L'enseignement de l'informatique au secours de l'algèbre...et réciproquement !

Guillaume CONNAN[1]
IUT d'informatique et IREM de Nantes
Octobre 2012

Résumé : l'algèbre, qu'elle soit classique, générale ou linéaire, a pratiquement disparu de nos enseignements secondaires. Parallèlement, les probabilités et l'informatique occupent une place prépondérante. Cependant, l'informatique n'est que très timidement enseignée en tant que science. Un très rapide tour d'horizon devrait pouvoir mettre en lumière les liens étroits qui l'unissent à l'algèbre. Une approche plus rigoureuse de l'informatique apparaît alors comme un moyen d'offrir une nouvelle vie à notre chère algèbre si malmenée depuis quelque temps. De manière plus générale, l'introduction de mathématiques discrètes pourrait apporter une touche ludique, moderne et néanmoins rigoureuse à notre enseignement secondaire.

Voici quelques activités proposées à des étudiants de première année d'IUT d'informatique. Les étudiant(e)s viennent majoritairement de S. Le choix effectué par la plupart des IUT est de débuter par des mathématiques nouvelles pour la plupart : les mathématiques discrètes.

Il n'y a donc aucun prérequis d'analyse mais on a besoin de prendre un peu de recul sur les bases du calcul algébrique vu en collège. En effet, il s'agit d'étudier des ordinateurs ou, comme le disent les anglo-saxons, des *computers* c'est-à-dire des « calculateurs ».

1- Définition récursive d'un ensemble

La notion d'ensemble est primordiale en informatique, en particulier en début de formation pour introduire les types de variables qui sont en fait des ensembles.
Voici une définition :

Un ensemble d'éléments est :

- *soit l'ensemble vide (noté {})*
- *soit construit en « ajoutant » un élément à un ensemble. On notera $x \oplus E$ cette opération.*

Cela se traduit par exemple en CAML :

```
type 'a ens =
   | Vide
   | Ens of ('a * 'a ens) ;;
```

1 guillaume.connan@univ-nantes.fr

Ainsi {3,{2,{1,Vide}}} est l'ensemble contenant les entiers 1, 2 et 3.
Plusieurs problèmes algébriques apparaissent alors, le premier étant celui de la notion d'*égalité* de deux ensembles. On introduit alors l'*axiome d'extensionnalité :*

Deux ensembles A et B sont égaux si, et seulement si, ils contiennent les mêmes éléments.

Les étudiants sont dès le premier mois confrontés à une avalanche de symboles « = » signifiant des choses bien différentes : égalité des contenus de mémoire, affectation, définition de variables et ici une égalité qui va contredire celle utilisée par la machine :

```
# let e1 = Ens(3,Ens(2,Ens(1,Vide)));;
val e1 : int ens = Ens (3, Ens (2, Ens (1, Vide)))
# let e2 = Ens(2,Ens(3,Ens(1,Vide)));;
val e2 : int ens = Ens (2, Ens (3, Ens (1, Vide)))
# e1 = e2 ;;
-: bool = false
```

Ici, la contradiction n'est qu'apparente car on a utilisé une structure de couple ordonné (élément, ensemble). Il va donc falloir créer différentes fonctions : une qui teste l'appartenance d'un élément à un ensemble, une autre qui calcule l'intersection de deux ensembles, une troisième qui teste l'inclusion. Le test d'égalité se résumera alors à un test de double inclusion.

Cette généralisation de la notion d'égalité va se poursuivre dans bien d'autres domaines : la congruence modulo un entier en arithmétique, l'équivalence de propositions logiques, l'équivalence d'automates, l'isomorphisme de graphes etc.

L'autre problème est celui de la définition de l'opérateur $\oplus$ qui n'est pas commutatif et n'est même pas une loi de composition interne ce qui bouleverse les habitudes. Les élèves de lycée rencontrent bien la composition des fonctions mais de manière de plus en plus anecdotique.
Une égalité de ce type est donc perturbante :

$$x \oplus \{y\} = y \oplus \{x\}$$

en convenant de noter $x \oplus \{\} = \{x\}$.

Un autre problème est celui de la distinction entre l'appartenance et l'inclusion. On les introduit comme des fonctions :

est inclus *: P(E) * P(E) → {Vrai,Faux}* et appartien*t : E * P(E) → {Vrai, Faux}*

avec leurs opérateurs en notation infixe associés $\subseteq$ et $\in$.
On peut alors poser quelques questions « à la mode informatique ». Si *E* est un ensemble d'entier donc de type « à la CAML » `int ens`, quel est le type de *P(E)* ? Après avoir introduit le produit cartésien, que est le type de *P(E * E),* de *P(E) * P(E)* ?

On découvre enfin de nouvelles propriétés d'opérations (toujours introduites comme des fonctions) comme l'*idempotence* de l'union, de l'intersection, de la différence symétrique, la *distributivité* mutuelle de l'intersection et de la réunion, les lois de De Morgan ou un « moins » devant une

parenthèse transforme l'intersection en union, la notion d'*élément neutre,* etc.

Enfin la notion de fonction n'est plus liée à l'analyse mais devient un objet algébrique et de programmation : algèbre et informatique tissent leurs premiers liens étroits.

Voici en résumé un premier chapitre qui permet de généraliser les calculs faits en collège et au lycée, de prendre du recul avec des règles qui semblaient immuables parce qu'on n'avait travaillé que dans des types retreints d'ensembles. Les yeux s'ouvrent donc...

2- Logique des propositions

Le chapitre suivant permet de généraliser encore plus la notion de calcul. Certains en ont déjà découvert un aspect avec l'étude du calcul booléen au lycée, en section ST2I et en S-SI mais avec leurs professeurs d'électronique et non pas en mathématique.

La première notion algébrique importante abordée est celle de formule bien formée. Disposant d'un *alphabet* constitué de *propositions atomiques* (représentées par la suite par des minuscules), de *constantes* logiques $\top$ et $\bot$, de *connecteurs* $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$ et $\neg$, de parenthèses « (» et «) » on définit l'ensemble des formules bien formées comme étant le plus petit ensemble F tel que :

- toute proposition atomique est un élément de F ;
- $\top$ et $\bot$ sont des éléments de F ;
- si $p \in F$, alors $(\neg p) \in F$;
- si $(p,q) \in F^2$, alors $(p \wedge q)$, $(p \vee q)$, $(p \rightarrow q)$, $(p \leftrightarrow q)$ sont des éléments de F ;
- il n'y a pas d'autre expression bien formée que celles décrites précédemment.

Voici les étudiants confrontés au calcul non plus sur des nombres mais sur des propositions logiques : les voici capables de mécaniser une pensée simple. D'un point de vue plus pragmatique, on aborde de manière plus formelle qu'au collège des règles de calcul qui généralisent celles sur les nombres en disposant cette fois d'une présentation concise mais suffisante pour que le professeur ne soit plus celui qui dira si « on a le droit » ou pas d'effectuer tel calcul.
C'est aussi l'occasion de reparler de règles de priorité :

- $\neg$ a priorité sur les autres opérateurs ;
- $\vee$ et $\wedge$ sont prioritaires sur $\rightarrow$ et $\leftrightarrow$.

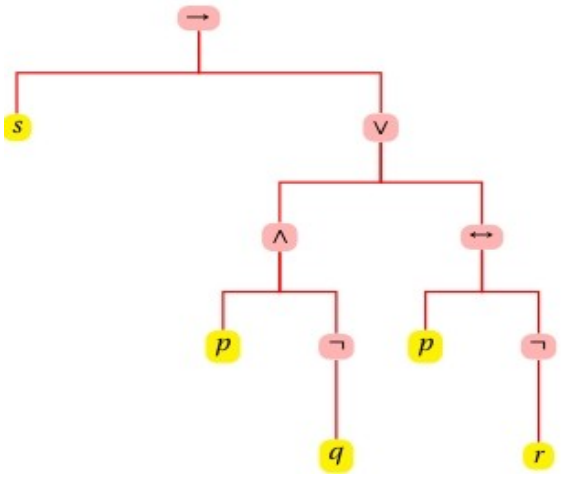
Cependant des expressions comme $p \vee q \wedge r$ demeurent ambiguës.
Le contexte informatique permet également d'introduire des arbres syntaxiques pour représenter des opérations.
Définissons tout d'abord un type de variables correspondant à nos définitions :

```
type atom = char  ;;
type formule =
      |Faux
      |Atomic of atom
      |Non of formule
      |Et of (formule * formule)
      |Ou of (formule * formule)
      |Implique of (formule * formule)
      |Equiv of (formule * formule);;
```

On peut alors se demander quelle formule représente :

```
Implique(Atomic 's',Ou(Et(Atomic 'p',Non (Atomic 'q')),(Equiv(Atomic
'p',Non(Atomic 'r'))))
```

Dessinons l'arbre correspondant à cette formule :



Il s'agit donc de $s \rightarrow (p \wedge \neg q) \vee (p \leftrightarrow \neg r)$. L'utilisation de calculatrices sur lesquelles on tape les expressions comme on les écrit sur sa feuille induit depuis plusieurs années une perte de lien avec la construction arborescente qui est primordiale. Au collège on se demande « est-ce une somme ? Est-ce un produit ? ». Ici, les connecteurs sont plus nombreux et demandent plus de concentration. Plusieurs types d'étude de cette proposition s'offrent alors, certaines d'ordre sémantique, d'autres d'ordre syntaxiques. On peut regrouper les valeurs prises par la proposition sous différents environnements dans une *table de vérité* ce qui nécessite de décomposer la proposition en sous-formules donc de bien appréhender sa structure arborescente. Par exemple, dans la proposition précédente, les sous-formules sont p, q, r, s, $\neg q$, $\neg r$, $p \wedge \neg q$, $p \leftrightarrow \neg r$, $(p \wedge \neg q) \vee (p \leftrightarrow \neg r)$ et la proposition elle-même.

On définit ensuite une nouvelle égalité entre propositions, l'équivalence logique : deux formules sont logiquement équivalentes si, et seulement si, elles admettent les mêmes environnements les rendant vraies.

L'emploi de tables de vérité permet alors de *démontrer* un certain nombre de formules et de ne plus les accepter comme une loi mystérieuse qui ne trouve sa légitimité que parce qu'elle a été énoncée par le professeur :

Équivalence entre connecteurs	$p \rightarrow q \equiv \neg p \vee q$
	$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p) \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$
Double négation	$\neg \neg p \equiv p$
Lois de De Morgan	$\neg(p \wedge q) \equiv \neg p \vee \neg q$
	$\neg(p \vee q) \equiv \neg p \wedge \neg q$
Idempotence	$p \vee p \equiv p \wedge p \equiv p$
Commutativité	$p \wedge q \equiv q \wedge p$
	$p \vee q \equiv q \vee p$
Associativité	$p \wedge (q \wedge r) \equiv (p \wedge q) \wedge r \equiv p \wedge q \wedge r$
	$p \vee (q \vee r) \equiv (p \vee q) \vee r \equiv p \vee q \vee r$

Contradiction	$p \wedge \neg p \equiv \bot$
Tiers exclus	$p \vee \neg p \equiv \top$
Lois de domination	$p \vee \top \equiv \top \quad p \wedge \bot \equiv \bot$
Lois d'identité	$p \vee \bot \equiv p \qquad p \wedge \top \equiv p$
Distributivité	$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
	$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
Absorption	$p \vee (p \wedge q) \equiv p$
	$p \wedge (p \vee q) \equiv p$

On en arrive à la mise sous *forme normale.* Si on appelle littéral toute formule atomique ou sa négation, une formule est dite sous forme normale conjonctive si, et seulement si, elle est composée d'une conjonction de disjonctions de littéraux.

Une formule est dite sous forme normale disjonctive si, et seulement si, elle est composée d'une disjonction de conjonctions de littéraux.

C'est en fait le pendant des factorisations et développements mais avec plus de règles donc plus de chance de se perdre. Les étudiants apprécient alors un schéma directeur : ne faire apparaître que des conjonctions et des disjonctions grâce aux équivalences entre connecteurs, réduire les négations avec les lois de De Morgan, utiliser enfin les diverses distributivités, absorptions et commutativité pour conclure sachant qu'on peut démontrer qu'une telle décomposition équivalente existe.

Cependant, les étudiants sont grandement handicapés au départ par leur manque de pratique du calcul en général. Avec un peu de pratique, montrer que $((p \to q) \wedge (q \to r)) \to (p \to r)$ est une tautologie se fait sans trop de mal.

On peut alors introduire de nouveaux connecteurs binaires (il en existe 16 mais pas tous logiquement utiles) pour varier les plaisirs et les propriétés :

- l'opérateur $\downarrow$ est défini par $1 \downarrow 1 = 1 \downarrow 0 = 0 \downarrow 1 = 0$ et $0 \downarrow 0 = 1$: « NOR » ;
- le connecteur $\uparrow$ est défini par $1 \uparrow 0 = 0 \uparrow 1 = 0 \uparrow 0 = 1$ et $1 \uparrow 1 = 0$: « NAND »
- le connecteur $\oplus$ est défini par : $1 \oplus 1 = 0$, $1 \oplus 0 = 1$, $0 \oplus 1 = 1$ et $0 \oplus 0 = 0$: « XOR ».

On peut alors multiplier les égalités :

$\neg x \equiv x \uparrow x \equiv x \downarrow x$, $x \wedge y \equiv (x \uparrow y) \uparrow (x \uparrow y)$, $x \vee y \equiv (x \downarrow y) \downarrow (x \downarrow y)$, $x \oplus y \equiv (x \vee y) \wedge \neg (x \wedge y)$,...

Cela permet de montrer entre autre que tous les connecteurs de la logique des propositions peuvent s'exprimer à l'aide de NAND. Il suffit alors de parler de l'architecture de l'ordinateur et des portes logiques et de faire rêver les étudiants en leur disant qu'avec une grosse boîte de portes NAND, on peut fabriquer un ordinateur (enfin presque...).

L'étude peut alors se poursuivre par l'étude de la logique des prédicats qui va permettre de généraliser la « mise en équation de la pensée ».

Soit par exemple f une fonction totale de E dans F et A et B deux parties de E. On veut démontrer que $f(A \cup B) \subseteq f(A) \cup f(B)$:

$$
\begin{aligned}
(\forall y)(y \in f(A \cup B) &\to (\exists x)((x \in A \cup B) \wedge (f(x) = y)) \\
&\to (\exists x)(((x \in A) \vee (x \in B)) \wedge (f(x) = y)) \\
&\to (\exists x)(((x \in A) \wedge (f(x) = y)) \vee ((x \in B) \wedge (f(x) = y))) \\
&\to ((\exists x)((x \in A) \wedge (f(x) = y)) \vee (\exists x)((x \in B) \wedge (f(x) = y))) \\
&\to ((y \in f(A)) \vee (y \in f(B))) \\
&\to (y \in f(A) \cup f(B)) \\
) &\to f(A \cup B) \subseteq f(A) \cup f(B)
\end{aligned}
$$

Pour faciliter la lecture, nous avons adopté la convention suivante :

$$A \rightarrow B$$
$$\rightarrow C$$

signifie $(A \rightarrow B) \wedge (B \rightarrow C)$

Grâce à l'emploi de règles de CALCUL, nous avons pu effectuer une DÉMONSTRATION : le calcul n'est plus seulement lié au*x* nombres, à l'arithmétique, il permet aussi de raisonner. Or un ordinateur calcule : il peut donc, dans certains cas, raisonner lui aussi, grâce à l'algèbre...

3- Langages

Calculer sur un langage, voici une autre nouveauté qui pique la curiosité des étudiants. On est habitué à « dialoguer » avec un ordinateur via un clavier et un écran : comment faire ? Il est impossible d'aborder la théorie des langages et la compilation de manière sophistiquée. Cependant, la seule introduction en 1ère année suffit à voir une fois encore le calcul autrement. Une des grandes tâches de la machine est en effet de reconnaître des motifs: c'est la partie essentielle de la *compilation*, c'est-à-dire la traduction de programmes écrits dans un langage « humain » (Caml, C, Java, etc.) dans un autre, notamment le langage machine.

Un *alphabet* est un ensemble fini non vide. Les éléments sont appelés des *symboles* ou des *caractères*.

Une chaîne est une juxtaposition de symboles. On appelle cette juxtaposition la concaténation

Soit A un alphabet. Soit λ_A la chaîne vide (ne comportant aucun symbole).

L'ensemble A^* des chaînes construites avec l'alphabet A est défini par :

1. $\lambda_A \in A^*$;
2. si $a \in A$ et $c \in A^*$, alors la chaîne construite à partir de c en concaténant a à droite est une chaîne de A^* ;
3. μ est un élément de A^* seulement s'il peut être obtenu à partir de λ_A par application de l'étape 2 un nombre fini de fois.

La concaténation des chaînes μ et ν est notée $\mu\nu$.

C'est l'occasion de parler de structures algébriques : la concaténation est une loi de composition interne sur A^* car en concaténant deux éléments de A^* , on obtient encore un élément de A^*. Cette loi est également associative et possède un élément neutre λ_A. Cela confère à A^* la structure de monoïde libre, selon les standards informatiques (les mathématiciens parlent plutôt de magma associatif unitaire et les anglo-saxons de *lattice*). C'est l'occasion de débattre sur l'utilité de distinguer certaines structures pour en connaître d'avance les propriétés.

L'associativité nous permet d'utiliser la notation puissance, mais de manière inhabituelle car nous ne disposons pas de la commutativité :

$$a^n = aa...a \ ; \ a^n b^k = aa...abb...b \ ; \ (ab)^n = abababab...ab$$

On définit des préfixes, des suffixes, des facteurs : par exemple *bac* est un facteur de *aaabacaaa.*

On définit une division partielle de chaînes : pour toutes chaînes μ et ν de A^* , le quotient à gauche de ν par μ est noté $\mu^{-1}\nu$ et est défini par $\mu^{-1}\nu = \xi$ si $\nu = \mu\xi$ et n'a pas de sens si μ n'est pas un préfixe de ν . On définit de même le quotient à droite. Si μ = *aababc*, le quotient à gauche de μ par *aab* est *abc*.

Les opérations affluent mais nous n'avons toujours pas parlé de langage : c'est simplement une partie de A^* . L'étude des langages va nous donner l'occasion d'introduire de nouvelles lois... Cette fois, les lois ne portent pas sur les chaînes mais sur les langages : on peut en faire la somme (en fait l'union), le produit (en fait la concaténation) et une nouvelle appelée *étoile de Kleene* : l'étoile de Kleene du langage L est l'union sur tous les entiers k des L^k , L^k étant l'ensemble des chaînes de longueurs k construites avec l' »alphabet » L... C'est ici que ça commence à chauffer ! Il faut réaliser que pour la moitié des étudiants, après deux mois de cours, la notation x^2 est irrémédiablement associée au carré d'un nombre or ils ont été confrontés au carré d'une relation au sens de la composition, au carré d'un ensemble au sens du produit cartésien, au carré d'un symbole au sens de la concaténation et au carré d'un langage au sens de la concaténation des langages : des années à (mal...) manipuler des nombres ont profondément lié le calcul aux nombres.
À peine remis de leurs émotions suite à cet assaut de notations familières mais utilisées dans des contextes totalement nouveaux, voici une nouvelle définition et des conventions de notations :

Langage rationnel (ou régulier)
Soit L un langage sur l'alphabet A. Il est dit rationnel s'il peut être obtenu récursivement uniquement à l'aide des opérations :

- *réunion (somme),*
- *concaténation (produit),*
- *étoile de Kleene*

en partant :

- *du langage vide ,*
- *des langages du type a avec* $a \in A \cup \lambda_A$.

Considérons l'alphabet A = *b, o, x* et L l'ensemble des chaînes finissant par *x.*
Par exemple, *box, bobox, xxbobbxoxbbxxoxxxx, x* appartiennent à ce langage.
On peut le décrire ainsi :

$$L = (\{b\} \cup \{o\} \cup \{x\})^* \cdot \{x\}$$

C'est donc un langage rationnel car obtenu à partir de singletons et d'opérations rationnelles. Par abus de notation, on se contentera souvent de l'écrire ainsi :

$$L = (b + o + x)^* x \text{ ou } \{b,o,x\}^* x \text{ ou } (b^* o^* x^*)^* x$$

Et tout ceci n'est qu'un début : viennent ensuite les notions d'expressions rationnelles, d'automates à états finis avec leurs nouvelles règles algébriques de calcul et de résolutions d'équations...

4- Arithmétique pour l'informatique

L'arithmétique est présente à divers niveaux en informatique. L'étude est axée sur une initiation à la cryptographie : notions de groupe, anneau, corps pour aborder l'inversion, la résolution d'équations dans $\mathbb{Z}/n\mathbb{Z}$, les permutations pour le chiffrement par blocs.

a- Structures mères

Sur la plupart des langages, les opérations sur les nombres sont des lois internes. Par exemple, sur

CAML :

```
# 1 + 1.5 ;;
^^^
"Error: This expression has type float but an expression was expected of
type int"

# 1. +. 1.5 ;;
- : float = 2.5
# 1 + 2 ;;
-: int = 3
```

Sur CAML, il existe une loi + pour additionner les entiers et une loi +. pour additionner les flottants. L'étude des chapitres précédents a permis d'avoir de nombreux exemples de magmas, monoïdes, groupes. On généralise cette fameuse écriture x^k qui a posé tant de problèmes auparavant.

b- Calcul modulaire

L'arithmétique, c'est aussi l'occasion de travailler pleinement sur les classes d'équivalences, avant de les revoir dans d'autres contextes informatiques comme lors de l'étude des graphes. Parler de classes d'équivalence permet de définir un nouveau type de variables si on y réfléchit en informaticien, ce qui permet d'éviter les confusions entre un entier et sa classe modulo *n.* Par exemple, on peut définir le calcul modulaire sur CAML sous forme d'un enregistrement :

```
type classe  =
  {representant : int ; modulo : int};;
```

Ensuite, on crée un opérateur qui simplifiera notre saisie :

```
let (%) a n = {representant = (a mod n); modulo = n};;
```

Par exemple, on peut définir la classe de 12 modulo 7 :

```
# 12%7;;
-: classe = {representant = 5; modulo = 7}
```

Ce n'est pas un entier, c'est une classe. Pour additionner les classes par exemple, on est obligé de définir une nouvelle opération :

```
exception  Classes_incompatibles;;

let (+%) a b =
  {modulo =
      if a.modulo = b.modulo
      then a.modulo
      else raise Classes_incompatibles ;
   representant =
      ( (a.representant) + (b.representant)) mod (a.modulo)
  };;
```

Par exemple, la somme de la classe de 12 modulo 7 et de la classe de 15 modulo 7 s'obtient par :

```
# (12%7) +% (15%7) ;;
-: classe = {representant = 6; modulo = 7}
```

En fait on peut créer une fonction d'ordre supérieur prenant l'opérateur en argument, d'une part pour économiser des copier-coller, d'autre part et surtout, pour monter d'un cran dans la modélisation algébrique, ce que permet un langage fonctionnel :

```
let loi_induite loi =
  fun a  b ->
  {modulo =
      if a.modulo= b.modulo
      then a.modulo
      else raise Classes_incompatibles ;
   representant =
      (loi (a.representant) (b.representant)) mod (a.modulo)
  };;

let ( +% ) = loi_induite  ( + ) ;;
let ( *% ) = loi_induite  ( * ) ;;
```

Alors par exemple :

```
# (9%5) *% (7%5);;
-: classe = {representant = 3; modulo = 5}
```

On peut alors dresser la table de multiplication de $\mathbb{Z}/7\mathbb{Z}$ et $\mathbb{Z}/8\mathbb{Z}$ à la main ou avec la machine :

```
 let table_modulaire n =
  for i = 1 to (n-1) do
    for j = 1 to (n-1) do
      print_int ((j%n *% (i%n)).representant);
      print_string " ";
    done;
    print_newline();
  done;;
```

et on obtient :

```
 # table_modulaire 8;;
1 2 3 4 5 6 7
2 4 6 0 2 4 6
3 6 1 4 7 2 5
4 0 4 0 4 0 4
5 2 7 4 1 6 3
6 4 2 0 6 4 2
7 6 5 4 3 2 1
-: unit = ()
```

```
# table_modulaire 7;;
  1 2 3 4 5 6
  2 4 6 1 3 5
  3 6 2 5 1 4
  4 1 5 2 6 3
  5 3 1 6 4 2
  6 5 4 3 2 1
- : unit = ()
```

ce qui permet de discuter sur l'existence de l'inverse d'un entier modulo *n.* On peut ensuite le déterminer par une recherche exhaustive (on multiplie jusqu'à obtenir $1_{\mathbb{Z}/n\mathbb{Z}}$) ou en utilisant les coefficients de Bézout de l'algorithme d'Euclide étendu. Cela ouvre ensuite la porte à la notion

d'anneau, de corps, du théorème d'Euler...

c- Permutations, cryptosystèmes par blocs.

Si nous allons plus avant dans le domaine de la cryptographie, il nous faut explorer rapidement l'ensemble des permutations des entiers : montrer que ($\mathfrak{S}_n$, o) est un groupe et étudier le lien avec les permutations de bits, à savoir les fonction de $\{0,1\}^n \to \{0,1\}^n$ définies par :

$$b_1 b_2 \dots b_n \to b_{\pi(1)} b_{\pi(2)} \dots b_{\pi(n)}$$

avec $\pi \in \mathfrak{S}_n$

On peut alors étudier le cryptosystème CBC (*Cipher Block Chaining)* Ce mode a été inventé par IBM en 1976. Son étude permet d'avoir un exemple simple de cryptosystème par blocs dont l'AES est le représentant phare mais trop compliqué à étudier.
On considère une chaîne de bits que l'on découpe en blocs de longueur fixe, par exemple 4 : on rajoute alors éventuellement des zéros en bout de chaîne.
Considérons le message clair *m* = 1001001111011010 et découpons-le :

$$m = 1001\ 0011\ 1101\ 1010 = b_1 b_2 b_3 b_4$$

On choisit une clé de chiffrement qui sera une permutation de $\mathfrak{S}_4$

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 4 & 2 \end{pmatrix}$$

Appelons E_π la permutation de bits associée et $c = c_1 c_2 c_3 c_4$ le message chiffré. L'idée est d'obtenir des blocs chaînés : chaque bloc en clair est « xoré » (avec l'opérateur *ou exclusif*) avec le bloc crypté précédent avant d'être crypté lui-même par permutation des bits. Ainsi, deux mêmes blocs pourront être cryptés différemment ce qui brouillera d'autant mieux les pistes.
Pour le premier bloc, on utilise un vecteur d'initialisation public. On a donc :

$$c_i = E_\pi(m_i \oplus c_{i-1}),\ c_0 = v_0$$

Pourquoi le ou exclusif ? C'est en fait l'addition modulaire dans le corps $\mathbb{Z}/2\mathbb{Z}$: le lien entre mathématiques abstraites et électronique concrète se resserre...
Chiffrons *m* avec $v_0 = 1010$ *:*

$$c_1 = E_\pi(m_1 \oplus v_0) = E_\pi(1001 \oplus 1010) = E_\pi(0011) = 1010$$
$$c_2 = E_\pi(m_2 \oplus c_1) = E_\pi(0011 \oplus 1010) = E_\pi(1001) = 0110$$
$$c_3 = E_\pi(m_3 \oplus c_2) = E_\pi(1101 \oplus 0110) = E_\pi(1011) = 1110$$
$$c_4 = E_\pi(m_4 \oplus c_3) = E_\pi(1010 \oplus 1110) = E_\pi(0100) = 0001$$

Ainsi *c* = 1010 0110 1110 0001

Comment déchiffrer ? Il s'agit de résoudre les équations $c_i = E_\pi(m_i \oplus c_{i-1})$ d'inconnue m_i dans $\{0,1\}^4$. Voilà qui demande de bien réfléchir aux étapes de la résolution algébrique : les « je passe de l'autre côté » fonctionnent ici bizarrement...
Il faut donc commencer par composer par la permutation inverse de E_π puis « xorer » par c_{i-1} car pour tout *x* dans {0,1}, $x \oplus x = 0$:

$$m_i = E_{\pi^{-1}}(c_i) \oplus c_{i-1}$$

Voici une **activité ludique pour des collégiens** et qui peut leur permettre de prendre du recul sur les résolutions d'équations : comment résoudre des équations dans des ensembles « bizarres » mais abordables : le ou exclusif et les permutations peuvent être introduits de manière simple sans rentrer

dans un trop grand formalisme. On peut de plus leur fournir une table des codes ASCII des principaux caractères en base 10, leur demander de les convertir en base 2 puis d'appliquer le cryptosystème précédent. Par exemple, « maman » sera chiffré en « eQ*yD » : c'est ludique et on fait des maths contemporaines...

5- Polynômes pour l'informatique

Les polynômes sont utilisés dans de nombreux domaines en informatique : modélisation des grands entiers, codes détecteurs et correcteurs d'erreurs, opérations sur les chaînes de bits...

a- Multiplication des grands entiers

Une fois vu un algorithme naïf de calcul du produit de deux polynômes, on peut s'intéresser, naïvement aussi, au problème du calcul avec des grands nombres.
Par exemple, sur des machines 32 bits, certains calculs posent des problèmes. En effet, les entiers sont codés sur 31 bits plus un bit pour le signe : on ne peut donc pas dépasser 2^{31} -1.
Prenons pour simplifier 123 456 789 $\times$ 987 654 321. On écrit chacun des deux nombres en base 1000 et on « remplace » les 1000 par des « X ». On est donc amené à effectuer le produit des deux polynômes $(123X^2+456X+789)$ et $(987X^2+654X+321)$. On obtient :

$$121401X^4+530514X^3+1116450X^2+662382X+253269$$

Il reste à remplacer les coefficients supérieurs à 1000 par des polynômes selon les puissances de 1000 :

$$(121X+401)X^4+(530X+514)X^3+(X^2+116X+450)X^2+(662X+382)X+253X+269$$

On développe et on ordonne :

$$121X^5+932X^4+631X^3+112X^2+635X+269$$

Le résultat est donc 121 932 631 112 635 269 : voici une activité qui peut également être menée au collège pour illustrer le calcul algébrique, la numération et le fait qu'une machine est limitée en mémoire et a donc besoin d'outils mathématiques pour aller au-delà de ses capacités basiques.

b- Schéma de Horner

Pour le calcul effectif du quotient et du reste dans la division euclidienne de deux polynômes , on peut appliquer l'algorithme classique ou bien utiliser le schéma de Horner très utilisé en algorithmique. On peut ouvrir une parenthèse historico-anecdotique à ce sujet : cette méthode porte le nom du britannique William George Horner (1786 - 1837) mais en fait elle fut publiée presque 10 ans auparavant par un horloger londo nien, Theophilius Holdred et simultanément par l'italien Paolo Ruffini (1765 - 1822) mais fut déjà utilisée par Newton 150 ans auparavant et par le chinois Zhu Shije cinq siècles plus tôt (vers 1300) et avant lui par le Persan Sharaf Al Din Al Muzaffar Ibn Muham Mad Ibn Al Muzaffar Al Tusi vers (1100) et avant lui par le Chinois Liu Hui (vers 200) révisant un des résultats présent dans Les Neuf Chapitres sur l'art mathématique publié avant la naissance de JC...
Par exemple, $P(2)=4+5\times2+6\times a^2+7\times2^3=4+2(5+2(6+2(7+2\times0)))$
L'intérêt est de réduire grandement le nombre de multiplications effectuées : elles sont de l'ordre du degré du polynôme au lieu d'être de l'ordre de son carré.

Sur machine, on peut même montrer l'intérêt de représenter un polynôme par la liste de ses coefficients et voir que l'on peut aussi bien calculer avec [1; 2 ; 3] qu'avec $1+2X+3X^2$:

```
let rec horner poly t =
  match poly with
  |[]          -> 0
  |tete::queue -> tete + t * (horner queue t);;
```

Alors pour calculer par exemple P(2) avec P = $1+2X+3X^2$:

```
# horner [1;2;3] 2;;
- : int = 17
```

Il faut cependant noter que Ruffini l'avait employée en fait comme un moyen de calculer rapidement le quotient et le reste d'un polynôme par (X – a) puis la dérivée *n*-eme d'un polynôme en un point. C'est ce qui peut être vu en travaux dirigés. En effet, écrivons la division euclidienne d'un polynôme $P=\sum_{i=0}^{i=n} a_i X^i$ par (X-t) :

$$P=(X-t)(b_n X^{n-1}+b_{n-1}X^{n-2}+\ldots+b_1)+b_0$$

En égalant les coefficients de même rang on est amené à résoudre un système linéaire d'inconnues les b_i:

$$\begin{cases} a_n=b_n \\ a_{n-1}=b_{n-1}-t\,b_n \\ \cdot \\ \cdot \\ \cdot \\ a_0=b_0-t\,b_1 \end{cases} \qquad \begin{cases} b_n=a_n \\ b_{n-1}=a_{n-1}+t\,a_n \\ \cdot \\ \cdot \\ \cdot \\ b_0=a_0+t\left(a_1+t\left(a_2+\ldots+t\left(a_{n-1}+ta_n\right)\right)\right) \end{cases}$$

On s'aperçoit donc que le reste est le calcul de P(*t*) par le schéma de Horner.
Un peu d'algèbre permet donc d'améliorer la performance du calcul du reste même si cela ne concerne ici que des diviseurs de degré 1...

c- Code détecteur d'erreurs

Les informations transitant à travers un réseau sont soumises à des influences pouvant gêner leur transmission : le message reçu n'est pas toujours le message émis.
Certains protocoles (ports USB, Bluetooth, PPP, Ethernet, Wifi,...) utilisent un *code de redondance cyclique* (CRC : cyclic redundancy check).
Supposons que l'on veuille transmettre une chaîne de bits $b_n b_{n-1}\ldots b_1 b_0$ qui sera associée à un polynôme de $\mathbb{Z}/2\mathbb{Z}$[X] :

$$C=b_n X^n+b_{n-1}X^{n-1}+\ldots+b_1 X+b_0$$

Émetteur et récepteur disposent d'un même polynôme de contrôle G de degré $p < n$. La chaîne

émise sera alors $C_e = X^p C + (X^p C \bmod G)$. La chaîne reçue ne sera pas forcément la même suite aux erreurs de transmission. On notera E l'erreur : $C_r = C_e + E$.
Le récepteur calcule alors le reste de C_r dans la division par G : compte tenu des propriétés de la somme des polynômes de $\mathbb{Z}/2\,\mathbb{Z}$[X], le résultat obtenu sera E modulo G.
Voyons un exemple simple : on veut transmettre 10101101 et on dispose de G = 10111. Pour déterminer la chaîne émise, il faut effectuer la division de X^p C par G. On l'effectue soit directement en base 2, soit en passant par les polynômes de $\mathbb{Z}/2\mathbb{Z}$[X].
On obtient $X^{11}+X^9+X^7+X^6+X^4=(X^4+X^2+X+1)(X^7+X^4+X)+X^3+X^2+X$. La chaîne émise est donc C_e = 101 011 011 110. Si C_r = 101 011 001 110, on a E = 1 000 et l'erreur est détectée. Si E est un multiple de G, alors elle ne sera pas détectée. Il faut donc choisir un bon générateur qui puisse détecter un maximum d'erreurs. Il existe certaines règles dans le choix de G qui permettent de détecter certaines erreurs et dont la justification n'est pas toujours accessible en Bac+2.
Pour information, voici quelques polynômes générateurs « de la vraie vie » :

- CRC-16-IBM pour les ports USB : $X^{16}+X^{12}+X^2+1$;
- CRC-16-CCITT pour Bluetooth, PPP : $X^{16}+X^{12}+X^5+1$;
- CRC-32-IEEE802.3 pour Ethernet, WiFi :
  $X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^8+X^7+X^5+X^4+X^2+X+1$.

En fait, ces polynômes corrigent les erreurs les plus probables. En cas de détection d'erreur, on demande alors simplement une retransmission.
Dans des situations plus délicates, on demandera au code d'être non seulement détecteur mais aussi correcteur. C'est le cas par exemple des CD-Rom. Certains de ces codes peuvent être abordés à l'IUT en explorant un peu plus les polynômes et les matrices à coefficients dans $\mathbb{Z}/2\mathbb{Z}$.

8- Enseignements ?

La liste des liens entre algèbre et informatique à un niveau élémentaire est encore longue: graphes, résolutions de systèmes, géométrie euclidienne, calculs en probabilités discrètes,...
Les quelques exemples évoqués suffisent à laisser penser que l'enseignement de l'informatique ne peut se faire sans des bases solides en calcul algébrique. Or l'informatique a été trop longtemps présentée dans l'enseignement secondaire comme un outil pour avoir de jolies figures ou des tableaux de gestion et pour nos élèves, elle n'apparaît pas vraiment comme une science à part entière. Certains collègues de mathématiques sont plus ou moins contraints de l'évoquer dans leurs cours mais surtout comme un outil d'illustration ou pour *éviter de faire des calculs* : il serait plus rassurant de constater que l'étude de l'informatique au contraire *nécessite de faire des calculs.*

D'un autre côté, la formation des étudiants en informatique ne pourra être que bancale s'ils ne disposent pas de bases solides en calcul. Les orientations des derniers programmes ne semblent pourtant pas aller dans ce sens. Cependant, des ouvertures pourraient exister, en MPS par exemple, où mathématiques et informatique sont trop souvent employées comme des outils secondaires mais pourraient au contraire devenir des objets d'étude à part entière et s'enrichir mutuellement.

On peut enfin rêver à l'introduction de mathématiques discrètes dans l'enseignement secondaire : cela permettrait de sortir des mathématiques du XVIIe pour rentrer dans celles du XXe avec peu

d'outils et sans sombrer dans le catalogue de théorèmes et recettes non démontrées comme cela est trop le cas avec le nouveau programmes de probabilités et de statistiques. Au contraire, avec très peu d'outils, mais de vrais outils, modernes et qui piquent la curiosité, on peut travailler différemment le calcul et découvrir des utilisations actuelles des mathématiques.

Citons pour finir un très grand informaticien parlant des liens entre mathématiques et informatique :

By now we all know that programming is as hard or as easy as proving, and that if programming a procedure corresponds to proving a theorem, designing a digital system corresponds to building a mathematical theory. The tasks are isomorphic. We also know that, while from an operational point of view a program can be nothing but an abstract symbol manipulator, the designer had better regard the program as a sophisticated formula.[…] Computing and Computing Science unavoidably emerge as an exercise in formal mathematics or, if you wish an acronym, as an exercise in VLSAL (Very Large Scale Apllication of Logic).[...]And from the mathematical community I have learned not to expect too much support either, as informality is the hallmark of the Mathematical Guild, whose members –like poor programmers-- derive their intellectual excitement from not quite knowing what they are doing.[...]In the next fifty years, Mathematics will emerge as The Art and Science of Effective Formal Reasoning.

Edsger Dijkstra (1930 – 2002), prix Turing 1972, lors d'une allocution du 14 juin 1989 dont le texte manuscrit est disponible ici : `http://www.cs.utexas.edu/~EWD/ewd10xx/EWD1051.PDF`