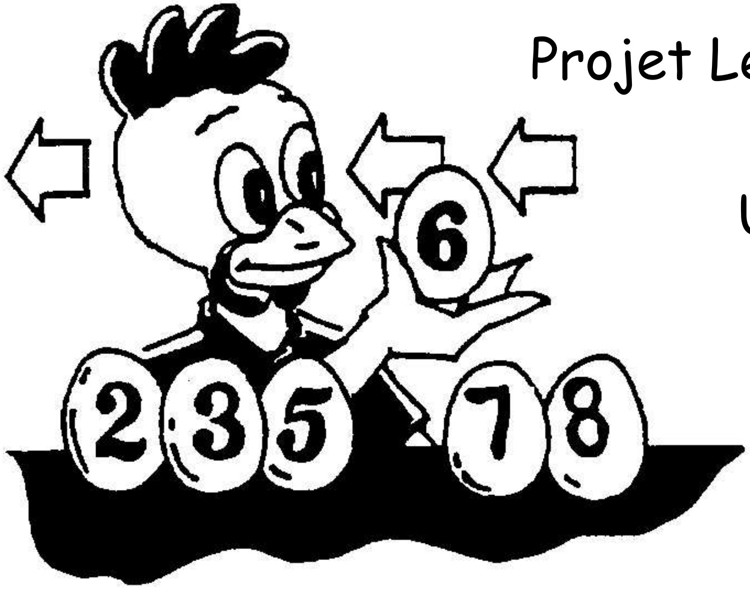
Vérification formelle en Coq:

L'algorithme de Knuth pour les nombres premiers

Laurent Théry

Projet Lemme INRIA Sophia-Antipolis
&
Université de L'Aquila



Motivations

Motivations

Lemme:
Application des méthodes formelles au calcul scientifique

Motivations

Lemme:
Application des méthodes formelles au calcul scientifique

Coq: l'outil de prédilection

Motivations

Lemme:
Application des méthodes formelles au calcul scientifique

Coq: l'outil de prédilection

Généricité

Motivations

Lemme:
Application des méthodes formelles au calcul scientifique

Coq: l'outil de prédilection

- Généricité
- Expressivité

Motivations

Lemme:
Application des méthodes formelles
au calcul scientifique

Coq: l'outil de prédilection

- Généricité
- Expressivité
- Maturité

Motivations

Lemme:

Application des méthodes formelles au calcul scientifique

Coq: l'outil de prédilection

- Généricité
- Expressivité
- Maturité

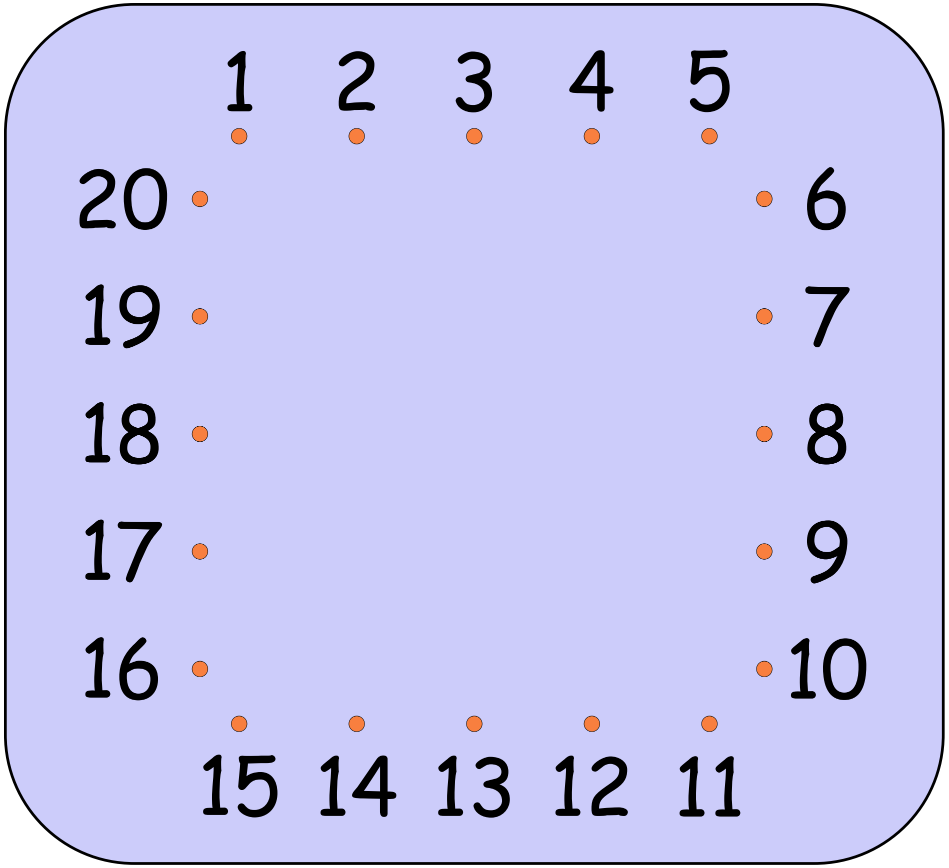
Illustration par un exemple

Petit Problème

Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?

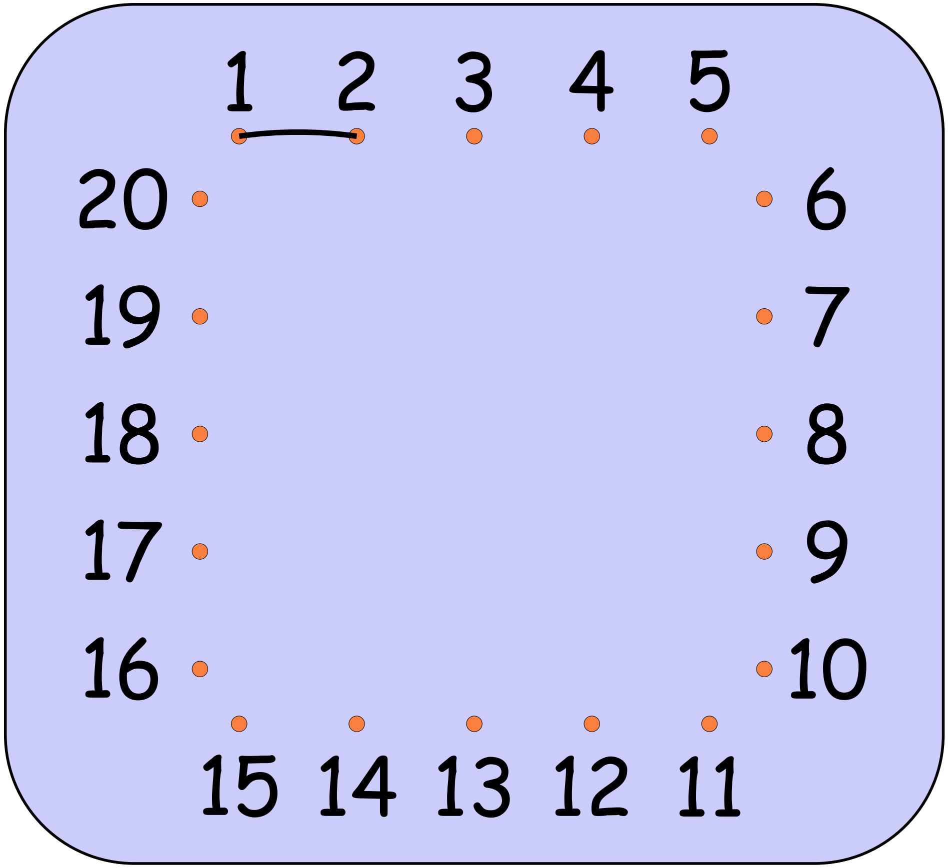
Petit Problème

Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?



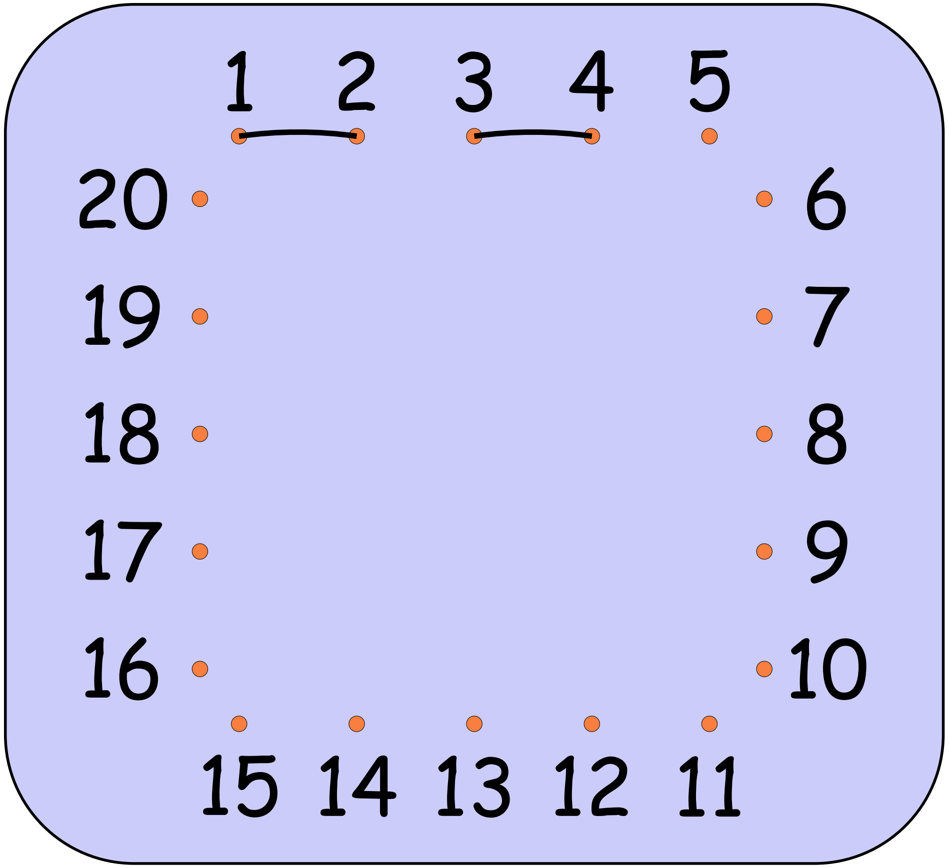
Petit Problème

Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?



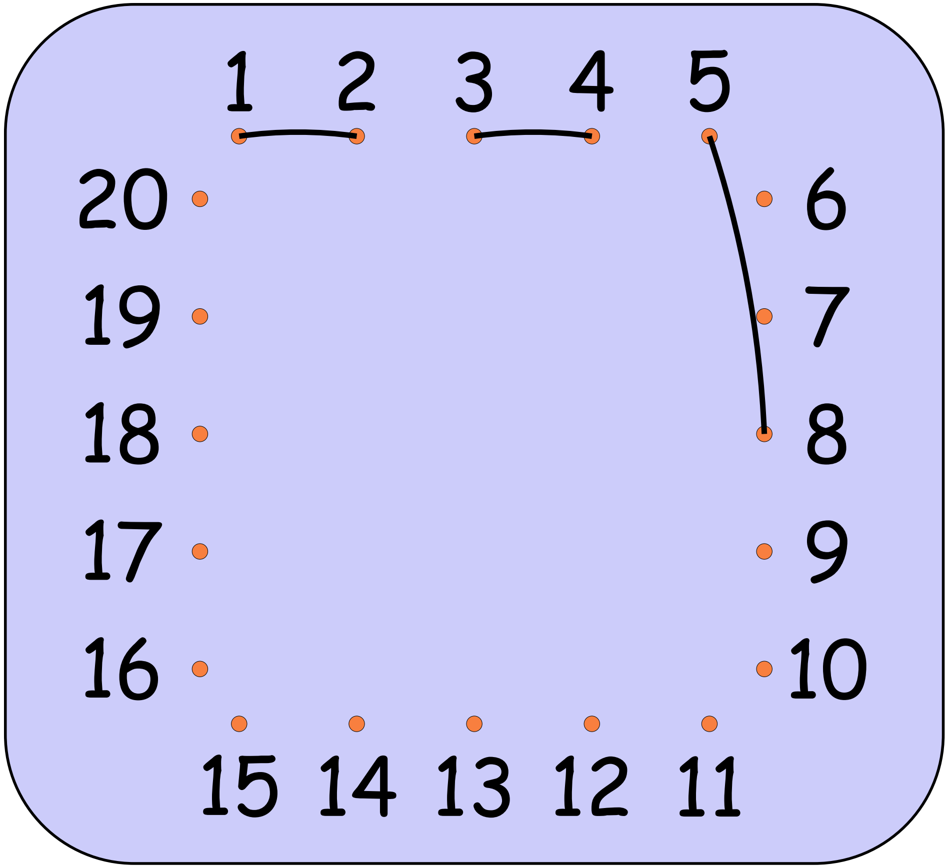
Petit Problème

Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?



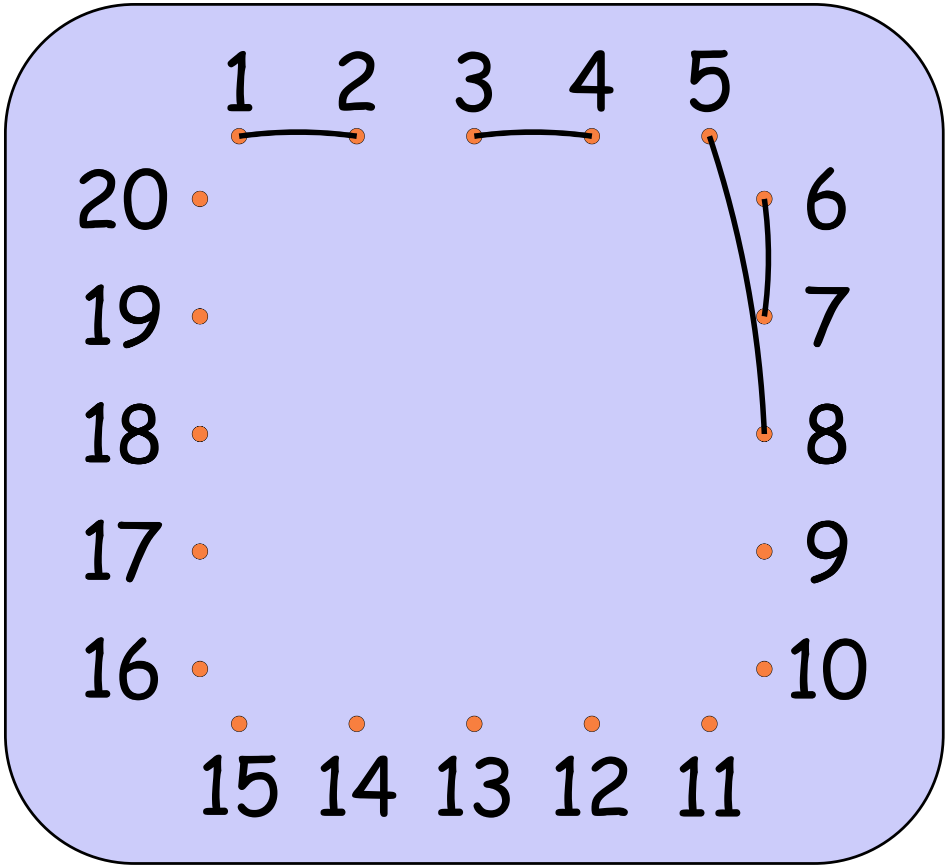
Petit Problème

Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?



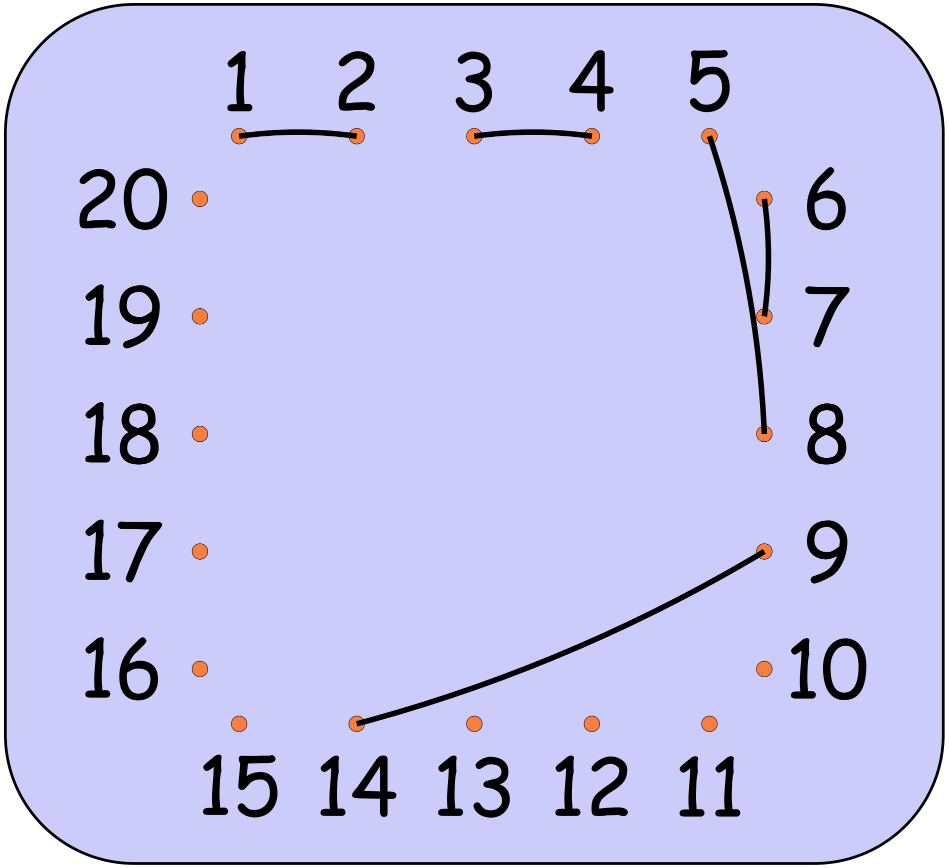
Petit Problème

Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?



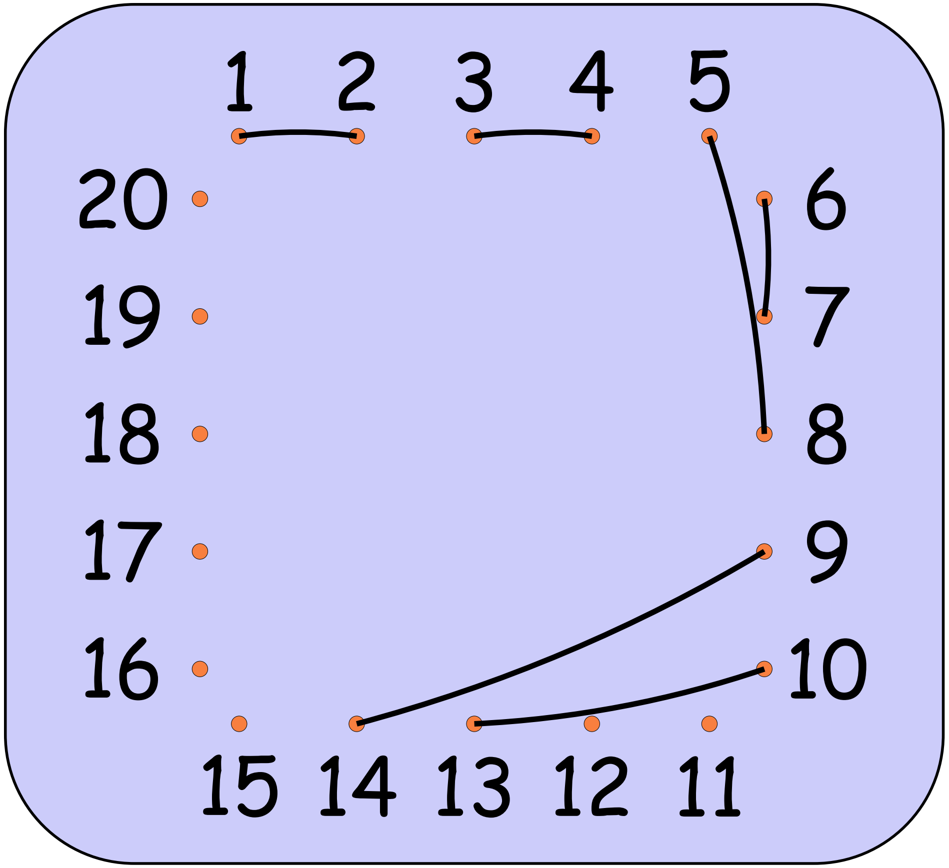
Petit Problème

Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?



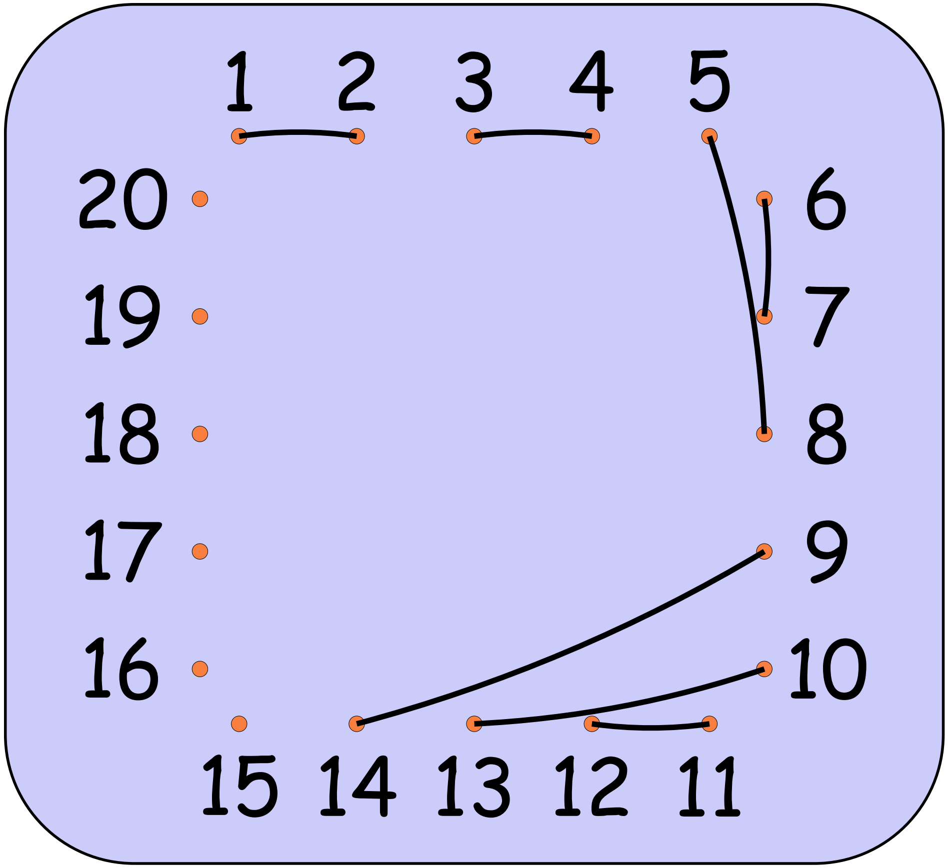
Petit Problème

Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?



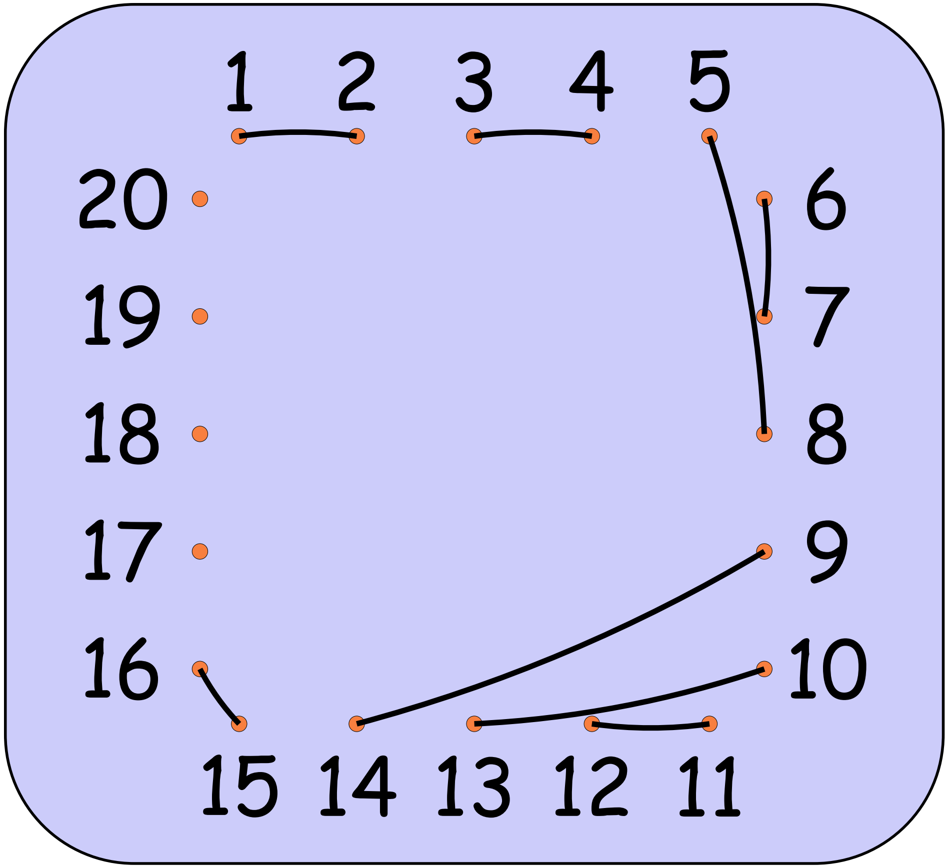
Petit Problème

Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?



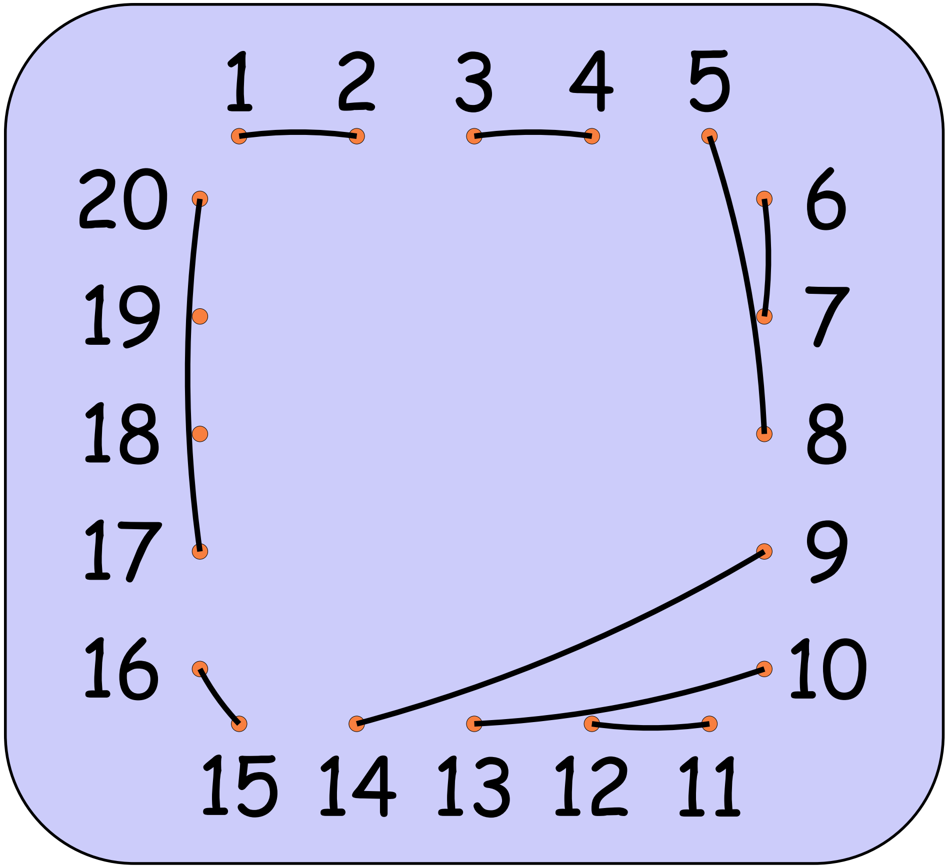
Petit Problème

Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?



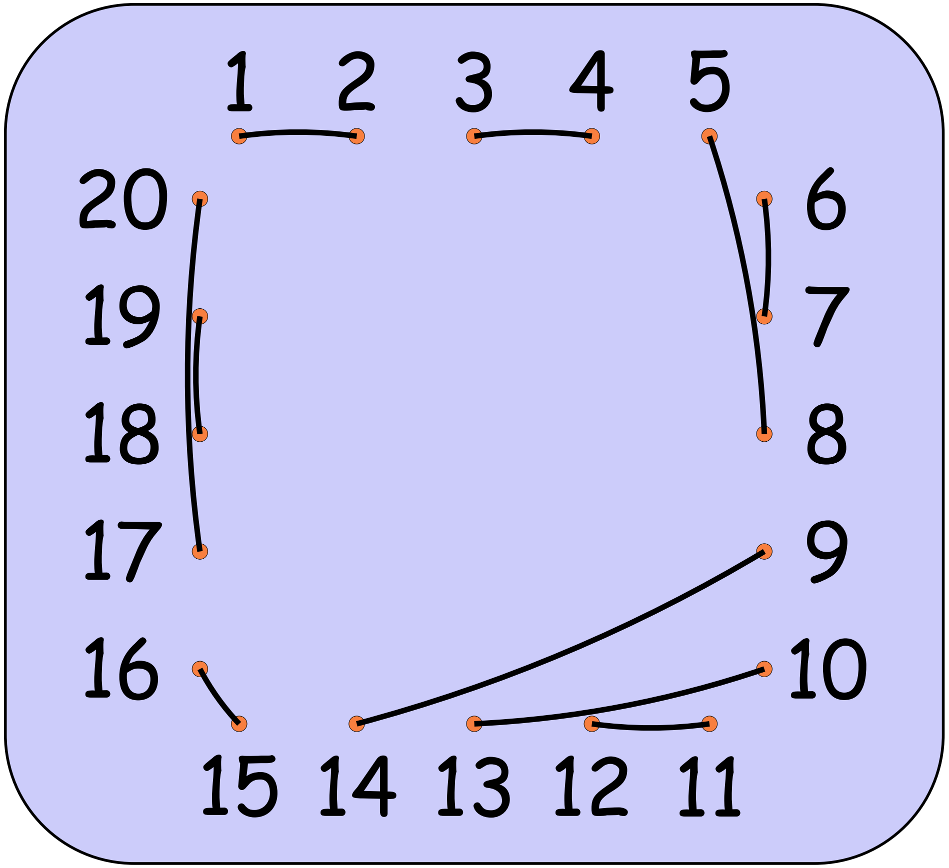
Petit Problème

Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?



Petit Problème

Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?



Coq

Coq

$\mathbb{N}$: Set

Coq

$\mathbb{N}$: `Set`

`O` : $\mathbb{N}$

`O`

Coq

$\mathbb{N}$: `Set`

$\quad$ `O` : $\mathbb{N}$

$\quad$ `S` : $\mathbb{N} \to \mathbb{N}$

`O , S(O), S(S(O)), ...`

Coq

```
Inductive ℕ: Set :=
    O : ℕ
|   S : ℕ→ℕ .
```

O , S(O), S(S(O)), ...

Coq

```
Inductive ℕ:  Set :=
   O : ℕ
|  S : ℕ→ℕ .
```

`O , S(O), S(S(O)), ...`

$$\begin{array}{l} \quad P(\mathtt{O}) \\ \wedge \\ \quad \forall n : \mathbb{N}.\ P(n) \Rightarrow P(\mathtt{S}(n)) \end{array} \Rightarrow \forall n : \mathbb{N}.\ P(n)$$

Coq: addition

```
Inductive ℕ:  Set:=
   O : ℕ
|  S : ℕ→ℕ.
```

```
Fixpoint + [a,b:nat] :  nat :=
Cases a of
     O => b
 | S(a') => S(a' + b)
end
```

Coq: multiplication

```
Inductive ℕ:  Set:=
    O : ℕ
|   S : ℕ→ℕ.
```

```
Fixpoint * [a,b:nat] :  nat :=
Cases a of
      O => O
 | S a' => b + (a' * b)
end.
```

Coq: divisibilité

Coq: divisibilité

$$\exists c\!:\mathbb{N}.\quad b \;\texttt{=}\; c \;\texttt{*}\; a$$

Coq: divisibilité

$$\texttt{Definition divides :}\ \ \mathbb{N} \to \mathbb{N} \to \texttt{Prop :=}$$
$$\lambda a\texttt{,}\, b\texttt{:}\mathbb{N}\texttt{.}\quad \exists c\texttt{:}\,\mathbb{N}\texttt{.}\quad b \texttt{ = } c \texttt{ * } a \texttt{ .}$$

Coq: divisibilité

$\texttt{Definition divides : } \mathbb{N} \rightarrow \mathbb{N} \rightarrow \texttt{Prop :=}$
$\quad \lambda a\texttt{,} b\texttt{:}\mathbb{N}\texttt{.} \quad \exists c\texttt{:}\mathbb{N}\texttt{.} \quad b \texttt{ = } c \texttt{ * } a \texttt{ .}$

$\texttt{Theorem divides\_trans:}$
$\forall a\texttt{,}b\texttt{,}c\texttt{:}\mathbb{N}\texttt{.}$

$$\begin{array}{c} \texttt{divides}(a,b) \\ \wedge \\ \texttt{divides}(b,c) \end{array} \Rightarrow \texttt{divides}(a,c).$$

Coq: primalité

$$\texttt{Definition prime}: \mathbb{N} \rightarrow \texttt{Prop} := \lambda a{:}\mathbb{N}.$$

$$\forall b{:}\mathbb{N}.\ \texttt{divides}(b, a) \Rightarrow \vee \begin{array}{l} b = \texttt{S}(\texttt{O}) \\ b = a \end{array}$$

Coq: primalité

$$\texttt{Definition prime}: \mathbb{N} \to \texttt{Prop} \;\texttt{:=}\; \lambda a{:}\mathbb{N}.$$

$$\bigwedge \begin{cases} \forall b{:}\mathbb{N}.\ \texttt{divides}(b, a) \Rightarrow \bigvee \begin{array}{l} b = \mathtt{S}(\mathtt{O}) \\ b = a \end{array} \\ a \neq \mathtt{S}(\mathtt{O}) \end{cases}$$

Algorithme de Knuth

Algorithme de Knuth

```
int[] firstPrimes(int n) {
  int[] res = new int[n];
  int number=2;

  for (int i=0; i<n; i++) {













    res[i]=number;
    number++;
  }
  return res;
}
```

Algorithme de Knuth

```
int[] firstPrimes(int n) {
  int[] res = new int[n];
  int number=2;
  boolean isPrime;
  for (int i=0; i<n; i++) {
    while (true) {
      isPrime=true;
      for(int j=2; j<number; j++) {
        if (number%j==0) {
          isPrime=false;
          break;
        }
      }
      if (isPrime) break;
      number++;
    }
    res[i]=number;
    number++;
  }
  return res;
}
```

Algorithme de Knuth

```
int[] firstPrimes(int n) {
  int[] res = new int[n];
  int number=2;
  boolean isPrime;
  for (int i=0; i<n; i++) {
    while (true) {
      isPrime=true;
      for(int j=0; j<i; j++) {
        if (number%res[j]==0) {
          isPrime=false;
          break;
        }
      }
      if (isPrime) break;
      number++;
    }
    res[i]=number;
    number++;
  }
  return res;
}
```

Algorithme de Knuth

```
int[] firstPrimes(int n) {
  int[] res = new int[n];
  res[0]=2;
  int number=3;
  boolean isPrime;
  for (int i=1; i<n; i++) {
    while (true) {
      isPrime=true;
      for(int j=0; j<i; j++) {
        if (number%res[j]==0) {
          isPrime=false;
          break;
        }
      }
      if (isPrime) break;
      number+=2;
    }
    res[i]=number;
    number+=2;
  }
  return res;
}
```

Algorithme de Knuth

```
int[] firstPrimes(int n) {
  int[] res = new int[n];
  res[0]=2;
  int number=3, snum;
  boolean isPrime;
  for (int i=1; i<n; i++) {
    while (true) {
      isPrime=true;
      snum = (int)Math.sqrt(number);
      for(int j=0; j<i && res[j]<= snum; j++) {
        if (number%res[j]==0) {
          isPrime=false;
          break;
        }
      }
      if (isPrime) break;
      number+=2;
    }
    res[i]=number;
    number+=2;
  }
  return res;
}
```

Algorithme de Knuth

```
int[] firstPrimes(int n) {
  int[] res = new int[n];
  res[0]=2;
  int number=3, snum;
  boolean isPrime;
  for (int i=1; i<n; i++) {
    while (true) {
      isPrime=true;
      snum = (int)Math.sqrt(number);
      for(int j=0; res[j]<= snum; j++) {
        if (number%res[j]==0) {
          isPrime=false;
          break;
        }
      }
      if (isPrime) break;
      number+=2;
    }
    res[i]=number;
    number+=2;
  }
  return res;
}
```

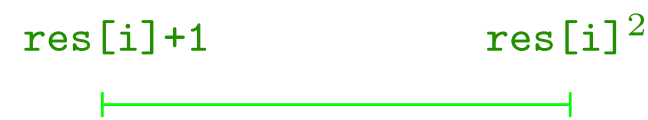
Algorithme de Knuth

```
int[] firstPrimes(int n) {
  int[] res = new int[n];
  res[0]=2;
  int number=3, snum;
  boolean isPrime;
  for (int i=1; i<n; i++) {
    while (true) {
      isPrime=true;
      snum = (int)Math.sqrt(number);
      for(int j=0; res[j]<= snum; j++) {
        if (number%res[j]==0) {
          isPrime=false;
          break;
        }
      }
      if (isPrime) break;
      number+=2;
    }
    res[i]=number;
    number+=2;
  }
  return res;
}
```

res[i]+1 $\qquad$ res[i]^2

Algorithme de Knuth

```java
int[] firstPrimes(int n) {
  int[] res = new int[n];
  res[0]=2;
  int number=3, snum;
  boolean isPrime;
  for (int i=1; i<n; i++) {
    while (true) {
      isPrime=true;
      snum = (int)Math.sqrt(number);
      for(int j=0; res[j]<= snum; j++) {
        if (number%res[j]==0) {
          isPrime=false;
          break;
        }
      }
      if (isPrime) break;
      number+=2;
    }
    res[i]=number;
    number+=2;
  }
  return res;
}
```



Théorème de Bertrand:

Why

Why

Vérification de programmes: ML, C, Java

Why

Vérification de programmes: ML, C, Java

Programme + annotations

Why

Vérification de programmes: ML, C, Java

Programme + annotations

Génération de conditions

Why

Vérification de programmes: ML, C, Java

Programme + annotations

Génération de conditions

Indépendant de Coq

Why

Vérification de programmes: ML, C, Java

Programme + annotations

Génération de conditions

Indépendant de Coq

Développé par J.C. Filliâtre

Algorithme en Ocaml

```
isPrime := true; number := 3;
snum := 0; i := 1; res[0] := 2;
while ((!i) < n) do
  isPrime:=true; snum := (sqr !number); j := 0;
  while (!isPrime && res[!j] <= !snum) do
    if (!number % res[!j]) = 0 then
      isPrime:=false
    else
      j := !j + 1
  done;
  if (!isPrime) then
    begin
      res[!i] := !number; i:=!i+1
    end;
  number := !number+2
done
```

Annotations

Annotations

$$\mathbf{P}$$

Annotations

$$\{\ \textit{Pré-conditions}\ \}$$
$$\mathbf{P}$$

Annotations

$$\{\ \textit{Pré-conditions}\ \}$$
$$\mathbf{P}$$
$$\{\ \textit{Post-conditions}\ \}$$

Annotations

$$\{\ \textit{Pré-conditions}\ \}$$
$$\mathbf{P}$$
$$\{\ \textit{Post-conditions}\ \}$$

Exemple

Annotations

$$\{\ \textit{Pré-conditions}\ \}$$
$$\mathbf{P}$$
$$\{\ \textit{Post-conditions}\ \}$$

Exemple

```
x := !x + 2;
```

Annotations

$$\{\ \textit{Pré-conditions}\ \}$$

$$\mathbf{P}$$

$$\{\ \textit{Post-conditions}\ \}$$

Exemple

{ `odd(x)` }

```
x := !x + 2;
```

Annotations

$$\{\ \textit{Pré-conditions}\ \}$$
$$\mathbf{P}$$
$$\{\ \textit{Post-conditions}\ \}$$

Exemple

```
{ odd(x) }
x := !x + 2;
{ odd(x) }
```

Annotations

$$\{\ \textit{Pré-conditions}\ \}$$
$$\mathbf{P}$$
$$\{\ \textit{Post-conditions}\ \}$$

Exemple

```
{ odd(x) }
x := !x + 2;
{ odd(x) }
```

Condition générée:

$$\forall x.\ \mathtt{odd}(x) \Rightarrow \mathtt{odd}(x+2)$$

Annotations

Boucle:

Annotations

Boucle:

```
while (C) do

        P

done
```

Annotations

Boucle:

```
while (C) do
 { invariant I

        P

 done
```

Annotations

Boucle:

```
while (C) do
 { invariant I
   variant  V
 }
       P

done
```

Annotations

Exemple:

Annotations

Exemple:

```
while (0 < !i) do


     x := !x + 2
done
```

Annotations

Exemple:

```
while (0 < !i) do
   { invariant odd(x)

   x := !x + 2
done
```

Annotations

Exemple:

```
while (0 < !i) do
  { invariant odd(x)
      variant  i
  }
  x := !x + 2
done
```

Annotations

Exemple:

```
while (0 < !i) do
    { invariant odd(x)
        variant  i
    }
    x := !x + 2
 done
```

Conditions générées:

$$\forall x.\ \mathsf{odd}(x) \Rightarrow \mathsf{odd}(x+2)$$

$$\forall i.\ 0 \leq i - 1 < i$$

Algorithme en Ocaml

```
isPrime := true; number := 3;
snum := 0; i := 1; res[0] := 2;
while ((!i) < n) do
  isPrime:=true; snum := (sqr !number); j := 0;
  while (!isPrime && res[!j] <= !snum) do
    if (!number % res[!j]) = 0 then
      isPrime:=false
    else
      j := !j + 1
  done;
  if (!isPrime) then
    begin
      res[!i] := !number; i:=!i+1
    end;
  number := !number+2
done
```

Post-condition

Post-condition

{

}

Post-condition

{

$$\forall k.\ 0 \le k < n \Rightarrow \texttt{prime}(res[k])$$

}

Post-condition

{

$$\forall k.\ 0 \leq k < n \Rightarrow \texttt{prime}(res[k])$$

$\wedge$

$$\forall k, j.\ 0 \leq k < j < n \Rightarrow res[k] < res[j]$$

}

Post-condition

{

$$\forall k.\ 0 \leq k < n \Rightarrow \texttt{prime}(res[k])$$

$\wedge$

$$\forall k, j.\ 0 \leq k < j < n \Rightarrow res[k] < res[j]$$

$\wedge$

$$\forall k.\ \wedge \begin{array}{c} 0 \leq k \leq res[n-1] \\ \texttt{prime}(k) \end{array} \Rightarrow \exists j.\ \wedge \begin{array}{c} 0 \leq j < n \\ res[j] = k \end{array}$$

}

Première Boucle

```
while ((!i) < n)
```

Première Boucle

```
while ((!i) < n) {
 invariant



 variant

}
```

Première Boucle

```
while ((!i) < n) {
 invariant
```

$$res[i-1] < number < 2 * res[i-1]$$

```
 variant

}
```

Première Boucle

```
while ((!i) < n) {
 invariant
```

$$res[i-1] < number < 2 * res[i-1]$$
$$\wedge\ \texttt{odd}(number)$$

```
 variant

}
```

Première Boucle

```
while ((!i) < n) {
 invariant
```

$$res[i-1] < number < 2 * res[i-1]$$
$$\wedge\ \mathtt{odd}(number)$$
$$\wedge\ \forall k.\ res[i-1] < k < number \Rightarrow \neg\mathtt{prime}(k)$$

```
 variant

}
```

Première Boucle

```
while ((!i) < n) {
 invariant
```

$$res[i-1] < number < 2 * res[i-1]$$
$$\wedge\ \texttt{odd}(number)$$
$$\wedge\ \forall k.\ res[i-1] < k < number \Rightarrow \neg\texttt{prime}(k)$$
$$\wedge\ \forall k.\ 0 \leq k < i \Rightarrow \texttt{prime}(res[k])$$

```
 variant

}
```

Première Boucle

```
while ((!i) < n) {
 invariant
```

$$
\begin{aligned}
& res[i-1] < number < 2 * res[i-1] \\
\wedge\ & \mathtt{odd}(number) \\
\wedge\ & \forall k.\ res[i-1] < k < number \Rightarrow \neg\mathtt{prime}(k) \\
\wedge\ & \forall k.\ 0 \leq k < i \Rightarrow \mathtt{prime}(res[k]) \\
\wedge\ & \forall k, j.\ 0 \leq k < j < i \Rightarrow res[k] < res[j]
\end{aligned}
$$

```
 variant

}
```

Première Boucle

while $((!i) < n)$ {

invariant

$$
\begin{array}{l}
res[i-1] < number < 2 * res[i-1] \\
\wedge\ \texttt{odd}(number) \\
\wedge\ \forall k.\ res[i-1] < k < number \Rightarrow \neg\texttt{prime}(k) \\
\wedge\ \forall k.\ 0 \leq k < i \Rightarrow \texttt{prime}(res[k]) \\
\wedge\ \forall k, j.\ 0 \leq k < j < i \Rightarrow res[k] < res[j] \\
\wedge\ \forall k.\ \wedge \begin{array}{c} 0 \leq k \leq res[i-1] \\ \texttt{prime}(k) \end{array} \Rightarrow \exists j.\ \wedge \begin{array}{c} 0 \leq j < i \\ res[j] = k \end{array}
\end{array}
$$

variant

}

Première Boucle

```
while ((!i) < n) {
 invariant
```

$$
\begin{array}{l}
\quad res[i-1] < number < 2 * res[i-1] \\
\wedge \ \mathtt{odd}(number) \\
\wedge \ \forall k.\ res[i-1] < k < number \Rightarrow \neg\mathtt{prime}(k) \\
\wedge \ \forall k.\ 0 \leq k < i \Rightarrow \mathtt{prime}(res[k]) \\
\wedge \ \forall k, j.\ 0 \leq k < j < i \Rightarrow res[k] < res[j] \\
\wedge \ \forall k.\ \wedge \begin{array}{c} 0 \leq k \leq res[i-1] \\ \mathtt{prime}(k) \end{array} \Rightarrow \exists j.\ \wedge \begin{array}{c} 0 \leq j < i \\ res[j] = k \end{array}
\end{array}
$$

```
 variant
```

$$(n - i, 2 * res[i-1] - number) \ \mathtt{for\ lexZ}$$

```
}
```

Seconde Boucle

```
while (!isPrime && res[!j] <= !snum) do
```

Seconde Boucle

```
while (!isPrime && res[!j] <= !snum) do {
 invariant







 variant

}
```

Seconde Boucle

```
while (!isPrime && res[!j] <= !snum) do {
 invariant
    if(isPrime)
      then
        ∀k. 0 ≤ k < j ⇒ ¬divides(res[k], number)
      else
        divides(res[j], number))

 variant

}
```

Seconde Boucle

```
while (!isPrime && res[!j] <= !snum) do {
 invariant
    if(isPrime)
      then
        ∀k. 0 ≤ k < j ⇒ ¬divides(res[k], number)
      else
        divides(res[j], number))
   ∧ 0 ≤ j < i
 variant

}
```

Seconde Boucle

```
while (!isPrime && res[!j] <= !snum) do {
 invariant
    if(isPrime)
      then
        ∀k. 0 ≤ k < j ⇒ ¬divides(res[k], number)
      else
        divides(res[j], number))
  ∧ 0 ≤ j < i
 variant
    isPrime + i − j
}
```

Conditions de Vérification

Conditions de Vérification

22 Conditions

Conditions de Vérification

22 Conditions

Toutes faciles à prouver sauf

$$res[i-1] < number < 2 * res[i-1]$$

Théorème de Bertrand

Pour n plus grand que 2, il y a toujours au moins un nombre premier strictement entre n et $2n$.

Théorème de Bertrand

Pour n plus grand que 2, il y a toujours au moins un nombre premier strictement entre n et $2n$.

Preuve par l'absurde

Théorème de Bertrand

Pour n plus grand que 2, il y a toujours au moins un nombre premier strictement entre n et $2n$.

Preuve par l'absurde

Borne Sup: $\mathrm{C}_{2n}^{n} < (2n)^{\sqrt{2n}/2-1} 4^{2n/3}$

Théorème de Bertrand

Pour n plus grand que 2, il y a toujours au moins un nombre premier strictement entre n et $2n$.

Preuve par l'absurde

Borne Sup: $\mathrm{C}_{2n}^{n} < (2n)^{\sqrt{2n}/2-1} 4^{2n/3}$

Borne Inf: $4^n \leq 2n\mathrm{C}_{2n}^{n}$

Théorème de Bertrand

Pour n plus grand que 2, il y a toujours au moins un nombre premier strictement entre n et $2n$.

Preuve par l'absurde

Borne Sup: $\mathrm{C}_{2n}^{n} < (2n)^{\sqrt{2n}/2-1} 4^{2n/3}$

Borne Inf: $4^n \leq 2n\mathrm{C}_{2n}^{n}$

Condition Nécessaire: $4^{n/3} < (2n)^{\sqrt{2n}/2}$

Exemple de propriétés

Borne Sup sur le Produit des Nombres Premiers

$$\prod_{p \leq n} p < 4^n$$

Exemple de propriétés

Borne Sup sur le Produit des Nombres Premiers

$$\prod_{p \leq n} p < 4^n$$

Par induction forte sur n

Exemple de propriétés

Borne Sup sur le Produit des Nombres Premiers

$$\prod_{p \leq n} p < 4^n$$

Par induction forte sur n

$2 < 4^2$

Exemple de propriétés

Borne Sup sur le Produit des Nombres Premiers

$$\prod_{p \leq n} p < 4^n$$

Par induction forte sur n

$2 < 4^2$

Si n est impair, $\prod_{p \leq n+1} p = \prod_{p \leq n} p < 4^n < 4^{n+1}$

Exemple de propriétés

Borne Sup sur le Produit des Nombres Premiers

$$\prod_{p \leq n} p < 4^n$$

Par induction forte sur n

$2 < 4^2$

Si n est impair, $\prod_{p \leq n+1} p = \prod_{p \leq n} p < 4^n < 4^{n+1}$

Si n est pair, $\prod_{p \leq 2m+1} p = (\prod_{p \leq m+1} p) \quad (\prod_{m+1 < p \leq 2m+1} p)$

Exemple de propriétés

Borne Sup sur le Produit des Nombres Premiers

$$\prod_{p \leq n} p < 4^n$$

Par induction forte sur n

$2 < 4^2$

Si n est impair, $\prod_{p \leq n+1} p = \prod_{p \leq n} p < 4^n < 4^{n+1}$

Si n est pair, $\prod_{p \leq 2m+1} p = (\prod_{p \leq m+1} p) \quad (\prod_{m+1 < p \leq 2m+1} p)$

$$\prod_{p \leq 2m+1} p < 4^{m+1} \quad (\prod_{m+1 < p \leq 2m+1} p)$$

Exemple de propriétés

Borne Sup sur le Produit des Nombres Premiers

$$\prod_{p \leq n} p < 4^n$$

Par induction forte sur n

$2 < 4^2$

Si n est impair, $\prod_{p \leq n+1} p = \prod_{p \leq n} p < 4^n < 4^{n+1}$

Si n est pair, $\prod_{p \leq 2m+1} p = (\prod_{p \leq m+1} p) \quad (\prod_{m+1 < p \leq 2m+1} p)$

$$\prod_{p \leq 2m+1} p < 4^{m+1} \quad (\prod_{m+1 < p \leq 2m+1} p)$$

$$\prod_{p \leq 2m+1} p < 4^{m+1} \mathrm{C}_{2m+1}^{m+1}$$

Exemple de propriétés

Borne Sup sur le Produit des Nombres Premiers

$$\prod_{p \leq n} p < 4^n$$

Par induction forte sur n

$2 < 4^2$

Si n est impair, $\prod_{p \leq n+1} p = \prod_{p \leq n} p < 4^n < 4^{n+1}$

Si n est pair, $\prod_{p \leq 2m+1} p = (\prod_{p \leq m+1} p) \quad (\prod_{m+1 < p \leq 2m+1} p)$

$$\prod_{p \leq 2m+1} p < 4^{m+1} \quad (\prod_{m+1 < p \leq 2m+1} p)$$

$$\prod_{p \leq 2m+1} p < 4^{m+1} \mathrm{C}_{2m+1}^{m+1}$$

$$\prod_{p \leq 2m+1} p < 4^{m+1} 4^m$$

Exemple de propriétés

Borne Sup sur le Produit des Nombres Premiers

$$\prod_{p \leq n} p < 4^n$$

Par induction forte sur n

$2 < 4^2$

Si n est impair, $\prod_{p \leq n+1} p = \prod_{p \leq n} p < 4^n < 4^{n+1}$

Si n est pair, $\prod_{p \leq 2m+1} p = (\prod_{p \leq m+1} p) \quad (\prod_{m+1 < p \leq 2m+1} p)$

$$\prod_{p \leq 2m+1} p < 4^{m+1} \quad (\prod_{m+1 < p \leq 2m+1} p)$$

$$\prod_{p \leq 2m+1} p < 4^{m+1} \mathrm{C}_{2m+1}^{m+1}$$

$$\prod_{p \leq 2m+1} p < 4^{m+1} 4^m$$

$$\prod_{p \leq 2m+1} p < 4^{2m+1}$$

Condition Nécessaire

$$4^{n/3} < (2n)^{\sqrt{2n}/2}$$

Condition Nécessaire

$$4^{n/3} < (2n)^{\sqrt{2n}/2}$$

Passage au logarithme:

$$\frac{n}{3}\, ln(4) < \frac{\sqrt{2n}}{2} ln(2n)$$

Condition Nécessaire

$$4^{n/3} < (2n)^{\sqrt{2n}/2}$$

Passage au logarithme:

$$\frac{n}{3}\,ln(4) < \frac{\sqrt{2n}}{2} ln(2n)$$

Simplification:

$$\sqrt{8n}\,ln(2) - 3\,ln(2n) < 0$$

Analyse de fonction

Analyse de fonction

Inégalité: $\sqrt{8n}\, ln(2) - 3\, ln(2n) < 0$

Analyse de fonction

Inégalité: $\sqrt{8n}\, ln(2) - 3\, ln(2n) < 0$

Fonction: $f(x) = \sqrt{8x}\, ln(2) - 3ln(2x)$

Analyse de fonction

Inégalité: $\sqrt{8n}\,ln(2) - 3\,ln(2n) < 0$

Fonction: $f(x) = \sqrt{8x}\,ln(2) - 3ln(2x)$

Évaluation: $f(2^7) = 2^5 ln(2) - 3.2^3 ln(2) > 0$

Analyse de fonction

Inégalité: $\sqrt{8n}\, ln(2) - 3\, ln(2n) < 0$

Fonction: $f(x) = \sqrt{8x}\, ln(2) - 3ln(2x)$

Évaluation: $f(2^7) = 2^5 ln(2) - 3.2^3 ln(2) > 0$

Variation: $f'(x) = \dfrac{\sqrt{2x}\, ln(2) - 3}{x}$

Réels dans Coq

Réels dans Coq

Axiomatisation

Réels dans Coq

Axiomatisation

Limite

Réels dans Coq

Axiomatisation

Limite

Continuité

Réels dans Coq

Axiomatisation

Limite

Continuité

Dérivabilité

Réels dans Coq

Axiomatisation

Limite

Continuité

Dérivabilité

...

Cas restants

Prouver que le théorème est vrai pour $n < 128$

Cas restants

Prouver que le théorème est vrai pour $n < 128$

Preuve cas par cas

Cas restants

Prouver que le théorème est vrai pour $n < 128$

- Preuve cas par cas
- Réflection:

Cas restants

Prouver que le théorème est vrai pour $n < 128$

- Preuve cas par cas
- Réflection:
  - Écrire un programme

Cas restants

Prouver que le théorème est vrai pour $n < 128$

- Preuve cas par cas
- Réflection:
  - Écrire un programme
  - Prouver sa correction

Cas restants

Prouver que le théorème est vrai pour $n < 128$

Preuve cas par cas

Réflection:

Écrire un programme

Prouver sa correction

Le faire tourner pour 128

Schéma de la Preuve

Programme de Knuth

Schéma de la Preuve

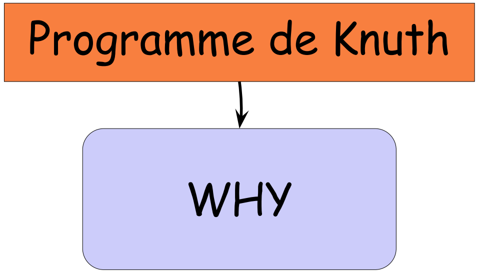


Schéma de la Preuve

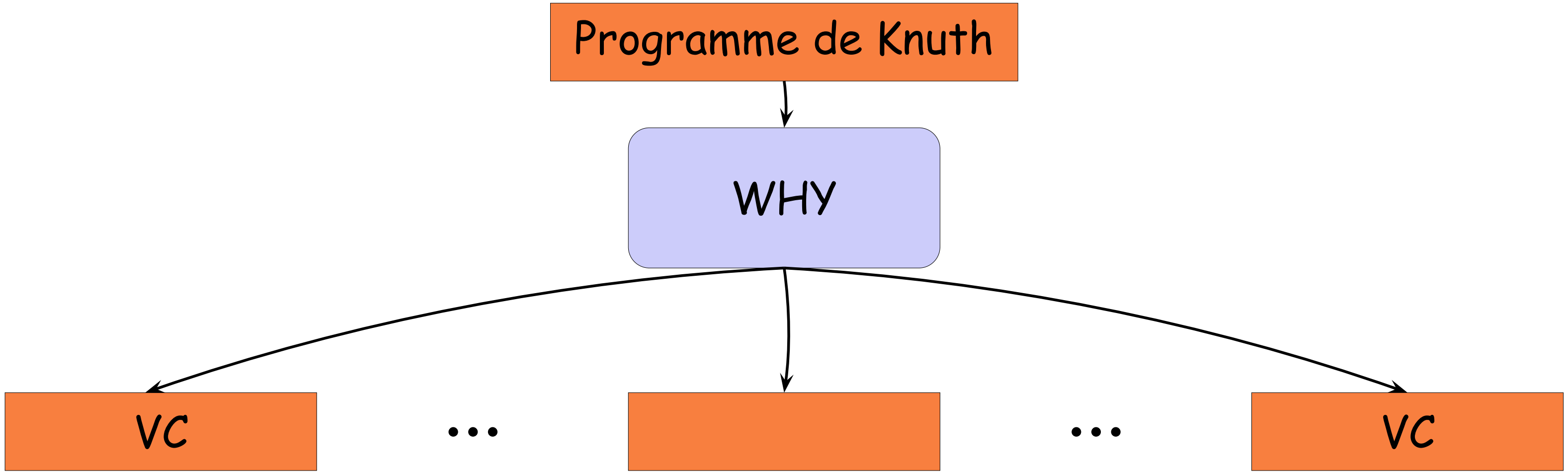


Schéma de la Preuve

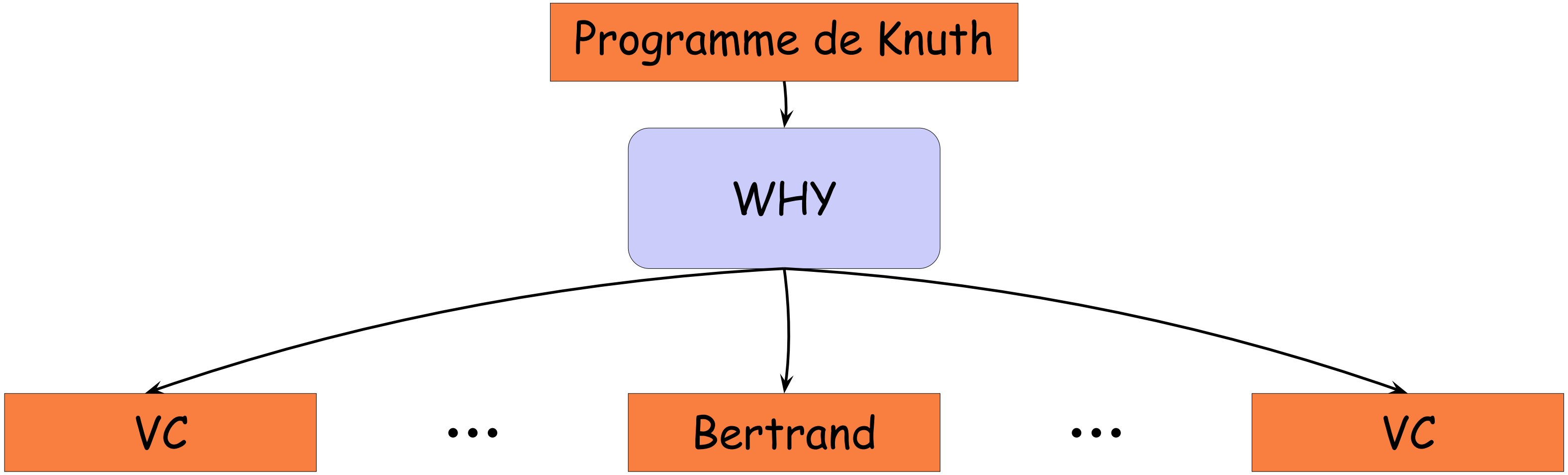


Schéma de la Preuve

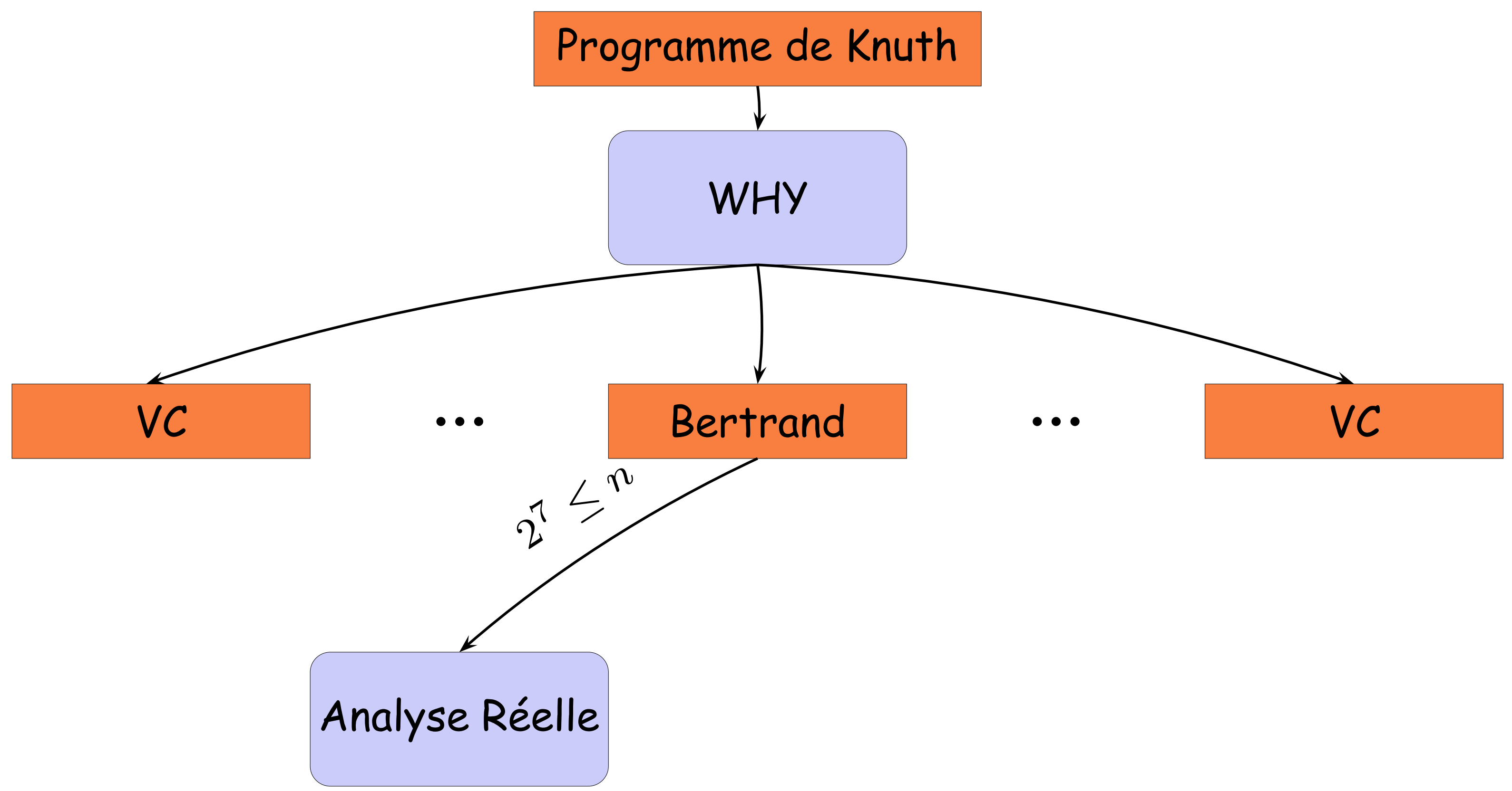
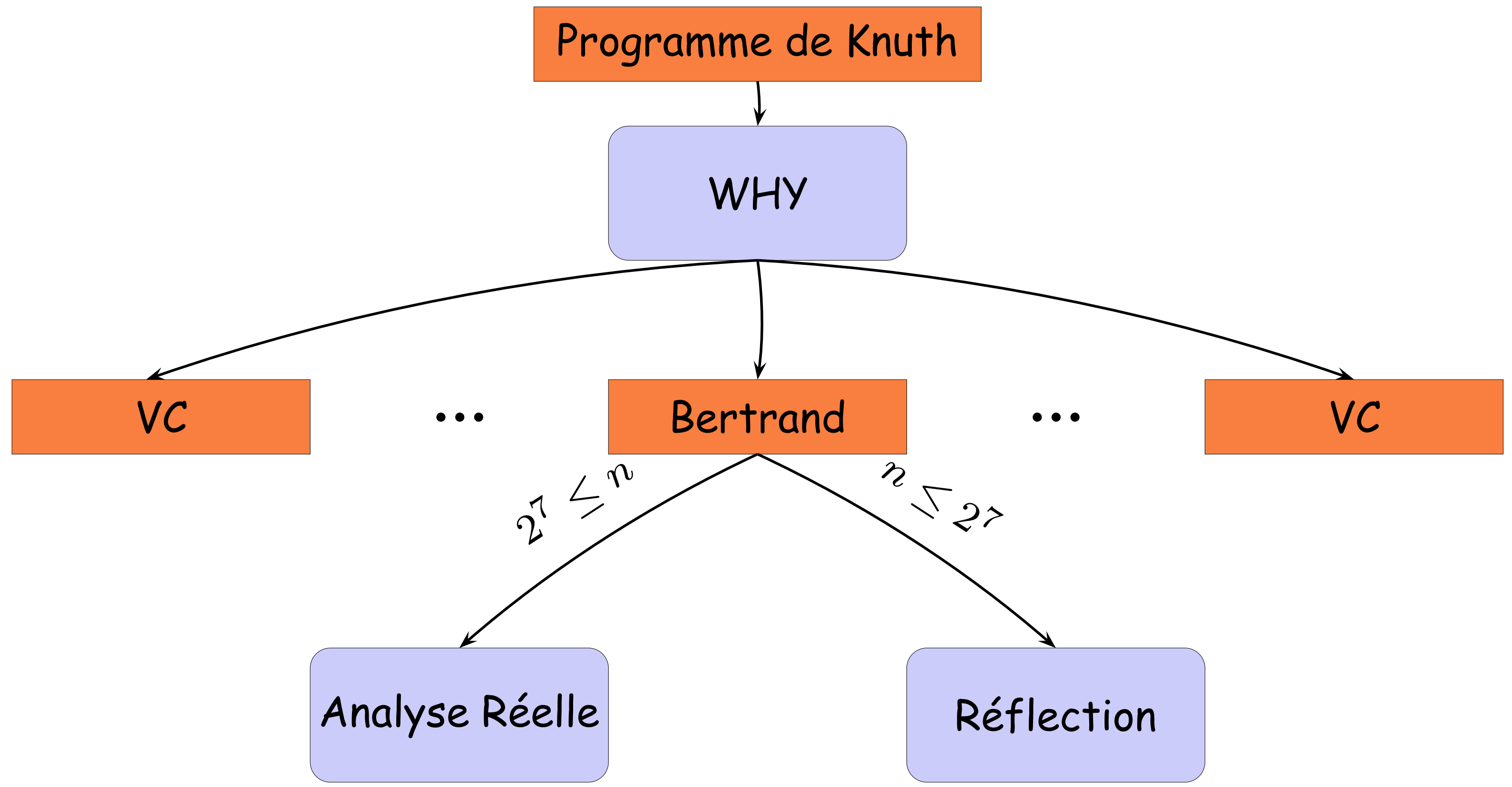


Schéma de la Preuve



Conclusions

Conclusions

Joli Exemple

Conclusions

Joli Exemple

Généricité de Coq

Conclusions

Joli Exemple

Généricité de Coq

Expressivité de Coq

Conclusions

Joli Exemple

Généricité de Coq

Expressivité de Coq

Maturité de Coq

Conclusions

Joli Exemple

Généricité de Coq

Expressivité de Coq

Maturité de Coq

Intérêts

Conclusions

Joli Exemple

Généricité de Coq

Expressivité de Coq

Maturité de Coq

Intérêts

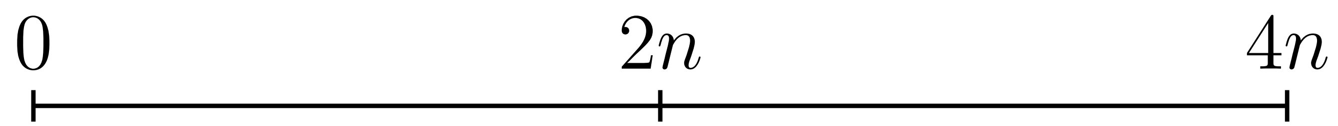
Prochain challenge?

Petit Problème

Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?

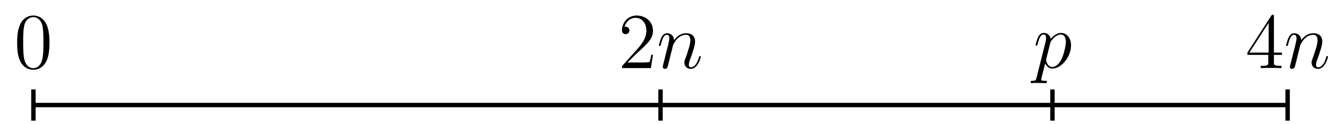
Petit Problème

Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?



Petit Problème

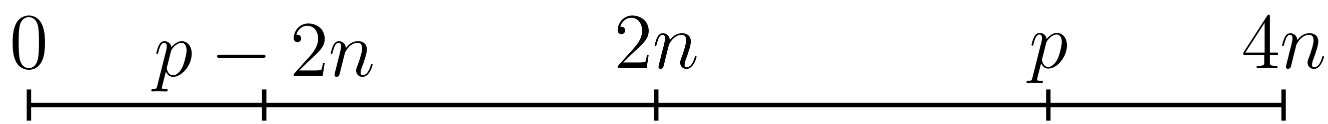
Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?



p premier

Petit Problème

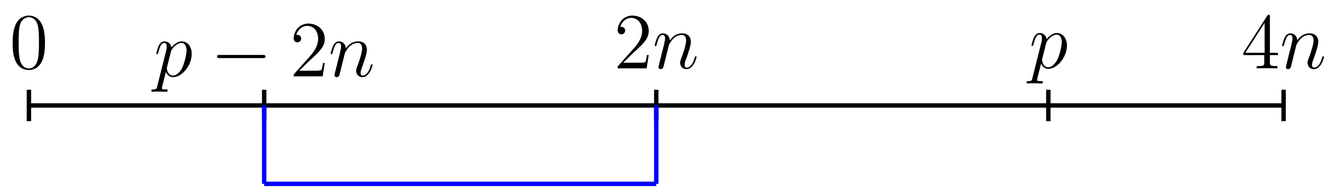
Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?



p premier

Petit Problème

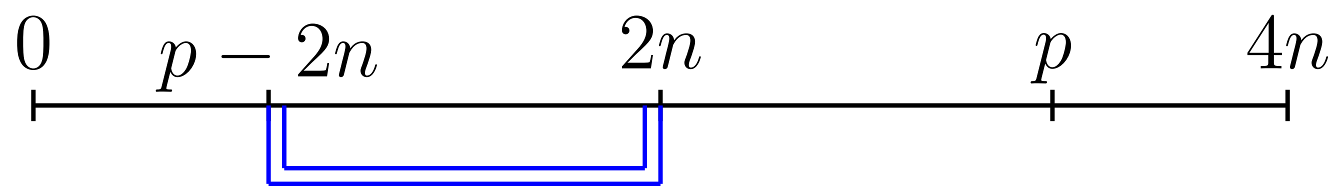
Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?



p premier

Petit Problème

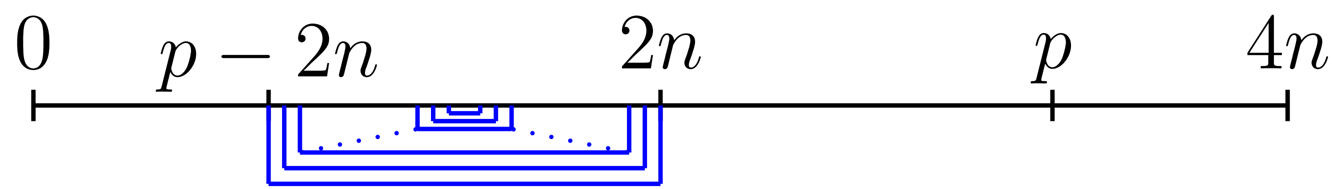
Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?



p premier

Petit Problème

Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?

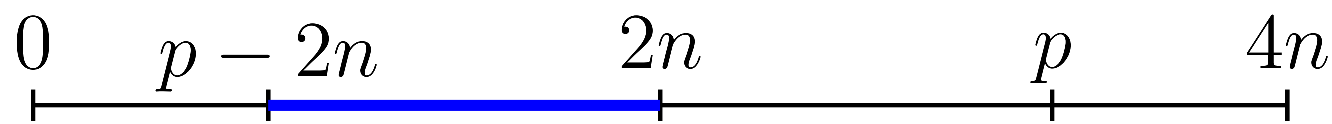


p premier

$p-2n$ impair

Petit Problème

Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?

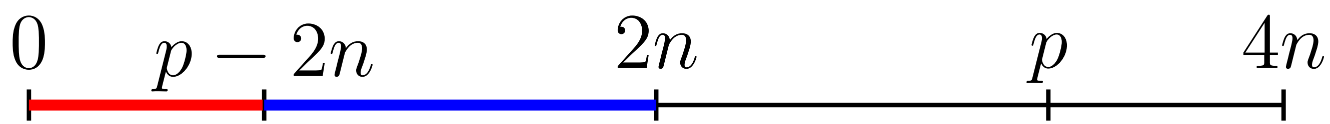


p premier

$p - 2n$ impair

Petit Problème

Partitionner les entiers de 1 à $2n$ en pairs (a_i, b_i) telles que $a_i + b_i$ est premier?



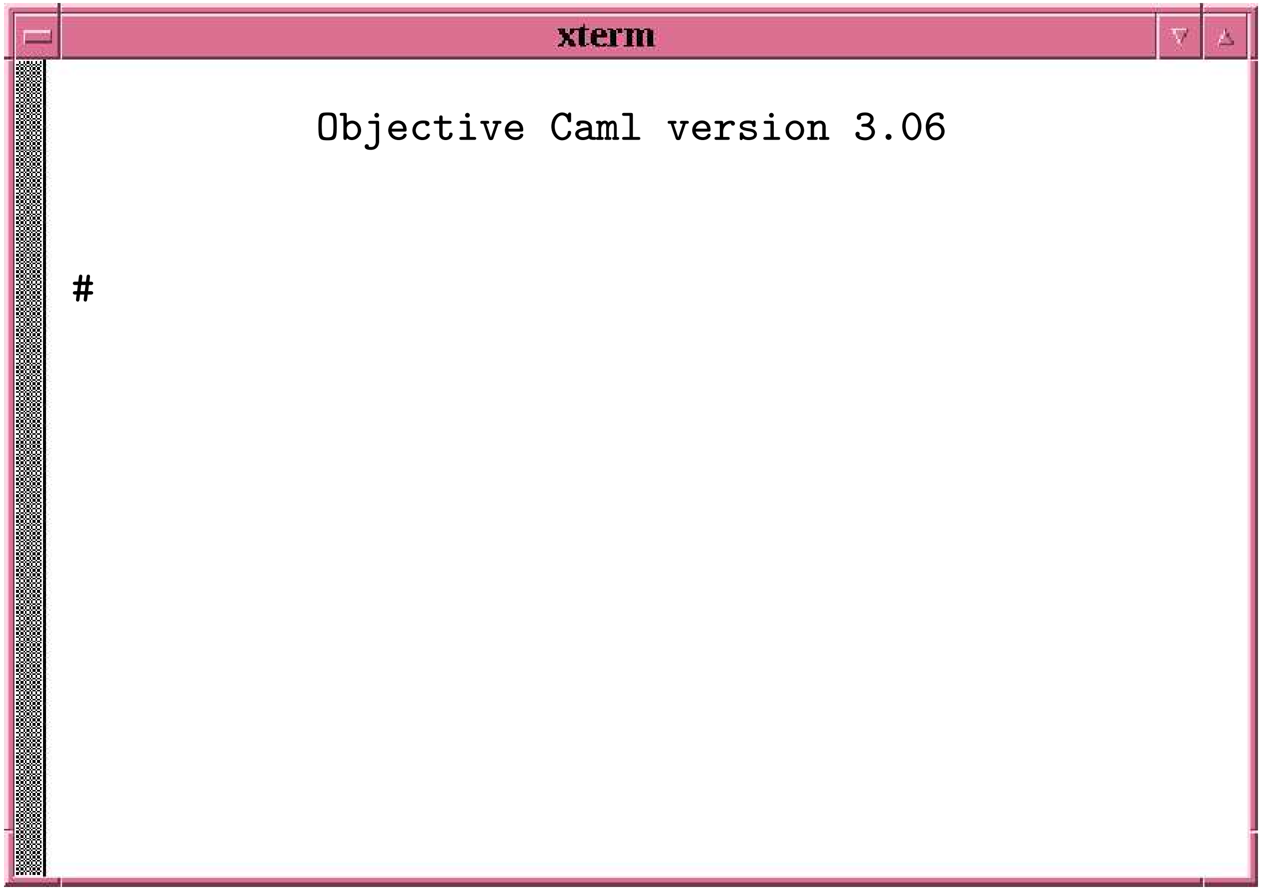
p premier

$p - 2n$ impair

$p - 2n - 1$ pair

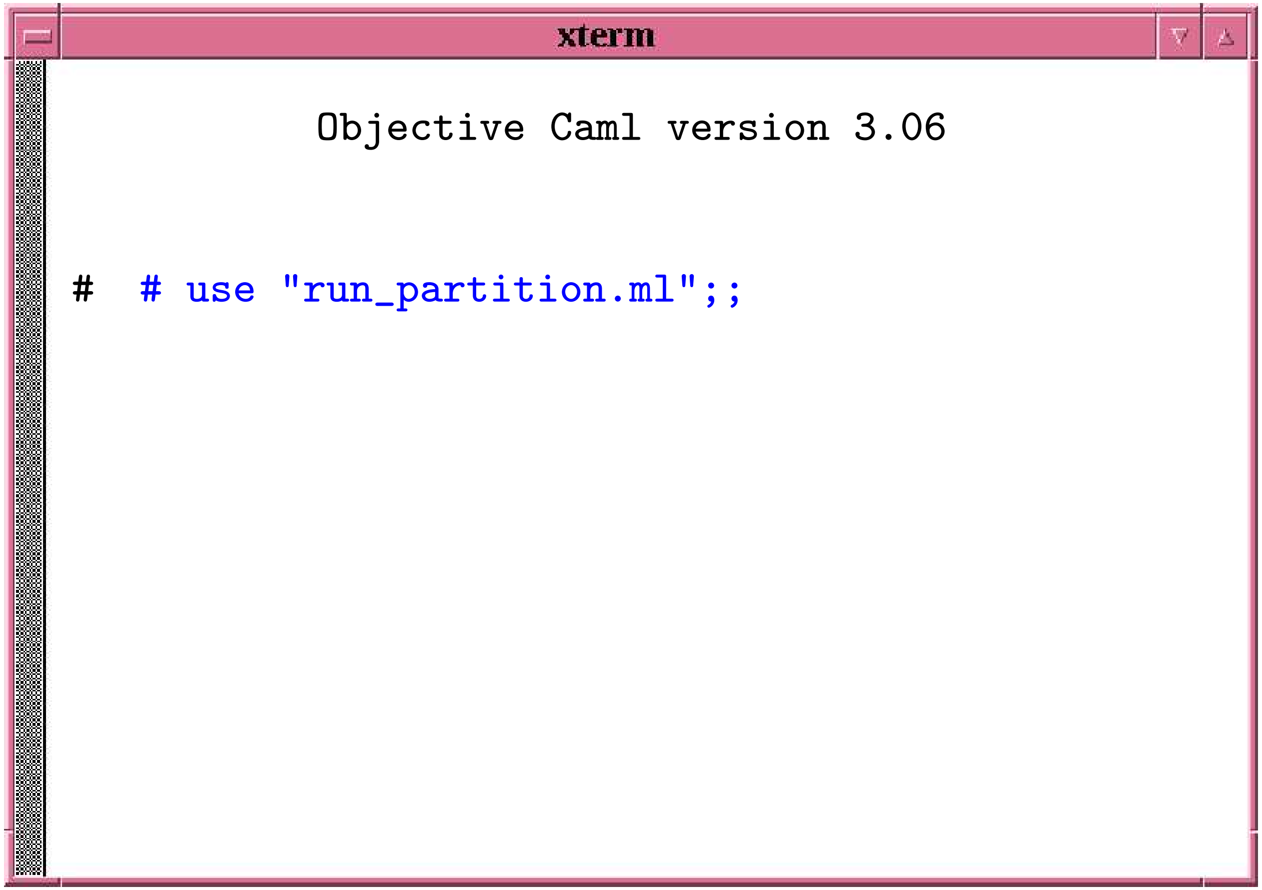
Petit Programme

Programme OCaml certifié:



Petit Programme

Programme OCaml certifié:



Petit Programme

Programme OCaml certifié:

```
xterm

        Objective Caml version 3.06


#  # use "run_partition.ml";;
val part :  int -> (int * int) list = <fun>
#
```

Petit Programme

Programme OCaml certifié:

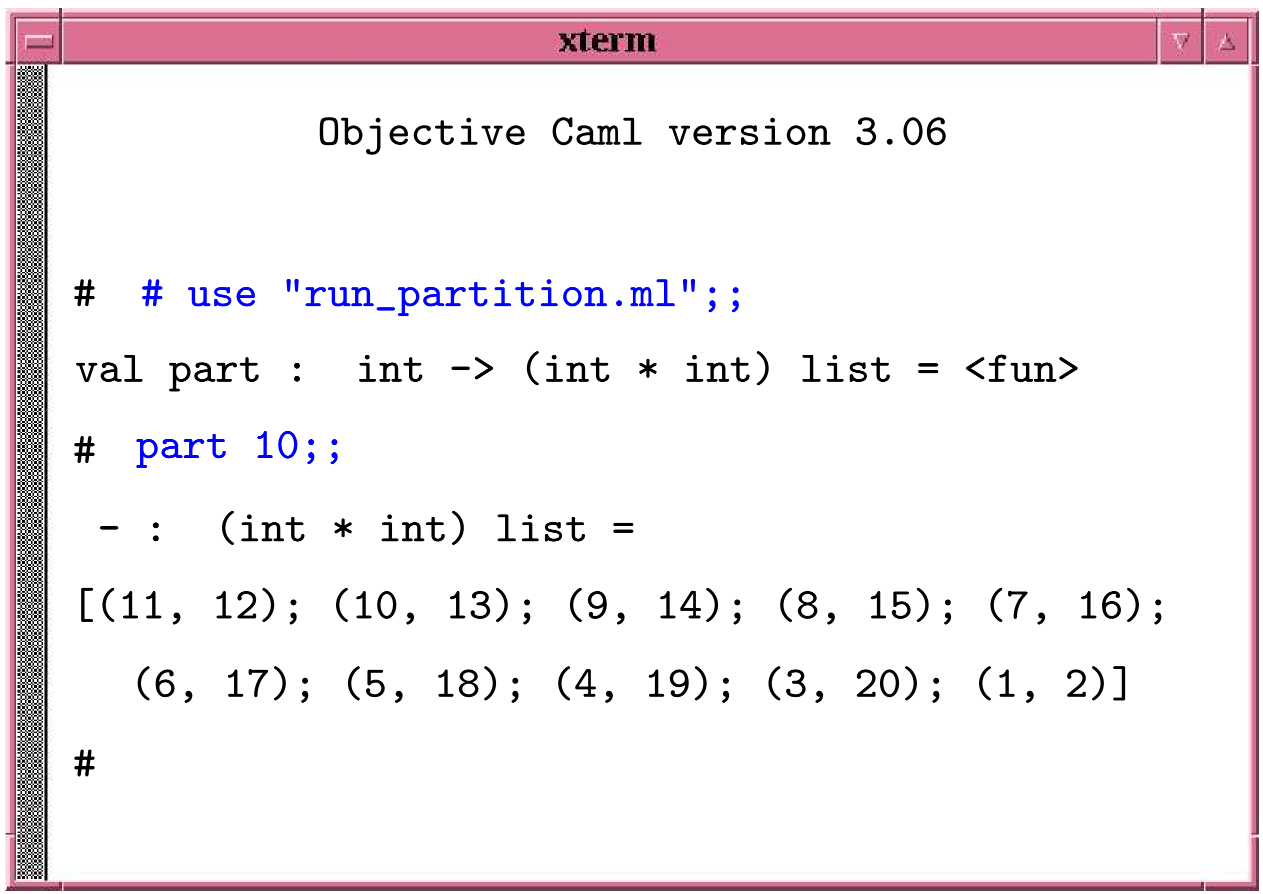
```
xterm

                    Objective Caml version 3.06


#  # use "run_partition.ml";;
val part :  int -> (int * int) list = <fun>
#  part 10;;
```

Petit Programme

Programme OCaml certifié:



```
                Objective Caml version 3.06


#  # use "run_partition.ml";;
val part :  int -> (int * int) list = <fun>
#  part 10;;
 - :  (int * int) list =
[(11, 12); (10, 13); (9, 14); (8, 15); (7, 16);
  (6, 17); (5, 18); (4, 19); (3, 20); (1, 2)]
#
```