ISC9000 Connaissance, Raisonnement et Prise de Décision

Modélisations logiques, inférence, et ontologies informatiques comme représentation de la connaissance

Roger Villemaire

Département d'informatique
UQAM

16 avril 2026

UQÀM
Université du Québec à Montréal

Plan

Plan

Données ouvertes

- *Société de l'information (ou de la connaissance)* : décisions éclairées, gain en productivité, problèmes complexes (environnement, ...) :
  - informatisation et large intégration de sources hétérogènes,
  - données *ouverte* (accessibles), *extensible* (ajouts) et utilisables par tou.te.s.
- Ex : connaissances bio-médicales, villes intelligentes, ...

Partage

- Mais le partage de données nécessite :
  - des formats normalisés,
  - des primitives à la sémantique claire,
  - et des méthodes algorithmiques efficaces pour exploiter l'information ainsi représentée.
- Pour atteindre ces objectifs l'approche ontologique est aujourd'hui dominante.

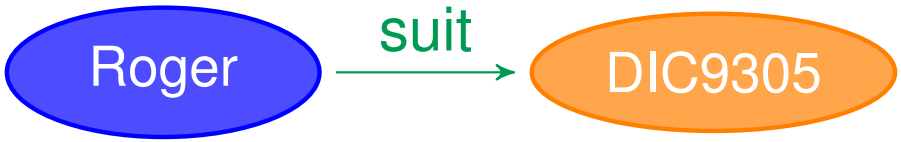
Ontologie (informatique)

- Une ontologie est une terminologie structurée, formelle, qu'on peut exploiter à l'aide d'algorithmes.
- Il s'agit de représenter la connaissance à l'aide des méthodes de la logique et des technologies du *Web Sémantique*[1], principalement :
  - le (Resource Description Framework) (RDF), pour représenter l'information,
  - (Web Ontology Language) (OWL) pour représenter les formules de la *Logique de Description* en RDF.

1. www.w3.org

RDF

- L'information est représentée par des *triplets* formés d'un *sujet*, d'un *prédicat*, et d'un *objet*.
- par exemple $(Roger, suit, DIC9305)$.



- Il s'agit donc d'une représentation par un graphe, dont les sommets sont des entités (sujets et objects), et les arêtes les couples des relations (prédicats).
- On parle donc aussi de Graphes de Connaissance (Knowledge Graphs).

Plan

Logique de Description

- La logique de description permet de représenter
  - des *constantes* qui dénotent des *individus* nommés (particuliers),
  - des *classes* regroupant des individus,
  - des *rôles*, soit des relations binaires entre individus,
  - ainsi que des contraintes sur ces notions.
- De plus, l'inférence de connaissances implicites est en général réalisable par des algorithmes qui permettent donc un raisonnement automatique [2].

2. par ex. à l'aide du *raisonneur* Hermit dans l'outil Protégé

Concepts

- Un *concept* dénote un ensemble d'*individus* :
  - *Etudiant*, représente le concept “être un étudiant”,
  - *Femme*, représente le concept “s'identifier comme une femme”,
  - *Cours*, représente le concept “être un cours”.

Rôles

- Un *rôle* dénote une *relation binaire* entre individus :
  - *suit*, représente le rôle “... suit ...”.

Plan

Expressions de concepts

- $\top$, le concept le *plus général*, applicable à tous les individus,
- $\bot$, le concept le *plus spécifique*, applicable à aucun individu,
- $\sqcap$, la *conjonction* de concepts,
- $\sqcup$, la *disjonction* de concepts,
- $\neg$, la *négation* d'un concept.

Expressions de concepts (suite)

- la quantification sur les rôles,
  - $\forall r.C$, “tous ses r sont des C”,
  - $\exists r.C$, “un de ses r est un C”,
- les *cardinalités* sur les rôles :
  - $\geq n\ r$, “au moins n, r”,
  - $\leq n\ r$, “au plus n, r”.

Exemples

- $\mathit{Etudiant} \sqcap \mathit{Femme}$, représente les étudiantes,
- $\exists \mathit{suit}.\mathit{Cours}$, représente toutes les entités qui suivent un cours.

Comparaisons de concepts

- $\sqsubseteq$, la *subsomption*,
- $\equiv$, l'*équivalence*,
  - En fait on peut définir $C \equiv D$ par $C \sqsubseteq D$ et $D \sqsubseteq C$.

Exemples

- $\exists suit.Cours \sqsubseteq Etudiant$,
- $Etudiant \sqcap Cours \equiv \bot$.

Plan

TBox

- au niveau *terminologique* (TBox), on formalise le domaine :
  - $\exists suit.Cours \sqsubseteq Etudiant$,
  - $Etudiant \sqcap Cours \equiv \bot$,

ABox

- Les assertions (ABox) représentent des données :
  - *Cours*(*DIC*9305),
  - *Roger suit DIC*9305 [3].

3. *Roger* et *DIC*9305 sont ici des constantes (individus nommés)

Plan

Définition

- La *logique du premier-ordre* permet de construire des expressions à l'aide de :
  - des *variables* $x, y, z, u, v, x_1, y_1, z_1, u_1, v_1, \ldots$,
  - des *constantes* $a, b, c, a_1, b_1, c_1, \ldots$,
  - des *symboles de relations* $R(x), S(x, y) \ldots$,
  - les *connecteurs logiques* $\wedge$ ("et"), $\vee$ ("ou"), $\neg$ ("négation"), $\rightarrow$ ("si...alors.."),
  - ainsi que les *quantificateurs*
    - *universel* $\forall x$ ("pour tout x"),
    - *existentiel* $\exists x$ ("il existe un x").

Exemple

- $\forall x \neg (Etudiant(x) \wedge Cours(x))$,
  “rien n’est en même temps un étudiant et un cours”.
- $\forall x (\exists y suit(x, y) \wedge Cours(y) \rightarrow Etudiant(x))$,
  “si x suit un cours y, alors x est un étudiant”.
- $\exists y Cours(y) \wedge (\forall x (Etudiant(x) \rightarrow suit(x, y)))$,
  “il y a un cours y que tous les étudiants suivent”.

Plan

Interprétation en logique du premier ordre : concepts

- Un concept C correspond à une relation unaire $C(x)$,
  - $\top$ correspond à $x = x$,
  - $\bot$ correspond à $x \neq x$,
  - $\sqcap$ correspond à la *conjonction* $\wedge$,
  - $\sqcup$ correspond à la *disjonction* $\vee$,
  - $\neg$ correspond à la *négation* $\neg$,

Interprétation en logique du premier ordre : rôles

- Un rôle r correspond à une relation binaire $r(x, y)$,
  - $\forall r.C$ correspond à $\forall y.(r(x, y) \rightarrow C(y))$,
  - $\exists r.C$ correspond à $\exists y.r(x, y) \wedge C(y)$,
  - $\geq n.\ r$ correspond à “il existe au moins n, y différents”,
  - $\leq n.\ r$ correspond à “il n’existe pas plus de n, y différents”.

Exemples

- $\exists suit.Cours \sqsubseteq Etudiant$ correspond à
  $\forall x(\exists ysuit(x, y) \wedge Cours(y) \rightarrow Etudiant(x))$,
- $Etudiant \sqcap Cours \equiv \bot$ correspond à
  $\forall x \neg(Etudiant(x) \wedge Cours(x))$.

Comparaison

- La logique de description restreint donc la logique du premier ordre en
  - ne permettant que des relations unaires et binaires,
  - et que des quantificateurs *gardés* $\forall y.(r(x,y) \rightarrow C(y))$ et $\exists y.r(x,y) \wedge C(y)$.
- Il s'agit donc d'un *fragment* de la logique du premier ordre.
- L'avantage est que la logique de description est *décidable* (dans la plupart de ses versions), alors que la logique du premier-ordre est *indécidable*.

Décidabilité

- Un système logique est *décidable* s'il existe un algorithme pour déterminer si un énoncé est une conséquece, ou non, d'un ensemble d'axiomes.
- Gödel a montré en 1930 que la logique du premier-ordre est *indécidable* et que ceci est la cas dès qu'on y formalise les propriétés de bases des entiers naturels avec l'addition et la multiplication.
- La logique de description a, de son côté, été construite de telle façon à ce qu'elle soit décidable. On peut d'ailleurs y employer la méthode des tableaux analytiques de Smullyan.

Plan

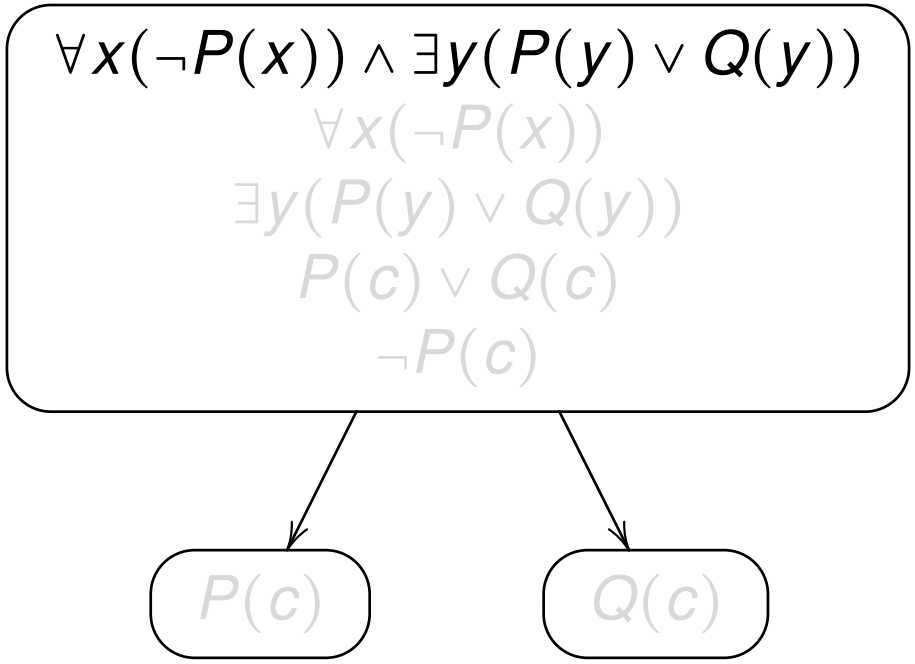
La forme normale négative

- Une formule est en *forme normale négative* (FNN) si la négation n'est utilisée que sur des symboles de relations.
- On peut transformer une formule en FNN en "poussant" les négations vers l'intérieur en utilisant les règles de de Morgan :
  - $\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$.
  - $\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$.
- et la dualité entre $\exists$ et $\forall$:
  - $\neg(\exists x\varphi) \equiv \forall x\neg\varphi$.
  - $\neg(\forall x\varphi) \equiv \exists x\neg\varphi$.

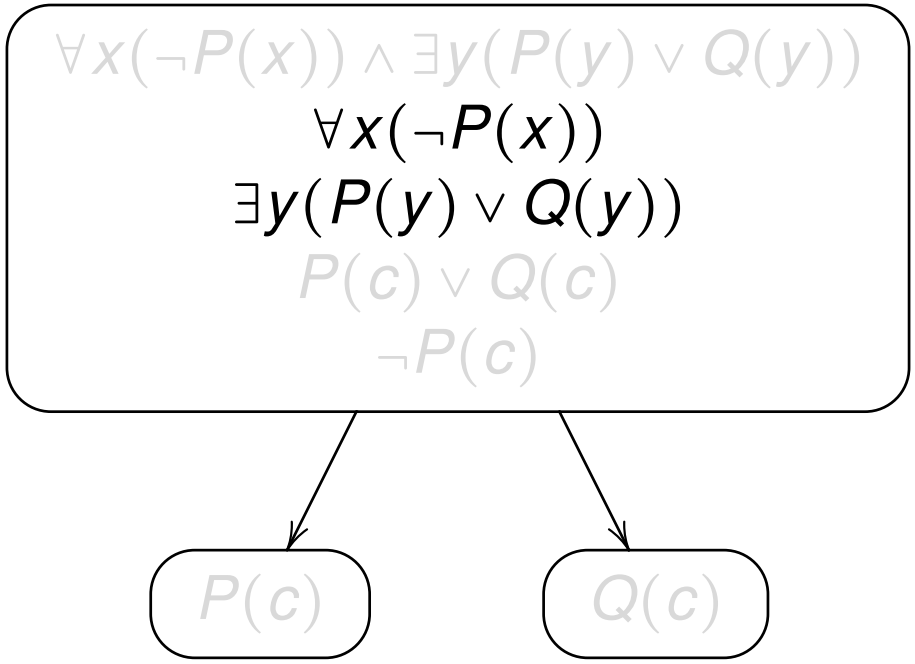
La méthode des tableaux

- 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,
  - Si $\exists x \varphi(x)$ alors on ajoute $\varphi(c)$ pour c une constante n'apparaissant pas sur cette branche,
  - Si $\forall x \varphi(x)$ alors on ajoute $\varphi(c)$ pour toutes les constantes c présentes sur cette branche (il doit y avoir au moins une constante sur la branche, sinon en rajouter une).

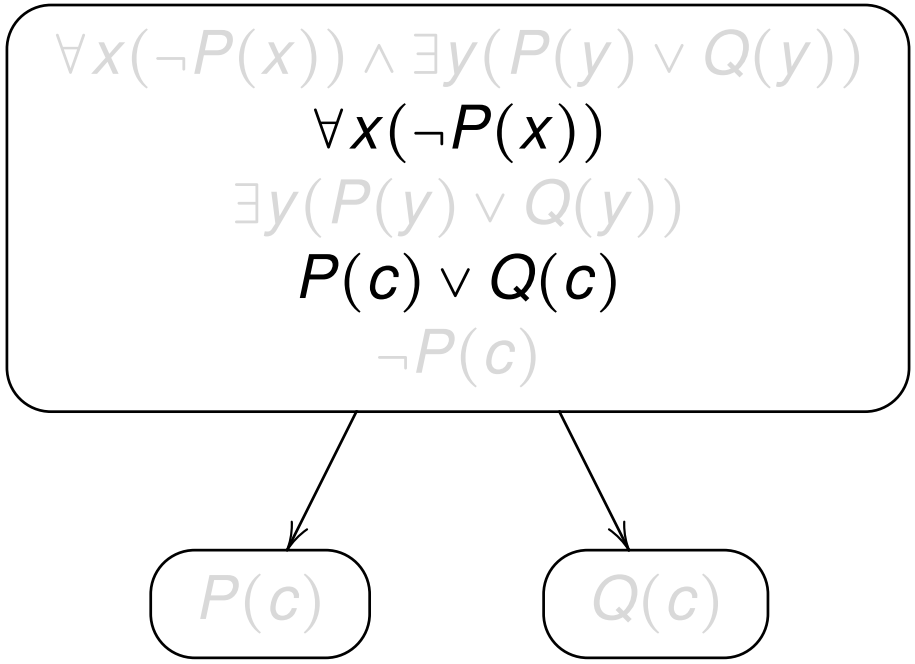
Exemple


$\forall x(\neg P(x)) \wedge \exists y(P(y) \vee Q(y))$
$\forall x(\neg P(x))$
$\exists y(P(y) \vee Q(y))$
$P(c) \vee Q(c)$
$\neg P(c)$
$P(c)$
$Q(c)$

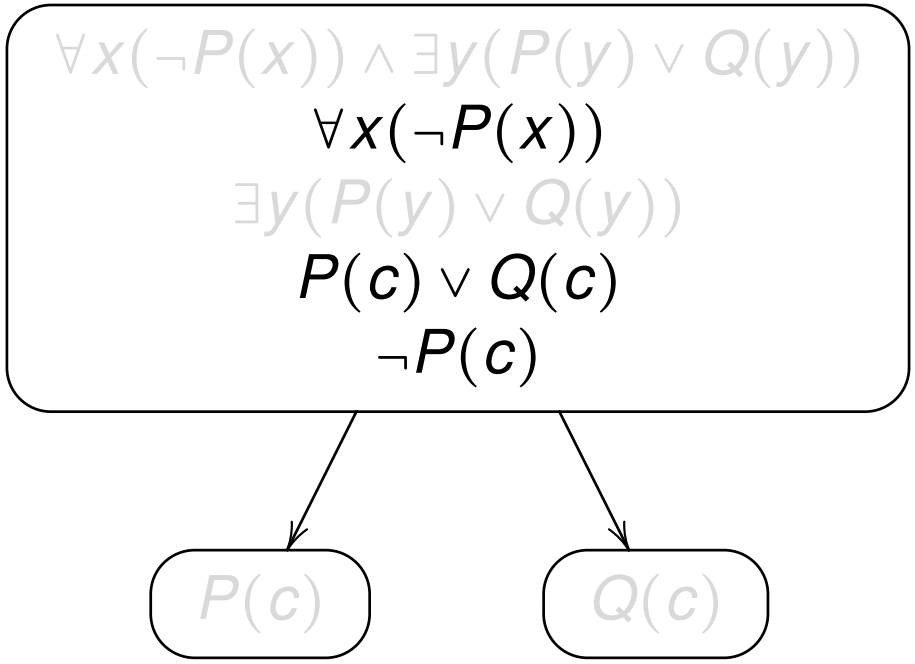
Exemple


$\forall x(\neg P(x)) \wedge \exists y(P(y) \vee Q(y))$
$\forall x(\neg P(x))$
$\exists y(P(y) \vee Q(y))$
$P(c) \vee Q(c)$
$\neg P(c)$
$P(c)$
$Q(c)$

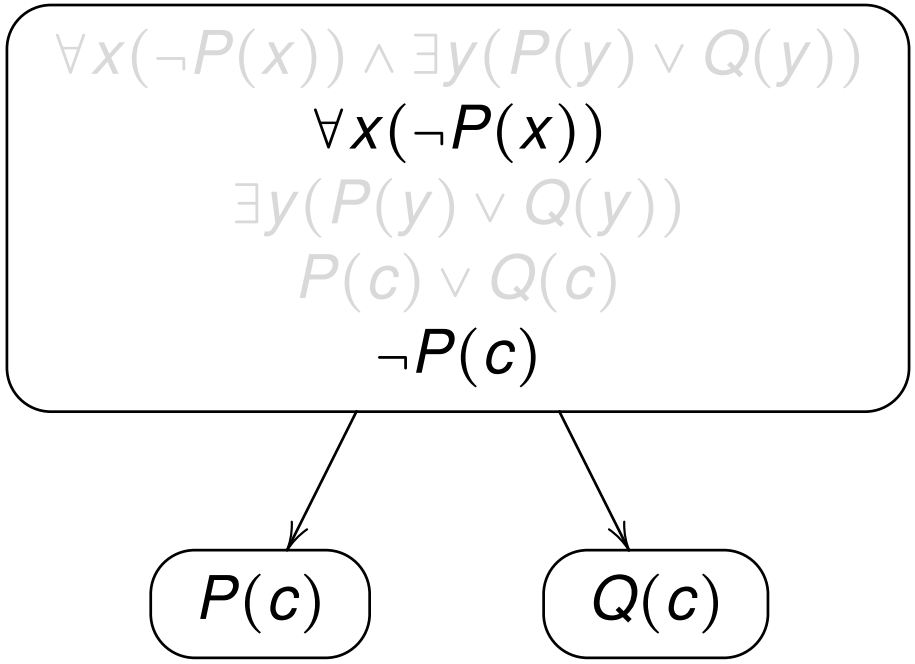
Exemple


$\forall x(\neg P(x)) \wedge \exists y(P(y) \vee Q(y))$
$\forall x(\neg P(x))$
$\exists y(P(y) \vee Q(y))$
$P(c) \vee Q(c)$
$\neg P(c)$
$P(c)$
$Q(c)$

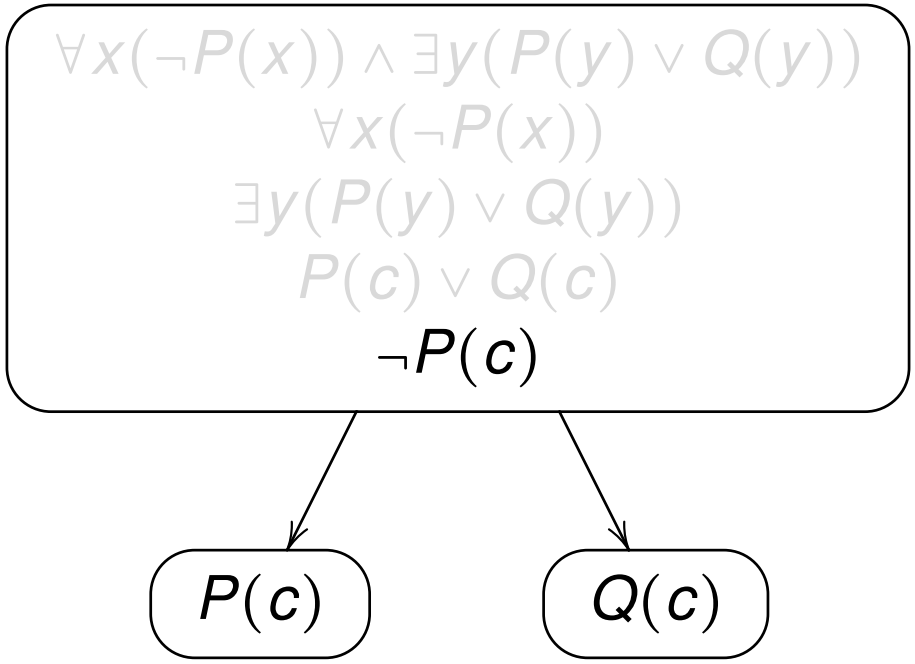
Exemple


$\forall x(\neg P(x)) \wedge \exists y(P(y) \vee Q(y))$
$\forall x(\neg P(x))$
$\exists y(P(y) \vee Q(y))$
$P(c) \vee Q(c)$
$\neg P(c)$
$P(c)$
$Q(c)$

Exemple


$\forall x(\neg P(x)) \wedge \exists y(P(y) \vee Q(y))$
$\forall x(\neg P(x))$
$\exists y(P(y) \vee Q(y))$
$P(c) \vee Q(c)$
$\neg P(c)$
$P(c)$
$Q(c)$

Exemple

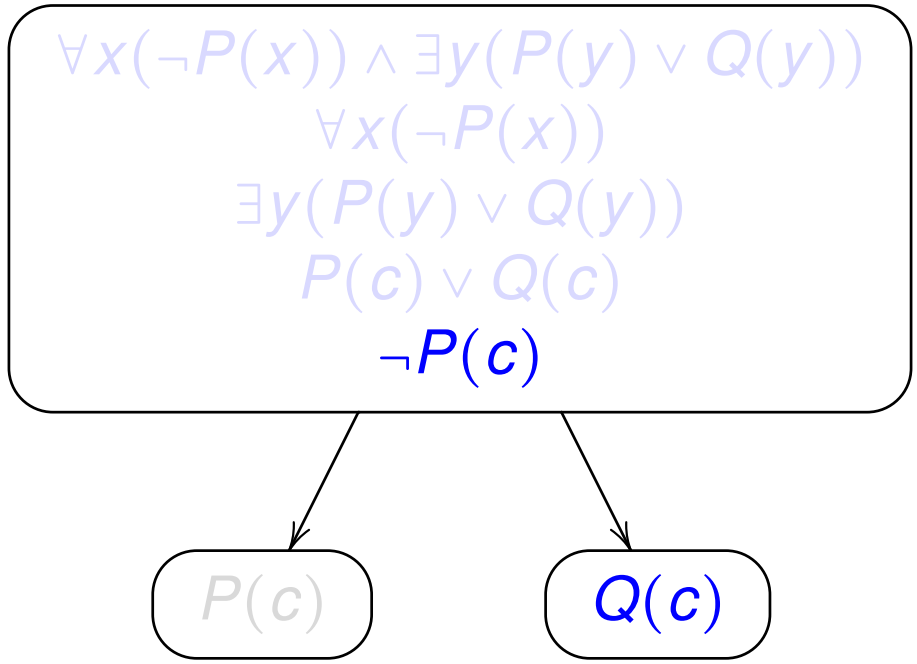

$\forall x(\neg P(x)) \wedge \exists y(P(y) \vee Q(y))$
$\forall x(\neg P(x))$
$\exists y(P(y) \vee Q(y))$
$P(c) \vee Q(c)$
$\neg P(c)$
$P(c)$
$Q(c)$

Définitions

- On dit qu'un tableau est *complet* si toutes les formules 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.

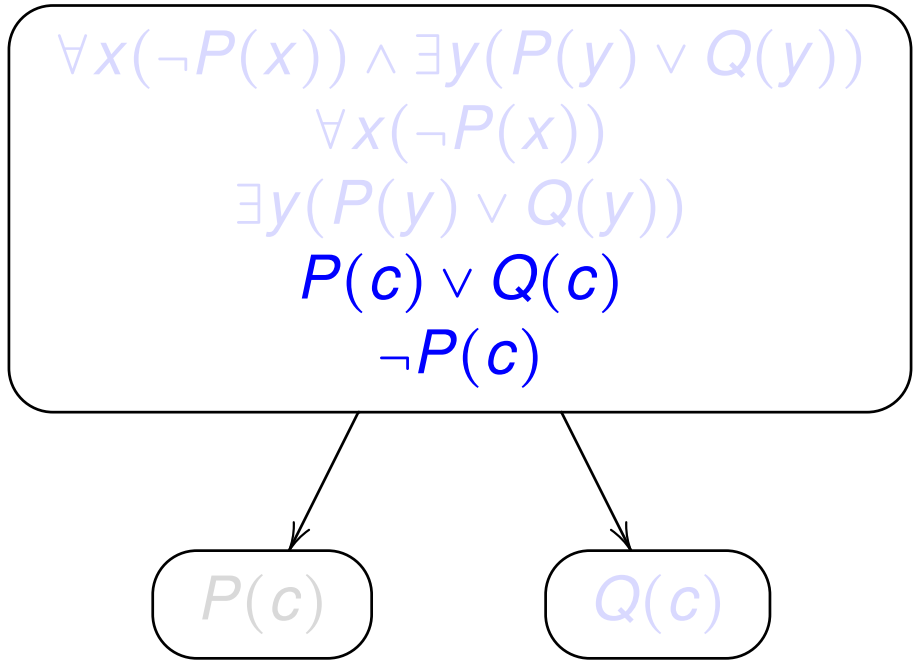
Modèle

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



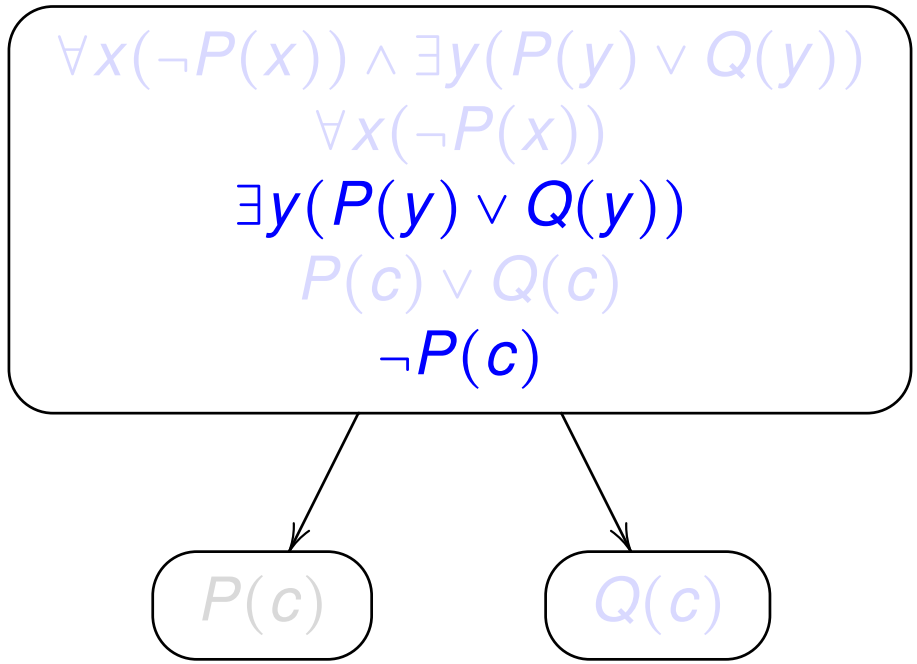
Modèle

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



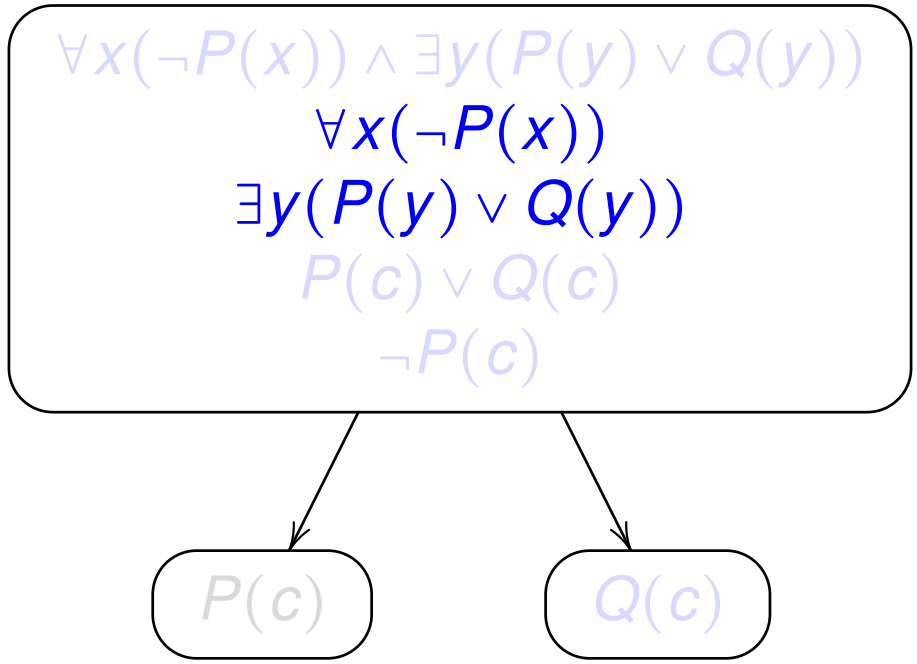
Modèle

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



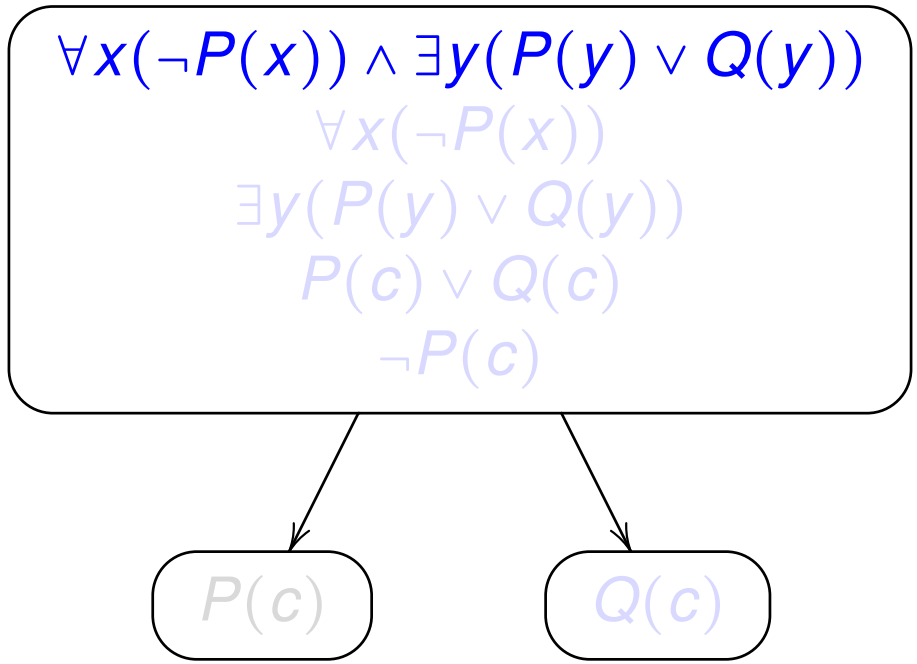
Modèle

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



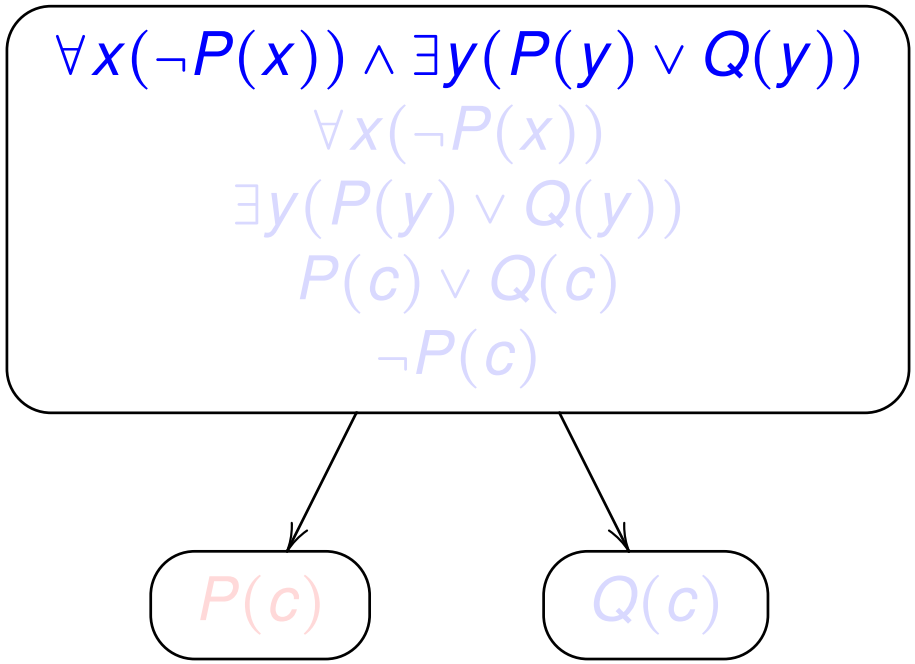
Modèle

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



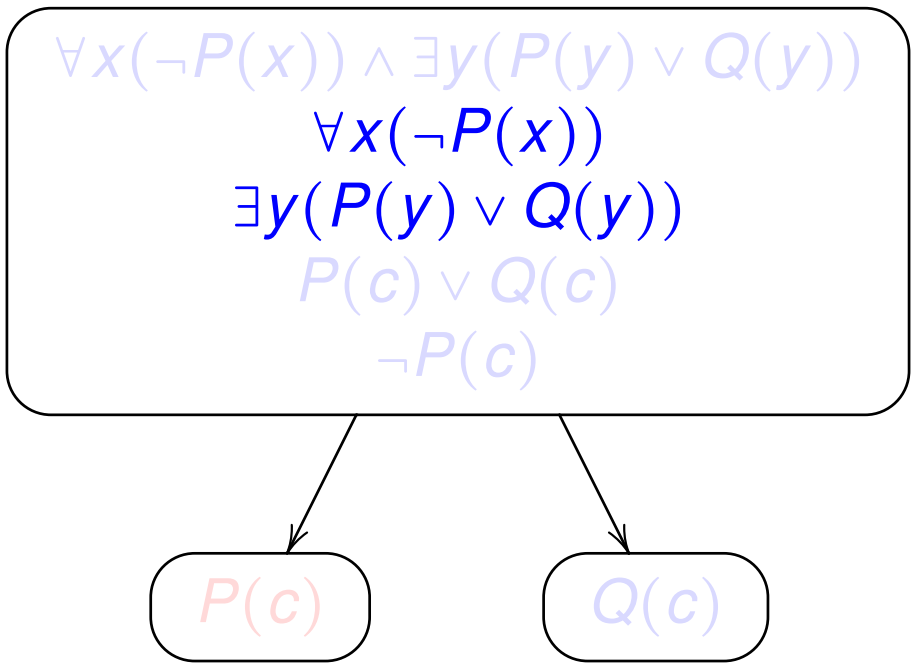
Branche

Une structure satisfaisant une formule (un *modèle*) doit satisfaire toutes les formules d'au moins une branche d'un tableau (pour cette formule).
Considérons $M = \{c\}$ avec $\neg P(c)$ et $Q(c)$.



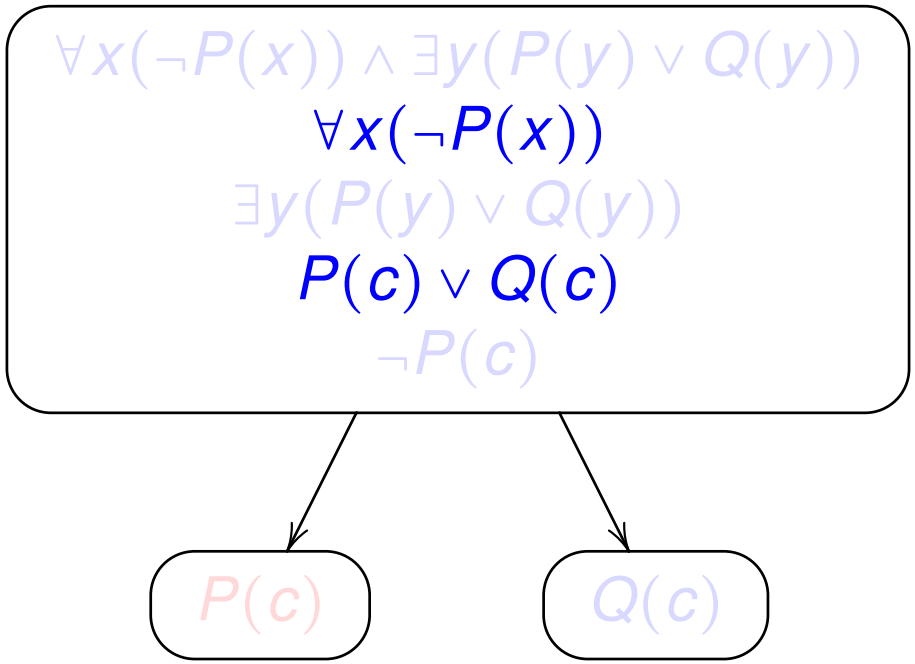
Branche

Une structure satisfaisant une formule (un *modèle*) doit satisfaire toutes les formules d'au moins une branche d'un tableau (pour cette formule).
Considérons $M = \{c\}$ avec $\neg P(c)$ et $Q(c)$.



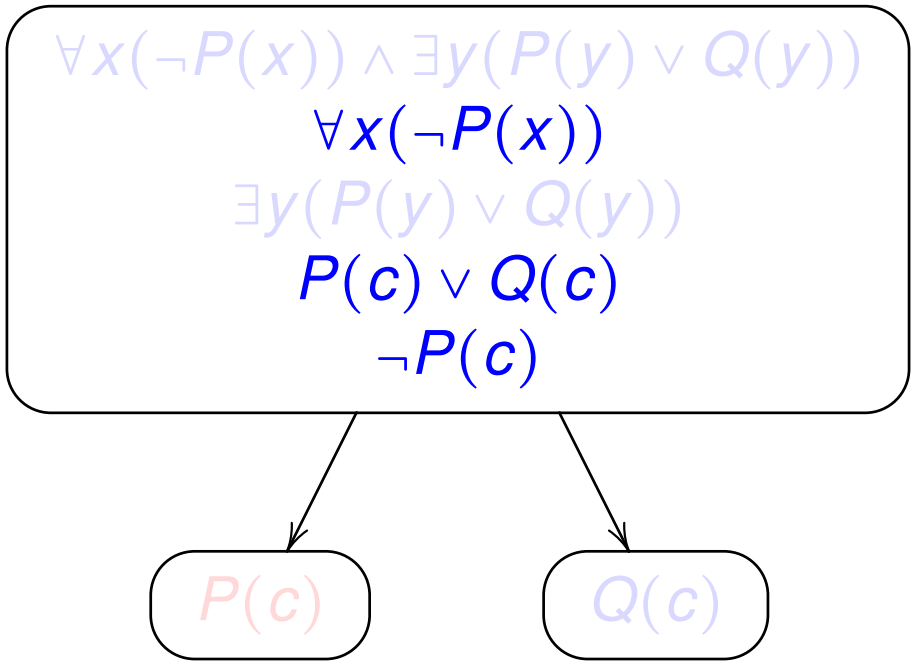
Branche

Une structure satisfaisant une formule (un *modèle*) doit satisfaire toutes les formules d'au moins une branche d'un tableau (pour cette formule).
Considérons $M = \{c\}$ avec $\neg P(c)$ et $Q(c)$.



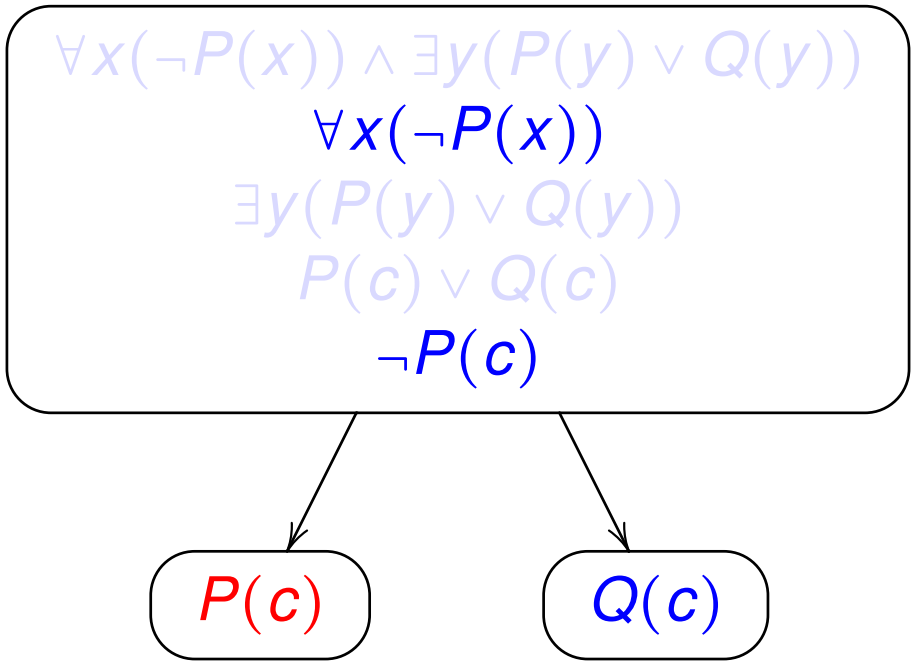
Branche

Une structure satisfaisant une formule (un *modèle*) doit satisfaire toutes les formules d'au moins une branche d'un tableau (pour cette formule).
Considérons $M = \{c\}$ avec $\neg P(c)$ et $Q(c)$.



Branche

Une structure satisfaisant une formule (un *modèle*) doit satisfaire toutes les formules d'au moins une branche d'un tableau (pour cette formule).
Considérons $M = \{c\}$ avec $\neg P(c)$ et $Q(c)$.



Réfutation et Démonstration

- On dit qu'un formule φ est *réfutable* si tout tableau complet pour φ est clos.
- On dit qu'un formule φ est *démontrable*, noté $\vdash \varphi$ si $\neg\varphi$ est réfutable.

Complétude

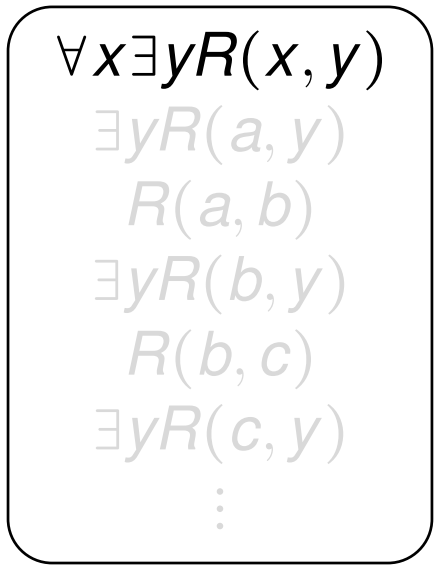
Théorème

φ est insatisfaisable si et seulement si φ est réfutable.

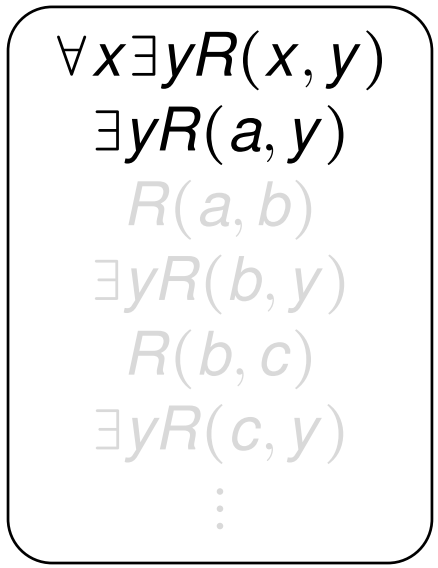
Démonstration et contre-exemple

- Si une formule est démontrable, toutes les branches du tableau pour sa négation seront closes. On aura donc un arbre fini, la réfutation de la négation de la formule, qui est une évidence finie du fait que la formule est démontrable et donc valide.
- Si une formule n'est pas démontrable, une branche au moins du tableau pour sa négation est ouverte. Cette branche est une évidence que la formule n'est pas démontrable et donc n'est pas une tautologie. Mais cette branche peut être infinie !

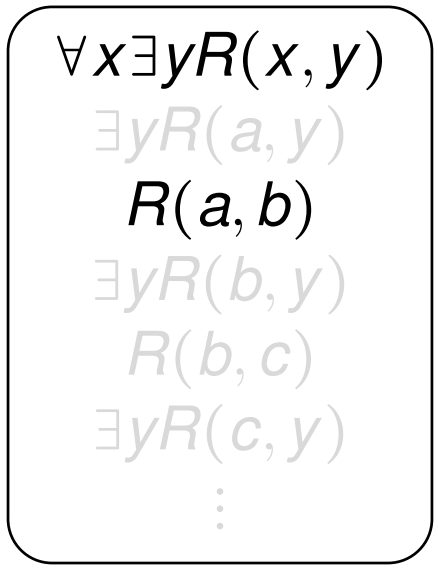
Exemple de branche infinie



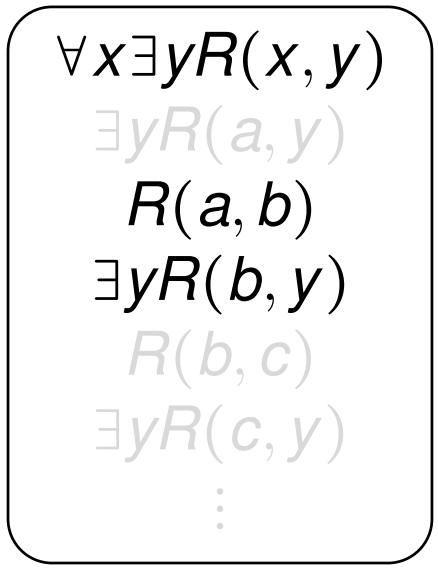
Exemple de branche infinie



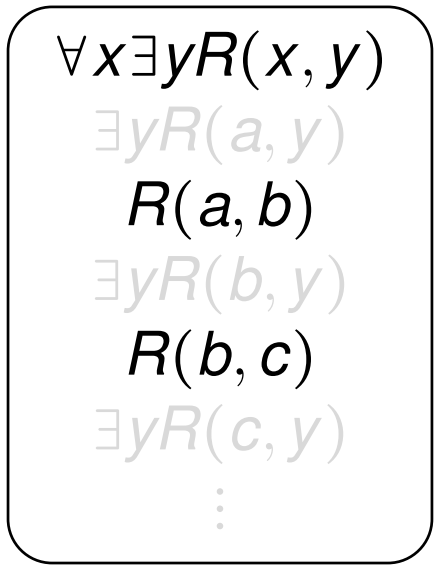
Exemple de branche infinie



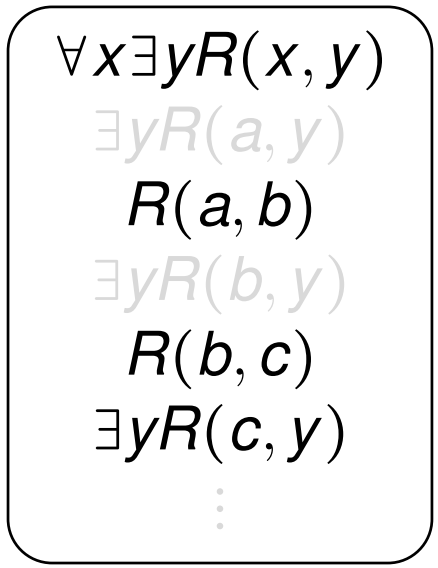
Exemple de branche infinie



Exemple de branche infinie



Exemple de branche infinie



Exemple de branche infinie

$$\forall x \exists y R(x, y)$$
$$\exists y R(a, y)$$
$$R(a, b)$$
$$\exists y R(b, y)$$
$$R(b, c)$$
$$\exists y R(c, y)$$
$$\vdots$$

Structures infinies

- Bien que dans certains cas, par exemple pour la plupart des logiques de description, il soit possible d'éviter de telles branches infinies, par exemple en réutilisant les constantes, ceci n'est pas le cas pour la logique du premier ordre.
- Par exemple, une formule sans modèle fini (par exemple, l'ordre total sans extrémités) ne pourra pas donner lieu à une brache finie.
- On peut néanmoins encoder certains types de braches infinies par une description finie, mais ceci, non plus, n'est pas toujours possible.

Plan

Profils OWL 2

- La norme OWL 2 du W3C introduit, en plus de OWL 2 DL, des profils, qui sont
  - des versions réduites du langage OWL 2 DL (fragments), limitant les constructions possibles,
  - et visant un certain équilibre en expressivité et performance de raisonnement.
- Il s'agit donc de permettre des applications plus performantes et le traitement de plus grands jeux de données.

Objectifs

- L'objectif d'une ontologie contruite à l'aide de la logique de description est de
  - représenter des connaissances d'intérêt, en vue
  - d'inférer des conséquences pour
  - compléter nos connaissances, ou encore les valider.

Processus

- On va donc
  - formaliser le minimum nécessaire aux buts visés,
  - en tentant de réutiliser des ontologies ou des fragments d'ontologies existantes,
  - tout en tâchant de tirer le maximum des moteurs d'inférence existants.
- Il s'agit donc de travailler de façon pragmatique, en fonction des objectifs visés.

Défis

- Déterminer et définir les concepts et rôles d'un domaine réel est une tâche non-univoque, qui nécessite un travail collectif important,
- déterminer les axiomes nécessaires est aussi difficile puisqu'il s'agit de donner une définition exacte de ces concepts et rôles.
- Il existe donc des méthodologies pour établir des concessus entre les experts,
- il reste qu'il s'agit d'une approche itérative puisque les choix qui sont faits ont un impact direct sur les inférences que l'on peut, ou ne peut pas, effectuer.
- Les problématiques abordées sont donc similaires à celles du génie logiciel pour le développement de code informatique.

Syntaxe de Protégé

- $\top$ (Thing), $\bot$ (Nothing),
- $\exists$ (some), $\forall$ (only),
- $\neg$ (not), $\sqcup$ (or), $\sqcap$ (and),
- $\geq$ (min), $\leq$ (max),= (exactly).