

IFT2105 - Introduction à l'informatique théorique

Brins de complexité du calcul (Sipser chap. 7)

Pierre McKenzie et Alizée Gagnon (pour Geňa Hahn)

DIRO, Université de Montréal

Avril 2018

Définition (Sipser 7.1)

Le *temps de calcul* de M est une fonction

$$f : \mathbb{N} \rightarrow \mathbb{N}$$

$$n \mapsto \max_{|w|=n} [\text{temps de calcul de } M \text{ sur } w].$$

Définition (Sipser 7.7)

$\text{TIME}(t(n)) = \{ L \mid \text{une } mT \text{ déterministe décide } L \text{ en temps } O(t(n)) \}.$

Définition (Sipser 7.12)

La *classe de complexité* P est

$$P = \bigcup_{k \in \mathbb{N}} \text{TIME}(n^k).$$

Exemple

Le langage **PATH**, soit

$$\left\{ \langle G, s, t \rangle \mid \text{un chemin de } s \text{ à } t \text{ existe dans le graphe} \right.$$

orienté G encodé sous forme de matrice d'adjacence

$\in P$.

Précisément,

$$\langle G, s, t \rangle = \underbrace{001010001 \cdots 0010111010101010110}_{\text{matrice d'adjacence de } G} \# \underbrace{0110100}_{\text{binaire de } s} \# \underbrace{11000101}_{\text{binaire de } t}$$

et **PATH** est le langage des $\langle G, s, t \rangle \in \{0, 1, \#\}^*$ satisfaisant la condition.

Exemple

Le langage **3-COL**, soit

$$\left\{ \text{matrice d'adjacence } \langle G \rangle \mid G \text{ est 3-coloriable} \right\}$$

$\in \text{TIME}(\simeq n \cdot 2^{\sqrt{n}})$.

Exemple

Le langage **SAT**, soit

$$\left\{ \langle \varphi \rangle \mid \varphi \text{ est une expression booléenne satisfaisable} \right\}$$

$\in \text{TIME}(\simeq \text{poly}(n) \cdot 2^n)$.

Qu'ont de spécial les polynômes ?

- P est robuste : une bande, k bandes, k têtes, et plusieurs autres modèles engendrent la même classe P .

Qu'ont de spécial les polynômes ?

- P est robuste : une bande, k bandes, k têtes, et plusieurs autres modèles engendrent la même classe P .
- Si $p(n)$ et $q(n)$ sont des polynômes alors $p(n) + q(n)$, $p(n)q(n)$ et $p(q(n))$ sont aussi des polynômes.

Qu'ont de spécial les polynômes ?

- P est robuste : une bande, k bandes, k têtes, et plusieurs autres modèles engendrent la même classe P .
- Si $p(n)$ et $q(n)$ sont des polynômes alors $p(n) + q(n)$, $p(n)q(n)$ et $p(q(n))$ sont aussi des polynômes.
- P correspond aux langages décidables en pratique en un temps réalisable.

Temps de calcul ininterrompu

Signification concrète d'un temps de calcul polynomial, sous l'hypothèse d'un ordinateur exécutant **un milliard d'opérations à la seconde** :

$t(n)$	taille 25	taille 50	taille 100	taille 200
n	0,025 μs	0,05 μs	0,1 μs	0,2 μs
n^2				

Temps de calcul ininterrompu

Signification concrète d'un temps de calcul polynomial, sous l'hypothèse d'un ordinateur exécutant **un milliard d'opérations à la seconde** :

$t(n)$	taille 25	taille 50	taille 100	taille 200
n	0,025 μs	0,05 μs	0,1 μs	0,2 μs
n^2	0,625 μs	2,5 μs	10 μs	40 μs
n^5				

Temps de calcul ininterrompu

Signification concrète d'un temps de calcul polynomial, sous l'hypothèse d'un ordinateur exécutant **un milliard d'opérations à la seconde** :

$t(n)$	taille 25	taille 50	taille 100	taille 200
n	0,025 μs	0,05 μs	0,1 μs	0,2 μs
n^2	0,625 μs	2,5 μs	10 μs	40 μs
n^5	9,5 ms	0,3 sec	10 sec	5 min
$2^{n/3}$				

Temps de calcul ininterrompu

Signification concrète d'un temps de calcul polynomial, sous l'hypothèse d'un ordinateur exécutant **un milliard d'opérations à la seconde** :

$t(n)$	taille 25	taille 50	taille 100	taille 200
n	0,025 μs	0,05 μs	0,1 μs	0,2 μs
n^2	0,625 μs	2,5 μs	10 μs	40 μs
n^5	9,5 ms	0,3 sec	10 sec	5 min
$2^{n/3}$	0,3 μs			

Temps de calcul ininterrompu

Signification concrète d'un temps de calcul polynomial, sous l'hypothèse d'un ordinateur exécutant **un milliard d'opérations à la seconde** :

$t(n)$	taille 25	taille 50	taille 100	taille 200
n	0,025 μs	0,05 μs	0,1 μs	0,2 μs
n^2	0,625 μs	2,5 μs	10 μs	40 μs
n^5	9,5 ms	0,3 sec	10 sec	5 min
$2^{n/3}$	0,3 μs	0,1 ms		

Temps de calcul ininterrompu

Signification concrète d'un temps de calcul polynomial, sous l'hypothèse d'un ordinateur exécutant **un milliard d'opérations à la seconde** :

$t(n)$	taille 25	taille 50	taille 100	taille 200
n	0,025 μs	0,05 μs	0,1 μs	0,2 μs
n^2	0,625 μs	2,5 μs	10 μs	40 μs
n^5	9,5 ms	0,3 sec	10 sec	5 min
$2^{n/3}$	0,3 μs	0,1 ms	10 sec	

Temps de calcul ininterrompu

Signification concrète d'un temps de calcul polynomial, sous l'hypothèse d'un ordinateur exécutant **un milliard d'opérations à la seconde** :

$t(n)$	taille 25	taille 50	taille 100	taille 200
n	0,025 μs	0,05 μs	0,1 μs	0,2 μs
n^2	0,625 μs	2,5 μs	10 μs	40 μs
n^5	9,5 ms	0,3 sec	10 sec	5 min
$2^{n/3}$	0,3 μs	0,1 ms	10 sec	37

Temps de calcul ininterrompu

Signification concrète d'un temps de calcul polynomial, sous l'hypothèse d'un ordinateur exécutant **un milliard d'opérations à la seconde** :

$t(n)$	taille 25	taille 50	taille 100	taille 200
n	0,025 μs	0,05 μs	0,1 μs	0,2 μs
n^2	0,625 μs	2,5 μs	10 μs	40 μs
n^5	9,5 ms	0,3 sec	10 sec	5 min
$2^{n/3}$	0,3 μs	0,1 ms	10 sec	37 siècles
2^n	33 μs	13		

Temps de calcul ininterrompu

Signification concrète d'un temps de calcul polynomial, sous l'hypothèse d'un ordinateur exécutant **un milliard d'opérations à la seconde** :

$t(n)$	taille 25	taille 50	taille 100	taille 200
n	0,025 μs	0,05 μs	0,1 μs	0,2 μs
n^2	0,625 μs	2,5 μs	10 μs	40 μs
n^5	9,5 ms	0,3 sec	10 sec	5 min
$2^{n/3}$	0,3 μs	0,1 ms	10 sec	37 siècles
2^n	33 μs	13 jrs	10^{11} siècles	

Temps de calcul ininterrompu

Signification concrète d'un temps de calcul polynomial, sous l'hypothèse d'un ordinateur exécutant **un milliard d'opérations à la seconde** :

$t(n)$	taille 25	taille 50	taille 100	taille 200
n	0,025 μs	0,05 μs	0,1 μs	0,2 μs
n^2	0,625 μs	2,5 μs	10 μs	40 μs
n^5	9,5 ms	0,3 sec	10 sec	5 min
$2^{n/3}$	0,3 μs	0,1 ms	10 sec	37 siècles
2^n	33 μs	13 jrs	10^{11} siècles	10^{41} siècles

L'ordinateur est démuni, voire peu utile, face à un problème dont les seuls algorithmes connus nécessitent un temps exponentiel !

Rechercher le temps polynomial !

NB : polynomial en fonction de quoi ?

Toujours en fonction de la **longueur de l'entrée**, que l'on appelle habituellement n .

NB : polynomial en fonction de quoi ?

Toujours en fonction de la **longueur de l'entrée**, que l'on appelle habituellement n .

Ex : $a \geq 0$ entier :

longueur de $\langle a \rangle$ est $n \simeq \log_2 a$.

Ainsi un temps de calcul de l'ordre de a sur entrée un nombre naturel a est **exponentiel** car $n = |\langle a \rangle| \simeq \log_2 a$ et ainsi $a \simeq 2^n$.

Dans **P** ou pas ?

Est-ce que **3-COL** $\in \text{TIME}(n \cdot 2^{\sqrt{n}})$ implique **3-COL** $\notin P$?

Est-ce que **SAT** $\in P$?

Un outil inspiré de l'indécidabilité

Rappel : La **réduction** " \leq " nous permettait de comparer la "difficulté" de langages comme L_d , A_{TM} , $VIDE_{GHC}$, etc.

On avait : $A \leq B$ et B décidable $\implies A$ décidable.

On aimerait : $A \preceq B$ et $B \in P \implies A \in P$.

Comment définir " \preceq " ?

En limitant le temps pris par Dji Dieng :

Définition (Sipser 7.29)

Le langage A se réduit polynomialement au langage B , noté $A \leq_m^p B$ si

$$A \leq B$$

à l'aide d'une fonction

$$f : \Sigma^* \rightarrow \Sigma^*$$

calculable en temps polynomial.

Théorème (Sipser 7.31, qu'on pourrait appeler "du Klingon")

Si $A \leq_m^P B$ et $B \in P$ alors $A \in P$.

NB : Ici $A \leq_m^P B$ essentiel car $A \leq B$ ne suffirait pas.

Justification du théorème : au tableau.

Exemple de réduction polynomiale

On ne sait pas si 3-COL est dans P.

On peut quand même démontrer :

Proposition

$$3\text{-COL} \leq_m^P 4\text{-COL}.$$

Au tableau.

Usage typique de la réductibilité polynomiale

Proposition

Si 4-COL est dans P alors 3-COL est dans P.

Preuve :

Usage typique de la réductibilité polynomiale

Proposition

Si 4-COL est dans P alors 3-COL est dans P.

Preuve :

Appliquer le théorème du Klingon à la réduction $3\text{-COL} \leq_m^P 4\text{-COL}$.

Incidentement, est-ce que $4\text{-COL} \leq_m^p 3\text{-COL}$?

Définition (Sipser 7.18)

Un *vérificateur polynomial* pour un langage A est une machine de Turing déterministe \mathcal{V} telle que

il existe un polynôme p ayant la propriété que *pour tout* $w \in \Sigma^*$:

① si $w \in A$ alors

il existe c tel que $\langle w, c \rangle \in L(\mathcal{V})$

Définition (Sipser 7.18)

Un *vérificateur polynomial* pour un langage A est une machine de Turing déterministe \mathcal{V} telle que

il existe un polynôme p ayant la propriété que *pour tout* $w \in \Sigma^*$:

① si $w \in A$ alors

il existe c tel que $\langle w, c \rangle \in L(\mathcal{V})$

② si $w \notin A$ alors

pour tout c , $\langle w, c \rangle \notin L(\mathcal{V})$

Définition (Sipser 7.18)

Un *vérificateur polynomial* pour un langage A est une machine de Turing déterministe \mathcal{V} telle que

il existe un polynôme p ayant la propriété que *pour tout* $w \in \Sigma^*$:

① si $w \in A$ alors

il existe c tel que $\langle w, c \rangle \in L(\mathcal{V})$

② si $w \notin A$ alors

pour tout c , $\langle w, c \rangle \notin L(\mathcal{V})$

③ le temps de calcul de \mathcal{V} sur $\langle w, c \rangle$ est au plus $p(|w|)$.

Note. Un c tel que $\langle w, c \rangle \in L(\mathcal{V})$ est appelé *certificat* ou *preuve* ou *témoin* de l'appartenance de w au langage A .

Remarques :

- Pas nécessaire de **calculer** le certificat, il suffit que celui-ci **existe** !

Remarques :

- Pas nécessaire de **calculer** le certificat, il suffit que celui-ci **existe** !
- Le temps polynomial du vérificateur est en fonction de $|\langle w \rangle|$ et non pas de $|\langle w, c \rangle|$.

Définition (Sipser 7.19)

La *classe de complexité* NP est

$\{ \text{langage } L : L \text{ possède un } \textit{vérificateur polynomial} \}$

Exemple

Théorème

Le langage 4-COL est dans NP.

Exemple

Théorème

Le langage 4-COL est dans NP.

Au tableau.

Définition

Le langage B est **NP-ardu** (“ardu” = “difficile”) si

- pour *tout* langage $A \in \text{NP}$, $A \leq_m^P B$.

Définition

Le langage B est **NP-complet** si en plus d'être **NP-ardu**,

- $B \in \text{NP}$.

Faits :

- Si B est NP-ardu et $B \in P$, alors $P = \text{NP}$
 - ▶ a fortiori, si B est NP-complet et $B \in P$, alors $P = \text{NP}$.
- Si $A \leq_m^P B$ et $B \in \text{NP}$ alors $A \in \text{NP}$.

- La “magie” est que si l’on résout **un seul** problème NP-complet (ou NP-ardu) efficacement (en temps polynomial), alors on aura résolu efficacement **tous** les problèmes de NP.
- Le langage

$\{ \langle M, w, \underbrace{1^n}_{n \text{ en unaire}} \rangle : \text{la MT non déterministe } M \text{ accepte } w \text{ et}$

il existe une branche de calcul qui mène à l’acceptation en au plus n transitions}

est NP-ardu. Est-il NP-complet ? (Cf. votre devoir)

$P = NP ?$

Intuitivement parlant :

- $P \approx$ l'ensemble des langages **décidables** efficacement.
- $NP \approx$ l'ensemble des langages dont un témoin de l'appartenance est **vérifiable** efficacement.

$P = NP ?$

Intuitivement parlant :

- $P \approx$ l'ensemble des langages **décidables** efficacement.
- $NP \approx$ l'ensemble des langages dont un témoin de l'appartenance est **vérifiable** efficacement.

“ $P = NP ?$ ” est l'un sept “problèmes mathématiques du millénaire” ;
passez réclamer votre prix de 1 000 000 \$US si vous résolvez cette
question : [Millenium prizes](#)

Qu'apprend-on d'utile si l'on apprend que $A \in \text{NP}$?

Qu'apprend-on d'utile si l'on apprend que $A \in \text{NP}$?

Théorème

$$\text{NP} \subseteq \text{EXPTIME}$$

où

$$\text{EXPTIME} = \bigcup_{k \geq 0} \text{TIME}(2^{n^k}).$$

Au tableau.

Et qu'apprend-on d'utile si l'on apprend que A est NP-ardu ?

Et qu'apprend-on d'utile si l'on apprend que A est NP-ardu ?

Que tous les langages de NP se réduisent à A , donc, intuitivement, que A est aussi difficile à décider que les pires langages de NP.

Pourquoi NP s'appelle NP

On peut définir une notion de temps de calcul d'une MT **non** déterministe (comme dans Sipser 7.9).

La classe **NP** fut d'abord définie en ces termes et le nom est resté :

NP = **N**ondeterministic **P**olynomial Time.

Si **B** est NP-complet alors

- 1 **B** n'est **pas trop** difficile (il est dans NP, donc dans EXPTIME, alors que ça aurait pu être pire, par exemple doublement exponentiel),
- 2 **B** est **au moins aussi** difficile que n'importe quel langage de NP (tout autre langage de NP se réduit à lui).

Théorème de Cook et Levin

Belle théorie, mais **existe-t-il** des langages NP-complets ?

Théorème de Cook et Levin

Belle théorie, mais **existe-t-il** des langages NP-complets ?

Possiblement (selon votre devoir) le langage

$\{ \langle M, w, \underbrace{1^n}_{n \text{ en unaire}} \rangle : \text{la MT non déterministe } M \text{ accepte } w \text{ et}$
il existe une branche de calcul qui mène à
l'acceptation en au plus n transitions.}

Mais **existe-t-il** des langages NP-complets **naturels** ?

Théorème (Sipser 7.37, théorème de Cook-Levin)

*Le langage **SAT** est NP-complet.*

- SATFNC
- 3-SAT
- HAMPATH = $\{ \langle G, s, t \rangle \mid \text{un chemin existe de } s \text{ à } t \text{ dans le graphe orienté } G \text{ rencontrant une et une seule fois chaque sommet} \}$.
- CLIQUE = $\{ \langle G, k \rangle \mid G \text{ contient un sous-graphe complet de taille } k \}$.
- SUBSET-SUM = $\{ \langle \{x_1, \dots, x_m\}, t \rangle \mid \text{les } x_i \text{ et } t \text{ sont des entiers, et } \exists \{y_1, \dots, y_l\} \subseteq \{x_1, \dots, x_m\} \text{ tel que } \sum_{i=1}^l y_i = t \}$
- STABLE = $\{ \langle G, k \rangle \mid \text{le graphe non orienté } G \text{ possède au moins } k \text{ sommets qu'aucune arête ne relie entre eux} \}$
- COUVERTURE = $\{ \langle G, s \rangle \mid \text{il existe un ensemble de } s \text{ sommets qui "couvre" le graphe non orienté } G \}$.

Preuve de Cook-Levin

Nous savons que SAT \in NP.

Reste à montrer : SAT est NP-ardu.

Soit donc A un langage **quelconque** de la classe NP.

À montrer : $A \leq_m^p \text{SAT}$

(20 prochains transparents)

Supposons momentanément que $A \in P$, accepté par

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$$

en temps $p(n)$.

Assertion

On peut construire, sur entrée w , une expression booléenne ϕ dont la satisfaisabilité (le cas échéant) exprime

que le tableau des configurations successives de la machine M sur entrée w mène M à son état acceptant.

Comment faire cela ?

	1	2	3	4	...		j		$p(n) + 3$
1	#	q_0	w_1	w_2	...	\square	\square	\square	#

	1	2	3	4	...	j	$p(n) + 3$
1	#	q_0	w_1	w_2	...	\square	\square	...	#
2	#	d	q_4	w_2	...	\square	\square	...	#

	1	2	3	4	...	j	$p(n) + 3$
1	#	q_0	w_1	w_2	...	□	□	...	#
2	#	d	q_4	w_2	...	□	□	...	#
3	#	d	c	q_{67}	...	□	□	...	#

	1	2	3	4	...	j	$p(n) + 3$
1	#	q_0	w_1	w_2	...	\sqcup	\sqcup	...	#
2	#	d	q_4	w_2	...	\sqcup	\sqcup	...	#
3	#	d	c	q_{67}	...	\sqcup	\sqcup	...	#
4	#	d	q_{31}	c	...	\sqcup	\sqcup	...	#

	1	2	3	4	...	j		$p(n) + 3$	
1	#	q_0	w_1	w_2	...	\sqcup	\sqcup	\sqcup	#
2	#	d	q_4	w_2	...	\sqcup	\sqcup	\sqcup	#
3	#	d	c	q_{67}	...	\sqcup	\sqcup	\sqcup	#
4	#	d	q_{31}	c	...	\sqcup	\sqcup	\sqcup	#
\vdots	#	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	#
\vdots	#	c	a	...	b	b	q_{52}	c	...	a	#
\vdots	#	c	a	...	b	q_8	b	d	...	a	#
t	#	c	a	...	b	f	q_{12}	d	...	a	#
	#	c	a	...	b	f	g	q_7	...	a	#
\vdots	#	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	#
\vdots	#	c	q_9	d	#
$p(n)$	#	q_a	c	a	#

- Utilisons dans ϕ les variables booléennes :

- Utilisons dans ϕ les variables booléennes :

$$\{ x_{t,j,\sigma} \mid 1 \leq t \leq p(n), 1 \leq j \leq p(n) + 3, \\ \sigma \in Q \cup \Gamma \cup \{\#\} \}.$$

- Utilisons dans ϕ les variables booléennes :

$$\{ x_{t,j,\sigma} \mid 1 \leq t \leq p(n), 1 \leq j \leq p(n) + 3, \\ \sigma \in Q \cup \Gamma \cup \{\#\} \}.$$

- Signification de $x_{t,j,\sigma}$:

- Utilisons dans ϕ les variables booléennes :

$$\{ x_{t,j,\sigma} \mid 1 \leq t \leq p(n), 1 \leq j \leq p(n) + 3, \\ \sigma \in Q \cup \Gamma \cup \{\#\} \}.$$

- Signification de $x_{t,j,\sigma}$:

une affectation satisfaisant ϕ où $x_{t,j,\sigma} = \text{VRAI}$ décrira un tableau dont la case (t, j) est σ

- Utilisons dans ϕ les variables booléennes :

$$\{ x_{t,j,\sigma} \mid 1 \leq t \leq p(n), 1 \leq j \leq p(n) + 3, \\ \sigma \in Q \cup \Gamma \cup \{\#\} \}.$$

- Signification de $x_{t,j,\sigma}$:

une affectation satisfaisant ϕ où $x_{t,j,\sigma} = \text{VRAI}$ décrira un tableau dont la case (t, j) est σ

une affectation satisfaisant ϕ où $x_{t,j,\sigma} = \text{FAUX}$ décrira un tableau dont la case (t, j) n'est pas σ

Construction :

$$w \mapsto \phi = \phi_{\text{case}} \wedge \phi_{\text{initiale}} \wedge \phi_{\text{accepte}} \wedge \phi_{\text{légale}}$$

où la satisfaction de

- ① ... ϕ_{case} assure qu'à chaque case du tableau est affecté un unique σ
- ② ... ϕ_{initiale} assure que la ligne 1 du tableau est la configuration initiale de M sur w
- ③ ... ϕ_{accepte} assure que la dernière ligne est une configuration acceptante de M
- ④ ... $\phi_{\text{légale}}$ assure que chaque ligne suit la précédente selon l'unique action légale de M .

ϕ_{case} : un unique symbole par case

ϕ_{case} : un unique symbole par case

Pour chaque $1 \leq t \leq p(n)$
et chaque $1 \leq j \leq p(n) + 3$
inclure dans ϕ_{case} :

ϕ_{case} : un unique symbole par case

Pour chaque $1 \leq t \leq p(n)$
et chaque $1 \leq j \leq p(n) + 3$
inclure dans ϕ_{case} :

$$\begin{aligned} & (x_{t,j,q_0} \vee x_{t,j,q_1} \vee \cdots \vee x_{t,j,a} \vee x_{t,j,b} \vee \cdots \vee x_{t,j,\#}) \\ & \quad \wedge \end{aligned}$$

$$\begin{aligned}
& \neg [(x_{t,j,q_0} \wedge x_{t,j,q_1}) \vee \\
& \quad (x_{t,j,q_0} \wedge x_{t,j,q_2}) \vee \\
& \quad \vdots \\
& \quad (x_{t,j,q_0} \wedge x_{t,j,a}) \vee \\
& \quad \vdots \\
& \quad (x_{t,j,q_0} \wedge x_{t,j,\#}) \vee \\
& \quad (x_{t,j,q_1} \wedge x_{t,j,q_2}) \vee \\
& \quad (x_{t,j,q_1} \wedge x_{t,j,q_3}) \vee \\
& \quad (x_{t,j,q_1} \wedge x_{t,j,\#}) \vee \\
& \quad \vdots \\
& \quad (x_{t,j,a} \wedge x_{t,j,b}) \vee \\
& \quad \vdots]
\end{aligned}$$

ϕ_{initiale} : première ligne = config initiale sur w

ϕ_{initiale} : première ligne = config initiale sur w

$x_{1,1,\#} \wedge$

$x_{1,2,q_0} \wedge$

$x_{1,3,w_1} \wedge$

$x_{1,4,w_2} \wedge$

\vdots

$x_{1,n+2,w_n} \wedge$

$x_{1,n+3,\sqcup} \wedge$

\vdots

$x_{1,p(n)+2,\sqcup} \wedge$

$x_{1,p(n)+3,\#}$

ϕ_{accepte} : dernière ligne = config acceptante

$$\begin{array}{c} x_{p(n),2,q_a} \\ \vee \\ x_{p(n),3,q_a} \\ \vee \\ x_{p(n),4,q_a} \\ \vee \\ \vdots \\ \vee \\ x_{p(n),p(n)+2,q_a} \end{array}$$

$\phi_{\text{légal}}$: suite légale de configurations

Pour chaque $1 \leq t < p(n)$ et $1 < j < p(n) + 3$,

inclure dans $\phi_{\text{légal}}$ un \vee de chaque

A	B	C
α	β	γ

possible, représenté par

$$(x_{t,j-1,A} \wedge x_{t,j,B} \wedge x_{t,j+1,C}$$

$$\wedge$$

$$x_{t+1,j-1,\alpha} \wedge x_{t+1,j,\beta} \wedge x_{t+1,j+1,\gamma})$$

- $w \mapsto \phi = \phi_{\text{case}} \wedge \phi_{\text{initiale}} \wedge \phi_{\text{accepte}} \wedge \phi_{\text{légale}}$
se construit en temps $\text{poly}(p(n))$

- $w \mapsto \phi = \phi_{\text{case}} \wedge \phi_{\text{initiale}} \wedge \phi_{\text{accepte}} \wedge \phi_{\text{légale}}$
se construit en temps $\text{poly}(p(n))$
- $w \in A \Rightarrow$ une suite de configs mène M sur entrée w à l'état $q_a \Rightarrow$

- $w \mapsto \phi = \phi_{\text{case}} \wedge \phi_{\text{initiale}} \wedge \phi_{\text{accepte}} \wedge \phi_{\text{légale}}$
se construit en temps $\text{poly}(p(n))$
- $w \in A \Rightarrow$ une suite de configs mène M sur entrée w à l'état $q_a \Rightarrow$ le tableau résultant prescrit une affectation satisfaisant ϕ
 \Rightarrow

- $w \mapsto \phi = \phi_{\text{case}} \wedge \phi_{\text{initiale}} \wedge \phi_{\text{accepte}} \wedge \phi_{\text{légal}}$
se construit en temps $\text{poly}(p(n))$
- $w \in A \Rightarrow$ une suite de configs mène M sur entrée w à l'état $q_a \Rightarrow$ le tableau résultant prescrit une affectation satisfaisant ϕ
 $\Rightarrow \phi \in \text{SAT}$.

- $w \mapsto \phi = \phi_{\text{case}} \wedge \phi_{\text{initiale}} \wedge \phi_{\text{accepte}} \wedge \phi_{\text{légale}}$
se construit en temps $\text{poly}(p(n))$
- $w \in A \Rightarrow$ une suite de configs mène M sur entrée w à l'état $q_a \Rightarrow$ le tableau résultant prescrit une affectation satisfaisant ϕ
 $\Rightarrow \phi \in \text{SAT}$.
- $\phi \in \text{SAT} \Rightarrow$

- $w \mapsto \phi = \phi_{\text{case}} \wedge \phi_{\text{initiale}} \wedge \phi_{\text{accepte}} \wedge \phi_{\text{légale}}$
se construit en temps $\text{poly}(p(n))$
- $w \in A \Rightarrow$ une suite de configs mène M sur entrée w à l'état $q_a \Rightarrow$ le tableau résultant prescrit une affectation satisfaisant ϕ
 $\Rightarrow \phi \in \text{SAT}$.
- $\phi \in \text{SAT} \Rightarrow \phi$ décrit un tableau \Rightarrow

- $w \mapsto \phi = \phi_{\text{case}} \wedge \phi_{\text{initiale}} \wedge \phi_{\text{accepte}} \wedge \phi_{\text{légal}}$
se construit en temps $\text{poly}(p(n))$
- $w \in A \Rightarrow$ une suite de configs mène M sur entrée w à l'état $q_a \Rightarrow$ le tableau résultant prescrit une affectation satisfaisant ϕ
 $\Rightarrow \phi \in \text{SAT}$.
- $\phi \in \text{SAT} \Rightarrow \phi$ décrit un tableau \Rightarrow ce tableau décrit un calcul acceptant à partir de la config initiale de M sur $w \Rightarrow$

- $w \mapsto \phi = \phi_{\text{case}} \wedge \phi_{\text{initiale}} \wedge \phi_{\text{accepte}} \wedge \phi_{\text{légal}}$
se construit en temps $\text{poly}(p(n))$
- $w \in A \Rightarrow$ une suite de configs mène M sur entrée w à l'état $q_a \Rightarrow$ le tableau résultant prescrit une affectation satisfaisant $\phi \Rightarrow \phi \in \text{SAT}$.
- $\phi \in \text{SAT} \Rightarrow \phi$ décrit un tableau \Rightarrow ce tableau décrit un calcul acceptant à partir de la config initiale de M sur $w \Rightarrow w \in A$.

- $w \mapsto \phi = \phi_{\text{case}} \wedge \phi_{\text{initiale}} \wedge \phi_{\text{accepte}} \wedge \phi_{\text{légale}}$
se construit en temps $\text{poly}(p(n))$
- $w \in A \Rightarrow$ une suite de configs mène M sur entrée w à l'état $q_a \Rightarrow$ le tableau résultant prescrit une affectation satisfaisant ϕ
 $\Rightarrow \phi \in \text{SAT}$.
- $\phi \in \text{SAT} \Rightarrow \phi$ décrit un tableau \Rightarrow ce tableau décrit un calcul acceptant à partir de la config initiale de M sur $w \Rightarrow w \in A$.

Fin preuve de l'assertion.

Faisons le point. Nous avons produit ϕ dont la satisfaisabilité exprime que l'unique suite de configurations d'un M déterministe partant de

	1	2	3	4		$p(n) + 3$	
1	#	q_0	w_1	w_2	...	w_n	□	□	...	□	#

se termine par une configuration acceptante.

Que faire maintenant pour gérer la situation générale d'un A dans NP ?

M = le vérificateur \mathcal{V} de A

$p(n)$ = le temps d'exécution (polynomial) de \mathcal{V} .

Désormais :

La satisfaction de ϕ doit impliquer l'initialisation de la première ligne du tableau à

	1	2	3	4	...	$n + 2$...		$p(n) + 4$		
1	#	q_0	w_1	w_2	...	w_n	\$	c_1	c_2	...	c_m	#

où $c_1 c_2 \dots c_m$ est n'importe quelle suite de symboles susceptible de constituer un certificat

(note : $m = p(n) - n$ sans perte de généralité).

ϕ_{initiale} : ligne 1 = config initiale sur w

devient donc

ϕ_{initiale} : ligne 1 = config initiale sur $w\$c_1 \dots c_m$

comme suit :

ϕ_{initiale} auparavant :

$$\begin{aligned} & x_{1,1,\#} \wedge \\ & x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge \\ & x_{1,4,w_2} \wedge \\ & \quad \vdots \\ & x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\sqcup} \wedge \\ & \quad \vdots \\ & x_{1,p(n)+2,\sqcup} \wedge \\ & x_{1,p(n)+3,\#} \end{aligned}$$

ϕ_{initiale} maintenant :

$x_{1,1,\#} \wedge$

$x_{1,2,q_0} \wedge$

$x_{1,3,w_1} \wedge$

$x_{1,4,w_2} \wedge$

\vdots

$x_{1,n+2,w_n} \wedge$

\wedge

$$\begin{aligned}
& x_{1,n+3,\$} \wedge \\
& \left(\bigvee_{\sigma \in \Sigma} x_{1,n+4,\sigma} \right) \wedge \\
& \left(\bigvee_{\sigma \in \Sigma} x_{1,n+5,\sigma} \right) \wedge \\
& \quad \vdots \\
& \left(\bigvee_{\sigma \in \Sigma} x_{1,p(n)+3,\sigma} \right) \wedge \\
& x_{1,n+p(n)+4,\#}
\end{aligned}$$

où Σ est l'alphabet de \mathcal{V} .

La nouvelle ϕ_{initiale} force donc le remplacement des \sqcup de la ligne 1 par des symboles de Σ .

Le reste est inchangé :

$$w \mapsto \phi = \phi_{\text{case}} \wedge \phi_{\text{initiale}} \wedge \phi_{\text{accepte}} \wedge \phi_{\text{légale}}.$$

demeure calculable en temps **polynomial**.

• $w \in A \Rightarrow$

• $w \in A \Rightarrow$

▶ pour un certain $c_1 c_2 \dots c_m$, \mathcal{V} accepte $w \$ c_1 c_2 \dots c_m \Rightarrow$

• $w \in A \Rightarrow$

- ▶ pour un certain $c_1 c_2 \dots c_m$, \mathcal{V} accepte $w \$ c_1 c_2 \dots c_m \Rightarrow$
- ▶ un tableau ayant $\# w \$ c_1 c_2 \dots c_m \#$ à la ligne 1 dépeint le calcul de \mathcal{V} sur $w \$ c_1 c_2 \dots c_m \Rightarrow$

• $w \in A \Rightarrow$

- ▶ pour un certain $c_1 c_2 \dots c_m$, \mathcal{V} accepte $w \$ c_1 c_2 \dots c_m \Rightarrow$
- ▶ un tableau ayant $\# w \$ c_1 c_2 \dots c_m \#$ à la ligne 1 dépeint le calcul de \mathcal{V} sur $w \$ c_1 c_2 \dots c_m \Rightarrow$
- ▶ ce tableau prescrit une affectation satisfaisant $\phi \Rightarrow$

• $w \in A \Rightarrow$

- ▶ pour un certain $c_1 c_2 \dots c_m$, \mathcal{V} accepte $w \$ c_1 c_2 \dots c_m \Rightarrow$
- ▶ un tableau ayant $\# w \$ c_1 c_2 \dots c_m \#$ à la ligne 1 dépeint le calcul de \mathcal{V} sur $w \$ c_1 c_2 \dots c_m \Rightarrow$
- ▶ ce tableau prescrit une affectation satisfaisant $\phi \Rightarrow$
- ▶ $\phi \in \text{SAT}$.

• $w \in A \Rightarrow$

- ▶ pour un certain $c_1 c_2 \dots c_m$, \mathcal{V} accepte $w \$ c_1 c_2 \dots c_m \Rightarrow$
- ▶ un tableau ayant $\# w \$ c_1 c_2 \dots c_m \#$ à la ligne 1 dépeint le calcul de \mathcal{V} sur $w \$ c_1 c_2 \dots c_m \Rightarrow$
- ▶ ce tableau prescrit une affectation satisfaisant $\phi \Rightarrow$
- ▶ $\phi \in \text{SAT}$.

• $\phi \in \text{SAT} \Rightarrow$

• $w \in A \Rightarrow$

- ▶ pour un certain $c_1 c_2 \dots c_m$, \mathcal{V} accepte $w \$ c_1 c_2 \dots c_m \Rightarrow$
- ▶ un tableau ayant $\# w \$ c_1 c_2 \dots c_m \#$ à la ligne 1 dépeint le calcul de \mathcal{V} sur $w \$ c_1 c_2 \dots c_m \Rightarrow$
- ▶ ce tableau prescrit une affectation satisfaisant $\phi \Rightarrow$
- ▶ $\phi \in \text{SAT}$.

• $\phi \in \text{SAT} \Rightarrow$

- ▶ ϕ prescrit un tableau ayant un certain $\# w \$ c_1 c_2 \dots c_m \#$ à la ligne 1 \Rightarrow

• $w \in A \Rightarrow$

- ▶ pour un certain $c_1 c_2 \dots c_m$, \mathcal{V} accepte $w \$ c_1 c_2 \dots c_m \Rightarrow$
- ▶ un tableau ayant $\# w \$ c_1 c_2 \dots c_m \#$ à la ligne 1 dépeint le calcul de \mathcal{V} sur $w \$ c_1 c_2 \dots c_m \Rightarrow$
- ▶ ce tableau prescrit une affectation satisfaisant $\phi \Rightarrow$
- ▶ $\phi \in \text{SAT}$.

• $\phi \in \text{SAT} \Rightarrow$

- ▶ ϕ prescrit un tableau ayant un certain $\# w \$ c_1 c_2 \dots c_m \#$ à la ligne 1 \Rightarrow
- ▶ ce tableau dépeint un calcul acceptant de \mathcal{V} sur $w \$ c_1 c_2 \dots c_m \Rightarrow$

• $w \in A \Rightarrow$

- ▶ pour un certain $c_1 c_2 \dots c_m$, \mathcal{V} accepte $w \$ c_1 c_2 \dots c_m \Rightarrow$
- ▶ un tableau ayant $\# w \$ c_1 c_2 \dots c_m \#$ à la ligne 1 dépeint le calcul de \mathcal{V} sur $w \$ c_1 c_2 \dots c_m \Rightarrow$
- ▶ ce tableau prescrit une affectation satisfaisant $\phi \Rightarrow$
- ▶ $\phi \in \text{SAT}$.

• $\phi \in \text{SAT} \Rightarrow$

- ▶ ϕ prescrit un tableau ayant un certain $\# w \$ c_1 c_2 \dots c_m \#$ à la ligne 1 \Rightarrow
- ▶ ce tableau dépeint un calcul acceptant de \mathcal{V} sur $w \$ c_1 c_2 \dots c_m \Rightarrow$
- ▶ $c_1 c_2 \dots c_m$ témoigne de l'appartenance de w à $A \Rightarrow$
- ▶ $w \in A$.

Donc $A \leq_m^p \text{SAT}$, et ceci conclut la preuve de Cook-Levin.

Aussi difficile de trouver un algorithme polynomial pour SAT que de trouver du coup un algo poly pour tous les langages de NP!!!

Aussi difficile de trouver un algorithme polynomial pour SAT que de trouver du coup un algo poly pour tous les langages de NP!!!

Un algorithme polynomial pour SAT fournirait :

- un algorithme polynomial pour 3-COL
- un algorithme polynomial pour HAMPATH
- un algorithme polynomial pour CLIQUE
- etc.

Définition (FNC, forme normale conjonctive)

Une expression booléenne est en *forme normale conjonctive* si elle ressemble à

$$(* \vee *) \wedge (* \vee * \vee * \vee * \vee *) \wedge (\cdots) \wedge \dots$$

où $*$ est une variable x ou sa négation \bar{x} .

SATFNC

$\text{SATFNC} = \{ \langle \phi \rangle : \phi \text{ est une expression booléenne en FNC et } \phi \text{ est satisfaisable} \}.$

Théorème

SATFNC est NP-complet.

$\text{SATFNC} = \{ \langle \phi \rangle : \phi \text{ est une expression booléenne en FNC et } \phi \text{ est satisfaisable} \}.$

Théorème

SATFNC est NP-complet.

Preuve :

① $\text{SATFNC} \in \text{NP}$ (exercice)

② SATFNC est NP-ardu

Suffit d'appliquer De Morgan à ϕ_{case} et de distribuer les \vee sur les \wedge dans $\phi_{\text{légal}}$ de la preuve de Cook-Levin.

Doit-on refaire la preuve de Cook-Levin à chaque fois qu'on veut démontrer qu'un problème est NP-complet ?

Doit-on refaire la preuve de Cook-Levin à chaque fois qu'on veut démontrer qu'un problème est NP-complet ?

Non ! (Alizée) :

- 1 Pour prouver **B** NP-complet, il suffit de

Doit-on refaire la preuve de Cook-Levin à chaque fois qu'on veut démontrer qu'un problème est NP-complet ?

Non ! (Alizée) :

- ❶ Pour prouver B NP-complet, il suffit de
 - ❶ montrer que $B \in \text{NP}$
 - ❷ identifier un langage A tel que
 - ❶ A est NP-ardu (ou NP-complet),
 - ❷ $A \leq_m^p B$.

Doit-on refaire la preuve de Cook-Levin à chaque fois qu'on veut démontrer qu'un problème est NP-complet ?

Non ! (Alizée) :

- ① Pour prouver **B** NP-complet, il suffit de
 - ① montrer que **B** \in NP
 - ② identifier un langage **A** tel que
 - ① **A** est NP-ardu (ou NP-complet),
 - ② **A** \leq_m^p **B**.

Souvent, l'étape 2.2.1 requiert une bonne dose d'ingéniosité.

Autres problèmes NP-complets

Littéralement, des milliers de problèmes tirés de tous les domaines : combinatoire, VLSI, confection d'horaires, logique, etc. dont quelques centaines dans le classique :

M. Garey et D. Johnson, *Computers and intractability—A guide to the theory of NP-completeness*, Freeman, 1979.

Pour chacun des langages NP-complets A de cette liste :

$$A \in P \text{ ssi } P = NP.$$

Le jeu Minesweeper : préparatifs

Et maintenant un langage NP-complet très loin des expressions booléennes, mais d'abord, pause publicitaire...

- Richard Kaye, Minesweeper is NP-complete, *The Mathematical Intelligencer*, Springer Verlag, vol. 22, no. 2, 2000, 9–15.

- Richard Kaye, Minesweeper is NP-complete, *The Mathematical Intelligencer*, Springer Verlag, vol. 22, no. 2, 2000, 9–15.

“...Finally, it is nice to know that to current knowledge, there may still be an efficient algorithm for Minesweeper, and finding it could solve one of mathematics’s most important open problems.”

- Ian Stewart, **Million-Dollar Minesweeper**, *Scientific American*, vol. 283, oct. 2000, 94–95.

“The Clay Mathematics Institute, a nonprofit educational foundation in Cambridge, Mass., is offering million-dollar prizes for the solutions to seven infamous unsolved problems. One of them is the P versus NP question. [...] A clever amateur may be able to solve it with the help of Minesweeper.”

- Lisa Lipman,

<http://news.excite.com/news/ap/001102/15/minesweeper-mystery>

BOSTON (AP) - “Minesweeper, a seemingly simple game included on most personal computers, could help mathematicians crack one of the field’s most intriguing problems.”

Retour aux préparatifs

Rappel : 3SAT est NP-complet.

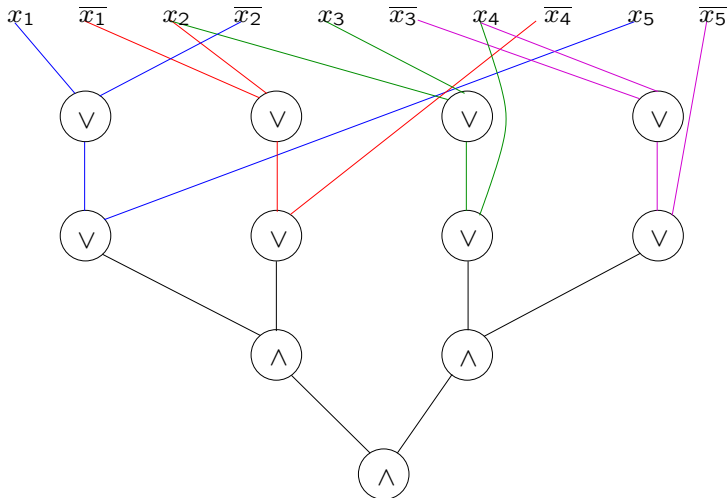
Remarques :

- On peut associer à un exemplaire de 3SAT un circuit booléen.
- On peut même supposer sans perte de généralité que ce circuit booléen ne contient pas de porte \vee .

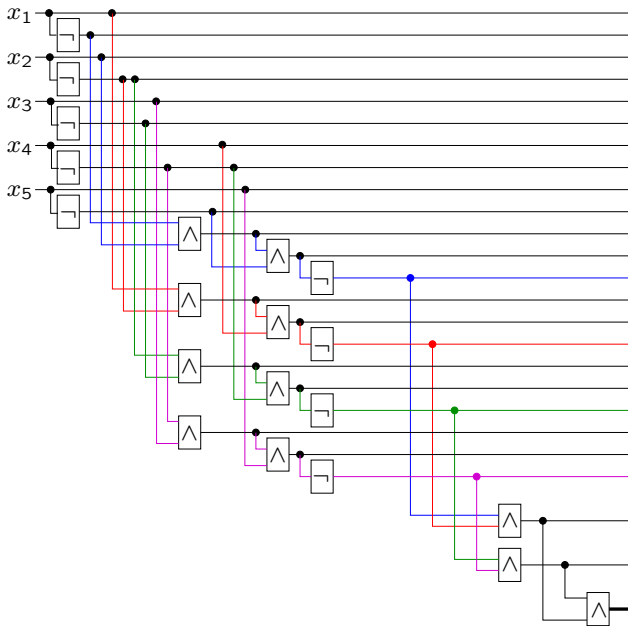
Par exemple :

$$(x_1 \vee \overline{x_2} \vee x_5) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_4}) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\overline{x_3} \vee x_4 \vee \overline{x_5})$$

donne lieu à



et au $\{x_i, \neg, \wedge\}$ -circuit équivalent (De Morgan) :



NEG-ET-SAT = $\{ \langle C \rangle : C \text{ est un } \{x_i, \neg, \wedge\}\text{-circuit et il existe une affectation aux portes } x_i \text{ qui rend VRAIE l'unique sortie de } C \}$.

Théorème

NEG-ET-SAT est NP-complet.

Preuve :

- **NEG-ET-SAT** \in NP : exercice.
- **NEG-ET-SAT** est NP-ardu : exercice
(facile si l'on choisit bien le langage NP-complet **A** dont on montrera ensuite que **A** \leq_m^p **NEG-ET-SAT**).

DÉMINEUR

Quel **problème de décision** pouvons-nous associer au jeu de démineur ??

Quel *problème de décision* pouvons-nous associer au jeu de démineur??

Problème considéré par Richard Kaye :

Donnée : tableau $m \times m$ de symboles de
 $\{ \bullet, \sqcup, 0, 1, 2, 3, 4, 5, 6, 7, 8 \}$

Question : est-il possible de remplir les cases
 \sqcup de manière à obtenir un tableau légal?

Le problème (plus intéressant :-)) que nous allons considérer :

DÉMINEUR :

Donnée : tableau $m \times m$ de symboles de $\{ \bullet, \sqcup, 0, 1, 2, 3, 4, 5, 6, 7, 8 \}$, et deux entiers $1 \leq i, j \leq m$

Question : est-il possible qu'une mine se trouve en position (i, j) ?

Le problème (plus intéressant :-)) que nous allons considérer :

DÉMINEUR :

Donnée : tableau $m \times m$ de symboles de $\{ \bullet, \sqcup, 0, 1, 2, 3, 4, 5, 6, 7, 8 \}$, et deux entiers $1 \leq i, j \leq m$

Question : est-il possible qu'une mine se trouve en position (i, j) ?

Le langage DÉMINEUR est l'ensemble des $\langle \text{tableau}, i, j \rangle$ pour lesquels la réponse à la question est oui.

Théorème (Kaye 2000)

DÉMINEUR est NP-complet.

Preuve :











- DÉMINEUR \in NP : exercice.
- NEG-ET-SAT \leq_m^p DÉMINEUR : pages suivantes.

Esquisse de $\text{NEG-ET-SAT} \leq_m^p \text{DÉMINEUR}$











L'idée : transformer le circuit C en un tableau de DÉMINEUR et se servir de fils propageant VRAI ou FAUX d'une porte du circuit à une autre :

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1			1			1			1			1	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

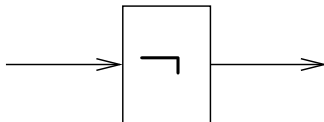
Un fil transportant la valeur VRAI :

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1			1			1			1			1	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Un fil transportant la valeur FAUX :

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1			1			1			1			1	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Transformer dans le circuit



en un bloc de cases :

				1	1	1						
1	1	1	1	2	●	2	1	1	1	1	1	...
	1			3		3			1			...
1	1	1	1	2	●	2	1	1	1	1	1	...
				1	1	1						

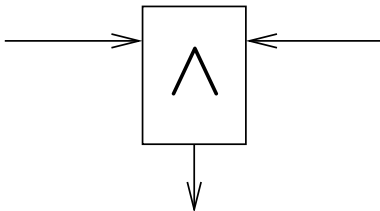
Un fil VRAI devient FAUX :

				1	1	1						
1	1	1	1	2	●	2	1	1	1	1	1	...
●	1	/	●	3	/	3	●	/	1	●	/	...
1	1	1	1	2	●	2	1	1	1	1	1	...
				1	1	1						

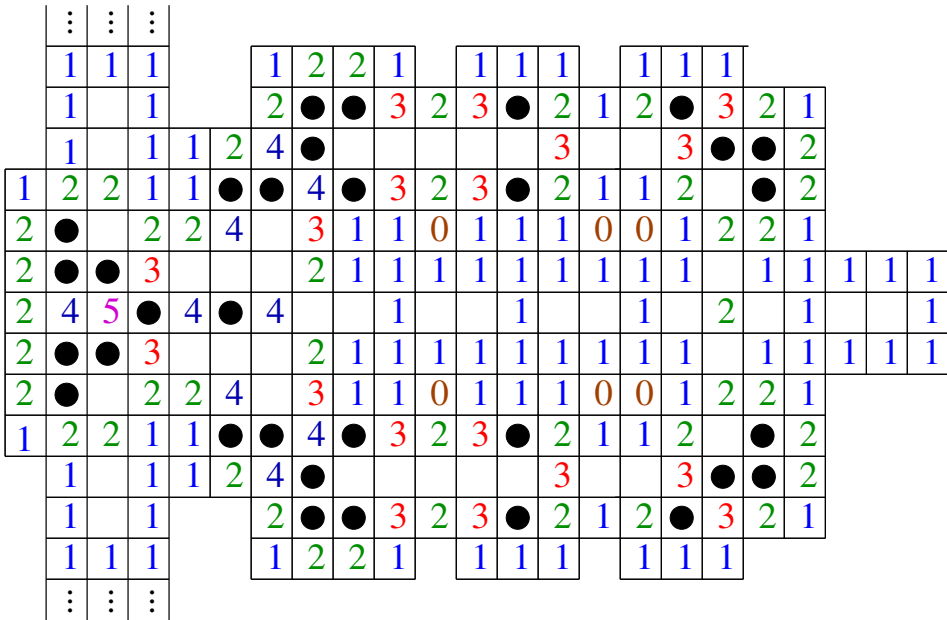
Un fil FAUX devient VRAI :

				1	1	1						
1	1	1	1	2	●	2	1	1	1	1	1	...
/	1	●	/	3	●	3	/	●	1	/	●	...
1	1	1	1	2	●	2	1	1	1	1	1	...
				1	1	1						

Transformer dans le circuit



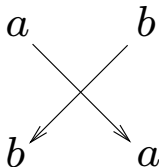
en un bloc de cases :



Que faire lorsque des fils **se croisent** dans le $\{x_i, \neg, \wedge\}$ -circuit de départ ?

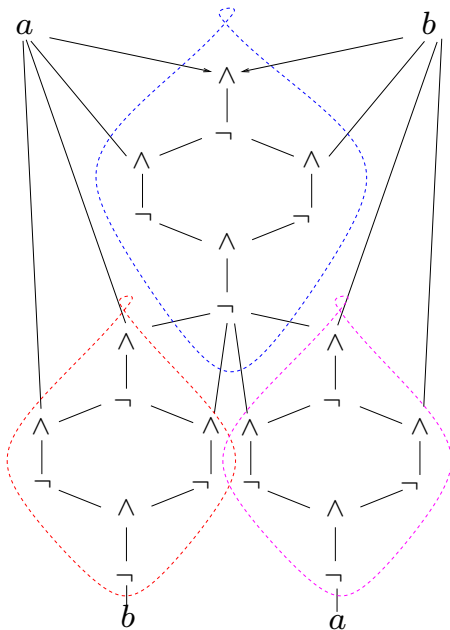
S'en débarrasser avant de commencer !

En effet, un croisement



équivalent à un $\{\neg, \wedge\}$ -circuit planaire... (huh ?)

On transforme donc d'abord le “layout” du circuit de départ pour éliminer tout croisement.

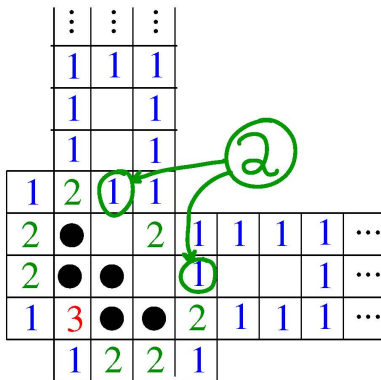


Que faire lorsqu'un fil **tourne** dans le "layout"
rectangulaire du $\{x_i, \neg, \wedge\}$ -circuit planaire ?

Remplacer



par l'amalgame de cases :



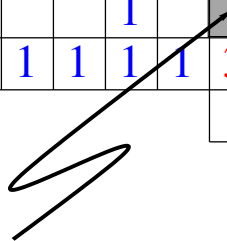
Comment **se termine** un fil ?

Par une fort jolie “mailloche” ...

										1	1	1
...	1	1	1	1	1	1	1	1	3	●	2	
...	1			1			1			●	3	
...	1	1	1	1	1	1	1	1	3	●	2	
										1	1	1

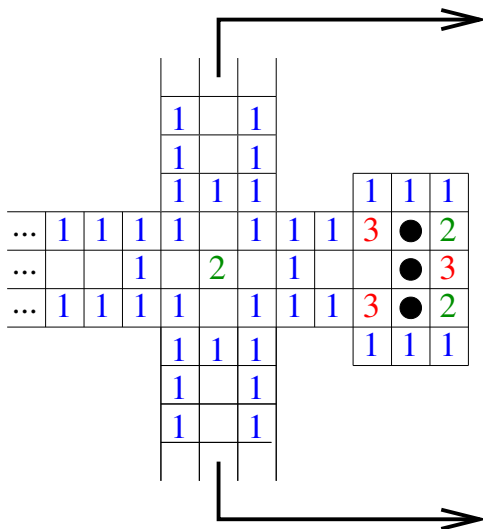
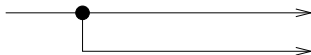
Note : le fil qui correspond à la porte de sortie du $\{x_i, \neg, \wedge\}$ -circuit se termine de la même façon.

									1	1	1
...	1	1	1	1	1	1	1	1	3	●	2
...	1			1			1			●	3
...	1	1	1	1	1	1	1	1	3	●	2
									1	1	1



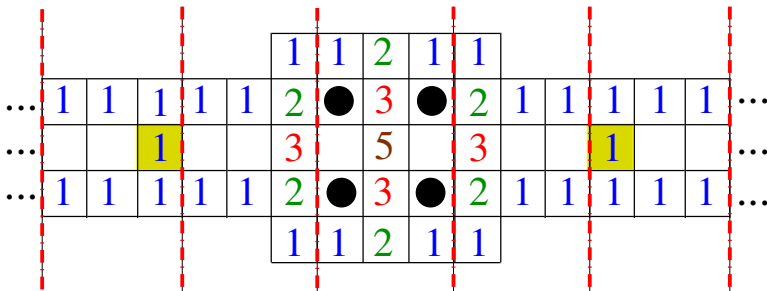
Note : position critique

Duplication d'un fil dans le circuit de départ :



Dernière technicalité : lorsque deux fils entrent dans un bloc simulant une porte \wedge , ils doivent être **en phase**, i.e. leurs 1s de référence doivent être à égale distance de la jonction du bloc.

Exemple de bidule modifiant la **phase** d'un fil :



Après tout ceci, il ne reste qu'à

- calculer $m \in O(|\langle C \rangle|^2)$ tel qu'un tableau T de dimension $m \times m$ englobe toute la construction,
- placer des 0 dans toutes les cases de ce tableau T qui sont à l'extérieur des fils,
- déterminer l'abscisse i et l'ordonnée j de la **position critique** (le long du fil de sortie).

Le résultat est $f(\langle C \rangle) = \langle T, i, j \rangle$.

Après tout ceci, il ne reste qu'à

- calculer $m \in O(|\langle C \rangle|^2)$ tel qu'un tableau T de dimension $m \times m$ englobe toute la construction,
- placer des 0 dans toutes les cases de ce tableau T qui sont à l'extérieur des fils,
- déterminer l'abscisse i et l'ordonnée j de la **position critique** (le long du fil de sortie).

Le résultat est $f(\langle C \rangle) = \langle T, i, j \rangle$.

On se convainc que f est calculable en temps polynomial.

Et pourquoi ça marche ?

- $\langle C \rangle \in \text{NEG-ET-SAT} \Rightarrow$
 - ▶ \exists une affectation σ qui “satisfait” $C \Rightarrow$

- $\langle C \rangle \in \text{NEG-ET-SAT} \Rightarrow$
 - ▶ \exists une affectation σ qui “satisfait” $C \Rightarrow$
 - ▶ σ induit des mines aux positions reflétant le statut VRAI ou FAUX de chaque porte et de chaque fil \Rightarrow

- $\langle C \rangle \in \text{NEG-ET-SAT} \Rightarrow$
 - ▶ \exists une affectation σ qui “satisfait” $C \Rightarrow$
 - ▶ σ induit des mines aux positions reflétant le statut VRAI ou FAUX de chaque porte et de chaque fil \Rightarrow
 - ▶ σ induit une mine en **position critique** \Rightarrow

- $\langle C \rangle \in \text{NEG-ET-SAT} \Rightarrow$
 - ▶ \exists une affectation σ qui “satisfait” $C \Rightarrow$
 - ▶ σ induit des mines aux positions reflétant le statut VRAI ou FAUX de chaque porte et de chaque fil \Rightarrow
 - ▶ σ induit une mine en **position critique** \Rightarrow
 - ▶ la **position critique** peut contenir une mine \Rightarrow

- $\langle C \rangle \in \text{NEG-ET-SAT} \Rightarrow$
 - ▶ \exists une affectation σ qui “satisfait” $C \Rightarrow$
 - ▶ σ induit des mines aux positions reflétant le statut VRAI ou FAUX de chaque porte et de chaque fil \Rightarrow
 - ▶ σ induit une mine en position critique \Rightarrow
 - ▶ la position critique peut contenir une mine \Rightarrow
 - ▶ $f(|\langle C \rangle|) = \langle T, i, j \rangle \in \text{DÉMINEUR}$.
- $\langle C \rangle \notin \text{NEG-ET-SAT} \Rightarrow$

- $\langle C \rangle \in \text{NEG-ET-SAT} \Rightarrow$
 - ▶ \exists une affectation σ qui “satisfait” $C \Rightarrow$
 - ▶ σ induit des mines aux positions reflétant le statut VRAI ou FAUX de chaque porte et de chaque fil \Rightarrow
 - ▶ σ induit une mine en position critique \Rightarrow
 - ▶ la position critique peut contenir une mine \Rightarrow
 - ▶ $f(|\langle C \rangle|) = \langle T, i, j \rangle \in \text{DÉMINEUR}$.
- $\langle C \rangle \notin \text{NEG-ET-SAT} \Rightarrow$
 - ▶ aucune affectation σ ne “satisfait” $C \Rightarrow$

- $\langle C \rangle \in \text{NEG-ET-SAT} \Rightarrow$
 - ▶ \exists une affectation σ qui “satisfait” $C \Rightarrow$
 - ▶ σ induit des mines aux positions reflétant le statut VRAI ou FAUX de chaque porte et de chaque fil \Rightarrow
 - ▶ σ induit une mine en position critique \Rightarrow
 - ▶ la position critique peut contenir une mine \Rightarrow
 - ▶ $f(|\langle C \rangle|) = \langle T, i, j \rangle \in \text{DÉMINEUR}$.
- $\langle C \rangle \notin \text{NEG-ET-SAT} \Rightarrow$
 - ▶ aucune affectation σ ne “satisfait” $C \Rightarrow$
 - ▶ toutes les affectations laissent la position critique libre \Rightarrow

- $\langle C \rangle \in \text{NEG-ET-SAT} \Rightarrow$
 - ▶ \exists une affectation σ qui “satisfait” $C \Rightarrow$
 - ▶ σ induit des mines aux positions reflétant le statut VRAI ou FAUX de chaque porte et de chaque fil \Rightarrow
 - ▶ σ induit une mine en position critique \Rightarrow
 - ▶ la position critique peut contenir une mine \Rightarrow
 - ▶ $f(|\langle C \rangle|) = \langle T, i, j \rangle \in \text{DÉMINEUR}$.
- $\langle C \rangle \notin \text{NEG-ET-SAT} \Rightarrow$
 - ▶ aucune affectation σ ne “satisfait” $C \Rightarrow$
 - ▶ toutes les affectations laissent la position critique libre \Rightarrow
 - ▶ $T(i, j)$ ne peut contenir de mine \Rightarrow
 - ▶ $f(|\langle C \rangle|) = \langle T, i, j \rangle \notin \text{DÉMINEUR}$.

- $\langle C \rangle \in \text{NEG-ET-SAT} \Rightarrow$
 - ▶ \exists une affectation σ qui “satisfait” $C \Rightarrow$
 - ▶ σ induit des mines aux positions reflétant le statut VRAI ou FAUX de chaque porte et de chaque fil \Rightarrow
 - ▶ σ induit une mine en position critique \Rightarrow
 - ▶ la position critique peut contenir une mine \Rightarrow
 - ▶ $f(|\langle C \rangle|) = \langle T, i, j \rangle \in \text{DÉMINEUR}$.
- $\langle C \rangle \notin \text{NEG-ET-SAT} \Rightarrow$
 - ▶ aucune affectation σ ne “satisfait” $C \Rightarrow$
 - ▶ toutes les affectations laissent la position critique libre \Rightarrow
 - ▶ $T(i, j)$ ne peut contenir de mine \Rightarrow
 - ▶ $f(|\langle C \rangle|) = \langle T, i, j \rangle \notin \text{DÉMINEUR}$.

Donc $\text{NEG-ET-SAT} \leq_m^p \text{DÉMINEUR}$.

Donc DÉMINEUR est NP-complet par le théorème de la contagion.