Chapitre 5
Les structures traitables

Claude-Guy Quimper

Introduction

- Nous avons vu dès le premier chapitre qu'un arbre de recherche peut être de taille exponentielle.
- Les solveurs peuvent passer des heures, voire des jours, à explorer ces arbres de recherche.
- Or, nous avons aussi vu que la vérification anticipée, le filtrage, la cohérence de singleton et la cohérence de chemin permettent de réduire la taille d'un arbre de recherche.

Introduction

- Nous voyons dans ce chapitre des problèmes où le filtrage est suffisamment puissant pour rendre la taille de l'arbre polynomial.
- Il sera donc possible de vérifier en temps polynomial si un problème a une solution.
- Bien sûr, ces problèmes ne sont pas NP-Complets puisqu'il est possible de les résoudre en temps polynomial.

Structures traitables

- On ne résout généralement pas des problèmes dans la classe de complexité P à l'aide d'un solveur. On utilise plutôt des algorithmes spécialisés qui sont plus efficaces.
- Cependant, les problèmes NP-Complets sont formés d'une combinaison de sous-problèmes qui peuvent être résolus en temps polynomial.
- Certains problèmes partagent des propriétés communes qui les rendent « faciles » à résoudre.
- On appelle « structure traitable » ces propriétés qui garantissent la résolution du problème en temps polynomial.

Comment éviter les retours arrière?

- Il y a deux façons d'éviter les retours arrière lors de la fouille d'un arbre de recherche.
  - On peut avoir une heuristique qui ne fait que de bons choix.
  - On peut avoir un filtrage assez fort pour éliminer tous les mauvais choix.
- Ces deux réponses sont en fait équivalentes. Si une heuristique est assez puissante pour identifier tous les bons choix, elle peut très bien filtrer les mauvais choix.
- Si un algorithme de filtrage est assez fort pour élimer tous les mauvais choix, l'heuristique ne peut plus se tromper.

Le cas trivial

- Considérons un problème avec une seule contrainte sur laquelle on applique la cohérence de domaine.
- À chaque branchement, on a la garantie qu'il existe un support de domaine, donc une solution au problème avec la variable et la valeur choisies.
- Un choix est donc toujours un bon choix et mène à une solution sans même faire un retour arrière.

Exemple du cas trivial

$$(A, B, C) \in \{(1,2,3), (1,3,2), (3,2,1), (2,1,3)\}$$
$$\mathrm{dom}(A) = \mathrm{dom}(B) = \mathrm{dom}(C) = \{1,2,3\}$$

- Le solveur branche en affectant la valeur 1 à la variable A. Après l'application de la cohérence de domaine, nous obtenons les domaines suivants.

$\mathrm{dom}(A) = \{1\}$ $\mathrm{dom}(B) = \{2,3\}$ $\mathrm{dom}(C) = \{2,3\}$

- Le solveur choisit ensuite une valeur pour B, disons 2, et la variable C est fixée à 3.
- Nous obtenons donc la solution A=1, B=2 et C=3 sans avoir eu à faire un retour arrière.

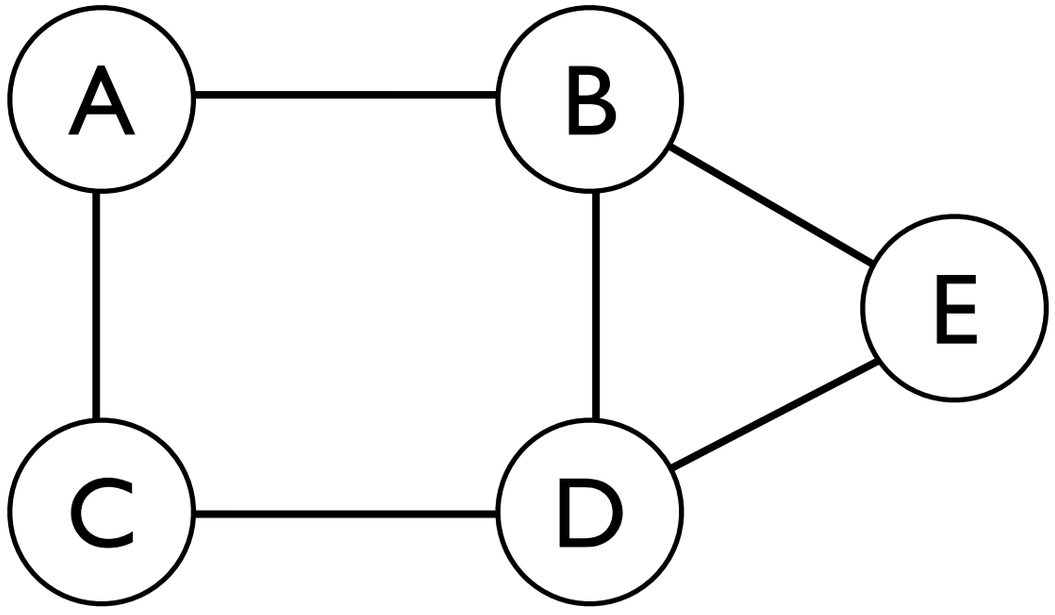
Le graphe de contraintes

- Le premier cas non trivial de structure traitable que nous allons étudier concerne les problèmes de satisfaction de contraintes binaires. Ce sont des problèmes où toutes les contraintes n'ont que deux variables dans leur portée.

- Nous allons utiliser le graphe de contraintes comme outils pour analyser la structure du problème.

- Le graphe de contraintes est un graphe où chaque variable du problème est un noeud et chaque contrainte binaire est représentée par une arête reliant les deux variables de sa portée.

Exemple de graphe de contraintes



$$A < B$$
$$A \neq C$$
$$D = 2C$$
$$D = 3B$$
$$E = 4B$$
$$2E < 4D$$

Les arbres

- **Définition (arbre):** Un arbre est un graphe qui ne comporte pas de cycle.
- **Définition (arbre orienté)**: Un arbre peut être orienté en promouvant un noeud au statut de racine. Chaque noeud possède donc un parent (à l'exception de la **racine**) et des enfants (à l'exception des **feuilles**).
- **Définition (hauteur d'un noeud)**: La hauteur d'une feuille est zéro. La hauteur d'un noeud interne est donnée par le maximum des hauteurs des enfants plus un.
- **Théorème**: Si le graphe de contraintes d'un problème est un arbre et que la cohérence de domaine est appliquée sur chaque contrainte, alors une fouille avec filtrage mène à une solution sans aucun retour arrière.

Démonstration

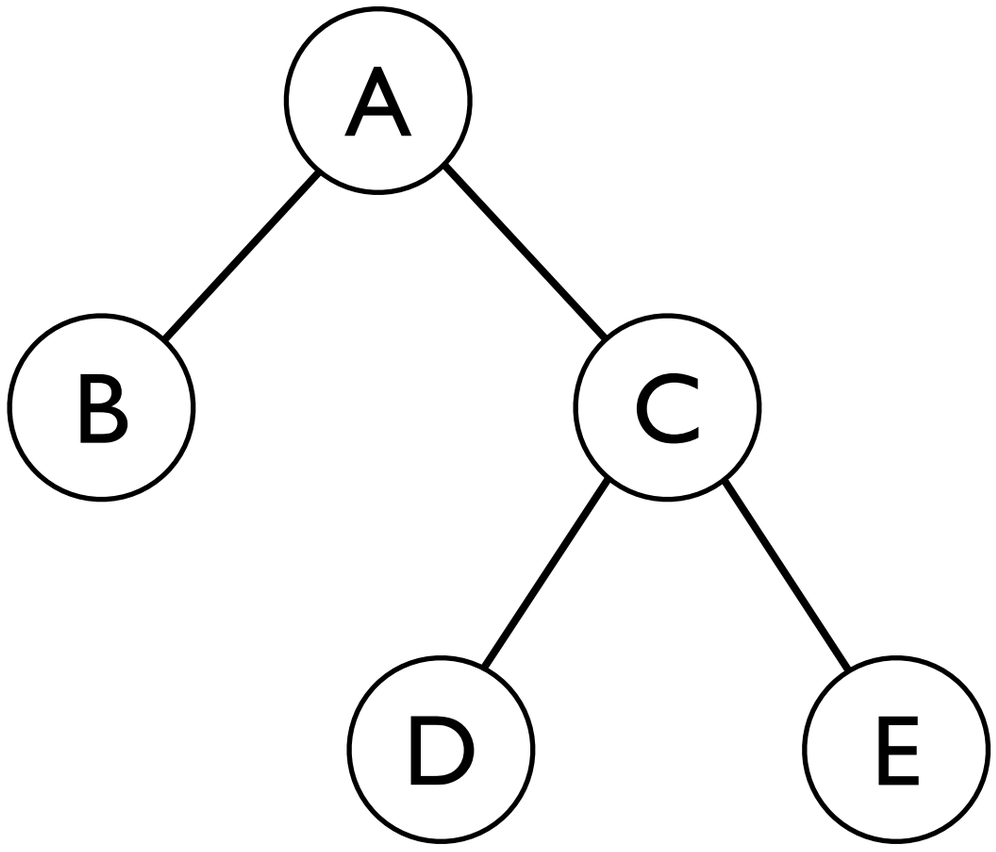
- **Hypothèse d'induction:** si la cohérence de domaine est maintenue sur les contraintes et qu'on instancie une variable de hauteur *h*, alors il existe une façon d'affecter les descendants de cette variable à des valeurs pour former une solution.

- **Case de base:** Les feuilles n'ont pas d'enfants donc l'hypothèse est vraie pour $h = 0$.

- **Étape d'induction**: Soit X_i une variable au niveau h+1. Puisque toutes les contraintes sont cohérentes de domaine, toutes les valeurs dans le domaine de X_i ont un support de domaine pour les contraintes reliant X_i à ses enfants. Si on fixe X_i à une valeur v, il existe donc une valeur pouvant être assignée à chaque enfant de X_i. En appliquant l'hypothèse d'induction, il existe donc une affectation valide pour tous les descendants de X_i.

Exemple

Qu'arrive-t-il si on instancie A=2?



$$A < B$$

$$A = 2C$$

$$C \mod 2 = D$$

$$E \mod 3 = C$$

$$\mathrm{dom}(A) = \{0, 2, 4\}$$

$$\mathrm{dom}(B) = \{1, 3, 5\}$$

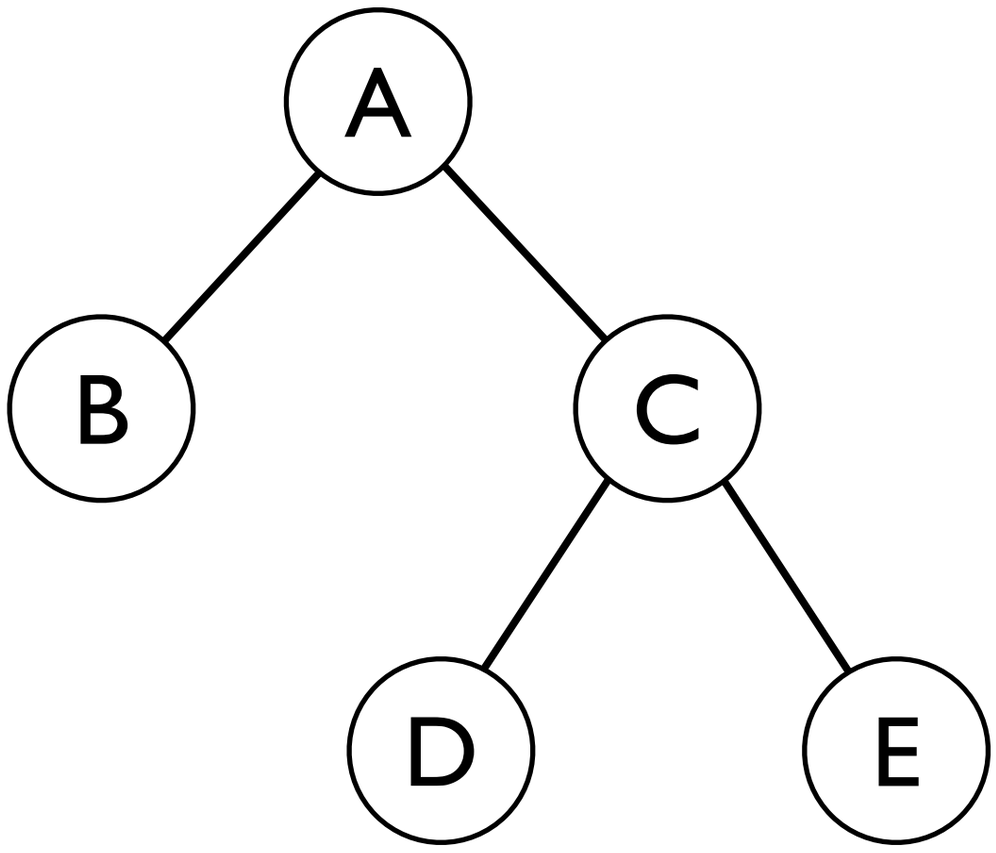
$$\mathrm{dom}(C) = \{0, 1, 2\}$$

$$\mathrm{dom}(D) = \{0, 1\}$$

$$\mathrm{dom}(E) = \{3, 4, 5, 7\}$$

Exemple

Qu'arrive-t-il si on instancie A=2?



$$A < B$$

$$A = 2C$$

$$C \mod 2 = D$$

$$E \mod 3 = C$$

$$\mathrm{dom}(A) = \{2\}$$

$$\mathrm{dom}(B) = \{3, 5\}$$

$$\mathrm{dom}(C) = \{1\}$$

$$\mathrm{dom}(D) = \{1\}$$

$$\mathrm{dom}(E) = \{4, 7\}$$

Application

- Modéliser un problème où un sous-ensemble des contraintes forme un arbre permet d'obtenir un fort filtrage sur ces contraintes.

- Considérez un problème de confection d'horaires de travail où un employé doit travailler pendant une partie de la journée. On crée une séquence de variable $X_1 ... X_n$ où $X_i = 1$ indique que l'employé travaille au temps i et $X_i = 0$ indique que l'employé ne travaille pas au temps i.

- On désire que l'employé se présente au travail et quitte sans avoir à revenir. On désire donc des séquences de la forme $0^a 1^b 0^c$, c'est-à-dire une séquence de 0 (possiblement vide) suivie d'une séquence de 1 (possiblement vide) suivie d'une séquence de 0 (possiblement vide).

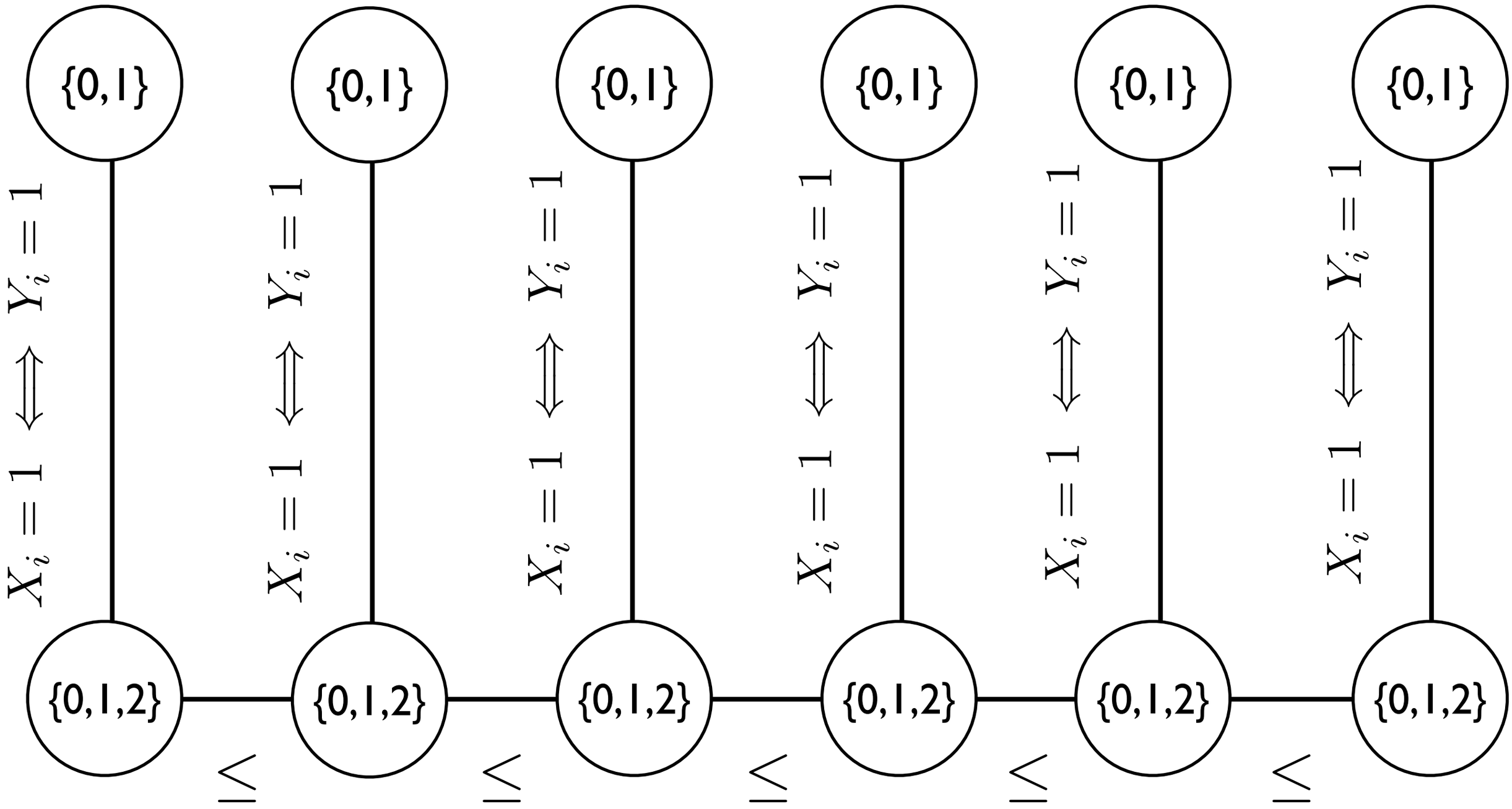
Modèle

- On crée deux séquences de variables $X_1, ..., X_n$ et $Y_1, ..., Y_n$.
- Le domaine de X_i est {0, 1}.
- Le domaine de Y_i est {0, 1, 2}.
- On ajoute les contraintes suivantes.

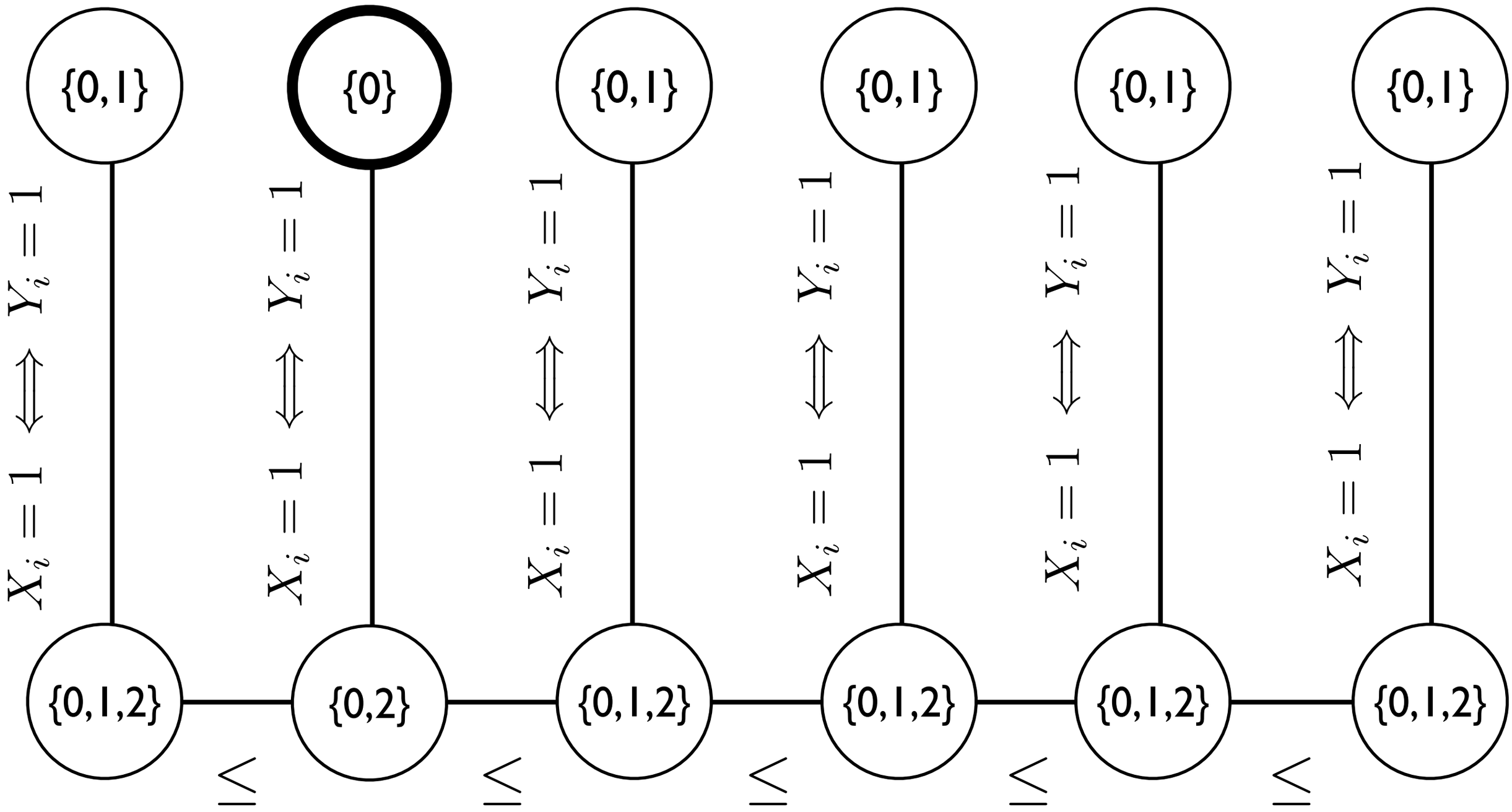
$$X_i = 1 \iff Y_i = 1$$

$$Y_i \leq Y_{i+1}$$

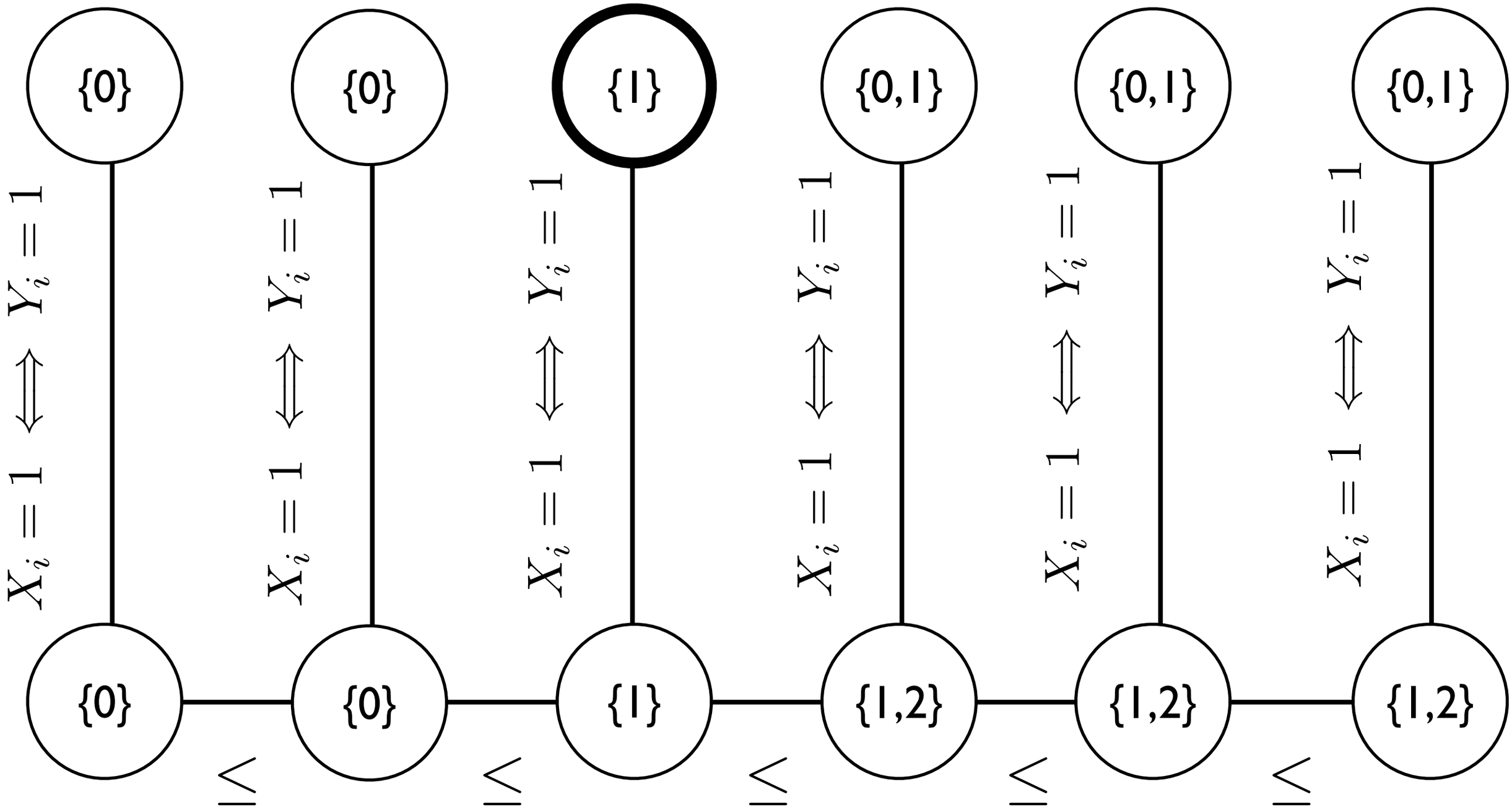
Graphe de contraintes



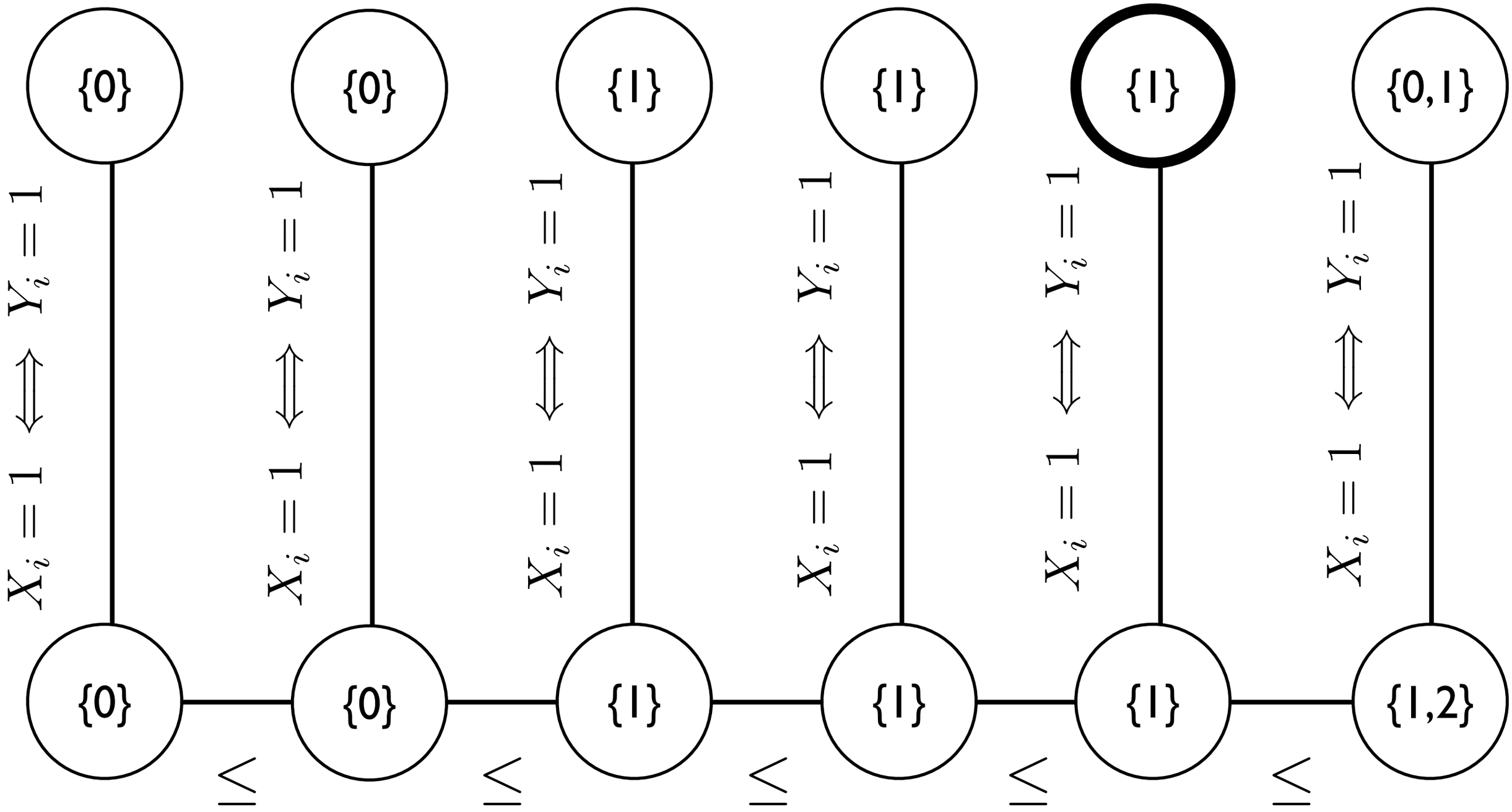
Graphe de contraintes



Graphe de contraintes



Graphe de contraintes



Avantages du modèle en arbre

- En encodant une partie du problème sous la forme d'un arbre, on s'assure que le filtrage est fort et qu'on réduit efficacement l'espace de recherche.
- Avec un mauvais modèle, le solveur pourrait faire des retours arrière seulement pour tenter de générer un horaire de travail valide.
- Avec une structure d'arbre, les choix faits par le solveur mènent nécessairement à un horaire de travail valide. Il ne lui reste qu'à trouver un horaire valide compatible avec les autres contraintes du problème.

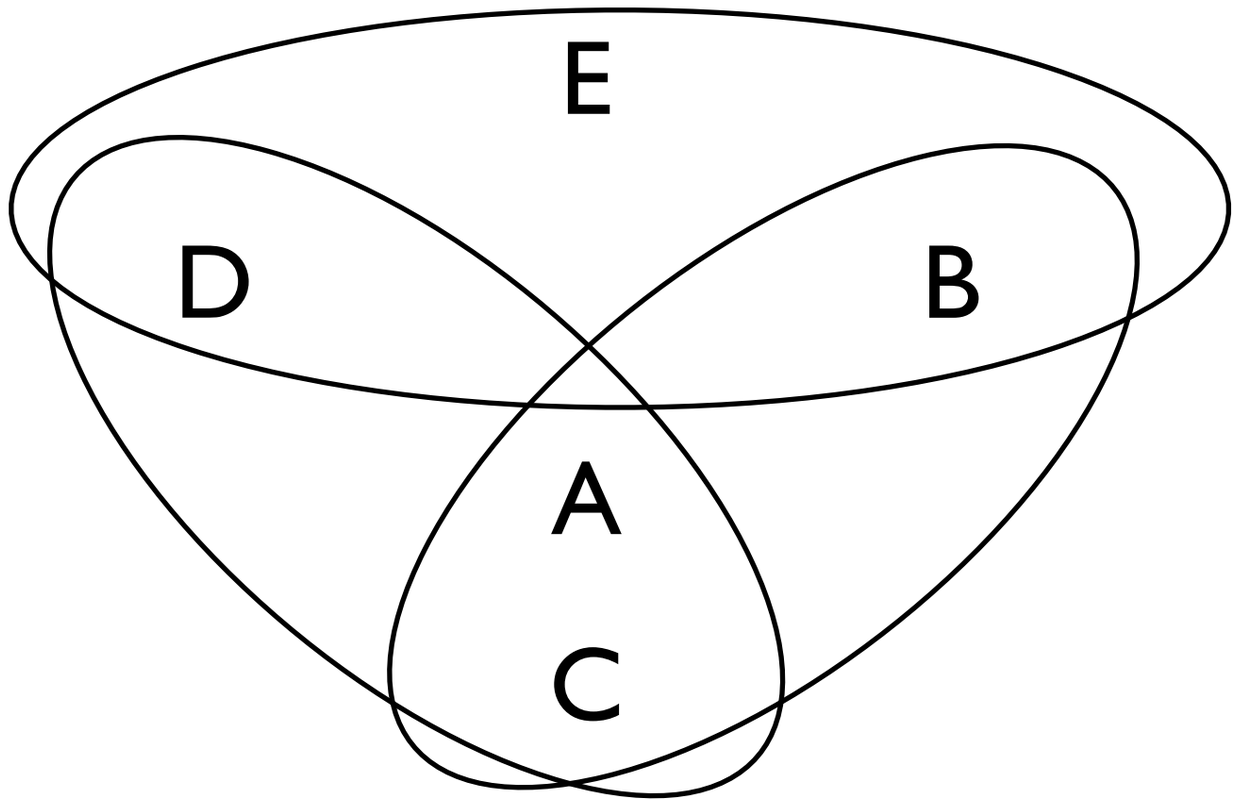
Les hypergraphes

- Un hypergraphe est un tuple (V, E) où V est un ensemble de noeuds et E est un ensemble d'hyperarêtes (ou simplement arête).
- Une hyperarête est un sous-ensemble de noeuds.
- Une arête $e \in E$ est adjacente à un noeud $n \in V$ si le noeud n est dans l'ensemble e.
- Lorsque toutes les hyperarêtes d'un graphe sont adjacentes à exactement deux noeuds, l'hypergraphe est en fait un graphe.

L'hypergraphe de contraintes

- L'hypergraphe de contraintes associé à un problème de satisfaction de contraintes est défini de la façon suivante.
- Les variables du problème sont les noeuds de l'hypergraphe.
- Les portées des contraintes sont les hyperarêtes de l'hypergraphe.

Exemple d'hypergraphe de contraintes



$$A + B = C$$

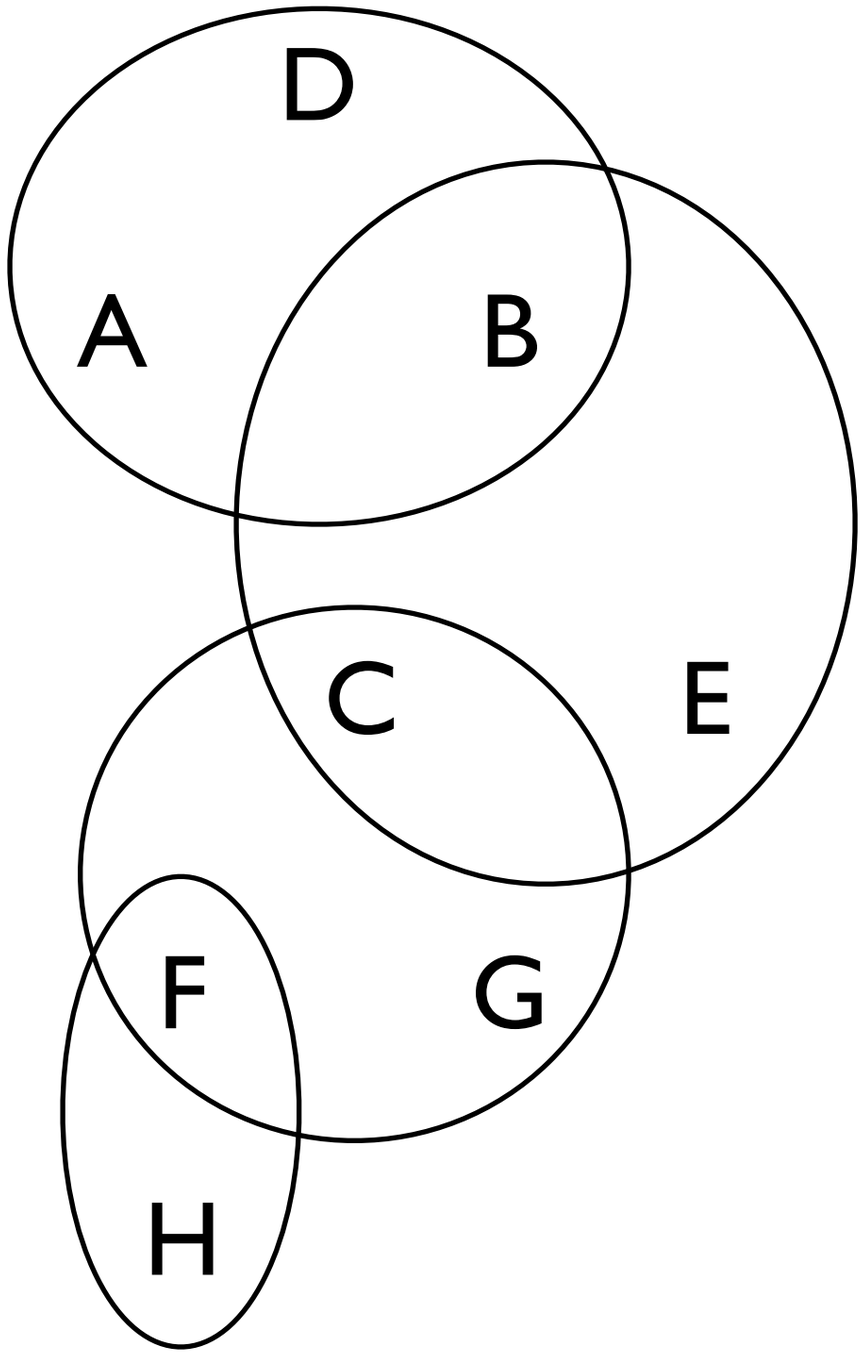
$$A + C = D$$

$$D - B = E$$

Les hyperarbres

- Un hyperarbre est un hypergraphe remplissant les conditions suivantes:
  - Deux arêtes se croisent sur au plus un noeud.
  - Il n'y a pas de cycle, c'est-à-dire qu'il n'y a pas de séquence de noeuds $n_1, n_2, ..., n_k, n_1$ où seuls le premier et le dernier noeud sont répétés et où n_i et n_{i+1} sont adjacents.
- **Théorème**: Si l'hypergraphe de contraintes est un hyperarbre, alors un solveur qui applique la cohérence de domaines sur les contraintes peut résoudre le problème sans retour arrière.
- **Preuve**: La même preuve que pour les arbres.

Un arbre arithmétique



$$A + B = D$$

$$C + E = B$$

$$F * G = C$$

$$F = \sqrt{H}$$

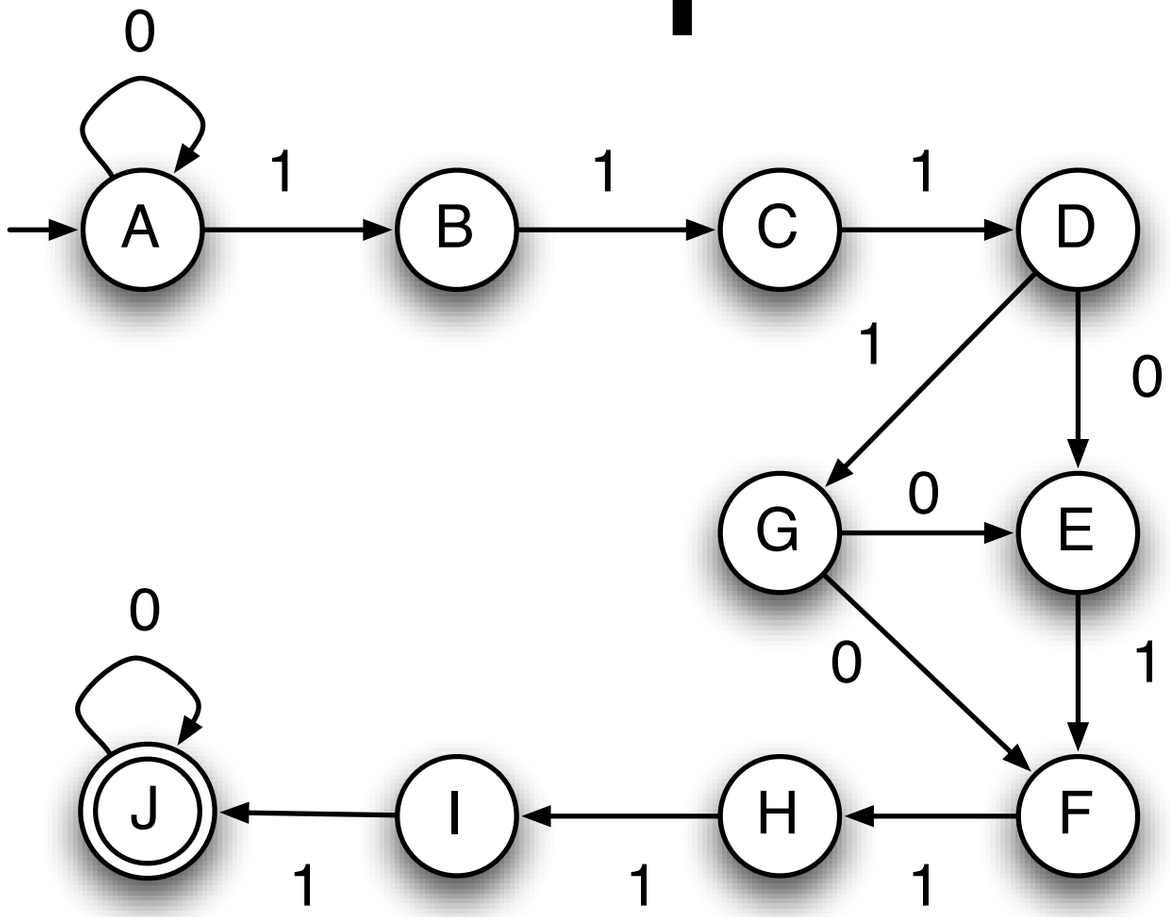
Note: Ce problème de satisfaction de contraintes est une décomposition de la contrainte

$$A + \sqrt{H}G + E = D$$

Automate

- Un automate est un tuple $(\sum, Q, q_0, F, T)$.
  - $\sum$ est un alphabet (un ensemble de caractère).
  - Q est un ensemble d'états.
  - $q_0 \in Q$ est l'état initial.
  - $F \subseteq Q$ est l'ensemble des états finaux.
  - $T \subseteq Q \times \sum \times Q$ est un ensemble de transitions.
- Une séquence de caractères $c_1, c_2, ..., c_n$ est reconnue par l'automate s'il existe une séquence d'états $q_1, q_2, ..., q_n$ tel que $(q_{i-1}, c_i, q_i) \in T$ et $q_n \in F$.

Exemple d'automate



$\Sigma = \{0, 1\}$
$Q = \{A, B, C, ..., J\}$
$q_0 = A$
$F = \{J\}$

	T	
A	0	A
A	1	B
B	1	C
C	1	D
D	1	G
D	0	E
E	1	F
F	1	H
G	0	E
G	0	F
H	1	I
I	1	J
J	0	J

Exemple d'automate

- Cet automate accepte les séquences de la forme 0*111(01|10|101)1110*
- Cette séquence peut représenter l'horaire de travail d'un employé pour une journée de travail.
  - Un 0 indique que l'employé ne travaille pas pendant une heure
  - Un 1 indique que l'employé travaille pendant une heure
  - Un employé travaille donc 7 ou 8 heures dans sa journée.
  - Il a le droit à une pause d'une heure dans sa journée.
  - Il ne peut pas travailler plus de 4 heures consécutives.

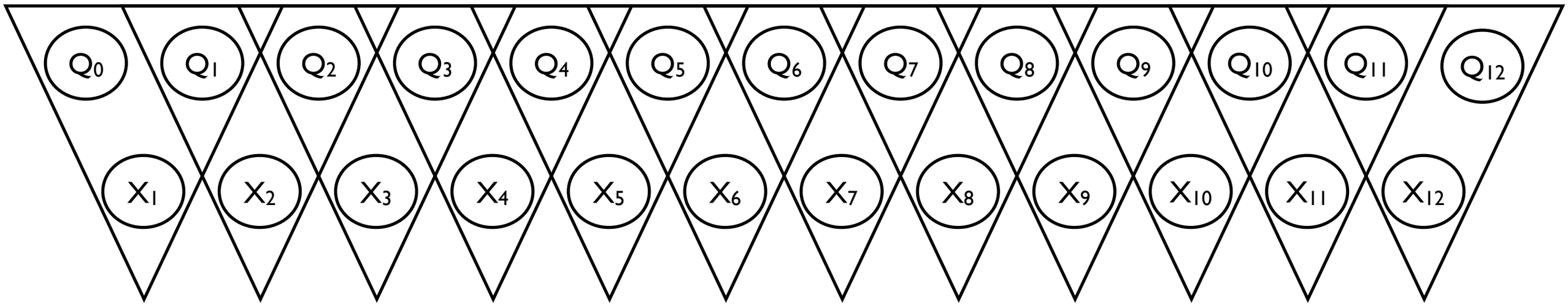
La contrainte *Regular*

- La contrainte Regular(X_1, ..., X_n, A) a dans sa portée une séquence de variable X_1 et X_n. Elle a aussi un automate A qui n'est pas une variable, mais un paramètre de la contrainte.
- La contrainte Regular est satisfaite si la séquence X_1, ..., X_n est reconnue par l'automate.
- Il existe un algorithme de filtrage qui applique la cohérence de domaine à la contrainte Regular.
- Cette contrainte peut cependant être encodée avec d'autres contraintes formant une structure traitable en forme d'arbre.

Encoder l'automate avec des contraintes

- Considérons une séquence de n variables $X_1, ..., X_n$ qui doit être reconnue par un automate $A = (\Sigma, Q, q_0, F, T)$.
- On a donc $dom(X_i) = \Sigma$
- On crée n+1 variables $Q_0 ... Q_n$ avec les domaines
  - $dom(Q_0) = \{q_0\}$
  - $dom(Q_i) = Q$ pour $0 < i < n$
  - $dom(Q_n) = F$
- On ajoute les contraintes Tableau suivantes: $(Q_{i-1}, X_i, Q_i) \in T$.

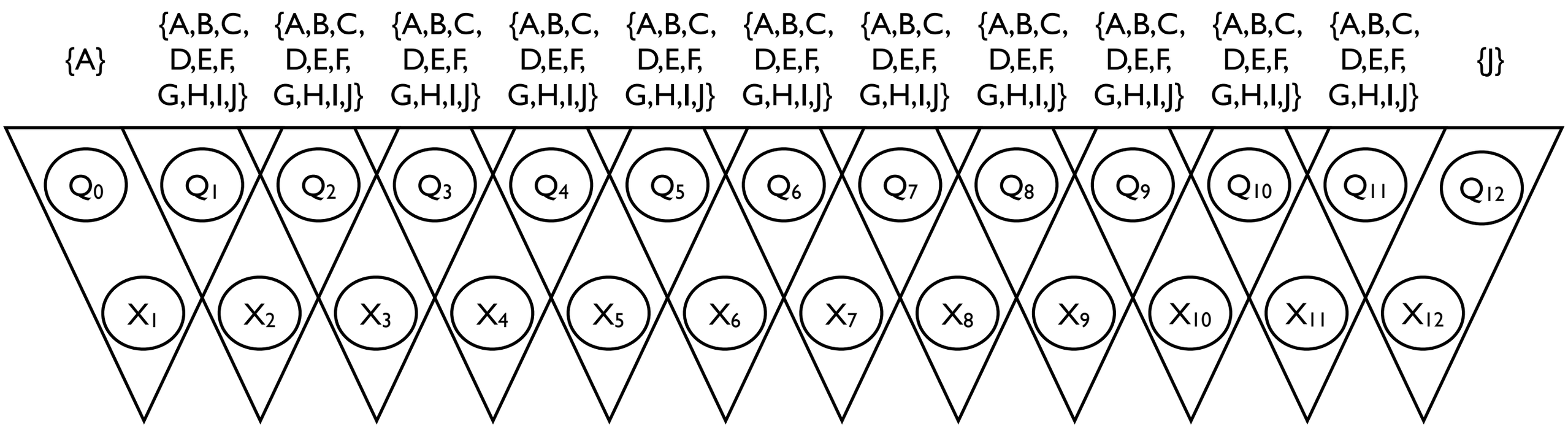
La structure d'arbre d'un langage régulier



$$\textsc{Tableau}(Q_{i-1}, X_i, Q_i, T)$$

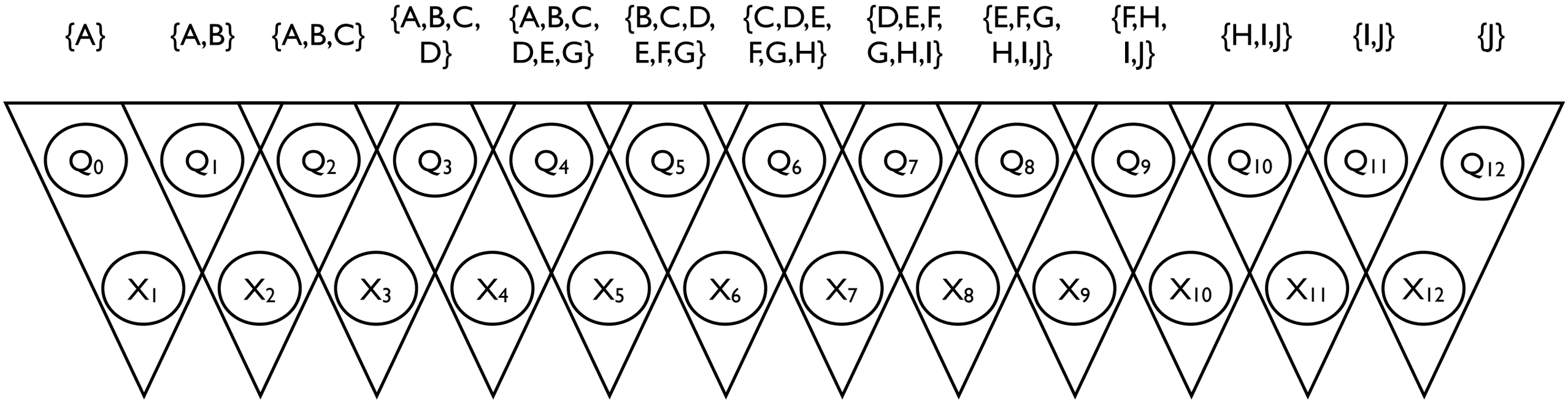
- La contrainte Regular peut donc être décomposée en un ensemble de contraintes Tableau. Appliquer la cohérence de domaine sur les contraintes Tableau applique la cohérence de domaine sur la contrainte Regular.

Exemple du filtrage de Regular



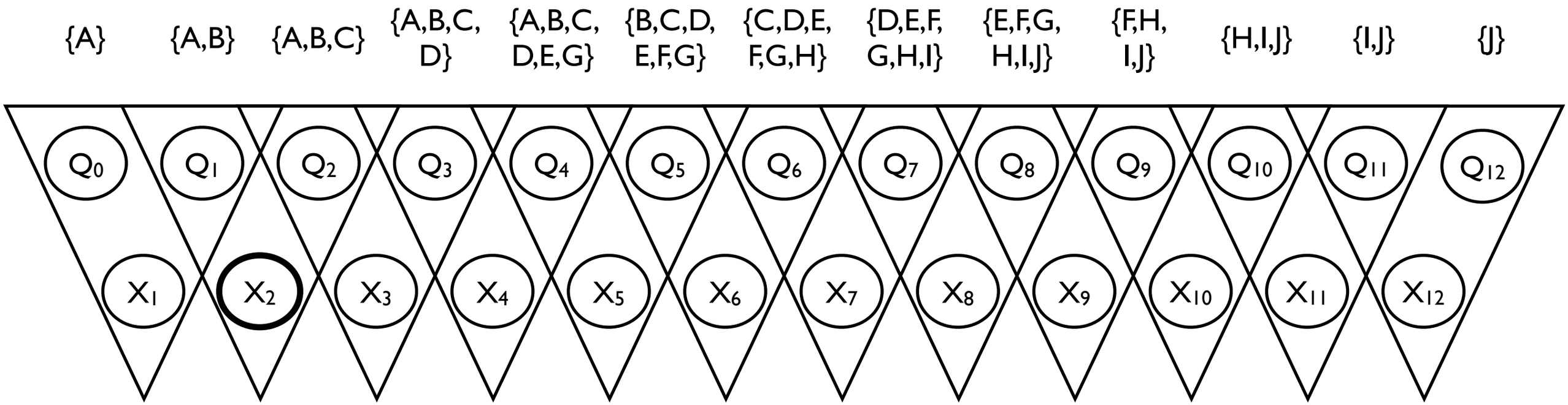
- Graphe de contraintes avec les domaines initiaux pour l'automate de la page 27.

Exemple du filtrage de Regular



- État des domaines après le filtrage des contraintes Tableau.

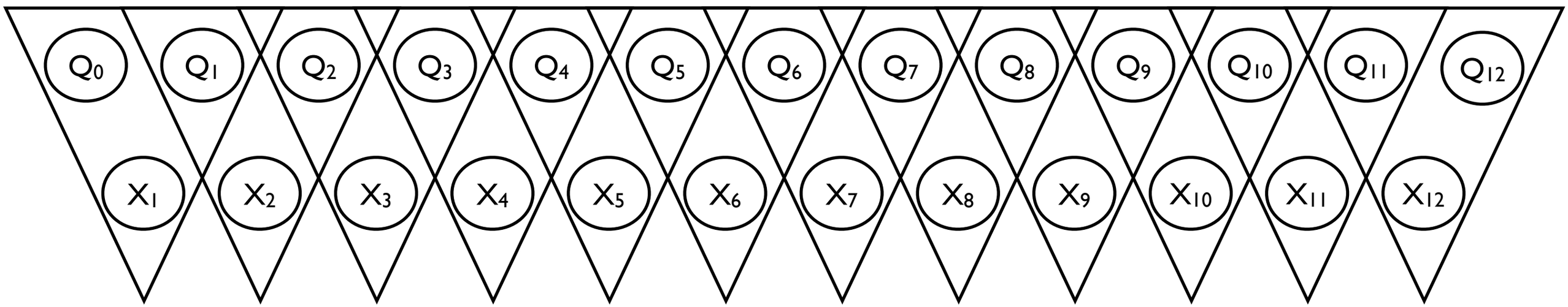
Exemple du filtrage de Regular



- Supposons que l'on branche sur $X_2 = 1$.

Exemple du filtrage de Regular

{A} {A,B} {B,C} {C,D} {D,E,G} {E,F,G} {E,F,H} {F,H,I} {H,I,J} {I,J} {J} {J} {J}

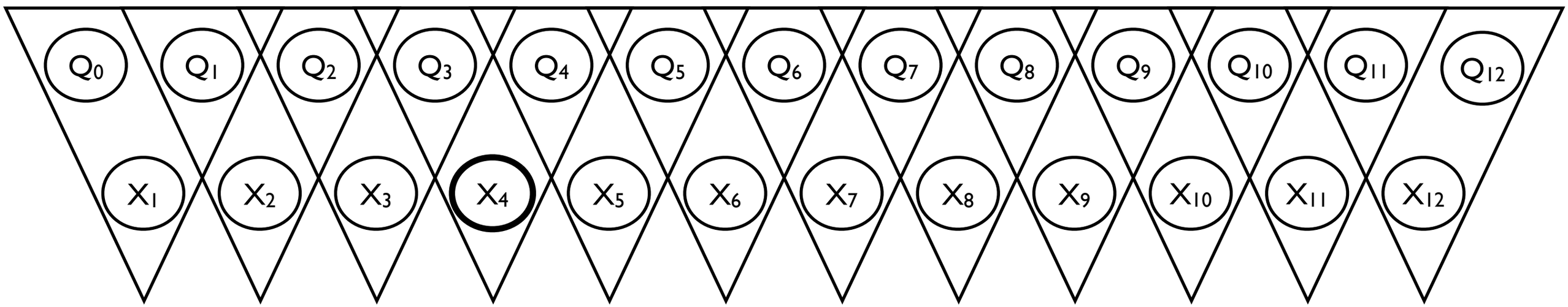


{0,1} {1} {1} {0,1} {0,1} {0,1} {1} {1} {0,1} {0,1} {0} {0}

- Supposons que l'on branche sur $X_2 = 1$.
- Voici l'état des domaines après le filtrage.

Exemple du filtrage de Regular

{A} {A,B} {B,C} {C,D} {D,E,G} {E,F,G} {E,F,H} {F,H,I} {H,I,J} {I,J} {J} {J} {J}

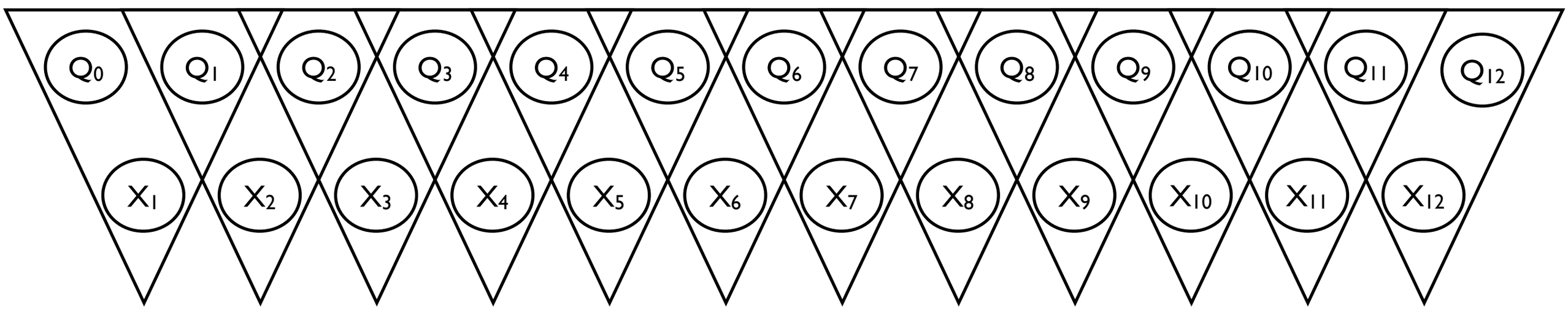


{0,1} {1} {1} **{0}** {0,1} {0,1} {1} {1} {0,1} {0,1} {0} {0}

- Supposons que l'on branche sur $X_4 = 0$.

Exemple du filtrage de Regular

{A} {B} {C} {D} {E} {F} {H} {I} {J} {J} {J} {J} {J}



{1} {1} {1} {0} {1} {1} {1} {1} {0} {0} {0} {0}

- Supposons que l'on branche sur $X_4 = 0$.
- Voici l'état des domaines après le filtrage.

Ordre lexicographique

- Il arrive qu'une solution soit exprimée par un tableau de valeurs, mais que l'ordre des lignes du tableau ait peu d'importance.
- C'est le cas d'un horaire de travail où deux lignes du tableau peuvent être permutées afin de générer une solution équivalente.
- Dans les horaires ci-dessous, les employés 3 et 4 sont permutés.
- Afin de limiter l'arbre de recherche aux solutions non symétriques, on peut imposer un ordre lexicographique sur les lignes comme c'est le cas sur le tableau de gauche.

Heure	9h	10h	11h	12h	13h	14h	15h	16h	17h	18h	19h	20h
Employé 1	1	1	1		1	1	1	1				
Employé 2		1	1	1	1		1	1	1			
Employé 3		1	1	1	1		1	1	1	1		
Employé 4				1	1	1		1	1	1	1	
Employé 5					1	1	1		1	1	1	1
Nb d'employés	1	3	3	3	5	3	4	4	4	3	2	1
Nb d'employés souhaité	1	2	3	4	5	4	2	3	4	4	2	1
Perte	0 $	20 $	0 $	20 $	0 $	20 $	40 $	20 $	0 $	20 $	0 $	0 $

Heure	9h	10h	11h	12h	13h	14h	15h	16h	17h	18h	19h	20h
Employé 1	1	1	1		1	1	1	1				
Employé 2		1	1	1	1		1	1	1			
Employé 3				1	1	1		1	1	1	1	
Employé 4		1	1	1	1		1	1	1	1		
Employé 5					1	1	1		1	1	1	1
Nb d'employés	1	3	3	3	5	3	4	4	4	3	2	1
Nb d'employés souhaité	1	2	3	4	5	4	2	3	4	4	2	1
Perte	0 $	20 $	0 $	20 $	0 $	20 $	40 $	20 $	0 $	20 $	0 $	0 $

Ordre lexicographique

- Soit $X_1, ..., X_n$ et $Y_1, ... Y_n$ deux séquences de variables.
- La séquence $X_1, ..., X_n$ est lexicographiquement plus petite que la séquence $Y_1, ... Y_n$ s'il existe un index P tel que $X_P < Y_P$ et $X_i = Y_i$ pour $i < P$.
- On écrit $X_1, ..., X_n \leq Y_1, ..., Y_n$.
- Exemple: *abbcd* ≤ *abcba*

Une structure traitable

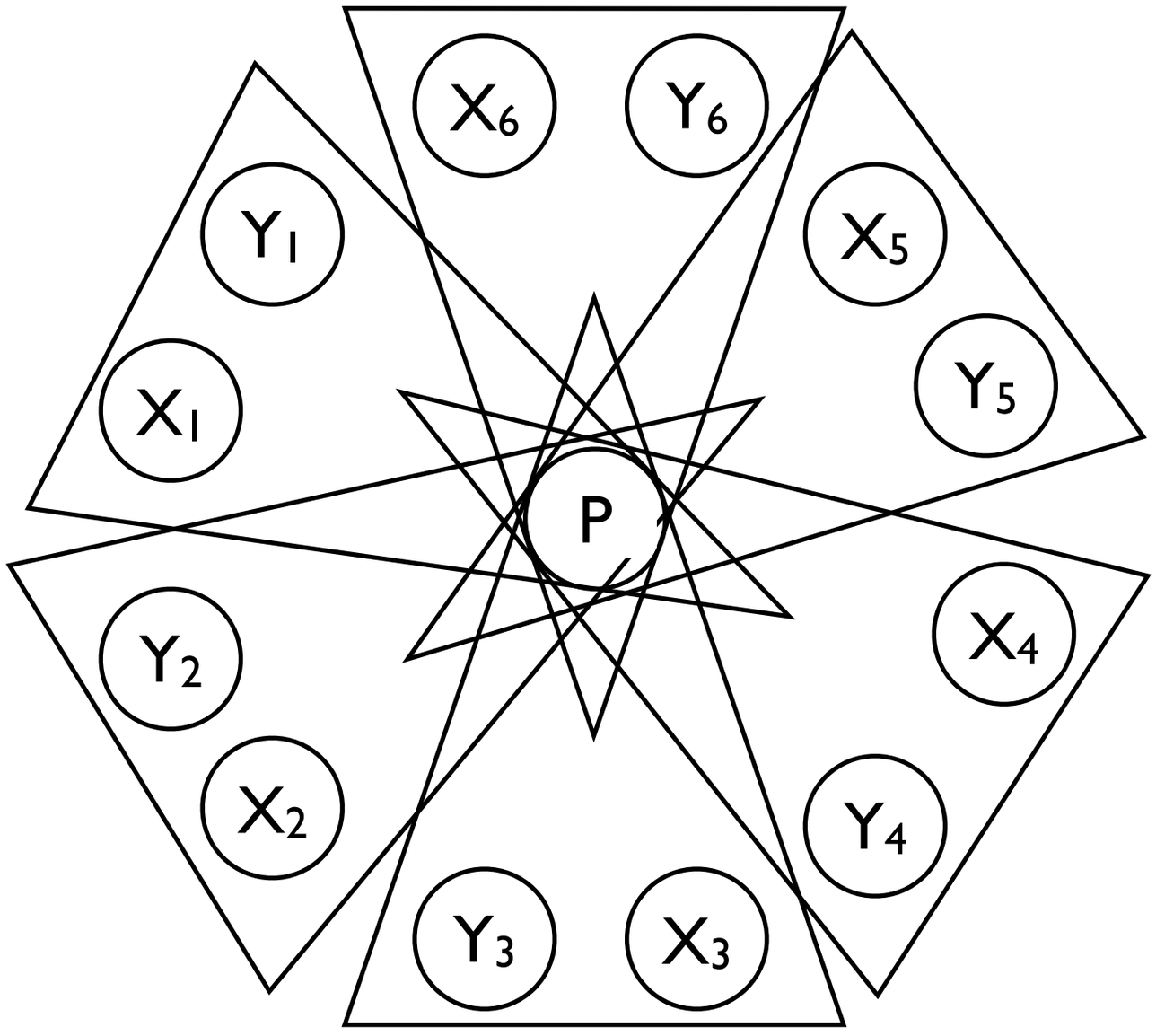
- On peut construire un modèle qui contraint une séquence de variables à être lexicographiquement plus petite qu'une autre séquence de variables.
- Soit P une variable indiquant le plus petit index où $X_P < Y_P$.
- Pour permettre deux séquences identiques, on permet P = n+1.
- Pour que la séquence $X_1 ... X_n$ soit lexicographiquement plus petite que la séquence de $Y_1 ... Y_n$, il faut que $X_i = Y_i$ pour tous les i < P.
- On définit la contrainte C de la façon suivante.

$$C(X_i, Y_i, P) \iff (i < P \Rightarrow X_i = Y_i) \wedge (i = P \Rightarrow X_i < Y_i)$$

- Cette contrainte s'assure que les paires de variables (X_i, Y_i) sont égales si i < P et que $X_P < Y_P$.

Ordre lexicographique

- Les contraintes encodant un ordre lexicographique forment une structure traitable en forme d'hyperarbre.
- Appliquer la cohérence de domaine sur chaque contrainte applique la cohérence de domaine sur la contrainte lexicographique.



Exemple

$$X_1, \ldots, X_7 \prec Y_1, \ldots, Y_7$$

dom(X_1) = {2}	dom(Y_1) = {2}
dom(X_2) = {1,2, 3}	dom(Y_2) = {1,2, 3}
dom(X_3) = {3}	dom(Y_3) = {1,2}
dom(X_4) = {1,2, 3}	dom(Y_4) = {1,2, 3}
dom(X_5) = {1,2,3}	dom(Y_5) = {1,2,3}
dom(X_6) = {1,2, 3}	dom(Y_6) = {1,2, 3}
dom(X_7) = {1,2, 3}	dom(Y_7) = {1,2, 3}

dom(P) = {1, 2, 3, 4, 5, 6, 7}

$$(i < P \Rightarrow X_i = Y_i) \wedge (i = P \Rightarrow X_i < Y_i)$$

- Par la règle du *dilemme constructif*, ces deux implications en impliquent une troisième que nous utilisons pour le filtrage.

$$i \leq P \Rightarrow X_i \leq Y_i$$

- D'après l'état des variables X_3 et Y_3, le domaine de P est filtré à {1, 2}.
- D'après l'état des variables X_1 et Y_1, la valeur 1 est retirée du domaine de P.
- Nous avons donc dom(P)= {2} ce qui retire les valeurs 3 du domaine de X_2 et 1 du domaine de Y_2.

Exemple

$$X_1, \ldots, X_7 \prec Y_1, \ldots, Y_7$$

dom(X_1) = {2}	dom(Y_1) = {2}
dom(X_2) = {1,2}	dom(Y_2) = {2, 3}
dom(X_3) = {3}	dom(Y_3) = {1,2}
dom(X_4) = {1,2, 3}	dom(Y_4) = {1,2, 3}
dom(X_5) = {1,2,3}	dom(Y_5) = {1,2,3}
dom(X_6) = {1,2, 3}	dom(Y_6) = {1,2, 3}
dom(X_7) = {1,2, 3}	dom(Y_7) = {1,2, 3}

dom(P) = {2}

$$(i < P \Rightarrow X_i = Y_i) \wedge (i = P \Rightarrow X_i < Y_i)$$

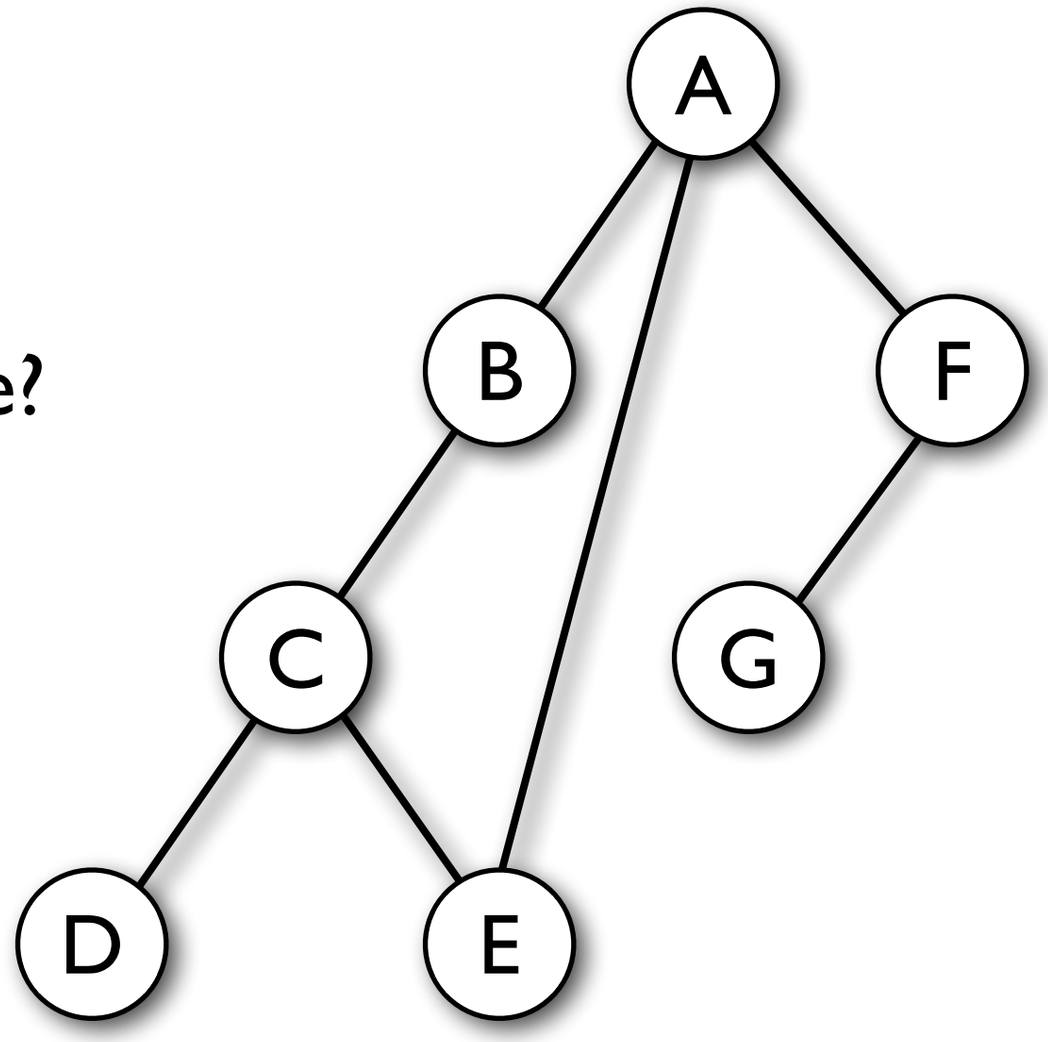
- Par la règle du *dilemme constructif*, ces deux implications en impliquent une troisième que nous utilisons pour le filtrage.

$$i \leq P \Rightarrow X_i \leq Y_i$$

- D'après l'état des variables X_3 et Y_3, le domaine de P est filtré à {1, 2}.
- D'après l'état des variables X_1 et Y_1, la valeur 1 est retirée du domaine de P.
- Nous avons donc dom(P)= {2} ce qui retire les valeurs 3 du domaine de X_2 et 1 du domaine de Y_2.

La largeur d'un arbre

- Ce graphe est-il un arbre?
- Sinon, ce graphe est-il presque un arbre?



La largeur d'un arbre

- Nous définissons une mesure, appelée **la largeur d'un arbre**, qui permettra de dire à quel point un graphe a une structure semblable à celle d'un arbre.

- La largeur d'un arbre nous donnera également un indice de la difficulté du problème que l'on désire résoudre.

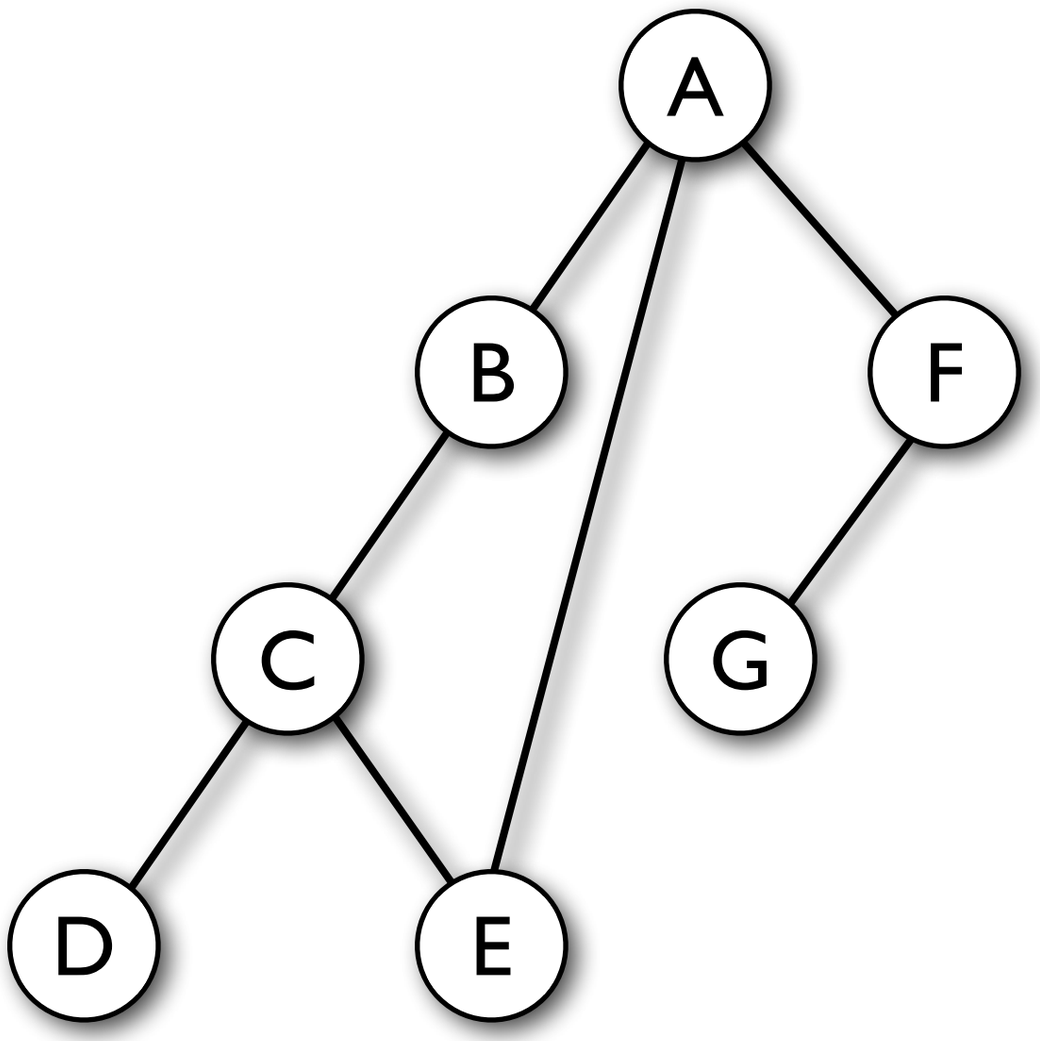
- Lorsque la largeur est bornée par une constante, le problème peut être résolu en temps polynomial.

Décomposition en arbre

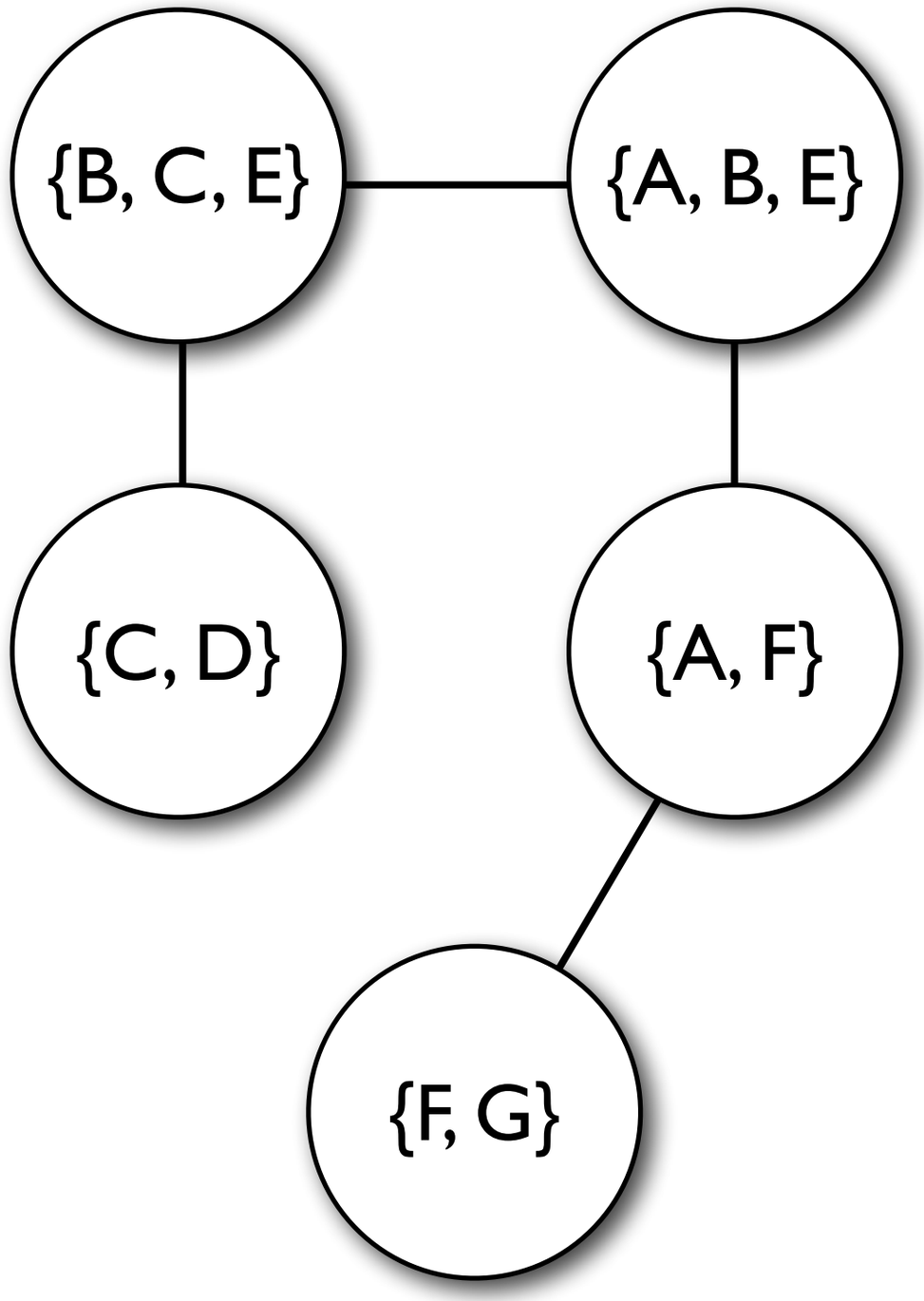
- On décompose un graphe de contraintes G en arbre T de la façon suivante.
- Chaque noeud de l'arbre T est identifié par un sous-ensemble des noeuds du graphe G. Soit i un noeud dans T, on représente par N_i le sous-ensemble de noeuds dans G associé au noeud i.
- Pour chaque arête (a, b) dans G, il existe au moins un noeud i dans T où $\{a, b\} \subseteq N_i$.
- Soit i et j deux noeuds de l'arbre et soit k un noeud de l'arbre se trouvant sur l'unique chemin reliant i à j. On a $N_i \cap N_j \subseteq N_k$.
  - Autrement dit, si un noeud dans G se trouve à deux endroits dans T, il doit également se trouver sur tout le chemin reliant un endroit à l'autre.

Exemple

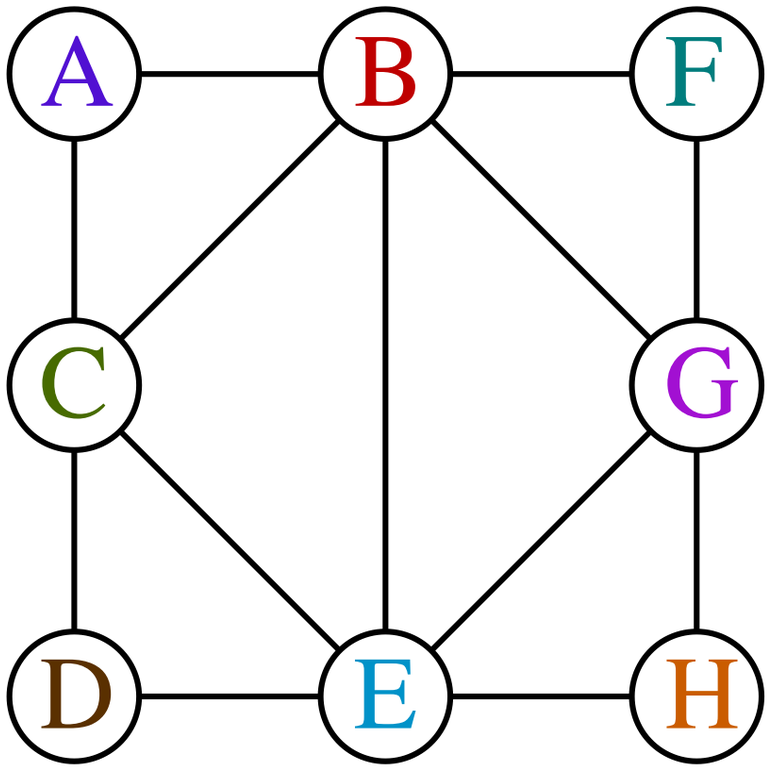
G

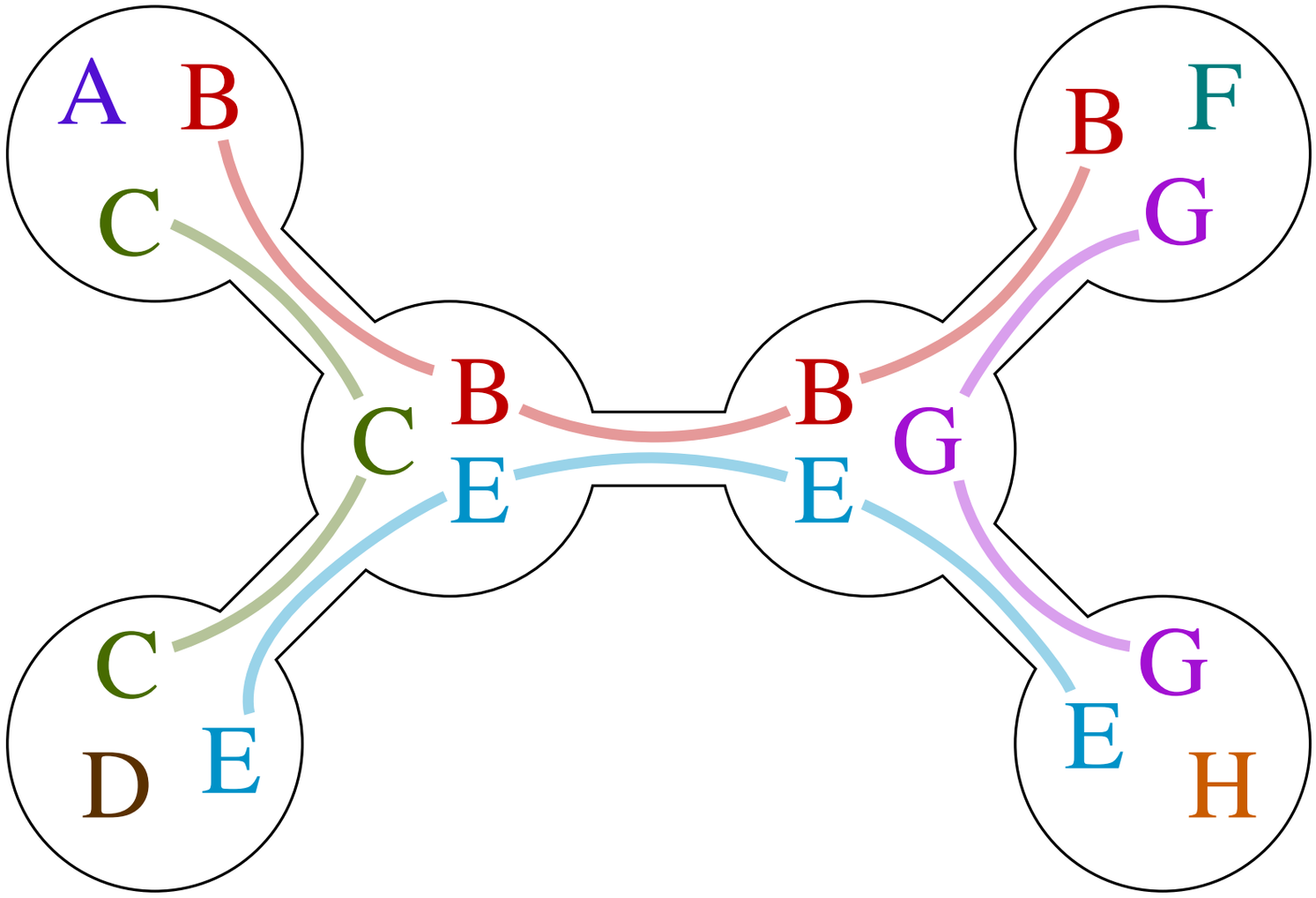


T



Autre exemple





Source: Wikipédia, rubrique « treewidth », janvier 2014.

Multiplicité des décompositions

- Il existe plusieurs façons de décomposer un graphe en arbre.
- La décomposition la plus triviale est celle ou l'arbre T n'a qu'un seul noeud étiqueté avec l'ensemble des noeuds du graphe G.
- Cependant, il existe des décompositions qui sont meilleures que d'autres.

La largeur d'un arbre.

- La **largeur d'une décomposition** en arbre est la cardinalité du plus grand ensemble associé à un noeud moins un.
- Les deux décompositions précédentes ont donc une largeur de 2.
- La **largeur d'arbre d'un graphe** est la plus petite largeur de ses décompositions.
- Il est malheureusement NP-Difficile de calculer la largeur d'arbre d'un graphe. On se contentera donc de dire que la largeur d'arbre d'un graphe est la plus petite largeur trouvée.

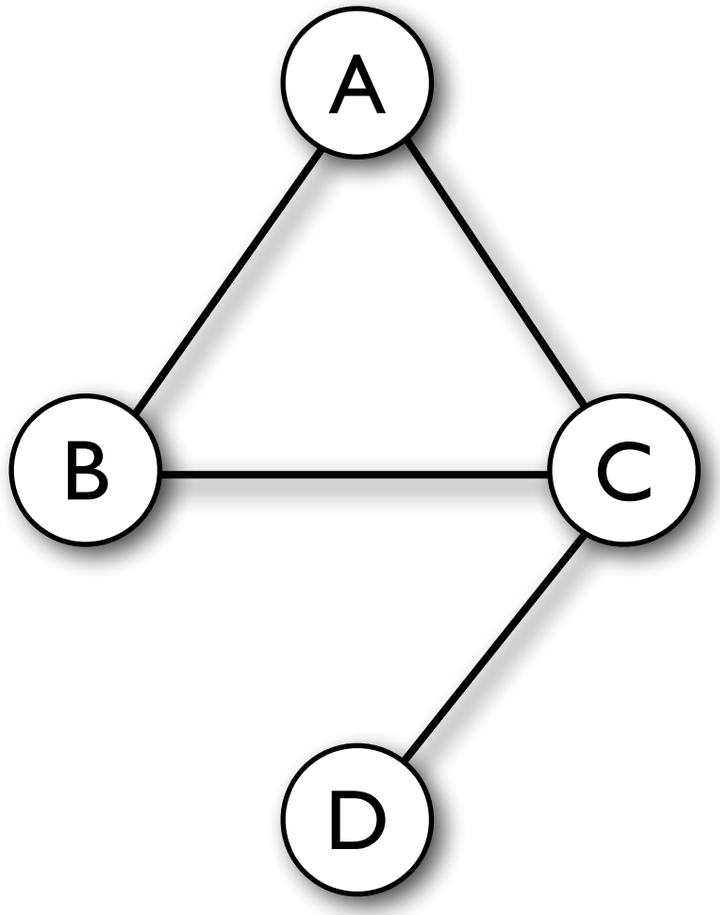
Problème avec une largeur d'arbre bornée

- Il est possible de résoudre en temps polynomial les problèmes dont le graphe de contraintes a une largeur d'arbre bornée par une constante *k*.
- Soit N_i l'ensemble des variables associées au noeud i.
- Chaque noeud i de l'arbre devient un problème de satisfaction de contraintes en soit formé des variables dans N_i et des contraintes dont la portée est un sous-ensemble de N_i. Nous appelons S_i l'ensemble des solutions du problème associé au noeud i.

Problème avec une largeur d'arbre bornée

- Il est ensuite possible de créer un problème de satisfaction de contraintes où chaque noeud de l'arbre est une variable X_i dont le domaine est l'ensemble des solutions S_i.
- Une arête (i, j) de l'arbre devient une contrainte forçant la solution choisie dans S_i à être compatible à celle choisie dans S_j.

Coloration d'un graphe



$A \neq B$

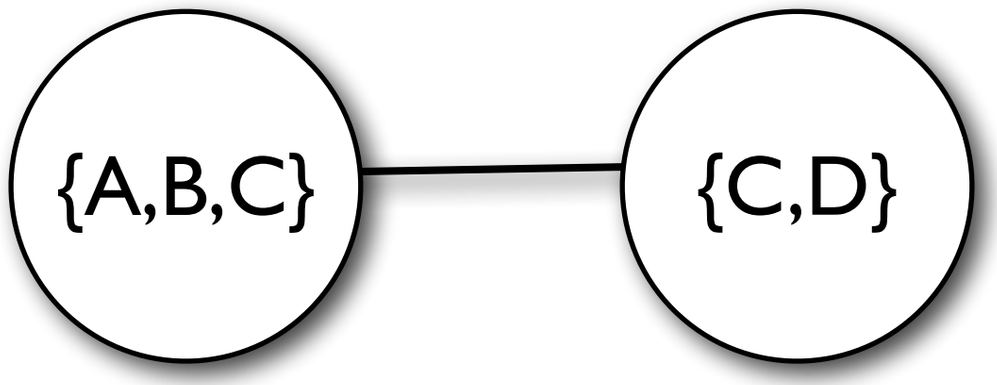
$A \neq C$

$B \neq C$

$D \neq C$

$$\mathrm{dom}(A) = \mathrm{dom}(B) = \mathrm{dom}(C) = \mathrm{dom}(D) = \{1, 2, 3\}$$

Décomposition en arbre



Solutions des sous-problèmes

	1, 2, 3	1, 2
$A \neq B$	1, 3, 2	1, 3
$A \neq C$	2, 1, 3	2, 1
$B \neq C$	2, 3, 1	2, 3
$D \neq C$	3, 1, 2	3, 1
	3, 2, 1	3, 2

$$\mathrm{dom}(A) = \mathrm{dom}(B) = \mathrm{dom}(C) = \mathrm{dom}(D) = \{1, 2, 3\}$$

Nouveau modèle

- Une variable par noeud de l'arbre
  - $X_{abc} \in \{(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)\}$
  - $X_{cd} \in \{(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)\}$
- Une contrainte par arête de l'arbre. Dans notre exemple, la contrainte tableau s'assure que la variable C prend la même valeur dans chaque sous-problème.

X_{abc}	X_{cd}
(1, 2, **3**)	(**3**, 1)
(1, 2, **3**)	(**3**, 2)
(1, 3, **2**)	(**2**, 1)
(1, 3, **2**)	(**2**, 3)
(2, 1, **3**)	(**3**, 1)
(2, 1, **3**)	(**3**, 2)
(2, 3, **1**)	(**1**, 2)
(2, 3, **1**)	(**1**, 3)
(3, 1, **2**)	(**2**, 1)
(3, 1, **2**)	(**2**, 3)
(3, 2, **1**)	(**1**, 2)
(3, 2, **1**)	(**1**, 3)

Complexité

- Quelle est la complexité de résoudre un problème dont la largeur d'arbre est bornée par une constante *k* et dont le plus grand domaine a une cardinalité bornée par une constante *d*?
- Résoudre un sous-problème peut se faire en temps $O(d^{k+1})$. Puisque *d* et *k* sont des constantes, résoudre un sous-problème se fait donc en temps constant!
- Une contrainte tableau aura au plus $d^{k+1} \times d^{k+1} = d^{2k+2}$ entrées et sera donc filtrée en temps constant.
- Finalement, l'arbre a au plus *n* noeuds.
- Si k et d sont des constantes, le problème est résolu en temps linaire: O(n).

Conclusions

- Il existe des problèmes de satisfaction de contraintes pouvant être résolus en temps polynomial par un solveur.
- C'est le cas lorsque le graphe de contraintes est:
  - Un arbre
  - Un hyperarbre
  - Un graphe dont la largeur d'arbre est bornée par une constante.

Conclusions

- Ces structures traitables peuvent être utilisées à même la modélisation de problèmes intraitables.
  - La contrainte Regular peut être décomposée en contraintes Tableau.
  - La contrainte d'ordre lexicographique peut aussi être décomposée en structure d'arbre.
- Un bon modèle pour un problème en est un qui exploite ces structures traitables et en tirent avantage pour obtenir un fort filtrage.

Références

- Handbook of Constraint Programming, sections 7.1.2 et 7.2
- Contrainte Regular: Gilles Pesant: A Regular Language Membership Constraint for Finite Sequences of Variables. CP 2004, pages 482-495.