

Logique

C₁) Logique propositionnelle

C₂) Logique prédictive.

C₁: Logique propositionnelle

I / Syntaxe

- ordre de priorité & associativité
- ⊢ inductive
- hauteur ff, substitution ff
- le principe d'induction
- TH
- DM

II / Sémantique

- stq formelle connecteurs
- stq TDV
- stq intuitive →
- satisfaisabilité & tautologie
- équivalence & ongts stq
- pples algébriques
- axiomatisation compétence
- syst. complet de connecteurs.

⑥

I/ Introduction

- émtq^t au XIX^e s.
- Deb XX : crise fond^t maths
- TH de complétude
- Pb logique
- Applications

I/ émtq^t au XIX^e s.

- ϕ , avoir vie bonne, se conforme pas Ω' , raisonnements syllogismes.
- Hstq^t, \emptyset maths utilisées des syllogismes pr^r dm^r TH.
- Depuis Aristote, très peu d'évol^t.
- Actu XIX^e : mathématisation de la logique
- Georges Boole, *The law of Thought*, invent^r logique propositionnelle.
- Gottlob Frege, *Begriffsschrift*, quantification.

II/ Deb XX : crise fond^t maths

- Paradoxe de Bertrand Russel dans la TH des ens. $U = \{x \mid x \notin x\}$.
 - $u \in U \Rightarrow$ par définit^r de U que $u \in u$
 - $u \notin U \Rightarrow$ $u \notin u$
- $[\underline{u \in u \Leftrightarrow u \notin u}]$ menace cohérence mathématiques.

• Prgm Hilbert : prgm finitistes
(user ens fini de principes : dm^r \forall maths)

• Kurt Gödel : DMQ^e prgm voué à ÉCHEC

• 1^{er} TH d'incomplétude : \exists proposition φ de l'arithmétique q^r n'est pas démontrable & dont la négation n'est pas démontrable.

• 2^{er} TH incompl. (1931) : Tout système logique contenant l'arithmétique est soit contradictoire, soit il ne peut dm^r sa propre cohérence.

→ on ne peut pas discerner certaines pp^ts sur objets finis.

III/ TH Complétude (1929) : Gödel a dm^r que le TH de arithmétique est démontrable.

G numérisé les élts.

- les n°s permettent de représenter énoncés logiqs
- arith. permet opérat's
- on pt représenter la logiq ^{el-m} ds arithm & la réduire à du calcul sur les n°s.

Alan Turing (1936), thèse de master, bis TH G:

- modèle de calcul q formalisme: machines ^{de} Turing
- exhiber machine q permet exécuter les autres : machine programmable.
- exhibe un pb q n'a pas sol^d algorithme : le pb de l'arrêt des machines de Turing.

• Syntaxe (ff programmables) & sémantique (modèles, grammaires, entités...)

II / Applications

- VDM - En programmation, recherche données de
S - optimisat' prog'm, éliminer code mort
- pb résolus, opt. sy contraintes
- prog'm^d p contraintes : pb graphes, jeux
- prog'm^d synchrone : généra't de code de machines à états finis par spec'^d logique.

- T - dmq automatique TH
- vérif. automatisée prog'ms
- certificat' prog'ms (logique de Hoare)
D - dmq^d continu algorithmes.

- On obtient un algorithme (^{reste} DE)
 - ↳ M dialectica de Gödel
 - ↳ CoQ (Curry Howard de Drujim)
 - ↳ extraire DM algorithmes
- article, *On the usual Effectiveness of Logic in computer science.*
 - complexité descriptive :
 - logiqs épistémiqs

VDM Vérification de Modèle (ff vaut de M fixé?)

S Satisfaisabilité (ff de ff est vaut ?)

T Tautologie (ff est vaut de \vee M ?)

D Démonstration (qu'est-ce dmq^d ?)

C1 Logique propositionnelle

(LP) \leftrightarrow articula^g vérité.

- énoncés vrais + construire autres vrais
- condit^g de vérité d'un énoncé
- dmg énoncé est gis vrai.

I Syntacce

① syntaxe : des termes ou formules.

Atoms

T et \perp : tautologie & absurd, true/false
(top/bottom)

FF composites

termes conjonc^g disjonc^g

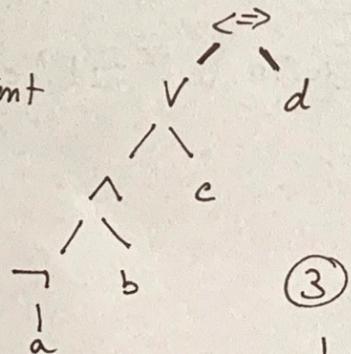
Syntaxe concrète & Abstraite.

ordre de priorité : $\neg > \wedge > \vee > \{\Rightarrow, \Leftrightarrow\}$

$$\neg a \wedge b \vee c \Leftrightarrow d$$

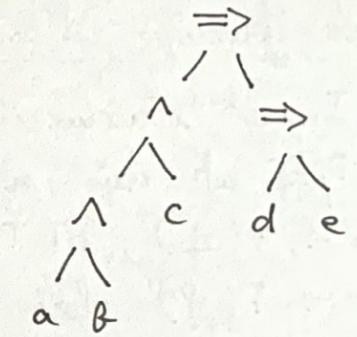
$$((\neg a \wedge b) \vee c) \Leftrightarrow d$$

dénotent



• Associativité :

$$(a \wedge b) \wedge c \Rightarrow (d \Rightarrow e) \text{ dmt}$$



• Défini^g inductive (réursive)

\rightarrow variable d'une ff : var(φ)

$$\text{var}(\varphi) = \emptyset$$

$$\text{var}(x) = \{x\}$$

$$\text{var}(\neg \varphi) = \text{var}(\varphi)$$

$$\text{var}(\varphi \text{ op } \psi) = \text{var}(\varphi) \cup \text{var}(\psi)$$

• Hauteur d'une ff : h(φ).

• Substitu^g ff : subs(φ, σ)

$$\text{subst}(\varphi, \sigma) = \varphi$$

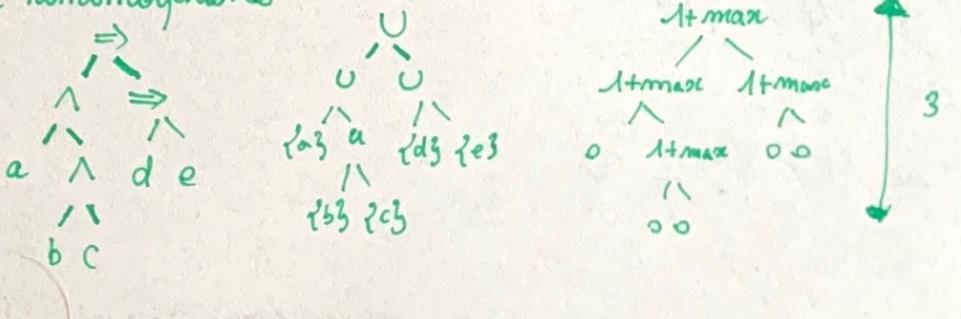
$$\text{subst}(x, \sigma) = \sigma(x)$$

$$\text{subst}(\neg \varphi, \sigma) = \neg \text{subst}(\varphi, \sigma)$$

si $\varphi \in \{T, \perp\}$.

\hookrightarrow ces définiti^gs st : homomorphismes (se base n^{qq chose})

@ homomorphismes



Le principe d'induction

Induction structurelle

Ppté est vraie n & ff ?

$$\rightarrow P(\perp), P(T), P(x)$$

$$\rightarrow P(\psi) \text{ alors } P'(\neg\psi)$$

$$\rightarrow \text{si } P(\psi) \wedge P(\psi) \text{ alors } P(\psi \text{ op } \psi) \quad \begin{matrix} \text{m t op} \\ \text{ds } \{\wedge, V, \Rightarrow, \Leftrightarrow\} \end{matrix}$$

\rightarrow induction : cas particulier de récurrence.

(TH) Une ff ψ des prop (x) et tte subs T,
si $\nvdash x$ dans X, $h(T(x)) \leq N$ alors
 $h(\text{subs}(\psi, T)) \leq h(\psi) + N$

(DM) par induction sur S^e de ψ :

$$\times \text{ cas } \psi = \perp \text{ ou } \psi = T$$

$$\times \text{ cas } \psi = \circ c$$

$$\times \text{ cas } \psi = \neg\psi$$

$$\times \text{ cas } \psi = \psi_1 \text{ op } \psi_2$$

II/ Sémantique

sémantique intuitive connecteurs

$$\circ T, \perp$$

◦ conjonction, disjonction, négation.

Sémantique formelle connecteurs :

Évalua^θ ff , valua^θ v.

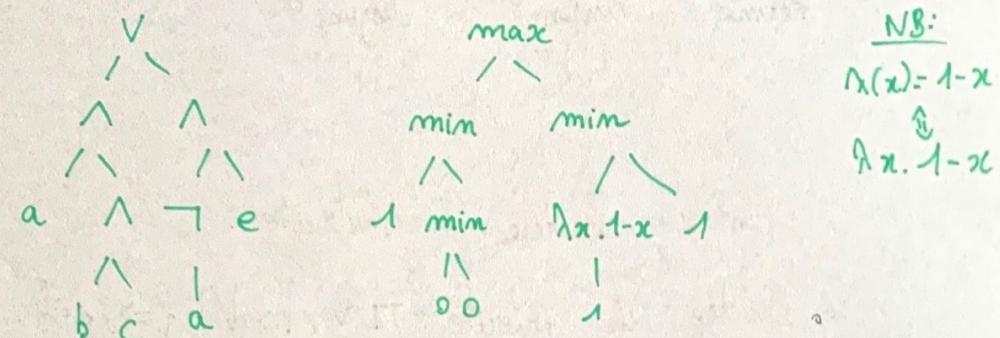
$$[\![T, v]\!] = 1 \quad [\![\neg\psi, v]\!] = 1 - [\![\psi, v]\!]$$

$$[\![\perp, v]\!] = 0 \quad [\![\psi \wedge \psi, v]\!] = \min([\![\psi, v]\!], [\![\psi, v]\!])$$

$$[\![x, v]\!] = v(x) \quad [\![\psi \vee \psi, v]\!] = \max(\dots)$$

@ T est valua^θ : a=1, b=0, c=0, d=1, e=1

Évalua^θ de $[\![a \wedge b \wedge c \vee \neg d \wedge e, T]\!]$



$$[\![a \wedge b \wedge c \vee \neg d \wedge e, T]\!] = \max([\![a \wedge b \wedge c, T]\!], [\![\neg d \wedge e, T]\!])$$

Sémantique par table de vérité

a	b	$\neg a$	$a \vee b$	$a \wedge b$
0	0	1	0	0
0	1	1	1	0
1	0	0	1	0
1	1	0	1	1

On prend le min pour \wedge , max pour \vee .
et $2^n - 1 - n$ pour \neg .

Sémantique intuitive de \Rightarrow

$\varphi \Rightarrow \psi$: si φ (est vraie) alors ψ (est vraie)

@ si le bord du carré est gris alors son intérieur est gris.

Table de vérité:

φ	ψ	$\varphi \Rightarrow \psi$
0	0	1
0	1	1
1	0	0
1	1	1

φ	ψ	$\varphi \Rightarrow \psi$	$\psi \Rightarrow \varphi$	$\varphi \Leftrightarrow \psi$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	1	1	1

$$\varphi \Leftrightarrow \psi = (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi) \quad (5)$$

a	b	c	$a = b$	$(a \Rightarrow b) \wedge c$	$\neg a$	$\neg a \wedge b$	$((a \Rightarrow b) \wedge c) \vee \neg a$
0	0	0	1	0	1	0	0
0	0	1	1	1	1	0	1
0	1	0	0	0	1	1	1
0	1	1	0	1	= 1	1	1
1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0
1	1	0	1	0	0	0	0
1	1	1	1	1	0	0	1

N^o lignes: 2^m , m: "variables"

Satisfaisabilité & Tautologie

[S] Une φ est s. s' \exists une valuation v tq

$$[\varphi, v] = 1.$$

On dit que v satisfait φ .

[T] Une φ est t. si \forall valuation v, $[\varphi, v] = 1$.

On note $I = \varphi$, & fait φ soit tautologie.

Équivalence & Conséquence sémantique

Équivalences sémantiques, $\varphi \equiv \psi$ lorsque tt valuation v,
 $[\varphi, v] = [\psi, v]$.

On peut internaliser ds logiq l'équivalence sémantique.
 $\varphi \equiv \psi$ mi $I = \varphi \Leftrightarrow \psi$

Congruence \equiv

Reflexivité $\varphi \equiv \varphi$

Symétrie $\varphi \equiv \psi \vdash \psi \equiv \varphi$

Transitive $\varphi \equiv \psi \wedge \psi \equiv \theta \vdash \varphi \equiv \theta$

Propriétés Algébriques

Commut

Associat

Distrib.

Identité: $\varphi \vee \perp \equiv \varphi, \varphi \wedge \top \equiv \varphi$

Zéro $\varphi \vee \top \equiv \top, \varphi \wedge \perp \equiv \perp$

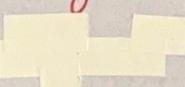
Idempotence $\varphi \vee \varphi \equiv \varphi, \varphi \wedge \varphi \equiv \varphi$

Absorption $\varphi \vee (\varphi \wedge \psi) \equiv \varphi, \varphi \wedge (\varphi \vee \psi) \equiv \varphi$

Complémentaire $\varphi \vee \neg \varphi \equiv \top, \varphi \wedge \neg \varphi \equiv \perp$

Double négation $\neg \neg \varphi \equiv \varphi$

Morgan $\neg(\varphi \vee \psi) \equiv \neg \varphi \wedge \neg \psi$

 $\neg(\varphi \wedge \psi) \equiv \neg \varphi \vee \neg \psi$

Définition $\varphi \Rightarrow \psi \equiv \neg \varphi \vee \psi$

Complexité de $\varphi \Leftrightarrow \psi \equiv (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$ équationnelle

Axiomatique de compétence.

si $\varphi \equiv \psi$, on peut réécrire φ équivalente données de axiomatique.

Système Complet de connecteurs

(SC): Si pr thé ff φ , \exists ff ψ tq:
→ ψ n'est construite à partir de connecteurs de φ
→ $\varphi \equiv \psi$.

@ SC $\{\vee, \neg\}, \{\wedge, \neg\}, \{\Rightarrow, \neg\}, \{\Rightarrow, \perp\}$

@ $\{\neg, \vee\}$, transformé \neg

- $\overline{\top} = \neg(x \vee \neg x)$
- $\overline{\perp} = x \vee \neg x$
- $\overline{x} = x$
- $\overline{\varphi \vee \psi} = \neg \varphi \vee \neg \psi$
- $\overline{\varphi \wedge \psi} = \neg(\neg \varphi \vee \neg \psi)$
- $\overline{\varphi \Rightarrow \psi} = \neg \varphi \vee \psi$
- $\overline{\varphi \Leftrightarrow \psi} = \neg(\neg \varphi \vee \neg \psi) \vee \neg(\neg \psi \vee \neg \varphi)$

 $\neg \varphi$ ne contient que \neg et \vee .

Listes en compréhension

o ens fermés.

► produit cartésien $I_1 \times I_2$.

$[(x, y) \text{ for } x \text{ in } I_1 \text{ for } y \text{ in } I_2]$

► intersection $I_1 \cap I_2$:

$[x \text{ for } x \text{ in } I_1 \text{ if } x \text{ in } I_2]$

► différence $I_1 - I_2$:

$[x \text{ for } x \text{ in } I_1 \text{ if } x \text{ not in } I_2]$

III / Algo Résolu SAT

→ algo Davis Putman (algo DP).

→ algo DPLL (Davis Putman Logemann Loveland)

Algo utilise opérations :

→ simplification

→ propagat unitaire

→ éliminat littéraux purs.

Simplification:

Dans clause : un littéral & un \neg littéral :

tjs satisfiable si une clause est tt le tps

(a n'est pas contrainte)

Propagat unitaire :

Si 1 seul littéral a : s'évalue vraie pour toutes clauses possibles.

@ $a \wedge (\neg a \vee b_1 \vee \dots \vee b_m) \wedge (a \vee c_1 \vee \dots \vee c_m) \wedge C_1 \wedge \dots \wedge C_p$

↳ impose la valeur de a : propagat la val^R des variables.

↳ j'élimine a .

$a=1$: $(c_1 \vee \dots \vee c_m) \wedge C_1 \wedge \dots \wedge C_p$

@ $\neg a \wedge (\neg a \vee b_1 \vee \dots \vee b_m) \wedge (a \vee c_1 \vee \dots \vee c_m) \wedge C_1 \wedge \dots \wedge C_p$

↳ impose la valeur: $a = 0$.

@ $x \neg a \wedge a \wedge C_1 \wedge \dots \wedge C_p \Rightarrow$ l'absurde.

la ff n'est pas satisfiable.

Éliminat des littéraux purs :

@ $a \wedge C_1 \wedge a \wedge \dots \wedge a \wedge C_p$: tjs positive $\Rightarrow a = 1$

ou
@ $\neg a \wedge C_1 \wedge \neg a \wedge \dots \wedge \neg a \wedge C_p$: tjs négative $\Rightarrow a = 0$

$\Rightarrow C_1 \wedge \dots \wedge C_p$.

M résultante :

soit $C_1 = a \vee b_1 \vee \dots \vee b_m$ et $C_2 = \neg a \vee c_1 \vee \dots \vee c_m$.
 $x = b_1 \vee \dots \vee b_m \vee c_1 \vee \dots \vee c_m$

est satisfiable si $C_1 \wedge C_2$ est satisfiable.

Si $C_1 \wedge C_2$ est satisfiable alors x l'est aussi.

Résultante pr pb SAT

$$(a \wedge b \dots \vee \dots) \wedge \dots (a \vee \dots \vee \dots) \wedge (\neg a \vee \dots)$$

→ remplacer des clauses par résultantes.

$$mbr\text{-résultante} = \text{max}(mbr\text{-ff contient } a, mbr\text{-ff contient } \neg a)$$

Algo Davis - Putnam

- faire simplificat si \textcircled{S}
- éliminer clauses unitaires sinon \textcircled{PU}
- éliminer littéraux plus sinon \textcircled{LP}
- éliminer variable de résultante \textcircled{R}

⇒ Transformer ff Ψ en Ψ' qui est satisfiable
 si Ψ' l'est; à la fin : \emptyset variables.

→ on a soit T, soit \perp .

$$@ (a \vee b) \wedge (\neg a \vee \neg c) \wedge (d)(d \vee \neg a) \\ \wedge (\neg d \vee e \vee f) \wedge (\neg b \vee c \vee e \vee \neg f) \wedge (\neg b \vee c)$$

⑤? non | \textcircled{PU} oui ⑥ → éliminer clauses contenant d.

$$(a \vee b) \wedge (\neg a \vee \neg c) \wedge (e \vee f) \wedge (\neg b \vee c \vee e \vee \neg f) \\ \wedge (\neg b \vee c).$$

⑤? non | \textcircled{PU} non | \textcircled{LP} oui → e apparaît tout seul ☺.
 → e : VRAI (val^r imposé)

$$(a \vee b) \wedge (\neg a \vee \neg c) \wedge (\neg b \vee c).$$

⑤? non | \textcircled{PU} non | \textcircled{LP} non | \textcircled{R} oui → choix n'implique pas de variable
 → i.e. sur a.

$$(b \vee \neg c) \wedge (\neg b \vee c) (c \vee \neg c)$$

\textcircled{R} sur b.

$$(c \vee \neg c) \quad \textcircled{S} \Rightarrow \textcircled{T}.$$

- Modif Algo pr avoir valeur.
- assignat de variables: on va garder en mémoire.
- qd on simplifie: on impose valeurs.
- \textcircled{PU} & \textcircled{LP} : garder en mémoire.
- pr calcul résultante:

$$(a \vee b_1 \vee \dots \vee b_m) \wedge (\neg a \vee c_1 \vee \dots \vee c_m)$$

$$\hookrightarrow a = \neg (b_1 \vee \dots \vee b_m) \vee c_1 \vee \dots \vee c_m.$$

Algo DPLL

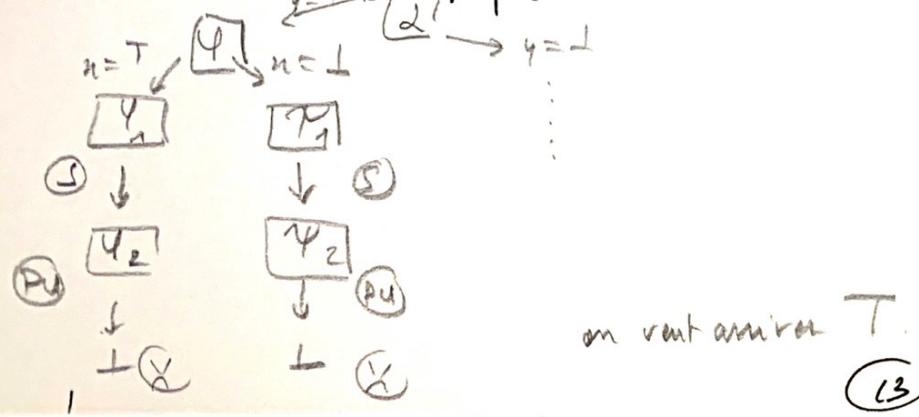
Ppté

Algo DPLL

- éliminer clauses unitaires.
- éliminer LP.
- simplifier résultante.

Alternative à résultante: propager clauses unitaires.

(si a vrai alors je propage la valeur de a part)



on rent arriver à T.

(13)

Algo DP·LL

- créer bcp clauses, us^e bcp mémoire.
 - mémoire virtuels.

Règles & valuations partielles

Pptó : soluto : valuta partecipante

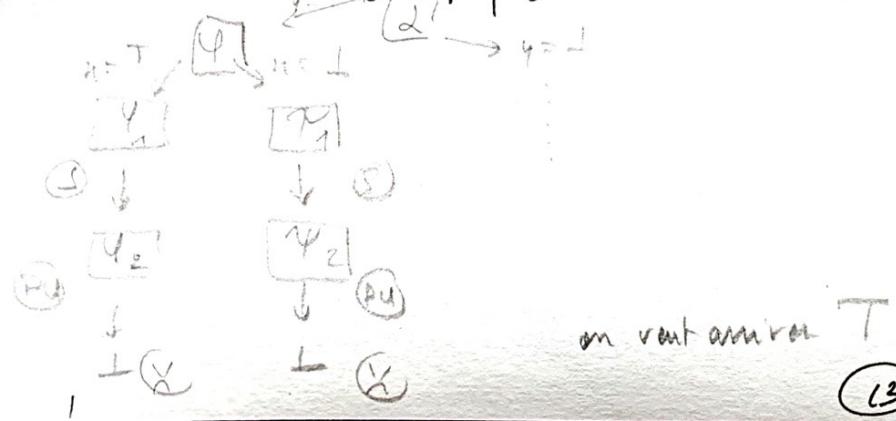
- n'a pas une val^R à chq var prop^{te} de signature.
 - chq règle assigne val^R vérité à variable
 - clauses simplifiées f assignat effectuées
 - si chx arrêt à ??! alors revenir en arrière: backtracking
 - sol^{VRS} SAT modernes : **apprentissage clauses** (determiner cause \Leftrightarrow backtracking)

Algò DPLL

- éliminer classes unitaires.
 - éliminer LP.
 - simplifier résultante.

Alternative à résultante: propager clauses unitaires.

(si a vrai alors je propage la valeur de a partit)



The expression $\{x \in I, p(x)\}$:

NB: Liste en compréhension: [x for x in l if p(x)]

Regles & valeurs partielles

- à except de résultante DP LL basé sur règles de algo DP mais on peut ajouter affect de variable à chaque élément :

- Simplification** (ici pas d'affectation) (i)
- Propagation unitaire** (ii)

$$(i) \quad (\cancel{a} \vee \cancel{b}_1 \vee \dots \vee \cancel{b}_m) \wedge c_1 \wedge \dots \wedge c_p$$

\Downarrow

$$c_1 \wedge \dots \wedge c_p.$$

$$(ii) \quad \begin{cases} \cancel{a} \wedge (\cancel{a} \vee b_1 \vee \dots \vee b_m) \wedge (a \vee c_1 \vee \dots \vee c_m) \wedge c_1 \wedge \dots \wedge c_p \\ a \wedge (\cancel{a} \vee b_1 \vee \dots \vee b_m) \wedge (\cancel{a} \vee c_1 \vee \dots \vee c_m) \wedge c_1 \wedge \dots \wedge c_p \end{cases}$$

\Downarrow

$$a=1 \quad (b_1 \vee \dots \vee b_m) \wedge c_1 \wedge \dots \wedge c_p.$$

$$\begin{cases} \cancel{a} \wedge (\cancel{a} \vee b_1 \vee \dots \vee b_m) \wedge (a \vee c_1 \vee \dots \vee c_m) \wedge c_1 \wedge \dots \wedge c_p \\ \cancel{a} \wedge (\cancel{a} \vee b_1 \vee \dots \vee b_m) \wedge (a \vee c_1 \vee \dots \vee c_m) \wedge c_1 \wedge \dots \wedge c_p \end{cases}$$

\Downarrow

$$a=0 \quad (c_1 \vee \dots \vee c_m) \wedge c_1 \wedge \dots \wedge c_p.$$

$$\cancel{a} \wedge a \wedge c_1 \wedge \dots \wedge c_p$$

\Downarrow

Élimination littéraux purs

$$\begin{aligned} & (\cancel{a} \vee b_1 \vee \dots \vee b_m) \wedge (\cancel{a} \vee c_1 \vee \dots \vee c_m) \wedge c_1 \\ & (\cancel{a} \vee b_1 \vee \dots \vee b_m) \wedge (\cancel{a} \vee c_1 \vee \dots \vee c_m) \wedge c_1 \\ & \qquad\qquad\qquad \Downarrow \\ & a=0 \quad c_1 \wedge \dots \wedge c_p. \end{aligned}$$

Recherche par cas

- on choisit une variable pivot, disons a
- on choisit une valeur pour cette variable
- on simplifie en propagant cette valeur

$a=1$

$$\begin{aligned} & (\cancel{a} \vee b_1 \vee \dots \vee b_m) \wedge (a \vee c_1 \vee \dots \vee c_m) \wedge c_1 \\ & (\cancel{a} \vee b_1 \vee \dots \vee b_m) \wedge (\cancel{a} \vee c_1 \vee \dots \vee c_m) \wedge c_1 \\ & \qquad\qquad\qquad \Downarrow \\ & (b_1 \vee \dots \vee b_m) \wedge c_1. \end{aligned}$$

Arbre de recherche (adr)

- maintenir adr pour pouvoir effectuer un backtrack

afin de pouvoir revenir sur des chaines effectuées lors des recherches par cas :

et nœuds de l'arbre de recherche :

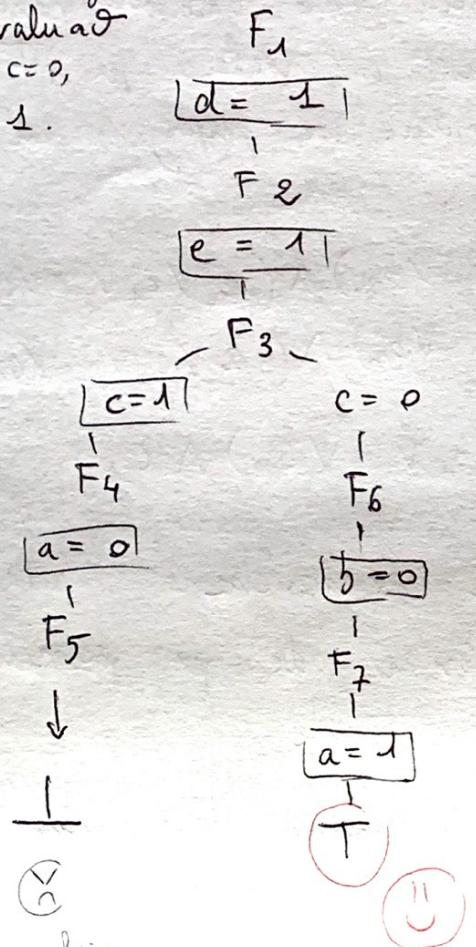
$$G_1 \wedge \dots \wedge G_m$$
$$\boxed{a=1} \quad \boxed{a=0}$$

$$D_1 \wedge \dots \wedge D_m \quad E_1 \wedge \dots \wedge E_p$$

② voir page ⑫

$$F_1 = (a \dots)$$

Donc F est satisfait
par la valuation
 $a=1, b=0, c=0,$
 $d=1, e=1.$



donc faire
backtrack

Saintje
workshop ⑭

L

Logique de 1^{er} ordre

La logiq propositionnelle (LP)

- syntaxe : représent^{ation} fls
- sémantiq → values
 - ↳ vrai en faise (vérific^{ation} modif)
- us^{ages} booléens
- résolu^t pb SAT
- codage pb à contraintes de SAT

Limites (LP)

↳ ne parle q vérités & n'pe objets.

LPO

point dépasser (LP)

Syntaxe Abstraite

Modélisa^{tion} objets & termes

en LPO, use termes pour modélisa objets & lang:

- @: - LP
- termes construits par opérations :

- ▷ constante: 0, gén^{érateur}
- ▷ opérat^{eur} unitaire : S, le successeur
- ▷ opér. binaires: + et x : addit^{ion} & multipliact^{ion}.

Termes typés : motiva^{tion} & exemple

→ on modifie, manipula^{tion} concepts modifiés

@ programm & entiers

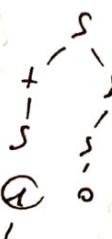
type Int

Expres	Type	Décr
i	Int	entier en écriture décimale
+	Int × Int → Int	op ^{erat^{eur}} 1 ^{er} addit ^{ion}
-	Int × Int → Int	
x	Int × Int → Int	
!	Var → Int	vérif ^{er} & val ^{uer} et à var

Expres	Type	Bool
		Décr

Expres	Type	Command
is then else	Bool × Cmd × Cmd → Cmd	
while	Bool × Cmd → Cmd	
:=	Var × Int → Cmd	
;	Cmd × Cmd → Command	

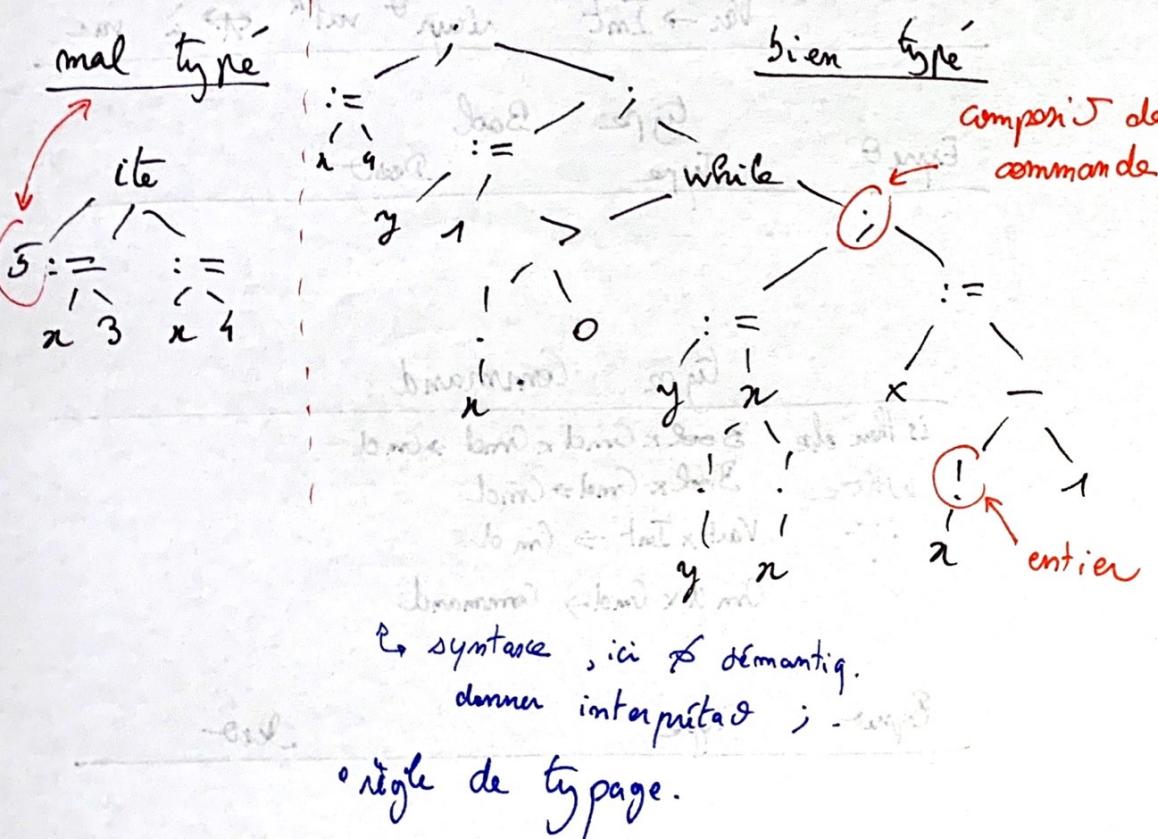
Expres	Type	Zero



Termes bien typé

- opérat^{re} typé, tbt; le des pt à un type:
- o soit t_1, \dots, t_m st tbt de types respectifs $\alpha_1, \dots, \alpha_m$: on note $t_1 : \alpha_1, \dots, t_m : \alpha_m$.
- o soit f un opérat^{re} de type $\alpha_1 \times \dots \times \alpha_m \rightarrow \alpha$.
- e le terme $f(t_1, \dots, t_m)$ est tbt de type α .

mal typé



Termes & variables

- une var mu constitue termes (types si termes typés).
- avoir termes paramétrés & éts termes.

Var(t) variable présente ds termes t.

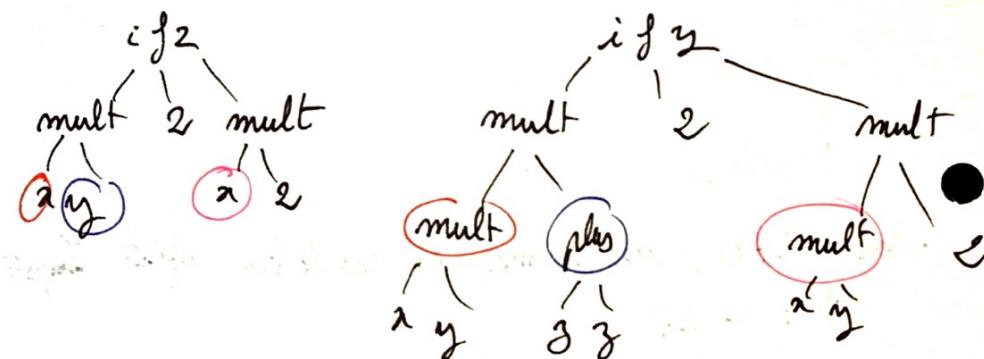
Substitution

$x_1 = t_1, \dots, x_m = t_m$ la sub. σ tq

$$\sigma(y) = \begin{cases} t_i & \text{si } y = x_i \\ \perp & \text{sinon.} \end{cases}$$

Appliq substitution à terme

L. σ ou $\sigma(t)$: le résultat de son appliq à



if2: opérat^{re} de 3 arguments (pas démantig)

Pb Unification

Unification

Termes t et u , trouver une substitution σ t.q.

$$\sigma(t) = \sigma(u) \text{ est pb unificat}$$

$$\text{modèle } \sigma = t = u$$

→ noticdes particularités: **filtrage**

Filtrage

Un pb unificat $t = u$ est :

→ un pb de f bsp: a ne contient pas variable.

Filtrage linéaire ou filtrage structural & programmable:

Interprétations des termes (donnée sémantique)

* Cas non typé.

Associer sémantique :

→ fixer un ensemble D , le domaine d'interprétation.

→ pr chq opérat de type $A_1 \times \dots \times A_m \rightarrow A_i$, associe une f

$$[[f]] : D^m \rightarrow D$$

On appelle modèle la paire $M = (D, [[\cdot]])$

@ Pour LP : σ, τ & opérat.

Pour interpréter un terme contenant des variables,

on sémantique $[[t, v]]_o$ d'un terme t est :

$$[[x, v]]_o = \sigma_x v(x).$$

$$[[f(t_1, \dots, t_n), v]]_o = [[f]]([[t_1, v]]_o, \dots, [[t_n, v]]_o)$$

* Cas typé :

Des types A_1, \dots, A_m

→ fixer famille d'ens $(D_i)_{i \in [1, m]}$

et domaine d'interprétation

→ pr chq opérat f de type $A_1 \times \dots \times A_m \rightarrow A_i$, on associe une f $[[f]]$ de $D_1 \times \dots \times D_m \rightarrow D_i$

On a modèle : $M = ((D_i)_{i \in [1, m]}, [[\cdot]])$

valuation

$$[[x, v]]_o = v(x)$$

$$[[f(t_1, \dots, t_n), v]]_o = [[f]]([[t_1, v]]_o, \dots, [[t_n, v]]_o)$$

si t est un terme de type A_i alors

$[[t, v]]_o$ est un élmt de D_i .

II / Les Relations

Vers la logiq :

- une env de **prédictats**.
- chq prédictat a une arité (type si + types).
- enrichir opérations.
- rajoute un type aux types des termes (**T**): **Bool**
- p chq opérat^e on ajoute un opérat^e de type $T \times \dots \times T \rightarrow \text{Bool}$ (on adapte au typage ^{pu}_{cas} types)
- on ajoute connect^s logiq $\wedge, \vee, \neg \dots$

Interprétat^a des **ff**

- **Bool** est interprét^a do 1's et 0's
- connect^s logiq idem q LP
- modèle : une telle interprétat^a
- les prédictats repris^t des **relat^s** entre les élts du domaine d'interprétat^a: c'est une représentat^a p **f caractéristiq**:

Se passer des termes

- il est possible de se passer des termes en ayant q des prédictats ; mln arguments
- p chq symbole de f d'arité m, on voit un prédictat p_f d'arité $m+1$ & on interprét^a: $\llbracket p_f \rrbracket (u_1, \dots, u_m, v) = \begin{cases} 1 & \text{si } v = \llbracket f \rrbracket (u_1, \dots, u_m) \\ 0 & \text{sinon} \end{cases}$

En ajoutant une variable p ss-terme, on peut ainsi transformer tt f en s ff ay^t m interprétat^a:

$$\leq (s(x), s(s(y))) \xrightarrow{\text{prédicat binaire}} p_s(x, sx) \wedge p_s(y, sy) \wedge p_<(sy, ssy) \wedge <(sx, ssy)$$

IV / La Quantificat^a

- \forall : universel
- \exists : existenciel

$$\forall x. \varphi$$

$$\exists x. \varphi$$

$$\forall x. \frac{}{\varphi}$$

$$\exists x. \frac{}{\varphi}$$

→ les quantif. t^os de var et des lieux de φ :

Satisfaisabilité

ψ est satisfait par modèle v si $\llbracket \psi, v \rrbracket_{of} = 1$: $v \models \psi$

$VL(\forall x. \varphi) = VL(\varphi) - \{x\}$ variable liée

$VL(\exists x. \varphi) = VL(\varphi) + \{x\}$

$VL(f(t_1, \dots, t_n)) = \bigcup_{i=1}^n VL(t_i)$

$VL(n) = \{n\}$.

• \bullet une ψ est dite telle que $VL(\psi) = \emptyset$.

Interprétation quantificat°

si M : modèle des opérand° & predicat°:

$$\llbracket \forall x. \varphi, v \rrbracket_{of} = \min \left\{ \llbracket \varphi[x=d] \rrbracket_{of} \mid d \in D \right\} \text{ "n"}$$

$$\llbracket \exists x. \varphi, v \rrbracket_{of} = \max \left\{ \llbracket \varphi[x=d] \rrbracket_{of} \mid d \in D \right\} \text{ "v"}$$

vaut 1 si tous les valuations valent 0.

FND
FNC

! ici 2 arguments

min(binaire)

min(bis) gomomorphie p élts
que tout do D

pt è fini :

→ éliminer quantificat° si ens fini $\neq \infty$

& remplacer par 1 ou 0.

⑤

→ les quantif. tels de var et des lieux de ff:

$$VL(\forall x. \varphi) = VL(\varphi) - \{x\}$$

$$VL(\exists x. \varphi) = VL(\varphi) - \{x\}$$

$$VL(f(t_1, \dots, t_n)) = \bigcup_{i=1}^n VL(t_i)$$

modèle au QL

$$VL(n) = \{x\}$$

une ff est clause si $VL(\varphi) = \emptyset$.

Interprétation quantificat°

- si M : modèle des opérand° & prédict°:

$$\llbracket \forall x. \varphi, v \rrbracket_{of} = \min \{ \llbracket \varphi, v[x=d] \rrbracket_{of} \mid d \in D \} \quad \text{v}$$
$$\llbracket \exists x. \varphi, v \rrbracket_{of} = \max \{ \llbracket \varphi, v[x=d] \rrbracket_{of} \mid d \in D \} \quad \checkmark$$

vaut 0 si tous les valuations valent 0.

FND
FNC
D ici 2 arguments
 \min (binnaire)

\min (ou paramétré p élts)
q se tut des D
pt è fini:

→ éliminer quantificat° si on finit à ∞
& remplacer par 1 ou 0.

Satisfaisabilité

φ est satisfait si et modélisé si.

$$\llbracket \varphi, v \rrbracket_{of} = 1 : \boxed{of, v \models \varphi}$$

Tautologie

$$v : of, v \models \varphi$$

φ est une tautologie si et modélisé si

Résumé: (LP) → énoncé vrai/faux

L^{1°}O: pplo n objets:

→ opérat° (@ + et * : opé binaires & ctlo)
→ prédict° (@ =)
→ langage logiq

④ Australie

Prédicats :

- **prédicats unaires**: couleur et état.

- **prédicats binaires**: voisin et colorie

$$\text{voisin} = \{\{WA, NT\}, \{WA, SA\}, \dots\}$$

symétrique

$$\text{colorie} = \{(x, WA), (V, NT), \dots\}$$

- Tout état est colorié par une couleur :

$$\forall x. \text{état}(x) \Rightarrow \exists y. \text{colorie}(y, x)$$

- Si deux états sont voisins ils sont coloriés par couleurs \neq :

$$\forall x y. \text{état}(x) \wedge \text{état}(y) \wedge \text{voisin}(x, y) \Rightarrow$$

$$\forall c_1. \forall c_2. \text{colorie}(c_1, x) \wedge \text{colorie}(c_2, y)$$

- Tout état est colorié par une couleur :

$$\forall x. \text{état}(x) \Rightarrow \exists y. \text{couleur}(y) \wedge \text{colorie}(y, x)$$

- Si 2 états sont voisins

$$\text{couleur} = \{r, v, b\}$$

$$\text{état} = \{WA, NT, Q, SA, NSW, V, T\}$$

Axiomatique

Ax 0 : tous les objets doivent vérifier

Ax 0 des entiers de Peano

$$\forall x. \neg S(x) = 0$$

$$\forall x. x = 0 \vee \exists y. x = S(y)$$

$$\forall x \forall y. S(x) = S(y) \Rightarrow x = y.$$

injectivité du successeur

$$\forall x \forall y. x + S(y) = S(x + y)$$

$$\forall x. x \neq 0 = 0$$

$$\forall x. \forall y. x \neq S(y) = (x \neq y) + x$$

• schéma d'induction, pr tte ff $\varphi(x, x_1, \dots, x_m)$:

$$\forall x_1 \dots \forall x_m. (\varphi(0, x_1, \dots, x_m) \wedge \forall x. \varphi(x, x_1, \dots, x_m))$$

$$\Rightarrow \varphi(S(x), x_1, \dots, x_m) \Rightarrow \forall x. \varphi(x, x_1, \dots, x_m)$$

Traduire du français vers logique

Approche de **Sujet / Prédicat**

• **sujet**: ce dont on parle (sujet grammatical)

• **prédicat**: ce qu'on en dit (groupe verbal)

④ Tous les étudiants apprennent l'anglais.

sujet

prédicat

$$\forall x. \text{student}(x) \Rightarrow \text{learn}(x, \text{english})$$

Traduction quantifiée - II

Tous le

$$\forall x. \underbrace{\text{ sujet}(x)}_{\text{repère pré expré & sujet}} \Rightarrow \underbrace{\text{ prédicat}(x)}_{\text{ - opt' e méd.}}$$

repère pré expré & sujet ~ opt' e méd.

@ . cté : english, german

• prédicats : student (unaire), learn (binaire),
learn(x,y) i.e. x apprend y .

\rightarrow Tous les étudiants qui apprennent l'anglais n'apprennent pas allemand.

$$\forall x. \underbrace{\text{student}(x)}_{\text{sujet}} \wedge \underbrace{\text{learn}(x, \text{english})}_{\text{prédicat}} \Rightarrow \neg \underbrace{\text{learn}(x, \text{german})}_{\text{prédicat}}$$

\rightarrow Aucun étudiant n'apprend l'anglais et l'allemand

$$\forall x. \text{student}(x) \Rightarrow \neg (\text{learn}(x, \text{german}) \wedge \text{learn}(x, \text{english}))$$

$$\forall x. \text{student}(x) \stackrel{=}{\Rightarrow} (\neg \text{learn}(x, \text{german}) \vee \neg \text{learn}(x, \text{english}))$$

il existe ...

$$\exists x. \text{ sujet}(x) \wedge \text{ prédicat}(x)$$

@ Certains étudiants apprennent l'anglais

$$\exists x. \text{student}(x) \wedge \text{learn}(x, \text{english})$$

@ Certains étudiants qui apprennent en n'apprennent pas allemand

$$\exists x. \text{student}(x) \wedge (\text{learn}(x, \text{german}) \wedge \neg \text{learn}(x, \text{english}))$$

Dualité traduction quantifiée

$$\neg (\forall x. \text{ sujet}(x) \Rightarrow \text{ prédicat}(x)) \equiv \exists x. \neg (\text{ sujet}(x) \Rightarrow \text{ prédicat}(x))$$

$$\equiv \exists x. \neg (\neg \text{ sujet}(x) \vee \text{ prédicat}(x))$$

$$\exists x. \text{ sujet}$$

$$\neg (\exists x. \text{ sujet}(x) \wedge \text{ prédicat}(x)) \equiv \forall x. \neg (\text{ sujet}(x) \wedge \text{ prédicat}(x))$$

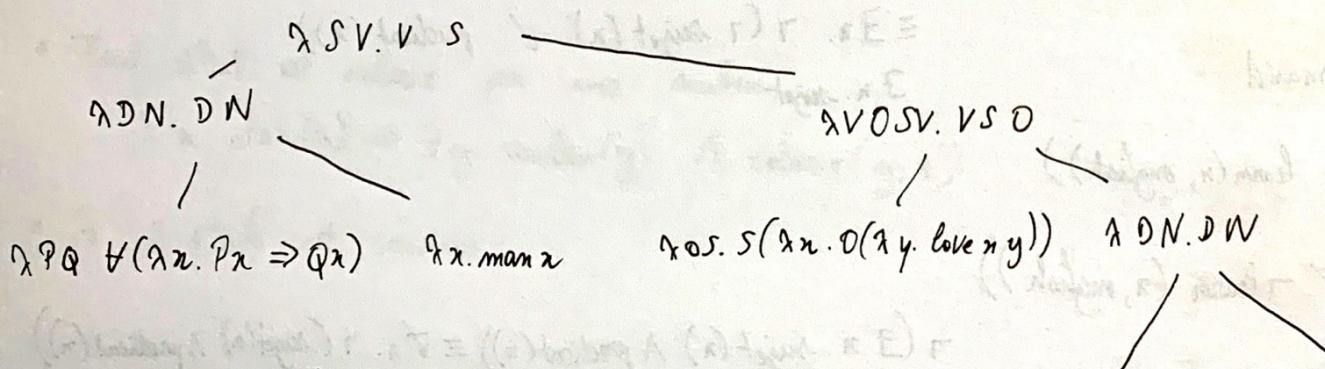
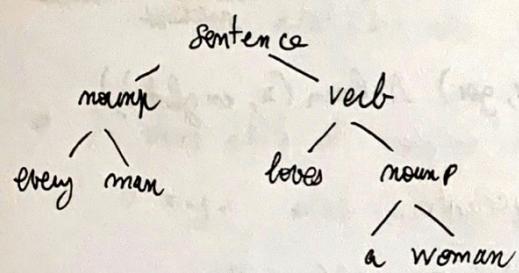
$$\equiv \forall x. \neg \text{ sujet}(x) \vee \neg \text{ prédicat}(x)$$

$$\equiv \exists x. \text{ sujet}(x) \Rightarrow \neg \text{ prédicat}(x)$$

En aller + loin : la sémantique de Montague

- linguistique & philosophie se sont intéressés à $\leftrightarrow^0 \leftrightarrow^{\text{texte et sens}}$
- 70's, Richard Montague :
 - English as a formal Language.
 - Universal Grammar
 - Pragmatics & intentional logic
 - The proper treatment of quantification in ordinary English.

Interprétation sémantique de grammatical



every man loves a woman : 2 sens :

- $\forall(\forall u. \text{man } u \Rightarrow \exists(\forall z. \text{woman } z \wedge \text{love } u z))$
- $\exists(\forall z. \text{woman } z \wedge \forall u. \text{man } u \Rightarrow \text{love } u z)$

ambiguité : Re/de Dicto ⑧

- Jean croit q qq manipule les médias :
- Il croit q il est une personne q manipule les médias
- Il q q dont Il croit qu'il manipule les médias.

Logique & Programmation

M & implementation

M pb $\xrightarrow{\text{implementation}}$ code du logiciel

OB : on a code concé implementer notre M et
mais pas correctement.

Pro : code \leftrightarrow **fonction des configurations** (part de faire)
optimisation

à language while

Environnements: mod configuration

Envi & cmd's

- Un envi: (m mod valua0) est f partielle à rapport jimi des variables ds les valus.
Repr'st l'état de la mémoire d'un prog.

- les cmd's t' envir:

$$x \leftarrow \text{True}$$

$$y \leftarrow 4$$

$$z \leftarrow -45$$

$$\frac{x \leftarrow \text{True} \quad y = z + y * z}{y \leftarrow -37} \quad z \leftarrow -45$$

- (R9) Le debugger de Thonny peut observer facilement l'act des commandes sur les environnements.

Logiq & envi

Variables & pp'tés et envi.

si on fixe x_1, \dots, x_m variables, une ff logiq $\Psi(x_1, \dots, x_m)$ est une ppte n'envi q assignent des valeurs aux variables x_1, \dots, x_m

$$E \models \Psi(x_1, \dots, x_m) \quad (E \text{ est envi q satisfait } \Psi(x_1, \dots, x_m))$$

$$E \not\models \Psi(x_1, \dots, x_m)$$

① x et y ne st pas PEE.

$\Psi(x, y) = \exists z. \neg z = 1 \wedge (\exists d_1. y = d_1 * z) \wedge (\exists d_2. x = d_2 * z)$
désigne les envi q assignent à la variable x & à var y
des mtrs q ont diviseur commun \neq de 1.

Pré-condids & post-condids

Transforma t' envi P prog

Étant donné prog p & 2 environnements E, E':

$$E \triangleright_p E'$$

Le fait que p transforme t'envi E en t'envi E'.

Paire de pré-condid & post

Étant donné un programme p les ff Ψ et Ψ' st appellées respectivement pré-condid & post-condid de p lorsq:

Pour tout E et E' tq $E \triangleright_p E'$,

si $E \models \Psi$ alors $E' \models \Psi'$.

Gm appelle (Ψ, Ψ') paire de pré/post-condids de p:

$$\Psi \triangleright_p \Psi'$$

Qqs @ Incrementation : $\pi = \pi + 1$

Il a 3 paires de pré / post-conditions :

- $(x=1, x=2)$, i.e. $x=1 \rightarrow x=x+1 \rightarrow x=2$
- $(x < 1, x < 2)$, i.e. $x < 1 \rightarrow x=x+1 \rightarrow x < 2$
- $(x \neq 0, x \neq 1)$, i.e. $x \neq 0 \rightarrow x=x+1 \rightarrow x \neq 1$.

@ Échange de valeur

Mgm p:

$$\begin{array}{l} z=2 \\ x=y \\ y=z \end{array}$$

- $(x=4 \wedge y=5, x=5 \wedge y=4)$, i.e. $x=4 \wedge y=5 \rightarrow p \rightarrow x=5 \wedge y=4$.

- $(\exists v. x=v \vee v=y = v \vee v \neq v, \exists v. y=v \vee v \wedge x=v \neq v \vee v \wedge z=x)$

$\exists v. x=v \vee v \wedge y=v \vee v \neq v \rightarrow p \rightarrow \exists v. y=v \vee v \wedge x=v \neq v \vee v \wedge z=x$

Raisonnement & pré / post-conditions

Closure par conjonction

Si (Ψ_1, Ψ_1) et (Ψ_2, Ψ_2) sont des paires de pré/post-conditions de p alors $(\Psi_1 \wedge \Psi_2, \Psi_1 \wedge \Psi_2)$ est une paire de pré/post-conditions.

Implications

Si (Ψ_1, Ψ_1) est une paire de pré/post-conditions de p, et

- $I = \Psi_2 \Rightarrow \Psi_1$.
- $I = \Psi_1 \Rightarrow \Psi_2$.

alors (Ψ_2, Ψ_2) est une paire de pré/post-conditions

Invariant de boucle

Invariant

Si (Ψ, Ψ) est une paire de pré/post-conditions de p, alors Ψ est invariant de p.

Invariant de boucle

Si $(c \wr \Psi, \Psi)$ est une paire de pré/post-conditions de p, on appelle invariant de boucle de .

while c : p

Démontrer un prgm

Etant donné un prgm p, pr d'm q qu'il calcule ^{au q ln} souhaité.

- spécifier de la logiq une pp'té Ψ de l'environnement ut
- spécifier d'une ff Ψ les envir qu le prgm est censé transformer (suite : spécificatio formelle).

Sémantique opérationnelle : système formel

$$\boxed{[e, E] = v}$$

$$E \triangleright x = e \triangleright E[x \leftarrow v]$$

$$E \triangleright c_1 \triangleright E' \quad E' \triangleright c_2 \triangleright E''$$

$$E \triangleright c_1; c_2 \triangleright E''$$

$$[e, E] = \text{True} \quad E \triangleright ct \triangleright E'$$

$$E \triangleright \text{if } e: ct \text{ else: } ce \triangleright E'$$

$$[e, E] = \text{err}$$

$$E \triangleright \text{if } c: ct \text{ else: } ce \triangleright \text{err}$$

$$[e, E] = \text{True} \quad E \triangleright cw \triangleright E' \quad E' \triangleright \text{while } e: cw \triangleright E''$$

$$E \triangleright \text{while } e: cw \triangleright E''$$

$$[e, E] = \text{False}$$

$$E \triangleright \text{while } e: cw \triangleright E$$

$$[e, E] = \text{err}$$

$$E \triangleright \text{while } e: cw \triangleright \text{err}$$

$$\boxed{[e, E] = \text{err}}$$

$$E \triangleright x = e \triangleright \text{err}$$

$$E \triangleright c_1 \triangleright \text{err}$$

$$E \triangleright c_1; c_2 \triangleright \text{err}$$

$$[e, E] = \text{False} \quad E \triangleright ce \triangleright E'$$

$$E \triangleright \text{if } c: ct \text{ else: } ce \triangleright E'$$

$$I = \varphi \Rightarrow \psi \quad I = \psi \Rightarrow \psi' \quad \varphi \triangleright c \triangleright \psi$$

$$\varphi' \triangleright c \triangleright \psi'$$

$$\varphi(e/x) \triangleright x = e \triangleright \psi$$

$$\frac{\varphi \triangleright c_1 \triangleright \psi \quad \psi \triangleright c_2 \triangleright \theta}{\varphi \triangleright c_1; c_2 \triangleright \theta}$$

$$\frac{\varphi \wedge c \triangleright ct \triangleright \psi}{\varphi \wedge \neg c \triangleright ce \triangleright \psi}$$

$$\varphi \triangleright \text{if } c: ct \text{ else } ce \triangleright \psi$$

$$\varphi \wedge c \triangleright cw \triangleright \psi$$

$$\varphi \triangleright \text{while } c: cw \triangleright \varphi \wedge \neg c$$

La notation $\varphi \triangleright c \triangleright \psi$ signifie que (φ, ψ) est une paire de pré-condit & post-condit du code c .

→ Debugger Thonny : simple use

↳ acquérir 1ère sémantiq opérationnelle, définie formell + sous la forme d'intuit logiq de Hoare.

$$\underline{NB}: \forall x \in A \quad \varphi(x)$$

$$\hookrightarrow \forall x \quad x \in A \Rightarrow \varphi(x)$$

$$\forall x \quad A(x) \Rightarrow \varphi(x)$$

φ : prédicat binaire.

n	y	$n \Rightarrow y$
00	1	
01	1	
10	0	
11	1	

11