Compilation de connaissances pour le comptage de modèles

Samuel Thomas

14 décembre 2016

Soutenance de thèse

Représentations des connaissances

Logique propositionnelle :

- Ensemble de variables booléennes PS et symboles $\top$ et $\bot$;
- Ensemble de connecteurs : $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\oplus$.

$$\Phi = x_2 \vee (\neg x_4 \wedge x_8) \qquad || \Phi || = 5$$

Réseaux de contraintes :

- Ensemble de variables $\mathcal{V}$ et leur domaines $\mathcal{D}$;
- Ensemble de contraintes $\mathcal{C}$:

$$N = V_2 + V_4 \leq 4, \text{ avec } D_2 = D_4 = \{1, 2, 3\} \qquad || N || = 6$$

Compilation de « connaissances »

- Résolution (d'instances) de problèmes partageant une même partie fixe
- **Pré-traitement** de la partie fixe en vue d'améliorer l'efficacité globale (temps calcul)
- **Problème clé : vers quel langage compiler ?**
  - Requêtes et transformations utiles
  - Expressivité et efficacité spatiale

Requêtes et transformations propositionnelles

Requêtes :

- **CO** : cohérence
- **VA** : validité
- **CE** : implication clausale
- **IM** : implicant
- **SE** : implication logique
- **EQ** : équivalence logique
- **CT** : comptage de modèles
- **ME** : énumération de modèles

Transformations :

- **CD** : conditionnement
- **FO** : oubli ($\exists X.\alpha$)
- **SFO** : oubli singleton ($X = \{x\}$)
- $\neg$**C** : négation
- $\vee$**C** : disjonction
- $\vee$**BC** : disjonction bornée
- $\wedge$**C** : conjonction
- $\wedge$**BC** : conjonction bornée

Efficacité spatiale

Soient $\mathcal{L}_1$ et $\mathcal{L}_2$ deux langages propositionnels :

- $\mathcal{L}_1$ est au moins aussi **succinct** que $\mathcal{L}_2$, noté $\mathcal{L}_1 \leq_s \mathcal{L}_2$, ssi $\exists p$ un polynôme tq. $\forall \beta \in \mathcal{L}_2$, $\exists \alpha \in \mathcal{L}_1$ tq. $\alpha \equiv \beta$ et $|\alpha| \leq p(|\beta|)$
- $\mathcal{L}_1$ est **strictement plus succinct** que $\mathcal{L}_2$, noté $\mathcal{L}_1 <_s \mathcal{L}_2$, ssi $\mathcal{L}_1 \leq_s \mathcal{L}_2$ et $\mathcal{L}_2 \not\leq_s \mathcal{L}_1$

Comptage de modèles

- Compter les modèles d'une formule propositionnelle est une tâche fondamentale pour plusieurs problèmes d'IA
  - inférence probabiliste
  - planification stochastique
  - ...
- Cependant cette tâche est calculatoirement difficile : #P-complète
- La puissance offerte par le comptage et sa complexité sont reflétées par le **théorème de Toda :**

Seinosuke Toda (prix Gödel 1998) :

$$\mathrm{PH} \subseteq \mathrm{P}^{\#\mathrm{P}}$$

Arbres de décision affine

Objectifs et méthodologie

- **Définir de nouveaux langages propositionnels offrant CT**
  - AFF satisfait **CT** mais est incomplet
  - d-DNNF et ses sous-ensembles (comme $\text{OBDD}_<$, DT) sont complets et satisfont **CT**
- **Idée de base : combiner AFF et DT**

Formules affines

- Une **clause affine** est une XOR-disjonction finie de littéraux (y compris $\bot$ et $\top$)
- Une **formule affine** est une conjonction finie de clauses affines

$$\Sigma = (x \oplus \overline{y} \oplus \overline{z}) \wedge (x \oplus u \oplus \overline{w}) \wedge (w \oplus y \oplus \overline{z})$$

- **Compter le nombre de modèles d'une formule affine est réalisable en temps polynomial**
- Obtenu par triangulation *via* l'algorithme du **pivot de Gauss**

$$\Phi = \left(\begin{array}{ccccc|c} x & y & z & & & 1 \\ x & & & u & w & 0 \\ & y & z & & w & 0 \end{array}\right) \begin{array}{l} (E1) \\ (E2) \\ (E3) \end{array}$$

Formules affines

- Une **clause affine** est une XOR-disjonction finie de littéraux (y compris $\bot$ et $\top$)
- Une **formule affine** est une conjonction finie de clauses affines

$$\Sigma = (x \oplus \overline{y} \oplus \overline{z}) \wedge (x \oplus u \oplus \overline{w}) \wedge (w \oplus y \oplus \overline{z})$$

- **Compter le nombre de modèles d'une formule affine est réalisable en temps polynomial**
- Obtenu par triangulation *via* l'algorithme du **pivot de Gauss**

$$\Phi = \left(\begin{array}{ccccc|c} x & y & z & & & 1 \\ & y & z & u & w & 1 \\ & y & z & & w & 0 \end{array}\right) \begin{array}{l} (E1) \\ (E2' = E1 \oplus E2) \\ (E3) \end{array}$$

Formules affines

- Une **clause affine** est une XOR-disjonction finie de littéraux (y compris $\bot$ et $\top$)
- Une **formule affine** est une conjonction finie de clauses affines

$$\Sigma = (x \oplus \overline{y} \oplus \overline{z}) \wedge (x \oplus u \oplus \overline{w}) \wedge (w \oplus y \oplus \overline{z})$$

- **Compter le nombre de modèles d'une formule affine est réalisable en temps polynomial**
- Obtenu par triangulation *via* l'algorithme du **pivot de Gauss**

$$\Phi = \left(\begin{array}{ccccc|c} x & y & z & & & 1 \\ & y & z & u & w & 1 \\ & & & u & & 1 \end{array}\right) \begin{array}{l} (E1) \\ (E2' = E1 \oplus E2) \\ (E3' = E2' \oplus E3) \end{array}$$

- 3 variables liées : x, y et u, et 2 variables libres : z et w
- Nombre de modèles $= 2^{\#VarsLibres} = 2^2 = 4$

Requêtes et transformations (AFF)

$\mathcal{L}$	CO	VA	CE	IM	EQ	SE	CT	ME
CNF	○	√	○	√	○	○	○	○
d-DNNF	√	√	√	√	?	○	√	√
$\text{OBDD}_<$	√	√	√	√	√	√	√	√
AFF	√	√	√	√	√	√	√	√

$\mathcal{L}$	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
CNF	√	○	√	√	√	•	√	•
d-DNNF	√	○	○	○	○	○	○	?
$\text{OBDD}_<$	√	○	√	○	√	○	√	√
AFF	√	√	√	√	√	!	!	!

Requêtes et transformations (AFF)

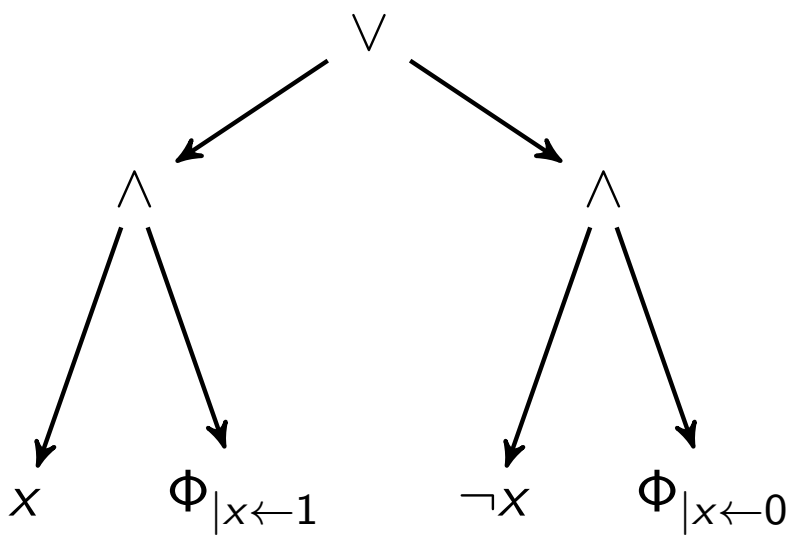
$\mathcal{L}$	CO	VA	CE	IM	EQ	SE	CT	ME
CNF	○	√	○	√	○	○	○	○
d-DNNF	√	√	√	√	?	○	√	√
$\text{OBDD}_<$	√	√	√	√	√	√	√	√
AFF	√	√	√	√	√	√	√	√

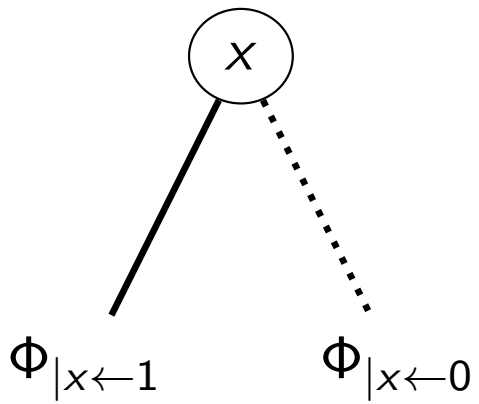
$\mathcal{L}$	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
CNF	√	○	√	√	√	•	√	•
d-DNNF	√	○	○	○	○	○	○	?
$\text{OBDD}_<$	√	○	√	○	√	○	√	√
AFF	√	√	√	√	√	!	!	!

AFF n'est pas complet pour la logique propositionnelle

Arbre de décision (DT)

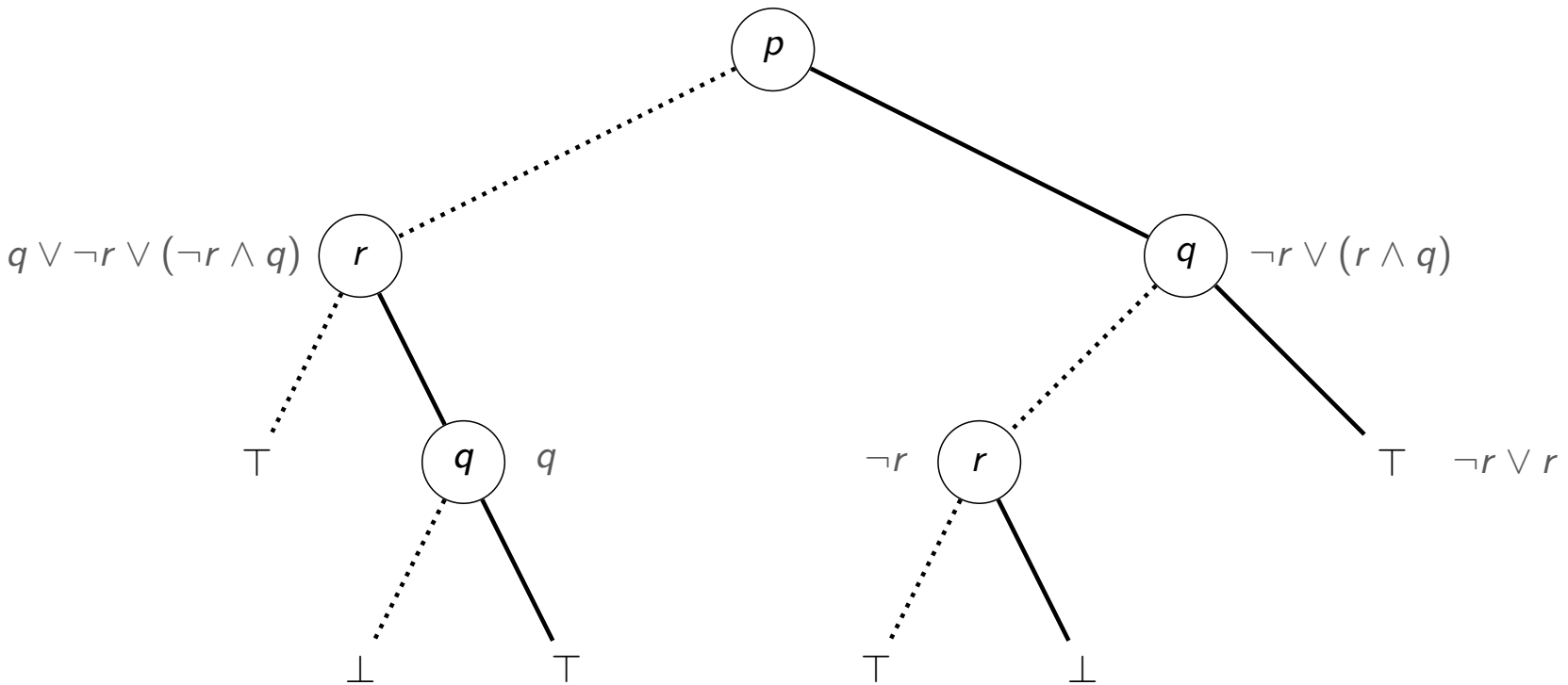
- **Expansion de Shannon** : $\Phi \equiv (x \wedge \Phi_{|x \leftarrow 1}) \vee (\neg x \wedge \Phi_{|x \leftarrow 0})$





Arbre de décision (DT)

- $\Phi = (q \wedge \neg p) \vee \neg r \vee (((\neg p \wedge \neg r) \vee (p \wedge r)) \wedge q)$



Requêtes et transformations (DT)

$\mathcal{L}$	CO	VA	CE	IM	EQ	SE	CT	ME
CNF	○	√	○	√	○	○	○	○
d-DNNF	√	√	√	√	?	○	√	√
$\text{OBDD}_<$	√	√	√	√	√	√	√	√
DT	√	√	√	√	√	√	√	√

$\mathcal{L}$	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
CNF	√	○	√	√	√	•	√	•
d-DNNF	√	○	○	○	○	○	○	?
$\text{OBDD}_<$	√	○	√	○	√	○	√	√
DT	√	○	√	○	√	○	√	√

Expressivité et efficacité spatiale (DT)

DT est **complet** mais est **peu efficace spatialement** :

- $\mathtt{CNF} <_s \mathtt{DT}$
- $\mathtt{DNF} <_s \mathtt{DT}$

Arbre de décision affine (ADT)

- Idée : utiliser une forme **généralisée** de l'expansion de Shannon

$$\Phi \equiv ((x \Leftrightarrow \neg\delta) \wedge \Phi_{|x \leftarrow \neg\delta}) \vee ((x \Leftrightarrow \delta) \wedge \Phi_{|x \leftarrow \delta})$$

pour toute formule propositionnelle Φ et δ et toute variable x

- Dans notre cas Φ est une ECNF et δ est une **clause affine**
- Une ECNF Φ est une conjonction de **clauses étendues** (i.e., des disjonctions de clauses affines)

$$\Phi = (x_1 \vee (x_2 \oplus x_3) \vee (x_1 \oplus \overline{x_3})) \wedge (x_1 \vee x_2 \vee \overline{x_5})$$

- Comme $x \Leftrightarrow \neg\delta \equiv x \oplus \delta$ et $x \Leftrightarrow \delta \equiv \overline{x} \oplus \delta$, on a :

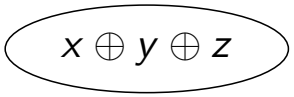
$$\Phi \equiv ((x \oplus \delta) \wedge \Phi_{|x \leftarrow \delta \oplus \top}) \vee ((\overline{x} \oplus \delta) \wedge \Phi_{|x \leftarrow \delta})$$

Arbre de décision affine (ADT) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$

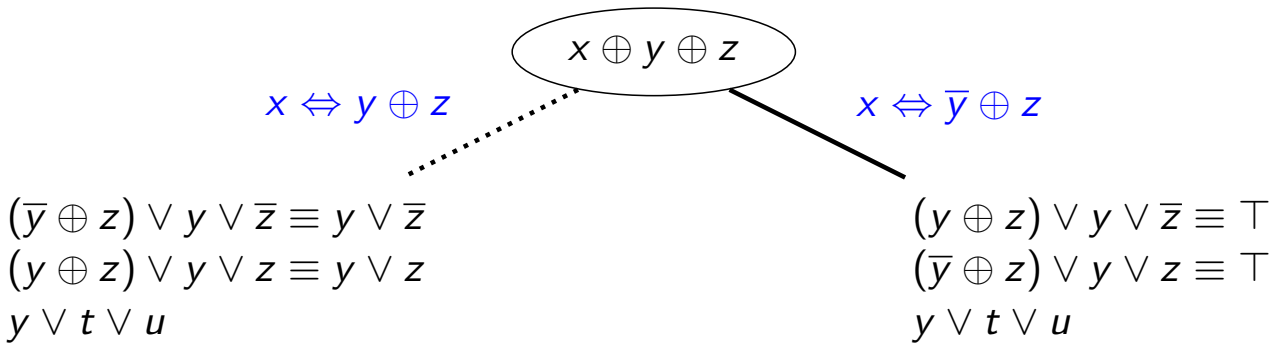
Arbre de décision affine (ADT) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



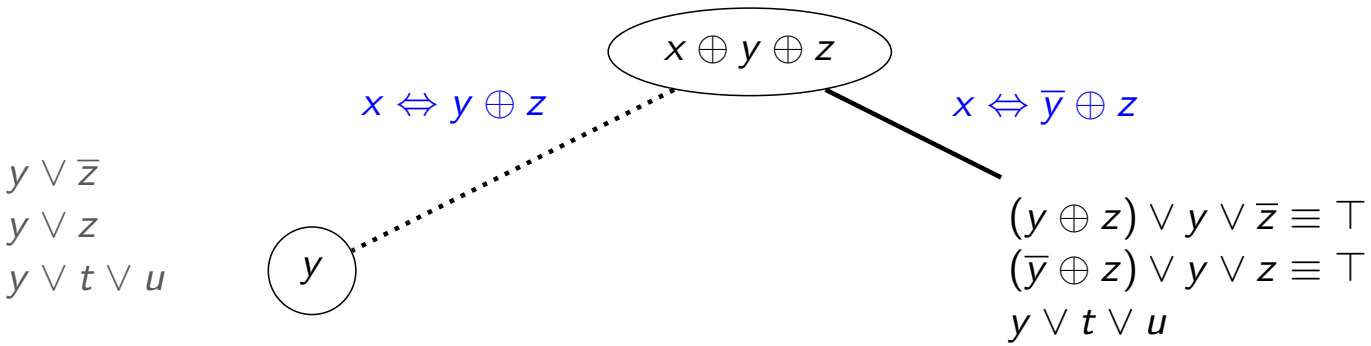
Arbre de décision affine (ADT) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



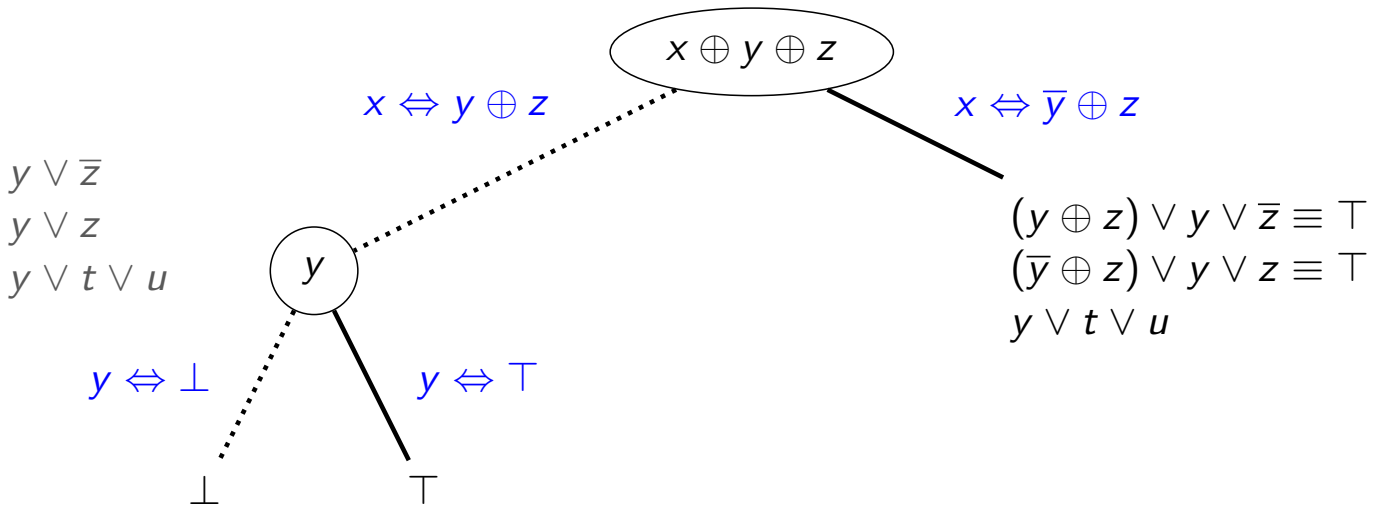
Arbre de décision affine (ADT) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



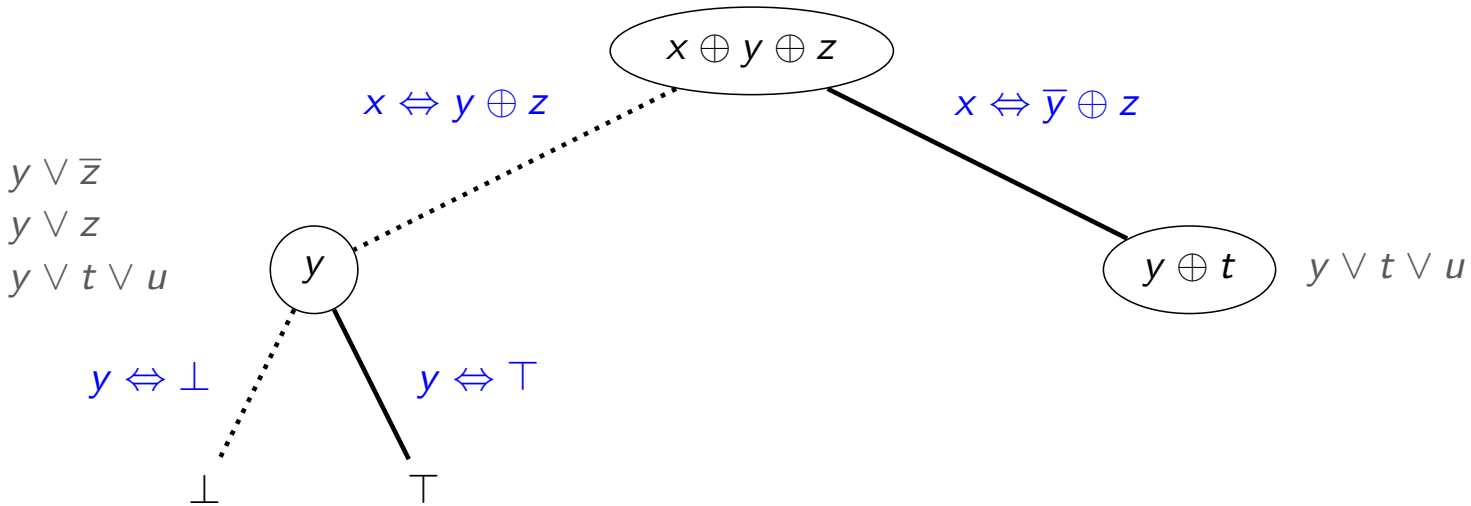
Arbre de décision affine (ADT) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



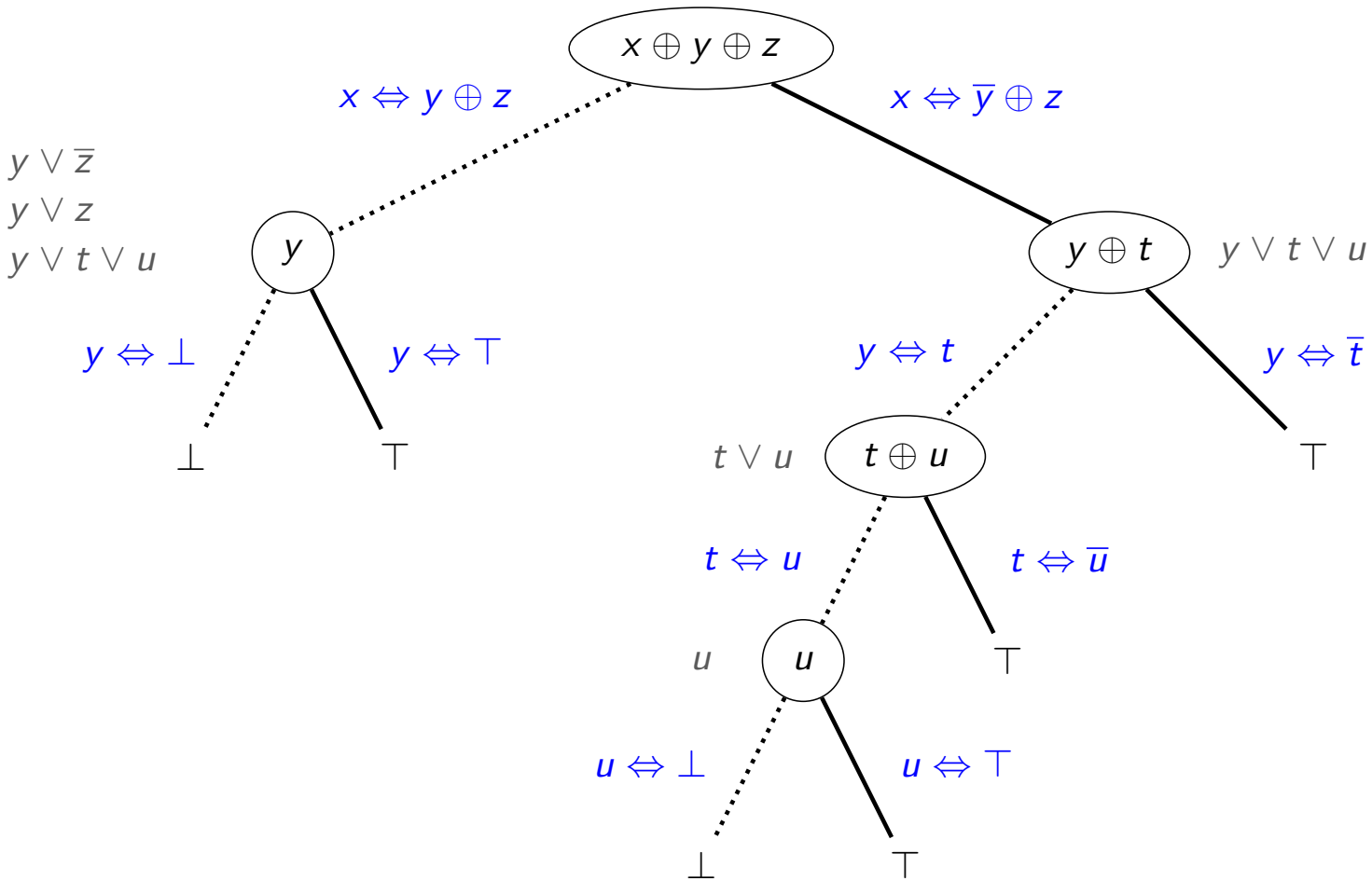
Arbre de décision affine (ADT) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



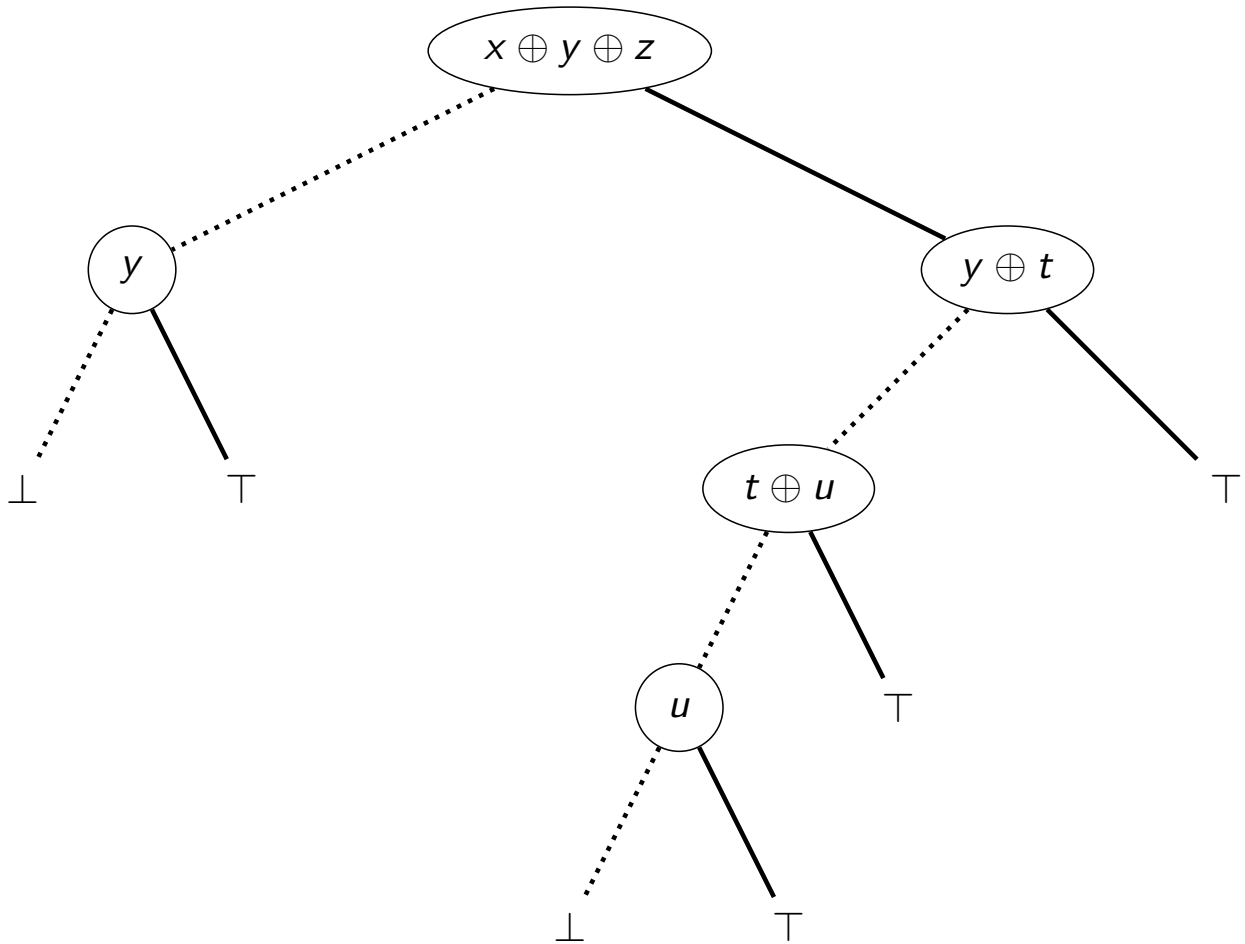
Arbre de décision affine (ADT) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



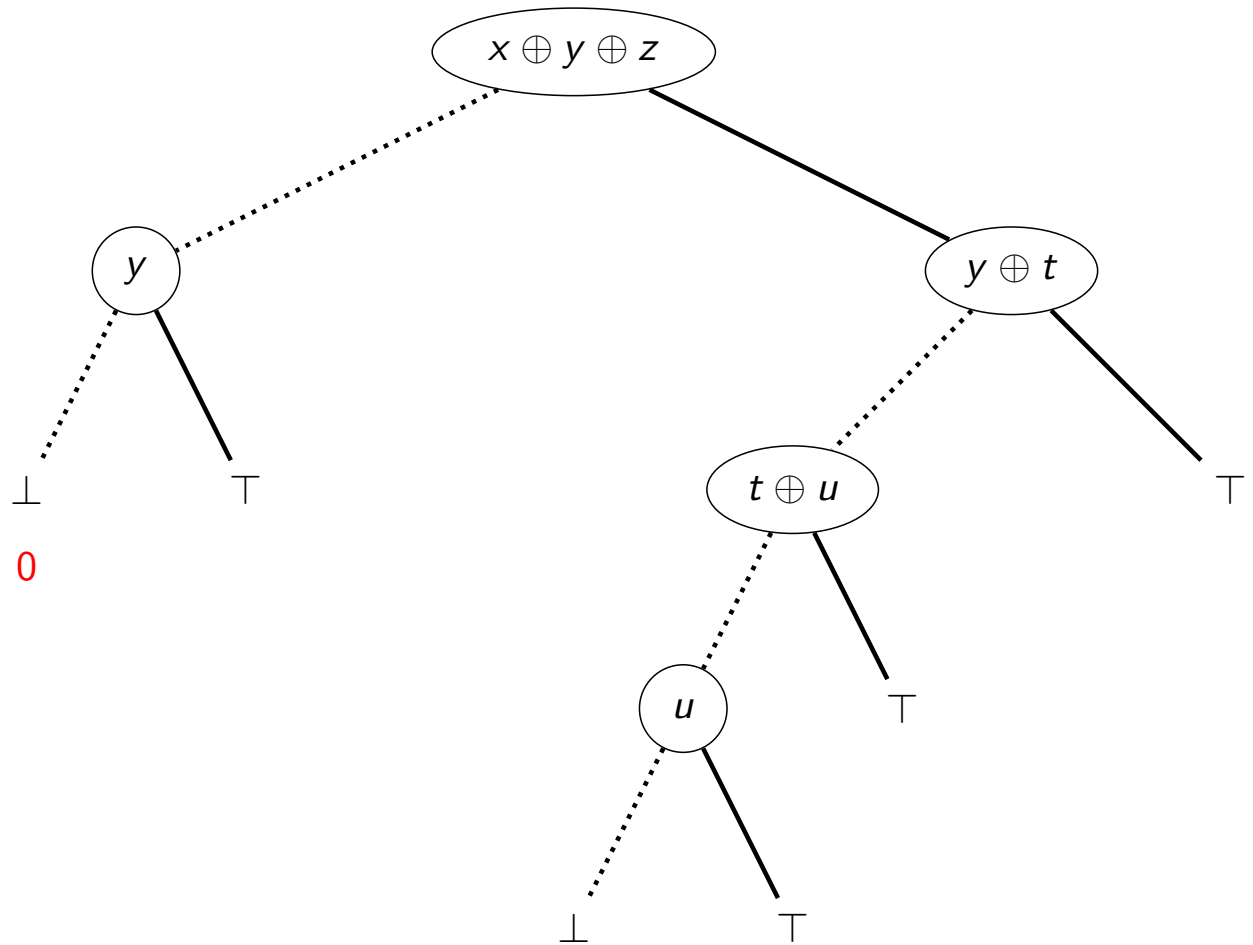
Arbre de décision affine (ADT) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



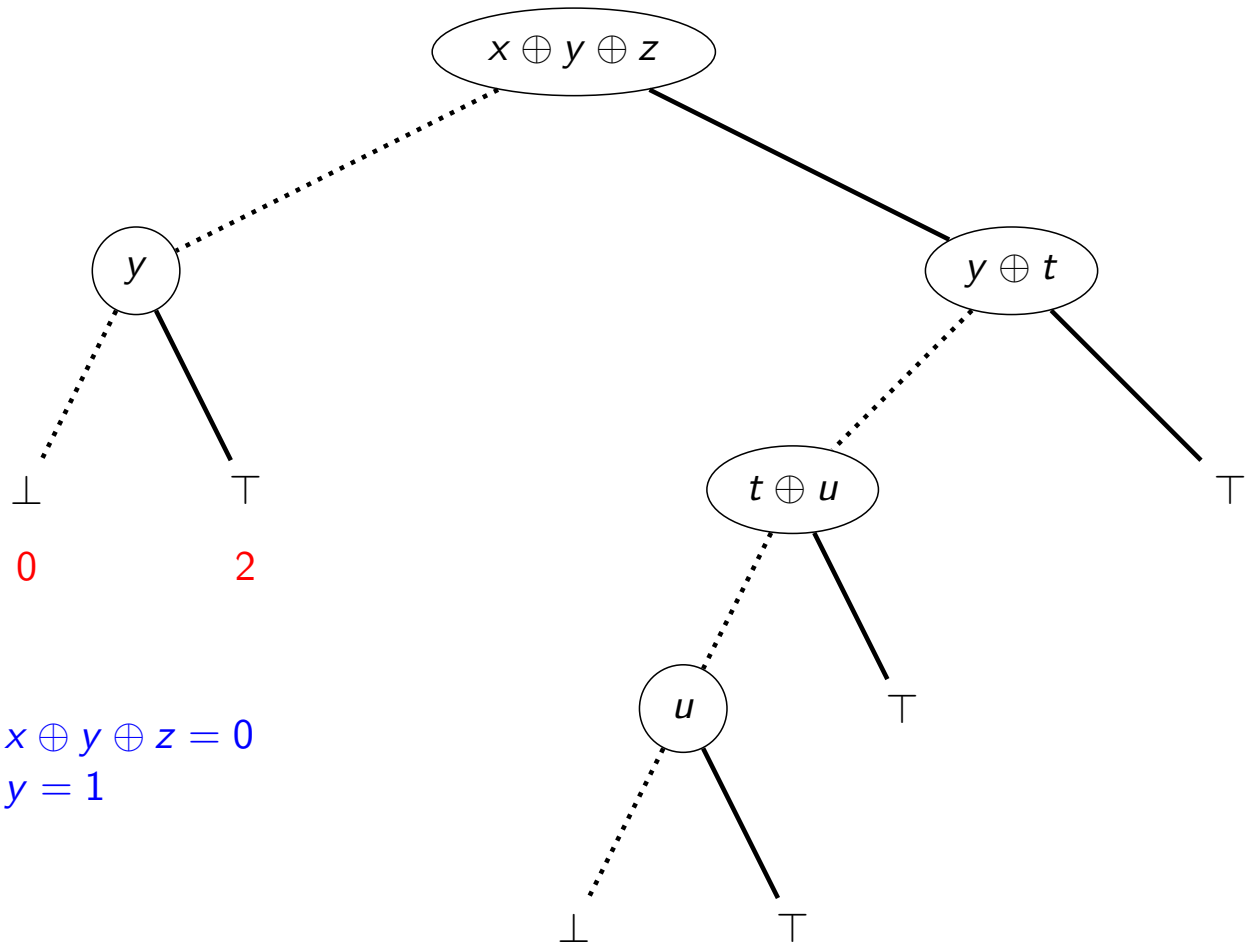
Arbre de décision affine (ADT) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



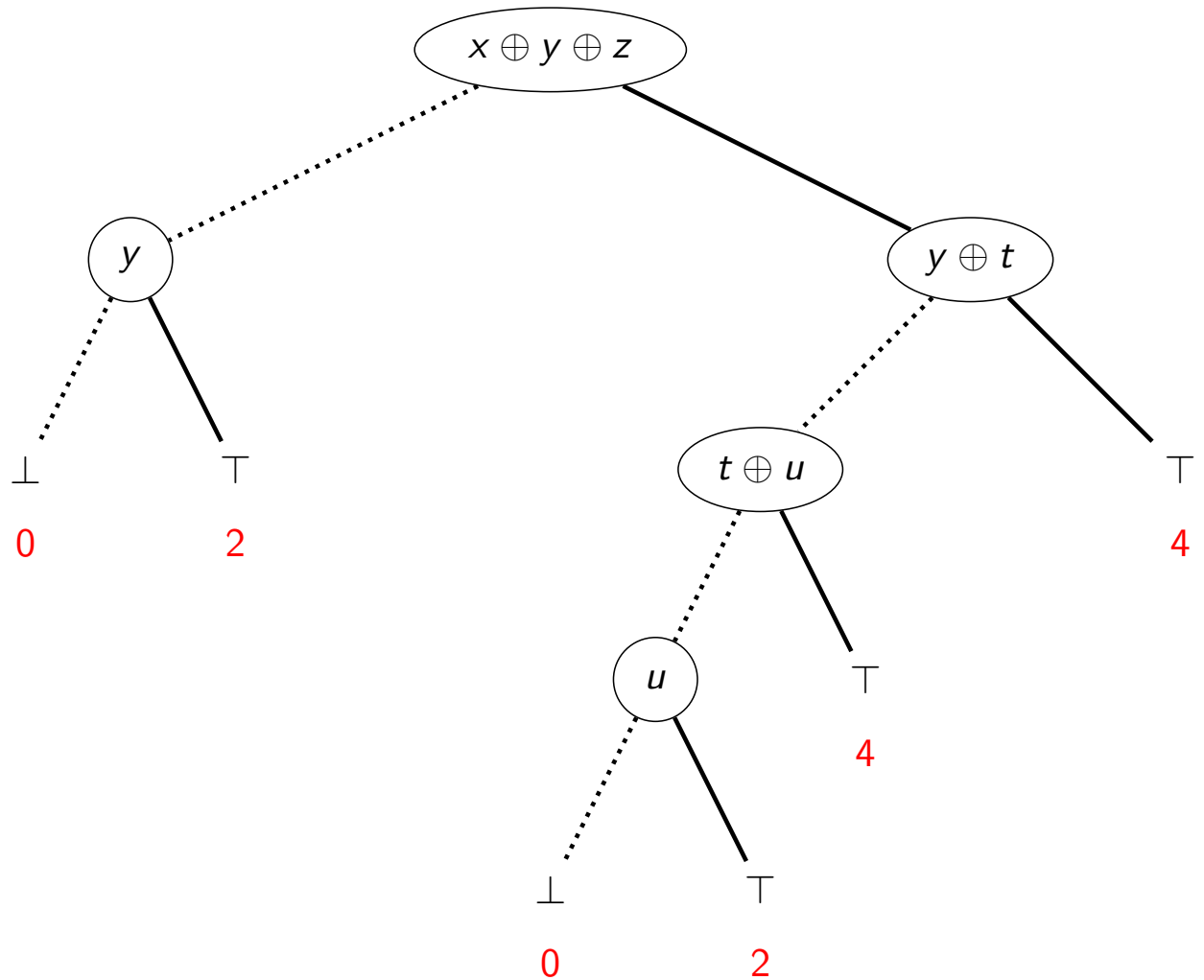
Arbre de décision affine (ADT) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



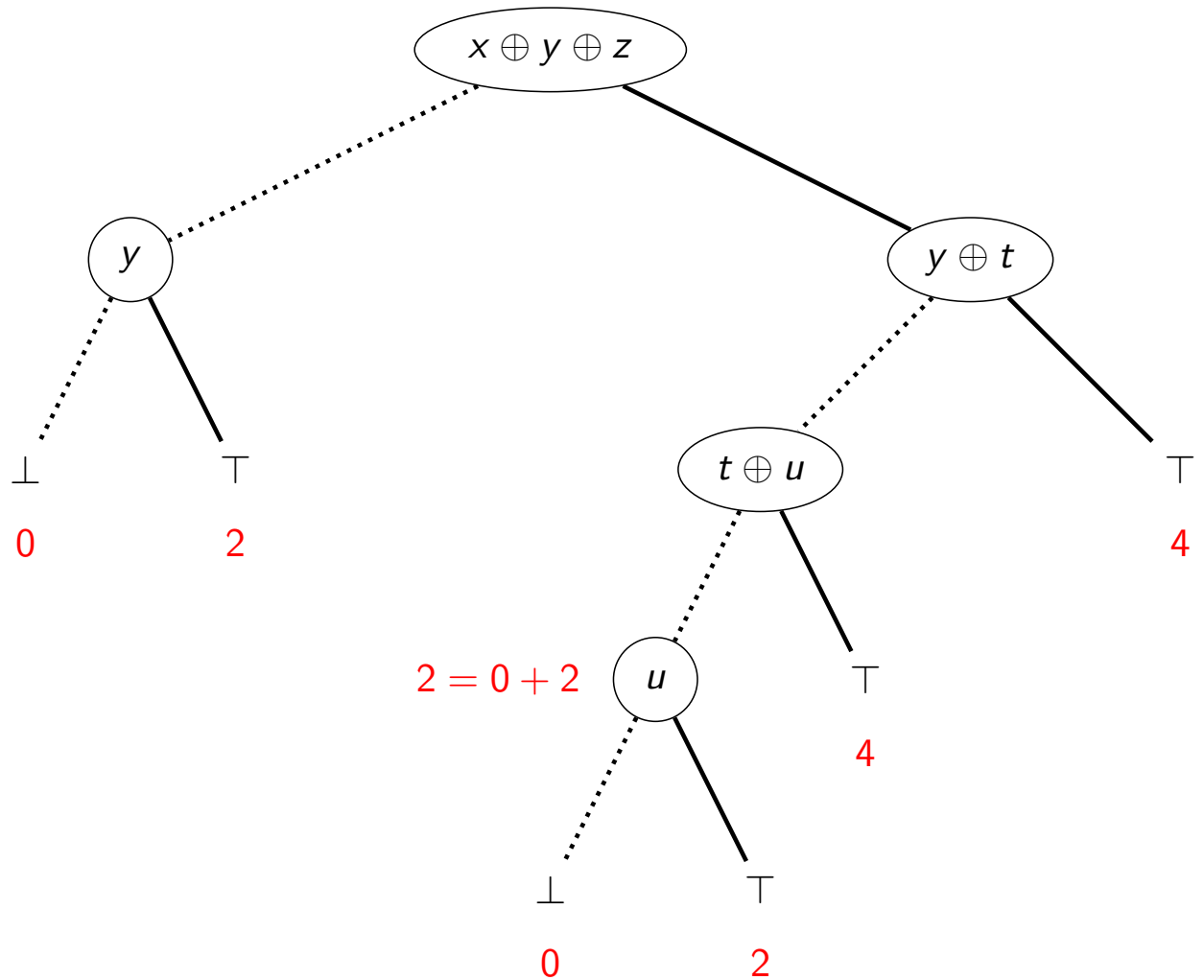
Arbre de décision affine (ADT) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



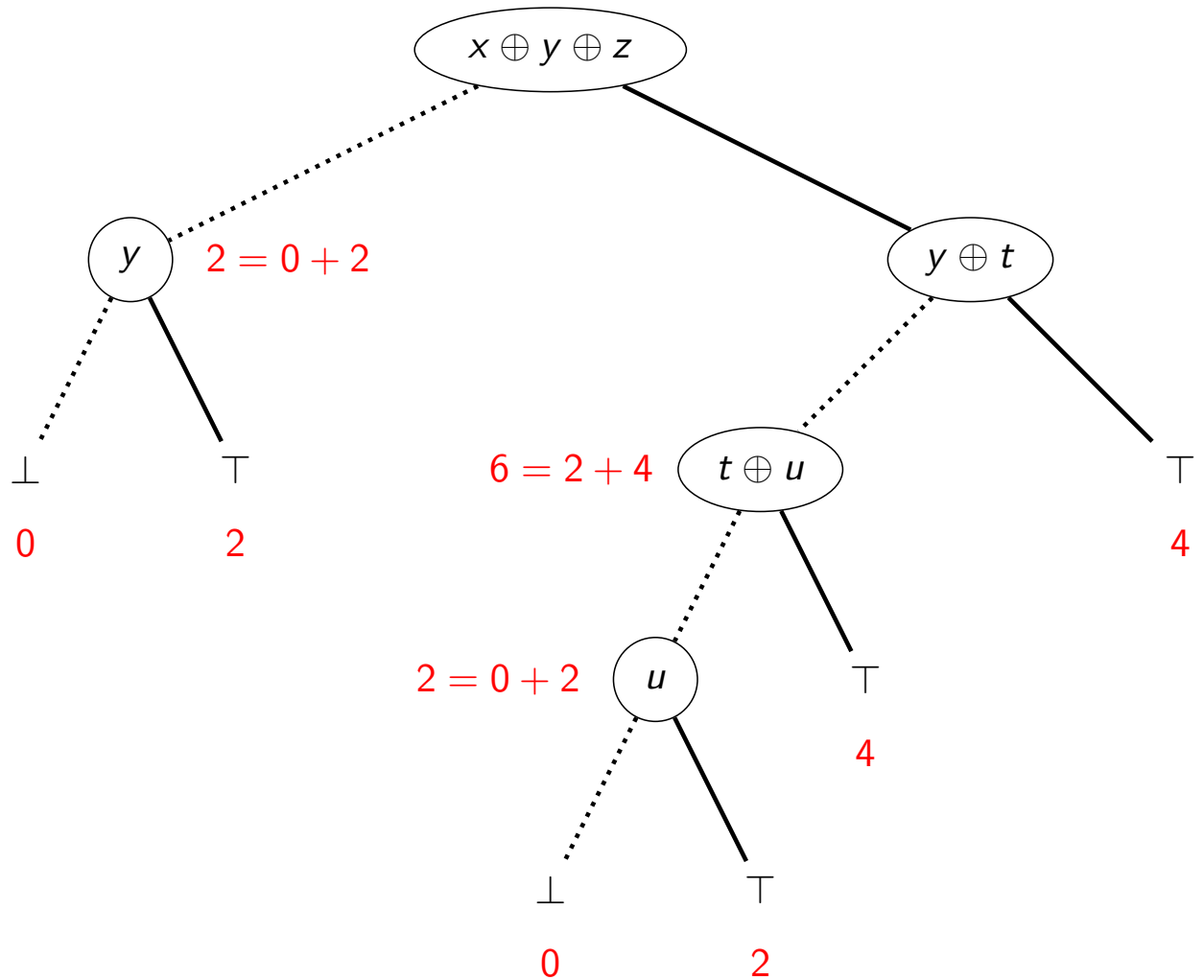
Arbre de décision affine (ADT) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



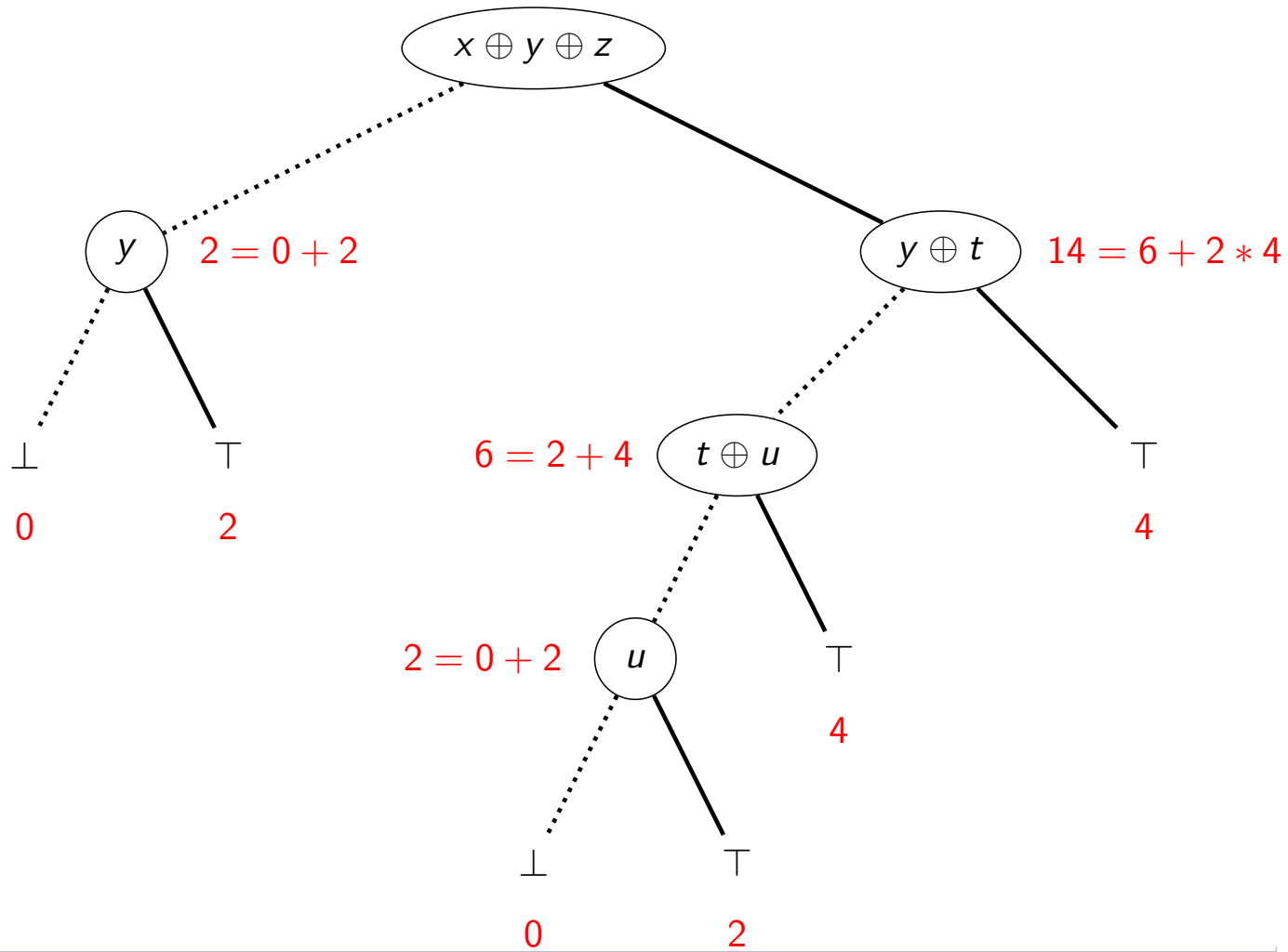
Arbre de décision affine (ADT) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



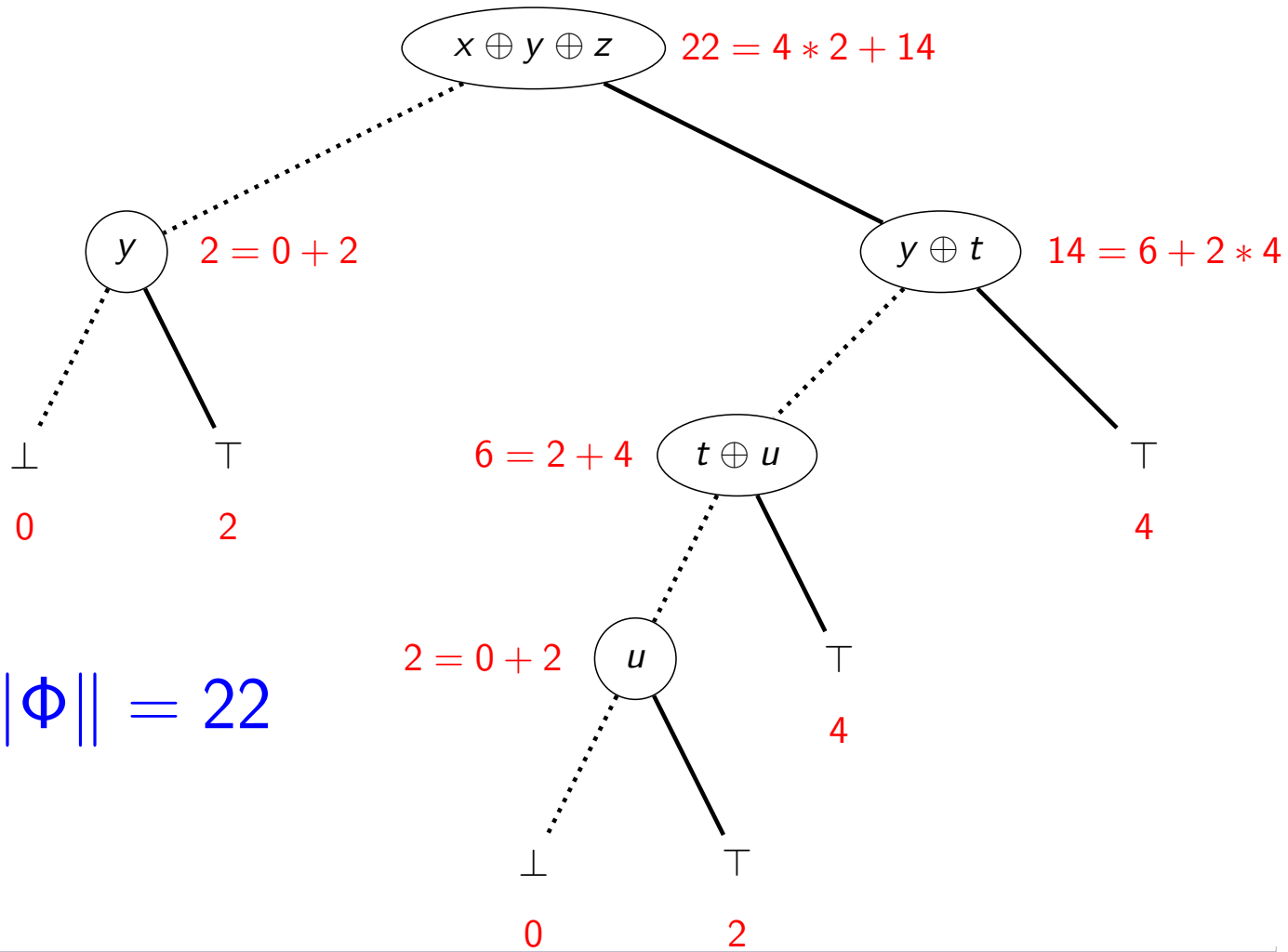
Arbre de décision affine (ADT) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



Arbre de décision affine (ADT) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



$\|\Phi\| = 22$

Requêtes et transformations (ADT)

$\mathcal{L}$	CO	VA	CE	IM	EQ	SE	CT	ME
CNF	○	√	○	√	○	○	○	○
d-DNNF	√	√	√	√	?	○	√	√
$\text{OBDD}_<$	√	√	√	√	√	√	√	√
DT	√	√	√	√	√	√	√	√
ADT	√	√	√	√	√	√	√	√

$\mathcal{L}$	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
CNF	√	○	√	√	√	•	√	•
d-DNNF	√	○	○	○	○	○	○	?
$\text{OBDD}_<$	√	○	√	○	√	○	√	√
DT	√	○	√	○	√	○	√	√
ADT	√	○	√	○	√	○	√	√

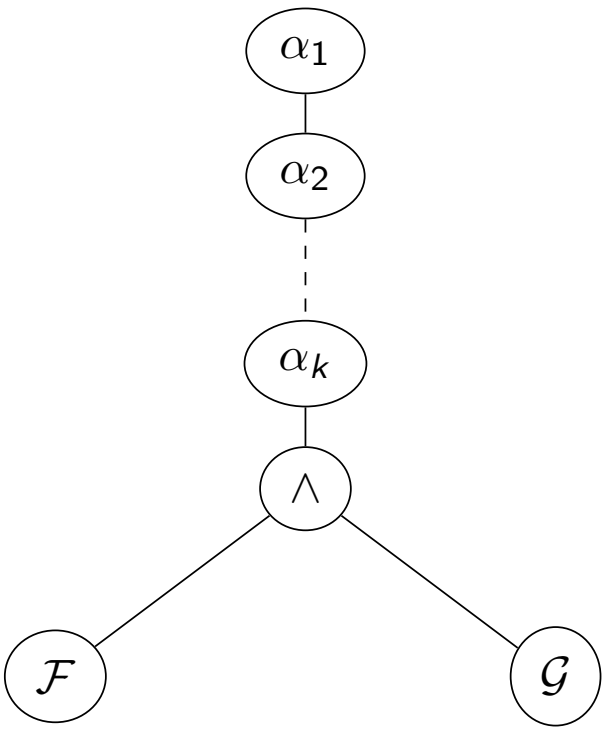
Expressivité et efficacité spatiale (ADT)

ADT est **complet** et **offre une meilleure efficacité spatiale que** DT :

- $\mathtt{CNF} \not\leq_s \mathtt{ADT}$
- $\mathtt{DNF} \not\leq_s \mathtt{ADT}$
- $\mathtt{OBDD}_< \not\leq_s \mathtt{ADT}$
- $\mathtt{d\text{-}DNNF}_T \not\leq_s \mathtt{ADT}$
- $\mathtt{ADT} <_s \mathtt{DT}$

Arbre de décision affine étendu ($\mathtt{EADT}_\wedge$)

- Utiliser **la décomposabilité** pour réduire la taille des ADT
- Introduire des nœuds $\wedge$ **affine décomposables**



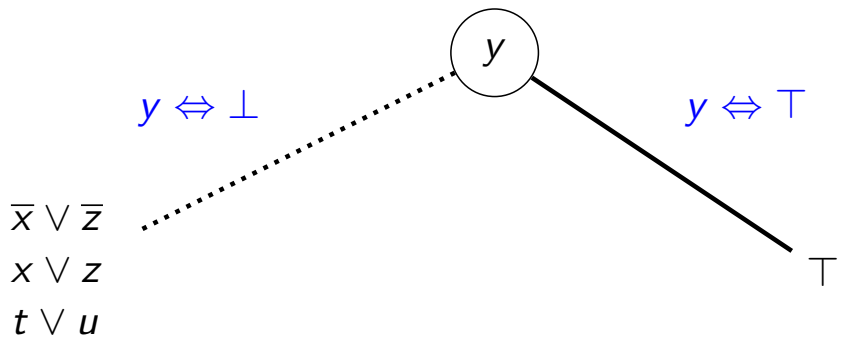
- $\forall i, \alpha_i$ est une clause affine
- le nœud $\wedge$ est **affine décomposable** si :
  - $Var(\mathcal{F}) \cap Var(\mathcal{G}) = \emptyset$;
  - Au plus une des formules $\mathcal{F}$ ou $\mathcal{G}$ a des variables en commun avec des clauses affines non unitaires ancêtres α_i

Arbre de décision affine étendu ($\mathtt{EADT}_{\wedge}$) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$

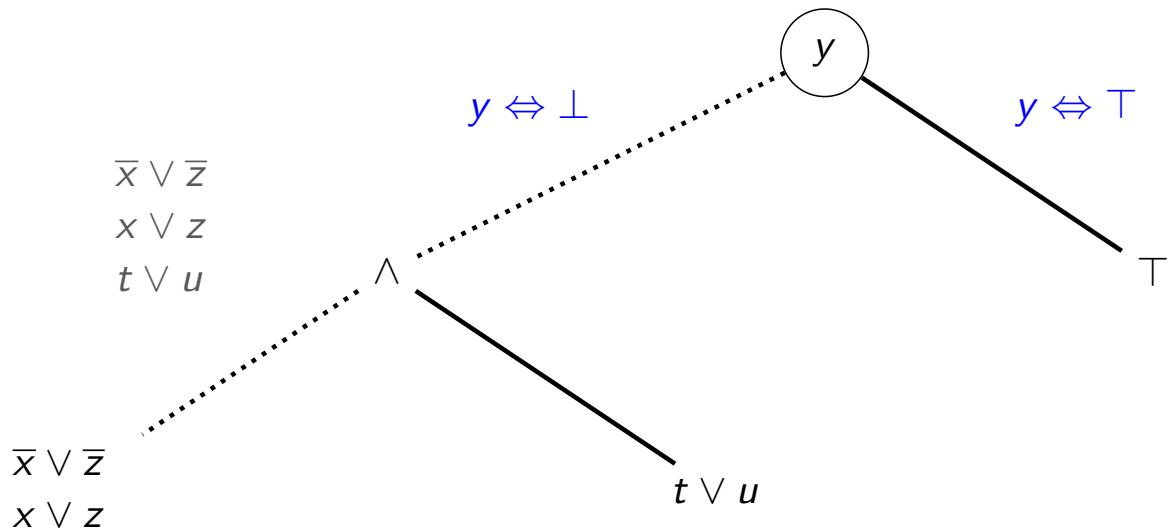
Arbre de décision affine étendu ($\mathtt{EADT}_\wedge$) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



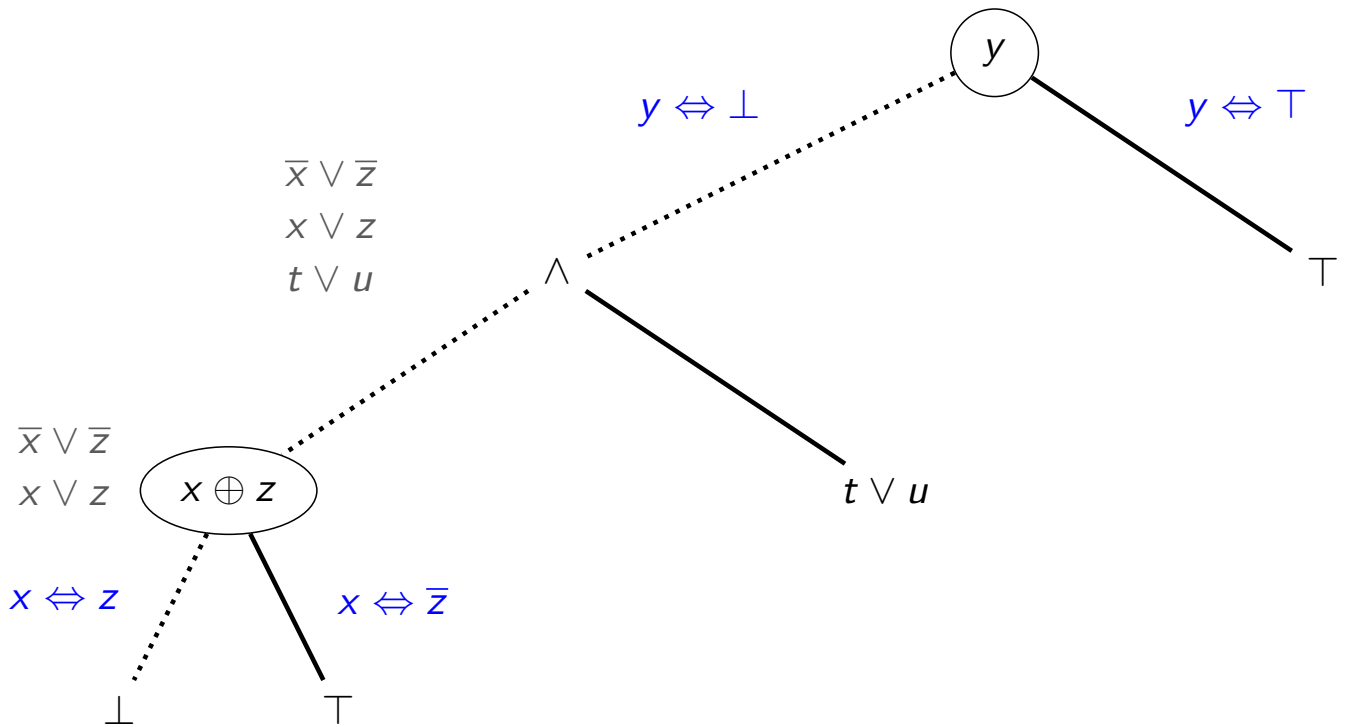
Arbre de décision affine étendu ($\mathtt{EADT}_{\wedge}$) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



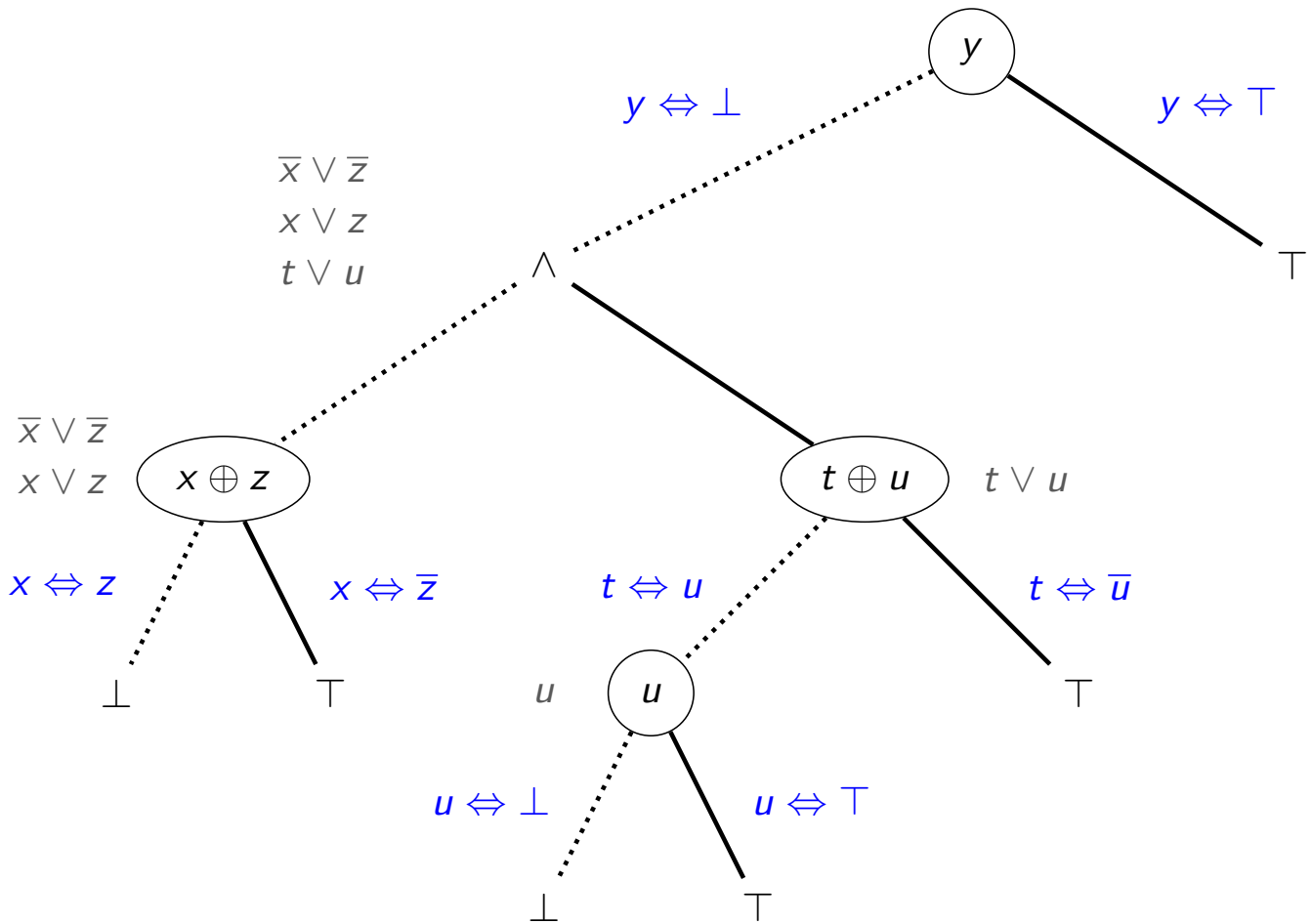
Arbre de décision affine étendu ($\mathtt{EADT}_\wedge$) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



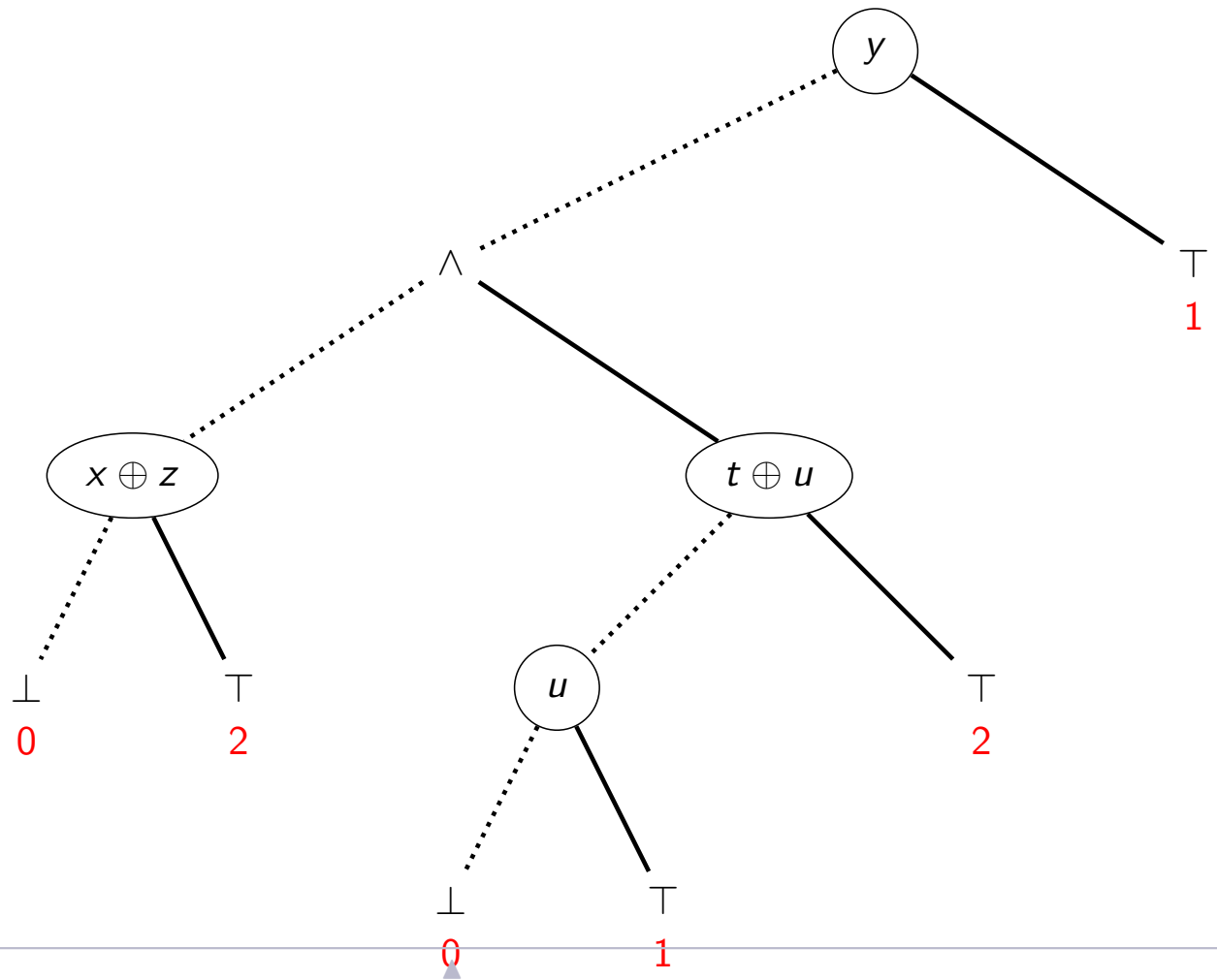
Arbre de décision affine étendu ($\mathtt{EADT}_\wedge$) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



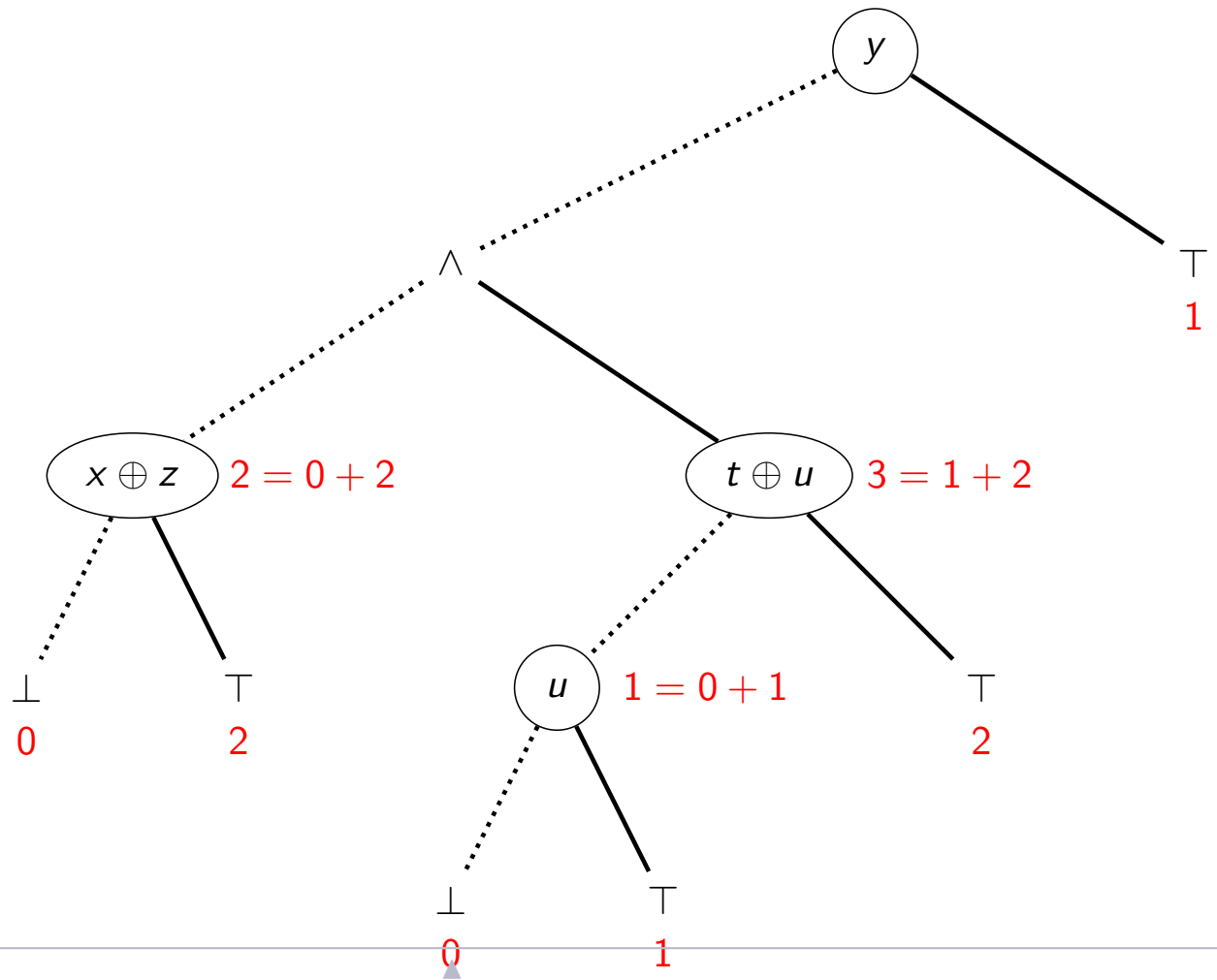
Arbre de décision affine étendu ($\mathtt{EADT}_\wedge$) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



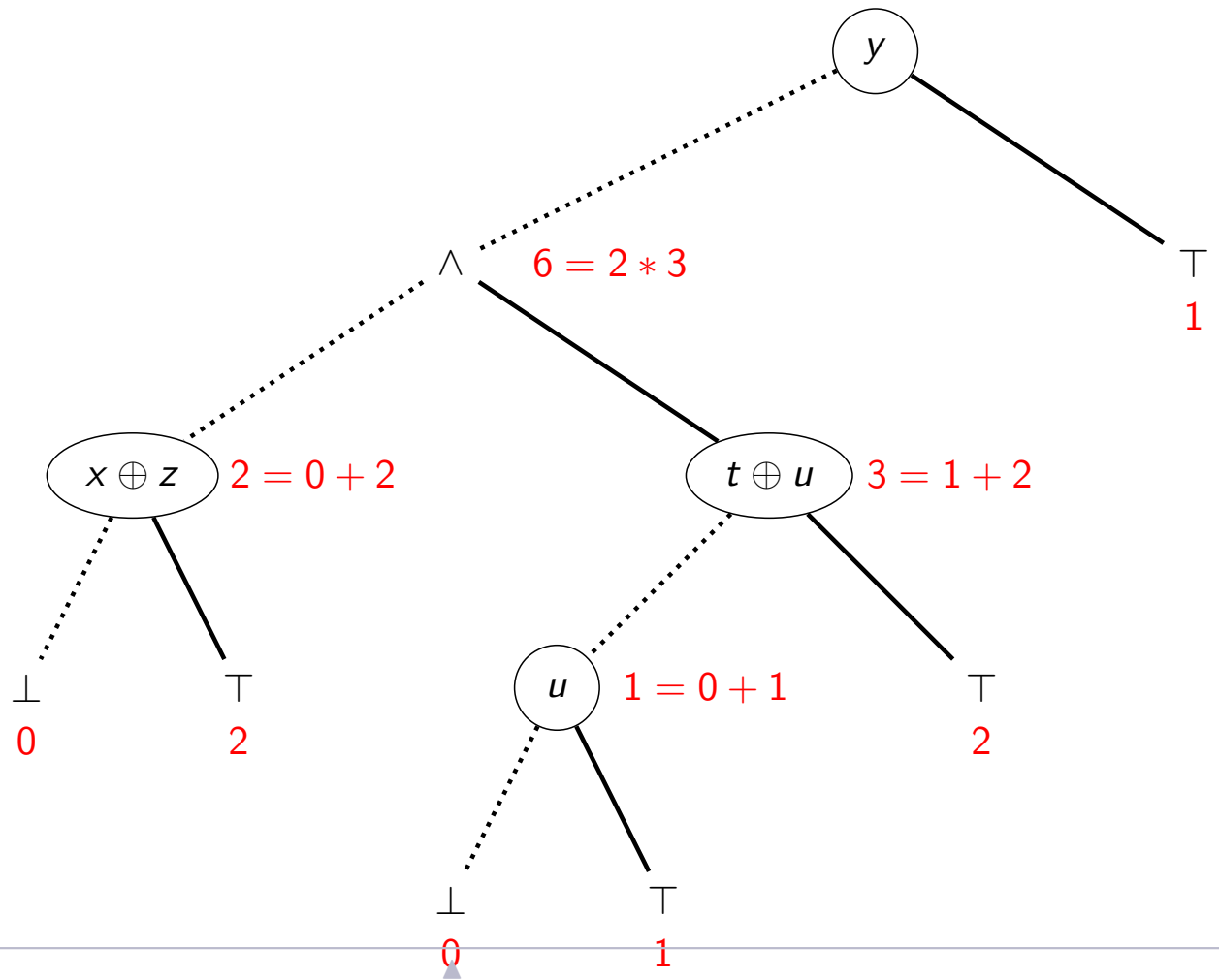
Arbre de décision affine étendu ($\mathtt{EADT}_\wedge$) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



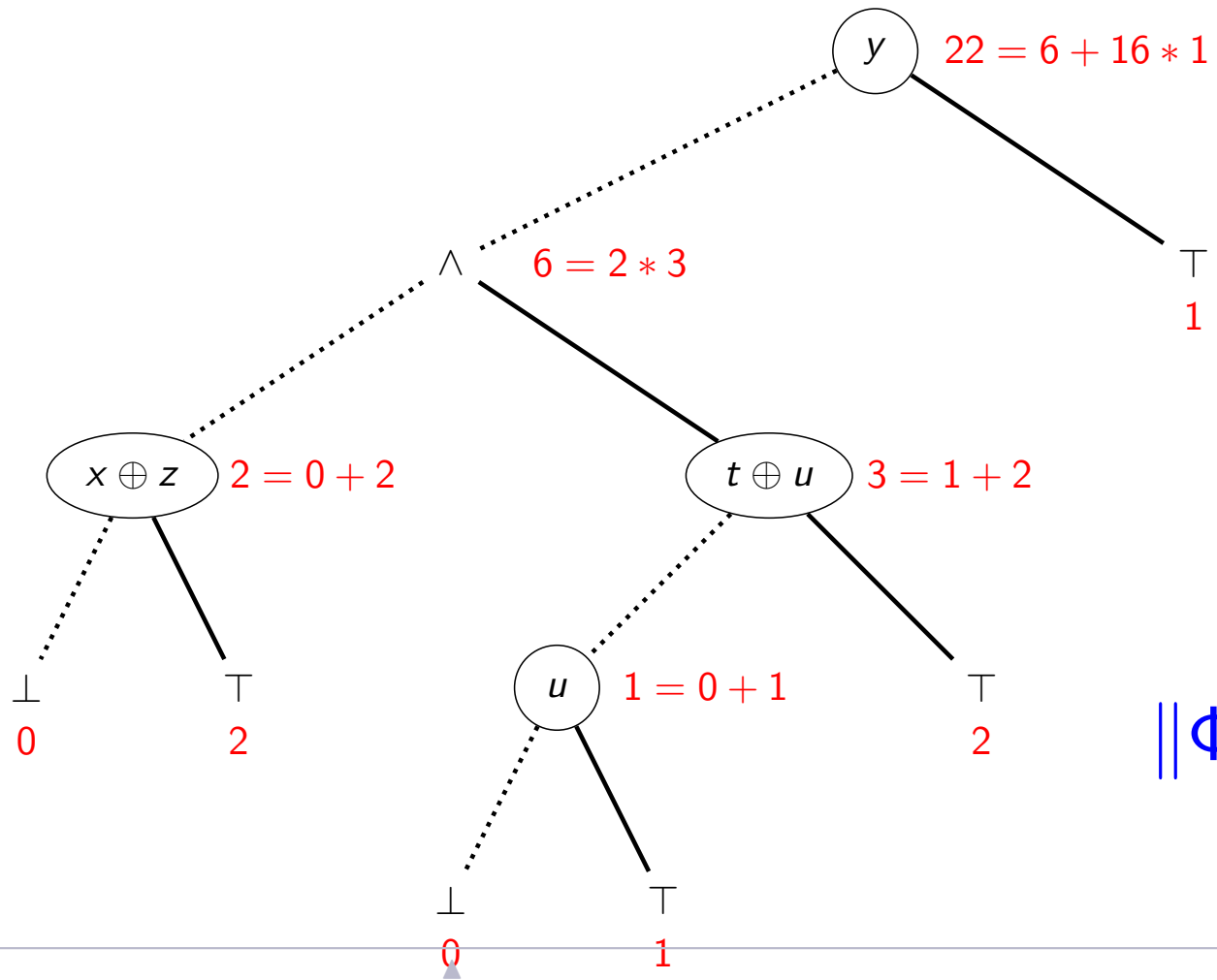
Arbre de décision affine étendu ($\mathtt{EADT}_{\wedge}$) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



Arbre de décision affine étendu ($\mathtt{EADT}_\wedge$) : construction et comptage des modèles

- $\Phi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee z) \wedge (y \vee t \vee u)$



$\|\Phi\| = 22$

Requêtes et transformations ($\mathtt{EADT}_\wedge$)

$\mathcal{L}$	CO	VA	CE	IM	EQ	SE	CT	ME
d-DNNF	√	√	√	√	?	○	√	√
$\mathtt{OBDD}_<$	√	√	√	√	√	√	√	√
DT	√	√	√	√	√	√	√	√
ADT	√	√	√	√	√	√	√	√
$\mathtt{EADT}_\wedge$	√	√	√	√	?	?	√	√

$\mathcal{L}$	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
d-DNNF	√	○	○	○	○	○	○	?
$\mathtt{OBDD}_<$	√	○	√	○	√	○	√	√
DT	√	○	√	○	√	○	√	√
ADT	√	○	√	○	√	○	√	√
$\mathtt{EADT}_\wedge$	√	○	?	○	○	○	?	?

Expressivité et efficacité spatiale ($\mathtt{EADT}_\wedge$)

$\mathtt{EADT}_\wedge$ est **complet** et **offre une meilleure efficacité spatiale que** $\mathtt{ADT}$:

- $\mathtt{CNF} \not\leq_s \mathtt{EADT}_\wedge$
- $\mathtt{DNF} \not\leq_s \mathtt{EADT}_\wedge$
- $\mathtt{OBDD}_< \not\leq_s \mathtt{EADT}_\wedge$
- $\mathtt{d\text{-}DNNF}_T \not\leq_s \mathtt{ADT}$
- $\mathtt{EADT}_\wedge <_s \mathtt{ADT}$

Un compilateur CNF-TO-EADT

- **Implémentation sur la base du solveur CDCL MINISAT**
- Extension de MINISAT aux formules ECNF
- Utilisation de l'heuristique de choix des variables VSIDS (w)
  - extension du poids VSIDS aux clauses, $\text{VSIDS}(\alpha) = \sum_{x \in \alpha} w(x)$
  - choisir une clause α qui maximise $\text{VSIDS}(\alpha)$
  - remplacer $x = max(\alpha)$ par la XOR-disjonction de $k-1$ littéraux de $\alpha \setminus \{x\}$ ayant le meilleur poids selon VSIDS
  - $k = 2$ pour nos expérimentations

Protocole expérimental

- Quad-Core Intel Xeon X550 avec 32Go de mémoire vive
- CNF choisie parmi différentes instances de la *SAT LIBrary*
- Pour chaque instance Φ 1000 requêtes γ ont été générées
  - chaque requête γ est un terme de 3 littéraux dont les variables sont choisies aléatoirement parmi celles de Φ (distribution uniforme)
  - le signe de chaque littéral est aussi choisi aléatoirement ($\frac{1}{2}$)
  - **l'objectif est de compter le nombre de modèles de $\Phi \mid \gamma$**
- Temps limité à 3h pour compiler Φ et à 3h par requête pour compter le nombre de modèles de $\Phi \mid \gamma$

Protocole expérimental

Comparaison de trois approches :

- approche directe (non compilée) CACHET
  - $\#F_{\text{CACHET}}$ le nombre de requêtes traitées
  - $\overline{Q}_{\text{CACHET}}$ indique le temps moyen nécessaire répondre aux requêtes réalisables
- approche par compilation d-DNNF c2d
  - C_{C2D} indique le temps de compilation
  - $\overline{Q}_{\text{C2D}}$ indique le temps moyen nécessaire pour répondre aux requêtes après compilation
- approche par compilation EADT
  - C_{EADT} indique le temps de compilation
  - $\overline{Q}_{\text{EADT}}$ indique le temps moyen nécessaire pour répondre aux requêtes après compilation

Résultats expérimentaux

Instances	Cachet		c2d		EADT	
nom	$\#F$	$\overline{Q}$	C	$\overline{Q}$	C	$\overline{Q}$
ais6	1000	0.531	1.23	4.10^{-5}	0.01	$< 1.10^{-7}$
ais10	80	0.573	–	–	410	$< 1.10^{-3}$
bmc-ibm-2	1000	0.569	–	–	0.37	2.10^{-5}
bmc-ibm-4	1000	5.412	1128	9.09	–	–
bw_large.a	1000	0.537	15.05	2.10^{-5}	$< 1.10^{-4}$	$< 1.10^{-7}$
bw_large.d	896	31.44	–	–	1.9	2.10^{-5}
bw medium	1000	0.526	2.39	2.10^{-5}	$< 1.10^{-4}$	$< 1.10^{-7}$
hanoi4	505	0.557	559.6	3.10^{-5}	0.13	$< 1.10^{-7}$
hanoi5	440	0.619	2240	8.10^{-5}	1.1	1.10^{-5}
logistics.a	993	1.266	–	–	6757	2.12
ssa7552-038	1000	0.634	20.99	0.042	–	–

Graphes de décision décomposables multivalués

CNF VS CN

CNF

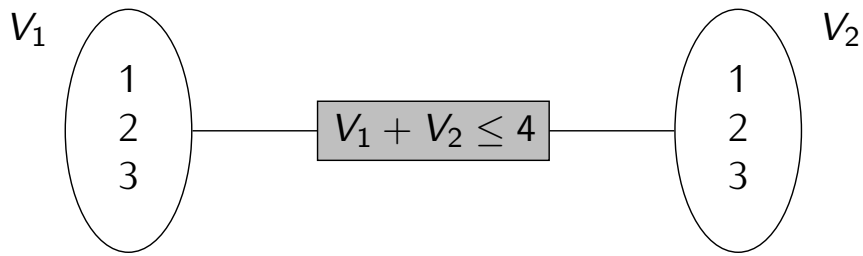
$(V_1 = 1) \vee (v_1 = 2) \vee (v_1 = 3)$
$\neg(V_1 = 1) \vee \neg(v_1 = 2)$
$\neg(V_1 = 1) \vee \neg(v_1 = 3)$
$\neg(V_1 = 2) \vee \neg(v_1 = 3)$

$(V_2 = 1) \vee (v_2 = 2) \vee (v_2 = 3)$
$\neg(V_2 = 1) \vee \neg(v_2 = 2)$
$\neg(V_2 = 1) \vee \neg(v_2 = 3)$
$\neg(V_2 = 2) \vee \neg(v_2 = 3)$

$\neg(V_1 = 2) \vee \neg(v_2 = 3)$
$\neg(V_1 = 3) \vee \neg(v_2 = 2)$
$\neg(V_1 = 3) \vee \neg(v_2 = 3)$

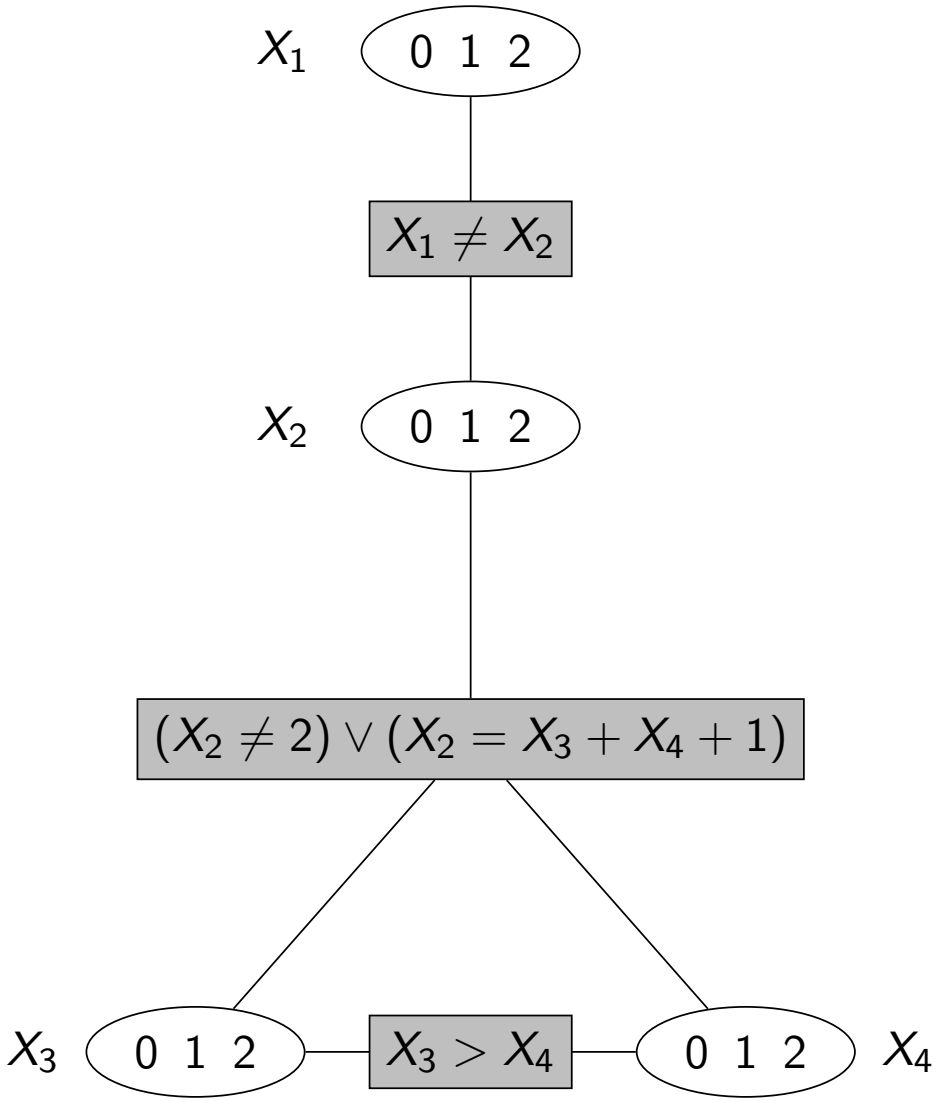
11 clauses

CN



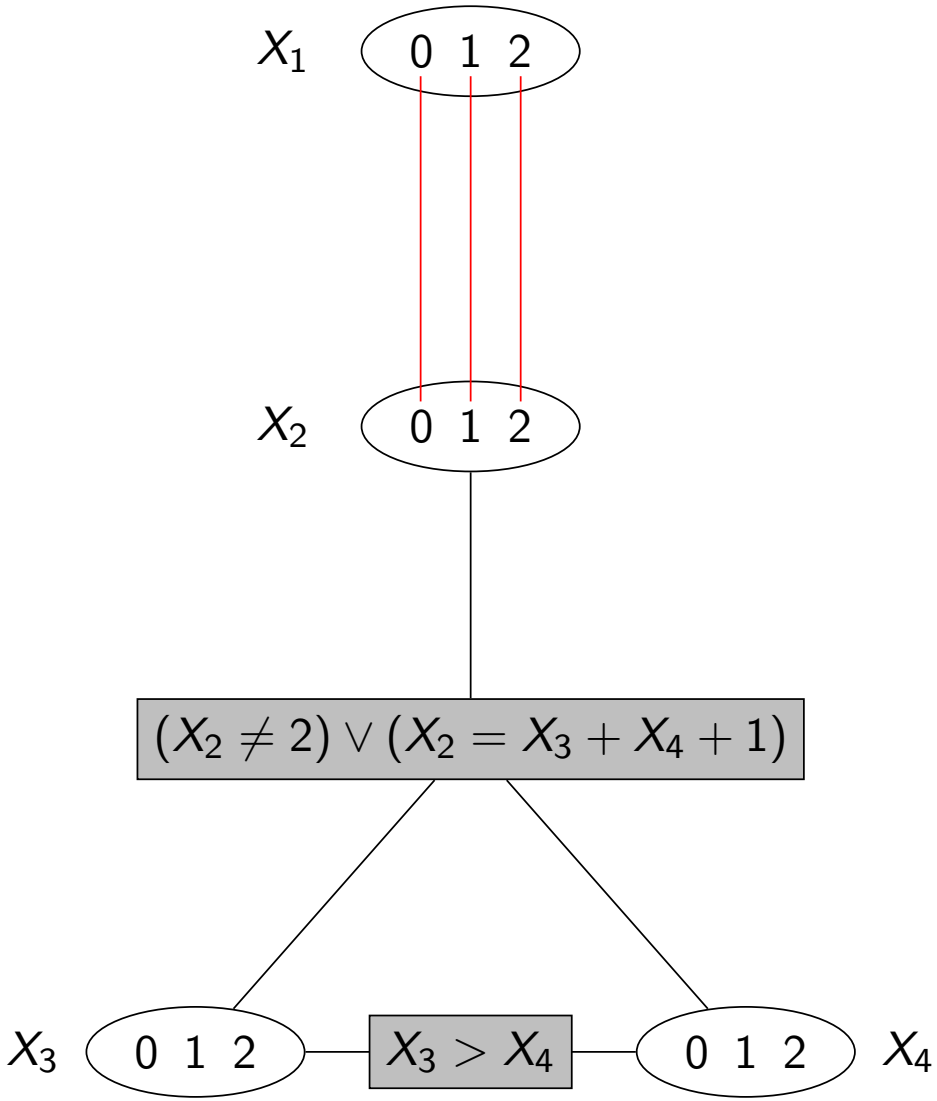
Le problème CSP

- $\mathtt{CN} = \langle \mathcal{X}, \mathcal{C} \rangle$
- $\mathcal{X} = \{X_1, \ldots, X_n\}$
- $\mathcal{D} = \{D_1, \ldots, D_n\}$
- $\mathcal{C} = \{C_1, \ldots, C_m\}$



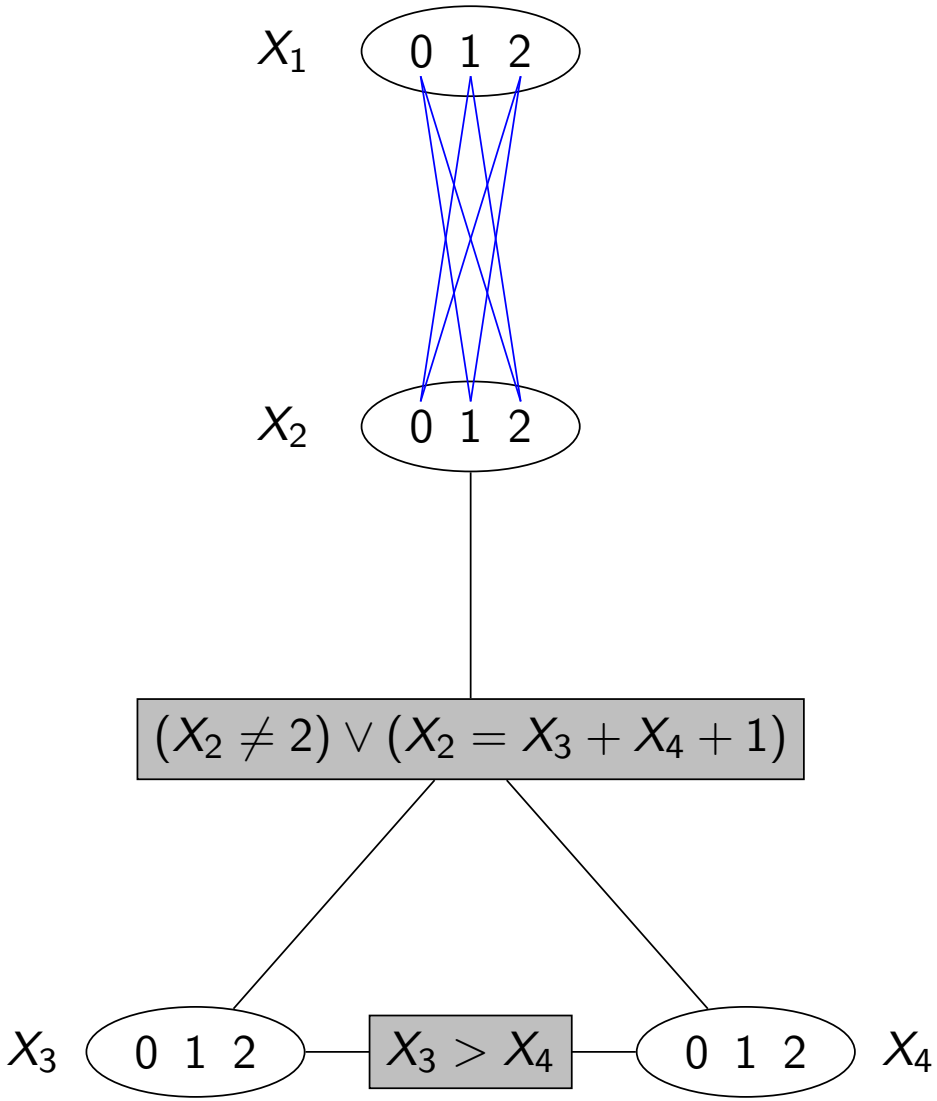
Le problème CSP

- $\mathtt{CN} = \langle \mathcal{X}, \mathcal{C} \rangle$
- $\mathcal{X} = \{X_1, \ldots, X_n\}$
- $\mathcal{D} = \{D_1, \ldots, D_n\}$
- $\mathcal{C} = \{C_1, \ldots, C_m\}$



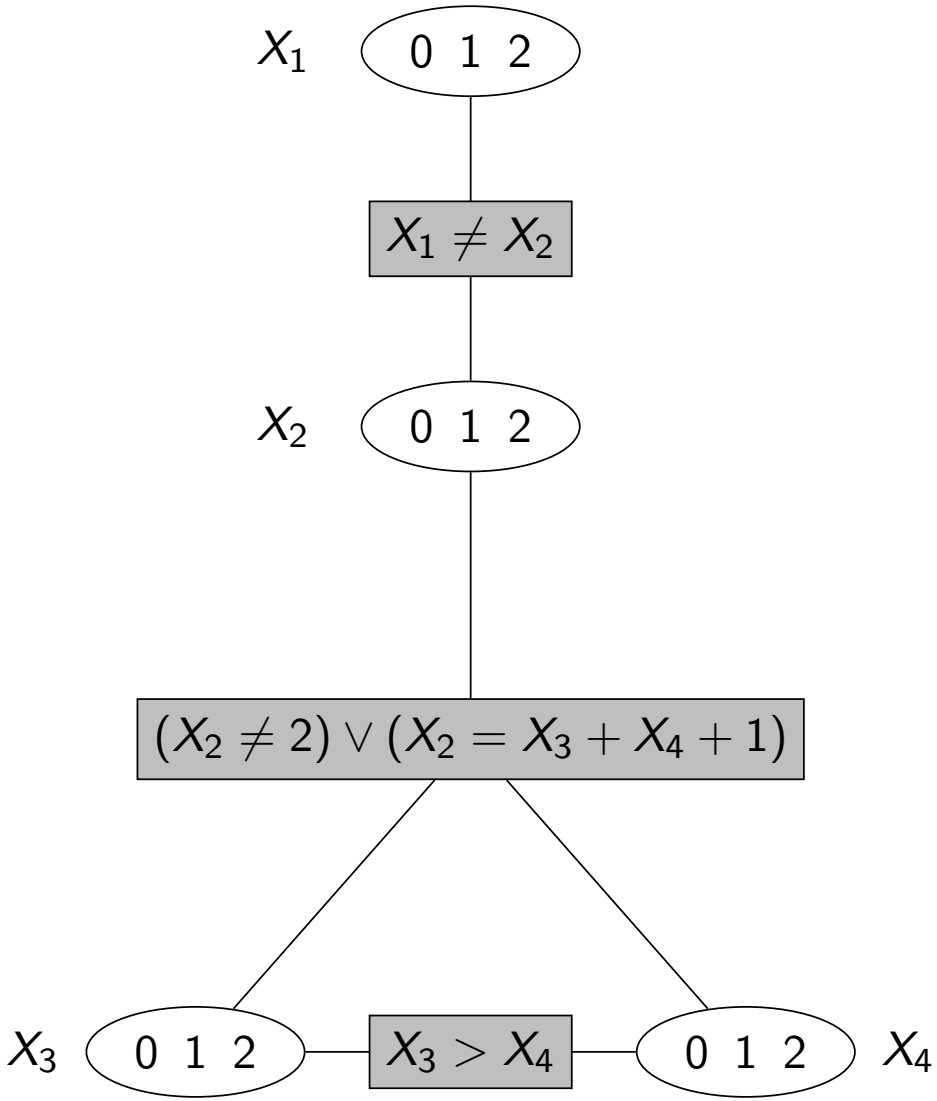
Le problème CSP

- $\mathtt{CN} = \langle \mathcal{X}, \mathcal{C} \rangle$
- $\mathcal{X} = \{X_1, \ldots, X_n\}$
- $\mathcal{D} = \{D_1, \ldots, D_n\}$
- $\mathcal{C} = \{C_1, \ldots, C_m\}$



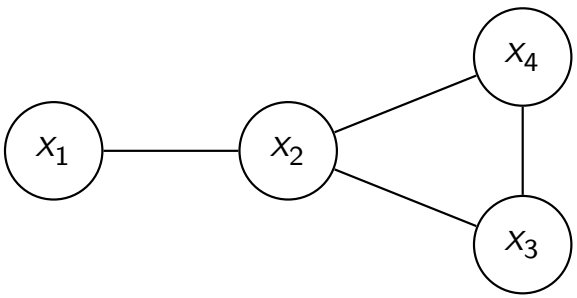
Le problème CSP

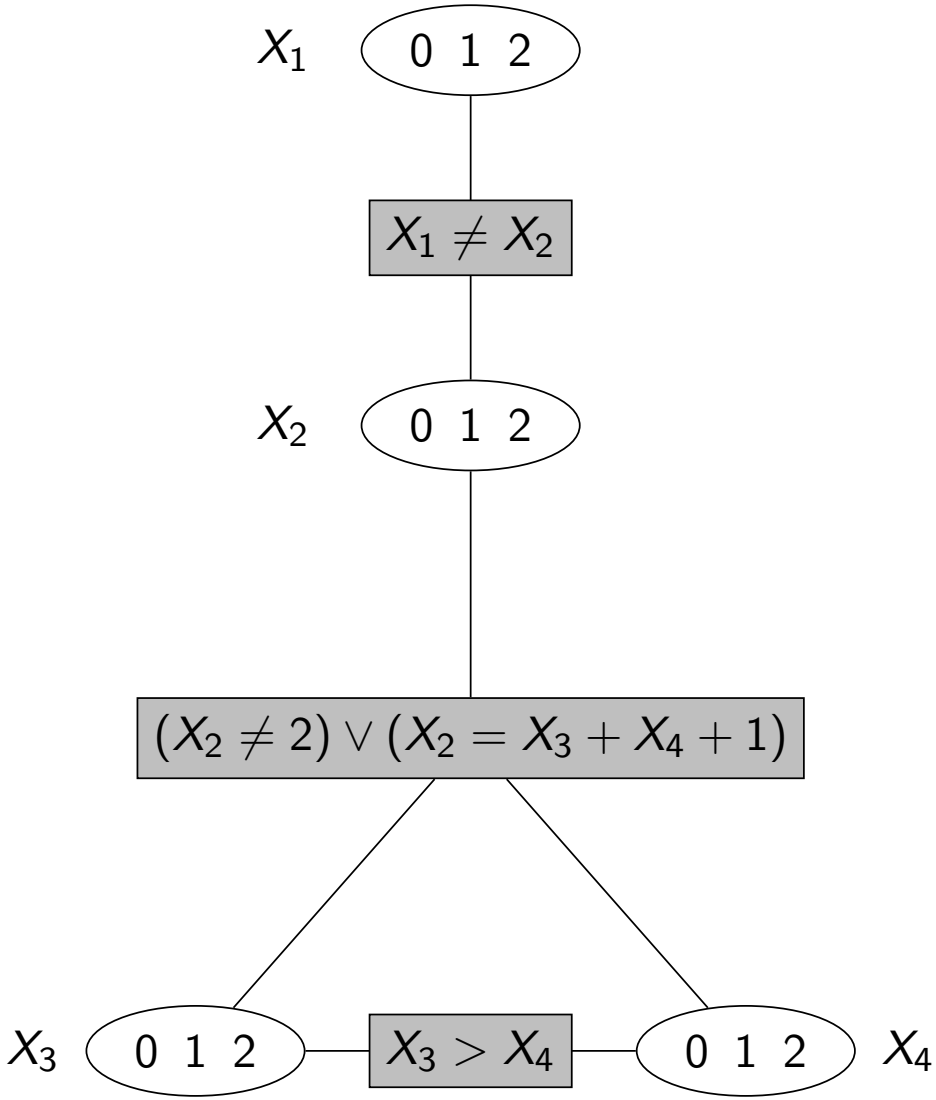
- $\mathtt{CN} = \langle \mathcal{X}, \mathcal{C} \rangle$
- $\mathcal{X} = \{X_1, \ldots, X_n\}$
- $\mathcal{D} = \{D_1, \ldots, D_n\}$
- $\mathcal{C} = \{C_1, \ldots, C_m\}$



Le problème CSP

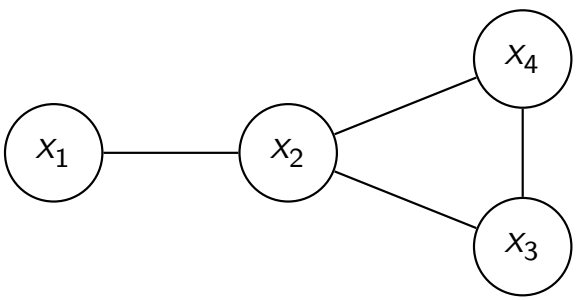
- $\mathtt{CN} = \langle \mathcal{X}, \mathcal{C} \rangle$
- $\mathcal{X} = \{X_1, \ldots, X_n\}$
- $\mathcal{D} = \{D_1, \ldots, D_n\}$
- $\mathcal{C} = \{C_1, \ldots, C_m\}$
- Graphe primal d'un CN :



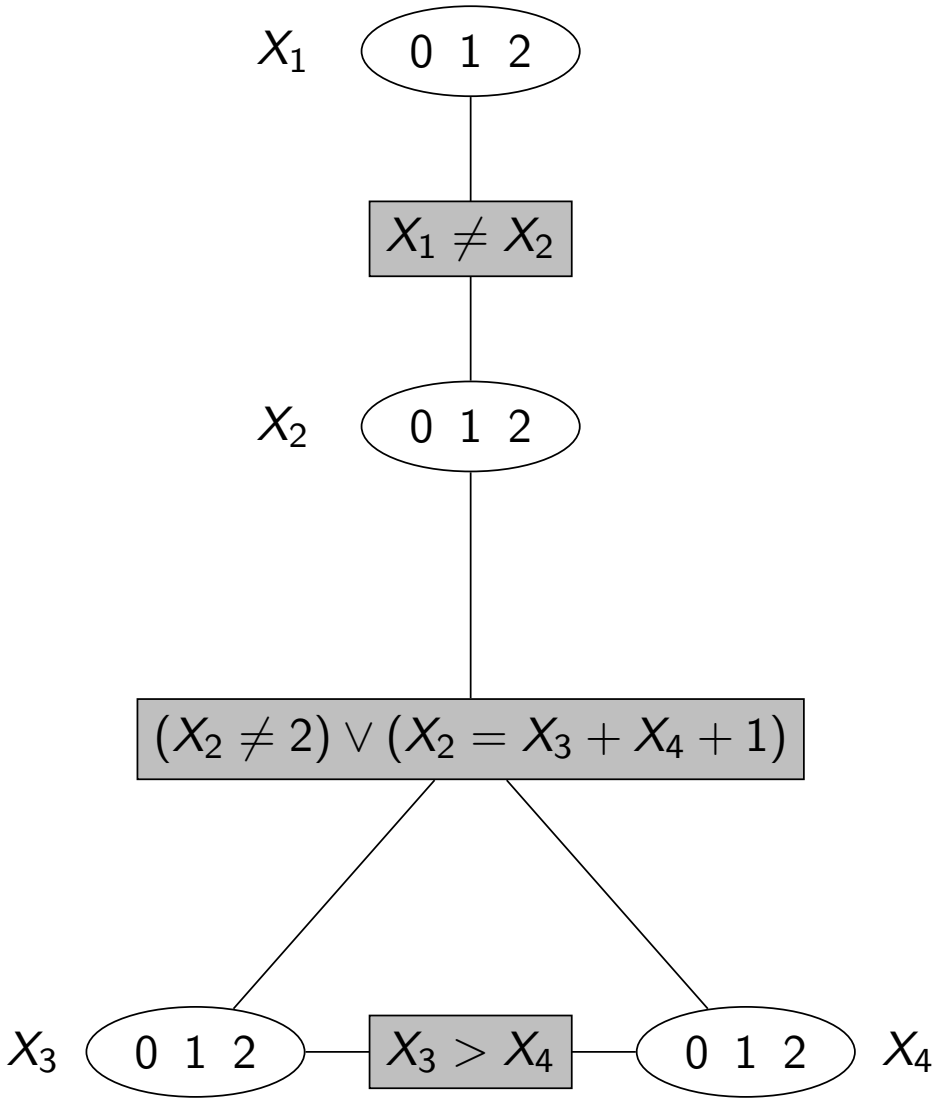


Le problème CSP

- $\mathtt{CN} = \langle \mathcal{X}, \mathcal{C} \rangle$
- $\mathcal{X} = \{X_1, \ldots, X_n\}$
- $\mathcal{D} = \{D_1, \ldots, D_n\}$
- $\mathcal{C} = \{C_1, \ldots, C_m\}$
- Graphe primal d'un CN :

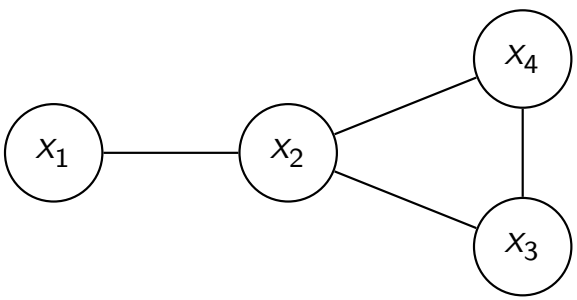


- Cohérence d'arc

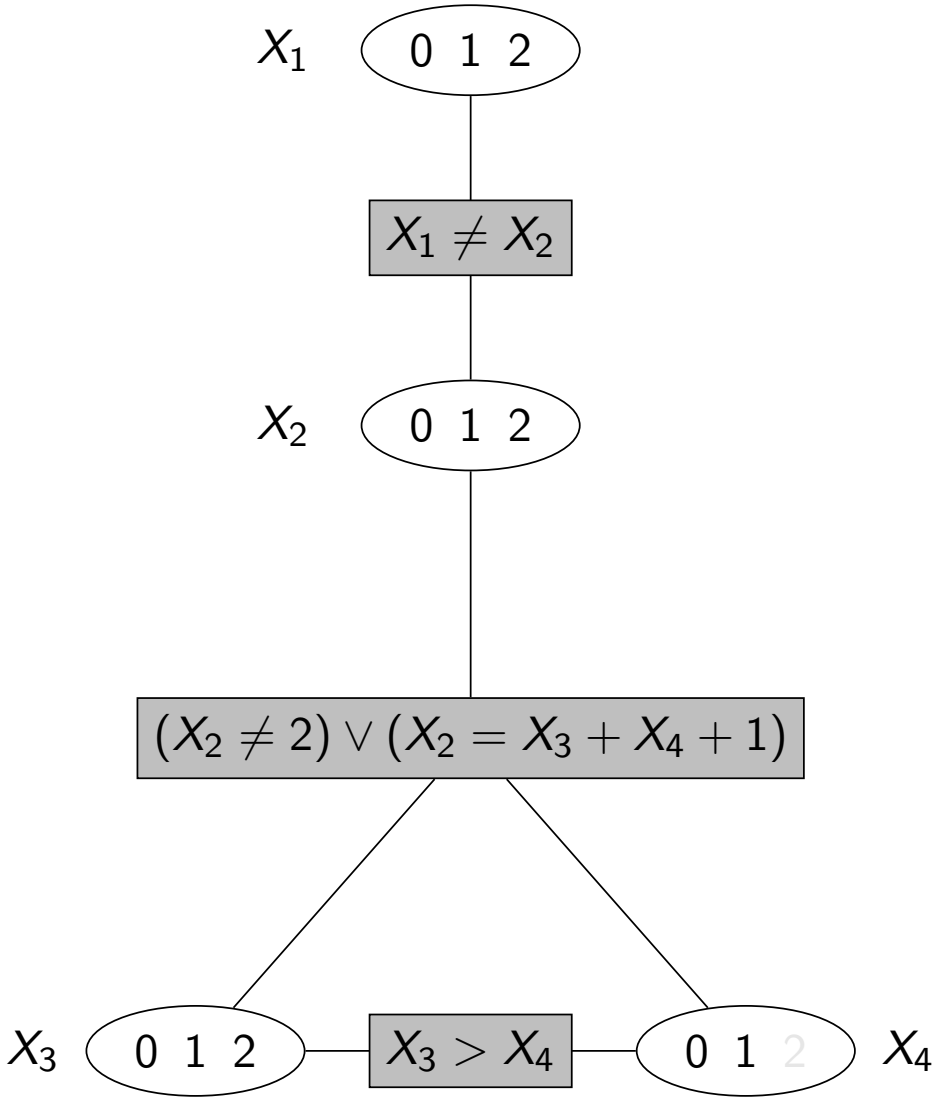


Le problème CSP

- $\mathtt{CN} = \langle \mathcal{X}, \mathcal{C} \rangle$
- $\mathcal{X} = \{X_1, \ldots, X_n\}$
- $\mathcal{D} = \{D_1, \ldots, D_n\}$
- $\mathcal{C} = \{C_1, \ldots, C_m\}$
- Graphe primal d'un CN :



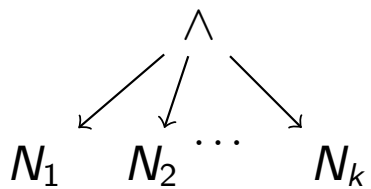
- Cohérence d'arc



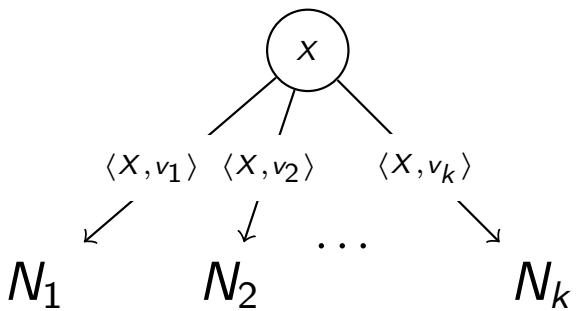
Le langage MDG

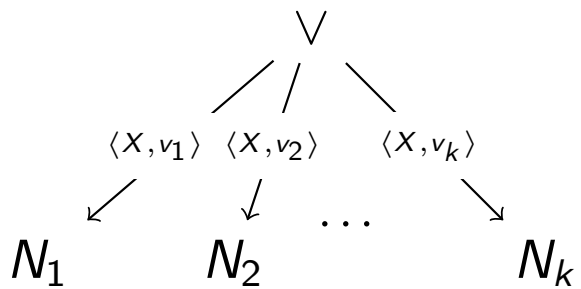
Le langage (read-once) MDG est l'ensemble des DAG à racine unique :

- les nœuds feuilles sont étiquetés par $\top$ (vrai) ou $\bot$ (faux)
- tout nœud interne est soit :
  - un **nœud** $\wedge$ du type $\wedge(N_1, \ldots, N_k)$



  - ou un **nœud de décision** associé à une variable $X_i \in \mathcal{X}$, i.e., un **nœud** $\vee$ déterministe $\vee(N_1, \ldots, N_k)$ tel que $D_i = \{v_1, \ldots, v_k\}$ et chaque arc est étiqueté par un assignation élémentaire $\langle Xi, v_j \rangle$

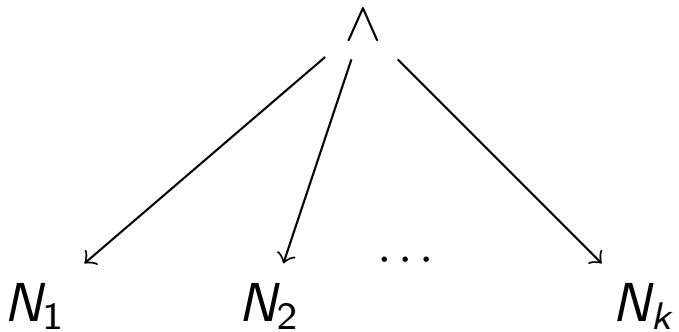




- Chaque variable apparait une fois au plus sur chaque chemin du MDG

Le langage MDDG

- Le langage MDDG est le sous-ensemble de MDG, où tout nœud $\wedge$ $N = \wedge(N_1, ..., N_k)$ est décomposable



$\forall i, j \in 1, \ldots, k$ si $i \neq j$,
alors $Var(N_i) \cap Var(N_j) = \emptyset$

- d-DNNF correspond au sous-ensemble des MDDG où toute variable possède un domaine booléen

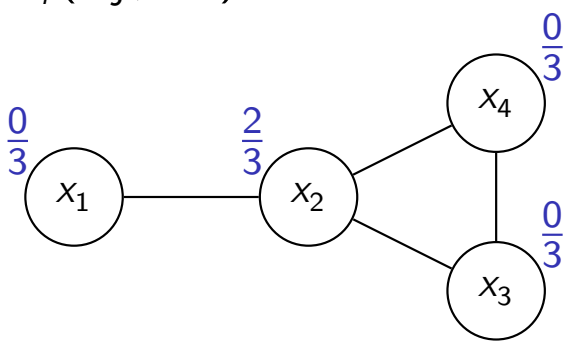
- Les **requêtes et transformations** offertes par MDDG sont les mêmes que celles offertes par d-DNNF

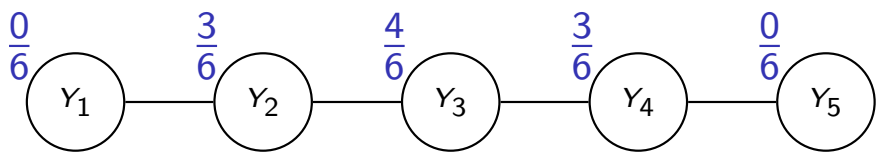
Heuristique de choix de variable

- **Les heuristiques de recherche de solution ne sont pas dédiées à la compilation**
- Utilisation d'une heuristique basée sur la **centralité d'intermédiarité**

$$bc(X_i) = \Sigma_{X_j \neq X_i \neq X_k} \frac{\sigma_{X_i}(X_j, X_k)}{\sigma(X_j, X_k)}$$

où X_i, X_j, X_k sont des nœuds d'un réseau de contraintes donné, $\sigma(X_j, X_k)$ est le nombre de plus courts chemins de X_j à X_k, et $\sigma_{X_i}(X_j, X_k)$ est le nombre de chemins passant par X_i

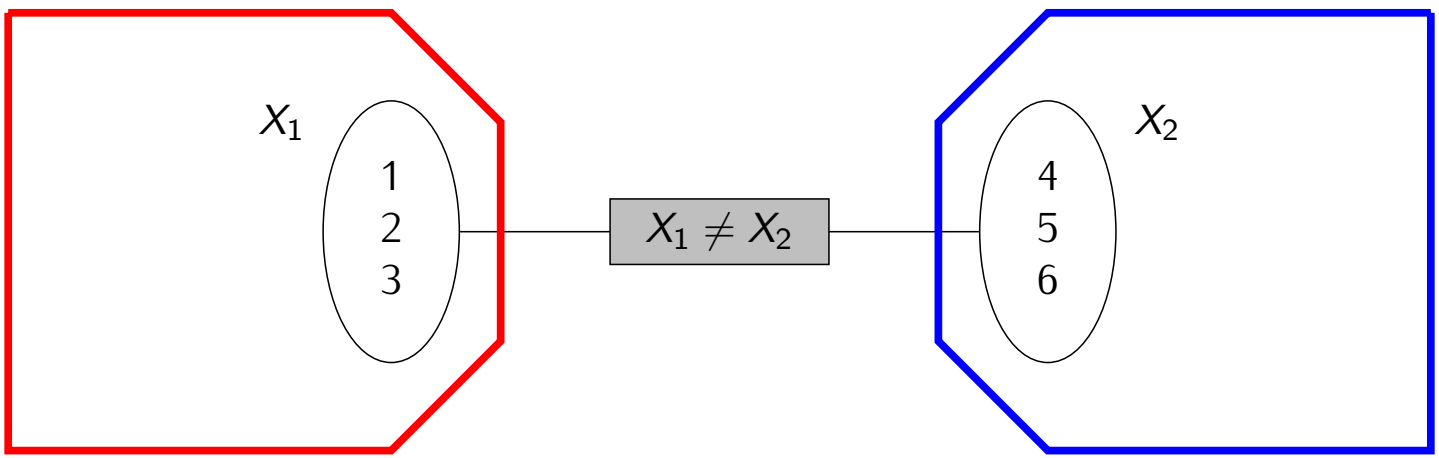




- Assigner en priorité les variables les plus centrales est une manière de favoriser la génération de **composantes connexes de même taille**

Contraintes universelles

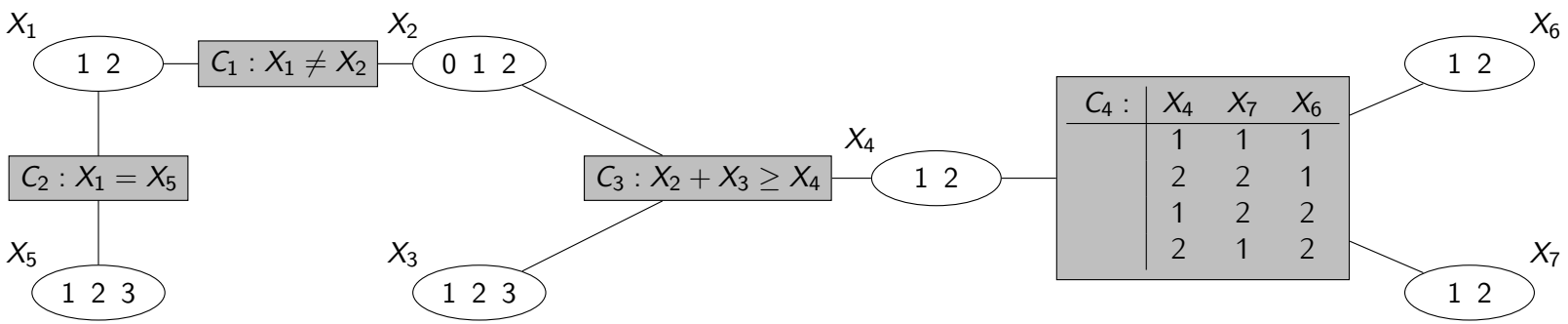
- Les contraints universelles sont des contraintes **nécessairement satisfaites** peu importe les valeurs considérées pour les variables de sa portée
- Une fois détectée, la contrainte universelle est **simplement supprimée** du réseau actuel



- L'objectif est de simplifier les traitements et de **favoriser la décomposition**

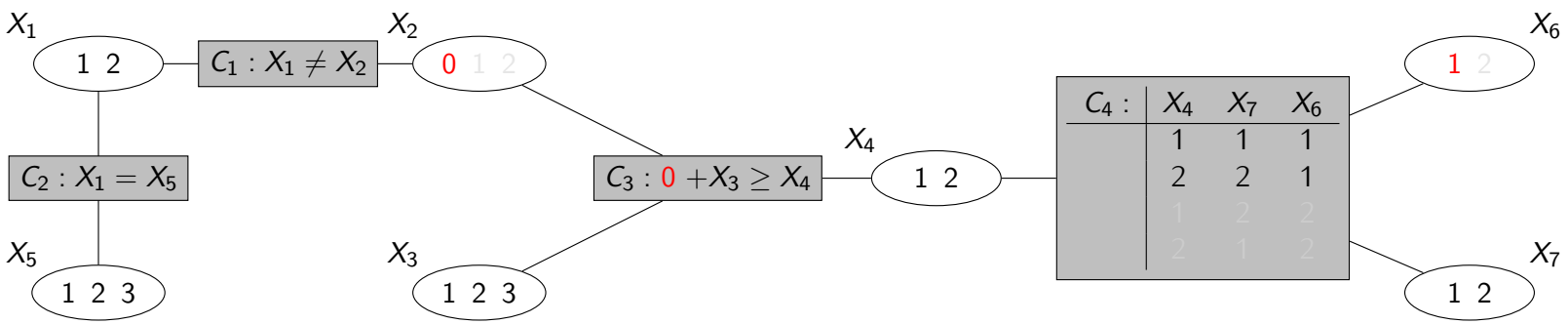
Le caching

- Le caching est une technique clé pour tout compilateur calculant une forme de DAG
- Deux réseaux de contraintes sont détectés comme “équivalents” quand ils sont identiques
- Pour un caching efficace, la taille des entrées doit être petite :
  - enregistrer le domaine courant des variables non assignées
  - $\forall C_i$ du CN courant, si C_i est AC, $|scp(C_i) > 2|$ et $\exists X_j \in scp(C_i)$ tel que D_i est modifié, alors une projection de C_i est stockée



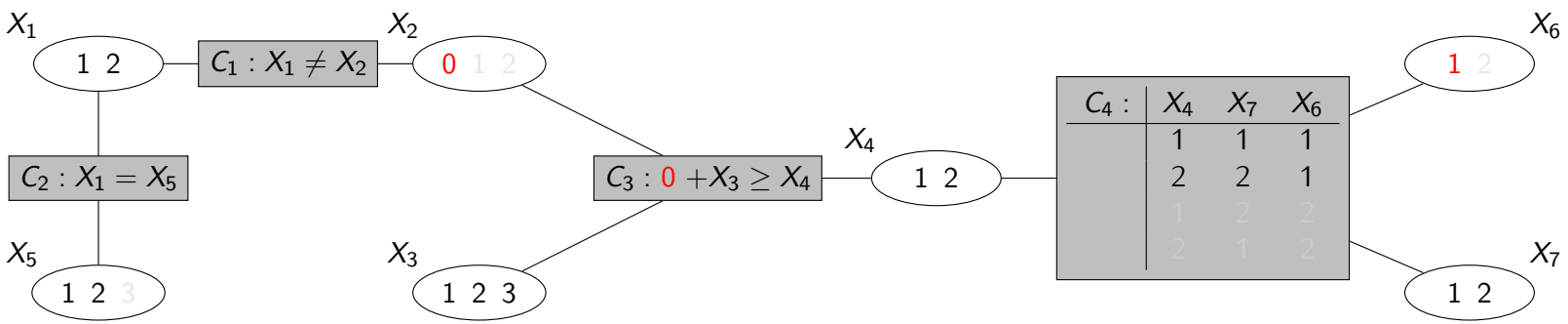
Le caching

- Le caching est une technique clé pour tout compilateur calculant une forme de DAG
- Deux réseaux de contraintes sont détectés comme “équivalents” quand ils sont identiques
- Pour un caching efficace, la taille des entrées doit être petite :
  - enregistrer le domaine courant des variables non assignées
  - $\forall C_i$ du CN courant, si C_i est AC, $|scp(C_i) > 2|$ et $\exists X_j \in scp(C_i)$ tel que D_i est modifié, alors une projection de C_i est stockée



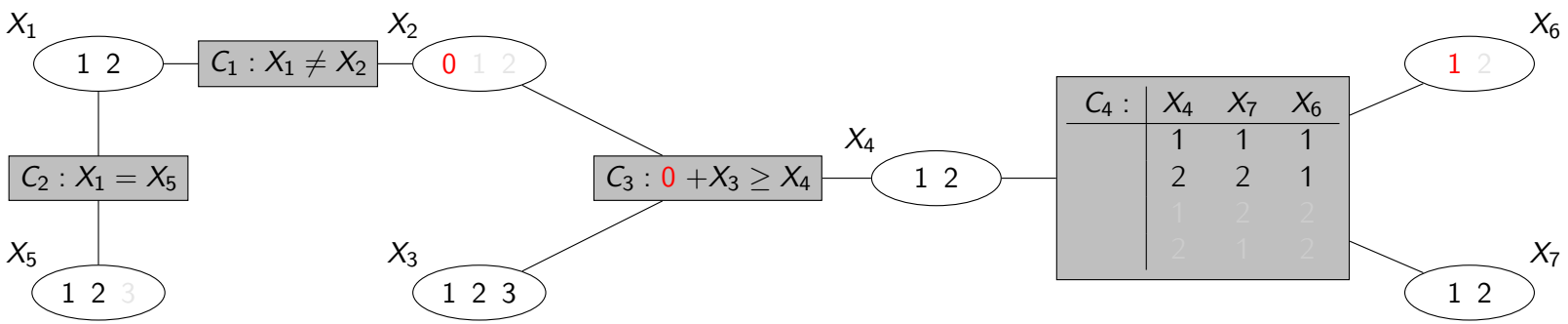
Le caching

- Le caching est une technique clé pour tout compilateur calculant une forme de DAG
- Deux réseaux de contraintes sont détectés comme “équivalents” quand ils sont identiques
- Pour un caching efficace, la taille des entrées doit être petite :
  - enregistrer le domaine courant des variables non assignées
  - $\forall C_i$ du CN courant, si C_i est AC, $|scp(C_i) > 2|$ et $\exists X_j \in scp(C_i)$ tel que D_i est modifié, alors une projection de C_i est stockée



Le caching

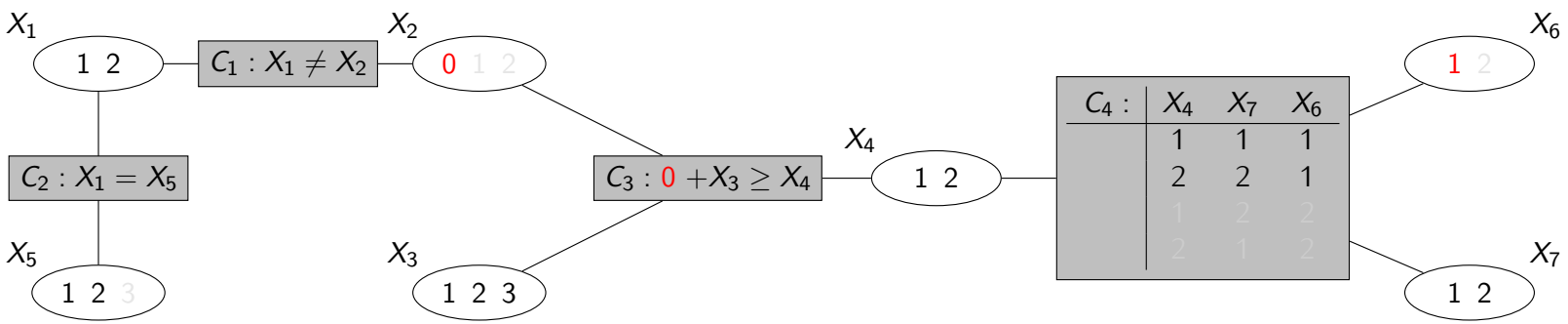
- Le caching est une technique clé pour tout compilateur calculant une forme de DAG
- Deux réseaux de contraintes sont détectés comme “équivalents” quand ils sont identiques
- Pour un caching efficace, la taille des entrées doit être petite :
  - enregistrer le domaine courant des variables non assignées
  - $\forall C_i$ du CN courant, si C_i est AC, $|scp(C_i) > 2|$ et $\exists X_j \in scp(C_i)$ tel que D_i est modifié, alors une projection de C_i est stockée



X_1	1	2	X_3	0	1	2	X_4	1	2	X_5	1	2	X_7	1	2											

Le caching

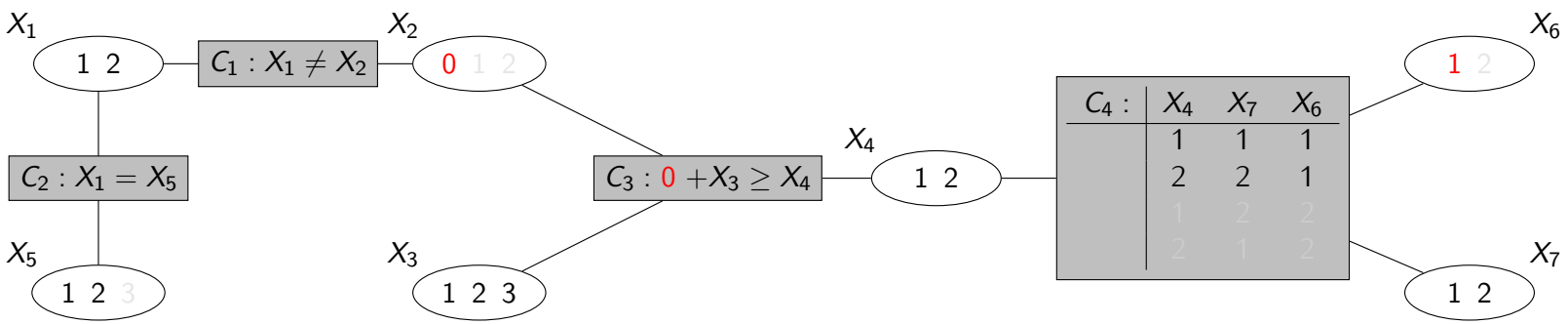
- Le caching est une technique clé pour tout compilateur calculant une forme de DAG
- Deux réseaux de contraintes sont détectés comme “équivalents” quand ils sont identiques
- Pour un caching efficace, la taille des entrées doit être petite :
  - enregistrer le domaine courant des variables non assignées
  - $\forall C_i$ du CN courant, si C_i est AC, $|scp(C_i) > 2|$ et $\exists X_j \in scp(C_i)$ tel que D_i est modifié, alors une projection de C_i est stockée



X_1	1	2	X_3	0	1	2	X_4	1	2	X_5	1	2	X_7	1	2	C_3	0	+	X_3	≥	X_4					

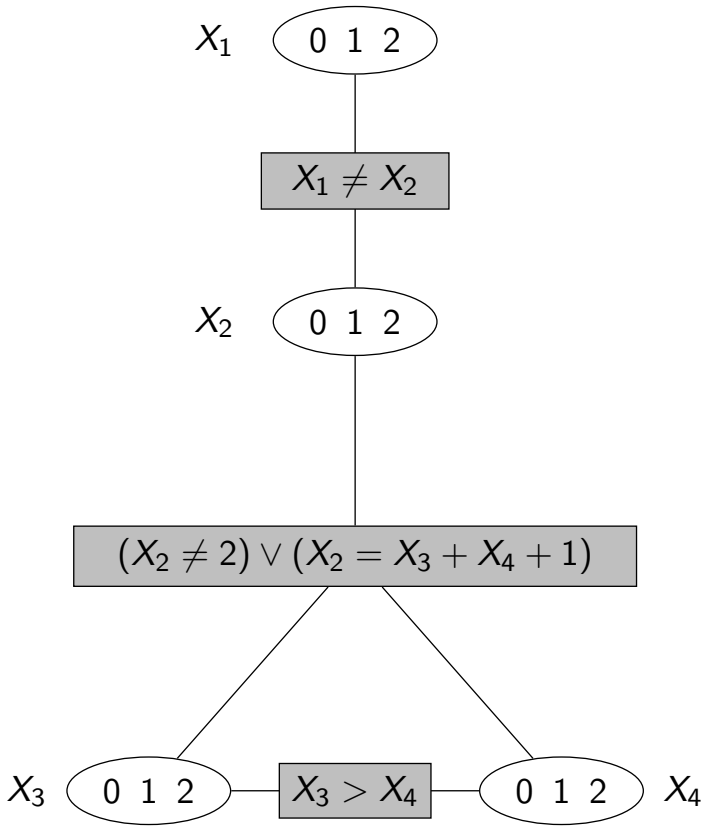
Le caching

- Le caching est une technique clé pour tout compilateur calculant une forme de DAG
- Deux réseaux de contraintes sont détectés comme “équivalents” quand ils sont identiques
- Pour un caching efficace, la taille des entrées doit être petite :
  - enregistrer le domaine courant des variables non assignées
  - $\forall C_i$ du CN courant, si C_i est AC, $|scp(C_i) > 2|$ et $\exists X_j \in scp(C_i)$ tel que D_i est modifié, alors une projection de C_i est stockée

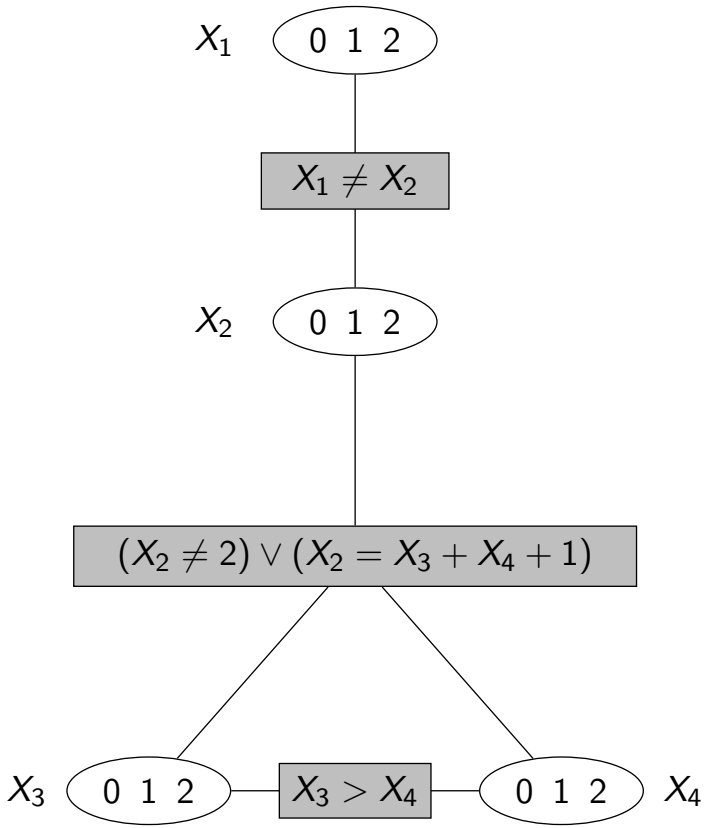


X_1	1	2	X_3	0	1	2	X_4	1	2	X_5	1	2	X_7	1	2	C_3	0	+	X_3	≥	X_4	C_4	1	2	2	1

Graphes de décision décomposables multivalués : construction

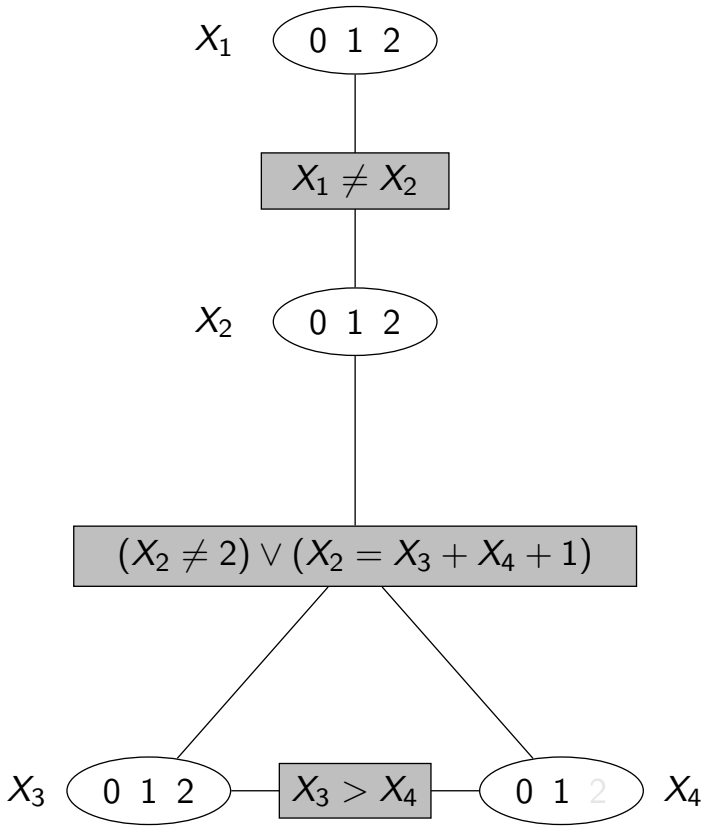


Graphes de décision décomposables multivalués : construction



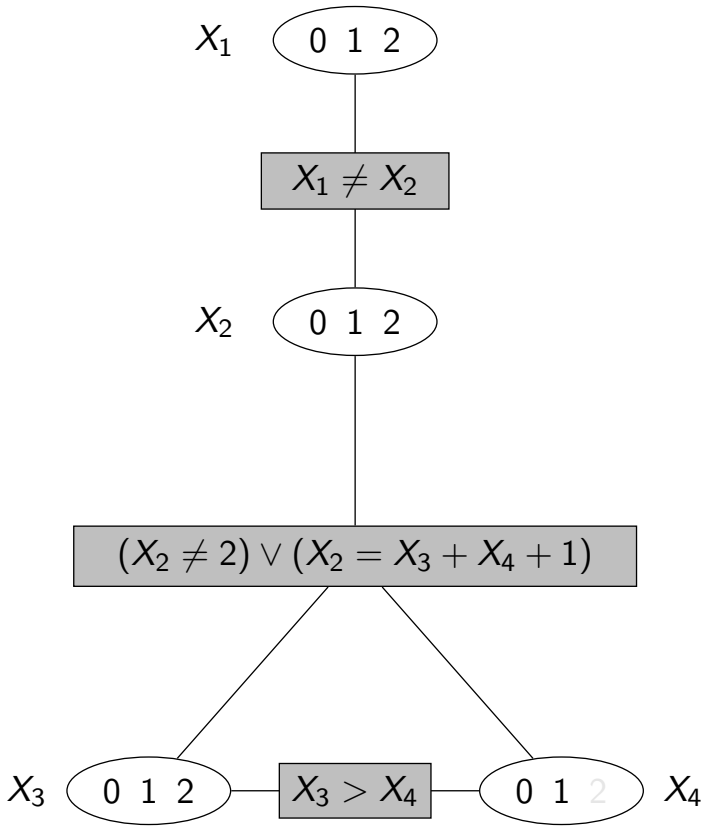
- Cohérence d'arc

Graphes de décision décomposables multivalués : construction



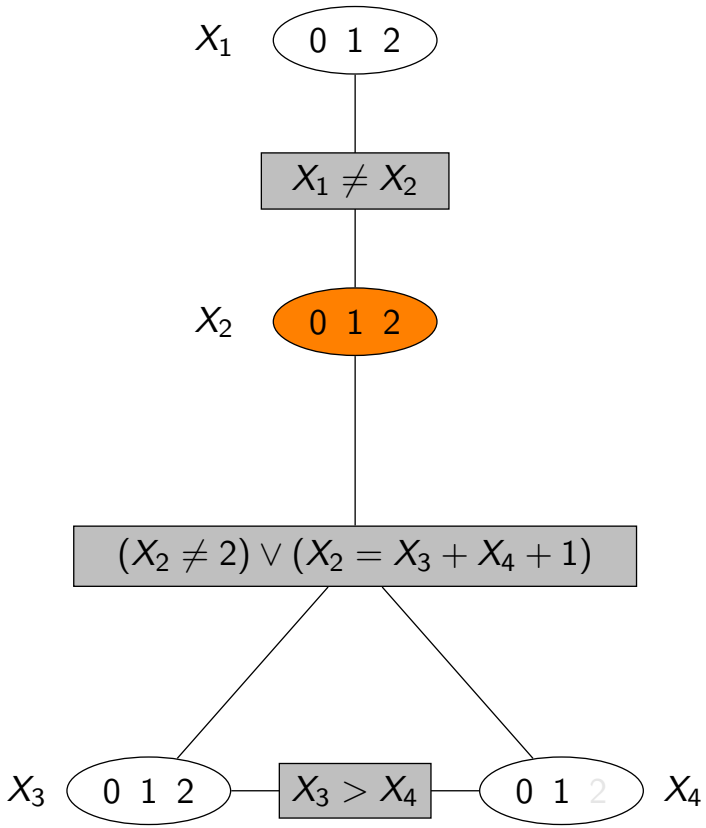
- Cohérence d'arc

Graphes de décision décomposables multivalués : construction



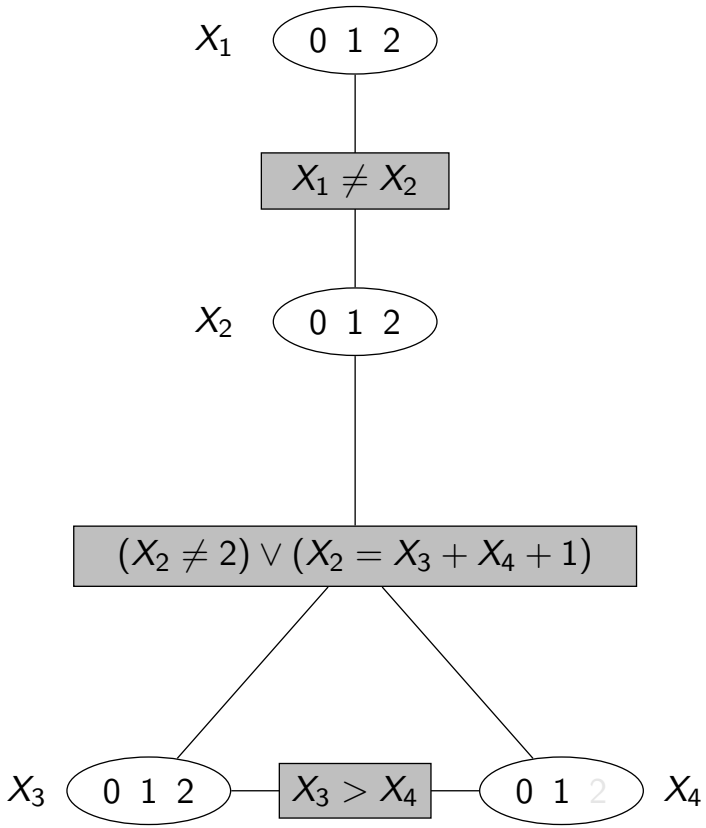
- Sélection de la variable

Graphes de décision décomposables multivalués : construction



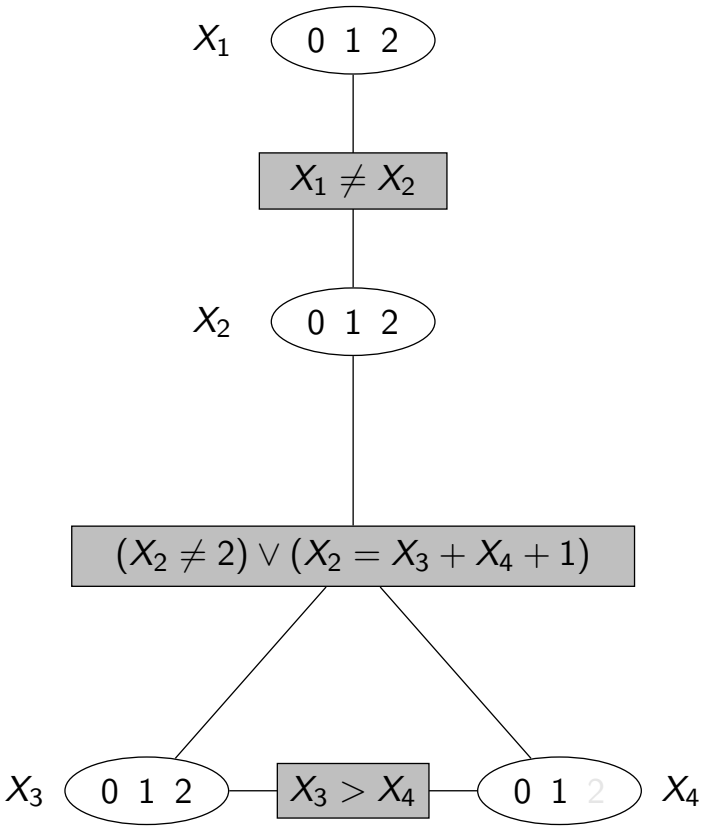
- Sélection de la variable

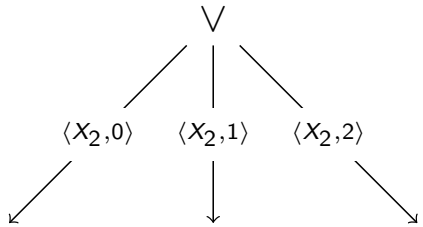
Graphes de décision décomposables multivalués : construction



- Nœud OU déterministe

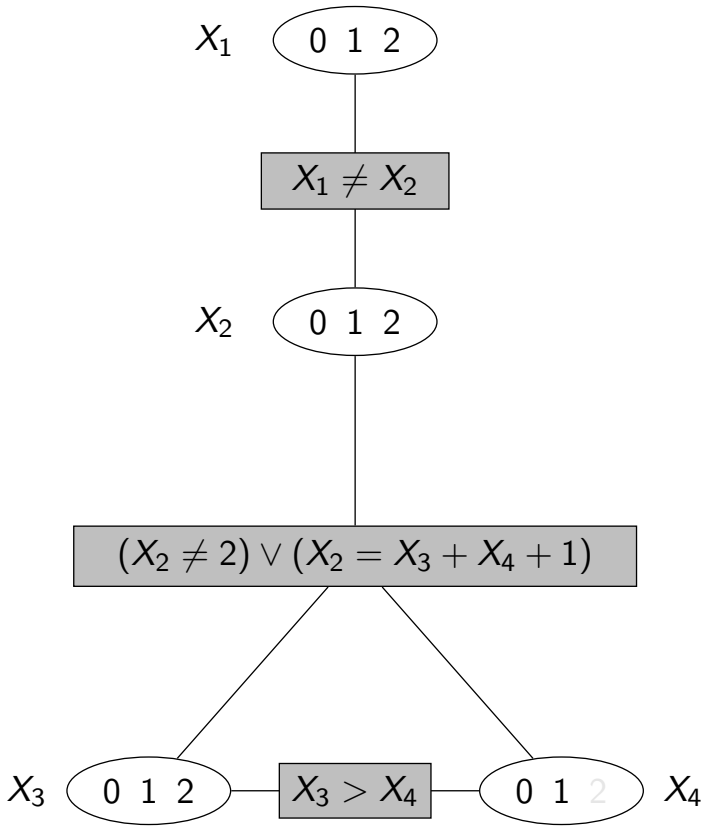
Graphes de décision décomposables multivalués : construction

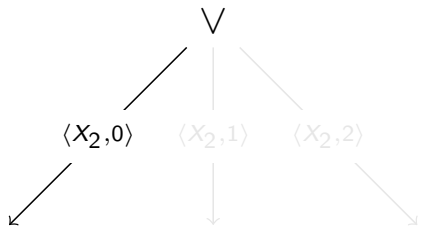




- Nœud OU déterministe

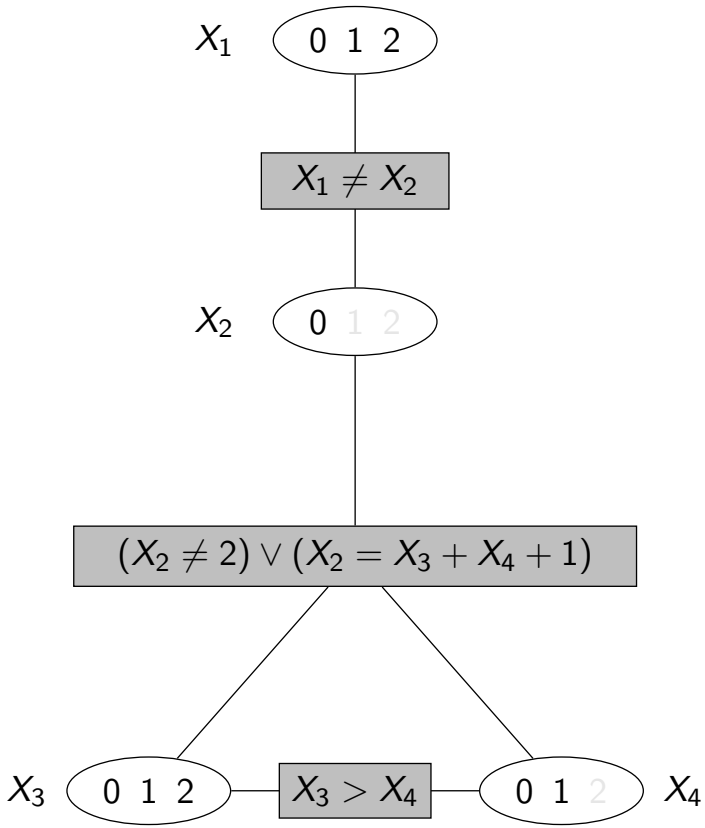
Graphes de décision décomposables multivalués : construction

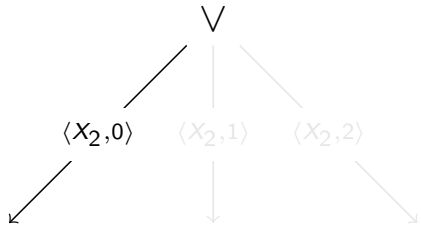




- Considère récursivement chaque branche

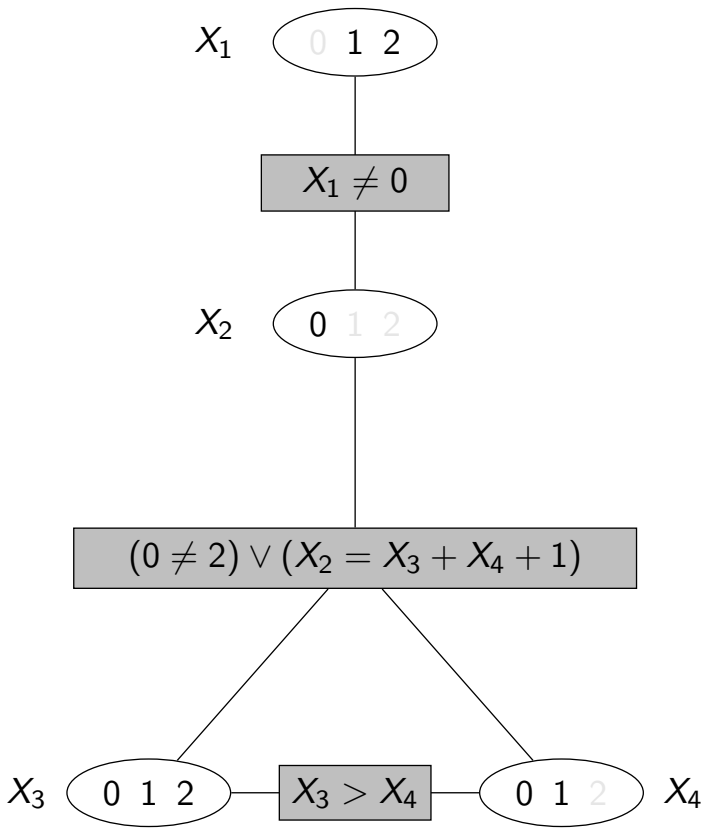
Graphes de décision décomposables multivalués : construction

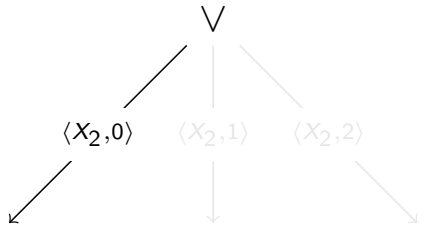




- X_2 est assigné à 0

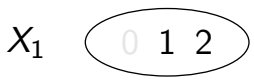
Graphes de décision décomposables multivalués : construction

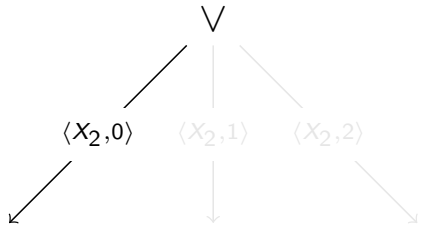


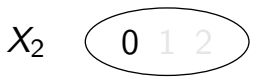


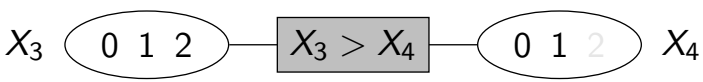
- Appliquer la cohérence d'arc

Graphes de décision décomposables multivalués : construction



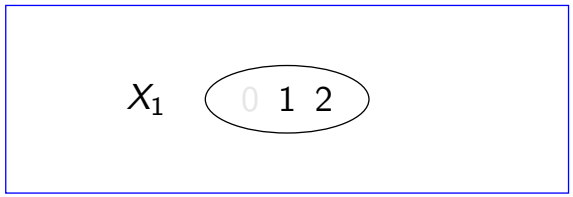


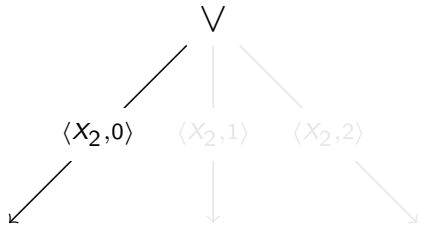


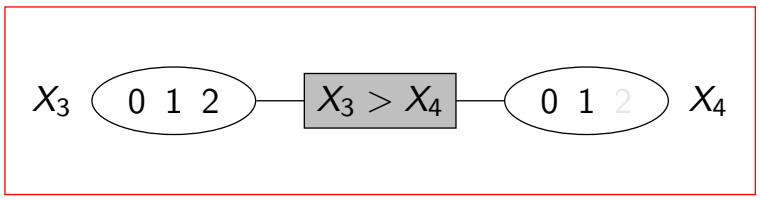


- Simplification du réseau de contraintes

Graphes de décision décomposables multivalués : construction

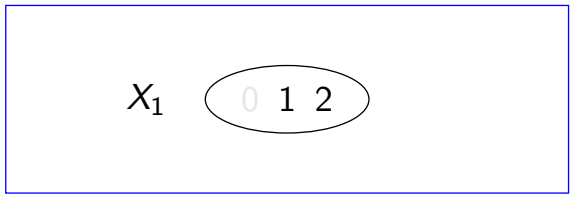


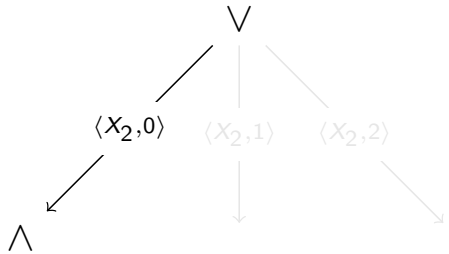


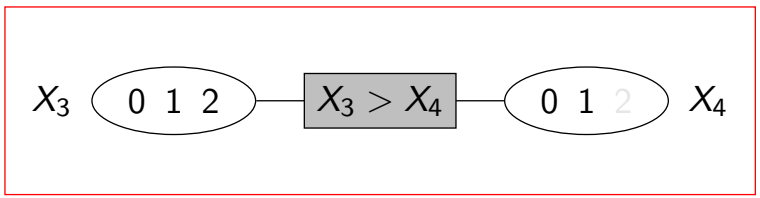


- Calcul des composantes connexes

Graphes de décision décomposables multivalués : construction

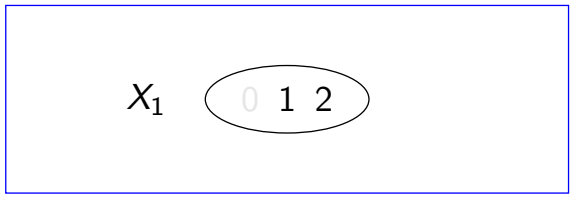


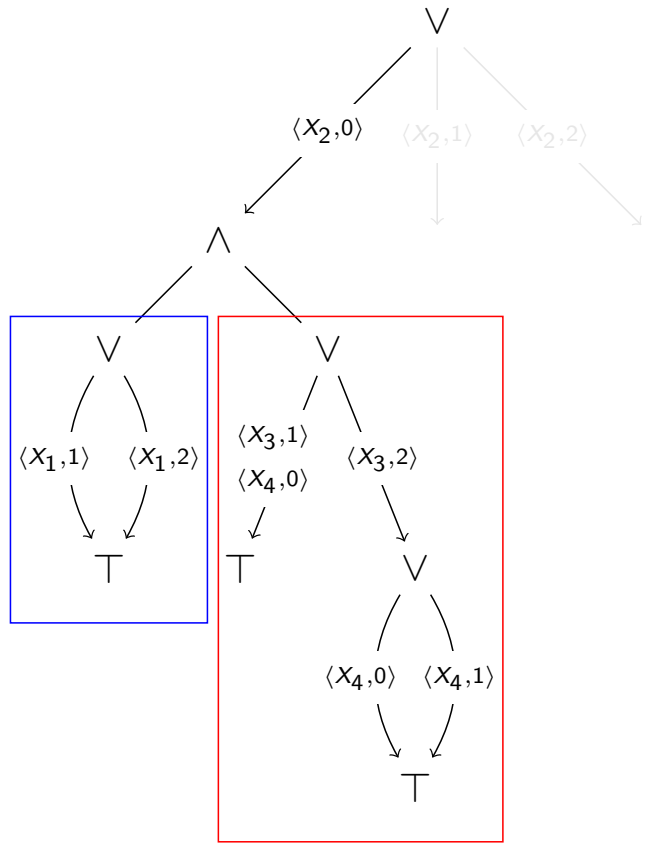


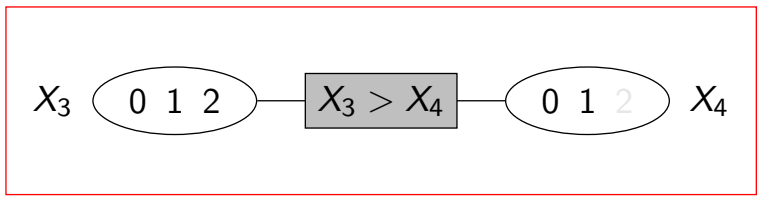


- Nœud ET décomposable

Graphes de décision décomposables multivalués : construction

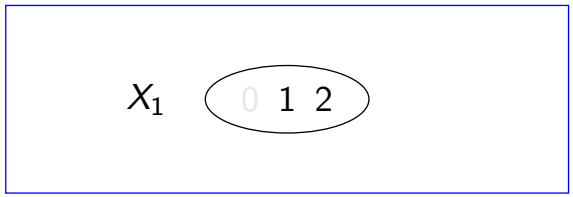


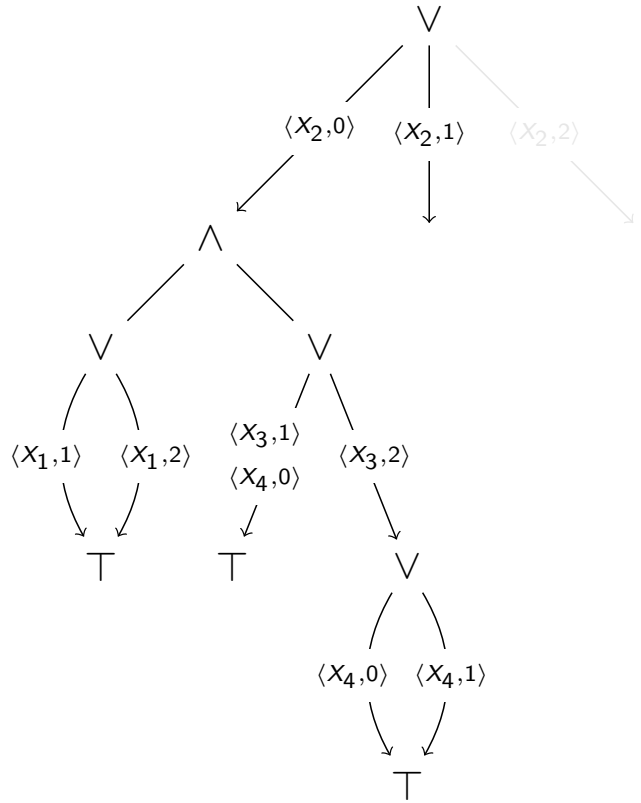


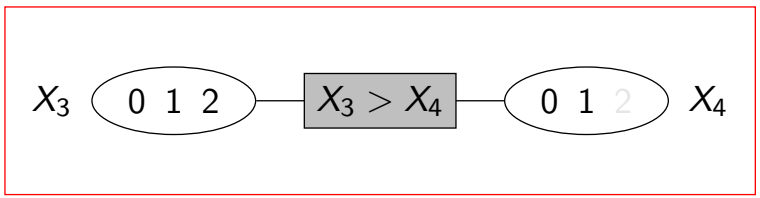


- Compilation séparée des sous-problèmes

Graphes de décision décomposables multivalués : construction

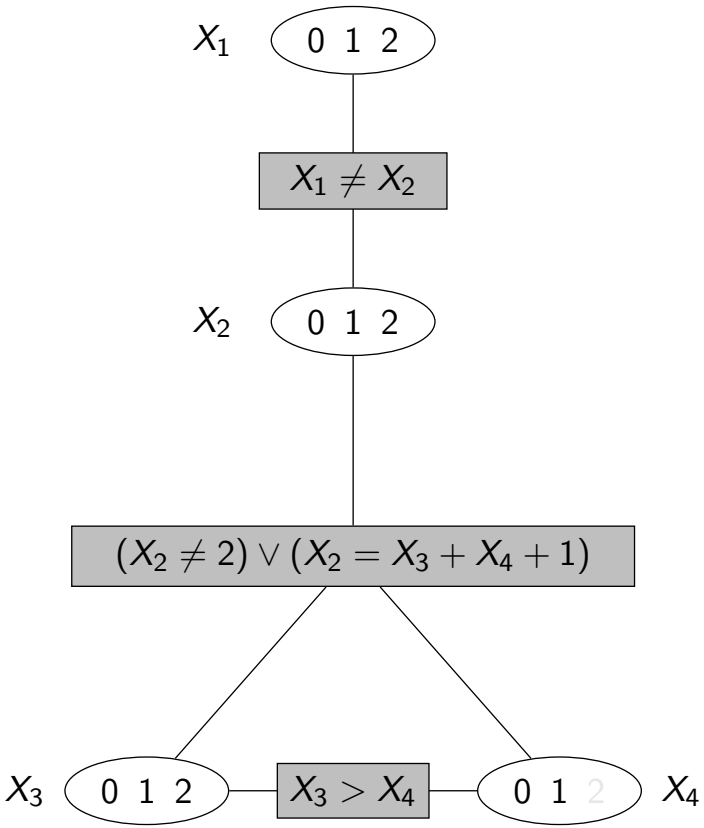


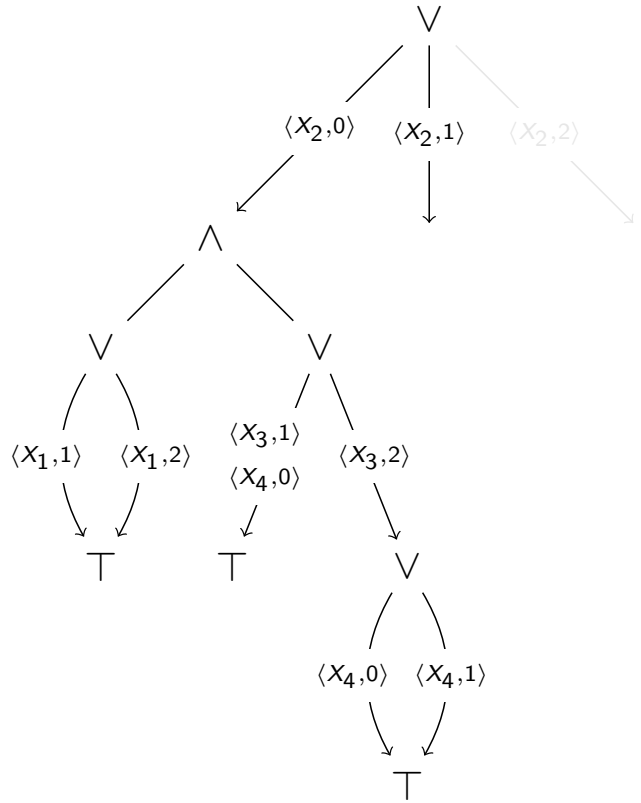




- Prise en compte de la branche suivante

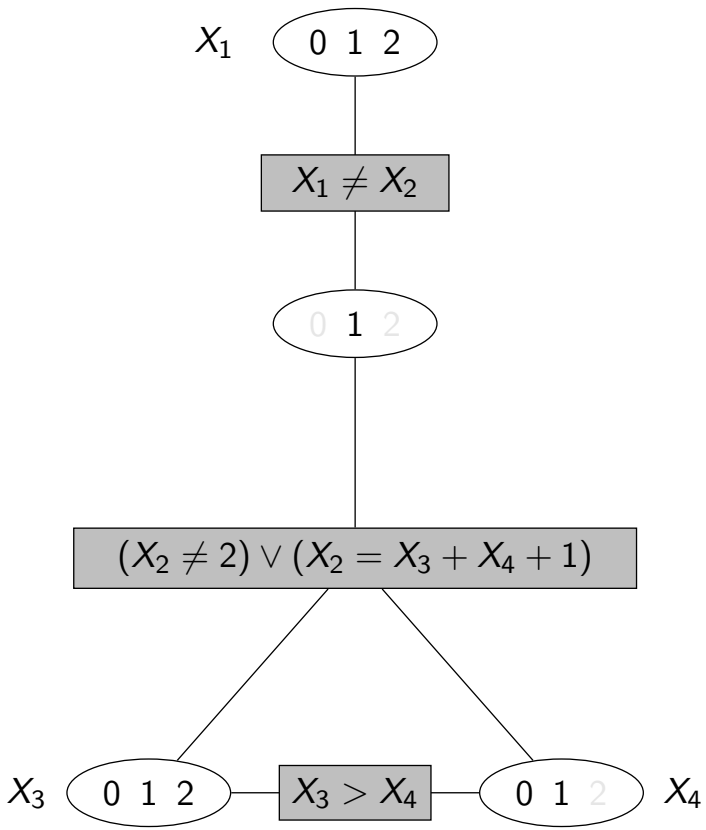
Graphes de décision décomposables multivalués : construction

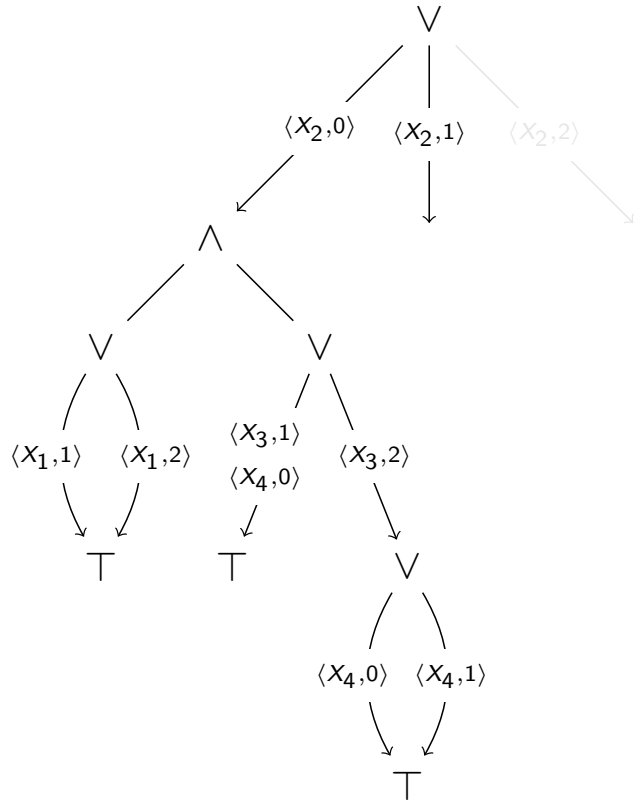




✦ Retour en arrière

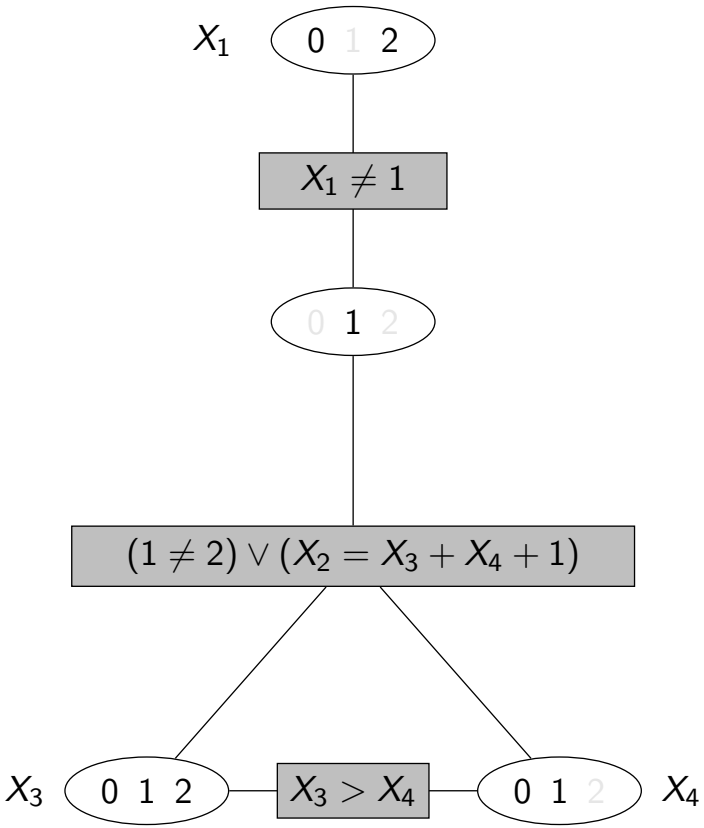
Graphes de décision décomposables multivalués : construction

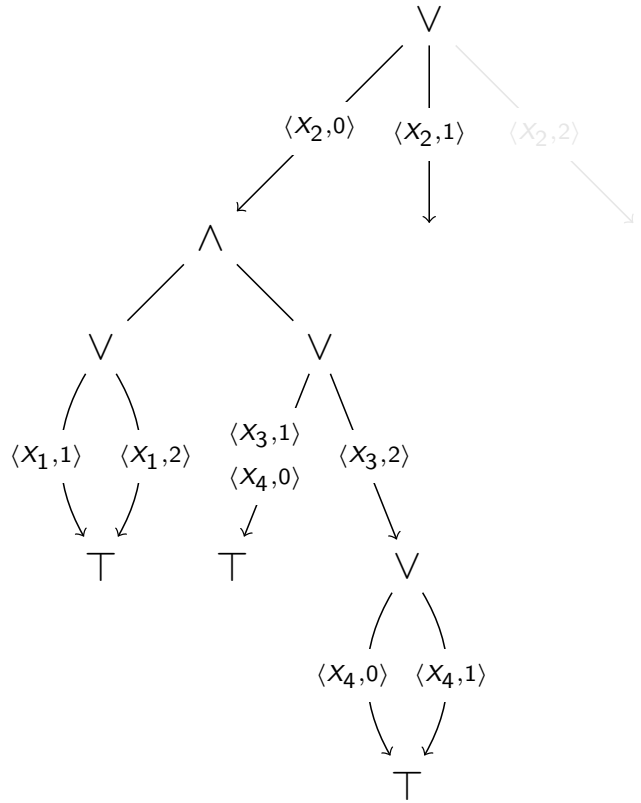




- X_2 est assignée à 1

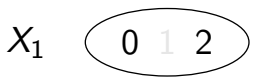
Graphes de décision décomposables multivalués : construction

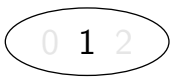


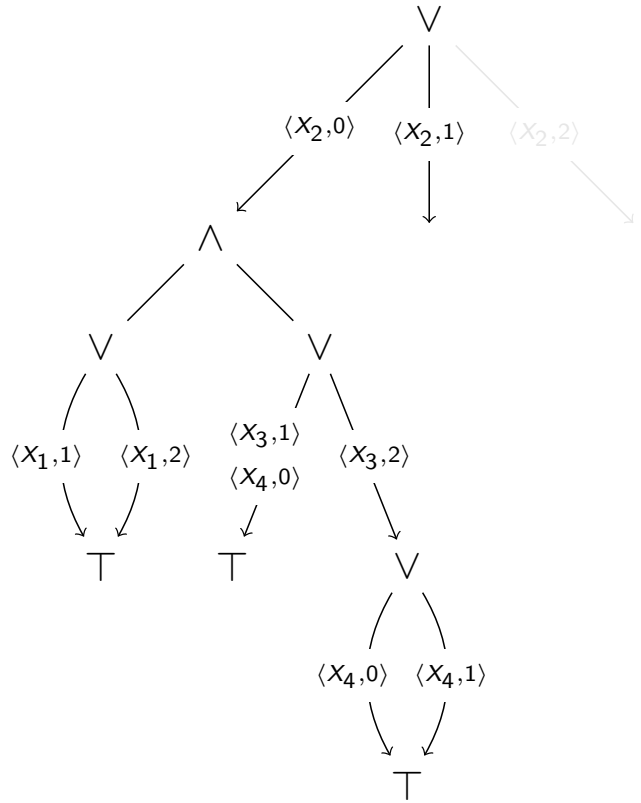


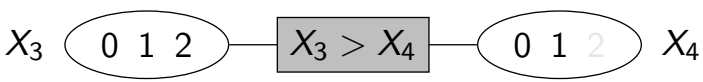
- Appliquer la cohérence d'arc

Graphes de décision décomposables multivalués : construction



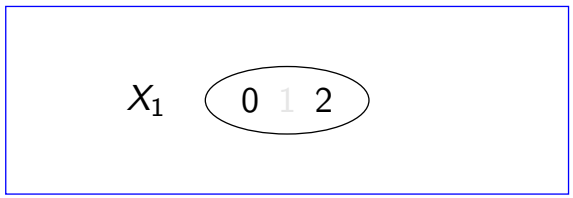


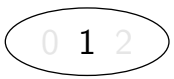


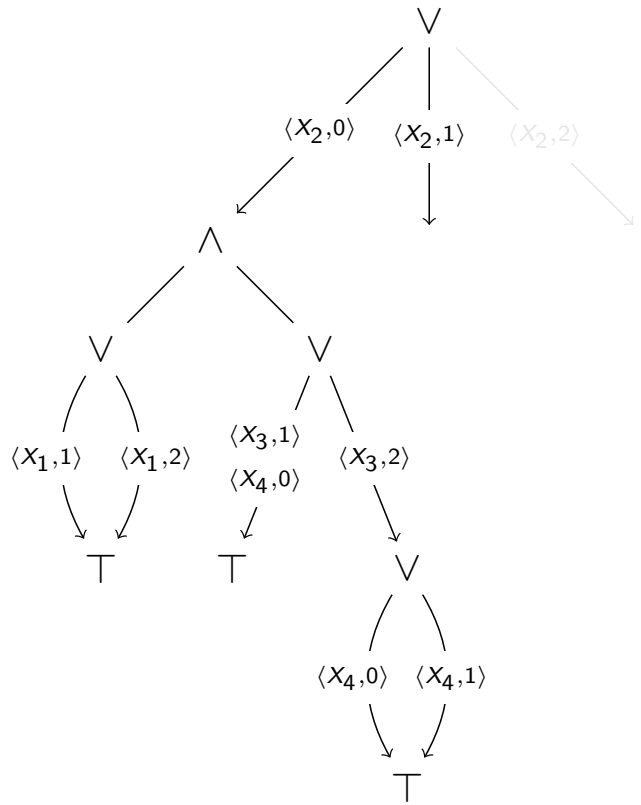


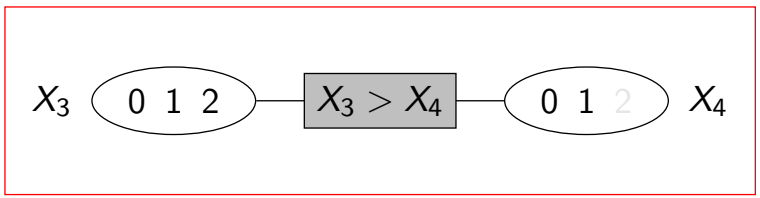
- Simplification du réseau de contraintes

Graphes de décision décomposables multivalués : construction



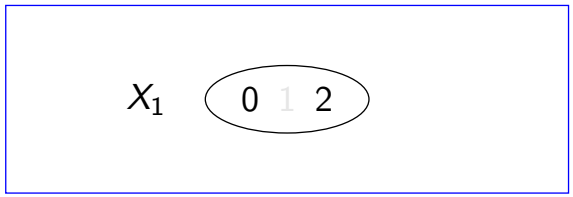


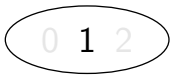


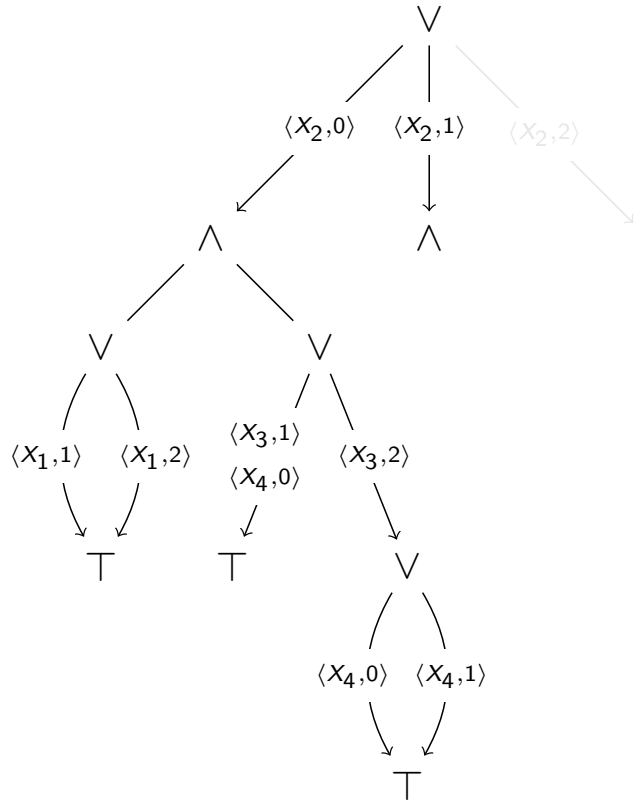


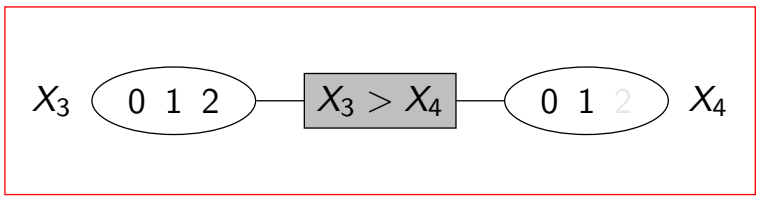
- Calcul des composantes connexes

Graphes de décision décomposables multivalués : construction



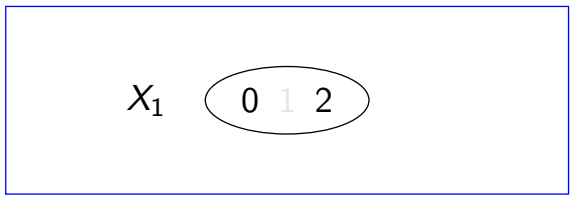


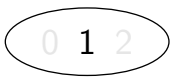


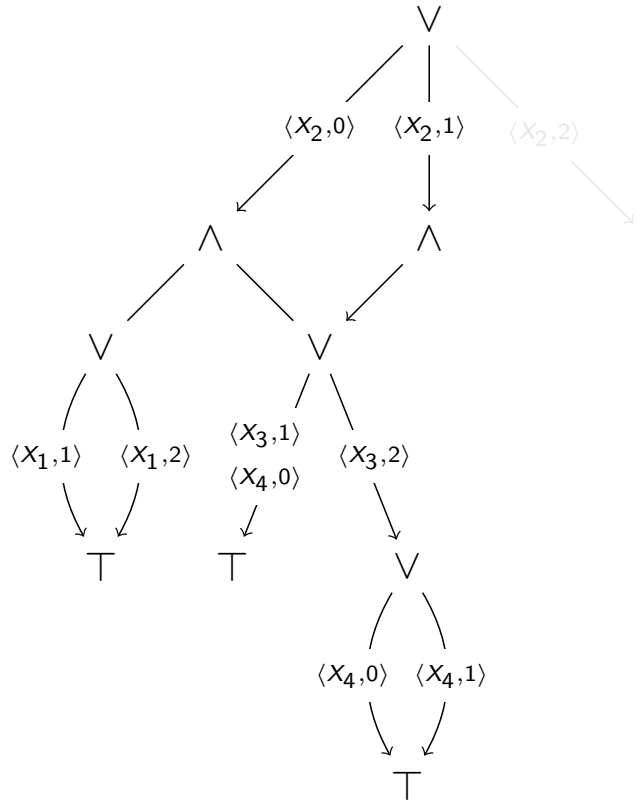


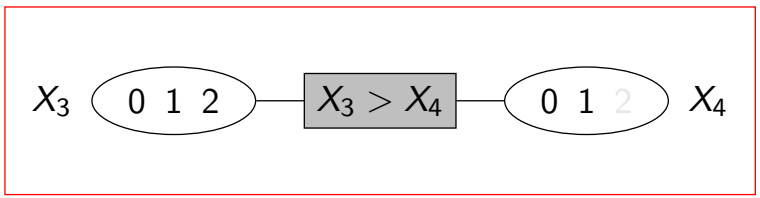
- Nœud ET décomposable

Graphes de décision décomposables multivalués : construction



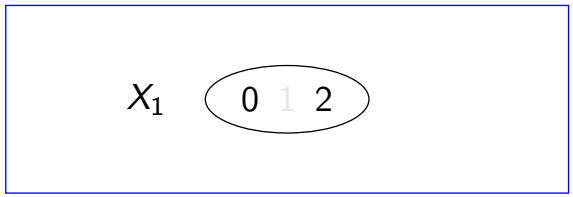


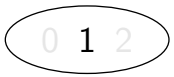


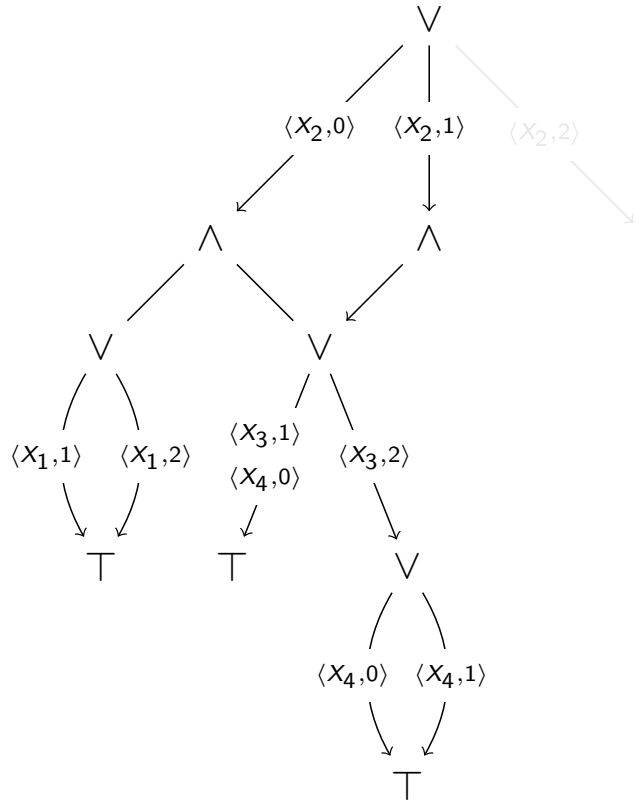


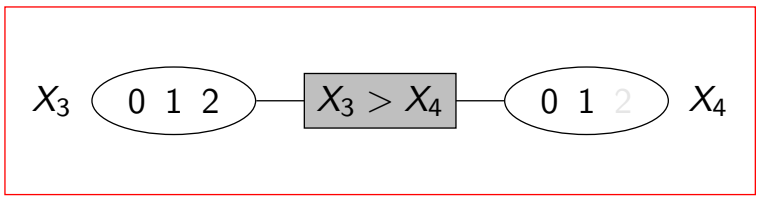
- Caching

Graphes de décision décomposables multivalués : construction





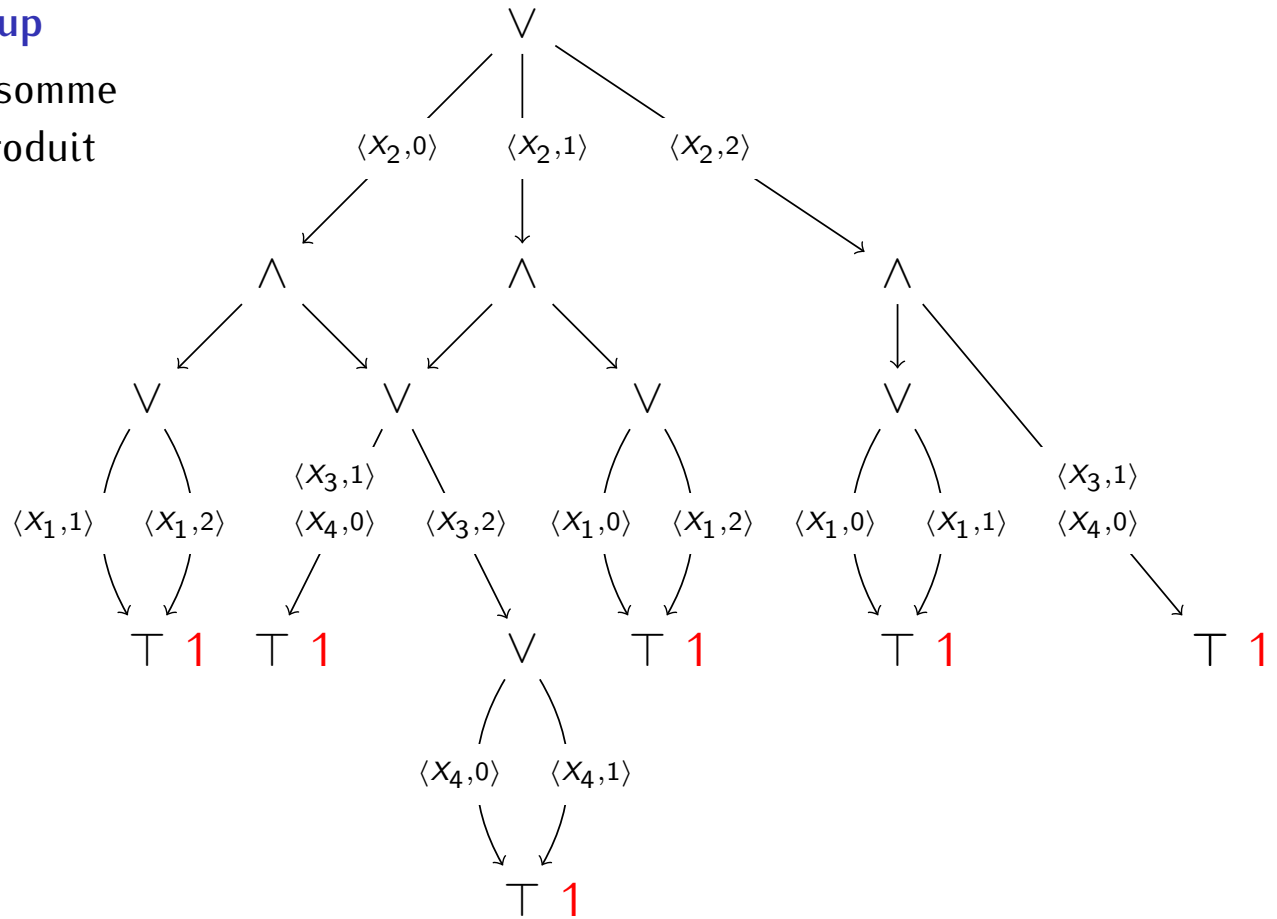




- ...

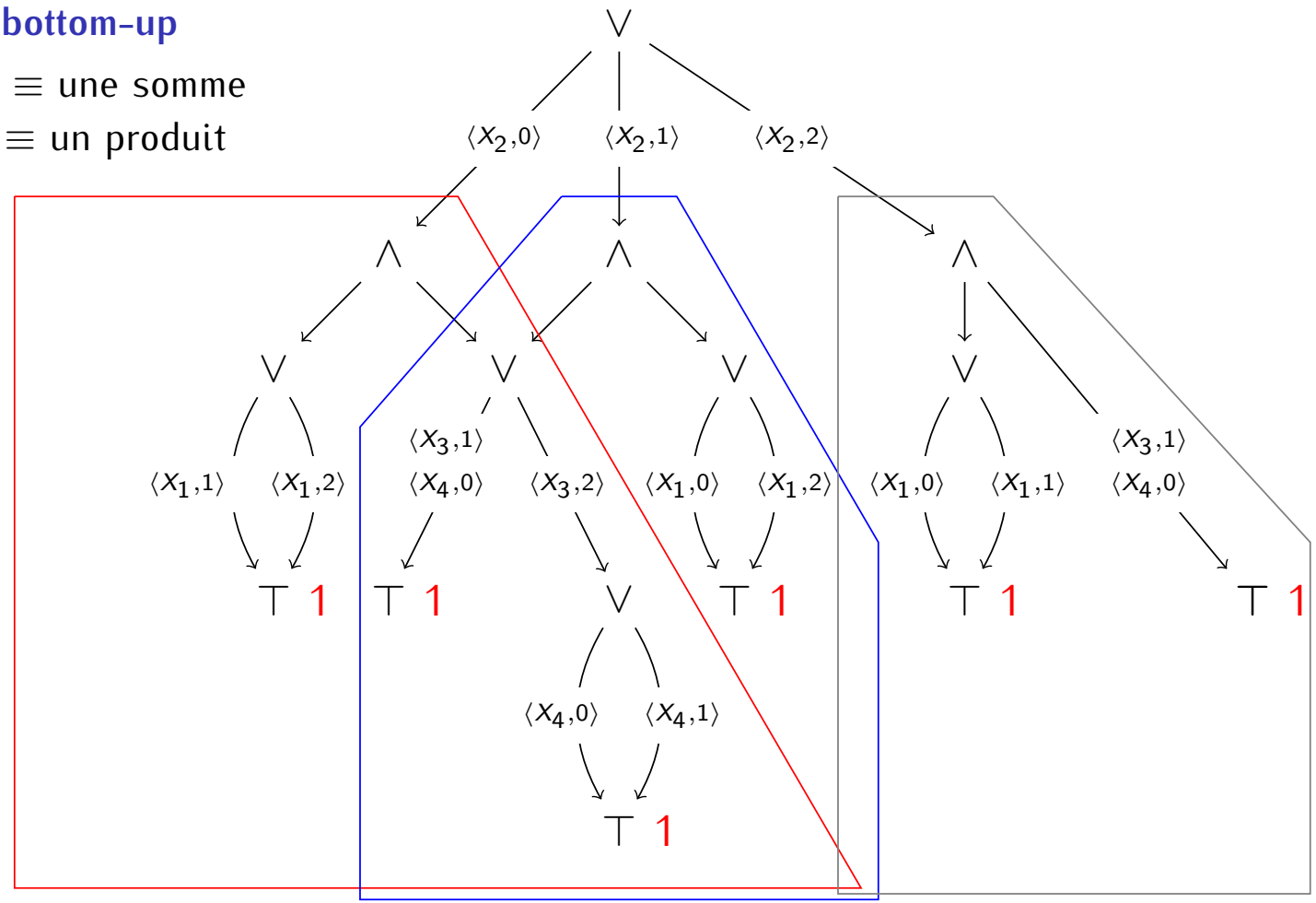
Graphes de décision décomposables multivalués : comptage de modèles

- ✧ **Approche bottom-up**
- ✧ Nœud OU ≡ une somme
- ✧ Nœud ET ≡ un produit



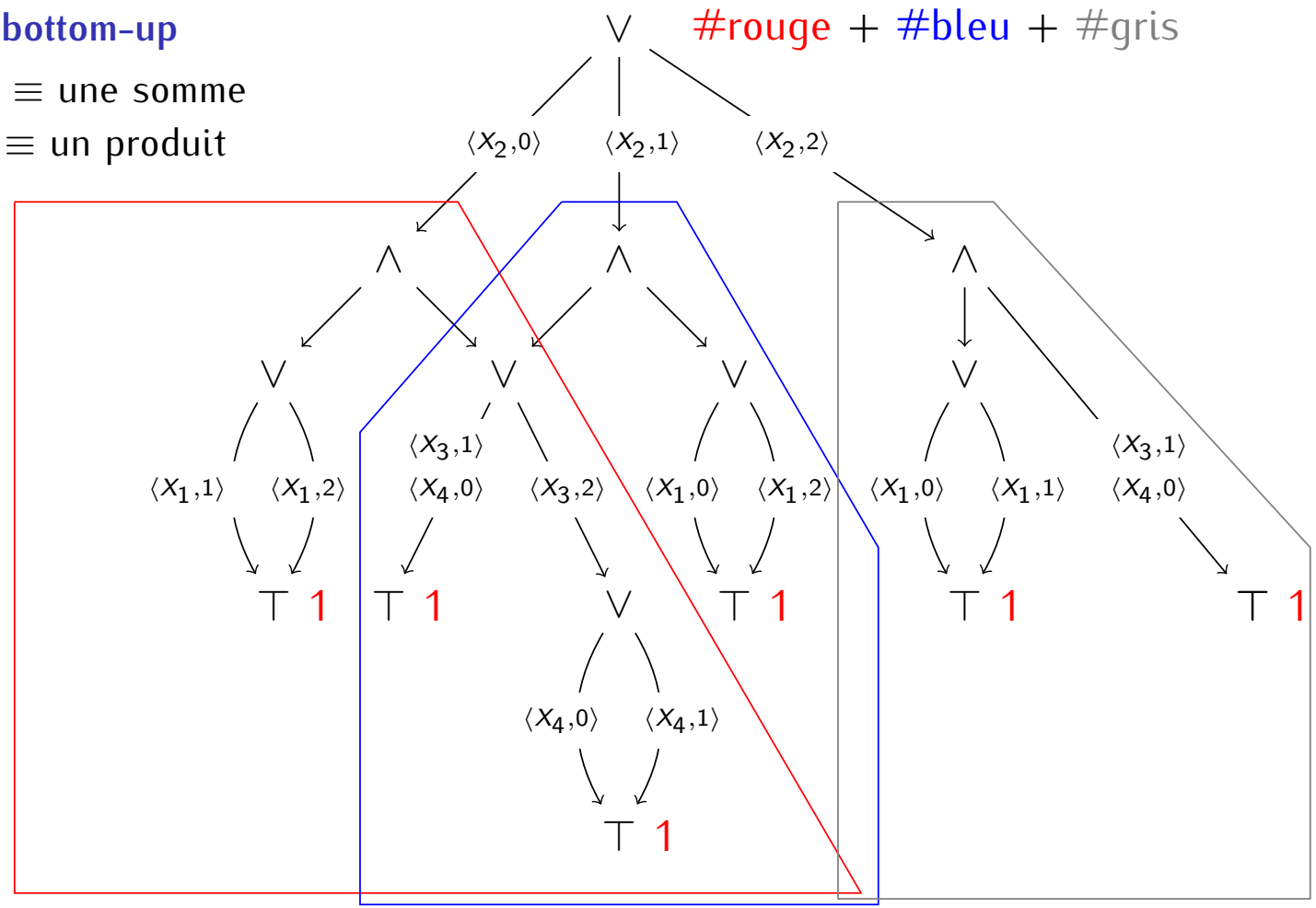
Graphes de décision décomposables multivalués : comptage de modèles

- Approche bottom-up
- Nœud OU ≡ une somme
- Nœud ET ≡ un produit



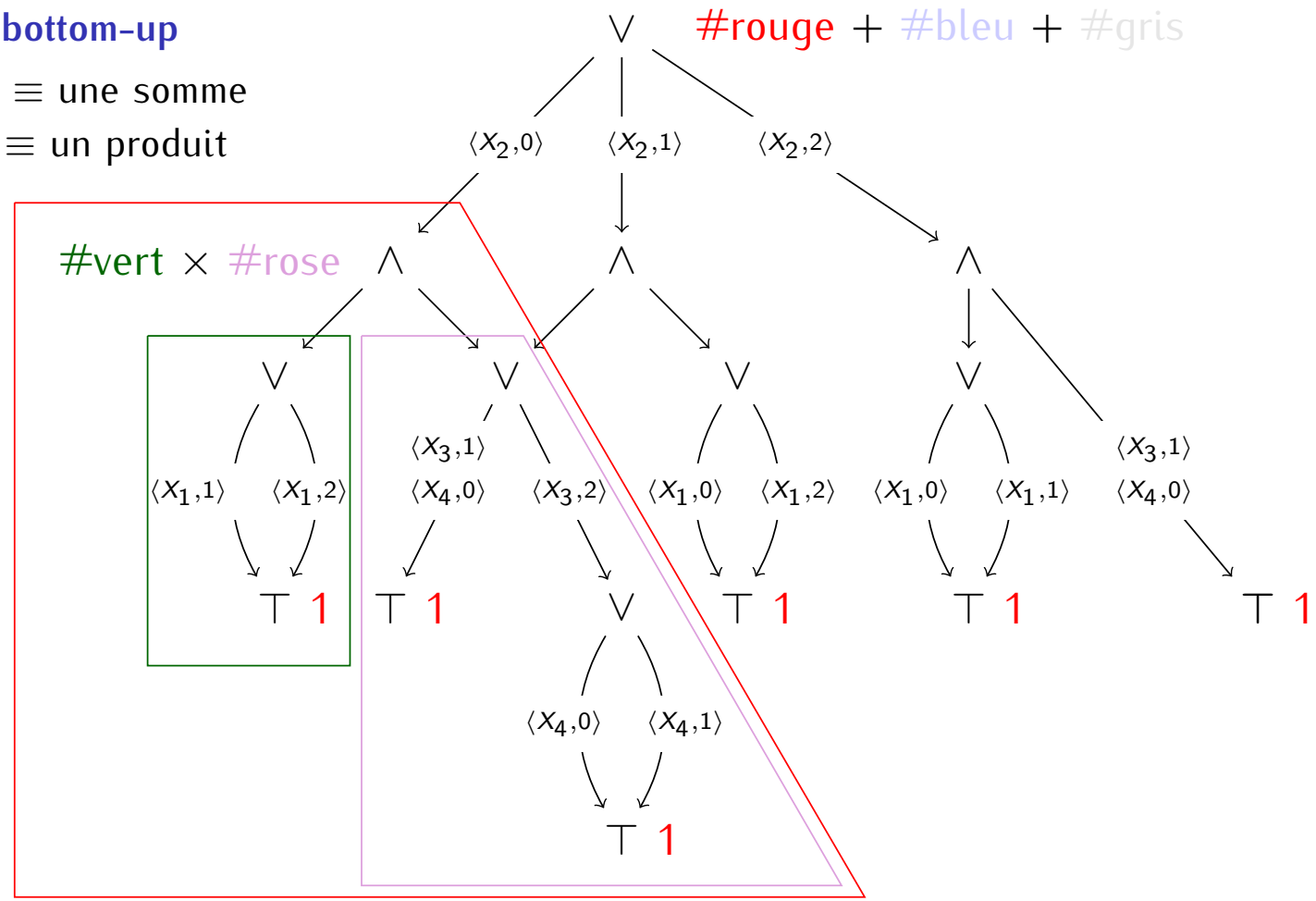
Graphes de décision décomposables multivalués : comptage de modèles

- Approche bottom-up
- Nœud OU ≡ une somme
- Nœud ET ≡ un produit



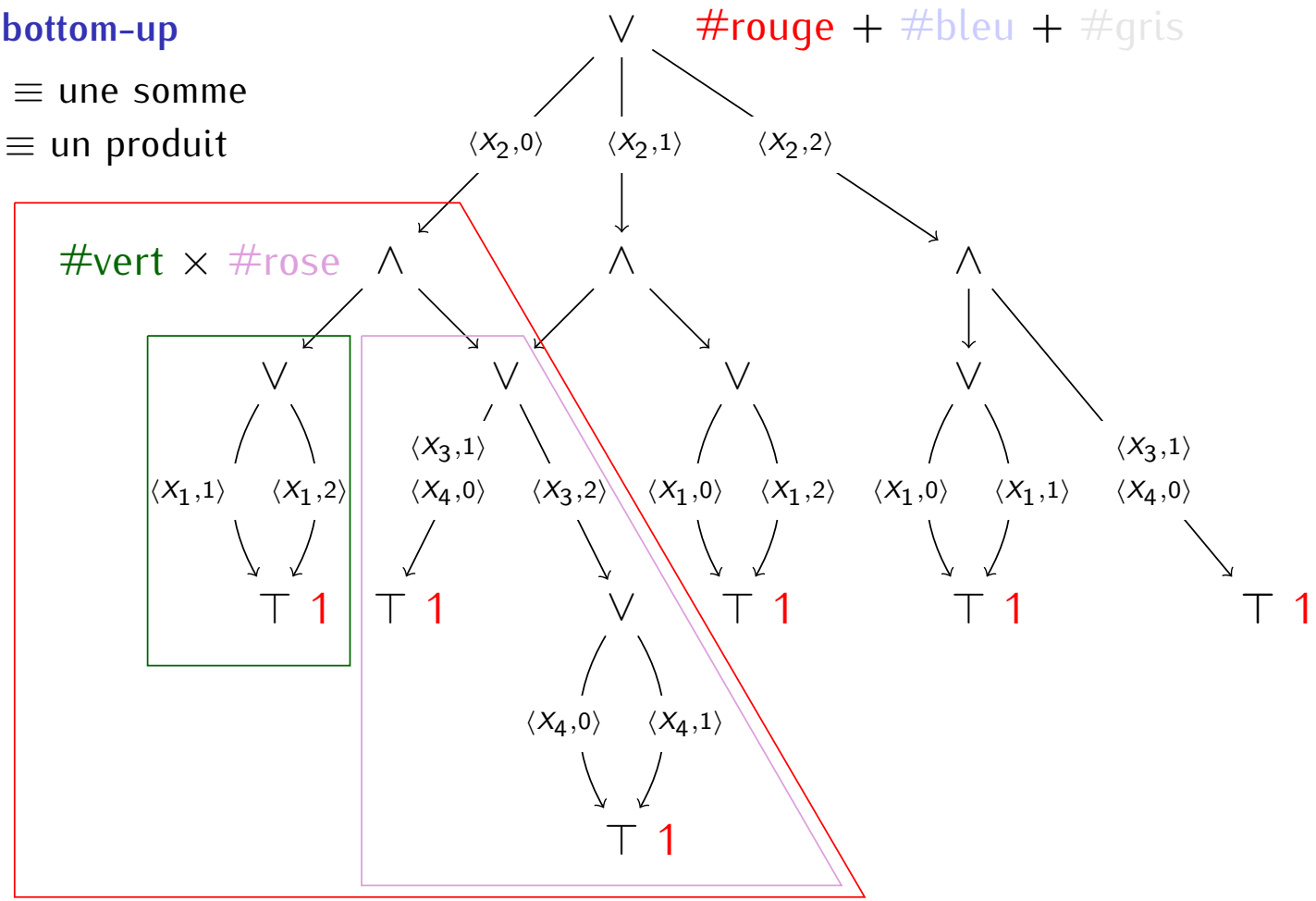
Graphes de décision décomposables multivalués : comptage de modèles

- ✧ **Approche bottom-up**
- ✧ Nœud OU ≡ une somme
- ✧ Nœud ET ≡ un produit



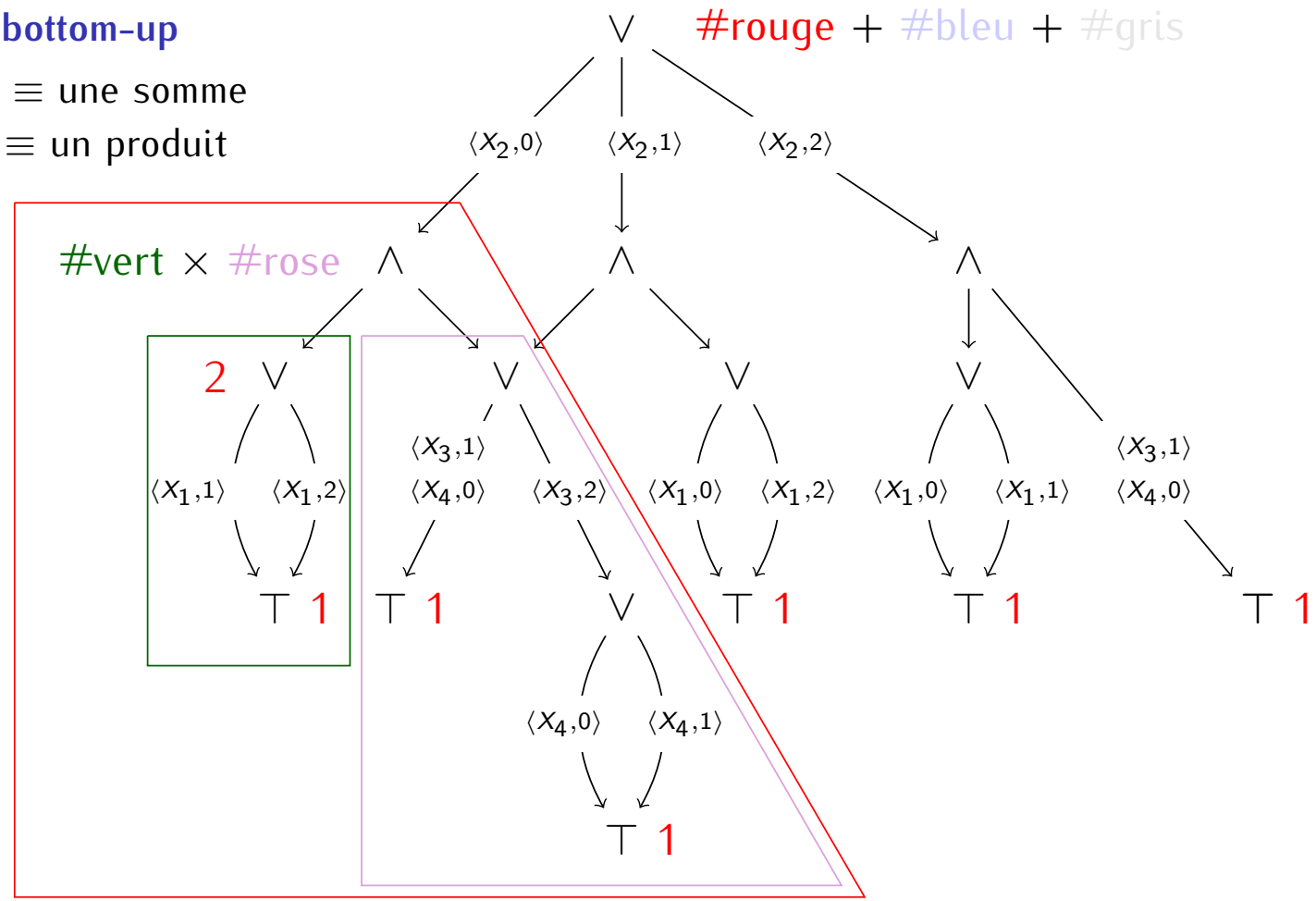
Graphes de décision décomposables multivalués : comptage de modèles

- ✧ **Approche bottom-up**
- ✧ Nœud OU ≡ une somme
- ✧ Nœud ET ≡ un produit



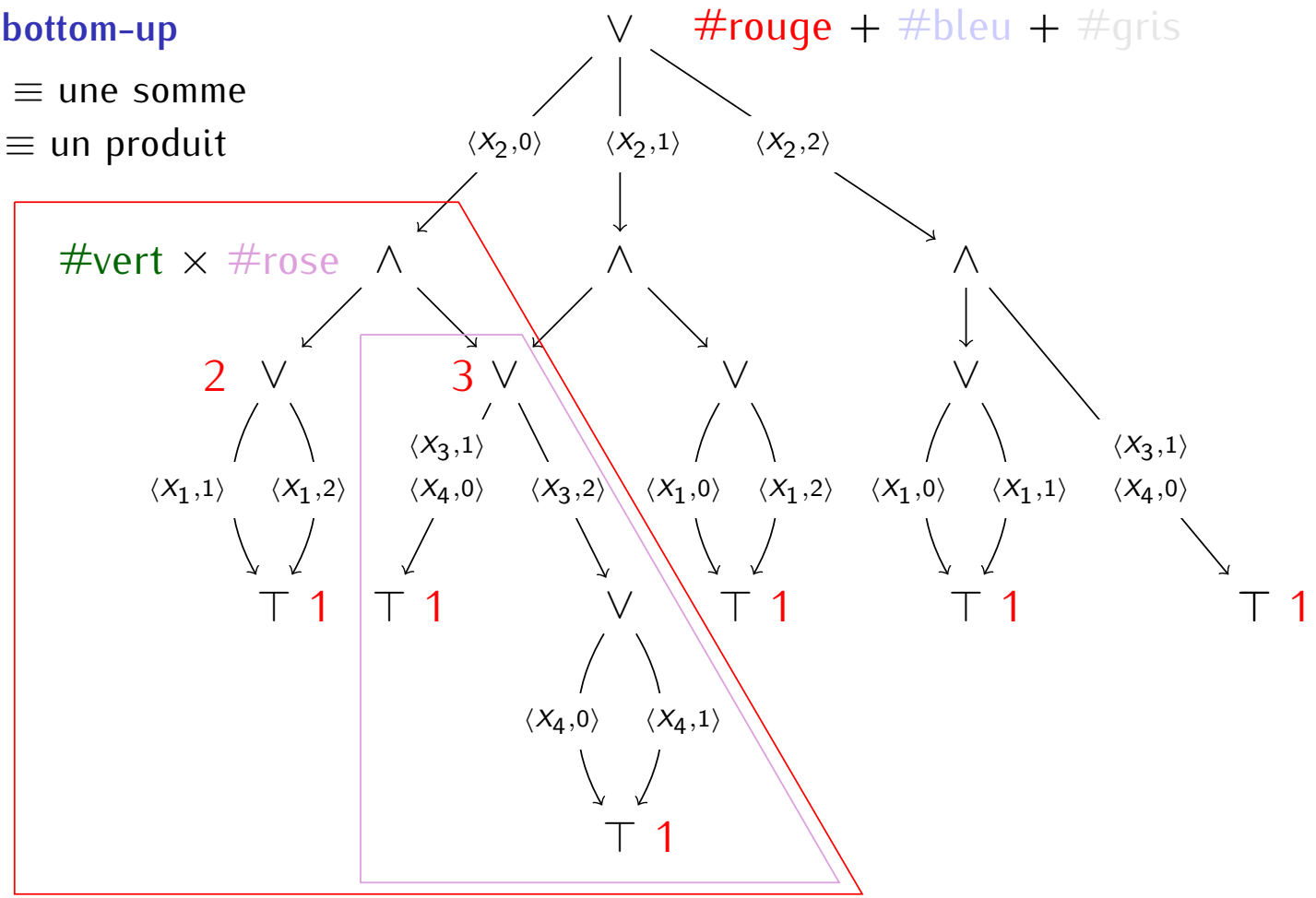
Graphes de décision décomposables multivalués : comptage de modèles

- Approche bottom-up
- Nœud OU ≡ une somme
- Nœud ET ≡ un produit



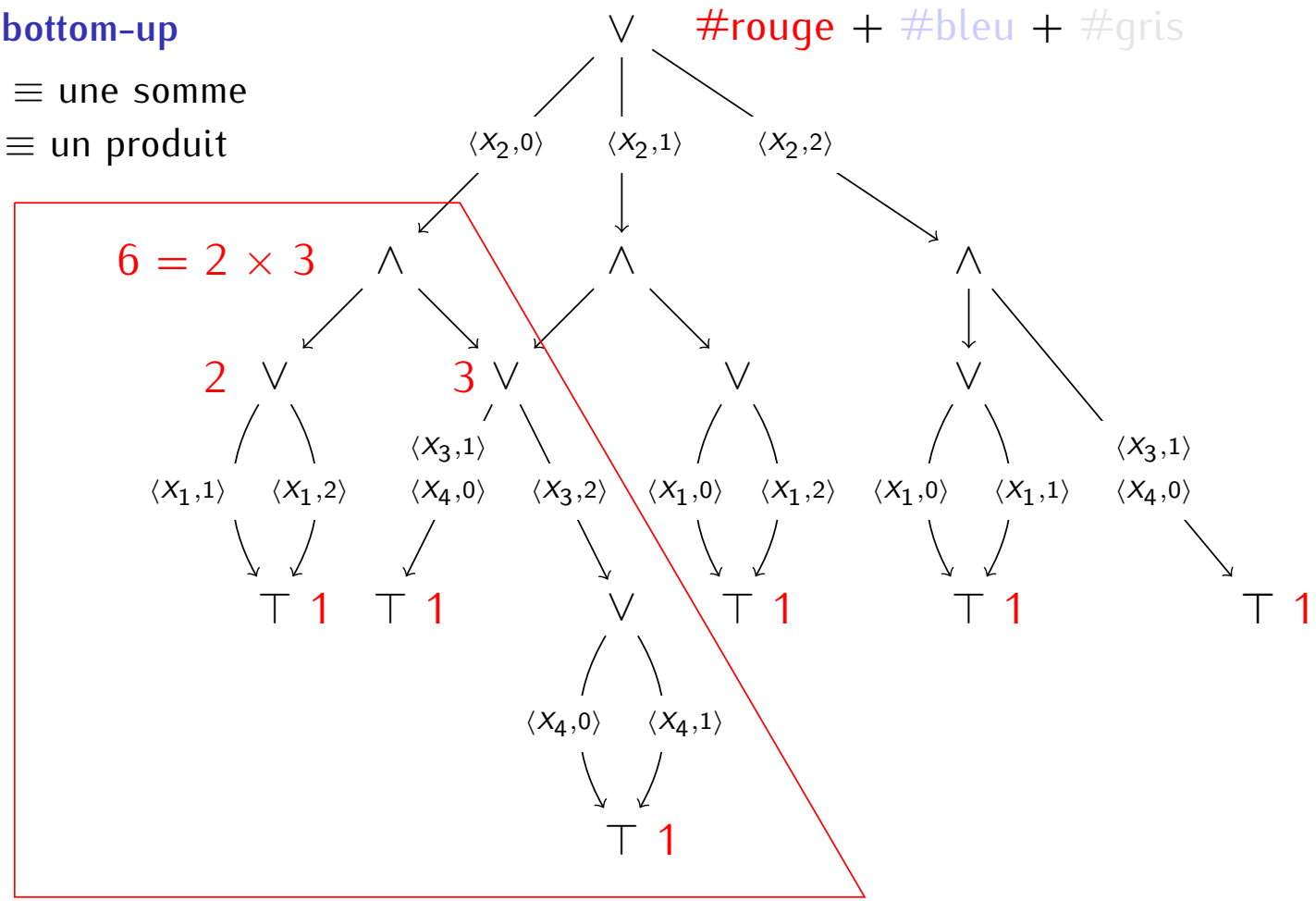
Graphes de décision décomposables multivalués : comptage de modèles

- ✧ **Approche bottom-up**
- ✧ Nœud OU ≡ une somme
- ✧ Nœud ET ≡ un produit



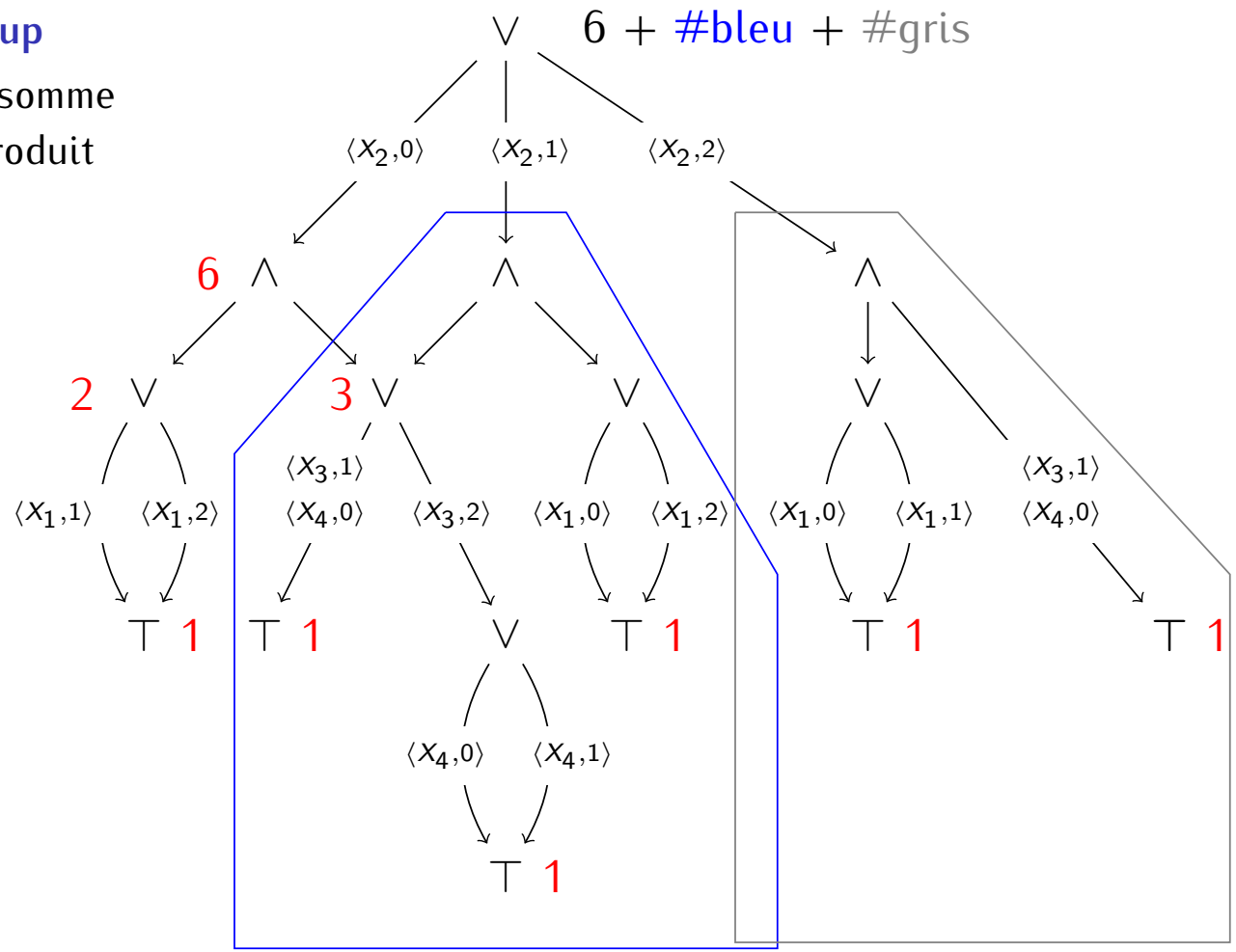
Graphes de décision décomposables multivalués : comptage de modèles

- ✧ **Approche bottom-up**
- ✧ Nœud OU ≡ une somme
- ✧ Nœud ET ≡ un produit



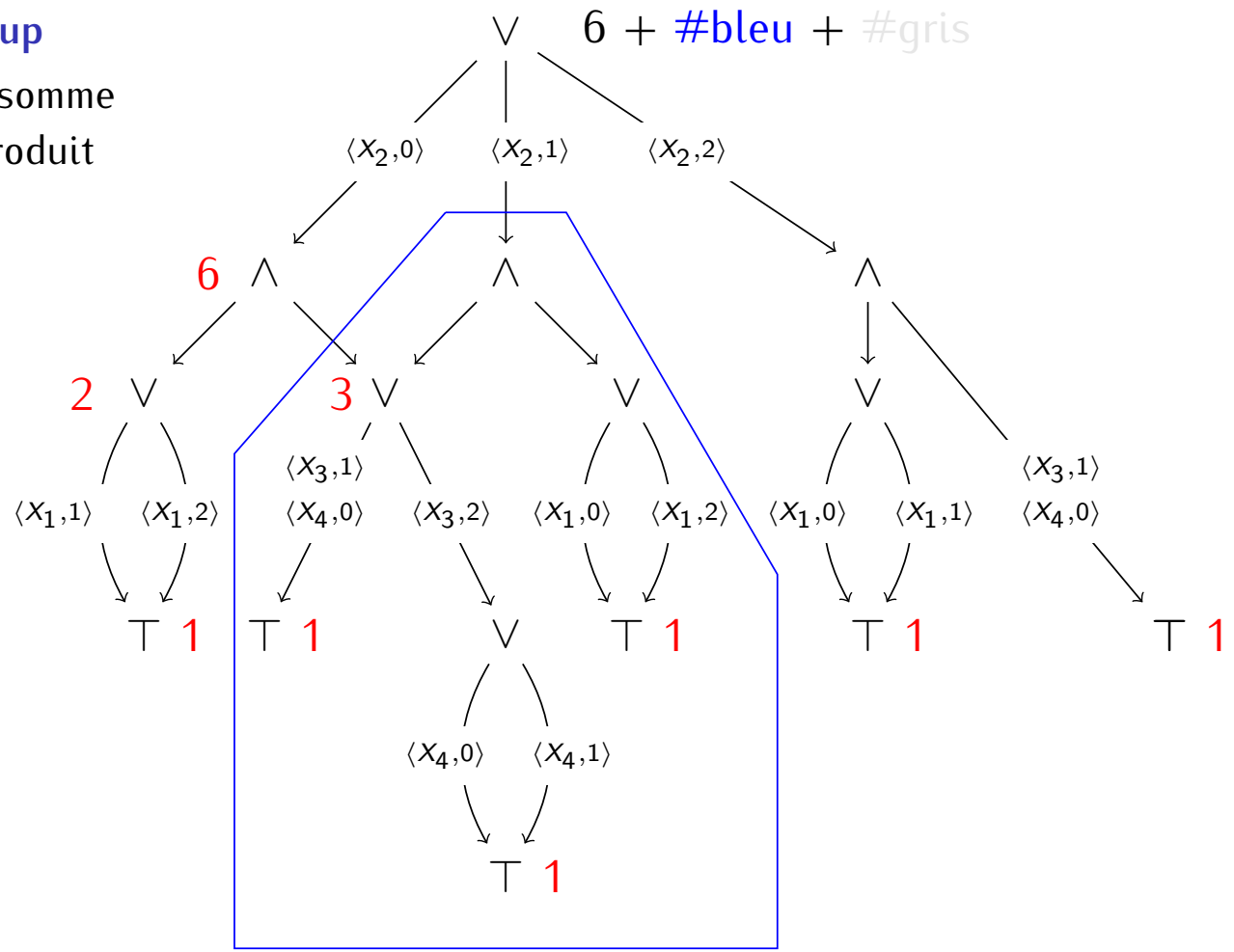
Graphes de décision décomposables multivalués : comptage de modèles

- ✧ **Approche bottom-up**
- ✧ Nœud OU ≡ une somme
- ✧ Nœud ET ≡ un produit



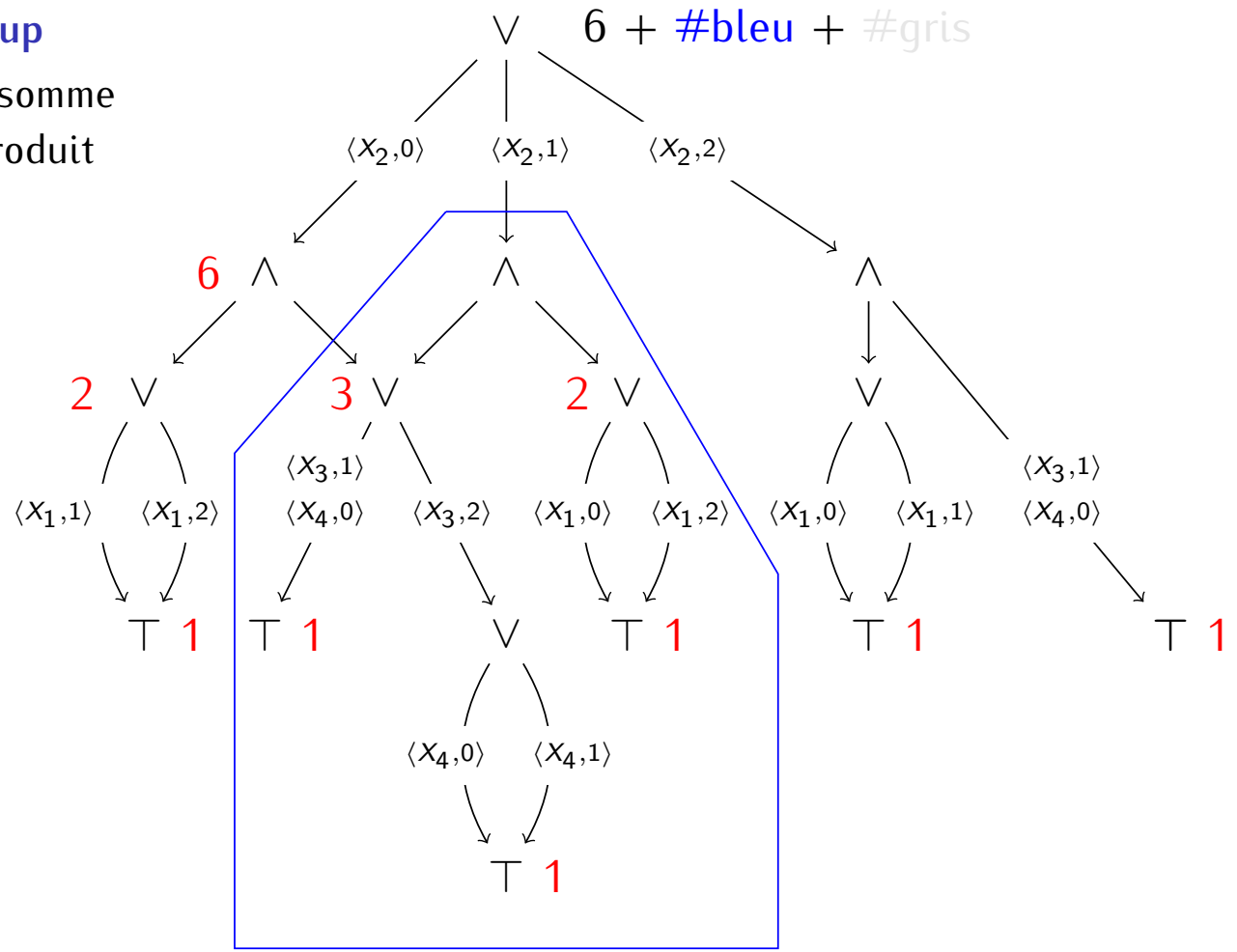
Graphes de décision décomposables multivalués : comptage de modèles

- ✧ **Approche bottom-up**
- ✧ Nœud OU ≡ une somme
- ✧ Nœud ET ≡ un produit



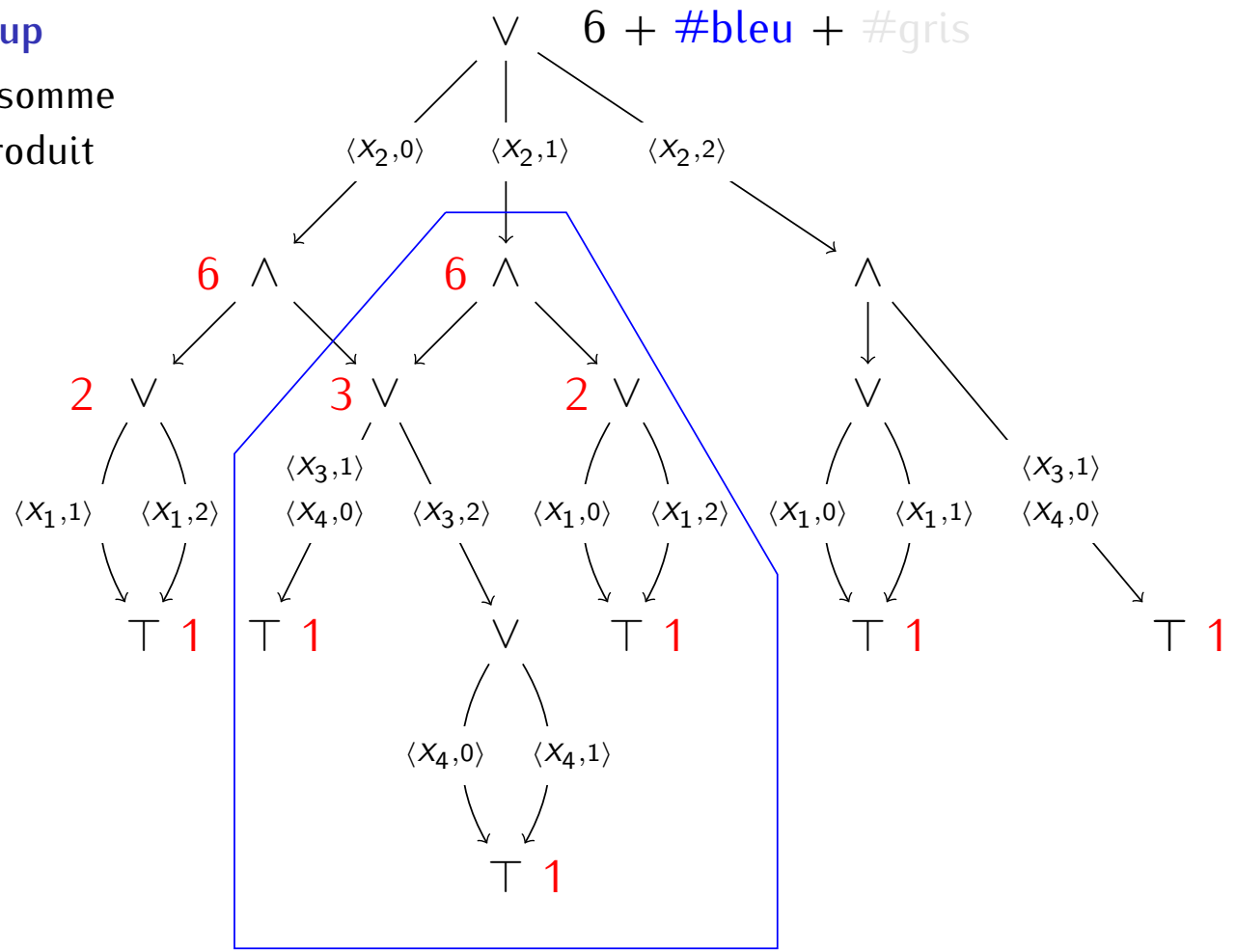
Graphes de décision décomposables multivalués : comptage de modèles

- ✧ **Approche bottom-up**
- ✧ Nœud OU ≡ une somme
- ✧ Nœud ET ≡ un produit



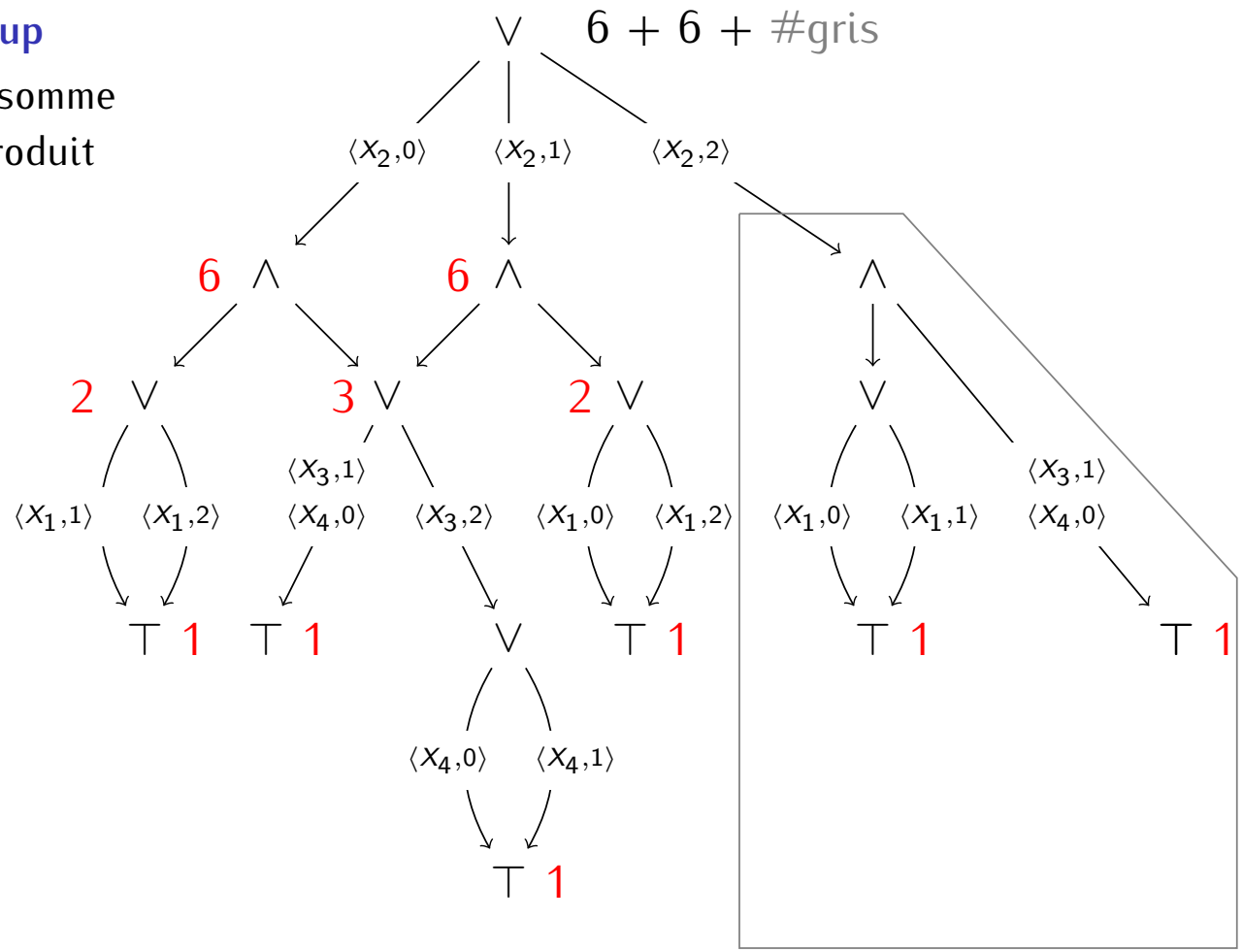
Graphes de décision décomposables multivalués : comptage de modèles

- ✧ **Approche bottom-up**
- ✧ Nœud OU ≡ une somme
- ✧ Nœud ET ≡ un produit



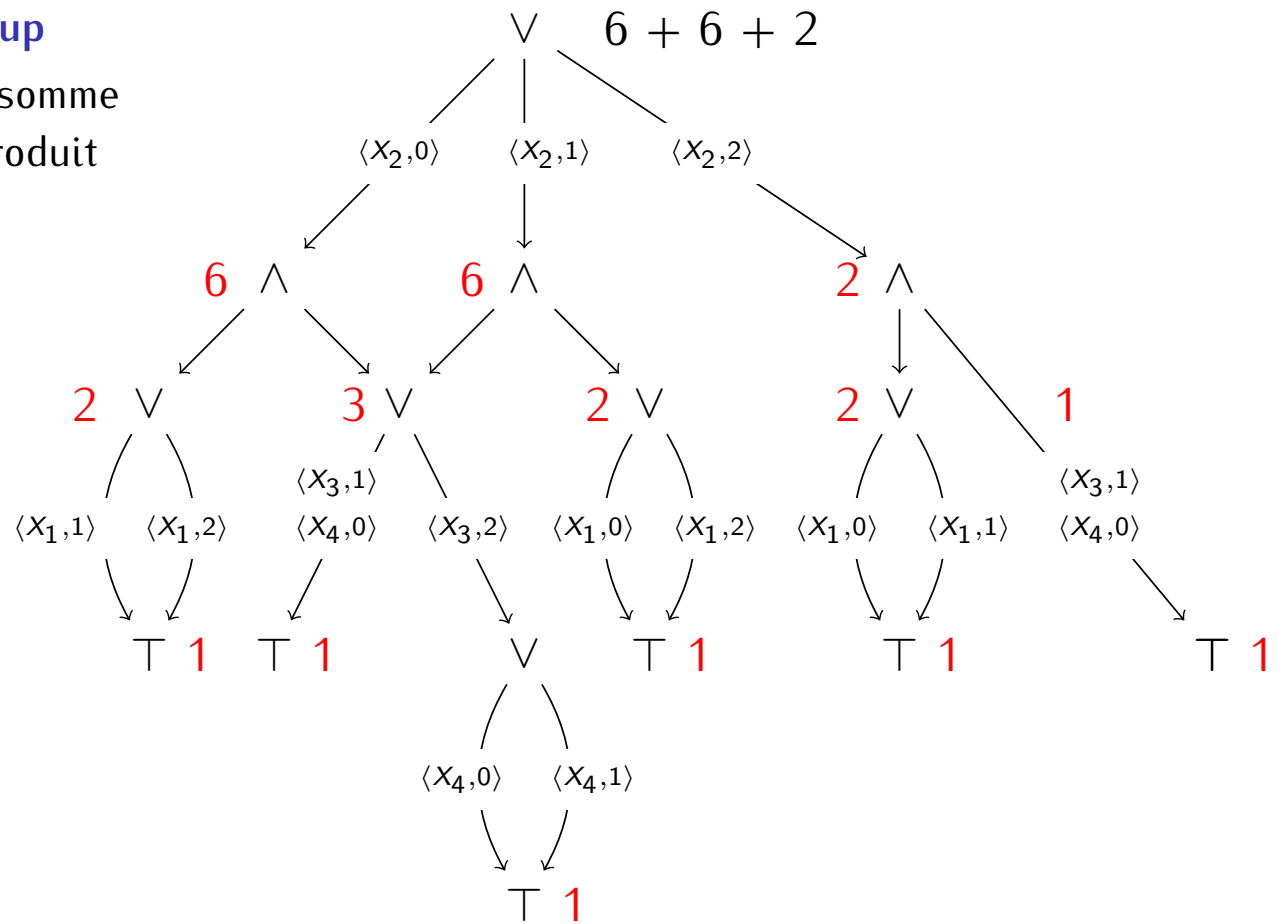
Graphes de décision décomposables multivalués : comptage de modèles

- **Approche bottom-up**
- Nœud OU ≡ une somme
- Nœud ET ≡ un produit



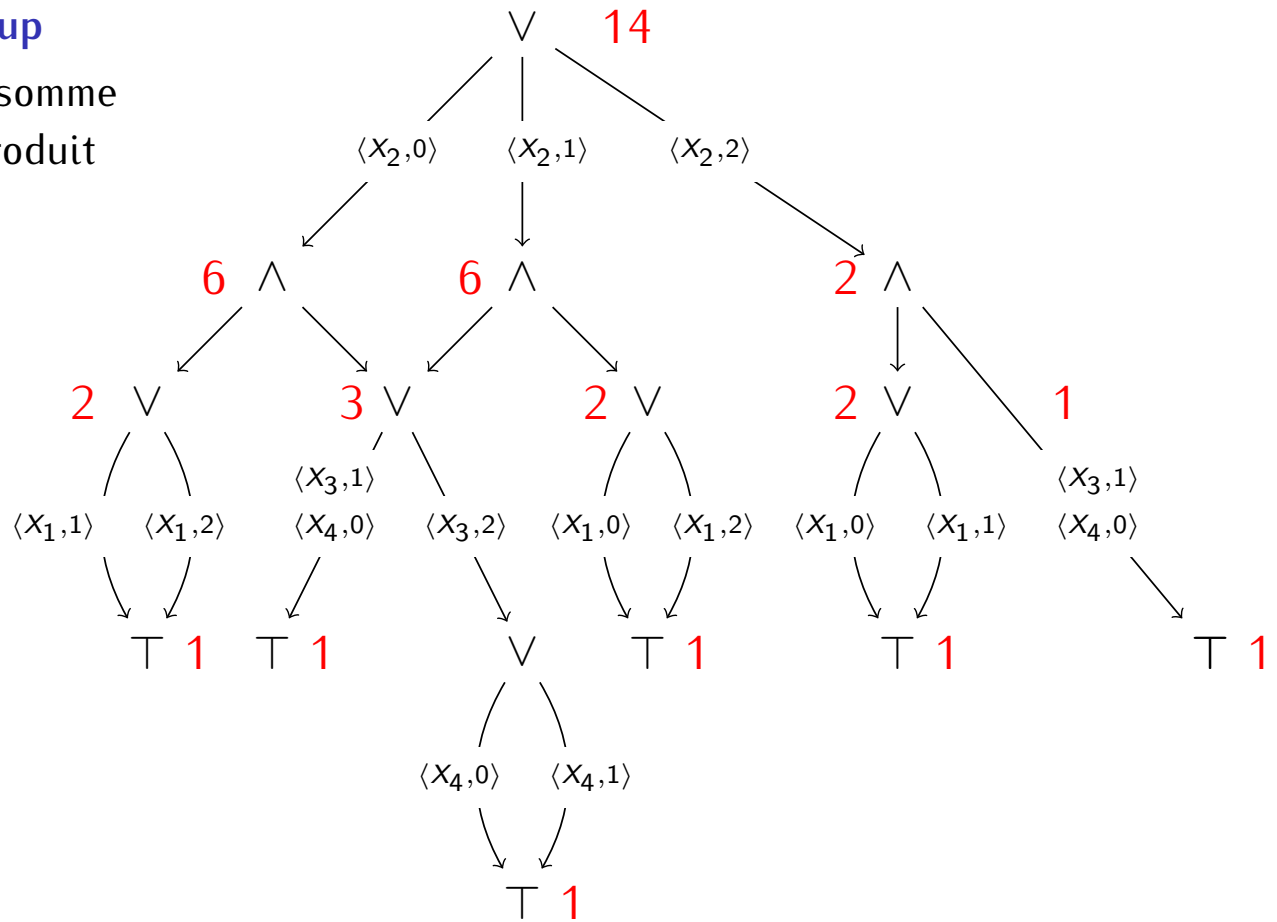
Graphes de décision décomposables multivalués : comptage de modèles

- ✧ **Approche bottom-up**
- ✧ Nœud OU ≡ une somme
- ✧ Nœud ET ≡ un produit



Graphes de décision décomposables multivalués : comptage de modèles

- Approche bottom-up
- Nœud OU ≡ une somme
- Nœud ET ≡ un produit



Protocole expérimental

- 173 CNs (définis en extension et en intension) de 15 ensembles de problèmes (incluant configuration, planification, allocation de fréquences)
- Compiler chaque réseau de contrainte en :
  - une représentation MDDG via notre compilateur, appelé cn2mddg
  - une CNF via le *sparse encoding* des domaines et un encodage mixte des contraintes
  - une représentation d-DNNF via le compilateur d-DNNF Dsharp
- Exécuté sur un Quad-core Intel XEON X5550 avec 32GB de mémoire vive
- Temps limité à 3600s pour la phase d'encodage en CNF (resp. la phase hors ligne de compilation) et un espace total de 8Go de mémoire pour stocker la formule CNF résultante (resp. la formule compilée)

Résultats expérimentaux

	CN				cn2mddg		CNF → d-DNNF			
Nom	$\#\mathcal{X}$	$\#C$	maxA	maxD	temps	taille	#pv	#pcl	temps	taille
rect-packingrpp09	2196	2353	10	36	1673.33	514754	37044	593518	375.66	16118647
ghoulomb3-4-5	2033	2051	11	26	15.17	5162	MO	MO	MO	MO
talent-concert	325	352	46	316	1277.21	404437	MO	MO	MO	MO
CostasArray10	110	338	4	19	10.39	13440	149564	841930	TO	-
CostasArray14	210	808	4	27	TO	-	988671	5568047	TO	-
photophoto2	89	133	21	11	499.93	9564220	685555	14326576	TO	-
rlfap-scen4	680	3967	2	44	3.47	52226	915553	4875002	-	MO
radiation04	781	569	9	5180	-	MO	MO	MO	MO	MO
still-life7x7	690	803	50	50	1819.87	738478	MO	MO	MO	MO
renault-mod-32	111	154	10	42	20.39	160238	222582	1755876	TO	-
renault-mod-11	111	149	10	42	16.22	41919	223718	1762294	3538.01	2399273
driverlogw-08c	408	9321	2	11	15.63	2931	9528	62825	6.42	139306

- `cn2mddg` 131 instances compilées (32 TO and 10 MO)
- `Dsharp` 83 instances compilées (24 TO and 39 MO)

Impact de chaque technique utilisée dans cn2mddg

- Nous avons réalisé une évaluation afin d'évaluer l'impact de chaque technique utilisée

	centralité	dom/wdeg
sans universelle	121	113
sans nœud ET	59	63
sans caching	118	112
tout	131	125

- Supprimer la décomposition affecte sensiblement les performances
- Lorsque la décomposition est prise en compte, la meilleure heuristique de choix de variable est la centralité
- **Chaque technique est utile !**

Conclusion

- **Définition et étude de la famille des langages des arbres de décision affine**
- Ces langages sont très attractifs lorsque **CT** est une requête clé
- EADT offre les mêmes requêtes et (au moins) les mêmes transformations que d-DNNF, sans être moins succinct que d-DNNF$_T$
- ADT offre les mêmes requêtes et transformations que OBDD$_<$, sans être moins succinct
- Conception et évaluation d'un compilateur plutôt performant

Conclusion

- Un algorithme top-down cn2mddg pour compiler des réseaux de contraintes à domaines finis en graphes de décision décomposables multivalués est étudié
- cn2mddg tire avantage de nouvelles techniques pour compiler :
  - **caching spécifique**
  - heuristique basée sur la **centralité d'intermédiarité**
  - **contraintes universelles**
- Alors que l'encodage en CNF peut tirer avantage des prouveurs SAT et être une approche compétitive pour le problème CSP, il semble être une mauvaise idée lorsque l'objectif est la **compilation** d'un réseau de contraintes

Perspectives

- Compléter les résultats manquants et obtenir des résultats supplémentaires sur l'efficacité spatiale de la famille des arbres de décisions affines
- Utiliser des techniques de pré-traitement pour améliorer l'efficacité du compilateur `EADT`
- Modifier le compilateur `EADT` pour éviter d'avoir à gérer des `ECNF`
- Étendre le compilateur `cn2mddg` à la prise en compte du comptage de modèles pondéré
- Comparer le langage `MDDG` aux autres langages cibles pour la compilation
- Considérer de nouvelles heuristiques de choix de variables encourageant la décomposition

Merci de votre attention

Compilation de connaissances pour le comptage de modèles

Samuel Thomas

14 décembre 2016

Soutenance de thèse