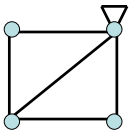
Bases mathématiques pour les méthodes formelles



IUT Bordeaux 1
département Informatique
Eric Sopena
Colette Johnen

Plan du cours

Chapitre 1

La Logique
Propositionnelle
(ou calcul propositionnel)

Plan du chapitre 1

- Syntaxe
- Interprétation
- Démonstrations Logiques

Syntaxe (1)

Nous noterons **P** = {**p, q, p', q', ...**} l'ensemble des propositions atomiques. La nature de ces propositions dépend de l'application visée...

Nous noterons **F(P)** l'ensemble des formules de la logique propositionnelle déduite de **P** selon les règles suivantes (indiquées sur le transparent suivant)

Syntaxe (2)

Définition (formule propositionnelle).
Une formule propositionnelle est une suite de symboles pris dans l'ensemble **P** ∪ { ⇒, ¬, (,) } et construite selon les règles suivantes :

1. tout symbole de **P** est une formule,
2. si **f** est une formule alors ¬**f** est une formule (négation)
3. si **f** et **f'** sont deux formules alors (**f** ⇒ **f'**) est une formule (implication)
4. toute formule obtenue par application des règles précédentes un nombre fini de fois

Exemple

Exemples de formules sur **P = {p, q}** :

- **p**
- **(p ⇒ ¬q)**
- **((¬p ⇒ q) ⇒ (p ⇒ q))**
- **¬(p ⇒ q)**

Les formule suivantes sont-elles syntaxiquement correctes (valides) sur **P** ?

p ⇒ (¬p ⇒ ¬(q ⇒ p'))
p ⇒ (¬p ⇒ ¬(q ⇒ p))
(p ⇒ (¬p ⇒ ¬(q ⇒ p)))

7

Interprétation (sémantique) (1)

Soit **I : P → B** (algèbre de Boole) une fonction d'interprétation, associant à chaque proposition atomique de **P** une valeur booléenne (**vrai** ou **faux**)

L'interprétation d'une formule est totalement déterminée par l'interprétation des propositions atomiques.

8

Interprétation (sémantique) (1)

Définition de la valeur (de vérité) d'une formule **f**

La valeur de vérité **I(f)** d'une formule **f** de **F(P** pour l'interprétation **I** est définie par :

1. si **f = p ∈ P**, alors **I(f) = I(p)**

2. si **f = ¬f'** alors **I(f) = $\overline{\text{I(f')}}$**

3. si **f = (f_1 ⇒ f_2)** alors
   I(f) = $\overline{\text{I}(f_1)}$ + I(f_2)

9

Interprétation (sémantique) - exemples

Soit **P = {p, q}.**
Soit **I** l'interprétation définie par
I(p) = vrai et **I(q) = faux**
Soit **f = (p ⇒ (q ⇒ ¬p))**

I(f) = I(p ⇒ (q ⇒ ¬p))	définition de **f**
= $\overline{\text{I(p)}}$ + I(q ⇒ ¬p)	implication
= $\overline{\text{I(p)}}$ + $\overline{\text{I(q)}}$ + I(¬p)	implication
= $\overline{\text{I(p)}}$ + $\overline{\text{I(q)}}$ + $\overline{\text{I(p)}}$	négation
= $\overline{\text{vrai}}$ + $\overline{\text{faux}}$ + $\overline{\text{vrai}}$	prop. atomiques
= faux + vrai + faux	Algèbre de Boole
= vrai	Algèbre de Boole

10

Interprétation (sémantique) (3)

Définition (validité, satisfiabilité). Soit **f** une formule ; on dit que

1. **f** est valide (ou encore, **f** est une tautologie) ssi **I(f) = vrai** pour toute interprétation **I**
2. **f** est satisfaisable ssi il existe une interprétation **I** pour laquelle **I(f) = vrai**
3. **f** est insatisfaisable ssi il existe une interprétation **I** pour laquelle **I(f) = faux**

11

Interprétation (sémantique) (3)

Exemples:

1. la formule **(p ⇒ p)** est valide
2. la formule **(p ⇒ ¬q)** est satisfaisable mais non valide
3. la formule **¬(p ⇒ p)** est insatisfaisable

12

Démonstrations logiques (1)

Un séquent est un couple de la forme (**F**,**f**), où **f** est une formule (**f** ∈ **F**) et F un ensemble fini de formules (**F** ⊂ **F**). L'ensemble **F** est l'ensemble des prémisses du séquent, la formule **f** sa conclusion.

Un séquent **(F,f)** est vrai dans une interprétation **I** ssi on a :

si pour toute formule **g** ∈ **F**, **I(g)** est égale à **vrai** alors **I(f)** est égale à **vrai** sinon **I(f)** peut être égale à **vrai** ou **faux**

Démonstrations logiques (1)

Un séquent est un couple de la forme (**F**,**f**), où **f** est une formule (**f** ∈ **F**) et F un ensemble fini de formules (**F** ⊂ **F**). L'ensemble **F** est l'ensemble des prémisses du séquent, la formule **f** sa conclusion.

Un séquent **(F,f)** est valide si il est vrai dans toute interprétation (on écrit alors F ├ **f**)

Démonstrations logiques (2)

Les définitions des séquents, des séquents vrais pour une interprétation **I** et des séquents valides permettent de formaliser la notion de conséquence sémantique.

Si toutes les prémisses du séquent sont vraies, alors la conclusion est vraie.

Formalisation du concept de Preuve …

Démonstrations logiques - Exemples

Théorème : Le séquent ({p, (p ⇒ q)}, q) est valide

Preuve :

1. Soit **I1** une interprétation telle que **I1(p)= vrai**, et **I1(p ⇒ q)= vrai**. Nous avons :

I1(p ⇒ q) = $\overline{\text{I1(p)}}$ + I1(q) interprétation
vrai = faux + I1(q)
hypothèse, définition du « ¬ » et de « + »
vrai = I1(q)

2. Soit **I2** une interprétation telle que **I2(p)= faux** ou **I2(p ⇒ q)= faux** alors le séquent est vrai dans **I2**. □

Démonstrations logiques - Exemples

seq1 est le séquent suivant **({p}, q)**
seq2 est le séquent suivant **({(p ⇒ q)}, q)**

p	q	seq1	(p ⇒ q)	seq2
0	0			
0	1			
1	0			
1	1			

Séquent prouvable (1)

Un séquent (**F**,**f**) est prouvable, ce que l'on notera **F** ├ **f**, s'il peut être construit en utilisant un nombre fini de fois les 6 règles suivantes :

(a) si **f** ∈ **F**, alors **F** ├ **f**
utilisation d'une hypothèse

(b) si **g** ∉ **F** et **F** ├ **f**, alors **F**,**g** ├ **f**
augmentation des hypothèses

(c) si **F** ├ (**f** ⇒ **f'**) et **F** ├ **f**, alors **F** ├ **f'**
détachement (modus ponens)

Séquent prouvable (2)

(d) si **F, f ⊢ f', alors F ⊢ (f ⇒ f')**

retrait d'une hypothèse (synthèse)

(e) **F ⊢ f** ssi **F ⊢ ¬¬f**

double négation

(f) si **F,f ⊢ f'** et **F,f ⊢ ¬f'** alors **F ⊢ ¬f**

raisonnement par l'absurde (contradiction)

La définition des séquents prouvables permet de formaliser la notion de conséquence logique (purement syntaxique, indépendante de toute interprétation)

Démonstrations logiques (3)

Une démonstration d'un séquent prouvable **F ⊢ f** est une suite finie de séquents prouvables $\mathbf{F}_i \vdash \mathbf{f}_i$, i = 1,...,n, telle que :

- $\mathbf{F}_n = \mathbf{F}$ et $\mathbf{f}_n = \mathbf{f}$
- chaque séquent de la suite est obtenu à partir des séquents qui le précède en appliquant l'une des 6 règles

Remarque : le premier séquent de la suite est nécessairement obtenu par utilisation d'une hypothèse

Démonstrations logiques - Exemple 1

1. p, q ⊢ p	hypothèse
2. p ⊢ (q ⇒ p)	synthèse de 1
3. ∅ ⊢ (p ⇒ (q ⇒ p))	synthèse de 2

Démonstrations logiques - Exemple 2

1. (p ⇒ q), ¬q, p ⊢ ¬q	hypothèse
2. (p ⇒ q), ¬q, p ⊢ p	hypothèse
3. (p ⇒ q), ¬q, p ⊢ (p ⇒ q)	hypothèse
4. (p ⇒ q), ¬q, p ⊢ q	modus ponens sur 2 et 3
5. (p ⇒ q), ¬q ⊢ ¬p	contradiction 1 et 4
6. (p ⇒ q) ⊢ (¬q ⇒ ¬p)	synthèse de 5

Syntaxe vs sémantique...

La notion de séquent valide est sémantique (référence aux interprétations) alors que la notion de séquent prouvable est purement syntaxique

Théorème : Un séquent est prouvable si et seulement s'il est valide.

Nous avons donc à la fois la cohérence (un séquent prouvable est valide) et la complétude (tous les valides sont prouvables).

Chapitre 2

La logique des prédicats du premier ordre

Plan du chapitre 2

- Syntaxe
- Interprétation
- Formules équivalentes
- Démonstration

25

Syntaxe : briques de base... (1)

1. ensemble **G** de symboles de fonctions
2. ensemble **X** de symboles de variables :

$$X = \{ x, y, \ldots \}$$

A chaque fonction **g** de **G** est associée une arité (ou rang) notée **rang**(g) correspondant au nombre de ces paramètres

On note **C** = { **a, b, ..., a', b', ...** } l'ensemble des symboles de constantes :

$$C = \{ g \in G \mid rang(g) = 0 \}$$

26

Syntaxe : briques de base. Exemple G_N

Soit **G_N** = { **zéro, succ, plus** }.

zéro est d'arité 0 (c'est une constante)
succ est d'arité 1
plus est d'arité 2

27

Syntaxe : ensemble de termes (2)

Nous noterons $T(G \cup X)$ l'ensemble ldes termes de la logique des prédicats du premier ordre déduite de $G \cup X$ selon les règles suivantes (indiquées sur le transparent suivant)

28

Syntaxe : un Terme (2)

Un terme sur $G \cup X$ est défini de la façon suivante :

1. si $t \in C \cup X$, alors **t** est un terme
2. si $f \in G$ avec $rang(f) = n > 0$, et si $t_1, t_2, \ldots, t_n$ sont n termes alors $f(t_1, t_2, \ldots, t_n)$ est un terme

Un terme clos est un terme sans variables

29

Syntaxe : Terme - Exemple G_N

Soit **G_N = { zéro, succ, plus }**
rang(zéro) = 0 rang(succ) = 1
rang(plus) = 2
Soit **X_N = {x, y}**

Quelques termes :

zéro	clos
succ(x)	non clos
succ(zéro)	clos
Succ(plus(succ(y),succ(succ(zéro))))	non clos

30

Syntaxe : les prédicats (1)

Un prédicat est une affirmation, portant sur des objets, qui peut être vraie ou fausse...

Nous utiliserons un ensemble **P** de symboles de Prédicats. A chaque symbole de prédicat **p**, est associée une arité notée **rang(p)**

Un symbole de relation d'arité 0 est un symbole propositionnel

31

Syntaxe : les prédicats - Exemple G_N

Soit **P_N** = { **pair, premier, inf** } avec

- **rang(pair) = 1,**
- **rang(premier) = 1**
- **rang(inf) = 2**

32

Syntaxe : les formules, enfin ! (1)

Nous noterons **F(P ∪ G ∪ X)** l'ensemble des formules de la logique des prédicats du premier ordre déduite de **P ∪ G ∪ X** selon les règles suivantes (indiquées sur le transparent suivant)

On notera **F(P ∪ G ∪ X)** l'ensemble des formules ainsi construites.

On utilise les symboles suivants :

{ ⇒, ¬, ∧, ∨, ∀, ∃ ,(,) }

33

Syntaxe : les formules, enfin ! (2)

Une formule (de la logique des prédicats du 1er ordre) sur **P ∪ G ∪ X** est définie ainsi :

(1) si **p** ∈ **P** avec **rang(p) = n**, et si $t_1, t_2, \ldots, t_n$ sont des termes, alors $p(t_1, t_2, \ldots, t_n)$ est une formule (atomique)

(2) si **f** et **f'** sont deux formules, alors

$\neg f$, $(f \Rightarrow f')$, $(f \wedge f')$ et $(f \vee f')$

sont des formules

(3) si **f** est une formule contenant la variable **x** $(x \in X)$ alors $\forall x\ f$ et $\exists x\ f$ sont des formules

34

Syntaxe : les formules - Exemple G_N

G_N = { zéro, succ, plus }
X_N = { x, y, z }
P_N = { pair, premier, inf }

Nous avons alors les formules suivantes :

pair (plus (x, x)) atomique
inf (zéro, succ (y)) atomique
∀x pair (plus (x,x))
∃y premier (y)
∃x (premier(x) ∧ inf(x, succ(succ(zéro))))

35

Syntaxe : les formules - Exemple G_N

Soit **G_N = { zéro, succ, plus }**
Soit **X_N = {x, y, z}**
Soit **P_N = { pair, premier, inf }**

Exprimez les formules suivantes :

- tout entier ajouté à son successeur est pair
- le double d'un entier > 1 n'est jamais premier
- la somme d'un entier x et d'un entier non nul est toujours supérieure à x
- aucun entier sauf 2 n'est pair et premier

36

Exemple **G_N**

- tout entier ajouté à son successeur est pair
- le double d'un entier > 1 n'est jamais premier
- la somme d'un entier x et d'un entier non nul est toujours supérieure à x
- aucun entier sauf 2 n'est pair et premier

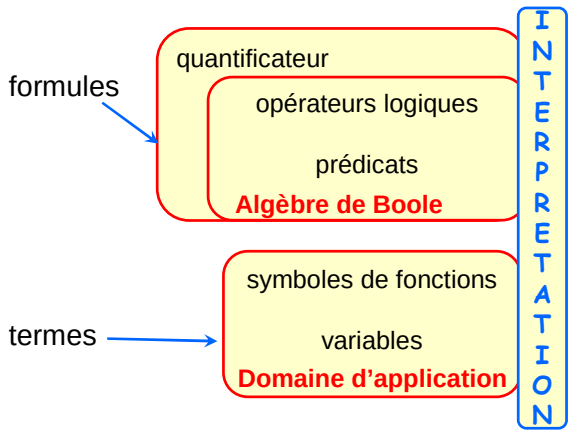
37

Logique propositionnelle versus logique des prédicats

Remarque: La logique propositionnelle est « contenue » dans la logique des prédicats du 1er ordre (on se restreint aux symboles propositionnels et aux symboles $\Rightarrow$ et $\neg$).

38

Interprétation (sémantique) (1)



39

Interprétation (sémantique) (2)

Soit **L = P ∪ G** (Prédicats, Fonctions);
une **L**-structure est
S =<L, E, Pred_S, Fonct_S> :

- **E** représente les valeurs possibles des objets
- **Pred_S** associe à chaque prédicat **p** une interprétation (fonction de $\mathbf{E}^{\mathbf{rang(p)}}$ à B **)**

  B = {vrai, faux}
- **Fonct_S** associe à chaque fonction **g** une interprétation (fonction de $\mathbf{E}^{\mathbf{rang(g)}}$ à **E)**

40

Interprétation (sémantique) (3)

Soit **L = P ∪ G ,** une L-structure est
S = < L, E, Pred_S, Fonct_S > où :

- **E** est un ensemble non vide (domaine d'interprétation)
- **Pred_S** une application, associant à chaque symbole **p ∈ P** une fonction $\mathbf{Pred\_S(g)=g_S: E^{rang(p)} \rightarrow B}$
- **Fonct_S** une application associant à chaque symbole **g ∈ G** une fonction $\mathbf{Fonct\_S(g)=g_S: E^{rang(g)} \rightarrow E}$
  (à chaque constante **c ∈ C**, **Fonct_S** associe un élément de **E** noté $\mathbf{c_S}$)

41

Interprétation (sémantique) - Exemple **G_N**

L-structure **S_N** est définie par :

- L = **P_N ∪ G_N**
- **E = N** (entiers naturels)
- **Pred_S** est défini par (**x, y ∈ N**) :
  $\mathbf{premier_S(x)}$**=vrai si x est premier**
  $\mathbf{pair_S(x)}$ **= vrai si x est pair**
  $\mathbf{inf_S(x,y)}$ **= vrai si x < y**
- **Fonct_S** est défini par
  $\mathbf{zéro_S}$ = 0 ∈ **N** = **E**
  $\mathbf{succ_S}$: **i → i + 1** ($\mathbf{N^1 \rightarrow N}$)
  $\mathbf{plus_S}$: **(i,j) → i+j** ($\mathbf{N^2 \rightarrow N}$)

42

Interprétation (sémantique) (4)

Il nous reste maintenant à donner à chaque variable une valeur du domaine **E**

Une valuation **v** sur une **L**-structure **S = <L, E, Pred_S, Fonct_S>** est une application de l'ensemble des variables **X** dans le domaine **E**.

43

Interprétation (sémantique) (5)

Nous pouvons maintenant valuer les termes :

Si **t** est un terme et **v** une valuation sur une **L**-structure **S=<L, E, Pred_S, Fonct_S>**, la valeur $v^*(t) \in E$ du terme **t** est défini par :

- si $t = a \in C$, $v^*(t) = a_S$
- si $t = x \in X$, $v^*(t) = v(x)$
- si $t = f(t_1, t_2, \ldots, t_n)$, alors $v^*(t) = f_S(v^*(t_1), v^*(t_2), \ldots, v^*(t_n))$

44

Interprétation (sémantique) - Exemple G_N

Soit **S_N** la **L**-structure définie précédemment sur $G\_N \cup P\_N$. La valuation **v** sur **S_N** est défini par **v(x) = 3** et **v(y) = 4**

$$
\begin{aligned}
& v^*(\text{plus}(\text{succ}(\text{zéro}), x)) \\
&= \text{plus}_S(v^*(\text{succ}(\text{zéro})), v^*(x)) \\
&= v^*(\text{succ}(\text{zéro})) + v^*(x) \\
&= \text{succ}_S(v^*(\text{zéro})) + x_S \\
&= (v^*(\text{zéro}) + 1) + 3 \\
&= (\text{zéro}_S + 1) + 3 \\
&= (0 + 1) + 3 \\
&= 4
\end{aligned}
$$

45

Interprétation (sémantique) (6)

Deux valuations **v** et **v'** sur une même **L**-structure sont congruentes sur $Y \subseteq X$, noté $v =_Y v'$, ssi $v(y) = v'(y)$ pour tout **y** de **Y**.

Nous pouvons maintenant donner une valeur de vérité aux formules de **L** pour une valuation v sur une **L**-structure.

46

Interprétation (sémantique) (7)

Soit **v** une valuation sur une **L**-structure **S=<L,…>**,
Soit **f** une formule de **L**. La valeur de vérité de la formule **f** pour **v** noté $I_v(f) \in B$, est défini par

1 si $f = p(t_1, \ldots, t_n)$ alors
$I_v(f) = p_S(v^*(t_1), \ldots, v^*(t_n))$

2 si $f = \neg f'$ alors $I_v(f) = \overline{I_v(f')}$

3 si $f = (f' \vee f'')$ alors
$I_v(f) = I_v(f') + I_v(f'')$

4. si $f = (f' \wedge f'')$ alors
$I_v(f) = I_v(f') \cdot I_v(f'')$

5. si $f = (f' \Rightarrow f'')$ alors
$I_v(f) = \overline{I_v(f')} + I_v(f'')$

47

Interprétation (sémantique suite) (8)

6 si $f = \forall x\ f'$ alors $I_v(f) = vrai$ ssi pour toute valuation **v'** telle que
$v' =_{X-\{x\}} v$, nous avons $I_{v'}(f') = vrai$

7 si f = ∃x f' alors $I_v(f) = vrai$ ssi il existe une valuation **v'** telle que
$v' =_{X-\{x\}} v$ et $I_{v'}(f') = vrai$

48

Interprétation (sémantique) - Exemple G_N

Soit **g = pair(succ(succ(zéro)))**
Nous avons :

Iv(g)= Iv(pair(succ(succ(zéro))))
= vrai ssi
$pair_S$ (v*(succ(succ(zéro)))) = vrai

v*(succ(succ(zéro)))) = $succ_S$(v*(succ zéro)))
= $succ_S$($succ_S$(v*(zéro)))
= $succ_S$($succ_S$(0))
= $succ_S$(1)
= 2

Ainsi, **Iv(g) = vrai** car **$pair_S$(2) = vrai**

Interprétation - Exemple G_N

Soit **X = {x, y, z}.** La valuation **v** sur **S_N** est défini par **v(x) = 3** et **v(y) = 4** et **v(z) = 8**

Soit **g = ∀z inf(plus(z,x), plus(z,y))**

Nous avons :
Iv(g)= Iv(∀z inf(plus(z,x), plus(z,y))) = vrai
ssi pour toute **v'** t.q. **v' =$_{\{x,y\}}$ v** on a
Iv'(inf(plus(z,x),plus(z,y))) = vrai
inf_s($plus_s$(z,x),$plus_s$(y,x)) = vrai
$plus_s$(z,x) < $plus_s$(z,y)
(v'(z) + v(x)) < (v'(z) + v(y))
(v'(z) + 3) < (v'(z) + 4)

d'où **Iv(g) = vrai**

Interprétation (sémantique) - Exemple G_N

Soit **X = {x, y, z},** La valuation **v** sur **S_N** est défini par **v(x)=1, v(y)=2 et v(z)=3.**
Soit **g = ∃y (premier(plus(y,x))).** Nous avons:

Iv(g) = Iv(∃y premier(plus(y,x))) =
vrai ssi il existe **v'** t.q. **v =$_{\{x,z\}}$ v** et
Iv'(premier(plus(y,x))) = vrai
$premier_s$($plus_s$(y,x)) = vrai
$premier_s$(Iv'(y) + Iv(x)) = vrai
Iv'(y)+1 doit être un nombre premier

Iv(g) = vrai car **v'** convient avec :
v'(x) =1, v'(y) = 4 et v(z) = 3

Formules équivalentes (1)

Deux formules **f** et **f'** sont équivalentes, notée **f ≡ f',** si pour toute **L**-structure **S** et pour toute valuation **v**, **Iv(f) = Iv(f').**

Les formules suivantes sont équivalentes :

¬(f ∧ f') ≡ (¬f ∨ ¬f')
si **f = (f'∧ f")** alors **Iv(f) = Iv(f').Iv(f")**

∀x f ≡ ¬∃x ¬f

∃x f ≡ ¬∀x ¬f

Les quantificateurs : ∀ ∃

∀x : quelque soit x tel que …

∀x (pair(x) ∨ ¬premier(x))
∀x ¬premier(x)
∀x (premier(x) ∧ pair(x))

Négation :
∀x ¬premier (x) ≡ ¬( )
∀x (premier(x) ∧ pair(x))
≡ ¬( )
∀x ¬(pair(x) ∨ ¬ premier(x))≡
¬ ( )

Les quantificateurs : ∀ ∃

∃y : il existe y tel que …

∃x ¬premier(x)
∃x (premier(x) ∧ pair(x))
∃x (premier(x) ∨ ¬pair(x))

Négation :
∃x ¬premier (x) ≡ ¬ ( )
∃x (premier(x) ∧ pair(x))
≡¬( )
∃x (premier(x) ∨ ¬pair(x))
≡ ¬( )

Les quantificateurs : ∀ ∃

Traduire en Français les formules suivantes sur **S_N**

1. ∀x ∃y `inf(x, y)`

2. ∃x ∀y `inf(x, y)`

Les formules sont-elles vraies ?
Ecrivez leur négation

55

Formules équivalentes (2)

A démontrer : Pour toutes **L**-structure **S** et valuation **v** :

`∀x (p1(x) ∧ p2(x))≡(∀x p1(x) ∧ ∀x p2(x))`

`∃x (p1(x) ∨ p2(x)) ≡ (∃x p1(x) ∨ ∃x p2(x))`

`si Iv(∃x (p1(x)∧ p2(x))) = vrai`
`alors Iv((∃x p1(x)∧ ∃x p2(x))) = vrai`

`si Iv((∀x p1(x) ∨ ∀x p2(x))) = vrai`
`alors Iv(∀x (p1(x) ∨ p2(x))) = vrai`

56

Formules équivalentes (3)

La réciproque des assertions sont fausses

`si Iv(∃x (p1(x)∧ p2(x))) = vrai`
`alors Iv((∃x p1(x)∧ ∃x p2(x))) = vrai`

`si Iv((∀x p1(x) ∨ ∀x p2(x))) = vrai`
`alors Iv(∀x (p1(x) ∨ p2(x))) = vrai`

Contre-exemple:
`(∃x premier(x) ∧ ∃x inf(x, succ(zéro)) )`
`∀x ( inf(x, succ(zéro)) ∨ inf(zéro, x) )`

57

Formules équivalentes (3)

A démontrer :
Soit * ∈ {∧,∨}, **f1** et **f2** deux formules, **x** une variable n'ayant aucune occurrence dans **f2**. Nous avons alors :

- `(∀x f1 * f2) ≡ ∀x (f1 * f2)`
- `(∃x f1 * f2) ≡ ∃x (f1 * f2)`
- `(f2 * ∀x f1) ≡ ∀x (f2 * f1)`
- `(f2 * ∃x f1) ≡ ∃x (f2 * f1)`
- `(∀x f1 ⇒ f2) ≡ ∀x (f1 ⇒ f2)`
- `(∃x f1 ⇒ f2) ▪ ∃x (f1 ⇒ f2)`

58

Satisfaisabilité, validité

Nous dirons qu'une formule **f** est :

- satisfaisable dans **S** (**S** est une **L**-structure) s'il existe une valuation **v** telle que **`v(f)=vrai`**
- satisfaisable s'il existe une **L**-structure **S** telle que **f** est satisfaisable dans **S**
- valide dans **S** si pour toute valuation **v**, **`v(F) = vrai`**
- universellement valide si elle est valide dans toute **L**-structure

59

Satisfaisabilité, validité – Exemple **S_N**

`g1 = ∀x inf ( x, plus(x,y) )` est satisfaisable mais non valide sur **N** (y = 0).

`g1= ∀x ∀y inf(x,plus(x,plus(y, succ(z))))`
est valide sur **N**
satisfaisable mais non valide sur **Z** (y+z+1 < 0)

`g2 = (∀x R(x) ⇒ R(y))` est universellement valide.

60

Séquents valides

La notion de séquent se définit comme dans le cas de la logique propositionnelle. Nous avons alors :

Un séquent (**F**,**f**) est valide dans **S**, noté $\mathbf{F} \models_{\mathbf{S}} \mathbf{f}$, si pour toute valuation **v** nous avons :

si pour toute $\mathbf{g} \in \mathbf{F}$, $\mathbf{Iv(g)=vrai}$,
$\mathbf{alors\ Iv(f) = vrai}$

Un séquent est universellement valide s'il est valide dans toute **L**-structure.

Séquents valides - Exemple **G_N**

- Le séquent
  $\mathbf{(\emptyset, \forall x\ \forall y\ inf(x,\ plus(x,\ plus(y,succ(z)))))}$
  est valide dans **S_N**

- Le séquent
  $\mathbf{(\{\forall x\ (R(x) \Rightarrow R'(x)), \exists y\ R(y)\},\ \exists z\ R'(z))}$
  est universellement valide

Chapitre 3

La logique de Hoare

Plan du chapitre 3

- Syntaxe
- Système déductifs
- Correction de programme

Introduction (1)

Le but de la logique de Hoare est de formaliser la preuve de la correction des programmes.

Rappelons que notre souhait est de prouver des choses du type :

<pré-condition>
programme
<post-condition>

Signification : si la pré-condition est vraie alors, après exécution du programme, la post-condition est vraie.

Introduction (2)

Pour cela, nous allons utiliser des règles de déduction qui, étape par étape (i.e. instruction par instruction), nous amènerons de la pré-condition à la post-condition.

Les formules (1)

Les formules ont la forme suivante :

{p} S {q}

où **p** et **q** sont deux prédicats (exprimés dans la logique des prédicats du premier ordre, avec **X = {variables du programme}**) et **S** un « extrait de programme »...

Signification intuitive. Si **p** est vrai alors, après exécution de l'extrait de programme **S**, **q** est vraie.

Les formules - Exemples

Des formules vraies de la logique de Hoare :

1. **{x ≥ 0} x ← x + 1 { x > 0 }**
2. **{x > 0} x ← 5 { x = 5 }**
3. **{(x > 0) ∧ (y ≥ 0)} z ← y/x**
   {(x > 0) ∧ (y ≥ 0) ∧ (z ≥ 0)}
4. **{(x ≠ 0) }**
   si (x=0) alors y ← x+1
   sinon [y ← x; x ← x-1]
   {(y ≠ 0)}

Les formules (2)

Remarque 1 : Pour des raisons de lisibilité, on s'autorise l'écriture de « x > 0 », plutôt que « sup(x,0) ».

Remarque 2 : On ne cherche pas à exprimer « tout ce qui est vrai », mais seulement ce qui nous intéresse...

Restriction : On ne considèrera que des extraits de programme ne permettant pas la synonymie de variables (pas de pointeurs par exemple). En effet, les règles de la logique de Hoare ne fonctionnent pas dans ce cas. Cette restriction est effective dans l'industrie !

Les règles de déduction

Nous allons utiliser des règles de déduction qui auront toutes le format suivant :

$$\frac{\text{prémisses}}{\text{conclusion}}$$

« si toutes les prémisses sont vraies, la conclusion est vraie » (cf. séquents)

Les prémisses et la conclusion sont des formules de la logique de Hoare.

Le système déductif (1)

Règle (a). Axiome

$$\frac{}{\{ p(t) \}\quad x \leftarrow t\quad \{ p(x) \}}$$

Intuition : Si une propriété est vraie pour **t** alors, après exécution de « **x ← t** », la propriété est vraie pour **x**.

Le système déductif (2)

Règle (b). Composition

$$\frac{\{ p \}\ S1\ \{ q \} \quad ; \quad \{ q \}\ S2\ \{ r \}}{\{ p \}\quad S1 ; S2\quad \{ r \}}$$

Intuition : Une règle naturelle pour l'enchaînement séquentiel de deux portions de programme...

Le système déductif (3)

Règle (c1). Conditionnelle 1

$$\frac{\{p \wedge b\}\ S\ \{q\} \quad ; \quad \{p \wedge \neg b\}\ \{q\}}{\{p\}\quad \text{si b alors S fsi}\quad \{q\}}$$

Intuition : Dans tous les cas, le prédicat **q** est vrai.

Remarque :

$\{p \wedge \neg b\}\ \{q\}$ signifie en fait $((p \wedge \neg b) \Rightarrow q)$

Le système déductif (4)

Règle (c2). Conditionnelle 2

$$\frac{\{p \wedge b\}\ S1\ \{q\} \quad ; \quad \{p \wedge \neg b\}\ S2\ \{q\}}{\{p\}\quad \text{si b alors S1 sinon S2 fsi}\quad \{q\}}$$

Intuition : Dans tous les cas, le prédicat **q** est vrai.

Remarque :

$\{p \wedge \neg b\}\ \{q\}$ signifie en fait $((p \wedge \neg b) \Rightarrow q)$

Le système déductif (5)

Règle (d). Conséquence

$$\frac{\{p \Rightarrow q\}\ ;\ \{q\}\ S\ \{r\}\ ;\ \{r \Rightarrow s\}}{\{p\}\quad S\quad \{s\}}$$

Intérêt de la règle « conséquence »

Nous voulons prouver :

$\{entier(a)\}$
si (a < 0) alors a ← -a fsi
$\{a \geq 0\}$

selon la règle (c) (conditionnelle), nous avons :

$\{ entier(a) \wedge (a \geq 0) \} - \{ a \geq 0 \}$

OK mais

$\{entier(a) \wedge (a<0)\}$ a = -a $\{a > 0\}$

AIE....

règle (d) : $(a > 0) \Rightarrow (a \geq 0)$, on peut conclure !

Boucle (1)

Technique de preuve de la correction partielle (cas des boucles).

Format général :

```
<pré-condition>
Tant_que <test> faire
        <corps>
Fin-tq
<post-condition>
```


Boucle (2)

Technique de preuve de la correction partielle (cas des boucles).

Idée: On exhibe un invariant de boucle, c'est-à-dire un prédicat **p** telle que :

- **<pré-condition> = vrai** ⇒ **p = vrai**
- **p** est vrai après chaque itération du **<corps>**
- (**p = vrai et <test> = faux**) ⇒ (**<post-condition> = vrai**)

Preuve de Correction - Exemple

pré-condition

```
(x, y entiers) et (x ≥ 0) et (y ≥ x) et
    (b = vrai) et (xx = x) et (yy = y)
```

boucle

```
    tant que ( xx ≠ yy ) faire
        si b alors xx ← xx + 1
        sinon yy ← yy – 1
        b ← non b
    fin-tant-que
```

post-condition

```
    xx = ( x + y + 1 ) div 2
```

(xx+yy+b) div 2 = (x+y+1) div 2
est-il l'invariant de boucle ?

Le système déductif (5)

Règle (e). Invariant de boucle

$$\frac{\{p \wedge b\}\ S\ \{p\}}{\{p\}\ \text{tant-que b faire S fin-tq}\ \{p \wedge \neg b\}}$$

Exemple Invariant de boucle

On veut prouver :

```
{ val ≥ 0 }
        cpt ← val; res ← 0
(*)     tant-que ( cpt ≠ 0 ) faire
            res ← res + val;
            cpt ← cpt – 1
        fin-tq
{ res = val² }
```

p = (val² = val.cpt + res)

Exemple Invariant de boucle

On a :
p = (val² = val.cpt + res)
b = (cpt ≠ 0)
S : res ← res + val; cpt ← cpt – 1

Il faut prouver que :

1. avant la boucle, on a **p = vrai**
2. **{p ∧ b} S {p} :**
   {(val² = val.cpt + res) ∧ (cpt ≠ 0)}
   res ← res + val; cpt ← cpt – 1
   {(val² = val.cpt + res)}
3. **p ∧ ¬b ⇒ res = val²**

Exemple Invariant de boucle

3.

(val² = val.cpt + res) ∧ (cpt = 0)
⇒ (res = val²)

1. **{ val² = val.val + 0 }**
   cpt ← val;
   { val² = val.cpt + 0 }
   res ← 0;
   {val² = val.cpt + res}

Exemple Invariant de boucle

2.

{(val² = val(cpt) + res)) ∧ (cpt ≠ 0)}
{val² = val(cpt-1) + res + val}
res ← res + val
{val² = val(cpt-1) + res}
cpt ← cpt – 1
{val² = val.cpt + res}

Prouvez la formule suivante :

```
{entier(a) ∧ entier(b) ∧
        (a ≥ 0) ∧ (b > 0)}
     q ← 0;r ← a
     tant-que ( r ≥ b ) faire
        q ← q + 1; r ← r – b
     fin-tq
{(a = qb + r) ∧ (r ≥ 0) ∧ (r < b)}
```

85

Remarque : autres structures ?

Certains langages de programmation permettent l'utilisation d'autres structures de contrôle. Il est cependant toujours possible de se ramener aux structures prises en compte dans notre système :

- Si-Alors-Sinon imbriqués
- boucle « Tant que »

86

Cohérence du système déductif

On peut montrer que le système déductif présenté est cohérent (tout ce qu'il prouve est vrai) mais il n'est pas complet (certaines formules vraies ne sont pas prouvables avec les règles de ce système...)

87

Correction totale (1)

Le système déductif présenté permet de prouver la correction partielle d'un programme : si celui-ci termine, alors le résultat produit est correct.

Pour obtenir la correction totale d'un programme, il est également nécessaire de prouver sa terminaison.

Pour cela, il faut examiner précisément :

⇒ les structures itératives (boucles)

⇒ la récursivité

88

Terminaison structure de « boucle »

En toute généralité, le problème de la terminaison d'un programme est un problème indécidable.

En pratique, nous sommes très souvent dans le cas où il est possible de prouver qu'un programme termine...

89

Preuve de Terminaison (1)

Technique de preuve de terminaison (cas des boucles).

Format général :

`Tant_que <test> faire <corps>`

Idée : On exhibe une variable entière **v**, appelée variant de boucle, telle que :

- **v** $\in$ $\mathbf{N}^+$ (avant le tant_que, et après chaque itération)
- **v** diminue à chaque itération

90

Preuve de Terminaison (2)

Technique de preuve de terminaison (cas des boucles).

v diminue à chaque itération. c'est-à-dire :

1. $v_0 \in N^+$ est la valeur initiale de **v**
2. Pour toute valeur **v**, v_1, telle que **<test>=vrai** on a $v_1 > v_2 \geq 0$ sachant que v_2 est la nouvelle valeur de **v** après l'exécution du **<corps>**

Preuve de Terminaison - Exemple

pré-condition.

```
(x, y entiers) et (x ≥ 0) et (y ≥ x)
et (b booléen)
```

programme.

```
xx ← x ; yy ← y ; b ← vrai
tant que ( xx ≠ yy )faire
    si b alors xx ← xx + 1 ;
    sinon yy ← yy – 1;
    fsi
    b ← non b
fin-tant-que
```

yy-xx est le variant de boucle?...

Prouver la correction totale de la fonction **Carre**

```
fonction Carre(val : entier):
             entier
{        cpt ← val; res ← 0
         tant-que ( cpt ≠ 0 ) faire
             res ← res + val;
             cpt ← cpt – 1
         fin-tq
         retourne (res);
}
```

Carre doit retourner le carré de **val**

Méthodologie de la Preuve

- Spécifier la fonction "carré"
- Prouver la correction totale du code
  - terminaison
  - correction

Prouver la correction totale de la fonction **Reste**

```
fonction Reste (a,b : entier): entier
{
     q ← 0;r ← a
     tant-que ( r ≥ b ) faire
        q ← q + 1; r ← r – b
     fin-tq
     retourne (r)
}
```

Reste doit retourner le reste de la division entière de **a** par **b**

Fin...

C'est fini !...