

# Eléments de logique pour l'informatique

Uli Fahrenberg

`uli@lmf.cnrs.fr`

*d'après Christine Paulin*

Département Informatique, Faculté des Sciences d'Orsay, Université Paris-Saclay

Licence Informatique - LDD Informatique, Mathématiques

2025–26

# Introduction au cours de logique

- 1 Motivations
- 2 Organisation du cours
- 3 Exemples d'usage de la logique

# La logique, chemin vers la vérité

- *Logique* vient du grec logos (raison, langage, raisonnement).
- La logique manipule des *énoncés* (phrases dont le sens est vrai ou faux).
  - “Tous les moutons ont 5 pattes”
  - “Aucun étudiant ne joue sur son téléphone pendant le cours de logique”
  - “Les trains ne passent pas à un passage à niveau ouvert”
- “La couleur du cheval blanc d’Henri IV” *n’est pas un énoncé*
- Importance du langage pour préciser de quoi on parle (souvent implicite)
- Un énoncé peut être vrai ou faux suivant la manière dont on *interprète* les mots
- La logique est une manière *scientifique* d’étudier la notion de vérité

- L'arithmétique étudie les propriétés des nombres :  $0, 1, 2, 3, \dots$
- La logique (classique) s'intéresse à un espace beaucoup plus simple : *vrai / faux*
- Les **connecteurs logiques** sont des *opérations* qui permettent de former de nouveaux énoncés
  - la **négation** d'un énoncé  $E$  : la négation d'un énoncé  $E$  est vraie si et seulement si l'énoncé  $E$  est faux
  - la **conjonction** : l'énoncé  $E_1$  *et*  $E_2$  est vrai si et seulement si les énoncés  $E_1$  et  $E_2$  sont simultanément vrais
  - la **disjonction** : l'énoncé  $E_1$  *ou*  $E_2$  est vrai si et seulement si l'un des deux énoncés  $E_1$  et  $E_2$  est vrai (ou les deux)
  - ...

# Énoncé paramétré et quantification

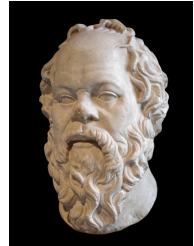
- Un énoncé logique *paramétré* représente une propriété (vraie ou fausse) d'un "*objet*" quelconque
- exemple : propriété pour un étudiant de *valider le cours de logique*
- "*valider le cours de logique*" sera vrai ou faux pour chaque étudiant considéré
  - *Toto valide le cours de logique*
- on peut former de nouveaux énoncés comme :
  - tous les objets vérifient la propriété,
  - il existe (au moins) un objet qui vérifie la propriété,
  - il existe précisément 42 objets qui vérifient la propriété.

# La logique, chemin vers la vérité

- Le **raisonnement** est un cheminement (une **déduction**, une **preuve**) qui permet de relier entre eux des énoncés : on distingue des *hypothèses*, et une *conclusion*.
- On veut s'assurer que dans toute situation où les *hypothèses sont vraies*, il en est de même de la conclusion.
- Un raisonnement suit des *règles* logiques précises.
- Le raisonnement est un procédé suffisamment élémentaire pour *convaincre*
  - modus ponens, preuve par l'absurde, preuve par récurrence ...
- **Montrer** qu'un énoncé est vrai est, en général, **indécidable** (il n'y a pas de programme qui répond à cette question).
- **Vérifier** qu'un raisonnement suit bien les règles du jeu peut, le plus souvent, s'effectuer mécaniquement.
- Certains raisonnements sont adaptés à l'humain, d'autres aux machines.

# Exemples de raisonnement

*Tous les hommes sont mortels  
or Socrate est un homme  
**donc** Socrate est mortel.*



*Tous les hommes sont mortels  
or l'âne Francis n'est pas un homme  
**donc** l'âne Francis est immortel.*



Dialogue entre le Logicien et le Vieux Monsieur, extrait de Rhinoceros (Ionesco)

- Je vais vous expliquer le syllogisme.
- Ah ! oui, le syllogisme !
- Le syllogisme comprend la proposition principale, la secondaire et la conclusion.
- Quelle conclusion ?
- Voici donc un syllogisme exemplaire. Le chat a quatre pattes. Isidore et Fricot ont chacun quatre pattes. Donc Isidore et Fricot sont chats.
- Mon chien aussi a quatre pattes.
- Alors, c'est un chat.
- Donc, logiquement, mon chien serait un chat.
- Logiquement, oui. Mais le contraire est aussi vrai.
- C'est très beau, la logique.
- A condition de ne pas en abuser...



- Démontrer c'est apporter une *évidence* du fait que quelque chose est vrai

- Plusieurs sortes de preuve 😊

(<http://www.pion.ch/Logic/preuves.html>)

- par l'exemple : *On démontre le cas  $n = 2$  qui contient la plupart des idées de la preuve générale.*
- par généralisation : *Ça marche pour 17, donc ça marche pour tout nombre réel.*
- par fin de cours : *Vue l'heure, je laisserai la preuve de ce théorème en exercice.*
- par probabilité : *Une recherche longue et minutieuse n'a mis à jour aucun contre-exemple.*
- par tautologie : *Le théorème est vrai car le théorème est vrai.*

- Dans ce cours 😐

- des méthodes rigoureuses
- transposables sur ordinateur

- La logique *formalise un langage*, en *définit le sens* et propose *des règles du jeu* qui permettent de se convaincre de la vérité d'une argumentation ou au contraire de la réfuter.
- *Questions logiques :*
  - avec quels *objets* joue-t-on ? que représentent-ils ? quelles règles ?
  - les règles du jeu sont-elles trop *laxistes* (on déduit des choses fausses, on dit que le système est *incohérent*)
  - les règles du jeu sont-elles trop *strictes* (on n'arrive pas à prouver quelque chose qui est pourtant correct, on dit que le système est *incomplet*)
  - peut-on changer les règles du jeu ? changer de style ?
- *Questions informatiques :*
  - un ordinateur peut-il *raisonner* ?
  - est-ce qu'il existe un *algorithme* pour dire qu'une formule est vraie, est fausse ?
  - étant données des hypothèses et une conclusion, peut-on reconstruire une déduction ?

- Le questionnement sur les fondements des mathématiques date du début du 20ème siècle avec la théorie des ensembles
- Quelques logiciens importants : Tarski, Russell, Hilbert, Gödel ...
- Des surprises :
  - le raisonnement ne peut pas se ramener au calcul
  - impact du langage sur la cohérence : paradoxe de Russell
    - tout est ensemble,  $x \in X$ , compréhension  $\{x|P(x)\}$
    - $X = \{x|x \notin x\}$ ,  $X \in X$  si et seulement si  $X \notin X$
    - il n'y a pas d'ensemble de tous les ensembles/il faut structurer les ensembles avec des *types*
  - théorème d'incomplétude de Gödel :
    - soit un système de déduction dans lequel on peut raisonner sur les entiers
    - il existe une formule *C* telle que on ne peut démontrer ni *C*, ni la négation de *C*
    - aucun système de démonstration "puissant" (ex. arithmétique, théorie des ensembles) ne capture toute la vérité...

- *Logique mathématique* : les énoncés mathématiques, le raisonnement sont l'objet de l'étude comme les nombres en algèbre, les fonctions en analyse, les espaces vectoriels . . .
- On raisonne sur ces objets, on établit des théorèmes :
  - *deux niveaux* différents qu'il ne faut pas confondre.
- Analogie informatique : programmes qui manipulent d'autres programmes (*les compilateurs*)

- Lien entre calcul booléen (vrai/faux) et circuits logiques (0/1)
- Une démarche analogue : machine universelle reposant sur un nombre limité d'opérations ; liens entre syntaxe et sémantique.
- Intelligence artificielle : munir un ordinateur qui sait calculer de capacité de raisonnement nécessite de transformer le raisonnement en calcul (méthodes symboliques liées à la logique, enjeu de l'explication des méthodes statistiques).
- Outil de modélisation : contraintes dans les bases de données, développement de programmes, web sémantique, apprentissage symbolique, ... (logique au service de l'informatique)
- Structures informatiques pour représenter des propriétés logiques, outils pour les manipuler (informatique au service de la logique).

# Introduction au cours de logique

- 1 Motivations
- 2 Organisation du cours
- 3 Exemples d'usage de la logique

- Prérequis
    - les bases du calcul booléen
    - compréhension des formules et preuves mathématiques élémentaires
  - Connaître et savoir manipuler le langage de la logique du premier ordre
    - savoir traduire des formules logiques en langue naturelle
    - savoir modéliser un problème en termes logiques
    - reconnaître certaines catégories de formules
    - savoir donner un sens aux formules de la logique
  - Savoir mettre en œuvre plusieurs notions de démonstration
  - Connaître les principales limites des méthodes d'un point de vue calcul
- LDD** Connaître et comprendre quelques résultats clés de la logique : complétude, compacité, démonstrations. . .

- Savoir présenter un raisonnement scientifiquement correct
- Apprendre un nouveau langage formel
- Distinguer syntaxe et sémantique
- Savoir manipuler des algorithmes sur des objets symboliques
- Méthodologie : mise en pratique d'objets mathématiques utiles en informatique



# Plan du cours

- 1 Maîtriser le langage logique
- 2 Donner du sens aux formules
- 3 Manipuler les formules de la logique
- 4 Automatiser les démonstrations



Serenella Cerrito.

Logique pour l'Informatique : une introduction à la déduction automatique.  
Vuibert Publisher Co, 2008.



Stéphane Devismes, Pascal Lafourcade, and Michel Lévy.

Informatique théorique : Logique et démonstration automatique, Introduction à la logique propositionnelle et à la logique du premier ordre.  
Ellipses, 2012.



Robert Cori and Daniel Lascar.

Logique Mathématique.  
Axiomes. Masson, 1993.



René David, Karim Nour, and Christophe Raffalli.

Introduction à la Logique, Théorie de la démonstration.  
Dunod, 2001.



Gilles Dowek.

La logique.  
Le Pommier, 2015.



Gilles Dowek.

Les démonstrations et les algorithmes.  
Les éditions de l'Ecole Polytechnique, 2010.



Pierre Le Barbenchon, Sophie Pinchinat, and François Schwarzentruher.

Logique : fondements et applications.  
Dunod, 2022.

- Espace ecampus des formations
  - emplois du temps, groupes, examens. . .
- Espace ecampus du cours *Eléments de logique pour l'informatique*
  - Espace partagé entre les parcours licence et LDD, classique et apprentissage
  - Notes de cours disponibles (distribuées)
  - Exercices pour le contrôle continu (à venir)
  - Annales des partiels-examens des années précédentes (beaucoup de corrigés)

- Feuille de présence cours-TD
- Deux cours cette semaine
- Démarrage des TD la semaine prochaine
- LDD IM+magistère : complément de cours + exercices
- Evaluation
  - Partiel (40%) + Examen final (50%)
    - un exercice sous forme *QCM* (contrôle notions élémentaires)
  - CC (10%) exercices en ligne, devoir
- Des tests à compléter sur ecampus (respecter les dates)
  - Enquête rentrée sur la page d'accueil
  - Tests (calcul propositionnel) dans la section *Maîtriser le langage logique*

- Uli Fahrenberg
  - responsable cours, chargé TD groupe 6
  - nouveau prof. à l'Université Paris-Saclay, anciennement à l'**EPITA**
  - aussi responsable L3 MAG
  - `uli@lmf.cnrs.fr`
- Christine Paulin
  - chargée TD groupe 5, ancienne responsable cours
- Aquilina Al Khoury
  - chargée TD groupe 4
- Jérémie Marrez
  - chargé TD groupe 3
- Adrien Durier
  - chargé TD groupe 2
- Gérald Forhan
  - chargé TD groupe 1

# Introduction au cours de logique

1 Motivations

2 Organisation du cours

3 Exemples d'usage de la logique

- Résoudre des problèmes
- Modéliser, prouver

# Jeu du Sudoku

8		4				2		9
		9				1		
1			3		2			7
	5		1		4		8	
				3				
	1		7		9		2	
5			4		3			8
		3				4		
4		6				3		1

Règles du jeu :

- les chiffres de 1 à 9 apparaissent une et une seule fois sur chaque ligne, chaque colonne et dans chaque cadran  $3 \times 3$

# Jeu du Sudoku

8		4				2		9
		9				1		
1			3		2			7
	5		1		4		8	
				3				
	1		7		9		2	
5			4		3			8
		3				4		
4		6				3		1

Règles du jeu :

- les chiffres de 1 à 9 apparaissent une et une seule fois sur chaque ligne, chaque colonne et dans chaque cadran  $3 \times 3$

Solution :

8	7	4	6	5	1	2	3	9
2	3	9	8	4	7	1	6	5
1	6	5	3	9	2	8	4	7
6	5	7	1	2	4	9	8	3
9	4	2	5	3	8	7	1	6
3	1	8	7	6	9	5	2	4
5	2	1	4	7	3	6	9	8
7	8	3	9	1	6	4	5	2
4	9	6	2	8	5	3	7	1



# Modélisation propositionnelle

- variables à valeur vrai/faux
- position  $p = (i, j)$  sur la ligne  $i$  et la colonne  $j$
- variable propositionnelle  $x_p^k$  : vraie si le chiffre  $k$  est à la position  $p$
- au total  $9 \times 9 \times 9 = 729$  variables, soit  $2^{729} \simeq 10^{219}$  possibilités
- exemples de règles du jeu
  - à généraliser pour chacune des 81 positions

$$x_{1,1}^1 \vee x_{1,1}^2 \vee x_{1,1}^3 \vee x_{1,1}^4 \vee x_{1,1}^5 \vee x_{1,1}^6 \vee x_{1,1}^7 \vee x_{1,1}^8 \vee x_{1,1}^9$$

- à généraliser pour chacune des 81 positions et chacune des 9 valeurs possibles

$$x_{1,1}^1 \Rightarrow (\neg x_{1,2}^1 \wedge \neg x_{1,3}^1 \wedge \neg x_{1,4}^1 \wedge \neg x_{1,5}^1 \wedge \neg x_{1,6}^1 \wedge \neg x_{1,7}^1 \wedge \neg x_{1,8}^1 \wedge \neg x_{1,9}^1)$$

- ...

- grille initiale : force certaines variables à vrai

$$x_{1,1}^8 \quad x_{1,3}^4 \quad x_{1,7}^2 \dots$$

# Exemple, complet (?)

8		4				2		9
		9				1		
1			3		2			7
	5		1		4		8	
				3				
	1		7		9		2	
5			4		3			8
		3				4		
4		6				3		1

$$(x_{1,1}^8 \wedge x_{1,3}^4 \wedge \dots \wedge x_{2,3}^9 \wedge \dots \wedge x_{9,9}^1) \wedge$$

$$(x_{1,1}^1 \Rightarrow (\neg x_{1,2}^1 \wedge \neg x_{1,3}^1 \wedge \neg x_{1,4}^1 \wedge \neg x_{1,5}^1 \wedge \neg x_{1,6}^1 \wedge \neg x_{1,7}^1 \wedge \neg x_{1,8}^1 \wedge \neg x_{1,9}^1)) \wedge$$

$$(x_{1,1}^2 \Rightarrow (\neg x_{1,2}^2 \wedge \neg x_{1,3}^2 \wedge \neg x_{1,4}^2 \wedge \neg x_{1,5}^2 \wedge \neg x_{1,6}^2 \wedge \neg x_{1,7}^2 \wedge \neg x_{1,8}^2 \wedge \neg x_{1,9}^2)) \wedge \dots$$

$$(x_{9,9}^9 \Rightarrow (\neg x_{9,1}^9 \wedge \neg x_{9,2}^9 \wedge \neg x_{9,3}^9 \wedge \neg x_{9,4}^9 \wedge \neg x_{9,5}^9 \wedge \neg x_{9,6}^9 \wedge \neg x_{9,7}^9 \wedge \neg x_{9,8}^9)) \wedge$$

$$(x_{1,1}^1 \Rightarrow (\neg x_{2,1}^1 \wedge \neg x_{3,1}^1 \wedge \neg x_{4,1}^1 \wedge \neg x_{5,1}^1 \wedge \neg x_{6,1}^1 \wedge \neg x_{7,1}^1 \wedge \neg x_{8,1}^1 \wedge \neg x_{9,1}^1)) \wedge \dots$$

$$(x_{1,1}^1 \Rightarrow (\neg x_{1,2}^1 \wedge \neg x_{1,3}^1 \wedge \neg x_{2,1}^1 \wedge \neg x_{2,2}^1 \wedge \neg x_{2,3}^1 \wedge \neg x_{3,1}^1 \wedge \neg x_{3,2}^1 \wedge \neg x_{3,3}^1)) \wedge \dots$$

$$(x_{1,1}^1 \vee x_{1,1}^2 \vee x_{1,1}^3 \vee x_{1,1}^4 \vee x_{1,1}^5 \vee x_{1,1}^6 \vee x_{1,1}^7 \vee x_{1,1}^8 \vee x_{1,1}^9) \wedge \dots$$

$$(x_{9,9}^1 \vee x_{9,9}^2 \vee x_{9,9}^3 \vee x_{9,9}^4 \vee x_{9,9}^5 \vee x_{9,9}^6 \vee x_{9,9}^7 \vee x_{9,9}^8 \vee x_{9,9}^9)$$

# Trouver une solution propositionnelle

- les formules peuvent être “simplifiées”

- Ensemble de **clauses** (disjonctions)

$$\neg x_{1,1}^1 \vee \neg x_{1,2}^1, \neg x_{1,1}^1 \vee \neg x_{1,3}^1, \neg x_{1,1}^1 \vee \neg x_{1,4}^1, \dots$$

- au total : 10287 clauses
- propager les variables résolues pour simplifier les clauses et résoudre plus de variables
- si  $x_{1,1}^8$  est vraie, alors
  - $\neg x_{1,1}^8 \vee \neg x_{1,2}^8$  devient  $\neg x_{1,2}^8$  qui sera propagé
  - $x_{1,2}^1 \vee x_{1,2}^2 \vee x_{1,2}^3 \vee x_{1,2}^4 \vee x_{1,2}^5 \vee x_{1,2}^6 \vee x_{1,2}^7 \vee x_{1,2}^8 \vee x_{1,2}^9$  devient  $x_{1,2}^1 \vee x_{1,2}^2 \vee x_{1,2}^3 \vee x_{1,2}^4 \vee x_{1,2}^5 \vee x_{1,2}^6 \vee x_{1,2}^7 \vee x_{1,2}^9$
  - $\neg x_{1,2}^8 \vee \neg x_{1,3}^8$  disparaît car toujours vrai
- Si toutes les clauses restantes ont au moins 2 variables alors on explore les deux possibilités pour une variable : vraie ou fausse
- Si on tombe sur une contradiction (clause fausse = règle du jeu non respectée), alors on revient en arrière pour explorer une autre branche.

- Modélisation : programme qui engendre les clauses du Sudoku
- Recherche de solution : procédure DPLL<sup>1</sup> qui cherche des valeurs de variables qui rendent vraies un ensemble de clauses (SAT : satisfiabilité)
- Mise en oeuvre en Ocaml :
  - 170 lignes pour les clauses et la procédure SAT (une solution ou toutes les solutions, sans optimisation)
  - 150 lignes pour la modélisation du Sudoku générique en la dimension

# Introduction au cours de logique

1 Motivations

2 Organisation du cours

3 Exemples d'usage de la logique

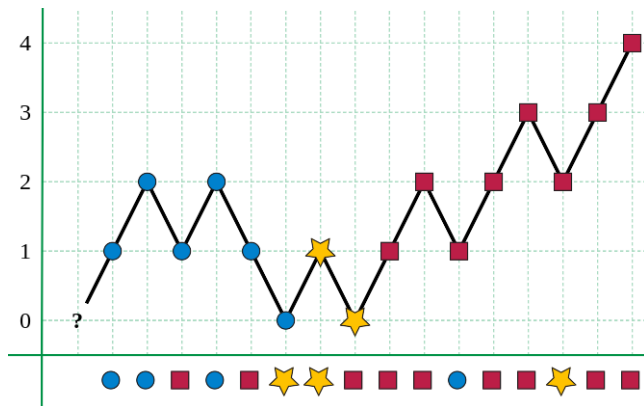
- Résoudre des problèmes
- **Modéliser, prouver**

- une **modélisation propositionnelle** se résout par calcul mais est peu naturelle et ne couvre que des propriétés *finies*
- la **logique des prédicats** permet de raisonner sur des ensembles potentiellement infinis d'objets, de manière plus concise

Preuve de programmes : Algorithme de vote majoritaire de Boyer-Moore pour chercher une valeur *m* ayant possiblement la majorité absolue dans un tableau *t* avec un seul compteur *c*

```
let majority t =  
  let rec fmaj i c m =  
    if i = Array.length t then m  
    else let x = t.(i) in  
      if c = 0 || m = x then fmaj (i+1) (c+1) x  
      else fmaj (i+1) (c-1) m  
  in fmaj 0 0 t.(0)
```

# Principe de l'algorithme



- examen séquentiel des valeurs ( $i$  de 0 à  $\text{length } t - 1$ )
- $m$  est le candidat majoritaire potentiel (aucun autre n'a la majorité)
- deux valeurs différentes  $m \neq x$  s'annulent mutuellement
- il y a  $c$  occurrences de  $m$  qui n'ont pas été annulées

- Propriété attendue :
  - aucune autre valeur que le résultat a la majorité absolue dans  $t$
  - $x \neq m$  apparaît au plus  $(\text{length } t)/2$  fois dans  $t$
- Invariant à l'étape  $i, c, m$  :
  - $x \neq m$  apparaît au plus  $(i - c)/2$  fois parmi les  $i$  premiers éléments de  $t$
  - $m$  apparaît au plus  $\frac{i-c}{2} + c$  fois parmi les  $i$  premiers éléments de  $t$
- Terminaison :  $(\text{length } t) - i$  décroît strictement en restant positif
- Théorie utilisée :
  - arithmétique (linéaire)
  - tableau (en lecture)
  - spécifique : nombre d'apparitions d'une valeur dans un segment de tableau



# Preuve de programme en Why3 : théorie logique

```
use int.Int use array.Array
```

```
(* nombre d'occurrences de x dans les i premiers éléments de t *)
```

```
function nbc (x:int) (t:array int) (i : int) : int
```

```
axiom nbc0 : forall x t. nbc x t 0 = 0
```

```
axiom nbceq : forall x t i.
```

```
    0<=i<length t -> t[i]=x -> nbc x t (i+1) = 1 + nbc x t i
```

```
axiom nbcdif : forall x t i.
```

```
    0<=i<length t -> t[i]<>x -> nbc x t (i+1) = nbc x t i
```

```
function nb (x:int) (t:array int) : int = nbc x t (length t)
```

```
(* majorité absolue *)
```

```
predicate maj (m :int) (t:array int) = length t < 2 * nb m t
```

```
(* invariant de l'algorithme *)
```

```
predicate inv (t : array int) (i : int) (c : int) (m : int)
```

```
    = 0<=c /\ 0<=i<length t
```

```
    /\ 2 * nbc m t i <= i+c /\ forall x. x<>m -> 2 * nbc x t i <=
```

# Preuve de programme en Why3 : programme annoté

*Logique de Hoare* (voir cours de GLA)

```
let majority (t : array int) : int
requires {length t <> 0}
ensures {forall x. x <> result -> not (maj x t)}
=
let rec fmaj (i : int) (c : int) (m : int) : int
  requires {inv t i c m}
  ensures {forall x. x <> result -> not (maj x t)}
  variant {length t - i}
  =
    if i = length t then m
    else let x = t[i] in
      if c = 0 || m = x then fmaj (i+1) (c+1) x
      else fmaj (i+1) (c-1) m
in fmaj 0 0 t[0]
```

Preuve automatique en utilisant alt-ergo (*SMT-solver*)



*That's all Folks!*

# 1—Maîtriser le langage logique

- 1 Définition du langage
- 2 Structure des formules
- 3 Formules vraies
- 4 Théorie et modélisation
- 5 Définition récursive sur les formules

# 1–Maîtriser le langage logique

- 1 Définition du langage
  - Objets
  - Formules
  - Traduire des énoncés
- 2 Structure des formules
- 3 Formules vraies
- 4 Théorie et modélisation
- 5 Définition récursive sur les formules

- On utilise un langage *formel* pour écrire les énoncés logiques
- Éviter les *ambiguïtés* du langage naturel et les notations imprécises.
  - Je peux t'offrir de l'eau *ou* du vin/Je peux t'offrir de l'eau *et* du vin
  - Le menu propose fromage *ou* dessert/Le menu propose fromage *et* dessert
  - S'il ne pleut pas, je sors jouer/S'il pleut, je ne sors pas jouer
- Moins de *redondance* que le langage naturel : plus simple à étudier, plus simple à implémenter
- Un énoncé écrit dans le langage de la logique sera appelé *formule*

# Introduction au cours de logique

- 1 Définition du langage
  - Objets
  - Formules
  - Traduire des énoncés
- 2 Structure des formules
- 3 Formules vraies
- 4 Théorie et modélisation
- 5 Définition récursive sur les formules

- Les énoncés parlent d'*objets*
  - entiers, individus, livres, ensembles, fonctions ...
- Dans le langage de la logique, un objet est représenté par un **terme**
- Un terme peut être une constante  $0, 1, \text{Martin}, \emptyset, \mathbb{N} \dots$ ,
- Un terme peut être construit à partir d'*opérations*  $+, \times, \cup, \dots$   
 $3 + 5, \mathbb{N} \times \mathbb{N}, \text{le père de Martin}, \dots$   
Une opération est un *symbole* associé à une **arité**, entier naturel qui représente le nombre d'arguments attendus.
- Dans une formule, on utilise des **variables** : symboles qui représentent des objets indéterminés  
 $x + 1, \text{un étudiant de licence}, \dots$



## Definition (Signature, arité)

Une signature est un ensemble de symboles  $\mathcal{F}$  chacun associé à un entier naturel appelé **arité**.

Un symbole d'arité 0 est appelé **constante**, un symbole d'arité 1 est dit **unaire**, un symbole d'arité 2 est dit **binaire**.

## Definition (Terme)

Etant donné une signature  $\mathcal{F}$  et un ensemble  $\mathcal{X}$  de variables, un terme  $t$  est soit une variable, soit formé d'un symbole  $f$  d'arité  $n$  et d'une suite ordonnée de  $n$  termes  $t_1, \dots, t_n$ .

- $f$  est le **symbole de tête**,  $t_1, \dots, t_n$  sont les **sous-termes** directs du terme  $t$ .
- $\mathcal{T}(\mathcal{F}, \mathcal{X})$  est l'ensemble des termes sur la signature  $\mathcal{F}$  et l'ensemble des variables  $\mathcal{X}$ .
- $\mathcal{T}(\mathcal{F})$  est l'ensemble des termes qui ne contiennent pas de variable, appelés aussi **termes clos**.

# Exemples de signature

- Entiers naturels :
  - constantes 0 et 1
  - opérations binaires : addition + et la multiplication  $\times$
  - notation **infixe** :  $(t + u)$ ,  $(t \times u)$
  - termes :  $x + 1$ ,  $(0 + 1)$ ,  $(1 + 1) \times (1 + 1 + 1)$ ,...
- Mots binaires de longueur arbitraire
  - constante  $\epsilon$  pour représenter le mot vide
  - deux fonctions unaires  $c_0$  et  $c_1$  pour représenter l'ajout d'un 0 ou d'un 1 en tête du mot
  - le mot 1011 est représenté par le terme  $c_1(c_0(c_1(c_1(\epsilon))))$ .
  - autres opérations possibles : concaténation de deux mots, décalage vers la gauche ou la droite

# Introduction au cours de logique

- 1 Définition du langage
  - Objets
  - **Formules**
  - Traduire des énoncés
- 2 Structure des formules
- 3 Formules vraies
- 4 Théorie et modélisation
- 5 Définition récursive sur les formules

# Formules atomiques

Les **formules atomiques** ne se décomposent pas en formules plus simples.

- $\top$  (top) : la propriété toujours vraie, tautologie ( $0 = 0$ )
- $\perp$  (bottom) : la propriété toujours fausse, l'absurde ( $0 = 1$ )
- **symbole de prédicat** associé à un ou plusieurs termes (suivant l'**arité**)  
propriétés de base des objets : ce qui ne s'explique pas en terme logique mais qui *s'observe*
  - Exemples de symboles
    - arité 0 (**variable propositionnelle**) : "signal-passage-à-niveau-ouvert",
    - arité 1 (**symbole unaire**, ensemble) : "yeux-bleus", "pair"
    - arité 2 (**symbole binaire**) : l'égalité  $=$ , la comparaison  $\leq$ , l'appartenance  $\in$ ,
    - arité quelconque : table d'une base de données
  - Exemples de formules atomiques
    - $0 = 1$ ,  $2 + 2 = 4$ ,  $x = x$ ,
    - $x + 1 \leq x$
    - *Martin a les yeux bleus* : yeux-bleus(Martin)
    - Etudiant(Durand, Bob, 347890, 01/01/1990)

# Exemples de signature : systèmes d'information

- modélisation logique des entités/ensembles et des tables/relation par des symboles de prédicat.
- Trajets de bus :
  - identifiants de ligne (numéro) des arrêts et des horaires : trois prédicats unaires `ligne`, `arrêt` et `horaire` pour séparer les objets de la logique suivant leur catégorie.  
`ligne(91-06), arrêt(Massy), arrêt(Saclay), horaire(06h00)...`
  - table qui tient à jour les rotations de bus avec le numéro de ligne, l'arrêt de départ, celui d'arrivée ainsi que l'horaire de départ : predicat `trajet` d'arité 4.  
`trajet(91-06, Massy, Saclay, 06h00)`

- Le choix des symboles de prédicat dépend de la modélisation
  - primitive de base versus notions dérivées via une formule
  - analogie avec les variables d'un problème mathématique ou physique
  - chaque symbole peut s'interpréter librement
  - on peut aussi parfois choisir des symboles de fonction au lieu de symboles de prédicat (notions différentes en logique).
- Exemples
  - *tables* primitives dans une base de données, versus résultat d'une requête
  - *interface* d'une bibliothèque dont on ne connaît pas l'implémentation
  - *axiomatisation* d'une *théorie*
    - entiers :  $x \leq y$  versus  $\exists n, y = x + n$
    - ordres :  $x = y$  versus  $(x \leq y \wedge y \leq x)$
    - hommes et femmes, versus  $H(x) \stackrel{\text{def}}{=} \neg F(x)$
    - *est-mère*( $x, y$ ) versus *x=mère*( $y$ )

- Un symbole est juste un nom (**syntaxe**),
- la **sémantique** lui attribue un *sens* en lui associant une relation mathématique entre les objets modélisés
- il y a parfois un sens usuel implicite (ex : ordre sur les entiers) mais d'un point de vue logique, *toutes les interprétations sont possibles*.
- les sens possibles des symboles seront restreints par l'introduction de **théories**

# Formules complexes

Les *formules complexes* (celles qui ne sont pas atomiques) se construisent à l'aide de **connecteurs** et de **quantificateurs** logiques.

Partie **propositionnelle** (connecteurs) :

- $\neg P$  la **négation** d'une formule, prononcée "**non**  $P$ "
- $P \wedge Q$  la **conjonction** de deux formules, prononcée " $P$  **et**  $Q$ "
- $P \vee Q$  la **disjonction** de deux formules, prononcée " $P$  **ou**  $Q$ "
- $P \Rightarrow Q$  l'**implication** de deux formules, prononcée " $P$  **implique**  $Q$ " ou bien "**si**  $P$  **alors**  $Q$ "

Et les **quantificateurs** du **premier ordre** :

- $\forall x, P$  la **quantification universelle**, prononcée "**pour tout**  $x$ ,  $P$ "
- $\exists x, P$  la **quantification existentielle**, prononcée "**il existe**  $x$  **tel que**  $P$ "

Une formule sans quantificateur  $\forall$  et  $\exists$  est dite **formule propositionnelle**



# Sens intuitif des connecteurs et quantificateurs

- $\neg P$  :  $P$  est faux
- $P \wedge Q$  :  $P$  et  $Q$  sont tous les deux vrais
- $P \vee Q$  : soit  $P$  soit  $Q$  est vrai (ou les deux)
- $P \Rightarrow Q$  : si  $P$  est vrai alors  $Q$  est vrai et si  $P$  est faux alors  $Q$  peut être vrai ou faux ( $P$  est faux ou bien  $Q$  est vrai)
- $\forall x, P$  :  $P$  est vrai pour toutes les *valeurs* possibles de  $x$
- $\exists x, P$  : il existe au moins une *valeur* de  $x$  pour laquelle  $P$  est vrai

- expressions pour représenter des conditions booléennes
- opérations pour la négation, la conjonction, la disjonction
- pas d'implication : on trouve à la place une conditionnelle

si *a* alors *b* sinon *c*

- *b* et *c* peuvent être des booléens ou représenter d'autres types d'objet
- les quantificateurs ne correspondent pas à des constructions de programme car en général (cas infini) ils ne sont pas *calculables*.

# Exercice

On suppose que  $a$ ,  $b$  et  $c$  sont des formules logiques.

Représenter la phrase **si  $a$  alors  $b$  sinon  $c$**  comme une formule logique n'utilisant que les connecteurs logiques

- faire la table de vérité
- donner une première représentation sans utiliser d'implication
- donner une seconde représentation qui contienne la formule  $a \Rightarrow b$

# Exemples de formules complexes

- tiers exclu :  $A \vee \neg A$
- modus-ponens :  $((A \Rightarrow B) \wedge A) \Rightarrow B$
- loi de Peirce :  $((A \Rightarrow B) \Rightarrow A) \Rightarrow A, ((\neg A) \Rightarrow A) \Rightarrow A$
- $\forall x, \text{yeux-bleus}(x)$
- $\exists x, \text{yeux-bleus}(x)$
- $\exists x, (\text{yeux-bleus}(x) \Rightarrow \forall y, \text{yeux-bleus}(y))$
- formule *paramétrée* : l'entier  $x$  est impair :  $\exists y, x = 2 \times y + 1$

# Différentes catégories syntaxiques : termes

- variables (objets) :  $\mathcal{X}$
- symboles de fonctions
  - constantes d'arité 0 :  $\mathcal{C}$
  - fonctions d'arité au moins 1 :  $\mathcal{F}$
- **termes** (ou objets)

$\text{term} \coloneqq \mathcal{X} \mid \mathcal{C} \mid \mathcal{F}(\text{list-terms})$

$\text{list-terms} \coloneqq \text{term} \mid \text{list-terms}, \text{term}$

# Différentes catégories syntaxiques : formules

- symboles de prédicats
  - d'arité 0 :  $\mathcal{V}$
  - d'arité au moins 1 :  $\mathcal{P}$
- **formules** logiques :

$$\begin{aligned} \text{form} ::= & \top \mid \perp \mid \mathcal{V} \mid \mathcal{P}(\text{list-terms}) \\ & \mid \neg \text{form} \mid (\text{form} \wedge \text{form}) \mid (\text{form} \vee \text{form}) \mid (\text{form} \Rightarrow \text{form}) \\ & \mid (\forall \mathcal{X}, \text{form}) \mid (\exists \mathcal{X}, \text{form}) \end{aligned}$$

# Notations infixes

- Notation standard :
  - Fonctions/Prédicats :  $\text{Symbole}(t_1, \dots, t_n)$
- Quelques notations usuelles **infixes** pour des symboles **binaires**  $f(t, u)$ 
  - $t \circ u$
  - exemples :  $t + u, t \times u, t = u, t \leq u \dots$
- Extension de la grammaire

$$\begin{aligned}\text{term} &\vdash (\text{term } \mathcal{F}_l \text{ term}) \\ \text{form} &\vdash (\text{term } \mathcal{P}_l \text{ term})\end{aligned}$$

les parenthèses évitent les ambiguïtés

# Notations, règles de parenthésage

- $A \Leftrightarrow B$  est la même chose que  $(A \Rightarrow B) \wedge (B \Rightarrow A)$
- plusieurs variables dans un quantificateur, par exemple :  $\forall x y, P$   
représente la formule  $\forall x, \forall y, P$
- attention  $\forall x \in A, P$  ne fait pas partie du langage ni  $\exists! x, P$
- l'usage de notations infixes rend nécessaire l'ajout de parenthèses dans la syntaxe des formules
  - comment interpréter  $P \wedge Q \vee R$ ?
    - 1  $(P \wedge Q) \vee R$
    - 2  $P \wedge (Q \vee R)$



- On appelle **logique du premier ordre** (ou **calcul des prédicats**) le langage logique défini par une signature (symboles de fonctions et de prédicats) et qui n'utilise que les connecteurs logiques et les quantificateurs sur les variables de termes tels que définis précédemment.
- Le **calcul propositionnel** (aussi appelé **logique propositionnelle**) est un cas particulier dans lequel la signature est réduite à des symboles de prédicats d'arité 0 (appelées **variables propositionnelles**) et dans lequel on n'utilise pas de quantificateurs.
- Il existe d'autres logiques (logique d'ordre supérieur, logiques temporelles, logiques modales. . .)

# Introduction au cours de logique

- 1 Définition du langage
  - Objets
  - Formules
  - Traduire des énoncés
- 2 Structure des formules
- 3 Formules vraies
- 4 Théorie et modélisation
- 5 Définition récursive sur les formules

## Langage

- $\text{ami}(x, y)$  :  $x$  est l'ami de  $y$
- $\text{joue}(x, y)$  :  $x$  joue avec  $y$
- constante  $\text{self}$  qui représente l'individu qui s'exprime.

Que signifient les formules suivantes en langage courant ?

- 1  $\forall x, (\text{ami}(\text{self}, x) \Rightarrow \neg \text{joue}(\text{self}, x))$
- 2  $\forall x, \text{joue}(\text{self}, x)$
- 3  $\forall x, \exists y, \text{joue}(x, y)$
- 4  $\forall y, \exists x, \text{joue}(y, x)$
- 5  $\exists y, \forall x, \text{joue}(x, y)$

# Exemple de traduction

- le langage comporte un symbole de fonction  $+$  binaire noté de manière infixe qui représente l'opération de sommation
- `pair`( $x$ ) représente la propriété " $x$  est un entier pair"
- Exprimer par une formule les propriétés :
  - *"la somme de deux entiers pairs est un entier pair"*
  - *"la somme de deux entiers impairs est un entier pair"*

- Connaître les notions de **signature** et **arité**.
- Distinguer les **termes** et les **formules**, et parmi les formules celles qui sont **atomiques**.
- Reconnaître les **termes** et les **formules** *syntactiquement* bien formés.
- Comprendre le sens intuitif des connecteurs propositionnels et quantificateurs logiques.
- Lire une formule logique et traduire une propriété exprimée en langue naturelle en utilisant le langage de la logique.



*That's all Folks!*

## Definition (Signature, arité)

Une signature est un ensemble de symboles  $\mathcal{F}$  chacun associé à un entier naturel appelé **arité**.

Un symbole d'arité 0 est appelé **constante**, un symbole d'arité 1 est dit **unaire**, un symbole d'arité 2 est dit **binaire**.

## Definition (Terme)

Etant donné une signature  $\mathcal{F}$  et un ensemble  $\mathcal{X}$  de variables, un terme  $t$  est soit une variable, soit formé d'un symbole  $f$  d'arité  $n$  et d'une suite ordonnée de  $n$  termes  $t_1, \dots, t_n$ .

- $f$  est le **symbole de tête**,  $t_1, \dots, t_n$  sont les **sous-termes** directs du terme  $t$ .
- $\mathcal{T}(\mathcal{F}, \mathcal{X})$  est l'ensemble des termes sur la signature  $\mathcal{F}$  et l'ensemble des variables  $\mathcal{X}$ .
- $\mathcal{T}(\mathcal{F})$  est l'ensemble des termes qui ne contiennent pas de variable, appelés aussi **termes clos**.

# La dernière fois

Les **formules atomiques** ne se décomposent pas en formules plus simples.

- $\top$  (top) : la propriété toujours vraie, tautologie ( $0 = 0$ )
- $\perp$  (bottom) : la propriété toujours fausse, l'absurde ( $0 = 1$ )
- **symbole de prédicat** associé à un ou plusieurs termes (suivant l'**arité**)  
propriétés de base des objets : ce qui ne s'explique pas en terme logique mais qui *s'observe*
  - Exemples de symboles
    - arité 0 (**variable propositionnelle**) : "signal-passage-à-niveau-ouvert",
    - arité 1 (**symbole unaire**, ensemble) : "yeux-bleus", "pair"
    - arité 2 (**symbole binaire**) : l'égalité  $=$ , la comparaison  $\leq$ , l'appartenance  $\in$ ,
    - arité quelconque : table d'une base de données
  - Exemples de formules atomiques
    - $0 = 1, 2 + 2 = 4, x = x,$
    - $x + 1 \leq x$
    - *Martin a les yeux bleus* : yeux-bleus(Martin)
    - Etudiant(Durand, Bob, 347890, 01/01/1990)



# La dernière fois

Les *formules complexes* (celles qui ne sont pas atomiques) se construisent à l'aide de **connecteurs** et de **quantificateurs** logiques.

Partie **propositionnelle** (connecteurs) :

- $\neg P$  la **négation** d'une formule, prononcée "**non**  $P$ "
- $P \wedge Q$  la **conjonction** de deux formules, prononcée " $P$  **et**  $Q$ "
- $P \vee Q$  la **disjonction** de deux formules, prononcée " $P$  **ou**  $Q$ "
- $P \Rightarrow Q$  l'**implication** de deux formules, prononcée " $P$  **implique**  $Q$ " ou bien "**si**  $P$  **alors**  $Q$ "

Et les **quantificateurs** du **premier ordre** :

- $\forall x, P$  la **quantification universelle**, prononcée "**pour tout**  $x$ ,  $P$ "
- $\exists x, P$  la **quantification existentielle**, prononcée "**il existe**  $x$  **tel que**  $P$ "

Une formule sans quantificateur  $\forall$  et  $\exists$  est dite **formule propositionnelle**

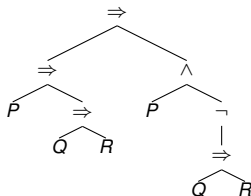
- On appelle **logique du premier ordre** (ou **calcul des prédicats**) le langage logique défini par une signature (symboles de fonctions et de prédicats) et qui n'utilise que les connecteurs logiques et les quantificateurs sur les variables de termes tels que définis précédemment.
- Le **calcul propositionnel** (aussi appelé **logique propositionnelle**) est un cas particulier dans lequel la signature est réduite à des symboles de prédicats d'arité 0 (appelées **variables propositionnelles**) et dans lequel on n'utilise pas de quantificateurs.
- Il existe d'autres logiques (logique d'ordre supérieur, logiques temporelles, logiques modales. . .)

# 1–Maîtriser le langage logique

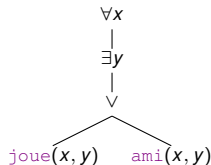
- 1 Définition du langage
- 2 **Structure des formules**
  - Représentation par des arbres
  - Règles de parenthésage
  - Variables libres et liées
- 3 Formules vraies
- 4 Théorie et modélisation
- 5 Définition récursive sur les formules

# Structure des formules

- Une formule se représente comme un arbre
  - les nœuds internes sont les connecteurs et les quantificateurs  $\forall x$ , et  $\exists x$
  - les feuilles sont les formules atomiques
- $(P \Rightarrow (Q \Rightarrow R)) \Rightarrow (P \wedge \neg(Q \Rightarrow R))$



- $\forall x, \exists y, (\text{joue}(x, y) \vee \text{ami}(x, y))$



représenter sous forme d'arbre les formules

- $(P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P)$
- $\forall x, ((\forall y, \neg \text{ami}(x, y)) \Rightarrow \text{joue}(x, x))$

# Règles de précedence

- comment interpréter  $P \wedge Q \vee R$ ?  $\forall x, P(x) \Rightarrow Q(x)$
- règles de **précedence** :
  - La précedence de  $\neg$  est la plus **forte**, vient ensuite la conjonction  $\wedge$  puis la disjonction  $\vee$  et finalement l'implication  $\Rightarrow$ .
  - Les connecteurs  $\wedge, \vee$  et  $\Rightarrow$  associent à **droite**



$A \Rightarrow B \Rightarrow C$  se parenthèse  $A \Rightarrow (B \Rightarrow C)$

- Les quantificateurs  $\forall$  et  $\exists$  ont une précedence plus **faible** que les autres connecteurs.



$\forall x, P \Rightarrow Q$  se parenthèse  $\forall x, (P \Rightarrow Q)$

# Exercice : Règles de précédence

- 1  $P \wedge R \wedge \neg Q \Rightarrow P \vee Q \wedge R$
- 2  $(P \Rightarrow Q \Rightarrow \forall x, R) \Rightarrow P \wedge \exists x, Q \Rightarrow R$
- 3  $(P \Rightarrow Q) \Rightarrow \neg \forall x, Q \Rightarrow \neg P$

# Grammaire avec ambiguïté

$$\text{term} ::= \mathcal{X} \mid \mathcal{C} \mid \mathcal{F}(\text{list-terms}) \mid \text{term } \mathcal{F}_I \text{ term} \mid (\text{term})$$
$$\text{list-terms} ::= \text{term} \mid \text{list-terms}, \text{term}$$
$$\begin{aligned} \text{form} ::= & \top \mid \perp \mid \mathcal{V} \mid \mathcal{P}(\text{list-terms}) \mid \text{term } \mathcal{P}_I \text{ term} \mid (\text{form}) \\ & \mid \neg \text{form} \mid \text{form} \wedge \text{form} \mid \text{form} \vee \text{form} \mid \text{form} \Rightarrow \text{form} \\ & \mid \forall \mathcal{X}, \text{form} \mid \exists \mathcal{X}, \text{form} \end{aligned}$$

- Les règles de précedence permettent de lever les ambiguïtés
- Les parenthèses peuvent être ajoutées librement pour faciliter la compréhension.



- les quantificateurs sont associés à une variable  $\forall x, P$  et  $\exists x, P$ , on dit que la variables est **liée**.
- une variable liée est dite **muette** : son nom peut être changé, sans changer le sens de la formule

$$\forall x, \exists y, x < y \qquad \forall t, \exists u, t < u$$

attention au problème de **capture** :  $\forall y, \exists y, y < y$

- en langage courant, souvent on ne mentionne pas le nom
  - tous les chats sont gris :  $\forall x, \text{chat}(x) \Rightarrow \text{gris}(x)$
  - il existe des chats qui ne sont pas gris :  $\exists x, \text{chat}(x) \wedge \neg \text{gris}(x)$

# Variables libres

- une occurrence  $x$  est **libre** ssi pas sous un quantificateur  $\forall x$ , ou  $\exists x$ ,
- les variables libres sont les *paramètres* de la formule :
  - exemple : “ $x$  est pair” :  $\exists y, x = 2 \times y$
  - la formule est vraie ou fausse en fonction de la valeur de la variable
  - analogie avec une procédure, méthode en programmation (donner des valeurs aux paramètres pour exécuter)
- une variable libre peut être remplacée par un terme plus complexe :  
**substitution**
  - “4 est pair” :  $\exists y, 4 = 2 \times y$
  - “ $2 \times z + 4$  est pair” :  $\exists y, 2 \times z + 4 = 2 \times y$
  - attention à la capture lorsqu’on substitue une variable par un terme :  
“ $3 \times y$  est pair” :
    - $\exists y, 3 \times y = 2 \times y$  (substitution incorrecte)
    - $\exists y', 3 \times y = 2 \times y'$  (substitution correcte après renommage)

- une même variable peut apparaître libre et liée dans une formule :

$$0 < x \times y \vee (\exists y, x < y) \wedge (\exists y, y + y < x)$$

- un terme qui ne contient pas de variables est appelé **terme clos**
- une formule qui ne contient pas de *variable libre* est appelée **formule close**
- Exercice :
  - donner les variables libres et liées de la formule

$$\forall b, b > 0 \Rightarrow \exists q, \exists r, a = b \times q + r \wedge r < b$$

- cette formule est-elle close ?

- une même variable peut apparaître libre et liée dans une formule :

$$0 < x \times \mathbf{y} \vee (\exists \mathbf{y}_1, x < \mathbf{y}_1) \wedge (\exists \mathbf{y}_2, \mathbf{y}_2 + \mathbf{y}_2 < x)$$

- un terme qui ne contient pas de variables est appelé **terme clos**
- une formule qui ne contient pas de *variable libre* est appelée **formule close**
- Exercice :
  - donner les variables libres et liées de la formule

$$\forall b, b > 0 \Rightarrow \exists q, \exists r, a = b \times q + r \wedge r < b$$

- cette formule est-elle close ?

# Formules syntaxiquement égales

- Deux formules  $P$  et  $Q$  sont **syntactiquement égales** si elles ont la même représentation sous forme d'arbre modulo le renommage des variables liées. On écrit alors  $P = Q$ .
- seul le lien entre l'utilisation de la variable et le quantificateur qui l'a introduit est important
- Il est relativement “facile” d'écrire un programme qui vérifie que deux formules sont égales
- Les variables liées compliquent la vérification et font que deux formules égales peuvent avoir des représentations différentes en machine

# Exercice : Formules syntaxiquement égales

Dire quelles formules sont égales ou différentes

①  $\forall x, \forall y, P(x, y)$

②  $\forall y, \forall x, P(x, y)$

③  $\forall y, \forall z, P(y, z)$

①  $\forall x, P(x) \Rightarrow Q(x) \Rightarrow \perp$

②  $\forall x, ((P(x) \Rightarrow Q(x)) \Rightarrow \perp)$

③  $\forall x, (P(x) \Rightarrow (Q(x) \Rightarrow \perp))$

④  $(\forall x, (P(x) \Rightarrow Q(x))) \Rightarrow \perp$

⑤  $(\forall x, P(x)) \Rightarrow (Q(x) \Rightarrow \perp)$

Comparer les représentations sous forme d'arbre !

- Connaître les règles de précédences sur les connecteurs et les quantificateurs de la logique.
- Représenter une formule sous forme d'arbre.
- Reconnaître les variables libres et les variables liées d'une formule logique.
- Connaître la définition de **terme clos** et **formule close**.

# 1—Maîtriser le langage logique

- 1 Définition du langage
- 2 Structure des formules
- 3 **Formules vraies**
  - Cas propositionnel
  - Modéliser un problème à l'aide de la logique
  - Formules avec quantificateurs
- 4 Théorie et modélisation
- 5 Définition récursive sur les formules



- **valeur de vérité**, ensemble des **booléens**  $\mathbb{B} = \{V, F\}$   
(parfois notées  $\{1, 0\}$ )
- on cherche les conditions dans lesquelles une formule est vraie ou fausse
- on parle de **sémantique** (sens de la formule) par opposition à la **syntaxe** (forme)
- plusieurs formules syntaxiquement différentes peuvent avoir le même sens
- la même formule peut avoir différents sens suivant l'interprétation des symboles de la signature
  - Je n'ai pas d'ami
  - $1 + 1 = 0$

# Formules (closes) vraies

- Les formules contiennent des symboles de fonctions et de prédicats qui correspondent à des *primitives*, de sens indéterminé
- On ne connaît la vérité d'une formule qu'*après* avoir défini l'*interprétation des symboles*
  - Si un programme utilise une *bibliothèque* :
    - il se compile en utilisant l'interface de la bibliothèque
    - il ne s'exécute qu'en présence du code d'implémentation de cette bibliothèque.
  - Une *base de données*
    - les requêtes se définissent en fonction de la structure (tables)
    - chaque "état" de la base de données correspond à une interprétation.

# Formules (closes) valides

- La valeur de vérité d'une formule (vrai ou faux) est définie par rapport à une interprétation des symboles qu'elle contient
- On ne peut pas dire *a priori* si une formule est vraie ou fausse
  - Par abus de langage, on dit parfois qu'une formule est vraie (resp. fausse) si elle est tout le temps vraie (resp. fausse)
- Une formule (indépendamment d'une interprétation) peut être
  - **valide** (**tautologie**) : vraie pour toutes les interprétations
  - **insatisfiable** : fausse pour toutes les interprétations
  - **satisfiable** : vraie pour au moins une interprétation
- analogie avec les équations

$$(x + 1)^2 = x^2 + 2x + 1 \qquad x^2 = -1 \qquad x^2 = 4$$

# Lien entre validité et satisfiabilité

- une formule valide est a fortiori satisfiable (perte d'information)
- un algorithme qui résout l'une des trois questions résout les autres.

## Proposition

*Les trois propriétés suivantes sont équivalentes*

- 1  $P$  est valide
- 2  $\neg P$  est insatisfiable
- 3  $\neg P$  n'est pas satisfiable

## Preuve:

- $P$  est valide ssi  $P$  est vrai pour toute interprétation (définition valide)  
ssi  $\neg P$  est faux pour toute interprétation (table de vérité de  $\neg$ )  
ssi  $\neg P$  est insatisfiable (définition insatisfiable)
- $Q$  est insatisfiable ssi  $Q$  est faux pour toute interprétation (définition insatisfiable)  
ssi il n'y a pas d'interprétation qui rend  $Q$  vrai (reformulation)  
ssi  $Q$  n'est pas satisfiable (définition de satisfiable)

# Modèle et valeur de vérité

- pour dire si une formule est vraie, il faut expliciter dans quelle **interprétation** on se place (on utilise parfois le terme **modèle**) : que représentent les symboles ?
- $2 \leq 4$ ,  $0 = 1$ , Martin a les yeux bleus,  
Bob X. est inscrit en L3 Info. à l'U. Paris-Saclay
- si on connaît les formules atomiques vraies alors la logique nous dit si une formule *propositionnelle* est vraie ou fausse
- table de vérité** :

$P$	$Q$	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$F$	$F$					
$F$	$V$					
$V$	$F$					
$V$	$V$					

Les formules suivantes sont-elles satisfiables ? valides ?

1  $\neg(p \wedge q) \Rightarrow \neg p \vee \neg q$

2  $\neg(p \wedge q) \Rightarrow \neg p \vee q$

3  $\neg p \wedge p$

4  $((p \Rightarrow q) \Rightarrow p) \Rightarrow p$

5  $(\neg p \Rightarrow p) \Rightarrow p$

# Modélisation propositionnelle de problèmes

- pour modéliser certains problèmes, on introduit des **variables propositionnelles** correspondant à des situations dont on cherche à déterminer si elles sont vraies ou fausses
- des formules logiques représentent les contraintes associées au problème
- on cherche à déterminer si le problème a une solution (satisfiabilité), le nombre de solutions . . .
- les outils informatiques qui réalisent ces tâches s'appellent des *SAT-solver*
- de nombreuses applications industrielles
  - vérification de hardware
  - résolution de contraintes, planification. . .

# Exercice : Enigme

Arthur, Bob et Casimir sont soupçonnés d'avoir peint en bleu le chat de la voisine. Ils font les déclarations suivantes :

- Arthur : Bob est coupable et Casimir est innocent.
  - Bob : Si Arthur est coupable, Casimir aussi.
  - Casimir : Je suis innocent mais au moins l'un des deux autres est coupable.
- 1 On pose :  $a$  = "Arthur est coupable",  $b$  = "Bob est coupable" et  $c$  = "Casimir est coupable". Avec ces notations transcrire les trois déclarations ci-dessus dans le langage de la logique propositionnelle (notées  $F_A$ ,  $F_B$  et  $F_C$ ).
  - 2 Construire la table de vérité des formules  $F_A$ ,  $F_B$  et  $F_C$ .
  - 3 En utilisant la question précédente, répondez aux questions suivantes :
    - 1 Montrer que si Casimir a menti alors Arthur aussi.
    - 2 Si Casimir a menti que peut-on dire de la déclaration de Bob ?
    - 3 En supposant que tous ont dit la vérité, qui est coupable qui est innocent ?
    - 4 En supposant que tous sont coupables, qui a dit vrai ? qui a menti ?
    - 5 Est-il possible que tous les innocents aient menti et que tous les coupables aient dit la vérité ?



# 1—Maîtriser le langage logique

- 1 Définition du langage
- 2 Structure des formules
- 3 Formules vraies**
  - Cas propositionnel
  - Modéliser un problème à l'aide de la logique
  - **Formules avec quantificateurs**
- 4 Théorie et modélisation
- 5 Définition récursive sur les formules

# Formules avec quantificateurs

- les *variables d'objets* représentent des inconnues dans un univers a priori indéterminé
$$\exists x, \text{yeux-bleus}(x) \quad \forall x y, x < y \Rightarrow (x + 1) \leq y$$
- une **interprétation** (**modèle**) va expliciter le **domaine** d'interprétation des objets (noté  $D$ ) non vide
  - Ex. : entiers signés ou non, sur 32 ou 64 bits, population dans une BD.
- on interprète les symboles de constante et les opérateurs dans ce domaine (ex. `maxint`, la division...)
- un terme  $t$  sans variable représente une valeur  $t_D$  (le résultat du calcul) dans le domaine  $t_D \in D$  (ex.  $2 + 3$ )
- un terme avec variable  $x + 3$  s'interprète comme une valeur  $t_D \in D$  en se donnant en plus un **environnement** qui définit les valeurs des variables
  - analogie avec la mémoire d'un ordinateur

# Interprétation des formules avec quantificateurs

- les symboles de prédicat `yeux-bleus`,  $\leq$ ... peuvent aussi s'interpréter de différentes manières. L'interprétation leur associe une **relation** entre les objets
  - relation unaire (`yeux-bleus`) : ensemble d'objets
  - relation binaire ( $\leq$ ) : graphe orienté
  - relations d'arité supérieure : tables
- analogie avec les langages de programmation
  - la **syntaxe** : les règles pour écrire un programme syntaxiquement correct
  - la **sémantique** : les règles qui expliquent quel sera le résultat de l'exécution d'un programme (dans un certain contexte)
  - plusieurs sens possibles pour le même programme
  - plusieurs compilateurs peuvent donner des résultats différents

# Vérité d'une formule quantifiée

- A quelle condition  $\forall x, P(x)$  est-il vrai ?
- on ne peut parler de la valeur de vérité d'une formule que dans une interprétation qui définit le domaine ( $D$ ) des objets, et l'interprétation des symboles (constantes, fonctions, prédicats) ainsi que la valeur des variables **libres** de la formule (**l'environnement**)
- $\forall x, P$  est vraie si pour tout objet  $d \in D$ , la formule  $P$  est vraie dans l'environnement dans lequel  $x$  a la valeur  $d$ .
- $\exists x, P$  est vraie s'il existe un objet  $d \in D$  tel que la formule  $P$  est vraie dans l'environnement dans lequel  $x$  a la valeur  $d$ .

# Interprétation des symboles

- Domaine  $D$  des objets : ensemble non vide
- A chaque constante on associe un élément du domaine
- A chaque symbole de fonction on associe une fonction sur le domaine
- Une formule atomique  $P(t_1, \dots, t_n)$  représente une vérité qui dépend de la valeur des arguments  $t_1, \dots, t_n$ .  
On interprète  $P$  par une relation  $n$ -aire sur l'ensemble  $D$  (à quelle condition sur les entrées la formule est vraie).
- Exemples : yeux-bleus, ami, ...

- Validité : vrai pour tout domaine, toute interprétation des symboles, toutes valeurs des variables libres
- Satisfiabilité : vrai pour au moins un domaine, une interprétation des symboles, une valeur des variables libres
- Insatisfiabilité : faux pour tout domaine, toute interprétation des symboles, toutes valeurs des variables libres

- Les formules suivantes sont-elles valides (vraies pour tout domaine et interprétation du prédicat  $P$ ) ? Sont-elles satisfiables ?

- 1  $(\forall x, P(x)) \Rightarrow (\exists x, P(x))$
- 2  $(\forall x, P(x)) \wedge (\exists x, \neg P(x))$
- 3  $\neg(\forall x, P(x)) \Rightarrow \exists x, \neg P(x)$
- 4  $(\exists x, P(x)) \Rightarrow \forall x, P(x)$
- 5  $(\forall x, P(x)) \Rightarrow \forall x, \neg P(x)$

## Remarque

- Quand il y a une infinité d'objets, on ne peut plus faire des tables de vérité !

## Trouver l'erreur

- Tout ce qui est rare est cher.
  - Un cheval bon marché est rare,
  - donc un cheval bon marché est cher
- 1 introduire des prédicats pour modéliser la situation
  - 2 exprimer les propriétés précédentes comme des formules logiques



- Tables de vérité des formules propositionnelles
- Définir une interprétation simple
- Evaluer la vérité d'une formule dans une interprétation
- Liens entre validité et satisfiabilité

Ces notions seront approfondies dans le chapitre 2

# 1—Maîtriser le langage logique

- 1 Définition du langage
- 2 Structure des formules
- 3 Formules vraies
- 4 Théorie et modélisation**
  - Exemple de modélisation propositionnelle avancée
- 5 Définition récursive sur les formules

# Pourquoi des théories ?

- un symbole peut être interprété de plusieurs manières différentes
  - `ami`, `joue`
  - `pair`
- Formule valide : vraie dans toutes les interprétations possibles
  - des vérités “*absolues*”
  - rien n’empêche une interprétation dans laquelle `ami(t, u)` est vrai mais `ami(u, t)` est faux ou bien `pair(1)` est vrai.
- Pour limiter les interprétations, on ajoute des **axiomes** (formule sans variables libres)

$$0 \neq 1 \qquad \forall x y, \text{ami}(x, y) \Rightarrow \text{ami}(y, x)$$

- Les axiomes sont des *vérités* qui n’ont pas besoin d’être démontrées.
  - on se limite aux interprétations qui rendent vrais les axiomes.
  - dans les démonstrations, les axiomes sont traités comme des hypothèses supplémentaires

## Definition (Théorie)

Une théorie est définie par un ensemble de symboles de fonctions et de prédicats (la **signature** de la théorie) et un ensemble de formules closes (sans variables libres) construites sur ce langage, appelés les **axiomes** de la théorie.

- Une théorie peut être définie par un ensemble fini ou infini d'axiomes.
- Un **modèle d'une théorie** est donné par une interprétation de la signature dans laquelle tous les axiomes ont pour valeur vraie.
- Une formule est **valide dans une théorie** si elle vraie dans tous les modèles de la théorie, on dit aussi que c'est une **conséquence logique** des axiomes de la théorie.
- axiomatiser une théorie (géométrie, . . . )
  - Partir d'un ensemble restreint d'objets de base et de leurs propriétés
  - Construire logiquement les concepts avancés et les théorèmes

# Axiomes pour les groupes I

Exemple type : les *entiers relatifs* ( $\mathbb{Z}$ )

- symboles de fonctions :
  - constante  $0$ ,
  - opération binaire  $+$ ,
  - opération unaire  $-$  (le  $t - u$  binaire n'est pas primitif :  $t + (-u)$ )
- symbole de prédicat binaire : égalité.
- 3 axiomes (+ ceux de l'égalité)
  - associativité :  $\forall x y z, (x + y) + z = x + (y + z)$
  - élément neutre :  $\forall x, x + 0 = x \wedge 0 + x = x$
  - inverse :  $\forall x, x + (-x) = 0 \wedge (-x) + x = 0$
- Modèle : entiers relatifs, groupe des permutations. ...
- Conséquence :  $\forall x, -(-x) = x$

$$--x = --x + 0 = --x + (-x + x) = (--x + -x) + x = 0 + x = x$$

(utilise les propriétés de symétrie, transitivité et congruence de l'égalité)

# Axiomes pour la géométrie

- Euclide : points, segment, droite, demi-droite et cercle
- Hilbert : points, droites, incidence (un point est sur une droite), un point est entre deux autres (segment), congruence de segments, angles, triangles
  - par deux points, il passe une et une seule droite
  - sur une droite, il y a au moins 2 points distincts et un point qui n'est pas sur la droite
  - parallèle : soit une droite  $d$  et un point  $x$  qui n'est pas sur la droite, il existe une unique droite qui passe par  $x$  et qu n'a pas de point commun avec  $d$
- Modélisation logique :
  - prédicats unaires  $P$  et  $D$  (points et droites)
  - prédicats binaires :  $x \in d$  et  $p = q \dots$

$$\begin{aligned} &\forall p q, P(p) \wedge P(q) \wedge \neg(p = q) \Rightarrow \\ &\exists d, D(d) \wedge p \in d \wedge q \in d \wedge (\forall d', D(d') \wedge p \in d' \wedge q \in d' \Rightarrow d = d') \end{aligned}$$

- la géométrie (espaces euclidiens) est un modèle de la théorie de Hilbert

# Théorie de l'égalité

Un symbole binaire quelconque, noté  $=$  de manière infixé.

- réflexivité :  $\forall x, x = x$
- symétrie :  $\forall x y, x = y \Rightarrow y = x$
- transitivité :  $\forall x y z, (x = y \wedge y = z) \Rightarrow x = z$
- congruence/symboles de fonction  $f$  (arité  $n$ ) :  
 $\forall x_1 \dots x_n y_1 \dots y_n, (x_1 = y_1 \wedge \dots \wedge x_n = y_n) \Rightarrow f(x_1, \dots, x_n) = f(y_1, \dots, y_n)$
- congruence/symboles de prédicat  $P$  (arité  $n$ ) :  
 $\forall x_1 \dots x_n y_1 \dots y_n, (x_1 = y_1 \wedge \dots \wedge x_n = y_n) \Rightarrow P(x_1, \dots, x_n) \Rightarrow P(y_1, \dots, y_n)$

Exemples

$$\forall x_1 x_2 y_1 y_2, (x_1 = y_1) \wedge (x_2 = y_2) \Rightarrow (x_1 + x_2) = (y_1 + y_2)$$

$$\forall x_1 y_1, (x_1 = y_1) \Rightarrow (-x_1) = (-y_1)$$

$$\forall x_1 y_1, (x_1 = y_1) \Rightarrow D(x_1) \Rightarrow D(y_1)$$

L'égalité préserve le prédicat d'égalité trivialement (symétrie et transitivité)

$$\forall x_1 x_2 y_1 y_2, (x_1 = y_1) \wedge (x_2 = y_2) \Rightarrow (x_1 = x_2) \Rightarrow (y_1 = y_2)$$

- on pourra écrire  $t \neq u$  pour la formule  $\neg(t = u)$
- pour n'importe quelle formule  $\Phi$  avec une variable libre  $x$  :

$$\forall x y, (x = y) \Rightarrow (\Phi \Rightarrow \Phi[x \leftarrow y])$$

Preuve par récurrence sur la structure de la formule (voir suite)

- le symbole d'égalité sera utilisé avec les axiomes précédents implicites
- l'interprétation de l'égalité est une relation d'équivalence quelconque (pas forcément l'égalité du domaine)
- les interprétations des symboles de fonction et de prédicat doivent respecter la congruence
  - exemple des rationnels, des ensembles finis...



# Théorie arithmétique (Peano)

- Constante :  $0$
- Opération unaire *successeur* (+1) :  $S(\_)$ ,
- Opérations binaires addition et multiplication :  $\_ + \_$  et  $\_ * \_$
- Symbole de prédicat binaire d'égalité ( $=$ )
- Axiomes de l'égalité
- Axiomes arithmétique
  - $\forall x, S(x) \neq 0$
  - $\forall x, x = 0 \vee \exists y, x = S(y)$  (inutile en présence de récurrence)
  - $\forall x y, S(x) = S(y) \Rightarrow x = y$
  - $\forall x, x + 0 = x$
  - $\forall x y, x + S(y) = S(x + y)$
  - $\forall x, x \times 0 = 0$
  - $\forall x y, x \times S(y) = (x \times y) + x$
- Récurrence (nombre dénombrable d'axiomes, un pour chaque formule  $\Phi$ )

$$\forall x_1 \dots x_n, \Phi[x \leftarrow 0] \Rightarrow (\forall x, \Phi \Rightarrow \Phi[x \leftarrow S(x)]) \Rightarrow \forall x, \Phi$$

- A chaque entier naturel  $n \in \mathbb{N}$  on fait correspondre un terme noté  $\tilde{n}$  correspondant à  $S^n(O)$ .
- A partir des symboles de la théorie, on peut définir de nouvelles notions
  - $t \leq u \stackrel{\text{def}}{=} \exists d, t + d = u$
- A partir des axiomes de la théorie, on peut déduire de nouvelles propriétés
  - $\forall x, 0 \leq x$
  - $\forall x y, x \leq y \Leftrightarrow S(x) \leq S(y)$
  - transitivité :  $\forall x y z, x \leq y \Rightarrow y \leq z \Rightarrow x \leq z$

# Application de la théorie de l'arithmétique

- La théorie arithmétique est suffisante pour *représenter* les fonctions et prédicats *calculables* sur les entiers.

à une fonction  $f \in \mathbb{N} \rightarrow \mathbb{N}$  correspond une formule  $F[x, y]$

$$f(n) = m \quad \text{ssi} \quad F[\tilde{n}, \tilde{m}] \text{ est vrai dans l'arithmétique de Peano}$$

- Exemples

- soustraction  $(x - y) : F[x, y, z] \stackrel{\text{def}}{=} y + z = x \vee x < y \wedge z = 0$
- minimisation (+ petit  $n$  tel que  $G(x, n)$ ) :  
 $\mu G[x, y] \stackrel{\text{def}}{=} G(x, y) \wedge \forall z, z < y \Rightarrow \neg G(x, z)$
- Fonctions complexes via un codage des couples et des suites pour simuler les calculs récursifs



*That's all Folks!*

## Definition (Théorie)

Une théorie est définie par

- un ensemble de symboles de fonctions et de prédicats (la **signature**) et
  - un ensemble de formules **closes** (sans variables libres) construites sur ce langage (les **axiomes**).
- 
- *syntaxe* et *sémantique*
  - Une théorie peut être définie par un ensemble fini ou infini d'axiomes.
  - Un **modèle d'une théorie** est donné par une interprétation de la signature dans laquelle tous les axiomes ont pour valeur vraie.
  - Une formule est **valide dans une théorie** si elle vraie dans tous les modèles de la théorie, on dit aussi que c'est une **conséquence logique** des axiomes de la théorie.
  - axiomatiser une théorie :
    - Partir d'un ensemble restreint d'objets de base et de leurs propriétés
    - Construire logiquement les concepts avancés et les théorèmes

# La dernière fois : la théorie de l'égalité

Un symbole binaire quelconque, noté  $=$  de manière infixé.

- réflexivité :  $\forall x, x = x$
- symétrie :  $\forall x y, x = y \Rightarrow y = x$
- transitivité :  $\forall x y z, (x = y \wedge y = z) \Rightarrow x = z$
- congruence/symboles de fonction  $f$  (arité  $n$ ) :  
 $\forall x_1 \dots x_n y_1 \dots y_n, (x_1 = y_1 \wedge \dots \wedge x_n = y_n) \Rightarrow f(x_1, \dots, x_n) = f(y_1, \dots, y_n)$
- congruence/symboles de prédicat  $P$  (arité  $n$ ) :  
 $\forall x_1 \dots x_n y_1 \dots y_n, (x_1 = y_1 \wedge \dots \wedge x_n = y_n) \Rightarrow P(x_1, \dots, x_n) \Rightarrow P(y_1, \dots, y_n)$

# Propriétés des théories

Soit une théorie  $\mathcal{A}$

- $\mathcal{A}$  est **récursive** s'il existe un algorithme qui étant donné une formule  $P$  permet de calculer si  $P \in \mathcal{A}$
- $\mathcal{A}$  est **cohérente** si elle possède au moins un modèle, en particulier elle ne permet pas de déduire  $\perp$  (n'importe quoi)
- $\mathcal{A}$  est **décidable** s'il existe un algorithme qui étant donné une formule  $P$  permet de savoir si  $P$  est valide dans la théorie ou pas
- $\mathcal{A}$  est **complète** si pour toute formule  $P$ , soit  $P$  est valide dans la théorie, soit  $\neg P$  est valide dans la théorie.

Quelques résultats

- exemples de théories décidables : calcul propositionnel, arithmétique linéaire (Presburger), ordres discrets, théorie des réels...
- une théorie récursive et complète est décidable
- théorème d'incomplétude de Gödel : toute théorie cohérente qui contient l'arithmétique est incomplète

# Modélisation propositionnelle avancée : Sudoku

8		4				2		9
		9				1		
1			3		2			7
	5		1		4		8	
				3				
	1		7		9		2	
5			4		3			8
		3				4		
4		6				3		1

- position  $p = (i, j)$  sur la ligne  $i$  et la colonne  $j$
- variable propositionnelle  $x_p^k$  : le chiffre  $k$  est à la position  $p$

- donner des ensembles de formules pour les propriétés suivantes :
  - $A(p)$  : au moins un chiffre à la position  $p$
  - $B(p)$  : pas deux chiffres différents à la position  $p$
  - $C(p_1, p_2, \dots, p_9)$  tout chiffre  $k = 1, \dots, 9$  se trouve au moins à l'une des positions  $p_1, p_2, \dots, p_9$
  - $D(p_1, p_2, \dots, p_9)$  pas deux fois le même chiffre à des positions différentes en utilisant  $D'(p, q)$  pour cette propriété concernant deux cases.
- relier la résolution du Sudoku et l'existence de modèles d'un ensemble de formules.



# Modélisation au premier ordre : Sudoku

- La modélisation propositionnelle introduit de nombreuses variables et axiomes.
- On peut considérer les positions et chiffres comme des objets de la logique
- Un symbole de prédicat à trois arguments  $\text{pos}(i, j, k)$  vrai lorsque le chiffre  $k$  est à la position  $(i, j)$  et l'égalité
- Au plus un seul chiffre  $k$  par case  $(i, j)$  :

$$\forall i j k_1 k_2, \text{pos}(i, j, k_1) \wedge \text{pos}(i, j, k_2) \Rightarrow k_1 \neq k_2$$


- Exprimer logiquement que les objets peuvent prendre exactement 9 valeurs différentes sans utiliser l'arithmétique.
- Comment représenter simplement que deux positions sont dans le même cadran ?

- Connaître la définition d'une théorie.
- Connaître la théorie de l'égalité.
- Distinguer modélisation en calcul propositionnel et modélisation en logique du premier ordre.

# 1–Maîtriser le langage logique

- 1 Définition du langage
- 2 Structure des formules
- 3 Formules vraies
- 4 Théorie et modélisation
- 5 **Définition récursive sur les formules**
  - Définir une fonction sur les formules
  - Raisonner sur les formules
  - Définition récursive sur les termes

# Définition récursive sur les formules

- définir des opérations mathématiques sur les formules (taille, substitution, valeur de vérité, transformations ...)
  - les implémenter sur machine
  - équations récursives :
    - une équation (et une seule) pour chaque construction possible de formule :
      - formules atomiques :  $\top$ ,  $\perp$ , prédicat
      - connecteurs propositionnels  $\neg A$ ,  $A \vee B$ ,  $A \wedge B$ ,  $A \Rightarrow B$ .
      - formules quantifiées  $\forall x, A$ ,  $\exists x, A$
    - le membre droit de l'équation peut contenir des appels récursifs à la fonction sur les sous-formules  $A$ ,  $B$
    - correspond à un calcul par parcours de l'arbre
-   $A$  et  $B$  sont des (meta)-variables mathématiques qui représentent des formules quelconques.
- facilite le raisonnement (cf TD)

# Exemple

compter le nombre de connecteurs propositionnels dans une formule :

$$\begin{aligned}\text{nbsymbp}(p) &= \text{si } p \text{ atomique} \\ \text{nbsymbp}(\neg A) &= 1 + \text{nbsymbp}(A) \\ \text{nbsymbp}(A \wedge B) &= \\ \text{nbsymbp}(A \vee B) &= \\ \text{nbsymbp}(A \Rightarrow B) &= \\ \text{nbsymbp}(\forall x, A) &= \\ \text{nbsymbp}(\exists x, A) &= \end{aligned}$$

- fonction `contient-neg` qui teste si une formule comporte au moins une négation
- fonction `nbatom` qui calcule le nombre de formules atomiques dans une formule

# Exemple

compter le nombre de connecteurs propositionnels dans une formule :

$$\begin{aligned} \text{nbsymbp}(p) &= 0 \text{ si } p \text{ atomique} \\ \text{nbsymbp}(\neg A) &= 1 + \text{nbsymbp}(A) \\ \text{nbsymbp}(A \wedge B) &= \\ \text{nbsymbp}(A \vee B) &= \\ \text{nbsymbp}(A \Rightarrow B) &= \\ \text{nbsymbp}(\forall x, A) &= \\ \text{nbsymbp}(\exists x, A) &= \end{aligned}$$

- fonction `contient-neg` qui teste si une formule comporte au moins une négation
- fonction `nbatom` qui calcule le nombre de formules atomiques dans une formule

# Exemple

compter le nombre de connecteurs propositionnels dans une formule :

$$\begin{aligned} \text{nbsymbp}(p) &= 0 \text{ si } p \text{ atomique} \\ \text{nbsymbp}(\neg A) &= 1 + \text{nbsymb}(A) \\ \text{nbsymbp}(A \wedge B) &= \text{nbsymb}(A) + 1 + \text{nbsymb}(B) \\ \text{nbsymbp}(A \vee B) &= \text{nbsymb}(A) + 1 + \text{nbsymb}(B) \\ \text{nbsymbp}(A \Rightarrow B) &= \text{nbsymb}(A) + 1 + \text{nbsymb}(B) \\ \text{nbsymbp}(\forall x, A) &= \text{nbsymb}(A) \\ \text{nbsymbp}(\exists x, A) &= \text{nbsymb}(A) \end{aligned}$$

- fonction `contient-neg` qui teste si une formule comporte au moins une négation
- fonction `nbatom` qui calcule le nombre de formules atomiques dans une formule

# Restriction au calcul propositionnel

- seulement des variables propositionnelles (symboles de prédicat sans argument)
- pas de quantificateur
- Définir **récurivement**  $\text{Vars}(P)$  ensemble des variables propositionnelles qui apparaissent dans  $P$

$$\begin{array}{lll} \text{Vars}(X) & = & X \in \mathcal{V}_p \\ \text{Vars}(\perp) & = & \\ \text{Vars}(\top) & = & \\ \text{Vars}(\neg P) & = & \\ \text{Vars}(P_1 \circ P_2) & = & \circ \in \{\wedge, \vee, \Rightarrow\} \end{array}$$



# Restriction au calcul propositionnel

- seulement des variables propositionnelles (symboles de prédicat sans argument)
- pas de quantificateur
- Définir **récurivement**  $\text{Vars}(P)$  ensemble des variables propositionnelles qui apparaissent dans  $P$

$$\begin{aligned}\text{Vars}(X) &= \{X\} & X \in \mathcal{V}_p \\ \text{Vars}(\perp) &= \emptyset \\ \text{Vars}(\top) &= \emptyset \\ \text{Vars}(\neg P) &= \text{Vars}(P) \\ \text{Vars}(P_1 \circ P_2) &= \text{Vars}(P_1) \cup \text{Vars}(P_2) & \circ \in \{\wedge, \vee, \Rightarrow\}\end{aligned}$$

# Substitution

Remplacer une variable propositionnelle  $X$  par une formule  $Q$  dans une formule propositionnelle  $P$  :

$$X[X \leftarrow Q] =$$

$$P[X \leftarrow Q] =$$

$$(\neg A)[X \leftarrow Q] =$$

$$(A \circ B)[X \leftarrow Q] =$$

$$X \in \mathcal{V}_p$$

$$P \text{ est atomique } P \neq X$$

$$\circ \in \{\wedge, \vee, \Rightarrow\}$$

## Exemple

Calculer :  $((\neg x \wedge y) \vee \neg y \Rightarrow x)[x \leftarrow (p \Rightarrow p)]$

# Substitution

Remplacer une variable propositionnelle  $X$  par une formule  $Q$  dans une formule propositionnelle  $P$  :

$$X[X \leftarrow Q] = Q$$

$$X \in \mathcal{V}_p$$

$$P[X \leftarrow Q] = P$$

$$P \text{ est atomique } P \neq X$$

$$(\neg A)[X \leftarrow Q] = \neg(A[X \leftarrow Q])$$

$$(A \circ B)[X \leftarrow Q] = (A[X \leftarrow Q] \circ B[X \leftarrow Q]) \quad \circ \in \{\wedge, \vee, \Rightarrow\}$$

## Exemple

Calculer :  $((\neg x \wedge y) \vee \neg y \Rightarrow x)[x \leftarrow (p \Rightarrow p)]$

$$((\neg(p \Rightarrow p) \wedge y) \vee \neg y \Rightarrow (p \Rightarrow p))$$

# Représentation des formules propositionnelles

```
type connecteur = Impl | Et | Ou
type form = Var of int | Bot | Top
           | Neg of form
           | Bin of form * connecteur * form

let rec vars = function
  Var n -> [n]
  | Bot | Top -> []
  | Neg f -> vars f
  | Bin(f,_,g) -> vars f @ vars g

let rec subst x q = function
  Var n -> if n = x then q else Var n
  | Bot -> Bot | Top -> Top
  | Neg f -> Neg (subst x q f)
  | Bin(f,o,g) -> Bin(subst x q f,o,subst x q g)
```

# Valeur de vérité (formule propositionnelle)

On se donne une interprétation  $I \in \mathcal{V}_p \rightarrow \mathbb{B}$ , et on définit  $\text{val}(I, P) \in \mathbb{B}$

$$\begin{aligned}\text{val}(I, \perp) &= F \\ \text{val}(I, \top) &= V \\ \text{val}(I, x) &= I(x) & x \in \mathcal{V}_p \\ \text{val}(I, \neg A) &= \text{si } \text{val}(I, A) = V \text{ alors } F \text{ sinon } V \\ \text{val}(I, A \wedge B) &= \text{si } \text{val}(I, A) = V \text{ alors } \text{val}(I, B) \text{ sinon } F \\ \text{val}(I, A \vee B) &= \text{si } \text{val}(I, A) = V \text{ alors } V \text{ sinon } \text{val}(I, B) \\ \text{val}(I, A \Rightarrow B) &= \text{si } \text{val}(I, A) = V \text{ alors } \text{val}(I, B) \text{ sinon } V\end{aligned}$$

## Exemple

- $P \stackrel{\text{def}}{=} ((x \Rightarrow y) \Rightarrow y) \Rightarrow x$ ,  $I \stackrel{\text{def}}{=} \{x \mapsto F, y \mapsto V\}$ , calculer  $\text{val}(I, P)$

## Propriété

- La valeur d'une formule ne dépend que de la valeur de l'interprétation pour les variables qui apparaissent dans la formule.

$$\text{val}(I_1, P) = \text{val}(I_2, P) \quad \text{si pour tout } x \in \text{Vars}(P), I_1(x) = I_2(x)$$

# Schéma de récurrence pour les formules

Soit une propriété mathématique  $\phi(P)$  qui dépend d'une formule  $P$ .

- Si on peut montrer que :

- 1  $\phi(p)$  est vérifiée lorsque  $p$  est une formule atomique (en particulier  $\phi(\top)$  et  $\phi(\perp)$  sont vérifiés)
- 2 pour une formule  $A$  quelconque, en supposant que  $\phi(A)$  est vérifié, on peut montrer  $\phi(\neg A)$
- 3 pour des formules  $A$  et  $B$  quelconques, en supposant que  $\phi(A)$  et  $\phi(B)$  sont vérifiées, on peut montrer  $\phi(A \wedge B)$  ainsi que  $\phi(A \vee B)$  et  $\phi(A \Rightarrow B)$
- 4 pour une formule  $A$  quelconque, et une variable  $x$ , en supposant que  $\phi(A)$  est vérifié, on peut montrer  $\phi(\forall x, A)$  et  $\phi(\exists x, A)$

- Alors on peut en déduire que pour toute formule logique  $P$ ,  $\phi(P)$  est vérifié.

# Exemple

Schéma utile pour montrer des propriétés de fonctions sur les formules définies récursivement.

$\text{nbatom}(p)$  : nombre d'occurrences de sous-formules atomiques

$$\begin{aligned}\text{nbatom}(p) &= 1 \text{ si } p \text{ atomique} \\ \text{nbatom}(\neg A) &= \text{nbatom}(A) \\ \text{nbatom}(\forall x, A) &= \text{nbatom}(A) \\ \text{nbatom}(\exists x, A) &= \text{nbatom}(A) \\ \text{nbatom}(A \wedge B) &= \text{nbatom}(A) + \text{nbatom}(B) \\ \text{nbatom}(A \vee B) &= \text{nbatom}(A) + \text{nbatom}(B) \\ \text{nbatom}(A \Rightarrow B) &= \text{nbatom}(A) + \text{nbatom}(B)\end{aligned}$$

**Lemme :** Pour toute formule  $P$ , on a  $\text{nbatom}(P) \leq 1 + \text{nbsymbp}(P)$ .

# Preuve par récurrence

- propriété  $\phi(P)$  à montrer par récurrence structurale sur  $P$  :  
 $\text{nbatom}(P) \leq 1 + \text{nbsymbp}(P)$ .
- Examen de chacun des cas possibles pour la formule  $P$ 
  - 1  $P$  formule atomique  $p$ . Par définition,  $\text{nbatom}(p) = 1$  et  $\text{nbsymbp}(p) = 0$  et donc  $\text{nbatom}(p) \leq 1 + \text{nbsymbp}(p)$ ,  $\phi(p)$  est vérifié.
  - 2  $P$  est une négation  $\neg A$ ,
    - $A$  quelconque vérifie l'hypothèse de récurrence  $\phi(A)$  ( $\text{nbatom}(A) \leq 1 + \text{nbsymbp}(A)$ ).
    - montrons  $\phi(\neg A)$
    - par définition,  $\text{nbatom}(\neg A) = \text{nbatom}(A)$  et  $\text{nbsymbp}(\neg A) = 1 + \text{nbsymbp}(A)$ .
    - En utilisant l'hypothèse de récurrence on a donc

$$\begin{aligned}\text{nbatom}(\neg A) = \text{nbatom}(A) &\leq 1 + \text{nbsymbp}(A) \\ &= \text{nbsymbp}(\neg A) \leq 1 + \text{nbsymbp}(\neg A)\end{aligned}$$

- donc  $\phi(\neg A)$  est vérifiée.

- 3 pour les autres cas, voir les notes de cours

On a bien examiné tous les cas possibles, on en conclut que pour toute formule logique  $P$ , on a  $\text{nbatom}(P) \leq 1 + \text{nbsymbp}(P)$ .



# Exemple

Schéma utile pour montrer des propriétés de fonctions sur les formules définies récursivement.

- Soit une formule propositionnelle  $Q$  et une variable propositionnelle  $X$ .
- On souhaite montrer que pour toute formule propositionnelle  $P$  telle que  $X \notin \text{Vars}(P)$ , on a  $P[X \leftarrow Q] = P$
- La propriété  $\phi(P)$  à montrer par récurrence structurelle sur  $P$  :  
si  $X \notin \text{Vars}(P)$  alors  $P[X \leftarrow Q] = P$
- Examen de chacun des cas possibles pour la formule  $P$

# Preuve détaillée

## 1 $P$ formule atomique :

Comme  $X \notin \text{Vars}(P)$ , on a que  $P \neq X$  et donc  $P[X \leftarrow Q] = P$  par définition de la substitution

## 2 $P$ de la forme $\neg A$ :

- L'hypothèse de récurrence sur  $A$  nous assure que si  $X \notin \text{Vars}(A)$  alors  $A[X \leftarrow Q] = A$
- Comme  $X \notin \text{Vars}(P)$  et  $\text{Vars}(\neg A) = \text{Vars}(A)$ , on a que  $X \notin \text{Vars}(A)$ .
- L'hypothèse de récurrence nous permet de déduire  $A[X \leftarrow Q] = A$
- Par définition de la substitution  $(\neg A)[X \leftarrow Q] = \neg(A[X \leftarrow Q])$
- On en déduit le résultat attendu :  $(\neg A)[X \leftarrow Q] = \neg A$

## 3 $P$ de la forme $A \circ B$ (les trois connecteurs $\vee, \wedge, \Rightarrow$ se comportent pareil) :

- L'hypothèse de récurrence sur  $A$  nous assure que si  $X \notin \text{Vars}(A)$  alors  $A[X \leftarrow Q] = A$  et de même pour  $B$ , si  $X \notin \text{Vars}(B)$  alors  $B[X \leftarrow Q] = B$
- Comme  $X \notin \text{Vars}(P)$  et  $\text{Vars}(A \circ B) = \text{Vars}(A) \cup \text{Vars}(B)$ , on a que  $X \notin \text{Vars}(A)$  et  $X \notin \text{Vars}(B)$ .
- Par hypothèses de récurrence, on déduit  $A[X \leftarrow Q] = A$  et  $B[X \leftarrow Q] = B$
- Par définition de la substitution  $(A \circ B)[X \leftarrow Q] = (A[X \leftarrow Q]) \circ (B[X \leftarrow Q])$
- On a donc le résultat attendu :  $(A \circ B)[X \leftarrow Q] = A \circ B$

On a examiné tous les cas possibles, on en conclut que pour toute formule propositionnelle  $P$ , on a bien que si  $X \notin \text{Vars}(P)$  alors  $P[X \leftarrow Q] = P$



*That's all Folks!*

- Construire des fonctions sur les termes et les formules en utilisant un système d'équations récursives
- Faire des raisonnements simples par récurrence sur la structure des termes ou des formules

# La dernière fois : définition récursive sur les formules

- définir des opérations mathématiques sur les formules (taille, substitution, valeur de vérité, transformations ...)
- les implémenter sur machine
- équations récursives :
  - une équation (et une seule) pour chaque construction possible de formule :
    - formules atomiques :  $\top$ ,  $\perp$ , prédicat
    - connecteurs propositionnels  $\neg A$ ,  $A \vee B$ ,  $A \wedge B$ ,  $A \Rightarrow B$ .
    - formules quantifiées  $\forall x, A$ ,  $\exists x, A$
  - le membre droit de l'équation peut contenir des appels récursifs à la fonction sur les sous-formules  $A$ ,  $B$
  - correspond à un calcul par parcours de l'arbre



$A$  et  $B$  sont des (meta)-variables mathématiques qui représentent des formules quelconques.

# La dernière fois : exemple

compter le nombre de connecteurs propositionnels dans une formule :

$$\begin{aligned} \text{nbsymbp}(p) &= 0 \text{ si } p \text{ atomique} \\ \text{nbsymbp}(\neg A) &= 1 + \text{nbsymbp}(A) \\ \text{nbsymbp}(A \wedge B) &= \text{nbsymbp}(A) + 1 + \text{nbsymbp}(B) \\ \text{nbsymbp}(A \vee B) &= \text{nbsymbp}(A) + 1 + \text{nbsymbp}(B) \\ \text{nbsymbp}(A \Rightarrow B) &= \text{nbsymbp}(A) + 1 + \text{nbsymbp}(B) \\ \text{nbsymbp}(\forall x, A) &= \text{nbsymbp}(A) \\ \text{nbsymbp}(\exists x, A) &= \text{nbsymbp}(A) \end{aligned}$$

# La dernière fois : schéma de récurrence pour les formules

Soit une propriété mathématique  $\phi(P)$  qui dépend d'une formule  $P$ .

- Si on peut montrer que :

- 1  $\phi(p)$  est vérifiée lorsque  $p$  est une formule atomique
- 2 pour une formule  $A$  quelconque, en supposant que  $\phi(A)$  est vérifié, on peut montrer  $\phi(\neg A)$
- 3 pour des formules  $A$  et  $B$  quelconques, en supposant que  $\phi(A)$  et  $\phi(B)$  sont vérifiées, on peut montrer  $\phi(A \wedge B)$  ainsi que  $\phi(A \vee B)$  et  $\phi(A \Rightarrow B)$
- 4 pour une formule  $A$  quelconque, et une variable  $x$ , en supposant que  $\phi(A)$  est vérifié, on peut montrer  $\phi(\forall x, A)$  et  $\phi(\exists x, A)$

- Alors on peut en déduire que pour toute formule logique  $P$ ,  $\phi(P)$  est vérifié.

# La dernière fois : exemple

$\text{nbatom}(p)$  : nombre d'occurrences de sous-formules atomiques

$$\begin{aligned}\text{nbatom}(p) &= 1 \text{ si } p \text{ atomique} \\ \text{nbatom}(\neg A) &= \text{nbatom}(A) \\ \text{nbatom}(\forall x, A) &= \text{nbatom}(A) \\ \text{nbatom}(\exists x, A) &= \text{nbatom}(A) \\ \text{nbatom}(A \wedge B) &= \text{nbatom}(A) + \text{nbatom}(B) \\ \text{nbatom}(A \vee B) &= \text{nbatom}(A) + \text{nbatom}(B) \\ \text{nbatom}(A \Rightarrow B) &= \text{nbatom}(A) + \text{nbatom}(B)\end{aligned}$$

**Lemme :** Pour toute formule  $P$ , on a  $\text{nbatom}(P) \leq 1 + \text{nbsymbp}(P)$ .



# La dernière fois : preuve par récurrence

- propriété  $\phi(P)$  à montrer par récurrence structurelle sur  $P$  :  
 $\text{nbatom}(P) \leq 1 + \text{nbsymbp}(P)$ .
- Examen de chacun des cas possibles pour la formule  $P$ 
  - 1  $P$  formule atomique  $p$ . Par définition,  $\text{nbatom}(p) = 1$  et  $\text{nbsymbp}(p) = 0$  et donc  $\text{nbatom}(p) \leq 1 + \text{nbsymbp}(p)$ ,  $\phi(p)$  est vérifié.
  - 2  $P$  est une négation  $\neg A$ ,
    - $A$  quelconque vérifie l'hypothèse de récurrence  $\phi(A)$  ( $\text{nbatom}(A) \leq 1 + \text{nbsymbp}(A)$ ).
    - montrons  $\phi(\neg A)$
    - par définition,  $\text{nbatom}(\neg A) = \text{nbatom}(A)$  et  $\text{nbsymbp}(\neg A) = 1 + \text{nbsymbp}(A)$ .
    - En utilisant l'hypothèse de récurrence on a donc

$$\begin{aligned}\text{nbatom}(\neg A) = \text{nbatom}(A) &\leq 1 + \text{nbsymbp}(A) \\ &= \text{nbsymbp}(\neg A) \leq 1 + \text{nbsymbp}(\neg A)\end{aligned}$$

- donc  $\phi(\neg A)$  est vérifiée.

- 3 pour les autres cas, voir les notes de cours

On a bien examiné tous les cas possibles, on en conclut que pour toute formule logique  $P$ , on a  $\text{nbatom}(P) \leq 1 + \text{nbsymbp}(P)$ .

# Exercice TD : Sous-formules

On se restreint aux formules du calcul propositionnel.

On dit qu'une formule  $Q$  est une **sous-formule** de  $P$  si  $Q = P$  ou bien si la formule  $Q$  apparaît sous un connecteur de  $P$ .

C'est-à-dire  $P = \neg P'$  et  $Q$  est une sous-formule de  $P'$  ou bien  $P = P_1 \circ P_2$  et  $Q$  est une sous-formule de  $P_1$  ou bien une sous-formule de  $P_2$  avec  $\circ$  un des connecteurs binaires :  $\{\vee, \wedge, \Rightarrow\}$ .

- 1 Donner toutes les sous-formules de la formule  $\neg(p \vee (q \wedge r)) \Rightarrow (p \wedge q)$
- 2 Donner les équations qui définissent la fonction **sf** qui à une formule propositionnelle  $P$  associe l'ensemble de ses sous-formules.
- 3 Trouver un majorant du nombre de sous-formules d'une formule  $P$  qui utilise  $n$  connecteurs logiques. Donner un exemple où ce majorant est atteint. Prouver ce résultat par récurrence structurelle sur la formule.
- 4 (optionnel) Même question pour un minorant du nombre de sous-formules.

# Définition récursive sur les termes

- Pour traiter le cas des formules atomiques en logique du premier ordre, il faut souvent prendre en compte les termes arguments des prédicats.
- Les termes sont définis à partir de la signature et contiennent possiblement des variables
- Les termes se représentent par des arbres donc chaque nœud est étiqueté par un symbole de fonction, le nombre de sous-arbres est donné par l'arité du symbole et les feuilles sont les constantes et les variables.



- Le même principe de définition récursive de fonctions (mathématiques) s'applique sur les termes.
- Une équation pour les variables et une pour chaque symbole de la signature avec de possibles appels récursifs sur les sous-termes.

$$\begin{aligned} G(x) &= \dots && x \text{ variable} \\ G(f(t_1, \dots, t_n)) &= \dots G(t_1) \dots G(t_n) \dots \end{aligned}$$

# Exemple

- Signature : constante  $c$ , fonction unaire  $f$ , fonction binaire  $g$ .
- On définit une fonction  $\text{clos}$  qui étant donné un terme  $t$  teste s'il est clos (pas de variables)

```
 $\text{clos}(x)$       = faux si  $x$  est une variable  
 $\text{clos}(c)$       = vrai  
 $\text{clos}(f(t))$    =  $\text{clos}(t)$   
 $\text{clos}(g(t,u))$  =  $\text{clos}(t)$  et  $\text{clos}(u)$ 
```

# Définition récursive sur les termes

## Definition

$\mathcal{F}$  signature,  $\mathcal{X}$  ensemble de variables et  $\mathcal{D}$  un ensemble quelconque.

Pour définir une application  $G \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \rightarrow \mathcal{D}$ , on se donne :

- 1 Une application  $V$  dans  $\mathcal{X} \rightarrow \mathcal{D}$  ;
- 2 Pour chaque constante  $c \in \mathcal{F}_0$ , un élément  $g_c \in \mathcal{D}$
- 3 Pour chaque symbole de fonction  $f \in \mathcal{F}_n$ , une application  $G_f$  dans

$$\underbrace{\mathcal{T}(\mathcal{F}, \mathcal{X}) \times \dots \times \mathcal{T}(\mathcal{F}, \mathcal{X})}_{n \text{ fois}} \times \underbrace{\mathcal{D} \times \dots \times \mathcal{D}}_{n \text{ fois}} \rightarrow \mathcal{D}$$

Il existe une unique application  $G$  dans  $\mathcal{T}(\mathcal{F}, \mathcal{X}) \rightarrow \mathcal{D}$  qui vérifie :

$$\begin{aligned} G(x) &= V(x) & (x \in \mathcal{X}) \\ G(c) &= g_c & (c \in \mathcal{F}_0) \\ G(f(t_1, \dots, t_n)) &= G_f(t_1, \dots, t_n, G(t_1), \dots, G(t_n)) & (f \in \mathcal{F}_n) \end{aligned}$$

# Fonctions génériques utiles

## Exemple (Taille d'un terme)

*size* compte le nombre de symboles dans un terme.

- si  $x \in \mathcal{X}$  alors  $size(x) = 0$
- si  $c \in \mathcal{F}_0$  alors  $size(c) = 1$
- si  $f \in \mathcal{F}_n$  alors  $size(f(t_1, \dots, t_n)) = 1 + size(t_1) + \dots + size(t_n)$

$t \stackrel{\text{def}}{=} plus(0, S(0))$  vérifie  $size(t) = 4$ .

## Exemple (Hauteur d'un terme)

*ht* : compte le nombre maximal de symboles imbriqués dans un terme.

- si  $x \in \mathcal{X}$  alors  $ht(x) = 0$
- si  $c \in \mathcal{F}_0$  alors  $ht(c) = 1$
- si  $f \in \mathcal{F}_n$  alors  $ht(f(t_1, \dots, t_n)) = 1 + \max(ht(t_1), \dots, ht(t_n))$

$ht(t) = 3$ .

# Exercice : variables d'un terme

## Exercice

Ecrire une fonction `vars` qui prend en argument un terme et renvoie l'ensemble des variables qui apparaissent dans ce terme.

# Exercice : variables d'un terme

## Exercice

Ecrire une fonction `vars` qui prend en argument un terme et renvoie l'ensemble des variables qui apparaissent dans ce terme.

## Solution

- si  $x \in \mathcal{X}$  alors  $\text{vars}(x) = \{x\}$
- si  $c \in \mathcal{F}_0$  alors  $\text{vars}(c) = \emptyset$
- si  $f \in \mathcal{F}_n$  alors  $\text{vars}(f(t_1, \dots, t_n)) = \text{vars}(t_1) \cup \dots \cup \text{vars}(t_n)$



# Substitution sur les termes

- remplacement simultanée de plusieurs variables par des termes.
- application  $\sigma \in \mathcal{X} \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{X})$ , appelée **substitution** qui associe un terme à chaque variable.
- On note  $\{x_1 \leftarrow u_1; \dots; x_n \leftarrow u_n\}$  la substitution  $\sigma$  telle que  $\sigma(x) = u_i$  si  $x = x_i$  et  $\sigma(x) = x$  sinon.
- On définit pour chaque terme  $t$ , le résultat de la **substitution** dans  $t$  de toute variable  $x$  par  $\sigma(x)$  que l'on note  $t[\sigma]$ .
- Si  $\sigma$  est de la forme  $\{x_1 \leftarrow u_1; \dots; x_n \leftarrow u_n\}$ , alors le terme  $t[\sigma]$  sera noté  $t[x_1 \leftarrow u_1; \dots; x_n \leftarrow u_n]$ .

# Définition et exemple

La définition de  $t[\sigma]$  se fait de manière récursive sur  $t$  :

- si  $x \in \mathcal{X}$  alors  $x[\sigma] = \sigma(x)$
- si  $c \in \mathcal{F}_0$  alors  $c[\sigma] = c$
- si  $f \in \mathcal{F}_n$  alors  $f(t_1, \dots, t_n)[\sigma] = f(t_1[\sigma], \dots, t_n[\sigma])$

## Exemple

$t = \text{plus}(\text{mult}(x, y), S(x))$  et  $\sigma = \{x \leftarrow \text{mult}(y, 0); y \leftarrow 0\}$ .  
On a  $t[\sigma] = \text{plus}(\text{mult}(\text{mult}(y, 0), 0), S(\text{mult}(y, 0)))$

# Propriétés de la substitution

- le résultat de  $t[\sigma]$  ne dépend que de la valeur de la substitution  $\sigma$  sur les variables de  $t$ .
- soit deux substitutions  $\sigma_1$  et  $\sigma_2$  et un terme  $t$ , si pour toute variable  $x \in \text{vars}(t)$  on a  $\sigma_1(x) = \sigma_2(x)$  alors  $t[\sigma_1] = t[\sigma_2]$ .
- La preuve se fait aisément par récurrence structurelle sur le terme  $t$  suivant le schéma ci-dessous.

# Récurrance sur les termes

On peut utiliser un schéma de preuve par récurrence sur les termes de  $\mathcal{T}(\mathcal{F}, \mathcal{X})$

Soit  $\phi(t)$  une propriété mathématique qui dépend d'un terme  $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ . Si :

- 1 pour toute variable  $x \in \mathcal{X} : \phi(x)$  ;
- 2 pour chaque constante  $c \in \mathcal{F}_0 : \phi(c)$  ;
- 3 pour chaque symbole  $f \in \mathcal{F}_n$  : pour tous termes  $t_1 \dots t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$  si les propriétés  $\phi(t_1) \dots \phi(t_n)$  sont vérifiées (hypothèses de récurrence), alors il en est de même de  $\phi(f(t_1, \dots, t_n))$  ;

alors pour tout terme  $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$  la propriété  $\phi(t)$  est vérifiée.

# Exemple

On montre pour tout terme  $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$  que  $\text{ht}(t) \leq \text{size}(t)$

La preuve se fait par récurrence sur la structure du terme  $t$ .

La propriété à montrer est  $\phi(t) \stackrel{\text{def}}{=} \text{ht}(t) \leq \text{size}(t)$

**variable** soit  $x \in \mathcal{X}$ ,  $\text{ht}(x) \leq \text{size}(x)$  vrai car  $\text{ht}(x) = 0$  et  $\text{size}(x) = 0$

**constante**  $\text{ht}(c) \leq \text{size}(c)$  vrai car  $\text{ht}(c) = 1 = \text{size}(c)$ .

**symbole** si  $f \in \mathcal{F}_n$  et  $t_1, \dots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$  sont des termes quelconques qui vérifient l'hypothèse de récurrence  $\text{ht}(t_i) \leq \text{size}(t_i)$ . On doit montrer  $\text{ht}(f(t_1, \dots, t_n)) \leq \text{size}(f(t_1, \dots, t_n))$ .

$$\begin{aligned} \text{ht}(f(t_1, \dots, t_n)) &= 1 + \max(\text{ht}(t_1), \dots, \text{ht}(t_n)) && \text{(déf de ht)} \\ &\leq 1 + \text{ht}(t_1) + \dots + \text{ht}(t_n) && \text{(car ht}(t_i) \geq 0) \\ &\leq 1 + \text{size}(t_1) + \dots + \text{size}(t_n) && \text{(hyp. de récurrence)} \\ &= \text{size}(f(t_1, \dots, t_n)) && \text{(déf de size)} \end{aligned}$$

On en déduit que  $\text{ht}(t) \leq \text{size}(t)$  est vérifié pour tout les termes du langage.

# Substitution sur les formules

- substitution  $P[x \leftarrow t]$  d'une variable  $x$  par un terme  $t$  dans une formule  $P$
- éviter la capture d'une variable du terme  $t$  par un des quantificateurs interne de  $P$ .
- définition récursive de  $P[x \leftarrow t]$ 
  - Formules atomiques :
    - $\perp[x \leftarrow t] = \perp$
    - $\top[x \leftarrow t] = \top$
    - $R(t_1, \dots, t_n)[x \leftarrow t] = R(t_1[x \leftarrow t], \dots, t_n[x \leftarrow t])$
  - $(\neg A)[x \leftarrow t] = \neg(A[x \leftarrow t])$
  - $\circ \in \{\wedge, \vee, \Rightarrow\} : (A \circ B)[x \leftarrow t] = (A[x \leftarrow t]) \circ (B[x \leftarrow t])$
  - $(\forall y, Q)[x \leftarrow t]$  :
    - si  $y = x$  alors  $(\forall y, Q) = \forall x, Q$  et comme  $x$  n'est pas libre dans  $\forall x, Q$  on a  $(\forall y, Q)[x \leftarrow t] = (\forall x, Q)[x \leftarrow t] = \forall x, Q$
    - si  $y \neq x$  et  $y \notin \text{vars}(t)$  alors  $(\forall y, Q)[x \leftarrow t] = \forall y, (Q[x \leftarrow t])$   
pas de risque de capture puisque la variable liée  $y$  n'apparaît pas dans  $t$
  - $(\exists y, Q)[x \leftarrow t]$  : traitement analogue à  $(\forall y, Q)[x \leftarrow t]$

# Exemples

- 1  $(\forall y, R(x, y))[x \leftarrow f(x)]$
- 2  $(\forall y, R(x, y))[x \leftarrow f(y)]$
- 3  $(\forall y, R(x, y))[y \leftarrow x]$

- La définition est *partielle*, elle n'est pas définie dans le cas des formules avec quantificateurs si une variable liée dans la formule apparaît aussi dans le terme que l'on veut substituer.
- En procédant à un renommage des variables liées dans les quantificateurs, on se ramène à une situation dans laquelle la substitution sera possible.



- Savoir construire des fonctions sur les termes et les formules en utilisant un système d'équations récursives.
- Faire des raisonnements simples par récurrence sur la structure des termes ou des formules.
- Savoir calculer la **substitution** d'une variable (libre) par un terme dans une formule en évitant les problèmes de capture.

## 2—Donner du sens aux formules

### 1 Interprétations et vérité

- Chapitre 1 : Les bases de la logique du premier ordre :
  - structure des termes et des formules
  - comment utiliser ce langage pour modéliser des situations ou problèmes
  - la vérité d'une formule logique dépend du monde dans lequel on interprète les symboles de la signature
- Chapitre 2 :
  - retour sur la notion d'interprétation de manière plus détaillée
  - définition de la notion de conséquence logique et d'équivalence.
  - étude de quelques modèles particuliers.

# Interprétations et vérité

## signature $(\mathcal{F}, \mathcal{R})$

- $\mathcal{F}$  l'ensemble des symboles de fonctions (termes)
- $\mathcal{R}$  l'ensemble des symboles de prédicat (formules atomiques).

## Definition (Interprétation)

Une **interprétation**  $I$  de la signature  $(\mathcal{F}, \mathcal{R})$  est donnée par

- un ensemble  $\mathcal{D}$  *non vide* appelé **domaine** de l'interprétation ;
- pour chaque symbole de fonction  $f \in \mathcal{F}_n$  d'arité  $n$ , une fonction  $f_I$   $n$ -aire sur  $\mathcal{D}$  (c'est à dire  $f_I : \mathcal{D}^n \rightarrow \mathcal{D}$  : pour chaque  $v_1, \dots, v_n \in \mathcal{D}$ , on a  $f_I(v_1, \dots, v_n) \in \mathcal{D}$ ) ;
- pour chaque symbole de prédicat  $R \in \mathcal{R}_n$  d'arité  $n$ , une relation  $n$ -aire  $R_I$  sur  $\mathcal{D}$  ( $R_I \subseteq \mathcal{D}^n$ ).  
 $R_I$  ensemble de  $n$ -uplets  $(v_1, \dots, v_n)$  avec  $v_i \in \mathcal{D}$  qui correspondent au cas où, *dans cette interprétation*, la relation  $R$  est vraie.

On utilise également la terminologie de **modèle** ou de **structure** pour parler de l'interprétation d'une signature.

# Exemple : interprétation

Signature : prédicat binaire  $P$ .

Domaine  $D \stackrel{\text{def}}{=} \{1, 2, 3, 4\}$ .

L'interprétation de  $P$  donnée par un tableau  $4 \times 4$ .

La formule atomique  $P(t, u)$  est vraie si  $t$  vaut  $i$  et  $u$  vaut  $j$  et que la case sur la ligne  $i$  et la colonne  $j$  est grisée.

Soient les formules :

- ①  $\exists x, \neg P(x, x)$
- ②  $\exists x, \forall y, P(x, y)$
- ③  $\forall x, \exists y, P(x, y)$
- ④  $\forall x y, P(x, y) \Rightarrow P(y, x)$

Donner la valeur des formules pour chaque interprétation de  $P$  :


(1)


(2)


(3)


(4)

## Exemple des rationnels

- signature pour manipuler des rationnels
  - constantes 0, 1 et -1,
  - une opération binaire de construction `frac`,
  - des opérations binaires `+` et `*`
  - un symbole de prédicat binaire pour l'égalité.
- interprétation de cette signature : représentation d'un rationnel
  - un couple  $(p, q) \in \mathbb{Z} \times \mathbb{N} \setminus \{0\}$  formé d'un entier relatif en numérateur et d'un entier naturel non nul en dénominateur.
  - Le domaine de l'interprétation est donc  $\mathcal{D} \stackrel{\text{def}}{=} \mathbb{Z} \times \mathbb{N} \setminus \{0\}$ .
- Autres domaines possibles
  - imposer que la fraction soit réduite (pas de diviseur commun du dénominateur et du numérateur),
  - autoriser un entier relatif en dénominateur

# Interprétation de chaque symbole

$$\begin{aligned}0_{\mathbb{Q}} &\stackrel{\text{def}}{=} (0, 1) \\1_{\mathbb{Q}} &\stackrel{\text{def}}{=} (1, 1) \\-1_{\mathbb{Q}} &\stackrel{\text{def}}{=} (-1, 1) \\\text{frac}_{\mathbb{Q}}((p_1, q_1), (p_2, q_2)) &\stackrel{\text{def}}{=} (p_1 q_2, q_1 p_2) && \text{si } p_2 > 0 \\\text{frac}_{\mathbb{Q}}((p_1, q_1), (p_2, q_2)) &\stackrel{\text{def}}{=} (0, 1) && \text{si } p_2 = 0 \\\text{frac}_{\mathbb{Q}}((p_1, q_1), (p_2, q_2)) &\stackrel{\text{def}}{=} (-p_1 q_2, -q_1 p_2) && \text{si } p_2 < 0 \\+_{\mathbb{Q}}((p_1, q_1), (p_2, q_2)) &\stackrel{\text{def}}{=} (p_1 q_2 + p_2 q_1, q_1 q_2) \\*_{\mathbb{Q}}((p_1, q_1), (p_2, q_2)) &\stackrel{\text{def}}{=} (p_1 p_2, q_1 q_2) \\=_{\mathbb{Q}}((p_1, q_1), (p_2, q_2)) &\stackrel{\text{def}}{=} (p_1 q_2 = q_1 p_2)\end{aligned}$$

- Aux tables primitives de la BD correspondent des symboles de prédicat (l'arité est le nombre de colonnes)
- Chaque état de la base de données correspond à une interprétation particulière des symboles.



# Cas propositionnel

- Dans le cas propositionnel (sans quantificateur), il n'y a pas de symboles de fonctions et tous les symboles de prédicats sont d'arité 0, donc des variables propositionnelles.
- Une interprétation revient donc à fixer une valeur booléenne pour chacune de ces variables.
- S'il y a  $n$  variables propositionnelles alors il y a  $2^n$  interprétations possibles.
- On peut les énumérer toutes et calculer la valeur de vérité de la formule propositionnelle pour chacune de ces interprétations, on retrouve ainsi les tables de vérité.

# Interprétation des symboles de prédicats

- Symbole de prédicat d'arité 0 : vrai, ou faux (*barrière\_ouverte*)
- Symbole de prédicat d'arité 1 : sous-ensemble de  $\mathcal{D}$  (les objets qui vérifient le prédicat).
- Symbole  $R$  de prédicat d'arité 2 : relation binaire sur  $\mathcal{D}$ .  
Représentation par un graphe (fini ou infini) : sommets éléments de  $\mathcal{D}$ , arête entre deux sommets  $a$  et  $b$  ssi  $R_I(a, b)$  est vraie.  
Représentation par une matrice carrée sur les sommets à valeur dans  $\{0, 1\}$ .
- Prédicat  $R$  d'arité plus grande, par exemple 4 alors l'interprétation est un ensemble de quadruplets  $(a, b, c, d)$ .  
Si l'ensemble est fini, on peut utiliser une table avec 4 colonnes.  
Les lignes de la table contiennent les quadruplets  $(a, b, c, d)$  pour lesquels l'interprétation  $R_I(a, b, c, d)$  est vraie.

La vérité d'une formule qui contient des variables libres dépend de la *valeur des variables*

**Exemple** Dans le modèle usuel des entiers :  $\exists n, \text{pair}(3 \times n + 1)$  est vrai,  $\forall n, \text{pair}(3 \times n + 1)$  est faux mais la valeur de vérité de  $\text{pair}(3 \times n + 1)$  dépend de la valeur de  $n$ .

## Definition (Environnement)

Soit  $\mathcal{X}$  l'ensemble des variables d'objets. Soit  $I$  une interprétation de la signature  $(\mathcal{F}, \mathcal{R})$  dont le domaine est  $\mathcal{D}$ , un **environnement** est une application  $\rho$  qui associe une valeur du domaine  $\mathcal{D}$  à chaque variable de  $\mathcal{X}$  (c'est à dire  $\rho : \mathcal{X} \rightarrow \mathcal{D}$ ).

Soit  $\rho$  un environnement, si  $x \in \mathcal{X}$  et  $d \in \mathcal{D}$ , on note  $\rho + \{x \mapsto d\}$  l'environnement qui vaut  $d$  pour la variable  $x$  et  $\rho(y)$  pour toutes les variables  $y \neq x$ .

# Valeur d'un terme dans une interprétation

- une signature  $(\mathcal{F}, \mathcal{R})$ , un ensemble de variables  $\mathcal{X}$
- une interprétation  $I \stackrel{\text{def}}{=} (\mathcal{D}, (f_I)_{f \in \mathcal{F}}, (R_I)_{R \in \mathcal{R}})$

## Definition (Valeur d'un terme)

Soit  $\rho$  un environnement,  $\rho \in \mathcal{X} \rightarrow \mathcal{D}$ .

On définit la valeur  $\text{val}_I(\rho, t)$  d'un terme  $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$  dans l'interprétation  $I$  et l'environnement  $\rho$ , c'est un élément du domaine  $\mathcal{D}$  :

$$\text{val}_I(\rho, x) = \rho(x) \quad \text{val}_I(\rho, f(t_1, \dots, t_n)) = f_I(\text{val}_I(\rho, t_1), \dots, \text{val}_I(\rho, t_n))$$

# Valeur d'une formule dans une interprétation

## Definition (Valeur d'une formule)

Soit  $\rho$  un environnement,  $\rho : \mathcal{X} \rightarrow \mathcal{D}$ .

On définit la valeur  $\text{val}_I(\rho, P)$  d'une formule  $P$  dans l'interprétation  $I$  et l'environnement  $\rho$ , c'est une valeur de vérité dans  $\{V, F\}$  :

$\text{val}_I(\rho, \perp)$	$= F$
$\text{val}_I(\rho, \top)$	$= V$
$\text{val}_I(\rho, R(t_1, \dots, t_n))$	$= \text{si } R_I(\text{val}_I(\rho, t_1), \dots, \text{val}_I(\rho, t_n)) \text{ alors } V \text{ sinon } F$
$\text{val}_I(\rho, \neg A)$	$= \text{si } \text{val}_I(\rho, A) = V \text{ alors } F \text{ sinon } V$
$\text{val}_I(\rho, A \wedge B)$	$= \text{si } \text{val}_I(\rho, A) = V \text{ alors } \text{val}_I(\rho, B) \text{ sinon } F$
$\text{val}_I(\rho, A \vee B)$	$= \text{si } \text{val}_I(\rho, A) = V \text{ alors } V \text{ sinon } \text{val}_I(\rho, B)$
$\text{val}_I(\rho, A \Rightarrow B)$	$= \text{si } \text{val}_I(\rho, A) = V \text{ alors } \text{val}_I(\rho, B) \text{ sinon } V$
$\text{val}_I(\rho, \forall x, A)$	$= \text{si pour tout } d \in \mathcal{D}, \text{val}_I(\rho + \{x \mapsto d\}, A) = V$ $\text{alors } V \text{ sinon } F$
$\text{val}_I(\rho, \exists x, A)$	$= \text{si il existe } d \in \mathcal{D} \text{ tel que } \text{val}_I(\rho + \{x \mapsto d\}, A) = V$ $\text{alors } V \text{ sinon } F$



*That's all Folks!*