

Plan

- ◆ *Introduction*
- ◆ ***SAT(isfiabilité) propositionnel***
 - *Problème SAT : définitions de base*
 - *Les classes polynomiales*
 - *Algorithmes de résolution*
 - ◆ *Résolution*
 - ◆ *Énumération, Simplifications, Heuristiques, Traitement des échecs*
 - ◆ *Recherche locale (GSAT, Taboo,...)*
 - ◆ *Méthodes hybrides*
 - *Instances de SAT*
 - ◆ *Problèmes structurés*
 - ◆ *problèmes aléatoires - phénomène de seuil*
 - ◆ *problèmes réels*
 - *Structures & résolution*
 - *Extensions*
- **Solveurs SAT modernes**
- **Applications**
- **Max-SAT, Partial Max-SAT, ...**
- ***Directions de recherche***

Références

Web

SAT :

- ◆ SATLIVE : <http://www.satlive.org>
- ◆ Sat Competition : <http://www.satcompetition.org/>
- ◆ SATLIB : <http://www.satlib.org>

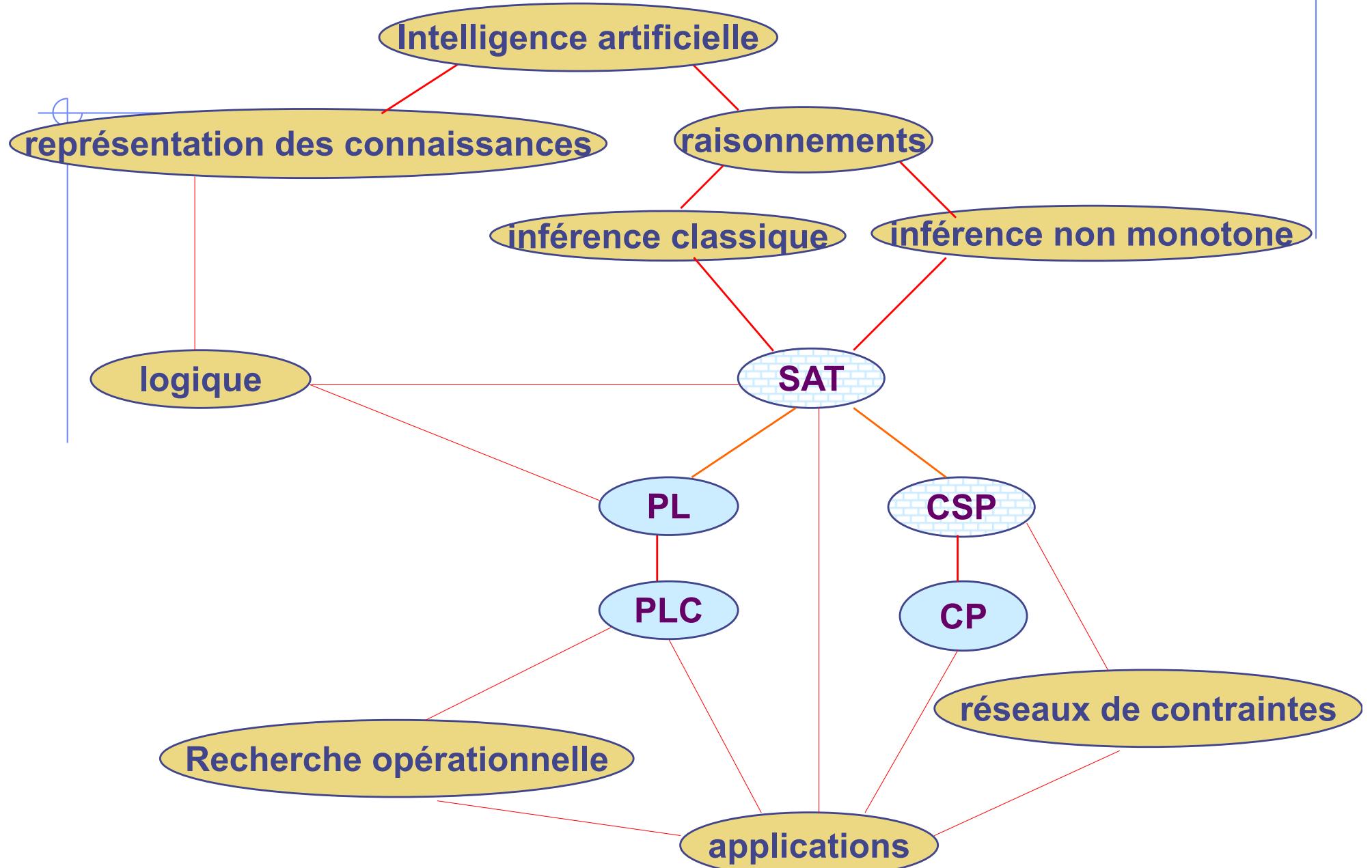
Conférences

IJCAI, AAAI, ECAI, CP, SAT, JFPLC,...

Livres & articles de synthèse

- ◆ Problème SAT : Progrès et Défis. Livre sous la direction de L. Sais. Hermes Publishing Ltd, London, mai 2008
- ◆ Handbook of Satisfiability: Vol 185 Frontiers in Artificial Intelligence and Applications, 2009
- ◆ **Handbook of Parallel Constraint Reasoning** - Editors: Hamadi, Youssef and Sais, Lakhdar (Eds.), Springer, 2018

Introduction & Motivation





Problème SAT : définitions de base

Notations & définitions

- ◆ Une **formule booléenne** :
 - \perp , T sont des formules
 - une variable (atome) propositionnelle A_1, A_2, A_3, \dots est une formule
 - si φ_1 et φ_2 sont des formules, alors $\neg\varphi_1, (\varphi_1 \wedge \varphi_2), (\varphi_1 \vee \varphi_2), (\varphi_1 \rightarrow \varphi_2), (\varphi_1 \leftrightarrow \varphi_2)$ sont des formules
 - **Littéral** : une variable propositionnelle A_i (**littéral positif**) ou sa négation $\neg A_i$ (**littéral négatif**)
 - **Atomes(φ)** : $\{A_1, A_2, A_3, \dots A_n\}$ apparaissant dans φ
 - Une formule booléenne peut être représenté par un arbre ou un DAG (graphe orienté acyclique)

Notations & définitions

- ◆ Une **interprétation complète** I de φ :

$$I : \text{Atomes}() \rightarrow \{\perp, \top\}$$

- ◆ Une **interprétation partielle** I de φ :

$$I : A \rightarrow \{\perp, \top\}, A \subset \text{Atomes}(\varphi)$$

- ◆ représentation d'une *interprétation* :,

- I peut être représentée par *un ensemble de littéraux* :

$$\text{Ex: } \{ I(A_1) = \top, I(A_2) = \perp \} \Rightarrow \{A_1, \neg A_2\}$$

- I peut être représentée par *une formule booléenne* :

$$\text{Ex: } \{ I(A_1) = \top, I(A_2) = \perp \} \Rightarrow A_1 \wedge \neg A_2$$

Notations & définitions

- ◆ $I \models \varphi$ (I satisfait φ)
 - $I \models A_i \Leftrightarrow I(A_i) = T$
 - $I \models \neg\varphi \Leftrightarrow \neg I \models \varphi$
 - $I \models \varphi_1 \wedge \varphi_2 \Leftrightarrow I \models \varphi_1$ et $I \models \varphi_2$
 - ...
- ◆ φ est satisfiable ssi $\exists I$ tq $I \models \varphi$ (I est appelé modèle de φ)
- ◆ $\varphi_1 \models \varphi_2 \Leftrightarrow \forall I$ tq $I \models \varphi_1$ alors $I \models \varphi_2$
(φ_2 est conséquence logique de φ_1)
- ◆ $\models \varphi \Leftrightarrow \forall I, I \models \varphi$ (φ est valide)
- ◆ φ est valide $\Leftrightarrow \neg\varphi$ est unsatisfiable

Équivalence et satisfiabilité

- ◆ φ_1 et φ_2 sont équivalentes ssi, $\forall I, I \models \varphi_1$ ssi $I \models \varphi_2$
- ◆ φ_1 et φ_2 sont équi-satisfiables ssi

$$\exists I_1 \text{ tq } I_1 \models \varphi_1 \text{ ssi } \exists I_2 \text{ tq } I_2 \models \varphi_2$$

- ◆ φ_1 et φ_2 sont équivalentes

$$\Downarrow \quad \Updownarrow$$

φ_1 et φ_2 sont équi-satisfiables

- ◆ Exemple :

$\varphi_1 \vee \varphi_2$ et $(\varphi_1 \vee \neg A_3) \wedge (\varphi_2 \vee A_3)$, avec $A_3 \notin \text{Atomes}$ ($\varphi_1 \vee \varphi_2$) sont équi-satisfiable mais pas équivalentes

Complexité

- ◆ Le problème de décider de la satisfiabilité d'une formule booléenne (**SAT**) est NP-complet [Cook : 71]
- ◆ Les problèmes les plus importants en logique (validité, déduction, équivalence, ...) peuvent être réduit à SAT, et sont (co)NP-complet. (*voir cours outils formels*)
- ◆ Exemple (déduction logique):

$$\varphi_1 \models \alpha \Leftrightarrow \varphi_1 \wedge \neg\alpha \text{ est unsatisfiable}$$

👉 « Il n'existe pas » d'algorithme de complexité polynomiale!
Question importante en théorie de la complexité.



Transformation en NNF, CNF

Polarité des sous-formules

Polarité : le nombre de négations modulo 2

- ◆ Occurrences positive/negative
 - φ apparaît positivement dans φ
 - si $\neg\varphi_1$ apparaît positivement [négativement] dans φ , alors φ_1 apparaît négativement [positivement] dans φ
 - si $\varphi_1 \wedge \varphi_2$ ou $\varphi_1 \vee \varphi_2$ apparaît positivement [négativement] dans φ , alors φ_1 et φ_2 apparaissent positivement[négativement] dans φ
 - $\varphi_1 \rightarrow \varphi_2$ apparaît positivement [négativement] dans φ , alors φ_1 apparaît négativement [positivement] dans φ et φ_2 apparaît positivement [négativement] dans φ
 - $\varphi_1 \leftrightarrow \varphi_2$ apparaît dans φ , alors φ_1 et φ_2 apparaissent positivement et négativement dans φ

Forme Normal Négative (NNF)

- ◆ φ est sous **forme NNF** ssi φ est donnée avec uniquement les connecteurs logique \wedge , \vee et \neg
- ◆ toute formule peut être réduite sous forme NNF :
 - 1) en remplaçant les \rightarrow et \leftrightarrow :

$$\varphi_1 \rightarrow \varphi_2 \Rightarrow \neg\varphi_1 \vee \varphi_2$$

$$\varphi_1 \leftrightarrow \varphi_2 \Rightarrow (\neg\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \neg\varphi_2)$$

- 2) descendre les négations au niveau atomique :

$$\neg(\varphi_1 \wedge \varphi_2) \Rightarrow \neg\varphi_1 \vee \neg\varphi_2$$

$$\neg(\varphi_1 \vee \varphi_2) \Rightarrow \neg\varphi_1 \wedge \neg\varphi_2$$

$$\neg\neg\varphi_1 \Rightarrow \varphi_1$$

- ◆ la réduction préserve l'équivalence,
- ◆ complexité linéaire.

Forme Normal Conjonctive (CNF)

- ◆ φ est sous **forme CNF (clausale)** ssi φ est une conjonction de disjonctions de littéraux :

$$\bigwedge_{i=1}^L \left(\bigvee_{j_i=1}^{K_i} A_{j_i} \right)$$

- ◆ La disjonction de littéraux $\bigvee_{j_i=1}^{K_i} A_{j_i}$ est appelé **clause**
- ◆ La forme CNF est plus facile à manipuler (par les algorithmes) :
 - ↳ liste de listes de littéraux
- ◆ **Exemple :**

$$\varphi = (\neg a \vee b) \wedge (\neg b \vee c) \wedge (\neg c \vee a) \wedge (a \vee b \vee c)$$


clause

Transformation en CNF

- ◆ Toute formule φ peut être réduite à $\text{CNF}(\varphi)$:
 1. réduire en NNF;
 2. Appliquer récursivement les lois de DeMorgan :
$$(\varphi_1 \wedge \varphi_2) \vee \varphi_3 \Rightarrow (\varphi_1 \vee \varphi_3) \wedge (\varphi_2 \vee \varphi_3)$$
 - ◆ Atomes($\text{CNF}(\varphi)$) = Atomes(φ)
 - ◆ $\text{CNF}(\varphi)$ est équivalente à φ
 - ◆ Si φ_1 est équivalente à φ_2 , alors $\text{CNF}(\varphi_1) = \text{CNF}(\varphi_2)$ modulo un ordre
 - ◆ Complexité exponentielle dans le pire des cas
-  Rarement utilisé en pratique.

Transformation en CNF : utilisation de variables supplémentaire

- ◆ Toute formule φ peut être réduite à $\text{CNF}(\varphi)$:
 1. réduire en NNF;
 2. Appliquer récursivement les règles :
$$\varphi \Rightarrow \varphi[A_i \vee A_j | B] \wedge \text{CNF}(B \leftrightarrow (A_i \vee A_j))$$
$$\varphi \Rightarrow \varphi[A_i \wedge A_j | B] \wedge \text{CNF}(B \leftrightarrow (A_i \wedge A_j))$$

A_i et A_j sont des littéraux, et B une variable supplémentaire
 - ◆ Atomes($\text{CNF}(\varphi)$) \supseteq Atomes(φ)
 - ◆ $\text{CNF}(\varphi)$ est équi-satisfiable à φ
 - ◆ Si φ_1 est équivalente à φ_2 , alors $\text{CNF}(\varphi_1) = \text{CNF}(\varphi_2)$ modulo un ordre
 - ◆ Complexité linéaire dans le pire des cas
- ↳ Souvent utilisé en pratique.

Transformation en CNF : version améliorée

- Comme dans le cas précédent, en appliquant (2.) les règles ci-dessous :

$\varphi \Rightarrow \varphi[A_i \vee A_j | B] \wedge \text{CNF}(B \rightarrow (A_i \vee A_j))$, si $A_i \vee A_j$ est positive

$\varphi \Rightarrow \varphi[A_i \vee A_j | B] \wedge \text{CNF}((A_i \vee A_j) \rightarrow B)$, si $A_i \vee A_j$ est négative

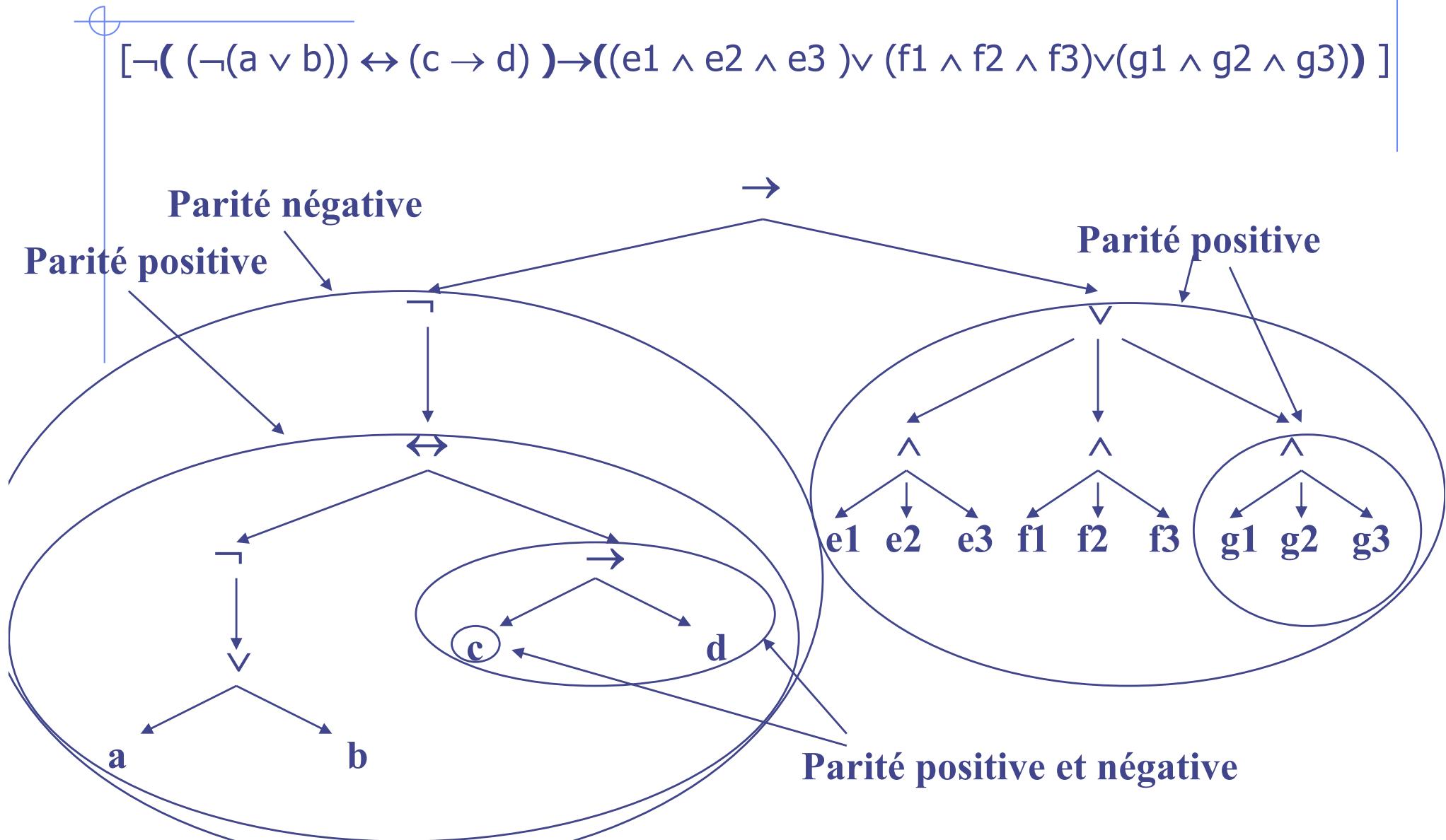
$\varphi \Rightarrow \varphi[A_i \wedge A_j | B] \wedge \text{CNF}(B \rightarrow (A_i \wedge A_j))$, si $A_i \wedge A_j$ est positive

$\varphi \Rightarrow \varphi[A_i \wedge A_j | B] \wedge \text{CNF}((A_i \wedge A_j) \rightarrow B)$, si $A_i \wedge A_j$ est négative



Réduction de la taille de la formule obtenue.

Représentation arborescente d 'une formule booléenne



Transformation en CNF : un exemple

Soit une formule $\varphi =$

$$[\neg((\neg(a \vee b)) \leftrightarrow (c \rightarrow d)) \rightarrow ((e_1 \wedge e_2 \wedge e_3) \vee (f_1 \wedge f_2 \wedge f_3) \vee (g_1 \wedge g_2 \wedge g_3))]$$

Transformation de φ en CNF :

1. $\varphi' = \text{NNF}(\varphi)$:

♦ **remplacer les \rightarrow et \leftrightarrow :**

- $[\neg(\neg(a \vee b)) \leftrightarrow (\neg c \vee d) \rightarrow ((e_1 \wedge e_2 \wedge e_3) \vee (f_1 \wedge f_2 \wedge f_3) \vee (g_1 \wedge g_2 \wedge g_3))]$
- $[\neg((\neg(a \vee b)) \rightarrow (\neg c \vee d)) \wedge ((\neg c \vee d) \rightarrow (\neg(a \vee b))) \rightarrow ((e_1 \wedge e_2 \wedge e_3) \vee (f_1 \wedge f_2 \wedge f_3) \vee (g_1 \wedge g_2 \wedge g_3))]$
- $[\neg(\neg(\neg(a \vee b)) \vee (\neg c \vee d)) \wedge (\neg(\neg c \vee d) \vee (\neg(a \vee b))) \rightarrow ((e_1 \wedge e_2 \wedge e_3) \vee (f_1 \wedge f_2 \wedge f_3) \vee (g_1 \wedge g_2 \wedge g_3))]$
- $[\neg(\neg(\neg(\neg(a \vee b)) \vee (\neg c \vee d)) \wedge (\neg(\neg c \vee d) \vee (\neg(a \vee b)))) \vee ((e_1 \wedge e_2 \wedge e_3) \vee (f_1 \wedge f_2 \wedge f_3) \vee (g_1 \wedge g_2 \wedge g_3))]$

Transformation en CNF : un exemple (suite)

- descendre les négations au niveau atomique :

$$\varphi' = [((a \vee b \vee \neg c \vee d) \wedge ((c \wedge \neg d) \vee (\neg a \wedge \neg b))) \vee ((e1 \wedge e2 \wedge e3) \vee (f1 \wedge f2 \wedge f3) \vee (g1 \wedge g2 \wedge g3))]$$

2. Appliquer les règles

$$[((a \vee b \vee \neg c \vee d) \wedge ((c \wedge \neg d) \vee (\neg a \wedge \neg b))) \vee ((e1 \wedge e2 \wedge e3) \vee (f1 \wedge f2 \wedge f3) \vee (g1 \wedge g2 \wedge g3))]$$

$$(i \rightarrow (c \wedge \neg d)) \wedge (j \rightarrow (\neg a \wedge \neg b)) \wedge (k \rightarrow (e1 \wedge e2 \wedge e3)) \wedge (l \rightarrow (f1 \wedge f2 \wedge f3)) \wedge (m \rightarrow (g1 \wedge g2 \wedge g3))$$

$$[((a \vee b \vee \neg c \vee d) \wedge (i \vee j)) \vee (k \vee l \vee m)]$$

$$(n \rightarrow ((a \vee b \vee \neg c \vee d) \wedge (i \vee j)))$$

$$(n \vee k \vee l \vee m)$$

Transformation en CNF : un exemple (suite)

→

$$\begin{aligned} & (n \vee k \vee l \vee m) \\ & (i \rightarrow (c \wedge \neg d)) \wedge \\ & (j \rightarrow (\neg a \wedge \neg b)) \wedge \\ & (k \rightarrow (e1 \wedge e2 \wedge e3)) \wedge \\ & (l \rightarrow (f1 \wedge f2 \wedge f3)) \wedge \\ & (m \rightarrow (g1 \wedge g2 \wedge g3)) \\ & (n \rightarrow ((a \vee b \vee \neg c \vee d) \wedge (i \vee j))) \end{aligned}$$

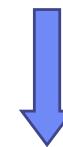
$$\begin{aligned} & ((n \vee k \vee l \vee m) \wedge \\ & (\neg i \vee c) \wedge (\neg i \vee \neg d) \wedge \\ & (\neg j \vee \neg a) \wedge (\neg j \vee \neg b) \wedge \\ & (\neg k \vee e1) \wedge (\neg k \vee e2) \wedge (\neg k \vee e3) \wedge \\ & (\neg l \vee f1) \wedge (\neg l \vee f2) \wedge (\neg l \vee f3) \wedge \\ & (\neg m \vee g1) \wedge (\neg m \vee g2) \wedge (\neg m \vee g3) \\ & (\neg n \vee a \vee b \vee \neg c \vee d) \wedge (\neg n \vee i \vee j) \end{aligned}$$

Forme CNF

Transformation en K-CNF

- ◆ K-CNF : est une formule CNF avec des clauses de k littéraux
- ◆ 2-CNF est polynomial
- ◆ K-CNF est NP-complet, $k \geq 3$
- ◆ toute formule k-SAT peut être réduite en 3-CNF :

$$(A_1 \vee A_2 \vee \dots \vee A_{k-1} \vee A_k)$$



$$(A_1 \vee A_2 \vee B_1) \wedge$$

$$(\neg B_1 \vee A_3 \vee B_2) \wedge$$

...

$$(\neg B_{k-4} \vee A_{k-2} \vee B_{k-3}) \wedge$$

$$(\neg B_{k-3} \vee \dots \vee A_{k-1} \vee A_k)$$

D'un exemple à représentation clausale

- ♦ **Exemple :** Soit un moteur d'un véhicule un hybride de moto, vélo, et d 'auto). Le problème est pour le conducteur, de détecter les pannes grâce à des indices.

Pannes

p1 = il n'y a plus d'eau
p2 = il n'y a plus de huile
p3 = la courroie est cassée
p4 = le redresseur est coupé
p5 = la batterie est vide
p6 = la batterie est en court circuit
p7 = le fusible est fondu

Indices

I1 = le voyant de température est rouge
I2 = l ' ampèremètre de charge est positif
I3 = le voyant de huile est rouge
I4 = le compte tour est positif
I5 = le moteur tourne
I6 = les phares marchent

Utilitaires

u1 = la bobine est alimenté
u2 = les circuits secondaires sont alimentés
u3 = il y a un court circuit

D'un exemple à représentation clausale

- ♦ Si le voyant de température est rouge alors, d 'une part il n 'y a plus d 'eau ou il n 'y a plus d 'huile ou la courroie est cassée, et d 'autre part les circuits secondaires sont alimentés.

$$i1 \rightarrow ((p1 \vee p2 \vee p3) \wedge u2)$$

clauses : $(\neg i1 \vee p1 \vee p2 \vee p3) \wedge (\neg i1 \vee u2)$

- ♦ Le voyant d 'huile est rouge si et seulement si les circuits secondaires sont alimentés, et, il n 'y a plus d 'huile ou le moteur ne tourne pas.

$$i3 \leftrightarrow u2 \wedge (p2 \vee \neg i5)$$

clauses : $(\neg i3 \vee u2) \wedge (\neg i3 \vee p2 \vee \neg i5) \wedge (\neg u2 \vee \neg p2 \vee i3) \wedge (\neg u2 \vee i5 \vee i3)$

- ♦ Une batterie en court circuit est vide

$$p6 \rightarrow p5$$

clause : $\neg p6 \vee p5$

- ♦ si l 'ampèremètre de charge est positif et la batterie est en court circuit alors le fusible est fondu

$$(i2 \wedge p6) \rightarrow p7$$

clause : $(\neg i2 \vee \neg p6 \vee p7)$

- ♦ ...

D'un exemple à représentation clausale

Coder en CNF les problèmes suivants:

- ◆ Puzzles Logiques :
 - *problème crypto-arithmétique :*
coder en CNF le problème consistant à trouver une affectation des lettres dans {0..9} de manière à satisfaire l 'addition : DONALD + GERALD = ROBERT, tout en évitant la solution triviale où les lettres reçoivent zéro comme valeur.
 - *Problème des pigeons (pigeon-hole problem) :*
mettre n pigeons dans n-1 pigeonniers, avec un pigeon par pigeonnier.
- ◆ *Problèmes mathématiques :*
 - Dans toute collection de 6 personnes, on doit trouver, 3 personnes qui se connaissent ou 3 personnes qui ne se connaissent pas. (instance du théorème de Ramsey)
 - ...



Problème SAT : Objectifs & historique

Objectifs

- ◆ Mots clés
 - Codage, Résolution, Extension, Application
- ◆ But :
 - ☞ Élargir la classe des problèmes traitables en pratique
 - Exhiber des restrictions (classes) polynomiales
 - Améliorer les algorithmes de résolution :
 - ◆ Proposer de nouveaux raffinements aux algorithmes
 - ◆ Mise au point de nouvelles approches
 - ◆ Évaluer les approches proposées :
 - complexité théorique des algorithmes
 - complexité en pratique

Historique

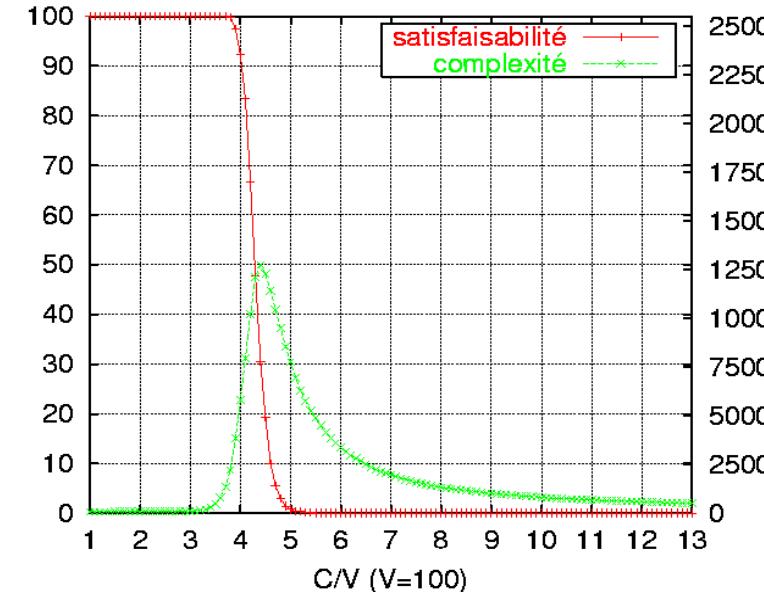


Problèmes structurés

- Les puzzles logiques
- Le problème des pigeons
- Les reines, ...

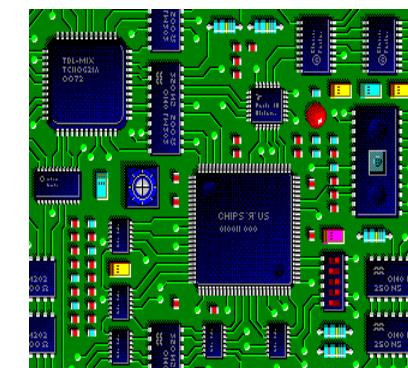
1990

~ 100 variables / 200 clauses



Génération aléatoire

- Phénomène de seuil



Applications

- VLSI, Planification
- Cryptographie, ...

2012

Millions de variables et de clauses

Quelques applications de SAT

- ◆ Satisfaction de contraintes
 - planification, emploi du temps
 - raisonnement temporel (Allen 1983)
 - conception et vérification de circuit (VLSI) (Larrabee 1992)
 - ...
- ◆ raisonnement (déductif)
 $\Sigma \models \alpha$ ssi $\Sigma \wedge \neg\alpha$ est **unsatisfiable**
- ◆ Autres modèles de raisonnement en AI (voir cours RCR)
 - diagnostic / abduction
 - raisonnement par défaut,
 - révision de croyances
 - ...
- ◆ Apprentissage
- ◆ Autres applications
 - cryptographie
 - model checking (vérification de modèles)
 - ...



Instances de SAT

Problèmes structurés

- ◆ *Puzzle logiques :*
 - e.g. reines, zebres, ...
- ◆ *Problèmes mathématiques :*
 - e.g. instances du théorème de Ramsey, de Schur, d 'Erdös, problèmes de quasi-groupes,...
- ◆ *Problèmes générés artificiellement :*
 - pigeons, formules de Tseitin,...
- ◆ Problèmes issus d 'applications réels

Exemple : problème des pigeons

- ◆ ***Exemple*** : problème des pigeons
 - soit les variables propositionnelles :
 $p(i,j)$ « le pigeon i est dans le pigeonnier j », $i=1..n$, $j=1..n-1$
 - ◆ *Ne pas laisser un pigeon libre* :
$$((p(1,1) \vee p(1,2) \vee \dots \vee p(1,n-1)) \wedge$$
$$((p(2,1) \vee p(2,2) \vee \dots \vee p(2,n-1)) \wedge$$

...
$$((p(n,1) \vee p(n,2) \vee \dots \vee p(n,n-1))$$
 - ◆ *mettre exactement un pigeon par pigeonnier*
$$(\neg p(1,1) \vee \neg p(2,1)) \wedge \dots \wedge (\neg p(1,1) \vee \neg p(n,1)) \wedge (\neg p(2,1) \vee \neg p(3,1)) \wedge$$
$$\dots \wedge (\neg p(2,1) \vee \neg p(n,1)) \wedge \dots \wedge (\neg p(n-1,1) \vee \neg p(n,1))$$

...
$$(\neg p(1,n-1) \vee \neg p(2,n-1)) \wedge \dots \wedge (\neg p(1,n-1) \vee \neg p(n,n-1)) \wedge (\neg p(2,n-1) \vee \neg p(3,n-1)) \wedge \dots \wedge (\neg p(2,n-1) \vee \neg p(n,n-1)) \wedge \dots \wedge (\neg p(n-1,n-1) \vee \neg p(n,n-1))$$

Problèmes aléatoires - phénomène de seuil

- ◆ Génération aléatoire de formule **k**-CNF (avec **V** variables et **C** clauses)
répéter
 1. Prendre uniformément k atomes parmi V
 2. Négativer chaque atome avec une probabilité 0.5
 3. Créer une clause avec les k littéraux résultants

Jusqu 'à (#clauses générées = C)

Fixed-clause length model

Phénomène de seuil (transition de phase)

- ◆ Expérimentations :
 - fixer k et V
 - pour des valeurs croissantes de C , générer et tester la satisfiabilité des (500, 1000, 10000, ...) instances avec k , V , C
 - tracer les courbes :
 - ◆ pourcentages d'instances satisfiables en fonction de C/V
 - ◆ moyenne du temps CPU (sur 500, 1000, ...) en fonction de C/V

... \Rightarrow

Phénomène de seuil

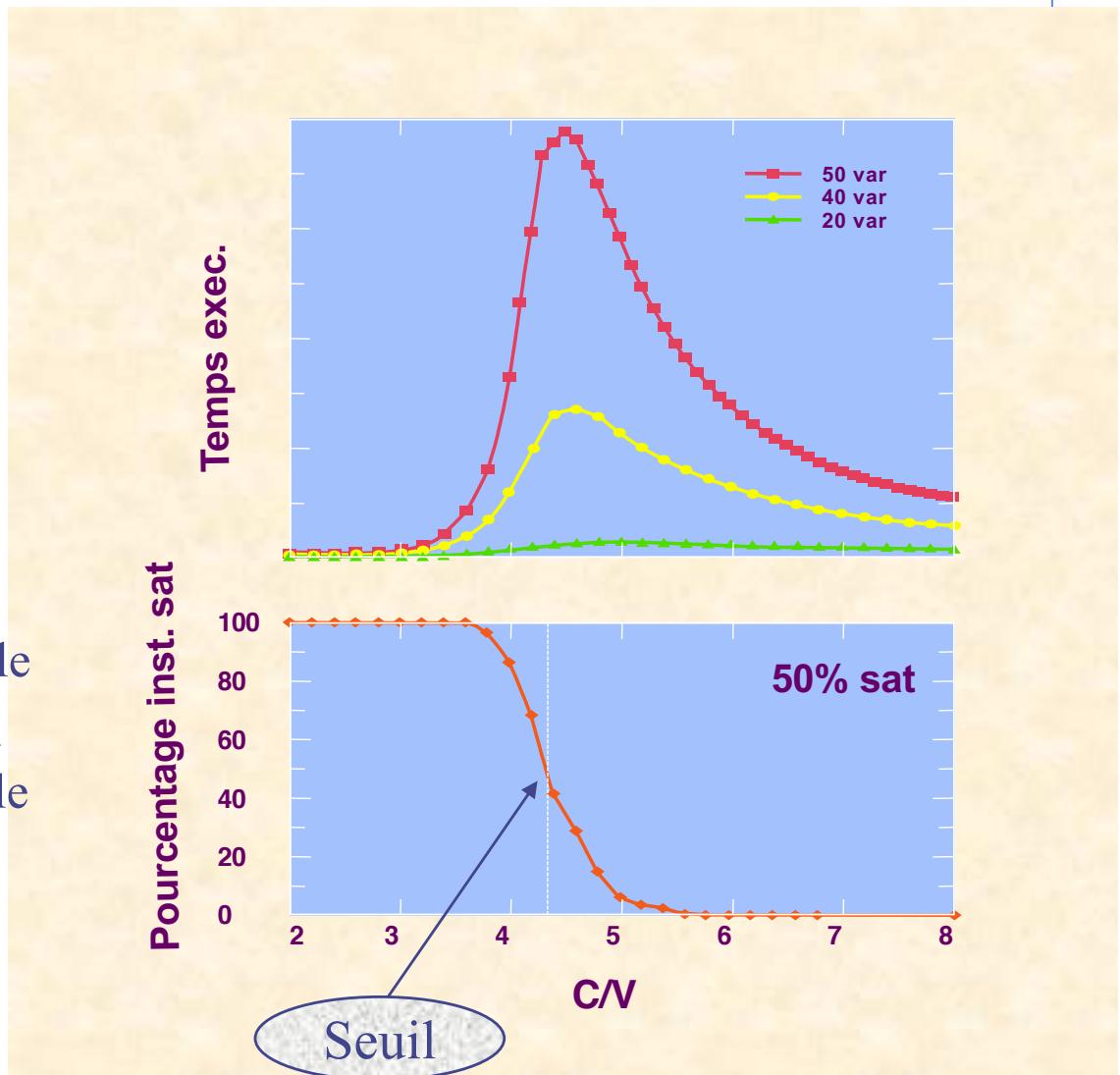
Transition de phase : %SAT

- En augmentant C/V, on passe de 100% d'instances satisfiable à 100% d'instances unsatisfiable
- pour $V \rightarrow \infty$, la transition tend vers une loi 0/1
- Phénomène constaté sur d'autres problèmes NP-complet

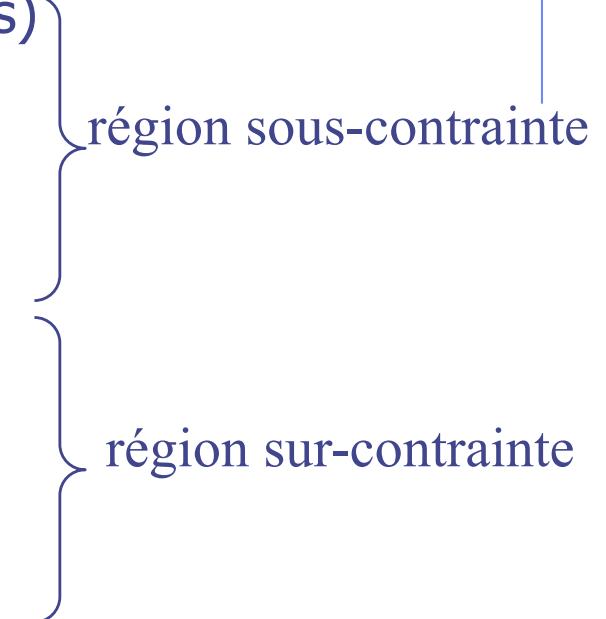
Difficulté de résolution

- En augmentant C/V, on distingue trois régions : facile, difficile, facile
- Le pique de difficulté se trouve au point où 50% d'instance satisfiable
-

Phénomène constaté sur d'autres problèmes NP-complet



Intuition

- ◆ faible rapport C/V : peu de clauses (contraintes)
 - beaucoup de solutions
 - facilement trouvé
 - ◆ rapport C/V élevé :
 - beaucoup de clauses de contraintes
 - inconsistance facilement détecté
- 
- région sous-contrainte
- région sur-contrainte

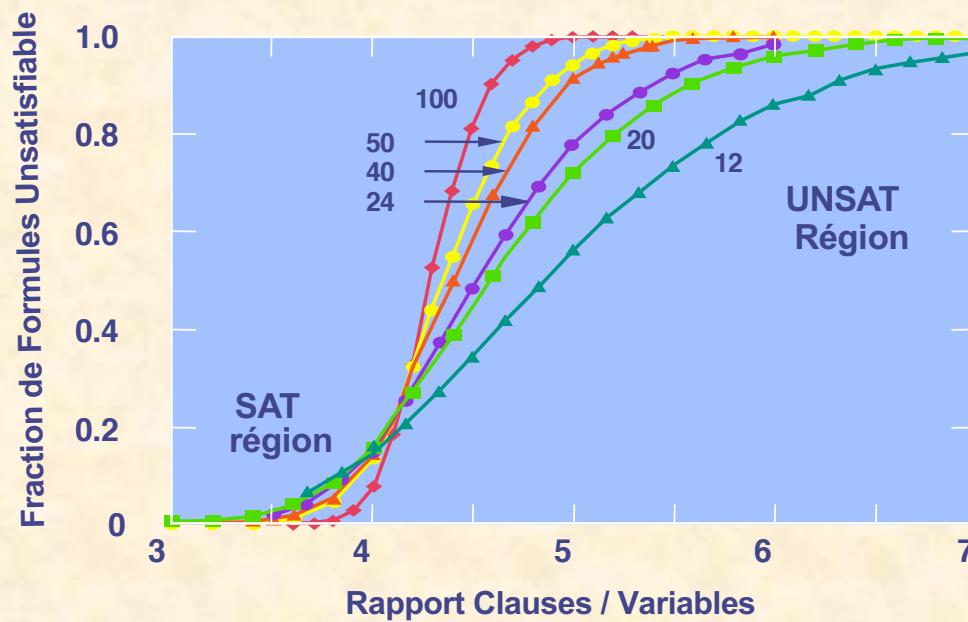
Remarque :

- pour une formule à n variables -> 2^n interprétations
- une clause C élimine (est falsifié par) $2^{n-|C|}$ interprétations

Zoom sur le seuil

- Redressement de la transition pour des valeurs élevées du nombre de variables

Transition de phase 3-SAT



Statut théorique du seuil

♦ Challenge : trouver la valeur théorique du seuil Problème...

♦ Statut actuel :

- 2-SAT : seuil ($C/V = 1$)

- 3-SAT seuil entre **3.003** et **4.54**

(Motwani et al. 1994; Broder and Suen 1993; Broder et al. 1992; Dubois 1990)

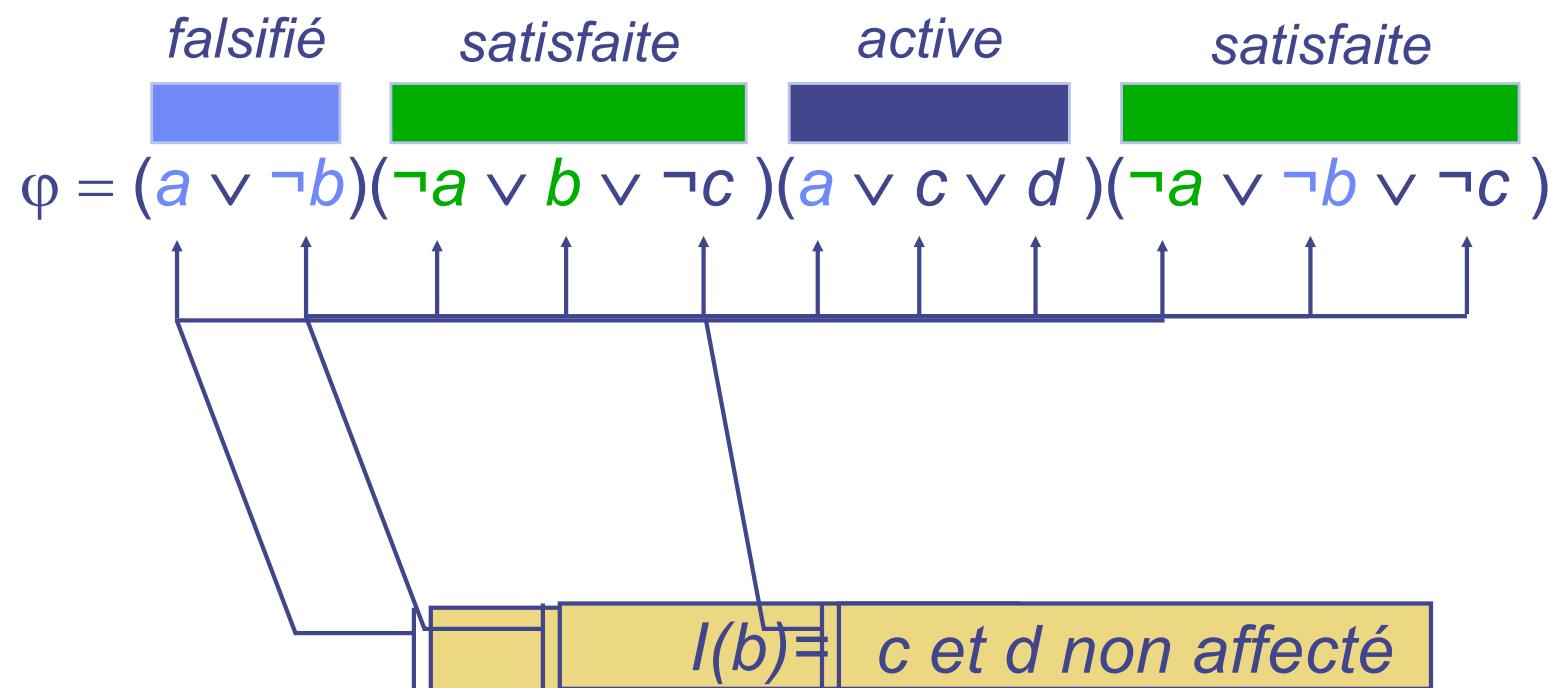
- k -SAT $k > 3$?



Quelques définitions

Définitions (suite)

Littéral & Clause : classification



Définitions (suite)

Règle de clause unitaire - implications : propagation unitaire

- ◆ Une clause active *est unitaire* si elle a exactement un littéral non affecté

Soient $\varphi = (a \vee c) \wedge (b \vee c) \wedge (\neg a \vee \neg b \vee \neg c)$,

et I une interprétation tq. $I(a) = T, I(b) = T$

- ◆ Une clause unitaire admet une seul option pour être satisfaite.

$$a \wedge b \rightarrow \neg c$$

i.e. c doit être affecté à \perp .

- ◆ La **propagation unitaire** (PU) consiste à affecter par propagation tout les littéraux unitaires (apparaissant dans une clause unitaire) d'une formule

Définitions (suite)

Règle du littéral pure (monotone)

- ♦ une variable est dite *pure* dans φ si ses littéraux sont soit tous positifs soit tous négatifs
- ♦ La satisfiabilité d'une formule ne change pas par l'affectation des littéraux purs de manière à satisfaire toutes les clauses où ils apparaissent.

$$\varphi = (a \vee c) \wedge (b \vee c) \wedge (b \vee \neg d) \wedge (\neg a \vee \neg b \vee d)$$

- ♦ affecter c à T ; si φ devient unsatisfiable, alors φ est aussi unsatisfiable avec c affecté à \perp .

Définitions (suite)

Règle de Resolution/Consensus

- ◆ *Résolution:*

Soient deux clauses : $\omega_1 = (\neg a \vee \alpha)$, $\omega_2 = (a \vee \beta)$, α et β sont des sous-clauses (disjonction de littéraux)

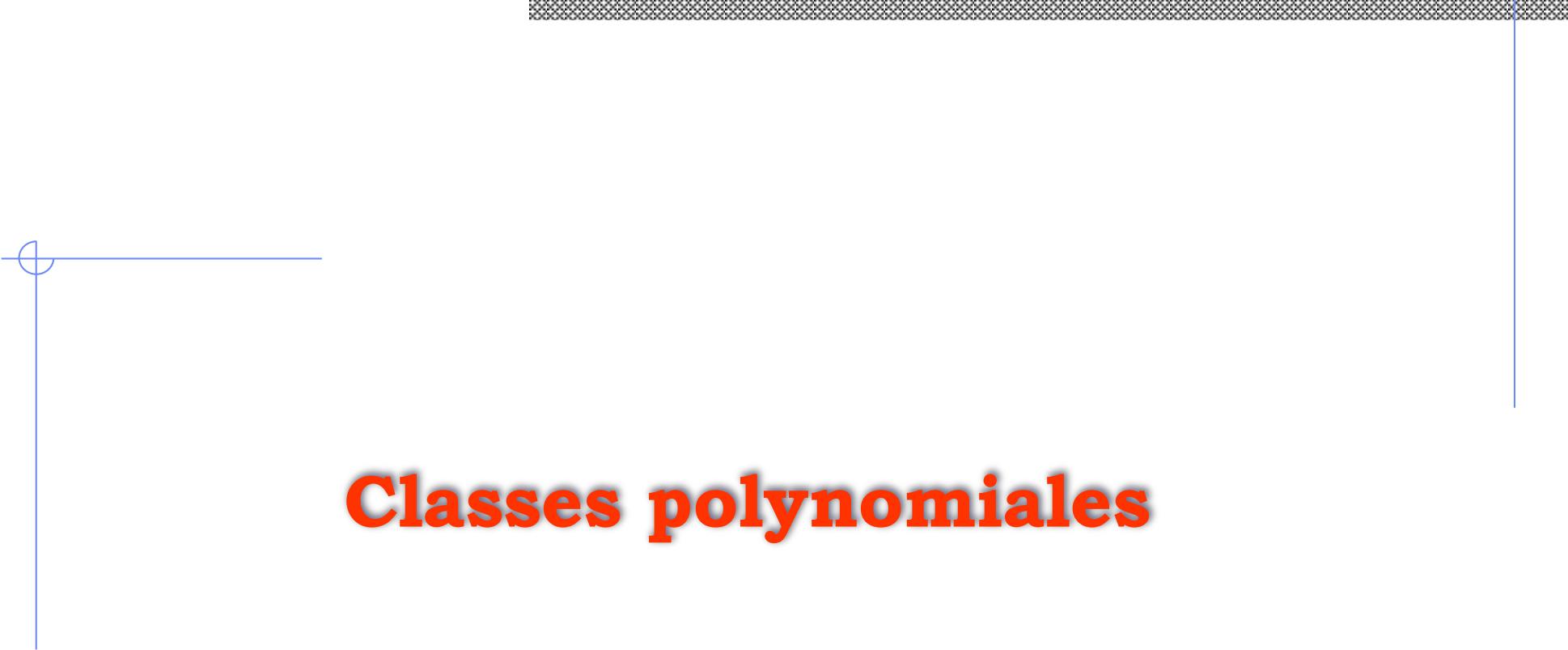
On appelle une **résolvante** de ω_1 et ω_2 la clause :

$$\text{res}(\omega_1, \omega_2, a) = (\alpha \vee \beta)$$

- ◆ $\omega_1 \wedge \omega_2 \models (\alpha \vee \beta)$

- ◆ Remarque :

la règle de clause unitaire est un cas particulier de la règle de résolution
 $(a) \wedge (\neg a \vee \alpha) \models \alpha$



Classes polynomiales

Classes polynomiales : 2-SAT

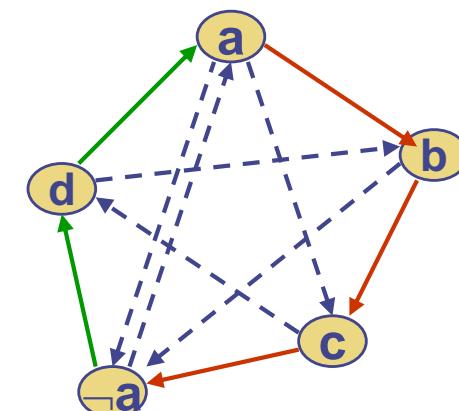
- ♦ Une **classe polynomiale** est un ensemble de formules qu 'il est possible de *reconnaitre* et de *résoudre* en temps polynomial.
- ♦ **2-SAT (clauses binaires)**

Exemple :

$$\varphi = (\neg a \vee b) \wedge (\neg b \vee c) \wedge (\neg c \vee \neg a) \wedge (a \vee d) \wedge (\neg d \vee a)$$

Algorithme : φ est satisfiable?

1. Construire Graphe(φ) = $G = (S, A)$
2. Fermeture transitive(G) = $G' = (S', A')$
3. Si $(\exists p \text{ tq } (p, \neg p) \in A' \text{ et } (\neg p, p) \in A')$
alors retourner faux
sinon retourner vrai



Remarque :

- $(a \rightarrow b) \Leftrightarrow (\neg b \rightarrow \neg a)$
- stabilité des clauses binaires par adjonction d 'un littéral unitaire

Classes polynomiales : Horn-SAT

♦ Clauses (formule) de Horn

Une clause est dite de Horn ssi elle contient au plus un littéral positif

Exemple : règle de prolog

- (a) $\longrightarrow a.$
- $(a \vee \neg b \vee \neg c)$ $\longrightarrow a:- b, c.$
- $(\neg a \vee \neg b \vee \neg c)$ $\longrightarrow :- a, b, c.$

Algorithme : φ est satisfiable?

1. Appliquer la règle de propagation unitaire
2. Si (φ contient une clause vide)
alors retourner faux
sinon retourner vrai

Remarque :

- la PU est complète pour les clauses de Horn
- les clauses de Horn sont stables par adjonction d 'un littéral unitaire

Classes polynomiales : Horn-renommable

♦ *Définitions :*

- Un **renommage** σ des variables $x_1, x_2, \dots x_k$ dans une formule CNF φ consiste à substituer toutes les occurrences des littéraux construits sur les x_i par leurs opposés.
 ⇒ $\sigma(\varphi)$ et φ sont équi-satisfiable.
- Un renommage σ est appelé **Horn-renomage** ssi $\sigma(\varphi)$ est une formule de Horn.
- Une **formule est dite Horn-renommable** ssi elle admet un Horn-renommage.

Classes polynomiales : Horn-renommable

- ◆ **Reconnaissance d'une formule Horn-renommable :**

- La recherche d'un Horn-renommage d'une formule φ se réduit au problème de la satisfiabilité d'un ensemble de clauses binaires (2-SAT)

Idée : associer à φ la formule $H(\varphi)$ définie comme suit :

$$\wedge \quad \wedge (p_i \vee p_j)$$

$$C_k \in \varphi \quad p_i, p_j \in C_k$$

$\wedge (p_i \vee p_j)$ exprime «au plus un des littéraux de C_k peut être faux»
 $p_i, p_j \in C_k$

Classes polynomiales : un exemple

- ♦ **Exemple 1 :** (formule horn renommable)

$$\varphi = (\neg a \vee b) \wedge (\neg b \vee c) \wedge (\neg c \vee a) \wedge (a \vee b \vee c)$$

$$H(\varphi) = (\neg a \vee b) \wedge (\neg b \vee c) \wedge (\neg c \vee a) \wedge (a \vee b) \wedge (a \vee c) \wedge (b \vee c)$$

$H(\varphi)$ est satisfiable $\Rightarrow \mu = \{a, b, c\}$ est un modèle de $H(\varphi)$

↳ μ est un Horn- renommage de φ

on obtient la formule de Horn :

$$(a \vee \neg b) \wedge (b \vee \neg c) \wedge (c \vee \neg a) \wedge (\neg a \vee \neg b \vee \neg c)$$

Classes polynomiales : un exemple

- ♦ ***Exemple 2 :***

$$\varphi = (\neg a \vee \neg b \vee \neg c) \wedge (\neg a \vee b) \wedge (\neg b \vee c) \wedge (\neg c \vee a) \wedge (a \vee b \vee c)$$

$$\begin{aligned} H(\varphi) = & (\neg a \vee \neg b) \wedge (\neg a \vee \neg c) \wedge (\neg b \vee \neg c) \wedge \\ & (\neg a \vee b) \wedge (\neg b \vee c) \wedge (\neg c \vee a) \wedge (a \vee b) \wedge (a \vee c) \wedge (b \vee c) \end{aligned}$$

$H(\varphi)$ est insatisfiable

☞ φ n'est pas Horn-renommable

Classes polynomiales :Q-Horn

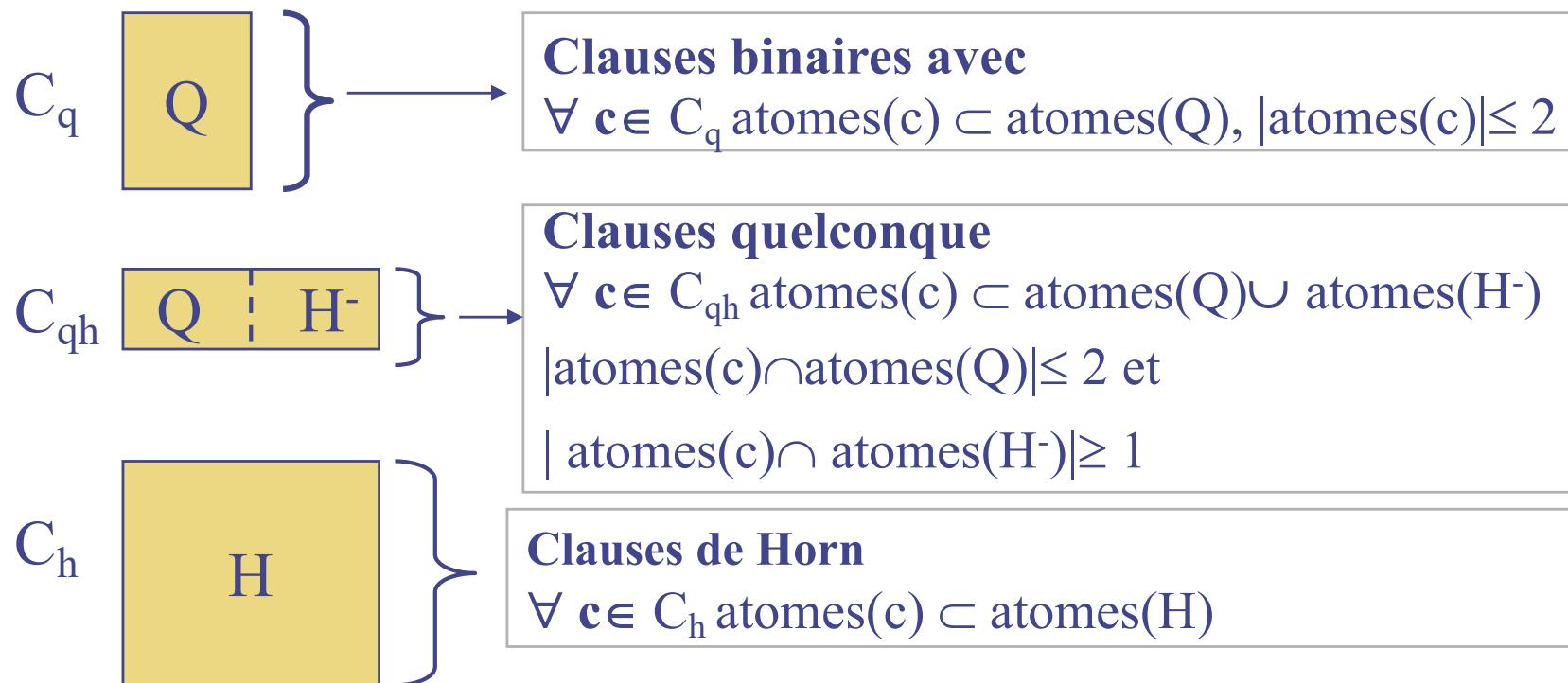
- ♦ **Q-Horn (généralisation de Horn-SAT et 2-SAT)**

Définition : une ensemble de clauses est dit **Q-Horn** si il existe une partition des variables en deux sous-ensembles disjoints H et Q tels que :

- aucune clause ne contient plus de deux variables de Q
- aucune clause ne contient plus d'un littéral positif de H
- aucune clause contenant un littéral positif de H ne contient un littéral de Q.

Classes polynomiales : Q-Horn

- Représentation schématique



- algorithme de reconnaissance en temps linéaire [J-J Hébrard]

Classes polynomiales

♦ Autres classes polynomiales

- Q-Horn renommable
- Formules bien imbriqués [Knuth]
- Hiérarchies de Gallo & Scutellà, généralisée par Dallal et Etherington
- ***Restrictions sur le nombre d 'occurrences :***

On définit **r,s-SAT** comme la classe des formules où les clauses ont exactement **r** littéraux et chaque variable admet au plus **s** occurrences.

- ♦ *, s-SAT (pour tout r et s<4) est polynomiale
- ♦ 3, 4-SAT est NP-complet
- ♦ toute instance r,r-SAT est satisfiable