Logique et Programmation Logique

Julien Narboux

Licence 2

September 6, 2021

Table des matières I

Table des matières II

Table des matières III

Logique et mathématiques

- Fondements des mathématiques
- Résultats négatifs

Logique et informatique

- La logique sert en programmation:
  - pour définir la sémantique des langages de programmation
  - pour la spécifications de programmes
  - pour l'analyse de programmes
  - pour la preuve de programmes
  - dans le cadre de la programmation logique (résolution de problèmes, systèmes experts, intelligence artificielle)
- La logique sert dans l'industrie; SAT et ses applications:
  - ordonnancement
  - synthèse et vérification de circuits
  - trafic aérien
- L'informatique sert en logique/mathématiques:
  - pour générer des preuves (démonstration automatique)
  - pour vérifier des preuves (assistants de preuve)

Logique et informatique

Il existe un lien profond entre logique et informatique: la correspondance de Curry-Howard.

Logique	Programmation
formule	type
preuve	programme
vérification d'une démonstration	vérification de type
normalisation des preuves	calcul

Objectifs du cours

- Comprendre la différence entre syntaxe et sémantique
- Comprendre la différence entre théorie et méta-théorie
- Découvrir la logique propositionnelle et la logique des prédicats
- Découvrir un autre style de programmation

Bibliographie

R. David, K. Nour, C. Raffalli, Introduction à la logique, théorie de la démonstration.

La logique

La logique peut être considérée comme une partie des mathématiques, c'est une science dont l'objet d'étude est le raisonnement, les théorèmes, ...
La théorie des groupes consiste à étudier les groupes.
La logique consiste à étudier les théorèmes.
On va donc être amené à énoncer des théorèmes à propos de théorèmes.
Il faut distinguer les deux !

Syntaxe vs sémantique

Syntaxe

La *syntaxe* d'un langage formel est l'ensemble des règles définissant si un mot est correct dans le langage.

Sémantique

La *sémantique* d'une langage est une relation entre les formules syntaxiquement correctes et leurs signification.

Exemple en C

`0x10` et 16 désignent l'entier mathématique: 16

René Magritte

Syntaxe vs sémantique

En logique on distinguera $\Gamma \models P$ et $\Gamma \vdash P$. Une formule est un objet syntaxique:

- de la craie sur un tableau
- des caractères dans une chaîne de caractères
- un arbre de syntaxe abstraite

On associe à une formule correcte une interprétation, un sens.

- $\models$ désigne la validité d'une formule: elle est définie à partir de toutes les interprétations possibles de la formule dans un univers donné.
- $\vdash$ désigne le fait d'être prouvable dans un système formel, i.e. l'existence d'un objet syntaxique: l'arbre de preuve.

Méta-théorie vs théorie

Il ne faut pas confondre la logique dans laquelle (appelée méta-théorie ou méta langage) on va raisonner et la logique (appelée théorie objet ou langage objet) que l'on étudie. La méta-théorie ne sera pas bien définie ce sont *les mathématiques usuelles*, tandis que la théorie objet sera définie avec précision.

Quelques méta-théorèmes courants:

- La correction d'un système de preuve formelle ($\vdash \Rightarrow \models$).
  Les formules prouvables avec le système formel sont des formules valides.
- La complétude d'un système de preuve formelle ($\models \Rightarrow \vdash$).
  Toutes les formules valides sont prouvable avec le système formel.

Table des matières

Calcul des propositions

Le calcul des propositions (aussi appelé logique propositionnelle, ou bien CP0) est une des logiques les plus simples, elle ne comporte que des *variables* et des *connecteurs* logiques.

Syntaxe

Soit V un ensemble dénombrable de symboles, dits variables propositionnelles.

$$V = \{a, b, c, d, e, \ldots\}$$

L'ensemble des formules de la logique propositionnelle est défini par *induction*:
Les formules de la logique propositionnelle L est le plus petit ensemble contenant V et clos par les opérations binaires $\wedge$, $\vee$, $\Rightarrow$ et par l'opérateur unaire $\neg$ et la constante $\bot$.

Exemples

Exemples de formules:

- a
- $a \Rightarrow b$
- $\neg(a \Rightarrow b)$
- $a \wedge b \vee c$

Exemples qui ne sont pas des formules:

- $a \Rightarrow\Rightarrow a$
- $a\neg b$

Parenthèsage

- Ordre des priorités: $\neg > \wedge > \vee$.
- Associtivité:
  - $\vee$ et $\wedge$ sont associatifs à gauche.
  - $\Rightarrow$ est associatif à droite.

Parenthèsage

- Ordre des priorités: $\neg > \wedge > \vee$.
- Associtivité:
  - $\vee$ et $\wedge$ sont associatifs à gauche.
  - $\Rightarrow$ est associatif à droite.

Exemples

- $A \Rightarrow B \Rightarrow C \Rightarrow D$
- $A \vee B \vee C$

Parenthèsage

- Ordre des priorités: $\neg > \wedge > \vee$.
- Associtivité:
  - $\vee$ et $\wedge$ sont associatifs à gauche.
  - $\Rightarrow$ est associatif à droite.

Exemples

- $A \Rightarrow (B \Rightarrow (C \Rightarrow D))$
- $A \vee B \vee C$

Parenthèsage

- Ordre des priorités: $\neg > \wedge > \vee$.
- Associtivité:
  - $\vee$ et $\wedge$ sont associatifs à gauche.
  - $\Rightarrow$ est associatif à droite.

Exemples

- $A \Rightarrow (B \Rightarrow (C \Rightarrow D))$
- $(A \vee B) \vee C$

En Ocaml

```
type formule =
  Faux
| Variable of string
| Non of formule
| Et of formule * formule
| Ou of formule * formule
| Implique of formule * formule
```

Equivalence, *etc.*

On considère que l'équivalence $A \Leftrightarrow B$ est une *notation* pour $A \Rightarrow B \wedge B \Rightarrow A$.

Remarque

On aurait pu choisir un autre ensemble de connecteurs.

- Avoir plus de connecteurs rendent les meta-theorèmes plus longs: plus de cas à traiter.
- Avoir moins de connecteurs obligent à définir des connecteurs à partir d'autres connecteurs.

Exercice

Quels sont les ensembles de connecteurs minimal ?

Sémantique

On appelle interprétation (ou valuation) une application I de V dans $\{0,1\}$. L'application I est étendue aux formules de la logique propositionnelle par récurrence sur la structure des formules:

- $I(\bot) = 0$
- $I(\neg A) = 1$ ssi $I(A) = 0$
- $I(A \vee B) = 1$ ssi $I(A) = 1$ ou $I(B) = 1$
- $I(A \wedge B) = 1$ ssi $I(A) = 1$ et $I(B) = 1$
- $I(A \Rightarrow B) = 1$ ssi $I(A) = 0$ ou $I(B) = 1$

On dit que I satisfait A ssi $I(A) = 1$.
On dit que I falsifie A ssi $I(A) = 0$.

En Ocaml

- Les valeurs de vérité sont représentées par des valeurs de type bool : 1 par `true`, 0 par `false`.
- Une interprétation peut être représentée par une liste de couples (nom de variable, valeur de vérité):
  `type interpretation = (string*bool) list`
- `interp_formule : formule -> interpretation -> bool`
- ```
  let rec interp_formule f i = match f with
  | Faux
  | Variable v -> ....
  ```
  ```

Tables de vérités

Définition

On appelle table de vérité la représentation sous forme d'un tableau de la fonction qui associe à chaque interprétation des variables propositionnelles dans $\{0, 1\}$ associe la valeur $I(f)$.

A	B	$A \wedge B$	$A \vee B$	$A \Rightarrow B$	$\neg A$
0	0	0	0	1	1
0	1	0	1	1	1
1	0	0	1	0	0
1	1	1	1	1	0

Les 16 connecteurs logiques d'arité 2

A	B	$\bot$	$\wedge$	?	?	?	?	*xor*	$\vee$	$\neg\vee$	$\Leftrightarrow$	?	?	?	?	$\neg\wedge$	$\top$
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

L'implication

Remarque:

L'implication ne représente pas un lien de cause à effet.

Exemple:

L'augmentation de la consommation de glaces entraine une augmentation du nombre de noyades.

Formules satisfiables, insatisfiables

satisfiable Une formule A est satisfiable s'il existe **au moins une** interprétation I qui satisfait A.

satisfiable Un ensemble de formules Γ est satisfiable s'il existe **au moins une** interprétation I qui satisfait toutes les formules de Γ.

insatisfiable Une formule A est insatisfiable (contradictoire) si elle n'est pas satisfiable.

insatisfiable Un ensemble de formules Γ est insatisfiable (contradictoire) si il n'est pas satisfiable.

Formules valides, conséquence

validité Une formule A est valide (tautologie) si **toute** interprétation satisfait A.
On note: $\models A$

conséquence Une formule A est conséquence logique d'un ensemble de formules Γ si toute interprétation qui satisfait Γ satisfait A.
On note: $\Gamma \models A$

Exemples

$$\models P \vee \neg P$$

$$\{P, Q\} \models Q \wedge P$$

Exercices

Montrer que:

$$\models (A \Rightarrow B) \vee \neg(A \Rightarrow B)$$

$$\{A \Rightarrow B\} \models \neg B \Rightarrow \neg A$$

Définition de l'équivalence sémantique

Deux formules A et B sont équivalentes (noté $A \equiv B$) ssi $\{A\} \models B$ et $\{B\} \models A$.

Meta-théorème de la déduction

$\Sigma \models F \Rightarrow G$ ssi $\Sigma, F \models G$

Preuve

- Supposons que $\Sigma \models F \Rightarrow G$ montrons que $\Sigma, F \models G$. Soit I une interprétation telle que I satisfait Σ, F. I satisfait Σ donc I satisfait $F \Rightarrow G$. Or $I(F) = 1$ donc $I(G) = 1$.
- Supposons que $\Sigma, F \models G$ montrons que $\Sigma \models F \Rightarrow G$. Soit I une interprétation qui satisfait Σ. Si I satisfait F alors I satisfait G et donc I satisfait $F \Rightarrow G$. Sinon I ne satisfait pas F. Donc I satifsait $F \Rightarrow G$.

Remarque

$A \equiv B$ ssi $\vDash A \Leftrightarrow B$

Quelques équivalences utiles

Lois de De Morgan

- $\models \neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$
- $\models \neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$

Lois de ditributivité

- $\models (A \vee B) \wedge C \Leftrightarrow (A \wedge C) \vee (B \wedge C)$
- $\models (A \wedge B) \vee C \Leftrightarrow (A \vee C) \wedge (B \vee C)$

Associativité/Commutativité

- $\models ((A \vee B) \vee C) \Leftrightarrow (A \vee (B \vee C))$
- $\models ((A \wedge B) \wedge C) \Leftrightarrow (A \wedge (B \wedge C))$
- $\models (A \vee B) \Leftrightarrow (B \vee A)$
- $\models (A \wedge B) \Leftrightarrow (B \wedge A)$

Quelques équivalences utiles

Règle de l'implication

- $\models A \Rightarrow B \Leftrightarrow \neg A \vee B$

Règles de simplification

- $\models \neg\neg A \Leftrightarrow A$
- $\models A \wedge \neg A \Leftrightarrow \bot$
- $\models A \wedge \bot \Leftrightarrow \bot$
- $\models A \wedge \top \Leftrightarrow A$
- $\models A \vee \neg A \Leftrightarrow \top$
- $\models A \vee \bot \Leftrightarrow A$
- $\models A \vee \top \Leftrightarrow \top$
- $\models A \wedge A \Leftrightarrow A$
- $\models A \vee A \Leftrightarrow A$

Quelques équivalences utiles

Implications imbriquées

- $\vDash (A \Rightarrow (B \Rightarrow G)) \Leftrightarrow (A \wedge B \Rightarrow G)$

Quelques équivalences utiles

Implications imbriquées

- $\models (A \Rightarrow (B \Rightarrow G)) \Leftrightarrow (A \wedge B \Rightarrow G)$
- $\models (H_1 \Rightarrow (H_2 \Rightarrow (H_3 \Rightarrow G))) \Leftrightarrow (H_1 \wedge H_2 \wedge H_3 \Rightarrow G)$

Quelques équivalences utiles

Implications imbriquées

- $\models (A \Rightarrow (B \Rightarrow G)) \Leftrightarrow (A \wedge B \Rightarrow G)$
- $\models (H_1 \Rightarrow (H_2 \Rightarrow (H_3 \Rightarrow G))) \Leftrightarrow (H_1 \wedge H_2 \wedge H_3 \Rightarrow G)$

Simplification

- $\models (A \vee P) \wedge (A \vee \neg P) \iff A$

Substitution

Si F et G sont des formules et p une variable propositionnelle, on définit par induction la formule $F[p := G]$ obtenue à partir de F en substituant G à p dans F.

- $\bot[p := G] = \bot$
- $q[p := G] = G$ si $q = p$; q sinon
- $(\neg F)[p := G] = \neg(F[p := G])$
- $(F_1 \wedge F_2)[p := G] = (F_1[p := G]) \wedge (F_2[p := G])$
- $(F_1 \vee F_2)[p := G] = (F_1[p := G]) \vee (F_2[p := G])$
- $(F_1 \Rightarrow F_2)[p := G] = (F_1[p := G]) \Rightarrow (F_2[p := G])$

Principe d'induction sur les formules

1. Pour toute variable x, $P(x)$.
2. Pour toute formule F, si $P(F)$ alors $P(\neg F)$.
3. Pour toutes formules F_1 F_2, si $P(F_1)$ et $P(F_2)$ alors $P(F_1 \wedge F_2)$.
4. Pour toutes formules F_1 F_2, si $P(F_1)$ et $P(F_2)$ alors $P(F_1 \vee F_2)$.
5. Pour toutes formules F_1 F_2, si $P(F_1)$ et $P(F_2)$ alors $P(F_1 \Rightarrow F_2)$.

alors

Pour toute formule F, on a $P(F)$.

Lemme

Soit I une interprétation. Soit I_0 l'interprétation définie par $I_0(p) = I(G)$ et $I_0(q) = I(q)$ pour tout $q \neq p$. Alors $I(F[p := G]) = I_0(F)$

Par induction sur F (on applique le principe d'induction structurelle)

1. $I(\bot[p := G]) = I(\bot) = 0 = I_0(\bot)$
2. soit q une variable propositionnelle.
   si $p = q$, $I(q[p := G]) = I(G) = I_0(G)$ par construction de I_0
   si $p \neq q$, $I(q[p := G]) = I(q) = I_0(q)$ par construction de I_0
3. hyps d'induction : $I(F_1[p := G]) = I_0(F_1)$ et $I(F_2[p := G]) = I_0(F_2)$.
   $I((F_1 \wedge F_2)[p := G]) = I(F_1[p := G] \wedge F_2[p := G]) = I(F_2[p := G])$ si $I(F_1[p := G]) = 1$, 0 sinon. $= I_0(F1 \wedge F2)$
4. de même pour F de la forme $F_1 \vee F_2$
5. de même pour F de la forme $F_1 \Rightarrow F_2$
6. de même pour F de la forme F_1 (1 seule hypothèse d'induction)

Lemmes de substitution

Corollaires

1. Si $\vDash F$ alors $\vDash F[p := G]$
2. Si $F \equiv F'$ alors $F[p := G] \equiv F'[p := G]$

Lemme

- Si $G \equiv G'$ alors $F[p := G] \equiv F[p := G']$

Forme clausale

Littéral

Un littéral est une variable x ou sa négation $\neg x$ (souvent noté $\overline{x}$).

Vocabulaire

On dit qu'un littéral est positif s'il ne comporte pas de négation $\neg$, négatif sinon.

Clause

Une clause est une disjonction de littéraux.

Formes canoniques

Forme normale conjonctive

On dit que F est sous forme normale conjonctive si F est une conjonction de disjonctions de formules atomiques ou de négations de formules atomiques.

Forme normale disjonctive

On dit que F est sous forme normale disjonctive si F est une disjonction de conjonction de formules atomiques ou de négations de formules atomiques.

Exemples de formules en forme normale conjonctive

$$(A \vee \neg B) \wedge (\neg C \vee A)$$

$$A$$

$$A \wedge B$$

Exemples de formules qui ne sont pas en forme normale

$$\neg(A \wedge B)$$

$$A \Rightarrow B$$

Un méta-théorème

Mise sous forme normale conjonctive

Pour toute formule F il existe une formule F' en FNC (forme normale conjonctive) telle que $F \equiv F'$.

Preuve = méthode

1. On élimine les $\Leftrightarrow$ en transformant $A \Leftrightarrow B$ par $A \Rightarrow B \wedge B \Rightarrow A$.
2. On élimine les $\Rightarrow$ en transformant $A \Rightarrow B$ par $\neg A \vee B$.
3. On propage les négations vers les feuilles avec les règles:
   - $\neg(A \vee B)$ devient $\neg A \wedge \neg B$
   - $\neg(A \wedge B)$ devient $\neg A \vee \neg B$
4. On simplifie les double-négations en transformant $\neg\neg A$ en A.
5. On distribue le $\wedge$ sur le $\vee$ avec les règles:
   - $A \vee (B \wedge C)$ devient $(A \vee B) \wedge (A \vee C)$
   - $(B \wedge C) \vee A$ devient $(A \vee B) \wedge (A \vee C)$
6. On simplifie.

Exemple

$\neg(A \Leftrightarrow B)$

Exemple

$$\neg(A \Leftrightarrow B)$$

$$\equiv$$

$$\neg(A \Rightarrow B \wedge B \Rightarrow A)$$

Exemple

$$\neg(A \Leftrightarrow B)$$

$$\equiv$$

$$\neg(A \Rightarrow B \wedge B \Rightarrow A)$$

$$\equiv$$

$$\neg((\neg A \vee B) \wedge (\neg B \vee A))$$

Exemple

$$\neg(A \Leftrightarrow B)$$

$$\equiv$$

$$\neg(A \Rightarrow B \wedge B \Rightarrow A)$$

$$\equiv$$

$$\neg((\neg A \vee B) \wedge (\neg B \vee A))$$

$$\equiv$$

$$(\neg(\neg A \vee B)) \vee (\neg(\neg B \vee A))$$

Exemple

$$\neg(A \Leftrightarrow B)$$

$$\equiv$$

$$\neg(A \Rightarrow B \wedge B \Rightarrow A)$$

$$\equiv$$

$$\neg((\neg A \vee B) \wedge (\neg B \vee A))$$

$$\equiv$$

$$(\neg(\neg A \vee B)) \vee (\neg(\neg B \vee A))$$

$$\equiv$$

$$(A \wedge \neg B) \vee (B \wedge \neg A)$$

Exemple

$$\neg(A \Leftrightarrow B)$$

$$\equiv$$

$$\neg(A \Rightarrow B \wedge B \Rightarrow A)$$

$$\equiv$$

$$\neg((\neg A \vee B) \wedge (\neg B \vee A))$$

$$\equiv$$

$$(\neg(\neg A \vee B)) \vee (\neg(\neg B \vee A))$$

$$\equiv$$

$$(A \wedge \neg B) \vee (B \wedge \neg A)$$

$$\equiv$$

$$(A \vee (B \wedge \neg A)) \wedge (\neg B \vee (B \wedge \neg A))$$

Exemple

$$\neg(A \Leftrightarrow B)$$

$$\equiv$$

$$\neg(A \Rightarrow B \wedge B \Rightarrow A)$$

$$\equiv$$

$$\neg((\neg A \vee B) \wedge (\neg B \vee A))$$

$$\equiv$$

$$(\neg(\neg A \vee B)) \vee (\neg(\neg B \vee A))$$

$$\equiv$$

$$(A \wedge \neg B) \vee (B \wedge \neg A)$$

$$\equiv$$

$$(A \vee (B \wedge \neg A)) \wedge (\neg B \vee (B \wedge \neg A))$$

$$\equiv$$

$$(A \vee (B \wedge \neg A)) \wedge (\neg B \vee (B \wedge \neg A))$$

Exemple

$$\neg(A \Leftrightarrow B)$$

$$\equiv$$

$$\neg(A \Rightarrow B \wedge B \Rightarrow A)$$

$$\equiv$$

$$\neg((\neg A \vee B) \wedge (\neg B \vee A))$$

$$\equiv$$

$$(\neg(\neg A \vee B)) \vee (\neg(\neg B \vee A))$$

$$\equiv$$

$$(A \wedge \neg B) \vee (B \wedge \neg A)$$

$$\equiv$$

$$(A \vee (B \wedge \neg A)) \wedge (\neg B \vee (B \wedge \neg A))$$

$$\equiv$$

$$(A \vee (B \wedge \neg A)) \wedge (\neg B \vee (B \wedge \neg A))$$

$$\equiv$$

$$((A \vee B) \wedge (A \vee \neg A)) \wedge ((\neg B \vee B) \wedge (\neg B \vee \neg A))$$

Exemple

$$\neg(A \Leftrightarrow B)$$

$$\equiv$$

$$\neg(A \Rightarrow B \wedge B \Rightarrow A)$$

$$\equiv$$

$$\neg((\neg A \vee B) \wedge (\neg B \vee A))$$

$$\equiv$$

$$(\neg(\neg A \vee B)) \vee (\neg(\neg B \vee A))$$

$$\equiv$$

$$(A \wedge \neg B) \vee (B \wedge \neg A)$$

$$\equiv$$

$$(A \vee (B \wedge \neg A)) \wedge (\neg B \vee (B \wedge \neg A))$$

$$\equiv$$

$$(A \vee (B \wedge \neg A)) \wedge (\neg B \vee (B \wedge \neg A))$$

$$\equiv$$

$$((A \vee B) \wedge (A \vee \neg A)) \wedge ((\neg B \vee B) \wedge (\neg B \vee \neg A))$$

$$\equiv$$

$$(A \vee B) \wedge (\neg B \vee \neg A)$$

Attention !

La formule peut grandir de manière exponentielle.

Clause de Horn

Définition

Une clause de Horn est une clause qui possède au plus un littéral positif.

Application

Les clauses de Horn jouent un rôle fondamental en programmation logique.

Méta-théorème

Décidabilité

La logique propositionnelle est décidable: il existe un algorithme qui termine dans tous les cas et qui permet de déterminer si une formule est satisfiable/valide.

Tableaux sémantiques

- Méthode pour décider de la satisfiabilité/validité d'une formule en logique propositionnelle.
- Idée: chercher un modèle en décomposant la formule en des ensembles de littéraux.
- Pour tester si un ensemble de littéraux est satisfiable, il suffit de regarder s'il contient un littéral et sa négation.
- La formule est satisfiable ssi un des ensembles de littéraux est satisfiable.

$A = p \wedge (\neg q \vee \neg p)$
$I(A) = 1$ ssi $I(p) = 1$ et $I(\neg q \vee \neg p) = 1$.
$I(\neg q \vee \neg p) = 1$ ssi $I(\neg q) = 1$ ou $I(\neg p) = 1$.
Pour résumer: $I(A) = 1$ ssi

1. $I(p) = 1$ et $I(\neg q) = 1$ ou
2. $I(p) = 1$ et $I(\neg p) = 1$.

Exemple

$$\dfrac{\dfrac{\dfrac{o}{p, \neg p, \neg q} \quad \dfrac{x}{q, \neg p, \neg q}}{p \vee q, \neg p, \neg q}}{\dfrac{p \vee q, \neg p \wedge \neg q}{(p \vee q) \wedge (\neg p \wedge \neg q)}}$$

Représentation sous forme d'arbre

La formule de départ est la racine de l'arbre.
Les feuilles sont constituées d'un ensemble de littéraux.
Les nœuds sont d'arité 1 ou 2.

Formules conjonctives

α	α_1	α_2
$\neg\neg A_1$	A_1	
$A_1 \wedge A_2$	A_1	A_2
$\neg(A_1 \vee A_2)$	$\neg A_1$	$\neg A_2$
$\neg(A_1 \Rightarrow A_2)$	A_1	$\neg A_2$
$A_1 \iff A_2$	$A_1 \Rightarrow A_2$	$A_2 \Rightarrow A_1$

Formules disjonctives

β	β_1	β_2
$A_1 \vee A_2$	B_1	B_2
$\neg(B_1 \wedge B_2)$	$\neg B_1$	$\neg B_2$
$(B_1 \Rightarrow B_2)$	$\neg B_1$	B_2
$\neg(B_1 \iff B_2)$	$\neg(B_1 \Rightarrow B_2)$	$\neg(B_2 \Rightarrow B_1)$

Theorem

A insatisfiable ssi le tableau sémantique de A est fermé.
A satisfiable ssi le tableau sémantique de A est ouvert.
A est valide ssi le tableau sémantique de $\neg A$ est fermé.

Table des matières

Systèmes formels

Un système formel est composé de 3 éléments:

1. le langage: un ensemble de formules
2. des axiomes: un sous ensemble des formules
3. des règles de déduction (ou règles d'inférence): des fonctions qui associent des formules à d'autres formules

Remarque: les axiomes peuvent être vus comme des règles sans prémisses.

Systèmes formels

Les manipulations effectuées dans un système formel sont purement syntaxiques et n'utilisent aucunement le sens que l'on peut associer aux formules.

Il est fréquent de noter les règles de la manière suivante:

$$\frac{P_1 \qquad P_2}{C} \text{ nom de la règle}$$

Les P_i sont appelées des prémisses, C la conclusion. La règle énonce le fait que de P_1 , P_2 on peut déduire C.

Preuve

Définition

Une preuve/déduction est un arbre dont:

- Les feuilles sont des axiomes.
- Les noeuds correspondent à des applications des règles de déduction.
- La racine est la formule prouvée.

Théorème

Définition

Un théorème est une formule dont il existe une preuve.

Quelle est la différence entre un théorème, un lemme, un corollaire ?

Du point de vue de la logique formelle:

aucune.

Dans la pratique des mathématiques:

lemme	assertion servant d'intermédiaire pour démontrer un théorème
corollaire	assertion qui découle directement d'un théorème

Quelle est la différence entre une preuve mathématique et une preuve formelle ?

On pourrait dire qu'une preuve mathématique est une suite d'arguments pour convaincre un mathématicien de l'existence d'une preuve formelle.

Exemple 1

- Le langage: $L = \{(a, b) \in \mathbb{N}^2\}$
- Règles de déduction:

$$\frac{(n, n)}{(n+1, n+1)}\ n \in \mathbb{N}$$

- Axiomes: $A = \{(0, 0)\}$

Exemples de théorèmes

$(0,0)$, $(3,3)$, $(4,4)\ldots$

Exemple de déduction

$$\dfrac{\dfrac{\dfrac{}{(0,0)}}{(1,1)}}{(2,2)}}{(3,3)}$$

Exemple de formule qui n'est pas un théorème

$$(2, 3)$$

Pourquoi ?

Exemple de formule qui n'est pas un théorème

$$(2, 3)$$

Pourquoi ?

Si c'était un théorème alors il existerait un arbre de déduction dont c'est la racine, mais:

- Ce n'est pas un axiome.
- Aucune règle ne s'applique.

C'est donc impossible.

Remarque

Nous venons d'utiliser une méthode *syntaxique* pour montrer qu'une formule n'est pas un théorème. Il existe des méthodes *sémantiques*.

Correction, complétude

Correction

On dit qu'un système est correct quand tous les théorèmes sont des formules valides.

Complétude

On dit qu'un système est complet quand toutes les formules valides sont des théorèmes.

Reprenons notre exemple

Définissons la validité pour cet exemple

On dit qu'une formule $F = (a, b)$ est valide ssi $a = b$.

L'exemple 1 est-il un système correcte ?

L'exemple 1 est-il un système complet ?

Cohérence, consistance

Cohérence

On dit qu'un système formel est cohérent si toutes les formules ne sont pas des théorèmes.

Consistance

On dit qu'un système formel (comportant un symbole de négation $\neg$) est consistant s'il existe aucune formule F telle que F et $\neg F$ soient des théorèmes.

L'exemple 1 est-il un système cohérent ?

Saturé

On dit qu'un système formel S est saturé si pour toute formule F qui n'est pas une théorème, le système formel S' obtenu en ajoutant F aux axiomes de S est inconsistant.

Décidable

On dit qu'un système formel S est décidable, si le problème de savoir si une formule est un théorème de S est décidable.

L'exemple 1 est-il un système saturé ?

L'exemple 1 est-il un système décidable ?

Exemple 2

- Le langage: $L = \{(a, b) \in \mathbb{N}^2\}$
- Règles de déduction:

$$\frac{(n, n)}{(n+1, n+1)}\ n \in \mathbb{N}$$

- Axiomes: $A = \{(1, 1)\}$

Exemple 3

- Le langage: $L = \{(a, b) \in \mathbb{N}^2\}$
- Règles de déduction:

$$\frac{(n, n)}{(n+2, n+2)}\ n \in \mathbb{N}$$

- Axiomes: $A = \{(0, 0)\}$

Exemple 4

- Le langage: $L = \{(a, b) \in \mathbb{N}^2\}$
- Règles de déduction: $\varnothing$
- Axiomes: $A = \{(n, n), n \in \mathbb{N}\}$

Exemple 5

- Le langage: $L = \{(a, b) \in \mathbb{N}^2\}$
- Règles de déduction:

$$\frac{(n, n)}{(n+1, n)}\ n \in \mathbb{N}$$

$$\frac{(n, n)}{(n, n+1)}\ n \in \mathbb{N}$$

- Axiomes: $A = \{(0, 0)\}$

Table des matières

Quelques exemples de définitions à l'aide de systèmes formels:

- λ-calcul
- sémantique

Lambda-termes

Définition

Les règles de formation des termes du λ-calcul sont les suivantes:

1 variable Toute variable est un terme.

2 abstraction Si x est une variable et M un terme, alors $x \mapsto M$ est un terme.

3 application Si M et N sont des termes, alors $M(N)$ est un terme.

Autre notation pour cette définition:

$$M, N ::= V | x \mapsto M | M(N)$$

Lambda-termes, notation historique

Définition

Les règles de formation des termes du λ-calcul sont les suivantes:

1 variable Toute variable est un terme.

2 abstraction Si x est une variable et M un terme, alors $(\lambda x.M)$ est un terme.

3 application Si M et N sont des termes, alors (MN) est un terme.

Autre notation pour cette définition:

$$M, N ::= V|\lambda x.M|MN$$

Jugement de typage

Un environnement est un ensemble d'association de types à des variables.

$$\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$$

Un *jugement* de typage est l'affirmation du type d'un terme dans un environnement.

$$\Gamma \vdash M : \sigma$$

Le terme M possède le type σ dans l'environnement Γ.

Règles de typage du lambda calcul simplement typé

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\ N : \tau}\ \text{App}$$

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma}\ \text{Var}$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau}\ \text{Abs}$$

Exemples

1. $\lambda x.x$
2. $(\lambda x.x)(\lambda x.x)$

Exemples

1. $\lambda x.x$
2. $(\lambda x.x)(\lambda x.x)$

$$\dfrac{\dfrac{}{x : \tau \vdash x : \tau}\,\text{Var}}{\vdash \lambda x.x : \tau \to \tau}\,\text{Abs}$$

Exemples

1. $\lambda x.x$
2. $(\lambda x.x)(\lambda x.x)$

$$\dfrac{\dfrac{}{x : \tau \vdash x : \tau}\,\text{Var}}{\vdash \lambda x.x : \tau \to \tau}\,\text{Abs}$$

$$\dfrac{\dfrac{\dfrac{}{x : \tau \to \tau \vdash x : \tau \to \tau}\,\text{Var}}{\vdash \lambda x.x : (\tau \to \tau) \to (\tau \to \tau)}\,\text{Abs} \qquad \dfrac{\dfrac{}{x : \tau \vdash x : \tau}\,\text{Var}}{\vdash \lambda x.x : \tau \to \tau}\,\text{Abs}}{\vdash (\lambda x.x)(\lambda x.x) : \tau \to \tau}\,\text{App}$$

Table des matières

Un exemple idiot

Considérons le système formel suivant:

- Le langage est composé des variables et des connecteurs: $\Rightarrow, \wedge, \vee, \forall, \exists$
- Les axiomes: l'ensemble des formules valides en logique des prédicats
- Règles d'inférence: aucune

Étudions ce système

- L'ensemble des théorèmes de ce système est le même que l'ensemble des axiomes.
- Donc tous les théorèmes sont des formules valides: ce système est correct.
- Il est aussi complet car toutes les formules valides sont des théorèmes.

Mais !

Ce système formel n'a pas d'intérêt car il est difficile de savoir si une formule est un axiome. Il faut donc compléter notre définition de système formel pour interdire ce genre d'exemples.

Problème

- Il faut que l'on puisse dire sans avoir à réfléchir si une formule est un axiome.
- Il faut que l'on puisse dire sans avoir à réfléchir si une formule est une conséquence d'autres au moyen d'une règle d'inférence.

Ensemble récursif

Attention !

Le mot récursif ici ne fait pas directement référence aux fonctions qui s'appellent elles-même récursivement.

Définition

On dit qu'un ensemble est récursif (ou décidable) si il s'agit d'un ensemble dont le test d'appartenance peut être réalisé par un programme informatique qui s'arrête toujours (sans tenir compte des limites de mémoire ou de temps de calcul des ordinateurs actuellement réalisables).

Ensemble récursivement énumérable

Définition

Un ensemble récursivement énumérable ou semi-décidable est un ensemble qui est le domaine de définition d'une fonction calculable.

Théorème

Le complémentaire d'un ensemble décidable est décidable.

Théorème

Tout ensemble récursif est récursivement énumérable.

Mais l'inverse n'est pas vrai !

Théorème

Un ensemble d'entiers E est récursif ssi E et $\mathbb{N}/E$ sont récursivement énumérables.

Corollaire

Une théorie récursivement axiomatisable et complète est décidable.

Décidabilité de la déduction

L'ensemble des déductions est décidable (récursif).

Enumerabilité des théorèmes

L'ensemble des théorèmes est semi-décidable (récursivement énumérable).

Attention !

Ne pas confondre décidabilité logique et décidabilité algorithmique. Dans ce cours on utilisera le mot décidabilité pour désigner la décidabilité algorithmique. Mais parfois, on dit qu'un proposition P est indécidable, pour exprimer le fait qu'elle est indépendante d'un certain ensemble d'axiomes Γ i.e. que ni $\Gamma \vdash P$ ni $\Gamma \vdash \neg P$ sont prouvables . C'est un abus de langage. C'est le cas notamment du célèbre postulat des Parallèles d'Euclide qui est indépendant des autres axiomes de la géométrie.

Formalismes pour la logique

1. Systèmes à la Hilbert
2. Systèmes à base de séquents
   - Déduction naturelle
   - Calcul de Gentzen

Table des matières

David Hilbert (1862-1943)

Les systèmes *à la Hilbert*

Les systèmes *à la Hilbert* comportent

- de nombreux schémas d'axiomes,
- peu de règles d'inférence (souvent une seule).
- On ne manipule pas les hypothèses explicitement.

La logique propositionnelle minimale

- Une syntaxe très simple.
- Deux axiomes.
- Une règle.

Langage

On se donne un seul connecteur binaire noté $\Rightarrow$ et des variables propositionnelles.

Exemples de formules

- P
- $A \Rightarrow A$
- $(A \Rightarrow B) \Rightarrow (B \Rightarrow A)$

Une règle

Le modus ponens:

$$\frac{A \Rightarrow B \qquad A}{B}\ \mathrm{MP}$$

Deux schémas d'axiomes

1. Axiome K: $A \Rightarrow (B \Rightarrow A)$
2. Axiome S: $(A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))$

Schémas d'axiomes

Définition

Un schéma d'axiomes est la définition d'une infinité d'axiomes à l'aide d'un motif.

Exemple:

Les axiomes suivants sont des instances du schéma $A \Rightarrow (B \Rightarrow A)$:

- $x \Rightarrow (y \Rightarrow x)$
- $(a \Rightarrow b) \Rightarrow (c \Rightarrow (a \Rightarrow b))$
- $(a \Rightarrow b) \Rightarrow ((a \Rightarrow b) \Rightarrow (a \Rightarrow b))$

Les systèmes à la Hilbert sont

- compliqués pour construire des preuves *dans* le système.
- plus adaptés pour construire des preuves *à propos* du système.

Exemple

$$\dfrac{\dfrac{\dfrac{}{(A \Rightarrow ((B \Rightarrow A) \Rightarrow A)) \Rightarrow ((A \Rightarrow (B \Rightarrow A)) \Rightarrow (A \Rightarrow A))}\,\mathrm{S} \qquad \dfrac{}{(A \Rightarrow ((B \Rightarrow A) \Rightarrow A)}\,\mathrm{K}}{(A \Rightarrow (B \Rightarrow A)) \Rightarrow (A \Rightarrow A)}\,\mathrm{MP} \qquad \dfrac{}{A \Rightarrow (B \Rightarrow A)}\,\mathrm{K}}{A \Rightarrow A}\,\mathrm{MP}$$

Le modèle

On utilise l'ensemble $\{0,1\}$ comme modèle. On fixe l'interprétation du symbole $\Rightarrow$ par la fonction de $\{0,1\}^2$ dans $\{0,1\}$ suivante:

A	B	$A \Rightarrow B$
0	0	1
0	1	1
1	0	0
1	1	1

Exercice

1. Montrer que les axiomes sont valides.
2. Montrer que la règle du modus ponens préserve la validité.
3. Que peut-on en déduire ?

Incomplétude

Formule de Peirce

$$((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$$

Exercice: montrer que la formule de Peirce est valide.
On peut montrer que la formule de Peirce n'est pas un théorème de la logique minimale: la logique minimale est donc incomplète.

Solutions

On peut soit:

1. trouver un modèle adapté à la logique minimale ou bien,
2. modifier le système pour en trouver un qui corresponde à notre modèle.

Logique propositionnelle à la Hilbert

On élargit le langage avec les connecteurs $\wedge$, $\vee$ et $\neg$.

Exemples de formules

$$A \wedge B \Rightarrow C \vee A$$

$$A \Rightarrow A$$

10 schémas d'axiomes, une règle.

Les schémas d'axiomes I

Règles de l'implication

$$P \Rightarrow Q \Rightarrow P$$

$$P \Rightarrow (Q \Rightarrow R) \Rightarrow ((P \Rightarrow Q) \Rightarrow (P \Rightarrow R))$$

Règles de la conjonction

$$A \wedge B \Rightarrow A$$

$$A \wedge B \Rightarrow B$$

$$A \Rightarrow (B \Rightarrow (A \wedge B))$$

Les schémas d'axiomes II

Règles de la disjonction

$$(P \Rightarrow R) \Rightarrow ((Q \Rightarrow R) \Rightarrow (P \vee Q \Rightarrow R))$$

$$P \Rightarrow (P \vee Q)$$

$$Q \Rightarrow (P \vee Q)$$

Règle de la négation

$$(\neg P \Rightarrow Q) \Rightarrow ((\neg P \Rightarrow \neg Q) \Rightarrow P)$$

$$(\neg P \Rightarrow \neg Q) \Rightarrow (Q \Rightarrow P)$$

La règle

Le modus ponens:

$$\frac{A \Rightarrow B \qquad A}{B}\ \text{MP}$$

Méta-théorèmes

Théorème d'adéquation/correction

Si $\vdash A$ alors $\models A$

Théorème de complétude

Si $\models A$ alors $\vdash A$

Théorème de la déduction

$A \vdash B$ ssi $\vdash A \Rightarrow B$

Décidabilité

La logique propositionnelle est *décidable*.

Cohérence

La logique propositionnelle est *cohérente*.

Table des matières

Gerhard Gentzen (1909-1945)

Notion de séquent

Les systèmes formels modélisant des logiques utilisent souvent un langage basé sur une structure appelée **séquent**. Il s'agit d'une paire (Γ, F) composée

- d'un multi-ensemble de formules Γ (l'ordre ne compte pas, répétitions possibles) et
- d'une formule F.

Traditionnellement on note cette paire:

$$\Gamma \vdash F$$

Intuitivement, un séquent permet de représenter le fait que des hypothèses Γ, on peut déduire F.

Remarque

La logique minimale à la Hilbert peut aussi être formalisée sous formes de séquents dans lesquels Γ est toujours vide:

- Le langage est composé des séquents $\vdash F$ ou F est un terme composé avec des variables propositionnelle et le connecteur $\Rightarrow$.
- Les axiomes sont:

$$\frac{}{\vdash A \Rightarrow (B \Rightarrow A)}$$

$$\frac{}{\vdash (A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))}$$

- Le modus ponens s'écrit:

$$\frac{\vdash A \Rightarrow B \qquad \vdash A}{\vdash B}\ \mathrm{MP}$$

Déduction naturelle

- On utilise des séquents.
- On ne manipule pas les hypothèses.

Règles pour la logique minimale

$$\frac{}{\Gamma \vdash A} \text{ si } A \in \Gamma$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \text{ Intro } \Rightarrow \qquad \frac{\Gamma \vdash A \Rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B} \text{ Elim } \Rightarrow$$

Preuve de la formule K

$$\dfrac{\dfrac{A, B \vdash A}{A \vdash B \Rightarrow A}\,\text{Intro} \Rightarrow}{\vdash A \Rightarrow B \Rightarrow A}\,\text{Intro} \Rightarrow$$

Preuve de la formule S

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A \Rightarrow B \Rightarrow C, A \Rightarrow B, A \vdash A \Rightarrow B \Rightarrow C \qquad A \Rightarrow B \Rightarrow C, A \Rightarrow B, A \vdash A}{A \Rightarrow B \Rightarrow C, A \Rightarrow B, A \vdash B \Rightarrow C}\,\text{MP} \qquad \ldots\ \text{X}\ \ldots}{A \Rightarrow B \Rightarrow C, A \Rightarrow B, A \vdash C}\,\text{MP}}{A \Rightarrow B \Rightarrow C, A \Rightarrow B \vdash A \Rightarrow C}\,\text{Intro} \Rightarrow}{A \Rightarrow B \Rightarrow C \vdash (A \Rightarrow B) \Rightarrow A \Rightarrow C}\,\text{Intro} \Rightarrow}{\vdash (A \Rightarrow B \Rightarrow C) \Rightarrow (A \Rightarrow B) \Rightarrow A \Rightarrow C}\,\text{Intro} \Rightarrow$$

X:

$$\dfrac{A \Rightarrow B \Rightarrow C, A \Rightarrow B, A \vdash A \Rightarrow B \qquad A \Rightarrow B \Rightarrow C, A \Rightarrow B, A \vdash A}{A \Rightarrow B \Rightarrow C, A \Rightarrow B, A \vdash B}\,\text{MP}$$

Théorèmes

Les théorèmes sont les formules ϕ telles que $\vdash \phi$ est un théorème du système formel.

Metathéorème

Les formules qui sont des théorèmes de la logique minimale propositionnelle à la Hilbert sont aussi des théorèmes de la logique minimale formalisée en déduction naturelle.

Autre exemple

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \Rightarrow B, C \Rightarrow A, C \vdash A \Rightarrow B} \quad \dfrac{\text{... X ...}}{A \Rightarrow B, C \Rightarrow A, C \vdash A}}{A \Rightarrow B, C \Rightarrow A, C \vdash B}\,\text{MP}}{A \Rightarrow B, C \Rightarrow A \vdash C \Rightarrow B}\,\text{Intro} \Rightarrow}{A \Rightarrow B \vdash (C \Rightarrow A) \Rightarrow C \Rightarrow B}\,\text{Intro} \Rightarrow}{\vdash (A \Rightarrow B) \Rightarrow (C \Rightarrow A) \Rightarrow C \Rightarrow B}\,\text{Intro} \Rightarrow$$

X:

$$\dfrac{\dfrac{}{A \Rightarrow B, C \Rightarrow A, C \vdash C \Rightarrow A} \quad \dfrac{}{A \Rightarrow B, C \Rightarrow A, C \vdash C}}{A \Rightarrow B, C \Rightarrow A, C \vdash A}$$

Logique propositionnelle

Syntaxe

- variables propositionnelles
- connecteurs: $\wedge$, $\vee$, $\neg$, $\Rightarrow$

Exemples

$$A \wedge B \Rightarrow B \wedge A$$

Logique propositionnelle intuitionniste

Règles de la logique minimale :

$$\frac{}{\Gamma \vdash A} \text{ si } A \in \Gamma$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \text{ Intro } \Rightarrow \qquad \frac{\Gamma \vdash A \Rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B} \text{ Elim } \Rightarrow$$

$+ \ldots$

Règles du $\bot$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash P}\ \text{Elim}\ \bot$$

Règle du $\neg$

$$\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}\ \text{Intro}\ \neg$$

$$\frac{\Gamma \vdash \neg A \qquad \Gamma \vdash A}{\Gamma \vdash \bot}\ \text{Elim}\ \neg$$

Règles du $\wedge$

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q}\ \text{Intro}\ \wedge$$

$$\frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash P}\ \text{Elim}\ \wedge\text{g}$$

$$\frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash Q}\ \text{Elim}\ \wedge\text{d}$$

Règles du $\vee$

$$\frac{\Gamma \vdash P}{\Gamma \vdash P \vee Q} \text{ Intro } \vee\text{g}$$

$$\frac{\Gamma \vdash Q}{\Gamma \vdash P \vee Q} \text{ Intro } \vee\text{d}$$

$$\frac{\Gamma \vdash P \vee Q \qquad \Gamma, P \vdash R \qquad \Gamma, Q \vdash R}{\Gamma \vdash R} \text{ Elim } \vee$$

Exemple de preuve

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \wedge B \vdash A \wedge B}}{A \wedge B \vdash B}\ \text{Elim}\ \wedge d \qquad \dfrac{\dfrac{}{A \wedge B \vdash A \wedge B}}{A \wedge B \vdash A}\ \text{Elim}\ \wedge g}{A \wedge B \vdash B \wedge A}\ \text{Intro}\ \wedge}{\vdash A \wedge B \Rightarrow B \wedge A}\ \text{Intro}\ \Rightarrow$$

Exercices

Prouver que:

- $\vdash A \vee B \Rightarrow B \vee A$
- $\vdash A \Rightarrow \neg\neg A$

Solutions

$$\dfrac{\dfrac{}{A \vee B \vdash A \vee B} \qquad \dfrac{\dfrac{}{A \vee B, A \vdash A}}{A \vee B, A \vdash B \vee A}\ \text{Intro}\ \vee_d \qquad \dfrac{\dfrac{}{A \vee B, B \vdash B}}{A \vee B, B \vdash B \vee A}\ \text{Intro}\ \vee_g}{\dfrac{A \vee B \vdash B \vee A}{\vdash A \vee B \Rightarrow B \vee A}\ \text{Intro}\ \Rightarrow}\ \text{Elim}\ \vee$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A, \neg A \vdash A} \qquad \dfrac{}{A, \neg A \vdash \neg A}}{A, \neg A \vdash \bot}\ \text{Elim}\ \neg}{A \vdash \neg\neg A}\ \text{Intro}\ \neg}{\vdash A \Rightarrow \neg\neg A}\ \text{Intro}\ \Rightarrow$$

Règles dérivées

Motivations

- Pour faire des preuves *à propos* d'un système formel (méta-théorèmes) pour raccourcir les preuves il est bon d'avoir le moins de règles possible.
- Mais pour faire des preuves *au sein* d'un système formel (théorèmes) pour raccourcir les preuves il est bon d'avoir le plus de règles possibles.

Définition

Une règle dérivée est un règle d'inférence qui se démontre avec les règles de base (ou d'autres règles dérivées déjà démontrées).

Pour démontrer une règle dérivée, on doit exhiber un arbre de dérivation dont la racine est le séquent conclusion de la règle dérivée et les feuilles non fermées (non instances d'axiomes) les prémisses de la règle dérivée.

Exemple de règle dérivable

On peut couper un $\wedge$ en deux dans les hypothèses:

$$\frac{\Gamma, A, B \vdash P}{\Gamma, A \wedge B \vdash P}$$

Logique classique vs intuitionniste

Les formules suivantes sont des formules valides en logique classique:

tiers exclu $P \vee \neg P$

élimination de la double négation $\neg\neg P \Rightarrow P$

loi de Peirce $((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$

Ces propositions impliquent qu'il y a des démonstrations qui ne construisent pas l'objet satisfaisant la proposition prouvée.
Certains mathématiciens ont refusé ces propositions: Brouwer, Heyting, ...

Exemple de preuve classique

Montrons que :

$$\exists x, y \notin \mathbb{Q}, x^y \in \mathbb{Q}$$

Considérons $\sqrt{2}^{\sqrt{2}}$.

a Si $\sqrt{2}^{\sqrt{2}} \in \mathbb{Q}$.
On choisit $x = \sqrt{2}$ et $y = \sqrt{2}$.

b Sinon $\sqrt{2}^{\sqrt{2}} \notin \mathbb{Q}$.
On choisit $x = \sqrt{2}^{\sqrt{2}}$ et $y = \sqrt{2}$.

$$x^y = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{\sqrt{2}\times\sqrt{2}} = \sqrt{2}^2 = 2 \in \mathbb{Q}$$

En fait, un théorème d'analyse affirme que $\sqrt{2}^{\sqrt{2}}$ est irrationnel et que c'est le cas "a" qu'il faut choisir, mais la démonstration fondée sur le tiers exclu ne le dit pas.

Réduction à l'absurde

$$\frac{\Gamma, \neg P \vdash \bot}{\Gamma \vdash P}\ \text{RAA}$$

Exercice

Montrer que cette règle préserve la validité.

Exercices

1. $\vdash \neg\neg\neg A \Rightarrow A$
2. $\vdash A \vee \neg A$
3. $\vdash ((A \Rightarrow B) \Rightarrow A) \Rightarrow A$

Solution I

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg\neg A, \neg A \vdash \neg\neg A} \quad \dfrac{}{\neg\neg A, \neg A \vdash \neg A}}{\neg\neg A, \neg A \vdash \bot}\ \text{elim } \neg}{\neg\neg A \vdash A}\ \text{RAA}}{\vdash \neg\neg A \Rightarrow A}\ \text{intro } \Rightarrow$$

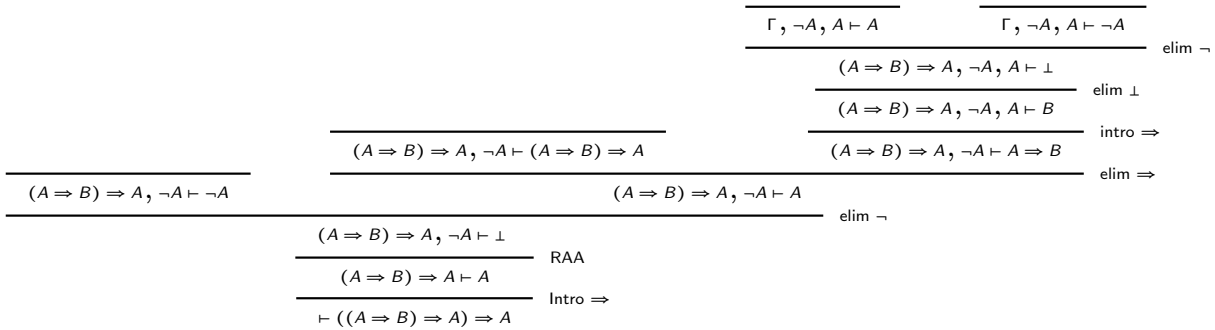
Solution II

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(A \vee \neg A), A \vdash A}}{\neg(A \vee \neg A), A \vdash A \vee \neg A}\ \text{intro } \vee\text{g} \qquad \dfrac{}{\neg(A \vee \neg A), A \vdash \neg(A \vee \neg A)}}{\neg(A \vee \neg A), A \vdash \bot}\ \text{elim } \neg}{\neg(A \vee \neg A) \vdash \neg A}\ \text{intro } \neg}{\neg(A \vee \neg A) \vdash A \vee \neg A}\ \text{intro } \vee\text{d}$$

$$\dfrac{\neg(A \vee \neg A) \vdash \neg(A \vee \neg A) \qquad \neg(A \vee \neg A) \vdash A \vee \neg A}{\neg(A \vee \neg A) \vdash \bot}\ \text{elim } \neg$$

$$\dfrac{\neg(A \vee \neg A) \vdash \bot}{\vdash A \vee \neg A}\ \text{RAA}$$

Solution III



Méta-théorèmes

Théorème d'adéquation/correction

Si $\vdash A$ alors $\models A$

Théorème de complétude

Si $\models A$ alors $\vdash A$

Théorème de la déduction

$A \vdash B$ ssi $\vdash A \Rightarrow B$

Preuve de correction

Pour prouver le théorème de correction, il faut prouver un énoncé plus général: Si $\Gamma \vdash G$ alors $\Gamma \models G$.
Preuve par induction sur la structure de la $\Gamma \vdash G$.

Table des matières

La logique des prédicats

La logique propositionnelle à laquelle on ajoute: $\forall$, $\exists$.

Signature

Une signature est composée de:

- un ensemble Σ_f de symboles de fonctions
- un ensemble Σ_P de symboles de prédicats

Terme

On appelle Σ-terme toute expression construite par application finie des règles suivantes:

- si x est une variable, alors x est un terme;
- si f est un symbole de fonction d'arité n (dans Σ) et si $t_1, \ldots, t_n$ sont n termes, alors $f(t_1, t_2, \ldots, t_n)$ est un terme.

Formule

On appelle Σ-formule toute expression construite par application finie des règles suivantes:

- si p un symbole de prédicat d'arité n (dans Σ) et si $t_1, \ldots, t_n$ sont n termes, alors $p(t_1, t_2, \ldots, t_n)$ est une formule;
- les expressions $\top$ et $\bot$ sont des formules;
- si A est une formule alors $\neg A$ est une formule;
- si A et B sont des formules alors $A \wedge B$, $A \vee B$, $A \Rightarrow B$ sont des formules;
- si x est une variable et A est une formule, alors $\forall x, A$ et $\exists x, A$ sont des formules.

Remarque: on a pas de variables propositionnelles.

Variables libres/variable liées

Une variable est liée quand le nom que l'on utilise pour définir un objet n'a pas d'importance.

Exemples:

$$\forall x, P(x) \equiv \forall y, P(y)$$

Occurences libres/liées

- Une même variable peut être liée et libre dans une formule, ou plus exactement elle peut avoir des occurrences libres et des occurrences liées.
- Une occurrence d'une variable est un endroit où la variable apparaît dans la formule (sauf derrière un quantificateur)

Variables libres/variable liées

FV : terme $\rightarrow$ ensemble de variables.

$$\begin{aligned} FV(x) &= \{x\} \\ FV(f(t_1, t_2, \ldots, t_n)) &= FV(t_1) \cup FV(t_2) \cup \ldots \cup FV(t_n) \end{aligned}$$

FV : formule $\rightarrow$ ensemble de variables.

$$
\begin{aligned}
FV(p(t_1, t_2, \ldots, t_n)) &= FV(t_1) \cup FV(t_2) \cup \ldots \cup FV(t_n) \\
FV(\neg A) &= FV(A) \\
FV(A \wedge B) &= FV(A) \cup FV(B) \\
FV(A \vee B) &= FV(A) \cup FV(B) \\
FV(A \Rightarrow B) &= FV(A) \cup FV(B) \\
FV(\exists x, P) &= FV(P) - \{x\} \\
FV(\forall x, P) &= FV(P) - \{x\}
\end{aligned}
$$

Vocabulaire

Un terme ou une formule sans variable libre est dit clos.
Une théorie est un ensemble de formules closes.

Substitution

Attention au renommage !

$$(\forall y, P)[x ::= t] = \forall y, P[x ::= t]$$

que si $x \neq y$ et $y \notin FV(t)$!

Sémantique

Pour définir l'interprétation il faut :

- un univers U
- pour chaque symbole de fonction f (des pixels allumés à l'écran), une fonction f_I de $U^n \to U$ pour interpréter les symboles de fonction.
- pour chaque symbole de prédicat p (des pixels allumés à l'écran), une fonction p_I de $U^n \to \{0, 1\}$ pour interpréter les symboles de prédicat.

Sémantique des termes

Étant donné une interprétation des variables, on définit la sémantique par récurrence sur la structure des termes.

$I_t[x] = I_V(x)$

$I_t[f(t_1, t_2, \ldots, t_n)] = f_I(I_t[t_1], I_t[t_2], \ldots, I_t[t_n])$

Sémantique des formules

On définit la sémantique par récurrence sur la structure des formules.

$$
\begin{aligned}
I_F[p(t_1, t_2, \ldots, t_n)] &= p_I(I_t[t_1], I_t[t_2], \ldots, I_t[t_n]) \\
I_F[A \wedge B] &= f_\wedge(I_F[A], I_F[B]) \\
I_F[A \vee B] &= f_\vee(I_F[A], I_F[B]) \\
I_F[\neg A] &= f_\neg(I_F[A]) \\
I_F[\forall x, P] &= min\{I_F[P[x ::= u]] \,|u \in U\} \\
I_F[\exists x, P] &= max\{I_F[P[x ::= u]] \,|u \in U\}
\end{aligned}
$$

On appelle modèle:

- un univers U non vide
- une fonction d'interprétation f_I pour chaque symbole de fonction de Σ
- une fonction d'interprétation p_I pour chaque symbole de prédicat de Σ
- une interprétation $I_V(x) \in U$ pour chaque variable libre de la formule

tels que $I[P] = V$.

- A est satisfiable si elle a un modèle.
- $\models A$ est valide si toutes les structures sont des modèles.
- $A \models B$, B est conséquence de A, si tout modèle de A est modèle de B.
- $A \equiv B$, ssi $A \models B$ et $B \models A$.

Exemples

Formule valide

$$\exists x \; \forall y \; P(x, y) \Rightarrow \forall y \; \exists x \; P(x, y)$$

Formule satisfiable mais non valide

$$\forall y \; \exists x \; P(x, y) \Rightarrow \exists x \; \forall y \; P(x, y)$$

Formule insatisfiable

$$\exists x \; P(x) \wedge \neg P(x)$$

Nécessité d'un système de preuve

Si U est infini, la définition de l'interprétation ne permet pas le calcul:

$$\begin{aligned} I_F[\forall x, P] &= min\{I_F[P[x ::= u]] \,|u \in U\} \\ I_F[\exists x, P] &= max\{I_F[P[x ::= u]] \,|u \in U\} \end{aligned}$$

Nous avons donc besoin d'un système formel de preuve.

Équivalences classiques

Permutation des quantificateurs

$$\forall x \forall y \ \phi \equiv \forall y \forall x \ \phi$$
$$\exists x \exists y \ \phi \equiv \exists y \exists x \ \phi$$

Échange de quantificateurs

$$\neg \forall x \phi \equiv \exists x \ \neg \phi$$
$$\neg \exists x \phi \equiv \forall x \ \neg \phi$$

Table des matières

Déduction naturelle pour le calcul des prédicats

On garde les règles pour la logique propositionnelle auxquelles on rajoute des règles pour $\forall$ et $\exists$.

Règles du $\forall$

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x\, A}\ \forall \text{ intro } (x \text{ n'est pas libre dans } \Gamma)$$

$$\frac{\Gamma \vdash \forall x\, A}{\Gamma \vdash A[x \leftarrow t]}\ \forall \text{ elim}$$

Règles du $\exists$

$$\frac{\Gamma \vdash A[x \leftarrow t]}{\Gamma \vdash \exists x\, A}\ \exists \text{ intro}$$

$$\frac{\Gamma \vdash \exists x\, A \qquad \Gamma, A \vdash B}{\Gamma \vdash B}\ \exists \text{ elim } (x \text{ n'est pas libre dans } \Gamma \text{ ni } B)$$

Exemples

$$\dfrac{\dfrac{\dfrac{}{\forall x\, P(x) \vdash \forall x\, P(x)}}{\forall x\, P(x) \vdash P(t)}}{\dfrac{\forall x\, P(x) \vdash \exists x\, P(x)}{\vdash \forall x\, P(x) \Rightarrow \exists x\, P(x)}}$$

Méta-théorèmes

Théorème d'adéquation

Si $\vdash A$ alors $\models A$.

Semi-décidabilité

La logique des prédicats est semi-décidable : il existe une procédure effective telle que pour toute formule A en entrée,

- si A est valide alors la procédure s'arrête et retourne 'oui'
- sinon ou bien la procédure s'arrête et retourne 'non', ou bien elle ne s'arrête pas.

Table des matières

Résolution

- La méthode de la *résolution* est une méthode de preuve *(semi-)automatique*.
- Cette méthode est à la base de la *programmation logique*.

Une méthode par réfutation

Pour prouver que $\Gamma \models A$ on va prouver que $\Gamma, \neg A \models \bot$.

Les règles de la résolution

coupure $$\frac{C \vee p \qquad D \vee \neg p}{C \vee D}$$

factorisation $$\frac{C \vee p \vee p}{C \vee p}$$

Méthode

1. Prendre la négation de la conclusion.
2. Mettre sous formes de clauses.
3. Appliquer les règles de la résolution.
   Les clauses produites peuvent être utilisées pour générer de nouvelles clauses.

Exemple

Etant donnés:

C_1 p

C_2 q

C_3 $\neg p \vee \neg q \vee r$

C_4 $\neg p \vee \neg q \vee \neg s \vee r$

C_5 $\neg r$

On dérive:

C_6 $\neg p \vee \neg q$ par résolution entre C_3 et C_5

C_7 $\neg q$ par résolution entre C_1 et C_6

C_8 $\perp$ par résolution entre C_7 et C_2

Correction et complétude

La méthode de résolution par réfutation est correcte et complète.

Optimisations

tautologies si une clause comporte à la fois A et $\neg A$ on peut la supprimer.

pures si une clause comporte un atome A (resp. $\neg A$) unifiable avec aucun atome $\neg A'$ (resp. A') dans une autre clause alors elle peut être supprimée.

Exemple du club Ecossais

- Tout membre non écossais port des chaussettes rouges
- Tout membre port un kilt ou ne porte pas de chaussettes rouges
- Les membres mariés ne sortent pas le dimanche
- Un membre sort le dimanche si et seulement s'il est écossais
- Tout membre qui porte un kilt est écossais et marié
- Tout membre écossais porte un kilt

Mise sous forme prénexe

Définition

Une formule F est sous forme prénexe si elle est de la forme : $Q_1 x_1 Q_n x_n G$ ou les Q_i sont $\forall$ ou $\exists$ et G est sans quantificateur.

Théorème

Toute formule est équivalente à une formule sous forme prénexe.

Attention

On suppose que x n'est pas libre dans C.

$$C \vee (\forall x A(x)) \equiv \forall x (C \vee A(x))$$

$$C \vee (\exists x A(x)) \equiv \exists x (C \vee A(x))$$

$$(\forall x A(x)) \vee C \equiv \forall x (A(x) \vee C)$$

$$(\exists x A(x)) \vee C \equiv \exists x (A(x) \vee C)$$

$$C \wedge (\forall x A(x)) \equiv \forall x (C \wedge A(x))$$

$$C \wedge (\exists x A(x)) \equiv \exists x (C \wedge A(x))$$

$$(\forall x A(x)) \wedge C \equiv \forall x (A(x) \wedge C)$$

$$(\exists x A(x)) \wedge C \equiv \exists x (A(x) \wedge C)$$

Méthode II

Traitement de $\neg$ et $\Rightarrow$

Attention

$$(\forall x A(x)) \Rightarrow C \equiv \exists x (A(x) \Rightarrow C)$$

$$(\exists x A(x)) \Rightarrow C \equiv \forall x (A(x) \Rightarrow C)$$

$$\neg \exists x A(x) \equiv \forall x \neg A(x)$$

$$\neg \forall x A(x) \equiv \exists x \neg A(x)$$

$$C \Rightarrow (\forall x A(x)) \equiv \forall x (C \Rightarrow A(x))$$

$$C \Rightarrow (\exists x A(x)) \equiv \exists x (C \Rightarrow A(x))$$

Exemples

- $(P(a) \wedge Q) \Rightarrow \exists x\, P(x) \equiv \exists x\, (P(a) \wedge Q) \Rightarrow P(x)$
- $\forall x\, P(x) \Rightarrow \forall z\, P(z) \equiv \ldots$
- $\forall xy\, P(x) \wedge P(y) \Rightarrow (\exists z\, P(z)) \Rightarrow (\forall x\, P(x)) \equiv \ldots$

Attention il y a plusieurs résultats possibles !

Skolémisation

Définition

Soit $Q_1 x_1 Q_2 x_2 \ldots Q_m x_m P(x_1, x_2, \ldots, x_m)$ une formule F sous forme prénexe. On appelle forme de Skolem de F la formule obtenue en enlevant tous les quantificateurs $\exists x_i$, et en remplaçant chacune des variables quantifiées existentiellement par un nouveau symbole de fonction $f_i(x_{j_1}, x_{j_2}, \ldots, x_{j_l})$ où les x_{j_i} sont les variables quantifiées universellement qui figurent avant x_i.

Théorème d'equi-satisfiabilité

Une formule close prénexe F est démontrable ssi la forme de Skolem de $\neg F$ est contradictoire (insatisfiable).

Exemples

- $\exists y\, \forall x\, P(x, y) \Rightarrow Q(x, y)$ donne : $\forall x\, P(x, c) \Rightarrow Q(x, c)$
- $\forall x\, \exists y\, P(x, y) \Rightarrow Q(x, y)$ donne : $\forall x\, P(x, f(x)) \Rightarrow Q(x, f(x))$
- $\forall x\, \exists y\, \forall z\, \exists t\, P(f(x), g(y), z, t)$ donne :
  $\forall x\, \forall z\, P(f(x), g((g_0(x)), z, g_1(x, z))$

Unification

Le problème de l'unification:
Etant donnés deux termes t_1 et t_2, comment trouver une substitution σ telle que $t_1\sigma = t_2\sigma$.

Exemples

En supposant que a et b sont des constantes et x et y des variables.

- $f(x, a)$ et $f(b, y)$: on remplace x par b et y par a: $f(a, b)$.
- $f(x, x)$ et $f(a, b)$: unification impossible.
- $f(x, x)$ et $g(a, a)$: unification impossible.
- $f(x, y)$ et $f(g(x), y)$: unification impossible.
- $f(x, y)$ et $f(z, x)$: plusieurs solutions on remplace x et y par z ou bien on remplace x, y et z par $sin(0)$.

Algorithme d'unification

x denote une variable et t un terme.

decomposition
$$\{f(t_1,\ldots,t_n) = f(u_1,\ldots,u_n)\} \cup E \to \{t_1 = u_1,\ldots,t_n = u_n\} \cup E$$

conflit
$$\{f(t_1,\ldots,t_n) = g(u_1,\ldots,u_n)\} \cup E \to Echec$$

elimination
$$\{x = t\} \cup E \to \{x = t\} \cup E[x ::= t] \text{ si } x \in FV(E) \text{ et } x \notin FV(t)$$

occurence
$$\{x = t\} \cup E \to Echec \text{ si } x \in FV(t) \text{ et } x \neq t$$

effacement
$$\{t = t\} \cup E \to E$$

inversion
$$\{t = x\} \cup E \to \{x = t\} \cup E \text{ si } t \text{ non variable.}$$

Unificateur le plus général

Définition

Soient s et t deux termes. Alors soit s et t ne sont pas unifiables, soit il existe un unificateur le plus général (mgu) σ tel que:
Pour tout unificateur σ' de s et t, il existe σ'' tel que $\sigma' = \sigma\sigma''$.

L'algorithme précédent fournit un mgu.

Exemples

En supposant que x,y,z et t sont des variables:

- unifier $u(h(x,y),g(x))$ avec $u(h(p(y),h(z,z)),t)$
- unifier $f(x,f(x,y))$ avec $f(g(y),f(g(y),z)$
- unifier $u(x,u(x,y))$ avec $u(t(y),u(t(y),t(x)))$

Résolution avec variables

1. Négation de la conclusion
2. Mise sous forme de clause:
   1. Mise en forme prénexe
   2. Skolémisation
   3. Mise sous forme normale conjonctive
3. Applications des règles

Résolution avec variables

coupure Si $mgu(p, p') = \sigma$ et $FV(C \vee p) \cap FV(D \vee \neg p') = \varnothing$

$$\frac{C \vee p \qquad D \vee \neg p'}{C\sigma \vee D\sigma}$$

factorisation Si $mgu(l, l') = \sigma$

$$\frac{C \vee l \vee l'}{C\sigma \vee l\sigma}$$

Exemple I

Etant donnés:

- $\forall x((S(x) \vee T(x)) \Rightarrow P(x))$
- $\forall x(S(x) \vee R(x))$
- $\neg R(a)$

Montrer $P(a)$.

Exemple II

Mise sous forme de clauses:

C_1 $\neg S(x) \vee P(x)$

C_2 $\neg T(x) \vee P(x)$

C_3 $S(x) \vee R(x)$

C_4 $\neg R(a)$

C_5 $\neg P(a)$

$$C_3 + C_4 \rightarrow C_6 : S(a)$$

$$C_1 + C_6 \rightarrow C_7 : P(a)$$

$$C_5 + C_7 \rightarrow \bot$$

Exemple

Étant donnés

$$
\begin{aligned}
H_1 : &\quad \forall x\ ((\exists y \neg P(x,y)) \Rightarrow \exists z (P(z,x) \wedge P(x,z))) \\
H_2 : &\quad \forall x\ \exists y\ P(x,y) \vee P(y,x) \\
H_3 : &\quad \forall xyz\ P(x,y) \wedge P(y,z) \Rightarrow P(x,z)
\end{aligned}
$$

On veut montrer que:

$$\exists x, P(x,x)$$

Mise en forme prénexe

$$
\begin{aligned}
H_1 : &\quad \forall x\ \exists z\ \forall y\ (\neg P(x,y) \Rightarrow (P(z,x) \wedge P(x,z))) \\
H_2 : &\quad \forall x\ \exists y\ P(x,y) \vee P(y,x) \\
H_3 : &\quad \forall xyz\ P(x,y) \wedge P(y,z) \Rightarrow P(x,z) \\
g : &\quad \forall x\ \neg P(x,x)
\end{aligned}
$$

Skolémisation

$$
\begin{aligned}
H_1 : &\quad \forall x\ \forall y\ (\neg P(x,y) \Rightarrow (P(f(x),x) \wedge P(x,f(x)))) \\
H_2 : &\quad \forall x\ P(x,g(x)) \vee P(g(x),x) \\
H_3 : &\quad \forall xyz\ P(x,y) \wedge P(y,z) \Rightarrow P(x,z) \\
g : &\quad \forall x\ \neg P(x,x)
\end{aligned}
$$

Mise sous forme de clauses

$$
\begin{aligned}
C_1 : &\quad P(x,y) \vee P(f(x),x) \\
C_2 : &\quad P(x,y) \vee P(x,f(x)) \\
C_3 : &\quad P(x,g(x)) \vee P(g(x),x) \\
C_4 : &\quad \neg P(x,y) \vee \neg P(y,z) \vee P(x,z) \\
C_5 : &\quad \neg P(x,x)
\end{aligned}
$$

Résolution

C_4 et C_5 donnent $C_6 : \neg P(u, v) \vee \neg P(v, u)$
factorisation C_2 donne $C_7 : P(x, f(x))$.
C_6 et C_7 donnent $C_8 : \neg P(f(x), x)$.
C_8 et C_1 donnent $C_9 : P(x, y)$.
C_9 et C_5 donnent $\perp$.

Propriétés

Théorème

La méthode de résolution est une procédure de réfutation correcte et complète.

- C'est une procédure de semi-décision pour la logique des prédicats.
- C'est une procédure de décision pour la logique propositionnelle car l'ensemble des clauses dérivables est fini.

Attention la règle de factorisation est nécessaire

Etant donnés:

$$C_1 : \neg R(x,a) \vee R(x,x) \quad C_2 : \neg R(v,f(y)) \vee \neg R(v,y)$$
$$C_3 : \neg R(z,z) \vee R(z,a) \quad C_4 : R(u,f(y)) \vee R(u,y)$$

On déduit:

$$\begin{array}{lll} C_5 : & R(f(y),a) \vee R(f(y),y) & (\text{coupure entre } C_3 \text{ et } C_4) \\ C_6 : & R(f(a),a) & (\text{factorisation de } C_5) \\ C_7 : & \neg R(f(y),a) \vee \neg (R(f(y),y) & (\text{coupure entre } C_1 \text{ et } C_2) \\ C_8 : & \neg R(f(a),a) & (\text{factorisation de } C_7) \\ C_9 : & \bot & (\text{coupure entre } C_6 \text{ et } C_8) \end{array}$$

La factorisation est nécessaire car la résolution entre clauses à deux littéraux donne des clauses à deux littéraux.

Renommage

Attention

Avant d'unifier il faut renommer les variables des deux clauses de manière à ce qu'elles n'aient plus de variables en commun.

Exemple

$R(x, a) \vee S(x)$ et $\neg R(f(x), x) \vee T(x)$
Sans renommage échec à cause de la règle "occur check".
On unifie $R(x, a)$ et $\neg R(f(y), y)$ et on obtient $S(f(a)) \vee T(a)$.

Table des matières

Les paradigmes

- Programmation impérative
- Programmation structurée
- Programmation fonctionnelle
- Programmation déclarative
- Programmation objet

Pouvoir d'expressivité

- langage programmation $\neq$ langage mathématique
- visée opérationnelle
- compromis entre *puissance d'expression* et *capacité d'exécution*

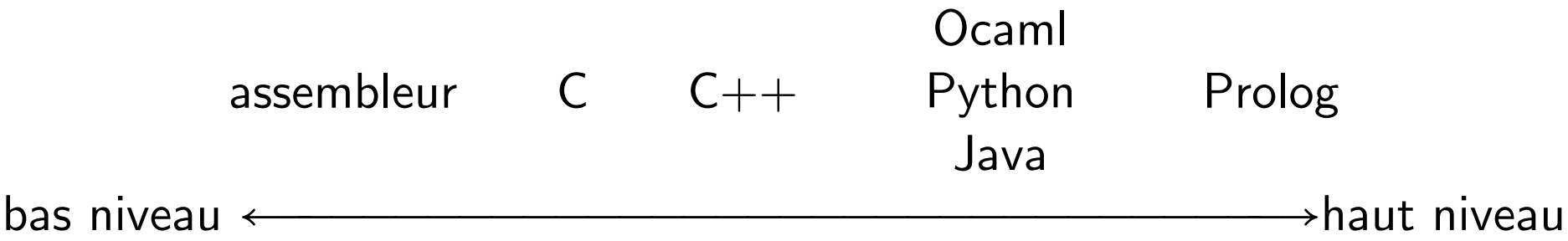
Langage de haut/bas niveau

langage de haut niveau

- Les calculs sont décrits de manière plus abstraite.
- ☺ Programmation plus facile : le programmeur ne doit pas gérer tous les détails (allocation mémoire).
- ☹ Maîtrise des détails plus faible.

langage de bas niveau

- Les calculs sont décrits dans un langage proche de l'exécution réelle sur le microprocesseur.
- ☹ Programmation plus difficile.
- ☺ Maîtrise des détails plus grande.



Exemple d'un programme de bas niveau

Mips

```
        blt   $t1, $t2, Then      # si t1 >= t2 saut à Then
        move  $t3, $t2            # t3 := t1
        j     End                 # saut à Fi
Then:   move  $t3, $t1            # t3 := t2
End:                              # suite du programme
```

Exemple d'un programme de haut niveau

Prolog

```
naive_sort(List,Sorted):-perm(List,Sorted),is_sorted(Sorted).

is_sorted([]). % la liste vide est triée
is_sorted)[_]). % toute liste à un élément est triée
is_sorted([X,Y|T]):-X=<Y,is_sorted([Y|T]).
 % si X <= Y et Y::T est triée alors X::Y::T est triée
```

Prolog : Programmation Logique

Prolog a été créé par Alain Colmerauer et Philippe Roussel vers 1972.

- Traitement des langues naturelles
- Intelligence artificielle

Prolog

La programmation logique est dite déclarative. En effet, ici on ne s'occupe pas de la manière d'obtenir le résultat ; par contre, le programmeur doit faire la description du problème à résoudre en listant les objets concernés, les propriétés et les relations qu'ils vérifient.

Prolog

- Un programme Prolog est composé de clauses.
- Les clauses sont des affirmations portant sur des atomes.
- Les atomes expriment des relations entre les termes.

Les termes I

- Les variables représentent des objets inconnus de l'univers. Syntaxiquement, une variable est une chaîne alpha-numérique commençant par une majuscule (ou par _).

Attention !

Une variable n'est pas un contenant auquel on peut affecter une valeur.

- Les termes élémentaires (ou termes atomiques, ou constantes) représentent les objets simples connus de l'univers.
  - les nombres: entiers ou flottants,
  - les identificateurs (parfois appelés symboles atomiques): un identificateur est une chaîne alpha-numérique commençant par une minuscule, les chaînes de caractères entre guillemets ("toto")

Les termes II

- Les termes composés représentent les objets composés (structurés) de l'univers de la forme:

$$foncteur(t_1, t_2, \ldots, t_n)$$

Attention !

On appelle foncteur un symbole de fonction ou de prédicat.

Exemples

```
toto.
humain(toto).
père(pierre, paul).
```

Pour s'entrainer

Vous pouvez installer swi-prolog ou alors utiliser en ligne:
`https://swish.swi-prolog.org/`

Les clauses

Une clause est une affirmation inconditionnelle (un fait) ou conditionnelle (une règle).

Les faits

Un fait est de la forme:

$$A.$$

ou A est un atome logique, et signifie que la relation définie par A est vraie (sans condition).

Exemple

$$pere(toto, paul).$$

indique que la relation "toto est le père de paul" est vraie.

Une variable dans un fait est quantifiée universellement.

Exemple

$$ancetre(adam, X).$$

Les règles

Une règle est de la forme:

$$A_0 : -A_1, \ldots, A_n.$$

ou $A_0, A_1, \ldots, A_n$ sont des atomes logiques.
Une telle règle signifie que la relation A_0 est vraie si les relations A_1 et ... et A_n sont vraies.

- A_0 est appelé tête de clause.
- A_1, ... , A_n est appelé corps de clause.
- Une variable apparaissant dans la tête d'une règle (et éventuellement dans son corps) est quantifiée universellement.
- Une variable apparaissant dans le corps d'une clause mais pas dans sa tête est quantifiée existentiellement.

Clauses de Horn

Une clause de Horn est une clause possédant au plus un littéral positif.

Faits pas de littéral négatif

$$p(x, y, f(x, z))$$

```
p(x,y,f(x,z)).
```

Régles un littéral positif et au moins un littéral négatif

$$p(x, y, f(x, z)) \vee \neg q(x) \vee \neg r(y, z)$$

$$q(x) \wedge r(y, z) \Rightarrow p(x, y, f(x, z)$$

```
p(x,y,f(x,z)) :-  q(x),r(y,z).
```

Questions : pas de littéral positif

$$\neg q(x, y) \vee \neg r(y, z)$$

Exemples de règles

```
père(X,Y) :- parent(X,Y), mâle(X).
mère(X,Y) :- parent(X,Y), femelle(X).
parent(X,Y) :- père(X,Y).
parent(X,Y) :- mère(X,Y).
mâle(X) :- père(X,_).
grandpère(X,Y) :- père(X,Z),parent(Z,Y).
```

Ordre entre les clauses

Un programme Prolog est une ensemble de paquets de clause comportant la même tête.
L'ordre des paquets n'importe pas mais l'ordre des clauses dans les paquets importe.

Aspects techniques

Pour charger un fichier on utilise:

- consult('toto.pl') ou bien ['toto.pl']
- reconsult('toto.pl')

Pour les commentaires: %

Les questions I

Pour exécuter un programme Prolog on pose une question. Une question est une suite d'atomes séparés par des virgules. Prolog répond *yes* ou *no*.

Exemple

```
mortel(X) :- humain(X).
humain(socrate).%
```

Si on demande:

```
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 5.8.0)
Copyright (c) 1990-2009 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
?- [essai].
% essai compiled 0.00 sec, 2,096 bytes
true.
?-  mortel(socrate).
true.
?- mortel(toto).
false.
```

Les questions II

La question peut comporter des variables qui sont quantifiées existentiellement.
Dans ce cas prolog donne l'ensemble des valeurs des variables.

Exemple

```
pere(toto, titi).
pere(titi, tom).
pere(titi, alex).

?- pere(toto,X), pere(X,Y).
X = titi,
Y = tom ;
X = titi,
Y = alex.
```

Algorithme de résolution

Stratégie de résolution Prolog

- Chaînage arrière : on part de ce qu'il faut prouver
- Profondeur d'abord : on essaie de prouver d'abord le premier sous but avant de s'attaquer au deuxième.
- Mécanisme de retour-arrière : en cas d'échec on revient au dernier point de choix

Chaînage arrière

Résolution d'un but $G = G_1, G_2, \ldots, G_n$.
Si n=0 le but est montré.
Sinon s'il existe une clause ayant une conclusion qui s'unifie avec G_1 on choisit la première.
Si c'est un fait retirer G_1 de la liste.
Si c'est une règle de la forme $B : -H_1, H_2, \ldots, H_n$, remplacer G_1 par $H_1, H_2, \ldots, H_n$ et se rappeler récursivement.
S'il n'existe aucune clause ayant une conclusion qui s'unifie avec G_1 revenir au dernier point de choix et choisir la clause suivante.

trace I

Exemple

```
p(X) :- q(X),r(X).
q(a).
q(b).
r(b).
r(c).
```

```
 ?- trace.
true.

[trace]  ?- p(X).
   Call: (6) p(_G368) ? creep
   Call: (7) q(_G368) ? creep
   Exit: (7) q(a) ? creep
   Call: (7) r(a) ? creep
```

trace II

```
   Fail: (7) r(a) ? creep
   Redo: (7) q(_G368) ? creep
   Exit: (7) q(b) ? creep
   Call: (7) r(b) ? creep
   Exit: (7) r(b) ? creep
   Exit: (6) p(b) ? creep
X = b.
```

Contrôle du succés

- true
- fail

Test de type

- var
- atomic

Exemple

```
?- var(X).
true.
?- var(t).
false.
?- atomic(X).
false.
?- atomic(2).
true.
?- atomic(toto).
true.
?- atomic(toto(3)).
false.
```

Opération arithmétiques

+	addition
-	soustraction
*	multiplication
/	division
//	division euclidienne
mod	modulo
**	puissance

Egalités

`X = Y`, `X \= Y` Teste si X et Y peuvent être unifiés ou pas. Par exemple,

```
?- [X,Y|R] = [a,b,c].
   X = a, Y = b, R = [c]
   ?- [X,Y,Z] = [a,b].
   no
```

Egalités

`X == Y`, `X \== Y` Teste si X et Y sont identiques ou pas en mémoire. Par exemple,

```
?- X = 3, Y = 1 * 3, X == Y.
no
?- X = a, [Y|_]= [a,b,c], X == Y.
X = a, Y = a
```

Egalités

`X =:= Y`, `X =\= Y` Teste si X et Y sont égaux après évaluation. Par exemple,

```
?- 3+2 =:= 1+1+3.
true.
```

variable is expression

Exemple

?- X is 3*4. X = 12 yes

Exemple

```
?- 8 = 5+3.
No
?- 8 is 5+3.
Yes
?- X is 5+3.
X = 8 ;
No
```

Exemple

```
entier(z).
entier(s(X)) :- entier(X).
```

Exemple

```
plus(z,X,X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
```

Exemple

```
fact(0,1).
fact(N,F) :- N > 0, M is N - 1,
        fact(M,Fm), F is N * Fm.
```

Factorielle avec accumulateur

```
fact(A,B) :- factaux(A,1,B).

factaux(1,A,A).
factaux(A,B,C) :- A>1, D is B*A, E is A-1, factaux(E, D, C).
```

Exemple Fibonacci

```
fib(0,1).
fib(1,1).
fib(N,F) :- N > 1, N1 is N - 1, N2 is N - 2,
    fib(N1,F1), fib(N2,F2), F is F1 + F2.
```

Exemple Fibonacci avec accumulateur

```
fib(0,1).
fib(1,1).
fib(N,F) :- N > 1, fib(N,1,1,F)

fib(2,F1,F2,F) :- F is F1 + F2.
fib(N,F1,F2,F) :- N > 2, N1 is N - 1, NF1 is F1 + F2,
    fib(N1,NF1,F1,F).
```

Exemple: minimum

```
minimum(M,N,M) :- M =< N.
minimum(M,N,N) :- N =< M.
```

Exemple: modulo

```
mod(X,Y,Z) :- less_than(Z,Y), times(Y,Q,W), plus(W,Z,X).
```

Exemple: Les tours de Hanoï

```
move(1,X,Y,_) :-
    write('Move top disk from '),
    write(X),
    write(' to '),
    write(Y),
    nl.
move(N,X,Y,Z) :-
    N>1,
    M is N-1,
    move(M,X,Z,Y),
    move(1,X,Y,_),
    move(M,Z,Y,X).
```

Les listes

Les listes sont définies récursivement:

1. l'atome [] est une liste vide ;
2. si T est une liste et H est un élément, alors le terme '.'(H, T) est une liste. Le foncteur . est aussi noté `[H|T]`.

	Prolog	OCaml	Haskell
Cons	.	::	:
Liste vide	[]	[]	[]
Exemple	`[1|[2|[3|[]]]]`	`1::2::3::[]`	`1:2:3:[]`
Exemple avec sucre syntaxique	`[1,2,3]`	`[1;2;3]`	`[1,2,3]`

Fonctions vs prédicats

Soit f une fonction d'arité n, alors on peut représenter f par un prédicat d'arité $n+1$:

$$P_f(x_1, x_2, \ldots, x_n, y) := \; y = f(x_1, x_2, \ldots, x_n)$$

Inversement, si on a un prédicat (ou relation) d'arité $n+1$, on peut le voir comme une fonction d'arité n à condition que prédicat ce soit fonctionnel, i.e.

$$\forall x_1 \ldots x_n y \; P(x_1, \ldots, x_n, y) \wedge P(x_1, \ldots, x_n, y') \Rightarrow y = y'$$

Exemple:

$5 = 2 + 3$ peut être représenté par : $estlasomme(2, 3, 5)$.

Composée de fonctions et prédicats

Ok mais comment une composée de fonctions ? Par exemple
$(2+3) * y = mult(somme(2,3), y)$

Exemple: concaténation de deux listes (rappel de prog fonc)

On veut une fonction concat qui met bout à bout deux listes:
`concat [1,2,3] [4,5] = [1,2,3,4,5]`

Exemple: concaténation de deux listes (rappel de prog fonc)

On veut une fonction concat qui met bout à bout deux listes:
`concat [1,2,3] [4,5] = [1,2,3,4,5]`

En Haskell

```
concat :: [a] -> [a] -> [a]
concat []     ys = ys
concat (x:xs) ys = x : (concat xs ys)
```

Exemple: concaténation en Prolog

Exemple

```
aconcat([],Y,Y).
aconcat([X|R],Y,[X|Q]) :- aconcat(R,Y,Q).
```

Utilisation

```
?- aconcat([1,2,3],[4,5,6],L).
L = [1, 2, 3, 4, 5, 6].
```

Utilisation 2

```
?- aconcat([1,2,3],L,[1,2,3,5,4,5,6]).
L = [5, 4, 5, 6].
```

Utilisation 3

```
?- aconcat(L,M,[1,2,3]).
L = [],
M = [1, 2, 3] ;
L = [1],
M = [2, 3] ;
L = [1, 2],
M = [3] ;
L = [1, 2, 3],
M = [] ;
false.
```

Notion de Mode

- + : l'argument doit être instancié
- - : l'argument doit être une variable
- ? : + ou -

```
%!      length(+List:list, -Length:int) is det.
%!      length(?List:list, -Length:int) is nondet.
%!      length(?List:list, +Length:int) is det.
%
%       True if List is a list of length Length.
```

Exemple

```
membre( X, [X | L]).
membre( X, [Y | L]) :- membre( X, L).

?- membre(d,[a,b,a]).
false.

?- membre(a,[b,a,b]).
true .
```

Exemple

```
enlever(X,[X|R],R).
enlever(X,[F|R],[F|S]) :- enlever(X,R,S).
```

Exemple

```
Exemples d'utilisation de enlever:
?- enlever( a, [a, c, b, a, b], L).
L = [c, b, a, b] ;
L = [a, c, b, b] ;
false
?- enlever( X, [a, b, b], L).
X=a L = [b, b] ;
X=b L = [a, b] ;
X=b L = [a, b] ;
false
```

Exemple

```
prefix([],List).
prefix([X|Prefix],[X|List]) :- prefix(Prefix,List).
```

```
?- prefix([a,b],[a,b,c]).
true.

?- prefix([a,d],[a,b,c]).
false.

?- prefix(X,[a,b,c]).
X = [] ;
X = [a] ;
X = [a, b] ;
X = [a, b, c] ;
false.
```

Chaînes de caractères

Une chaîne de caractère est représentée pas une liste de codes ASCII.

```
 ?- X="titi".
X = [116, 105, 116, 105].
```

Coupures

Les coupures permettent d'interdire les retours arrière sur des points de choix, c'est un mécanisme de contrôle extra logique.
On distingue les:

coupures vertes qui éliminent des points de choix inutiles (optimisation) et

coupures rouges qui changent la sémantique du programme.

Exemple de cut

```
a(X, Y) :- b(X), !, c(Y).
b(1).
b(2).
b(3).

c(1).
c(2).
c(3).
```

Exemple de cut

```
?-  a(X,Y).
X = 1,
Y = 1 ;
X = 1,
Y = 2 ;
X = 1,
Y = 3.
```

Exemple de cut

```
a(X) :- b(X), !, c(X).
b(1).
b(2).
b(3).

c(2).
```

Exemple de cut

```
?- a(X).
false
?- a(2).
true
```

Exemple de cut: cas avec deux clauses

```
a(X) :- b(X), !, c(X).
a(X) :- d(X).

b(1).
b(4).

c(3).

d(4).
```

Exemple de cut

```
?- a(X).
false
?- a(4).
false
```

Exemple

```
max(X,Y,X) :- X >= Y.
max(X,Y,Y) :- X < Y.
```

Avec cut:

```
max(X,Y,X) :- X >= Y , !.
max(X,Y,Y) :- X < Y.
```

Avec cut 2:

```
max(X,Y,X) :- X >= Y , !.
max(X,Y,Y) :- .
```

Not

```
not(P) :- call(P), !, fail.
not(P).
```

Contrôle de la base de connaissances

- asserta(clause)
- assertz(clause) ou assert(clause)
- retract(clause)

Entrées-Sorties

- write
- nl
- put(code ascii)
- get0(X)

Exemple

```
?- write(5).
5
true.
?- write('Toto'), nl, write ('titi').
Toto
titi
true.
?- write(5),nl,write(6).
5
6
true.
?- get0(X).
|: o

X = 111.

?- put(111).
```

Afficher liste

```
affiche_liste([H|T]):-write(H),write(' '),affiche_liste(T).
affiche_liste([]) :- nl.

?- affiche_liste([toto,titi,tutu]).
toto titi tutu
true.
```