de l'hôtel de Hilbert, Jean-Pierre Messager :
https://framagit.org/jpython/math/-/tree/master/Hotel_Hilbert

Tout... sur LaTeX, Vincent Lozano, Framabook 2013 :
https://framabook.org/tout-sur-latex/

Très courte initiation à LaTeX, Maxime Chupin :
https://www.ceremade.dauphine.fr/~chupin/LaTeX/initLaTeX.pdf

Références

What Every Computer Scientist Should Know About Floating-Point Arithmetic, David Goldberg, Computing Surveys 1991 : https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html

Le *Frido*, tout le programme de l'agrégation de mathématiques, librement disponible : http://laurent.claessens-donadello.eu/frido.html

Solaris, Stanislas Lem, 1961. En français chez Denoël.

La voix du maître, Stanislas Lem, 1976. En français chez Denoël.

La cité des permutants, Greg Egan, 1984. En français au Livre de Poche.

Cours *Culture et histoire de l'informatique*, Jean-Pierre Messager : https://framagit.org/jpython/culthistinfo

Références

Logicomix, Apóstolos K. Doxiàdis, Christos Papadimitriou, Annie Di Donna (roman graphique), Vuibert 2018

Histoire d'algorithmes – Du caillou à la puce, Jean-Luc Chabert, Evelyne Barbin, et *al.*, Belin 2010

Histoire universelle des chiffres, Georges Ifrah, Robert Laffont 1994

Alan Turing – L'homme qui inventa l'informatique, David Leavitt, Dunod 2007

Science Made Stupid: How to Discomprehend the World Around Us, Tom Weller : http://www.chrispennello.com/tweller/

Web documentaire sur la cryptographie : https://www.ssi.gouv.fr/particulier/bonnes-pratiques/crypto-le-webdoc/cryptologie-art-ou-science-du-secret/