Le treillis de Goldbach à 4 lettres et 16 règles : la récurrence de la note `ia251.pdf` démontrée en Lean

Denise Vella-Chemla pilotant l'ia claude, 7 novembre 2026

Résumé

Le treillis de Goldbach code chaque décomposition $n = p + q$ d'un nombre pair par l'une de quatre lettres, comme les quatre bases de l'ADN, et se construit colonne après colonne par seize règles de réécriture. La note 251 en donnait un programme Lean qui *calcule* les colonnes. On démontre ici, et l'on a vérifié en Lean, que ce programme engendre exactement la colonne définie directement par la primalité de p et de $n - p$, pour tout n pair ≥ 6. Ce théorème ne dit rien de l'existence d'une lettre a dans chaque colonne, c'est-à-dire de la conjecture de Goldbach elle-même : il établit que la formalisation est fidèle au treillis.

1. Le treillis et ses quatre lettres

Pour n pair, $n \geq 6$, on considère les décompositions $n = p + q$ avec p impair, $3 \leq p \leq n/2$ et $q = n - p$. La lettre de (p, q) est

lettre	p	q
`a`	premier	premier
`b`	composé	premier
`c`	premier	composé
`d`	composé	composé

On note $L(p, q)$ cette lettre. Une colonne contient au moins un `a` si et seulement si n est somme de deux nombres premiers.

Les seize règles de réécriture sont la table d'une unique loi $X \cdot Y$ sur les quatre lettres :

$X \backslash Y$	`a`	`b`	`c`	`d`
`a`	`a`	`b`	`a`	`b`
`b`	`a`	`b`	`a`	`b`
`c`	`c`	`d`	`c`	`d`
`d`	`c`	`d`	`c`	`d`

Convention utilisée dans toute la note : la lettre $X \cdot Y$ prend le statut de p (premier ou composé) sur l'opérande de droite Y et le statut de q sur l'opérande de gauche X. Autrement dit, pour tous entiers p, q, r, s,

$$\text{Lettre}(p, q) \cdot \text{Lettre}(r, s) = \text{Lettre}(r, q). \tag{1}$$

Si l'on écrit les couples dans l'ordre (q, p), la loi s'écrit $(x_1, y_1) \cdot (x_2, y_2) = (x_1, y_2)$: c'est une bande rectangulaire (associative, idempotente, non commutative). Les ambiguïtés rencontrées dans certaines notes viennent de l'oubli de préciser l'ordre des coordonnées.

2. La colonne définie directement

Pour n pair ≥ 6, soit $m(n) = \lfloor (n-2)/4 \rfloor$. Les entiers impairs de $[3, n/2]$ sont exactement les $p_i = 3 + 2i$ pour $0 \leq i < m(n)$. La colonne de n est le mot

$$\mathrm{col}(n) = \mathrm{Lettre}(p_0, n - p_0)\ \mathrm{Lettre}(p_1, n - p_1) \cdots \mathrm{Lettre}(p_{m(n)-1}, n - p_{m(n)-1}).$$

Par exemple col(6) = `a` (car $6 = 3 + 3$) et col(98) = `cccbccbcadcbbcaddabccbcd`.

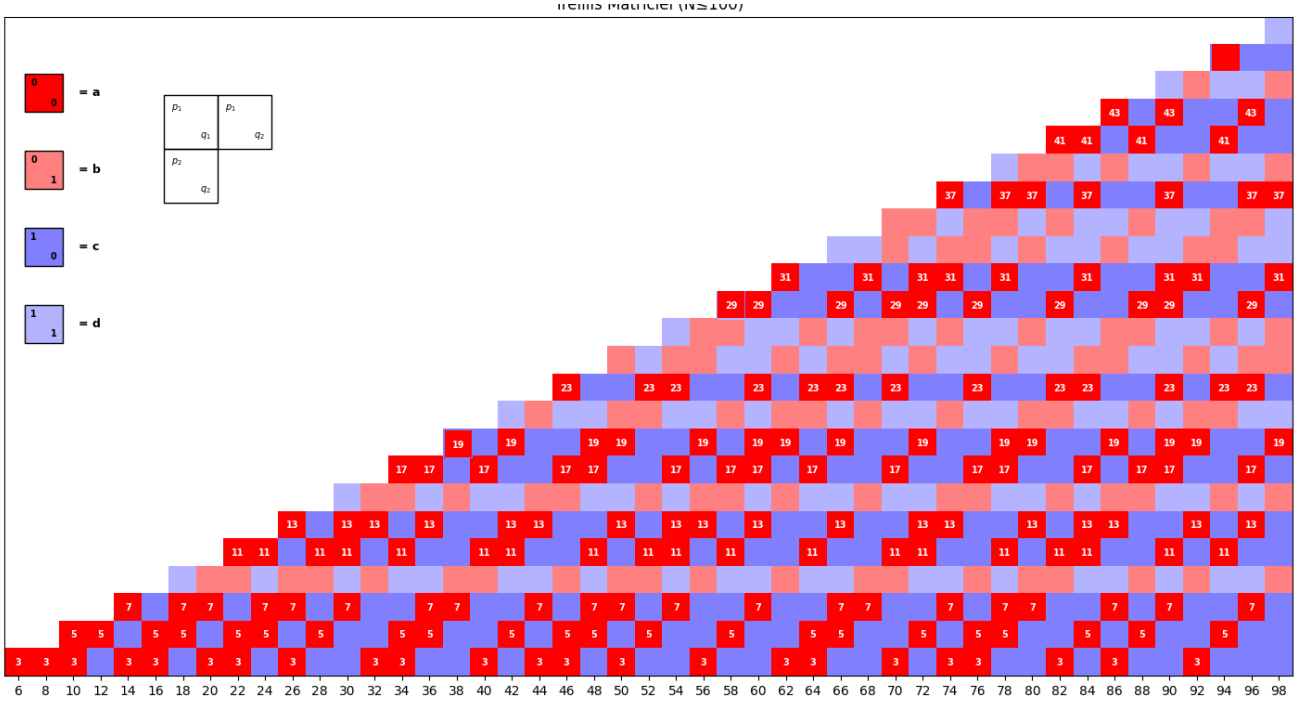


FIGURE 1 : le treillis pour n pair, $6 \leq n \leq 102$: abscisse n, ordonnée p (petit sommant impair). Les cases de couleur rouge opaque sont les lettres `a` (p et $n - p$ premiers), celles de couleur rouge clair les `b`, les cases de couleur bleu opaque les `c`, celles de couleur bleu clair les `d`. Une colonne verticale contient au moins une case rouge opaque si et seulement si n est somme de deux nombres premiers.

3. La récurrence de la note 251

La note 251 construit col$(n+2)$ à partir de col(n) sans consulter à nouveau la primalité, sauf pour deux lettres. Trois ingrédients :

1. *les transitions.* Si X_i désigne la lettre de rang i de col(n), on forme les produits $X_i \cdot X_{i+1}$ pour $0 \leq i \leq m(n) - 2$.
2. *la tête.* On place devant eux la lettre `a` si $(n+2) - 3$ est premier, la lettre c sinon (c'est la lettre de $(3, n-1)$, et 3 est premier).
3. *le palier.* Si $m(n+2) = m(n) + 1$, on ajoute une lettre finale : a si la dernière lettre de col(n) est a ou b, et d si c'est c ou d.

On note $T_n(w)$ le mot ainsi obtenu à partir d'un mot w de la colonne n.

4. Énoncé et démonstration

Théorème 1. *Pour tout n pair, $n \geq 6$, on a $T_n(\mathrm{col}(n)) = \mathrm{col}(n+2)$. En conséquence, le mot engendré par la récurrence à partir de a pour $n = 6$ est, à chaque étape, égal à* $\mathrm{col}(n)$.

Lemme 2 (décalage)**.** *Pour $0 \leq i \leq m(n) - 2$, la lettre de rang $i+1$ de* $\mathrm{col}(n+2)$ *est $X_i \cdot X_{i+1}$.*

Démonstration. On a $X_i = \mathrm{Lettre}(p_i, n - p_i)$ et $X_{i+1} = \mathrm{Lettre}(p_i + 2, n - p_i - 2)$. Par (1), $X_i \cdot X_{i+1} = \mathrm{Lettre}(p_i + 2, n - p_i)$. Or $p_i + 2 = p_{i+1}$ et $(n+2) - p_{i+1} = n - p_i$: c'est la lettre de rang $i+1$ de $\mathrm{col}(n+2)$. □

Lemme 3 (tête)**.** *La lettre de rang 0 de* $\mathrm{col}(n+2)$ *est a si $n-1$ est premier, c sinon.*

Démonstration. C'est $\mathrm{Lettre}(3, n-1)$, avec 3 premier. □

Lemme 4 (longueurs et palier)**.** *Si $n \equiv 2 \pmod 4$, alors $m(n+2) = m(n)$. Si $n \equiv 0 \pmod 4$, alors $m(n+2) = m(n) + 1$, et la lettre de rang $m(n)$ de* $\mathrm{col}(n+2)$ *est celle de $(n/2+1,\ n/2+1)$; elle est obtenue à partir de la dernière lettre de* $\mathrm{col}(n)$ *par la règle du palier.*

Démonstration. On a $m(n) = \lfloor (n-2)/4 \rfloor$, d'où les deux valeurs de $m(n+2)$. Si $n \equiv 0 \pmod 4$, le dernier p de $\mathrm{col}(n)$ est $n/2 - 1$ (impair), avec $q = n/2 + 1$; le nouveau dernier est $p = q = n/2 + 1$ pour la colonne $n+2$. Sa lettre est a si $n/2+1$ est premier et d sinon. Or $n/2+1$ est le q de la dernière lettre de $\mathrm{col}(n)$, laquelle est a ou b si q est premier, c ou d sinon : c'est exactement la règle du palier. □

Démonstration du théorème 1. Les mots $T_n(\mathrm{col}(n))$ et $\mathrm{col}(n+2)$ ont les mêmes lettres de rang 0 (tête), de rang $1, \ldots, m(n)-1$ (lemme du décalage), et, au palier, de rang $m(n)$ (lemme précédent). Les longueurs coïncident par le lemme des longueurs. La seconde assertion se déduit de la première par récurrence sur le nombre d'étapes, avec $\mathrm{col}(6) = a$. □

5. La formalisation en Lean

Le fichier `Goldbach251Preuve.lean` (https://denisevellachemla.eu/treillis-conj-Goldbach-preuve-ia251.pdf) reprend mot pour mot les définitions de `GardeMain.lean`, qui est le programme de la note 251 (https://denisevellachemla.eu/ia251.pdf). Parmi les dix fichiers Lean retrouvés à la date de cette note, c'est celui dont le code coïncide avec celui du PDF ; les seules différences sont les numéros de page et les accolades des chaînes `s!"..."`, perdues à l'export. Le fichier de preuve n'utilise pas Mathlib : il commence par `import Init`.

On y ajoute la colonne directe :

```
def letterOf (p q : Nat) : Letter := letterB (isPrime p) (isPrime q)

def mkCol (n : Nat) : Nat -> Nat -> Word
  | _, 0 => []
  | i, m + 1 => letterOf (3 + 2*i) (n - (3 + 2*i)) :: mkCol n (i+1) m

def colOf (n : Nat) : Word := mkCol n 0 (targetLength n)
```

où `mkCol n i m` liste les lettres des rangs $i, \ldots, i+m-1$. Les étapes de la preuve sont les suivantes.

rule_letterB, lastStep_letterB	la loi (1) et la règle du palier, par distinction des cas sur quatre booléens
head_eq	la règle de tête
mkCol_snoc, getLast_mkCol	dernière lettre de `mkCol` (palier)
ng_mkCol	le lemme de décalage : `nextGeneration` fait passer de la colonne n à la colonne $n+2$
trans_palier, trans_stable	les deux branches de `transitionWithContext`
step_mkCol, step_col	le théorème 1, avec $n = 4u+8$ au palier
seq_correct, recurrence_eq_colonne	la récurrence sur le nombre d'étapes

Le fichier se compile sans erreur. La commande `#print axioms` sur le théorème final répond `propext`, `Classical.choice`, `Quot.sound` : ce sont les trois axiomes standard de Lean, et `sorryAx` n'y figure pas, donc la preuve ne contient aucun trou. Les commandes `#eval` de contrôle affichent le mot de $n = 98$, identique à col(98).

Vérification indépendante. Un portage du programme en Python, comparé à la colonne définie par la primalité, ne donne aucun écart pour tous les n pairs de 6 à 20 000. Cette vérification numérique précède et recoupe la preuve, mais ne la remplace pas.

Un piège de syntaxe. La première version du fichier laissait un but non fermé au palier : en Lean, `a :: b ++ c` se lit `(a :: b) ++ c`, et non `a :: (b ++ c)`. Les deux listes sont égales par définition de `++`, mais la tactique `rw` ne le voit pas ; une ligne `rfl` a suffi.

6. Ce que le théorème dit, et ce qu'il ne dit pas

Le théorème 1 établit que la formalisation Lean de la note 251 est fidèle : la récurrence par règle de tête, transitions et palier n'ajoute ni ne retire rien à la définition par la primalité. Les lemmes se transportent sur toute suite de bits à la place de la suite des nombres premiers, comme l'avaient observé plusieurs notes précédentes : la démonstration n'utilise de la primalité que le fait que 3 est premier.

Il n'en résulte aucune information sur la présence d'une lettre a dans chaque colonne : cette présence est exactement la conjecture de Goldbach, et ce théorème est une réécriture fidèle, pas un raccourci. Il reste néanmoins un résultat vérifié par machine sur un objet que j'aime, et qui est maintenant spécifié sans ambiguïté, ce qui n'était pas le cas des tentatives de preuve antérieures, où des cas étaient oubliés ou contradictoires.

7. Notes voisines : ia251, ia252, ia235

Trois notes de septembre 2026 touchent à des questions proches sans se recouvrir.

La note https://denisevellachemla.eu/ia251.pdf : le programme du treillis, qui calcule les mots de $n = 6$ à 118 par la récurrence. Son défaut est de ne rien garantir : rien n'assurait que les mots calculés soient ceux que définit la primalité, pour tout n. Le théorème 1 comble ce manque.

La note https://denisevellachemla.eu/ia252.pdf : un programme `find_decomposants`, qui liste les $p \leq n/2$ vérifiant les deux conditions de congruence ($p \not\equiv 0$ et $p \not\equiv n$ modulo q pour tout $q \leq \sqrt{n}$), et un énoncé de Goldbach de cette forme laissé avec `sorry`. Son défaut : les modules q y sont testés à partir de 3 et non de 2. Des p pairs ou composés passent alors le test : pour $n = 8$ on obtient $p \in \{3, 4\}$, pour $n = 12$ on obtient $p = 4$, et pour 498 des 998 nombres pairs de 6 à 2000 la liste contient au moins un faux candidat. Pour $n = 98$ aucun problème : on trouve 19, 31, 37. En testant les modules à partir de 2, on ne trouve aucun faux candidat jusqu'à 2000. Il suffit donc de remplacer 3 par 2 dans la borne des q, dans le théorème et dans `satisfies_condition`.

La note https://denisevellachemla.eu/ia235.pdf (de lumo) : la formalisation Lean du lemme de Leila Schneps (un entier sans petit diviseur est premier) et le théorème `candidate_implies_goldbach`, qui démontre, pour les modules $q \geq 2$, qu'un p vérifiant les deux conditions donne une décomposition $n = p + (n - p)$ en deux nombres premiers.