Compilation de connaissances

Hélène Fargier et Pierre Marquis

IRIT-CNRS
CRIL-CNRS/Université d'Artois

Exposé Journées IAF'08, Paris, jeudi 23 octobre 2008

Qu'est-ce que la «compilation de connaissances» ?

- Une famille d'approches pour pallier **la difficulté calculatoire d'un certain nombre de problèmes d'IA**
- S'appuyant sur un principe de **pré-traitement** d'informations disponibles pour améliorer certaines tâches **du point de vue calculatoire**
- Relèvent d'une problématique de **traduction** :
  - **Phase «hors-ligne» :** mettre une partie Σ de l'information disponible sous une **forme compileée** $comp(\Sigma)$
  - **Phase «en-ligne» :** exploiter la forme compilée $comp(\Sigma)$ (et le reste α des informations disponibles) pour réaliser les tâches visées

La compilation de connaissances : un thème de recherche assez récent

- La terminologie «compilation de connaissances» date de la fin des années 80 (les tâches visées concernaient le raisonnement propositionnel)
- **Le thème s'est développé** depuis lors :
  - du point de vue théorique (concepts, algorithmes, etc.)
  - du point de vue pratique (jeux d'essai, programmes, applications, etc.)

La compilation de connaissances : une vieille idée

- Pré-traiter des informations pour améliorer l'efficacité de calculs est **une vieille idée**
- Améliorer l'efficacité de calculs signifie (typiquement) diminuer le temps calcul
- Elle trouva et trouve encore à s'appliquer dans de nombreux domaines de l'informatique (même avant l'ère de l'informatique «moderne»)

Exemple : la table de logarithmes

- Une **«forme compilée»** utile pour améliorer l'efficacité de divers calculs (depuis plus de trois siècles)
- $\Sigma \subseteq [1, 10)$
- $comp(\Sigma) =$ ensemble des couples $\langle x, log_{10}(x) \rangle$ pour chaque $x \in \Sigma$
- α est une description de ce qui doit être calculé ; par exemple $\sqrt[5]{1234}$
  - $\sqrt[5]{1234} = (1.234 \times 10^3)^{\frac{1}{5}}$
  - $log_{10}(\sqrt[5]{1234}) = log_{10}((1.234 \times 10^3)^{\frac{1}{5}})$
  - $= \frac{log_{10}(1.234)+3}{5}$
  - Rechercher $log_{10}(1.234)$ dans la table

$$\langle 1.234, 0.09131516 \rangle \in comp(\Sigma)$$

  - Calculer $\frac{log_{10}(1.234)+3}{5} = \frac{0.09131516+3}{5} = 0.618263032$
  - Rechercher dans la table l'antécédent par log_{10} de la valeur obtenue

$$\langle 4.152054371, 0.618263032 \rangle \in comp(\Sigma)$$

Qu'est-ce que de la «connaissance» ?

- Doit être pris dans un sens très large (pas seulement des croyances vraies)
- Même signification que dans «représentation des connnaissances»
- Des informations et la description de **processus pour les exploiter**
- Les informations sont souvent représentées par des formules Σ, α, ... issues de **langages logiques**

$$\langle L, \vdash \rangle$$

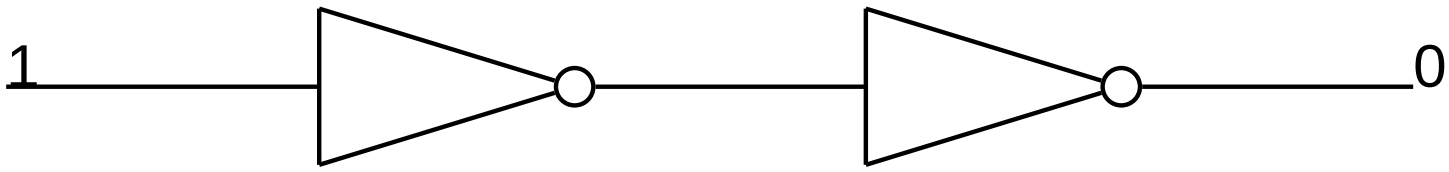
Qu’est-ce qu’ «exploiter des connaissances» ?

- Quelles sont les **tâches** à améliorer du point de vue calcul via la compilation ?
- Une problématique **dépendant du domaine considéré** en général
- Souvent des **combinaisons de requêtes et transformations élémentaires**

Requêtes et transformations élémentaires

- **Requêtes**
  - **Inférence :** est-ce que $\Sigma \vdash \alpha$?
  - **Cohérence :** existe-t-il α tq $\Sigma \nvdash \alpha$?
  - ...
- **Transformations**
  - **Conditionnement :** rendre des atomes vrai ou faux dans Σ
  - **Fermeture par connecteurs :** calculer une représentation dans L of $\alpha \oplus \beta$ à partir de $\alpha \in L$ et de $\beta \in L$
  - **Oubli :** quand elle est définie, calculer une représentation de la conséquence la plus générale selon $\vdash$ de $\Sigma \in L$,qui ne contient aucun des atomes donnés (à oublier)
  - ...

Un exemple : diagnostic fondé sur la cohérence



Un exemple : diagnostic fondé sur la cohérence

- $S = \langle SD, OBS \rangle$ regroupe la **description** SD du système considéré et des **observations** OBS
- SD décrit le comportement des composants du système et sa topologie :

$$\neg ab - inv_1 \Rightarrow (out - inv_1 \Leftrightarrow \neg in - inv_1)$$

$$\neg ab - inv_2 \Rightarrow (out - inv_2 \Leftrightarrow \neg in - inv_2)$$

$$out - inv_1 \Leftrightarrow in - inv_2$$

- OBS est une conjonction de littéraux qui décrit les entrées et sorties du système :

$$in - inv_1 \wedge \neg out - inv_2$$

- Δ est un **diagnostic** pour S ssi c'est une conjonction de ab-littéraux telle que $\Delta \wedge SD \wedge OBS$ est cohérent

$$ab - inv_1, ab - inv_2, ab - inv_1 \wedge ab - inv_2$$

Un exemple : diagnostic fondé sur la cohérence

- **Tâche :** engendrer tous les **diagnostics** de $S = \langle SD, OBS \rangle$

$$mod(\exists(PS \setminus AB).(SD \mid OBS))$$

$$= mod(\exists(PS \setminus (AB \cup Var(OBS)).SD) \mid OBS)$$

$$= mod(ab - inv_1 \vee ab - inv_2)$$

- Cette tâche peut être vue comme une combinaison de **conditionnement**, **oubli** et **énumération de modèles**
- **Autre tâche :** décider si Δ donné est un **diagnostic** de $S = \langle SD, OBS \rangle$
- Ce problème de décision DIAGNOSTIC peut être résolu par combinaison de **conditionnement** et **oubli**

Quand la compilation de connaissances est-elle utile ?

Deux conditions sont nécessaires :

- Certaines informations sont **plus sujettes au changement que d'autres**
- Illustration prototypique : le **problème de l'inférence**, i.e. un ensemble de couples $\{\langle \Sigma, \alpha \rangle\}$
  - Une «base de connaissances» Σ (la partie fixe)
  - Des requêtes α à son sujet (la partie variable)
- Autre exemple : le **problème du diagnostic** (version décision), i.e. un ensemble de couples $\{\langle SD, \langle OBS, \Delta \rangle \rangle\}$
  - La description du système *SD* (la partie fixe)
  - Des observations *OBS* et des diagnostics "possibles" Δ à son sujet (la partie variable)
- Certaines requêtes/transformations visées deviennent **«moins difficiles»**, et l'effort de calcul «hors-ligne» (les ressources consommées) reste **«raisonnable»**

Evaluer l'intérêt de la compilation

La vision théorique : le niveau problème

- Requêtes/transformations **«moins difficiles»** : supprimer des **sources de complexité** (e.g. baisser dans la hiérarchie polynomiale)
- Pré-traitement **«raisonnable»** : la **taille** de la forme compilée $comp(\Sigma)$ est **polynomiale** dans la taille de Σ (ne pas oublier que la complexité d'un algorithme est fonction de la taille de son entrée)
- Cette condition sur la taille est **cruciale**

Evaluer l'intérêt de la compilation

La vision empirique : le niveau instance

- Fait référence à une **fonction de compilation spécifique** *comp*
- Considère **un ensemble de n instances** du problème, partageant la même partie fixe Σ
- Détermine les ressources de calcul nécessaires pour résoudre les n instances en utilisant l'approche par compilation
- Détermine les ressources de calcul nécessaires pour résoudre les n instances en utilisant une approche «directe», sans compilation
- **Compare les ressources** consommées par les deux approches

Calcule des statistiques résumant les comparaisons effectuées pour plusieurs parties fixes Σ

Vision théorique vs. vision empirique

- Deux **approches complémentaires** ayant chacune ses inconvénients
- Le niveau problème : une application correspond à une / un sous-ensemble d'instances **spécifiques** du problème (pas forcément les pires)
- La compilation peut se révéler utile pour certaines instances d'un problème, même si celui-ci est «non compilable»
- Le niveau instance : est spécifique à une fonction de compilation *comp* particulière et **demande plus d'informations** (ensemble d'instances et algorithme «direct» de référence)
- Un consensus sur l'ensemble d'instances et l'algorithme de référence à utiliser peut être difficile à obtenir

Travaux passés

- Développement et évaluation de fonctions de compilation pour le raisonnement propositionnel (en particulier, la déduction clausale)
- **Développement d'un cadre formel pour la notion de «compilabilité»** et étude de la compilabilité de diverses tâches (raisonnement, diagnostic, planification, etc.)
- **Construction d'une carte pour la compilation de formules propositionnelles**
- Exploitation des techniques de compilation propositionnelles à d'autres types d'inférence
- **Etude de langages cibles pour la compilation dans d'autres cadres formels**
- **Applications au diagnostic, à la configuration, à la planification, etc.**
- ...

Qu'est-ce que la «compilabilité» ?

- **Intuition :** un problème (de décision) est **compilable dans une classe de complexité** C s'il est dans C lorsque la partie fixe Σ de toute instance a été pré-traitée, i.e., traduite «hors-ligne» en une structure de données (sa forme compilée) **de taille polynomiale** en $|\Sigma|$
- Plusieurs **classes de compilabilité** organisées en hiérarchies (structurées comme PH) ont été introduites
- Elles permettent de classer un problème en **compilable dans** C, ou **non compilable dans** C (souvent sous les hypothèses habituelles de la théorie de la complexité)

Problèmes de décision = langages L de couples

- $\langle \Sigma, \alpha \rangle \in L$
- Σ : la **partie fixe**
- α : la **partie variable**
- **Exemples :**

DÉDUCTION CLAUSALE = $\{\langle \Sigma, \alpha \rangle \mid \Sigma$ une formule NNF et α une clause telle que $\Sigma \models \alpha\}$

DIAGNOSTIC = $\{\langle SD, \langle OBS, \Delta \rangle \rangle \mid SD$ une formule NNF, OBS une conjonction de littéraux et Δ une conjonction d'*ab*-lttéraux tels que Δ est un diagnostic de $\langle SD, OBD \rangle\}$

compC

- C = une classe de complexité fermée par réduction polynomiale et admettant des problèmes complets pour ces réductions
- Un langage de couples L **appartient à** compC ssi il existe une fonction **polysize** *comp* et un langage de couples $L' \in$ C tels que pour tout couple $\langle \Sigma, \alpha \rangle$, $\langle \Sigma, \alpha \rangle \in L$ ssi $\langle comp(\Sigma), \alpha \rangle \in L'$
- Pour toute classe de complexité admissible C, on a l'inclusion C $\subseteq$ compC

compC

- Prouver l'appartenance à compC : suivre la définition !
- Prouver la non-appartenance à compC : un problème souvent plus difficile
- Les classes C/poly sont souvent utiles à cet effet

Machines de Turing à avis

- Une **machine de Turing à avis** est une machine de Turing équipée d'un oracle spécial A, qui peut être n'importe quelle fonction (éventuellement non récursive !)
- Pour une entrée s, un second ruban de la machine est automatiquement chargé avec $A(|s|)$ et le calcul de la machine procède normalement en s'appuyant sur les deux entrées, s et $A(|s|)$

C/poly

- Une machine de Turing à avis utilise un **avis polynomial** si son oracle A est **polysize**
- C/poly est la classe de tous les langages L pour lesquels il existe une fonction polysize A telle que $\{\langle A(|s|), s\rangle \mid s \in L\}$ appartient à C

P/poly vs. PH

- **Si** NP $\subseteq$ P/poly **alors** $\Pi_2^p = \Sigma_2^p$ **(donc** PH **s'effondre au deuxième niveau)**
- Si NP $\subseteq$ coNP/poly alors $\Pi_3^p = \Sigma_3^p$ (donc PH s'effondre au troisième niveau)

DÉDUCTION CLAUSALE $\notin$ compP

- Soit n un entier naturel
- Soit Σ_n^{max} la formule CNF

$$\bigwedge_{\gamma_i \in 3-C_n} \neg holds_i \vee \gamma_i$$

- $3 - C_n$ est l'ensemble de toutes les clauses contenant 3 littéraux, qui peuvent être engendrée à partir de $\{x_1, \ldots, x_n\}$ et les atomes $holds_i$ sont des nouveaux atomes, ne figurant pas parmi $\{x_1, \ldots, x_n\}$
- $|\Sigma_n^{max}| \in \mathcal{O}(n^3)$

DÉDUCTION CLAUSALE $\notin$ compP

- Toute formule 3-CNF α_n construite sur les atomes de $\{x_1, \ldots, x_n\}$ est en bijection avec le sous-ensemble S_{α_n} des atomes $holds_i$ tel que γ_i est une clause de α_n ssi $holds_i \in S_{\alpha_n}$
- α_n est incohérente ssi

$$\Sigma_n^{max} \models \gamma_{\alpha_n} = \bigvee_{holds_i \in S_{\alpha_n}} \neg holds_i$$

DÉDUCTION CLAUSALE $\notin$ compP

- Supposons qu'il existe une fonction (de compilation) polysize *comp* telle que décider si $comp(\Sigma) \models \gamma \in$ P
- Alors 3-SAT $\in$ P/poly :
  - Soit α une formule 3-CNF
  - Si $|Var(\alpha)| = n$, la machine charge d'abord son ruban oracle avec
    $$A(n) = comp(\Sigma_n^{max})$$
  - Elle détermine ensuite en temps polynomial (déterministe) si $comp(\Sigma_n^{max}) \models \gamma_\alpha$
- Comme 3-SAT est complet pour NP, cela aurait pour conséquence NP $\subseteq$ P/poly

La carte pour la compilation de formules propositionnelles

Une **évaluation multi-critère** de langages cibles pour la compilation

- **Requêtes :** opérations pour extraire de l'information d'une forme compilée sans la changer
- **Transformations :** opérations modifiant les formes compilées
- **Efficacité spatiale :** la capacité d'un langage à représenter de l'information en utilisant (relativement) «peu d'espace mémoire»

NNF et ses sous-ensembles

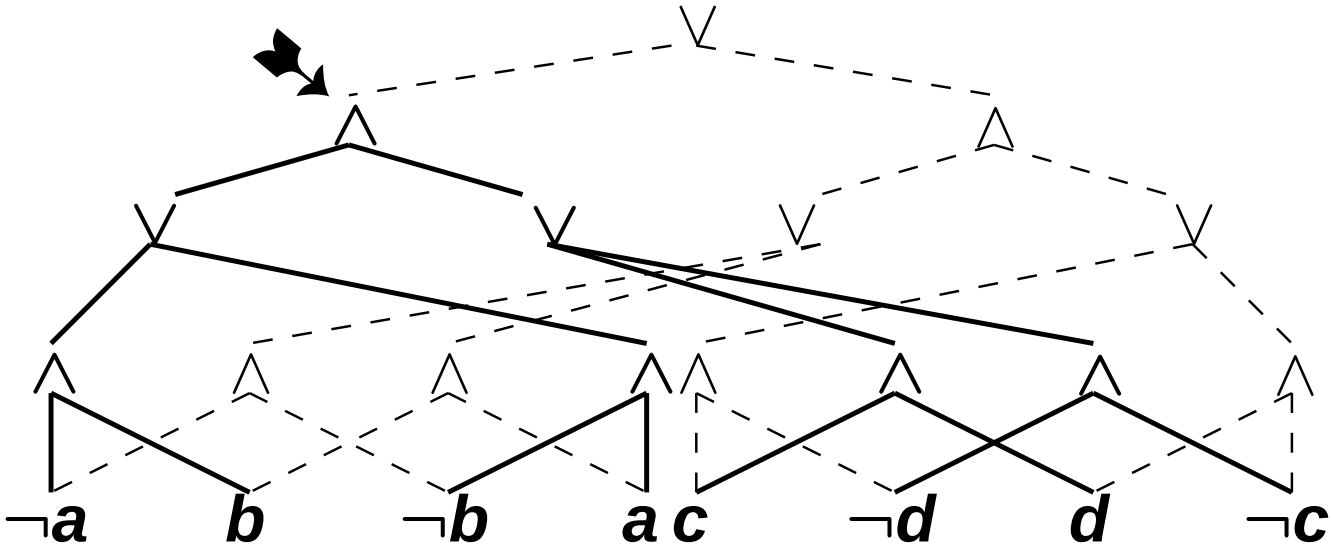


FIG.: Une formule NNF.

Requêtes

Problèmes de décision ou de recherche / propriétés des fragments

- **CO** (cohérence)
- **CE** (déduction clausale)
- **VA** (validité)
- **EQ** (équivalence)
- **SE** (déduction)
- **IM** (impliquants)
- **CT** (comptage des modèles)
- **ME** (énumération des modèles)
- ...

Transformations

Problèmes de recherche / propriétés des fragments

- **CD** (conditionnement)
- $\wedge$ **C** ($\wedge$**BC**) (fermeture selon $\wedge$)
- $\vee$**C** ($\vee$**BC**) (fermeture selon $\vee$)
- $\neg$**C** (fermeture selon $\neg$)
- **FO** (**SFO**) (oubli)
- ...

CO

- Soit **L** un sous-ensemble de NNF
- **L** satisfait **CO** ssi il existe un algorithme en temps polynomial associant toute formule Σ de **L**
  - à 1 si Σ est cohérent,
  - et à 0 sinon

CD

- Soit **L** un sous-ensemble de NNF
- **L** satisfait **CD** ssi il existe un algorithme en temps polynomial associant toute formule Σ de **L** et tout terme cohérent γ à une formule de **L** qui est équivalente à $\Sigma \mid \gamma$
- $\Sigma \mid \gamma$ est la formule obtenue en remplaçant dans Σ chaque occurrence d'un atome $x \in \mathit{Var}(\gamma)$ par *vrai* si x est un littéral positif du terme γ et par *faux* si $\neg x$ est un littéral négatif du terme γ

Requêtes et transformations ne sont pas (toujours) indépendantes

Soit **L** un sous-ensemble de NNF

- Si **L** satisfait **SE**, alors il satisfait **CE** et **EQ**
- Si **L** satisfait **ME**, alors il satisfait **CO**
- Si **L** satisfait **CO** and **CD**, alors il satisfait **CE**
- Si **L** satisfait **CT**, alors il satisfait **CO** and **VA**
- Si **L** satisfait **CO**, $\wedge$**C** et $\neg$**C**, alors il satisfait **SE**
- Si **L** satisfait **VA**, $\vee$**C** et $\neg$**C**, alors il satisfait **SE**
- Si **L** contient L_{PS} et satisfait $\wedge$**C** et $\vee$**BC**, alors il ne satisfait pas **CO** sauf si P = NP
- Si **L** satisfait **FO**, alors il satisfait **CO**
- ...

Efficacité spatiale

L'efficacité spatiale (ou compacité) capture **la capacité d'un langage à représenter de l'information en utilisant (relativement) «peu d'espace mémoire»**

- $\mathbf{L}_1$ est **au moins aussi succinct que** $\mathbf{L}_2$, noté $\mathbf{L}_1 \leq_s \mathbf{L}_2$, ssi il existe un polynôme p tel que pour toute formule $\alpha \in \mathbf{L}_2$, il existe une formule équivalente $\beta \in \mathbf{L}_1$ telle que $|\beta| \leq p(|\alpha|)$
- Un **pré-ordre** $\leq_s$ sur les sous-ensembles de NNF

Fragments, requêtes et transformations

- DNNF satisfait **CO**, **CE**, **ME**, **CD**, **FO**, $\vee$**C**
- d-DNNF satisfait **CO**, **VA**, **CE**, **IM**, **CT**, **ME**, **CD**
- $\text{OBDD}_<$ satisfait **CO**, **VA**, **CE**, **IM**, **EQ**, **CT**, **ME**, **CD**, **SFO**, $\wedge$**BC**, $\vee$**BC**, $\neg$**C**
- DNF satisfait **CO**, **CE**, **ME**, **CD**, **FO**, $\wedge$**BC**, $\vee$**C**
- PI satisfait **CO**, **VA**, **CE**, **IM**, **EQ**, **SE**, **ME**, **CD**, **FO**, $\vee$**BC**
- IP satisfait **CO**, **VA**, **CE**, **IM**, **EQ**, **SE**, **ME**, **CD**, $\wedge$**BC**
- DNNF **ne satisfait aucune propriété parmi VA**, **IM**, **EQ**, **SE**, **CT**, $\wedge$**BC**, $\neg$**C** sauf si P = NP
- ...

L'efficacité spatiale des fragments propositionnels

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

Compacité vs. non-compacité

Plusieurs types de preuve :

- DNNF $\leq_s$ DNF : facile car DNNF $\supseteq$ DNF
- DNF $\not\leq_s$ DNNF : **arguments combinatoires**

$$\bigwedge_{i=0}^{n-1} (\neg x_{2i} \vee x_{2i+1}) \in \text{DNNF}$$

- DNNF $\not\leq_s$ CNF : **exploite les résultats de non-compilabilité**
  - DNNF satisfait **CE**
  - DÉDUCTION CLAUSALE à partir de formules CNF Σ n'est pas dans compP sauf si PH s'effondre

Tirer parti de la carte pour la compilation

- **Identifier** les requêtes et transformations requises
- **Sélectionner** les fragments qui les satisfont (en tant que propriétés)
- **Choisir** un des plus succincts parmi eux

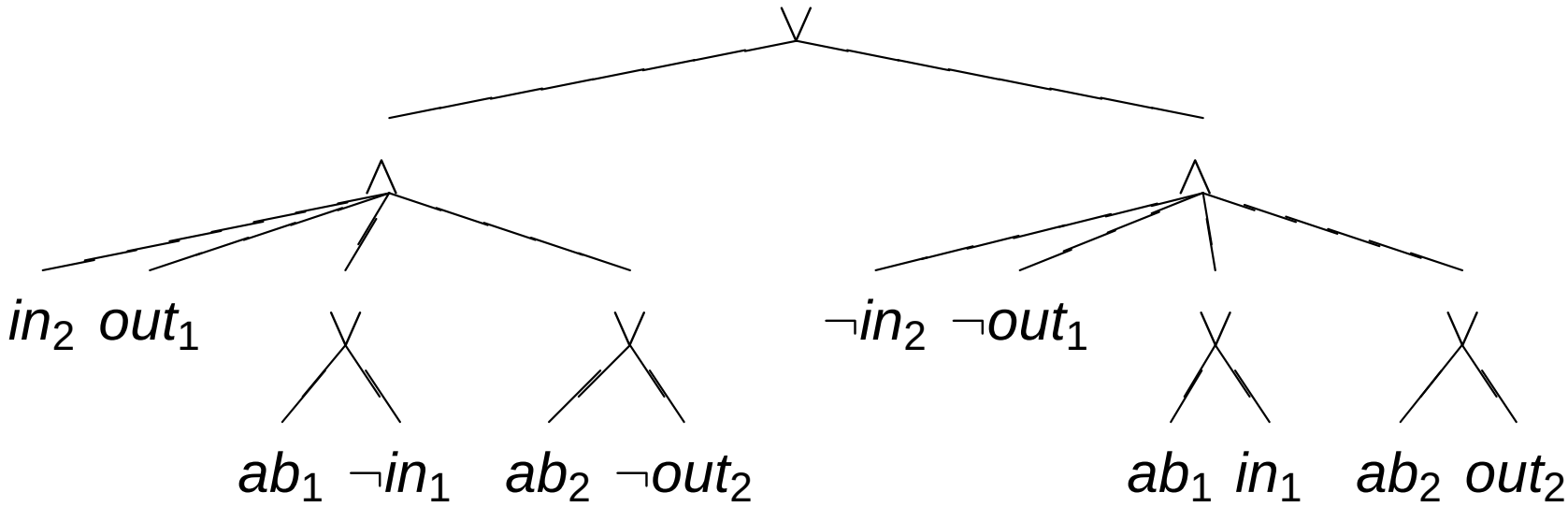
Un exemple : diagnostic fondé sur la cohérence

DIAGNOSTIC n'est pas dans compP sauf si PH s'effondre

- **FO**, **CD** sont requises
- `DNNF`, `DNF`, `PI` les satisfont
- `DNNF` et `PI` sont les plus succincts des trois
- Explique le succès des fragments `DNNF` et `PI` pour le diagnostic fondé sur la cohérence ?

Diagnostic fondé sur la cohérence – en détails

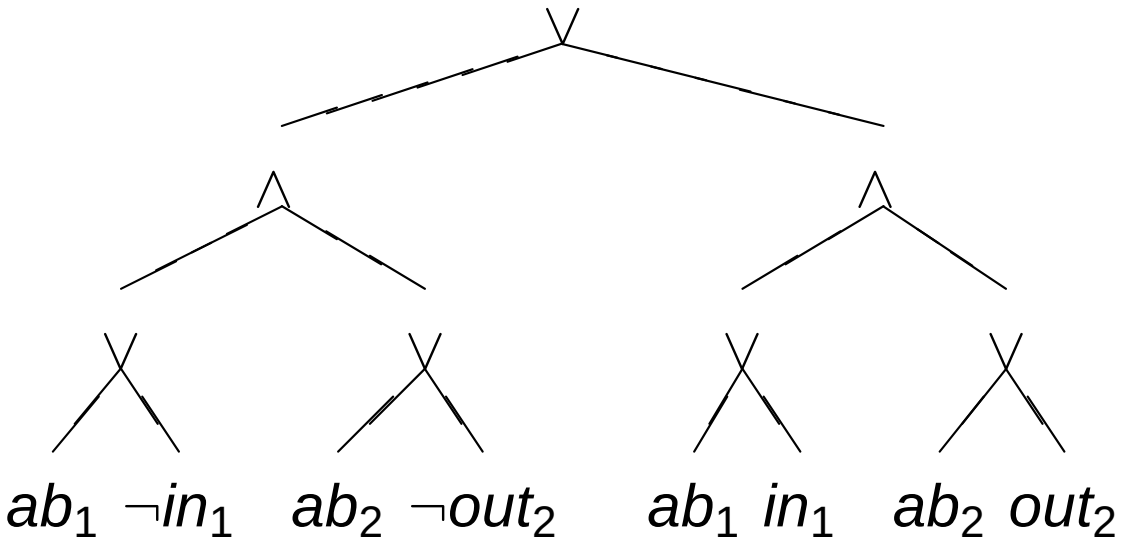
1- Représenter le système par une DNNF (compiler)



2. Oublier les variables intermédaires (projeter sur les variables d'intérêt)

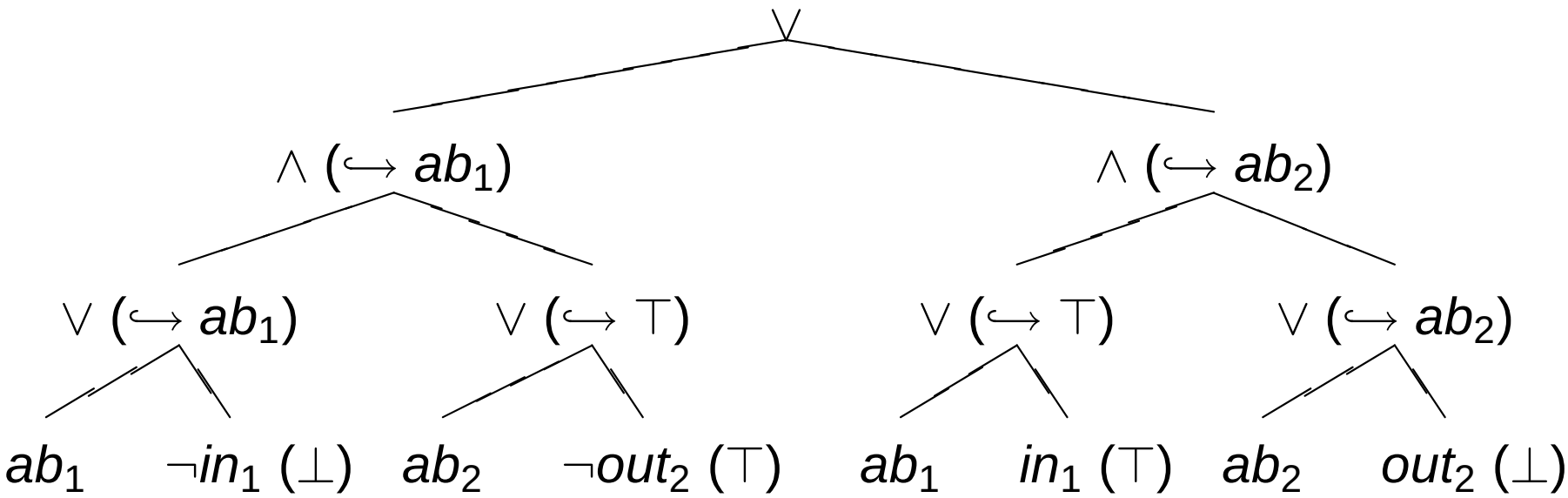
Diagnostic fondé sur la cohérence – en détails

1+2- Représenter le système par une DNNF et oublier les variables intermédaires



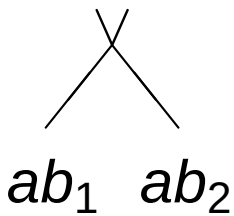
Diagnostic fondé sur la cohérence – en détails

3. En ligne, conditionner la `DNNF` par l'observation (e.g. $in_1, \neg out_2$) puis la réduire :



Diagnostic fondé sur la cohérence – en détails

3. En ligne, conditionner la `DNNF` par l'observation (eg $in_1, \neg out_2$) puis la réduire :



4. Trouver un (ou plusieurs) diagnostic(s) = énumérer les modèles : (ab_1, ab_2), $(ab_1, \neg b_2)$, $(\neg ab_1, b_2)$

Diagnostic minimisant le nombre de fautes

- Représenter le système par une formule logique
- Pondérer les littéraux :
  - sur les hypothèses : $ab_1?1:0$, $ab_2?1:0$
  - sur les littéraux observables :
    ($in_1?0:+\infty$, $out_2?+\infty:0$)
- Coût d'un modèle de la formule = somme des coûts hérités des littéraux
  - $ab_1, \neg ab_2, in_1, \neg out_2 : 1$
  - $\neg ab_1, \neg ab_2, in_1, out_2 : +\infty$
  - $ab_1, ab_2, in_1, \neg out_2 : 2$

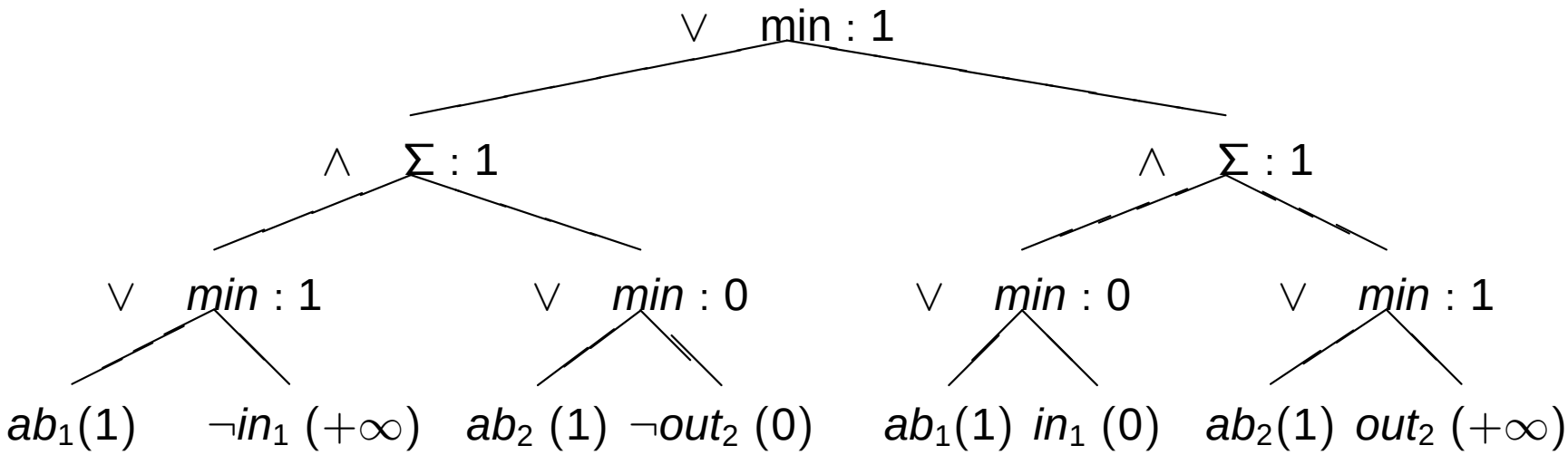
  Diagnostic minimal = modèle de coût min

Recherche de diagnostic optimal dans une DNNF

- Associer des poids aux feuilles (0, 1, $+\infty$) .
- Coût de $\phi_1 \wedge \phi_2$ = coût(ϕ_1) + coût(ϕ_2) car pas de variables en commun
- Coût de $\phi_1 \vee \phi_2 = min$(coût(ϕ_1), coût(ϕ_2)) (je peux choisir par quel côté satisfaire la disjonction)

Recherche de diagnostic minimal dans une DNNF

Calcul du coût min, en ligne, par propagation dans le graphe de la DNNF.



Enumération des modèles optimaux par un parcours retour de la racine vers les feuilles

Recherche de diagnostic optimal dans un OBDD

- On peut voir un OBDD comme un automate déterministe, noeuds étiquetés par des variables, arcs par des valeurs de ces variables.
- Modèle = chemin de la source vers le puits
- Étiqueter les arcs par des poids (0, 1, $+\infty$)
- Diagnostic optimal = chemin de la source vers racine de coût minimal

Compilation (hors ligne) puis problème de plus court chemin (en ligne)

Utilisation du principe automate + valuations : configuration interactive

- Ensemble des produits possibles (catalogue) : formule logique, CSP
- Actions utilisateur : choix de valeur pour des variables, annulation de choix
- Système : éliminer des choix possibles les valeurs incompatibles avec les choix courants

Utilisation du principe automate + valuations : configuration interactive

- Ensemble des produits possibles compilé hors ligne, e.g. à partir d'un CSP .

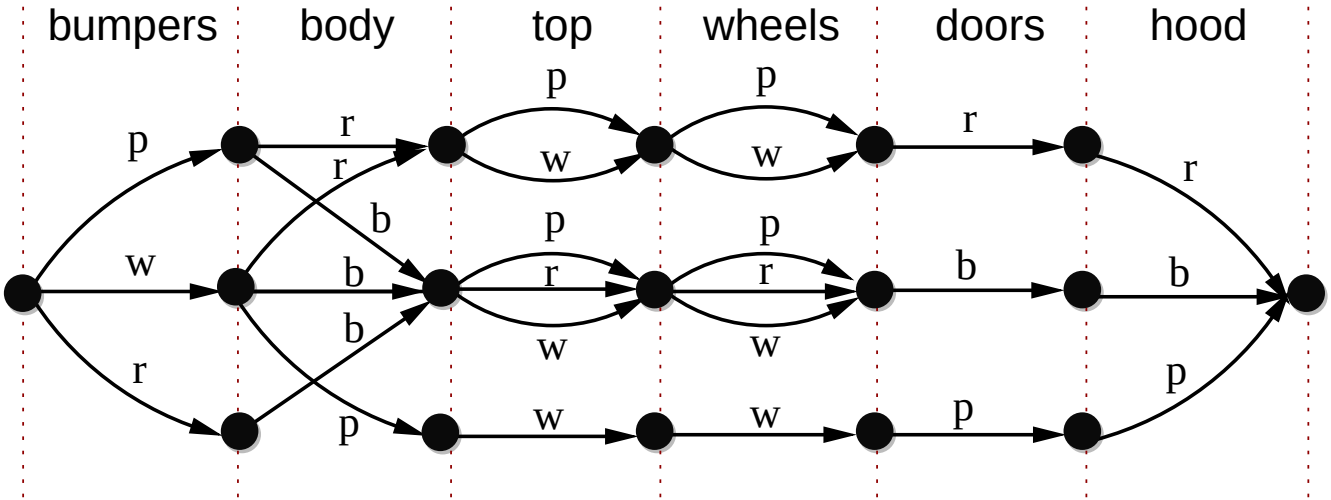


FIG.: Un petit problème de configuration de voiture

Utilisation du principe automate + valuations : configuration interactive

- Choix d'une valeur : poser un poids 1 sur les valeurs alternatives
- Coût d'un chemin : nombre de choix utilisateur violés
- Éliminer les valeurs n'étant supportées par aucun chemin de coût nul.

Remarque : ici aussi, on pourrait travailler sur des DNNF.....

D’autres exemples

- En planification :
  - *En planification non déterministe* : compiler la fonction de transition par un OBDD pour opérer une récursion arrière
  - *En planification conformante (robuste)* : représenter le problème par une DNNF déterministe pour voir évoluer l’ensemble des états couvrables
  - *En planification probabiliste* : représenter par des *ADD* (un OBDD dont les feuilles sont des entiers) la table de probabilité $< tat_i, action, tat_{i+1} >$ et des fonctions d’utilité.
- En raisonnement : sur des bases stratifiées, sur des bases possibilistes, pour faire de la révision, . . .

Prenons un peu de recul

- Qu'est ce qui fait que ca marche ?
  $\Rightarrow$ voir la carte de compilation

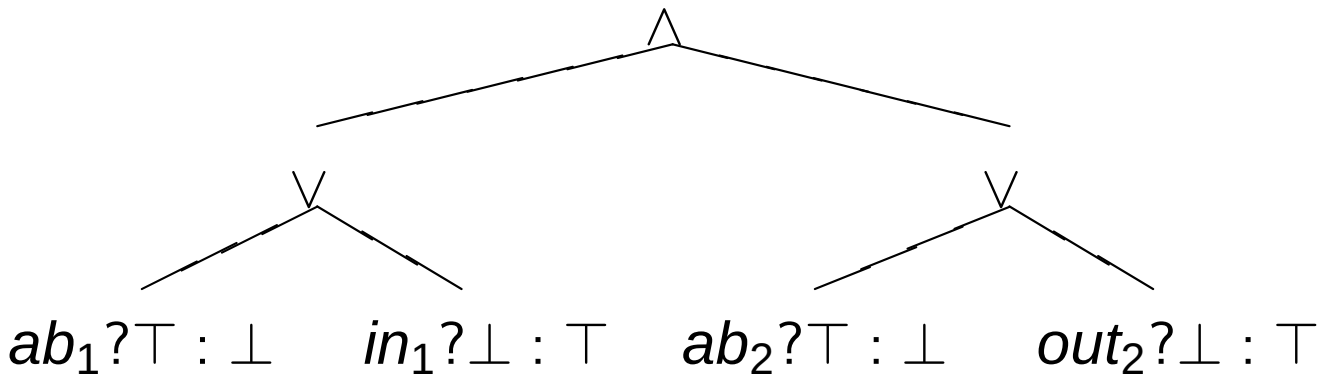
- Oui, mais dans la carte de compilation, il n'y a pas de valuations ? ? ?
  $\Rightarrow$ faire une carte de compilation de formes valuées

Formes Normales Negatives *Valuées*

- E un ensemble ordonné de valuations (e.g. $[0, 1]$) ;
- $X = \{x_1, \ldots, x_n\}$ un ensemble de variables à domaines discrets.
  D_Y l'ensemble des affectations possibles de $Y \subseteq X$.
- Considérer des "fonctions locales" $f : D_Y \mapsto E$.
  - $E = \{\top, \bot\}$: littéraux, constantes, contraintes, fonctions booléennes
  - $E = \mathbb{N}$: contraintes ou formules pondérées
  - $E = [0, 1]$: probabilités, unitaires ou jointes, fonctions d'utilité, contraintes floues, ...
- OP un ensemble d'opérateurs sur les valuations de E :
  - $+$, $\times$, max, min ...
  - opérateurs associatifs, commutatifs, monotones et possédant chacun un élément neutre.

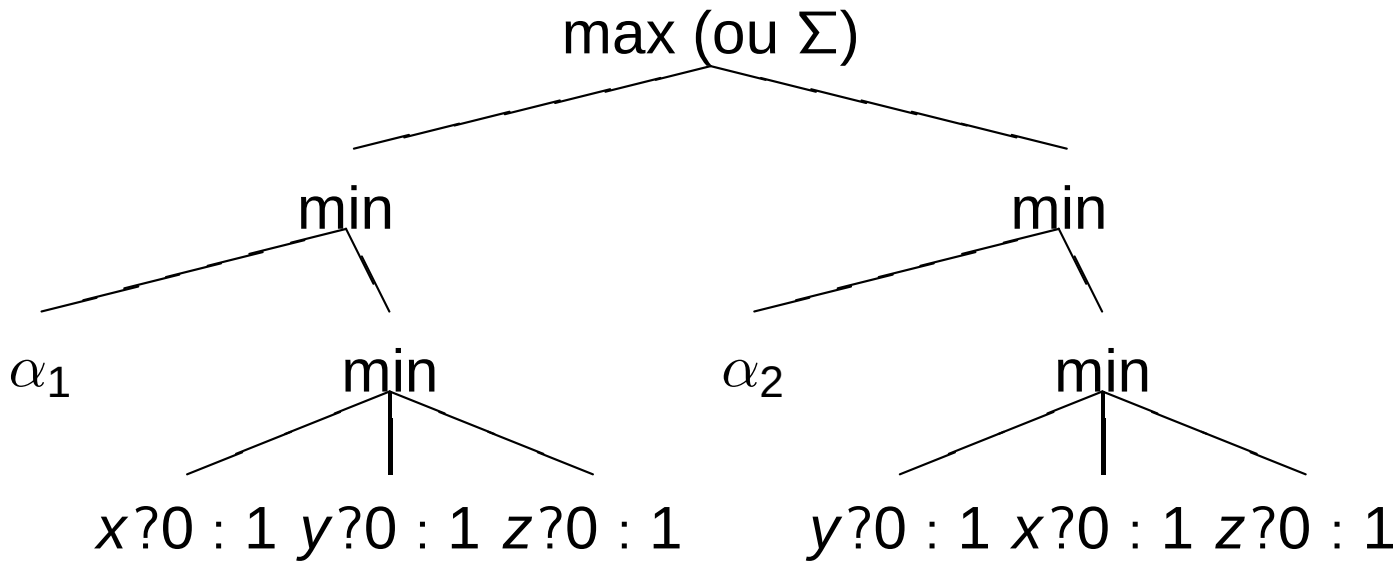
Formes Normales Negatives *Valuées*

- VNNF = un graphe, dont les noeuds internes sont étiquetés par des opérateurs et les feuilles par des fonctions locales
  - NNF de la log. propositionnelle : les feuilles sont les littéraux, les noeuds internes sont étiquetés par $\vee$ et $\wedge$



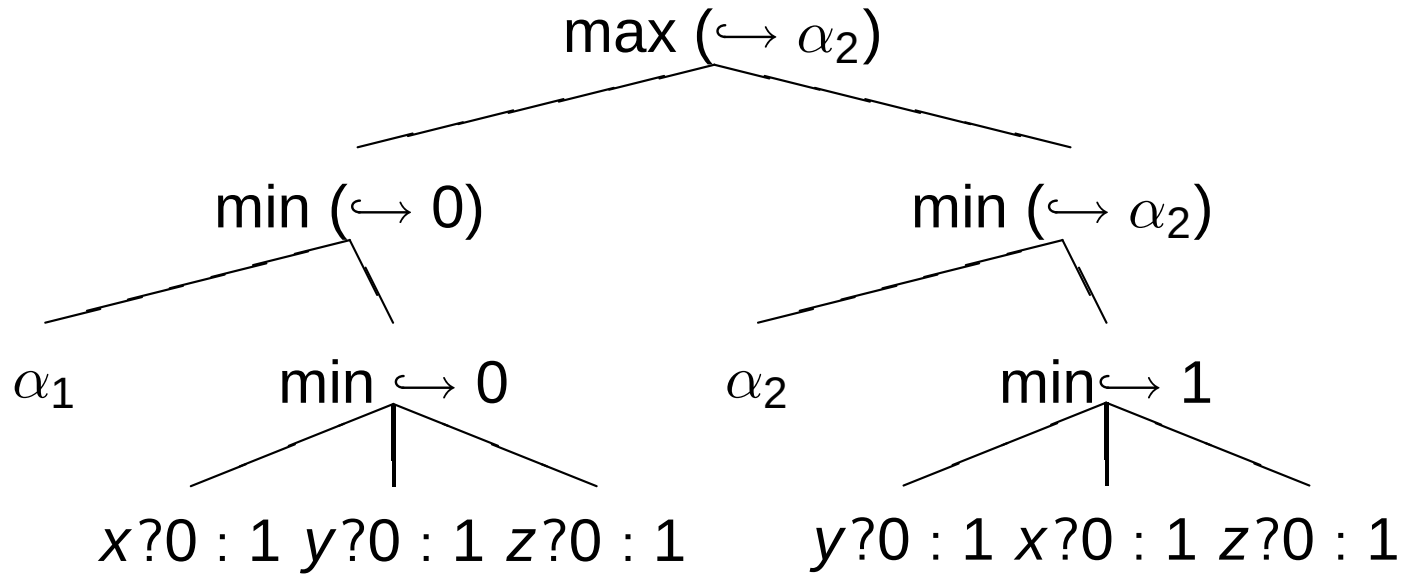
Formes Normales Negatives *Valuées*

- VNNF = noeuds internes étiquetés par des opérateurs, feuilles par des fonctions locales
  - CNF de la log. possibiliste : feuilles sont des "littéraux" $x?0:1$ ou $x?1:0$, noeuds étiquetés par min et max
  - CNF de la log. pondérée : feuilles sont des "littéraux" $x?0:1$ ou $x?1:0$, noeuds étiquetés par min et Σ



Evaluation d'une VNNF

- Pour une affectation de X, chaque feuille rend une valeur, à combiner selon le schéma indiqué par la VNNF.



Evaluation pour $(x \leftarrow \mathit{vrai}, y \leftarrow \mathit{vrai}, z \leftarrow \mathit{vrai})$

Formes Normales Négatives Valuées

- Une VNNF représente de façon compacte une fonction de D^n dans E...
- Requêtes/transformations : conditionner, énumérer les modèles minimaux, éliminer une variable *selon un opérateur*
  - $\exists x \phi$ est vu comme la VNNF $\max_{v \in D_x} \phi_{x \leftarrow v}$
  - Compter les modèles : calculer la VNNF $\Sigma_{v \in D_x} \phi_{x \leftarrow v}$
- Oui, mais toute VNNF ne permet pas une réponse efficace à toute requête/transformation

Propriétés remarquables des «bons» langages cibles pour la compilation

- *La décomposabilité* : un noeud N étiqueté $\oplus$ est $\otimes$-décomposable ssi
  - Soit $\oplus = \otimes$
  - Soit $\oplus$ est distributive sur $\otimes$ et les enfants de N ne partagent aucune variable

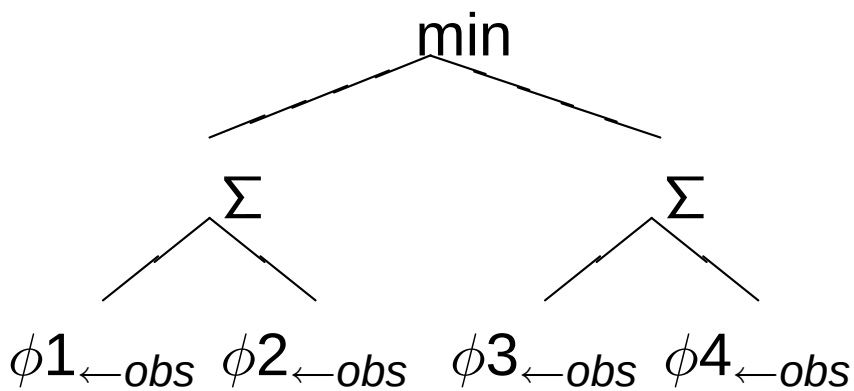
  DNNF «classiques» : formules $\vee$-décomposables (que des $\vee$ et des $\wedge$ décomposables)

- La $\otimes$-décomposabilité assure que $(\phi \oplus \psi)_{y \leftarrow vrai} \otimes (\phi \oplus \psi)_{y \leftarrow faux}$ est équivalent à $(\phi_{y \leftarrow vrai} \otimes \phi_{y \leftarrow faux}) \oplus (\psi_{y \leftarrow vrai} \otimes \psi_{y \leftarrow faux})$.

Propriétés remarquables des «bons» langages cibles pour la compilation

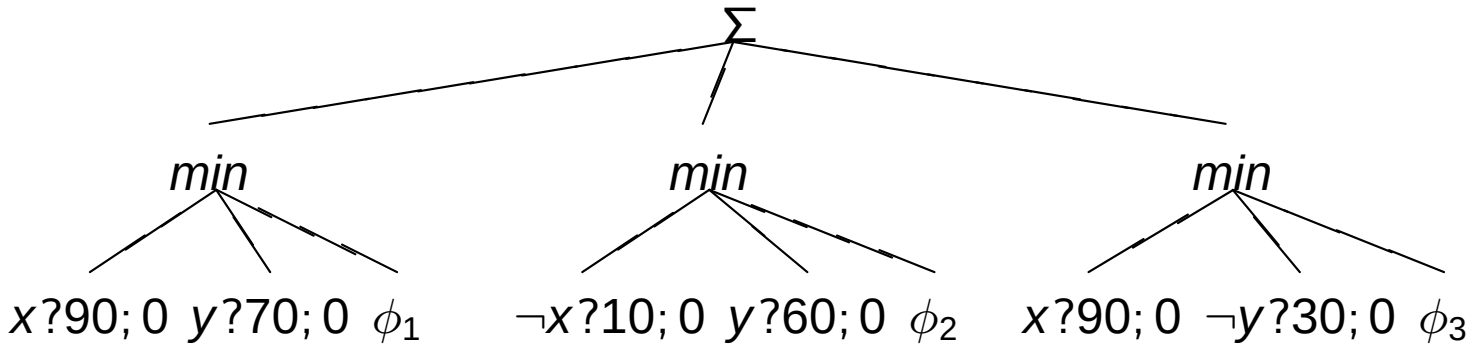
Un noeud *N* étiqueté $\oplus$ est $\otimes$-décomposable ssi soit $\oplus = \otimes$, soit $\oplus$ est distributive sur $\otimes$ et les enfants du *N* ne partagent aucune variable

En diagnostic, représentation de la fonction de vraissemblance par un graphe min-décomposable :
coût min de $\phi_1 + \phi_2$ = coût min de ϕ_1 + coût min de ϕ_2



Propriétés remarquables des «bons» langages cibles pour la compilation

- *Le déterminisme* : un noeud étiqueté $\oplus$ est *déterministe* ssi au plus l'un de ses fils peut rendre une valeur non neutre pour $\oplus$



Les noeuds déterministes ont une rôle de «sélecteur» plus qu'un rôle d'aggrégateur.

- Le déterminisme assure que $(\phi \oplus \psi)_{y \leftarrow vrai} \otimes (\phi \oplus \psi)_{y \leftarrow faux}$ est équivalent à $(\phi_{y \leftarrow vrai} \otimes \phi_{y \leftarrow faux}) \otimes (\psi_{y \leftarrow vrai} \otimes \psi_{y \leftarrow faux})$ *pour tout opérateur* $\otimes$ *de même élément neutre que* $\oplus$

Propriétés remarquables des «bons» langages cibles pour la compilation

- Le déterminisme des noeuds $\vee$ (et plus largement des noeuds *max*), assorti de la $\wedge$-décomposabilité permet de de compter le nombre d'affectations de coût supérieur ou égal à v
  - $Card_{\geq v}(\phi \vee \psi) = Card_{\geq v}(\phi) + Card_{\geq v}(\psi)$
  - $Card_{\geq v}(\phi \wedge \psi) = Card_{\geq v}(\phi) \times Card_{\geq v}(\psi)$

  Ex : calcul du nombre de mondes couverts en planification conformante

Répères bibliographiques pour cet exposé

- Darwiche Marquis 02 : premiere carte de compilation, étendue depuis
- Torta Torasso depuis DX'04 : diagnostic sur des OBDD ;
  (Darwiche) Huang depuis AAAI'05 : diagnostic sur des Dnnf
- Giunchiglia Traverso, ECP'97 et suivants : "*planning as model checking*" ;
  Jensen Veloso, AIJ'99, Cimati et al., AIJ'03 : récursion arrière sur des OBDD ;
  Hoey et al., UAI'99 : récursion arrière sur des ADD.
- Amilhastre et al. AIJ'02 : configuration sur des automates ;
  voir aussi les travaux de T. Hadzic depuis 2004.
- Et d'autres, en raisonnement, en SAT, QBF, . . .
- Compilateurs : Huang Darwiche JAIR'07 (d-DNNF), Huang et al. AAAI'06 (circuits arithmétiques), Somezi (cudd), Amilhastre et al. 99 (automates)

Quelques remarques pour finir

- Dans le cas propositionnel, on a rarement besoin du déterminisme ;
- Il a été montré que les formes non déterministes peuvent être exponentiellement plus compactes
- Mais en pratique, on utilise presque toujours des formes compilées déterministes
- Car les deux types de compilateurs disponibles (OBDD, (d)-DNNF) calculent des diagrammes de décision
  - DNNF actuelles : plutôt des "*trees of OBDD*"

Bon, mais on peut toujours utiliser un compilateur déterministe et traiter la forme compilée sans essayer de garder le déterminisme.
Et puis il y a Horn, 2 CNF, cluster trees, etc....

Quelques remarques pour finir

Compilateurs de formes valuées :

- Automates + poids unitaires : pouvoir d'expression limité
- *Algebraic decision diagrams* : pas bien performants en espace ;
- *Affine algebraic decision diagrams* : pas de compilateur dispo ;
- ACE : fondé sur le compilateur de (d)-DNNF – pas de code source

Bon, alors, au boulot ! ! !