

L2 - Logique

Olivier Gasquet, Jan-Georg Smaus, Martin Strecker

Université de Toulouse/IRIT

Année 2012/2013

Plan

- 1 Motivation
 - Problématique et histoire
 - Applications en informatique
- 2 Logique des propositions
- 3 Métathéorie
- 4 Logique du premier ordre
- 5 Autres méthodes de preuve

Note : Ne tentez même pas à comprendre 2, 4 et 5.

Problématique

Logique : du grecque **logos** (mot, énoncé, propos)

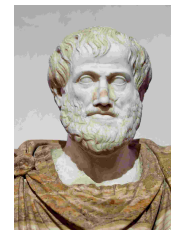
Histoire :

- Premiers développements au 4ème siècle av. J.-C.
- Scolastique médiévale : discipline essentielle (**trivium** : grammaire, rhétorique, logique)
- Refonte à la fin du 19ème siècle
- Crise existentielle au début du 20ème siècle

Objectifs de cette section :

- Connaître le développement historique de la discipline
- Apprécier ses limitations
- (sans comprendre tous les détails)

Histoire de la logique : Aristote



Aristote (-384 .. -322)

- Disciple de Platon
- Précepteur d'Alexandre le Grand
- Contributions essentielles aux sciences naturelles, l'éthique, la logique
- "Le Philosophe" par excellence de la philosophie médiévale

Histoire de la logique : Aristote

Le syllogisme d'Aristote (1)

Instance :

- **Prémisse majeure :**
Tous les hommes sont mortels
- **Prémisse mineure :**
Tous les Grecs sont des hommes
- **Conclusion :**
Tous les Grecs sont mortels

En général :

- Tous les B sont des C
- Tous les A sont des B
- Tous les A sont des C

Histoire de la logique : Aristote

Le syllogisme d'Aristote (2)

Observation essentielle :

- On peut reconnaître un argument valide par sa **forme**
- ...sans regarder la **signification** des mots

En terminologie moderne :

- On peut raisonner de manière **syntactique**
- ...sans connaissance de la **sémantique**

Histoire de la logique : Aristote

Le syllogisme d'Aristote (3)

Structure générale : deux prémisses, une conclusion

Quatre formes d'énoncé :

- Tout A est B (donc : $A \subseteq B$, pour $A \neq \{\}$)
- Aucun A n'est B (donc : $A \cap B = \{\}$, pour $A \neq \{\}$)
- Quelque A est B (donc : $A \cap B \neq \{\}$)
- Quelque A n'est pas B (donc : $A \not\subseteq B$)

Lesquels des syllogismes suivants sont valides ?

- | | |
|---------------------------|---------------------------|
| ● P_1 : Tout B est C | ● P_1 : Quelque B est C |
| ● P_2 : Quelque A est B | ● P_2 : Quelque A est C |
| ● C : Quelque A est C | ● C : Quelques A est C |

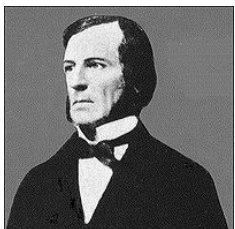
Histoire de la logique : Leibniz



Leibniz (1646 - 1716)

- Codage binaire des nombres
- Construction de l'un des premiers calculateurs (mécanique, décimal)
- Développement du calcul infinitésimal (en concurrence avec Newton)
- **Ars combinatoria** : l'art de dériver des vérités de manière **calculatoire**, basé sur
 - une **characteristica universalis**, un langage mathématique non-ambigu
 - un **calculus ratiocinator**, un calcul / une machine manipulant la **characteristica**

Histoire de la logique : Boole

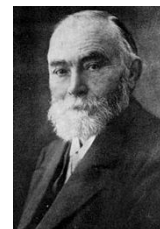


Boole (1815 - 1864)

Livre : An Investigation of the Laws of Thought

- Traitement **algébrique** de la logique propositionnelle
- (ce n'est pas "l'algèbre de Boole" actuelle !)
- Procédure de décision pour la logique propositionnelle

Histoire de la logique : Frege

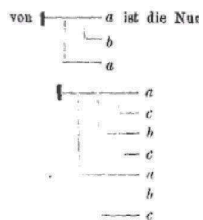


Frege (1848 - 1925)

- Fondateur de la logique "moderne" :
 - les connecteurs "essentiels" de la logique des propositions : \rightarrow et \neg
 - les quantificateurs de la logique des prédicats
 - un calcul formel

Histoire de la logique : Frege

"Begriffsschrift" de Frege (1879)



Les jugements

$\vdash a \rightarrow b \rightarrow a$
et

$\vdash (c \rightarrow b \rightarrow a) \rightarrow$
 $((c \rightarrow b) \rightarrow (c \rightarrow a))$

- Distinction entre
 - **formule** : représente une proposition (qui peut être vraie ou fausse)
 - **jugement** : formule dont on constate la vérité (dans un calcul donné)
- Notation deux-dimensionnelle qui a laissé des traces :
 - négation (\neg)
 - jugement (\vdash)

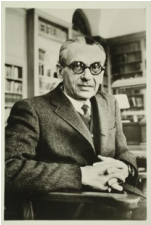
Histoire de la logique : Russel



Russel (1872 - 1970)

- Livre : **Principia Mathematica** (1910 - 1913, avec A. N. Whitehead)
But : formalisation de la mathématique à partir de quelques notions élémentaires
- Découverte d'un paradoxe (1903) dans la théorie des ensembles de Cantor
 \rightsquigarrow crise des fondements de la mathématique :
Consistance des axiomes ?
- (En plus : prix Nobel de littérature, emprisonné suite à des campagnes contre l'armement nucléaire, ...)

Histoire de la logique : Gödel



Gödel (1906 - 1978)

Le cataclysme : “**Sur des propositions indécidables de Principia Mathematica**” (1931)

- Toute théorie mathématique “suffisamment expressive” est
 - incomplète (ne peut pas prouver tout énoncé vrai)
 - **ou** contradictoire
- Surtout, elle ne permet pas de démontrer sa propre cohérence
- \rightsquigarrow échec du programme rationaliste dans la tradition Leibniz - Hilbert

Histoire de la logique - Résumé

Quelques repères dans un paysage vaste :

- Syllogisme d'Aristote : Possibilité de raisonner en s'appuyant sur la **syntaxe**, sans connaissance de la **sémantique**
- Leibniz : Raisonnement exécutable par une machine (**calcul**)
- Forme moderne de la logique (Boole, Frege)
- Le programme rationaliste mis à mal : Russel, Gödel

Conclusion : La logique

- a des limites
- ne fournit pas de certitude absolue
- mais est pourtant utile . . .

Pour plus d'information sur le développement de la logique au tournant du 20ème siècle : Jean Van Heijenoort : From Frege to Gödel

Plan

- 1 Motivation
 - Problématique et histoire
 - Applications en informatique
- 2 Logique des propositions
- 3 Métathéorie
- 4 Logique du premier ordre
- 5 Autres méthodes de preuve

Note : Il ne faut toujours pas comprendre 2, 4 et 5.

Applications de la logique

La logique a d'importantes applications en informatique :

- Intelligence artificielle
- Sûreté de fonctionnement
- Sécurité

Objectifs de cette section :

- Comprendre le rôle de la logique dans l'informatique
- Gagner une intuition des méthodes mises en œuvre

Application : Intelligence artificielle (1)

But :

- Comprendre mieux le fonctionnement de l'intelligence humaine
- Découvrir les facettes d'un comportement rationnel

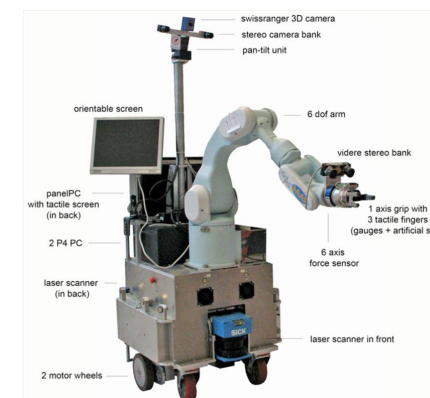
Applications :

- Simuler le comportement humain : comportement grégaire en économie (rationalité \pm réduite)
- Guider l'action humaine : prévention de conflits
- Imiter le comportement humain : robotique

Application : Intelligence artificielle (2)

Robotique :

- **Contrôle des actions :**
Comment déplacer un livre d'une chaise sur une table ?
- **Contrôle des mouvements :**
Comment se déplacer d'une salle à une autre ?
- **Interaction homme-machine :**
Comment interpréter une commande en langage naturel ?



Robot Jido (LAAS)

Application : Sûreté de fonctionnement (1)

But : S'assurer du bon fonctionnement du matériel et logiciel de systèmes critiques :

- Transport (avions, trains)
- Nucléaire
- Systèmes médicaux

Contexte :

- Systèmes embarqués de plus en plus sophistiqués
- **Exemple :** Système **fly by wire** de l'A330

à lire : Wired : History's worst software bugs



Application : Sûreté de fonctionnement (2)



Explosion de la fusée Ariane 5 (juin 1996) :

- 40 sec. après son lancement
- À bord : 4 satellites de recherche
- Dispersion de 745 tonnes de débris, y compris substances toxiques
- Coût des dégâts : 290 millions d'Euros

Contexte :

- Premier lancement de la fusée
- Lors du développement : Réutilisation de logiciel de l'Ariane 4 (autres caractéristiques de vol, accélération moins forte)
- Cause : dépassement arithmétique

Application : Sûreté de fonctionnement (3)

- Exemple du domaine médical : **Therac-25** (entre 1985 et 1987)
 - Lors d'une radiothérapie, 6 personnes reçoivent une surdose massive (facteur 100)
 - Conséquence : 3 décès
 - Causes : Multiples, surtout : problème de synchronisation de deux tâches
- Mai 2007, à Toulouse :
 - 145 personnes irradiés au CHU de Rangueil
 "c'est encore une fois une déficience de l'informatique qui serait en cause [...] L'étalonnage était mal fait [...]"
 - lire l'article dans le Parisien

Application : Sécurité

Problème :

- Des virus et vers informatiques infectent des systèmes connectés par Internet
- Exemple** : Ver "Slammer" (janvier 2003)
 - infecte 75.000 machines en 30 min
 - entre autres : blocage du système de surveillance de la centrale nucléaire Davis-Besse (Ohio) pendant 5h

Causes de la vulnérabilité :

- Langages de programmation de bas niveau (assembleur, C)
- Incompréhension du fonctionnement des protocoles de communication
- Cryptage insuffisant

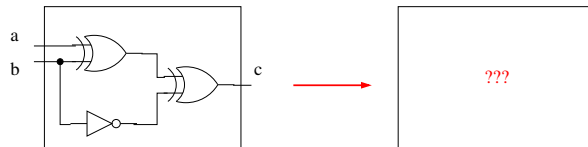
à lire : Wired : Slammer

Méthodes : Preuves d'équivalence

Circuits électroniques

Problème :

- Simplifiez le circuit suivant (table de vérité ...)
- Vérifiez que le résultat est le même



Modélisation : $(a \oplus b) \oplus (\neg b) = ???$

où : \oplus est "xor"

Remarque : Formule en **logique propositionnelle**

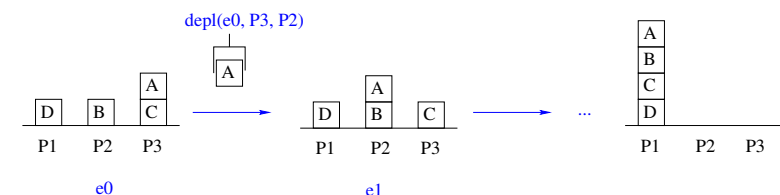
Question : Est-ce que la méthode est aussi praticable pour 1.000 variables ?

Méthodes : Systèmes à état

Robotique / Planification :

Exemple :

- Un **état** est caractérisé par des objets A, B, C, D empilés arbitrairement sur trois positions P_1, P_2, P_3
- Opérations** : Le robot peut déplacer un seul objet au sommet d'une pile vers le sommet d'une autre pile
- But** : empiler A, B, C, D (dans l'ordre) sur P_1
- Solution** : séquence d'opérations qui atteignent le but



Méthodes : Systèmes à état

Modélisation :

- Prédicat ternaire `sur` : un bloc est **sur** un autre bloc / sur une position dans un état
Exemples : `sur(D, P1, e0)` et `sur(A, B, e1)`
- Fonction ternaire `depl` : convertir un état en déplaçant le sommet d'une position vers une autre
Exemple : L'état `e1` est : `depl(e0, P3, P2)`

à faire :

- Décrivez entièrement l'état `e1`
- Trouvez la séquence d'opérations menant au but

Méthodes : Systèmes à état

Recherche d'une solution : On peut

- décrire le but par une formule logique :
"Pour tout état initial, il existe un état final ayant la propriété ..."
- faire une preuve constructive \rightsquigarrow séquence d'opérations

Remarque : Formule et preuve en logique des prédicats

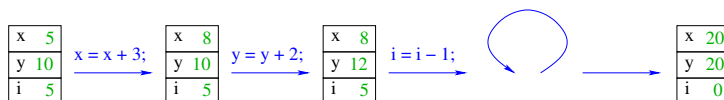
Questions plus générales :

- Comment trouver une solution pour n'importe quel état initial ?
 \rightsquigarrow stratégie de preuve
- Est-ce qu'on peut **assurer** qu'une solution existe toujours ?
 \rightsquigarrow complétude de la stratégie
- ...aussi si on a seulement deux positions ?

Méthodes : Systèmes à état

Programmes impératifs :

- Un **état** est caractérisé par un ensemble de variables et de valeurs associées.
- Opérations :
 - Affectations
 - Séquences, structures de contrôle, boucles



Méthodes : Systèmes à état

Pré- /Postconditions :

```

{ y = 2 * x ∧ x ≥ 0
  ∧ Y = y ∧ X = x }
i = x;
while (i > 0) {
  x = x + 3;
  y = y + 2;
  i = i - 1;
}
{ y = x }
  
```

 \rightsquigarrow plus de détails dans le cours "Programmation impérative"

Correction du programme :

- Est-ce que le programme transforme tout état qui satisfait la **précondition** en un état qui satisfait la **postcondition** ?
- Vérification à l'aide d'un **invariant** :
 $i \geq 0$
 $\wedge x = X + 3 * (X - i)$
 $\wedge y = Y + 2 * (X - i)$

Résumé

La logique est essentielle dans des domaines tels que

- l'intelligence artificielle
- la sûreté de fonctionnement et la sécurité

La logique permet de décrire


- l'équivalence fonctionnelle et comportementale (**exemple** : circuit)
- les propriétés des systèmes de transition :
 - **exemple planification** :
Donné : État initial, état final.
But : Recherche de séquence d'actions
 - **exemple programme impératif** :
Donné : Séquence d'actions (le programme)
But : Vérification de rapport entre un état initial et un état final

Plan

Quand on parle de “la logique”, il y a (au moins) deux dimensions :

- **Quelle** logique : logique propositionnelle, logique du premier ordre ...
- Quel **aspect** : syntaxe, sémantique, méthodes de preuves ...

Ça nous donne une “carte du terrain” :

		Lo. prop.	Lo. prem. ord	...
Syntaxe				
Sémantique				
Méth. preuve	• Déd. nat.			
	• Tableaux			
	• Résolution			
Outils	• SAToulouse			

Langages logiques (1)

Les **langages logiques** sont :

- des abstractions du langage naturel ou mathématique :
 - **Lang. nat.** : “Il fait nuit, **et** la lumière est allumée”
 - **Logique** : $n \wedge l$
 - avec : $n \equiv$ il fait nuit ; $l \equiv$ la lumière est allumée
- ...qui ne permettent pas d'exprimer certaines nuances :
 - **Lang. nat.** : “Il fait nuit **mais** la lumière est allumée”
 - **Logique** : $n \wedge l$
- ...mais plus précis :
 - **Lang. nat.** : “Je suis au bureau ou je suis dans le jardin et je lis un livre”
 - **Logique** : $(b \vee j) \wedge l$ ou $b \vee (j \wedge l)$
 - avec : $b \equiv$ je suis au bureau ; $j \equiv$ je suis dans le jardin ;
 $l \equiv$ je lis un livre

Langages logiques (2)

La **logique des propositions** LProp

- est une logique très simple
- fait une abstraction très grossière

Elle ne permet pas d'exprimer

- des rapports entre des **individus** :
“entre deux nombres, il y a un troisième nombre”
 \rightsquigarrow logique des prédicats, plus tard ...
- des rapports **temporels** :
“S’il fait nuit, la lumière sera finalement allumée”
 \rightsquigarrow logiques temporelles
- des souhaits, obligations, possibilités, ... :
“Je pense que s’il fait nuit, la lumière devrait être allumée”
 \rightsquigarrow logiques modales

Abstractions de la Lprop (1)

La logique des propositions

- remplace des **propositions** entières par des **variables propositionnelles**
il fait nuit $\rightsquigarrow n$; la lumière est allumée $\rightsquigarrow l$
- ...et les relie par des **connecteurs logiques** :
 - “Il fait nuit, **et** la lumière est allumée” : $n \wedge l$
 - “Il fait nuit, **ou** la lumière est allumée” : $n \vee l$
 - “**S**’il fait nuit, **alors** la lumière est allumée” : $n \longrightarrow l$

Abstractions de la Lprop (2)

Une **proposition** correspond à une **phrase élémentaire** en français :

- “Eve mange une grande pomme”
ou simplement : “Eve mange”

mais pas à

- un substantif : “Eve”, “une pomme”
- un verbe : “mange”
- un adjectif : “grand”

Pour avoir une correspondance juste : **réécrire** les phrases :

- **Lang. nat.** : “Eve mange une pomme et un abricot”
- **Logique** : $p \wedge a$
- avec : $p \equiv$ “Eve mange une pomme”; $a \equiv$ “Eve mange un abricot”
- mais pas(!) : $p \equiv$ “Eve mange une pomme”; $a \equiv$ “un abricot”

Abstractions de la Lprop (3)

Inversion de l'ordre des connecteurs :

- “Je vais à la plage s’il fait beau”
 \equiv “S’il fait beau, je vais à la plage”
en logique : $b \longrightarrow p$
avec : $b \equiv$ il fait beau ; $p \equiv$ je vais à la plage

Attention à l'**ambiguïté** du langage naturel :

- “Je vais au cinéma ou à la plage s’il fait beau”
 - 1 $b \longrightarrow (c \vee p)$
 - 2 $c \vee (b \longrightarrow p)$

Définition formelle de la Lprop (1)

Définition inductive : Soit $PROP$ un ensemble de **variables propositionnelles** (typiquement $\{p, q, \dots\}$).

On définit l'ensemble $FORM$ des formules par **induction**. $FORM$ est le plus petit ensemble qui satisfait les conditions :

- 1 **Variable propositionnelle** : Pour tout $p \in PROP$, $p \in FORM$
- 2 **Constante “faux”** : $\perp \in FORM$
- 3 **Négation** : Si $A \in FORM$,
alors $(\neg A) \in FORM$
- 4 **Conjonction** (“et”) : Si $A \in FORM$ et $B \in FORM$,
alors $(A \wedge B) \in FORM$
- 5 **Disjonction** (“ou”) : Si $A \in FORM$ et $B \in FORM$,
alors $(A \vee B) \in FORM$
- 6 **Implication** (“si ... alors”) : Si $A \in FORM$ et $B \in FORM$,
alors $(A \longrightarrow B) \in FORM$

(A et B représentent des chaînes de caractères, c.à.d., A et B sont des **métavariabes**).

Définition formelle de la Lprop (2)

Comprendre une définition inductive :

$((p \wedge (q \vee r)) \rightarrow (\neg s)) \in FORM$, parce que :

- $(p \wedge (q \vee r)) \in FORM$, parce que :
 - $p \in FORM$, parce que $p \in PROP$
 - $(q \vee r) \in FORM$, parce que
 - $q \in FORM$, parce que $q \in PROP$
 - $r \in FORM$, parce que $r \in PROP$
- $(\neg s) \in FORM$, parce que
 - $s \in FORM$, parce que $s \in PROP$

Définition formelle de la Lprop (3)

Comprendre une définition inductive :

"FORM est le **plus petit** ensemble ..."

- $(p \bowtie q) \notin FORM$, parce qu'aucune règle ne génère $(p \bowtie q)$
- ☕ $\notin FORM$, ☹ $\notin FORM$, ⚽ $\notin FORM$, parce que ...
- $((p \wedge (q \vee r)) \rightarrow (\neg s)) \notin FORM$, parce que il y a une parenthèse ")" en trop.
- $(A \vee p) \in FORM$ ou $(A \vee p) \notin FORM$?
 - si A représente une formule oui, sinon non

Conventions syntaxiques (1)

On peut omettre des parenthèses selon les conventions suivantes :

- **Associativité** : Les opérateurs binaires associent à droite :

- $A \wedge B \wedge C$ correspond à $(A \wedge (B \wedge C))$
- $A \rightarrow B \rightarrow C$ correspond à $(A \rightarrow (B \rightarrow C))$

Attention : $(A \rightarrow (B \rightarrow C))$ et $((A \rightarrow B) \rightarrow C)$ ont une sémantique (à voir ultérieurement) différente !

- **Priorité** : La priorité décroissante des opérateurs est $\neg, \wedge, \vee, \rightarrow$.

Exemples :

- $(A \wedge B \vee C) \equiv ((A \wedge B) \vee C)$
- $(\neg A \vee B) \equiv ((\neg A) \vee B)$
- $(A \wedge B \rightarrow A \vee B) \equiv ((A \wedge B) \rightarrow (A \vee B))$

Conventions syntaxiques (2)

On peut introduire d'autres connecteurs comme **abréviations** :

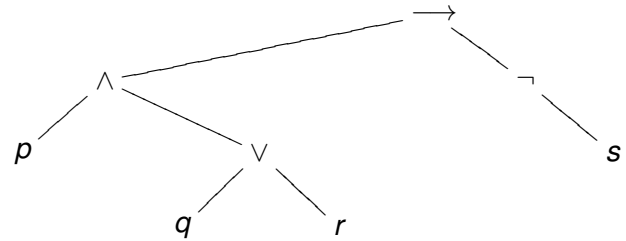
- **Constante "vrai"** : \top défini par $\neg \perp$
- **Double-implication** : $(A \leftrightarrow B)$ défini par $((A \rightarrow B) \wedge (B \rightarrow A))$
- **Ou exclusif** : $(A \oplus B)$ défini par $((A \vee B) \wedge (\neg(A \wedge B)))$

En fait, l'opérateur \neg est souvent regardé comme une abréviation : $\neg A$ défini par $A \rightarrow \perp$. Ça se clarifie quand on regardera la sémantique. Ici, les lettres majuscules A, B, C, \dots **représentent** des formules, c.à.d., les lettres A, B, C ne sont pas littéralement des formules (des variables propositionnelles) !

Arbres syntaxiques (1)

Plusieurs représentations de la même formule :

- Arbre syntaxique
- Représentation textuelle : parenthésage maximal
- Représentation textuelle : parenthésage minimal



$((p \wedge (q \vee r)) \rightarrow (\neg s))$

$p \wedge (q \vee r) \rightarrow \neg s$

Arbres syntaxiques (2)

- Il est assez compliqué de décrire formellement comment on peut transformer entre les trois représentations, pourtant intuitivement c'est assez simple.
- Ce qui compte, c'est la représentation textuelle en parenthésage minimal

Résumé

Syntaxe de la logique des propositions

- Rapport avec le langage naturel
- Définition inductive de l'ensemble *FORM*
- Différentes représentations : textuelle, arborescente

Plan

	Lo. prop.	Lo. prem. ord	...
Syntaxe	✓		
Sémantique	✗		
Méth. preuve	• Déd. nat.		
	• Tableaux		
	• Résolution		
Outils	• SAToulouse		

Problématique

Vu jusqu'à maintenant : la structure du langage logique : **Syntaxe**
(grec : **syn-taxis** : composition)

Dans cette section : la signification du langage logique : **Sémantique**
(grec : **sema** : signe)

Dichotomie syntaxe / sémantique dans tout langage :

- en logique et dans les langages de programmation
- en linguistique (signifiant/signifié \sim le mot/la chose)

Tout simplement, on va maintenant aborder la question : quand est-ce qu'une formule est **vraie** (= 1) ou **fausse** (= 0) ?

Valuation

- Normalement, on ne peut pas simplement dire qu'une formule propositionnelle est vraie ou fausse. Ça dépend en fait des **variables** de la formule.
- Une **valuation** décrit quelle valeur de vérité est associée à une variable propositionnelle
- **Formellement** : une valuation est une fonction $v : PROP \Rightarrow \{0, 1\}$
- Donc : $v(p) = 0$: " p est faux" ; $v(p) = 1$: " p est vrai" ;

Interprétation

Chaque valuation peut être étendue de *PROP* à *FORM* de la manière suivante :

- **(Variable)** : $v(p) = v(p)$
- **(Constante)** : $v(\perp) = 0$
- **(Négation)** : $v(\neg A) = 1 - v(A)$
- **(Conjonction)** : $v(A \wedge B) = \min(v(A), v(B))$
- **(Disjonction)** : $v(A \vee B) = \max(v(A), v(B))$
- **(Implication)** : $v(A \rightarrow B) = \max(1 - v(A), v(B))$

C'est une définition récursive.

On parle ainsi d'une **interprétation**.

Interprétations et tables de vérité

Table de vérité :

p	q	r	$p \wedge q$	$\neg r$	$(p \wedge q) \vee (\neg r)$
0	0	0	0	1	1
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	0	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	1	0	1

Chaque **ligne** de la table de vérité correspond à une **valuation**.

Exemple : valuation v avec : $v(p) = 1, v(q) = 0, v(r) = 0$

$$v((p \wedge q) \vee (\neg r)) = \max(v(p \wedge q), v(\neg r)) = \max(\min(v(p), v(q)), 1 - v(r)) = \max(\min(1, 0), 1 - 0) = \max(0, 1) = 1$$

Modèles

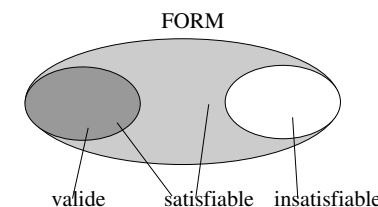
Modèle d'une formule : Une interprétation v est

- un **modèle** d'une formule A si $v(A) = 1$
On dit : v satisfait A
Ex. : $v(p) = 0, v(q) = 1$ est un modèle de $p \vee q$
- un **contre-modèle** d'une formule A si $v(A) = 0$
On dit : v falsifie A
Ex. : $v(p) = 0, v(q) = 1$ est un contre-modèle de $p \wedge q$

Modèle d'un ensemble de formules : Une interprétation v est

- un **modèle** d'un ensemble H si $v(A) = 1$ pour tout $A \in H$
Ex. : $v(p) = 0, v(q) = 1$ est un modèle de $\{p \vee q, q\}$
- un **contre-modèle** d'un ensemble H si $v(A) = 0$ pour au moins un $A \in H$
Ex. : $v(p) = 0, v(q) = 1$ est un contre-modèle de $\{p \vee q, p\}$

Validité (1)



- Une formule A est **valide** (une **tautologie**) si, pour toute valuation v , on a $v(A) = 1$
- Une formule A est **satisfiable** s'il existe une valuation v telle que $v(A) = 1$
- Une formule A est **insatisfiable** s'il n'existe pas de valuation v telle que $v(A) = 1$

Validité (2)

Déterminez si ces formules sont valides / satisfiables / insatisfiables :

- 1 $p \wedge \neg p$
- 2 $(p \vee q) \rightarrow q$
- 3 $(p \wedge q) \rightarrow p$

Complétez :

- Une formule A est **valide** si sa table de vérité ...
- Une formule A est **satisfiable** si sa table de vérité ...
- Une formule A est **insatisfiable** si sa table de vérité ...

Rapports entre les notions :

- Si A est valide, qu'est-ce que vous savez de $\neg A$?
- Si A est satisfiable, est-ce que $\neg A$ est insatisfiable ?

Conséquence (1)

Motivation : En mathématiques / informatique, on tire des **conclusions** à partir d'un ensemble d'**hypothèses**.

Exemple :

- H_1 : La fonction f est monotone
- H_2 : La fonction g est monotone
- C : La fonction $f \circ g$ est monotone

Notation : $\{H_1, H_2\} \models C$

On dit alors que C est une **conséquence (logique)** de H_1, H_2 .

Signification : Tout modèle de $\{H_1, H_2\}$ est aussi un modèle de C .

Conséquence (2)

Cas particulier : Ensemble d'hypothèses vide :

On écrit $\models C$ au lieu de $\{\} \models C$

Différence subtile entre :

- C : une formule (qui peut être vraie ou fausse)
- $\models C$: une formule dont on affirme la validité

Exercices : Vrai ou faux ?

- $\{A, B\} \models A \vee B$
- $\{A, B\} \models A \wedge B$
- $\{A, B\} \models A \rightarrow B$
- $\{A \rightarrow B, B\} \models A$
- $\{A \rightarrow B, A\} \models B$
- $\{A \rightarrow B, \neg B\} \models \neg A$

Équivalence

Deux formules A et B sont **équivalentes** si elles ont les mêmes modèles. **Notation :** $A \equiv B$

Exercices :

- Montrer que $A \wedge B \equiv B \wedge A$
- Montrer que $A \rightarrow B \not\equiv B \rightarrow A$
- Montrer que $\neg A \equiv A \rightarrow \perp$

Méta-langage : \models et \equiv ne sont pas des constructeurs de formules !

Interdit : "formules" comme $(A \models B) \equiv (A \rightarrow B)$

Résumé

- Différence syntaxe / sémantique
- On définit la **sémantique** d'une formule à l'aide d'une fonction d'**interprétation**
- Correspondance entre interprétations et tables de vérité
- Modèles d'une formule
- Validité, satisfiabilité
- Conséquence, équivalence logique

Plan

		Lo. prop.	Lo. prem. ord	...
Syntaxe		✓		
Sémantique		✓		
Méth. preuve	• Déd. nat.	✗		
	• Tableaux			
	• Résolution			
Outils	• SAToulouse			

Introduction

On s'intéresse désormais à des méthodes pour déterminer la **validité** d'une formule A .

Méthodes possibles :

Construction de la table de vérité

- **Question :** Quelle est la taille de la table pour une formule avec n variables propositionnelles ?
- \sim Méthode non praticable si n est "grand"

Méthode astucieuse : recherche plus ciblée :

- Méthode directe : Vérifie que la formule est **valide**
Déduction naturelle

Vraiment "astucieuse" ?

- **Oui :** On peut **souvent** éviter un coût exponentiel
- **Non :** On ne sait pas **toujours** l'éviter (à l'heure actuelle)

Autres méthodes plus tard ...

Validité et Prouvabilité

Validité : $\{H_1, \dots, H_n\} \models C$

- une notion **sémantique**
- définie à l'aide d'**interprétations** :
toute interprétation qui satisfait $\{H_1, \dots, H_n\}$ satisfait aussi C
- voir définitions de validité et conséquence

Prouvabilité : $\{H_1, \dots, H_n\} \vdash C$

- une notion **syntactique**
- relative à un **calcul** : avec les règles du calcul, on peut déduire C à partir des hypothèses $\{H_1, \dots, H_n\}$
- (ici, le calcul est la **déduction naturelle**)

Critères pour un "bon" calcul :

- **Correction :** si $\{H_1, \dots, H_n\} \vdash C$, alors $\{H_1, \dots, H_n\} \models C$
- **Complétude :** si $\{H_1, \dots, H_n\} \models C$, alors $\{H_1, \dots, H_n\} \vdash C$

Déduction naturelle : exemple (1)

Exemple de raisonnement "naturel"

Quelques propositions :

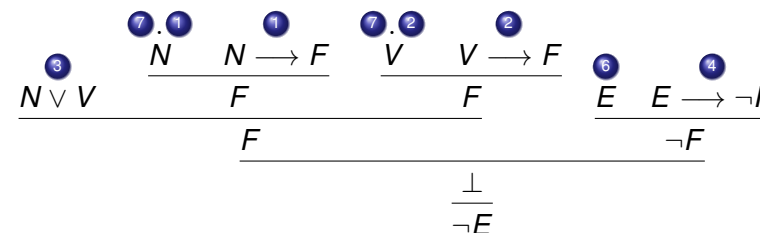
- 1 Quand il neige, il fait froid : $(N \rightarrow F)$
- 2 Quand il y a du verglas, il fait froid : $(V \rightarrow F)$
- 3 Il neige ou il y a du verglas $(N \vee V)$
- 4 En été, il ne fait pas froid $(E \rightarrow \neg F)$
- 5 Donc, il n'est pas été : $\neg E$

Dérivation de 5 à partir de 1 ... 4 :

- 6 Pour montrer 5, supposons E , et dérivons une contradiction (\perp)
- 7 Distinction de cas (avec 3) :
 - 1 Soit N . Alors, avec 1, on a F .
 - 2 Soit V . Alors, avec 2, on a F .
 Donc, de 1, 2, 3 on peut conclure F .
- 8 Avec 6 et 4, inférer $\neg F$.
- 9 7 et 8 permettent de conclure \perp , comme demandé dans 6.

Déduction naturelle : exemple (2)

Arbre de dérivation :



Nous dirons : Sous les hypothèses 1 ... 4, on peut **déduire** ou **dérivé** la **conclusion** 5.

Ce fait est exprimé par le **jugement**

$\{N \rightarrow F, V \rightarrow F, N \vee V, E \rightarrow \neg F\} \vdash \neg E$

Histoire : Hilbert



Hilbert (1862 - 1943)

- Contributions essentielles aux fondements de la mathématique :
 - Axiomatisation de la géométrie Euclidienne
 - ... et dans la suite : axiomatisation de la mathématique
- "Programme de Hilbert" :
 - Trouver un système d'axiomes et règles de déduction qui assurent la consistance de la mathématique
 - (mis à mal par les résultats de Gödel)

Histoire : Gentzen



Gentzen (1909 - 1945)

- Développement de deux calculs (1934) :
 - Déduction naturelle**, pour rendre l'axiomatisation de Hilbert plus utilisable
 - Calcul des séquents** : une procédure **effective** de preuve
 - Les deux calculs sont équivalents ("élimination des coupures")
- Méthode des tableaux :
 - Variante du calcul des séquents
 - ... introduite vers 1955 par E.W.Beth

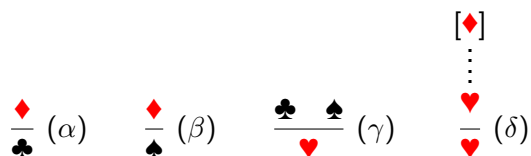
Voir article The Development of Proof Theory

Déduction naturelle : un exemple abstrait

Nous avons déjà donné un exemple qui était très **intuitif** (neige, verglas ...).

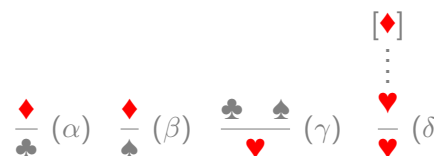
Pour mieux apprécier que la déduction est un processus entièrement mécanique et que nous ne pouvons pas utiliser notre intuition sémantique comme bon nous semble, il est utile de présenter un exemple **abstrait**, sans aucune intuition.

- Langage $\mathcal{L} = \{\heartsuit, \clubsuit, \spadesuit, \diamondsuit\}$.
- Système déductif donné par les **règles** :

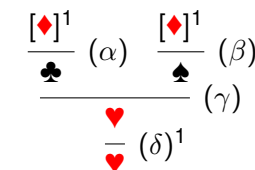


Preuve de ♥

Les règles :



Le preuve :



On fait une **hypothèse**. Pour l'instant, elle est **ouverte**

On applique α .

De la même façon avec β .

On applique γ .

On applique δ , en déchargeant les deux occurrences de \diamond . On met une étiquette à la règle et aux parenthèses pour les associer. Comme ça, la dérivation est devenue une **preuve** : il n'y a plus d'hypothèses ouvertes.

Règles d'inférence (1)

Une règle

$$\frac{C^1 \quad \dots \quad C^m}{C}$$

est composée de

- 0, 1 ou plusieurs **antécédents**
- un seul **conséquent**

Lecture informelle :

“Si tous les antécédents sont prouvables, alors aussi le conséquent”

Règles d'inférence (2)

Il y a aussi des règles qui contiennent une ou plusieurs **dérivations hypothétiques** :

$$\frac{\begin{array}{c} [A] \\ \vdots \\ B \end{array} \quad \dots \quad \dots}{C}$$

Lecture informelle :

“Si B est prouvable en utilisant A , alors C est prouvable”

On peut ensuite **décharger** la hypothèse A .

Typologie des règles : Règles pour \wedge

Pour chaque connecteur, il y a une (éventuellement deux)

- **règle(s) d'introduction** (connecteur dans le conséquent), par exemple $(I\wedge)$
- **règle(s) d'élimination** (connecteur dans l'antécédent **principal**), par exemple $(E\wedge_1)$ et $(E\wedge_2)$

- Règle d'introduction :

$$\frac{A \quad B}{A \wedge B} (I\wedge)$$

- Règles d'élimination :

$$\frac{A \wedge B}{A} (E\wedge_1) \quad \frac{A \wedge B}{B} (E\wedge_2)$$

Règles pour \rightarrow

- Intro :

$$\frac{\begin{array}{c} [A] \\ \vdots \\ B \end{array} \quad A \rightarrow B}{A \rightarrow B} (I\rightarrow)$$

- Elim :

$$\frac{A \rightarrow B \quad A}{B} (E\rightarrow)$$

Note : $(E\rightarrow)$ aussi appelée **modus ponens**

Règles pour \neg

- Intro :

$$\frac{\begin{array}{c} [A] \\ \vdots \\ \perp \end{array}}{\neg A} (I\neg)$$

- Elim :

$$\frac{\neg A \quad A}{\perp} (E\neg)$$

Observation ?

Ce sont des cas spéciaux des règles pour \longrightarrow , en considérant que $\neg A$ n'est qu'une abbréviatiion pour $A \longrightarrow \perp$.

Règles pour \perp

- Intro ? Il n'y en a pas !!!
- Elim :

$$\frac{\perp}{A} (E\perp)$$

Note : $(E\perp)$ aussi appelée **ex falso quodlibet**

Règles pour \vee (1)

$$\frac{A}{A \vee B} (I\vee_1) \quad \frac{B}{A \vee B} (I\vee_2)$$

$$\frac{\begin{array}{cc} [A] & [B] \\ \vdots & \vdots \\ A \vee B & C \end{array}}{C} (E\vee)$$

Note : $(E\vee)$ correspond à une distinction de cas.

Logique intuitionniste et classique

Nous avons maintenant des règles pour \wedge , \rightarrow , \neg , \perp , et \vee . Est-ce que c'est suffisant pour prouver **toutes les tautologies** de la logique propositionnelle ?

Considérez $\neg(A \wedge B) \rightarrow \neg A \vee \neg B$. On ne peut pas la prouver !

Les règles que nous avons introduit jusqu'à maintenant définissent la logique **intuitionniste**. Pour avoir la logique **classique**, il faut ...

Double négation

$$\frac{\neg\neg A}{A} (\neg\neg)$$

Dans la littérature, on trouve deux autres formulations :

$$\frac{}{A \vee \neg A} (TND) \quad (\text{tertium non datur})$$

et

$$\frac{[\neg A] \dots A}{A} (clas.)$$

Toutes ces formulations sont équivalentes ...

De $(\neg\neg)$ à $(clas.)$

$$\frac{[\neg A] \dots \frac{A}{\neg\neg A} (I\neg)}{\frac{}{A} (\neg\neg)} (E\neg)$$

De $(clas.)$ à (TND)

$$\frac{[\neg(A \vee \neg A)]^2 \quad \frac{[A]^1}{A \vee \neg A} (I\vee_1)}{\neg A} (E\neg) \quad \frac{}{\neg A} (I\neg)^1 \quad \frac{A \vee \neg A}{A \vee \neg A} (I\vee_2) \quad (clas.)^2$$

De (TND) à $(\neg\neg)$

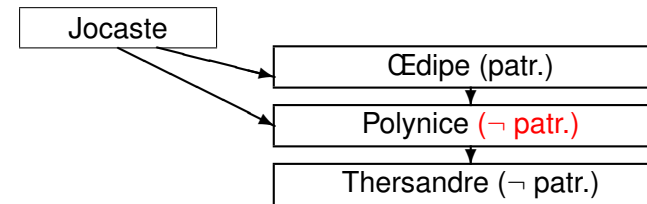
$$\frac{\frac{}{A \vee \neg A} (TND) \quad [A] \quad \frac{\neg\neg A \quad [\neg A]}{A} (E\neg)}{A} (E\vee)$$

Les trois formulations sont donc équivalentes et à partir de maintenant, vous êtes autorisés d'utiliser celle que vous semble plus apte.

Exemple du raisonnement classique

- Locaste est la mère d'Œdipe.
- Jocaste et Oedipe sont les parents de Polynice.
- Polynice est le père de Thersandre.
- Œdipe est un patricide
- Thersandre n'est pas un patricide.

Exemple du raisonnement classique (2)



Est-ce que Jocaste a un fils qui est un patricide qui lui-même a un fils qui n'est pas un patricide ?

Cas 2 : Si Polynice n'est pas un patricide, alors Jocaste a un fils (Polynice) qui est un patricide et qui lui-même a un fils (Thersandre) qui n'est pas un patricide.

Preuve de $\neg(A \wedge B) \rightarrow \neg A \vee \neg B$

$$\begin{array}{c}
 \frac{\frac{\frac{A \vee \neg A}{(clas.)} \quad \frac{\frac{\frac{[\neg(A \wedge B)]^3}{\perp} \quad \frac{[A]^2 \quad [B]^1}{A \wedge B} (I \wedge)}{\neg B} (I \neg)^1}{\neg A \vee \neg B} (I \vee_2)}{\neg A \vee \neg B} (E \vee)^2 \quad \frac{[\neg A]^2}{\neg A \vee \neg B} (I \vee_1)}{\neg(A \wedge B) \rightarrow \neg A \vee \neg B} (I \rightarrow)^3
 \end{array}$$

Règles dérivées :

Règles pour \leftrightarrow : On peut définir $A \leftrightarrow B$ comme : $(A \rightarrow B) \wedge (B \rightarrow A)$

Dérivez les règles $(I \leftrightarrow)$:

$$\frac{\begin{array}{c} [A] \\ \vdots \\ B \end{array} \quad \begin{array}{c} [B] \\ \vdots \\ A \end{array}}{A \leftrightarrow B} (I \leftrightarrow)$$

et $(E \leftrightarrow_1)$, $(E \leftrightarrow_2)$:

$$\frac{A \leftrightarrow B}{A \rightarrow B} (E \leftrightarrow_1) \quad \frac{A \leftrightarrow B}{B \rightarrow A} (E \leftrightarrow_2)$$

Jugement

Quand on peut construire un arbre de déduction avec les règles ci-dessus, dont les hypothèses (toujours ouvertes) sont H_1, \dots, H_n est dont la conclusion est C , alors on a le droit d'écrire le **jugement** $\{H_1, \dots, H_n\} \vdash C$.

Note : en fait, il est possible de présenter la déduction naturelle autour des jugements (**notation explicite**).

Correction et Complétude

On ne le prouve pas ici, mais :

- La déduction naturelle avec les règles ci-dessus est correcte pour LProp : si $\{H_1, \dots, H_n\} \vdash C$, alors $\{H_1, \dots, H_n\} \models C$
- La déduction naturelle avec les règles ci-dessus est complète pour LProp : si $\{H_1, \dots, H_n\} \models C$, alors $\{H_1, \dots, H_n\} \vdash C$

On abordera cette question plus en détail pour un autre calcul plus tard ...

Déduction Naturelle : Résumé

Principes :

- Motivation : Modéliser le raisonnement humain
- Structure de base : Arbre de déduction
 - **Lecture** : De haut en bas (antécédent \rightsquigarrow conséquent)
 - **Construction** : De bas en haut

Règles :

- ...d'introduction : connecteur dans le conséquent
- ...d'élimination : connecteur dans l'antécédent

Subtil :

- Décharge = modification des hypothèses en vigueur dans un sous-arbre
- Difficile à automatiser (pas de **propriété de sous-formule**, il faut inventer des formules)

Plan

		Lo. prop.	Lo. prem. ord	...
Syntaxe		✓		
Sémantique		✓		
Méth. preuve	• Déd. nat.	✓		
	• Tableaux			
	• Résolution			
Outils	• SAToulouse	✗		

Un outil pédagogique

SAToulouse a été conçu par l'équipe pédagogique du module "Logique" pour

- avoir un format simple d'écriture de formules
- permettre la vérification de satisfiabilité d'une formule en un clic
- donner un modèle lisible d'une formule satisfiable

Autres outils (voir liens sur la page Moodle) :

- Chaff : expressif, mais difficile à apprendre
- Minisat, Sat4j : format "assembleur" de bas niveau, difficile à comprendre

Le pourquoi et comment de SAToulouse

But : montrer la satisfiabilité d'une formule F

- Si F est satisfiable, SAToulouse fournit un modèle (variables interprétées comme "vraies", les autres étant "fausses")
- Sinon, SAToulouse affiche "insatisfiable"

Démarche :

- 1 écrire une ou plusieurs formules dans le panneau central (SAToulouse prend la **conjonction** des lignes)
- 2 Appuyer sur le "cerveau" pour vérifier la satisfiabilité
- 3 Eventuellement : Raffiner en rajoutant d'autres formules, et recommencer (pour obtenir un autre modèle)

Syntaxe de SAToulouse

- Une syntaxe textuelle, inspirée de Lisp
- Une syntaxe \LaTeX , à composer avec l'éditeur syntaxique à gauche

Correspondances :

Textuel	\LaTeX
p	p
(p i)	p_i
(p (i + 1))	p_{i+1}
(p i j)	$p_{i,j}$
(A and B)	$(A \wedge B)$
(A or B)	$(A \vee B)$
(bigand i (1 2 3) (p i))	$\bigwedge_{i \in \{1,2,3\}} p_i$
(bigor i (1 2 3) (p i))	$\bigvee_{i \in \{1,2,3\}} p_i$
(bigand i (1 2 3) (i diff j) (p i j))	$\bigwedge_{i \in \{1,2,3\} i \neq j} p_{i,j}$

Limites (actuelles) de SAToulouse

- **Opérations sur indices** : limitées à des sommes d'indices et constantes :
 p_{i+j} , q_{i+3} , mais pas : q_{2*i-5}
- **Contraintes** : limitées à des inégalités :
 $i \neq j$, $i \neq j + 2$, mais pas : $i \geq j$

Message publicitaire : Intéressé(e) par un stage pour étendre SAToulouse ?

Prenez contact avec nous pour en discuter !

Plan

- 1 Motivation
- 2 Logique des propositions
- 3 **Métathéorie**
- 4 Logique du premier ordre
- 5 Autres méthodes de preuve

Où sommes-nous ?

- Nous savons écrire des formules (propositionnelles) : **syntaxe**
- Nous savons ce que ça veut dire qu'une formule est vraie ou fausse : **sémantique**
- Nous connaissons une méthode astucieuse (le déduction naturelle) pour dériver qu'une formule est valide : **méthode de preuve**

Maintenant, au lieu de regarder toujours une formule à la fois comme auparavant, nous allons étudier le formalisme de la logique propositionnelle de manière plus générale : **métathéorie**

- On ne peut pas dire que ce qu'on fera maintenant soit "syntaxe" ou "sémantique", par contre ça fait souvent la liaison entre les deux.
- Nous traitons toujours la logique propositionnelle, mais plutôt comme exemple ; les techniques et concepts qu'on regarde ici sont en fait beaucoup plus générales que ça (donc : nouvelle section).

Plan

	Lo. prop.	Lo. prem. ord	...
Syntaxe	✓ 		
Sémantique	✓ 		
Méth. preuve <ul style="list-style-type: none"> • Déd. nat. • Tableaux • Résolution 	✓		
Outils <ul style="list-style-type: none"> • SAToulouse 	✓		

Plan

- 1 Motivation
- 2 Logique des propositions
- 3 **Métathéorie**
 - Induction
 - Substitutions
 - Un lien entre syntaxe et sémantique
- 4 Logique du premier ordre
- 5 Autres méthodes de preuve

Induction : histoire



Maurolico (1494-1575)

Première preuve par induction



Peano (1858 - 1932)

Première axiomatisation des nombres naturels

Triade

Nous verrons une “triade” :

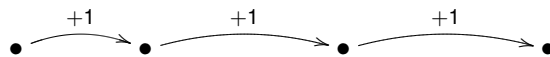
- définition inductive d'un ensemble
- fonctions récursives
- preuves par induction

Définition inductive : nombres naturels

Les nombres naturels comme ensemble inductif

 Nat est le plus petit ensemble qui satisfait les conditions :

- 1 Zéro : $0 \in Nat$
- 2 Successeur : Si $n \in Nat$, alors $suc(n) \in Nat$

Note : $suc(n)$ est une autre manière d'écrire $n + 1$ 

repr. décimale 0 1 2 3

repr. unaire 0 $suc(0)$ $suc(suc(0))$ $suc^3(0)$

Récursion : nombres naturels

Récursion sur les nombres naturels

Exemple : La fonction d'addition $n + m$ peut être définie par :

- cas Zéro [eq. Z] : $0 + m = m$
- cas Successeur [eq. S] : $suc(n) + m = suc(n + m)$

Preuves par induction : nombres naturels

Schéma d'induction sur les nombres naturels :

- **base (Zéro)** : Si $P(0)$,
- **hérédité (Successeur)** : et si pour tout n , $P(n)$ implique $P(\text{succ}(n))$
- alors on peut conclure $\forall n. P(n)$

Preuves par induction : exemple

Montrer que 0 est l'élément neutre à droite : $\forall n. (n + 0 = n)$

- 1 Identifier le prédicat P : Ici : $P(n) \equiv (n + 0 = n)$
- 2 Preuve pour le cas de base : Montrer : $P(0)$, donc : $0 + 0 = 0$ (par [eq. Z])
- 3 Preuve d'hérédité : Montrer : si $P(n)$, alors $P(\text{succ}(n))$
 - Hypothèse d'induction [eq. H] : $n + 0 = n$
 - Montrer : $\text{succ}(n) + 0 = \text{succ}(n)$. Calculer :
 - $\text{succ}(n) + 0 = \text{succ}(n + 0)$ (par [eq. S])
 - $= \text{succ}(n)$ (par [eq. H])

Exercice : Montrer : l'addition est commutative : $\forall n. \forall m. (n + m = m + n)$

Retenir :

- $+$ est définie par **réursion** sur le 1er argument
- donc : **induction** sur le 1er argument de $+$

Définition inductive : formules

Rappel : définition inductive de $FORM$ (voir définition page 7) : $FORM$ est le plus petit ensemble qui satisfait les conditions :

- 1 **Variable propositionnelle** : Pour tout $p \in PROP$, $p \in FORM$
- 2 **Constante "faux"** : $\perp \in FORM$
- 3 **Négation** : Si $A \in FORM$, alors $(\neg A) \in FORM$
- 4 **Conjonction** ("et") : Si $A \in FORM$ et $B \in FORM$, alors $(A \wedge B) \in FORM$
- 5 **Disjonction** ("ou") : Si $A \in FORM$ et $B \in FORM$, alors $(A \vee B) \in FORM$
- 6 **Implication** ("si ... alors") : Si $A \in FORM$ et $B \in FORM$, alors $(A \rightarrow B) \in FORM$

Récursion : formules

Exemple : La fonction nbp (nombre de parenthèses) est définie par :

- **Variable** [eq. V] : $\text{nbp}(p) = 0$
- **Constante** [eq. C] : $\text{nbp}(\perp) = 0$
- **Négation** [eq. N] : $\text{nbp}(\neg A) = \text{nbp}(A) + 2$
- **Conjonction** [eq. C] : $\text{nbp}(A \wedge B) = \text{nbp}(A) + \text{nbp}(B) + 2$
- **Disjonction** [eq. D] : $\text{nbp}(A \vee B) = \text{nbp}(A) + \text{nbp}(B) + 2$
- **Implication** [eq. I] : $\text{nbp}(A \rightarrow B) = \text{nbp}(A) + \text{nbp}(B) + 2$

Quelques formules :

- $\text{nbp}(((p \vee q) \wedge r)) = 4$
- $\text{nbp}(((p \wedge q) \vee (\neg(r \vee \perp)))) = 8$

Preuves par induction : formules

Schéma d'induction sur les formules :

- **base (Variable)** : Si $P(p)$
- **base (Constante)** : et si $P(\perp)$
- **hérédité (Négation)** : et si pour toute formule A , $P(A)$ implique $P(\neg A)$
- **hérédité (Conjonction)** : et si pour tout A, B , $P(A)$ et $P(B)$ implique $P(A \wedge B)$
- **hérédité (Disjonction)** : et si pour tout A, B , $P(A)$ et $P(B)$ implique $P(A \vee B)$
- **hérédité (Implication)** : et si pour tout A, B , $P(A)$ et $P(B)$ implique $P(A \rightarrow B)$
- alors on peut conclure : $\forall A \in \text{FORM}. P(A)$

Preuve par induction : exemple

Preuve par induction Exemple : Montrer que le nombre de parenthèses est toujours pair : $\forall f. \text{nbp}(f) \bmod 2 = 0$

- **Identifier le prédicat P** : Ici : $P(A) \equiv (\text{nbp}(A) \bmod 2 = 0)$
- **Cas de base (Variable)** : Montrer : $P(p)$, donc : $\text{nbp}(p) \bmod 2 = 0 \bmod 2 = 0$ (par [eq. V])
- **Cas de base (Constante)** : Montrer : $P(\perp)$, donc : $\text{nbp}(\perp) \bmod 2 = 0 \bmod 2 = 0$ (par [eq. C])
- **Hérédité (Négation)** : Montrer : si $P(A)$, alors $P(\neg A)$
 - Hypothèse d'induction [eq. H] : $\text{nbp}(A) \bmod 2 = 0$
 - Montrer : $\text{nbp}(\neg A) \bmod 2 = 0$. Calculer :
 - $\text{nbp}(\neg A) \bmod 2 = (\text{nbp}(A) + 2) \bmod 2$ (par [eq. N])
 - $= \text{nbp}(A) \bmod 2$ (arithmétique)
 - $= 0$ (par [eq. H])

Preuve par induction : exemple (cont.)

- **Hérédité (Conjonction)** : Montrer : si $P(A)$ et $P(B)$, alors $P(A \wedge B)$
 - Hypothèses d'induction
 - [eq. H_1] : $\text{nbp}(A) \bmod 2 = 0$
 - [eq. H_2] : $\text{nbp}(B) \bmod 2 = 0$
 - Montrer : $\text{nbp}(A \wedge B) \bmod 2 = 0$. Calculer :
 - $\text{nbp}(A \wedge B) \bmod 2 = (\text{nbp}(A) + \text{nbp}(B) + 2) \bmod 2$ (par [eq. C])
 - $= (\text{nbp}(A) \bmod 2 + \text{nbp}(B) \bmod 2) \bmod 2$ (arithmétique)
 - $= (0 + \text{nbp}(B) \bmod 2) \bmod 2$ (par [eq. H_1])
 - $= (0 + 0) \bmod 2$ (par [eq. H_2])
 - $= 0$ (arithmétique)
- **Hérédité (Disjonction et Implication)** : Similaire

Induction : résumé

La Triade :

- **ensembles inductifs** :
 - générés à partir de **cas de base**,
 - en appliquant des **règles de construction**
- **Fonctions récursives** (récursion structurelle) :
 - décomposent une structure inductive composée
 - s'arrêtent dans les cas de base
 ...et synthétisent un résultat en remontant
- **Preuves par induction structurelle** : Permettent de conclure qu'une propriété est satisfaite pour **tout** élément d'un type inductif
 - si elle est satisfaite pour les cas de base
 - si elle est héréditaire pour les éléments composés

Plan

- 1 Motivation
- 2 Logique des propositions
- 3 Métathéorie
 - Induction
 - Substitutions
 - Un lien entre syntaxe et sémantique
- 4 Logique du premier ordre
- 5 Autres méthodes de preuve

Substitutions (1)

Une substitution remplace **toutes** les occurrences d'une variable propositionnelle p dans une formule A par une formule B .

Notation : $A[B/p]$

Exemple :

$$\begin{aligned}
 & (q \wedge (p \rightarrow (\neg p)))[(r \vee s)/p] \\
 = & (q[(r \vee s)/p] \wedge (p \rightarrow (\neg p))[(r \vee s)/p]) \\
 = & (q \wedge (p[(r \vee s)/p] \rightarrow (\neg p)[(r \vee s)/p])) \\
 = & (q \wedge ((r \vee s) \rightarrow (\neg p[(r \vee s)/p]))) \\
 = & (q \wedge ((r \vee s) \rightarrow (\neg(r \vee s))))
 \end{aligned}$$

Substitutions (2)

Contre-exemples (dessinez les arbres syntaxiques !) :

- Pourquoi $(\neg p)[(r \vee s)/p] \neq (\neg r \vee s) ? ?$
- Pourquoi $(p \rightarrow (\neg p))[(r \vee s)/p] \neq ((r \vee s) \rightarrow (\neg p)) ? ?$

Exercices :

- Donner une définition (par récursion sur A) de : $A[B/p]$
- Définir une fonction $nbocc(p, A)$ qui calcule le nombre d'occurrences d'une variable propositionnelle p dans une formule A
- Si $nbocc(p, A) = 0$, alors $A[B/p] = ???$ (**preuve !**)
- Définir une fonction $taille(A)$ qui calcule la taille d'une formule (\equiv nombre de nœuds de l'arbre syntaxique)
- Quel est le rapport entre $taille(A)$, $taille(B)$, $nbocc(p, A)$ et $taille(A[B/p])$?

Plan

- 1 Motivation
- 2 Logique des propositions
- 3 Métathéorie
 - Induction
 - Substitutions
 - Un lien entre syntaxe et sémantique
- 4 Logique du premier ordre
- 5 Autres méthodes de preuve

Un lien entre syntaxe et sémantique

- La **substitution** est une notion **syntactique**.
- L'**équivalence** est une notion **sémantique** : deux formules A et B sont **équivalentes** si elles ont les mêmes modèles.
- Nous disposons maintenant des outils pour prouver un lien intéressant ...

Équivalence

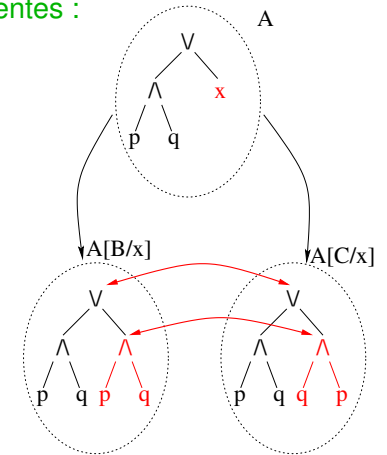
Rappel : $A[B/p]$

Substitution d'une formule B pour une variable p dans une formule A

Remplacement de sous-formules équivalentes :

Exemple : Soient

- $B := (p \wedge q)$, $C := (q \wedge p)$
- $A := (p \wedge q) \vee x$
- donc : $A[B/x] = (p \wedge q) \vee (p \wedge q)$
- donc : $A[C/x] = (p \wedge q) \vee (q \wedge p)$



Proposition : Si $B \equiv C$,

alors $A[B/x] \equiv A[C/x]$

Preuve : Par induction sur la structure de A

Complétez !

Plan

- 1 Motivation
- 2 Logique des propositions
- 3 Métathéorie
- 4 Logique du premier ordre
 - Syntaxe (langage)
 - Sémantique (théorie des modèles)
 - Méthode de preuve : déduction naturelle
- 5 Autres méthodes de preuve

Plan

	Lo. prop.	Lo. prem. ord	...
Syntaxe	✓	✗	
Sémantique	✓		
Méth. preuve	• Déd. nat.	✓	
	• Tableaux		
	• Résolution		
Outils	• SAToulouse	✓	

Motivation (1)

La logique des **propositions** est limitée en expressivité.

- elle ne permet pas de parler d'individus :
"Si l'objet $o1$ est bleu, alors $o1$ est aussi rond"
 - Tentative de modélisation : $B \rightarrow R$
Problème : Traduire "Si $o1$ est bleu, alors $o1$ est rond et $o2$ carré"
 - Tentative de modélisation : $B1 \rightarrow R1 \wedge C2$
Problème : Différence structurelle avec $X \rightarrow R1 \wedge C2$ ou $B2 \rightarrow R1 \wedge C2$?
- elle n'est pas concise (voir quantif. sur ensembles finis) :
"Tout objet bleu est rond" (on connaît les objets $o1, o2, o3$) :
 $(B1 \rightarrow R1) \wedge (B2 \rightarrow R2) \wedge (B3 \rightarrow R3)$
- elle ne permet pas de parler d'une infinité d'individus :
"Tout objet bleu est rond"
- elle ne permet pas de parler d'actions / opérations :
"Tout objet repeint deux fois apparaît neuf"

Motivation (2)

La logique des **prédicats** LPred raffine la logique des **propositions**.
Individus et prédicats

- Langage naturel : "Si l'objet $o1$ est bleu, alors $o1$ est aussi rond"
- LProp : $B1 \rightarrow R1$, où
 - $B1, R1$ sont des phrases élémentaires
- LPred : $B(o1) \rightarrow R(o1)$, où
 - $o1$ est le **sujet** de la phrase : "l'objet $o1$ "
 - B est le **prédicat** : "est bleu"
 - On garde les connecteurs de LProp (\rightarrow)

Motivation (3)

Quantificateurs

- Langage naturel : "Tout objet bleu est rond"
- LProp : Pas exprimable en général
- LPred : $\forall x.(B(x) \rightarrow R(x))$

Opérations / Fonctions

- Langage naturel : "Tout objet repeint deux fois apparaît neuf"
- LProp : Pas exprimable en général
- LPred : $\forall x.N(r(r(x)))$, où
 - x est un individu
 - r est une fonction qui transforme des individus ("repeindre")
 - N est un prédicat

Syntaxe de LPred : Termes et formules

Dans la logique propositionnelle, les expressions syntaxiques étaient les **formules** (peuvent être vraies ou fausses).

Dans la logique des prédicats, nous avons deux **catégories syntaxiques** : les **termes** et les **formules**. Un terme représente un individu.

Syntaxe de LPred : Termes

Termes : Soit

- VAR un ensemble de variables d'individus
- FON_n un ensemble des fonctions n -aires

L'ensemble $TERM$ des termes est défini par induction comme le plus petit ensemble qui satisfait :

- 1 **Variable** : si $x \in VAR$, alors $x \in TERM$
- 2 **Application de fonction** : si $t_1 \in TERM, \dots, t_n \in TERM$ et $f \in FON_n$, alors $f(t_1, \dots, t_n) \in TERM$

On appelle des éléments de FON_0 des **constantes**.
Souvent, on écrit c au lieu de $c()$.

Exemples : Soient $f \in FON_2$, $x, y \in VAR$, $g \in FON_1$, $\pi \in FON_0$

- $f(x, c) \in TERM$
- $f(x, g(f(y, \pi))) \in TERM$

Syntaxe de LPred : Formules

Formules : Soit $PRED_n$ un ensemble des prédicats n -aires

L'ensemble $FORM$ des formules est défini par induction comme le plus petit ensemble qui satisfait :

- 1 **Application de prédicat** : si $t_1 \in TERM, \dots, t_n \in TERM$ et $P \in PRED_n$, alors $P(t_1, \dots, t_n) \in FORM$
- 2 **Constante "faux"** : $\perp \in FORM$
- 3 **Négation** : Si $A \in FORM$, alors $(\neg A) \in FORM$
- 4 **Connecteurs binaires** : Si $A \in FORM$ et $B \in FORM$, alors $(A \wedge B) \in FORM, (A \vee B) \in FORM, (A \rightarrow B) \in FORM$
- 5 **Quantificateur universel** ("pour tout") : Si $A \in FORM$ et $x \in VAR$, alors $(\forall x. A) \in FORM$
- 6 **Quantificateur existentiel** ("il existe") : Si $A \in FORM$ et $x \in VAR$, alors $(\exists x. A) \in FORM$

à noter : $FORM$ dépend de $TERM$, mais pas inversement

Fonctions et Prédicats

Utilisation : Fonctions et prédicats traduisent des **verbes**.

Les **fonctions** décrivent la transformation d'un ou plusieurs objet(s) :

- "une tarte faite de farine et de pommes" : $fairetarte(farine, pomme)$

Les **prédicats** décrivent un état / une situation :

- "Eve achète une pomme" : $Acheter(eve, pomme)$
- "Eve mange une tarte de pomme" :
 $Manger(eve, fairetarte(farine, pomme))$

Un prédicat ne peut pas prendre des formules comme arguments :

"Eve mange les pommes qu'elle achète" :

- **Faux** : $Manger(eve, Acheter(eve, pomme))$
- **Correct** : $\forall p. Acheter(eve, p) \rightarrow Manger(eve, p)$

Convention : nom de fonctions : minuscules ; de prédicats : majuscules

Limites de l'expressivité

Contournables : par exemple : propriétés temporelles :

- "Si le composant tombe en panne, il émet éventuellement un signal"
- Peut être codé par :
 $\forall t. Panne(c, t) \rightarrow (\exists t'. t' > t \wedge Signal(c, t'))$

Essentielles :

- Utilisation des fonctions comme objets :
 - Quantification sur des fonctions : "Toute fonction a une inverse" :
Interdit : $\forall f. \exists g. \forall x. f(g(x)) = x$
 - Fonctions qui prennent des fonctions comme argument :
Interdit : $\forall f. \forall x. f(inv(f, x)) = x$
- De même : Quantification sur des prédicats ou ensembles

\rightsquigarrow inapproprié pour raisonner sur certains programmes fonctionnels

Variables libres et variables liées

- Toute occurrence d'une variable dans une formule est **liée** ou **libre** ou **liante**.
- Exemple :

$$(Q(x) \vee \exists x. \forall y. P(f(x), z) \wedge Q(y)) \vee \forall x. R(x, z, g(x))$$

$$(Q(x) \vee \exists x. \forall y. P(f(\mathbf{x}), z) \wedge Q(\mathbf{y})) \vee \forall x. R(\mathbf{x}, z, g(\mathbf{x}))$$

$$(Q(\mathbf{x}) \vee \exists x. \forall y. P(f(\mathbf{x}), \mathbf{z}) \wedge Q(\mathbf{y})) \vee \forall x. R(\mathbf{x}, \mathbf{z}, g(\mathbf{x}))$$

$$(Q(\mathbf{x}) \vee \exists \mathbf{x}. \forall y. P(f(\mathbf{x}), \mathbf{z}) \wedge Q(\mathbf{y})) \vee \forall \mathbf{x}. R(\mathbf{x}, \mathbf{z}, g(\mathbf{x}))$$
Liées ? Libres ? Liantes ?
- Une formule sans occurrences libres est **close**.

Exercice : écrire la définition formelle !

Substitutions (1)

- **Substitution dans un terme :**
 $t[s/x]$, où x est une variable et t et s sont des termes :
 - 1 **Variable** : $x[s/x] = s$ et $y[s/x] = y$ pour $y \neq x$
 - 2 **Application de fonction** : $f(t_1, \dots, t_n)[s/x] = f(t_1[s/x], \dots, t_n[s/x])$
- **Substitution dans une formule :**
 $F[s/x]$, où x est une variable, s est un terme et F une formule :
 - 1 **Application de prédicat** : $P(t_1, \dots, t_n)[s/x] = P(t_1[s/x], \dots, t_n[s/x])$
 - 2 **Const. "faux", connecteurs propositionnels** : comme d'habitude
 - 3 **Quantificateur universel** :
 - $(\forall x. A)[s/x] = (\forall x. A)$
 - $(\forall y. A)[s/x] = (\forall y. (A[s/x]))$
 - 4 **Quantificateur existentiel** :
 - $(\exists x. A)[s/x] = (\exists x. A)$
 - $(\exists y. A)[s/x] = (\exists y. (A[s/x]))$

Substitutions (2)

Dans le cas des quantificateurs : Une **substitution** est dite **saine** si la variable quantifiée y diffère des variables de s

- **Subst. saine** : $(\forall y. R(y, x))[f(z)/x] = (\forall y. R(y, f(z)))$
- **Subst. pas saine** : $(\forall y. R(y, x))[f(y)/x] = (\forall y. R(y, f(y)))$

Pour obtenir une substitution saine :

Renommer des variables liées avant d'effectuer la substitution

$$(\forall y. R(y, x))[f(y)/x] = (\forall y_0. R(y_0, x))[f(y)/x] = (\forall y_0. R(y_0, f(y)))$$

Exercices : Calculer les substitutions (saines !) de :

- $(\forall x. \exists y. R(x, y) \wedge P(z))[f(v)/z]$
- $(\forall x. \exists y. R(x, y) \wedge P(z))[f(x)/z]$
- $(\forall x. \exists y. R(x, y) \wedge P(z))[f(v)/x]$

Plan

- 1 Motivation
- 2 Logique des propositions
- 3 Métathéorie
- 4 Logique du premier ordre
 - Syntaxe (langage)
 - Sémantique (théorie des modèles)
 - Méthode de preuve : **déduction naturelle**
- 5 Autres méthodes de preuve

Plan

	Lo. prop.	Lo. prem. ord	...
Syntaxe	✓	✓	
Sémantique	✓	✗	
Méth. preuve	• Déd. nat.	✓	
	• Tableaux		
	• Résolution		
Outils	• SAToulouse	✓	

Valuation et Interprétation

Rappel : Logique propositionnelle

- Une **valuation** v détermine la valeur de vérité d'une variable propositionnelle. **Exemple** : $v(A) = 0, v(B) = 1$
- Une **interprétation** v étend une valuation à une formule. **Exemple** : $v(\neg A \wedge B) = 1$

Logique des prédicats : Plus complexe :

- Distinction entre "termes" et "formules" :
 \rightsquigarrow deux fonctions d'interprétation différentes

Interprétation des termes (1)

Premier exemple intuitif : **Quelle est la signification** du terme $f(x, y)$?

- Si x représente 2 et y représente 3 et f est la fonction d'addition, alors $f(x, y)$ représente $2 + 3$, donc 5.
- Si x représente 2 et y représente 3 et f est la fonction de multiplication, alors $f(x, y)$ représente $2 * 3$, donc 6.

En général :

Une **valuation pour les termes** est donnée par

$(D, v_{VAR}, \{v_{FON_n} \mid n \in \text{Nat}\})$:

- Un domaine d'interprétation D : un ensemble non vide
- $v_{VAR} : VAR \Rightarrow D$ est une valuation pour les variables
- $v_{FON_n} : FON_n \Rightarrow D^n \Rightarrow D$ est une valuation pour les fonctions

Interprétation des termes (2)

Encore un exemple illustratif :

- Domaine $D = \text{Nat}$
- Interprétation des variables : $v_{VAR} : VAR \Rightarrow \text{Nat}$ avec
 $v_{VAR}(x) = 2, v_{VAR}(y) = 3$
- Interprétation des fonctions binaires :
 $v_{FON_2}(f)$ la fonction d'addition
 $v_{FON_2}(g)$ la fonction de multiplication

Le terme $f(x, g(x, y))$ est interprété comme $2 + (2 * 3) = 8$

Interprétation des termes (3)

Définition formelle :

Une **interprétation des termes** v est l'extension de $(D, v_{VAR}, \{v_{FON_n} | n \in \text{Nat}\})$ à $TERM$:

Définition récursive :

- **(Variable)** : $v(x) = v_{VAR}(x)$
- **(Application)** : $v(f(t_1, \dots, t_n)) = v_{FON_n}(f)(v(t_1), \dots, v(t_n))$

Exemple :

$v = (D, v_{VAR} = [x \mapsto 2, y \mapsto 3], \{v_{FON_2} = [f \mapsto (+), g \mapsto (*)]\})$

- $v(f(x, g(x, y))) = 2 + 6 = 8$
 - $v_{FON_2}(f) = (+)$
 - $v(x) = v_{VAR}(x) = 2$
 - $v(g(x, y)) = 2 * 3 = 6$
 - $v_{FON_2}(g) = (*)$
 - $v(x) = v_{VAR}(x) = 2$
 - $v(y) = v_{VAR}(y) = 3$

Interprétation des formules (1)

Quelle est la signification de la formule (élémentaire) $P(x)$?

- 1 Si x représente 2 et P représente la propriété "être un nombre pair", alors $P(x)$ est une proposition vraie.
- 2 Si x représente π et P représente la propriété "être un nombre rationnel", alors $P(x)$ est une proposition fausse.

Une **valuation pour les formules** est donnée par

$(D, v_{VAR}, \{v_{FON_n} | n \in \text{Nat}\}, \{v_{PRED_n} | n \in \text{Nat}\})$:

- $(D, v_{VAR}, \{v_{FON_n} | n \in \text{Nat}\})$ est une valuation pour les termes
- $v_{PRED_n} : PRED_n \Rightarrow D^n \Rightarrow \{0, 1\}$ est une valuation pour les prédicats

Interprétation des formules (2)

Une **interprétation des formules** v est l'extension de

$v = (D, v_{VAR}, \{v_{FON_n} | n \in \text{Nat}\}, \{v_{PRED_n} | n \in \text{Nat}\})$ à $FORM$:

- **Application de prédicat** :
 $v(P(t_1, \dots, t_n)) = v_{PRED_n}(P)(v(t_1), \dots, v(t_n))$
- **Constante "faux", négation, connecteurs binaires** :
comme pour la logique propositionnelle
- **Quantificateur universel** :
$$v(\forall x. \phi) = \begin{cases} 1 & \text{ssi } v_{[x:=d]}(\phi) = 1 \text{ pour tout } d \in D \\ 0 & \text{autrement} \end{cases}$$
- **Quantificateur existentiel** :
$$v(\exists x. \phi) = \begin{cases} 1 & \text{ssi } v_{[x:=d]}(\phi) = 1 \text{ pour quelque } d \in D \\ 0 & \text{autrement} \end{cases}$$

Actualisation d'une fonction pour un argument :

$$v_{[x:=d]}(y) = \begin{cases} d & \text{si } x = y \\ v(y) & \text{si } x \neq y \end{cases}$$

Interprétation des formules (3)

Cas spécial : Domaine fini. Soit

- $N_3 = \{0, 1, 2\}$
- \oplus l'addition modulo 3 : $x \oplus y \equiv (x + y) \pmod{3}$

Soit la valuation v avec

- Domaine N_3
- $v_{VAR} = [x_1 \mapsto 1, x_2 \mapsto 2]$
- $\{v_{FON_0} = [n_0 \mapsto 0, n_1 \mapsto 1, n_2 \mapsto 2], v_{FON_2} = [a \mapsto \oplus]\}$
- $\{v_{PRED_1} = [P \mapsto \text{"est un nombre pair"}]\}$

Interprétation de :

- $v(P(x_1)) = v_{PRED_1}(P)(v_{VAR}(x_1)) = \text{pair}(1) = 0$
- $v(\exists x. P(a(x_2, x))) = 1$ car $v_{[x:=1]}(P(a(x_2, x))) = \text{pair}(2 \oplus 1) = 1$

Interprétation des formules (4)

Exercices (dans le domaine N_3) :

- Développez $v(\forall x.P(a(x_2, x)))$
- Comparez $v(\exists x.P(x))$ et $v(P(n_0) \vee P(n_1) \vee P(n_2))$
- Comparez $v(\forall x.P(x))$ et $v(P(n_0) \wedge P(n_1) \wedge P(n_2))$

Les notions de **modèle**, **satisfiabilité**, **validité**, **conséquence** et **équivalence logique** sont définies comme en logique propositionnelle.

Exercices (en général) :

- Montrez : $\exists x.(P(x) \vee Q(x)) \equiv (\exists x.P(x)) \vee (\exists x.Q(x))$
- Montrez : $\forall x.(P(x) \wedge Q(x)) \equiv (\forall x.P(x)) \wedge (\forall x.Q(x))$
- Vous savez que $\neg(F_0 \vee F_1) \equiv (\neg F_0 \wedge \neg F_1)$.
Alors : $\neg(\exists x.P(x)) \equiv ???$
- Vous savez que $\neg(F_0 \wedge F_1) \equiv (\neg F_0 \vee \neg F_1)$.
Alors : $\neg(\forall x.P(x)) \equiv ???$

Interprétation des formules (4)

Vrai ou faux ?

- Si vrai, fournissez un modèle fini (avec \wedge et \vee)
(c'est un argument de plausibilité, pas une preuve) !
- Si faux, fournissez un contre-modèle

$\forall x.\forall y.R(x, y)$ $\equiv \forall y.\forall x.R(x, y)$	vrai	$(A_{1,1} \wedge A_{1,2}) \wedge (A_{2,1} \wedge A_{2,2})$ $\equiv (A_{1,1} \wedge A_{2,1}) \wedge (A_{1,2} \wedge A_{2,2})$
$\forall x.\exists y.R(x, y)$ $\equiv \exists y.\forall x.R(x, y)$	faux	contre-mod. : domaine : Nat , $v_{PRED}(R) = (<)$
$\exists x.\exists y.R(x, y)$ $\equiv \exists y.\exists x.R(x, y)$		
$\exists x.(P(x) \wedge Q(x))$ $\equiv (\exists x.P(x)) \wedge (\exists x.Q(x))$		
$\forall x.(P(x) \vee Q(x))$ $\equiv (\forall x.P(x)) \vee (\forall x.Q(x))$		

Interprétation des formules : Égalité (1)

- Ci-dessus, on avait dit que la sémantique d'une formule n'est pas "gravée dans la pierre" : ça dépend d'une valuation v . Même pour un prédicat comme $<$.
- Éviter de définir des v qui sont contraires à l'intuition, par exemple de définir $<$ comme "supérieur".
- Pour le prédicat $=$, c'est encore plus fort : Indépendamment de v_{PRED} , nous définissons

$$v(s = t) = 1 \quad \text{ssi} \quad v(s) = v(t)$$

- Pour $=$, la sémantique est "gravée dans la pierre" (pourtant elle dépend implicitement de la sémantique des termes).

Égalité : exemple

Considérons encore une fois

- $N_3 = \{0, 1, 2\}$
- \oplus l'addition modulo 3 : $x \oplus y \equiv (x + y) \bmod 3$

Soit la valuation v avec

- Domaine N_3
- $v_{VAR} = [x_1 \mapsto 1, x_2 \mapsto 2]$
- $\{v_{FON_0} = [n_0 \mapsto 0, n_1 \mapsto 1, n_2 \mapsto 2], v_{FON_2} = [a \mapsto \oplus]\}$
- $\{v_{PRED_1} = [P \mapsto \text{"est un nombre pair"}]\}$

Interprétation de :

- $v(x_1 = n_1) = 1$,
- $v(a(n_2, n_1) = n_0) = 1$,
- $v(n_1 = n_0) = 0$

Plan

- 1 Motivation
- 2 Logique des propositions
- 3 Métathéorie
- 4 Logique du premier ordre
 - Syntaxe (langage)
 - Sémantique (théorie des modèles)
 - Méthode de preuve : déduction naturelle
- 5 Autres méthodes de preuve

Plan

	Lo. prop.	Lo. prem. ord	...
Syntaxe	✓	✓	
Sémantique	✓	✓	
Méth. preuve	• Déd. nat.	✓	
	• Tableaux		
	• Résolution		
Outils	• SAToulouse	✓	

Déduction Naturelle pour LPred

Extension de la déduction naturelle pour LProp :

- Les règles pour LProp restent en vigueur
- Nouvelles règles pour les quantificateurs : $(I\exists)$, $(E\exists)$, $(I\forall)$, $(E\forall)$
- Construction d'un arbre de dérivation ...comme pour LProp

Exemple d'une preuve "naturelle" de $\forall x.\exists y.x < y$

- 1 Soit x une valeur arbitraire (mais fixe), pour laquelle il faut prouver $\exists y.x < y$.
- 2 Pour montrer $\exists y.x < y$, il faut trouver un y qui satisfait $x < y$ (et qui peut dépendre de x). On choisit $x + 1$ pour y .
- 3 On vérifie que $x < x + 1$ est satisfait.

Règles : $(I\exists)$

$$\frac{A[t/x]}{\exists x.A} (I\exists)$$

Condition : $[t/x]$ est une substitution saine.

Lecture informelle :

A est vrai pour un t . Donc, il existe un x pour lequel A est vrai.

Règles : ($E\forall$)

$$\frac{\forall x.A}{A[t/x]} (E\forall)$$

Condition : $[t/x]$ est une substitution saine.

Lecture informelle :

A est vrai pour tout x . Donc, il est vrai en particulier pour un t (que je peux choisir).

Règles : (\forall) (1)

$$\frac{A}{\forall x.A} (\forall)$$

Condition : x n'est pas libre dans les hypothèses de la prémisse.

Lecture informelle :

- Si A est vrai pour n'importe quel x , alors aussi $\forall x.A$ est vrai.
- **Condition :** ne pas découpler les variables qui se réfèrent au même objet

Règles : (\forall) (2)

Exemple d'une preuve incorrecte (!) qui ne respecte pas la condition :

$$\frac{\frac{\frac{[x=0]^1}{\forall x.x=0} (\forall) \quad x=0 \rightarrow (\forall x.x=0) (I \rightarrow)^1}{\forall x.(x=0 \rightarrow (\forall x.x=0))} (\forall) \quad 0=0 \rightarrow (\forall x.x=0) (E\forall)}$$

Règles : ($E\exists$) (1)

$$\frac{\frac{\exists x.A}{C} \quad \begin{array}{c} [A] \\ \vdots \\ C \end{array}}{C} (E\exists)$$

Condition : x n'est libre ni dans les hypothèses de C (sauf A), ni dans C

Lecture informelle : Pour montrer C , sachant que $\exists x.A$, il suffit de postuler A (pour un x dont on ne connaît pas l'identité exacte) et de montrer C .

Règles : $(E\exists)$ (2)

Exemple d'une preuve :

$$\begin{array}{c}
 \frac{[P(x) \wedge Q(x)]^1}{P(x)} (E\wedge_1) \\
 \frac{\frac{[P(x) \wedge Q(x)]^2}{\exists x.(P(x) \wedge Q(x))} (I\exists) \quad \frac{P(x)}{\exists x.P(x)} (I\exists)^1}{\exists x.P(x)} (E\exists)^1 \\
 \frac{\exists x.P(x)}{(\exists x.(P(x) \wedge Q(x))) \longrightarrow (\exists x.P(x))} (I \longrightarrow)^2
 \end{array}$$

Règles : $(E\exists)$ (3)Exemple d'une preuve **incorrecte** (!) :

$$\begin{array}{c}
 \frac{[P(x) \wedge Q(x)]^1}{P(x)} (E\wedge_1) \\
 \frac{[P(x) \wedge Q(x)]^2}{P(x)} (E\exists)^1 \\
 \frac{P(x)}{\forall x.P(x)} (I\forall) \\
 \frac{(\exists x.P(x) \wedge Q(x)) \longrightarrow (\forall x.P(x))}{(\exists x.P(x) \wedge Q(x)) \longrightarrow (\forall x.P(x))} (I \longrightarrow)^2
 \end{array}$$

Discussion :

- L'application de $(I\forall)$ est correcte : x n'apparaît pas **libre** dans l'hypothèse
- L'application de $(E\exists)$ est incorrecte : x apparaît libre dans la conclusion

Égalité (1)

On avait dit que la sémantique de $=$ dépend de la sémantique des termes ; après, elle est "gravée dans la pierre" :

Par exemple, soit la valuation v avec domaine $N_3 = \text{Nat}$;

$\{v_{FON_0} = [n_0 \mapsto 0, n_1 \mapsto 1, n_2 \mapsto 2], v_{FON_2} = \dots\}$.

- $v(a(n_2, n_1) = n_0) = \mathbf{1}$ pour $v_{FON_2}[a \mapsto \oplus]$ (addition modulo 3)
- $v(a(n_2, n_1) = n_0) = \mathbf{0}$ pour $v_{FON_2}[a \mapsto +]$ (addition habituelle)

Il y a pourtant certaines exigences minimales : L'égalité est

- **réflexive**, **transitive**, **symétrique** ;
- On peut remplacer un terme par un terme "égal" (**congruence**).

Égalité (2)

Réflexivité ("règle d'introduction") :

$$\frac{}{t = t} (refl)$$

Congruence ("règle d'élimination") :

$$\frac{t_1 = t_2 \quad A[t_1/x]}{A[t_2/x]} (congr)$$

Définition de l'égalité de Leibniz :

- Deux objets sont égaux s'ils sont indistinguables (par rapport à toute propriété A)

Exercices : Montrez : l'égalité est

- symétrique : $t_1 = t_2 \longrightarrow t_2 = t_1$
- transitive : $t_1 = t_2 \longrightarrow t_2 = t_3 \longrightarrow t_1 = t_3$

Égalité (3)

Deux preuves ...

Correct :

$$\frac{\frac{\overline{x = x} \text{ (refl)}}{\exists y. x = y} \text{ (I}\exists\text{)}}{\forall x. \exists y. x = y} \text{ (I}\forall\text{)}$$

Incorrect :

$$\frac{\frac{\overline{x = x} \text{ (refl)}}{\forall x. x = x} \text{ (I}\forall\text{)}}{\exists y. \forall x. x = y} \text{ (I}\exists\text{)}$$

Trouvez l'erreur !

Égalité : théorie

- La sémantique, même si elle est “gravée dans la pierre”, dépend de la sémantique des termes.
- Pour modéliser un certain domaine, on peut fixer un ensemble T d'égalités et/ou inégalités. On appelle un tel ensemble une **théorie**. Après on fera des dérivation de $T \vdash \phi$ pour des formules en question.
- Exemple : $T = \{\forall x. \neg(0 = s(x)), \forall x, y. suc(x) = suc(y) \rightarrow x = y, \forall x. 0 + x = x, \forall x, y. s(x) + y = s(x + y)\}$
Exercice : Démontrez $T \vdash \neg(s(0) = s(s(0)))$,
 $T \vdash s(0) + s(0) = s(s(0))$.

Logique du premier ordre : Résumé

- La logique du premier ordre est beaucoup plus expressive que le logique propositionnelle :
 - Elle permet de parler d'individus ;
 - Elle permet de parler d'une infinité d'individus ;
 - Elle est plus concise ...
- On a maintenant deux **catégories syntaxiques** : les **termes** et les **formules**.
- On a les quantificateurs \forall, \exists et on introduit quatre règles de dérivation pour traiter \forall, \exists dans la déduction naturelle.
- La logique du premier ordre est la logique “par excellence” :
 - Il y a des logiques plus expressives, mais elles sont beaucoup moins connues.
 - Il y a d'autres logiques (modale, temporelle, descriptive ...) qui peuvent être traduites dans la logique du premier ordre.

Plan

- 1 Motivation
- 2 Logique des propositions
- 3 Métathéorie
- 4 Logique du premier ordre
- 5 Autres méthodes de preuve
 - Méthode des tableaux
 - Résolution

Plan

		Lo. prop.	Lo. prem. ord	...
Syntaxe		✓	✓	
Sémantique		✓	✓	
Méth. preuve	• Déd. nat.	✓	✓	
	• Tableaux	✗		
	• Résolution			
Outils	• SAToulouse	✓		

Motivation (1)

Dédution naturelle :

- correspond à un style de raisonnement “naturel”
- difficile à automatiser : pas de propriété de sous-formule

Calcul des tableaux :

- **Avantage** : Facile à mécaniser
 - ...parce qu'il suffit de décomposer des formules données
 - ...pas nécessaire d'“inventer” quelque chose
- **Désavantage** : Ne correspond pas à un raisonnement “naturel”

Méthode des tableaux : Idée (1)

Exemple : Déterminer si $(\neg q) \wedge (p \rightarrow q)$ est satisfiable.

p	q	$\neg q$	$p \rightarrow q$	$(\neg q) \wedge (p \rightarrow q)$
0	0	1	1	1
0	1	0	1	0
1	0	1	0	0
1	1	0	1	0

- $v((\neg q) \wedge (p \rightarrow q)) = 1$, donc

- $v(\neg q) = 1$, donc $v(q) = 0$

- et $v(p \rightarrow q) = 1$, donc

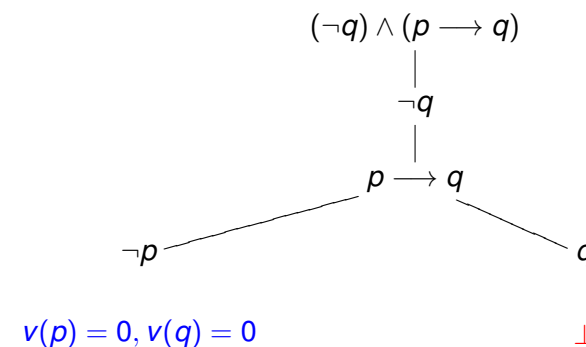
① $v(p) = 0$

② ou $v(q) = 1$ (impossible parce que $v(q) = 0$)

Résultat : $v((\neg q) \wedge (p \rightarrow q)) = 1$ pour la valuation $v(q) = 0$, $v(p) = 0$

Méthode des tableaux : Idée (2)

Le même raisonnement, avec arbre de recherche :



Structure de données (1)

Un **arbre de recherche** est :

- ① une feuille
- ② un nœud **unaire** dont le fils est un arbre de recherche
- ③ un nœud **binaire** dont les deux fils sont des arbres de recherche

Une **branche** est un chemin menant de la racine à une feuille.

Une branche peut être

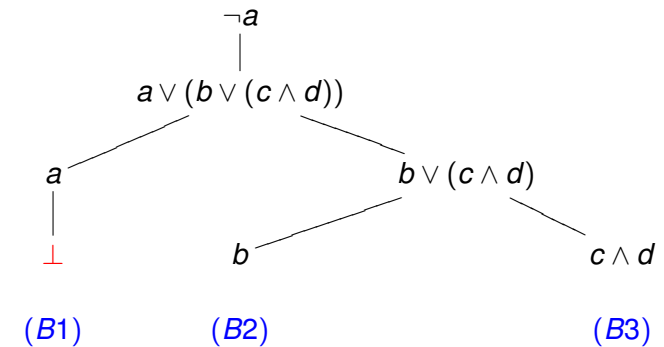
- ① **fermée** si sa feuille est marquée \perp
- ② **ouverte** si elle n'est pas fermée.

Un **nœud** sur une branche ouverte peut être

- ① **passif** si une règle a déjà été appliquée à ce nœud sur cette branche
- ② **actif** autrement

Structure de données (2)

Structure de base :

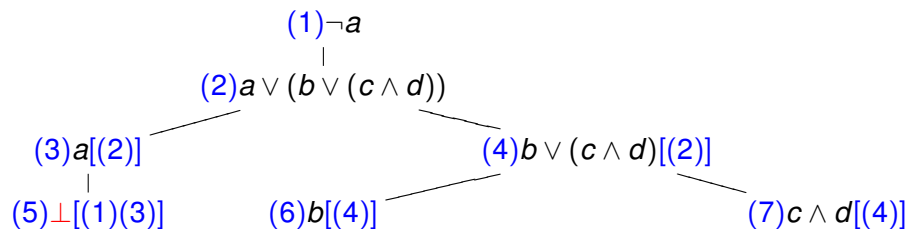


- Branche (B1) est fermée
- Branche (B2) est ouverte, avec nœuds actifs $\neg a$ et b
- Branche (B3) est ouverte, avec nœuds actifs $\neg a$ et $c \wedge d$

Structure de données (3)

Information administrative :

- Chaque nœud a un numéro, unique dans l'arbre
- Les nœuds dérivés sont annotés de leur origine : [nœud(s)]



Règles (1)

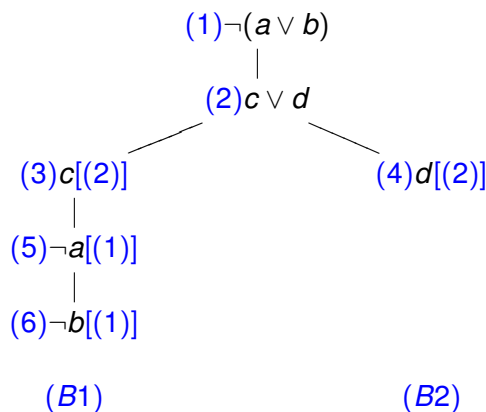
Condition d'applicabilité :

- **Fermeture** peut être appliquée à toute branche ouverte
- les autres règles peuvent être appliquées à tout nœud actif d'une branche ouverte

Effet de l'application d'une règle :

- **Fermeture** : Si la branche contient A et $\neg A$, créer un fils avec \perp
- **Règles conjonctives** : créent deux fils sur une seule branche :
 - **Règle \wedge** : Pour $A \wedge B$, créer deux fils avec A et B
 - **Règle $\neg \vee$** : Pour $\neg(A \vee B)$, créer deux fils avec $\neg A$ et $\neg B$
 - **Règle $\neg \rightarrow$** : Pour $\neg(A \rightarrow B)$, créer deux fils avec A et $\neg B$
- **Règles disjonctives** : créent deux fils sur deux branches :
 - **Règle \vee** : Pour $A \vee B$, créer un fils avec A et un autre avec B
 - **Règle $\neg \wedge$** : Pour $\neg(A \wedge B)$, créer un fils avec $\neg A$ et un autre avec $\neg B$
 - **Règle \rightarrow** : Pour $A \rightarrow B$, créer un fils avec $\neg A$ et un autre avec B
- **Règle pour $\neg \neg$** : Pour $\neg \neg A$, créer un fils avec A

Règles (2)



- Dans (B1)
 - Règle \vee n'est plus applicable
 - Règle $\neg\vee$ n'est plus applicable
 - Dans (B2)
 - Règle \vee n'est plus applicable
 - Règle $\neg\vee$ est encore applicable à nœud (1)
- Noter :** "Applicabilité" d'une règle est relative à une branche !

Algorithme

Algorithme : Tant que le tableau contient des branches ouvertes qui admettent l'application d'une règle :

- 1 sélectionner une telle branche B
- 2 sélectionner un nœud n dans B qui admet l'application de R
- 3 appliquer R

Résultat : Si l'algorithme termine,

- toutes les branches sont fermées
 \leadsto le tableau est insatisfiable
- **ou bien** il existe des branches ouvertes, sans règles applicables
 \leadsto le tableau est satisfiable
 \leadsto chaque branche ouverte contient uniquement des littéraux (p ou $\neg p$, pour $p \in PROP$)
 \leadsto permet la construction d'un modèle

Satisfiabilité et validité

Pour vérifier la **satisfiabilité** d'une formule F :

- 1 Appliquer l'algorithme des tableaux à F
- 2 Résultat :
 - toutes les branches sont fermées
 $\leadsto F$ est insatisfiable
 - il existe des branches ouvertes
 $\leadsto F$ est satisfiable
 \leadsto on peut construire un modèle de F

Pour vérifier la **validité** d'une formule F :

- 1 Appliquer l'algorithme des tableaux à $\neg F$
- 2 Résultat :
 - toutes les branches sont fermées
 $\leadsto \neg F$ est insatisfiable
 donc : F est valide
 - il existe des branches ouvertes
 $\leadsto \neg F$ est satisfiable
 donc : F n'est pas valide
 le modèle de $\neg F$ est un **contre-modèle** de F

Extensions : Conséquence logique

Au lieu de vérifier $\models F$ (" F est valide"), on s'intéresse souvent à : $\{H_1, \dots, H_n\} \models F$ (" F est valide sous hypothèses $H_1 \dots H_n$ ")
 Ceci est le cas seulement si $\{H_1, \dots, H_n, \neg F\}$ est insatisfiable

Algorithme :

- 1 Mettre les formules $H_1, \dots, H_n, \neg F$ sur une branche du tableau
- 2 Développer le tableau
- 3 Résultat :
 - toutes les branches sont fermées : $\{H_1, \dots, H_n\} \models F$
 - il existe des branches ouvertes : $\{H_1, \dots, H_n\} \not\models F$

Algorithme : Propriétés (1)

Terminaison : L'algorithme de calcul des tableaux termine.

Argument informel :

- Définir le **poids** d'une branche comme la "liste" des tailles de ses nœuds actifs
- Le poids des branches diminue lors de l'application des règles
- ...et ceci ne peut pas continuer à l'infini

Exemple :

- Avant application de la règle :
Branche : $[a \wedge b; c \vee (b \wedge \neg a)]$
Poids de la branche : $[3; 6]$
- Après application de la règle \vee :
Branches $[a \wedge b; c \vee (b \wedge \neg a); c]$ et $[a \wedge b; c \vee (b \wedge \neg a); (b \wedge \neg a)]$
Poids des branches : $[3; 1]$ et $[3; 4]$
Il y a plus de branches, mais à moindre poids

Algorithme : Propriétés (2)

Correction de l'algorithme :

Informellement :

- L'algorithme ne trouve pas de "faux modèles"
- Si l'algorithme dit que le tableau est **satisfiable**, il l'est effectivement

Quelques définitions :

- Une branche B est dite **satisfiable** par une interprétation v si pour toutes les formules actives F de B , on a $v(F) = 1$
- Un tableau est dit **satisfiable** par une interprétation v s'il existe une branche qui est satisfiable par v

Algorithme : Propriétés (3)

Correction de l'algorithme : suit de la

Correction de l'application d'une règle :

Si le tableau T' est le résultat de l'application de la règle R à T , si T' est satisfiable par v , alors aussi T est satisfiable par v

Preuve : Soit T' satisfiable par v , avec B' satisfiable par v

- B' est déjà dans T .
Alors : T est satisfiable par v
- autrement : examiner les règles :
 - Règle \wedge : alors $B' = [\dots; p \wedge q; p; q]$, avec $v(p) = v(q) = 1$
alors $v(p \wedge q) = 1$, donc $B = [\dots; p \wedge q]$ dans T est satisfiable.
 - **Prouvez les autres règles**

Algorithme : Propriétés (4)

Complétude de l'algorithme :

Informellement : L'inverse de la correction :

- L'algorithme ne perd pas de modèles
- Si l'algorithme dit que le tableau est **insatisfiable**, il l'est effectivement

Complétude de l'algorithme : suit de la

Complétude de l'application d'une règle :

Si le tableau T' est le résultat de l'application de la règle R à T , si T est satisfiable par v , alors aussi T' est satisfiable par v

Preuve : (un peu moins en détail) :

- Règle \vee : Soit $B = [\dots; p \vee q]$ satisfiable par v , donc
 - ① $v(p) = 1$, donc $B' = [\dots; p \vee q; p]$ satisfiable par v
 - ② **ou** : $v(q) = 1$, donc $B' = [\dots; p \vee q; q]$ satisfiable par v
- **Prouvez les autres règles**

Correction et complétude

Exercice 1 : Vous avez besoin d'une règle pour le "ou exclusif" ($A \oplus B$) défini par $((A \vee B) \wedge \neg(A \wedge B))$

On propose la règle :

- Si le nœud contient $A \oplus B$, créer un fils contenant A et $\neg B$

Prouvez avec cette règle :

- $(A \oplus B) \longrightarrow (A \longrightarrow \neg B)$
- $(A \oplus B) \longrightarrow \neg B$

Cette règle est-elle correcte ? complète ?

Sinon, proposez une règle correcte et complète.

Exercice 2 :

- Notre règle de **Fermeture** permet de fermer une branche contenant A et $\neg A$, pour n'importe quelle formule A .
- Introduisez une règle **Fermeture'** qui permet de fermer une branche uniquement pour p et $\neg p$, où $p \in PROP$?
- **Question :** Conséquences pour la correction / complétude ?

Stratégies

Observation : L'algorithme ci-dessus est non-déterministe :

- sélection d'une branche
- sélection d'un nœud sur la branche

Des **stratégies** rendent l'algorithme plus déterministe, en imposant des préférences :

- Si vous avez le choix entre la règle (\wedge) et la règle (\vee), laquelle préférez-vous ?
- **Généralisez !**

Plan

- 1 Motivation
- 2 Logique des propositions
- 3 Métathéorie
- 4 Logique du premier ordre
- 5 Autres méthodes de preuve
 - Méthode des tableaux
 - Résolution

Plan

	Lo. prop.	Lo. prem. ord	...
Syntaxe	✓	✓	
Sémantique	✓	✓	
Méth. preuve <ul style="list-style-type: none"> • Déd. nat. • Tableaux • Résolution 	✓	✓	
	✓		
	✗		
Outils <ul style="list-style-type: none"> • SAToulouse 	✓		

Motivation et Historique (1)

Résolution : Une méthode de déduction assez récente :

Alan Robinson :

A Machine-Oriented Logic Based on the Resolution Principle (1965)

But :

- Contexte : Premières applications en Intelligence Artificielle
- Implantation simple et efficace sur ordinateur
- ...sans ambition d'être compréhensible pour des humains

Principe :

- Une seule règle d'inférence appliquée à un ensemble de **clauses**
- ...nécessitant un pré-traitement de formules structurées

Motivation et Historique (2)

Idée : Propagation de faits

Donné :

- des faits : A, C
- des implications : $A \rightarrow B, C \rightarrow B \rightarrow D, B \rightarrow D \rightarrow F$
- un but : prouver F

Démarche : Propagation des faits dans les implications
("modus ponens") :

1.
 - A et $A \rightarrow B$ donnent B
 - C et $C \rightarrow B \rightarrow D$ donnent $B \rightarrow D$
2.
 - B et $B \rightarrow D \rightarrow F$ donnent $D \rightarrow F$
 - B et $B \rightarrow D$ donnent D
3.
 - D et $D \rightarrow F$ donnent F

Ceci est la base du langage de programmation **Prolog**

Motivation et Historique (3)

Les bases de la résolution :

- Au lieu d'implications ($A \rightarrow B$) : des disjonctions : $\neg A \vee B$
- La règle de résolution : A et $\neg A \vee B$ donnent B

...dans un cadre plus général :

- Des **clauses** avec plusieurs variables positives et négatives : $\neg A \vee B \vee C$
- Résolution entre clauses : $A \vee B \vee \neg D$ et $\neg A \vee B \vee C$ donnent $B \vee C \vee \neg D$

Étapes principales : Pour une formule F

1. Construction d'une forme clausale de F
2. Application de la résolution

Formes Normales

Un **littéral** est une variable propositionnelle ou sa négation,

par exemple : $p, \neg q$, **mais pas** : $p \vee \neg q$

Une formule est en

- **Forme Normale Disjonctive (FND)** si elle est une disjonction de conjonctions de littéraux
- **Forme Normale Conjonctive (FNC)** si elle est une conjonction de disjonctions de littéraux

Exercice : FND, FNC, ou ... ???

- | | |
|---|---|
| • $(\neg A \vee B) \wedge (C \vee D)$ | • $(A \vee \perp) \wedge (C \wedge \neg D)$ |
| • $(\neg A \wedge B) \vee (C \wedge D)$ | • $A \vee (C \vee \neg D)$ |
| • $\neg(A \wedge B) \vee (C \wedge D)$ | • $A \wedge \neg B$ |

Conversion en Forme Normale Conjonctive

- 1 Éliminer les connecteurs autres que \neg, \wedge, \vee
 - 1 Réécrire $A \leftrightarrow B$ en $(A \rightarrow B) \wedge (B \rightarrow A)$
 - 2 Réécrire $A \rightarrow B$ en $\neg A \vee B$
- 2 Tirer les négations à l'intérieur, éliminer les doubles négations :
 - $\neg(A \wedge B)$ devient $\neg A \vee \neg B$
 - $\neg(A \vee B)$ devient $\neg A \wedge \neg B$
 - $\neg\neg A$ devient A
- 3 Distribuer \vee sur \wedge :
 - $A \vee (B \wedge C)$ devient $(A \vee B) \wedge (A \vee C)$
 - et pareil pour $(B \wedge C) \vee A$
- 4 Faire des simplifications triviales, par exemple de $A \wedge \neg A$, $A \vee \neg A$, $A \vee \perp \dots$

Exercice : Convertir $\neg((A \wedge \neg B) \rightarrow (A \leftrightarrow \neg B))$ en FNC

Clauses

Une **clause** est un ensemble de littéraux.

Une clause représente la disjonction de ses littéraux.

Exemple : $\{A, \neg B\}$ représente $A \vee \neg B$

Notes :

- Deux formules syntaxiquement différentes peuvent être représentées par la même clause : $\{A, \neg B\}$ représente $A \vee \neg B$ et $A \vee \neg B \vee A$
- ...mais ces formules sont équivalentes : $(A \vee \neg B) \equiv (A \vee \neg B \vee A)$
- La clause vide $\{\}$ représente \perp

Un **ensemble de clauses** représente la conjonction des disjonctions des littéraux des clauses.

Exemple : $\{\{A, \neg B\}, \{B, \neg A\}\}$ représente $(A \vee \neg B) \wedge (B \vee \neg A)$

Règle de Résolution

Étant donné :

- une clause $C = \{\ell_1 \dots \ell_m, p\}$
- une clause $C' = \{\ell'_1 \dots \ell'_n, \neg p\}$

La **résolvante** de C et C' (écrit $res(C, C')$)

est la clause $\{\ell_1 \dots \ell_m, \ell'_1 \dots \ell'_n\}$

(**Petite imprécision :** on ne détaille pas quel littéral p est éliminé.)

Exemples :

- $res(\{A, B\}, \{\neg C, \neg B\}) = \{A, \neg C\}$
- $res(\{\neg A, B, C\}, \{\neg B, A\}) = \{\neg A, A, C\}$
- $res(\{\neg A, B, C\}, \{\neg B, A\}) = \{\neg B, B, C\}$

Cas spécial : La résolvante peut être vide : $res(\{A\}, \{\neg A\}) = \{\}$

Preuve par Résolution

Donné : Un ensemble de clauses \mathcal{E}

Algorithme :

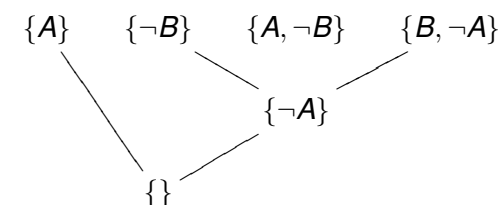
Tant que $\{\} \notin \mathcal{E}$ et tant qu'il existe $C, C' \in \mathcal{E}$ avec $res(C, C') \notin \mathcal{E}$:

- $\mathcal{E} := \mathcal{E} \cup \{res(C, C')\}$

Résultats possibles :

- 1 $\{\} \in \mathcal{E}$. Alors \mathcal{E} est insatisfiable
- 2 $\{\} \notin \mathcal{E}$ et impossible de former de nouvelles résolvantes.
Alors \mathcal{E} est satisfiable

Exemple d'un ensemble de clauses insatisfiable :



Résolution : Propriétés (1)

Préservation des interprétations :

$\mathcal{I}(\mathcal{E} \cup \{res(\mathcal{C}, \mathcal{C}')\}) = \mathcal{I}(\mathcal{E})$ pour $\mathcal{C}, \mathcal{C}' \in \mathcal{E}$

Preuve : Soit $\mathcal{C} = \{\ell_1 \dots \ell_m, p\}$, $\mathcal{C}' = \{\ell'_1 \dots \ell'_n, \neg p\}$

- Soit \mathcal{I} un modèle de $\mathcal{E} \cup \{res(\mathcal{C}, \mathcal{C}')\}$. Alors, \mathcal{I} est un modèle de \mathcal{E} .
- Soit \mathcal{I} un modèle de \mathcal{E} ,
donc surtout : $\mathcal{I}(\mathcal{C}) = 1$ et $\mathcal{I}(\mathcal{C}') = 1$
donc $\mathcal{I}(\ell_1 \vee \dots \vee \ell_m \vee p) = 1$ et $\mathcal{I}(\ell'_1 \vee \dots \vee \ell'_n \vee \neg p) = 1$.
Alors, $\mathcal{I}(\ell_1 \vee \dots \vee \ell_m \vee \ell'_1 \vee \dots \vee \ell'_n) = 1$
Pourquoi ? Complétez les détails !
Donc : \mathcal{I} est un modèle de $\mathcal{E} \cup \{res(\mathcal{C}, \mathcal{C}')\}$.

Résolution : Propriétés (2)

Correction : Étant donné \mathcal{E} . Si l'algorithme de résolution fourni un \mathcal{E}' avec $\{\} \in \mathcal{E}'$, alors \mathcal{E} n'est pas satisfiable.

Preuve : par induction sur le nombre n d'itérations de l'algorithme.

- $n = 0$: Alors, $\mathcal{E} = \{\mathcal{C}_1, \dots, \mathcal{C}_k, \{\}\}$ représente une formule
 $\mathcal{C}_1 \wedge \dots \wedge \mathcal{C}_k \wedge \perp \equiv \perp$
- $n \rightarrow n + 1$:
D'après la préservation des interprétations :
Si $\mathcal{E} \cup \{res(\mathcal{C}, \mathcal{C}')\}$ n'a pas de modèle, alors \mathcal{E} non plus.

Résolution : Propriétés (3)

Complétude : Étant donné \mathcal{E} . Si l'algorithme de résolution fourni un \mathcal{E}' avec $\{\} \notin \mathcal{E}'$, alors \mathcal{E} est satisfiable.

Preuve : par induction sur le nombre n d'itérations de l'algorithme.

- $n = 0$: Si
 - $\{\} \notin \mathcal{E}$
 - et \mathcal{E} est **saturé** (n'admet pas la dérivation d'une nouvelle résolvente)
alors \mathcal{E} a un modèle.
voir : Extraction d'un modèle
- $n \rightarrow n + 1$:
D'après la préservation des interprétations :
Tout modèle de $\mathcal{E} \cup \{res(\mathcal{C}, \mathcal{C}')\}$ est un modèle de \mathcal{E} .

Extraction d'un modèle (1)

Substitution d'une constante dans un ensemble de clauses :

- $\mathcal{E}[\perp/p]$ est l'ensemble de clauses où :
 - le littéral p est éliminé de toutes les clauses
 - toute clause contenant $\neg p$ est éliminée**Exemple :** $\{\{A, \neg B\}, \{\neg A, C\}, \{A, C\}\}[\perp/A] = \{\{\neg B\}, \{C\}\}$
- $\mathcal{E}[\top/p]$ est l'ensemble de clauses où :
 - toute clause contenant p est éliminée
 - le littéral $\neg p$ est éliminé de toutes les clauses**Exemple :** $\{\{A, \neg B\}, \{\neg A, C\}, \{A, C\}\}[\top/A] = \{\{C\}\}$

Justifiez cette définition !

Montrez : Si \mathcal{E} est saturé, alors aussi $\mathcal{E}[\perp/p]$ et $\mathcal{E}[\top/p]$.

Extraction d'un modèle (2)

Construction d'un modèle par preuve de :

Si $\{\} \notin \mathcal{E}$ et \mathcal{E} est saturé, alors \mathcal{E} a un modèle.

Induction sur le nombre n de variables p_1, \dots, p_n dans \mathcal{E} .

La preuve construit une valuation v pour p_1, \dots, p_n tq. $\mathcal{I}_v(\mathcal{E}) = 1$

- $n = 0$: \mathcal{E} est de la forme :
 - $\{\}$: Alors, \mathcal{E} est valide
 - $\{\{\}\}$: impossible, parce que $\{\} \notin \mathcal{E}$
- $n \rightarrow n + 1$:
 - ① Construire $\mathcal{E}' := \mathcal{E}[\perp/p_{n+1}]$, qui est saturé. Deux cas :
 - $\{\} \in \mathcal{E}'$: Alors, \mathcal{E}' n'a pas de modèle.
 - $\{\} \notin \mathcal{E}'$: Par hyp. d'induction, il existe valuation v' tq. $\mathcal{I}_{v'}(\mathcal{E}') = 1$.
Alors, pour v défini comme $v'(p_{n+1} := 0)$, on a $\mathcal{I}_v(\mathcal{E}) = 1$
 - ② Construire $\mathcal{E}'' := \mathcal{E}[\top/p_{n+1}]$, et procéder en analogie à (1)

Pourquoi est-ce qu'on ne peut pas avoir $\{\} \in \mathcal{E}'$ et $\{\} \in \mathcal{E}''$ en même temps ?

Extraction d'un modèle (3)

Exemple : Ensemble de clauses initial : $\{\{A, \neg B\}, \{B, \neg C\}, \{A, C\}\}$

Ensemble saturé : $\mathcal{E} = \{\{A, \neg B\}, \{B, \neg C\}, \{A, C\}, \{A, \neg C\}, \{A\}\}$

Construction des modèles :

- $\mathcal{E}[\perp/A] = \{\{\neg B\}, \{B, \neg C\}, \{C\}, \{\neg C\}, \{\}\}$ n'a pas de modèle
- $\mathcal{E}[\top/A] = \{\{B, \neg C\}\}$
 - $\{\{B, \neg C\}\}[\perp/B] = \{\neg C\}$
 - $\{\neg C\}[\perp/C] = \{\}$ est valide
 \rightsquigarrow valuation $v_0(A) = 1, v_0(B) = 0, v_0(C) = 0$
 - $\{\neg C\}[\top/C] = \{\{\}\}$ n'a pas de modèle
 - $\{\{B, \neg C\}\}[\top/B] = \{\}$ est valide
 \rightsquigarrow valuation $v_1(A) = 1, v_1(B) = 1$

Optimisations

Subsomption : La clause \mathcal{C} **subsume** la clause \mathcal{C}' si $\mathcal{C} \subseteq \mathcal{C}'$.

Signification :

Si la clause \mathcal{C} représente la formule C et \mathcal{C}' représente C' , alors $C \rightarrow C'$.

Exemple : $\{A\}$ subsume $\{A, \neg B\}$, et $A \rightarrow (A \vee \neg B)$

Constat : Si $\mathcal{E} \cup \{\mathcal{C}, \mathcal{C}'\}$ est insatisfiable et $\mathcal{C} \subseteq \mathcal{C}'$, alors déjà $\mathcal{E} \cup \{\mathcal{C}\}$ est insatisfiable.

Idée de preuve : Si $\neg(E \wedge C \wedge C')$ et $C \rightarrow C'$, alors $\neg(E \wedge C)$

Optimisation : "On peut supprimer les clauses subsumées par d'autres"

Dérivez la clause vide à partir de

$\{\{A, \neg B\}, \{\neg A, C\}, \{A\}, \{B, C\}, \{\neg C\}\}$

Résolution : Résumé (1)

Idée générale : La résolution permet de vérifier qu'un ensemble de clauses est insatisfiable.

Pour vérifier la validité d'une formule F :

- ① Convertir $\neg F$ en Forme Normale Conjonctive
- ② Construire l'ensemble de clauses correspondant
- ③ Appliquer l'algorithme de résolution :
 - Si $\{\}$ est une résolvante, alors F est valide
 - Si on ne peut pas obtenir $\{\}$, alors F n'est pas valide
(contre-modèle par extraction d'un modèle de l'ensemble de clauses saturé)

Résolution : Résumé (2)

Astuces Pour vérifier que $\{H_1, \dots, H_N\} \models F$:

- Convertir $\neg((H_1 \wedge \dots \wedge H_n) \rightarrow F)$ en FNC
- équivalent à convertir $(H_1 \wedge \dots \wedge H_n \wedge \neg F)$ en FNC
- équivalent à convertir H_1 en FNC, $\dots H_n$ en FNC, $\neg F$ en FNC et prendre l'union des clauses
- vérifier que cet ensemble est insatisfiable

Vérifier que $\{A \rightarrow B, B \wedge C \rightarrow \neg D, A, D\} \models \neg C$

Plan

- 1 Motivation
- 2 Logique des propositions
- 3 Métathéorie
- 4 Logique du premier ordre
- 5 Autres méthodes de preuve

La fin, ou presque ...

Annexe

Annexe

Digression : Classes de complexité (1)

(Mise en garde : Simplification très grossière !)

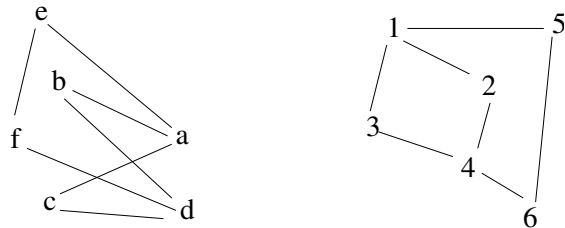
Classe P :

- Classe des problèmes dont le temps de calcul est borné par un **polynome** (en fonction de la taille de l'entrée)
- **Exemple** : Trier une liste de n éléments nécessite n^2 opérations (et on peut faire mieux)
- **Exemple** : Trouver le plus court chemin entre deux villes (pour un pays avec n villes) nécessite n^3 opérations
- **Exemple** : Étant donnée une interprétation v et une formule A , la **vérification** si $v(A) = 1$ nécessite $|A|$ opérations

Digression : Classes de complexité (2)

Classe NP (non-déterministe polynomial) :

- Classe des problèmes résolubles en
 - devinant une solution (instantanément)
 - vérifiant cette solution (en temps polynomial)
- **Exemple** : Isomorphisme de graphes
Est-ce que les deux graphes sont les “mêmes”
(à renommage et déplacement de nœuds près) ?



Chiffrez la complexité d'un algorithme “bête”

Digression : Classes de complexité (3)

Question $P = NP$

- Problème : Comment deviner une solution ?
- Dans le pire cas : Faire une recherche exhaustive
Complexité : exponentielle
- **Question ouverte** : Est-ce qu'on pourrait résoudre les problèmes de NP en temps polynomial ?

Satisfiabilité SAT et NP :

- SAT est parmi les plus difficiles de NP (NP -complet)
- Si on pouvait résoudre SAT en temps polynomial, alors aussi tous les autres problèmes de NP

Pour résumer :

- Vérifier si une interprétation est un modèle : classe P
- Trouver une interprétation qui est un modèle : NP -complet
surtout : le **pire temps connu** est exponentiel