DIC9305 Logique, informatique et sciences cognitives

Calcul propositionnel

Roger Villemaire

Département d'informatique
UQAM

28 mars 2024

UQÀM
Université du Québec à Montréal

Plan

Plan

Définition

- Le *calcul propositionnel* est la logique des propositions, des affirmations *vraies* (valeur 1) ou *fausses* (valeur 0), et des connecteurs *booléens* :
  - la négation $\neg$,
  - la *conjonction* $\wedge$ (le “et” logique),
  - la *disjonction* $\vee$ (le “ou” logique).
- On utilise aussi fréquemment :
  - l’*implication* $P \rightarrow Q \equiv \neg P \vee Q$ (si ... alors),
  - la *double implication* $P \leftrightarrow Q \equiv (P \rightarrow Q) \wedge (Q \rightarrow P)$ (si et seulement si).

Exemple de modélisation

Localiser des objets sur le carrelage 2x2 d'une pièce.

$O_{1,1}$	$O_{1,2}$
$O_{2,1}$	$O_{2,2}$

- Les variables :
  - $O_{i,j}$: "un objet est présent à la ligne i, colonne j".
- Quelques formules propositionnelles :
  - Il y a un objet sur la ligne 1 : $O_{1,1} \vee O_{1,2}$.
  - Il n'y a pas d'objet sur la ligne 2 : $\neg(O_{2,1} \vee O_{2,2})$.
  - Il y a des objets aux positions $(1,1)$ et $(2,1)$: $O_{1,1} \wedge O_{2,1}$.
  - S'il y a un objet en $(1,2)$, il y en a un en $(2,2)$: $O_{1,2} \rightarrow O_{2,2}$.

Plan

Sémantique

- Une *affectation* (ou *valuation*) v est une fonction qui associe des valeurs de vérité (*Vrai* (1) ou *Faux* (0)) aux variables propositionnelles.
  - On définit récursivement le sens d'une formule propositionnelle :
    - $v \models P$, pour une variable P, si $v(P) = \mathit{Vrai}$.
    - $v \models \neg\varphi$, si $v \not\models \varphi$.
    - $v \models \varphi \wedge \psi$, si $v \models \varphi$ et $v \models \psi$.
    - $v \models \varphi \vee \psi$, si $v \models \varphi$ ou $v \models \psi$.
    - $v \models \varphi \rightarrow \psi$, si $v \models \psi$ lorsque $v \models \varphi$.
    - $v \models \varphi \leftrightarrow \psi$, si $v \models \varphi$ ssi $v \models \psi$.
- On note par $\models \varphi$, le fait que φ est une *tautologie* (ou est *valide*), c.-à-d. satisfaite par toute les affectations.

Exemple

Soient $v(O_{1,1}) = 1, v(O_{1,2}) = 0, v(O_{2,1}) = 0, v(O_{2,2}) = 1$.

$O_{1,1} = 1$	$O_{1,2} = 0$
$O_{2,1} = 0$	$O_{2,2} = 1$

- $v \models O_{1,1} \vee O_{1,2}$, donc il y a un objet sur la ligne 1.
- $v \not\models \neg(O_{2,1} \vee O_{2,2})$, donc il est faux qu'il n'y a pas d'objet sur la ligne 2 !
- $v \not\models O_{1,1} \wedge O_{2,1}$, donc il n'y a pas d'objets (simultanément) aux positions $(1, 1)$ et $(2, 1)$.
- $v \models O_{1,2} \rightarrow O_{2,2}$, car il est vrai que s'il y a un objet en $(1, 2)$, il y en a un en $(2, 2)$. ATTENTION !

Plan

Fondation

Le calcul propositionnel est la fondation sur laquelle repose toutes les autres logiques classiques.

- Une proposition représente un énoncé.
- Les connecteurs booléens, sont les opérateurs fondamentaux permettant de combiner les propositions pour formuler des énoncées plus complexes.

Formalisation

Des problèmes complexes sont formalisables en calcul propositionnel.

- Représenter les données par un ensemble de variables propositionnelles.
- Représenter les relations, contraintes etc. à l'aide des connecteurs propositionnels.
- Cette méthode a été appliquée à :
  - la résolution de contraintes,
  - d'autres logiques, ...
- En général la traduction est volumineuse et doit être faite par un logiciel.
- Il peut y avoir de nombreuses traductions qui donnent lieu à des solutions très variables en efficacité.

Exemple : le Sudoku

- $C_{i,j,k}$: la case (i, j) contient la valeur k $(i, j, k \in 1..9)$.
- On peut formaliser en logique propositionnelle :
  - Chaque case contient une et une seule valeur.
  - Chaque valeur n'apparaît qu'une seule fois sur chaque ligne.
  - Chaque valeur n'apparaît qu'une seule fois sur chaque colonne.
  - Chaque valeur n'apparaît qu'une seule fois dans chacun des 9 carrés.
  - Certaines cases ont des valeurs déjà déterminées.
- Si C est la conjonction de toutes ces propriétés, on peut chercher la (ou les) solution(s) en tentant de satisfaire C.
- On peut aussi répondre à des questions comme "doit-il y avoir nécessairement un 2 sur la case $(5, 3)$", en vérifiant si $C \rightarrow C_{5,3,2}$ est une tautologie.

Plan

Les deux grandes questions algorithmiques

- Vérification de modèle (*model-checking*) : Étant donnée une structure (affectation) fixe, déterminer si une formule est satisfaite ou non.
  - Ceci peut être déterminé en temps $O(N)$ où N est la taille de la formule. Donc rapidement.
- Satisfiabilité (SAT) : Déterminer s'il existe une structure (affectation) qui satisfait un certaine formule.
  - Ce problème est NP-complet, donc en pratique tous les algorithmes connus sont de temps exponentiel.
  - Néanmoins, on a aujourd'hui de très efficace *SAT-solvers* qui peuvent traiter de très grandes formules (millions de variables).

Satisfaisabilité et validité

- φ est une tautologie ssi $\neg\varphi$ n'est pas satisfaisable.
- φ est satisfaisable ssi $\neg\varphi$ n'est pas une tautologie
- Donc satisfaction et détermination des tautologies sont des problèmes équivalents.

La forme normale négative

- Une formule est en *forme normale négative* (FNN) si la négation n'est utilisée que sur des variables.
  - $\neg P \wedge \neg((\neg P \vee Q) \wedge (P \vee \neg Q))$ n'est pas en FNN.
  - $\neg P \wedge (P \wedge \neg Q) \vee (\neg P \wedge Q)$ est une FNN.
- On peut transformer une formule en FNN en "poussant" les négations vers l'intérieur en utilisant les règles de de Morgan :
  - $\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$.
  - $\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$.

Exemple

- $\neg(P \vee (\neg P \vee Q) \wedge (P \vee \neg Q))$
- $\neg P \wedge \neg((\neg P \vee Q) \wedge (P \vee \neg Q))$
- $\neg P \wedge (\neg(\neg P \vee Q) \vee \neg(P \vee \neg Q))$
- $\neg P \wedge ((P \wedge \neg Q) \vee (\neg P \wedge Q))$
- On passe sur chaque conjonction et chaque disjonction, une et une seule fois. La complexité est donc linéaire par rapport à la taille de la formule.

Exemple

- $\neg(P \vee (\neg P \vee Q) \wedge (P \vee \neg Q))$
- $\neg P \wedge \neg((\neg P \vee Q) \wedge (P \vee \neg Q))$
- $\neg P \wedge (\neg(\neg P \vee Q) \vee \neg(P \vee \neg Q))$
- $\neg P \wedge ((P \wedge \neg Q) \vee (\neg P \wedge Q))$
- On passe sur chaque conjonction et chaque disjonction, une et une seule fois. La complexité est donc linéaire par rapport à la taille de la formule.

Exemple

- $\neg(P \vee (\neg P \vee Q) \wedge (P \vee \neg Q))$
- $\neg P \wedge \neg((\neg P \vee Q) \wedge (P \vee \neg Q))$
- $\neg P \wedge (\neg(\neg P \vee Q) \vee \neg(P \vee \neg Q))$
- $\neg P \wedge ((P \wedge \neg Q) \vee (\neg P \wedge Q))$
- On passe sur chaque conjonction et chaque disjonction, une et une seule fois. La complexité est donc linéaire par rapport à la taille de la formule.

Exemple

- $\neg(P \vee (\neg P \vee Q) \wedge (P \vee \neg Q))$
- $\neg P \wedge \neg((\neg P \vee Q) \wedge (P \vee \neg Q))$
- $\neg P \wedge (\neg(\neg P \vee Q) \vee \neg(P \vee \neg Q))$
- $\neg P \wedge ((P \wedge \neg Q) \vee (\neg P \wedge Q))$
- On passe sur chaque conjonction et chaque disjonction, une et une seule fois. La complexité est donc linéaire par rapport à la taille de la formule.

Exemple

- $\neg(P \vee (\neg P \vee Q) \wedge (P \vee \neg Q))$
- $\neg P \wedge \neg((\neg P \vee Q) \wedge (P \vee \neg Q))$
- $\neg P \wedge (\neg(\neg P \vee Q) \vee \neg(P \vee \neg Q))$
- $\neg P \wedge ((P \wedge \neg Q) \vee (\neg P \wedge Q))$
- On passe sur chaque conjonction et chaque disjonction, une et une seule fois. La complexité est donc linéaire par rapport à la taille de la formule.

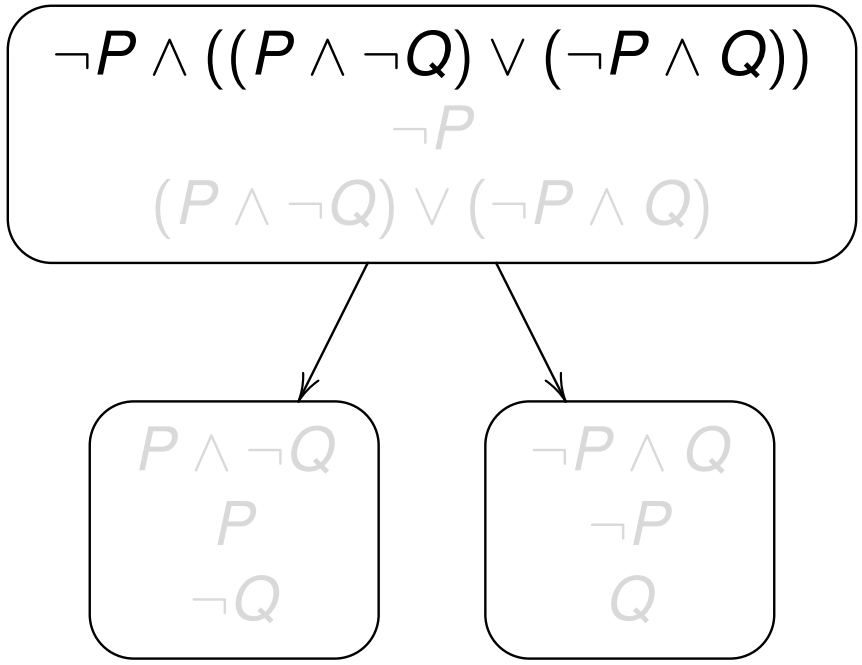
FNN, $\wedge$ et $\vee$

- On peut toujours se restreindre à des FNN ne contenant que $\neg$, $\wedge$ et $\vee$.
  - On remplace $\varphi \leftrightarrow \psi$ par la formule équivalente $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$.
  - On remplace $\varphi \rightarrow \psi$ par la formule équivalente $\neg\varphi \vee \psi$.
- Dorénavant, on supposera que les formules ne contiennent que $\neg$, $\wedge$ et $\vee$ et sont en FNN.

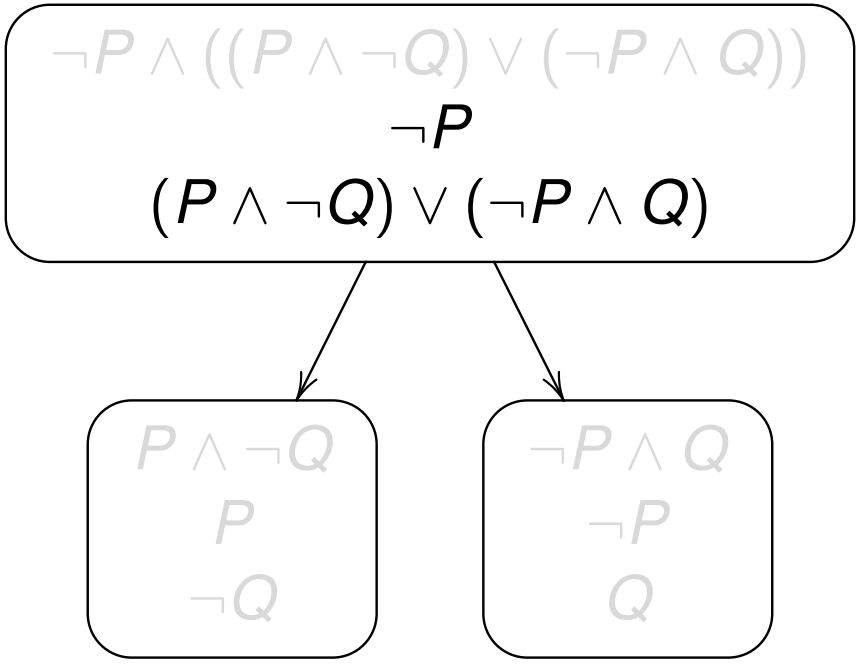
La méthode des tableaux (analytiques de Smullyan)

- On construit un arbre en débutant avec une racine contenant une formule (en FNN) à satisfaire et pour tout noeud $\mathcal{N}$ on applique :
  - Si $\varphi \wedge \psi \in \mathcal{N}$ alors on ajoute φ et ψ à $\mathcal{N}$.
  - Si $\varphi \vee \psi \in \mathcal{N}$ alors on crée deux descendants $\mathcal{N}_1$ et $\mathcal{N}_2$ de $\mathcal{N}$, contenant respectivement φ et ψ.

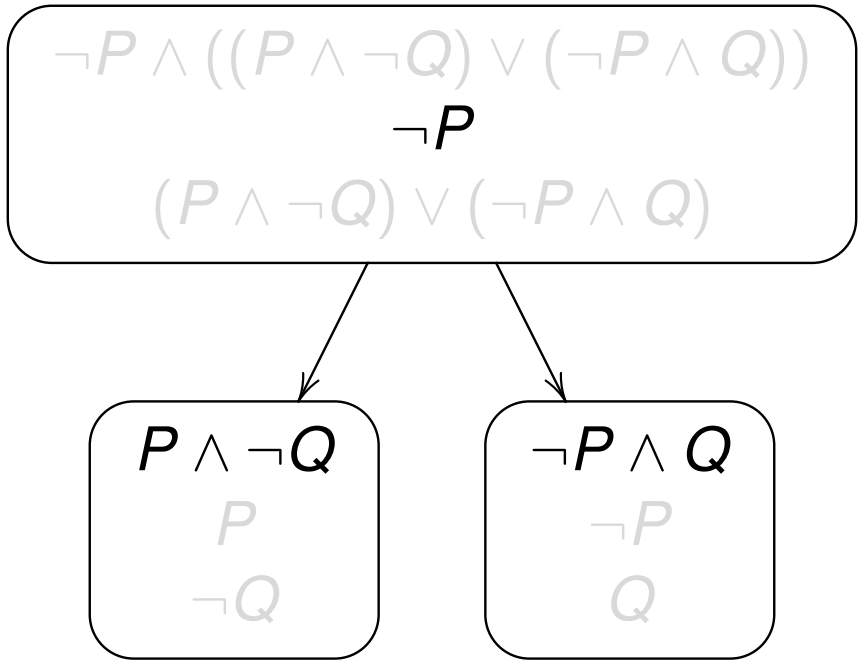
Exemple


$\neg P \wedge ((P \wedge \neg Q) \vee (\neg P \wedge Q))$
$\neg P$
$(P \wedge \neg Q) \vee (\neg P \wedge Q)$
$P \wedge \neg Q$
P
$\neg Q$
$\neg P \wedge Q$
$\neg P$
Q

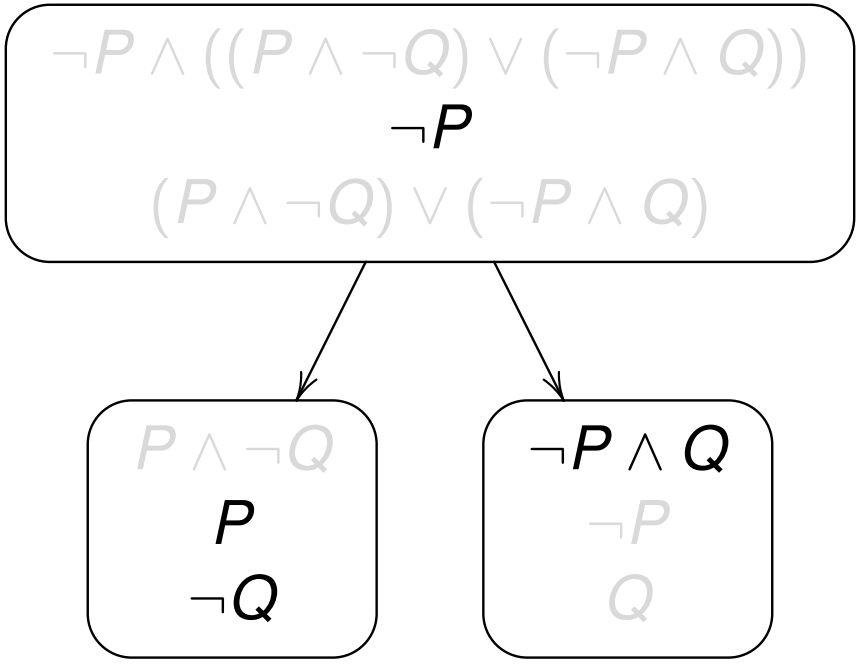
Exemple


$\neg P \wedge ((P \wedge \neg Q) \vee (\neg P \wedge Q))$
$\neg P$
$(P \wedge \neg Q) \vee (\neg P \wedge Q)$
$P \wedge \neg Q$
P
$\neg Q$
$\neg P \wedge Q$
$\neg P$
Q

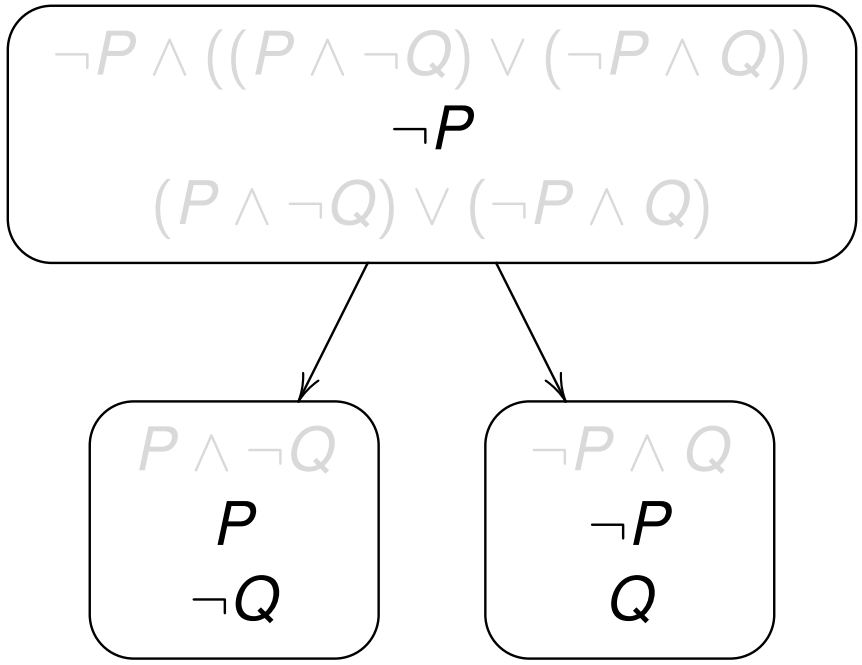
Exemple



Exemple


$\neg P \wedge ((P \wedge \neg Q) \vee (\neg P \wedge Q))$
$\neg P$
$(P \wedge \neg Q) \vee (\neg P \wedge Q)$
$P \wedge \neg Q$
P
$\neg Q$
$\neg P \wedge Q$
$\neg P$
Q

Exemple

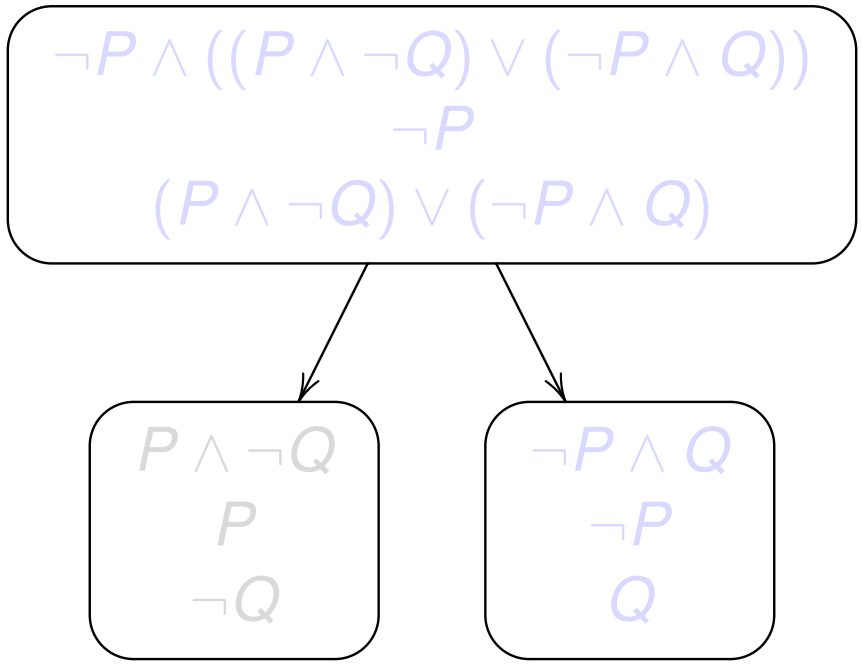


Définitions

- On dit qu'un tableau est *complet* si toutes les conjonctions et les disjonctions du tableau ont été traitées par les règles ci-dessus.
- On dit qu'une branche d'un tableau est *close* si elle contient P et $\neg P$, pour un certain P.
- On dit qu'un tableau est *clos* si toutes ses branches sont closes.

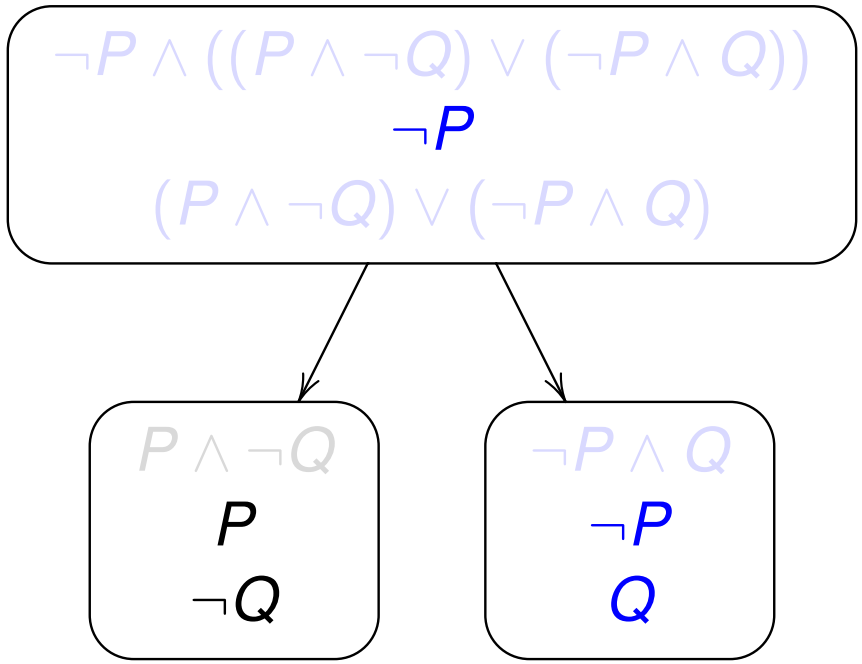
Affectation

Une branche ouverte d'un tableau permet de définir une affectation satisfaisant la formule de départ.



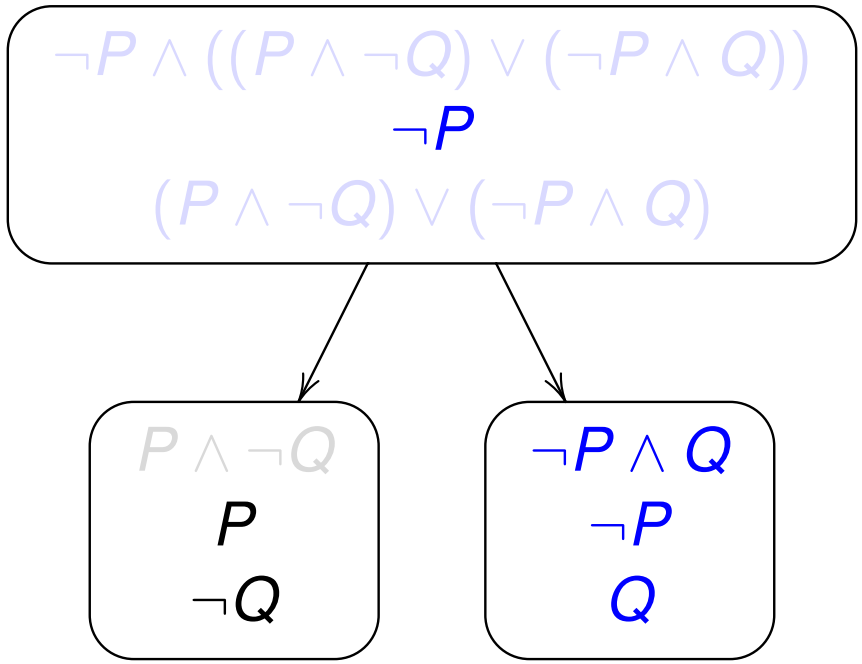
Affectation

Une branche ouverte d'un tableau permet de définir une affectation satisfaisant la formule de départ.



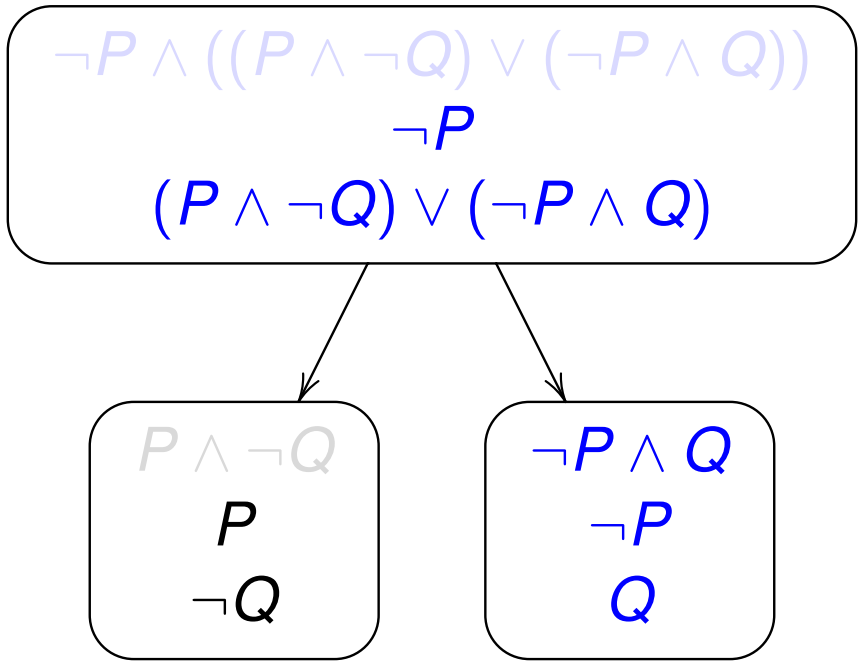
Affectation

Une branche ouverte d'un tableau permet de définir une affectation satisfaisant la formule de départ.



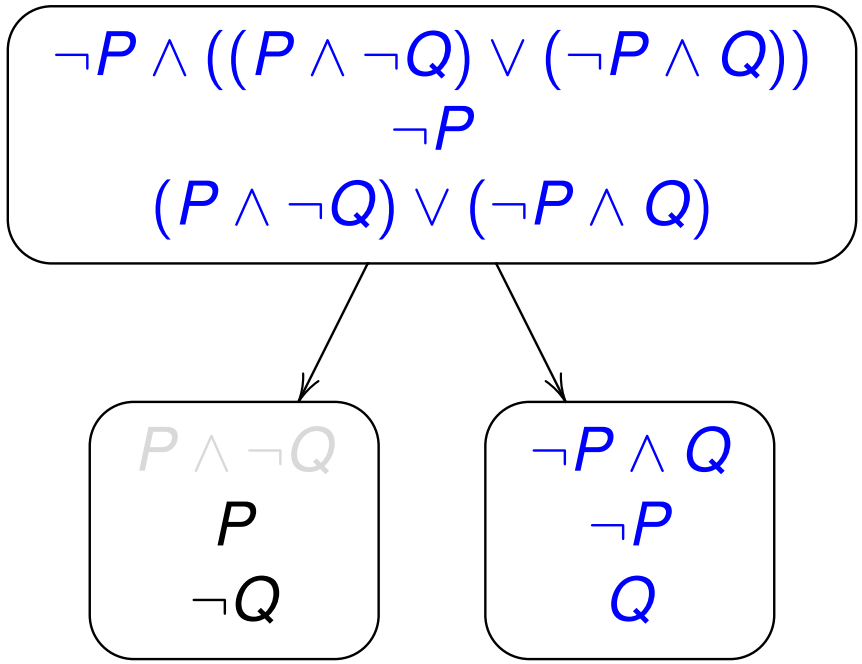
Affectation

Une branche ouverte d'un tableau permet de définir une affectation satisfaisant la formule de départ.



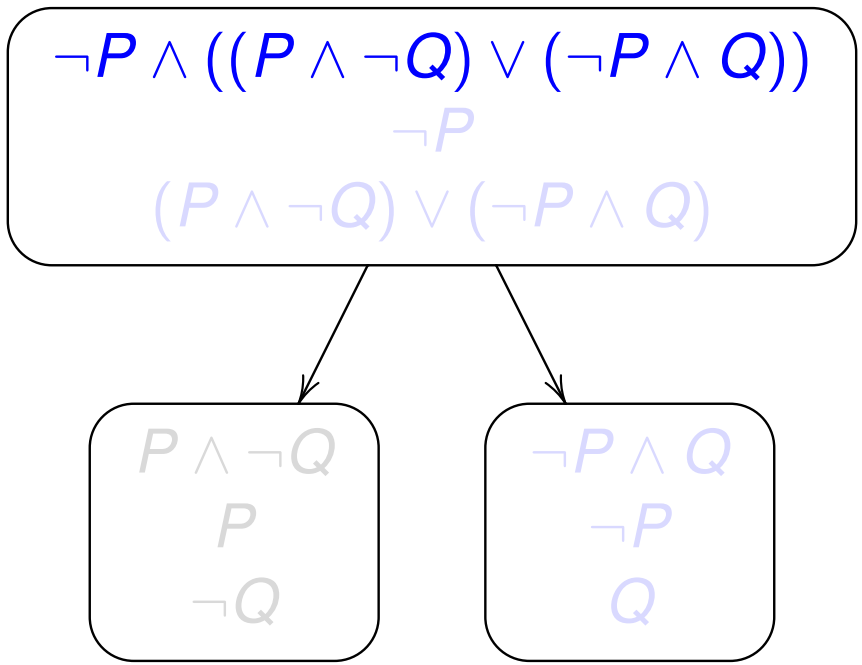
Affectation

Une branche ouverte d'un tableau permet de définir une affectation satisfaisant la formule de départ.



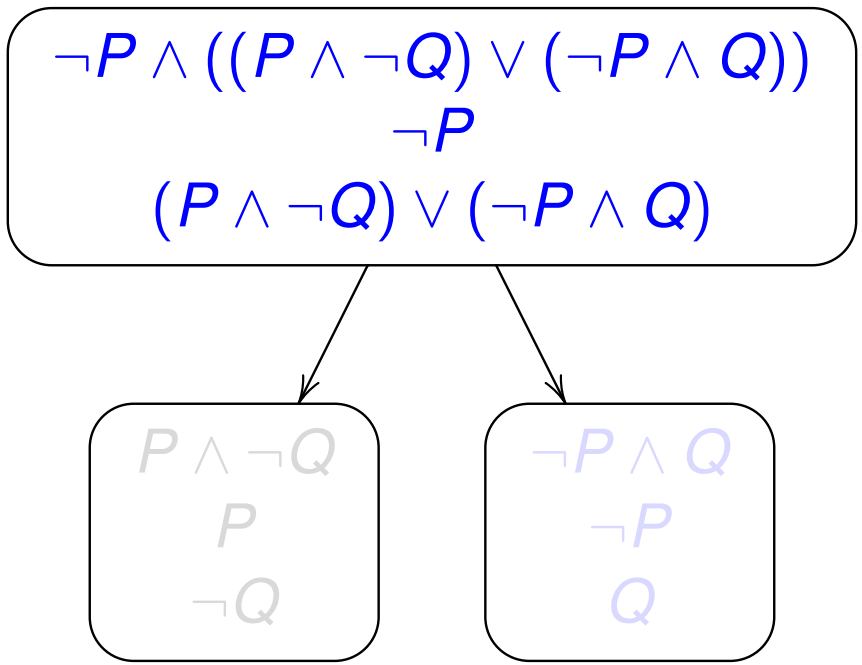
Branche

Une affectation satisfaisant une formule doit satisfaire toutes les formules d'au moins une branche d'un tableau.
Considérons $v(P) = 0$, $v(Q) = 1$.



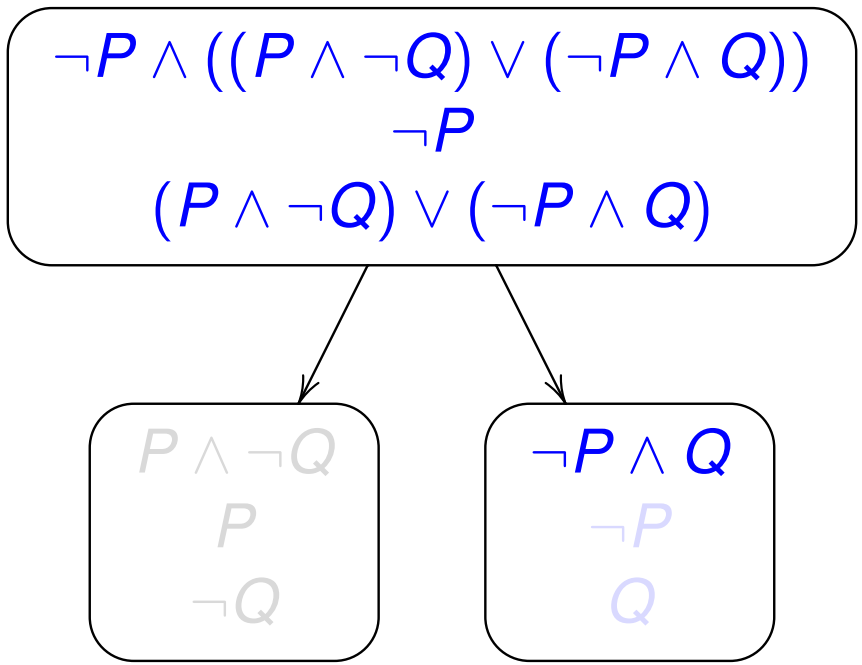
Branche

Une affectation satisfaisant une formule doit satisfaire toutes les formules d'au moins une branche d'un tableau.
Considérons $v(P) = 0$, $v(Q) = 1$.



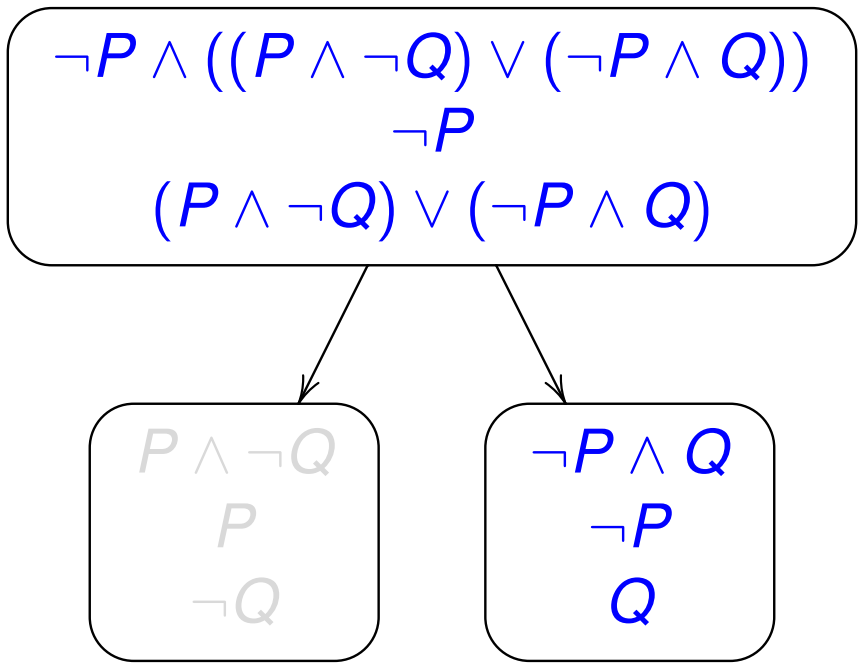
Branche

Une affectation satisfaisant une formule doit satisfaire toutes les formules d'au moins une branche d'un tableau.
Considérons $v(P) = 0$, $v(Q) = 1$.



Branche

Une affectation satisfaisant une formule doit satisfaire toutes les formules d'au moins une branche d'un tableau.
Considérons $v(P) = 0$, $v(Q) = 1$.



Réfutation

- On dit qu'une formule φ est *réfutable* si tout tableau complet pour φ est clos.

Complétude

Théorème

φ est insatisfaisable si et seulement si φ est réfutable.

Démonstration

(=>) Considérons un tableau complet pour φ. Si ce tableau avait une branche ouverte, les littéraux (variables et négations de variables) apparaissant sur cette branche serait une affectation satisfaisant φ, mais ceci est impossible car φ est insatisfaisable. Toutes les branches sont donc closes, le tableau est donc clos.
(<=) Réciproquement, considérons une affectation. Si cette affectation satisfaisait φ, elle satisferait toutes les formules d'au moins une branche d'un tableau complet pour φ. Cette branche ne serait donc pas close (elle ne peut contenir simultanément P et $\neg P$). Mais ceci contredit le fait que tout tableau complet pour φ est clos.

Interprétation

- Deux interprétations de la complétude :
  - Les tableaux forment une méthode *syntaxique* assez puissante pour que toute formule réfutable soit insatisfaisable.
  - La notion d'affectation est une *sémantique* assez souple pour que toute formule insatisfaisable soit réfutable.

Algorithmique

- La construction d'un tableau se fait récursivement sur la structure de la formule, donc c'est une inférence dirigée par la structure de la formule.
  - En fait, on peut montrer que la complexité temporelle est exponentielle par rapport au nombre d'*alternations* de conjonctions et de disjonctions.

Autres méthodes

- Il existe d'autres méthodes pour déterminer si une formule est une tautologie.
- Les méthodes les plus efficaces aujourd'hui sont probablement les dérivés de l'algorithme DPLL.
- Néanmoins la méthode des tableaux est :
  - généralisable à plusieurs autres logiques (modale, de description, etc),
  - utilisée en pratique dans plusieurs outils.

Plan

Conclusion

- La logique propositionnelle est :
  - à la base de presque toutes les autres logiques,
  - un outil de formalisation très utile en pratique.
- Néanmoins :
  - une formalisation d'un problème réel en logique propositionnelle n'est réalisable en pratique que par un outil,
  - il est donc nécessaire de considérer des logiques plus expressives permettant une formalisation plus naturelle.