

# Eléments de logique pour l'informatique

*Christine Paulin-Mohring*

Faculté des Sciences d'Orsay – Université Paris-Saclay

## Résumé

Ce document constitue les notes du cours proposé aux étudiants de troisième année des Licences de la mention Informatique (parcours Informatique et parcours MIAE) et de la mention Informatique, Mathématiques (parcours Informatique et parcours Informatique, Mathématiques) sur la période 2020-2025.

L'objectif est de donner les éléments pour comprendre et savoir utiliser le langage de la logique du premier ordre (syntaxe et sémantique) et de présenter les principes de base de méthodes de démonstration automatique.

Les exercices présentés dans ces notes sont corrigés en annexe (voir page 113). Un index des principales définitions est également donné en page 112.

# Table des matières

<b>Introduction</b>	<b>3</b>
Autour de la logique . . . . .	3
Organisation du cours . . . . .	6
<b>1 Maîtriser le langage logique</b>	<b>9</b>
1.1 Définition du langage . . . . .	9
1.1.1 Objets . . . . .	9
1.1.2 Formules atomiques . . . . .	11
1.1.3 Formules complexes . . . . .	12
1.1.4 Traduire des énoncés en formule logique . . . . .	14
1.2 Structure des formules . . . . .	16
1.2.1 Représentation des formules comme des arbres . . . . .	16
1.2.2 Notations, règles de parenthésage . . . . .	16
1.2.3 Variables libres, liées . . . . .	17
1.3 Formule vraie . . . . .	18
1.3.1 Le cas propositionnel . . . . .	18
1.3.2 Formule avec quantificateurs . . . . .	21
1.4 Théories et modélisation . . . . .	22
1.4.1 Définitions autour des théories . . . . .	22
1.4.2 Exemples de théories . . . . .	23
1.5 Définition récursive sur les formules . . . . .	27
1.5.1 Définir une fonction . . . . .	27
1.5.2 Raisonner sur les formules . . . . .	28
1.5.3 Définition récursive sur les termes . . . . .	29
<b>2 Donner du sens aux formules</b>	<b>33</b>
2.1 Interprétations et vérité . . . . .	33
2.1.1 Définitions . . . . .	33
2.1.2 Cas propositionnel . . . . .	34
2.1.3 Valeur de vérité d'une formule . . . . .	34
2.1.4 Lien entre substitution et valeur de vérité . . . . .	37
2.2 Validité et satisfiabilité . . . . .	38
2.2.1 Définitions . . . . .	39
2.2.2 Substitution et validité . . . . .	39
2.3 Conséquence logique, équivalence . . . . .	42
2.3.1 Propriétés de la conséquence logique . . . . .	42
2.3.2 Equivalences remarquables . . . . .	44
2.4 Modèles particuliers . . . . .	45
2.4.1 Modèle fini . . . . .	46

2.4.2	Modèle de Herbrand . . . . .	47
<b>3</b>	<b>Manipuler les formules de la logique</b>	<b>51</b>
3.1	Formes normales . . . . .	51
3.1.1	Forme normale de négation . . . . .	51
3.1.2	Clauses . . . . .	53
3.1.3	Formes normales conjonctives et disjonctives . . . . .	54
3.1.4	Skolémisation . . . . .	60
3.2	Représentations alternatives des formules . . . . .	64
3.2.1	Restrictions sur les connecteurs . . . . .	64
3.2.2	Connecteurs alternatifs . . . . .	64
3.2.3	Diagrammes de décision binaire . . . . .	65
3.3	Systèmes de déduction . . . . .	69
3.3.1	Preliminaires . . . . .	69
3.3.2	Système de Hilbert . . . . .	74
3.3.3	Déduction naturelle . . . . .	76
3.3.4	Calcul des séquents multi-conclusions . . . . .	81
<b>4</b>	<b>Automatiser les démonstrations</b>	<b>88</b>
4.1	Peut-on automatiser les démonstrations ? . . . . .	88
4.1.1	Décidabilité de la logique propositionnelle . . . . .	88
4.1.2	Semi-décidabilité de la logique du premier ordre . . . . .	89
4.2	Cas propositionnel . . . . .	91
4.2.1	Système G . . . . .	91
4.2.2	Résolution propositionnelle . . . . .	93
4.2.3	Satisfiabilité . . . . .	96
4.3	Unification . . . . .	97
4.3.1	Définitions . . . . .	98
4.3.2	Ensemble des solutions . . . . .	99
4.3.3	Ordre sur les termes et filtrage . . . . .	100
4.4	Résolution générale . . . . .	101
4.4.1	Résolution au premier ordre . . . . .	101
4.4.2	Questions de stratégie . . . . .	105
4.5	Calcul des séquents . . . . .	107
4.6	Théories décidables . . . . .	109
	<b>Conclusion</b>	<b>111</b>
	<b>Index</b>	<b>112</b>
<b>A</b>	<b>Correction des exercices du cours</b>	<b>114</b>
A.1	Exercices du chapitre 1 . . . . .	114
A.2	Exercices du chapitre 2 . . . . .	120
A.3	Exercices du chapitre 3 . . . . .	124
A.4	Exercices du chapitre 4 . . . . .	132

# Introduction

## Autour de la logique

Cette section donne quelques éléments de repère sur la logique. Que recouvre ce terme, comment nous aborderons cette thématique dans le cours et les usages qui peuvent en être fait en particulier en informatique.

### La logique, chemin vers la vérité

Logique vient du grec logos (raison, *langage*, *raisonnement*).

**Énoncé** La logique manipule des *énoncés* (on parle de manière équivalente de *propriété*, ou de *formule*) qui sont des phrases auxquelles on peut attribuer un sens qui est une *valeur de vérité*, vrai ou faux. Les phrases suivantes sont des énoncés.

- “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”

Par contre la phrase “la couleur du cheval blanc d’Henri IV” *n’est pas un énoncé*. En effet cette phrase désigne une couleur et pas une valeur de vérité.

Le *langage* est essentiel pour préciser de quoi on parle. En effet, un énoncé peut être vrai ou faux suivant la manière dont on *interprète* les mots. La logique est une manière *scientifique* et systématique d’étudier la notion de vérité.

**Raisonnement** La logique ne s’intéresse pas seulement aux énoncés et à la notion de vérité. Elle concerne également la notion de raisonnement. Tout comme en informatique, il peut y avoir plusieurs manières d’arriver à un résultat.

Le raisonnement est un cheminement (une *déduction*) qui permet à partir d’énoncés que l’on suppose vrais (que l’on appelle des *hypothèses* ou encore des *axiomes*), de construire de nouveaux énoncés appelés *conséquences* ou *conclusion*. Un raisonnement suit un ensemble de règles logiques données.

**Exemple 1 (Raisonnements)** *Le raisonnement “classique” suivant est correct.*

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

*Le raisonnement suivant est par contre beaucoup plus douteux. Il part d’hypothèses possibles pour aboutir à une conclusion qui est fausse.*

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

Un raisonnement suit des *règles* précises. C'est un procédé suffisamment élémentaire pour *convaincre* que si les hypothèses sont vraies alors la conclusion que l'on a déduite est aussi vraie. Quelques exemples de raisonnement sont :

— modus ponens, preuve par l'absurde, preuve par récurrence ...

Montrer qu'un énoncé est vrai ou faux n'est pas en général *décidable* (il n'y a pas de programme qui répond à cette question). Vérifier qu'un raisonnement suit bien les règles du jeu peut en général s'effectuer mécaniquement. Certains raisonnements sont adaptés à l'humain, d'autres aux machines...

La logique étudie des *règles du jeu* qui permettent de construire des raisonnements logiques. On se pose des questions comme :

- avec quels objets joue-t-on ? quelles règles peut-on utiliser ?
- les règles du jeu sont-elles trop laxistes (on déduit des choses fausses)
- les règles du jeu sont-elles trop strictes (on n'arrive pas à prouver quelque chose qui est pourtant correct)
- peut-on changer les règles du jeu ? changer de style ?

**Chaos de la logique.** L'extrait suivant est tiré de la pièce de théâtre *Rhinoceros*, d'Eugène Ionesco. C'est un dialogue entre un Logicien et un Vieux Monsieur, qui sous couvert de raisonnement logique, énonce des absurdités.

- 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...
- Autre syllogisme : tous les chats sont mortels. Socrate est mortel. Donc Socrate est un chat.
- Et il a quatre pattes. C'est vrai, j'ai un chat qui s'appelle Socrate.
- Vous voyez...
- Socrate était donc un chat !
- La logique vient de nous le révéler.

**Démonstration.** Démontrer c'est apporter une *évidence* du fait que quelque chose est vrai. Il existe plusieurs sortes de justification, plus ou moins rigoureuses :

(<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 : *Vu 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, on s'intéresse à des méthodes rigoureuses mais aussi à des formats de preuve qui vont être transposables sur ordinateur.

**Une approche fondatrice pour les mathématiques.** Le questionnement sur les fondements des mathématiques (se donner un langage et un ensemble de règles du jeu acceptables par tous) s'est développé du début du

20ème siècle avec la théorie des ensembles. Quelques logiciens importants dans ce domaine sont Tarski, Russell, Hilbert et Gödel. Des résultats surprenants à l'époque ont été le paradoxe de Russell ainsi que le théorème d'incomplétude de Gödel.

Le *paradoxe de Russel* a montré qu'un ensemble d'énoncés de la théorie des ensembles proposé initialement comme fondement des mathématiques était *incohérent* c'est-à-dire qu'il permettait de déduire une *contradiction* et à partir de là n'importe quelle propriété. Il a fallu restreindre l'ensemble des énoncés, en particulier interdire l'existence d'un ensemble de tous les ensembles pour obtenir la théorie des ensembles de Zermelo-Fraenkel que l'on utilise de nos jours.

Le *théorème d'incomplétude de Gödel* répond négativement à la question de l'existence d'un ensemble d'hypothèses qui pourrait fonder les mathématiques. Le théorème dit que si un ensemble d'énoncés, vus comme des hypothèses, a des « bonnes » propriétés, comme d'être récursif (étant donné une formule de la logique, on peut décider si c'est ou non une des hypothèses), d'être *cohérent* (on ne peut pas déduire toutes les formules à partir de ces hypothèses) et d'être suffisamment expressif pour modéliser les entiers naturels, alors il existe une formule qui, dans ce système, ne peut ni être prouvée, ni être réfutée (c'est-à-dire que, à partir de ces hypothèses, on ne peut prouver ni la formule, ni la négation de cette formule).

En informatique, on peut définir de manière universelle l'ensemble des fonctions “calculables” et on connaît plusieurs systèmes pour les représenter (Machine de Turing, lambda-calcul, fonctions récursives). En logique, aucun système “raisonnable” n'est suffisamment puissant pour permettre de démontrer toutes les « vérités ».

**Meta-mathématique.** En logique mathématique, les formules et le raisonnement sont les objets de l'étude comme les nombres en arithmétique, les fonctions en analyse, les espaces vectoriels en algèbre linéaire.

On va raisonner sur les objets de base de la logique, à savoir les énoncés mathématiques et les raisonnements, et établir des théorèmes mathématiques sur ces objets : il y a *deux niveaux* différents qu'il ne faut pas confondre.

- La formule logique et le raisonnement, objets de l'étude et pour lesquels on utilisera des symboles spécifiques.
- Les théorèmes que l'on établira par des preuves sur ces objets pour lesquels on utilisera plutôt le langage naturel.

Cette situation se retrouve aussi en informatique où on écrit parfois des programmes qui manipulent d'autres programmes (*les compilateurs*).

**Logique et informatique.** Il existe des liens très étroits entre logique et informatique dont voici quelques illustrations :

- Le calcul booléen a des liens avec les circuits logiques (on peut utiliser des formules booléennes pour représenter des circuits et des circuits pour implanter des fonctions booléennes).
- Logique et informatique suivent une démarche analogue : machine pour calculer ou raisonner reposant sur un nombre limité d'opérations ; liens entre syntaxe (une suite de caractères répondant à des contraintes syntaxiques) et sémantique (le ou les sens possibles attribués à ces phrases).
- Dans le domaine de l'Intelligence Artificielle dite « symbolique » : on munit un ordinateur qui sait calculer de capacité de raisonnement, cela nécessite de transformer le raisonnement en calcul. L'Intelligence Artificielle actuelle utilise plutôt des méthodes d'apprentissage statistiques en s'appuyant sur des exemples. Néanmoins il existe un enjeu de savoir expliquer et justifier a posteriori les résultats, ce qui nous ramène à la logique.
- La logique est un outil de modélisation utilisé dans le domaine des bases de données et du développement de programmes (génie logiciel). La logique sert alors à décrire le comportement attendu des systèmes informatiques et vérifier la conformité des implémentations.

**Programmer la logique** La logique se programme sur ordinateur : on introduit alors des structures informatiques pour représenter des propriétés logiques et on construit des outils pour les manipuler.

On se pose alors des questions informatiques sur la logique auxquelles nous apporterons des réponses dans ce cours :

- 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 ?

## Organisation du cours

### Objectifs

L'objectif du cours est de se familiariser avec le formalisme logique de base, à savoir la logique du premier ordre. On introduira la notion de terme, de formule, de démonstration, de validité, le lien entre syntaxe et sémantique. Ce cours met en pratique des outils mathématiques utilisés en informatique tels que la récurrence structurelle et les définitions par système d'inférence.

Le programme du cours couvre les éléments suivants :

- Langage logique
- Système de règles logiques pour construire des preuves
- Termes : signature, preuve par récurrence structurelle, définition récursive de fonctions
- Calcul des prédicats : syntaxe, variables libres et liées, sémantique, équivalence
- Modèle, modèle de Herbrand
- Exemples de théories
- Démonstration automatique, formes normales, résolution, séquents

### Compétences attendues

#### Prérequis

- les bases du calcul booléen
- compréhension des formules et preuves mathématiques élémentaires (dont les preuves par récurrence)

#### Compétences logiques

- 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 terme logique
  - 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

#### Compétences générales

- Etre capable de présenter un raisonnement scientifiquement correct
- Apprendre un nouveau langage formel
- Distinguer syntaxe et sémantique
- Connaître et savoir construire des algorithmes sur des objets symboliques
- Méthodologie : mettre en pratique des outils 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

La correction des exercices qui illustrent le cours est donnée en annexe.

### Notations du cours

Le cours introduit de nouvelles notions qui constituent le vocabulaire de la logique, ces notions doivent être connues et comprises. Elles sont introduites en rouge souligné comme dans substitution. Ces notions font (en général) l’objet d’une définition et on les retrouve dans l’index du cours page page 112.

D’autres éléments du cours sont simplement “surlignés” par leur affichage en italique, pour ressortir dans le texte comme des éléments *importants* mais qui se réfèrent au langage courant sans faire l’objet d’une définition spécifique au cours.

# Bibliographie

- [1] Serenella Cerrito. Logique pour l'Informatique : une introduction à la déduction automatique. Vuibert Publisher Co, 2008.
- [2] Robert Cori and Daniel Lascar. Logique Mathématique. Axiomes. Masson, 1993.
- [3] René David, Karim Nour, and Christophe Raffalli. Introduction à la Logique, Théorie de la démonstration. Dunod, 2001.
- [4] 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.
- [5] Gilles Dowek. Les démonstrations et les algorithmes. Les éditions de l'Ecole Polytechnique, 2010.
- [6] Gilles Dowek. La logique. Le Pommier, 2015.
- [7] Yannis Delmas-Rigoutsos et René Lalement. La Logique ou l'art de raisonner. Le Pommier, 2000.

# Chapitre 1

## Maîtriser le langage logique

La *logique* s'intéresse à la notion de *vérité* et de *raisonnement*. Un raisonnement est un discours qui permet d'établir la *vérité* d'un fait. Dans ce cours, nous nous intéressons à la *logique mathématique* qui s'appuie sur un langage *formel* pour décrire les formules, une définition rigoureuse de la notion de vérité d'un énoncé et un ensemble de *règles* fixé pour effectuer un raisonnement logique. Une démonstration logique ne sera donc pas un discours en langue naturel mais une suite bien précise de constructions qui obéissent à des règles du jeu définies.

Dans ce chapitre nous introduisons le langage de la *logique du premier ordre*, appelée aussi *calcul des prédicats*.

### 1.1 Définition du langage

On utilisera un langage *formel* pour écrire les propositions afin d'éviter les ambiguïtés du langage naturel ou des notations imprécises. Dans le langage naturel, le même mot peut refléter des situations logiques différentes. On le voit dans les énoncés ci-dessous avec différents sens pour le connecteur *ou* qui a souvent dans le langage courant un sens exhaustif (l'un des deux mais pas les deux), ou encore dans la dernière phrase dans laquelle le concept d'être une étoile représente deux réalités différents (entre l'astre et la vedette).

- Je peux t'offrir de l'eau *ou* du vin
- Le menu propose fromage *ou* dessert
- Demain, il y aura de la pluie *ou* du soleil
- Maryline Monroe *et* le soleil sont des *étoiles*

Un langage formel présente également moins de redondance que le langage naturel dans lequel la même situation logique peut s'exprimer de plusieurs manières différentes. Les langages formels sont donc plus simples à étudier et à représenter en machine.

#### 1.1.1 Objets

Une *formule* décrit une propriété (vraie ou fausse) qui en général va parler d'*objets* (on appellera *termes* les expressions qui représentent les objets), par exemple des entiers, des figures géométriques, mais aussi des ensembles, des fonctions, des éléments dans une base de données ...

Pour décrire les objets, on introduira un langage spécifique formé de *symboles* qui sont juste des suites de caractères spécifiques. Chaque symbole utilisé dans le langage des objets est défini avec une *arité* qui est un entier naturel représentant le nombre d'arguments qu'il faut associer au symbole pour représenter un nouvel objet.

Lorsqu'un symbole est d'arité 0, on dit que c'est une *constante*. Cela représente donc un objet sans nécessité d'argument supplémentaire. Ce sont les objets de base. Par exemple on peut introduire 0, 1,  $\emptyset$  (notation pour l'ensemble vide) comme des constantes particulières.

Un symbole d'arité 1 est dit symbole **unaire**. Le symbole lui-même ne représente pas un objet, par contre, si on l'associe à un terme (on dit parfois que l'on applique le symbole au terme), alors on forme ainsi un nouveau terme qui représente un objet. On peut par exemple introduire un symbole unaire **sqrt** pour représenter l'opération de racine carrée. On pourra alors former un nouveau terme **sqrt(1)**. Attention le terme **sqrt(1)** est différent du terme **1**. Dans le cas des ensembles, on peut introduire un symbole unaire **P** pour représenter l'ensemble des parties. Le terme **P(∅)** représente l'ensemble des parties de l'ensemble vide (qui est un ensemble qui contient un unique élément).

Un symbole d'arité 2 est dit symbole **binaire**, c'est le cas des symboles pour représenter les opérations arithmétiques **+**, **×** ou bien l'union de deux ensembles. Il faut commencer par construire deux termes avant de pouvoir les associer au symbole binaire pour construire un nouveau terme.

**Définition 1.1.1 (Signature)** Une signature est un ensemble de symboles  $\mathcal{F}$  chacun associé à une 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.

Le langage des objets utilise également des **variables d'objets** qui sont aussi des symboles qui représentent des objets, mais contrairement aux constantes, elles ne désignent pas un objet particulier, mais sont une manière de nommer un objet arbitraire comme par exemple dans l'expression  $x + 1$ .

**Définition 1.1.2 (Terme)** Étant 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  $t_1, \dots, t_n$ . On dit alors que  $f$  est le **symbole de tête** et que  $t_1, \dots, t_n$  sont les sous-termes directs du terme  $t$ .

On note  $\mathcal{T}(\mathcal{F}, \mathcal{X})$  l'ensemble des termes sur la signature  $\mathcal{F}$  et l'ensemble des variables  $\mathcal{X}$ . On notera  $\mathcal{T}(\mathcal{F})$  le sous-ensemble de ces termes qui ne contiennent pas de variable, appelés aussi **termes clos**.

La manière d'écrire le terme peut varier suivant les systèmes mais la structure du terme (variable ou symbole associé à 0, 1 ou plusieurs termes) reste la même. Si  $f$  est un symbole de fonction binaire et que  $t$  et  $u$  sont des termes, le terme obtenu avec comme symbole de tête  $f$  et comme sous-termes  $t$  et  $u$  se notera par défaut  $f(t, u)$ . Pour quelques symboles spécifiques, on pourra utiliser une notation dite **infixe** qui consiste à placer le symbole principal entre les deux termes comme par exemple pour l'addition  $t + u$ .

### Exemples de signature d'objets

- Pour représenter les entiers naturels, on peut se donner comme constantes **0** et **1** et comme opérations binaires l'addition **+** et la multiplication **×**. L'entier **3** pourra alors être représenté par le terme  $1 + 1 + 1$ . Il n'y a pas unicité de la représentation, par exemple **1** peut se représenter (entre autres) par les termes **1** ou  $1 \times 1$  ou  $1 + 0$ .
- Pour représenter des mots binaires (formés de **0, 1**) de longueur arbitraire, on peut se donner une constante  $\epsilon$  pour représenter le mot vide et deux fonctions unaires  $c_0$  et  $c_1$  pour représenter l'ajout d'un **0** ou d'un **1** en tête du mot.  
Ainsi le mot **1011** sera représenté par le terme  $c_1(c_0(c_1(c_1(\epsilon))))$ .  
D'autres opérations peuvent être ajoutées comme la concaténation de deux mots ou le décalage vers la gauche ou la droite ainsi qu'un prédicat d'égalité.
- Si on veut représenter uniquement des mots binaires sur 4 bits, on peut se donner une signature avec les constantes **0** et **1** et une fonction de construction  $P$  d'arité 4. Le mot **1011** sera représenté par le terme  $P(1, 0, 1, 1)$ . On voit sur cet exemple que la signature permet aussi de représenter des termes qui ne correspondent pas à des mots de 4 bits comme **0** et **1** mais aussi  $P(P(1, 1, 0, 0), 0, 1, 1)$ . L'univers des termes est plus gros que l'ensemble des objets que l'on veut modéliser. On pourra alors utiliser la logique pour caractériser l'ensemble des objets intéressants. Par exemple ici une propriété **bit(b)** qui est vraie lorsque  $b$  est égal à **0** ou **1** et une propriété **mot(m)** qui sera vraie si  $m$  est de la forme  $P(b_1, b_2, b_3, b_4)$  et que chaque  $b_i$  vérifie la propriété **bit(b<sub>i</sub>)**.

### 1.1.2 Formules atomiques

Les **formules atomiques** représentent une notion qui peut être vraie ou fausse et que l'on ne peut pas décomposer sur une base logique. Il s'agit plutôt de propriétés que l'on observe plutôt que des formules logiques sur lesquelles on peut raisonner.

Parmi les formules atomiques, on distingue deux objets spéciaux : la formule notée  $\top$  (aussi appelé **formule vraie**, ou **tautologie**) qui est toujours vraie et la formule  $\perp$  qui est toujours fausse, aussi appelée **contradiction**.

Les formules atomiques qui ne sont pas  $\top$  ou  $\perp$  sont construites à partir de **symboles de prédicat**. Elles servent à représenter des propriétés de base des objets.

Un **symbole de prédicat** est juste une suite de caractères à laquelle on va pouvoir attribuer ultérieurement un sens. Comme pour les symboles d'objets, on associe à un symbole de prédicat un entier naturel que l'on appelle une **arité**. Cet entier correspond au nombre d'arguments représentant des objets qu'il faudra associer au symbole pour en faire une formule. On peut faire une analogie entre un symbole de prédicat et une procédure en programmation à laquelle on passe des arguments avant de pouvoir l'exécuter.

On dispose souvent (mais pas tout le temps) d'un symbole de prédicat binaire (arité 2) pour représenter l'égalité. On note de manière **infixe**  $t = u$ , le symbole d'égalité appliqué aux deux termes  $t$  et  $u$ .

**Définition 1.1.3 (Formule atomique)** Une **formule atomique** est soit un des symboles  $\top$  ou  $\perp$ , soit un symbole de prédicat associé à un ou plusieurs objets (le nombre d'objets attendu correspond à l'**arité** du prédicat).

La plupart des modélisations utilisent un symbole de prédicat binaire  $=$  qui modélise le fait que deux objets sont égaux, c'est-à-dire qu'ils vont représenter la même entité.

#### Exemple 1.1 (Egalité)

On se donne un langage dans lequel on a des constantes **0**, **1**, **2** et **4** ainsi qu'un symbole de fonction binaire  $+$  et un symbole de prédicat binaire  $=$ .

- Les formules suivantes sont atomiques, le symbole de prédicat utilisé est l'égalité qui est un prédicat binaire (d'arité 2) :  $0 = 1$ ,  $2 + 2 = 4$ ,  $x = x$ ,  $x = y$
- Par contre la formule  $0 \neq 1$  qui sera définie comme la négation de  $0 = 1$  n'est pas une formule atomique.

Lorsqu'on modélise des systèmes d'information, l'organisation en ensembles et en tables sera représentée logiquement par des symboles de prédicat.

**Exemple 1.2 (Systèmes d'information)** Une table à  $n$  colonnes dans une base de données relationnelle peut se modéliser en logique comme un symbole de prédicat d'arité  $n$ .

- La formule atomique  $\text{Etudiant}(\text{Durand}, \text{Bob}, 347890, 01/01/1990)$  correspondra à l'existence d'une entrée  $(\text{Durand}, \text{Bob}, 347890, 01/01/1990)$  dans la table "**Etudiant**".
- Un système qui modélise des trajets de bus distinguera dans ses données des chaînes de caractères qui représentent indifféremment des lignes, des arrêts et des horaires. On pourra introduire trois prédicats unaires **ligne**, **arrêt** et **horaire** pour séparer les objets de la logique suivant leur catégorie. Pour modéliser une 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, on peut introduire un prédicat **trajet** d'arité 4. La formule atomique  $\text{trajet}()$  représente le fait qu'un bus de numéro **91.06** effectue un trajet de **Massy-Palaiseau** gare à **Christ de Saclay** à **6 : 00**.

Un symbole de prédicat est juste un nom (on dit que c'est de la **syntaxe**). La **sémantique** lui attribue un **sens** en lui associant une relation mathématique entre les objets modélisés. Il y a parfois un sens implicite (par exemple l'égalité ou l'ordre sur les entiers) mais d'un point de vue logique, de nombreuses interprétations sont possibles.

L'arité d'un symbole de prédicat peut être **0**. Le symbole de prédicat représente alors à lui tout seul une formule vraie ou fausse. On parle alors de **variable propositionnelle**. On verra que c'est un cas important, car avec

seulement des variables propositionnelles, la logique (que l'on appelle alors **logique propositionnelle** ou calcul propositionnel) devient décidable. De nombreux problèmes peuvent se modéliser en logique propositionnelle, quitte à introduire des milliers de variables propositionnelles et de formules.

Les symboles se choisissent en fonction du problème à modéliser et il y a souvent plusieurs manières de faire.

**Exemple 1.3 (Modélisation)** Si dans un problème on cherche à modéliser la situation *Paul est mon ami*, alors il suffit d'introduire une variable propositionnelle (qui pourra être vraie ou fausse) et qui représente le fait que Paul est (ou non) mon ami. Si par contre on veut modéliser une phrase comme « *tous mes amis habitent Paris* », alors il faut un symbole de prédicat unaire *ami* tel que *ami(t)* est vrai lorsque *t* est mon ami et un symbole de prédicat unaire *paris* tel que *paris(t)* est vrai lorsque *t* habite Paris, en introduisant en plus une constante *Paul* on peut facilement traduire la notion *Paul est mon ami*. Si maintenant la propriété à montrer est « *les amis de mes amis sont mes amis* » alors il faudra passer à un symbole de relation binaire tel que *amis(t, u)* est vrai si *t* et *u* sont amis et utiliser une constante *self* pour représenter la personne qui s'exprime.

Etant donnée une situation à représenter en logique, il peut être demandé de la modéliser et donc de trouver les bons symboles à introduire pour pouvoir représenter le problème. Dans d'autres situations, le langage peut être imposé (comme lorsque on doit utiliser l'interface d'une bibliothèque donnée) et il faudra se limiter aux symboles autorisés.

**Notation infixe** La notation standard pour un symbole *f* d'arité au moins 1 (symbole de fonction ou de prédicat) est  $f(t_1, \dots, t_n)$ .

Pour certains symboles binaires (d'arité 2), on utilise parfois plutôt une notation **infixe**, c'est-à-dire que l'on écrira  $t \circ u$  plutôt que  $f(t, u)$ . Par exemple :  $t + u$ ,  $t \times u$  pour des termes et  $t = u$ ,  $t \leq u$  pour des formules atomiques.

### 1.1.3 Formules complexes

Les **formules complexes** se construisent à partir des formules atomiques à l'aide de **connecteurs logiques**. On distingue la partie dite **propositionnelle** :

- $\neg P$  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**"

on ajoute 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**"

La quantification universelle peut se voir comme une conjonction généralisée sur tous les *x* possibles (il peut y en avoir une infinité, pas forcément dénombrable). La quantification existentielle peut se voir comme une disjonction généralisée, elle aussi possiblement infinie.

**Exemple 1.4 (Formules complexes)** On se place dans un langage dans lequel on a un prédicat unaire *yeux-bleus*. La formule atomique *yeux-bleus(x)* représente le fait que l'objet *x* a les yeux bleus. On introduit également une variable propositionnelle *A*.

- tout le monde a les yeux bleus :  $\forall x, \text{yeux-bleus}(x)$
- il existe une personne qui a les yeux bleus :  $\exists x, \text{yeux-bleus}(x)$
- il existe une personne telle que si cette personne a les yeux bleus alors tout le monde a les yeux bleus :  $\exists x, (\text{yeux-bleus}(x) \Rightarrow \forall y, \text{yeux-bleus}(y))$ <sup>1</sup>

1. Cette affirmation peut sembler paradoxale, c'est pourtant une propriété toujours vraie. En effet soit tout le monde a les yeux bleus et l'affirmation est vraie, soit il existe une personne qui n'a pas les yeux bleus, que l'on nomme *a*. La propriété  $\text{yeux-bleus}(a) \Rightarrow \forall y, \text{yeux-bleus}(y)$  est vraie car de la forme  $A \Rightarrow B$  avec *A* est faux. On en déduit le résultat.

- S'il existe une personne qui a les yeux bleus alors tout le monde a les yeux bleus :  $(\exists x, \text{yeux-bleus}(x)) \Rightarrow (\forall y, \text{yeux-bleus}(y))$ <sup>2</sup>
- tiers exclu :  $A \vee \neg A$  (soit une formule est vraie, soit sa négation est vraie)
- élimination des doubles négations :  $(\neg \neg A) \Rightarrow A$ , (une formule dont la négation est fausse est vraie)
- raisonnement par l'absurde  $(\neg A \Rightarrow A) \Rightarrow A$  (si  $A$  est vrai sous l'hypothèse  $\neg A$  alors  $A$  est vrai sans hypothèse)

**Exemple 1.5 (Formules paramétrées)** Les formules logiques permettent aussi d'énoncer des propriétés d'objets. On se place dans un langage qui contient les constantes 1 et 2, les opérations binaires + et \* et le symbole d'égalité.

- on peut exprimer le fait que l'entier  $x$  est impair à l'aide de la formule logique  $\exists y, x = 2 \times y + 1$ . Cette formule n'est pas a priori vraie ou fausse, cela dépendra de la valeur que l'on donne à  $x$ .
- De même on peut exprimer le fait que l'entier  $x$  est pair à l'aide de la formule logique  $\exists y, x = 2 \times y$ .
- A l'aide de ces définitions, on pourra exprimer la propriété qui dit qu'un entier qui n'est pas pair est impair :  $\forall x, ((\forall y, \neg(x = 2 \times y)) \Rightarrow (\exists y, x = 2 \times y + 1))$

**Notations** On pourrait ajouter le connecteur logique d'équivalence dans la définition des formules complexes. Nous utiliserons plutôt la notation  $A \Leftrightarrow B$  comme un raccourci pour la formule  $(A \Rightarrow B) \wedge (B \Rightarrow A)$ .

On peut introduire plusieurs variables dans un quantificateur, par exemple :  $\forall x y, P$  représente la formule  $\forall x, \forall y, P$ .

En mathématique usuelle, on utilise parfois d'autres notations logiques comme  $\exists! x, A$  pour « il existe un et un seul  $x$  qui vérifie  $A$  » ou encore  $\forall x \in E, A$  (la propriété  $A$  est vérifiée pour tous les objets  $x$  dans  $E$ ) et  $\exists x \in E, A$  (il existe un objet  $x$  dans  $E$  qui vérifie la propriété  $A$ ).

Néanmoins ces notations logiques ne sont pas primitives et cachent des constructions plus complexes. L'unicité par exemple présuppose l'existence d'un symbole d'égalité qui nous permet de dire si deux termes représentent le même objet (et donc de compter). La restriction d'un quantificateur sur un ensemble demande une modélisation des ensembles et de la notion d'appartenance, de plus elle se traduit par une implication dans le cas du quantificateur universel et par une conjonction dans le cas existentiel. Les exercices 1.1 et 1.3 donnent des exemples. Ces notations de haut niveau ne feront pas partie a priori du langage logique utilisé dans ce cours.

**Différentes catégories syntaxiques** Dans le langage de la logique, on distingue deux catégories syntaxiques, les termes et les formules. En particulier, il y aura des symboles de fonctions (comme 0 ou +) qui permettent de construire des termes et des symboles de prédicat pour construire les formules. La catégorie des termes (expressions qui représentent des objets) se construit à partir des données suivantes

- variables (objets) :  $\mathcal{V}$
- symboles de fonctions
  - constantes d'arité 0 :  $\mathcal{C}$
  - fonctions d'arité au moins 1 :  $\mathcal{F}$
  - symboles binaires infixes :  $\mathcal{F}_I$

et se décrit syntaxiquement par les règles :

$$\begin{aligned} \text{term} &\equiv \mathcal{V} \mid \mathcal{C} \mid \mathcal{F}(\text{list-terms}) \mid (\text{term } \mathcal{F}_I \text{ term}) \\ \text{list-terms} &\equiv \text{term} \mid \text{list-terms, term} \end{aligned}$$

La catégorie des formules logiques se construit à partir des symboles de prédicats

- d'arité 0 :  $\mathcal{X}$  (variables propositionnelles, à ne pas confondre avec les variables d'objets qui apparaissent dans les quantificateurs notées ici  $\mathcal{V}$ ).

2. Noter la différence de parenthésage avec la phrase précédente. Cette nouvelle affirmation, contrairement à la précédente n'est pas toujours vérifiée.



- d'arité au moins 1 :  $\mathcal{P}$
- symboles binaires infixes :  $\mathcal{P}_I$

et se décrit syntaxiquement par les règles :

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

Les règles précédentes correspondent à une représentation arborescente des termes et des formules sur laquelle nous reviendrons dans la section 1.2. Lorsqu'on écrit des formules comme des suites de caractère, il est nécessaire d'introduire des parenthèses de manière à délimiter les constructions et à éviter les ambiguïtés. Cela doit être pris en compte si on écrit un programme qui "lit" des formules logiques entrées de manière séquentielle par l'utilisateur pour les représenter ensuite en machine comme des arbres.

L'accumulation des parenthèses alourdit parfois les notations et des conventions permettent d'en éliminer un grand nombre. On choisit souvent une présentation ambiguë de la syntaxe, sans mettre systématiquement de parenthèses autour des opérateurs mais en ajoutant la possibilité de parenthéser n'importe quelle sous-expression sans en altérer le sens.

La description syntaxique des termes et des formules devient alors :

$$\begin{aligned} \text{term} \Rightarrow & \mathcal{V} \mid \mathcal{C} \mid \mathcal{F}(\text{list-terms}) \mid \text{term } \mathcal{F}_I \text{ term} \mid (\text{term}) \\ \text{list-terms} \Rightarrow & \text{term} \mid \text{list-terms}, \text{term} \end{aligned}$$

Si les termes n'utilisent que la notation préfixe pour les symboles alors il n'y a pas besoin de parenthèses additionnelles au niveau des termes. Par contre si certains symboles utilisent une notation infix, il faudra aussi autoriser des termes parenthésés, par exemple pour lever une possible ambiguïté sur un terme comme  $x - y - z$  qui pourrait se lire comme  $(x - y) - z$  ou comme  $x - (y - z)$ .

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

**Calcul des prédicats et calcul propositionnel.** 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 (variables propositionnelles) et dans lequel on n'utilise pas de quantificateurs.

Il existe beaucoup d'autres logiques comme des logiques multi-sortes dans lesquelles les termes sont séparés dans des classes différentes (à la manière des types dans les langages de programmation) ou encore des logiques d'ordre supérieur dans lesquels on peut aussi quantifier sur les formules ou encore des logiques temporelles qui intègrent des quantificateurs spéciaux pour raisonner sur le comportement de systèmes. La logique du premier ordre est le socle de référence de toutes ces logiques et l'outil de base pour le raisonnement mathématiques.

### 1.1.4 Traduire des énoncés en formule logique

On suppose que l'on se place dans un langage avec les prédicats  $\text{amis}(x, y)$  qui représente le fait que  $x$  et  $y$  sont amis et une constante  $\text{self}$  qui représente l'individu qui s'exprime. On cherche à traduire en une formule logique la phrase « les amis de mes amis sont mes amis ». On peut procéder par étape



1. identifier les différentes entités qui sont mentionnées dans la phrase et leur faire correspondre un terme. Ici, il y a moi, mes amis et les amis de ces amis. Moi est représenté par la constante `self`, mes amis sont une entité variable que l'on choisit d'appeler `x` et les amis de mes amis sont une autre entité variable que l'on choisit d'appeler `y`.
2. identifier les formules atomiques en jeu.  
Ici, le fait que `x` représente mes amis se traduit par la formule atomique `amis(x, self)` le fait que `y` représente les amis de `x` se traduit par `amis(y, x)`. Le fait que les amis de mes amis soient mes amis se traduit par `amis(y, self)`.  
Ces formules atomiques devront apparaître dans la formule finale.
3. identifier les liens logiques entre les formules atomiques. Ici on a une conséquence "si `x` est mon ami et `y` et `x` sont amis alors `y` est mon ami". Donc la phrase peut se traduire par

$$(\text{amis}(x, \text{self}) \wedge \text{amis}(y, x)) \Rightarrow \text{amis}(y, \text{self})$$

4. Il reste à lier les variables `x` et `y` par des quantificateurs, ici il s'agit bien d'exprimer une propriété pour n'importe quel `x` et n'importe quel `y`. Ce qui donne au final

$$\forall x y, ((\text{amis}(x, \text{self}) \wedge \text{amis}(y, x)) \Rightarrow \text{amis}(y, \text{self}))$$

5. On peut procéder à une étape de vérification qui consiste à relire (sans a priori) la formule logique proposée et à se convaincre qu'elle représente la même vérité que la phrase qu'il était demandé de traduire.

**Exercice 1.1** On ajoute au langage précédent le prédicat `joue(x, y)` qui représente le fait que `x` joue avec `y` et un prédicat d'égalité. Traduire en formules logiques les expressions suivantes :

1. je ne joue qu'avec mes amis
2. tout le monde a un ami
3. je n'ai pas d'ami avec qui jouer
4. j'ai au moins deux amis (on utilisera la relation d'égalité)

**Exercice 1.2** On se place dans un langage qui contient un symbole de fonction binaire `+` qui représente l'addition, un symbole de prédicat unaire `pair` tel que `pair(x)` exprime que l'entier `x` est pair.

- Écrire la formule qui dit que la somme de deux entiers pairs est un entier pair. On fera attention à bien distinguer les expressions qui représentent des objets (termes) de celles qui représentent des énoncés (formules).

**Exercice 1.3** En logique du premier ordre, lorsque l'on écrit la formule  $\forall x, P$  ou  $\exists x, P$  on englobe tous les objets possibles (sans préciser dans quel univers se placent ces objets). En mathématiques, la quantification est restreinte à un ensemble particulier. On écrira par exemple dans le cas des entiers naturels  $\forall x \in \mathbb{N}, P$  ou  $\exists x \in \mathbb{N}, P$ .

En utilisant un symbole de prédicat unaire `N`, tel que `N(x)` représente le fait que  $x \in \mathbb{N}$ , donner des formules de la logique du premier ordre qui représentent les deux quantifications ensemblistes  $\forall x \in \mathbb{N}, P$  et  $\exists x \in \mathbb{N}, P$ .

### Compétences 1.1

- Connaître les notions de signature et arité.
- Distinguer les termes et les formules, et parmi les formules celles qui sont atomiques.

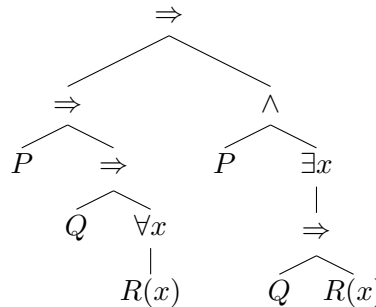
- Reconnaître les *termes* et les *formules syntaxiquement bien formés*.
- Comprendre le sens intuitif des connecteurs propositionnels et quantificateurs logiques.
- Savoir lire une formule logique et traduire une propriété exprimée en langue naturelle en utilisant le langage de la logique.

## 1.2 Structure des formules

### 1.2.1 Représentation des formules comme des arbres

Une formule peut se voir comme un arbre dont les nœuds internes sont les *connecteurs logiques* (les connecteurs propositionnels sont binaires, sauf la négation qui est unaire) et les *quantificateurs*  $\forall x, \exists x$  qui sont des nœuds internes unaires. Les feuilles de l'arbre sont les formules atomiques.

On suppose que  $P$  et  $Q$  sont des variables propositionnelles et que  $R$  est un symbole de prédicat unaire. On peut dessiner sous forme d'arbre la formule  $(P \Rightarrow (Q \Rightarrow (\forall x, R(x)))) \Rightarrow (P \wedge (\exists x, (Q \Rightarrow R(x))))$  :



Cette représentation arborescente pourra être utilisée si on veut représenter les formules en machine. C'est aussi elle qui va nous servir pour construire des fonctions sur les formules et aussi pour raisonner sur les formules.

**Exercice 1.4** Donner la forme arborescente de la formule  $(\forall x, (R(x) \Rightarrow Q)) \Rightarrow (\neg Q \Rightarrow \neg(\exists x, R(x)))$ .

### 1.2.2 Notations, règles de parenthésage

La définition des formules complexes correspond à une méthode de construction des formules à partir des formules atomiques. Pour lever toute ambiguïté, on peut mettre des parenthèses autour de chaque formule non atomique, néanmoins cela alourdit considérablement l'écriture et peut nuire à la lisibilité. Si on supprime les parenthèses et que l'on écrit simplement  $P \wedge Q \vee R$  il y a une ambiguïté : cette formule peut se construire de deux manières différentes suivant si on utilise d'abord la règle pour la disjonction puis la conjonction ou bien dans l'autre sens. Pour lever cette ambiguïté, on utilise des parenthèses et on pourra écrire  $P \wedge (Q \vee R)$  ou bien  $(P \wedge Q) \vee R$ . Les parenthèses sont importantes car les deux formules ne sont pas vraies en même temps (prendre  $P$  faux et  $Q$  et  $R$  vrai).

Comme pour les formules arithmétiques, il y a des règles de *précédence* pour les connecteurs logiques qui évitent l'utilisation systématique des parenthèses. La précedence de  $\neg$  est la plus forte, vient ensuite la conjonction  $\wedge$  puis la disjonction  $\vee$  et finalement l'implication  $\Rightarrow$ .

Dire que  $\neg$  a une précedence plus forte que  $\wedge$  signifie que la formule  $\neg P \wedge Q$  doit se parenthéser en  $(\neg P) \wedge Q$  et non pas en  $\neg(P \wedge Q)$ . De même dire que  $\wedge$  a une précedence plus forte que  $\Rightarrow$  signifie que  $P \Rightarrow Q \wedge R$  se parenthése comme  $P \Rightarrow (Q \wedge R)$  et non pas  $(P \Rightarrow Q) \wedge R$ . On commence par mettre les parenthèses autour du connecteur dont la précedence est la plus forte.

Les connecteurs  $\wedge, \vee$  et  $\Rightarrow$  *associent à droite*, ce qui signifie que la formule  $P \Rightarrow Q \Rightarrow R$  se parenthése en  $P \Rightarrow (Q \Rightarrow R)$  et non pas  $(P \Rightarrow Q) \Rightarrow R$ .

⚠ L'ordre des parenthèses ne change pas le sens des formules pour les connecteurs  $\wedge$  et  $\vee$  qui sont associatifs. Par contre, dans le cas de l'implication, les formules  $P \Rightarrow (Q \Rightarrow R)$  et  $(P \Rightarrow Q) \Rightarrow R$  ne disent pas du tout la même chose, il faut donc être très attentif à ne pas se tromper.

**Exemple 1.6**  $P \wedge R \wedge \neg Q \Rightarrow P \vee Q \wedge R$  représente donc la même formule que  $(P \wedge (R \wedge (\neg Q))) \Rightarrow (P \vee (Q \wedge R))$ .

**Exercice 1.5** Pour chacune des formules suivantes, donner sa représentation sous forme d'arbre ainsi qu'une forme avec des parenthèses sans changer le sens :

1.  $(P \Rightarrow Q \Rightarrow R) \Rightarrow P \wedge Q \Rightarrow R$
2.  $(P \Rightarrow Q) \Rightarrow \neg Q \Rightarrow \neg P$

**Précédence des quantificateurs** Les quantificateurs  $\forall$  et  $\exists$  ont une précédence plus faible que les autres opérateurs. Leur *portée* s'étend donc aussi loin que possible.

**Exemple 1.7**  $\forall x, P \Rightarrow Q$  représente  $\forall x, (P \Rightarrow Q)$  et non pas  $(\forall x, P) \Rightarrow Q$ .

**Exercice 1.6** Exprimer sous une forme logique l'adage suivant : “Tout ce qui ne nous tue pas nous rend plus fort”. On se donne comme symbole de prédicat de base  $\text{Tue}(x)$  et  $\text{Fort}(x)$ . Représenter la formule sous forme d'arbre. Écrire la formule avec toutes les parenthèses puis avec le minimum de parenthèses.

### 1.2.3 Variables libres, liées

Dans les formules  $\forall x, P$  et  $\exists x, P$ , la variable  $x$  est dite *liée* (on dit aussi *muette*), en effet les occurrences de  $x$  sont reliées au quantificateur correspondant et on peut changer le nom de manière cohérente dans  $P$  et le quantificateur sans changer le sens de la formule :

$\forall x, \exists y, x < y$  exprime la même propriété que  $\forall t, \exists u, t < u$ . Une variable  $x$  qui apparaît dans une formule  $P$  mais qui n'est pas dans une sous-expression de la forme  $\forall x, Q$  ou  $\exists x, Q$  est dite *libre*. Par exemple dans la formule  $\exists y, x < y$ , la variable  $x$  est libre et  $y$  est liée. Cette formule représente le fait que “Il existe un nombre plus grand que  $x$ ”.

En langage courant, le plus souvent, on n'introduit pas de nom explicite pour les variables liées, comme dans les exemples suivants :

- Tous les chats sont gris :  $\forall x, \text{chat}(x) \Rightarrow \text{gris}(x)$
- Il existe des chats qui ne sont pas gris :  $\exists x, \text{chat}(x) \wedge \neg \text{gris}(x)$

**Définition 1.2.1 (Variables libres,  $VI(P)$ )** Soit une formule  $P$  et  $x$  une variable. On dit que  $x$  est libre dans la formule  $P$  et on écrira  $x \in VI(P)$  si :

- $P$  est une formule atomique  $R(t_1, \dots, t_n)$  et  $x$  apparaît dans l'un des termes  $t_i$  ;
- $P$  est une négation  $\neg A$  et  $x \in VI(A)$  ;
- $P$  est une formule composée  $A \wedge B$ ,  $A \vee B$ ,  $A \Rightarrow B$  et  $x \in VI(A)$  ou  $x \in VI(B)$  ;
- $P$  est une formule quantifiée  $\forall y, A$  ou  $\exists y, A$  avec  $x \neq y$  et  $x \in VI(A)$ .

⚠ Une variable peut à la fois apparaître en position libre et en position liée, ou bien apparaître liée dans plusieurs quantificateurs comme  $y$  dans la formule  $0 < x \times y \vee (\exists y, x < y) \wedge (\exists y, y + y < x)$ . Cependant ces trois positions ne représentent pas la même chose car on peut toujours choisir un nom nouveau pour les variables liées. La formule pourra s'écrire :  $0 < x \times y \vee (\exists y_1, x < y_1) \wedge (\exists y_2, y_2 + y_2 < x)$ .

**Exercice 1.7** Donner les variables libres de la formule :  $\forall b, b > 0 \Rightarrow \exists q, \exists r, a = b \times q + r \wedge r < b$ .

Les variables libres représentent des *paramètres* de la formule par exemple dans la formule  $\exists y, x = 2 \times y$ , la variable  $x$  est libre, la formule représente la notion “ $x$  est pair”, qui dépend de  $x$ . Cette formule est vraie ou fausse en fonction de la valeur de la variable  $x$ . On peut faire une analogie avec une variable globale d'un programme qui devra avoir une valeur pour permettre une exécution.

- Une variable libre peut être remplacée par un terme plus complexe, on parle de *substitution*, par exemple
- “4 est pair” correspond à la formule  $\exists y, 4 = 2 \times y$

- “ $2 \times z + 4$  est pair” correspond à la formule  $\exists y, 2 \times z + 4 = 2 \times y$

Attention au phénomène de **capture** lorsqu’on substitue une variable par un terme. Supposons que l’on veuille écrire la formule “ $3 \times y$  est pair” :

- Si, dans la formule “ $x$  est pair”, on remplace naïvement par  $x$  par  $3 \times y$  on obtient la formule  $\exists y, 3 \times y = 2 \times y$  qui n’a pas du tout le même sens (cette formule ne parle plus de  $y$ , elle dit qu’il existe un nombre  $y$  tel que  $3 \times y = 2 \times y$  ce qui est vrai car il suffit de prendre  $y = 0$  alors que clairement  $3 \times y$  n’est pas toujours pair. Cette substitution est incorrecte.
- Pour obtenir le bon résultat, il faut d’abord renommer la variable  $y$  qui est liée (et donc muette) dans la formule  $\exists y, x = 2 \times y$  pour obtenir par exemple  $\exists y', x = 2 \times y'$ . On peut alors procéder au remplacement de  $x$  par  $3 \times y$  et obtenir la formule  $\exists y', 3 \times y = 2 \times y'$ .

On notera  $P[x \leftarrow t]$  la formule obtenue en remplaçant dans  $P$  les occurrences libres de la variable  $x$  par le terme  $t$ , après avoir si nécessaire renommé les variables liées de  $P$  afin d’éviter de capturer les variables qui apparaissent dans  $t$ . La définition précise de la substitution dans les formules sera donnée plus tard (définition 1.5.3).

### Définition 1.2.2 (Terme clos, formule close)

- 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**.

### Compétences 1.2

- Connaître les règles de précedence sur les connecteurs et les quantificateurs de la logique.
- Savoir 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**.

## 1.3 Formule vraie

Il y a deux **valeurs de vérité** vrai et faux. On les représente par les éléments de l’ensemble  $\mathbb{B} = \{V, F\}$  des booléens. L’élément  $V$  correspond à vrai et  $F$  à faux. On utilise parfois les entiers  $\{1, 0\}$  à la place, mais il est préférable de ne pas confondre valeur de vérité et valeur entière, d’où le choix de notations spécifiques.

Pour définir quand une formule quelconque est vraie, il faut se placer dans un **modèle** (on parle aussi d’**interprétation**) qui explicite de quels objets on parle et également pour chaque symbole de prédicat (propriété atomique), nous dit pour quels objets il est vérifié.

Par exemple, dans le modèle usuel des mathématiques  $2 \leq 4$  est vrai mais  $0 = 1$  est faux. L’énoncé « Anne est l’amie de Bob » peut être vrai dans certaines situations et faux dans d’autres. L’énoncé « Tout le monde a les yeux bleus » sera vrai ou faux en fonction de notre interprétation de « tout le monde », suivant si on se restreint à une famille donnée ou bien si on considère l’ensemble de la population française.

### 1.3.1 Le cas propositionnel

Une formule logique complexe contient des connecteurs logiques qui ont un sens précis.

La formule atomique  $\perp$  est toujours fausse et la formule atomique  $\top$  est toujours vraie.

Les tables suivantes rappellent les valeurs de vérité des connecteurs propositionnels en fonction des valeurs de vérité des composants de la formule.

$P$	$\neg P$
$V$	$F$
$F$	$V$

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

Pour savoir si une formule propositionnelle est vraie, il “suffit” de la décomposer pour se ramener à la vérité des formules atomiques qui la composent.

Si une formule propositionnelle ne contient pas de variables propositionnelles ni de symboles de prédicats, alors on peut calculer sa valeur de vérité en utilisant les tables ci-dessus.

Par exemple  $((\perp \Rightarrow \perp) \Rightarrow \perp) \Rightarrow \perp$  est une formule propositionnelle qui est toujours vraie.

Les formules sans symboles de prédicat ne sont pas très intéressantes à étudier. On va donc se placer dans le cas d’une formule propositionnelle (pas de quantificateur) qui ne contient comme symboles de prédicat que des variables propositionnelles (symbole d’arité 0, sans argument).

Une variable propositionnelle dans une formule peut prendre soit la valeur vrai, soit la valeur faux. C’est une *inconnue* du problème. Une formule propositionnelle contient un nombre fini de variables propositionnelles que l’on note  $X_1, \dots, X_n$ . Chacune peut prendre comme valeur soit vrai soit faux. Il y a donc en tout  $2^n$  possibilités. Chaque choix correspond à une *interprétation* différente. Si on fixe une interprétation, alors on peut calculer la valeur de vérité de la formule dans cette interprétation en utilisant les tables ci-dessus.

**Définition 1.3.1 (Table de vérité)** Soit une formule  $P$  qui contient les variables propositionnelles  $X_1, \dots, X_n$ . La table de vérité de la formule  $P$  est un tableau à  $n + 1$  colonnes étiquetées par  $X_1, \dots, X_n, P$ .

Chaque ligne correspond à une interprétation différente de  $X_1, \dots, X_n$  et contient dans la colonne  $P$ , la valeur de la formule  $P$  dans cette interprétation.

La table de vérité contient a priori  $2^n$  lignes (une par interprétation). Cependant la valeur de vérité d’une formule ne dépend parfois que de la valeur d’une ou deux variables propositionnelles, on s’autorisera alors à n’écrire qu’une seule ligne, en laissant les valeurs des autres variables indéterminées (et donc cette ligne représentera plusieurs interprétations pour lesquelles la valeur de  $P$  est la même).

**Exemple 1.8** Soit la formule  $P$  définie par  $(A \Rightarrow B \Rightarrow C) \Rightarrow (A \Rightarrow B) \Rightarrow A \Rightarrow C$  avec  $A, B$  et  $C$  des variables propositionnelles.

Avant de construire la table de vérité, il est préférable de parenthéser la formule pour lever toute ambiguïté, ici on obtient  $(A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))$ .

La formule  $P$  a trois variables propositionnelles  $A, B$  et  $C$ , et donc la table de vérité a 8 lignes que l’on peut énumérer avant de calculer la valeur de la formule sur chaque ligne.

$A$	$B$	$C$	$P$
$V$	$V$	$V$	?
$V$	$V$	$F$	?
$V$	$F$	$V$	?
$V$	$F$	$F$	?
$F$	$V$	$V$	?
$F$	$V$	$F$	?
$F$	$F$	$V$	?
$F$	$F$	$F$	?

Plutôt que cette approche « brute-force » qui est propice aux erreurs de calcul, on peut procéder de manière plus subtile en traitant en priorité les cas pour lesquels le calcul de la formule est immédiat (par exemple une conjonction est fausse si l’un des membres est faux, quelle que soit la valeur de l’autre membres).

Dans notre exemple la formule est vraie trivialement si  $C$  est vrai. On peut donc faire une première ligne

$A$	$B$	$C$	$P$
*	*	$V$	$V$

Cette ligne couvre 4 interprétations différentes (toutes les valeurs combinées possibles pour  $A$  et  $B$ ). Les autres interprétations à considérer sont celles où  $C$  a la valeur faux. La formule  $P$  est aussi vraie lorsque  $A$  est faux car alors  $A \Rightarrow C$  est toujours vrai. On peut donc compléter la table de vérité.

$A$	$B$	$C$	$P$
*	*	$V$	$V$
$F$	*	$F$	$V$

Cette ligne ne couvre que deux cas supplémentaires (puisque les cas avec  $C$  vrai ont déjà été comptés). Il ne reste donc que deux interprétations à considérer dans lesquelles  $A$  est vraie et  $C$  est faux. Si  $B$  est faux alors  $A \Rightarrow B$  est faux et donc  $P$  est vrai et si  $B$  est vrai alors  $B \Rightarrow C$  est faux donc  $A \Rightarrow (B \Rightarrow C)$  est faux et donc  $P$  est vrai. Ce qui donne au final la table.

$A$	$B$	$C$	$P$
*	*	$V$	$V$
$F$	*	$F$	$V$
$V$	$V$	$F$	$V$
$V$	$F$	$F$	$V$

**Validité, satisfiabilité.** On va s'intéresser à savoir si une formule est vraie *pour toutes les interprétations possibles* de ces prédicats (pour tous les modèles). Si c'est le cas on dira que la formule est valide ou encore que c'est une tautologie. Si la formule est vraie pour certaines interprétations, on dit que la formule est satisfiable, si elle est fausse dans toutes les interprétations alors elle est dite insatisfiable.

La terminologie satisfiable est un anglicisme couramment utilisé, on trouvera également dans les livres les qualificatifs de formule satisfaisable ou insatisfaisable qui correspondent aux mêmes définitions.

⚠ Attention insatisfiable est la négation de satisfiable, mais ne pas être valide (au moins une interprétation rend fausse la formule) ne signifie pas être insatisfiable (toutes les interprétations rendent fausses la formule). Une formule valide (toutes les interprétations rendent vraie la formule) est a fortiori satisfiable (au moins une interprétation rend vraie la formule), une formule non valide (au moins une interprétation rend la formule fausse) peut être satisfiable (une autre interprétation rend vraie la formule) ou insatisfiable (la formule est tout le temps fausse).

Il y a cependant un lien direct entre validité et satisfiabilité grâce à l'opération de négation sur les formules. En effet pour toute formule  $P$  (propositionnelle ou non), La formule  $P$  est valide si et seulement si la formule  $\neg P$  est insatisfiable. Savoir résoudre un des problèmes est donc équivalent à savoir résoudre l'autre.

**Exemple 1.9** Soit  $P$  et  $Q$  deux variables propositionnelles.

- $\neg(P \wedge Q) \Rightarrow (\neg P \vee \neg Q)$  est une tautologie (formule valide)
- $\neg(P \wedge Q) \Rightarrow (\neg P \vee Q)$  est vrai lorsque  $Q$  est vrai et lorsque  $P$  est faux mais est faux lorsque  $P$  est vrai et  $Q$  est faux (formule satisfiable).
- $\neg P \wedge P$  n'est jamais vérifié (formule insatisfiable).

**Exercice 1.8** Les formules suivantes sont-elles valides ?

1.  $((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$
2.  $(\neg P \Rightarrow P) \Rightarrow P$

**Résoudre des problèmes avec la logique.** Le calcul propositionnel permet de modéliser et de résoudre des problèmes. En plus de pouvoir décider si une formule est valide, on peut également répondre à la question *pour quelles valeurs des formules atomiques la propriété sera-t-elle vérifiée ?* ou bien à la question *combien y-a-t-il de solutions différentes ?*



**Exercice 1.9 (Enigme du tigre et de la princesse)** Des prisonniers sont soumis à un test qui peut se résoudre en utilisant la logique. Ils doivent choisir entre deux portes derrière lesquelles il peut y avoir soit une princesse soit un tigre. S'ils ouvrent une porte derrière laquelle il y a un tigre, ils sont dévorés.

Des inscriptions sur les portes guident leur choix :

La porte 1 comporte la mention “Il y a une princesse derrière cette porte et un tigre derrière l'autre”, la porte 2 comporte la mention “Il y a une princesse derrière une des portes et un tigre derrière l'autre”, on sait de plus qu'une seule de ces inscriptions est vraie.

On introduit les variables propositionnelles  $P_1$  (resp.  $P_2$ ) pour représenter la présence d'une princesse derrière la porte 1 (resp. la porte 2).

- Donner les formules propositionnelles représentant le problème. Quelle porte est-il préférable d'ouvrir ?

### 1.3.2 Formule avec quantificateurs

Le sens commun de la formule  $\forall x, P$  est que  $P$  doit être vrai pour tous les  $x$  mais cela ne peut avoir un sens que si on explique ce que représente  $x$ .

Avant de pouvoir dire quelle est la valeur de vérité de  $\forall x, P$ , il faut donc préciser notre interprétation des symboles. S'il y a des quantificateurs, c'est qu'il y a des objets et donc il faut préciser de quels objets on parle. On se donne donc un ensemble, appelé domaine de l'interprétation (on dira aussi plus simplement domaine) qui représente l'univers auquel appartiennent nos objets. Cet ensemble doit être *non vide*. On fait correspondre à chaque constante, un élément de notre domaine. Deux constantes qui ont des noms différents dans la logique peuvent parfaitement correspondre à la même valeur dans le domaine. A chaque symbole de fonction de la logique, va correspondre une fonction sur le domaine. Pour faire un parallèle avec la programmation, les symboles utilisés de la logique correspondent à l'interface d'un module à partir duquel on peut construire de nouveaux programmes alors que l'interprétation va correspondre à l'implémentation du module qui va permettre d'exécuter les programmes. Le comportement de ces programmes peut complètement changer en fonction de l'implémentation du module.

Une formule atomique  $P(t_1, \dots, t_n)$  représente une vérité qui dépend évidemment de la valeur des arguments  $t_1, \dots, t_n$ . On peut choisir l'interprétation de  $P$  parmi toutes les relations  $n$ -aires sur le domaine de l'interprétation.

Si on a un prédicat unaire comme *yeux-bleus* l'interprétation revient à choisir le sous-ensemble des individus qui ont les yeux bleus. Si on a une relation binaire comme *amis* ou  $\leq$ , l'interprétation peut se représenter comme un graphe orienté dont les sommets sont les éléments du domaine et dans lequel une arête de  $s$  à  $t$  sera présente lorsque la relation est vraie entre  $s$  et  $t$ .

Même si la formule ne contient qu'un symbole de prédicat, si le domaine est infini, le nombre d'interprétations possibles est lui-même infini et donc on ne peut plus faire une table de vérité qui énumère toutes les interprétations.

Par contre si on exhibe une interprétation qui rend vraie la formule, on sait qu'elle est satisfiable et si on trouve un contre-exemple (une interprétation qui rend fausse la formule) alors on sait qu'elle n'est pas valide.

Nous verrons dans les chapitres 2 et 4 des méthodes pour simplifier la recherche d'un modèle.

**Vérité d'une formule quantifiée sans variable libre.** La vérité d'une formule sans variable libre mais avec des quantificateurs du premier ordre dépend en général de l'exploration de la vérité d'un ensemble infini de formules (obtenues en interprétant les variables par des objets arbitraires).

- La formule  $\forall x, P$  est vraie si et seulement si pour tout objet  $t$  du modèle, la formule  $P$  (qui dépend de  $x$ ) est vraie dans un monde où l'objet  $x$  a la valeur  $t$ . vraie.
- La formule  $\exists x, P$  est vraie si et seulement si il existe un objet  $t$  du modèle tel que la formule  $P$  est vraie dans un monde où l'objet  $x$  a la valeur  $t$ .

**Exercice 1.10** Les formules suivantes sont-elles vraies pour toute interprétation du prédicat  $P$  ?

1.  $(\forall x, P(x)) \Rightarrow \exists x, P(x)$
2.  $\neg(\forall x, P(x)) \Rightarrow \exists x, \neg P(x)$

### Compétences 1.3

- Savoir construire la table de vérité d'une formule propositionnelle (prérequis).
- Savoir utiliser la logique propositionnelle pour modéliser et résoudre des problèmes simples.
- Comprendre intuitivement la notion d'interprétation et de modèle (sera approfondi au chapitre 2).

## 1.4 Théories et modélisation

Une formule est un élément de syntaxe qui utilise des symboles qui peuvent avoir de nombreuses interprétations différentes.

Si au contraire on s'intéresse à certaines opérations et certaines propriétés d'un univers particulier, on peut *modéliser* cet univers en introduisant un ensemble de symboles et un ensemble d'énoncés  $\mathcal{A}$  tel que notre univers corresponde à un modèle qui rend vraies toutes les formules de  $\mathcal{A}$ .

On pourra alors travailler au niveau de la logique à partir de cet ensemble d'axiomes et en déduire des propriétés de l'univers qui nous intéresse.

### 1.4.1 Définitions autour des théories

Les mathématiciens se sont intéressés depuis l'antiquité à trouver des présentations axiomatiques de théories.

**Axiomes pour la géométrie.** Par exemple Euclide a formalisé un ensemble d'axiomes pour la géométrie. Dans sa représentation, il choisit comme primitives les notions de points, de segment, de droite, de demi-droite et de cercle et les relie par des axiomes. Le logicien Hilbert a proposé une formulation alternative. Si on ne s'intéresse qu'à la géométrie plane et la notion de point et de droite ainsi qu'à la relation d'incidence (un point appartient à une droite), les axiomes de Hilbert sont les suivants :

- Par deux points distincts, passe une et une seule droite.
- Sur une droite sont situés au moins deux points, et pour une droite donnée, il existe au moins un point qui n'est pas sur la droite.
- Axiome des parallèles. Soient une droite  $d$  et un point  $A$  non sur  $d$  ; il existe au plus une droite passant par  $A$ , qui n'a aucun point commun avec  $d$ .

Pour formaliser cette théorie en logique du premier ordre, il nous faut des prédicats unaires  $P$  et  $D$  pour représenter respectivement les points et les droites et deux prédicats binaires (notés de manière infixe) :  $\in$  pour la propriété pour un point d'être sur une droite et  $=$  pour l'égalité (afin de pouvoir traduire la propriété d'unicité). Le premier axiome ci-dessus se traduit par la formule

$$\forall p, q, P(p) \wedge P(q) \wedge \neg(p = q) \Rightarrow \exists d, (D(d) \wedge p \in d \wedge q \in d \wedge (\forall d', D(d') \wedge p \in d' \wedge q \in d' \Rightarrow d = d'))$$

Pour une modélisation complète de la géométrie euclidienne, Hilbert introduit d'autres notions primitives pour modéliser les distances. Une relation ternaire entre les points représente le fait qu'un point est entre deux autres. Ce qui permet d'introduire la notion de segment  $AB$  (les points situés entre les points  $A$  et  $B$ ). Les notions de demi-droite, d'angle et de triangle sont aussi dérivés. Les relations de congruence sont introduites entre segments, angles et triangles pour modéliser les notions de distance et le fait que les figures géométriques se "déplacent" dans l'espace sans changer de forme.



**Axiomes pour les groupes.** La théorie des groupes est une généralisation du modèle des entiers relatifs : on se donne une constante  $0$ , une opération binaire  $+$ , une opération unaire  $-$  et un prédicat binaire d'égalité. On suppose que ces opérateurs vérifient un certain nombre de propriétés logiques

- associativité :  $\forall x y z, (x + y) + z = x + (y + z)$
- élément neutre :  $\forall x, x + 0 = x \wedge 0 + x = x$
- inverse :  $\forall x, x + (-x) = 0 \wedge (-x) + x = 0$

et on en déduit divers conséquences qui seront vraies pour les entiers relatifs mais aussi pour toutes les autres structures de groupe.

**Définition 1.4.1 (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.

Un modèle d'une théorie est donné par une interprétation de la signature (c'est-à-dire un domaine correspondant à l'ensemble dans lequel on interprète les objets et des fonctions et relations sur ce domaine) telle que, dans cette interprétation, tous les axiomes ont pour valeur vraie.

Une théorie peut être définie par un ensemble fini ou infini d'axiomes.

Soit une théorie définie par un ensemble  $\mathcal{A}$  d'axiomes. Au lieu de s'intéresser à la validité d'une formule, on va se poser la question *est-ce que la formule est vraie dans tous les modèles de la théorie ?* ou encore est-ce qu'on peut par un raisonnement qui suppose vrais tous les axiomes de la théorie, déduire cette propriété.

Si une formule  $P$  est vraie dans tous les modèles d'une théorie, on dit que la formule est vraie dans la théorie  $\mathcal{A}$  ou encore qu'elle est conséquence des axiomes  $\mathcal{A}$ .

Une théorie peut ne pas avoir de modèle, auquel cas on dit que la théorie est incohérente. Dans une théorie incohérente, même la formule  $\perp$  est vraie !

La théorie  $\mathcal{A}$  est dite complète si pour toute formule  $P$  soit  $P$  est vraie dans la théorie, soit  $\neg P$  est vraie dans la théorie.

La plupart des théories ne sont pas complètes. Si on pense à la théorie des groupes et que l'on regarde la formule qui dit que le groupe est commutatif ( $\forall x y, x + y = y + x$ ) alors la théorie ne permet de prouver ni cette formule ni sa négation puisqu'il existe à la fois des groupes commutatifs et des groupes non-commutatifs.

Pour faire des mathématiques, on aimerait pouvoir définir un ensemble d'axiomes qui nous permette de capturer avec précision et rigueur ce qui est vrai ou faux.

Le théorème d'incomplétude de Gödel nous dit que toute théorie (dont on peut reconnaître les axiomes) qui contient l'arithmétique est forcément incomplète et que quelques soient les axiomes que l'on ajoute la théorie restera incomplète à moins de devenir incohérente.

Une théorie est décidable, si on sait, pour toute formule  $P$ , décider si  $P$  est ou non une conséquence des axiomes. La théorie vide n'est pas décidable. La théorie de l'arithmétique linéaire (sans multiplication) est décidable ainsi que la théorie des réels.

## 1.4.2 Exemples de théories

### Égalité

Une théorie importante est celle de l'égalité. Elle contient un symbole de prédicat binaire  $=$  qui sera noté de manière infixé. Les axiomes nécessaires sont les suivants.

**Définition 1.4.2 (Théorie de l'égalité)** La théorie de l'égalité est donnée par les axiomes qui expriment que c'est une relation d'équivalence.

- réflexivité :  $\forall x, x = x$
- symétrie :  $\forall x y, x = y \Rightarrow y = x$
- transitivité :  $\forall x y z, x = y \Rightarrow y = z \Rightarrow x = z$

Pour chaque symbole de fonction  $f$   $n$ -aire on ajoute un axiome qui exprime le fait que c'est une congruence

$$\forall x_1 \dots x_n y_1 \dots y_n, x_1 = y_1 \wedge \dots \wedge x_n = y_n \Rightarrow f(x_1, \dots, x_n) = f(y_1, \dots, y_n)$$

et pour chaque symbole de prédicat  $P$   $n$ -aire on ajoute un axiome qui exprime le fait que la propriété est stable par égalité :

$$\forall x_1 \dots x_n y_1 \dots y_n, x_1 = y_1 \wedge \dots \wedge x_n = y_n \Rightarrow P(x_1, \dots, x_n) \Rightarrow P(y_1, \dots, y_n)$$

Les axiomes de congruence sont importants en effet le symbole d'égalité peut en logique s'interpréter de manière très variée par n'importe quelle relation d'équivalence (réflexive, transitive et symétrique) de même pour l'interprétation des fonctions et des prédicats qui peuvent ne pas vérifier les propriétés de congruence.

Supposons que l'on veuille modéliser les rationnels, on pourrait introduire des symboles de fonction pour le dénominateur et le numérateur. Néanmoins la fonction dénominateur n'est pas une congruence (avec l'égalité usuelle sur les rationnels) en effet on a  $\frac{1}{2} = \frac{2}{4}$  mais  $2 \neq 4$ . Il faudra donc soit se passer des fonctions d'accès au numérateur et au dénominateur, soit distinguer une égalité de représentation dans laquelle deux rationnels sont égaux s'ils ont même numérateur et même dénominateur et donc  $\frac{1}{2} \neq \frac{2}{4}$  et introduire une autre relation d'équivalence  $=_{\mathbb{Q}}$  pour lier des fractions qui représentent le même rationnel. Ces considérations sont très semblables à ce que l'on rencontre en programmation. Si les langages de programmation fournissent une notion d'égalité générique, elle correspondra à une égalité des représentations des objets et il est laissé à l'utilisateur le soin d'introduire des égalités plus larges en fonction des interprétations souhaitées.

La plupart des théories vont contenir un symbole d'égalité et donc inclure ces axiomes. On utilisera plus simplement la notation  $t \neq u$  pour représenter la formule  $\neg(t = u)$ .

## Arithmétique

**Définition 1.4.3 (Théorie arithmétique de Peano (PA))** Le langage est composé de la constante  $0$  du symbole de fonction unaire  $S$  et des symboles binaires  $+$  et  $\times$  ainsi que du symbole de prédicat d'égalité.

Les sept axiomes suivants constituent ce que l'on appelle l'arithmétique élémentaire. Cette théorie est notée  $PA_0$ .

1.  $\forall x, S(x) \neq 0$
2.  $\forall x, x = 0 \vee \exists y, x = S(y)$  (inutile en présence de récurrence)
3.  $\forall x y, S(x) = S(y) \Rightarrow x = y$
4.  $\forall x, x + 0 = x$
5.  $\forall x y, x + S(y) = S(x + y)$
6.  $\forall x, x \times 0 = 0$
7.  $\forall x y, x \times S(y) = (x \times y) + x$

La théorie  $PA_0$  n'est pas suffisante pour prouver une propriété comme la symétrie de l'addition.

Pour obtenir l'arithmétique de Peano, on lui ajoute une infinité d'axiomes pour le schéma de preuve par récurrence. Pour chaque formule  $P$  ayant comme variables libres  $x_1, \dots, x_n, x$ , on aura comme axiome la formule

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

On rappelle que  $P[x \leftarrow t]$  représente la formule obtenue en remplaçant dans la formule  $P$  toutes les occurrences libres de la variable  $x$  par le terme  $t$ .

A tout entier  $n \in \mathbb{N}$ , on peut associer un terme de l'arithmétique de Peano  $S^n(0)$  que l'on notera  $\tilde{n}$ .

On peut se demander pourquoi choisir l'addition et la multiplication et pas d'autres fonctions. En fait ces deux opérations permettent de représenter logiquement toutes les autres fonctions calculables sur les entiers sous la forme de relation. Par exemple, si on veut la soustraction, on pourra définir le fait que  $z$  représente la différence entre  $x$  et  $y$  par la formule  $z + y = x$ . On peut aussi représenter les opérations de division euclidienne et de reste modulo qui permettent de faire des codages pour des couples et des suites d'entiers.

**Exercice 1.11** On définit la propriété  $t \leq u \stackrel{\text{def}}{=} \exists n, n + t = u$ . Montrer que les propriétés de base de l'ordre sur les entiers sont vraies dans la théorie de l'arithmétique. Pour la transitivité, on pourra utiliser sans la démontrer la propriété d'associativité de l'addition.

- $\forall x, 0 \leq x$
- $\forall x, y, x \leq y \Leftrightarrow S(x) \leq S(y)$
- transitivité :  $\forall x, y, z, x \leq y \Rightarrow y \leq z \Rightarrow x \leq z$

**Exercice 1.12** 1. Définir par une formule logique contenant trois variables libres  $x, y$  et  $d$  le fait que  $d$  est le résultat de la division entière de  $x$  par  $y$ . On notera  $\text{div}(x, y, d)$  cette formule.

2. Définir par une autre formule logique contenant trois variables libres  $x, y$  et  $r$  le fait que  $r$  est le reste modulo de la division entière de  $x$  par  $y$ .

3. Que pensez-vous de la valeur de vérité de la formule  $\text{div}(x, 0, d)$  dans laquelle on a remplacé  $y$  par 0?

4. Si la division entière est représentée par une fonction à deux arguments, on peut écrire une propriété comme  $(x/y)/z = x/(y \times z)$ . Exprimer le même résultat en utilisant le prédicat  $\text{div}$  à trois arguments.

### Modélisation en logique propositionnelle

Si on se place dans le fragment propositionnel (pas de quantificateurs) alors on peut savoir si une formule est satisfiable, valide ou non. Les SAT-solvers sont des outils qui permettent de traiter des problèmes propositionnels. L'idée est d'avoir des variables pour des propositions atomiques qui peuvent valoir vrai ou faux, puis de décrire un problème sous la forme de contraintes sur ces variables exprimées comme des formules propositionnelles. L'outil détermine s'il existe une manière d'affecter une valeur vrai ou faux à chaque variable propositionnelle qui rende le problème vrai.

**Exemple 1.10** On peut modéliser un problème de Sudoku sous une forme propositionnelle. Pour cela on introduit  $9^3 = 729$  variables  $X_{ijk}$  pour  $i, j, k \in [1, 9]$ . L'interprétation est que  $X_{ijk}$  est vrai si le chiffre  $k$  est sur la  $i$ -ème ligne et la  $j$ -ème colonne. On peut ensuite exprimer les contraintes :

1. Au plus un seul chiffre  $k$  par case  $(i, j)$  : Par exemple pour indiquer que sur la case  $(1, 1)$  il n'y a pas plusieurs chiffres on va dire que s'il y a un 1 alors il ne peut y avoir ni un 2, ni un 3, etc c'est-à-dire

$$X_{111} \Rightarrow \neg X_{112} \wedge \neg X_{113} \wedge \neg X_{114} \wedge \neg X_{115} \wedge \neg X_{116} \wedge \neg X_{117} \wedge \neg X_{118} \wedge \neg X_{119}$$

On écrit cette formule de manière plus synthétique :  $X_{111} \Rightarrow \bigwedge_{l \neq 1} \neg X_{11l}$ .

Au final il faudra écrire  $9^3$  formules pour toutes les valeurs de  $i, j, k \in [1, 9]$  de la forme :

$$X_{ijk} \Rightarrow \bigwedge_{l \neq k} \neg X_{ijl}$$

En utilisant le fait que la formule  $A \Rightarrow (B \wedge C)$  est équivalente à  $(A \Rightarrow B) \wedge (A \Rightarrow C)$  et que  $A \Rightarrow B$  est équivalent à  $\neg A \vee B$  on peut transformer l'ensemble de formules précédents en des formules plus symétriques  $\neg X_{ijk} \vee \neg X_{ijl}$  pour tout  $i, j \in [1, 9]$  et pour toutes les paires d'éléments  $k \neq l$ . On a au total  $9 \times 9 \times 9 \times 4$  formules.

2. Chaque chiffre  $k$  apparaît sur chaque ligne  $i$  au moins dans une colonne  $j$ . La formule qui dit que le chiffre 1 apparaît au moins une fois sur la première ligne s'écrit :

$$X_{111} \vee X_{121} \vee X_{131} \vee X_{141} \vee X_{151} \vee X_{161} \vee X_{171} \vee X_{181} \vee X_{191}$$

que l'on écrit de manière synthétique  $\bigvee_{j=1..9} X_{1j1}$  et au final il faudra écrire  $9^2$  formules (une pour chaque ligne  $i$  et pour chaque chiffre  $k$ ) de la forme  $\bigvee_{j=1..9} X_{ijk}$

3. Chaque chiffre  $k$  apparaît au plus une fois sur chaque ligne  $i$  (s'il est dans la colonne  $j$ , il n'est pas dans une autre colonne) :

La formule qui dit que le chiffre 1 apparaît au plus une fois sur la première ligne et la première colonne alors il n'apparaît pas dans une autre colonne s'écrit :

$$X_{111} \Rightarrow \neg X_{121} \wedge \neg X_{131} \wedge \neg X_{141} \wedge \neg X_{151} \wedge \neg X_{161} \wedge \neg X_{171} \wedge \neg X_{181} \wedge \neg X_{191}$$

que l'on écrit de manière synthétique  $X_{111} \Rightarrow \bigwedge_{l \neq 1} \neg X_{1l1}$ . Au final il faudra écrire  $9^3$  formules pour toutes les valeurs de  $i, j, k \in [1, 9]$  de la forme :  $X_{ijk} \Rightarrow \bigwedge_{l \neq j} \neg X_{ilk}$ . Comme dans le cas de la propriété au plus un chiffre par case, une formulation alternative est de considérer toutes les formules  $\neg X_{ijk} \vee \neg X_{ilk}$  avec  $l \neq j$

4. Même chose pour les colonnes et les carrés.

5. Lorsque la case  $(i, j)$  est pré-remplie avec le chiffre  $k$ , on introduira la contrainte :  $X_{ijk}$ .

On obtient ainsi un ensemble d'axiomes (on pourra estimer l'ordre de grandeur). Chaque modèle de cette théorie représente une solution du Sudoku. Il peut n'y en avoir aucune ou au contraire plusieurs. Le nombre de variables est  $9^3$ , ce qui fait que le nombre d'interprétations est  $2^{9^3} = 2^{729} \simeq 10^{219}$ . Un traitement manuel de la table de vérité n'est pas envisageable mais il n'est pas très compliqué d'écrire un programme qui résout un tel problème automatiquement en utilisant la logique.

Les SAT-solvers utilisent des algorithmes sophistiqués pour pouvoir traiter des formules avec un très grand nombre de variables propositionnelles. Ils sont utilisés pour vérifier des programmes, ou bien pour résoudre des problèmes de planification qui seront modélisés de manière logique.

Sur le problème du Sudoku, on voit qu'utiliser des formules propositionnelles nécessite l'introduction de beaucoup de variables propositionnelles et d'axiomes. Au contraire en logique du premier ordre, il est possible de considérer les positions et les chiffres comme des objets de la logique et d'utiliser donc des variables d'objets pour les représenter. Au lieu des 729 variables propositionnelles, on introduit un seul symbole de prédicat à trois arguments  $\text{pos}(i, j, k)$  qui sera vrai lorsque le chiffre  $k$  est à la position  $(i, j)$ . On a également besoin du symbole binaire d'égalité sur les objets.

Cela permet de factoriser la description avec un nombre plus réduit d'axiomes.

**Exercice 1.13** — A l'aide du symbole de prédicat  $\text{pos}(i, j, k)$  défini ci-dessus et de la relation d'égalité, donner des axiomes en logique du premier ordre qui décrivent les règles du Sudoku.

- Le langage proposé est-il suffisant pour décrire la contrainte que le même chiffre n'apparaît pas dans le même "cadran" de la grille. Que proposez-vous pour résoudre ce problème ?
- On veut contraindre les objets de cette théorie à être dans un ensemble avec exactement 9 valeurs. Comment peut-on traduire cette contrainte en formule logique ?

#### Compétences 1.4

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

## 1.5 Définition récursive sur les formules

On a vu que les formules pouvaient se représenter comme des arbres. On introduit dans cette section une méthode générale de définition récursive sur la structure des formules qui nous permet de définir des opérations ou propriétés mathématiques sur les formules ou bien de les manipuler par ordinateur.

Les formules atomiques pouvant faire référence à des termes, nous donnons également les éléments pour définir des fonctions de manière récursive sur les termes et raisonner par récurrence structurale.


### 1.5.1 Définir une fonction

On peut définir une fonction  $G$  de l'ensemble des formules logiques dans un ensemble  $\mathcal{D}$  quelconque en se donnant un ensemble d'équations que doit satisfaire cette fonction. Les équations sont de la forme  $G(t) = u$  avec  $u$  un élément de l'ensemble  $\mathcal{D}$ . Il faut une équation pour chacune des constructions possibles de formule : formules atomiques (en particulier  $\top$ ,  $\perp$ ), formules composées d'un connecteur propositionnel  $\neg A$ ,  $A \wedge B$ ,  $A \vee B$ ,  $A \Rightarrow B$  ou d'un quantificateur  $\forall x, A$ ,  $\exists x, A$ .

Dans le cas des connecteurs propositionnels et des quantificateurs, le membre droit  $u$  peut utiliser la valeur de  $G$  sur des sous-formules de  $t$  (comme  $G(A)$  et  $G(B)$ ). On parle de définition récursive.

On peut par exemple définir une fonction `nbsymbp` qui compte le nombre de *connecteurs propositionnels* dans une formule :

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

 Dans les définitions ci-dessus  $p$ ,  $A$  ou  $B$  ne sont pas des symboles logiques mais des variables *mathématiques* qui nous servent à définir la fonction (comme la variable  $x$  dans une définition  $f(x) = x + 1$ ), elles représentent n'importe quelle formule complexe.

Ce qui est important dans cette définition est qu'elle est *complète* car elle couvre tous les cas de formules, que pour chaque formule il n'y a qu'une seule équation qui s'applique et donc que la définition est *non-ambigüe* et finalement on dit qu'elle est *bien fondée* car les références à la fonction  $G$  dans les membres droits des équations se font sur des formules strictement plus petites que le membre gauche.

Ces équations définissent également un moyen de calcul de la valeur de  $G$  pour une formule donnée. Par exemple dans le cas de la formule  $A \stackrel{\text{def}}{=} (\forall x, P \Rightarrow Q) \Rightarrow \neg Q \Rightarrow \neg \exists x, P$  on aura  $\text{nbsymbp}(A) = 5$ .

**Exercice 1.14** Modifier la définition précédente pour définir une fonction `nbsymb` qui compte le nombre de symboles logiques dans la formule, à savoir les connecteurs propositionnels comme dans `nbsymbp` mais aussi les quantificateurs  $\forall$  et  $\exists$ .

**Exercice 1.15** Donner les équations qui définissent la profondeur  $\text{prof}(A)$  d'une formule  $A$ , c'est-à-dire qui donne le nombre maximal de connecteurs et quantificateurs imbriqués. Dans le cas de la formule  $A$  précédente on aura  $\text{prof}(A) = 4$ .

### Restriction au calcul propositionnel

Le calcul propositionnel est le sous-ensemble du calcul des prédicats qui ne contient que des symboles de prédicat d'arité 0, qui sont appelés des variables propositionnelles et pas de quantificateur ni de termes. On peut de la même manière faire une définition récursive sur la structure d'une formule propositionnelle en décomposant le cas des formules atomiques en trois cas  $\perp$ ,  $\top$  et les variables propositionnelles et en éliminant les deux cas des quantificateurs.

**Calcul de la valeur de vérité d'une formule propositionnelle** Une fois que l'on se donne une *interprétation*  $I$  qui à chaque variable propositionnelle associe sa valeur de vérité, on peut définir une fonction  $\text{val}(I, P) \in \mathbb{B}$  qui calcule la valeur de vérité d'une formule propositionnelle quelconque  $P$  par les équations suivantes

$$\begin{aligned}
 \text{val}(I, \perp) &= F \\
 \text{val}(I, \top) &= V \\
 \text{val}(I, p) &= I(p) && \text{si } p \text{ est une variable propositionnelle} \\
 \text{val}(I, \neg P) &= \text{si } \text{val}(I, P) = T \text{ alors } F \text{ sinon } V \\
 \text{val}(I, P \wedge Q) &= \text{si } \text{val}(I, P) = V \text{ alors } \text{val}(I, Q) \text{ sinon } F \\
 \text{val}(I, P \vee Q) &= \text{si } \text{val}(I, P) = V \text{ alors } V \text{ sinon } \text{val}(I, Q) \\
 \text{val}(I, P \Rightarrow Q) &= \text{si } \text{val}(I, P) = V \text{ alors } \text{val}(I, Q) \text{ sinon } V
 \end{aligned}$$

### 1.5.2 Raisonner sur les formules

Pour établir un résultat pour toutes les formules, on peut utiliser un raisonnement par récurrence sur la *structure* de la formule. Soit une propriété  $\phi(P)$  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.

Ce principe de preuve est très utile lorsque l'on veut montrer des propriétés qui parlent de fonctions sur les formules qui sont elles-mêmes définies récursivement.

**Exemple 1.11** On peut définir de manière récursive le nombre d'occurrences de sous-formules atomiques dans une formule logique :

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

On peut ensuite montrer que pour toute formule  $P$ , on a  $\text{nbatom}(P) \leq 1 + \text{nbsymbp}(P)$ .

Pour faire cette preuve, on commence par expliciter la propriété  $\phi(P)$  que l'on cherche à montrer par récurrence structurelle sur  $P$ . Dans notre cas, il s'agit de  $\text{nbatom}(P) \leq 1 + \text{nbsymbp}(P)$ . On doit examiner chacun des cas possibles pour la formule  $P$  :

1. Cas où  $P$  est une formule atomique  $p$ . Par définition de  $\text{nbatom}$  et  $\text{nbsymbp}$ , on a  $\text{nbatom}(p) = 1$  et  $\text{nbsymbp}(p) = 0$  donc on vérifie aisément que  $\text{nbatom}(p) \leq 1 + \text{nbsymbp}(p)$  et donc  $\phi(p)$  est vérifié.
2. Cas où  $P$  est une négation de la forme  $\neg A$  avec  $A$  une formule quelconque. On suppose (hypothèse de récurrence) que  $\phi(A)$  est vérifié et donc que  $\text{nbatom}(A) \leq 1 + \text{nbsymbp}(A)$ . Il faut montrer que  $\phi(\neg A)$  est également vérifié. Par définition de  $\text{nbatom}$  et  $\text{nbsymbp}$ , on a  $\text{nbatom}(\neg A) = \text{nbatom}(A)$  et  $\text{nbsymbp}(\neg A) = 1 + \text{nbsymbp}(A)$ . En utilisant l'hypothèse de récurrence on a donc

$$\text{nbatom}(\neg A) = \text{nbatom}(A) \leq 1 + \text{nbsymbp}(A) = \text{nbsymbp}(\neg A) \leq 1 + \text{nbsymbp}(\neg A)$$

et donc on a bien montré que  $\phi(\neg A)$  était vérifiée.



3. Cas où  $P$  est une conjonction, disjonction ou implication de deux formules quelconques  $A$  et  $B$ , on note  $P = A \circ B$  avec  $\circ$  l'un des trois connecteurs. En effet, dans cet exemple, ils jouent tous les trois le même rôle, il n'est donc pas nécessaire de distinguer les trois cas.

On suppose (hypothèses de récurrence) que  $\phi(A)$  et  $\phi(B)$  sont vérifiés et donc que  $\text{nbatom}(A) \leq 1 + \text{nbsymbp}(A)$  et  $\text{nbatom}(B) \leq 1 + \text{nbsymbp}(B)$ .

Il faut montrer que  $\phi(A \circ B)$  est également vérifié. Par définition de  $\text{nbatom}$  et  $\text{nbsymbp}$ , on a  $\text{nbatom}(A \circ B) = \text{nbatom}(A) + \text{nbatom}(B)$  et  $\text{nbsymbp}(A \circ B) = 1 + \text{nbsymbp}(A) + \text{nbsymbp}(B)$ .

En utilisant les hypothèses de récurrence on a donc

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

et donc on a bien montré que  $\phi(A \circ B)$  était vérifiée.

4. Cas où  $P$  est une formule quantifiée de la forme  $\forall x, A$  ou  $\exists x, A$  avec  $A$  quelconque.

On suppose (hypothèse de récurrence) que  $\phi(A)$  est vérifié et donc que  $\text{nbatom}(A) \leq 1 + \text{nbsymbp}(A)$ .

Il faut montrer que  $\phi(\forall x, A)$  et  $\phi(\exists x, A)$  sont également vérifiés. On traite seulement le cas  $\forall x, A$  car celui de  $\exists x, A$  est identique.

Par définition de  $\text{nbatom}$  et  $\text{nbsymbp}$ , on a  $\text{nbatom}(\forall x, A) = \text{nbatom}(A)$  et  $\text{nbsymbp}(\forall x, A) = \text{nbsymbp}(A)$ . En utilisant l'hypothèse de récurrence on a donc

$$\text{nbatom}(\forall x, A) = \text{nbatom}(A) \leq 1 + \text{nbsymbp}(A) = 1 + \text{nbsymbp}(\forall x, A)$$

et donc on a bien montré que  $\phi(\forall x, A)$  était vérifiée.

La même preuve nous donne que  $\phi(\exists x, A)$  est vérifié.

On a bien examiné tous les cas possibles et prouvé par récurrence sur la structure de la formule  $P$ , que pour toute formule logique  $P$ , on a  $\text{nbatom}(P) \leq 1 + \text{nbsymbp}(P)$ .

### 1.5.3 Définition récursive sur les termes

En logique du premier ordre, les formules font référence à des objets qui sont représentés syntaxiquement par des termes qui apparaissent en argument des symboles de prédicat au niveau des formules atomiques.

Soit  $\mathcal{F}$  une *signature*, c'est-à-dire un ensemble de symboles de fonctions définis avec leur arité et  $\mathcal{X}$  un ensemble de variables. On note  $\mathcal{F}_n$  l'ensemble des symboles de  $\mathcal{F}$  d'arité  $n$  et on rappelle que  $\mathcal{T}(\mathcal{F}, \mathcal{X})$  représente l'ensemble des termes bien formés construits à partir de cette signature et de ces variables.

De la même manière que l'on fait une définition récursive sur les formules, on peut construire une fonction  $G$  de manière récursive sur les termes, en se donnant un ensemble d'équations. Il y aura une équation dans le cas où le terme est une variable  $x$  et une équation pour chaque symbole d'objet. Si on a un symbole  $f$  d'arité  $n$  alors il y aura une équation  $G(f(t_1, \dots, t_n)) = u$  avec la possibilité pour  $u$  de faire référence aux résultats de la fonction  $G$  sur les sous termes et donc de mentionner non seulement les termes  $t_1, \dots, t_n$  mais aussi les valeurs  $G(t_1), \dots, G(t_n)$ .

**Exemple 1.12** On se donne un langage avec une constante  $c$ , une fonction unaire  $f$  et une fonction binaire  $g$ .

On définit une fonction  $\text{clos}$  qui étant donné un terme  $t$  teste s'il est clos, c'est-à-dire s'il ne contient aucune variable, en utilisant les équations suivantes :

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

**Définition 1.5.1 (Définition récursive sur les termes)** Soit  $\mathcal{F}$  une signature,  $\mathcal{X}$  un ensemble de variables et  $\mathcal{D}$  un ensemble quelconque.

On suppose que l'on veut définir une application  $G$  qui prend en argument un terme et renvoie un élément de  $\mathcal{D}$ , on a donc  $G \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \rightarrow \mathcal{D}$ . Pour cela on va se donner les objets suivants :

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 :

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

Le fait que l'on puisse définir une telle application est une conséquence du fait que l'égalité sur les termes correspond à un critère *syntactique*. Deux termes sont égaux s'ils ont la même représentation sous forme d'arbre, c'est-à-dire que deux termes qui débutent par des symboles différents sont différents.

En effet ce schéma permet de définir une application  $G$  telle que  $G(c) = 1$  pour les constantes et  $G(t) = 2$  pour tous les termes qui commencent par un symbole de fonction. Si on avait  $\text{plus}(0, 0) = 0$  alors on en déduirait  $G(\text{plus}(0, 0)) = G(0)$  et donc  $2 = 1$ .

**Exemple 1.13 (Taille d'un terme)** Le schéma de définition récursive précédent permet de définir l'application *size* qui compte le nombre de symboles dans un terme.

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

Dans le cas d'une signature sur les entiers qui contient la constante  $0$ , le symbole unaire  $S$  et le symbole binaire  $\text{plus}$ , soit  $t$  le terme  $\text{plus}(0, S(0))$ , il vérifie  $\text{size}(t) = 4$ .

**Exemple 1.14 (Hauteur d'un terme)** Un autre exemple de définition récursive est l'application *ht* qui compte le nombre maximal de symboles imbriqués dans un terme.

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

Pour le terme  $t$  précédent on a  $\text{ht}(t) = 3$ .

**Exercice 1.16** Ecrire une fonction *vars* qui prend en argument un terme et renvoie l'ensemble des variables qui apparaissent dans ce terme.

## Substitution

Nous avons déjà utilisé sans la définir formellement la notion de remplacement d'une variable par un terme. Ici on généralise cette opération par le remplacement en parallèle de plusieurs variables par des termes. Pour cela, on se donne une application  $\sigma \in \mathcal{X} \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{X})$ , appelée *substitution* qui associe un terme à chaque variable. On notera  $\{x_1 \leftarrow u_1; \dots; x_n \leftarrow u_n\}$  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  $t[\sigma]$ .



**Définition 1.5.2 (Substitution sur les termes,  $t[\sigma]$ )** Si  $\sigma$  est de la forme  $\{x_1 \leftarrow u_1; \dots; x_n \leftarrow u_n\}$ , alors le terme  $t[\sigma]$  sera noté  $t[x_1 \leftarrow u_1; \dots; x_n \leftarrow u_n]$ .

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

On peut montrer que le résultat de  $t[\sigma]$  ne dépend que de la valeur de la substitution  $\sigma$  sur les variables de  $t$ . De manière plus précise cela revient à montrer que si on a deux substitutions  $\sigma_1$  et  $\sigma_2$  ainsi qu'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 structurale sur le terme  $t$  suivant le schéma ci-dessous.

### Récurrence sur les termes

On a également un schéma de preuve par récurrence sur les termes de  $\mathcal{T}(\mathcal{F}, \mathcal{X})$  qui s'exprime de la manière suivante. Soit  $\phi(t)$  une propriété mathématique qui dépend d'un terme  $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ . On suppose :

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 1.16** On peut par exemple montrer la propriété suivante pour tout terme  $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ ,  $ht(t) \leq \text{size}(t)$ .

**Preuve:** La preuve se fait par récurrence sur la structure du terme  $t$ . La propriété  $\phi(t)$  à montrer est définie comme  $ht(t) \leq \text{size}(t)$ .

**variable** si  $t$  est une variable  $x \in \mathcal{X}$ , alors il faut montrer  $ht(x) \leq \text{size}(x)$  qui est vrai car  $ht(x) = 0$  et  $\text{size}(x) = 0$

**constante** si  $t$  est une constante  $c$  alors il faut montrer  $ht(c) \leq \text{size}(c)$  qui est vrai car  $ht(c) = 1 = \text{size}(c)$ .

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

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

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

**Exercice 1.17** Sur la signature des arbres binaires qui comporte une constante  $\epsilon$  pour représenter une feuille et un symbole  $N$  binaire pour représenter un nœud interne. Définir une fonction *feuilles* qui compte le nombre de feuilles et une fonction *noeuds* qui compte le nombre de nœuds internes. Montrer que pour tout arbre  $t$  qui ne contient pas de variable, on a  $\text{feuilles}(t) = \text{noeuds}(t) + 1$ .

### Substitution sur les formules

On a défini formellement la fonction de substitution d'une variable par un terme dans un autre terme. On peut faire de même avec la définition de la substitution d'une variable par un terme dans une formule quelconque. On construit la fonction à l'aide d'équations récursives sur la structure de la formule.

La définition de la substitution d'une variable  $x$  par un terme  $t$  dans une formule  $P$  doit prendre en compte le problème de la capture possible d'une des variables du terme  $t$  par un des quantificateurs interne de  $P$ .

**Définition 1.5.3 (Substitution  $P[x \leftarrow t]$  d'une variable par un terme dans une formule)** Soit  $x$  une variable et  $t$  un terme.

- Formules atomiques :
  - $\perp[x \leftarrow t] = \perp$
  - $\top[x \leftarrow t] = \top$
  - $R(t_1, \dots, t_n)[x \leftarrow t] = R(t_1[x \leftarrow t], \dots, t_n[x \leftarrow t])$
- Soit  $A$  une formule,  $(\neg A)[x \leftarrow t] = \neg(A[x \leftarrow t])$
- Soient  $A$  et  $B$  des formules, si  $\circ \in \{\wedge, \vee, \Rightarrow\}$  alors  $(A \circ B)[x \leftarrow t] = (A[x \leftarrow t]) \circ (B[x \leftarrow t])$
- Soit  $P$  une formule et  $y$  une variable :
  - si  $y = x$  alors  $(\forall y, Q)[x \leftarrow t] = \forall x, Q$  et comme  $x$  n'est pas libre dans  $\forall x, Q$  on a  $(\forall y, Q)[x \leftarrow t] = (\forall x, Q)[x \leftarrow t] = \forall x, Q$
  - de même si  $y = x$  alors  $(\exists y, Q)[x \leftarrow t] = (\exists x, Q)[x \leftarrow t] = \exists x, Q$
  - si  $y \neq x$  et  $y \notin \text{vars}(t)$  alors  $(\forall y, Q)[x \leftarrow t] = \forall y, (Q[x \leftarrow t])$  (la substitution se propage sous le quantificateur, sans risque de capture puisque la variable liée  $y$  n'apparaît pas dans  $t$ )
  - si  $y \neq x$  et  $y \in \text{vars}(t)$  alors  $(\forall y, Q)[x \leftarrow t] = \forall y, (Q[x \leftarrow t])$

Cette définition est donc partielle dans le cas des formules avec quantificateurs si une variable liée dans la formule apparaît aussi dans le terme que l'on veut substituer. Cependant en procédant à un renommage des variables liées dans les quantificateurs, on peut toujours se ramener à une situation dans laquelle la substitution sera possible.

**Exercice 1.18** La définition 1.4.2 nous donne les axiomes de la théorie de l'égalité pour un langage quelconque. Soient  $t$  et  $u$  deux termes du langage et  $P$  une formule quelconque. On rappelle que  $P[x \leftarrow t]$  représente la formule  $P$  dans laquelle on remplace la variable libre  $x$  par le terme  $t$ .

Montrer par récurrence sur la structure de la formule, que la règle de substitution dans un contexte arbitraire est valide :

$$t = u \Rightarrow P[x \leftarrow t] \Rightarrow P[x \leftarrow u]$$

- On pourra faire la preuve dans le cas particulier d'une signature qui ne contient qu'une constante  $a$ , un symbole de fonction binaire  $g$  et un symbole de prédicat binaire  $Q$ .
- On donnera l'ensemble des axiomes de la théorie de l'égalité associée à ces symboles.
- On commencera par montrer par récurrence sur la structure du terme  $v$  que la formule suivante est valide :

$$t = u \Rightarrow v[x \leftarrow t] = v[x \leftarrow u]$$

### Compétences 1.5

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

## Chapitre 2

# Donner du sens aux formules

Le chapitre précédent nous a permis d'introduire les bases de la logique du premier ordre. Nous avons principalement vu la structure des termes et des formules et comment utiliser ce langage pour modéliser des situations ou problèmes. La vérité d'une formule logique dépend du monde dans lequel on l'interprète. Nous revenons ici sur la notion d'interprétation de manière plus détaillée et définissons la notion de conséquence logique et d'équivalence. Nous étudierons ensuite quelques modèles particuliers.

### 2.1 Interprétations et vérité

Nous nous plaçons dans la suite dans un langage logique sur une signature  $(\mathcal{F}, \mathcal{R})$  avec  $\mathcal{F}$  l'ensemble des symboles de fonctions qui servent à construire les termes et  $\mathcal{R}$  l'ensemble des symboles de prédicat qui permettent de construire les formules atomiques.

#### 2.1.1 Définitions

**Définition 2.1.1 (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 que  $f_I \in \mathcal{D}^n \rightarrow \mathcal{D}$ , pour chaque  $x_1, \dots, x_n \in \mathcal{D}$ , on a  $f_I(x_1, \dots, x_n) \in \mathcal{D}$ ) ;
- pour chaque symbole de relation  $R \in \mathcal{R}_n$  d'arité  $n$ , une relation  $n$ -aire  $R_I$  sur  $\mathcal{D}$  (c'est-à-dire que  $R_I \subseteq \mathcal{D}^n$  est un ensemble de  $n$ -uplets  $(v_1, \dots, v_n)$  de valeurs dans  $\mathcal{D}$  qui correspond à tous les cas pour lesquels on considère, dans cette interprétation, que la relation  $R$  est vraie).

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

**Cas des symboles d'arité 0.** Dans la définition ci-dessus, on peut regarder de plus près le cas des symboles d'arité 0.

Un symbole de fonction d'arité 0 est aussi appelé constante et un symbole de prédicat d'arité 0 est aussi appelé variable propositionnelle. Leur interprétation rentre dans le cas général. En effet l'ensemble  $\mathcal{D}^0$  contient exactement un élément, à savoir le 0-uplet noté  $()$  avec aucune valeur à l'intérieur.

- Pour une constante  $c \in \mathcal{F}_0$ , définir une fonction dans  $\mathcal{D}^0 \rightarrow \mathcal{D}$  revient à donner la valeur de la fonction pour  $()$ , et donc revient à se donner une constante  $c_I$  du domaine  $\mathcal{D}$ .
- Pour un symbole de prédicat  $X \in \mathcal{R}_0$ , l'interprétation  $X_I$  est une relation sur  $\mathcal{D}^0$ , c'est-à-dire un sous-ensemble d'un ensemble à un élément  $\{()\}$ . Il y a exactement deux relations possibles sur un ensemble qui a un seul élément. La relation vide, qui correspond au cas où  $X$  est interprété comme faux et la relation avec l'ensemble complet qui correspond au cas où  $X$  est interprété comme vrai.

**Interprétation et implantation.** Du point de vue informatique, on peut voir les différentes interprétations d'une même signature comme les différentes implémentations d'une même bibliothèque. L'implantation d'un symbole de prédicat est une fonction booléenne sur les termes.

**Exemple 2.1** Par exemple, les langages de programmation ont une notion primitive d'entiers relatifs mais pas de nombres rationnels. On peut se donner une signature pour manipuler des rationnels qui contiendra des constantes  $0$ ,  $1$  et  $-1$ , une opération binaire de construction  $\text{frac}$ , des opérations binaires  $+$  et  $*$  et un symbole de prédicat binaire pour l'égalité.

On peut proposer une interprétation de cette signature. Il faut décider comment on va représenter un rationnel, on choisit de le faire en prenant 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  $\mathcal{D} \stackrel{\text{def}}{=} \mathbb{Z} \times \mathbb{N} \setminus \{0\}$ . D'autres choix sont possibles comme d'imposer que la fraction soit réduite (pas de diviseur commun du dénominateur et du numérateur), ou encore autoriser un entier relatif en dénominateur.

On pourra définir alors une interprétation de chacun des symboles de notre signature :

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

### Interprétation des symboles de prédicats d'arité 0, 1 ou 2 ou plus

- Un symbole de prédicat d'arité 0 aura pour interprétation soit vrai, soit faux
- Un symbole de prédicat d'arité 1 aura pour interprétation n'importe quel sous-ensemble du domaine  $\mathcal{D}$ .
- Un symbole  $R$  de prédicat d'arité 2 aura pour interprétation n'importe quelle relation binaire du domaine  $\mathcal{D}$ . Une telle relation se visualise sous forme de graphe orienté dont les sommets sont les éléments de  $\mathcal{D}$  et dans lequel on met une arête entre deux sommets  $a$  et  $b$  exactement lorsque on souhaite que l'interprétation  $R_I(a, b)$  soit vraie. Le graphe pourra être fini ou infini.
- Si on a un symbole de prédicat  $R$  d'arité plus grande, par exemple 4 alors l'interprétation est un ensemble de quadruplets  $(a, b, c, d)$ . On peut représenter cette relation comme une table avec 4 colonnes. Les lignes de cette table seront formées de tous les quadruplets  $(a, b, c, d)$  pour lesquels l'interprétation  $R_I(a, b, c, d)$  est vraie. Cette représentation ne peut être utilisée que si  $R_I$  est vraie sur seulement un nombre fini de quadruplets.

#### 2.1.2 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.


#### 2.1.3 Valeur de vérité d'une formule

Se donner une interprétation de la signature est une étape nécessaire mais pas suffisante pour pouvoir définir la valeur de vérité d'une formule quelconque. En effet celle-ci peut contenir des variables libres auxquelles il

faut aussi attribuer une valeur. Dans la suite, on note  $\mathcal{X}$  l'ensemble des variables qui apparaissent dans les termes et les formules.

**Définition 2.1.2 (Environnement)** 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  $\rho \in \mathcal{X} \rightarrow \mathcal{D}$ ).

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  $y \neq x$ .

 Il ne faut pas confondre une substitution qui associe un terme syntactique à une variable avec un environnement qui associe à une variable une valeur sémantique du domaine d'interprétation.

Pour faire une analogie informatique, la substitution est analogue à une opération de remplacement que l'on peut faire dans un éditeur, alors que l'environnement va correspondre à l'état de la mémoire pour chaque variable du programme au moment de son exécution.

Etant données une signature  $(\mathcal{F}, \mathcal{R})$  et une interprétation  $I \stackrel{\text{def}}{=} (\mathcal{D}, (f_I)_{f \in \mathcal{F}}, (R_I)_{R \in \mathcal{R}})$  de cette signature (que l'on supposera fixée dans la suite) on peut définir en fonction d'un environnement  $\rho$ , la valeur d'un terme et d'une formule.

**Définition 2.1.3 (Valeur d'un terme et d'une formule dans une interprétation)** 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  $\mathcal{D}$  :

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

- 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  $\{V, F\}$  :

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

On notera  $I, \rho \models A$  lorsque la formule  $A$  est vraie dans l'interprétation  $I$  et l'environnement  $\rho$ , c'est-à-dire lorsque  $\text{val}_I(\rho, A) = V$ . Si la formule  $A$  est close, alors sa valeur dans une interprétation ne dépend pas de l'environnement (voir proposition 2.1.2) et on écrira simplement  $I \models A$  pour indiquer que  $A$  est vraie dans l'interprétation  $I$  (et n'importe quel environnement).

**Proposition 2.1.1** Soit  $I$  une interprétation  $I$  et  $\rho$  un environnement.

- $I, \rho \not\models \perp$
- $I, \rho \models \top$
- $I, \rho \models R(t_1, \dots, t_n)$  si et seulement si  $(\text{val}_I(\rho, t_1), \dots, \text{val}_I(\rho, t_n))$  appartient à l'interprétation de  $R$
- $I, \rho \models \neg A$  si et seulement si  $I, \rho \not\models A$

- $I, \rho \models A \wedge B$  si et seulement si  $I, \rho \models A$  et  $I, \rho \models B$
- $I, \rho \models A \vee B$  si et seulement si  $I, \rho \models A$  ou  $I, \rho \models B$
- $I, \rho \models A \Rightarrow B$  si et seulement si  $I, \rho \not\models A$  ou  $I, \rho \models B$
- $I, \rho \models \forall x, A$  si et seulement si pour tout  $d \in \mathcal{D}$ , on a  $I, \rho + \{x \mapsto d\} \models A$
- $I, \rho \models \exists x, A$  si et seulement si il existe  $d \in \mathcal{D}$  tel que  $I, \rho + \{x \mapsto d\} \models A$

**Preuve:** Ces propriétés sont une simple reformulation de la définition de la valeur d'une formule et du fait que  $I, \rho \models A$  est défini comme  $\text{val}_I(\rho, A) = V$   $\square$

Contrairement au cas de la logique propositionnelle, on ne peut pas en général calculer cette valeur de vérité car dans le cas des quantificateurs, il faut a priori raisonner sur l'ensemble des objets  $d \in \mathcal{D}$  qui peut être infini.

**Exemple 2.2** Les bases de données traitent généralement de problèmes sur des ensembles finis. On peut s'intéresser à un ensemble de personnes et vouloir représenter les relations de parenté (filiation et liens entre frères et soeurs).

On modélise une telle base de données en logique du premier ordre. On suppose que la signature est juste composée d'un ensemble de constantes pour représenter les individus, par exemple *Alice*, *Bob*, *Charles*, *Denis*, *Eric*, *Fanny*, *Georges* et *Hélène* et de deux symboles de relation binaire *parent* et *fratrie* ainsi qu'un symbole d'égalité binaire. La relation *parent*( $x, y$ ) exprime que  $x$  est le père ou la mère de  $y$  et *fratrie*( $x, y$ ) exprime que  $x$  et  $y$  sont dans la même fratrie.

Le modèle peut utiliser comme domaine l'ensemble des constantes

$\{\text{Alice}, \text{Bob}, \text{Charles}, \text{Denis}, \text{Eric}, \text{Fanny}, \text{Georges}, \text{Hélène}\}$

ou bien une autre représentation comme des entiers par exemple, ou des caractères.

On va prendre ici  $\mathcal{D} \stackrel{\text{def}}{=} \{A, B, C, D, E, F, G, H\}$ . On interprète chaque constante par un élément de  $\mathcal{D}$ . Il serait parfaitement légitime de choisir une interprétation qui associe le même élément du domaine à deux constantes différentes du langage (par exemple si un individu est connu sous deux identités différentes). Ici on choisit de représenter chaque constante du langage par un élément différent du domaine.

$\text{Alice}_I = A \quad \text{Bob}_I = B \quad \text{Charles}_I = C \quad \text{Denis}_I = D$   
 $\text{Eric}_I = E \quad \text{Fanny}_I = F \quad \text{Georges}_I = G \quad \text{Hélène}_I = H$

On a ensuite le choix sur l'interprétation des relations, par exemple :

$\text{parent}_I = \{(A, B), (G, B), (B, C), (B, H)\}$   
 $\text{fratrie}_I = \{(C, H), (H, C)\}$   
 $=_I = \{(A, A), (B, B), (C, C), (D, D), (E, E), (F, F), (G, G), (H, H)\}$

On s'attend à ce que cette interprétation satisfasse les propriétés suivantes :

- la relation de fratrie est symétrique ;
- deux personnes sont dans la même fratrie si et seulement si elles ont un parent commun ;
- une personne ne peut avoir plus de deux parents.

**Exercice 2.1** Soient les formules suivantes sans variable libre. Sont-elles vraies si on les interprète dans  $\mathbb{N}$ ,  $\mathbb{Z}$ ,  $\mathbb{Q}$  avec les conventions usuelles pour l'interprétation des opérations et des relations.

1.  $\forall x, \exists y, x < y$
2.  $\forall x, \exists y, y < x$
3.  $\forall x, y, x < y \Rightarrow x + 1 \leq y$
4.  $\forall x, y, z, x \leq y \Rightarrow x \times z \leq y \times z$

On constate que des propriétés vraies ou fausses dans une interprétation ne le sont pas forcément dans une autre, même si les interprétations se correspondent sur les formules atomiques. Cela vient du fait que les quantifications ne vont pas faire référence aux mêmes ensembles sous-jacent. En mathématiques "usuelles" (qui s'appuient implicitement sur la théorie des ensembles) on distinguerait les formules en relativisant les quantificateurs. Par exemple  $\forall x \in \mathbb{N}, \exists y \in \mathbb{N}, y < x$  (qui est faux) versus  $\forall x \in \mathbb{Z}, \exists y \in \mathbb{Z}, y < x$  qui est vrai.



### Propriétés de la valeur d'une formule dans une interprétation

On vérifie que la définition de la valeur de vérité d'une formule est bien compatible avec l'opération de renommage des variables liées dans les formules. C'est-à-dire que si  $y \notin \text{VI}(P)$  et  $P[x \leftarrow y]$  est bien définie alors  $\text{val}_I(\rho, (\forall x, P)) = \text{val}_I(\rho, (\forall y, P[x \leftarrow y]))$  et  $\text{val}_I(\rho, (\exists x, P)) = \text{val}_I(\rho, (\exists y, P[x \leftarrow y]))$

**Proposition 2.1.2** *La valeur d'un terme ne dépend que de la valeur de l'environnement sur les variables de ce terme et de l'interprétation des symboles de fonction et des constantes qui apparaissent dans ce terme.*

*La valeur de vérité d'une formule ne dépend que de la valeur de l'environnement sur les variables libres de cette formule et de l'interprétation des symboles de fonction, constantes et relations qui apparaissent dans la formule.*

**Preuve:** La preuve se fait sans difficulté par récurrence sur la structure du terme puis de la formule.  $\square$

En particulier pour un terme clos ou une formule close, la valeur est indépendante de l'environnement et on écrira simplement  $\text{val}_I(P)$  au lieu de  $\text{val}_I(\rho, P)$ . On omettra également l'indice  $I$  lorsque l'interprétation est implicite dans le contexte.

#### 2.1.4 Lien entre substitution et valeur de vérité

La propriété ci-dessous fait le lien entre substitution et environnement. On suppose que l'on a une formule  $A$  qui contient une variable libre  $x$  et un terme  $t$ . Remplacer  $x$  par  $t$  dans  $A$  puis calculer la valeur de  $A[x \leftarrow t]$  revient à d'abord évaluer le terme  $t$  en une valeur  $v$  puis calculer la valeur de  $A$  (qui contient la variable  $x$  libre) dans un environnement où  $x$  a la valeur  $v$ .

**Proposition 2.1.3 (Vérité et substitution)** *Soit une formule  $P$  qui contient une variable libre  $x$  et soit  $t$  un terme. On suppose que la formule substituée  $P[x \leftarrow t]$  est définie (pas de capture). Soit  $\rho$  un environnement. Interpréter la formule dans laquelle on remplace  $x$  par  $t$  revient à interpréter la formule de base en complétant l'environnement avec pour valeur associée à la variable  $x$ , l'interprétation du terme  $t$ .*

$$\text{val}_I(\rho, P[x \leftarrow t]) = \text{val}_I(\rho + \{x \mapsto \text{val}_I(\rho, t)\}, P)$$

**Preuve:** La preuve se fait en montrant d'abord un résultat analogue pour les termes.

Soit  $u$  un terme, on a :

$$\text{val}_I(\rho, u[x \leftarrow t]) = \text{val}_I(\rho + \{x \mapsto \text{val}_I(\rho, t)\}, u)$$

Ce premier résultat se montre par simple récurrence sur la structure du terme  $u$ , de la manière suivante :

— Si  $u$  est la variable  $x$  alors  $u[x \leftarrow t] = t$  et la formule à montrer se réécrit en

$$\text{val}_I(\rho, t) = \text{val}_I(\rho + \{x \mapsto \text{val}_I(\rho, t)\}, x) = (\rho + \{x \mapsto \text{val}_I(\rho, t)\})(x)$$

qui est vrai par définition de  $\rho + \{x \mapsto \text{val}_I(\rho, t)\}$ .

— Si  $u$  est une variable  $y$  différente de  $x$ , alors  $y[x \leftarrow t] = y$  et la formule à montrer se réécrit en

$$\text{val}_I(\rho, y) = \text{val}_I(\rho + \{x \mapsto \text{val}_I(\rho, t)\}, y)$$

qui est vraie par définition de  $\rho + \{x \mapsto \text{val}_I(\rho, t)\}$  (les deux expressions sont égales à  $\rho(y)$ ).

— Si  $u$  est une constante  $c$  alors  $c[x \leftarrow t] = c$  et la formule à montrer se réécrit en

$$\text{val}_I(\rho, c) = \text{val}_I(\rho + \{x \mapsto \text{val}_I(\rho, t)\}, c)$$

qui est vraie par définition de  $\text{val}_I$  (les deux expressions sont égales à  $c$ ).

- Si  $u$  est une expression  $f(u_1, \dots, u_n)$  alors  $f(u_1, \dots, u_n)[x \leftarrow t] = f(u_1[x \leftarrow t], \dots, u_n[x \leftarrow t])$  et la formule à montrer se réécrit en

$$\text{val}_I(\rho, f(u_1[x \leftarrow t], \dots, u_n[x \leftarrow t])) = \text{val}_I(\rho + \{x \mapsto \text{val}_I(\rho, t)\}, f(u_1, \dots, u_n))$$

On a alors

$$\begin{aligned} & \text{val}_I(\rho, f(u_1[x \leftarrow t], \dots, u_n[x \leftarrow t])) \\ &= f_I(\text{val}_I(\rho, u_1[x \leftarrow t]), \dots, \text{val}_I(\rho, u_n[x \leftarrow t])) && \text{def de val}_I \\ &= f_I(\text{val}_I(\rho + \{x \mapsto \text{val}_I(\rho, t)\}, u_1), \dots, \text{val}_I(\rho + \{x \mapsto \text{val}_I(\rho, t)\}, u_n)) && \text{hyp. de récurrence} \\ &= \text{val}_I(\rho + \{x \mapsto \text{val}_I(\rho, t)\}, f(u_1, \dots, u_n)) && \text{def de val}_I \end{aligned}$$

On montre ensuite par récurrence sur la structure de la formule  $P$  que pour n'importe quel environnement  $\rho$ , on a

$$\text{val}_I(\rho, P[x \leftarrow t]) = \text{val}_I(\rho + \{x \mapsto \text{val}_I(\rho, t)\}, P)$$

On ne montre ici que le cas des formules atomiques  $R(u_1, \dots, u_n)$  et le cas d'une formule quantifiée  $\forall x, A$ .

- Le résultat établi pour les termes permet de montrer le cas des formules atomiques car

$$\begin{aligned} & \text{val}_I(\rho, R(u_1, \dots, u_n)[x \leftarrow t]) \\ &= \text{val}_I(\rho, R(u_1[x \leftarrow t], \dots, u_n[x \leftarrow t])) && \text{def de substitution} \\ &= R_I(\text{val}_I(\rho, u_1[x \leftarrow t]), \dots, \text{val}_I(\rho, u_n[x \leftarrow t])) && \text{def de val}_I \\ &= R_I(\text{val}_I(\rho + \{x \mapsto \text{val}_I(\rho, t)\}, u_1), \dots, \text{val}_I(\rho + \{x \mapsto \text{val}_I(\rho, t)\}, u_n)) && \text{prop. des termes} \\ &= \text{val}_I(\rho + \{x \mapsto \text{val}_I(\rho, t)\}, R(u_1, \dots, u_n)) && \text{def de val}_I \end{aligned}$$

- Supposons que  $P$  est de la forme  $\forall z, A$  et on suppose que  $z \neq x$  (sinon la substitution est constante et le résultat immédiat) et que la substitution est bien définie à savoir la variable  $z$  n'apparaît pas libre dans  $t$ . On a alors  $(\forall z, A)[x \leftarrow t] = \forall z, (A[x \leftarrow t])$ .

Pour calculer la valeur d'une formule  $\forall z, B$  dans un environnement  $\rho'$  quelconque, on fixe un élément  $d \in \mathcal{D}$  et on regarde la valeur de la formule  $B$  dans l'environnement  $\rho' + \{z \mapsto d\}$ .

On se donne donc un environnement  $\rho$  et pour calculer  $\text{val}_I(\rho, \forall z, (A[x \leftarrow t]))$  on se donne  $d \in \mathcal{D}$  et on calcule

$$\begin{aligned} & \text{val}_I(\rho + \{z \mapsto d\}, A[x \leftarrow t]) \\ &= \text{val}_I((\rho + \{z \mapsto d\}) + \{x \mapsto \text{val}_I(\rho + \{z \mapsto d\}, t)\}, A) && \text{hypothèse de récurrence} \\ &= \text{val}_I((\rho + \{z \mapsto d\}) + \{x \mapsto \text{val}_I(\rho, t)\}, A) && \text{car } z \notin \text{vars}(t) \\ &= \text{val}_I((\rho + \{x \mapsto \text{val}_I(\rho, t)\}) + \{z \mapsto d\}, A) && \text{car } x \neq z \end{aligned}$$

Donc pour tout  $d$ ,  $\text{val}_I(\rho + \{z \mapsto d\}, A[x \leftarrow t])$  est vraie si et seulement si  $\text{val}_I((\rho + \{x \mapsto \text{val}_I(\rho, t)\}) + \{z \mapsto d\}, A)$  est vrai. On en déduit que  $\text{val}_I(\rho, \forall z, (A[x \leftarrow t]))$  est vrai si et seulement si  $\text{val}_I(\rho + \{x \mapsto \text{val}_I(\rho, t)\}, \forall z, A)$  est vrai, qui est le résultat souhaité.

Le même raisonnement permet de conclure dans le cas d'un quantificateur existentiel. La seule subtilité dans la preuve est que l'hypothèse de récurrence est appliquée à l'environnement  $\rho + \{z \mapsto d\}$  au lieu de  $\rho$  et donc que la propriété que l'on cherche à montrer par récurrence sur la formule  $P$  doit bien inclure cette possibilité de faire varier l'environnement.

□

## 2.2 Validité et satisfiabilité

La vérité d'une formule se définit par rapport à une interprétation qui fixe le sens des symboles de fonctions et de prédicats et un environnement qui détermine la valeur des variables libres. Si la formule n'a pas de



variables libres, sa vérité est la même quelque soit l'environnement et ne dépend que de l'interprétation des symboles.

Lorsqu'on a juste une formule, elle peut être vraie dans toutes les interprétations et tous les environnements possibles, dans aucune interprétation et aucun environnement ou bien être vraie dans certaines interprétations et certains environnements et fausse dans d'autres.

### 2.2.1 Définitions

**Définition 2.2.1 (Validité, satisfiabilité, modèle)** Soit  $A$ , une formule du calcul des prédicats sur une signature  $(\mathcal{F}, \mathcal{R})$ .

- La formule est dite *valide* (on dit aussi que c'est une *tautologie*) si sa valeur de vérité est vraie pour toute interprétation de la signature et tout environnement.
- La formule est dite *satisfiable* si sa valeur de vérité est vraie pour au moins une interprétation de la signature et un environnement. Une interprétation et un environnement qui rendent vraie la formule forment un *modèle de la formule*.
- La formule est dite *insatisfiable* (on dit aussi *contradictoire*) si sa valeur de vérité est fausse pour toute interprétation de la signature et tout environnement.

On étend ces notions à un ensemble  $\mathcal{E}$  fini ou infini de formules :

- L'ensemble  $\mathcal{E}$  est *valide* si pour toute interprétation  $I$ , tout environnement  $\rho$  et toute formule  $P \in \mathcal{E}$  on a  $I, \rho \models P$  (toutes les formules sont vraies dans toutes les interprétations).
- L'ensemble  $\mathcal{E}$  est *satisfiable* s'il existe une interprétation  $I$  et un environnement  $\rho$  tels que pour toute formule  $P \in \mathcal{E}$  on a  $I, \rho \models P$  (il existe une interprétation qui rend vraies toutes les formules de  $\mathcal{E}$ ).
- L'ensemble  $\mathcal{E}$  est *insatisfiable* si pour toute interprétation  $I$  et tout environnement  $\rho$ , il existe une formule  $P \in \mathcal{E}$  telle que  $I, \rho \not\models P$  (il n'existe pas d'interprétation et d'environnement qui rend vraies toutes les formules ou de manière équivalente, toute interprétation et environnement rendent fausse au moins une formule de  $\mathcal{E}$ ).

Une formule valide est a fortiori satisfiable mais le contraire est faux. Par contre la négation nous permet de passer de la validité à la satisfiabilité.

**Proposition 2.2.1 (Lien entre validité et satisfiabilité)** La formule  $A$  est valide si et seulement si  $\neg A$  est insatisfiable.

**Preuve:** La formule  $A$  est valide si et seulement si la formule  $A$  est vraie dans toute interprétation de la signature et tout environnement. Ceci est équivalent à dire que la formule  $\neg A$  est fausse dans toute interprétation de la signature et tout environnement, ce qui est la définition de insatisfiable.  $\square$

**Exercice 2.2** Dire si les formules suivantes sont valides, satisfiables, insatisfiables :

- $(\exists x, \neg P(x)) \Rightarrow \neg \forall x, P(x)$
- $(\exists x, \neg P(x)) \Rightarrow \exists x, \neg Q(x)$
- $(\exists x, \neg P(x)) \wedge \forall x, P(x)$

### Proposition 2.2.2

- Une formule  $A$  avec un variable libre  $x$  est valide si et seulement si la formule  $\forall x, A$  est valide.
- Une formule  $A$  avec un variable libre  $x$  est satisfiable si et seulement si la formule  $\exists x, A$  est satisfiable.
- Une formule  $A$  avec un variable libre  $x$  est insatisfiable si et seulement si la formule  $\exists x, A$  est insatisfiable.

### 2.2.2 Substitution et validité

La validité et l'insatisfiabilité qui sont des propriétés de toutes les interprétations ne changent pas lorsque l'on effectue une substitution dans une formule ou un ensemble de formules.

### Remplacement d'une variable par un terme

- Si une formule  $P$  est valide (resp. insatisfiable) alors  $P[x \leftarrow t]$  est valide (resp. insatisfiable).
- Si un ensemble de formules  $\mathcal{E}$  est valide (resp. insatisfiable) alors  $\mathcal{E}[x \leftarrow t] \stackrel{\text{def}}{=} \{P[x \leftarrow t] \mid P \in \mathcal{E}\}$  est valide (resp. insatisfiable).

Les réciproques sont fausses. Si on prend une signature avec un prédicat unaire  $Q$  et une constante  $c$ . La formule  $Q(x) \Rightarrow Q(c)$  n'est pas valide. En effet il suffit de prendre une interprétation dont le domaine contient exactement deux éléments  $0, 1$  et dans laquelle on interprète la constante  $c$  par la valeur  $0$ . On choisit d'avoir  $Q_I(1)$  vrai et  $Q_I(0)$  faux. Si  $Q(x) \Rightarrow Q(c)$  était valide alors la formule serait vraie dans l'interprétation ainsi définie et avec un environnement dans lequel  $x$  a la valeur  $1$ , ce qui n'est pas le cas. Maintenant si on substitue la variable  $x$  par la constante  $c$  on obtient la formule  $Q(c) \Rightarrow Q(c)$  qui est évidemment valide.

On a le même comportement avec l'insatisfiabilité en choisissant comme formule  $P$  la formule  $Q(x) \wedge \neg Q(c)$ . La formule  $P$  elle-même est satisfiable (prendre la même interprétation que précédemment). Par contre si on remplace  $x$  par la constante  $c$  on obtient la formule  $Q(c) \wedge \neg Q(c)$  qui est évidemment insatisfiable.

La satisfiabilité simple n'est pas préservée, au contraire on a que si  $P[x \leftarrow t]$  est satisfiable alors  $P$  est satisfiable mais le contraire est faux comme le montre l'exemple précédent.

### Remplacement d'un symbole de prédicat par une formule

Lorsqu'on écrit une formule comme  $A \vee \neg A$ , le symbole  $A$  peut correspondre à une variable mathématique qui représente n'importe quelle formule de la logique, ou au contraire le symbole  $A$  peut être une variable propositionnelle (symbole de prédicat d'arité 0), auquel cas on a exactement une formule syntaxique qui est différente d'autres formules de même format comme  $\perp \vee \neg \perp$ . Nous allons montrer que l'ambiguïté entre ces deux points de vue ne pose pas de problème, car on peut passer de l'un à l'autre sans difficulté.

**Remplacer une variable propositionnelle par une formule.** Généralisons l'exemple de  $A \vee \neg A$  et supposons que l'on a une formule  $P$  dans laquelle il y a une variable propositionnelle  $X$ . On suppose que  $P$  est valide et on veut montrer que si on remplace la variable  $X$  par n'importe quelle formule  $A$  alors la formule obtenue reste valide.

On commence par définir l'opération de remplacement d'une variable propositionnelle par une formule.

**Définition 2.2.2 (Remplacement d'une variable propositionnelle par une formule,  $P[X \leftarrow Q]$ )** Soit  $P$  et  $Q$  des formules et  $X$  une variable propositionnelle, le remplacement de  $X$  par  $Q$  dans  $P$  est une formule notée  $P[X \leftarrow Q]$  qui est définie de manière récursive sur la structure de  $P$ .

- Si  $P$  est une formule atomique : si  $P$  est la variable propositionnelle  $X$  alors  $P[X \leftarrow Q] = Q$  sinon  $P[X \leftarrow Q] = P$
- Si  $P$  est de la forme  $\neg A$  alors  $P[X \leftarrow Q] = \neg(A[X \leftarrow Q])$
- Si  $P$  est de la forme  $A \circ B$  avec  $\circ$  l'une des connecteurs propositionnels alors  $P[X \leftarrow Q] = (A[X \leftarrow Q]) \circ (B[X \leftarrow Q])$
- Si  $P$  est de la forme  $\forall x, A$  (resp.  $\exists x, A$ ) avec la variable  $x$  qui n'est pas libre dans la formule  $Q$ , alors  $P[X \leftarrow Q] = \forall x, (A[X \leftarrow Q])$  (resp.  $P[X \leftarrow Q] = \exists x, (A[X \leftarrow Q])$ )

**Proposition 2.2.3** Soit  $P$  et  $Q$  des formules et  $X$  une variable propositionnelle, si  $P$  est une formule valide (resp. insatisfiable) alors il en est de même de la formule  $P[X \leftarrow Q]$  obtenue en remplaçant  $X$  par  $Q$  dans  $P$ .

**Preuve:** Nous allons faire la preuve dans le cas de la validité. Le résultat découle du fait que dans une interprétation  $I$  et un environnement  $\rho$  quelconques, la valeur de  $P[X \leftarrow Q]$  est égale à la valeur de  $P$  dans un environnement dans lequel la variable  $X$  a pour interprétation la valeur de  $Q$ . Ce résultat s'obtient facilement par récurrence sur la structure de  $P$ .

Maintenant si une formule est valide, elle est vraie tout le temps donc en particulier si on fixe une valeur pour une de ses variables (ici  $X$ ).  $\square$

**Remplacer un symbole de prédicat par une formule paramétrée.** On peut généraliser la construction précédente pour remplacer un symbole de prédicat par une formule. Le symbole de prédicat est utilisé associé à des termes, ce qui nous permet d'utiliser des formules paramétrées par des variables qui pourront être instanciées dans le remplacement.

Soit une formule  $P$  qui est construite sur une signature qui comporte un symbole de relation  $n$ -aire  $R$ . Soit une formule  $Q$  et  $n$  variables  $x_1, \dots, x_n$ . On peut remplacer dans la formule  $P$  toutes les sous-formules atomiques de la forme  $R(t_1, \dots, t_n)$  par la formule  $Q[x_1 \leftarrow t_1, \dots, x_n \leftarrow t_n]$ . On note le résultat de cette opération  $P[R(x_1, \dots, x_n) \leftarrow Q]$ .

**Exemple 2.3** Soit une signature avec une constante  $c$  et un symbole de prédicat unaire  $Q$  et un symbole d'égalité binaire.

Soit  $A$  la formule  $P(c) \Rightarrow \exists x, P(x)$  alors  $A[P(z) \leftarrow z = z]$  est défini comme la formule :

$$(c = c) \Rightarrow \exists x, x = x$$

La définition reprend exactement les lignes du remplacement d'une variable propositionnelle sauf dans le cas de base dans lequel il faut, de plus, remplacer les paramètres de la formule  $Q$  par les arguments du symbole de prédicat  $R$ .

**Définition 2.2.3 (Remplacement d'un symbole de prédicat par une formule,  $P[R(x_1, \dots, x_n) \leftarrow Q]$ )** Soit  $P$  et  $Q$  des formules,  $R$  un symbole de prédicat d'arité  $n$  et  $x_1, \dots, x_n$ ,  $n$  variables d'objet. Le remplacement de  $R$  par  $Q$  paramétré par  $x_1, \dots, x_n$  dans  $P$  est une formule notée  $P[R(x_1, \dots, x_n) \leftarrow Q]$  qui est définie de manière récursive sur la structure de  $P$ .

- Si  $P$  est une formule atomique alors si  $P$  est de la forme  $R(t_1, \dots, t_n)$  on a  $P[R(x_1, \dots, x_n) \leftarrow Q] = Q[x_1 \leftarrow t_1, \dots, x_n \leftarrow t_n]$  sinon  $P[R(x_1, \dots, x_n) \leftarrow Q] = P$
- Si  $P$  est de la forme  $\neg A$  alors  $P[R(x_1, \dots, x_n) \leftarrow Q] = \neg(A[R(x_1, \dots, x_n) \leftarrow Q])$
- Si  $P$  est de la forme  $A \circ B$  avec  $\circ$  l'une des connecteurs propositionnels alors  $P[R(x_1, \dots, x_n) \leftarrow Q] = (A[R(x_1, \dots, x_n) \leftarrow Q]) \circ (B[R(x_1, \dots, x_n) \leftarrow Q])$
- Si  $P$  est de la forme  $\forall x, A$  (resp.  $\exists x, A$ ) avec la variable  $x$  qui n'est pas libre dans la formule  $Q$ , alors  $P[R(x_1, \dots, x_n) \leftarrow Q] = \forall x, (A[R(x_1, \dots, x_n) \leftarrow Q])$  (resp.  $P[R(x_1, \dots, x_n) \leftarrow Q] = \exists x, (A[R(x_1, \dots, x_n) \leftarrow Q])$ )

Tout comme dans le cas d'une variable propositionnelle, la validité est préservée lorsqu'on remplace un symbole de prédicat par une formule arbitraire.

**Proposition 2.2.4** Soit une formule  $P$  qui est construite sur une signature qui comporte un symbole de relation  $n$ -aire  $R$ . Soit une formule  $Q$  et  $n$  variables  $x_1, \dots, x_n$ .

Si  $P$  est valide (resp. insatisfiable) alors il en est de même de la formule  $P[R(x_1, \dots, x_n) \leftarrow Q]$ .

**Preuve:** On se donne une interprétation quelconque  $I$  pour les symboles de la formule  $P[R(x_1, \dots, x_n) \leftarrow Q]$  et un environnement  $\rho$  pour les variables libres. On doit montrer que dans cette interprétation la formule  $P[R(x_1, \dots, x_n) \leftarrow Q]$  est vraie (resp. fausse).

Tous les symboles et les variables qui apparaissent dans  $P$  apparaissent aussi dans  $P[R(x_1, \dots, x_n) \leftarrow Q]$  à l'exception peut-être de  $R$ . On construit une interprétation  $I'$  qui est définie comme  $I$  pour tous les symboles sauf dans le cas de  $R$  pour lequel on pose  $R_{I'}(d_1, \dots, d_n)$  est vrai si et seulement si  $\text{val}_I(\rho + \{x_1 \mapsto d_1, \dots, x_n \mapsto d_n\}, Q)$  est vrai.

Pour n'importe quel environnement  $\rho$ , on a

$$\text{val}_{I'}(\rho, P) = \text{val}_I(\rho, P[R(x_1, \dots, x_n) \leftarrow Q])$$

(cette propriété se montre par récurrence sur la structure de la formule  $P$ ).

Si  $P$  est valide, on a  $\text{val}_I(\rho, P)$  est vrai. On en déduit que  $\text{val}_I(\rho, P[R(x_1, \dots, x_n) \leftarrow Q])$  est vrai et comme cela vaut pour toutes les interprétations  $I$ , on en déduit la validité de la formule  $P[R(x_1, \dots, x_n) \leftarrow Q]$ .

□

## 2.3 Conséquence logique, équivalence

**Définition 2.3.1 (Conséquence logique, équivalence)** Si  $\mathcal{E}$  est un ensemble de formules et  $A$  une formule, on dit que  $A$  est conséquence logique de  $\mathcal{E}$  et on note  $\mathcal{E} \models A$  si toute interprétation  $I$  et environnement  $\rho$  qui rendent vraies les formules de  $\mathcal{E}$  rendent aussi vraie la formule  $A$ . C'est-à-dire que si  $I, \rho \models B$  pour toutes les formules  $B \in \mathcal{E}$  (noté  $I, \rho \models \mathcal{E}$ ) alors  $I, \rho \models A$ .

On dit que  $A$  et  $B$  sont des formules équivalentes et on écrit  $A \equiv B$  si  $A \models B$  et  $B \models A$ . C'est-à-dire que pour toute interprétation  $I$  et environnement  $\rho$ , on a  $I, \rho \models A$  si et seulement si  $I, \rho \models B$ .

**Exemple 2.4** Soit le syllogisme “Tous les hommes sont mortels, Socrate est un homme, donc Socrate est mortel”, on peut le modéliser dans le cadre logique en se donnant comme signature deux symboles de prédicat unaires  $M$  et  $H$  pour représenter le fait d’être mortel et d’être un homme et une constante d’objet  $\text{Socrate}$ .

Le syllogisme, peut se représenter comme une conséquence logique

$$(\forall x, H(x) \Rightarrow M(x)), H(\text{Socrate}) \models M(\text{Socrate})$$

La propriété est vraie quelle que soit notre interprétation de la notion d’être mortel ou d’être un homme et quelle que soit notre interprétation de qui est Socrate.

Si on fait la même chose avec “Tous les hommes sont mortels, l’âne Francis n’est pas un homme donc l’âne Francis est immortel” avec une constante  $\text{AneFrancis}$ , on peut montrer au contraire qu’il n’y a pas de conséquence logique

$$(\forall x, H(x) \Rightarrow M(x)), \neg H(\text{AneFrancis}) \not\models \neg M(\text{AneFrancis})$$

Il suffit de se placer dans un univers où il n’y a pas d’homme et où tous les objets sont mortels.

On aura également

$$(\forall x, H(x) \Rightarrow M(x)), \neg H(\text{AneFrancis}) \not\models M(\text{AneFrancis})$$

en se plaçant dans un univers (de science fiction) où il n’y a pas d’homme et où tous les objets sont immortels.

### 2.3.1 Propriétés de la conséquence logique

**Proposition 2.3.1** Pour toute formule  $P$  et ensemble de formules  $\mathcal{E}$ , on a les propriétés suivantes :

- Pour toutes formules  $A_1, \dots, A_n$ , on montre que  $A_1, \dots, A_n \models P$  si et seulement si  $A_1 \wedge \dots \wedge A_n \Rightarrow P$  est valide
- si  $\mathcal{E}$  est un ensemble insatisfiable de formules alors pour toute formule  $P$ , on a  $\mathcal{E} \models P$
- $\mathcal{E}$  est un ensemble insatisfiable de formules si et seulement si  $\mathcal{E} \models \perp$
- $\mathcal{E} \models P$  si et seulement si l’ensemble de formules  $\mathcal{E} \cup \{\neg P\}$  est insatisfiable.
- $\mathcal{E}, P \models Q$  si et seulement si  $\mathcal{E} \models P \Rightarrow Q$

**Preuve:** On traite ici le cas de formules closes pour ne pas avoir à introduire d’environnement mais la preuve est identique s’il y a des variables en introduisant l’environnement  $\rho$  en plus de l’interprétation  $I$ .

- $A_1, \dots, A_n \models P$  si et seulement si toute interprétation qui rend vraies toutes les formules  $A_1, \dots, A_n$  rend vraie  $P$ . On considère une interprétation  $I$  quelconque. Si  $I$  rend vraies toutes les formules  $A_1, \dots, A_n$  alors elle rend vraie  $P$  et donc aussi  $A_1 \wedge \dots \wedge A_n \Rightarrow P$ . Si  $I$  rend faux une des formules  $A_1, \dots, A_n$  alors elle rend faux  $A_1 \wedge \dots \wedge A_n$  et donc elle rend vraie  $A_1 \wedge \dots \wedge A_n \Rightarrow P$ . On a donc montré que, quelle que soit l’interprétation, la formule  $A_1 \wedge \dots \wedge A_n \Rightarrow P$  était vraie et donc

cette formule est valide.

Réciproquement si  $A_1 \wedge \dots \wedge A_n \Rightarrow P$  est valide et si  $I$  est une interprétation qui rend vraies toutes les formules  $A_1, \dots, A_n$ , alors  $I$  rend vraie  $P$ .

- Si  $\mathcal{E}$  est un ensemble insatisfiable de formules, alors l'ensemble des interprétations qui rendent vraie les formules de  $\mathcal{E}$  est vide. Et donc n'importe quelle interprétation dans cet ensemble rend vraie la formule  $P$ . Par définition,  $\mathcal{E} \models P$  est alors vérifié.
- Etant donnée la question précédente, il suffit de montrer que si  $\mathcal{E} \models \perp$  alors  $\mathcal{E}$  est insatisfiable. Supposons que  $\mathcal{E}$  soit satisfiable, il y aurait alors une interprétation qui rend vraie les formules de  $\mathcal{E}$ , mais alors cette interprétation devrait rendre vraie la formule  $\perp$  ce qui est contraire à la définition de  $\perp$ .
- Supposons que  $\mathcal{E} \models P$  et montrons que toute interprétation  $I$  rend fausse l'une des formules de  $\mathcal{E} \cup \{\neg P\}$ . Soit  $I$  rend fausse l'une des formules de  $\mathcal{E}$  et donc a fortiori une des formules de  $\mathcal{E} \cup \{\neg P\}$ , soit  $I$  rend vraies toutes les formules de  $\mathcal{E}$  et donc aussi  $P$  et donc  $I$  rend fausse la formule  $\neg P$ . Réciproquement si  $\mathcal{E} \cup \{\neg P\}$  est insatisfiable, soit  $I$  une interprétation qui rend vraies toutes les formules de  $\mathcal{E}$ , elle rend fausse la formule  $\neg P$  (sinon  $\mathcal{E} \cup \{\neg P\}$  serait satisfiable) et donc elle rend vraie la formule  $P$  et donc  $\mathcal{E} \models P$ .
- Soit une interprétation  $I$ .
  - Si  $\mathcal{E}, P \models Q$  et  $I \models \mathcal{E}$ ,
    - si  $I \models P$  alors  $I \models \mathcal{E}, P$  donc  $I \models Q$  et donc  $I \models P \Rightarrow Q$  d'où le résultat
    - si  $I \not\models P$  alors  $P$  est faux dans l'interprétation  $I$  donc  $P \Rightarrow Q$  est vrai et donc  $I \models P \Rightarrow Q$  on en conclut que  $\mathcal{E} \models P \Rightarrow Q$
  - En sens inverse, si  $\mathcal{E} \models P \Rightarrow Q$  et  $I \models \mathcal{E}, P$ . On a  $I \models \mathcal{E}$  donc  $I \models P \Rightarrow Q$  et  $I \models P$ . On en déduit que  $I \models Q$  et donc  $\mathcal{E}, P \models Q$ .

□

La relation de conséquence logique est stable par substitution et remplacement. On étend les notations  $P[x \leftarrow t]$  et  $P[R(x_1, \dots, x_n) \leftarrow Q]$  à un ensemble de formules  $\mathcal{E}$  en appliquant la transformation à chacune des formules de l'ensemble.

**Proposition 2.3.2** Soit  $P$  une formule,  $\mathcal{E}$  un ensemble de formules tel que  $\mathcal{E} \models P$ .

- si  $x$  est une variable d'objet et  $t$  un terme alors  $\mathcal{E}[x \leftarrow t] \models P[x \leftarrow t]$
- si  $R$  est un symbole de prédicat  $n$ -aire, si  $x_1, \dots, x_n$  sont  $n$  variables et  $Q$  est une formule alors  $\mathcal{E}[R(x_1, \dots, x_n) \leftarrow Q] \models P[R(x_1, \dots, x_n) \leftarrow Q]$

**Preuve:** La preuve suit les lignes des preuves de préservation de la validité par substitution en utilisant le fait que pour n'importe quelle formule  $P$ , interprétation  $I$  et environnement  $\rho$ , on a (proposition 2.1.3

$$\text{val}_I(\rho, P[x \leftarrow t]) = \text{val}_I(\rho + \{x \mapsto \text{val}_I(\rho, t)\}, P)$$

et (d'après la preuve de la proposition 2.2.3)

$$\text{val}_I(\rho, P[R(x_1, \dots, x_n) \leftarrow Q]) = \text{val}_{I'}(\rho, P)$$

avec  $I'$  définie comme  $I$  pour tous les symboles sauf  $R$  pour lequel on pose  $R_{I'}(d_1, \dots, d_n)$  est vrai si et seulement si  $\text{val}_I(\rho + \{x_1 \mapsto d_1, \dots, x_n \mapsto d_n\}, Q)$  est vrai. □

On en déduit facilement les mêmes propriétés pour l'équivalence. Soit  $P_1, P_2$  deux formules telles que  $P_1 \equiv P_2$ . On a  $P_1[x \leftarrow t] \equiv P_2[x \leftarrow t]$  et  $P_1[R(x_1, \dots, x_n) \leftarrow Q] \equiv P_2[R(x_1, \dots, x_n) \leftarrow Q]$ .

**Conséquence et opérateurs logiques.** La propriété de conséquence logique se combine avec les connecteurs et les quantificateurs.

**Proposition 2.3.3** Soit  $\mathcal{E}$  un ensemble de formules,  $A_1, A_2, B_1$  et  $B_2$  des formules. On suppose que  $\mathcal{E}, A_1 \models B_1$  et  $\mathcal{E}, A_2 \models B_2$ , on a alors

- $\mathcal{E}, \neg B_1 \models \neg A_1$
- $\mathcal{E}, A_1 \wedge A_2 \models B_1 \wedge B_2$
- $\mathcal{E}, A_1 \vee A_2 \models B_1 \vee B_2$
- $\mathcal{E}, B_1 \Rightarrow A_2 \models A_1 \Rightarrow B_2$
- si de plus la variable  $x$  n'apparaît pas libre dans  $\mathcal{E}$ , alors  $\mathcal{E}, \forall x, A_1 \models \forall x, B_1$  et  $\mathcal{E}, \exists x, A_1 \models \exists x, B_1$

**Preuve:** La preuve se fait très facilement en utilisant la définition de  $\models$ . On fera attention à la négation et l'implication qui "changent le sens" de la relation. Au niveau des quantificateurs, la condition que la variable  $x$  n'est pas libre dans l'ensemble  $\mathcal{E}$  est essentielle, comme le montre l'exemple suivant avec deux prédicats  $P(x)$  et  $Q(x)$ . On a  $P(x), P(x) \Rightarrow Q(x) \models Q(x)$  qui est trivialement vrai. Si maintenant on pouvait appliquer la règle avec  $\mathcal{E} \stackrel{\text{def}}{=} \{P(x)\}$ ,  $A_1 \stackrel{\text{def}}{=} P(x) \Rightarrow Q(x)$  et  $B_1 \stackrel{\text{def}}{=} Q(x)$ , on en déduirait  $P(x), (\forall x, P(x) \Rightarrow Q(x)) \models \forall x, Q(x)$ . Il suffit de prendre un modèle avec deux éléments  $\{0, 1\}$  avec  $P(0)$  et  $Q(0)$  vrai,  $P(1)$  et  $Q(1)$  faux. On a bien (pour  $x \mapsto 0$ ),  $P(x)$  et  $\forall x, P(x) \Rightarrow Q(x)$  qui sont vrais mais  $\forall x, Q(x)$  est faux.  $\square$

### 2.3.2 Equivalences remarquables

Les équivalences ci-dessous sont des propriétés de base de la logique qui pourront être utilisées sans justification.

#### Cas propositionnel

Dans le cas propositionnel, on prouve les équivalences par exemple en montrant que les deux formules ont la même table de vérité.

**Lois algébriques.** L'ensemble des booléens avec les opérations de conjonction et de disjonction forme ce que l'on appelle une Algèbre de Boole.

- La conjonction et la disjonction sont associatifs et commutatifs :

$$P \wedge Q \equiv Q \wedge P \quad P \vee Q \equiv Q \vee P \quad (P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R) \quad (P \vee Q) \vee R \equiv P \vee (Q \vee R)$$

- les opérateurs  $\vee$  et  $\wedge$  sont distributifs l'un par rapport à l'autre

$$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R) \quad P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$$

- $\top$  et  $\perp$  sont des éléments neutres ou absorbants :

$$P \wedge \top \equiv P \quad P \vee \top \equiv \top \quad P \vee \perp \equiv P \quad P \wedge \perp \equiv \perp$$

Certains connecteurs peuvent se définir en fonction d'autres :

$$\neg P \equiv P \Rightarrow \perp \quad P \Rightarrow Q \equiv \neg P \vee Q \quad P \Leftrightarrow Q \equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P) \equiv (P \wedge Q) \vee (\neg P \wedge \neg Q)$$

**Exercice 2.3** Donner des formules équivalentes pour  $\perp \Rightarrow P$ ,  $\top \Rightarrow P$ ,  $P \Rightarrow \top$



## Lois de de Morgan

Les lois de de Morgan établissent le comportement de la négation par rapport aux autres connecteurs :

$$\begin{array}{lll} \neg \perp & \equiv \top & \neg(P \wedge Q) \equiv \neg P \vee \neg Q & \neg \exists x, P(x) \equiv \forall x, \neg P(x) \\ \neg \top & \equiv \perp & \neg(P \vee Q) \equiv \neg P \wedge \neg Q & \neg \forall x, P(x) \equiv \exists x, \neg P(x) \\ \neg \neg P & \equiv P & \neg(P \Rightarrow Q) \equiv P \wedge \neg Q & \end{array}$$

En utilisant les lois de de Morgan et les formules équivalentes pour l'implication, on peut associer à toute formule  $P$ , une formule équivalente  $Q$ , telle que  $Q$  ne contient pas le symbole  $\Rightarrow$  et le symbole de négation n'apparaît que sur les formules atomiques de la forme  $R(t_1, \dots, t_n)$  qui commencent par un symbole de prédicat.

**Définition 2.3.2 (Forme normale de négation)** Une formule est dite en forme normale de négation si elle ne contient que les connecteurs logiques  $\wedge, \vee$ , les quantificateurs  $\forall, \exists$  et que le symbole de négation n'apparaît que devant une formule atomique  $R(t_1, \dots, t_n)$ .

## Equivalence et quantificateurs

**Proposition 2.3.4** On se place dans un langage avec deux symboles de prédicat unaire  $H$  et  $G$  et un symbole de prédicat binaire  $R$ . Les équivalences suivantes sont vérifiées :

- On peut permuter deux quantificateurs de même nature :  
 $\forall x, \forall y, R(x, y) \equiv \forall y, \forall x, R(x, y) \quad \exists x, \exists y, R(x, y) \equiv \exists y, \exists x, R(x, y)$
- Du fait des liens entre quantification universelle et conjonction d'une part, quantification existentielle et disjonction d'autre part, on peut réarranger l'enchaînement de ces connecteurs :  
 $\forall x, (G(x) \wedge H(x)) \equiv (\forall x, G(x)) \wedge (\forall x, H(x)) \quad \exists x, (G(x) \vee H(x)) \equiv (\exists x, G(x)) \vee (\exists x, H(x))$
- On peut éliminer un quantificateur sur la variable  $x$  si  $x$  n'est pas libre dans la formule. Si  $x \notin \text{VI}(A)$  alors  $\forall x, A \equiv A$  et  $\exists x, A \equiv A$ .
- On peut “sortir” d'un quantificateur sur la variable  $x$ , une sous-formule dans laquelle la variable n'est pas libre. Si  $x \notin \text{VI}(A)$  alors :  
 $\forall x, (A \wedge G(x)) \equiv A \wedge \forall x, G(x) \quad \forall x, (A \vee G(x)) \equiv A \vee \forall x, G(x)$   
 $\exists x, (A \wedge G(x)) \equiv A \wedge \exists x, G(x) \quad \exists x, (A \vee G(x)) \equiv A \vee \exists x, G(x)$

**Exercice 2.4** En utilisant les équivalences précédentes, montrer les équivalences suivantes :

- si  $x \notin \text{VI}(A)$  alors  $\forall x, (A \Rightarrow H(x)) \equiv A \Rightarrow \forall x, H(x) \quad \exists x, (A \Rightarrow H(x)) \equiv A \Rightarrow \exists x, H(x)$
- si  $x \notin \text{VI}(A)$  alors  $\forall x, (G(x) \Rightarrow A) \equiv (\exists x, G(x)) \Rightarrow A \quad \exists x, (G(x) \Rightarrow A) \equiv (\forall x, G(x)) \Rightarrow A$

Une application de la dernière équivalence est le principe dit du “buveur” : dans un bar quelconque, il existe une personne telle que si cette personne boit alors tout le monde boit. Il suffit de prendre pour  $G(x)$  la formule  $x$  boit et pour  $A$  la formule  $\forall x, x$  boit

D'autres équivalences de même nature ne sont pas vérifiées comme le montre l'exercice suivant.

**Exercice 2.5** Trouver des interprétations qui justifient les résultats suivants :

- $\forall x, \exists y, R(x, y) \not\equiv \exists y, \forall x, R(x, y)$
- $\forall x, (G(x) \vee H(x)) \not\equiv (\forall x, G(x)) \vee (\forall x, H(x))$
- $\exists x, (G(x) \wedge H(x)) \not\equiv (\exists x, G(x)) \wedge (\exists x, H(x))$

## 2.4 Modèles particuliers

On rappelle qu'une interprétation d'un langage est constituée d'un ensemble non vide, appelé le domaine et qui représente l'univers dans lequel les objets du langage sont interprétés et pour chaque symbole de fonction



ou de prédicat, une fonction ou une relation sur le domaine qui explicite l'interprétation du symbole du langage correspondant. Un **modèle d'une formule close** (ou d'un ensemble de formules closes) est une interprétation du langage qui rend vraie la formule (ou toutes les formules de l'ensemble). On parle de **modèle d'un langage** comme un synonyme d'interprétation du langage.

Certaines propriétés comme la validité ou l'insatisfiabilité s'appuient sur le caractère vrai ou faux d'une formule dans *toutes les interprétations*, ce qui en général représente une infinité de situations. Par contre la satisfiabilité demande juste d'exhiber un seul modèle de la formule, il suffit alors de bien choisir. Nous allons dans ce chapitre étudier quelques classes d'interprétations qui peuvent être utiles.

### 2.4.1 Modèle fini

On s'intéresse à un domaine fini particulier  $\mathcal{D} = \{d_1; \dots; d_n\}$ . On supposera que l'on a dans le langage des termes sans variable  $\{c_1; \dots; c_n\}$  tels que  $\text{val}(c_i) = d_i$ . On peut toujours se ramener à ce cas en introduisant dans le langage de nouvelles constantes  $\{c_1; \dots; c_n\}$  et en étendant l'interprétation des symboles de départ avec pour l'interprétation de  $c_i$  la valeur  $d_i$ .

Dans une telle interprétation, pour toute formule  $P$  et toute variable libre  $x$  dans  $P$ , les deux formules suivantes sont vraies.

$$(\forall x, P) \Leftrightarrow (P[x \leftarrow c_1] \wedge \dots \wedge P[x \leftarrow c_n]) \quad (\exists x, P) \Leftrightarrow (P[x \leftarrow c_1] \vee \dots \vee P[x \leftarrow c_n])$$

Si on applique cette technique à tous les quantificateurs d'une formule close on se retrouve avec une formule qui ne contient que des connecteurs propositionnels et qui ne contient plus de variables. On peut alors remplacer chaque sous-formule atomique  $R(t_1, \dots, t_p)$  par  $R(c_{i_1}, \dots, c_{i_p})$  si  $\text{val}(t_k) = d_{i_k}$ . On peut maintenant introduire une variable propositionnelle pour chaque sous-formule atomique (cela peut faire beaucoup de variables) et utiliser un solveur propositionnel pour calculer les propriétés de la formule.

**Exercice 2.6** On se place dans un langage qui contient  $n$  constantes  $\{c_1; \dots; c_n\}$  et un symbole de prédicat binaire pour l'égalité.

On s'intéresse uniquement aux interprétations dans lesquelles le symbole d'égalité est interprété comme l'égalité sur le domaine. C'est-à-dire que l'on aura  $t = u$  vrai dans une interprétation  $I$  et un environnement  $\rho$  si et seulement si les termes  $t$  et  $u$  ont la même valeur dans le domaine, c'est-à-dire  $\text{val}_I(\rho, t) = \text{val}_I(\rho, u)$ .

1. Que peut-on dire du cardinal du domaine d'une interprétation qui rend vraie la formule  $\forall x, x = c_1 \vee \dots \vee x = c_n$  ?
2. On se donne un symbole de prédicat  $P$  unaire. On rappelle que dans la théorie de l'égalité, un des axiomes exprime la stabilité de ce prédicat par rapport à l'égalité, à savoir  $\forall x y, x = y \Rightarrow (P(x) \Rightarrow P(y))$ . On note  $\mathcal{E}$  les axiomes de la théorie de l'égalité.

Montrer que  $\mathcal{E}, (\forall x, x = c_1 \vee \dots \vee x = c_n), (P(c_1) \wedge \dots \wedge P(c_n)) \models \forall x, P(x)$

en déduire  $\mathcal{E}, (\forall x, x = c_1 \vee \dots \vee x = c_n), (\exists x, P(x)) \models P(c_1) \vee \dots \vee P(c_n)$

**Exercice 2.7** On se donne un symbole de prédicat  $P$  unaire. On veut montrer que la formule  $(\exists x, P(x)) \Rightarrow \forall x, P(x)$  est vraie dans toute interprétation qui a juste un élément.

1. Trouver une formule  $A$  tel que ce problème soit équivalent à montrer que  $A \models (\exists x, P(x)) \Rightarrow \forall x, P(x)$
2. En déduire la propriété souhaitée.
3. Que peut-on dire de la formule si le domaine de l'interprétation a plus de deux éléments ?

### 2.4.2 Modèle de Herbrand

Pour établir qu'une formule est insatisfiable, il faut a priori raisonner sur tous les modèles et en particulier on peut choisir un domaine arbitraire pour l'interprétation. Le logicien Jacques Herbrand (1908-1931) a établi un résultat très important qui permet de restreindre la recherche à un seul domaine, appelé **domaine de Herbrand**, qui est le domaine des termes clos. A ce domaine est associé une interprétation des termes de manière dite *syntactique*.

Dans cette partie, on introduit la notion de **modèle de Herbrand** et on va montrer que pour certaines formules (dites universelles), la satisfiabilité de la formule se ramène à l'existence d'un modèle de Herbrand dans lequel la formule est vraie. On verra dans le chapitre 3 que l'on peut toujours ramener la question de la satisfiabilité d'une formule quelconque à la satisfiabilité d'une formule universelle. Les modèles de Herbrand seront donc un outil pour simplifier en général la recherche d'un modèle d'une formule.

#### Modèle des termes clos


Un langage du premier ordre est défini par une signature formée d'un ensemble  $\mathcal{F}$  de symboles de fonctions (dont les constantes) et d'un ensemble  $\mathcal{R}$  de symboles de prédicats.

On rappelle que les termes clos sont ceux formés sur la signature  $\mathcal{F}$  qui ne contiennent pas de variable, on note  $\mathcal{T}(\mathcal{F})$  l'ensemble des termes clos. S'il n'y a pas de constante dans l'ensemble  $\mathcal{F}$  alors l'ensemble des termes clos est vide. On montre ce résultat par l'absurde en supposant que l'ensemble  $\mathcal{T}(\mathcal{F})$  n'est pas vide et en regardant le terme clos de plus petite taille que l'on note  $t$  : comme il n'y a pas de constante, le terme  $t$  commence par un symbole de fonction qui a donc au moins un argument  $u$ , or le terme  $u$  est lui-même clos et sa taille est strictement plus petite que celle de  $t$ , d'où une contradiction.

Comme un domaine doit être un ensemble non vide, dans toute la suite de cette section, on supposera que la signature  $\mathcal{F}$  contient au moins une constante et on l'ajoutera si nécessaire à la signature des formules que l'on souhaite traiter.

**Définition 2.4.1 (Domaine de Herbrand)** Le **domaine de Herbrand** d'un langage logique de signature  $(\mathcal{F}, \mathcal{R})$  est l'ensemble  $\mathcal{T}(\mathcal{F})$  des termes clos formés à partir des symboles de  $\mathcal{F}$ .

**Exemple 2.5** Si la signature est formée d'une constante  $a$  et d'un symbole de fonction unaire  $f$ , le domaine de Herbrand contient les termes  $a, f(a), f(f(a)), \dots, f^n(a), \dots$

 Le domaine de Herbrand est formé de termes clos qui sont représentés comme des arbres et donc deux termes syntaxiquement différents correspondent à deux éléments différents. Si on regarde les entiers avec la constante  $0$ , la fonction successeur unaire  $S$ , la fonction binaire  $+$ , alors les termes clos  $0+S(0)$ ,  $S(0)+0$  et  $S(0)$  sont tous différents. L'interprétation de l'égalité dans les entiers ne correspondra pas à l'égalité du domaine mais à une relation d'équivalence qui identifiera les trois expressions précédentes.

Une fois que l'on connaît le domaine, il faut interpréter chaque symbole de la signature. Dans le cas des interprétations de Herbrand, c'est particulièrement simple car chaque symbole de fonction est interprété par lui-même.

**Définition 2.4.2 (Interprétation de Herbrand (fonctions))** Une **interprétation de Herbrand** est une interprétation  $H$  construite sur le domaine  $\mathcal{T}(\mathcal{F})$  dans laquelle chaque symbole de fonction  $f$  d'arité  $n$  est interprété par une fonction  $f_H$  définie par :

$$f_H(t_1, \dots, t_n) \stackrel{\text{def}}{=} f(t_1, \dots, t_n) \quad t_i \in \mathcal{T}(\mathcal{F})$$

**Proposition 2.4.1** Dans n'importe quelle interprétation de Herbrand, la valeur d'un terme clos est lui-même.

*Preuve:* La preuve se fait par simple récurrence sur la structure des termes. □

Dans une interprétation, il faut également définir l'interprétation des symboles de prédicat. Il faut donc associer à chaque symbole de prédicat  $R$ , l'ensemble des éléments du domaine qui rendent vraies la formule. Dans une interprétation de Herbrand, les éléments sont des termes clos. Interpréter le symbole de prédicat  $R$  revient donc à définir l'ensemble des  $n$ -uplets de termes clos  $(t_1, \dots, t_n)$  tels que  $R(t_1, \dots, t_n)$  est vrai.

**Définition 2.4.3 (Base de Herbrand)** La *base de Herbrand* est l'ensemble des formules atomiques closes construites sur le langage.

**Exemple 2.6** Dans un langage avec une constante  $a$ , un symbole de fonction  $f$ , un symbole de prédicat unaire  $P$  et un symbole de prédicat binaire  $R$ , la base de Herbrand est composé des formules atomiques  $P(a), P(f(a)), \dots, P(f^n(a)), \dots, R(f^n(a), f^p(a)), \dots$

Il y a en général une infinité de formules atomiques closes. Comme il n'y a qu'un nombre fini de symboles dans une formule, la base de Herbrand associée aux symboles d'une formule sera toujours finie ou dénombrable.

Un modèle de Herbrand fixe l'interprétation des termes mais pas celle des prédicats. Il y a autant de modèles de Herbrand que de manière d'assigner des valeurs de vérité aux formules atomiques closes.

**Définition 2.4.4 (Modèle de Herbrand)** Une *modèle de Herbrand* est défini en assignant des valeurs de vérité à chacune des formules de la base de Herbrand (c'est-à-dire à chaque formule atomique close).

### Théorème de Herbrand

Nous allons montrer maintenant le théorème de Herbrand qui nous permet d'établir la satisfiabilité d'une formule en ne cherchant des modèles que parmi les modèle de Herbrand.

**Proposition 2.4.2** Une formule close sans quantificateur est satisfiable si et seulement si il existe une interprétation de Herbrand qui rend vraie la formule.

**Preuve:** Si la formule est close et sans quantificateur, alors toutes les sous-formules atomiques sont dans la base de Herbrand, la formule se comporte donc comme une formule propositionnelle (en considérant les éléments de la base de herbrand comme des variables propositionnelles). S'il y a une interprétation de Herbrand qui rend vraie la formule, alors la formule est satisfiable. Si maintenant on a un modèle  $I$  de la formule alors on construit une interprétation de Herbrand en prenant comme valeur pour les objets de la base de Herbrand la valeur de la formule atomique (close) dans l'interprétation  $I$ . La valeur de la formule atomique  $R(t_1, \dots, t_n)$  est donc définie comme  $\text{val}_I(R(t_1, \dots, t_n))$ . La valeur de la formule initiale est alors la même dans l'interprétation  $I$  et dans l'interprétation de Herbrand et est donc vraie.  $\square$

On vient de montrer que dans le cas de formules closes sans quantificateur, il suffit de s'intéresser aux interprétations de Herbrand. On peut généraliser ce résultat à des formules un peu plus complexes, à savoir les formules dites universelles.

**Définition 2.4.5 (Formule universelle)** Une formule logique est dite *universelle* si elle est de la forme

$$\forall x_1 \dots x_n, A$$

avec  $A$  une formule sans quantificateur.

**Proposition 2.4.3** Une formule universelle close est satisfiable si et seulement si il existe une interprétation de Herbrand qui rend vraie la formule.

**Preuve:** S'il existe une interprétation de Herbrand qui rend vraie la formule alors celle-ci est satisfiable.

Montrons le sens inverse. On suppose que la formule est vraie dans une interprétation  $I$  quelconque. On cherche à construire un modèle de Herbrand  $H$  qui rend vraie la formule. La valeur de la formule atomique  $R(t_1, \dots, t_n)$  dans la base de Herbrand est donc définie comme  $\text{val}_I(R(t_1, \dots, t_n))$ .

Il suffit de montrer que pour tout terme  $t_1, \dots, t_n \in \mathcal{T}(\mathcal{F})$  la valeur de  $A$  dans l'environnement  $\iota \stackrel{\text{def}}{=} \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$  est vraie dans l'interprétation  $H$ . Or  $\text{val}_H(\iota, A) = \text{val}_H(A[x_1 \leftarrow t_1; \dots, x_n \leftarrow t_n])$  en utilisant la propriété 2.1.3 et le fait que dans un modèle de Herbrand la valeur d'un terme clos est lui-même c'est-à-dire que  $\text{val}_H(t) = t$ . Comme notre modèle de Herbrand a été choisi pour donner la même valeur aux formules atomiques que  $I$  et que  $A[x_1 \leftarrow t_1; \dots, x_n \leftarrow t_n]$  est une formule propositionnelle close, on a  $\text{val}_H(A[x_1 \leftarrow t_1; \dots, x_n \leftarrow t_n]) = \text{val}_I(A[x_1 \leftarrow t_1; \dots, x_n \leftarrow t_n]) = \text{val}_I(\{x_1 \mapsto \text{val}_I(t_1), \dots, x_n \mapsto \text{val}_I(t_n)\}, A) = V$  car  $\forall x_1 \dots x_n, A$  est vraie dans le modèle  $I$ .  $\square$

**Définition 2.4.6 (Instance close d'une formule)** Soit une formule  $A$  dont les variables libres sont  $\{x_1, \dots, x_n\}$ . Une formule de la forme  $A[x_1 \leftarrow t_1, \dots, x_n \leftarrow t_n]$  avec  $t_1, \dots, t_n \in \mathcal{T}(\mathcal{F})$  est appelée *instance close* de  $A$ .

Avant de montrer le théorème de Herbrand lui-même, on énonce un autre résultat important de la logique propositionnelle qui est le théorème dit de *compacité*.

**Proposition 2.4.4 (Théorème de compacité)** Soit  $\mathcal{E}$  un ensemble possiblement infini de formules propositionnelles,  $\mathcal{E}$  est satisfiable si et seulement si tout sous-ensemble fini de formules de  $\mathcal{E}$  est satisfiable.

Dire qu'un ensemble infini de formules  $\mathcal{E}$  est satisfiable, c'est dire qu'il y a une interprétation qui rend vraie toutes les formules de  $\mathcal{E}$ . Dire que tous les sous-ensembles finis de formules est satisfiable, c'est dire que pour chaque sous-ensemble fini  $G \subseteq \mathcal{E}$ , on a une interprétation qui satisfait les formules de  $G$ . La force de ce théorème est de dire que l'on peut "*assembler*" les interprétations locales pour construire une solution globale. La preuve dans le cas d'un nombre dénombrable de variables propositionnelles fait l'objet d'un exercice de TD.

Une conséquence immédiate de ce théorème est que si un ensemble infini de formules propositionnelles est insatisfiable alors il existe un sous-ensemble fini de ces formules qui est insatisfiable.

**Proposition 2.4.5 (Théorème de Herbrand)** Soit  $B \stackrel{\text{def}}{=} \forall x_1 \dots x_n, A$  une formule universelle close du calcul des prédicats avec  $A$  sans quantificateur.

- la formule  $B$  est satisfiable si et seulement si tout sous-ensemble fini  $\{A_1, \dots, A_p\}$  d'instances closes de  $A$  est satisfiable;
- de manière équivalente : la formule  $B$  est insatisfiable si et seulement si il existe un ensemble fini  $\{A_1, \dots, A_p\}$  d'instances closes de  $A$  qui est insatisfiable.

**Preuve:** On montre d'abord que la satisfiabilité de la formule  $\forall x_1 \dots x_n, A$  se ramène à la satisfiabilité d'un ensemble dénombrable de formules closes  $\{A[x_1 \leftarrow t_1, \dots, x_n \leftarrow t_n] \mid t_1, \dots, t_n \in \mathcal{T}(\mathcal{F})\}$ . On utilise les mêmes arguments que précédemment,  $\forall x_1 \dots x_n, A$  est vraie dans une interprétation de Herbrand si pour tous les termes  $t_1, \dots, t_n \in \mathcal{T}(\mathcal{F})$ ,  $A[x_1 \leftarrow t_1; \dots, x_n \leftarrow t_n]$  est vraie dans ce modèle. Et donc en assemblant les éléments précédents, on a que  $\forall x_1 \dots x_n, A$  est satisfiable si et seulement si l'ensemble (fini ou dénombrable) de toutes les formules  $A[x_1 \leftarrow t_1; \dots, x_n \leftarrow t_n]$  pour  $t_1, \dots, t_n \in \mathcal{T}(\mathcal{F})$  est satisfiable.

On utilise ensuite le théorème de compacité pour conclure.  $\square$

On revient au problème de la satisfiabilité d'une formule close dans le calcul des prédicats. On remarque que le domaine de Herbrand qui nous intéresse est celui formé de la signature correspondant uniquement aux symboles qui apparaissent dans la formule  $A$ , auquel on ajoute éventuellement une constante.

**Exemple 2.7** Soit la formule  $\neg((\forall x, P(x)) \Rightarrow \exists x, P(x))$  elle est équivalente à  $(\forall x, P(x)) \wedge \forall x, \neg P(x)$  et donc à  $\forall x y, P(x) \wedge \neg P(y)$ . Le domaine de Herbrand est composé d'une seule constante  $a$  et la base de Herbrand a une seule formule atomique close  $P(a)$ . Il y a donc deux modèles de Herbrand. Celui dans lequel  $P(a)$  est vrai et celui dans lequel  $P(a)$  est faux. Dans les deux modèles, la formule  $P(a) \wedge \neg P(a)$  qui est la seule instance close de  $P(x) \wedge \neg P(y)$  est fautive. On peut en déduire que la formule  $\forall x y, P(x) \wedge \neg P(y)$  est insatisfiable et donc que  $(\forall x, P(x)) \Rightarrow \exists x, P(x)$  est valide.

**Exercice 2.8** Soit la formule  $\forall x y, (P(x) \wedge Q(y) \wedge (\neg P(a) \vee \neg Q(a)))$ .

- Quel est le domaine de Herbrand et la base de Herbrand associées ?
- Donner toutes les interprétations de Herbrand possibles.
- La formule est-elle satisfiable ?

**Exercice 2.9** — La formule  $A$  définie comme  $(\exists x, P(x)) \Rightarrow \forall x, P(x)$  est-elle valide ?

- Construire la formule  $B$  qui est la négation de  $A$ .
- Donner la base de Herbrand associée à la formule  $B$ . Combien y-a-t-il de modèles de Herbrand ?
- Est-ce qu'il y a un modèle de Herbrand qui rend vraie la formule  $B$  ?
- Peut-on en déduire que  $B$  est insatisfiable ?

## Conclusion

Le théorème de Herbrand nous permet de ramener un problème sémantique général (existence d'un modèle) du premier ordre (avec quantificateurs) à un problème propositionnel que l'on va pouvoir traiter avec des outils de nature syntaxique. Même si le calcul propositionnel est décidable, cela ne donne pas une méthode de décision pour le calcul des prédicats. En effet l'ensemble des formules dont on doit prouver le caractère insatisfiable est infini en général. Si l'ensemble est insatisfiable, il y aura un sous-ensemble fini insatisfiable, et on pourra le trouver en énumérant tous les ensembles. Par contre si l'ensemble est satisfiable, alors n'importe quel sous-ensemble fini est satisfiable, et donc on ne trouvera pas de sous-ensemble insatisfiable mais on ne pourra pas conclure par une simple énumération.

Dans le chapitre 3, nous allons étudier des méthodes de transformation de formules qui nous permettront entre autres de nous ramener au cas des formules universelles et dans le chapitre 4, nous étudierons différentes techniques pour prouver la validité ou la satisfiabilité des formules en raisonnant uniquement sur leur forme syntaxique.

## Chapitre 3

# Manipuler les formules de la logique

Dans ce chapitre, nous allons aborder la logique et les formules sur le plan syntaxique, c'est-à-dire que nous chercherons à établir des propriétés logiques des formules (validité, satisfiabilité) par des transformations sans passer par la notion de modèle. Nous étudierons tout d'abord des mises en formes canoniques de formules logiques et en particulier la mise en forme clausale. Nous introduirons également la représentation des formules propositionnelles comme des diagrammes de décision binaire. Finalement nous introduirons des systèmes de déduction pour les formules et en particulier le calcul des séquents.

### 3.1 Formes normales

Dans le chapitre 2 nous avons introduit la forme normale de négation d'une formule. Une formule en forme normale de négation ne contient pas de connecteur  $\Rightarrow$  (ou  $\Leftrightarrow$ ) et les seules négations portent sur des symboles de prédicat. Grace aux lois de de Morgan, il est possible de propager les symboles de négation jusqu'aux symboles de prédicat et donc toute formule est équivalente à une formule en forme normale de négation. De plus cette transformation ne change pas (l'ordre de grandeur de) la taille de la formule.

#### 3.1.1 Forme normale de négation

**Définition 3.1.1 (Littéral)** On appelle littéral une formule qui est soit une formule atomique, soit de la forme  $\neg R(t_1, \dots, t_n)$  avec  $R$  un symbole de prédicat.

Dans la suite, on parlera d'instance de prédicat pour désigner une formule atomique  $R(t_1, \dots, t_n)$  avec  $R$  un symbole de prédicat. Une formule atomique est soit une instance de prédicat soit de la forme  $\top$  ou  $\perp$ . Dans le cas propositionnel, les instances de prédicat sont juste les variables propositionnelles.

**Proposition 3.1.1** Toute formule est équivalente à une formule en forme normale de négation, c'est-à-dire qui ne comporte pas de connecteur  $\neg$  sauf dans un littéral et n'utilise que les connecteurs  $\top, \perp, \vee$  et  $\wedge$  et les quantificateurs  $\forall$  et  $\exists$ .

**Preuve:** Pour définir un algorithme qui calcule la forme normale de négation d'une formule, on construit deux fonctions de manière récursive :  $fnn$  qui étant donnée une formule quelconque  $P$ , construit une formule en forme normale de négation équivalente à  $P$  et  $neg$  qui étant donnée une formule quelconque  $P$ , construit une

formule en forme normale de négation équivalente à  $\neg P$ .

$fnn(p)$	$= p$	si $p$ atomique
$fnn(\neg A)$	$= neg(A)$	
$fnn(A \circ B)$	$= fnn(A) \circ fnn(B)$	$\circ \in \{\vee, \wedge\}$
$fnn(A \Rightarrow B)$	$= neg(A) \vee fnn(B)$	
$fnn(\forall x, A)$	$= \forall x, fnn(A)$	
$fnn(\exists x, A)$	$= \exists x, fnn(A)$	
$neg(\top)$	$= \perp$	
$neg(\perp)$	$= \top$	
$neg(R(t_1, \dots, t_n))$	$= \neg R(t_1, \dots, t_n)$	$R$ symbole de prédicat
$neg(\neg A)$	$= fnn(A)$	
$neg(A \wedge B)$	$= neg(A) \vee neg(B)$	
$neg(A \vee B)$	$= neg(A) \wedge neg(B)$	
$neg(A \Rightarrow B)$	$= fnn(A) \wedge neg(B)$	
$neg(\forall x, A)$	$= \exists x, neg(A)$	
$neg(\exists x, A)$	$= \forall x, neg(A)$	

On montre ensuite par récurrence structurale sur la formule  $P$  que  $fnn(P)$  et  $neg(P)$  sont en forme normale de négation et que de plus  $fnn(P) \equiv P$  et  $neg(P) \equiv \neg P$ .

Dans le membre droit de chaque équation, il y a le même nombre de connecteurs logiques  $\wedge, \vee, \forall, \exists$  que dans le membre gauche. Le symbole d'implication est remplacé par le symbole  $\vee$ . On ajoute un symbole de négation uniquement dans le cas de la fonction  $neg$  appliquée à une instance de prédicat. Dans le cas de la formule  $a_0 \Rightarrow a_1 \Rightarrow \dots \Rightarrow a_n$  qui a  $n$  connecteurs logiques, la forme normale de négation  $\neg a_0 \vee \neg a_1 \dots \vee a_n$  en a  $2n$ . Cela correspond au pire cas. On peut montrer que si  $P$  est une formule avec  $n$  connecteurs, alors  $fnn(P)$  a au plus  $2n$  connecteurs et  $neg(P)$  au plus  $2n + 1$  connecteurs.

D'autre part, les littéraux pourront très souvent se traiter "directement" comme les formules atomiques sans introduire plus de complexité.  $\square$

**Exemple 3.1** Soient  $p$  et  $q$  des variables propositionnelles et soit la formule  $((p \Rightarrow q) \Rightarrow p) \Rightarrow p$  que l'on veut mettre en forme normale de négation. On élimine d'abord les implications, cette formule devient :

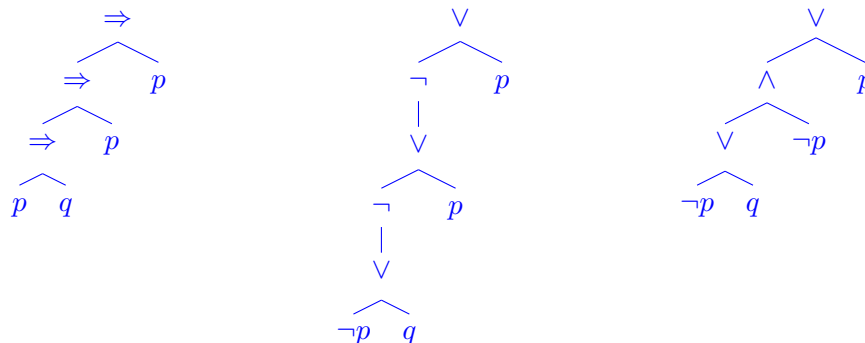
$$\neg(\neg(\neg p \vee q) \vee p) \vee p.$$

On peut ensuite repousser les négations en commençant par l'extérieur. On obtient la formule :

$$((\neg p \vee q) \wedge \neg p) \vee p.$$

Cette formule est en forme normale de négation. On pourra remarquer que sur le plan sémantique, cette formule peut encore se simplifier en  $\neg p \vee p$  en utilisant le fait que  $(a \vee b) \wedge a \equiv a$  puis en  $\top$ .

Ci-dessous les représentations sous forme d'arbre, de la formule initiale, puis après élimination des implications et enfin sous forme normale de négation :





### 3.1.2 Clauses

**Définition 3.1.2 (Clause)** On appelle *clause* une formule qui n'utilise que des littéraux et la disjonction.

⚠ Un littéral tout seul est une clause dans laquelle il n'y a pas de disjonction. En particulier  $\top$  et  $\perp$  sont des clauses. Si on veut faire apparaître une disjonction, on peut toujours réécrire le littéral  $l$  en  $l \vee \perp$ .

**Exemple 3.2** Soit  $p, q$  des variables propositionnelles.

- $\perp, p, \top, \neg p, p \vee q, p \vee q \vee \neg p$  sont des clauses.
- $p \wedge q, p \wedge \top, p \Rightarrow q$  ne sont pas des clauses.

⚠ Être une clause est une propriété *syntaxique* des formules. Une formule peut être équivalente à une clause sans être une clause elle-même.

#### Clause sous forme simplifiée

Les clauses peuvent se simplifier et être représentées par des ensembles de littéraux.

La disjonction est associative et commutative, on peut donc réarranger les littéraux librement à l'intérieur de la formule sans en changer le sens et en restant syntaxiquement une clause.

On a également  $P \vee P \equiv P$ ,  $P \vee \neg P \equiv \top$ ,  $P \vee \top \equiv \top$ ,  $P \vee \perp \equiv P$ .

Une clause peut donc soit être de la forme  $\perp$  ou  $\top$ , soit s'écrire  $l_1 \vee l_2 \vee \dots \vee l_n$  avec  $l_i$  un littéral autre que  $\perp$  ou  $\top$ . On peut représenter une clause comme un ensemble de littéraux tous distincts. La clause qui correspond à la formule atomique  $\perp$  est appelée *clause vide* elle correspond par convention à un ensemble vide de littéraux. Si une clause contient une instance de prédicat et sa négation alors elle est de la forme  $p \vee \neg p \vee Q$  et se simplifie en  $\top$ . On appelle *clause triviale*, une clause qui soit contient le littéral  $\top$ , soit contient une instance de prédicat  $p$  et sa négation  $\neg p$ .

On peut toujours réécrire une clause de manière à ce que chaque instance de prédicat  $R(t_1, \dots, t_n)$  apparaisse au plus une fois sous forme *positive* (littéral  $R(t_1, \dots, t_n)$ ) ou *négative* (littéral  $\neg R(t_1, \dots, t_n)$ ).

**Proposition 3.1.2** Une clause est équivalente à  $\top$  ou  $\perp$  ou bien à une disjonction d'instances de prédicats (sous forme positive ou négative) dans laquelle chaque instance apparaît au plus une fois.

**Preuve:** Si une instance de prédicat  $p$  apparaît deux fois sous forme positive ou deux fois sous forme négative, alors en utilisant  $q \vee q \equiv q$  on peut retirer une des occurrences sans changer la valeur de la formule.

Si une instance de prédicat  $p$  apparaît une fois sous forme positive et une fois sous forme négative alors, en utilisant  $p \vee \neg p \equiv \top$ , on peut remplacer ces deux occurrences par la formule  $\top$ .

Une clause qui contient  $\top$  est simplement équivalente à  $\top$

Si une clause est de la forme  $P \vee \perp$  on peut la simplifier en la clause  $P$ . □

Une clause est donc caractérisée par un ensemble d'instances de prédicats, chacun associé à un *signe* pour indiquer s'il apparait en position positive ou négative. On peut représenter une clause non triviale par un ensemble d'instances de prédicat associés à des booléens (*true* si l'instance apparait positivement et *false* si elle apparait négativement).

Par exemple la clause  $p \vee \neg q$  peut se représenter par l'ensemble  $\{(p, \text{true}); (q, \text{false})\}$ .

**Par convention**, la clause associée à un ensemble vide de littéraux est interprétée comme la formule fausse  $\perp$ . En effet une disjonction de formules est vraie lorsque l'une des formules qui forme la disjonction est vraie. S'il n'y a pas de formule, alors la propriété est fausse.

**Exercice 3.1** Donner les équations récursives d'une fonction *clauseset* qui étant donné un ensemble d'instances de prédicats, chacune associée à un booléen représentant son signe, construit la formule correspondante sous forme de clause.

Si  $L = \{l_1, \dots, l_n\}$  est un ensemble fini de littéraux, on notera  $\tilde{L}$  la formule logique correspondant à la clause  $l_1 \vee \dots \vee l_n$ .

**Proposition 3.1.3** Soient  $L_1$  et  $L_2$  des ensembles de littéraux, si  $L_1 \subseteq L_2$  alors  $\tilde{L}_1 \models \tilde{L}_2$

**Preuve:** Si  $L_1 \subseteq L_2$  alors soit  $L_3 = L_2 \setminus L_1$ . On a  $L_2 = L_1 \cup L_3$  et  $\tilde{L}_2 \equiv \tilde{L}_1 \vee \tilde{L}_3$ . Or  $A \models A \vee B$ , d'où  $\tilde{L}_1 \models \tilde{L}_2$ .  $\square$

La réciproque est vraie à condition que la clause correspondant à  $L_2$  ne soit pas triviale.

**Proposition 3.1.4** Soient  $L_1$  et  $L_2$  des ensembles de littéraux clos, si  $\tilde{L}_1 \models \tilde{L}_2$  et  $L_2$  n'est pas une clause triviale alors  $L_1 \subseteq L_2$

**Preuve:** si  $\tilde{L}_1 \models \tilde{L}_2$  et  $L_1 \not\subseteq L_2$ , alors il existe un littéral  $l \in L_1$  tel que  $l \notin L_2$ .

Comme  $L_2$  porte sur des instances de prédicat clos différentes (sinon elle serait triviale), on peut choisir une interprétation  $J$  telle que  $J \not\models \tilde{L}_2$ . Il suffit de choisir l'interprétation  $J$  pour que tous les littéraux de  $L_2$  soient faux.

L'instance de prédicat du littéral  $l \in L_1$  pourrait apparaître dans  $L_2$  mais avec un signe opposé (équivalent à  $\neg l$ ).

On a que  $J \not\models \neg l$  par construction de  $J$  donc  $J \models l$ .

Dans le cas où l'instance de prédicat sous-jacente à  $l$  n'apparaît pas dans  $L_2$  on peut étendre  $J$  pour que le littéral  $l$  soit vrai.

Comme  $l \in L_1$ , et que  $J \models l$ , on a que  $J \models \tilde{L}_1$  et donc une contradiction avec le fait que  $\tilde{L}_1 \models \tilde{L}_2$  et que  $J \not\models \tilde{L}_2$ .  $\square$

En particulier deux clauses non triviales sont logiquement équivalentes si et seulement si elles correspondent aux mêmes ensembles de littéraux.

Par contre il existe des formules logiques qui ne sont pas équivalentes à des clauses.

En effet si on se donne  $n$  variables propositionnelles différentes, on peut construire  $3^n$  clauses différentes non équivalentes (pour chaque variable, on choisit si elle apparaît ou non dans la clause, et lorsqu'elle apparaît avec quel signe). Si on ajoute la clause triviale, on a donc  $3^n + 1$  clauses qui ne sont pas logiquement équivalentes.

Or le nombre de formules non logiquement équivalentes avec  $n$  variables propositionnelles est  $2^{2^n}$  (nombre de tables de vérité différentes portant sur les  $2^n$  interprétations). Pour  $n = 0$  et  $n = 1$  ces deux nombres coïncident, mais pour  $n = 2$  on a seulement 10 clauses et par contre 16 formules non logiquement équivalentes. Il y a donc certaines "vérités" que l'on ne peut pas représenter par une clause. On va voir dans la suite que par contre chaque formule peut être rendue équivalente à la conjonction d'un ensemble fini de clauses.

### 3.1.3 Formes normales conjonctives et disjonctives

Dans la suite, on ne s'intéresse qu'à la partie propositionnelle du langage c'est-à-dire à des formules qui ne contiennent pas de quantificateur.

#### Définition 3.1.3 (Forme normale conjonctive, forme clause)

Une formule est dite en forme normale conjonctive (abrégé en FNC, CNF en anglais) si elle s'écrit comme une conjonction de clauses.

On peut représenter une formule en forme normale conjonctive par un ensemble de clauses, on parle alors de forme clause.

Par convention, une conjonction d'un ensemble vide de formules est définie comme la formule vraie  $\top$ . En effet, la conjonction de formules est vraie lorsque toutes les formules qui forment la conjonction sont vraies. S'il n'y a pas de formule, alors la propriété est vraie.

### Reconnaître une formule en forme normale conjonctive

On caractérise une formule en forme normale conjonctive en disant que cette formule est une conjonction de littéraux. Mais il faut faire attention qu'il y a des cas de limites :

- Conjonction d'un ensemble vide de formules : toujours vrai  $\top$
- Disjonction d'un ensemble vide de formules : toujours faux  $\perp$
- Conjonction d'un ensemble réduit à une formule  $\{A\}$  : la formule elle-même  $A$
- Disjonction d'un ensemble réduit à une formule  $\{A\}$  : la formule elle-même  $A$

Une formule en forme normale disjonctive peut très bien ne comporter aucun symbole  $\vee$  ni symbole  $\wedge$ . Une disjonction de littéraux est en forme normale conjonctive, de même pour une conjonction de littéraux. Pour qu'une formule soit en forme normale conjonctive, il faut et il suffit qu'elle ne comporte que les connecteurs logiques  $\vee$ ,  $\wedge$  et des littéraux, et de plus que dans la représentation sous forme d'arbre, il n'y ait pas de connecteur  $\wedge$  *en dessous* d'un connecteur  $\vee$ .

#### Exemple 3.3 (Forme normale conjonctive)

- Les formules  $\perp$ ,  $\top$ ,  $p \vee \neg q$ ,  $p \wedge \neg q$ ,  $(p \vee q) \wedge \neg q$  sont en forme normale conjonctive.
- Les formules  $p \Rightarrow q$ ,  $\neg(p \wedge q)$ ,  $(p \wedge q) \vee q$ ,  $(p \wedge (q \vee r)) \vee q$  ne sont pas en forme normale conjonctive.

### Mise en forme normale conjonctive

**Proposition 3.1.5** *Toute formule  $P$  propositionnelle admet une forme normale conjonctive, c'est-à-dire qu'il existe une formule  $Q$  équivalente à  $P$  et qui est en forme normale conjonctive.*

Pour démontrer ce résultat, nous allons utiliser une autre propriété qui dit que toute fonction booléenne  $f$  à  $n$  arguments peut être associée à une formule  $Q$  en forme normale conjonctive, telle que la table de vérité de  $Q$  correspondra à la fonction  $f$ .

On commence par définir plus précisément la notion de fonction booléenne associée à une formule propositionnelle. Cette notion suppose que les variables propositionnelles de la formule sont ordonnées.

**Définition 3.1.4 (Fonction booléenne associée à une formule,  $[P]_{\mathbb{B}}$ )** Soit  $P$  une formule du calcul propositionnel qui contient  $n$  variables propositionnelles nommées  $p_1, \dots, p_n$ . La fonction booléenne associée à  $P$  est une fonction de  $\mathbb{B}^n \rightarrow \mathbb{B}$  notée  $[P]_{\mathbb{B}}$ .

Elle est définie par  $[P]_{\mathbb{B}}(b_1, \dots, b_n) = \text{val}(I_{\{b_1, \dots, b_n\}}, P)$  avec  $I_{\{b_1, \dots, b_n\}}$  l'interprétation dans laquelle chaque variable  $p_i$  a la valeur  $b_i$ .

La fonction  $[P]_{\mathbb{B}}$  correspond à la table de vérité de la formule  $P$ , dans laquelle les variables sont ordonnées suivant les colonnes et chaque ligne correspond à une entrée de la fonction.

**Exemple 3.4** Soit la formule  $P \stackrel{\text{def}}{=} p_1 \vee p_2 \Rightarrow p_3$ . La table de vérité et la fonction associée sont donnés ci-dessous

$p_1$	$p_2$	$p_3$	$P$
$\neg$	$\neg$	$V$	$V$
$V$	$\neg$	$F$	$F$
$F$	$V$	$F$	$F$
$F$	$F$	$F$	$V$

$$\begin{aligned}
 [P]_{\mathbb{B}}(V, V, V) &= V \\
 [P]_{\mathbb{B}}(V, V, F) &= F \\
 [P]_{\mathbb{B}}(V, F, V) &= V \\
 [P]_{\mathbb{B}}(V, F, F) &= F \\
 [P]_{\mathbb{B}}(F, V, V) &= V \\
 [P]_{\mathbb{B}}(F, V, F) &= F \\
 [P]_{\mathbb{B}}(F, F, V) &= V \\
 [P]_{\mathbb{B}}(F, F, F) &= V
 \end{aligned}$$

**Proposition 3.1.6** Pour toute fonction booléenne  $f$  à  $n$  arguments, on peut trouver une formule  $Q$  en forme normale conjonctive telle que  $f = [Q]_{\mathbb{B}}$ .

**Preuve:** On s'intéresse à l'ensemble  $Z$  des  $n$ -uplets pour lesquels la fonction  $f$  est fausse, c'est-à-dire

$$Z \stackrel{\text{def}}{=} \{(b_1, \dots, b_n) \mid f(b_1, \dots, b_n) = F\}$$

Dans la table de vérité, il suffit de s'intéresser aux lignes où la valeur de la formule est  $F$ . L'ensemble  $Z$  est fini. Si cet ensemble est vide alors  $f = [\top]_{\mathbb{B}}$  qui est en forme normale conjonctive. On cherche une formule propositionnelle  $Q$  qui dépend des variables  $p_1, \dots, p_n$  correspondant à  $f$ . La variable propositionnelle  $p_i$  correspond à la propriété "le  $i$ -ème argument de  $f$  est vrai". La propriété  $Q$  doit être vraie lorsque la fonction est vraie et donc doit exprimer que les arguments ne sont pas dans  $Z$ . Pour un élément  $(b_1, \dots, b_n)$ , la formule qui dit que les arguments ne correspondent pas à cette valeur est  $l_1 \vee l_2 \vee \dots \vee l_n$  avec  $l_i = p_i$  si  $b_i = F$  et  $l_i = \neg p_i$  si  $b_i = V$ . Cette formule est une clause. Il suffit ensuite de prendre la conjonction des clauses pour chaque élément de  $Z$ .  $\square$

**Exemple 3.5** Soit la formule  $P \stackrel{\text{def}}{=} (p_1 \Rightarrow p_2) \Rightarrow (\neg p_1 \wedge p_2)$ , on commence par écrire la table de vérité :

$p_1$	$p_2$	$p_1 \Rightarrow p_2$	$\neg p_1 \wedge p_2$	$(p_1 \Rightarrow p_2) \Rightarrow (\neg p_1 \wedge p_2)$
V	V	V	F	F
V	F	F	F	V
F	V	V	V	V
F	F	V	F	F

Les lignes qui nous intéressent sont la première et la dernière et on doit trouver une formule qui dit exactement que l'on n'est pas dans un de ces deux cas.

La formule qui dit que l'on n'est pas dans le cas de la ligne 1 est  $\neg p_1 \vee \neg p_2$  et la formule qui dit que l'on n'est pas dans le cas de la ligne 4 est  $p_1 \vee p_2$ .

La forme normale conjonctive est donc  $(\neg p_1 \vee \neg p_2) \wedge (p_1 \vee p_2)$ .

Si on parle d'une formule  $P$  quelconque, on peut lui associer une fonction booléenne  $[P]_{\mathbb{B}}$ . Cette fonction booléenne peut, d'après le résultat précédent, se représenter par une formule  $Q$  en forme normale conjonctive. On a donc  $[P]_{\mathbb{B}} = [Q]_{\mathbb{B}}$ . Ces deux formules ont même table de vérité par construction et donc sont équivalentes.

### Mise en forme normale conjonctive syntaxique

Construire la table de vérité d'une formule est trop complexe en général et donc on va plutôt utiliser des méthodes syntaxiques. La formule  $(a \wedge b) \vee c$  n'est pas en forme normale conjonctive car il y a une disjonction qui porte sur une formule qui n'est pas un littéral. La règle de distributivité permet d'inverser les conjonctions et les disjonctions.

$$\begin{aligned}
 (a \wedge b) \vee c &\equiv (a \vee c) \wedge (b \vee c) \\
 (a_1 \wedge \dots \wedge a_n) \vee (b_1 \wedge \dots \wedge b_p) &\equiv (a_1 \vee b_1) \wedge \dots \wedge (a_1 \vee b_p) \\
 &\quad \wedge \dots \wedge (a_n \vee b_1) \wedge \dots \wedge (a_n \vee b_p) \\
 &\equiv \bigwedge_{(i=1..n, j=1..p)} (a_i \vee b_j) \\
 (a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3) &\equiv (a_1 \vee a_2 \vee a_3) \wedge (a_1 \vee a_2 \vee b_3) \\
 &\quad \wedge (a_1 \vee b_2 \vee a_3) \wedge (a_1 \vee b_2 \vee b_3) \\
 &\quad \wedge (b_1 \vee a_2 \vee a_3) \wedge (b_1 \vee a_2 \vee b_3) \\
 &\quad \wedge (b_1 \vee b_2 \vee a_3) \wedge (b_1 \vee b_2 \vee b_3) \\
 (a_1 \wedge b_1) \vee \dots \vee (a_n \wedge b_n) &\equiv \bigwedge_{c_i \in \{a_i, b_i\}} c_1 \vee \dots \vee c_n
 \end{aligned}$$

On en déduit un algorithme de mise en forme normale conjonctive. Pour cela, on construit une fonction  $\text{fnc}(P)$  qui calcule un ensemble de clauses équivalent à  $P$  et on utilise de manière auxiliaire une fonction  $\text{fnc-neg}(P)$  qui construit un ensemble de clauses équivalent à  $\neg P$ . On aura besoin d'une autre fonction auxiliaire  $\text{sh}$ , qui prend en argument deux ensembles de clauses  $E$  et  $E'$  et qui construit un nouvel ensemble de clauses avec toutes les combinaisons possibles  $C \vee C'$  avec  $C \in E$  et  $C' \in E'$ , c'est-à-dire que  $\text{sh}(E, E') = \{C \vee C' \mid C \in E, C' \in E'\}$ . On note  $\bigwedge(E)$  la conjonction des clauses de  $E$ . On a par distributivité :

$$\bigwedge(\text{sh}(E, E')) \equiv (\bigwedge(E)) \vee (\bigwedge(E'))$$

. Si  $E$  contient  $n$  clauses et si  $E'$  contient  $p$  clauses alors  $\text{sh}(E, E')$  peut contenir jusqu'à  $n \times p$  clauses (il pourrait y en avoir moins si certaines clauses se simplifient). Pour faire cette opération de manière efficace, il sera utile de représenter les clauses comme des ensembles de littéraux, mais dans la suite on considère les clauses comme des formules ordinaires.

$\text{fnc}(\perp) = \{\perp\}$	$\text{fnc-neg}(\perp) = \emptyset$
$\text{fnc}(\top) = \emptyset$	$\text{fnc-neg}(\top) = \{\perp\}$
$\text{fnc}(x) = \{x\}$	$\text{fnc-neg}(x) = \{\neg x\}$ $x$ instance de prédicat
$\text{fnc}(\neg P) = \text{fnc-neg}(P)$	$\text{fnc-neg}(\neg P) = \text{fnc}(P)$
$\text{fnc}(P \wedge Q) = \text{fnc}(P) \cup \text{fnc}(Q)$	$\text{fnc-neg}(P \vee Q) = \text{fnc-neg}(P) \cup \text{fnc-neg}(Q)$
$\text{fnc}(P \vee Q) = \text{sh}(\text{fnc}(P), \text{fnc}(Q))$	$\text{fnc-neg}(P \wedge Q) = \text{sh}(\text{fnc-neg}(P), \text{fnc-neg}(Q))$
$\text{fnc}(P \Rightarrow Q) = \text{sh}(\text{fnc-neg}(P), \text{fnc}(Q))$	$\text{fnc-neg}(P \Rightarrow Q) = \text{fnc}(P) \cup \text{fnc-neg}(Q)$

On a recours à deux fonctions pour calculer la forme normale conjonctive car on part d'une formule quelconque qui n'est pas déjà en forme normale de négation.

**Exercice 3.2** Donner les équations récursives pour définir une fonction  $\text{fnn2fnc}$  qui étant donnée une formule en forme normale de négation, calcule l'ensemble des clauses correspondant à sa forme clauseale.

### Forme normale disjonctive

On définit de manière duale la notion de forme normale disjonctive.

#### Définition 3.1.5 (Forme normale disjonctive)

- Une conjonction élémentaire est une formule uniquement composée de littéraux et de conjonctions.
- Une formule  $P$  est en forme normale disjonctive (abrégé en FND, DNF en anglais), si elle s'écrit comme une disjonction de conjonctions élémentaires.

Comme dans le cas des clauses, les conjonctions élémentaires peuvent se simplifier de manière à ne garder que celles dans lesquelles une variable propositionnelle apparaît une seule fois, en effet si  $p$  et  $\neg p$  apparaissent dans la même conjonction alors celle-ci est équivalente à  $\perp$ . Comme  $\perp \vee Q \equiv Q$ , on peut simplement supprimer la conjonction élémentaire de l'ensemble des conjonctions.

#### Exemple 3.6 (Forme normale disjonctive)

- Les formules  $\perp, \top, p \vee \neg q, p \wedge \neg q, (p \wedge q) \vee q$  sont en forme normale disjonctive.
- Les formules  $p \Rightarrow q, \neg(p \vee q), (p \vee q) \wedge \neg q, (p \wedge (q \vee r)) \vee q$  ne sont pas en forme normale disjonctive.

**Proposition 3.1.7** Pour toute formule propositionnelle  $P$ , il existe une formule  $Q$  équivalente en forme normale disjonctive.

**Preuve:** On remarque que si  $P$  est en forme normale conjonctive, alors la forme normale de négation de  $\neg P$  est en forme normale disjonctive.

Il suffit donc de prendre la forme normale de négation de la négation de la forme normale conjonctive de la formule  $\neg P$  pour obtenir une formule en forme normale disjonctive équivalent à  $P$ .

$$\text{fnd}(P) = \text{fnn}(\neg \text{fnc}(\neg P))$$

□

**Forme Normale Disjonctive à partir de la table de vérité** Comme précédemment, on peut aussi construire directement la FND à partir de la table de vérité de la formule  $P$ . On suppose que les variables propositionnelles sont  $\{p_1, \dots, p_n\}$ . On s'intéresse à l'ensemble  $O$  des  $n$ -uplets  $(b_1, \dots, b_n)$ , pour lesquels la table de vérité donne la valeur vrai. Pour chaque ligne on construit une conjonction élémentaire  $l_1 \wedge l_2 \wedge \dots \wedge l_n$  avec  $l_i = p_i$  si  $b_i = V$  et  $l_i = \neg p_i$  si  $b_i = F$ . Il suffit ensuite de prendre la disjonction de ces formules pour trouver une formule équivalente à  $P$ .

**Exemple 3.7** En reprenant la formule  $P$  de l'exemple 3.5, on obtient la forme normale disjonctive :

$$(p_1 \wedge \neg p_2) \vee (\neg p_1 \wedge p_2)$$

Si une formule est en forme normale disjonctive alors il est facile d'en trouver un modèle. On suppose que les branches de la disjonction sont en forme "simplifiée" c'est-à-dire qu'une variable propositionnelle apparaît dans un seul littéral. Il suffit de choisir une des branches de la disjonction. Pour chaque variable propositionnelle  $x$  qui apparaît dans un littéral  $l$  on prend dans le modèle  $x \mapsto V$  si  $l = x$  et  $x \mapsto F$  si  $l = \neg x$ . Il peut arriver qu'il n'y ait aucune composante en forme simplifiée : c'est le cas lorsque la formule initiale est équivalente à  $\perp$ . Dans ce cas, il n'y a pas de modèle, la formule est insatisfiable.

**Efficacité** Cependant, la mise en forme normale n'est pas forcément la meilleure manière de trouver un modèle. En effet, cette forme normale peut être exponentiellement plus grosse que la formule initiale. C'est le cas par exemple de la formule  $(a_1 \vee b_1) \wedge \dots \wedge (a_n \vee b_n)$  qui a  $2n$  variables et  $2n - 1$  connecteurs. La forme normale disjonctive s'écrit  $\bigvee_{c_i \in \{a_i, b_i\}} (c_1 \wedge \dots \wedge c_n)$  dans laquelle il y a  $2^n$  disjonctions.

### Forme normale conjonctive efficace pour la satisfiabilité

Lorsqu'une formule propositionnelle est en forme normale conjonctive, il est facile de voir si elle est valide. En effet si l'ensemble des clauses n'est pas vide, on en choisit une qui est de la forme  $l_1 \vee \dots \vee l_n$  que l'on rend fausse en choisissant une interprétation dans laquelle chacun des  $l_i$  est faux ce qui est possible car ils portent tous sur des variables propositionnelles différentes. Si une des clauses de la forme normale conjonctive est fausse alors il en est de même de la formule.

La forme normale conjonctive de la formule  $(a_1 \wedge b_1) \vee \dots \vee (a_n \wedge b_n)$  s'écrit  $\bigwedge_{c_i \in \{a_i, b_i\}} c_1 \vee \dots \vee c_n$  qui contient  $2^n$  clauses. Cette complexité est intrinsèque car liée à la complexité de décider si une formule propositionnelle est valide.

La forme normale conjonctive est équivalente à la formule initiale. Pour la démonstration automatique, cette équivalence n'est pas forcément nécessaire. Si on ne s'intéresse qu'à la satisfiabilité, on peut se permettre de transformer la formule  $P$  en une formule  $Q$ , telle que  $P$  et  $Q$  ne sont pas équivalentes, mais seulement equisatisfiables. C'est-à-dire que  $P$  est satisfiable si et seulement si  $Q$  est satisfiable. Cela nous permet d'introduire de nouvelles variables propositionnelles dans  $Q$  pour éviter l'explosion combinatoire.

Dire que  $P$  et  $Q$  sont équivalents c'est dire que pour toute interprétation  $I$ , on a  $I \models P$  si et seulement si  $I \models Q$  alors que dire que  $P$  et  $Q$  sont equisatisfiables c'est dire qu'il existe une interprétation  $I$  telle que  $I \models P$  si et seulement si il existe une interprétation  $J$  telle que  $J \models Q$ . Deux formules équivalentes sont equisatisfiables mais le contraire n'est pas vrai.

**Proposition 3.1.8 (Forme clause equisatisfiable)** Pour toute formule propositionnelle  $P$ , il existe un ensemble de clauses  $clause(P)$  dont la taille est linéaire par rapport à la taille de  $P$  et qui est equisatisfiable, plus précisément :

- $clause(P) \models P$ .
- pour toute interprétation  $I$ , si  $I \models P$  alors il existe une interprétation  $J$  telle que  $J \models clause(P)$  qui de plus coïncide avec  $I$  sur les symboles de  $P$ .



La construction se fait de manière récursive sur la structure de la formule que l'on supposera initialement en forme normale de négation. La formule ne contient donc que des conjonctions, des disjonctions et des littéraux.

- Si  $P = P_1 \wedge P_2$  alors il suffit de décomposer  $P_1$  et  $P_2$  en clauses et de les rassembler :  $\text{clause}(P) = \text{clause}(P_1) \cup \text{clause}(P_2)$
- Si  $P$  est un littéral ou une disjonction de littéraux alors  $P$  est déjà une clause et  $\text{clause}(P) = \{P\}$
- Si  $P = P_1 \vee P_2$  mais que  $P$  n'est pas une disjonction de littéraux, cela signifie que  $P$  contient une conjonction. En réarrangeant les disjonctions, la formule  $P$  est logiquement équivalente à  $l_1 \vee \dots \vee l_n \vee (R_1 \wedge Q_1) \vee \dots \vee (R_p \wedge Q_p)$  (on met à plat toutes les disjonctions qui portent sur des littéraux  $l_i$  puis celles qui ont une conjonction qu'il va falloir éliminer).

On remplace chaque conjonction  $R_i \wedge Q_i$  par une nouvelle variable propositionnelle  $x_i$  et on ajoute des formules qui disent que  $x_i \Rightarrow (R_i \wedge Q_i)$ .

$$\begin{aligned} \text{clause}(P) = & \{l_1 \vee \dots \vee l_n \vee x_1 \vee \dots \vee x_p\} \\ & \cup \text{clause}(\neg x_1 \vee R_1) \cup \text{clause}(\neg x_1 \vee Q_1) \\ & \cup \dots \\ & \cup \text{clause}(\neg x_p \vee R_p) \cup \text{clause}(\neg x_p \vee Q_p) \end{aligned}$$

**Exemple 3.8** Soit la formule  $P$  définie comme  $(a_1 \wedge b_1) \vee (a_2 \wedge \neg b_2) \vee (\neg a_3 \wedge (b_3 \vee c)) \vee a_4 \vee \neg b_4$ . On introduit les variables  $x_1$ ,  $x_2$  et  $x_3$  pour représenter respectivement  $(a_1 \wedge b_1)$  ( $x_1$ ),  $(a_2 \wedge \neg b_2)$  ( $x_2$ ), et  $(\neg a_3 \wedge (b_3 \vee c))$  pour ( $x_3$ ). L'ensemble  $\text{clause}(P)$  est donc :

$$\{x_1 \vee x_2 \vee x_3 \vee a_4 \vee \neg b_4, \neg x_1 \vee a_1, \neg x_1 \vee b_1, \neg x_2 \vee a_2, \neg x_2 \vee \neg b_2, \neg x_3 \vee a_3, \neg x_3 \vee b_3 \vee c\}$$

Dans l'ensemble de clauses obtenu, on a le même nombre de connecteurs  $\vee$  et  $\wedge$  que dans la formule initiale plus un nombre de variables et de clauses égal au nombre de connecteurs  $\wedge$  qui se trouvent sous un connecteur  $\vee$  dans la formule initiale (ici 3).

**Preuve:** La définition de la fonction  $\text{clause}$  est bien formée même si elle ne suit pas le schéma récursif structurel usuel. Elle traite tous les cas (pour une formule propositionnelle en forme normale de négation) et les appels récursifs se font sur des formules qui contiennent strictement moins de connecteurs  $\wedge$ .

Concernant la taille des clauses obtenues, on observe qu'une variable  $x$  est introduite chaque fois qu'une expression  $P \wedge Q$  apparaît sous une disjonction. On crée alors deux sous problèmes  $\neg x \vee P$  et  $\neg x \vee Q$ , cette opération fait disparaître une conjonction sous une disjonction sans changer la situation des autres conjonctions.

Montrons maintenant les liens logiques entre l'ensemble des clauses ainsi obtenu et la formule initiale. Plus précisément, on se donne une interprétation  $I$  et on fait le lien entre la vérité de  $P$  dans cette interprétation et celle de  $\text{clause}(P)$ . Le cas intéressant est celui dans lequel on introduit une nouvelle variable propositionnelle. Les autres cas se traitent directement ou par hypothèse de récurrence.

Soit  $P \stackrel{\text{def}}{=} A \vee (R \wedge Q)$ , on suppose qu'il n'y a pas de conjonction dans les formules  $A$ ,  $R$  et  $Q$ , on se ramène à ce cas en utilisant les hypothèses de récurrence.

On a  $\text{clause}(P) \stackrel{\text{def}}{=} \{A \vee x, \neg x \vee R, \neg x \vee Q\}$  avec  $x$  une variable propositionnelle qui n'apparaît pas dans  $A \vee (R \wedge Q)$ .

- Si  $I \models \{A \vee x, \neg x \vee R, \neg x \vee Q\}$  alors :
  - si  $I \models x$ , comme  $I \models \neg x \vee R$  on en déduit  $I \models R$  et de même  $I \models Q$ , donc  $I \models A \vee (R \wedge Q)$
  - si  $I \not\models x$ , comme  $I \models A \vee x$  on en déduit  $I \models A$ , donc  $I \models A \vee (R \wedge Q)$
 dans les deux cas on a bien que si  $I \models \text{clause}(P)$  alors  $I \models P$  donc  $\text{clause}(P) \models P$
- Si  $I \models A \vee (R \wedge Q)$  alors il faut trouver une interprétation  $x_J$  de la variable  $x$  pour laquelle on aura  $I, x_J \models \{A \vee x, \neg x \vee R, \neg x \vee Q\}$ . On choisit pour  $x_J$  la valeur de vérité  $\text{val}(I, Q \wedge R)$  de la formule  $Q \wedge R$  dans l'interprétation  $I$ .



- si  $I \models Q \wedge R$  alors l'interprétation de  $x$  est  $V$  et de plus  $I \models Q$  et  $I \models R$ . On a donc  $I, x_J \models \{A \vee x, \neg x \vee R, \neg x \vee Q\}$
- si  $I \not\models Q \wedge R$  alors l'interprétation de  $\neg x$  est  $V$  et de plus  $I \models A$ . On a donc  $I, x_J \models \{A \vee x, \neg x \vee R, \neg x \vee Q\}$

on a donc montré qu'une interprétation de  $P$  pouvait s'étendre en une interprétation de  $\text{clause}(P)$ .

La formule  $P$  est satisfiable exactement lorsque l'ensemble de clauses  $\text{clause}(P)$  l'est.

□

**Exercice 3.3** — En utilisant une méthode analogue montrer que pour toute formule propositionnelle  $P$ , on peut trouver un ensemble de clauses qui ont au plus trois littéraux tel que cet ensemble est satisfiable si et seulement si la formule  $P$  est satisfiable.

### 3.1.4 Skolémisation

Les transformations précédentes traitent de la partie propositionnelle des formules (sans quantificateur), nous allons dans cette section montrer comment on traite la partie avec quantificateurs.

#### Forme prénexe

Dans la section 2.3.2 nous avons établi des propriétés d'équivalence qui permettent de ramener les quantificateurs en tête de formule. Si  $x$  n'est pas libre dans la formule  $A$  alors :

$$\begin{aligned} \forall x, (A \wedge G(x)) &\equiv A \wedge \forall x, G(x) & \forall x, (A \vee G(x)) &\equiv A \vee \forall x, G(x) \\ \exists x, (A \wedge G(x)) &\equiv A \wedge \exists x, G(x) & \exists x, (A \vee G(x)) &\equiv A \vee \exists x, G(x) \end{aligned}$$

⚠ Attention la remontée des quantificateurs au travers d'une négation change le quantificateur (loi de Morgan) de même lorsqu'il est dans la partie gauche d'une implication. La remontée des quantificateurs se fait *après* avoir mis la formule en forme normale de négation.

⚠ La remontée naïve des quantificateurs peut engendrer une *capture* des variables : la condition que  $x$  n'est pas libre dans la formule  $A$  est essentielle.

Si  $x$  est libre dans  $A$  (en plus d'être liée dans  $\exists x, G(x)$ ) alors on renomme la variable liée en choisissant une variable  $z$  qui n'est pas libre dans  $A$  ni dans  $\exists x, G(x)$ , on a alors  $A \vee \exists x, G(x) = A \vee \exists z, G(z) \equiv \exists z, A \vee G(z)$

**Proposition 3.1.9 (Forme prénexe)** Toute formule logique  $P$  est équivalente à une formule de la forme

$$Qx_1, \dots, Qx_n, A$$

avec  $Q$  l'un des quantificateurs  $\forall$  ou  $\exists$  et  $A$  une formule propositionnelle (sans quantificateur). Une formule de la forme  $Qx_1, \dots, Qx_n, A$  est dite en *forme prénexe*.

**Preuve:** On met la formule  $P$  en forme normale de négation, on utilise les équivalences rappelées ci-dessus pour remonter les quantificateurs en renommant si nécessaire les variables liées. □

⚠ La forme prénexe n'est pas unique. L'ordre des quantificateurs en particulier peut varier en fonction de l'ordre dans lequel on choisit de les faire remonter.

#### Élimination des quantificateurs existentiels

On a vu qu'intervertir un quantificateur existentiel et un quantificateur universel en général ne préservait pas l'équivalence entre les formules. Nous allons montrer ici que, si on ne s'intéresse qu'à la satisfiabilité, alors on peut se ramener, grâce à une transformation appelée la skolemisation, à une formule prénexe qui n'aura que des quantificateurs universels.

La skolemisation est une opération (introduite par le logicien norvégien Thoralf Albert Skolem au 20ème siècle) qui permet de se débarrasser des quantifications existentielles en enrichissant les signatures et en préservant la satisfiabilité des formules.

L'idée de base est que si on s'intéresse à la formule  $\forall x, \exists y, P(x, y)$  cette formule est vérifiée si pour tout élément  $d$  du domaine, on peut trouver un élément  $e$  du domaine tel que  $I, \{x \mapsto d, y \mapsto e\} \models P(x, y)$ . En général, la valeur  $e$  retenue pour  $y$  dépend de la valeur  $d$  choisie pour  $x$ . On peut exprimer  $e$  en fonction de  $d$  et donc le problème de trouver un modèle de  $\forall x, \exists y, P(x, y)$  est analogue au problème de trouver un modèle pour  $\forall x, P(x, f(x))$  dans lequel  $f$  est un nouveau symbole de fonction.

De manière plus précise on prouve aisément la propriété de conséquence logique

$$\forall x, P(x, f(x)) \models \forall x, \exists y, P(x, y)$$

si pour tout  $x$ , on a établi que  $f(x)$  était dans la relation  $P$  avec  $x$ , alors a fortiori on sait que pour tout  $x$ , il existe un  $y$  qui est dans la relation  $P$  avec  $x$ .


On montre également que si la formule  $\forall x, \exists y, P(x, y)$  est satisfiable alors on peut, à partir d'un modèle de cette formule, trouver un modèle de la formule  $\forall x, P(x, f(x))$  en choisissant la "bonne" interprétation pour la fonction  $f$ .

**Définition 3.1.6 (Elimination d'un quantificateur existentiel)** Soit une formule  $A$  dans laquelle apparaît une sous-formule  $\exists y, B$ . On suppose qu'il y a  $n$  variables libres dans  $\exists y, B$ , c'est-à-dire  $VI(\exists y, B) = \{x_1, \dots, x_n\}$ . On introduit alors dans la signature un nouveau symbole de fonction  $f$  à  $n$  arguments (si  $n = 0$  alors  $f$  est juste une constante). On introduit la formule  $A'$  qui est la formule  $A$  dans laquelle on a remplacé la sous-formule  $\exists y, B$  par  $B[y \leftarrow f(x_1, \dots, x_n)]$ .

Pour que cette opération soit licite il suffit que les variables  $x_1, \dots, x_n$  ne soient pas liées dans  $B$  ce qui peut toujours être garanti quite à effectuer des renommages de variables liées dans  $B$ .

**Exemple 3.9** On suppose que l'on a un symbole de prédicat binaire  $P$ . On élimine le quantificateur existentiel dans les formules suivantes :

- $\forall x, \exists y, P(x, y)$  : la sous-formule existentielle à traiter est  $\exists y, P(x, y)$ . Elle a une seule variable libre  $x$ . On introduit donc un nouveau symbole de fonction  $f$  avec un seul argument et on élimine le quantificateur existentiel en remplaçant  $y$  par  $f(x)$ . La formule devient  $\forall x, P(x, f(x))$
- $\forall x y, \exists z, P(x, z) \wedge P(z, y)$  : la sous-formule existentielle à traiter est  $\exists z, P(x, z) \wedge P(z, y)$ . Elle a deux variables libres  $x, y$ . On introduit donc un nouveau symbole de fonction  $g$  avec deux arguments et on élimine le quantificateur existentiel en remplaçant  $z$  par  $g(x, y)$ . La formule devient  $\forall x y, P(x, g(x, y)) \wedge P(g(x, y), y)$
- $\exists x, \forall y, P(x, y)$  : la sous-formule existentielle à traiter est la formule elle-même. Elle n'a pas de variable libre. On introduit donc un nouveau symbole de constante  $a$  et on élimine le quantificateur existentiel en remplaçant  $x$  par  $a$ . La formule devient  $\forall y, P(a, y)$

 Dans beaucoup d'ouvrages, l'élimination du quantificateur existentiel se fait après avoir mis la formule en forme prénexe. On choisit alors l'arité du symbole de fonction introduit en fonction du nombre de quantifications universelles préalables. L'idée étant que si on a une alternance  $\forall x, \exists y$  alors le choix de  $y$  peut dépendre de  $x$ .

Dans l'approche présentée ici, on s'intéresse à la sous-formule  $\exists y, A$  et aux dépendances de la condition que doit vérifier  $y$ . Si  $A$  ne dépend pas de  $x$  alors dès qu'on a une valeur pour  $y$  qui vérifie  $A$ , cette valeur fonctionnera pour toute valeur de  $x$ , il n'est donc pas utile de faire dépendre  $y$  de la valeur de  $x$ .

La méthode prénexe est bien sûr correcte mais peut amener à des dépendances artificielles. Ainsi si on part d'une formule  $(\forall x, P(x)) \vee \exists y, \neg P(y)$ , la formule  $\exists y, \neg P(y)$  étant close, le symbole introduit pour remplacer l'existentielle est une constante  $a$  et on obtient la formule  $(\forall x, P(x)) \vee \neg P(a)$ . Le domaine de Herbrand associé est constitué de la seule constante  $a$ . Une des formes prénexe est  $\forall x, \exists y, P(x) \vee \neg P(y)$  et, par la méthode

prénexe, il faut introduire un symbole de fonction unaire  $f$  et la formule devient  $(\forall x, P(x) \vee \neg P(f(x)))$ , le domaine de Herbrand associé est alors infini.

Avoir un domaine de Herbrand plus restreint simplifie la recherche de modèle.

**Proposition 3.1.10** Soit  $A$  une formule en forme normale de négation. Soit  $A'$  obtenue à partir de  $A$  en éliminant un quantificateur existentiel suivant la procédure précédente. On a alors  $A' \models A$  et tout modèle de  $A$  peut être transformé en un modèle de  $A'$  qui coïncide avec celui de  $A$  sauf pour l'interprétation du nouveau symbole  $f$ .

**Preuve:** La première propriété est une conséquence de  $B[y \leftarrow f(x_1, \dots, x_n)] \models \exists y, B$  et du fait que  $A$  est en forme normale de négation. En effet on vérifie aisément les propriétés de stabilité de la conséquence logique (en l'absence de négation) : si  $Q \models R$  alors pour n'importe quelle formule  $P$  on a

$$Q \wedge P \models R \wedge P \quad Q \vee P \models R \vee P \quad \forall x, Q \models \forall x, R \quad \exists x, Q \models \exists x, R$$

Pour la seconde propriété on suppose construit un modèle  $I$  de la formule  $A$ . La formule  $B$  a comme variables libres les variables  $x_1 \dots x_n, y$ . On regarde dans le modèle l'ensemble des  $n+1$ -uplets  $(d_1, \dots, d_n, e)$  tels que  $I, \{x_1 \mapsto d_1, \dots, x_n \mapsto d_n, y \mapsto e\} \models B$ . À partir de cette relation, on peut construire une fonction  $f_J \in \mathcal{D}^n \rightarrow \mathcal{D}$  telle que  $I, \{x_1 \mapsto d_1, \dots, x_n \mapsto d_n, y \mapsto f_J(d_1, \dots, d_n)\} \models B$  s'il existe  $e$  tel que  $I, \{x_1 \mapsto d_1, \dots, x_n \mapsto d_n, y \mapsto e\} \models B$  et  $f_J(d_1, \dots, d_n) = d$  avec  $d$  une valeur arbitraire de  $\mathcal{D}$  si pour tout  $e$ ,  $I, \{x_1 \mapsto d_1, \dots, x_n \mapsto d_n, y \mapsto e\} \not\models B$ . On peut alors construire une interprétation  $J$  sur le même domaine  $\mathcal{D}$  que  $I$ , qui étend l'interprétation  $I$  avec l'interprétation du symbole de fonction  $f$  comme la fonction  $f_J$ .

Cette interprétation rend vraie la formule  $(\exists y, B) \Leftrightarrow B[y \leftarrow f(x_1, \dots, x_n)]$  pour tout environnement  $\{x_1 \mapsto d_1, \dots, x_n \mapsto d_n\}$ . Elle rend donc aussi vraie la formule  $\forall x_1 \dots x_n, ((\exists y, B) \Leftrightarrow B[y \leftarrow f(x_1, \dots, x_n)])$ . Par ailleurs cette interprétation rend vraie la formule  $A$  car  $J$  est une extension de  $I$  qui est un modèle de  $A$ . Comme par ailleurs on peut montrer  $\forall x_1 \dots x_n, ((\exists y, B) \Leftrightarrow B[y \leftarrow f(x_1, \dots, x_n)]) \models A \Leftrightarrow A'$  car  $A'$  est obtenue à partir de  $A$  en remplaçant  $\exists y, B$  par  $B[y \leftarrow f(x_1, \dots, x_n)]$ , on en déduit que  $J$  est un modèle de  $A'$ .  $\square$

On peut itérer l'opération précédente pour éliminer tous les quantificateurs existentiels d'une formule. Chaque élimination de quantificateur introduit un nouveau symbole de fonction.

Si on applique ces transformations à une formule  $A$  en forme normale de négation, on obtient une formule  $B$  sans quantificateur existentiel telle que

- $B \models A$
- Tout modèle de  $A$  peut être étendu sur les symboles de  $B$  qui ne sont pas dans  $A$  en un modèle de  $B$ .

**Proposition 3.1.11** Soit  $A$  une formule en forme normale de négation et  $B$  une formule obtenue par élimination des quantificateurs existentiels de  $A$ .

- Si  $B$  est valide alors  $A$  l'est aussi.
- Si  $B$  est satisfiable alors  $A$  l'est aussi.
- Si  $B$  est insatisfiable alors  $A$  l'est aussi.

**Preuve:** Les deux premières propriétés sont des conséquences directes de  $B \models A$  alors que la dernière vient du fait que, s'il existait un modèle de  $A$ , alors il y en aurait aussi un de  $B$ .  $\square$

**Exemple 3.10** Soit la formule  $(\exists x, P(x)) \wedge (\exists z, \neg P(z))$ . La skolemisation (utilisée deux fois) introduit deux nouvelles constantes  $a$  et  $b$  et donne la formule  $P(a) \wedge \neg P(b)$ . Cette formule est satisfiable, par contre si on avait utilisé deux fois la même constante  $a$  on aboutirait à la formule  $P(a) \wedge \neg P(a)$  qui est insatisfiable.

**Proposition 3.1.12 (Skolemisation, Forme de Skolem)** Soit  $A$  une formule close alors il existe une formule  $B$  en forme normale de négation et sans quantificateur telle que si  $V(B) = \{x_1, \dots, x_n\}$  on a

- $(\forall x_1 \dots x_n, B) \models A$
- si  $A$  a un modèle alors il en est de même de  $\forall x_1 \dots x_n, B$

L'opération de transformation s'appelle la skolémisation. On dit que la formule  $\forall x_1 \dots x_n, B$  est la forme de Skolem de  $A$  ou encore la formule skolémisée associée à  $A$ .

**Preuve:** Il suffit d'éliminer les symboles d'implication et mettre la formule  $A$  en forme normale de négation. On applique ensuite l'élimination des quantificateurs existentiels suivant la procédure vue précédemment. On fait ensuite remonter les quantificateurs universels (en s'assurant au préalable qu'il n'y en n'a pas deux différents qui portent sur une variable de même nom) et en utilisant les équivalences  $A \circ \forall x, B \equiv \forall x, (A \circ B)$  lorsque  $x$  n'est pas libre dans  $B$  et  $\circ \in \{\vee, \wedge\}$ . Une fois que tous les quantificateurs universels sont en tête (c'est-à-dire que la formule est en forme prénexe) il suffit de les éliminer pour obtenir la formule  $B$ .  $\square$

Cette opération peut nous permettre de montrer la validité d'une formule. En effet la validité d'une formule  $A$  est équivalente à l'insatisfiabilité de la formule  $\neg A$  qui se ramène donc à l'insatisfiabilité de la forme skolémisée de  $\neg A$ . Comme la forme skolémisée est une formule universelle, le théorème de Herbrand nous permet de ramener la satisfiabilité à l'existence d'un modèle construit sur le domaine de Herbrand.

**Exemple 3.11** Soit la formule  $A \stackrel{\text{def}}{=} (\forall x, (P(x) \Rightarrow Q(x))) \Rightarrow (\forall x, P(x)) \Rightarrow \forall x, Q(x)$ . Pour montrer que  $A$  est valide, on montre que  $\neg A$  est insatisfiable. Pour cela, on skolemise  $\neg A$  en passant par les étapes suivantes :

1. Forme normale de négation de  $\neg A$  :  $(\forall x, (\neg P(x) \vee Q(x))) \wedge (\forall y, P(y)) \wedge \exists z, \neg Q(z)$
2. Elimination du quantificateur existentiel :  $(\forall x, (\neg P(x) \vee Q(x))) \wedge (\forall y, P(y)) \wedge \neg Q(a)$
3. Mise en forme prénexe :  $\forall xy, ((\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(a))$
4. Domaine de Herbrand :  $\mathcal{D} = \{a\}$
5. Instances closes :  $\{(\neg P(a) \vee Q(a)) \wedge P(a) \wedge \neg Q(a)\}$  insatisfiable

D'après le théorème de Herbrand, la formule  $\forall xy, ((\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(a))$  est insatisfiable et donc il en est de même de la formule  $\neg A$  donc  $A$  est valide.

**Définition 3.1.7 (Fermeture universelle)** Soit  $P$  une formule dont les variables libres sont  $\{x_1, \dots, x_n\}$ . On appelle fermeture universelle de la formule  $P$  et on notera  $\forall(P)$  la formule  $\forall x_1, \dots, x_n, P$ . Cette formule est close (sans variable libre). L'opération de permutation de deux quantificateurs universels ne change pas le sens de la formule donc l'ordre dans lequel les variables apparaissent dans les quantificateurs n'a pas d'importance.

**Skolemisation d'un ensemble de formules.** Si on se donne un ensemble de formules  $A_1, \dots, A_n$  on peut traiter la skolémisation de manière indépendante pour chaque formule (en introduisant évidemment des symboles différents). On obtient pour chaque  $A_i$  une formule  $B_i$  sans quantificateurs telle que  $\forall(B_i) \models A_i$ . On en déduit aisément que :

- $\forall(B_1), \dots, \forall(B_n) \models A_1 \wedge \dots \wedge A_n$
- S'il existe un modèle des formules  $A_1, \dots, A_n$  alors on a aussi un modèle de l'ensemble des formules  $\{\forall(B_1), \dots, \forall(B_n)\}$ .

### Forme clausale d'une formule avec quantificateur

**Définition 3.1.8 (Forme clausale d'une formule avec quantificateur)** La forme clausale d'une formule  $P$  est un ensemble de clauses  $\mathcal{C} = \{C_1, \dots, C_p\}$ . Cet ensemble de clauses s'obtient en construisant la forme de skolem de la formule  $P$  qui s'écrit  $\forall x_1 \dots x_n, A$  avec  $A$  formule propositionnelle. L'ensemble de clauses  $\mathcal{C} = \{C_1, \dots, C_p\}$  est ensuite obtenu en mettant en forme normale conjonctive la formule  $A$ . On a alors

- $C_1, \dots, C_p \models A$
- $\forall(C_1), \dots, \forall(C_p) \models P$
- S'il existe un modèle de  $P$  alors on peut l'étendre en un modèle des formules  $\forall(C_1), \dots, \forall(C_p)$ .

**Exercice 3.4** On part de la formule  $A \stackrel{\text{def}}{=} (\exists x, \forall y, R(x, y)) \Rightarrow (\forall y, \exists x, R(x, y))$ .

- Donner la forme normale de négation de la formule  $\neg A$
- Donner la forme de skolem de la formule  $\neg A$
- Donner la forme clausale de  $\neg A$
- la formule  $\neg A$  est-elle satisfiable ? La formule  $A$  est-elle valide ?

## 3.2 Représentations alternatives des formules

Dans cette section, nous explorons quelques voies alternatives aux connecteurs usuels pour représenter les formules de la logique. Nous détaillerons sous forme principalement d'exercice la représentation à l'aide de diagrammes de décision binaires.

### 3.2.1 Restrictions sur les connecteurs

On a vu au travers des équivalences que de nombreuses formules différentes syntaxiquement pouvaient représenter la même vérité. Une question naturelle qui se pose est alors de savoir si on peut se passer complètement de certains connecteurs logiques et quantificateurs, sans perdre en expressivité.


Les lois de de Morgan nous permettent, à l'aide de la négation, de supprimer les implications, l'un des deux quantificateurs  $\forall$  ou  $\exists$  et l'un des connecteurs  $\vee$  ou  $\wedge$ .

On peut éliminer la négation en utilisant l'équivalence  $\neg A \equiv A \Rightarrow \perp$  et représenter n'importe quelle formule logique en utilisant uniquement les connecteurs logiques  $\perp, \Rightarrow$  et un quantificateur au choix. Par exemple la conjonction de deux formules  $A$  et  $B$  peut se représenter par la formule  $(A \Rightarrow B \Rightarrow \perp) \Rightarrow \perp$  dont on vérifie qu'elle ne peut être vraie que si  $A \Rightarrow B \Rightarrow \perp$  est faux et donc que  $A$  et  $B$  sont vrais.

On peut éliminer  $\perp$  en le remplaçant par  $a \wedge \neg a$  avec  $a$  une variable propositionnelle. De même  $\top$  peut être remplacé par  $a \vee \neg a$  ou  $a \Rightarrow a$ .

Par contre on ne peut pas se débarrasser simultanément de la négation et de l'implication. En effet, si une formule  $P$  n'utilise que les connecteurs  $\vee$  et  $\wedge$ , alors la fonction booléenne correspondant  $[P]_{\mathbb{B}}$  est croissante (en considérant l'ordre  $F < V$ ). Donc on ne peut représenter avec juste  $\vee$  et  $\wedge$  des fonctions booléennes comme la négation qui ne sont pas croissantes.

La mise en forme normale conjonctive montre que l'on peut représenter toutes les formules avec les deux quantificateurs, les connecteurs  $\vee$  et  $\wedge$  (dont on peut même restreindre l'imbrication) et des négations limitées aux littéraux.

 La skolémisation élimine les quantifications existentielles mais la formule obtenue *n'est pas équivalente* à la formule initiale, elle peut être fausse dans des interprétations qui rendent vraies la formule initiale.

### 3.2.2 Connecteurs alternatifs

Il est possible d'introduire d'autres connecteurs propositionnels binaires qui vont nous donner des représentations différentes des formules.

Le plus connu est le ou exclusif noté  $\oplus$  qui est défini pour satisfaire les équivalences

$$x \oplus y \equiv x \vee y \wedge \neg(x \wedge y) \equiv (x \wedge \neg y) \vee (\neg x \wedge y) \equiv \neg(x \Leftrightarrow y)$$

On peut coder toutes les fonctions booléennes à l'aide des seuls connecteurs  $\oplus, \wedge, \top$ . Les opérations  $\oplus$  et  $\wedge$  munissent l'ensemble des booléens d'une structure d'anneau de Boole.

L'opération  $\oplus$  est utile comme opérateur simple de cryptographie bit-à-bit d'un message. On utilise le fait que  $x \oplus x = \perp$  et  $x \oplus \perp = x$ , ainsi que l'associativité de  $\oplus$ . Si on dispose d'une clé secrète **key**, le message  $m$  crypté sera  $m \oplus \text{key}$ , il pourra être décrypté en lui appliquant la même opération  $(m \oplus \text{key}) \oplus \text{key}$ .

Il est même possible de représenter toutes les fonctions booléennes à l'aide uniquement de variables propositionnelles et d'un unique connecteur binaire : la barre de Sheffer  $x \mid y$  également appelée opérateur **NAND**.

et qui est défini comme la négation de la conjonction  $x \mid y \equiv \neg(x \wedge y)$ , on vérifie aisément que  $x \mid x \equiv \neg x$ ,  $x \mid \neg x \equiv \top$ ,  $(x \mid y) \mid (x \mid y) \equiv \neg(x \mid y) \equiv x \wedge y$  et ces constructions sont suffisantes pour reconstruire toutes les formules propositionnelles.

### 3.2.3 Diagrammes de décision binaire

Une représentation alternative de la partie propositionnelle des formules est l'utilisation d'un connecteur de conditionnelle à trois arguments.  $\mathbf{IF}(P, Q, R)$ . C'est d'une part une construction naturelle dans les langages de programmation, mais surtout, en limitant les conditionnelles à porter sur des variables propositionnelles, cette représentation des formules propositionnelles devient très efficace. Elle est largement utilisée en démonstration automatique.

**IF-expressions.** On introduit un connecteur logique  $\mathbf{IF}(P, Q, R)$  qui a la même table de vérité que la formule  $(P \wedge Q) \vee (\neg P \wedge R)$ , ou de manière équivalente à  $(P \Rightarrow Q) \wedge (\neg P \Rightarrow R)$ .

On représente n'importe quelle fonction booléenne en utilisant ce connecteur ainsi que les formules  $\perp$  et  $\top$ . Pour s'en convaincre il suffit de remarquer que :

$$\begin{aligned}\mathbf{IF}(P, \perp, \top) &\equiv \neg P \\ \mathbf{IF}(P, Q, \perp) &\equiv P \wedge Q \\ \mathbf{IF}(P, \top, Q) &\equiv P \vee Q \\ \mathbf{IF}(P, Q, \top) &\equiv P \Rightarrow Q\end{aligned}$$

On remarque ensuite qu'il est possible de simplifier la partie condition, en effet si le premier argument du  $\mathbf{IF}$  n'est pas une variable propositionnelle alors une des équivalences suivantes va s'appliquer :

$$\mathbf{IF}(\top, P, Q) \equiv P \quad \mathbf{IF}(\perp, P, Q) \equiv Q \quad \mathbf{IF}(\mathbf{IF}(B, P, Q), R, S) \equiv \mathbf{IF}(B, \mathbf{IF}(P, R, S), \mathbf{IF}(Q, R, S))$$

On peut donc se ramener à des expressions conditionnelles dans lesquelles la condition est toujours une variable propositionnelle. On remarquera néanmoins que construire cette forme "simplifiée" peut faire grossir la taille de la formule de manière exponentielle.

En effet si on se donne des variables propositionnelles  $p, p_1, \dots, p_k, q_1, \dots, q_k$ , on peut construire une suite de formules  $A_0 = p, A_{n+1} = \mathbf{IF}(A_n, p_n, q_n)$ . La formule  $A_n$  contient  $n$  connecteurs  $\mathbf{IF}$  et chaque variable propositionnelle apparaît exactement une fois.

Si on veut simplifier  $A_{n+1}$ , on obtiendra la suite de transformations suivante :

$$\begin{aligned}A_{n+1} &= \mathbf{IF}(A_n, p_n, q_n) = \mathbf{IF}(\mathbf{IF}(A_{n-1}, p_{n-1}, q_{n-1}), p_n, q_n) \\ &\equiv \mathbf{IF}(A_{n-1}, \mathbf{IF}(p_{n-1}, p_n, q_n), \mathbf{IF}(q_{n-1}, p_n, q_n)) \\ &\equiv \mathbf{IF}(A_{n-2}, \mathbf{IF}(p_{n-2}, \mathbf{IF}(p_{n-1}, p_n, q_n), \mathbf{IF}(q_{n-1}, p_n, q_n))), \mathbf{IF}(p_{n-2}, \mathbf{IF}(p_{n-1}, p_n, q_n), \mathbf{IF}(q_{n-1}, p_n, q_n)))\end{aligned}$$

A chaque étape, on diminue de un le nombre de  $\mathbf{IF}$  dans la condition mais on double le nombre de ceux qui apparaissent dans les branches et on en ajoute deux supplémentaires.

Un des enjeux sera donc de limiter cette explosion par le choix d'une méthode de construction et d'une représentation adéquate. Néanmoins, on gardera en tête que le comportement exponentiel dans le pire des cas est de toute manière inévitable lorsqu'on veut résoudre des problèmes généraux du calcul propositionnel.

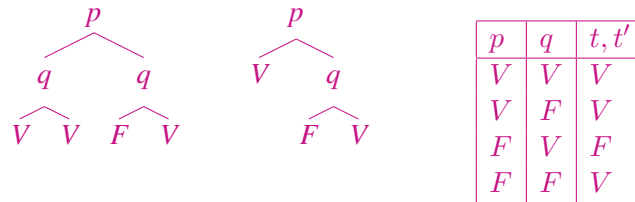
On s'intéresse dans la suite à des formules du calcul propositionnel, construites sur un ensemble de variables propositionnelles totalement ordonné.

**Définition 3.2.1 (Arbre de décision binaire, BDT)** Une *arbre de décision binaire* (BDT) est un arbre binaire dont les nœuds sont étiquetés par des variables propositionnelles et les feuilles sont formées des constantes booléennes  $V$  ou  $F$ . On impose de plus que les variables propositionnelles des fils soient strictement plus grandes que celle du père.



L'arbre de décision binaire définit une fonction booléenne sur l'ensemble des variables présentes dans l'arbre. Lorsque l'on passe un nœud étiqueté par la variable  $x$ , le sous arbre gauche correspond au cas où  $x$  a la valeur  $V$  et le sous-arbre droit au cas où  $x$  a la valeur  $F$ . Une branche de l'arbre définit donc une valeur pour les variables propositionnelles et la valeur de la fonction est déterminée par le booléen présent au niveau de la feuille. Comme toutes les variables ne sont pas forcément présentes dans une branche, celle-ci représente en général une valeur pour un ensemble d'arguments.

**Exemple 3.12** Si on a deux variables propositionnelles  $p < q$ , on peut construire les arbres de décision binaire  $t$  et  $t'$  suivants auxquels sont associés la même fonction booléenne dont on donne le tableau :



On définit simplement de manière récursive la valeur  $\text{vald}(I, t)$  d'un arbre de décision binaire  $t$  dans une interprétation des variables propositionnelles  $I$  :

$$\begin{aligned} \text{vald}(I, b) &= b \text{ si } b \in \{V, F\} \\ \text{vald}(I, \text{IF}(x, P, Q)) &= \text{vald}(I, P) \text{ si } I(x) = V \\ \text{vald}(I, \text{IF}(x, P, Q)) &= \text{vald}(I, Q) \text{ si } I(x) = F \end{aligned}$$

Comme pour les formules ordinaires, pour une interprétation  $I$  et un arbre  $t$ , on notera  $I \models t$  la propriété  $\text{vald}(I, t) = V$ .

### Exercice 3.5 (Arbre de décision binaire.)

1. Donner les équations récursives qui définissent une fonction  $\text{form}$ , qui prend en argument un BDT et renvoie une formule qui a la même table de vérité et qui n'utilise que des littéraux, des conjonctions et des disjonctions.  
Transformer cette fonction pour obtenir directement un ensemble de clauses représentant la forme normale conjonctive.
2. Définir par des équations récursives une fonction  $\text{notd}$  qui prend en argument un BDT  $t$  qui représente une formule  $A$  et renvoie un autre BDT qui représente la formule  $\neg A$ .
3. Définir par des équations récursives une fonction  $\text{conj d}$  qui prend en argument deux BDT  $t$  et  $u$  qui représentent respectivement les formules  $A$  et  $B$  et renvoie un autre BDT qui représente la formule  $A \wedge B$ . On fera attention à respecter la contrainte sur l'ordre des variables propositionnelles dans les BDT.
4. Que peut-on dire des feuilles d'un BDT qui correspond à une formule valide ? satisfiable ? insatisfiable ?
5. Proposer un algorithme pour trouver un modèle d'une formule.

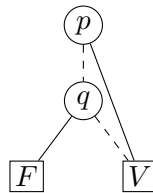
Les arbres de décision binaire ont deux défauts : des arbres différents peuvent représenter des formules équivalentes (par exemple  $\text{IF}(x, P, P) \equiv P$ ) et leur taille est en général exponentielle en le nombre de variables propositionnelles.

**Définition 3.2.2 (Diagramme de décision binaire, BDD)** Un diagramme de décision binaire (BDD) est obtenu à partir d'un arbre de décision en assurant de plus les conditions suivantes :



1. *réduction* : aucun nœud n'a deux fils identiques, cela correspond au fait que l'expression  $\mathbf{IF}(p, A, A)$  est logiquement équivalente à  $A$  et donc on peut remplacer un nœud qui a deux fois le même fils par ce fils ;
2. *partage* : l'arbre est transformé en graphe orienté acyclique (DAG), c'est-à-dire que deux sous-arbres identiques sont partagés. Dans un graphe, il n'y a plus de notion de sous-arbre gauche et de sous-arbre droit : on étiquette donc les arcs par une valeur de vérité  $V$  ou  $F$ . Graphiquement on distingue par un arc plein le cas où la variable vaut  $V$  et par un arc pointillé le cas où la variable vaut  $F$ .
3. *variables ordonnées* : comme dans les BDT, on va demander que les variables soient ordonnées : la variable d'un fils est toujours strictement supérieure à la variable du père, on parle alors de diagramme de décision binaire ordonné (ou OBDD).

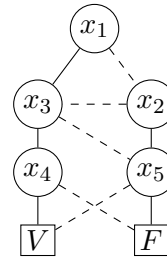
**Exemple 3.13** L'arbre donné dans l'exemple 3.12 correspond au diagramme suivant :



Pour construire un OBDD à partir d'une formule propositionnelle  $P$ , il suffit de partir de la plus petite variable  $x$  qui apparaît dans la formule et de construire récursivement l'OBDD  $P_V$  correspondant à la formule  $P[x \leftarrow \top]$  pour la branche "vrai" et l'OBDD  $P_F$  correspondant à la formule  $P[x \leftarrow \perp]$  pour la branche "faux". Si les deux OBDDs sont les mêmes alors  $P_V$  est le résultat attendu, sinon on introduit le nœud  $x$  avec un arc plein vers  $P_V$  et un arc pointillé vers  $P_F$ .

- Exercice 3.6**
1. Construire les OBDDs des formules  $(z \Leftrightarrow t)$  puis  $(x \Leftrightarrow y) \wedge (z \Leftrightarrow t)$  en prenant comme ordre  $x < y < z < t$ .
  2. En partant des feuilles, annoter chaque nœud dans les deux OBDD précédents par une formule équivalente qui n'utilise que des littéraux et les symboles  $\vee$  et  $\wedge$ .
  3. Montrer que deux formules sont équivalentes si et seulement si elles sont représentées par le même OBDD. En déduire comment, étant donné un OBDD, on peut savoir si la formule associée est valide, satisfiable ou insatisfiable.
  4. Pour représenter un OBDD, on associe un numéro unique à chaque sommet du graphe et on utilise une table  $G$  qui permet de stocker pour chaque sommet la variable associée et les arêtes sortantes. On dispose des opérations suivantes :
    - $\text{noeud}(G, n)$  : teste si  $n$  est un sommet du graphe qui correspond à un nœud interne ;
    - $\text{feuille}(G, n)$  : teste si  $n$  est un sommet du graphe qui correspond à une feuille ( $V$  ou  $F$ ) ;
    - pour un nœud interne :  $\text{var}(G, n)$  donne la variable propositionnelle associée,  $\text{vrai}(G, n)$  est le numéro du sommet qui correspond au cas où la variable est vraie et  $\text{faux}(G, n)$  est le numéro du sommet qui correspond au cas où la variable est fausse ;
    - pour une feuille :  $\text{valeur}(G, n)$  renvoie un booléen correspondant à la valeur  $V$  ou  $F$  de la feuille.
- (a) À l'aide des fonctions ci-dessus, exprimer les propriétés sur la table  $G$  qui assurent que le graphe correspond bien à un OBDD : partage maximal, réduction et ordre sur les variables propositionnelles.
- (b) Soit un graphe  $G$  correspondant à un OBDD et un sommet  $n$ , définir une fonction  $\text{form}(G, n)$  qui calcule la formule logique associée au sommet  $n$ , expliquer comment exploiter le partage pour ne pas recalculer deux fois le même résultat.

- (c) Expliquer comment construire une fonction *if* qui étant donné un graphe  $G$  correspondant à un OBDD, une variable propositionnelle  $x$  et deux sommets  $n$  et  $m$  de  $G$ , renvoie un graphe  $G'$  et un nœud  $p$ , tel que  $\text{form}(G', p) \equiv \mathbf{IF}(x, \text{form}(G, n), \text{form}(G, m))$  et  $\text{form}(G', k) = \text{form}(G, k)$  pour tous les sommets de  $G$  ( $G'$  contient possiblement un sommet de plus que  $G$ , mais ne modifie pas les sommets de  $G$ ). On utilisera une table  $T$  qui permet de retrouver si une formule est déjà représentée dans le graphe : il suffit de donner le nom de la variable et les sommets correspondant au cas vrai et au cas faux, si le nœud apparaît dans le graphe, la table donnera le sommet correspondant.
- (d) On veut compter le nombre de modèles d'une formule représentée par un OBDD.
- i. Soit l'OBDD suivant sur les variables propositionnelles  $x_1, x_2, x_3, x_4, x_5$  :



annoter chaque nœud  $n$  du graphe pour donner le nombre d'interprétations concernant les variables plus grandes que celle du nœud  $n$  qui satisfont la formule associée à  $n$ .

- ii. On suppose donnée une fonction *indice* qui calcule pour chaque sommet de l'OBDD un entier : si le sommet est un nœud, l'indice est la position de la variable correspondante dans l'ordre (la plus petite variable a le numéro 1) et si le sommet est une feuille, l'indice est  $n + 1$  avec  $n$  le nombre total de variables.  
 Construire une fonction *nbsat* qui étant donné un OBDD représenté par un graphe  $G$  et un sommet  $n$ , retourne le nombre de modèles de la formule en ne considérant que les variables plus grandes que celle du sommet.
- iii. Sachant que la variable racine d'un OBDD n'est pas forcément la plus petite variable, en déduire le nombre de modèles de la formule en considérant toutes les variables.

### Compétences 3.1

- Savoir reconnaître les formes syntaxiques des formules : clauses, forme normale de négation, forme normale conjonctive et disjonctive, forme prénexe, forme clausale.
- Savoir skolémiser une formule.
- Comprendre et savoir mettre en œuvre les algorithmes pour transformer les formules (principalement forme normale de négation, et forme clausale)
- Être capable de manipuler des connecteurs logiques alternatifs.

### 3.3 Systèmes de déduction

On s'intéresse maintenant à la possibilité de déduire de manière syntactique qu'une formule  $P$  est une conséquence logique d'un ensemble de formules  $\mathcal{E}$ , c'est-à-dire que la formule  $P$  sera vraie dans toute interprétation qui rend les formules de  $\mathcal{E}$  vraies.

Nous allons introduire des systèmes de déduction qui définissent des relations sur les formules en se donnant un certain nombre de règles du jeu, liées à leur structure logique.

Les relations peuvent porter sur les formules elle-mêmes ou bien sur une structure un peu plus compliquée appelée séquent qui prend en compte à la fois un ensemble d'hypothèses et une ou même un ensemble de conclusions.

Avant d'introduire les systèmes spécifiques à la logique, nous donnons le cadre général des définitions par règles d'inférence.

#### 3.3.1 Préliminaires

##### Système et règles d'inférence

Les règles d'inférence sont une manière de définir une relation mathématique  $R$  qui est très utile en informatique théorique. Dans tout ce qui suit, on s'intéresse à des propriétés mathématiques générales et non pas à des formules de la logique (même si les notations peuvent se ressembler). Une relation  $n$ -aire  $R$  est juste un sous-ensemble du produit de  $n$  ensembles  $A_1 \times \dots \times A_n$ .

Dans la suite on considère une relation sur un ensemble  $A$  qui pourra être choisi si nécessaire comme un ensemble produit. La relation est donc simplement un sous-ensemble  $R$  de  $A$ .

On écrira de manière générique  $R(t)$  la propriété qui dit que  $t \in R$ .

On se donne donc un ensemble  $A$ , on veut définir une relation  $R$  sous-ensemble de  $A$ . L'idée est de caractériser la relation  $R$  par des conditions que cette relation  $R$  doit vérifier.

Ces conditions sont traditionnellement représentées par des règles d'inférence. Une règle d'inférence se présente sous la forme d'une fraction :

$$\frac{P_1 \dots P_k}{R(t)}$$

Le dénominateur (aussi appelé la conclusion de la règle) est forcément un cas particulier de la relation que l'on cherche à définir. Le numérateur contient un nombre quelconque de conditions (que l'on appellera aussi les prémisses de la règle d'inférence).

La règle peut utiliser des variables mathématiques  $x_1 \dots x_p$ , appelés paramètres.

L'interprétation de la règle est que pour n'importe quelles valeurs possibles de  $x_1 \dots x_p$ , si les conditions  $P_1 \dots P_k$  sont vérifiées alors la propriété  $R(t)$  est également vérifiée.

S'il n'y a aucune condition  $k = 0$ , cela veut dire que  $R(t)$  est toujours vrai.

Par exemple on peut définir le fait qu'un nombre entier est pair avec les règles d'inférences suivantes :

$$(p_0) \frac{}{\text{pair}(0)} \quad (p_s) \frac{\text{pair}(x)}{\text{pair}(x+2)}$$

Dans la deuxième règle,  $x$  est un paramètre représentant n'importe quel entier, c'est-à-dire que la règle est valable si on remplace  $x$  par n'importe quel terme. Par exemple  $\frac{\text{pair}(2)}{\text{pair}(4)}$  mais aussi  $\frac{\text{pair}(1)}{\text{pair}(3)}$ .

Il y a plusieurs relations qui vérifient ces propriétés (par exemple l'ensemble des entiers  $\mathbb{N}$ ). La relation définie par les règles d'inférence est par construction l'intersection de toutes les relations qui vérifient les conditions. Ce sera la la plus petite relation (au sens de l'inclusion) ayant cette propriété.

Le fait qu'une telle relation existe et a les bonnes propriétés est un théorème qui nécessite des conditions supplémentaires sur  $P_i$ . Nous nous intéresserons au cas simple dans lequel chaque  $P_i$  est soit un cas particulier  $R(u)$  de la relation à définir, soit une condition auxiliaire qui ne parle pas de  $R$ .

**Définition 3.3.1 (Relation définie par des règles d'inférences)** Soit un ensemble  $A$ . Une relation  $R$  sur  $A$  peut se définir par un système d'inférence qui est un ensemble fini de règles d'inférence.

Chaque règle d'inférence est de la forme  $\frac{R(u_1) \dots R(u_n) P_1 \dots P_k}{R(t)}$  où  $t, u_1, \dots, u_n \in A$  et  $P_1, \dots, P_k$

sont des conditions auxiliaires qui ne mentionnent pas  $R$ .

La règle d'inférence peut être paramétrée par des variables mathématiques  $x_1 \dots x_p$ .

Chaque règle d'inférence représente une condition sur la relation  $R$  qui s'énonce ainsi : pour n'importe quelles valeurs possibles de  $x_1 \dots x_p$ , si les conditions  $R(u_1) \dots R(u_n)$  et  $P_1 \dots P_k$  sont vérifiées alors la propriété  $R(t)$  est également vérifiée.

La relation  $R$  ainsi définie est la plus petite relation qui vérifie l'ensemble des conditions associées aux règles d'inférence.

**Preuve:** Cette définition nécessite une preuve à savoir l'existence d'une plus petite relation qui vérifie les conditions associées aux règles d'inférence.

On note  $\text{COND}(R)$  l'ensemble des conditions associées aux règles d'inférence de  $R$  et on note  $R_0$  l'intersection  $\bigcap \{R \subseteq A \mid \text{COND}(R)\}$  de toutes les relations qui vérifient la condition.

Cette intersection est bien définie car il y a au moins une relation qui vérifie la condition à savoir la relation qui correspond à l'ensemble  $A$  tout entier.

Il faut montrer que  $R_0$  vérifie les conditions.

Une première remarque est que par définition de l'intersection, on a que  $R_0(t)$  est vérifié si et seulement si  $R(t)$  est vérifié pour toutes les relations  $R$  qui vérifient  $\text{COND}(R)$ .

Soit une règle d'inférence  $\frac{P_1 \dots P_k}{R(t)}$  paramétrée par des variables  $x_1 \dots x_p$ .

Soit donc  $x_1 \dots x_p$  quelconques, on suppose que les prémisses  $P_1 \dots P_k$  sont vérifiées pour la relation  $R_0$  et on doit montrer que  $R_0(t)$  est vérifié.

Par définition de l'intersection on doit montrer que  $R(t)$  est vrai pour n'importe quelle relation qui satisfait les conditions  $\text{COND}(R)$ .

Parmi les conditions que satisfait  $R$  on a la règle d'inférence  $\frac{P_1 \dots P_k}{R(t)}$ , donc pour montrer  $R(t)$ , il suffit de montrer que les prémisses  $P_1 \dots P_k$  sont vérifiées pour la relation  $R$ .

Les prémisses soit ne mentionnent pas  $R$  soit sont de la forme  $R(u)$ .

- si la prémisses ne mentionne pas  $R$ , comme elle est vraie pour  $R_0$ , elle est aussi vraie pour  $R$ ;
- si la prémisses est de la forme  $R(u)$ , alors comme la prémisses est vraie pour  $R_0$  on a  $R_0(u)$  est vérifié et comme  $R$  vérifie toutes les conditions, on a  $R_0 \subseteq R$  et donc  $R(u)$  est aussi vrai.

On en déduit que  $R$  vérifie les prémisses de la règle donc aussi la conclusion et donc comme  $R(t)$  est vrai pour toute relation qui vérifie les conditions, on a aussi  $R_0(t)$ . Ce qui conclut la démonstration du fait que  $R_0$  vérifie les conditions des règles d'inférence.  $\square$

**Remarques :** Ce résultat se généralise au cas où les prémisses sont des propriétés croissantes de la relation à définir (c'est-à-dire que si on a une condition  $P[R]$  qui dépend de la relation  $R$  alors si  $R \subseteq S$  et  $P[R]$  est vérifié, il en est de même de  $P[S]$ ). Une condition pourrait donc être de la forme  $\forall y, R(u) \vee B$ , voir faire apparaître la relation  $R$  à gauche d'un nombre pair de symboles d'implication.

Par contre la croissance des prémisses est essentielle. Si on essaie de définir de la même manière une relation être pair avec les deux règles suivantes :

$$\frac{}{\text{pair}(0)} \quad \frac{\neg \text{pair}(x)}{\text{pair}(x+1)}$$

La relation “tout le temps vraie” ( $\text{pair} = \mathbb{N}$ ) vérifie ces conditions, de même pour la relation “être pair” qui contient tous les entiers pairs ( $\text{pair} = \{2k | k \in \mathbb{N}\}$ ) et qui vérifie les conditions. Mais de manière plus surprenante, la relation “être 0 ou impair” qui contient 0 ainsi que tous les entiers impairs ( $\text{pair} = \{0\} \cup \{2k+1 | k \in \mathbb{N}\}$ ) vérifie aussi les conditions. Donc si on prend l’intersection de toutes les relations qui vérifient les conditions, il reste juste la relation qui contient 0. Mais cette relation ne vérifie pas les conditions puisque elle ne contient pas 1 et donc elle devrait contenir  $1+1=2$  ce qui n’est pas le cas.

Dans nos exemples les seules prémisses qui mentionnent la relation à définir  $R$  sont de la forme  $R(u)$  et donc ne présentent pas ce problème.

**Point fixe de Tarski.** Le résultat précédent est un cas particulier d’un théorème plus général, appelé théorème de point fixe de Tarski (attribué à Bronisław Knaster et Alfred Tarski), qui établit l’existence d’un point fixe pour toute fonction monotone sur un espace ayant de bonnes propriétés.

**Proposition 3.3.1 (Théorème de point fixe de Tarski)** *On se donne  $(E, \leq)$  un ensemble muni d’une structure de treillis complet, c’est-à-dire que  $\leq$  est un ordre partiel et que toute partie  $A \subseteq E$  admet une borne supérieure notée  $\text{sup}(A)$  dans  $E$ .*

*Toute fonction croissante de  $E$  dans  $E$  admet un plus petit et un plus grand point fixe.*

**Preuve:** Par définition la borne supérieure est le plus petit des majorants c’est donc un majorant (pour tout  $x \in A$ , on a  $x \leq \text{sup}(A)$ ) et c’est le plus petit donc si on a  $e \in E$  qui est un majorant (pour tout  $x \in A$ , on a  $x \leq e$ ) alors  $\text{sup}(A) \leq e$ .

L’existence d’une borne supérieure implique l’existence d’une *borne inférieure* pour tout ensemble (la borne inférieure de l’ensemble  $A$  est la borne supérieure de l’ensemble des minorants de  $A$ ), on la notera  $\text{inf}(A)$ .

On regarde l’ensemble  $A = \{x \in E | f(x) \leq x\}$  et on pose  $a = \text{inf}(A)$ .

Les propriétés de la borne inférieure sont

1. si  $f(x) \leq x$  alors  $a \leq x$  (minorant)
2. si  $y$  est un minorant ( $y \leq x$  pour tout  $x$  tel que  $f(x) \leq x$ ), alors  $y \leq a$  (plus grand des minorants)

La preuve suit alors les étapes suivantes :

- Montrons  $f(a) \leq a$  en appliquant la deuxième propriété ; il suffit de montrer que  $f(a)$  est un minorant de  $A$  et donc que  $f(a) \leq x$  pour tout  $x$  tel que  $f(x) \leq x$ . Or si  $f(x) \leq x$ , alors la première propriété nous dit que  $a \leq x$ . Comme  $f$  est croissante, on a  $f(a) \leq f(x)$  et par transitivité  $f(a) \leq x$ .
- De  $f(a) \leq a$ , en appliquant la croissance de  $f$  on déduit  $f(f(a)) \leq f(a)$  et donc  $f(a) \in A$  et par la première propriété, on a donc  $a \leq f(a)$
- On en déduit que  $f(a) = a$ ,  $a$  est donc un point fixe.
- Comme tous les points fixes de  $f$  sont dans l’ensemble  $A$ , on a que  $a$  est le plus petit.
- De manière analogue, le plus grand point fixe sera défini comme  $\text{sup}(\{x \in E | x \leq f(x)\})$ .

□

### Arbre de dérivation

Lorsque l’on introduit un système d’inférence pour définir une relation, on utilise ces mêmes règles d’inférence pour *justifier* que la relation est vérifiée. De manière générale, une preuve de  $R(a)$  se construit par étapes comme une séquence de faits (dérivation) de la forme  $R(b_1), \dots, R(b_n)$  où chaque  $R(b_i)$  s’obtient par une

règle dont les prémisses sont dans  $R(b_1), \dots, R(b_{i-1})$ . En pratique on représente la dérivation comme un arbre dont la racine est la conclusion et chaque nœud correspond à un cas particulier d'une règle d'inférence.

$$\begin{array}{c} p_0 \overline{\text{pair}(0)} \\ p_s \overline{\text{pair}(2)} \\ p_s \overline{\text{pair}(4)} \end{array}$$

**Définition 3.3.2 (Arbre de dérivation)** Etant donné un système d'inférence pour une relation  $R \subseteq A$ , on construit un arbre de dérivation pour ce système. Les nœuds de cet arbre sont étiquetés par des instances de la relation de la forme  $R(a)$ .

Chaque nœud correspond à un cas particulier d'une règle d'inférence  $\frac{R(u_1) \dots R(u_n) P_1 \dots P_k}{R(t)}$ .

C'est-à-dire qu'il existe des valeurs pour les paramètres  $x_1, \dots, x_p$  de la règle pour lesquelles  $t = a$ .

On écrit  $\frac{R(b_1) \dots R(b_n) P'_1 \dots P'_k}{R(a)}$  la règle obtenue pour ce cas particulier des paramètres.

Pour que la règle soit applicable, il faut que l'ensemble des conditions auxiliaires  $P'_1 \dots P'_k$  soient vérifiées.

Un nœud correspondant à cette règle aura  $n$  fils (un fils par prémisses de la forme  $R(b_i)$ ), chaque fils est étiqueté par cette prémisses.

L'arbre est **complet** s'il n'y a plus de feuilles (application de règles sans prémisses portant sur  $R$ , c'est-à-dire  $n = 0$ ) et il constitue alors une preuve que la racine est dans la relation.

Un arbre **incomplet** constitue une preuve du fait que si l'ensemble des propriétés correspondant aux feuilles de l'arbre sont vérifiées alors la propriété à la racine de l'arbre est aussi vérifiée.

L'arbre se dessine en juxtaposant les règles d'inférence instanciées, il est recommandé de nommer chaque règle et de reporter le nom en marge de la barre de fraction. De même les conditions qui doivent être vérifiées peuvent être rappelées en annotation du nœud. On peut aussi noter les valeurs prises par les paramètres de la règle de manière à les reporter uniformément dans les prémisses et la conclusion.

**Exercice 3.7 (Déplacement d'un robot)** On veut modéliser le déplacement d'un robot dans le plan. Les coordonnées sont données par des couples d'entiers naturels. Le robot part de la position  $(0, 0)$ . Les seuls déplacements possibles sont d'une case  $(x, y)$  à la case  $(x + 1, y + 2)$  ou bien à la case  $(x + 2, y + 1)$

1. Définir par un système d'inférence la relation  $\text{pos}$  sur  $\mathbb{N} \times \mathbb{N}$  correspondant à toutes les positions accessibles au robot.
2. Construire un arbre de dérivation pour montrer  $\text{pos}(3, 3)$ .
3. Justifier que  $\text{pos}(1, 1)$  implique  $\text{pos}(3, 5)$ .

### Preuve par minimalité de la relation

Le fait que la relation définie est la **plus petite** qui satisfait les règles d'inférence, nous permet également de dériver un schéma de preuve puissant qui dit que si on se donne une autre relation  $S$  qui vérifie les mêmes conditions associées aux règles d'inférence que  $R$  alors pour tout  $t$ , si  $R(t)$  on a aussi  $S(t)$ . On appellera ce schéma de preuve, une **preuve par minimalité** de la définition de  $R$ .

Dans le cas de la définition de **pair**, cela se traduit par le principe suivant :

### Proposition 3.3.2 (Principe de minimalité sur les entiers pairs.)

— Soit  $\phi(n)$  une propriété à montrer sur les entiers, si on peut démontrer les deux propositions suivantes :



1.  $\phi(0)$  est vérifié ;
  2. pour tout  $n$  tel que  $\text{pair}(n)$ , si  $\phi(n)$  est vérifié alors  $\phi(n+2)$  est vérifié ;
- alors on peut en déduire que pour tout  $n$  tel que  $\text{pair}(n)$  on a  $\phi(n)$  est vérifié.

Par exemple on peut utiliser ce principe pour prouver que si  $n$  vérifie  $\text{pair}$  alors il existe  $k$  tel que  $n = 2k$ . Il suffit de vérifier les deux conditions du principe précédent avec pour propriété  $\phi(n)$ , l'énoncé "il existe  $k$  tel que  $n = 2k$ ", c'est-à-dire :

1. il existe  $k$  tel que  $0 = 2k$  (il suffit de prendre  $k = 0$ )
2. soit  $n$  tel que  $\text{pair}(n)$ , supposons qu'il existe  $k$  tel que  $n = 2k$ , montrons qu'il existe  $k'$  tel que  $n+2 = 2k'$ . Il suffit de prendre  $k' = k+1$ .

L'exemple  $\text{pair}$  est particulièrement élémentaire et on peut trouver d'autre formulations équivalentes sans utiliser de règles d'inférence comme  $\exists k, n = 2k$  ou bien  $n \bmod 2 = 0$ .

**Définition 3.3.3 (Schéma général de minimalité)** Soit  $R \subseteq A$  définie par un système d'inférence. Le schéma de preuve par minimalité pour la définition de  $R$

- Soit  $\phi(x)$  une propriété paramétrée par  $x \in A$
- Pour chaque règle d'inférence de la relation  $R$ , de la forme  $\frac{R(u_1) \dots R(u_n) P_1 \dots P_k}{R(t)}$ , dont les paramètres sont  $x_1, \dots, x_p$ , il faut vérifier que la propriété  $\phi$  est préservée par la règle, ce qui s'écrit : pour n'importe quelles valeurs des paramètres  $x_1, \dots, x_p$  qui vérifient les prémisses de la règle (on a  $R(u_1) \dots R(u_n)$  et  $P_1 \dots P_k$ ) et tels que  $\phi(u_1) \dots \phi(u_n)$  sont vérifiés, alors  $\phi(t)$  est vérifié
- Si on a vérifié toutes les conditions précédentes, alors  $\phi(t)$  est vérifié pour tous les éléments  $t$  tels que  $R(t)$ .

**Exercice 3.8** On reprend la définition de l'exercice 3.7 des positions d'un robot.

1. Enoncer le principe de minimalité associé à cette définition.
2. Utiliser ce principe pour montrer que pour tout  $x, y$ , si  $\text{pos}(x, y)$  alors  $x + y \bmod 3 = 0$
3. En déduire que  $\text{pos}(1, 1)$  est faux

Les systèmes d'inférence sont utiles pour définir des relations générales qui ne sont pas des fonctions. Par exemple, si on suppose donnée une relation  $\text{ami}(x, y)$  qui représente le fait que  $x$  a déclaré que  $y$  est son ami dans un réseau social. Alors on peut simplement définir une relation  $\text{reseau}(x, y)$  qui représente le fait qu'il y a une chaîne d'amitiés entre  $x$  et  $y$ .

Il suffit de traduire les conditions suivantes :

1. Si  $x$  et  $y$  sont amis alors ils sont dans le même réseau.
2. Si  $x$  et  $y$  sont dans le même réseau et  $y$  et  $z$  sont amis alors  $x$  et  $z$  sont dans le même réseau.
3. Il n'y a pas d'autre manière d'être dans le même réseau : si quelqu'un est dans mon réseau, il est soit mon ami, soit l'ami de quelqu'un dans mon réseau

Ce qui s'exprime par les règles d'inférence suivantes :

$$\frac{\text{ami}(x, y)}{\text{reseau}(x, y)} \quad \frac{\text{reseau}(x, y) \quad \text{ami}(y, z)}{\text{reseau}(x, z)}$$

Il peut y avoir plusieurs systèmes d'inférence différents qui définissent la même relation (au sens où on peut prouver  $\text{reseau}(x, y)$  si et seulement si  $\text{reseau}_1(x, y)$  si et seulement si  $\text{reseau}_2(x, y)$ ) :

$$\frac{\text{ami}(x, y)}{\text{reseau}_1(x, y)} \quad \frac{\text{ami}(x, y) \quad \text{reseau}_1(y, z)}{\text{reseau}_1(x, z)}$$

ou encore :

$$\frac{\text{ami}(x, y)}{\text{reseau}_2(x, y)} \quad \frac{\text{reseau}_2(x, y) \quad \text{reseau}_2(y, z)}{\text{reseau}_2(x, z)}$$



Les systèmes d'inférence ont des applications en logique pour définir des systèmes de déduction, ils sont également beaucoup utilisés en informatique pour modéliser la sémantique des langages de programmation, par exemple les règles de typage ou les traductions entre langages en particulier dans le domaine de la compilation. C'est aussi une technique qui permet de modéliser des systèmes informatiques ayant des comportements non déterministes (systèmes parallèles et communicants).

### Systèmes de déduction

On va s'intéresser ici à des systèmes d'inférence qui vont nous permettre de prouver la validité de formules en utilisant leurs propriétés syntaxiques. On parlera de système de déduction logique, ou système de preuve.

Lorsqu'on fait un raisonnement, on a pour habitude de travailler de manière générale avec un ensemble d'hypothèses  $\Gamma$  et une formule  $P$ .

**Définition 3.3.4 (Séquent)** On appelle séquent un couple formé d'un ensemble fini de formules  $\Gamma$  (les hypothèses) et une formule  $P$  appelée la conclusion. On note le séquent  $\Gamma \vdash P$ .

La formule associée au séquent  $P_1, \dots, P_n \vdash Q$  est définie comme  $(P_1 \wedge \dots \wedge P_n) \Rightarrow Q$

Un séquent  $\Gamma \vdash P$  est valide si  $P$  est conséquence logique de  $\Gamma$  ( $\Gamma \models P$ ) ou de manière équivalente si la formule associée au séquent est valide.

**Notations associées aux séquents** Si  $\Gamma$  est un ensemble fini de formules  $\{A_1, \dots, A_n\}$ , on écrit  $A_1, \dots, A_n \vdash P$  pour la propriété  $\Gamma \vdash P$ . Si  $\Gamma$  est l'ensemble vide, on note simplement  $\vdash P$ . Si  $Q$  est une formule, on note  $\Gamma, Q$  l'ensemble  $\Gamma$  auquel on a ajouté la formule  $Q$  soit  $\Gamma \cup \{Q\}$  de même si  $\Gamma$  et  $\Delta$  sont des ensembles finis de formules, on note  $\Gamma, \Delta$  l'ensemble  $\Gamma \cup \Delta$  formé par leur union.


L'ensemble des variables libres d'un ensemble de formules  $\Gamma$  est par définition l'union des variables libres des formules de  $\Gamma$ , il est noté  $\text{Vl}(\Gamma)$ .

**Séquents prouvables** Nous allons définir par des systèmes d'inférence des relations pour capturer l'ensemble des séquents "prouvables". Traditionnellement, on n'introduit pas de notation supplémentaire. Ainsi suivant le contexte, la notation  $\Gamma \vdash P$  représentera l'objet séquent (un couple formé d'un ensemble d'hypothèses et d'une conclusion) ou bien la propriété qui dit que ce séquent est prouvable dans le système considéré.

Un arbre de preuve pour un séquent  $\Gamma \vdash P$  est un arbre de dérivation complet qui établit que le séquent est prouvable.

Faire une preuve d'une formule  $P$  se fera en construisant un arbre de dérivation dont la racine sera le séquent  $\vdash P$  (sans hypothèse).

On n'introduira évidemment que des systèmes corrects : tout séquent prouvable est valide et on cherchera à ce que les systèmes soient complets à savoir que tout séquent valide soit prouvable.

 Les règles d'inférence pour faire des preuves reposent sur la syntaxe des formules, ainsi il n'est pas correct de transformer dans un séquent une formule par une autre formule équivalente (sauf si une règle nous autorise explicitement à le faire).

### 3.3.2 Système de Hilbert

On peut définir un système simple pour dériver des formules valides dans une logique minimale (avec juste des variables propositionnelles et le connecteur  $\Rightarrow$ ). Soit  $\text{MIN}$  l'ensemble des formules de la logique minimale. On définit une relation sur cet ensemble, qui va représenter des formules prouvables.

La relation que l'on définit est notée  $\vdash P$  avec  $P \in \text{MIN}$ . Elle est donnée par le système d'inférence suivant qui comporte deux "axiomes" (des règles sans prémisses) et la règle de Modus Ponens :

$$(K) \frac{}{\vdash p \Rightarrow q \Rightarrow p} \quad (S) \frac{}{\vdash (p \Rightarrow q \Rightarrow r) \Rightarrow (p \Rightarrow q) \Rightarrow (p \Rightarrow r)} \quad (mp) \frac{\vdash p \Rightarrow q \quad \vdash p}{\vdash q}$$

**Exemple 3.14** On peut établir dans ce système que  $\vdash p \Rightarrow p$ , en construisant l'arbre de dérivation suivant :

$$\begin{array}{c}
 \begin{array}{c} S \\ \hline \vdash (p \Rightarrow (p \Rightarrow p) \Rightarrow p) \Rightarrow (p \Rightarrow p \Rightarrow p) \Rightarrow (p \Rightarrow p) \end{array} \quad \begin{array}{c} K \\ \hline \vdash p \Rightarrow (p \Rightarrow p) \Rightarrow p \end{array} \\
 \begin{array}{c} mp \\ \hline \vdash (p \Rightarrow p \Rightarrow p) \Rightarrow (p \Rightarrow p) \end{array} \quad \begin{array}{c} K \\ \hline \vdash p \Rightarrow p \Rightarrow p \end{array} \\
 \hline
 \vdash p \Rightarrow p
 \end{array}$$

Si on veut montrer que toute formule  $P$  telle que  $\vdash P$  est prouvable est aussi valide (si  $\vdash P$  alors  $\models P$ ) on va le faire en utilisant le principe de minimalité associé à la définition de  $\vdash$ .

Il nous suffit de montrer les trois propriétés, une pour chaque règle d'inférence pour n'importe quelles formules  $p, q$  et  $r$ .

1.  $\models p \Rightarrow q \Rightarrow p$
2.  $\models (p \Rightarrow q \Rightarrow r) \Rightarrow (p \Rightarrow q) \Rightarrow (p \Rightarrow r)$
3. si  $\models p \Rightarrow q$  et  $\models p$  alors  $\models q$

ce qui se vérifie simplement.

On étend ce système au cas du raisonnement sous hypothèses. Les règles vues précédemment sont transformées pour intégrer les hypothèses. Une règle est ajoutée qui dit que toute formule qui apparaît dans les hypothèses  $\Gamma$  est prouvable.

$$\begin{array}{c}
 \text{(hyp)} \frac{p \in \Gamma}{\Gamma \vdash p} \quad \text{(K)} \frac{}{\Gamma \vdash p \Rightarrow q \Rightarrow p} \quad \text{(S)} \frac{}{\Gamma \vdash (p \Rightarrow q \Rightarrow r) \Rightarrow (p \Rightarrow q) \Rightarrow (p \Rightarrow r)} \quad \text{(mp)} \frac{\Gamma \vdash p \Rightarrow q \quad \Gamma \vdash p}{\Gamma \vdash q}
 \end{array}$$

Si une formule est prouvable à partir d'hypothèses  $\Gamma$  est est a fortiori prouvable si on ajoute des hypothèses. Cette propriété essentielle s'appelle l'affaiblissement.

**Proposition 3.3.3 (Affaiblissement des hypothèses)** Si  $\Gamma \vdash p$ , alors pour tout ensemble  $\Delta$  tel que  $\Gamma \subseteq \Delta$ , on a  $\Delta \vdash p$ .

**Preuve:** La preuve se fait simplement en utilisant le principe de minimalité associé à la définition. Soit  $\Gamma$  et  $\Delta$  deux ensembles de formules telles que  $\Gamma \subseteq \Delta$ , la propriété à prouver est  $\Delta \vdash p$ .

On vérifie simplement que la propriété  $\Delta \vdash p$  est bien stable pour les quatre règles d'inférence qui définissent la relation  $\Gamma \vdash p$ .

Le seul cas non trivial est celui de la règle hypothèse pour laquelle on a  $p \in \Gamma$ , comme  $\Gamma \subseteq \Delta$ , on a aussi  $p \in \Delta$ , on en déduit  $\Delta \vdash p$ .  $\square$

Une autre propriété d'affaiblissement qui nous sera utile est la suivante.

**Proposition 3.3.4 (Affaiblissement de la conclusion)** Si  $\Gamma \vdash p$  alors  $\Gamma \vdash q \Rightarrow p$ .

**Preuve:** C'est une simple déduction sous hypothèse :

$$\begin{array}{c}
 K \frac{}{\Gamma \vdash p \Rightarrow q \Rightarrow p} \quad \Gamma \vdash p \\
 mp \frac{}{\Gamma \vdash q \Rightarrow p}
 \end{array}$$

$\square$

**Proposition 3.3.5 (Théorème de déduction)** Soit  $\Gamma$  un ensemble de formules et  $p, q$  deux formules. On a  $\Gamma, p \vdash q$  si et seulement si  $\Gamma \vdash p \Rightarrow q$ .

**Preuve:**

- Si  $\Gamma \vdash p \Rightarrow q$  alors
  - par le théorème d'affaiblissement des hypothèses, on a aussi  $\Gamma, p \vdash p \Rightarrow q$
  - par la règle hypothèse,  $\Gamma, p \vdash p$
  - par la règle modus-ponens, on a donc  $\Gamma, p \vdash q$
- Réciproquement, on suppose que  $\Gamma, p \vdash q$  on va montrer  $\Gamma \vdash p \Rightarrow q$  en utilisant le principe de minimalité associé à la définition de la relation  $\Gamma, p \vdash q$ .
  1. règle hypothèse  $\Gamma, p \vdash q$  avec  $q \in \Gamma, p$  :
    - soit  $q \in \Gamma$  et alors on a  $\Gamma \vdash q$  et donc par affaiblissement de la conclusion on a  $\Gamma \vdash p \Rightarrow q$ ,
    - si  $q = p$  alors on obtient la dérivation de  $\Gamma \vdash p \Rightarrow p$  comme dans l'exemple 3.14.
  2. pour les deux règles *S* et *K*, elles sont dérivables quelque soit l'ensemble d'hypothèses, il suffit donc à chaque fois d'appliquer la proposition d'affaiblissement de la conclusion
  3. pour la règle modus ponens qui est 
$$\frac{\Gamma, p \vdash r \Rightarrow q \quad \Gamma, p \vdash r}{\Gamma, p \vdash q}$$
 on a donc  $\Gamma \vdash p \Rightarrow r \Rightarrow q$  et  $\Gamma \vdash p \Rightarrow r$ , on construit alors la dérivation suivante :

$$\frac{\begin{array}{c} S \frac{\Gamma \vdash (p \Rightarrow r \Rightarrow q) \Rightarrow (p \Rightarrow r) \Rightarrow (p \Rightarrow q)}{\Gamma \vdash (p \Rightarrow r \Rightarrow q) \Rightarrow (p \Rightarrow r) \Rightarrow (p \Rightarrow q)} \quad \Gamma \vdash p \Rightarrow r \Rightarrow q \\ mp \frac{\Gamma \vdash (p \Rightarrow r \Rightarrow q) \Rightarrow (p \Rightarrow r) \Rightarrow (p \Rightarrow q) \quad \Gamma \vdash p \Rightarrow r \Rightarrow q}{\Gamma \vdash (p \Rightarrow r \Rightarrow q) \Rightarrow (p \Rightarrow r) \Rightarrow (p \Rightarrow q)} \\ mp \frac{\Gamma \vdash (p \Rightarrow r \Rightarrow q) \Rightarrow (p \Rightarrow r) \Rightarrow (p \Rightarrow q) \quad \Gamma \vdash p \Rightarrow r \Rightarrow q}{\Gamma \vdash p \Rightarrow q} \end{array}}$$

□

Dans le système de Hilbert, la relation de déduction est définie avec un ensemble fixe d'hypothèses, le théorème de déduction peut se voir comme une règle “dérivée” qui transforme simultanément l'ensemble des hypothèses et la conclusion.

L'idée des calculs de séquents est d'intégrer dans les règles d'inférence des manipulations sur l'ensemble des hypothèses. Il en existe plusieurs variantes. Nous présentons ici la déduction naturelle et le calcul des séquents à conclusions multiples, deux systèmes qui ont été introduits par Gerhard Gentzen.

### 3.3.3 Déduction naturelle

Nous donnons ici les principes de base de la **déduction naturelle**, un système qui est proche du raisonnement mathématique usuel.

Nous allons définir une relation qui correspond au sous-ensemble des séquents “prouvables en déduction naturelle”. Dans cette section, la notation  $\Gamma \vdash P$  représente soit l'objet séquent, soit la propriété qui dit que ce séquent est prouvable en déduction naturelle.

Nous présentons chacune des règles sous forme de règles d'inférence

$$\frac{\Gamma_1 \vdash A_1 \quad \dots \quad \Gamma_n \vdash A_n}{\Gamma \vdash P}$$

qui permet de construire un **arbre de preuve** dont la racine (en bas) représente le séquent que l'on cherche à justifier et les feuilles sont des conditions suffisantes.

**Règles de base** Une première série de règles permet de terminer une preuve dans les cas “élémentaires” :

- Hypothèse : si la conclusion fait partie des hypothèses alors la preuve est terminée.

$$(hyp) \frac{A \in \Gamma}{\Gamma \vdash A}$$

- Si la conclusion est la formule atomique toujours vraie alors la preuve est terminée.

$$(\text{ax}) \frac{}{\Gamma \vdash \top}$$

**Règles d'introduction** Les règles suivantes expliquent comment ramener la preuve d'une formule à des preuves de certaines de ses sous-formules. On les appelle des **règles d'introduction** :

- Si on veut prouver  $A \wedge B$  alors on peut se ramener à démontrer d'une part la conclusion  $A$  d'autre part la conclusion  $B$ . Les hypothèses ne changent pas.

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

- Si on veut montrer  $A \vee B$  alors on a deux possibilités : on peut se ramener à démontrer la conclusion  $A$  avec les mêmes hypothèses ou bien on peut se ramener à démontrer la conclusion  $B$  avec les mêmes hypothèses. Attention l'utilisation de l'une ou l'autre de ces règles peut ne pas aboutir car il y a des situations où  $A \vee B$  est vrai sans qu'on puisse conclure que  $A$  est vrai ou  $B$  est vrai ... Par exemple, on peut déduire  $B \vee A$  à partir de l'hypothèse  $A \vee B$  mais à partir de l'hypothèse  $A \vee B$ , on ne peut déduire ni  $B$  ni  $A$ .

$$(\vee I_g) \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \quad (\vee I_d) \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$$

- Si on veut montrer  $\neg A$  alors on peut se ramener à démontrer une contradiction en ajoutant la propriété  $A$  dans les hypothèses.

$$(\neg I) \frac{\Gamma, A \vdash \perp}{\Gamma \vdash \neg A}$$

- Si on veut montrer  $A \Rightarrow B$  alors on peut se ramener à démontrer la conclusion  $B$  en ajoutant la propriété  $A$  dans les hypothèses.

$$(\Rightarrow I) \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}$$

- Si on veut montrer  $\forall x, P$  alors on se ramène à prouver  $P$  pour un  $x$  quelconque. La contrainte  $x$  quelconque impose que le nom  $x$  ne soit pas déjà utilisé ailleurs et donc qu'il ne soit pas libre dans les hypothèses. On peut toujours se ramener à ce cas là, car si  $x$  était utilisé dans les hypothèses, comme  $x$  est lié dans la formule  $\forall x, P$ , on peut procéder à un renommage : on choisit un nom  $y$  qui n'apparaît pas par ailleurs, et on remplace  $\forall x, P$  par  $\forall y, P[x \leftarrow y]$

$$(\forall I) \frac{\Gamma \vdash P \quad x \notin \text{VI}(\Gamma)}{\Gamma \vdash \forall x, P}$$

- Si on veut montrer  $\exists x, P$  alors il suffit de produire un objet  $t$ , et on se ramène à prouver  $P[x \leftarrow t]$ . Attention si on choisit mal l'objet  $t$ , le séquent en prémisses peut ne pas être prouvable.

$$(\exists I) \frac{\Gamma \vdash P[x \leftarrow t]}{\Gamma \vdash \exists x, P}$$

Ces règles sont suffisantes pour prouver les propriétés suivantes (sans hypothèses) :

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

mais elles ne sont pas suffisantes pour montrer une propriété comme  $(A \wedge B) \Rightarrow (A \vee B)$ . En effet elles se contentent d'ajouter des connecteurs dans la conclusion, nous allons expliquer maintenant des règles qui permettent d'"enlever" des connecteurs.

**Règles d'élimination** Les règles suivantes expliquent comment utiliser une preuve d'une formule en fonction du connecteur principal de cette formule. On appelle ces règles des règles d'élimination.

- Si on a prouvé l'absurde (ce qui signifie que les hypothèses sont contradictoires) alors avec les mêmes hypothèses, on peut prouver n'importe quoi :

$$(\perp E) \frac{\Gamma \vdash \perp}{\Gamma \vdash C}$$

- Si on a prouvé  $\neg A$  et que l'on sait aussi prouver  $A$  alors on a une contradiction et donc on peut prouver n'importe quoi :

$$(\neg E) \frac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash C}$$

- Si on a prouvé  $A \wedge B$  alors on peut en déduire  $A$  et on peut en déduire  $B$  :

$$(\wedge E_g) \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \quad (\wedge E_d) \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}$$

- Si on a prouvé  $A \vee B$  alors on peut raisonner par cas, en considérant deux instances de notre problème, l'un avec l'hypothèse  $A$  en plus et l'autre avec l'hypothèse  $B$  en plus :

$$(\vee E) \frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}$$

- Si on a prouvé  $A \Rightarrow B$  et que l'on peut prouver  $A$  alors on peut en déduire  $B$ , on retrouve la règle de Modus-Ponens :

$$(\Rightarrow E) \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}$$

- Si on a prouvé  $\forall x, P$ , alors on peut en déduire que  $P$  est vrai pour n'importe quel objet  $t$  :

$$(\forall E) \frac{\Gamma \vdash \forall x, P}{\Gamma \vdash P[x \leftarrow t]}$$

- Si on a prouvé  $\exists x, P$ , alors on peut donner un nom à cet objet (que l'on ne connaît pas) et ajouter l'hypothèse que cet objet satisfait  $P$ . Comme on ne connaît rien de l'objet (sauf qu'il existe), le nom que l'on choisit doit être neuf et donc ne doit pas être déjà utilisé. Comme le nom  $x$  est lié dans  $\exists x, P$ , on peut toujours procéder à un renommage  $\exists y, P[x \leftarrow y]$  pour choisir un nom nouveau qui convient.

$$(\exists E) \frac{\Gamma \vdash \exists x, P \quad \Gamma, P \vdash C \quad x \notin \text{VI}(\Gamma, C)}{\Gamma \vdash C}$$

**Raisonnement par l'absurde** Enfin, les règles précédentes ne suffisent pas à prouver toutes les formules valides (par exemple  $A \vee \neg A$ ), il faut y ajouter le schéma de raisonnement *classique* qui capture ce que l'on appelle le *raisonnement par l'absurde*. Si on veut prouver  $A$  alors on peut supposer  $\neg A$  et en déduire une contradiction.

$$\frac{\Gamma, \neg A \vdash \perp}{\Gamma \vdash A}$$

**Règles dérivées** On remarque que si une formule  $A$  est une conséquence d'hypothèses  $\Gamma$  alors a fortiori  $A$  est une conséquence d'un ensemble d'hypothèses  $\Delta$  qui contient les hypothèses de  $\Gamma$ . C'est vrai pour la règle hypothèse, puis c'est préservé par toutes les règles. Ce résultat peut s'énoncer de la manière suivante :

**Proposition 3.3.6 (Affaiblissement des hypothèses)** Si  $\Gamma \vdash A$  et  $\Gamma \subseteq \Delta$  alors  $\Delta \vdash A$

On peut donc oublier des hypothèses inutiles lorsque l'on construit une preuve. Attention par contre à ne pas supprimer d'hypothèse qui peuvent être nécessaires pour compléter la preuve.

À partir des règles de base que l'on s'est données, on peut déduire d'autres schémas de preuve *naturels* que l'on pourra utiliser librement.

— Preuve par coupure

$$\text{Cut} \frac{\Gamma, A \vdash B \quad \Gamma \vdash A}{\Gamma \vdash B}$$

C'est une simple conséquence des règles pour le connecteur  $\Rightarrow$

$$\Rightarrow E \frac{\Rightarrow I \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \quad \Gamma \vdash A}{\Gamma \vdash B}$$

— Pour chaque connecteur logique, on peut spécialiser la règle d'élimination au cas où la formule à éliminer est déjà dans les hypothèses. Cela remplace une branche de l'arbre qui se conclurait par l'utilisation d'une règle hypothèse par une condition de la forme  $P \in \Gamma$  qui se vérifie trivialement. Pour la conjonction et la quantification universelle, la forme de la règle est légèrement différente, elle nous permet d'enrichir l'ensemble des hypothèses pour continuer la preuve.

— Contradiction :

$$(\perp H) \frac{\perp \in \Gamma}{\Gamma \vdash C}$$

— Negation :

$$(\neg H) \frac{\Gamma \vdash A \quad \neg A \in \Gamma}{\Gamma \vdash C}$$

— Conjonction :

$$(\wedge H) \frac{\Gamma, A, B \vdash C \quad A \wedge B \in \Gamma}{\Gamma \vdash C}$$

— Disjonction :

$$(\vee H) \frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C \quad A \vee B \in \Gamma}{\Gamma \vdash C}$$

— Implication :

$$(\Rightarrow H) \frac{\Gamma, B \vdash C \quad \Gamma \vdash A \quad A \Rightarrow B \in \Gamma}{\Gamma \vdash C}$$

— Quantification universelle :

$$(\forall H) \frac{\Gamma, P[x \leftarrow t] \vdash C \quad \forall x, P \in \Gamma}{\Gamma \vdash C}$$

— Quantification existentielle :

$$(\exists H) \frac{\Gamma, P \vdash C \quad x \notin \text{Vl}(\Gamma, C) \quad \exists x, P \in \Gamma}{\Gamma \vdash C}$$

## Récapitulatif des règles de la déduction naturelle

hypothèse	(hyp) $\frac{}{A, \Gamma \vdash A}$		
absurde	(abs) $\frac{\neg A, \Gamma \vdash \perp}{\Gamma \vdash A}$		
	introduction	elimination	élimination hypothèse (dérivée)
$\perp$		$\frac{\Gamma \vdash \perp}{\Gamma \vdash C}$	$\frac{\perp \in \Gamma}{\Gamma \vdash C}$
$\top$	$\frac{}{\Gamma \vdash \top}$		
$\neg$	$\frac{\Gamma, A \vdash \perp}{\Gamma \vdash \neg A}$	$\frac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash C}$	$\frac{\Gamma \vdash A \quad \neg A \in \Gamma}{\Gamma \vdash C}$
$\wedge$	$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$	$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \quad \wedge E \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}$	$\frac{\Gamma, A, B \vdash C \quad A \wedge B \in \Gamma}{\Gamma \vdash C}$
$\vee$	$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$	$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}$	$\frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C \quad A \vee B \in \Gamma}{\Gamma \vdash C}$
$\Rightarrow$	$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}$	$\frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}$	$\frac{\Gamma, B \vdash C \quad \Gamma \vdash A \quad A \Rightarrow B \in \Gamma}{\Gamma \vdash C}$
$\forall$	$\frac{\Gamma \vdash P \quad x \notin \text{VI}(\Gamma)}{\Gamma \vdash \forall x, P}$	$\frac{\Gamma \vdash \forall x, P}{\Gamma \vdash P[x \leftarrow t]}$	$\frac{\Gamma, P[x \leftarrow t] \vdash C \quad \forall x, P \in \Gamma}{\Gamma \vdash C}$
$\exists$	$\frac{\Gamma \vdash P[x \leftarrow t]}{\Gamma \vdash \exists x, P}$	$\frac{\Gamma \vdash \exists x, P \quad \Gamma, P \vdash C \quad x \notin \text{VI}(\Gamma, C)}{\Gamma \vdash C}$	$\frac{\Gamma, P \vdash C \quad x \notin \text{VI}(\Gamma, C) \quad \exists x, P \in \Gamma}{\Gamma \vdash C}$

**Exemple** Dérivation de  $\neg A \vee B \Rightarrow A \Rightarrow B$ .

Au départ il n'y a pas d'hypothèse, le séquent à montrer est  $\vdash \neg A \vee B \Rightarrow A \Rightarrow B$ . On commence par faire des introductions du connecteur  $\Rightarrow$ , on a donc deux hypothèses  $\neg A \vee B, A$  et on doit montrer  $B$ .

On raisonne par cas sur la preuve de  $\neg A \vee B$  (vraie par hypothèse).

- On doit montrer  $B$  sous les hypothèses  $\neg A \vee B, A, \neg A$  ce qui est vrai par contradiction ( $A$  et  $\neg A$  sont vrais par hypothèse);
- On doit aussi montrer  $B$  sous les hypothèses  $\neg A \vee B, A, B$  vrai par hypothèse.

L'arbre de preuve correspondant s'écrit :

$$\begin{array}{c}
 \text{hyp} \frac{}{\neg A \vee B, A, \neg A \vdash A} \\
 \neg H \frac{}{\neg A \vee B, A, \neg A \vdash B} \quad \text{hyp} \frac{}{\neg A \vee B, A, B \vdash B} \\
 \vee H \frac{}{\neg A \vee B, A \vdash B} \\
 \Rightarrow I \frac{}{\neg A \vee B \vdash A \Rightarrow B} \\
 \Rightarrow I \frac{}{\vdash \neg A \vee B \Rightarrow A \Rightarrow B}
 \end{array}$$

**Exercice 3.9 (Preuve en déduction naturelle)** Construire des démonstrations en déduction naturelle des formules et séquents suivants :



1.  $A \Rightarrow (B \Rightarrow A)$
2.  $(A \Rightarrow \neg A) \Rightarrow \neg A$
3.  $(\exists x, P(x)) \Rightarrow \neg \forall x, \neg P(x)$
4.  $\neg \exists x, P(x) \vdash \forall y, \neg P(y)$

**Exercice 3.10 (Raisonnement par l'absurde en déduction naturelle)** Construire des arbres de dérivation pour faire les preuves en déduction naturelle des séquents suivants :

1.  $P \vdash \neg \neg P$  et  $\neg \neg P \vdash P$  (pour ce séquent, commencer par la règle de raisonnement par l'absurde)
2.  $\neg(\neg P \vee P) \vdash \neg P$ , en déduire en utilisant le raisonnement par l'absurde une dérivation de  $\vdash \neg P \vee P$
3. Paradoxe du buveur :  $B \stackrel{\text{def}}{=} \exists x, (\neg P(x) \Rightarrow \forall y, \neg P(y))$ . On suppose l'existence dans le langage d'un objet  $a$ . On procèdera en démontrant d'abord les séquents  $\forall x, \neg P(x) \vdash B$  et  $\exists x, P(x) \vdash B$  et on pourra utiliser les séquents démontrés précédemment ainsi que la règle de coupure.

### Déduction naturelle et programmation fonctionnelle

La déduction naturelle a des liens très étroits avec la programmation fonctionnelle typée. Cette correspondance est connue sous le nom d'**isomorphisme de Curry-Howard**. Plus précisément on peut faire correspondre les formules logiques avec des types. Par exemple l'implication  $A \Rightarrow B$  correspond au type des fonctions de  $A$  dans  $B$  et la conjonction  $A \wedge B$  correspond au type des paires formées d'éléments de  $A$  et de  $B$ . Les hypothèses du séquent correspondent aux types des variables du programme. Chaque règle de preuve correspond à une construction de programme. Ainsi une preuve d'un séquent va correspondre à un programme bien typé.

Les calculs qui peuvent se faire dans le langage de programmation correspondent à des transformations de preuves.

Le fait que les constructions de programmes fonctionnels soient plus sûres que leur correspondant impératifs ou objets et que leur preuve se fasse plus simplement est lié à cette similarité avec nos modes de raisonnement usuels. Néanmoins, afin de permettre l'expression de l'ensemble des fonctions calculables, les langages de programmation autorisent des constructions récursives générales et boucles non bornées. Les méthodes de preuves sont elles beaucoup plus restrictives afin de rester correctes.

L'isomorphisme de Curry-Howard est également à la base d'un système de preuve interactif comme COQ (ou LEAN). La preuve est représentée par un terme fonctionnel qui doit satisfaire des règles de typage pour garantir la correction de la preuve.

### 3.3.4 Calcul des séquents multi-conclusions

Les règles de bases de la déduction naturelle permettent de faire naviguer des formules entre hypothèse et conclusion mais ne transforment pas la structure des formules dans les hypothèses. Or de telles règles sont utiles (nous les avons d'ailleurs fait apparaître comme des règles dérivées).

Par ailleurs la déduction naturelle repose sur un certain nombre de règles qui ne sont pas "inversibles" c'est-à-dire que la conclusion peut être vraie sans que les prémisses le soient. C'est le cas des règles d'introduction de la disjonction et de l'existentielle ou des règles d'élimination de la conjonction, de la quantification universelle et de l'implication.

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \quad \frac{\Gamma \vdash P[x \leftarrow t]}{\Gamma \vdash \exists x, P} \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \quad \frac{\Gamma \vdash \forall x, P}{\Gamma \vdash P[x \leftarrow t]} \quad \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}$$

L'utilisation de ces règles peut nous mener à des impasses. L'utilisation des règles d'élimination d'hypothèses pour la conjonction et la quantification universelle permet de traiter ces connecteurs sans perdre d'information.

$$\frac{\Gamma, A, B \vdash C \quad A \wedge B \in \Gamma}{\Gamma \vdash C} \quad \forall H \frac{\Gamma, P[x \leftarrow t] \vdash C \quad \forall x, P \in \Gamma}{\Gamma \vdash C}$$

Mais cela ne résoud pas les questions de la disjonction, de l'existentielle et de l'implication.

On voit qu'il y a une dissymétrie dans les séquents entre les hypothèses qui comportent plusieurs formules et la conclusion qui est limitée à une seule formule. L'idée du calcul des séquents est d'autoriser aussi un ensemble de formules à droite du symbole  $\vdash$ .

**Définition 3.3.5 (Séquent à conclusions multiples)** On appelle *séquent à conclusions multiples* (ou séquent multi-conclusions) un couple formé de deux ensembles finis de formules  $\Gamma$  (les *hypothèses*) et  $\Delta$  (les *conclusions*). On note le séquent  $\Gamma \vdash \Delta$ .

La *formule associée au séquent multi-conclusions*  $P_1, \dots, P_n \vdash Q_1, \dots, Q_p$  est définie comme  $(P_1 \wedge \dots \wedge P_n) \Rightarrow (Q_1 \vee \dots \vee Q_p)$ . Si  $\Delta = \emptyset$  ( $p = 0$ ) alors  $(Q_1 \vee \dots \vee Q_p)$  est par définition la formule  $\perp$  et la formule associée au séquent s'écrit  $(P_1 \wedge \dots \wedge P_n) \Rightarrow \perp$  ce qui est équivalent à  $\neg(P_1 \wedge \dots \wedge P_n)$ .

Un séquent  $\Gamma \vdash \Delta$  est *valide* si et seulement si la formule associée au séquent est valide ou, de manière équivalente, si pour toute interprétation  $I$  telle que  $I \models \Gamma$ , il existe  $P \in \Delta$  tel que  $I \models P$ .

On voit donc que l'ensemble des hypothèses à gauche du signe  $\vdash$  est interprété comme une *conjonction* (toutes les hypothèses sont vraies) alors que l'ensemble des conclusions à droite du signe  $\vdash$  est interprété comme une *disjonction* (au moins une des conclusions est vraie).

Les règles vont alors pouvoir prendre une forme symétrique entre gauche et droite. Chaque règle porte sur une des formules dans les hypothèses (on dit que c'est une règle *gauche*) ou dans les conclusions (on dit que c'est une règle *droite*) et va éliminer le connecteur ou quantificateur principal de cette formule pour se ramener à un ou plusieurs séquents à prouver.

Nous appellerons *système G* le système de déduction défini par l'ensemble des règles suivantes.

**Règles du système G** Les règles permettent de décomposer les connecteurs soit dans la partie gauche du séquent, soit dans la partie droite. On utilise des lettres grecques  $\Gamma, \Delta$  pour représenter des ensembles arbitraires (finis) de formules. On rappelle que l'ordre des formules n'intervient pas dans la définition d'un ensemble, ainsi les ensembles  $A, B, C$  et  $C, A, B$  sont égaux.

Un premier jeu de règles permet de conclure une preuve dans des cas triviaux (une contradiction dans les hypothèses, une propriété triviale dans les conclusions ou bien une conclusion qui est également une hypothèse) :

$$(\text{JOK}) \frac{}{\perp, \Gamma \vdash \Delta} \quad (\text{TRIV}) \frac{}{\Gamma \vdash \Delta, \top} \quad (\text{HYP}) \frac{}{A, \Gamma \vdash \Delta, A}$$

Nous donnons ensuite les règles pour chaque connecteur : on retrouve la règle JOK qui est la règle gauche de  $\perp$  et la règle TRIV qui est la règle droite de  $\top$ .

	gauche	droite
$\perp$	$\frac{}{\perp, \Gamma \vdash \Delta}$	$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, \perp}$
$\top$	$\frac{\Gamma \vdash \Delta}{\top, \Gamma \vdash \Delta}$	$\frac{}{\Gamma \vdash \Delta, \top}$
$\neg$	$\frac{\Gamma \vdash \Delta, A}{\neg A, \Gamma \vdash \Delta}$	$\frac{A, \Gamma \vdash \Delta}{\Gamma \vdash \Delta, \neg A}$
$\wedge$	$\frac{A, B, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta}$	$\frac{\Gamma \vdash \Delta, A \quad \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \wedge B}$
$\vee$	$\frac{A, \Gamma \vdash \Delta \quad B, \Gamma \vdash \Delta}{A \vee B, \Gamma \vdash \Delta}$	$\frac{\Gamma \vdash \Delta, A, B}{\Gamma \vdash \Delta, A \vee B}$
$\Rightarrow$	$\frac{\Gamma \vdash \Delta, A \quad B, \Gamma \vdash \Delta}{A \Rightarrow B, \Gamma \vdash \Delta}$	$\frac{A, \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \Rightarrow B}$
$\forall$	$\frac{P[x \leftarrow t], (\forall x, P), \Gamma \vdash \Delta}{(\forall x, P), \Gamma \vdash \Delta}$	$\frac{\Gamma \vdash \Delta, P \quad x \notin \text{VI}(\Gamma, \Delta)}{\Gamma \vdash \Delta, (\forall x, P)}$
$\exists$	$\frac{P, \Gamma \vdash \Delta \quad x \notin \text{VI}(\Gamma, \Delta)}{(\exists x, P), \Gamma \vdash \Delta}$	$\frac{\Gamma \vdash \Delta, (\exists x, P), P[x \leftarrow t]}{\Gamma \vdash \Delta, (\exists x, P)}$

**Exemple 3.15 (Preuve propositionnelle en calcul des séquents)** On peut montrer  $\neg(A \wedge B) \vdash \neg A \vee \neg B$ .

$$\begin{array}{c}
 \text{hyp } \frac{}{A \vdash \neg B, A} \quad \text{hyp } \frac{}{B \vdash \neg A, B} \\
 \neg d \frac{}{\vdash \neg A, \neg B, A} \quad \neg d \frac{}{\vdash \neg A, \neg B, B} \\
 \wedge d \frac{}{\vdash \neg A, \neg B, A \wedge B} \\
 \neg g \frac{}{\neg(A \wedge B) \vdash \neg A, \neg B} \\
 \vee d \frac{}{\neg(A \wedge B) \vdash \neg A \vee \neg B}
 \end{array}$$

**Exemple 3.16 (Preuves avec quantificateurs en calcul des séquents)**

—  $\exists x, (p(x) \vee q(x)) \vdash (\exists x, p(x)) \vee (\exists x, q(x))$

$$\begin{array}{c}
 \text{hyp } \frac{}{p(x) \vdash p(x), (\exists x, p(x)), (\exists x, q(x))} \quad \vdots \\
 \exists d \frac{}{p(x) \vdash (\exists x, p(x)), (\exists x, q(x))} \quad \frac{}{q(x) \vdash (\exists x, p(x)), (\exists x, q(x))} \\
 \vee g \frac{}{p(x) \vee q(x) \vdash (\exists x, p(x)), (\exists x, q(x))} \\
 \exists g \frac{}{\exists x, (p(x) \vee q(x)) \vdash (\exists x, p(x)), (\exists x, q(x))} \\
 \vee d \frac{}{\exists x, (p(x) \vee q(x)) \vdash (\exists x, p(x)) \vee (\exists x, q(x))}
 \end{array}$$

—  $\neg(\exists y, p(y)) \vdash \forall x, \neg p(x)$

$$\begin{array}{c}
 \text{hyp} \frac{}{p(x) \vdash p(x), (\exists y, p(y))} \\
 \exists d \frac{}{p(x) \vdash \exists y, p(y)} \\
 \neg d \frac{}{\vdash (\exists y, p(y)), \neg p(x)} \\
 \forall d \frac{}{\vdash (\exists y, p(y)), (\forall x, \neg p(x))} \\
 \neg g \frac{}{\neg(\exists y, p(y)) \vdash \forall x, \neg p(x)}
 \end{array}$$

### Propriétés du Calcul des séquents

On établit simplement des propriétés sur la structure des preuves. Un séquent prouvable le reste si on ajoute des hypothèses ou des conclusions.

**Proposition 3.3.7 (Affaiblissement)** Si  $\Gamma \vdash \Delta$  alors  $\Gamma, \Gamma' \vdash \Delta, \Delta'$

**Preuve:** On utilise une preuve par minimalité de la définition de  $\Gamma \vdash \Delta$ . □

Si un séquent contient des variables libres et est prouvable alors il en est de même pour le séquent dans lequel la variable libre a été remplacée par un terme arbitraire.

**Proposition 3.3.8 (Substitution)** Si  $\Gamma \vdash \Delta$  alors  $\Gamma[x \leftarrow t] \vdash \Delta[x \leftarrow t]$

**Preuve:** On utilise une preuve par minimalité de la définition de  $\Gamma \vdash \Delta$ . □

Une autre propriété importante est que toutes les règles sont inversibles c'est-à-dire qu'une interprétation rend vrai un séquent en conclusion d'une règle si et seulement si elle rend vrais les séquents qui apparaissent dans les prémisses.

**Proposition 3.3.9** Les règles du système  $G$  sont correctes et inversibles.

**Preuve:** On regarde les règles une par une en utilisant les propriétés des connecteurs et quantificateurs.

— On peut se ramener à la validité des formules associées. Si  $\Gamma = A_1, \dots, A_n$  on note  $M$  la formule  $A_1 \wedge \dots \wedge A_n$  et si  $\Delta = B_1, \dots, B_p$  on note  $P$  la formule  $B_1 \vee \dots \vee B_p$ .

Si on s'intéresse à la règle gauche de l'implication :  $\frac{\Gamma \vdash \Delta, A \quad \Gamma, B \vdash \Delta}{\Gamma, A \Rightarrow B \vdash \Delta}$

Les prémisses se traduisent par les formules  $M \Rightarrow P \vee A$  et  $B \wedge M \Rightarrow P$  la conclusion se traduit par la formule  $(A \Rightarrow B) \wedge M \Rightarrow P$  On vérifie que les formules  $(M \Rightarrow P \vee A) \wedge (B \wedge M \Rightarrow P)$  et  $(A \Rightarrow B) \wedge M \Rightarrow P$  sont équivalentes (par exemple à l'aide de tables de vérité).

— Une autre manière de raisonner est d'analyser directement la valeur de vérité d'un séquent. Un séquent est faux dans une interprétation, si et seulement si toutes les hypothèses sont vraies et toutes les conclusions sont fausses.

On peut donc faire directement les tables de vérité des trois séquents qui apparaissent dans la règle gauche de l'implication et on remarque que le séquent conclusion est vrai si et seulement si les deux séquents de prémisses sont vrais.

$\Gamma$	$\Delta$	$A$	$B$	$\Gamma \vdash \Delta, A$	$\Gamma, B \vdash \Delta$	$\Gamma, A \Rightarrow B \vdash \Delta$
$F$	$-$	$-$	$-$	$V$	$V$	$V$
$V$	$V$	$-$	$-$	$V$	$V$	$V$
$V$	$F$	$F$	$-$	$F$	$-$	$F$
$V$	$F$	$V$	$V$	$V$	$F$	$F$
$V$	$F$	$V$	$F$	$V$	$V$	$V$

— Pour les règles des quantificateurs

$$(\forall g) \frac{P[x \leftarrow t], (\forall x, P), \Gamma \vdash \Delta}{(\forall x, P), \Gamma \vdash \Delta} \quad (\exists d) \frac{\Gamma \vdash \Delta, (\exists x, P), P[x \leftarrow t]}{\Gamma \vdash \Delta, (\exists x, P)}$$

on remarque qu'elles sont inversibles (cas particulier d'affaiblissement). On peut donc utiliser plusieurs fois le  $\forall$  en hypothèse ou proposer plusieurs témoins pour un  $\exists$  en conclusion. □

**!** Des présentations alternatives du calcul des séquents utilisent plutôt des multi-ensembles (des éléments peuvent apparaître plusieurs fois) à la fois dans les hypothèses et les conclusions. Cela nécessite d'introduire une règle explicite dite de *contraction* qui remplace deux formules égales par une seule. Cela a un impact sur les règles  $\forall g$  et  $\exists d$  dans lesquels les formules  $\forall x, P, \exists x, P$  ne sont pas reportées systématiquement dans les prémisses, l'utilisateur doit explicitement avoir recours à la contraction pour garder la formule quantifiée dans le séquent. Notre formulation garantit que toutes les règles sont inversibles (pas de perte d'information).

**Proposition 3.3.10 (Propriétés de la sous-formule)** *toutes les formules qui apparaissent dans un arbre de preuve de  $\Gamma \vdash \Delta$  sont des sous-formules de  $\Gamma, \Delta$  (avec la définition que  $P[x \leftarrow t]$  est une sous-formule de  $\forall x, P$  et de  $\exists x, P$ )*

Ce système a de bonnes propriétés pour l'automatisation du raisonnement dans le cas propositionnel. En effet chaque séquent qui apparaît en prémisses d'une règle d'inférence contient moins de connecteurs logiques que celui qui est en conclusion et donc le processus qui consiste à appliquer les règles logiques s'arrête forcément. La profondeur de l'arbre est au maximum égale au nombre de symboles dans le séquent. Nous reviendrons sur ce point dans le chapitre 4

### Règle de coupure

**Définition 3.3.6 (Règle de coupure)** *On peut ajouter au système la règle dite de *coupure*.*

$$\frac{\Gamma \vdash \Delta, A \quad \Gamma', A \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}$$

Cette règle est correcte, réversible mais elle ne préserve pas la propriété de la sous formule. Un théorème établit que la règle de coupure est redondante :

**Proposition 3.3.11 (Elimination des coupures)** *Si  $\Gamma \vdash \Delta$  en utilisant la règle de coupure alors  $\Gamma \vdash \Delta$  sans la règle de coupure.*

**Preuve:** La preuve se fait en transformant les coupures pour faire "diminuer" leur taille. Elle est assez technique puisqu'il faut explorer tous les cas, de plus l'argument de terminaison n'est pas immédiat. Nous donnons ici quelques cas spécifiques.

— Cas d'une règle hypothèse

$$\text{cut} \frac{\Gamma_0, P \vdash \Delta, P \quad \Gamma', P \vdash \Delta'}{\Gamma_0, P, \Gamma' \vdash \Delta, \Delta'}$$

se transforme en un affaiblissement

$$\frac{\Gamma', P \vdash \Delta'}{\Gamma_0, P, \Gamma' \vdash \Delta, \Delta'}$$

— Interaction règles gauche/droite  $P = \forall x, Q, x \notin V(\Gamma, \Delta)$

$$\text{cut} \frac{\forall d \frac{\Gamma \vdash \Delta, Q}{\Gamma \vdash \Delta, (\forall x, Q)} \quad \forall g \frac{\Gamma', (\forall x, Q), Q[x \leftarrow t] \vdash \Delta'}{\Gamma', (\forall x, Q) \vdash \Delta'}}{\Gamma, \Gamma' \vdash \Delta, \Delta'}$$

se transforme en

$$\text{cut} \frac{\Gamma \vdash \Delta, Q[x \leftarrow t] \quad \text{cut} \frac{\Gamma \vdash \Delta, (\forall x, Q) \quad \Gamma', (\forall x, Q), Q[x \leftarrow t] \vdash \Delta'}{\Gamma', Q[x \leftarrow t] \vdash \Delta'}}{\Gamma, \Gamma' \vdash \Delta, \Delta'}$$

□

**Raisonnement par l'absurde** Contrairement au cas de la déduction naturelle, il n'est pas nécessaire d'ajouter une règle pour traiter les preuves par l'absurde. Celles-ci se font avec les règles standard mais tirent partie de la présence de plusieurs formules de conclusion et de la possibilité de commencer à décomposer une des conclusions et d'utiliser les éléments collectés pour établir une autre des conclusions.

**Exemple 3.17 (Preuve du tiers exclu)** Ceci est illustré sur la preuve de  $\vdash \neg P \vee P$ . Après avoir coupé la disjonction en deux conclusions, on commence par essayer de prouver  $\neg P$ , on suppose donc que  $P$  est vrai et on devrait établir une contradiction, mais on a aussi la possibilité de s'attaquer à une autre des conclusions, ici  $P$  qui permet de conclure.

$$\begin{array}{c} \text{hyp} \frac{}{P \vdash P} \\ \neg d \frac{}{\vdash P, \neg P} \\ \vee d \frac{}{\vdash P \vee \neg P} \end{array}$$

**Exercice 3.11 (Preuve en calcul des séquents)** Prouver en utilisant les règles du calcul des séquents les propriétés suivantes :

1.  $A \Rightarrow (B \Rightarrow A)$
2.  $(\neg A \Rightarrow A) \Rightarrow A$
3.  $\neg \forall x, \neg P(x) \vdash \exists x, P(x)$
4.  $P \vdash \neg \neg P$  et  $\neg \neg P \vdash P$
5. Paradoxe du buveur :  $\exists x, (P(x) \Rightarrow \forall y, P(y))$ .

Le calcul des séquents à conséquences multiples nous fournira directement une procédure de décision de la validité pour les formules propositionnelles.

**Complétude** Nous avons construit des systèmes de preuves pour des calculs des séquents. Ces systèmes sont corrects dans le sens où si  $\Gamma \vdash P$  alors  $\Gamma \models P$ . Le théorème de complétude de Gödel nous dit que la réciproque est vraie c'est-à-dire que si une formule est valide alors elle est démontrable dans un système de déduction (la déduction naturelle ou le calcul des séquents sont deux tels systèmes).

Ce théorème reste vrai quand on raisonne dans une théorie donnée. Nous ferons dans le chapitre suivant la preuve pour le cas propositionnel. L'extension pour le calcul des séquents au premier ordre est plus technique (voir par exemple [5]).

**Déduction naturelle et calcul des séquents** Nous avons présenté deux systèmes de preuve pour raisonner sur des séquents. Il n'est pas très difficile de transformer les preuves entre ces deux systèmes. On établit les deux résultats suivants :

- Si  $\Gamma \vdash P$  en déduction naturelle alors  $\Gamma \vdash P$  en calcul des séquents
- Si  $\Gamma \vdash \Delta$  en calcul des séquents alors  $\Gamma, \neg\Delta \vdash \perp$  en déduction naturelle avec la règle de raisonnement par l'absurde.

Dans les deux cas, on raisonne par minimalité de la définition des séquents prouvable dans un système en montrant que chaque règle d'un système peut se dériver dans l'autre système.

### Conclusion

Cette partie nous a permis d'introduire la notion de système de preuves pour la logique et la structure de séquents qui généralise la notion de formule et est plus adaptée pour le raisonnement mécanisé. Les techniques de mise en forme clausale seront utiles pour faire des démonstrations par résolution ou pour les méthodes de décision de satisfiabilité. L'automatisation des preuves est le sujet du chapitre 4.



## Chapitre 4

# Automatiser les démonstrations

Dans ce chapitre nous introduisons plusieurs techniques pour automatiser les démonstrations en calcul propositionnel et logique du premier ordre.

### 4.1 Peut-on automatiser les démonstrations ?

Automatiser les démonstrations consiste à trouver un algorithme qui étant donnée une formule quelconque va pouvoir répondre à la question *est-ce que cette formule est valide ?*

On a vu que ce problème était équivalent à trouver un algorithme qui, pour une formule quelconque, répond à la question *est-ce que cette formule est satisfiable ?* En effet la validité de la formule  $A$  est équivalente à l'insatisfiabilité de la formule  $\neg A$ .

#### 4.1.1 Décidabilité de la logique propositionnelle

**Proposition 4.1.1** *La validité d'une formule propositionnelle est une propriété décidable.*

**Preuve:** Etant donnée une formule de la logique propositionnelle, elle a un nombre fini de variables,  $n$ . On peut calculer sa valeur de vérité pour chacune des  $2^n$  interprétations et en fonction du résultat, répondre à la question de la validité.  $\square$

Evidemment la méthode donnée ci-dessus n'est pas efficace et d'autres techniques existent pour rendre plus effective la procédure. On pourrait se dire que la question de la satisfiabilité d'une formule (*problème SAT*) est plus simple puisqu'il suffit de trouver une interprétation qui rend vraie la formule. Or ce n'est pas le cas, le problème reste intrinsèquement dur. On rappelle les définitions liées à la *NP-complétude* dans le cas du problème SAT :

- SAT est un *problème NP* : on peut construire une machine de Turing non-déterministe (à partir de chaque état, on peut choisir parmi un nombre fini de transitions possibles), chaque exécution est de taille polynomiale et le problème a une solution si et seulement si il existe un chemin d'exécution (au moins) qui réussit. Dans le cas du problème de la satisfiabilité d'un ensemble de formules, chaque étape peut faire un choix sur une variable (valeur vrai ou faux) et aboutir au calcul de la valeur de vérité des formules dans l'interprétation choisie. L'ensemble est satisfiable si et seulement si il y a au moins un chemin d'exécution qui donne la valeur vraie.
- SAT est *NP-complet* : tout problème NP est reductible de manière polynomiale à un problème SAT (codage propositionnel du problème).

**Proposition 4.1.2 (théorème de Cook, 71)** *Le problème de satisfiabilité d'une formule est un problème NP-complet en le nombre de variables propositionnelles.*

**Preuve:** (idée) La preuve du fait que le problème de satisfiabilité est NP-complet va constituer à ramener la résolution du problème NP à une question de satisfiabilité en logique propositionnelle, en faisant un codage logique de la machine de Turing qui résoud le problème initial.  $\square$

On peut chercher des solutions plus efficaces pour des classes particulières de formules.

Si la formule est déjà en **forme normale disjonctive** avec des conjonctions élémentaires simplifiées, alors la satisfiabilité est immédiate car la formule est soit la formule  $\perp$  donc insatisfiable soit de la forme  $C \vee D$  avec  $C$  une conjonction élémentaire et cette conjonction élémentaire a un modèle qui est un modèle de la formule complète. La formule est donc satisfiable. Ce résultat associé au théorème de Cook implique que la complexité de la satisfiabilité se retrouve dans la complexité de mettre une formule quelconque en FND.

On a vu que l'on avait des méthodes linéaires pour ramener la satisfiabilité d'une formule à la satisfiabilité d'une formule en forme normale conjonctive. Cependant, même en se limitant à une formule en FNC dont les clauses propositionnelles ont au plus trois littéraux (*problème dit 3-SAT*), le problème SAT reste NP-complet.

### 4.1.2 Semi-décidabilité de la logique du premier ordre

Contrairement au cas propositionnel, dans le cas de la logique du premier ordre il y a une infinité d'interprétations possibles pour les symboles. On ne peut donc pas les énumérer toutes en général ou construire des tables de vérité.

Dans le cas des **formules universelles** (avec uniquement des quantificateurs universels en forme prénexe), le théorème de Herbrand nous permet de nous restreindre à une recherche parmi les modèles de Herbrand (interprétation sur le domaine des termes clos). Associé au théorème de compacité, il nous donne une méthode algorithmique pour les formules universelles. Par ailleurs la skolémisation permet toujours de se ramener au cas des formules universelles.

Pour savoir si une formule  $P$  de la logique du premier ordre (ou un ensemble de formules) est insatisfiable.

1. mettre la formule  $P$  en forme de Skolem  $\forall x_1 \dots x_n, A$  avec  $A$  sans quantificateur. La formule  $P$  est satisfiable si et seulement si  $\forall x_1 \dots x_n, A$  est satisfiable.
2. énumérer les instances closes de  $A$  :  $A_1, \dots, A_n, \dots$  (en remplaçant les variables  $x_1, \dots, x_n$  par des termes clos), et pour chaque préfixe  $A_1, \dots, A_p$ , utiliser une méthode propositionnelle pour tester la satisfiabilité :
  - (a) si  $A_1, \dots, A_p$  est insatisfiable alors  $\forall x_1 \dots x_n, A$  est insatisfiable (en effet dans une interprétation où  $\forall x_1 \dots x_n, A$  est vrai, a fortiori toutes les instances  $A_1, \dots, A_p$  sont aussi vraies).
  - (b) si  $A_1, \dots, A_p$  est satisfiable et s'il n'existe pas de nouvelle instance close (cas où le domaine de Herbrand est fini) alors  $\forall x_1 \dots x_n, A$  est satisfiable car l'interprétation qui rend vraies toutes les formules de  $A_1, \dots, A_p$  définit un modèle de Herbrand dans lequel la formule  $\forall x_1 \dots x_n, A$  est vraie.
  - (c) sinon on ajoute une nouvelle instance  $A_{p+1}$  et on relance le problème

On peut établir les propriétés suivantes de l'algorithme :

- Si l'algorithme s'arrête alors la réponse donnée est correcte (simple application de la définition de la satisfiabilité) et on peut conclure que  $P$  est ou non satisfiable..
- Si la formule initiale est insatisfiable alors l'algorithme s'arrête et renvoie la bonne réponse (application du théorème de Herbrand et de la compacité).
- Si la formule initiale est satisfiable et le domaine de Herbrand infini, alors l'algorithme ne s'arrête pas. En effet il y a une infinité d'instances closes et tous les sous-ensembles seront satisfiables donc nécessiteront de relancer la recherche.

On en déduit les résultats suivants :

#### Proposition 4.1.3 (Semi-décidabilité de la logique du premier-ordre)

- *Le problème d'insatisfiabilité d'une formule du calcul des prédicats est semi-décidable.*

— Le problème de validité d'une formule du calcul des prédicats est semi-décidable.

**Preuve:** Le premier résultat est une conséquence de l'algorithme présenté ci-dessus et le second en découle en utilisant le fait que tester la validité de  $A$  est équivalent à tester l'insatisfiabilité de la formule  $\neg A$ .  $\square$

On peut s'interroger sur la possibilité de faire mieux, c'est-à-dire de décider de la validité d'une formule. La réponse est négative.

**Proposition 4.1.4 (Indécidabilité de la logique du premier-ordre)** Les problèmes de satisfiabilité ainsi que de validité d'une formule du calcul des prédicats sont indécidables.

**Preuve:** La preuve se fait en ramenant un problème indécidable comme l'arrêt d'une machine de Turing ou la correspondance de Post à un problème de validité en logique du premier ordre.  $\square$

**Exercice 4.1 (Logique et problème de Post)** On prend comme problème indécidable la *correspondance de Post*. Soit un ensemble à deux éléments  $a$  et  $b$  appelés *caractères*, on appelle *mot* une suite finie de caractères. On note  $\epsilon$  le mot vide qui correspond à une suite sans caractères et on note par simple juxtaposition  $uv$ , la concaténation de deux mots. On se donne un ensemble fini de couples de mots  $\{(u_1, v_1); \dots; (u_n, v_n)\}$ . Une *correspondance de Post* est une manière d'assembler ces paires de mots en une séquence (non vide)  $i_1, \dots, i_k$  pour que le mot  $u_{i_1} \dots u_{i_k}$  formé en regardant la première composante soit le même que le mot  $v_{i_1} \dots v_{i_k}$  formé en regardant la seconde composante.

**Exemple.** Soit le système  $S \stackrel{\text{def}}{=} \{(a, baa); (ab, aa); (bba, bb)\}$ .

Avec les notations que précédentes, on pose  $u_1 \stackrel{\text{def}}{=} a, v_1 \stackrel{\text{def}}{=} baa, u_2 \stackrel{\text{def}}{=} ab, v_2 \stackrel{\text{def}}{=} aa, u_3 \stackrel{\text{def}}{=} bba, v_3 \stackrel{\text{def}}{=} bb$ . Le mot  $u_3 u_2 u_3 u_1 = bba ab bba a$  est égal au mot  $v_3 v_2 v_3 v_1 = bb aa bb baa$ . On a donc bien une correspondance de Post pour ce système. Par contre si on regarde le système  $\{(ab, aa); (bba, bb)\}$  on montre qu'il n'y a pas de solution : en effet un tel mot solution ne peut pas commencer par  $(ab, aa)$  (car cela donne des mots dont la deuxième lettre diffère) ni par  $(bba, bb)$  car alors le premier mot sera toujours plus long que le second.

On peut prouver qu'un système particulier a ou n'a pas de solution mais on ne peut pas décider en général (à l'aide d'un programme) si un ensemble de couples de mots  $\{(u_1, v_1); \dots; (u_n, v_n)\}$  admet une correspondance de Post.

Nous allons maintenant montrer comment on peut ramener ce problème à un problème logique.

Pour cela on considère un langage avec une constante  $e$  pour représenter le mot vide et deux symboles de fonction unaires  $a$  et  $b$  qui permettent l'ajout de la lettre  $a$  ou de la lettre  $b$  au début d'un mot. Ainsi le mot  $baa$  est représenté dans la logique par le terme  $b(a(a(e)))$ . Le domaine de Herbrand (ensemble des termes clos) associé à ce langage correspond à l'ensemble des mots sur l'alphabet  $\{a, b\}$  dans le sens où on peut faire correspondre un terme à un mot et réciproquement.

On introduit également un prédicat binaire  $P$  et les formules suivantes :

$$\begin{aligned} A_0 &\stackrel{\text{def}}{=} P(e, e) & A_1 &\stackrel{\text{def}}{=} \forall xy, P(x, y) \Rightarrow P(a(x), b(a(y))) \\ A_2 &\stackrel{\text{def}}{=} \forall xy, P(x, y) \Rightarrow P(a(b(x)), a(a(y))) & A_3 &\stackrel{\text{def}}{=} \forall xy, P(x, y) \Rightarrow P(b(b(a(x))), b(b(y))) \end{aligned}$$

On remarque que l'axiome  $A_i$  correspond au couple  $(u_i, v_i)$  du problème de Post. On étudie le lien entre une solution au problème de Post et le fait que la formule  $\exists x, P(a(x), a(x)) \vee P(b(x), b(x))$  est conséquence logique de  $A_0, A_1, A_2, A_3$ .

Pour cela on introduit une relation binaire sur l'ensemble des termes clos :  $u \sim v$  si et seulement si il existe  $k \geq 0$  et une séquence  $i_1, \dots, i_k$  (éventuellement vide) tels que le terme  $u$  correspond au mot  $u_{i_1} \dots u_{i_k}$  et le terme  $v$  correspond au mot  $v_{i_1} \dots v_{i_k}$ .

1. Montrer que l'interprétation de Herbrand dans laquelle  $P(u, v)$  est vrai si et seulement si  $u \sim v$  est un modèle des formules  $A_0, A_1, A_2, A_3$ .
2. Montrer que si  $u \sim v$  alors  $A_0, A_1, A_2, A_3 \models P(u, v)$ . On pourra raisonner par récurrence sur la taille de la séquence  $i_1, \dots, i_k$ .

3. Montrer que  $A_0 \models \exists x, P(x, x)$ .
4. Montrer que si le système  $S$  admet une solution pour le problème de correspondance de Post alors  $A_0, A_1, A_2, A_3 \models \exists x, (P(a(x), a(x)) \vee P(b(x), b(x)))$ .
5. Sans utiliser le fait que l'on connaît une solution pour le problème de Post du système  $S$ , montrer réciproquement que si  $A_0, A_1, A_2, A_3 \models \exists x, (P(a(x), a(x)) \vee P(b(x), b(x)))$  alors le système de Post  $S$  admet une solution.

La preuve de semi-décidabilité nous a donné un algorithme naïf pour chercher une preuve d'une formule. En effet la méthode énumère *au hasard* l'ensemble des instances d'une formule et repose à chaque fois la question de la satisfiabilité. Pour améliorer cela, on peut chercher à engendrer des instances qui vont pouvoir interagir avec d'autres parties de la formule pour conduire à une contradiction.

**Compétences 4.1** — *Savoir que la logique propositionnelle est décidable tout en étant un problème intrinsèquement difficile*  
 — *Savoir que la logique du premier ordre n'est pas décidable mais qu'il existe des algorithmes qui peuvent ne pas terminer mais qui peuvent identifier les formules valides ou insatisfiables.*

## 4.2 Cas propositionnel

Dans cette section nous explorons plusieurs méthodes pour répondre à la question de la validité ou de la satisfiabilité d'une formule propositionnelle qui ne contient donc pour symboles que des variables propositionnelles et qui ne comporte pas de quantificateur.

### 4.2.1 Système G

Le système G est un calcul des séquents introduit au chapitre précédent (section 3.3.4).

Nous en rappelons les règles dans le cas propositionnel.

On a une règle “*hypothèse*” (HYP)  $\frac{}{A, \Gamma \vdash \Delta, A}$  et pour chaque connecteur, une règle gauche et une règle droite rappelées dans le tableau ci-dessous.

	gauche	droite
$\perp$	$\frac{}{\perp, \Gamma \vdash \Delta}$	$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, \perp}$
$\top$	$\frac{\Gamma \vdash \Delta}{\top, \Gamma \vdash \Delta}$	$\frac{}{\Gamma \vdash \Delta, \top}$
$\neg$	$\frac{\Gamma \vdash \Delta, A}{\neg A, \Gamma \vdash \Delta}$	$\frac{A, \Gamma \vdash \Delta}{\Gamma \vdash \Delta, \neg A}$
$\wedge$	$\frac{A, B, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta}$	$\frac{\Gamma \vdash \Delta, A \quad \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \wedge B}$
$\vee$	$\frac{A, \Gamma \vdash \Delta \quad B, \Gamma \vdash \Delta}{A \vee B, \Gamma \vdash \Delta}$	$\frac{\Gamma \vdash \Delta, A, B}{\Gamma \vdash \Delta, A \vee B}$
$\Rightarrow$	$\frac{\Gamma \vdash \Delta, A \quad B, \Gamma \vdash \Delta}{A \Rightarrow B, \Gamma \vdash \Delta}$	$\frac{A, \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \Rightarrow B}$

Ce système a de bonnes propriétés pour l'automatisation du raisonnement. En effet chaque séquent qui apparaît en prémisses d'une règle d'inférence contient moins de connecteurs logiques que celui qui est en conclusion et donc le processus qui consiste à appliquer les règles logiques s'arrête forcément.

On a vu aussi que toutes les règles étaient *correctes* et *inversibles* dans le sens qu'une interprétation rend vrai un séquent en conclusion d'une règle si et seulement si elle rend vrais tous les séquents qui apparaissent dans les prémisses.

En particulier, lorsqu'on construit un arbre de dérivation, si un nœud n'est pas valide, alors il existe une interprétation qui rend faux le séquent correspondant et on peut en déduire que la racine de la dérivation est également fautive pour cette interprétation. Et donc le séquent à la racine n'est pas valide. On en déduit une méthode de décision de la validité d'un séquent.

**Proposition 4.2.1 (Validité d'un séquent)** Soit un arbre de dérivation suivant les règles du système  $G$  pour un séquent  $\Gamma \vdash \Delta$  dont les feuilles ne contiennent plus de connecteurs logiques.

Le séquent  $\Gamma \vdash \Delta$  est valide si et seulement si la règle hypothèse s'applique à chacune des feuilles.

**Preuve:** Si les feuilles  $A_1, \dots, A_n \vdash B_1, \dots, B_p$  ne contiennent plus de connecteurs logiques, les formules  $A_1, \dots, A_n, B_1, \dots, B_p$  sont des variables propositionnelles.

Si toutes les feuilles sont des instances de la règle "hypothèse", l'arbre peut être complété et la conclusion est alors valide.

S'il existe une feuille de l'arbre  $A_1, \dots, A_n \vdash B_1, \dots, B_p$  qui n'est pas une instance de la règle "hypothèse" alors elle ne contient que des variables propositionnelles distinctes. On peut donc construire une interprétation qui rend vraies les variables  $A_i$  et fausses toutes les variables  $B_j$ . Comme les règles sont inversibles, le nœud qui est le parent de cette feuille est également faux dans cette interprétation. Et donc la racine de l'arbre qui est le séquent qui nous intéresse est faux dans cette interprétation. Ce séquent n'est pas valide.  $\square$

**Proposition 4.2.2 (Complétude du système  $G$  pour la logique propositionnelle)** Le système  $G$  est complet pour la logique propositionnelle, à savoir si le séquent  $\Gamma \vdash \Delta$  est valide alors  $\Gamma \vdash \Delta$  peut être prouvé par les règles du système  $G$ .

**Preuve:** Le résultat découle de la propriété 4.2.1. On montre la contraposée à savoir que si le séquent n'est pas prouvable alors il n'est pas valide. En effet, en décomposant tous les connecteurs, on peut toujours construire un arbre dont les feuilles sont uniquement composées de variables propositionnelles. Une des feuilles de cet arbre n'est pas une instance de la règle hypothèse (sinon le séquent serait prouvable) et donc par le résultat précédent, le séquent  $A_1, \dots, A_n \vdash B_1, \dots, B_p$  n'est pas valide.  $\square$

**Exemple.** Soit la formule  $\vdash A \vee (B \wedge C) \Rightarrow (A \vee B) \wedge C$ . Un arbre de dérivation partiel est :

$$\begin{array}{c} \vdots \\ \frac{A \vee (B \wedge C) \vdash A \vee B}{\wedge d} \quad \vee g \frac{\boxed{A \vdash C} \quad B \wedge C \vdash C}{A \vee (B \wedge C) \vdash C} \\ \hline \frac{A \vee (B \wedge C) \vdash A \vee B \quad A \vee (B \wedge C) \vdash C}{A \vee (B \wedge C) \vdash (A \vee B) \wedge C} \\ \hline \Rightarrow d \frac{A \vee (B \wedge C) \vdash (A \vee B) \wedge C}{\vdash A \vee (B \wedge C) \Rightarrow (A \vee B) \wedge C} \end{array}$$

On voit qu'une des feuilles  $A \vdash C$  n'est pas valide et donc on en déduit (sans finir la construction de l'arbre) que le séquent  $\vdash A \vee (B \wedge C) \Rightarrow (A \vee B) \wedge C$  n'est pas prouvable non plus.

**Exercice 4.2 (Preuve à l'aide du système  $G$ )** En utilisant des arbres de dérivation dans le système  $G$  pour les formules suivantes, dire si elles sont valides ou non :

1.  $\vdash ((p \wedge q) \Rightarrow r) \Rightarrow (\neg p \vee q) \Rightarrow (p \Rightarrow r)$
2.  $\vdash ((p \wedge q) \Rightarrow r) \wedge (\neg p \vee q) \Rightarrow p$

### 4.2.2 Résolution propositionnelle

**Rappels.** Une clause est une disjonction de littéraux et peut se représenter par un ensemble de littéraux. Nous noterons une clause  $l_1 \vee \dots \vee l_n$ , chaque littéral apparaîtra une seule fois et l'ordre est indifférent.

Le cas où cet ensemble est vide correspond à la proposition  $\perp$  qui est toujours fausse.

Si la clause contient la formule  $\top$  ou contient une variable propositionnelle et sa négation, elle est toujours vraie (clause triviale). Nous nous intéressons à la satisfiabilité d'un ensemble de clauses, et une clause toujours vraie est satisfiable dans toutes les interprétations. Une clause triviale peut donc être retirée de l'ensemble considéré sans changer le problème ( $\mathcal{E}$  est satisfiable si et seulement si  $\mathcal{E} \cup \{\top\}$  est satisfiable). Une clause non triviale peut donc toujours se représenter par un ensemble de littéraux qui sont soit des variables propositionnelles soit des négations de variables propositionnelles et la même variable n'apparaît pas deux fois.

**Règle de résolution.** On suppose donné un ensemble de clauses  $\mathcal{E}$ . On cherche à savoir si ces clauses sont satisfiables, c'est-à-dire s'il existe une interprétation  $I$  telle que  $I \models \mathcal{E}$ . La notation  $I \models \mathcal{E}$  représente le fait que pour toute formule  $P \in \mathcal{E}$ , on a  $I \models P$ .

Pour cela on va se donner un système de règles qui nous permet de dériver de nouvelles clauses  $C$  telle que si  $I \models \mathcal{E}$  alors on a aussi  $I \models C$ . Le but du jeu est d'aboutir à la clause vide qui représente  $\perp$  et qui est insatisfiable et donc d'en conclure que l'ensemble de clauses initial n'est pas satisfiable (il n'existe pas d'interprétation  $I$  telle que  $I \models \mathcal{E}$ ).

**Définition 4.2.1 (Dédution par résolution)** Soit  $C$  et  $C'$  deux clauses propositionnelles et  $p$  une variable propositionnelle, la règle de résolution a la forme suivante :

$$\frac{C \vee \neg p \quad C' \vee p}{C \vee C'}$$

La notation  $C \vee C'$  construit la clause qui contient tous les littéraux de  $C$  et  $C'$  en supprimant les doublons.

Soit  $\mathcal{E}$  un ensemble de clauses, une dédution par résolution à partir de  $\mathcal{E}$  est une suite de clauses  $C_1, \dots, C_n$  telle que pour toute clause  $C_i$  dans cette suite soit  $C_i \in \mathcal{E}$ , soit il existe deux clauses  $C_j$  et  $C_k$  telles que  $j, k < i$  et telle que  $C_i$  soit le résultat de la règle de résolution appliquée à  $C_j$  et  $C_k$ .

C'est-à-dire que  $\frac{C_j \quad C_k}{C_i}$  est un cas particulier de  $\frac{C \vee \neg p \quad C' \vee p}{C \vee C'}$

De manière équivalente, l'ensemble des clauses déductibles par résolution à partir d'un ensemble de clauses  $\mathcal{E}$  est le plus petit ensemble qui contient les clauses de  $\mathcal{E}$  et qui satisfait la condition associée à la règle de résolution vue comme une règle d'inférence.

**Exemple 4.1 (Dédution par résolution)** On se donne l'ensemble  $\mathcal{E}$  suivant  $\{\neg p \vee \neg q \vee r, \neg r, p\}$ . On peut faire les étapes de résolution suivantes (on a mis en indice de chaque règle la variable propositionnelle sur laquelle porte la résolution) :

$$\begin{array}{c} (p) \frac{\neg p \vee \neg q \vee r \quad p}{\neg q \vee r} \\ (r) \frac{\neg q \vee r \quad \neg r}{\neg q} \end{array}$$

On obtient la déduction par résolution suivante :

$$\neg p \vee \neg q \vee r, p, \neg q \vee r, \neg r, \neg q$$

#### Définition 4.2.2 (Preuve par résolution)



- une réfutation par résolution de  $\mathcal{E}$  est une déduction par résolution à partir de  $\mathcal{E}$  qui contient la clause vide  $\perp$ ;
- une preuve par résolution (on dit aussi une preuve par réfutation) d'une formule quelconque  $A$  à partir d'un ensemble de formules  $\mathcal{E}$  est une réfutation à partir de la forme clausale de l'ensemble de formules  $\mathcal{E} \cup \{\neg A\}$ .

**Exemple 4.2 (réfutation)** Soit l'ensemble  $\mathcal{E}$  suivant  $\{\neg p \vee \neg q \vee r, \neg r, p, q\}$ . On peut construire une réfutation :

$$\begin{array}{c}
 (p) \frac{\neg p \vee \neg q \vee r \quad p}{\neg q \vee r} \\
 (r) \frac{\neg q \vee r \quad \neg r}{\neg q} \\
 (q) \frac{\neg q \quad q}{\perp}
 \end{array}$$

**Exemple 4.3 (Preuve par résolution)** Soit  $A$  la formule définie comme  $\neg(p \vee (q \wedge r)) \Rightarrow (\neg q \vee \neg r)$ . Pour faire une preuve par résolution de  $A$ , on met la formule  $\neg A$  en forme clausale :

$$\neg(\neg(p \vee (q \wedge r)) \Rightarrow (\neg q \vee \neg r)) \equiv \neg(p \vee (q \wedge r)) \wedge (q \wedge r) \equiv \neg p \wedge (\neg q \vee \neg r) \wedge q \wedge r$$

On a donc quatre clauses :  $\{\neg p, (\neg q \vee \neg r), q, r\}$ . On construit une réfutation de la manière suivante :

$$\begin{array}{c}
 (r) \frac{\neg q \vee \neg r \quad r}{\neg q} \\
 (q) \frac{\neg q \quad q}{\perp}
 \end{array}$$

La formule  $\neg A$  est instatisfiable et donc la formule  $A$  est valide.

**Exercice 4.3** Faire une réfutation par résolution de l'ensemble des formules  $\{p \vee q, \neg p \vee \neg q, p \vee \neg q, \neg p \vee q\}$ .

**Proposition 4.2.3 (Correction de la résolution)** Soit  $\mathcal{E}$  un ensemble de clauses

- Pour toute clause  $C$  déductible par résolution à partir de  $\mathcal{E}$ , on a  $\mathcal{E} \models C$ , c'est-à-dire que toute interprétation qui rend vraies les formules de  $\mathcal{E}$ , rend vraie  $C$ .
- S'il existe une réfutation par résolution de  $\mathcal{E}$ , alors  $\mathcal{E}$  est insatisfiable.
- Soit une formule  $A$  et  $\mathcal{E}$  un ensemble de formules, s'il existe une preuve par résolution de  $A$  à partir de  $\mathcal{E}$  alors  $A$  est conséquence logique de  $\mathcal{E}$  ( $\mathcal{E} \models A$ ).

**Preuve:**

- Pour établir que  $\mathcal{E} \models C$ , on se donne une interprétation  $I$  telle que  $I \models \mathcal{E}$  et on établit que  $I \models C$  par minimalité de la définition de l'ensemble des clauses déductibles.
  - Si  $C \in \mathcal{E}$  alors c'est vrai.
  - Sinon,  $C$  est obtenue par résolution à partir de deux clauses  $C_1 \vee p$  et  $C_2 \vee \neg p$  et on a alors  $C = C_1 \vee C_2$ .  
La minimalité nous donne que  $I \models C_2 \vee \neg p$  et  $I \models C_1 \vee p$ . On raisonne par cas suivant la valeur de la variable  $p$  dans l'interprétation  $I$ . Si  $p$  est vrai alors  $\neg p$  est faux et donc  $I \models C_2$  et si  $p$  est faux alors  $I \models C_1$ . Dans les deux cas, on a bien  $I \models C_1 \vee C_2$ , c'est-à-dire  $I \models C$ .  
On en déduit que  $\mathcal{E} \models C$ .
- Supposons qu'il existe une réfutation par résolution de  $\mathcal{E}$ . Si  $\mathcal{E}$  était satisfiable, d'après le résultat précédent on aurait une interprétation dans laquelle  $\perp$  est vrai, ce qui n'est pas possible. On en déduit que  $\mathcal{E}$  est insatisfiable.
- Si on a une preuve par résolution de  $A$  à partir d'un ensemble de formule  $\mathcal{E}$  alors on sait que la forme clausale de  $\mathcal{E} \cup \{\neg A\}$  est insatisfiable donc  $\mathcal{E} \cup \{\neg A\}$  est insatisfiable et donc  $\mathcal{E} \models A$ .

□



## Complétude

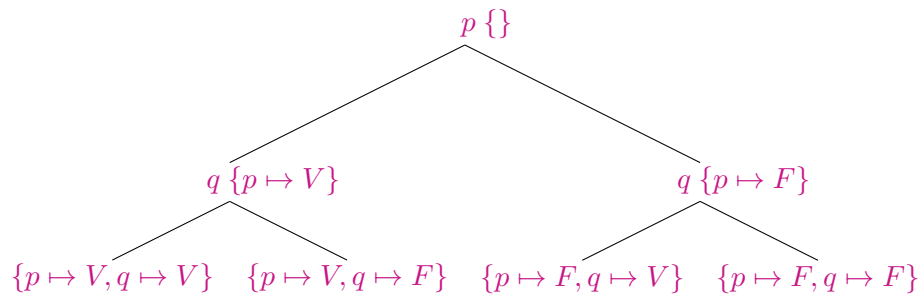
On peut se poser la question de savoir si la méthode de résolution peut toujours nous donner la solution. C'est-à-dire si on se donne un ensemble  $\mathcal{E}$  de clauses insatisfiable, est-ce qu'il existe une réfutation de  $\mathcal{E}$  par la méthode de résolution. La réponse est oui.

**Proposition 4.2.4 (Complétude de la résolution propositionnelle)** Soit  $\mathcal{E}$  un ensemble de clauses insatisfiable, alors il existe une réfutation par résolution de  $\mathcal{E}$ .

**Preuve:** On va faire la preuve dans le cas où d'un nombre fini ou dénombrable de variables propositionnelles  $(p_i)_{i \in \mathbb{N}}$ .

On construit un **arbre binaire de décision**. Chaque niveau est étiqueté par une variable propositionnelle, en partant de  $p_0$ , le sous-arbre gauche correspond au cas où cette variable vaut  $V$  et le sous-arbre droit, au cas où elle vaut  $F$ . Chaque branche (possiblement infinie) correspond donc à une interprétation logique des variables propositionnelles  $(p_i)_{i \in \mathbb{N}}$ .

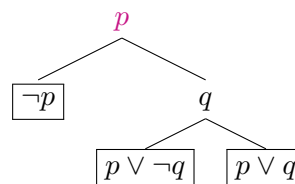
Si on a juste deux variables  $p$  et  $q$ , l'arbre de décision se présentera ainsi (on a indiqué pour chaque sommet l'interprétation correspondante)



Chaque nœud interne de l'arbre correspond à une interprétation partielle des variables. Au niveau de la racine, aucune variable n'a de valeur.

Si l'ensemble  $\mathcal{E}$  est insatisfiable alors pour chaque interprétation  $I$ , il existe une clause  $C \in \mathcal{E}$  telle que  $I \not\models C$ . Comme  $C$  n'a qu'un nombre fini de variables, on peut positionner la formule dans l'arbre le plus haut possible telle que l'interprétation  $I$  restreinte à  $p_0, \dots, p_n$  rend faux la clause  $C$ . Il est possible de faire cela pour toutes les branches de l'arbre. On va appeler un tel arbre un **arbre de réfutation de  $\mathcal{E}$** .

Par exemple un arbre de réfutation pour l'ensemble de clauses  $\{p \vee q, p \vee \neg q, \neg p\}$  est :



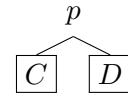
Un arbre à branchement fini dont toutes les branches sont finies est lui-même fini (lemme de König). Donc on a un arbre fini, et chaque feuille contient une clause de  $\mathcal{E}$  qui est fausse dans l'interprétation correspondant à la branche. Soit un arbre de réfutation pour un ensemble  $\mathcal{E}$  de clauses insatisfiable. Il se peut que cet arbre soit réduit à une feuille, celle-ci doit correspondre à une formule qui est fausse dans l'interprétation vide, il ne peut s'agir que de  $\perp$  et donc  $\perp \in \mathcal{E}$  et on a une réfutation triviale par résolution.

Si l'arbre n'est pas vide, on a un nœud de la forme  $p$  avec  $p$  une variable propositionnelle et



$C$  et  $D$  des clauses. Soit  $I$  l'interprétation des variables autres que  $p$  qui correspond au chemin de la racine de l'arbre jusqu'à  $p$ . Attention  $I$  ne définit pas la valeur de toutes les variables propositionnelles, c'est seulement une interprétation partielle et donc toutes les clauses ne peuvent pas être évaluées dans cette interprétation.

On a  $I, \{p \mapsto V\} \not\models C$  et  $I, \{p \mapsto F\} \not\models D$  par définition des arbres de réfutation. La variable propositionnelle  $p$  apparaît dans  $C$  et dans  $D$  (sinon on aurait pu faire remonter l'une de ces clauses plus haut dans l'arbre). On a forcément  $C = C' \vee \neg p$  et  $D = D' \vee p$  (sinon les clauses seraient vraies à l'endroit où elles sont positionnées). Ce qui signifie que l'on peut faire une étape de résolution entre les deux clauses, qui nous donnera une clause  $C' \vee D'$  telle que  $I \not\models C' \vee D'$ . On peut donc remplacer le sous arbre



clause  $C' \vee D'$ .

On voit donc qu'au fur et à mesure que l'on construit la déduction par résolution, on a un ensemble de clauses pour lequel l'arbre de réfutation diminue. Au final on aura un arbre réduit à une feuille  $\perp$  ce qui correspond à une réfutation.  $\square$

Evidemment cette démonstration n'est pas une manière effective de guider la recherche d'une réfutation par la méthode de résolution puisque si on avait l'arbre de réfutation on pourrait conclure immédiatement que l'ensemble de clauses est insatisfiable sans avoir recours à la résolution. Il faut parfois une peu d'intuition pour aboutir à la clause vide. Suivant la forme des clauses on peut avoir des stratégies plus précises.

Il se peut qu'en faisant une étape de résolution on trouve la clause triviale  $\top$  ou que l'on retombe sur une clause déjà connue. Cela ne signifie pas forcément que l'on s'est trompé ou que l'ensemble de clauses est satisfiable mais juste que les étapes de calcul faites étaient inutiles. Il faut alors chercher d'autres combinaisons ou s'interroger si l'ensemble de clauses de départ est bien insatisfiable.

### 4.2.3 Satisfiabilité

Il existe des méthodes qui, au lieu de chercher une contradiction dans un ensemble de clauses, vont au contraire chercher à construire une interprétation qui satisfait l'ensemble des clauses en collectant les contraintes sur les variables induites par ces clauses.

La procédure DPLL (du nom de ses inventeurs Davis, Putnam, Logemann et Loveland) permet de dire si une formule en forme normale conjonctive (représentée par un ensemble de clauses) est satisfiable. Pour cela, l'algorithme cherche à construire une interprétation qui rend la formule vraie.

L'algorithme DPLL travaille sur des problèmes de la forme :  $I \gg \Delta$  avec  $I$  une interprétation partielle des variables propositionnelles de la forme  $\{x_1 \mapsto b_1; \dots; x_n \mapsto b_n\}$  (que l'on peut aussi représenter comme un ensemble de littéraux  $x_i$  si  $b_i = V$  et  $\neg x_i$  si  $b_i = F$ ) et  $\Delta$  un ensemble de clauses  $C_1, \dots, C_n$ . On note  $\Delta, C$  le problème qui contient toutes les clauses de  $\Delta$  plus la clause  $C$ . Le but de l'algorithme est de construire une interprétation qui étend  $I$ , c'est-à-dire de la forme  $\{x_1 \mapsto b_1; \dots; x_n \mapsto b_n; x_{n+1} \mapsto b_{n+1} \dots x_p \mapsto b_p\}$  et qui rend vraies toutes les clause  $C_i$ .

Les règles sont les suivantes :

Il y a deux cas triviaux. Le premier cas s'il n'y a plus de clauses, l'interprétation  $I$  répond au problème. Le second cas si l'une des clauses  $C_i$  est la clause vide alors le problème n'a pas de solution. Cela correspond aux deux règles :

$$\text{SUCCESS} \frac{}{I \gg \emptyset} \quad \text{CONFLICT} \frac{}{I \gg \Delta, \perp}$$

Par ailleurs on peut simplifier les problèmes : lorsque l'interprétation  $I$  fixe la valeur de la variable  $x$ , alors les clauses  $C$  qui contiennent un littéral  $l$  de la forme  $x$  ou  $\neg x$  se simplifient :

- si  $I \models l$  alors la clause est trivialement vraie dans cette interprétation et peut être supprimée ;
- si  $I \not\models l$  alors on peut retirer le littéral  $l$  de la clause.

Ceci est représenté par les deux règles suivantes :

$$\text{BCP}_V \frac{I \gg \Delta \quad I \models l}{I \gg \Delta, (l \vee C)} \quad \text{BCP}_F \frac{I \gg \Delta, C \quad I \not\models l}{I \gg \Delta, (l \vee C)}$$

Maintenant si aucune des règles précédentes ne s'applique, alors il faut préciser l'interprétation en choisissant une valeur pour une nouvelle variable. Un cas simple est si une clause ne contient qu'un seul littéral alors pour

que la clause soit vraie il faut choisir la valeur de la variable pour rendre ce littéral vrai. Ceci s'exprime par les règles suivantes :

$$\text{ASSUME}_V \frac{I + \{x \mapsto V\} \gg \Delta \quad x \notin I}{I \gg \Delta, x} \quad \text{ASSUME}_F \frac{I + \{x \mapsto F\} \gg \Delta \quad x \notin I}{I \gg \Delta, \neg x}$$

Si maintenant, il n'y a aucune clause réduite à un littéral, alors il n'y a pas de choix évident. On va donc choisir une variable qui n'a pas encore de valeur et explorer les deux branches : celle où cette variable est interprétée par  $V$  et celle où elle est interprétée par  $F$ .

$$\text{UNSAT} \frac{I + \{x \mapsto V\} \gg \Delta \quad I + \{x \mapsto F\} \gg \Delta \quad x \notin I}{I \gg \Delta}$$

On part d'un ensemble de clauses  $\Delta$  et d'une interprétation vide. On construit un arbre en utilisant les règles précédentes et on cherche à arriver à une feuille de succès de la forme  $I \gg \emptyset$ . L'interprétation  $I$  est une solution de notre problème, elle rend vraies l'ensemble des clauses  $\Delta$ .

Si toutes les feuilles de l'arbre correspondent à des conflits alors l'ensemble de clauses initial n'est pas satisfiable.

**Exercice 4.4** 1. Appliquer l'algorithme précédent aux formules

(a)  $(\neg p \vee \neg q \vee \neg r) \wedge (\neg p \vee q \vee r) \wedge (p \vee q \vee \neg r)$

(b)  $(p \vee r) \wedge (\neg q \vee p) \wedge \neg p \wedge \neg r$

2. Justifier que la construction de l'arbre par les règles DPLL est toujours finie (trouver une quantité qui diminue le long des branches de l'arbre).
3. Montrer que les règles de DPLL sont correctes. C'est-à-dire que si un arbre DPLL de racine  $I \gg \Delta$  contient une feuille de succès  $J \gg \emptyset$  alors l'interprétation  $J$  étend l'interprétation  $I$  (c'est-à-dire que  $I(x) = J(x)$  pour toutes les variables  $x$  qui apparaissent dans  $I$ ) et l'interprétation  $J$  rend vraies toutes les clauses de  $\Delta$ .
4. En déduire une méthode pour trouver un modèle d'une formule.
5. A votre avis, la méthode est-elle complète, c'est-à-dire s'il existe un modèle est-on sûr de le trouver ? Peut-on trouver avec cette méthode tous les modèles d'une formule ?

**Compétences 4.2** — Savoir faire une preuve d'une formule propositionnelle par la construction d'un arbre dans le système  $G$  ou l'utilisation de la résolution.

## 4.3 Unification

On a vu que la question de la satisfiabilité nécessitait de produire des instances de formules, c'est-à-dire de remplacer des variables par des termes ayant de bonnes propriétés. Dans cette section, nous allons introduire un outil, l'**unification**, qui nous sera utile dans la méthode de résolution pour produire des instances particulières de formules. L'unification traite une classe de problèmes très simples à savoir des équations entre termes avec variables. C'est un problème qui a des applications multiples comme la question de l'inférence de types dans les langages fonctionnels.

L'unification est également l'ingrédient majeur des *systèmes de réécritures* qui sont des théories logiques dans lesquelles le seul symbole de prédicat est celui de l'égalité et les axiomes en plus de ceux de l'égalité sont de la  $\forall(t = u)$  de tels systèmes sont à la fois utilisés pour le raisonnement et comme langage de programmation.

### 4.3.1 Définitions

On se donne une signature  $\mathcal{F}$  pour les termes et un ensemble  $\mathcal{X}$  de variables. On rappelle qu'une substitution est une application  $\sigma \in \mathcal{X} \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{X})$ . On s'intéresse à des substitutions dites à *support fini* c'est-à-dire telles que  $\sigma(x) = x$  sauf pour un nombre fini de variables. De telles substitutions se représentent facilement comme des ensembles finis  $\{x \leftarrow \sigma(x) \mid x \in \mathcal{X} \wedge \sigma(x) \neq x\}$ . Ainsi, la substitution identité telle que  $\sigma(x) = x$  pour tout  $x \in \mathcal{X}$  se représentera simplement par un ensemble vide  $\{\}$ .

Étant donnée une substitution, on a défini dans la section 1.5.3 une application de  $\mathcal{T}(\mathcal{F}, \mathcal{X}) \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{X})$  qui à un terme  $t$  associe le terme noté  $t[\sigma]$  dans lequel on a remplacé chaque variable  $x$  par  $\sigma(x)$ .

**Définition 4.3.1 (Composition de substitutions)** On peut *composer* deux substitutions  $\sigma_1$  et  $\sigma_2$  en une nouvelle substitution notée  $\sigma_1\sigma_2$  définie par  $(\sigma_1\sigma_2)(x) = \sigma_1(x)[\sigma_2]$ . Pour tout terme  $t$  : on a  $t[\sigma_1\sigma_2] = (t[\sigma_1])[\sigma_2]$ .

⚠ D'un point de vue fonctionnel, la notation  $\sigma_1\sigma_2$  correspond à appliquer la substitution  $\sigma_1$  suivie de  $\sigma_2$ , ce qui d'un point de vue mathématiques correspond à la composition de fonctions  $\sigma_2 \circ \sigma_1$ .

**Remarque.** Il ne faut pas confondre la composition de deux substitutions qui correspond à deux substitutions *successives* avec la substitution *simultanée*. Ainsi si on pose  $\sigma_1 = \{x \leftarrow y\}$  et  $\sigma_2 = \{y \leftarrow z\}$ . On a  $\sigma_1\sigma_2 = \{x \leftarrow z; y \leftarrow z\}$  et  $(x + y)[\sigma_1\sigma_2] = z + z$  alors que la substitution simultanée est définie par  $\{x \leftarrow y; y \leftarrow z\}$  et donne le résultat  $(x + y)[x \leftarrow y; y \leftarrow z] = y + z$ .

**Définition 4.3.2 (Problème d'unification)** Un problème d'unification est donné par deux termes avec variables  $t$  et  $u$ . Ce problème admet une solution s'il existe une substitution  $\sigma$  telle que  $t[\sigma] = u[\sigma]$ . On dira que  $\sigma$  est solution du problème  $t \stackrel{?}{=} u$ , on dira aussi que  $\sigma$  est un *unificateur* des termes  $t$  et  $u$  ou encore qu'il *unifie* les termes  $t$  et  $u$ .

De manière plus générale, on peut se donner un ensemble de couples de termes  $\{(t_i, u_i) \mid i \in I\}$ . Résoudre ce problème d'unification revient à chercher une substitution  $\sigma$  telle que  $t_i[\sigma] = u_i[\sigma]$  pour tout  $i \in I$ .

Il s'agit de la résolution d'une équation (ou d'un système d'équations), on trouve des valeurs pour les variables qui apparaissent dans  $t$  et  $u$ , telles que si on remplace les variables par ces valeurs, alors les deux termes obtenus sont *syntactiquement* égaux. On rappelle que les termes  $f(t_1, \dots, t_n)$  et  $g(u_1, \dots, u_p)$  sont (syntactiquement) égaux uniquement lorsque les symboles  $f$  et  $g$  sont égaux, on a alors la même arité donc  $p = n$  et on doit avoir récursivement  $t_i = u_i$  pour  $i = 1 \dots n$ .

**Exemple 4.4** Si on se donne deux expressions sur la signature pour les nombres  $plus(x, 0)$  et  $plus(y, z)$  une solution est  $\{x \leftarrow 0; y \leftarrow 0; z \leftarrow 0\}$ . Une autre solution est  $\{x \leftarrow y; z \leftarrow 0\}$ . Certaines équations n'ont pas de solution, par exemple  $plus(x, 0)$  et  $0$  car on s'intéresse à une égalité *syntactique* entre les termes et les symboles  $plus$  et  $0$  sont distincts.

**Exercice 4.5** La signature sur les arbres est donnée avec la constante  $\epsilon$  et la fonction binaire  $N$ . Les problèmes d'unification suivants ont-ils des solutions ? si oui lesquelles ?

1.  $N(x, \epsilon) \stackrel{?}{=} N(\epsilon, y)$
2.  $N(N(x, y), \epsilon) \stackrel{?}{=} N(\epsilon, y)$
3.  $N(x, \epsilon) \stackrel{?}{=} x$

### 4.3.2 Ensemble des solutions

Si  $\sigma_1$  est une solution du problème d'unification  $t \stackrel{?}{=} u$  alors il en est de même de  $\sigma_1\sigma_2$  pour toute substitution  $\sigma_2$ . En effet on a  $t[\sigma_1] = u[\sigma_1]$  car  $\sigma_1$  est solution du problème et donc  $t[\sigma_1\sigma_2] = t[\sigma_1][\sigma_2] = u[\sigma_1][\sigma_2] = u[\sigma_1\sigma_2]$ .

Si on a deux solutions  $\sigma$  et  $\tau$  a notre problème d'unification, on dira que  $\sigma$  est plus générale que  $\tau$  s'il existe une substitution  $\sigma'$  telle que  $\sigma\sigma' = \tau$ .

**Exemple 4.5** Si on reprend l'exemple 4.4, on a une solution  $\sigma \stackrel{\text{def}}{=} \{x \leftarrow y; z \leftarrow 0\}$  et une solution  $\tau \stackrel{\text{def}}{=} \{x \leftarrow 0; y \leftarrow 0; z \leftarrow 0\}$ . On a  $\tau = \sigma\{y \leftarrow 0\}$  donc  $\sigma$  est plus général.

Une bonne propriété de l'unification sur les termes syntaxiques est que si un problème a une solution alors il existe une solution qui est plus générale que toutes les autres. Et donc pour calculer l'ensemble de toutes les solutions, il suffit de calculer cette substitution que l'on appelle aussi unificateur principal.

**Proposition 4.3.1 (Unification, algorithme de Robinson)** Soit  $E = \{t_i \stackrel{?}{=} u_i | i \in I\}$  un problème d'unification. Si ce problème admet une solution, alors il existe un unificateur principal  $\sigma$  et celui-ci est calculé par la fonction suivante :

1 :	$\text{unif}(\emptyset) = \{\}$	identité
2 :	$\text{unif}(\{x \stackrel{?}{=} x\} \cup E) = \text{unif}(E)$	
3 :	$\text{unif}(\{x \stackrel{?}{=} t\} \cup E) = \text{echec}$	$x \in \text{vars}(t), t \neq x$
4 :	$\text{unif}(\{x \stackrel{?}{=} t\} \cup E) = \{x \leftarrow t\} \text{unif}(E[x \leftarrow t])$	$x \notin \text{vars}(t)$
5 :	$\text{unif}(\{t \stackrel{?}{=} x\} \cup E) = \text{unif}(\{x \stackrel{?}{=} t\} \cup E)$	$t \notin \mathcal{X}$
6 :	$\text{unif}(\{f(t_1, \dots, t_n) \stackrel{?}{=} g(u_1, \dots, u_p)\} \cup E) = \text{echec}$	$f \neq g$
7 :	$\text{unif}(\{f(t_1, \dots, t_n) \stackrel{?}{=} f(u_1, \dots, u_n)\} \cup E) = \text{unif}(\{t_1 \stackrel{?}{=} u_1; \dots; t_n \stackrel{?}{=} u_n\} \cup E)$	

**Preuve:** On vérifie que chaque cas possible pour l'ensemble  $E$  est couvert par une équation unique : l'ensemble est soit vide soit il contient une équation et cette équation a son terme gauche qui est ou non une variable et si le terme gauche n'est pas une variable on regarde le cas où le terme droit est ou non une variable.

Un problème plus délicat à justifier est la terminaison de cette définition vue comme un procédé de calcul. Dans les cas récurrents, on regarde ce qui diminue :

- dans le cas (2) le nombre de problèmes diminue mais par ailleurs il augmente dans le cas (7);
- dans le cas (7) ce qui diminue c'est la taille globale du problème (nombre de symboles) car on a fait disparaître deux occurrences de  $f$ , mais la taille du problème augmente par ailleurs dans le cas (4) lorsque l'on remplace la variable  $x$  par  $t$ ;
- dans le cas (4), la taille du problème grossit mais on a fait disparaître une variable ( $x$ ).

En résumé pour un problème  $E$  si on regarde le triplet  $\text{nbvars}(E), \text{taille}(E), \text{nbeqs}(E)$  avec  $\text{nbvars}(E)$  le nombre de variables différentes dans  $E$ ,  $\text{taille}(E)$  le nombre d'occurrences de symboles dans  $E$ , et  $\text{nbeqs}(E)$  le nombre d'équations dans  $E$ , alors on remarque que chaque appel se fait sur un problème strictement plus petit si on prend l'ordre lexicographique, sauf celui où on change l'équation  $t = x$  en  $x = t$ . Mais cette règle n'est appliquée que lorsque  $t$  n'est pas une variable et sera donc immédiatement suivie de l'application d'une règle qui fait diminuer le "poids" du problème. Une autre possibilité est de définir  $\text{taille}(t \stackrel{?}{=} u) \stackrel{\text{def}}{=} 2 \times \text{nbsymb}(t) + \text{nbsymb}(u)$  au lieu de  $\text{taille}(t \stackrel{?}{=} u) \stackrel{\text{def}}{=} \text{nbsymb}(t) + \text{nbsymb}(u)$ .

On a alors  $\text{taille}(x \stackrel{?}{=} t) < \text{taille}(t \stackrel{?}{=} x)$  lorsque  $t$  n'est pas une variable.

On vérifie que les cas d'échec correspondent à des problèmes sans solution et que dans les cas où une solution est trouvée, il s'agit bien de l'unificateur principal.  $\square$

**Exercice 4.6** Soit  $g$  une fonction binaire,  $f$  une fonction unaire et  $a$  une constante. Résoudre les problèmes d'unification suivants :

1.  $g(x, f(y)) \stackrel{?}{=} g(f(y), z)$
2.  $g(g(x, f(y)), z) \stackrel{?}{=} g(g(y, z), a)$

L'algorithme de Robinson est naturel mais n'est pas efficace à cause en particulier de l'opération de substitution lorsqu'on élimine l'équation  $x \stackrel{?}{=} t$ . On peut utiliser une représentation des termes qui rend cette opération moins coûteuse. Il existe d'autres algorithmes plus complexes mais plus efficaces.

**Autres applications de l'unification.** L'unification est une opération utilisée dans de nombreuses applications, par exemple dans l'algorithme de typage des programmes Ocaml afin de trouver le type le plus général que peut traiter une fonction. L'unification se généralise également à des théories plus complexes dans lesquelles l'égalité entre termes n'est plus syntaxique mais prend en compte des propriétés des opérations comme la commutativité ou l'associativité. Suivant les théories, on va perdre la décidabilité ou l'existence d'un unificateur principal.

**Unification de littéraux.** L'unification sur les termes se généralise en une unification entre des littéraux. Si on a deux littéraux  $P$  et  $Q$ , ils s'unifient s'il existe une substitution  $\sigma$  telle que  $P[\sigma] = Q[\sigma]$ . Si  $P$  est de la forme  $R(t_1, \dots, t_n)$  (resp.  $\neg R(t_1, \dots, t_n)$ ) alors  $\sigma$  est un unificateur de  $P$  et  $Q$  si et seulement si  $Q$  est de la forme  $R(u_1, \dots, u_n)$  (resp.  $\neg R(u_1, \dots, u_n)$ ) et  $\sigma$  est un unificateur de l'ensemble de problèmes  $\{t_i \stackrel{?}{=} u_i \mid i = 1 \dots n\}$ .

**Exercice 4.7** Résoudre les problèmes d'unification suivants entre littéraux :

1.  $(plus(x, 0) \leq x) \stackrel{?}{=} (plus(0, y) \leq y)$
2.  $(plus(x, 0) \leq x) \stackrel{?}{=} \neg(y \leq z)$
3.  $(plus(x, 0) \leq x) \stackrel{?}{=} (y \leq plus(y, 0))$

**Exercice 4.8** On se donne  $n$  termes  $t_1, \dots, t_n$  et on cherche à unifier ces  $n$  termes, c'est-à-dire trouver une substitution  $\sigma$  telle que  $t_1[\sigma] = \dots = t_n[\sigma]$ .

On dispose d'un algorithme `unif` qui donne l'unificateur principal de deux termes, comment peut-on procéder ?

### 4.3.3 Ordre sur les termes et filtrage

**Définition 4.3.3 (Instance)** On définit un ordre sur les termes avec variables

$$t \leq u \stackrel{\text{def}}{=} \text{il existe } \sigma \text{ tel que } u = t[\sigma]$$

Si  $u = t[\sigma]$ , on dit que  $t$  est *plus général* que  $u$  et que  $u$  est une *instance* de  $t$ .

Un terme avec variables  $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$  représente un *ensemble* de termes clos (sans variables), les instances closes de  $t$  :

$$\text{inst}(t) \stackrel{\text{def}}{=} \{u \in \mathcal{T}(\mathcal{F}) \mid \text{il existe } \sigma \text{ tel que } u = t[\sigma]\}$$

**Proposition 4.3.2** Si  $t \leq u$  alors  $\text{inst}(u) \subseteq \text{inst}(t)$ .

**Preuve:** Si  $t \leq u$  alors  $u = t[\sigma]$  et toute instance close de  $u$  est de la forme  $u[\tau] = t[\sigma][\tau] = t[\sigma\tau]$  et est donc une instance close de  $t$ .  $\square$



**Définition 4.3.4 (Filtrage, motif)**

- Le problème du *filtrage* consiste à se donner deux termes  $t$  et  $u$  et à rechercher  $\sigma$  tel que  $t[\sigma] = u$ .
- Un *motif* est un terme dans lequel il n'y a pas deux fois la même variable.

Le langage Ocaml permet de définir une expression par cas suivant un ensemble de motifs

```
match e with 1 -> "un" | 2 -> "deux" | x -> "beaucoup"
type form = Var of int | Bot | Neg of form
           | Bin of form * form
let rec print_form e = match e with
  Var n -> string_of_int n | Bot -> "False"
  | Neg f -> "~"^(print_form f)
  | Bin(f,g) -> (print_form f)^^"/\\"^(print_form g)
```

A l'exécution,  $e$  a pour valeur un terme clos ( $v$ )

- la première branche dont le motif  $p$  correspond (ie.  $p[\sigma] = v$  ou encore  $v \in \text{inst}(p)$ ) est exécutée
- on donne aux variables du motif les valeurs de la substitution  $\sigma$

On peut analyser les motifs

- Est-il possible de changer l'ordre des motifs, de les exécuter en parallèle ?
- Tous les cas sont-ils couverts ?
- Certains cas sont-ils inutiles ?

**Exercice 4.9** On se donne une définition avec des motifs  $p_1, \dots, p_n$ , traduire les propriétés précédentes en conditions sur les ensembles  $\text{inst}(p_1), \dots, \text{inst}(p_n)$

**Compétences 4.3** — Connaître la définition de l'unification (syntaxique).

- Savoir utiliser l'algorithme de Robinson pour calculer l'unificateur principal.

## 4.4 Résolution générale

On s'intéresse ici au problème de savoir si un ensemble de formules de la logique du premier ordre est satisfiable ou non, c'est-à-dire s'il existe une interprétation qui rend vraies simultanément toutes les formules. On rappelle qu'un ensemble fini de formules est satisfiable si et seulement si la conjonction de toutes les formules est elle-même une formule satisfiable. On a vu au chapitre 3 que l'on pouvait se ramener au cas où les formules manipulées sont uniquement des clauses (disjonction de littéraux), universellement quantifiées dans le cas de la logique du premier ordre.

La technique de preuve par résolution est importante car elle est à la base non seulement de systèmes de démonstration automatique mais aussi de systèmes de programmation logique. Le plus connu est le langage de programmation PROLOG, dans lequel le programme est modélisé comme un ensemble de clauses et un calcul se fait par la recherche d'une preuve.

Nous avons traité le cas de la résolution en logique propositionnelle, nous voyons ici comment la méthode se généralise en logique du premier ordre.

### 4.4.1 Résolution au premier ordre

En logique propositionnelle, une clause est un ensemble de littéraux (variable propositionnelle ou négation de variable propositionnelle) et on se ramène au cas où chaque variable propositionnelle apparaît une seule fois.

Dans le cas du premier ordre, les littéraux sont de la forme  $R(t_1, \dots, t_n)$  ou  $\neg R(t_1, \dots, t_n)$  avec  $R$  un symbole de prédicat et  $t_i$  des termes. Le même symboles peut donc apparaître plusieurs fois dans la clause de manière positive ou négative associée à des termes différents.



Par ailleurs les termes qui apparaissent dans les clauses peuvent contenir des variables. De fait une clause  $C$  qui contient les variables  $\{x_1, \dots, x_n\}$  est juste une représentation sans quantificateur de la formule  $\forall(C)$  (notation pour  $\forall x_1 \dots x_n, C$ ).

Une autre interprétation de la clause avec variables  $C$  est l'ensemble des instances closes (sans variable) de la formule c'est-à-dire l'ensemble des clauses  $C[\sigma_0]$  avec  $\sigma_0$  une substitution qui remplace chaque variable de  $C$  par un terme clos. Le théorème de Herbrand nous a permis d'établir que la satisfiabilité de la formule  $\forall(C)$  était équivalente à la satisfiabilité de l'ensemble (potentiellement infini) de toutes les instances closes de  $C$ . C'est ce qui est exploité dans la résolution au premier ordre.

On part d'un ensemble de clauses (disjonction de littéraux vue comme un ensemble de littéraux). On rappelle que l'unification sur les termes s'étend de manière naturelle à l'unification sur les littéraux. Deux littéraux  $L_1$  et  $L_2$  s'unifient s'ils sont de la forme  $R(t_1, \dots, t_n)$  et  $R(u_1, \dots, u_n)$  ou bien  $\neg R(t_1, \dots, t_n)$  et  $\neg R(u_1, \dots, u_n)$  et que le problème  $(t_i \stackrel{?}{=} u_i)_{i=1 \dots n}$  admet une solution. Dans ce cas on note  $\text{unif}(L_1, L_2)$  l'unificateur principal.

### Règles pour la résolution

**Définition 4.4.1** La méthode de résolution en calcul des prédicats utilise trois règles qui manipulent des clauses :

1. *Factorisation* : si  $\sigma$  est l'unificateur principal de  $L_1$  et  $L_2$ , la règle de factorisation s'écrit :

$$\frac{L_1 \vee L_2 \vee C}{L_1[\sigma] \vee C[\sigma]}$$

2. *Renommage* : si  $\sigma$  est une bijection sur les variables, la règle de renommage s'écrit :

$$\frac{C}{C[\sigma]}$$

3. *Résolution binaire* : si les clauses  $L_1 \vee C$  et  $\neg L_2 \vee C'$  n'ont pas de variables communes et si  $\sigma$  est l'unificateur principal de  $L_1$  et  $L_2$ , la règle de résolution s'écrit :

$$\frac{L_1 \vee C \quad \neg L_2 \vee C'}{C[\sigma] \vee C'[\sigma]}$$

L'ensemble des clauses que l'on peut déduire par résolution à partir d'un ensemble de clauses  $\mathcal{E}$  est le plus petit ensemble défini par les règles précédentes et qui contient  $\mathcal{E}$  (et donc vérifie la règle  $\frac{C \in \mathcal{E}}{C}$ ).

Les définitions de réfutation par résolution et de preuve par résolution sont les mêmes que dans le cas propositionnel (définition 4.2.2) mais avec la notion de déduction par résolution générale.

**Exemple 4.6** Soit l'ensemble de clauses  $P(x), \neg P(y) \vee Q(y), \neg Q(a)$  avec  $P$  et  $Q$  deux symboles de prédicat unaires et  $a$  une constante.

On construit la déduction par résolution (on indique pour chaque règle la substitution qui est utilisée)

$$\begin{array}{c} [y \leftarrow x] \frac{\neg P(y) \vee Q(y) \quad P(x)}{Q(x)} \quad \neg Q(a) \\ [x \leftarrow a] \frac{\quad}{\perp} \end{array}$$

La règle de renommage sert juste à préparer les clauses pour appliquer la règle de résolution dont une des conditions est que les deux clauses ne partagent pas de variable.

La règle de factorisation permet de faire disparaître plusieurs littéraux lors d'une étape de résolution binaire. Elle est indispensable comme l'illustre l'exemple ci-dessous.

Ces deux règles sont triviales dans le cas propositionnel. La règle de résolution générale généralise la règle de résolution propositionnelle.

**Exemple 4.7 (Règle de factorisation)** On souhaite montrer que l'ensemble  $\{P(x, y) \vee P(y, x), \neg P(u, z) \vee \neg P(z, u)\}$  est insatisfiable. Si on applique la règle de résolution avec les formules  $P(x, y) \vee P(y, x)$  et  $\neg P(u, z) \vee \neg P(z, u)$  alors on obtient

$$[x \leftarrow u, y \leftarrow z] \frac{P(x, y) \vee P(y, x) \quad \neg P(u, z) \vee \neg P(z, u)}{P(z, u) \vee \neg P(z, u)}$$

qui est la clause triviale et donc n'aboutit pas à une contradiction.

Par contre, on peut commencer par effectuer des factorisations sur les deux clauses dont on déduit facilement une contradiction

$$\begin{array}{c} [x \leftarrow y] \frac{P(x, y) \vee P(y, x)}{P(y, y)} \quad [z \leftarrow u] \frac{\neg P(u, z) \vee \neg P(z, u)}{\neg P(u, u)} \\ [y \leftarrow u] \frac{\quad}{\perp} \end{array}$$

**Exemple 4.8 (Preuve par résolution)** Soit  $A \stackrel{\text{def}}{=} \exists x, \forall y, (S(y) \Rightarrow R(x)) \Rightarrow (S(x) \Rightarrow R(y))$ . Pour prouver  $A$ , on fait une réfutation de la formule  $\neg A$ .

On commence par mettre  $\neg A$  en forme clausale.

- La forme normale de négation de  $\neg A$  est  $\forall x, \exists y, (\neg S(y) \vee R(x)) \wedge S(x) \wedge \neg R(y)$
- On skolemise pour éliminer le quantificateur existentiel. Cette opération introduit un nouveau symbole de fonction que l'on notera  $f$  et qui est unaire (dépendance de la propriété par rapport à  $x$ ). On obtient la formule  $\forall x, (\neg S(f(x)) \vee R(x)) \wedge S(x) \wedge \neg R(f(x))$
- On découpe alors en clauses, il y en a trois  $\{(\neg S(f(x)) \vee R(x)), S(x), \neg R(f(x))\}$

On procède ensuite aux étapes de résolution en vue de produire la clause vide :

$$\begin{array}{c} [y \leftarrow f(x)] \frac{\neg S(f(x)) \vee R(x) \quad S(y)}{R(x)} \quad \neg R(f(z)) \\ [x \leftarrow f(z)] \frac{\quad}{\perp} \end{array}$$

On conclut :  $\neg A$  est insatisfiable donc  $A$  est valide.

**Exercice 4.10** En utilisant la résolution, montrer que

$$(\exists x, \neg P(x)), (\forall x, (P(x) \vee Q(x))) \models \exists x, Q(x)$$

### Propriétés de la résolution

La correction de la méthode de résolution établit que si l'on peut déduire une clause  $C$  à partir d'un ensemble de clauses  $\mathcal{E}$  alors la formule  $\forall(C)$  est conséquence logique de l'ensemble des formules  $\{\forall(D) \mid D \in \mathcal{E}\}$ .

On commence par établir quelques propriétés simples qui relient entre elles les clauses généralisées et leurs instances. Soient  $A$  et  $B$  deux formules,  $x$  une variable et  $t$  un terme.

- $(\forall x, A) \models A[x \leftarrow t]$
- Si  $A \models B$  alors  $(\forall x, A) \models B$
- Si  $A \models B$  et  $x \notin \text{VI}(A)$  alors  $A \models \forall x, B$

Ces résultats se généralisent au cas de plusieurs formules et plusieurs variables. Si  $\sigma$  est une substitution alors  $\forall(C) \models C[\sigma]$  qui se déduit du lien entre substitution et vérité :

$$\text{val}(\rho, C[\sigma]) = \text{val}(\{x \mapsto \text{val}(\rho, \sigma(x)) \mid x \in \text{VI}(C)\}, C)$$

Par ailleurs si  $\mathcal{E}$  est un ensemble de formules, si  $\mathcal{E} \models B$  et si les variables libres de  $B$  ne sont pas libres dans les formules de  $\mathcal{E}$ , alors  $\mathcal{E} \models \forall(B)$ .

**Proposition 4.4.1 (Correction de la résolution)** Soit  $\mathcal{E}$  un ensemble de clauses, chacune des étapes possibles de déduction par résolution produit une clause  $C$  telle que  $\{\forall(D) \mid D \in \mathcal{E}\} \models \forall(C)$ .

En particulier s'il existe une réfutation de  $\mathcal{E}$ , alors  $\{\forall(D) \mid D \in \mathcal{E}\} \models \perp$  et donc  $\{\forall(D) \mid D \in \mathcal{E}\}$  est insatisfiable.

**Preuve:** On va montrer la correction de chacune des règles en montrant que la généralisation de la conclusion (clause sous la barre) est une conséquence logique de la généralisation des hypothèses (clause(s) au-dessus de la barre) :

1. Factorisation. Soit  $\sigma$  l'unificateur principal de  $L_1$  et  $L_2$ , la règle de factorisation s'écrit

$$\frac{L_1 \vee L_2 \vee C}{L_1[\sigma] \vee C[\sigma]}$$

On veut montrer  $\forall(L_1 \vee L_2 \vee C) \models \forall(L_1[\sigma] \vee C[\sigma])$ .

On a  $\forall(L_1 \vee L_2 \vee C) \models L_1[\sigma] \vee L_2[\sigma] \vee C[\sigma]$  mais comme  $L_1[\sigma] = L_2[\sigma]$  (propriété de l'unificateur), on peut simplifier et on obtient  $\forall(L_1 \vee L_2 \vee C) \models L_1[\sigma] \vee C[\sigma]$  puis  $\forall(L_1 \vee L_2 \vee C) \models \forall(L_1[\sigma] \vee C[\sigma])$  car  $\forall(L_1 \vee L_2 \vee C)$  ne contient pas de variable libre.

2. Renommage. Si  $\sigma$  est une bijection sur les variables, la règle de renommage s'écrit  $\frac{C}{C[\sigma]}$ .

On a  $\forall(C) \equiv \forall(C[\sigma])$  (il s'agit d'une simple permutation des quantificateurs universels) d'où le résultat  $\forall(C) \models \forall(C[\sigma])$ .

3. Résolution binaire. Si  $\sigma$  est l'unificateur principal de  $L_1$  et  $L_2$ , la règle de résolution s'écrit

$$\frac{L_1 \vee C \quad \neg L_2 \vee C'}{C[\sigma] \vee C'[\sigma]}$$

On veut montrer  $\forall(L_1 \vee C), \forall(\neg L_2 \vee C') \models \forall(C[\sigma] \vee C'[\sigma])$ .

On a  $\forall(L_1 \vee C) \models L_1[\sigma] \vee C[\sigma]$  et  $\forall(\neg L_2 \vee C') \models \neg L_2[\sigma] \vee C'[\sigma]$ .

On a  $L_1[\sigma] = L_2[\sigma]$  (propriété de l'unificateur) donc le même raisonnement que dans le cas propositionnel nous donne que  $L_1[\sigma] \vee C[\sigma], \neg L_2[\sigma] \vee C'[\sigma] \models C[\sigma] \vee C'[\sigma]$ .

Et donc  $\forall(L_1 \vee C), \forall(\neg L_2 \vee C') \models C[\sigma] \vee C'[\sigma]$  et finalement  $\forall(L_1 \vee C), \forall(\neg L_2 \vee C') \models \forall(C[\sigma] \vee C'[\sigma])$  car les variables libres de  $C[\sigma] \vee C'[\sigma]$  ne sont pas libres dans  $\forall(L_1 \vee C), \forall(\neg L_2 \vee C')$ .

□

La méthode de résolution est aussi *complète* dans le cas de la logique du premier ordre, c'est-à-dire que si  $\mathcal{E}$  est un ensemble de clauses et que  $\perp$  est une conséquence logique de  $\{\forall(C) \mid C \in \mathcal{E}\}$  alors on peut déduire la clause vide par la méthode de résolution à partir de l'ensemble  $\mathcal{E}$ .

Ce résultat utilise le Théorème de Herbrand et la compacité qui nous permettent d'établir qu'il existe un ensemble fini insatisfiable d'instances closes des clauses de  $\mathcal{E}$  :

$$\mathcal{F} = \{C_1[\sigma_1], \dots, C_n[\sigma_n]\}$$

L'ensemble  $\mathcal{F}$  est propositionnel, on peut donc par le théorème de complétude de la résolution propositionnelle construire une réfutation par résolution de cet ensemble.

La clé de la preuve est ensuite le *Lemme de relèvement* qui permet d'établir que les étapes faites au niveau de la déduction propositionnelle peuvent se simuler au travers des règles de résolution au premier ordre.

**Proposition 4.4.2 (Lemme de relèvement)** *Soit  $C$  et  $D$  deux clauses du premier ordre sans variable commune. Si on peut faire une étape de résolution propositionnelle entre deux instances closes*

$$\frac{C[\sigma] \quad D[\sigma]}{P}$$

*alors il existe une clause  $E$  et une substitution  $\tau$  telle que  $E$  peut être obtenu à partir des clauses  $C$  et  $D$  par des règles de résolution du premier ordre et  $P = E[\tau]$ .*

Nous ne détaillerons pas la preuve ici. On pourra regarder plus précisément les points suivants.

1. pourquoi peut-on utiliser la même substitution pour  $C$  et  $D$  ?
2. à quoi sert la règle de factorisation ? de renommage ?

#### 4.4.2 Questions de stratégie

La méthode de résolution consiste à combiner des clauses dans le but d'obtenir la clause vide. Néanmoins il y a de nombreuses manières différentes de la faire et on peut s'interroger s'il existe des stratégies plus efficaces que d'autres.

##### Eliminer les clauses inutiles

Supposons qu'un des symboles de prédicat  $R$  n'apparaisse que de manière positive dans les clauses. Alors toutes les clauses dans lesquelles ce symbole apparaît peuvent être éliminées.

**Exercice 4.11** 1. Soit  $\mathcal{E}$  un ensemble de clauses, tel que chacune des clauses contient au moins un littéral  $R(t_1, \dots, t_n)$ . Montrer que  $\mathcal{E}$  est satisfiable.

2. Soit  $\mathcal{E}$  un ensemble de clauses, qui ne contient aucun littéral négatif  $\neg R(u_1, \dots, u_n)$ . Montrer que  $\mathcal{E}$  est satisfiable si et seulement si l'ensemble  $\{C \in \mathcal{E} \mid C \text{ ne contient pas de littéral } R(t_1, \dots, t_n)\}$  est satisfiable.

Le même résultat s'applique si le symbole  $R$  n'apparaît que de manière négative dans les clauses.

Il arrive que dans une dérivation, on puisse retomber sur une clause qui ressemble à une clause déjà rencontrée. Supposons qu'on ait à la fois une clause  $C$  et une clause  $C[\sigma] \vee D$ , alors la clause  $C[\sigma] \vee D$  peut être ignorée. En effet on a  $\forall(C) \models \forall(C[\sigma] \vee D)$  donc  $\mathcal{E}, \forall(C), \forall(C[\sigma] \vee D) \models \perp$  si et seulement si  $\mathcal{E}, \forall(C) \models \perp$ .

##### Stratégie unitaire et clauses de Horn

Lorsque l'on combine deux clauses par la méthode de résolution, on peut obtenir en général une clause qui est aussi grosse voir plus grosse que les clauses dont on est parti. Sauf si une des deux clauses est composée d'un seul littéral (on dit que la clause est unitaire) auquel cas la clause obtenue après résolution est plus petite que la clause dont on était parti.

Malheureusement, la stratégie de choisir de faire toujours une résolution avec une clause réduite à un littéral ne fonctionne pas. En effet il y a des ensembles de clauses contradictoires qui ne contiennent pas de clause unitaire et la stratégie de choisir toujours une clause unitaire peut nous amener à ne jamais considérer ces clauses. C'est ce qui se passe dans l'exemple suivant :

$$\{P(a), \neg P(x) \vee P(f(x)), \neg p \vee \neg q, \neg p \vee q, p \vee \neg q, p \vee q\}$$

La stratégie unitaire engendre un ensemble infini de clauses  $\{P(f(a)), P(f(f(a))), \dots, P(f^n(a)), \dots\}$  qui ne mène pas à une contradiction alors que l'ensemble  $\{\neg p \vee \neg q, \neg p \vee q, p \vee \neg q, p \vee q\}$  est lui insatisfiable (comme montré dans l'exercice 4.3).

Les **clauses de Horn** sont des cas particuliers de clauses qui contiennent au plus un seul littéral positif (sans négation). Les règles de la résolution appliquées à des clauses de Horn produisent des clauses de Horn.

L'intérêt des clauses de Horn réside dans le résultat suivant.

**Proposition 4.4.3 (Résolution unitaire pour les clauses de Horn)** *Soit  $S$  un ensemble de clauses de Horn, si  $S$  est insatisfiable, alors on peut dériver la clause vide en utilisant à chaque étape de résolution au moins une clause unitaire (résolution unitaire).*

**Preuve:** La preuve suit les étapes suivantes

- Si toutes les clauses ont un littéral négatif alors l'ensemble est satisfiable. En effet il suffit alors de choisir une interprétation dans laquelle tous les symboles de prédicats sont interprétés comme l'ensemble vide. Toutes les clauses auront alors leur littéral négatif qui vaut vrai et seront donc vraies.
- Si l'ensemble est insatisfiable alors il existe une clause sans littéral négatif donc avec un seul littéral positif puisque ce sont des clauses de Horn.
- Toute dérivation de la clause vide contient au moins une étape de résolution unitaire, puisqu'elle doit impliquer une des clauses sans littéral négatif.
- On peut transformer les étapes de résolution pour faire “remonter” les résolutions unitaires en utilisant la transformation suivante (présentée dans le cas propositionnel) :

$$\frac{\frac{\neg A \vee C \vee l \quad C' \vee \neg l}{\neg A \vee C \vee C'} \quad A}{C \vee C'}$$

$$\frac{\frac{\neg A \vee C \vee l \quad A}{C \vee l} \quad C' \vee \neg l}{C \vee C'}$$

- En itérant cette opération on peut construire une dérivation dont toutes les étapes de résolution sont unitaires.

□

**Application à la programmation logique** En 1972, l'informaticien Alain Colmerauer qui s'intéressait au traitement automatique des langues introduit, avec Philippe Roussel, le langage ProLog.

Le principe de base est de modéliser un problème à l'aide de Clauses de Horn.

Une clause avec un littéral positif  $p$  :  $\neg p_1 \vee \dots \vee \neg p_n \vee p$  est écrite sous la forme  $p : \neg p_1, \dots, p_n$ . et peut se lire comme  $(p_1 \wedge \dots \wedge p_n) \Rightarrow p$  qui est bien équivalent à  $\neg p_1 \vee \dots \vee \neg p_n \vee p$

Lorsque la clause ne contient pas de littéral positif, elle s'écrit sous la forme  $:\neg p_1, \dots, p_n$ . et peut se lire comme  $(p_1 \wedge \dots \wedge p_n) \Rightarrow \perp$  qui est bien équivalent à  $\neg p_1 \vee \dots \vee \neg p_n \equiv \neg(p_1 \wedge \dots \wedge p_n)$ .

Lorsque la clause est composée uniquement d'un littéral positif  $p$ , elle s'écrit simplement  $p$ .

Pour exécuter un programme, on va poser des questions sous la forme d'une **conjonction de littéraux positifs** qui peuvent contenir des variables.

Le système va rechercher les instances de ce programme qui sont conséquence logique des clauses du programme. Pour cela il va utiliser la méthode de résolution pour rechercher des contradictions à partir de la négation de ces prédicats.

**Exemple 4.9** Une base qui modélise les relations de parenté peut se décrire avec les clauses suivantes.

```

1  fratrie (X,Y) :- parent (Z,X), parent (Z,Y), X \= Y.
2  parent (X,Y) :- pere (X,Y).
3  parent (X,Y) :- mere (X,Y).
4  pere (tom, sally).
5  pere (tom, erica).

```

Les questions que l'on peut poser sont "est-ce que *sally* et *erica* sont dans la même fratrie" ou bien qui est dans la fratrie de *sally*. Cela se décrira en ProLog par les clauses suivantes :

```

?-fratrie ( sally , erica ).
?-fratrie ( sally , x ).

```

Supposons que la question soit

```

6  ?-fratrie ( sally , erica ).

```

Les seuls clauses unitaires sont les clauses 4 et 5. on peut combiner les clauses 2 et 4 ainsi que 2 et 5 pour obtenir les clauses unitaires suivantes :

```

7  parent (tom, sally).
8  parent (tom, erica).

```

On peut ensuite faire la résolution de la clause 1 avec ces clauses 7 puis 8 et on obtient

```

9  fratrie ( sally , Y) :- parent (tom, Y), sally \= Y.
10 fratrie ( sally , erica ).

```

La contrainte *sally*  $\neq$  *erica* est résolue indépendamment.

La clause 10 se combine alors avec la question 6 pour obtenir la clause vide ce qui établit le résultat.

Si maintenant la question est de chercher toutes les personnes dans la fratrie de *sally*, cela se traduit par :

```

6  ?-fratrie ( sally , x ).

```

il s'agit de trouver pour quelles valeurs de *x* ce problème a une solution. Les étapes vont être analogues

```

7  parent (tom, sally).
8  parent (tom, erica).

9  fratrie ( sally , Y) :- parent (tom, Y), sally \= Y.
10 fratrie ( sally , erica ).

```

La clause 10 se combine alors avec la question 6 pour obtenir la clause vide au travers de la substitution *x*  $\leftarrow$  *erica* qui est renvoyée à l'utilisateur.

Ce langage a des applications en intelligence artificielle, linguistique, bases de données, etc.

Un important aspect est la possibilité de retour en arrière (*back-tracking*) après avoir trouvé une solution pour énumérer l'ensemble des instances possibles solutions du problème.

**Compétences 4.4** Savoir effectuer une preuve par réfutation d'une formule arbitraire en appliquant toutes les étapes.

## 4.5 Calcul des séquents

Nous avons vu que le système G avait des bonnes propriétés dans le cas propositionnel. Au premier ordre, les règles  $\forall g$  et  $\exists d$  nécessitent de trouver le bon terme à appliquer et il peut aussi être nécessaire de les appliquer plus d'une fois sans que l'on puisse a priori borner cette recherche.

Le calcul des séquents est la base théorique des méthodes de démonstration automatique appelées *méthodes des tableaux*.

Le principal ingrédient est d'ajouter au langage des termes une nouvelle sorte de variables, les *inconnues* qui seront utilisées lors de l'application des règles  $\forall g$  et  $\exists d$  à la place du terme  $t$  à “deviner”. On décompose tous les connecteurs pour chercher des solutions au niveau des feuilles afin de pouvoir appliquer la règle axiome. Là encore c'est l'unification qui va nous permettre de trouver des solutions pour nos inconnues.

Les règles modifiées sont les suivantes, au lieu d'un terme quelconque,  $t$  nous remplaçons la variable liée par une “inconnue  $\mathbf{X}$ ” :

$$(\forall'g) \frac{P[x \leftarrow \mathbf{X}], (\forall x, P), \Gamma \vdash \Delta}{(\forall x, P), \Gamma \vdash \Delta} \quad (\exists'd) \frac{\Gamma \vdash \Delta, (\exists x, P), P[x \leftarrow \mathbf{X}]}{\Gamma \vdash \Delta, (\exists x, P)}$$

Un arbre de dérivation qui contient des inconnues ne constitue pas une preuve. Il sera souvent incomplet car les inconnus peuvent empêcher l'application de la règle hypothèse aux feuilles.

#### Exemple 4.10

$$\begin{array}{c} \exists'd \frac{R(\mathbf{X}, y), (\forall x, R(x, y)) \vdash (\exists y, R(x, y)), R(x, \mathbf{Y})}{R(\mathbf{X}, y), (\forall x, R(x, y)) \vdash \exists y, R(x, y)} \\ \forall'g \frac{}{\forall x, R(x, y) \vdash \exists y, R(x, y)} \\ \forall d(2) \frac{}{\forall x, R(x, y) \vdash \forall x, \exists y, R(x, y)} \\ \exists g(1) \frac{}{\exists y, \forall x, R(x, y) \vdash \forall x, \exists y, R(x, y)} \end{array}$$

Pour que l'arbre puisse être complété par une règle hypothèse, on voit qu'il faut appliquer la substitution  $\{\mathbf{X} \leftarrow x, \mathbf{Y} \leftarrow y\}$

Une difficulté supplémentaire est que les règles droites pour le  $\forall$  et gauche pour le  $\exists$  ont des conditions sur les noms des variables ordinaires de termes ; et que cela ajoute des contraintes sur les solutions pour les inconnues. Il faut donc vérifier que les solutions trouvées par l'unification respectent ces conditions. C'est le cas de l'exemple précédent, les règles  $\exists g(1)$  qui introduit  $y$  et  $\forall d(2)$  qui introduit  $x$  sont utilisées avant l'introduction d'inconnues et respectent la condition que les variables introduites ne sont pas utilisées ailleurs.

Si maintenant on essaie avec la même méthode de faire une preuve de la propriété réciproque.

$$\begin{array}{c} \forall d(2) \frac{R(\mathbf{X}, y), (\forall x, \exists y, R(x, y)) \vdash (\exists y, \forall x, R(x, y)), R(x, \mathbf{Y})}{R(\mathbf{X}, y), (\forall x, \exists y, R(x, y)) \vdash (\exists y, \forall x, R(x, y)), (\forall x, R(x, \mathbf{Y}))} \\ \exists g(1) \frac{}{(\exists y, R(\mathbf{X}, y)), (\forall x, \exists y, R(x, y)) \vdash (\exists y, \forall x, R(x, y)), (\forall x, R(x, \mathbf{Y}))} \\ \forall'g \frac{}{\forall x, \exists y, R(x, y) \vdash (\exists y, \forall x, R(x, y)), (\forall x, R(x, \mathbf{Y}))} \\ \exists'd \frac{}{\forall x, \exists y, R(x, y) \vdash \exists y, \forall x, R(x, y)} \end{array}$$

Il est tentant de compléter la preuve comme précédemment en appliquant la substitution  $\{\mathbf{X} \leftarrow x, \mathbf{Y} \leftarrow y\}$ . Cependant cela nous donnerait l'arbre suivant (pour ne pas alourdir l'arbre, on a retiré les formules dupliquées dans les règles  $\exists d$  et  $\forall g$ )

$$\begin{array}{c} \text{hyp} \frac{}{R(x, y) \vdash R(x, y)} \\ \forall d(2) \frac{}{R(x, y) \vdash \forall x, R(x, y)} \\ \exists g(1) \frac{}{\exists y, R(x, y) \vdash \forall x, R(x, y)} \\ \forall g \frac{}{\forall x, \exists y, R(x, y) \vdash \forall x, R(x, y)} \\ \exists d \frac{}{\forall x, \exists y, R(x, y) \vdash \exists y, \forall x, R(x, y)} \end{array}$$



L'arbre n'a plus de feuilles mais ne constitue pas néanmoins un arbre de preuve (le séquent est d'ailleurs non valide). En effet la condition que  $y$  est une variable non utilisée dans la règle  $\exists g(1)$  n'est pas vérifiée puisque  $y$  est libre dans la conclusion  $\forall x, R(x, y)$  et de même la condition de la règle  $\forall d(2)$  qui impose que  $x$  n'est pas utilisé par ailleurs n'est pas respectée puisque  $x$  est libre dans l'hypothèse  $R(x, y)$ . On voit donc que l'utilisation des règles  $\exists g$  et  $\forall d$  va imposer des restrictions sur les variables qui peuvent être utilisées pour les valeurs des inconnues. Ces conditions devront être vérifiées globalement lorsqu'on cherche les solutions pour compléter les feuilles par des règles hypothèses.

## 4.6 Théories décidables

**A propos de théories** Une théorie est un ensemble (fini ou infini) de formules *closes*. Une théorie peut se voir comme un ensemble de contraintes sur les symboles de la signature. Des exemples sont la théorie des ensembles, fondement des mathématiques modernes, l'arithmétique de Peano, cadre logique plus restreint mais qui permet de parler des fonctions calculables ou encore dans le cadre informatique des théories correspondant à des structures de données comme la théorie des piles.

Soit  $\mathcal{A}$  une théorie, on s'intéresse aux formules qui sont vraies dans une théorie  $\mathcal{A}$ , c'est-à-dire que toute interprétation qui *satisfait* les axiomes de la théorie, satisfait la formule  $P$ , ce qui est noté  $\mathcal{A} \models P$ . La calcul des prédicats est indécidable en général mais certaines théories sont décidables.

- théorie de l'égalité (Shostak)
- théorie de l'arithmétique linéaire (Presburger)
- certaines théories d'ensembles ordonnés

**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 ne permet pas de déduire  $\perp$
- $\mathcal{A}$  est *décidable* s'il existe un algorithme qui étant donné une formule  $P$  permet de savoir si  $P$  est prouvable dans la théorie ou pas
- $\mathcal{A}$  est *complète* si pour toute formule  $P$ , on peut démontrer  $P$  ou bien on peut démontrer  $\neg P$

**Théories incomplètes** Beaucoup de théories sont incomplètes

- théorie des ordres : une relation binaire  $R$ , réflexive, transitive et anti-symétrique
- la propriété qui dit que  $R$  est une relation totale n'est pas montrable dans cette théorie mais n'est pas non plus réfutable (il y a des ordres qui sont totaux et des ordres qui ne le sont pas).

**Proposition 4.6.1 (Décidabilité des théories complètes)** Une théorie complète, récursive sur un ensemble dénombrable de symboles est décidable.

**Preuve:** Les preuves de  $\mathcal{A} \vdash P$  peuvent être énumérées car la théorie est récursive et l'ensemble des symboles est dénombrable. On cherche en parallèle des preuves de  $\mathcal{A} \vdash P$  et de  $\mathcal{A} \vdash \neg P$ , l'un des deux va aboutir.  $\square$

### Théorème d'incomplétude de Gödel

**Proposition 4.6.2** Toute théorie non contradictoire qui contient l'arithmétique élémentaire  $PA_0$  n'est pas complète.

Il n'y a pas de théorie "universelle" (ex. théorie des ensembles) qui permette de montrer soit  $A$  soit  $\neg A$ , pour toute formule.

- langage : constante  $O$ , symbole unaire  $S$ , symboles binaires  $+$  et  $\times$ , symbole de prédicat d'égalité.

- axiomes de l'arithmétique élémentaire
  - $\forall x, S(x) \neq O$
  - $\forall x, x = O \vee \exists y, x = S(y)$  (inutile en présence de récurrence)
  - $\forall x y, S(x) = S(y) \Rightarrow x = y$
  - $\forall x, x + O = x$
  - $\forall x y, x + S(y) = S(x + y)$
  - $\forall x, x \times O = O$
  - $\forall x y, x \times S(y) = (x \times y) + x$
- A tout entier  $n \in \mathbb{N}$ , on associe le terme  $S^n(0)$  noté  $\tilde{n}$ .
- il faut ajouter la récurrence (ensemble dénombrable d'axiomes) pour avoir l'arithmétique de Peano

# Conclusion

La logique est une base essentielle pour tout scientifique qui se doit de savoir justifier les solutions qu'il propose. C'est également un outil utilisé dans de nombreuses applications informatique :

- modélisation des connaissances ;
- contraintes sur les bases de données ;
- propriétés théoriques des bases de données (modèles finis) ;
- intelligence artificielle : systèmes experts, modélisation des comportements dans les jeux ;
- spécifications formelles de bibliothèques, de programmes (invariants de boucle), test, vérification ;
- méthodes algorithmiques de résolution de problèmes...

Ce cours a présenté les éléments de base de la logique : calcul propositionnel et calcul des prédicats du premier ordre. Il existe de nombreuses autres logiques adaptées à des besoins spécifiques : logique multi-sortée, logique d'ordre supérieur, logiques temporelles ou modales. Les notions de base de syntaxe, d'interprétation et de validité s'adaptent à ces cas.

Une des difficultés pour l'informaticien est d'avoir à écrire des programmes qui manipulent des formules logiques pour les transformer et établir certaines propriétés. Cela nécessite de bien comprendre la distinction entre syntaxe et sémantique et de manipuler des objets qui ont des structures essentiellement arborescentes (avec la difficulté supplémentaire des variables liées). Une technique utile largement illustrée dans ce cours est de définir des fonctions de manière récursive sur la structure des formules, de définir des propriétés des formules par des systèmes d'inférence et d'utiliser les principes de récurrence associé pour établir les propriétés mathématiques des objets manipulés.

La logique est également un domaine qui illustre la richesse des solutions algorithmiques pour résoudre des problèmes : transformation de formules, tables de vérité, systèmes de preuve, résolution... Nous avons présenté des algorithmes élémentaires mais dont les principes sont au cœur de systèmes opérationnels plus avancés.

# Index

- arbre de décision binaire, 65
- arbre de dérivation, 72
- Arithmétique de Peano, 24
- Arité, 10
- Axiomes, 23
  
- Base de Herbrand, 48
- BDD, 66
- BDT, 65
- Binaire
  - Symbole binaire, 10
  
- Calcul des prédicats, 14
- Calcul propositionnel, 14
- clause, 53
  - clause triviale, 53
  - clause vide, 53
- clauses de Horn, 106
- Clos
  - Formule close, 18
  - Terme clos, 18
- Conjonction, 12
- conjonction élémentaire, 57
- Connecteur logique, 12
- Constante, 10
- Conséquence logique, 42
- correspondance de Post, 90
  
- diagramme de décision binaire, 66
- Disjonction, 12
- Domaine de Herbrand, 47
- déduction naturelle, 80
- Définition récursive sur les termes, 30
  
- Environnement, 35
- equisatisfiable, 58
- Equivalence, 13, 42
  
- fermeture universelle, 63
- filtrage, 101
  
- FNC, 54
- FND, 57
- fonction booléenne associée à une formule, 55
- forme clausale, 54, 63
- Forme normale de négation, 45
- forme normale disjonctive, 54, 57
- forme prénexe, 60
- Formule
  - Formule atomique, 11
  - Formule close, 18
  - Formule complexe, 12
  - Formule insatisfiable, 20
  - Formule satisfiable, 20
  - Formule valide, 20
- formule skolémisée, 62
- Formule universelle, 48
  
- Herbrand
  - Base de Herbrand, 48
  - Domaine de Herbrand, 47
  - Interprétation de Herbrand, 47
  - Modèle de Herbrand, 48
  
- Implication, 12
- indécidabilité, 90
- Infixe
  - Notation infix, 10
- Insatisfiable (formule), 20
- Instance close, 49
- instance de prédicat, 51
- Interprétation, 33
- Interprétation de Herbrand, 47
- isomorphisme de Curry-Howard, 81
  
- littéral, 51
- Logique du premier ordre, 14
- Logique propositionnelle, 14
  
- minimalité, 73

- Modèle
  - Modèle d'une théorie, 23
- Modèle de Herbrand, 48
- motif, 101
- Négation, 12
- OBDD, 66
- Précédence, 16
- Quantificateur logique, 12
- Quantification existentielle, 12
- Quantification universelle, 12
- règles d'inférence, 70
- résolution
  - déduction par résolution, 102
  - preuve par résolution, 93
  - règles de résolution, 102
  - réfutation par résolution, 93
  - résolution propositionnelle, 93
- Satisfiable (formule), 20
- Signature, 10, 23
- Skolem
  - forme de, 62
- Skolemisation, 62
- Substitution
  - Substitution dans les formules, 18
  - Substitution sur les formules, 32
  - Substitution sur les termes, 31
- Symbole, 9
  - Symbole binaire, 10
  - Symbole de prédicat, 11
  - Symbole de tête, 10
  - Symbole unaire, 10
- système d'inférence, 70
- système G, 82
- séquent, 74
  - formule associée, 74, 82
  - séquent multi-conclusions, 82
- Table de vérité, 19
- Tautologie, 20
- Terme, 10
  - Sous-terme, 10
  - Symbole de tête, 10
  - Terme clos, 10, 18
- Théorie, 23
  - Théorie complète, 23
  - Théorie décidable, 23
  - Théorie incohérente, 23
- Théorie de l'égalité, 23
- Unaire
  - Symbole unaire, 10
- unificateur, 98
  - unificateur principal, 99
- unification, 99
- Valeur de vérité, 18
- Valide (formule), 20
- Variable
  - Variable liée, 17
  - Variable propositionnelle, 11
  - Variables libres, 17
- Variables
  - Variables d'objet, 10

# Annexe A

## Correction des exercices du cours

### A.1 Exercices du chapitre 1

#### Exercice 1.1

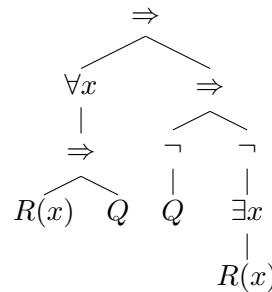
1.  $\forall x, (\text{joue}(\text{self}, x) \Rightarrow \text{ami}(\text{self}, x))$
2.  $\forall x, \exists y, \text{ami}(x, y)$
3.  $\neg(\exists x, (\text{joue}(\text{self}, x) \wedge \text{ami}(\text{self}, x)))$  ou de manière équivalente  $\forall x, ((\neg \text{joue}(\text{self}, x)) \vee (\neg \text{ami}(\text{self}, x)))$
4.  $\exists x y, (\text{joue}(\text{self}, x) \wedge \text{joue}(\text{self}, y) \wedge \neg(x = y))$  sans la mention de  $x \neq y$  la formule est vraie dès qu'il existe une personne avec qui je joue.

**Exercice 1.2**  $\forall x y, ((\text{pair}(x) \wedge \text{pair}(y)) \Rightarrow \text{pair}(x + y))$

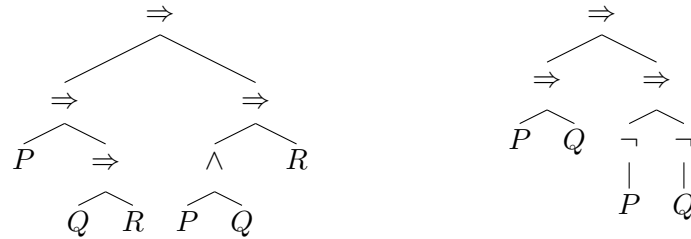
#### Exercice 1.3

- $\forall x \in \mathbb{N}, P$  correspond à  $\forall x, (N(x) \Rightarrow P)$
- $\exists x \in \mathbb{N}, P$  correspond à  $\exists x, (N(x) \wedge P)$

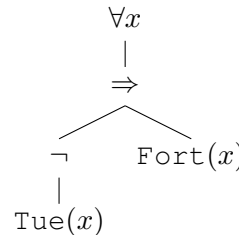
**Exercice 1.4** La formule parenthésée est  $(\forall x, (R(x) \Rightarrow Q)) \Rightarrow ((\neg Q) \Rightarrow (\neg(\exists x, R(x))))$  et sous forme d'arbre :



**Exercice 1.5** Les formules parenthésées sont  $(P \Rightarrow (Q \Rightarrow R)) \Rightarrow ((P \wedge Q) \Rightarrow R)$  et  $(P \Rightarrow Q) \Rightarrow ((\neg Q) \Rightarrow (\neg P))$  et sous forme d'arbre

**Exercice 1.6**

- $\forall x, \neg \text{Tue}(x) \Rightarrow \text{Fort}(x)$  correspond au minimum de parenthèses
- $\forall x, ((\neg \text{Tue}(x)) \Rightarrow \text{Fort}(x))$  correspond à la formule complètement parenthésée
- La forme arborescente est



**Exercice 1.7** La formule :  $\forall b, b > 0 \Rightarrow \exists q, \exists r, a = b \times q + r \wedge r < b$  a une seule variable libre :  $a$ .

**Exercice 1.8**

1. La formule  $((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$  est valide. En effet soit  $P$  est vrai et elle est vraie. Soit  $P$  est faux, mais alors  $P \Rightarrow Q$  est vrai et donc  $(P \Rightarrow Q) \Rightarrow P$  est faux et donc  $((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$  est vrai. Dans toutes les interprétations possibles, la formule est vraie. Donc elle est valide.
2. La formule  $(\neg P \Rightarrow P) \Rightarrow P$  est valide. En effet soit  $P$  est vrai et elle est vraie. Soit  $P$  est faux, mais alors  $\neg P$  est vrai et donc  $\neg P \Rightarrow P$  est faux et donc  $(\neg P \Rightarrow P) \Rightarrow P$  est vrai. Dans toutes les interprétations possibles, la formule est vraie. Donc elle est valide.

**Exercice 1.9**

- L'inscription sur la porte 1 correspond à la formule  $A_1 \stackrel{\text{def}}{=} P_1 \wedge \neg P_2$ ,
- L'inscription sur la porte 2 correspond à la formule  $A_2 \equiv (P_1 \wedge \neg P_2) \vee (P_2 \wedge \neg P_1)$ ,
- Une seule de ces inscriptions est vraie donc la formule  $(A_1 \wedge \neg A_2) \vee (A_2 \wedge \neg A_1)$  est vraie.

On peut faire une table de vérité de cette dernière formule en fonction des valeurs de  $P_1$  et  $P_2$  et chercher dans quels cas elle est vraie.

On peut aussi raisonner directement.

Comme  $A_2$  est de la forme  $A_1 \vee B$ , si  $A_1$  est vrai alors  $A_2$  est vrai et donc la troisième formule est fausse. Donc  $A_1$  est faux et  $A_2$  est vrai et on a  $P_2 \wedge \neg P_1$  donc il faut ouvrir la deuxième porte.

**Exercice 1.10** La formule  $(\forall x, P(x)) \Rightarrow \exists x, P(x)$  est vraie quelle que soit l'interprétation de  $P$ . En effet si on se donne une interprétation  $I$  sur un domaine  $\mathcal{D}$ , qui rend vraie  $\forall x, P(x)$ . L'interprétation de  $P$  est vraie pour toutes les valeurs dans le domaine  $\mathcal{D}$ . Comme le domaine est non vide, on peut choisir  $d \in \mathcal{D}$ . L'interprétation de  $P$  est vraie pour  $d$  et donc  $\exists x, P(x)$  est vraie dans cette interprétation.

La formule  $\neg(\forall x, P(x)) \Rightarrow \exists x, \neg P(x)$  est vraie quelle que soit l'interprétation de  $P$ . En effet si on se donne une interprétation  $I$  sur un domaine  $\mathcal{D}$ , qui rend vraie  $\neg(\forall x, P(x))$ . Cette interprétation rend faux  $\forall x, P(x)$  ce qui veut dire qu'il existe  $d \in \mathcal{D}$  pour lequel l'interprétation  $P$  est fausse et donc l'interprétation de  $\exists x, \neg P(x)$  est vraie.



**Exercice 1.11**  $t \leq u \stackrel{\text{def}}{=} \exists n, n + t = u$ .

- $\forall x, 0 \leq x$  se réécrit en  $\forall x, \exists n, n + 0 = x$ . Soit  $x$  quelconque, on choisit pour  $n$  la valeur  $x$ . Il faut alors montrer  $x + 0 = x$  qui est l'axiome 4 de la théorie arithmétique.
- $\forall xy, x \leq y \Leftrightarrow S(x) \leq S(y)$  se réécrit en  $\forall xy, (\exists n, n + x = y) \Leftrightarrow (\exists n, n + S(x) = S(y))$ . Soit  $x$  et  $y$  quelconques, on montre les deux côtés de l'équivalence.  
Si  $\exists n, n + x = y$ , soit  $n$  tel que  $n + x = y$ , on a  $S(n + x) = S(y)$  (propriété de la théorie de l'égalité). On en déduit  $n + S(x) = S(y)$  (axiome 5 de l'arithmétique) et donc  $\exists n, n + S(x) = S(y)$ .  
Réciproquement si  $\exists n, n + S(x) = S(y)$ , soit  $n$  tel que  $n + S(x) = S(y)$ , on en déduit  $S(n + x) = S(y)$  (axiome 5 de l'arithmétique). On en déduit  $n + x = y$  (axiome 3 de l'arithmétique) et donc  $\exists n, n + x = y$ .
- transitivité :  $\forall xyz, x \leq y \Rightarrow y \leq z \Rightarrow x \leq z$  se réécrit en  $(\exists n, n + x = y) \Rightarrow (\exists m, m + y = z) \Rightarrow (\exists p, p + x = z)$ .  
Soit  $x, y$  et  $z$  quelconques, on suppose  $\exists n, n + x = y$  et  $\exists m, m + y = z$  vrai et donc soit  $n$  et  $m$  tels que  $n + x = y$  et  $m + y = z$ . Par la théorie de l'égalité, on a  $m + (n + x) = m + y$  et par la transitivité de l'égalité on en déduit  $m + (n + x) = z$ . L'associativité de l'addition (non démontrée) permet d'en déduire que  $(m + n) + x = z$  et donc  $\exists p, p + x = z$  ce qui conclut la preuve.

**Exercice 1.12** Les définitions utilisent la notion d'ordre strict  $t < u$  qui peut se définir par  $\exists n, S(n + t) = u$ .

1. Le fait que  $r$  est le résultat de la division entière de  $x$  par  $y$  peut se représenter par la formule  $\exists r, x = d \times y + r \wedge r < y$ .
2. Le fait que  $r$  est le reste modulo de la division entière de  $x$  par  $y$  peut se représenter par la formule  $\exists d, x = d \times y + r \wedge r < y$ .
3. Si on remplace  $y$  par 0 dans la définition de divisibilité, on obtient la formule  $\exists r, x = d \times 0 + r \wedge r < 0$  qui se développe en  $\exists r, x = d \times 0 + r \wedge (\exists n, S(n + r) = 0)$  formule qui est toujours fausse si on se place dans un monde qui satisfait les axiomes de la théorie arithmétique et en particulier  $\forall x, S(x) \neq 0$ .
4. Pour représenter le résultat de la division de  $a$  par  $b$  à l'aide d'un prédicat à trois arguments, il suffit d'introduire une nouvelle variable  $d$  et d'utiliser la formule  $\text{div}(a, b, d)$ . Dans le cas de  $(x/y)/z = x/(y \times z)$ , il y a trois opérations de division, on introduit donc trois nouvelles variables  $d_1, d_2$  et  $d_3$ .  $d_1$  est le résultat de la division de  $x$  par  $y$ , ce qui se traduit par  $\text{div}(x, y, d_1)$ ,  $d_2$  est le résultat de la division de  $x/y$  (c'est-à-dire  $d_1$ ) par  $z$  ce qui se traduit par  $\text{div}(d_1, z, d_2)$ , finalement  $d_3$  est le résultat de la division de  $x$  par  $y \times z$  ce qui se traduit par  $\text{div}(x, y \times z, d_3)$ . Il suffit alors d'exprimer que sous ses hypothèses on a  $d_2 = d_3$ . Ce qui s'écrit

$$\forall d_1 d_2 d_3, \text{div}(x, y, d_1) \wedge \text{div}(d_1, z, d_2) \wedge \text{div}(x, y \times z, d_3) \Rightarrow d_2 = d_3$$

Il est possible de montrer que la relation  $\text{div}$  est fonctionnelle c'est-à-dire que  $\text{div}(x, y, d_1) \wedge \text{div}(x, y, d_2) \Rightarrow d_1 = d_2$ , la formule précédente peut donc se simplifier en

$$\forall d_1 d_2, \text{div}(x, y, d_1) \wedge \text{div}(d_1, z, d_2) \Rightarrow \text{div}(x, y \times z, d_2)$$

**Exercice 1.13** Il est nécessaire en plus du prédicat  $\text{pos}(i, j, k)$  qui exprime que le chiffre  $k$  est sur la case  $(i, j)$  de disposer du prédicat d'égalité pour pouvoir dire qu'un chiffre apparaît au plus une fois.

1. Au plus un seul chiffre  $k$  par case  $(i, j)$  :  $\forall i j k, \text{pos}(i, j, k) \Rightarrow \forall l, \neg l = k \Rightarrow \neg \text{pos}(i, j, l)$
2. Chaque chiffre  $k$  apparaît sur chaque ligne  $i$  au moins dans une colonne  $j$  :  $\forall i k, \exists j, \text{pos}(i, j, k)$
3. Chaque chiffre  $k$  apparaît au plus une fois sur chaque ligne  $i$  (s'il est dans la colonne  $j$ , il n'est pas dans une autre colonne) :  $\forall i j_1 j_2 k, \text{pos}(i, j_1, k) \wedge \text{pos}(i, j_2, k) \Rightarrow j_1 = j_2$

4. Le cas des colonnes est analogue à celui des lignes. On traite ici la notion de carré. Si on veut éviter de traiter chaque valeur de ligne et de colonne individuellement, on peut introduire un symbole de fonction **tiers** unaire qui, étant donné une ligne ou une colonne, va nous dire dans quel tiers du tableau elle se trouve (premier, deuxième ou troisième). On peut ainsi tester si deux positions  $(i_1, j_1)$  et  $(i_2, j_2)$  seront dans le même carré si et seulement si les indices de ligne sont dans le même tiers du tableau et de même pour les indices de colonnes.

Pour représenter le fait que chaque chiffre  $k$  apparaît au plus une fois dans chaque carré on utilisera la formule :

$$\begin{aligned} \forall i_1 i_2 j_1 j_2 k, \text{tiers}(i_1) = \text{tiers}(i_2) \wedge \text{tiers}(j_1) = \text{tiers}(j_2) \\ \Rightarrow \text{pos}(i_1, j_1, k) \wedge \text{pos}(i_2, j_2, k) \\ \Rightarrow i_1 = i_2 \wedge j_1 = j_2 \end{aligned}$$

On peut aussi représenter le fait que chaque chiffre apparaîtra au moins une fois dans chaque carré de la manière suivante :

$$\forall i_1 j_1 k, \exists i_2 j_2, \text{tiers}(i_1) = \text{tiers}(i_2) \wedge \text{tiers}(j_1) = \text{tiers}(j_2) \wedge \text{pos}(i_2, j_2, k)$$

Les deux formules sont redondantes dès lors qu'il y a 9 valeurs et 9 cases. En effet, on a écrit qu'il y avait au moins une valeur par case et qu'il ne pouvait pas y avoir deux fois la même valeur, donc chaque valeur va apparaître au moins une fois car une application injective (resp. surjective) d'un ensemble fini dans un ensemble de même cardinal est bijective.

5. On exprime dans la logique le fait qu'on a 9 valeurs possibles pour les lignes, les colonnes et les chiffres dans les cases. Cela se fait en ajoutant 9 constantes :  $1, \dots, 9$ . Il faut un axiome qui dit que l'univers est formé uniquement de ces 9 éléments  $\forall x, x = 1 \vee \dots \vee x = 9$  auquel il faut ajouter les 36 axiomes qui disent que les constantes sont différentes deux à deux  $\neg 1 = 2, \neg 1 = 3, \dots, \neg 1 = 9, \neg 2 = 3, \dots, \neg 8 = 9$ .

Il faudra aussi ajouter les 9 axiomes qui permettent de raisonner sur la fonction **tiers** à savoir  $\text{tiers}(1) = 1, \text{tiers}(2) = 1, \text{tiers}(3) = 1, \dots, \text{tiers}(9) = 3$ .

#### Exercice 1.14

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

#### Exercice 1.15

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

#### Exercice 1.16

- 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  $\text{vars}(f(t_1, \dots, t_n)) = \text{vars}(t_1) \cup \dots \cup \text{vars}(t_n)$

**Exercice 1.17** On considère ici des termes qui ne contiennent pas de variable

— définition de **feuilles**

$$\text{feuilles}(\epsilon) = 1 \quad \text{feuilles}(N(l, r)) = \text{feuilles}(l) + \text{feuilles}(r)$$

— définition de **noeuds**

$$\text{noeuds}(\epsilon) = 0 \quad \text{noeuds}(N(l, r)) = 1 + \text{noeuds}(l) + \text{noeuds}(r)$$

— preuve par récurrence que pour tout terme  $t$  sans variable, on a  $\text{feuilles}(t) = \text{noeuds}(t) + 1$ .

— cas où  $t$  est la constante  $\epsilon$  on a  $\text{feuilles}(\epsilon) = 1$  et  $\text{noeuds}(\epsilon) = 0$  d'où le résultat.

— cas où  $t$  est de la forme  $N(l, r)$  avec  $l$  et  $r$  deux sous-arbres qui vérifient l'hypothèse de récurrence  $\text{feuilles}(l) = \text{noeuds}(l) + 1$  et  $\text{feuilles}(r) = \text{noeuds}(r) + 1$ .

On a

$$\begin{aligned} \text{feuilles}(N(l, r)) &= \text{feuilles}(l) + \text{feuilles}(r) && \text{définition de feuilles} \\ &= \text{noeuds}(l) + 1 + \text{noeuds}(r) + 1 && \text{hypothèses de récurrence} \\ &= \text{noeuds}(N(l, r)) + 1 && \text{définition de noeuds} \end{aligned}$$

On en déduit que la propriété est vraie pour tous les arbres.

**Exercice 1.18** La théorie de l'égalité contient les axiomes pour la réflexivité, la symétrie et la transitivité ainsi que des axiomes pour les symboles de fonctions et les prédicats d'arité au moins 1. Dans le cas de la signature qui est donnée, il y aura donc deux axiomes

1.  $\forall x_1 x_2 y_1 y_2, x_1 = x_2 \wedge y_1 = y_2 \Rightarrow g(x_1, y_1) = g(x_2, y_2)$
2.  $\forall x_1 x_2 y_1 y_2, x_1 = x_2 \wedge y_1 = y_2 \Rightarrow Q(x_1, y_1) \Rightarrow Q(x_2, y_2)$

On veut montrer que la formule  $t = u \Rightarrow P[x \leftarrow t] \Rightarrow P[x \leftarrow u]$  est valide pour n'importe quelle formule  $P$  du langage.

On commence par montrer un cas particulier à savoir que la formule  $t = u \Rightarrow v[x \leftarrow t] = v[x \leftarrow u]$  est valide pour n'importe quel terme  $v$  du langage. On se place dans une interprétation qui rend vraie la formule  $t = u$  (et les axiomes de l'égalité) et on utilise une récurrence sur la structure du terme  $v$  pour montrer que  $v[x \leftarrow t] = v[x \leftarrow u]$  est également vraie dans cette interprétation.

- si  $v$  est une variable  $z$  alors soit  $z = x$  et la formule devient  $t = u$  qui est vraie par hypothèse sur l'interprétation, soit  $z \neq x$  et la formule devient  $z = z$  qui est vraie par réflexivité de l'égalité.
- si  $v$  est la constante  $a$  alors la formule devient  $a = a$  qui est vraie par réflexivité de l'égalité.
- si  $v$  est de la forme  $g(v_1, v_2)$  avec  $v_1$  et  $v_2$  qui vérifient l'hypothèse de récurrence à savoir  $v_1[x \leftarrow t] = v_1[x \leftarrow u]$  et  $v_2[x \leftarrow t] = v_2[x \leftarrow u]$ . On a  $g(v_1, v_2)[x \leftarrow t] = g(v_1[x \leftarrow t], v_2[x \leftarrow t])$ , par définition de la substitution. En appliquant les hypothèses de récurrence et l'axiome 1 de l'égalité ci-dessus, on obtient  $g(v_1, v_2)[x \leftarrow t] = g(v_1[x \leftarrow u], v_2[x \leftarrow u])$  et donc finalement  $g(v_1, v_2)[x \leftarrow t] = g(v_1, v_2)[x \leftarrow u]$ .

On montre ensuite par récurrence sur la formule  $P$  que pour tous les termes  $t$  et  $u$  et toute interprétation qui rend vraie  $t = u$ , on a  $P[x \leftarrow t] \Rightarrow P[x \leftarrow u]$  est vraie quelque soit l'environnement pour les valeurs des variables libres de  $P[x \leftarrow t]$ .

- Si  $P$  est une formule atomique : Si  $P$  est  $\perp$  ou  $\top$  qui ne contient pas  $x$  alors il faut montrer  $P \Rightarrow P$  qui est une tautologie.

Si  $P$  est de la forme  $Q(v_1, v_2)$  alors il faut montrer  $Q(v_1, v_2)[x \leftarrow t] \Rightarrow Q(v_1, v_2)[x \leftarrow u]$  Par définition de la substitution, cela revient à montrer  $Q(v_1[x \leftarrow t], v_2[x \leftarrow t]) \Rightarrow Q(v_1[x \leftarrow u], v_2[x \leftarrow u])$  en utilisant l'axiome 2 de la théorie de l'égalité, il suffit de montrer  $v_1[x \leftarrow t] = v_1[x \leftarrow u] \wedge v_2[x \leftarrow t] = v_2[x \leftarrow u]$ . Or cette propriété est vraie d'après ce que nous avons établi précédemment.

- Si  $P$  est de la forme  $\neg A$  alors il faut montrer  $(\neg A)[x \leftarrow t] \Rightarrow (\neg A)[x \leftarrow u]$ , c'est-à-dire  $\neg(A[x \leftarrow t]) \Rightarrow \neg(A[x \leftarrow u])$  en prenant la contraposée, on est ramené à montrer  $A[x \leftarrow u] \Rightarrow A[x \leftarrow t]$ . Comme une interprétation qui rend vraie  $t = u$  rend aussi vraie  $u = t$  par symétrie de l'égalité, on peut appliquer l'hypothèse de récurrence à  $A$  avec les termes  $u$  et  $t$ .
- Si  $P$  est de la forme  $A \circ B$  avec  $\circ$  l'un des 2 connecteurs propositionnels  $\wedge$  ou  $\vee$ , alors il faut  $(A \circ B)[x \leftarrow t] \Rightarrow (A \circ B)[x \leftarrow u]$ , c'est-à-dire  $(A[x \leftarrow t] \circ B[x \leftarrow t]) \Rightarrow (A[x \leftarrow u] \circ B[x \leftarrow u])$  or par hypothèse de récurrence on  $A[x \leftarrow t] \Rightarrow A[x \leftarrow u]$  et  $B[x \leftarrow t] \Rightarrow B[x \leftarrow u]$ . On conclut en utilisant les propriétés des connecteurs (si  $A_1 \Rightarrow A_2$  et  $B_1 \Rightarrow B_2$  alors  $A_1 \wedge B_1 \Rightarrow A_2 \wedge B_2$  et  $A_1 \vee B_1 \Rightarrow A_2 \vee B_2$ ).
- Si  $P$  est de la forme  $A \Rightarrow B$ , le raisonnement est presque le même sauf que la propriété  $(A_1 \Rightarrow B_1) \Rightarrow (A_2 \Rightarrow B_2)$  est une conséquence de  $A_2 \Rightarrow A_1$  et  $B_1 \Rightarrow B_2$ . Il faut donc intervertir  $t$  et  $u$  comme dans le cas de la négation.
- Si  $P$  est de la forme  $\forall z, A$ . On choisit la variable liée  $z$  pour qu'elle ne soit pas libre ni dans  $t$  ni dans  $u$  (sinon la substitution n'est pas définie) et qu'elle soit différente de  $x$  (sinon  $(\forall x, A)[x \leftarrow v] = (\forall x, A)$  et la propriété est trivialement vraie.  
On doit montrer  $(\forall z, A)[x \leftarrow t] \Rightarrow (\forall z, A)[x \leftarrow u]$ , c'est-à-dire  $(\forall z, (A[x \leftarrow t])) \Rightarrow \forall z, (A[x \leftarrow u])$  Par hypothèse de récurrence on sait que  $(A[x \leftarrow t]) \Rightarrow (A[x \leftarrow u])$  quelle que soit la valeur que l'on donne pour  $z$  dans l'environnement. On en déduit que  $(\forall z, (A[x \leftarrow t])) \Rightarrow (A[x \leftarrow u])$  est vraie pour toute valeur possible de  $z$ . En effet si on prend une valeur pour  $z$  et que  $\forall z, (A[x \leftarrow t])$  est vrai alors  $A[x \leftarrow t]$  est a fortiori vrai pour cette valeur de  $z$  et donc  $A[x \leftarrow u]$ . De  $(\forall z, (A[x \leftarrow t])) \Rightarrow (A[x \leftarrow u])$  vrai pour toutes les valeurs de  $z$ , on déduit que  $(\forall z, (A[x \leftarrow t])) \Rightarrow \forall z, (A[x \leftarrow u])$  est vrai.  
On procède de manière analogue pour le quantificateur existentiel. Si  $(A[x \leftarrow t]) \Rightarrow (A[x \leftarrow u])$  est vrai pour toute valeur de  $z$  alors  $(\exists z, (A[x \leftarrow t])) \Rightarrow \exists z, (A[x \leftarrow u])$  est vrai. En effet si  $\exists z, (A[x \leftarrow t])$  est vrai alors il existe une valeur  $v_0$  telle que  $A[x \leftarrow t]$  est vraie pour cette valeur de  $z$ , donc  $A[x \leftarrow u]$  est vraie pour cette même valeur de  $z$  (hypothèse de récurrence) et donc  $\exists z, A[x \leftarrow u]$  est vraie.

## A.2 Exercices du chapitre 2

**Exercice 2.1** Dans chacune des interprétations (qui fixent le domaine et sur ce domaine le sens des symboles  $\leq$ ,  $<$  et  $\times$ ), chaque formule a une valeur de vérité  $V$  ou  $F$  qu'il convient de justifier.

1.  $\forall x, \exists y, x < y$ 
  - Cette formule est vraie dans les trois interprétations  $\mathbb{N}$ ,  $\mathbb{Z}$  et  $\mathbb{Q}$ . En effet si on se donne n'importe quelle valeur  $n$  pour  $x$ , il suffit de choisir la valeur  $n + 1$  pour  $y$ . Le résultat découle du fait que  $n < n + 1$  dans les trois interprétations.
2.  $\forall x, \exists y, y < x$ 
  - Cette formule est fausse dans l'interprétation  $\mathbb{N}$ . En effet si on choisit la valeur  $0$  pour  $x$ , quelle que soit  $n \in \mathbb{N}$  pour  $y$ , la propriété  $n < 0$  est toujours fausse.
  - Cette formule est vraie dans les deux interprétations  $\mathbb{Z}$  et  $\mathbb{Q}$ . En effet si on se donne n'importe quelle valeur  $n$  pour  $x$ , il suffit de choisir la valeur  $n - 1$  pour  $y$ . Le résultat découle du fait que  $n - 1 < n$  dans les deux interprétations.
3.  $\forall x y, x < y \Rightarrow x + 1 \leq y$ 
  - Cette formule est vraie dans les deux interprétations  $\mathbb{N}$  et  $\mathbb{Z}$ . En effet si on se donne n'importe quelles valeurs  $n$  et  $m$  pour  $x$  et  $y$ , si  $x < y$  est vrai alors  $n < m$  et donc  $0 < m - n \in \mathbb{N}$ . Un entier strictement positif est supérieur ou égal à 1, donc  $1 \leq m - n$  donc  $n + 1 \leq m$ . Ceci établit donc le résultat.
  - Cette formule est fausse dans l'interprétation  $\mathbb{Q}$ . En effet si on choisit la valeur  $0$  pour  $x$ , et la valeur  $0,5$  pour  $y$ , on a bien  $x < y$  qui est vrai car  $0 < 0,5$  mais  $x + 1 \leq y$  est faux car correspond  $1 \leq 0,5$ .
4.  $\forall x y z, x \leq y \Rightarrow x \times z \leq y \times z$ 
  - Cette formule est vraie dans l'interprétation  $\mathbb{N}$ . En effet si on se donne n'importe quelles valeurs  $n$  et  $m$  et  $p$  dans  $\mathbb{N}$  pour  $x$ ,  $y$  et  $z$ , si  $x \leq y$  est vrai dans cet environnement alors  $n \leq m$ , comme  $p \in \mathbb{N}$  (donc positif), on a  $n \times p \leq m \times p$  et donc  $x \times z \leq y \times z$  est vrai ce qui établit le résultat.
  - Cette formule est fausse dans les interprétations  $\mathbb{Z}$  et  $\mathbb{Q}$ . En effet si on choisit la valeur  $0$  pour  $x$ , la valeur  $1$  pour  $y$  et la valeur  $-1$  pour  $z$ . On a bien  $x \leq y$  est vrai car  $0 \leq 1$ . Mais  $x \times z \leq y \times z$  est faux car correspond à  $0 \leq -1$ .

### Exercice 2.2

- La formule  $(\exists x, \neg P(x)) \Rightarrow \neg \forall x, P(x)$  est valide. On peut justifier cela avec les lois de de Morgan, car  $\neg \forall x, P(x) \equiv \exists x, \neg P(x)$  et donc la formule est un cas particulier de  $A \Rightarrow A$  qui est une tautologie. On peut aussi montrer le résultat en revenant à la définition des interprétations.  
Si on se donne une interprétation  $I$  sur un domaine  $\mathcal{D}$  telle que  $\exists x, \neg P(x)$  est vrai dans cette interprétation alors il existe  $d \in \mathcal{D}$ , tel que  $\neg P(x)$  est vrai dans cette interprétation et dans l'environnement dans lequel la variable  $x$  a la valeur  $d$ . On a donc  $P(x)$  faux dans l'interprétation  $I$  et le même environnement. On en déduit que  $\forall x, P(x)$  faux dans l'interprétation  $I$  et donc  $\neg \forall x, P(x)$  est vrai.  
Si on part de la formule  $\exists x, \neg P(x) \Rightarrow \neg \forall x, P(x)$  qui se parenthèse comme  $\exists x, (\neg P(x) \Rightarrow \neg \forall x, P(x))$ , le résultat est le même. Cette formule est valide. On peut le montrer en utilisant le fait que  $\exists x, (\neg P(x) \Rightarrow \neg \forall x, P(x)) \equiv (\forall x, \neg P(x)) \Rightarrow \neg \forall x, P(x)$  et que si  $\forall x, \neg P(x)$  est vrai alors a fortiori  $\exists x, \neg P(x)$  est vrai puis se ramener au cas précédent. On peut aussi faire un raisonnement direct avec une interprétation  $I$  sur un domaine  $\mathcal{D}$ . On choisit n'importe quel  $d \in \mathcal{D}$  (possible car  $\mathcal{D} \neq \emptyset$ ). On doit montrer que  $\neg P(x) \Rightarrow \neg \forall x, P(x)$  est vraie dans l'interprétation  $I$  et dans l'environnement dans lequel la variable  $x$  a la valeur  $d$ . On conclut de la même manière que précédemment.
- La formule  $(\exists x, \neg P(x)) \Rightarrow \exists x, \neg Q(x)$  est satisfiable. Il suffit de prendre une interprétation sur un domaine  $\mathcal{D}$  formé d'un seul élément  $a$  et telle que l'interprétation de  $Q$  est l'ensemble vide. On a donc que  $Q(x)$  est faux dans l'environnement dans lequel la variable  $x$  a la valeur  $a$ , donc  $\neg Q(x)$  est vraie et donc aussi  $\exists x, \neg Q(x)$  et finalement  $(\exists x, \neg P(x)) \Rightarrow \exists x, \neg Q(x)$ .

La formule  $(\exists x, \neg P(x)) \Rightarrow \exists x, \neg Q(x)$  est non valide. Il faut pour cela trouver une interprétation qui rend fausse la formule, donc qui rend vraie  $\exists x, \neg P(x)$  et fausse  $\exists x, \neg Q(x)$ . On peut à nouveau prendre une interprétation sur un domaine  $\mathcal{D}$  formé d'un seul élément  $a$  et telle que l'interprétation de  $P$  est l'ensemble vide et l'interprétation de  $Q$  est l'ensemble  $\mathcal{D}$  tout entier.

Le résultat est le même avec le parenthésage  $\exists x, (\neg P(x) \Rightarrow \exists x, \neg Q(x))$ , la formule est satisfiable mais non valide.

- La formule  $(\exists x, \neg P(x)) \wedge \forall x, P(x)$  est insatisfiable. Il faut montrer qu'elle est fausse dans toutes les interprétations. Pour cela on se donne une interprétation  $I$  sur un domaine  $\mathcal{D}$  quelconque. Si la formule était vraie dans cette interprétation, on aurait  $\exists x, \neg P(x)$  et donc on aurait que  $\neg P(x)$  est vrai dans l'interprétation  $I$  et dans l'environnement dans lequel la variable  $x$  a la valeur  $d$ . On aurait donc que  $P(x)$  est faux dans l'interprétation  $I$  et dans l'environnement dans lequel la variable  $x$  a la valeur  $d$  ce qui contredit le fait que  $\forall x, P(x)$  est vrai.

On peut aussi utiliser les lois de de Morgan  $\neg \forall x, P(x) \equiv \exists x, \neg P(x)$  ce qui montre que la formule est équivalente à  $(\neg \forall x, P(x)) \wedge (\forall x, P(x))$  qui est une instance de la formule propositionnelle  $\neg A \wedge A$  qui est insatisfiable.

Le résultat est le même avec le parenthésage  $\exists x, (\neg P(x) \wedge \forall x, P(x))$ , on peut alors commencer par appliquer l'équivalence  $\exists x, (\neg P(x) \wedge \forall x, P(x)) \equiv (\exists x, \neg P(x)) \wedge (\forall x, P(x))$  car  $\forall x, P(x)$  ne dépend pas de  $x$ , ce qui nous ramène au cas précédent.

**Exercice 2.3** On a  $\perp \Rightarrow P \equiv \top$ ,  $\top \Rightarrow P \equiv P$ ,  $P \Rightarrow \top \equiv \top$ . On peut justifier par des tables de vérité ou bien utiliser  $A \Rightarrow B \equiv \neg A \vee B$  et utiliser les lois connues pour les connecteurs  $\perp$ ,  $\top$  et  $\vee$ .

**Exercice 2.4** On suppose que  $x \notin \text{VI}(A)$ .

- $\forall x, (A \Rightarrow H(x)) \equiv \forall x, (\neg A \vee H(x)) \equiv \neg A \vee \forall x, H(x) \equiv A \Rightarrow \forall x, H(x)$
- $\exists x, (A \Rightarrow H(x)) \equiv \exists x, (\neg A \vee H(x)) \equiv \neg A \vee \exists x, H(x) \equiv A \Rightarrow \exists x, H(x)$
- $\forall x, (G(x) \Rightarrow A) \equiv \forall x, (\neg G(x) \vee A) \equiv (\forall x, \neg G(x)) \vee A \equiv \neg(\exists x, G(x)) \vee A \equiv (\exists x, G(x)) \Rightarrow A$
- $\exists x, (G(x) \Rightarrow A) \equiv \exists x, (\neg G(x) \vee A) \equiv (\exists x, \neg G(x)) \vee A \equiv \neg(\forall x, G(x)) \vee A \equiv (\forall x, G(x)) \Rightarrow A$

**Exercice 2.5** Trouver des interprétations qui justifient les résultats suivants :

- $\forall x, \exists y, R(x, y) \not\equiv \exists y, \forall x, R(x, y)$   
On a  $\exists y, \forall x, R(x, y) \models \forall x, \exists y, R(x, y)$  mais le sens inverse est faux  $\forall x, \exists y, R(x, y) \not\models \exists y, \forall x, R(x, y)$ . Pour montrer ce résultat, il suffit de trouver une interprétation dans laquelle  $\exists y, \forall x, R(x, y)$  est vraie et  $\forall x, \exists y, R(x, y)$  est fausse.  
Il suffit de prendre une interprétation avec un domaine qui contient deux éléments 0 et 1 avec pour la relation  $R$  la relation d'égalité. On a  $x = x$  et  $0 \neq 1$
- $\forall x, (G(x) \vee H(x)) \not\equiv (\forall x, G(x)) \vee (\forall x, H(x))$   
On a  $(\forall x, G(x)) \vee (\forall x, H(x)) \models \forall x, (G(x) \vee H(x))$  mais le sens inverse est faux  $\forall x, (G(x) \vee H(x)) \not\models (\forall x, G(x)) \vee (\forall x, H(x))$   
Pour montrer ce résultat, il suffit de trouver une interprétation dans laquelle  $\forall x, (G(x) \vee H(x))$  est vraie et  $(\forall x, G(x)) \vee (\forall x, H(x))$  est fausse.  
Il suffit de prendre une interprétation avec un domaine qui contient deux éléments 0 et 1 et dans laquelle  $G$  est vraie pour 0 et faux pour 1 et  $H$  est faux pour 0 et vraie pour 1. Dans cette interprétation on a bien  $\forall x, (G(x) \vee H(x))$  est vrai mais  $\forall x, G(x)$  et  $\forall x, H(x)$  sont faux.
- $\exists x, (G(x) \wedge H(x)) \not\equiv (\exists x, G(x)) \wedge (\exists x, H(x))$   
On a  $\exists x, (G(x) \wedge H(x)) \models (\exists x, G(x)) \wedge (\exists x, H(x))$  mais le sens inverse est faux  $(\exists x, G(x)) \wedge (\exists x, H(x)) \not\models \exists x, (G(x) \wedge H(x))$   
On prend la même interprétation que dans le cas précédent. On a bien  $\exists x, G(x)$  est vrai (prendre  $x \mapsto 0$  et  $\exists x, H(x)$  est vrai (prendre  $x \mapsto 1$ ) et donc leur conjonction est vraie. Par contre  $\exists x, (G(x) \wedge H(x))$  est faux car aucun des éléments du domaine ne rend vraie la formule.



**Exercice 2.6** Le langage contient  $n$  constantes  $\{c_1; \dots; c_n\}$ . On note  $I, \rho \models A$  lorsque la formule  $A$  est vraie dans l'interprétation  $I$  et l'environnement  $\rho$ , c'est-à-dire lorsque  $\text{val}_I(\rho, A) = V$ . On s'intéresse aux interprétations dans lesquelles le symbole d'égalité est interprété par l'égalité dans le domaine. C'est-à-dire que  $I, \rho \models t = u$  si et seulement si  $\text{val}_I(\rho, t) = \text{val}_I(\rho, u)$

1. Le cardinal du domaine d'une interprétation qui rend vraie la formule  $\forall x, x = c_1 \vee \dots \vee x = c_n$  est inférieur ou égal à  $n$ . En effet la formule dit que tout élément du domaine est égal à l'interprétation d'une des constantes  $c_i$  ce qui fait qu'il y a au plus  $n$  valeurs possibles. Par contre rien n'empêche deux constantes différentes d'être interprétées par la même valeur dans le domaine, donc on peut avoir strictement moins de  $n$  éléments.

Pour justifier le résultat plus formellement, on se donne une interprétation  $I$  sur un domaine  $D$  et on appelle  $C$  l'ensemble des interprétations des constantes  $\{c_1; \dots; c_n\}$ , à savoir  $C = \{\text{val}_I(c_i) \mid 1 \leq i \leq n\}$ . On a  $C \subseteq D$  et  $|C| \leq n$ . On va montrer que  $D = C$ . Soit donc  $d \in D$ . Comme  $I \models \forall x, x = c_1 \vee \dots \vee x = c_n$  on a  $I, \{x \mapsto d\} \models x = c_1 \vee \dots \vee x = c_n$  et donc comme une disjonction est vraie lorsque l'une des parties est vraie, il existe  $i$  tel que  $I, \{x \mapsto d\} \models x = c_i$  et donc  $d = \text{val}_I(\{x \mapsto d\}, x) = \text{val}_I(\{x \mapsto d\}, c_i)$  et donc  $d \in C$ .

Les deux ensembles étant égaux ont même cardinal qui est inférieur à  $n$ .

2. Montrons que  $(\forall x, x = c_1 \vee \dots \vee x = c_n), (P(c_1) \wedge \dots \wedge P(c_n)) \models \forall x, P(x)$

On se donne une interprétation  $I$  de l'égalité et telle que  $I \models \forall x y, x = y \Rightarrow (P(x) \Rightarrow P(y))$ ,  $I \models \forall x, x = c_1 \vee \dots \vee x = c_n$  et  $I \models P(c_1) \wedge \dots \wedge P(c_n)$ . Soit  $d \in D$ . On a  $I, \{x \mapsto d\} \models x = c_1 \vee \dots \vee x = c_n$  donc il existe  $i$  tel que  $I, \{x \mapsto d\} \models x = c_i$ , on a aussi  $I, \{x \mapsto d\} \models P(c_i)$  on en déduit que  $I, \{x \mapsto d\} \models P(x)$  et donc  $I \models \forall x, P(x)$ .

On pourrait faire le même raisonnement pour l'existentielle. On peut aussi réutiliser le résultat précédent. On peut remplacer  $P(x)$  par  $\neg P(x)$  en utilisant la propriété 2.3.2 de remplacement d'un symbole de prédicat par une formule paramétrée. On a donc  $(\forall x, x = c_1 \vee \dots \vee x = c_n), (\neg P(c_1) \wedge \dots \wedge \neg P(c_n)) \models \forall x, \neg P(x)$

On utilise ensuite le fait que pour tout ensemble de formule  $\mathcal{E}$  et deux formules  $A$  et  $B$ ,  $\mathcal{E}, A \models B$  est équivalent à  $\mathcal{E}, \neg B \models \neg A$  (on peut utiliser le fait que  $\mathcal{E}, A \models B$  est équivalent à  $\mathcal{E} \models A \Rightarrow B$  et la propriété de contraposée  $A \Rightarrow B \equiv \neg B \Rightarrow \neg A$ ).

### Exercice 2.7

- Il suffit d'introduire une constante  $c$  et de prendre comme formule  $A$  la formule  $\forall x, x = c$
- On a (cas particulier de l'exercice précédent)  $\forall x, x = c \models (\forall x, P(x)) \Leftrightarrow P(c)$  et  $\forall x, x = c \models (\exists x, P(x)) \Leftrightarrow P(c)$  on en déduit que  $\forall x, x = c \models (\forall x, P(x)) \Leftrightarrow (\exists x, P(x))$
- S'il y a deux éléments  $a, b$  dans le domaine, on choisit une interprétation  $I$  telle que  $I, \{x \mapsto a\} \models P(x)$  et  $I, \{x \mapsto b\} \not\models P(x)$ . On a bien  $I \models \exists x, P(x)$  mais  $I \not\models \forall x, P(x)$

**Exercice 2.8** Soit la formule  $\forall x y, (P(x) \wedge Q(y) \wedge (\neg P(a) \vee \neg Q(a)))$ . Le langage des termes ne contient que la constante  $a$ , celui des prédicats contient les deux symboles  $P$  et  $Q$ .

- Le domaine de Herbrand est l'ensemble des termes clos, ici réduit à la constante  $a$ .
- La base de Herbrand est l'ensemble des formules atomiques closes, ici réduite à deux formules  $P(a)$  et  $Q(a)$ .
- Il y a quatre interprétations possibles suivant que  $P(a)$  et  $Q(a)$  valent vrai ou faux.
- La formule est universelle, elle est satisfiable si et seulement si l'ensemble de ses instances closes est satisfiable. Ici il y a une seule instance close  $P(a) \wedge Q(a) \wedge (\neg P(a) \vee \neg Q(a))$ . On peut faire une table



de vérité

$P(a)$	$Q(a)$	$P(a) \wedge Q(a) \wedge (\neg P(a) \vee \neg Q(a))$
$F$	$\neg$	$F$
$V$	$F$	$F$
$V$	$V$	$F$

La formule est fausse dans tous les modèles de Herbrand, elle est universelle et donc, d'après le théorème de Herbrand, elle est fausse dans n'importe quelle interprétation et elle est donc insatisfiable.

### Exercice 2.9

- La formule  $(\exists x, P(x)) \Rightarrow \forall x, P(x)$  n'est pas valide (contre exemple avec un modèle à deux éléments)
- La négation de cette formule, notée  $B$  est  $(\exists x, P(x)) \wedge \exists x, \neg P(x)$
- Il n'y a pas de constante dans le langage, on en ajoute une  $a$  et le domaine de Herbrand est réduit à cette constante  $a$ . La base de Herbrand contient une formule atomique  $P(a)$  et il y a donc deux interprétations possibles suivant que  $P(a)$  est vraie ou fausse.
- Il n'y a pas de modèle de Herbrand qui rend vraie la formule  $B$ .
- On ne peut pas appliquer le théorème de Herbrand car la formule n'est pas universelle. D'ailleurs la formule est satisfiable (on peut construire un modèle avec deux éléments ou dire que si elle était satisfiable alors sa négation serait valide, ce qui n'est pas le cas comme montré à la question 1).

### A.3 Exercices du chapitre 3

**Exercice 3.1** La fonction `clauseset` prend pour argument un ensemble d'instances de prédicats associés à des booléens et renvoie une formule de la logique correspondant à la clause associée.

```
clauseset(s) = si s = ∅ alors ⊥
              sinon soit (p, b) ∈ s, s' = s \ (p, b), A = si b alors p sinon ¬p
                  dans si s' = ∅ alors A sinon A ∨ clauseset(s')
```

**Exercice 3.2** L'argument étant déjà en forme normale de négation il n'y a pas de symbole d'implication ni de négation autre que dans un littéral. On peut donc simplifier la fonction `fnc`.

```
fnn2fnc(⊥) = {⊥}
fnn2fnc(⊤) = ∅
fnn2fnc(l) = {l} l littéral
fnn2fnc(P ∧ Q) = fnn2fnc(P) ∪ fnn2fnc(Q)
fnn2fnc(P ∨ Q) = sh(fnn2fnc(P), fnn2fnc(Q))
```

**Exercice 3.3** Toute formule  $P$  peut se ramener à un ensemble de clauses  $\mathcal{E}$  (équivalent ou équisatisfiable).

Tout modèle de la formule initiale est (ou peut s'étendre) en un modèle de chacune des clauses de  $\mathcal{E}$ .

On va d'abord montrer que l'on peut transformer une clause  $C$  de  $\mathcal{E}$  en un ensemble de clauses à 3 littéraux équisatisfiable.

Si une clause  $C$  a strictement plus de trois littéraux alors elle s'écrit  $l_1 \vee l_2 \vee D$  avec  $l_1, l_2$  des littéraux et  $D$  une clause qui a deux littéraux de moins que  $C$ .

On introduit une nouvelle variable propositionnelle  $x_D$ , on transforme la clause  $C$  en l'ensemble de clauses  $\{l_1 \vee l_2 \vee x_D, \neg x_D \vee D\}$ . La clause  $l_1 \vee l_2 \vee x_D$  a trois littéraux et la clause  $\neg x_D \vee D$  a strictement moins de littéraux que  $C$  ce qui permet d'itérer le processus jusqu'à qu'il ne reste plus que trois littéraux. Si on a un modèle de  $\{l_1 \vee l_2 \vee x_D, \neg x_D \vee D\}$ , alors suivant la valeur de  $x_D$  dans ce modèle on aura que  $l_1 \vee l_2$  ou  $D$  est vrai et donc  $l_1 \vee l_2 \vee D$  est vrai. Dans le sens inverse, s'il existe un modèle de  $l_1 \vee l_2 \vee D$  alors il suffit de choisir pour valeur de  $x_D$  la valeur de la clause  $D$  dans cette interprétation pour obtenir un modèle de  $\{l_1 \vee l_2 \vee x_D, \neg x_D \vee D\}$ .

On a ainsi traité une clause de la forme clausale de  $P$ , il faut justifier que si on met à plat tous les ensembles de clauses à trois littéraux obtenues pour toutes les clauses de  $\mathcal{E}$ , on a toujours un ensemble de clauses équisatisfiable. Cela découle du fait qu'on introduit à chaque fois des variables nouvelles et qu'on ne modifie pas l'interprétation des symboles de la formule originale, les modèles trouvés pour chacun des ensembles de clauses à trois littéraux peuvent ainsi se recoller pour donner un modèle de l'ensemble complet de clauses.

**Exercice 3.4**  $A \stackrel{\text{def}}{=} (\exists x, \forall y, R(x, y)) \Rightarrow (\forall y, \exists x, R(x, y))$ .

- La forme normale de négation de la formule  $\neg A$  est  $(\exists x, \forall y, R(x, y)) \wedge (\exists y, \forall x, \neg R(x, y))$
- La forme de skolem de la formule  $\neg A$  introduit deux constantes  $a$  et  $b$   $(\forall y, R(a, y)) \wedge \forall x, \neg R(x, b)$
- La forme clausale de  $\neg A$  a deux clauses  $\{R(a, y); \neg R(x, b)\}$
- L'ensemble des instances closes dans le modèle de Herbrand contient  $\{R(a, b); \neg R(a, b)\}$  qui est insatisfiable. La formule  $\neg A$  est donc insatisfiable et la formule  $A$  est valide.

#### Exercice 3.5

1. On introduit tout d'abord la fonction `form` qui étant donné un arbre de décision binaire (BDT) calcule une formule en forme normale de négation qui représente la même fonction booléenne.

$$\begin{aligned} \text{form}(V) &= \top \quad \text{form}(F) = \perp \\ \text{form}(\mathbf{IF}(x, P, Q)) &= (x \wedge \text{form}(P)) \vee (\neg x \wedge \text{form}(Q)) \end{aligned}$$

On peut alternativement utiliser la définition  $\text{form}(\mathbf{IF}(x, P, Q)) = (\neg x \vee \text{form}(P)) \wedge (x \vee \text{form}(Q))$ . On transforme cette fonction en une nouvelle fonction `formc` qui calcule directement la forme normale conjonctive sous la forme d'un ensemble de clauses.

$$\begin{aligned} \text{formc}(V) &= \emptyset \quad \text{formc}(F) = \{\perp\} \\ \text{formc}(\mathbf{IF}(x, P, Q)) &= \{\neg x \vee C \mid C \in \text{formc}(P)\} \cup \{x \vee C \mid C \in \text{formc}(Q)\} \end{aligned}$$

2. La fonction `notd` se définit simplement par un parcours de l'arbre qui se contente de remplacer les feuilles par leur négation.

$$\begin{aligned} \text{notd}(V) &= F \quad \text{notd}(F) = V \\ \text{notd}(\mathbf{IF}(x, P, Q)) &= \mathbf{IF}(x, \text{notd}(P), \text{notd}(Q)) \end{aligned}$$

On vérifie simplement par récurrence sur la structure de l'arbre qu'on a bien le bon comportement à savoir que pour un arbre  $t$  et une interprétation  $I$  quelconque, on a  $I \models \text{notd}(t)$  si et seulement si  $I \not\models t$ . C'est évident pour les feuilles. Dans le cas d'une construction  $\mathbf{IF}(x, P, Q)$ , il suffit de distinguer les cas où  $I(x) = V$  et  $I(x) = F$  et de conclure par hypothèse de récurrence.

3. Pour implanter la conjonction de deux BDT, la principale difficulté est que les deux arbres ne sont pas forcément construits sur la même séquence de variables. La fonction doit donc réconcilier deux branches en intercalant possiblement des variables qui seraient présentes sur une seule des branches et en préservant la contrainte de variables ordonnées de manière croissante sur chaque branche.

$$\begin{aligned} \text{conj}(b_1, b_2) &= b_1 \text{ et } b_2 \\ \text{conj}(\mathbf{IF}(x, P_1, Q_1), \mathbf{IF}(y, P_2, Q_2)) &= \mathbf{IF}(x, \text{conj}(P_1, P_2), \text{conj}(Q_1, Q_2)) \quad \text{si } x = y \\ \text{conj}(\mathbf{IF}(x, P, Q), R) &= \mathbf{IF}(x, \text{conj}(P, R), \text{conj}(Q, R)) \quad \text{si vars de } R \text{ plus grandes que } x \\ \text{conj}(R, \mathbf{IF}(y, P, Q)) &= \mathbf{IF}(y, \text{conj}(R, P), \text{conj}(R, Q)) \quad \text{si vars de } R \text{ plus grandes que } y \end{aligned}$$

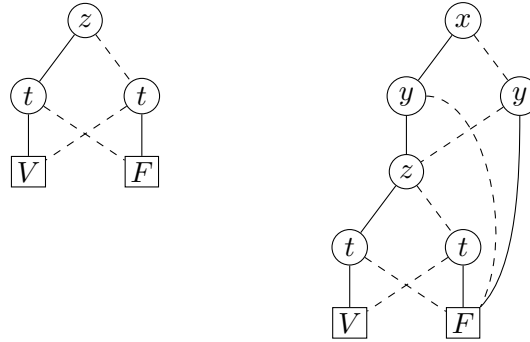
Les deux derniers cas couvrent également la situation dans laquelle  $R$  est une feuille (pas de variable). La terminaison de cette définition est justifiée par le fait que le nombre de symboles  $\mathbf{IF}$  est strictement plus petit dans les arguments des appels à la fonction `conj` à droite.

4. Le BDT d'une formule valide n'a que des feuilles  $V$ , celui d'une formule insatisfiable n'a que des feuilles  $F$  et une formule satisfiable aura au moins une feuille  $V$ .
5. Pour trouver un modèle, on parcourt l'arbre pour chercher une feuille avec la valeur  $V$ . On renvoie `insat` si la formule est insatisfiable et sinon une interprétation (sous forme de liste de couples variable/valeur).

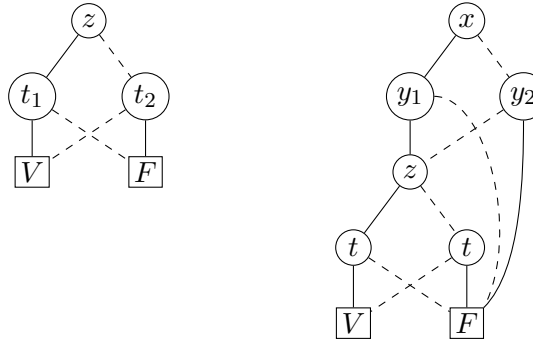
$$\begin{aligned} \text{mod}(V) &= [] \quad \text{mod}(F) = \text{insat} \\ \text{mod}(\mathbf{IF}(x, P, Q)) &= (x, V) :: \text{mod}(P) \quad \text{si } \text{mod}(P) \neq \text{insat} \\ \text{mod}(\mathbf{IF}(x, P, Q)) &= (x, F) :: \text{mod}(Q) \quad \text{si } \text{mod}(P) = \text{insat et } \text{mod}(Q) \neq \text{insat} \\ \text{mod}(\mathbf{IF}(x, P, Q)) &= \text{insat} \quad \text{si } \text{mod}(P) = \text{insat et } \text{mod}(Q) = \text{insat} \end{aligned}$$

**Exercice 3.6**

1. L'ordre sur les variables est  $x < y < z < t$ . Les OBDDs des formules  $(z \Leftrightarrow t)$  et  $(x \Leftrightarrow y) \wedge (z \Leftrightarrow t)$  sont



2. On associe à chaque sommet du graphe une formule équivalente au sous-graphe issu de ce sommet qui n'utilise que des littéraux et les symboles  $\vee$  et  $\wedge$ .



Formules associées à chaque sommet :

- La formule associée à la feuille  $V$  est  $\top$  et à la feuille  $F$  est  $\perp$
- si un nœud est étiqueté par la variable  $x$  dont le fils vrai correspond à la formule  $A$  et le fils faux à la formule  $B$  alors la formule associée au nœud  $x$  est  $x \wedge A \vee \neg x \wedge B$  qui peut parfois se simplifier

$$\begin{aligned}
 t_1 &: t \\
 t_2 &: \neg t \\
 z &: (z \wedge t) \vee (\neg z \wedge \neg t) \\
 y_1 &: y \wedge ((z \wedge t) \vee (\neg z \wedge \neg t)) \\
 y_2 &: \neg y \wedge ((z \wedge t) \vee (\neg z \wedge \neg t)) \\
 x &: (x \wedge y \wedge ((z \wedge t) \vee (\neg z \wedge \neg t))) \vee (\neg x \wedge \neg y \wedge ((z \wedge t) \vee (\neg z \wedge \neg t)))
 \end{aligned}$$

3. Deux formules sont équivalentes si et seulement si elles sont représentées par le même OBDD.

Si deux formules ont le même OBDD, elles ont a fortiori la même table de vérité et sont donc équivalentes.

Dans l'autre sens, on montre que deux OBDD  $t$  et  $u$  différents ne sont pas associés à des formules équivalentes, on raisonne par récurrence sur le nombre de variables propositionnelles qui apparaissent dans les OBDD.

S'il n'y a pas de variable alors on a deux valeurs booléennes qui sont différentes et donc les formules initiales ne sont pas équivalentes. Si l'un des OBDD n'est pas une valeur booléenne, on considère  $x$  la plus petite variable, on peut supposer que c'est  $t$  qui débute par la variable  $x$ . Si  $x$  n'apparaît pas dans  $u$  alors cela veut dire que la formule  $Q$  associée à  $u$  ne dépend pas de  $x$ . Par contre les deux OBDD fils  $n$  et  $m$  de  $x$  dans  $t$  sont différents. Par hypothèse de récurrence (moins de variables) ils ne sont pas

équivalents donc il existe une interprétation  $I$  qui rend vraie l'une et fausse l'autre. On regarde la valeur de  $u$  dans cette interprétation, elle est différente soit de l'interprétation de  $n$ , soit de l'interprétation de  $m$  et il suffit de fixer  $x$  à la valeur correspondante pour trouver une interprétation dans laquelle les deux formules diffèrent.

Dans le cas où  $x$  apparaît dans  $u$  alors il est forcément en tête (plus petite variable) et si les OBDD  $t$  et  $u$  sont différents cela signifie qu'ils n'ont pas le même fils vrai ou pas le même fils faux. Dans les deux cas par récurrence on trouve une interprétation qui ne donne pas la même valeur et on l'étend avec la bonne valeur de  $x$ .

On peut ainsi simplement caractériser la validité et satisfiabilité d'une formule

- une formule valide a un OBDD qui est réduit à une feuille  $V$ ,
- une formule insatisfiable a un OBDD qui est réduit à une feuille  $F$ .
- tout OBDD qui n'est pas juste une feuille  $F$  correspond à une formule satisfiable
- un modèle de la formule est donné par n'importe quelle branche qui mène jusqu'à une feuille  $V$ .

4. (a) partage maximal (nœud et feuilles), il n'y a pas deux sommets différents qui ont les mêmes caractéristiques : pour tous sommets  $n$  et  $m$ ,
  - si  $\text{noeud}(G, n)$  et  $\text{noeud}(G, m)$  et  $\text{var}(G, n) = \text{var}(G, m)$  et  $\text{vrai}(G, n) = \text{vrai}(G, m)$  et  $\text{faux}(G, n) = \text{faux}(G, m)$  alors  $n = m$
  - si  $\text{feuille}(G, n)$  et  $\text{feuille}(G, m)$  et  $\text{valeur}(G, n) = \text{valeur}(G, m)$  alors  $n = m$
- (b) absence de tests inutiles : pour tout sommet  $n$ ,  
si  $\text{noeud}(G, n)$  alors  $\text{vrai}(G, n) \neq \text{faux}(G, n)$
- (c) ordre sur les variables : pour tout sommet  $n$ ,  
si  $\text{noeud}(G, n)$  et  $\text{noeud}(G, \text{vrai}(G, n))$  alors  $\text{var}(G, n) < \text{var}(G, \text{vrai}(G, n))$   
si  $\text{noeud}(G, n)$  et  $\text{noeud}(G, \text{faux}(G, n))$  alors  $\text{var}(G, n) < \text{var}(G, \text{faux}(G, n))$

5. La fonction naïve a la forme suivante

```

form(G, n) =
  si feuille(G, n) alors si valeur(G, n) alors T sinon ⊥
  sinon soit x = var(G, n)
    soit P = form(vrai(G, n))
    soit Q = form(faux(G, n))
    (¬ x ∨ P) ∧ (x ∨ Q)

```

Pour justifier de la terminaison on remarque que la plus petite variable présente dans le graphe traité augmente strictement dans le cas d'un appel et donc qu'on aboutit toujours à une feuille.

Pour exploiter le partage, il faut utiliser des techniques de programmation dynamique en stockant dans une table les résultats déjà calculés pour chaque sommet. Si la valeur a déjà été calculée, on la renvoie sinon on la calcule comme précédemment et on l'enregistre avant de renvoyer la valeur.

6.  $G$  est le graphe courant,  $x$  une variable propositionnelle,  $n$  et  $m$  sont deux sommets de  $G$ . On veut calculer un nouveau graphe  $G'$  et un nœud  $p$  tel que  $\text{form}(G', p) \equiv \mathbf{IF}(x, \text{form}(G, n), \text{form}(G, m))$  et  $\text{form}(G', k) = \text{form}(G, k)$  pour tous les sommets de  $G$ . On utilise une table  $T(G)$  qui permet de retrouver si une formule est déjà représentée dans le graphe : soit une variable  $y$  et deux sommets  $n$  et  $m$ ,  $(x, n, m) \mapsto p \in T(G)$  ssi le graphe  $G$  a un sommet  $p$  étiqueté par la variable  $x$  dont le fils "vrai" est le sommet  $n$  et le fils "faux" est le sommet  $m$ .

```

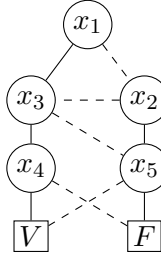
if(G, x, n, m) =
  si (x, n, m) ↦ p dans T alors (G, p)
  sinon si n=m alors (G, n)
  sinon soit p = nouveau sommet

```

soit  $G' = G + p$  /  $\text{var}(p)=x$ ,  $\text{vrai}(p)=n$ ,  $\text{faux}(p)=m$   
 ajouter  $(x,n,m) \mapsto p$  dans  $T$   
 $(G', p)$

7. On veut compter le nombre de modèles d'une formule représentée par un OBDD.

(a) Soit l'OBDD suivant sur les variables propositionnelles  $x_1, x_2, x_3, x_4, x_5$  :



on annote l'OBDD pour donner le nombre d'interprétations concernant les variables plus grandes que celle du nœud  $n$  qui satisfont la formule associée à  $n$ .

$x_5:1$	$\{x_5 \mapsto F\}$
$x_4:2$	$\{x_4 \mapsto V; x_5 \mapsto F\}, \{x_4 \mapsto V; x_5 \mapsto V\}$
$x_3:2 + 2 = 4$	$\{x_3 \mapsto V\} \cup I(x_4), \{x_3 \mapsto F, x_4 \mapsto V/F\} \cup I(x_5)$
$x_2:4 + 4 = 8$	$\{x_2 \mapsto F\} \cup I(x_3), \{x_2 \mapsto V, x_3 \mapsto V/F, x_4 \mapsto V/F\} \cup I(x_5)$
$x_1:8 + 8$	$\{x_1 \mapsto F\} \cup I(x_2), \{x_1 \mapsto V, x_2 \mapsto V/F\} \cup I(x_3)$

(b) la fonction `indice` calcule pour chaque sommet de l'OBDD un entier qui est la position de la variable dans l'ordre si le sommet est un nœud, et si le sommet est une feuille, l'indice est  $n + 1$  avec  $n$  le nombre total de variables.

`nbsat` calcule pour un graphe  $G$  et un sommet  $n$ , le nombre de modèles de la formule en ne considérant que les variables plus grandes que celle du sommet.

```

nbsat (G, n) =
  si feuille(G, n)
  alors si valeur(G, n) alors 1 sinon 0
  sinon soit p = indice(n)
        soit v = vrai(G, n)
        soit f = faux(G, n)
        2^(indice(v)-indice(n)-1) × nbsat (G, v)
        + 2^(indice(f)-indice(n)-1) × nbsat (G, f)

```

(c) Il faut ajouter les interprétations des variables plus petites que celle qui apparaît au sommet de l'OBDD

$\text{nbsat1}(G, n) = 2^{(\text{indice}(n)-1)} \times \text{nbsat}(G, n)$

**Exercice 3.9** On utilise les règles de la déduction naturelle.

1. Pour la formule  $A \Rightarrow (B \Rightarrow A)$  on se ramène par deux introduction de  $\Rightarrow$  à prouver  $A$  sous les hypothèses  $A, B$  ce qui est vrai par hypothèse.

Cette preuve s'écrit sous forme d'arbre de dérivation :

$$\begin{array}{c} \text{hyp} \frac{}{A, B \vdash A} \\ \Rightarrow I \frac{}{A \vdash B \Rightarrow A} \\ \Rightarrow I \frac{}{\vdash A \Rightarrow (B \Rightarrow A)} \end{array}$$

2. Pour la formule  $(A \Rightarrow \neg A) \Rightarrow \neg A$  on se ramène par une introduction de  $\Rightarrow$  et une introduction de  $\neg$  à prouver l'absurde  $\perp$  sous les hypothèses  $A \Rightarrow \neg A, A$ . Pour cela on va utiliser l'élimination de la négation en essayant de prouver à la fois  $A$  (vrai par hypothèse) et  $\neg A$ . Cette dernière propriété s'obtient simplement par Modus Ponens en utilisant  $A \Rightarrow \neg A$  vrai par hypothèse et  $A$  qui est aussi vrai par hypothèse.

$$\begin{array}{c} \text{hyp} \frac{}{A \Rightarrow \neg A, A \vdash A \Rightarrow \neg A} \quad \text{hyp} \frac{}{A \Rightarrow \neg A, A \vdash A} \quad \text{hyp} \frac{}{A \Rightarrow \neg A, A \vdash A} \\ \Rightarrow E \frac{}{A \Rightarrow \neg A, A \vdash \neg A} \quad \neg E \frac{}{A \Rightarrow \neg A, A \vdash \perp} \\ \neg I \frac{}{A \Rightarrow \neg A \vdash \neg A} \\ \Rightarrow I \frac{}{\vdash (A \Rightarrow \neg A) \Rightarrow \neg A} \end{array}$$

3. Pour montrer  $(\exists x, P(x)) \Rightarrow \neg \forall x, \neg P(x)$ , on effectue deux introductions ( $\Rightarrow$  suivi de  $\neg$ ). On a pour hypothèses  $\exists x, P(x)$  et  $\forall x, \neg P(x)$  et il faut montrer une contradiction. Pour cela on procède par élimination de la propriété existentielle  $\exists x, P(x)$  qui est vraie par hypothèse et on peut donc introduire un nouvel objet  $y$  et une hypothèse  $P(y)$ , on doit toujours montrer une contradiction, ce que l'on fait en montrant à la fois  $\neg P(y)$  et  $P(y)$ . La première propriété s'obtient par élimination de la quantification universelle  $\forall x, \neg P(x)$  sur l'objet  $y$  et la seconde est juste une hypothèse. Cela donne l'arbre de dérivation suivant :

$$\begin{array}{c} \text{hyp} \frac{}{(\forall x, \neg P(x)), P(y) \vdash \forall x, \neg P(x)} \\ \forall E \frac{}{(\exists x, P(x)), (\forall x, \neg P(x)), P(y) \vdash \neg P(y)} \quad \text{hyp} \frac{}{(\exists x, P(x)), \forall x, \neg P(x), P(y) \vdash P(y)} \\ \neg E \frac{}{(\exists x, P(x)), (\forall x, \neg P(x)), P(y) \vdash \perp} \\ \exists H \frac{}{(\exists x, P(x)), (\forall x, \neg P(x)) \vdash \perp} \\ \neg I \frac{}{\exists x, P(x) \vdash \neg \forall x, \neg P(x)} \\ \Rightarrow I \frac{}{\vdash (\exists x, P(x)) \Rightarrow \neg \forall x, \neg P(x)} \end{array}$$

4.

$$\begin{array}{c} \text{hyp} \frac{}{\neg \exists x, P(x), P(y) \vdash P(y)} \\ \exists I \frac{}{(\neg \exists x, P(x)), P(y) \vdash \exists x, P(x)} \\ \neg H \frac{}{(\neg \exists x, P(x)), P(y) \vdash \perp} \\ \neg I \frac{}{(\neg \exists x, P(x)) \vdash \neg P(y)} \\ \forall I \frac{}{\neg \exists x, P(x) \vdash \forall y, \neg P(y)} \end{array}$$



**Exercice 3.10** On utilise les règles de la déduction naturelle en particulier la règle de raisonnement par l'absurde.

1.

$$\begin{array}{c} \text{hyp} \frac{}{P, \neg P \vdash P} \\ \neg E \frac{}{P, \neg P \vdash \perp} \\ \neg I \frac{}{P \vdash \neg \neg P} \end{array} \qquad \begin{array}{c} \text{hyp} \frac{}{\neg \neg P, \neg P \vdash \neg P} \\ \neg E \frac{}{\neg \neg P, \neg P \vdash \perp} \\ \text{abs} \frac{}{\neg \neg P \vdash P} \end{array}$$

2.

$$\begin{array}{c} \text{hyp} \frac{}{\neg(\neg P \vee P), P \vdash P} \\ \vee I_d \frac{}{\neg(\neg P \vee P), P \vdash \neg P \vee P} \\ \neg E \frac{}{\neg(\neg P \vee P), P \vdash \perp} \\ \neg I \frac{}{\neg(\neg P \vee P) \vdash \neg P} \\ \vee I_g \frac{}{\neg(\neg P \vee P) \vdash \neg P \vee P} \\ \neg E \frac{}{\neg(\neg P \vee P) \vdash \perp} \\ \text{abs} \frac{}{\vdash \neg P \vee P} \end{array}$$

3.

$$\begin{array}{c} \text{hyp} \frac{}{(\forall x, \neg P(x)), \neg P(a) \vdash \forall y, \neg P(y)} \\ \Rightarrow I \frac{}{(\forall x, \neg P(x)) \vdash \neg P(a) \Rightarrow \forall y, \neg P(y)} \\ \exists I \frac{}{(\forall x, \neg P(x)) \vdash \exists x, (\neg P(x) \Rightarrow \forall y, \neg P(y))} \end{array}$$

s'il existe quelqu'un qui boit alors c'est lui qu'on choisit et d'affirmer qu'il est non buveur introduit une contradiction et nous permet de déduire que personne ne boit ...

$$\begin{array}{c} \text{hyp} \frac{}{P(x), \neg P(x) \vdash P(x)} \\ \neg E \frac{}{P(x), \neg P(x) \vdash \forall y, \neg P(y)} \\ \Rightarrow I \frac{}{P(x) \vdash \neg P(x) \Rightarrow \forall y, \neg P(y)} \\ \exists I \frac{}{P(x) \vdash \exists x, (\neg P(x) \Rightarrow \forall y, \neg P(y))} \\ \exists E \frac{}{(\exists x, P(x)) \vdash \exists x, (\neg P(x) \Rightarrow \forall y, \neg P(y))} \end{array}$$

On raisonne par cas si  $\exists x, P$  ou  $\neg(\exists x, P)$

$$\begin{array}{c} \text{cut} \frac{}{\vdash (\exists x, P) \vee \neg(\exists x, P)} \vee E \frac{}{(\exists x, P) \vee \neg(\exists x, P) \vdash B} \text{cut} \frac{}{\vdash B} \end{array}$$

**Exercice 3.11** On utilise les règles du calcul des séquents à conclusions multiples.

1.

$$\begin{array}{c} \text{hyp } \frac{}{A, B \vdash A} \\ \Rightarrow d \frac{}{A \vdash (B \Rightarrow A)} \\ \Rightarrow d \frac{}{\vdash A \Rightarrow (B \Rightarrow A)} \end{array}$$

2.

$$\begin{array}{c} \text{hyp } \frac{}{A \vdash A} \quad \text{hyp } \frac{}{A \vdash A} \\ \neg d \frac{}{\vdash A, \neg A} \\ \Rightarrow g \frac{}{(\neg A \Rightarrow A) \vdash A} \\ \Rightarrow d \frac{}{\vdash (\neg A \Rightarrow A) \Rightarrow A} \end{array}$$

3.

$$\begin{array}{c} \text{hyp } \frac{}{P(y), \vdash P(y), (\exists x, P(x))} \\ \exists d \frac{}{P(y) \vdash (\exists x, P(x))} \\ \neg d \frac{}{\vdash (\exists x, P(x)), \neg P(y)} \\ \forall d \frac{}{\vdash (\exists x, P(x)), (\forall x, \neg P(x))} \\ \neg g \frac{}{\neg \forall x, \neg P(x) \vdash \exists x, P(x)} \end{array}$$

4. La négation se traite simplement en faisant changer de côté la formule.

$$\begin{array}{cc} \text{hyp } \frac{}{P \vdash P} & \text{hyp } \frac{}{P \vdash P} \\ \neg g \frac{}{P, \neg P \vdash} & \neg d \frac{}{\vdash P, \neg P} \\ \neg d \frac{}{P \vdash \neg \neg P} & \neg g \frac{}{\neg \neg P \vdash P} \end{array}$$

5. pour traiter le lemme du buveur il est nécessaire de commencer par introduire un objet arbitraire  $a$  afin de pouvoir décomposer ensuite le reste de la formule qui nous donnera l'élément pour conclure la preuve.

$$\begin{array}{c} \text{hyp } \frac{}{\neg P(a), P(z) \vdash \exists x, (\neg P(x) \Rightarrow \forall y, \neg P(y)), (\forall y, \neg P(y)), P(z)} \\ \neg g \frac{}{\neg P(a), P(z), \neg P(z) \vdash \exists x, (\neg P(x) \Rightarrow \forall y, \neg P(y)), (\forall y, \neg P(y))} \\ \Rightarrow d \frac{}{\neg P(a), P(z) \vdash \exists x, (\neg P(x) \Rightarrow \forall y, \neg P(y)), (\neg P(z) \Rightarrow \forall y, \neg P(y))} \\ \exists d \frac{}{\neg P(a), P(z) \vdash \exists x, (\neg P(x) \Rightarrow \forall y, \neg P(y))} \\ \neg d \frac{}{\neg P(a) \vdash \exists x, (\neg P(x) \Rightarrow \forall y, \neg P(y)), \neg P(z)} \\ \forall d \frac{}{\neg P(a) \vdash \exists x, (\neg P(x) \Rightarrow \forall y, \neg P(y)), (\forall y, \neg P(y))} \\ \Rightarrow d \frac{}{\vdash \exists x, (\neg P(x) \Rightarrow \forall y, \neg P(y)), (\neg P(a) \Rightarrow \forall y, \neg P(y))} \\ \exists d \frac{}{\vdash \exists x, (\neg P(x) \Rightarrow \forall y, \neg P(y))} \end{array}$$

## A.4 Exercices du chapitre 4

### Exercice 4.1

- Montrons que l'interprétation de Herbrand dans laquelle  $P(u, v)$  est vrai si et seulement si  $u \sim v$  est un modèle des formules  $A_0, A_1, A_2, A_3$ . Il faut montrer :
  - $e \sim e$  (vrai avec la séquence vide),
  - Soit  $u$  et  $v$  deux termes quelconques, si  $u \sim v$  alors  $a(u), b(a(v))$  qui est vrai en ajoutant 1 en tête de la séquence qui fait correspondre  $u$  et  $v$  et de même pour les deux autres propriétés en utilisant le 2ème et le 3ème couple.
- On montre que si  $u \sim v$  alors  $A_0, A_1, A_2, A_3 \models P(u, v)$  par récurrence sur la taille de la séquence  $i_1, \dots, i_k$ .
  - Si la séquence est vide alors les termes  $u$  et  $v$  correspondent au mot vide donc au terme  $e$ . Il faut montrer  $A_0, A_1, A_2, A_3 \models P(e, e)$  ce qui est vrai ( $A_0$ ).
  - On suppose la propriété vraie pour toutes les séquences de taille  $k$  et on veut montrer que c'est vrai pour une séquence de taille  $k+1$ . On a donc une séquence  $i_0, i_1, \dots, i_k$  et un terme  $u$  qui correspond au mot  $u_{i_0} u_{i_1} \dots u_{i_k}$  et le terme  $v$  correspond au mot  $v_{i_0} v_{i_1} \dots v_{i_k}$ . On introduit aussi le terme  $u'$  qui correspond au mot  $u_{i_1} \dots u_{i_k}$  et le terme  $v'$  qui correspond au mot  $v_{i_1} \dots v_{i_k}$ . Par hypothèse de récurrence on a  $A_0, A_1, A_2, A_3 \models P(u', v')$  et en utilisant la propriété  $A_{i_0}$  on en déduit  $A_0, A_1, A_2, A_3 \models P(u, v)$ . On pourra détailler le cas  $i_0 = 1$  et admettre que les autres cas sont pareils.
- On a  $A_0 \models \exists x, P(x, x)$ , trivialement en prenant le terme  $e$  pour  $x$ .
- Une solution du problème de Post nous donne une séquence et un mot non vide  $u$  tel que  $A_0, A_1, A_2, A_3 \models P(u, u)$ . Comme ce mot est non vide, il s'écrit de la forme  $a(u')$  ou  $b(u')$  donc on a bien  $A_0, A_1, A_2, A_3 \models \exists x, (P(a(x), a(x)) \vee P(b(x), b(x)))$ .
- Il suffit de se placer dans le modèle dans lequel  $P(u, v)$  est vrai si et seulement si  $u \sim v$ . Le modèle valide  $A_0, A_1, A_2, A_3$  et donc valide  $\exists x, (P(a(x), a(x)) \vee P(b(x), b(x)))$  d'où l'existence d'un terme  $u$  tel que  $a(u) \sim a(u)$  est vrai ou bien  $b(u) \sim b(u)$ . Dans les deux cas on a une solution au problème de Post.

### Exercice 4.2

- L'arbre de dérivation pour le séquent  $\vdash ((p \wedge q) \Rightarrow r) \Rightarrow (\neg p \vee q) \Rightarrow (p \Rightarrow r)$  est :

$$\begin{array}{c}
 \text{hyp} \frac{}{p \vdash r, q, p} \\
 \neg g \frac{}{\neg p, p \vdash r, q} \quad \frac{}{q, p \vdash r, q} \\
 \text{hyp} \frac{}{(\neg p \vee q), p \vdash r, p} \quad \vee g \frac{}{(\neg p \vee q), p \vdash r, q} \\
 \wedge d \frac{}{(p \wedge q), p \vdash r, p \wedge q} \quad \text{hyp} \frac{}{r, (\neg p \vee q), p \vdash r} \\
 \Rightarrow g \frac{}{((p \wedge q) \Rightarrow r), (\neg p \vee q), p \vdash r} \\
 \Rightarrow d \frac{}{((p \wedge q) \Rightarrow r), (\neg p \vee q) \vdash (p \Rightarrow r)} \\
 \Rightarrow d \frac{}{((p \wedge q) \Rightarrow r) \vdash (\neg p \vee q) \Rightarrow (p \Rightarrow r)} \\
 \Rightarrow d \frac{}{\vdash ((p \wedge q) \Rightarrow r) \Rightarrow (\neg p \vee q) \Rightarrow (p \Rightarrow r)}
 \end{array}$$

Les feuilles de l'arbre correspondent à des règles hypothèses donc la formule est valide.



3. On montre que les règles de DPLL sont correctes. C'est-à-dire que si un arbre DPLL de racine  $I \gg \Delta$  contient une feuille de succès  $J \gg \emptyset$  alors l'interprétation  $J$  étend l'interprétation  $I$  (c'est-à-dire que  $I(x) = J(x)$  pour toutes les variables  $x$  qui apparaissent dans  $I$ ) et l'interprétation  $J$  rend vraies toutes les clauses de  $\Delta$ .

On raisonne par minimalité sur l'arbre de dérivation DPLL.

- Si on a une feuille, soit c'est une feuille de succès  $I \gg \emptyset$  et alors comme  $\Delta$  est vide on a que  $I$  rend vrai les clauses de  $\Delta$  et la seule feuille de succès dans cet arbre est la feuille elle-même. Si c'est une feuille de conflit alors il n'y a pas de feuille de succès et donc la propriété est vraie.
- Si on a une règle  $\text{BCP}_V$  alors comme on a une seule prémisses  $I \gg \Delta$ , si l'arbre contient une feuille de succès  $J \gg \emptyset$ , elle est dans ce sous-arbre et donc  $J$  étend  $I$  et rend vrai les formules de  $\Delta$ . On a  $J(l) = I(l) = V$  et donc  $J$  rend vrai aussi  $l \vee C$ .
- Si on a une règle  $\text{BCP}_F$  alors comme on a une seule prémisses  $I \gg \Delta, C$ , si l'arbre contient une feuille de succès  $J \gg \emptyset$ , elle est dans ce sous-arbre et donc  $J$  étend  $I$  et rend vrai les formules de  $\Delta, C$  et donc  $J$  rend vrai aussi  $l \vee C$ .
- Si on a une règle  $\text{ASSUME}_V$  alors comme on a une seule prémisses  $I + \{x \mapsto V\} \gg \Delta$ , si l'arbre contient une feuille de succès  $J \gg \emptyset$ , elle est dans ce sous-arbre et donc  $J$  étend  $I + \{x \mapsto V\}$  et rend vrai les formules de  $\Delta$ . On a  $J(x) = V$  et donc  $J$  rend vrai aussi  $\Delta, x$ .
- Si on a une règle  $\text{ASSUME}_F$  alors comme on a une seule prémisses  $I + \{x \mapsto F\} \gg \Delta$ , si l'arbre contient une feuille de succès  $J \gg \emptyset$ , elle est dans ce sous-arbre et donc  $J$  étend  $I + \{x \mapsto F\}$  et rend vrai les formules de  $\Delta$ . On a  $J(x) = F$  et donc  $J$  rend vrai aussi  $\Delta, \neg x$ .
- Si on a une règle UNSAT alors il y a deux prémisses donc la feuille de succès  $J \gg \emptyset$  peut se trouver dans l'une des deux branches (ou dans les deux). Dans les deux cas on a que  $J$  étend  $I + \{x \mapsto V/F\}$  et donc  $J$  étend  $I$  et  $J$  rend vrai les formules de  $\Delta$ .

4. Pour trouver un modèle d'une formule :

- (a) On met la formule en forme normale conjonctive, on obtient ainsi un ensemble de clauses  $\Delta$ .
- (b) On applique la méthode DPLL en partant du problème  $\{\} \gg \Delta$ .
- (c) Si on arrive à une feuille de succès  $I \gg \emptyset$  alors  $I$  est un modèle (on peut choisir des valeurs arbitraires pour les variables de  $\Delta$  qui ne seraient pas affectées dans  $I$ ).

L'arbre n'est pas unique et on peut choisir différentes stratégies pour trouver une branche de succès plus rapidement.

5. Il suffit de montrer que si on a un arbre DPLL de racine  $I \gg \Delta$  alors si  $K$  est un modèle de  $\Delta$  qui étend  $I$  alors il existe une feuille de succès  $J \gg \emptyset$  telle que  $K$  étend  $J$ . On regarde chaque règle :

- si on a une règle de succès, c'est trivial. Si c'est une feuille de conflit alors il n'y a pas de modèle de  $\perp$  et donc la propriété est vraie.
- Pour les règles BCP, si on a un modèle  $K$  de  $\Delta, (l \vee C)$  qui étend  $I$  alors c'est aussi un modèle de  $\Delta$  et donc on a bien une feuille de succès dans la règle  $\text{BCP}_V$  et pour  $\text{BCP}_F$ , comme  $K(l) = I(l) = F$  alors c'est aussi un modèle de  $\Delta, C$  et donc on a une feuille de succès qui convient.
- Dans le cas des règles ASSUME, si  $K$  est un modèle de  $\Delta, l$  qui étend  $I$  et que  $l$  n'a pas de valeur dans  $I$  alors on a forcément  $K(l) = V$  et donc  $K$  étend  $I + \{x \mapsto b\}$  et valide  $\Delta$  et donc il y a bien une feuille de succès qui convient.
- Pour la règle UNSAT si  $K$  est un modèle de  $\Delta$  qui étend  $I$  on regarde la valeur de  $K(x)$ , si  $K(x) = V$  alors  $K$  étend  $I + \{x \mapsto V\}$  et donc on trouvera la branche de succès dans le sous-arbre gauche et si  $K(x) = F$  alors on trouvera la feuille dans le sous-arbre droit correspondant à la branche  $I + \{x \mapsto F\}$ .

Si on construit tout l'arbre alors on va trouver tous les modèles en regardant toutes les feuilles de succès et en considérant toutes les extensions.

6. On peut utiliser cette méthode pour prouver la validité d'une formule. Si on a une formule  $A$ , on trouve la forme clausale correspondant à  $\neg A$  et on construit l'arbre DPLL. Si toutes les feuilles sont des conflits alors  $\neg A$  est insatisfiable et donc  $A$  est valide.

**Exercice 4.5**

1.  $N(x, \epsilon) \stackrel{?}{=} N(\epsilon, y)$

Solution :  $\{x \leftarrow \epsilon; z \leftarrow \epsilon\}$

2.  $N(N(x, y), \epsilon) \stackrel{?}{=} N(\epsilon, y)$

Pas de solution : on simplifie en utilisant (7) qui amène l'équation  $N(x, y) \stackrel{?}{=} \epsilon$  qui échoue (règle 6) car il faudrait rendre égaux un terme qui commence par le symbole  $N$  avec un terme qui commence par le symbole  $\epsilon$ .

3.  $N(x, \epsilon) \stackrel{?}{=} x$

Pas de solution : échec règle (3) après avoir appliqué la règle (5) pour retourner l'équation. Quel que soit le terme qui remplace la variable  $x$ , la partie gauche de l'équation aura toujours plus de symboles que la partie droite.

**Exercice 4.6** Pour chaque problème, on transforme l'ensemble des équations en indiquant la règle utilisée et on construit progressivement la substitution

1.

règle	substitution	équations
(7)		$g(x, f(y)) \stackrel{?}{=} g(f(y), z)$
(4)		$x \stackrel{?}{=} f(y), f(y) \stackrel{?}{=} z$
(5)	$\{x \leftarrow f(y)\}$	$f(y) \stackrel{?}{=} z$
(4)	$\{x \leftarrow f(y)\}$	$z \stackrel{?}{=} f(y)$
(1)	$\{x \leftarrow f(y)\} \{z \leftarrow f(y)\}$	$\emptyset$

Il faut ensuite développer la substitution résultat, ici on a

$$\{x \leftarrow f(y)\} \{z \leftarrow f(y)\} = \{x \leftarrow f(y); z \leftarrow f(y)\}$$

2.

règle	substitution	équations
(7)		$g(g(x, f(y)), z) \stackrel{?}{=} g(g(y, z), a)$
(7)		$g(x, f(y)) \stackrel{?}{=} g(y, z), z \stackrel{?}{=} a$
(4)		$x \stackrel{?}{=} y, f(y) \stackrel{?}{=} z, z \stackrel{?}{=} a$
(6)	$\{z \leftarrow a\}$ echec	$x \stackrel{?}{=} y, f(y) \stackrel{?}{=} a$

**Exercice 4.7**

1.  $(\text{plus}(x, 0) \leq x) \stackrel{?}{=} (\text{plus}(0, y) \leq y)$

Solution  $\{x \leftarrow 0; y \leftarrow 0\}$

2.  $(\text{plus}(x, 0) \leq x) \stackrel{?}{=} \neg(y \leq z)$

Pas de solution car un littéral positif ne peut être égal à un littéral négatif.

3.  $(\text{plus}(x, 0) \leq x) \stackrel{?}{=} (y \leq \text{plus}(y, 0))$

Pas de solution : le problème se décompose en  $\text{plus}(x, 0) \stackrel{?}{=} y; x \stackrel{?}{=} \text{plus}(y, 0)$  qui se simplifie par les règles (5) puis (4), on obtient  $\{y \leftarrow \text{plus}(x, 0)\}(x \stackrel{?}{=} \text{plus}(\text{plus}(x, 0), 0))$  qui conduit à un échec suivant la règle (3).

**Exercice 4.8** Il suffit de résoudre le problème :  $\{t_1 \stackrel{?}{=} t_2; t_2 \stackrel{?}{=} t_3 \dots t_{n-1} \stackrel{?}{=} t_n\}$

On peut se ramener à une seule équation :  $F(t_1, t_2, \dots, t_{n-1}) = F(t_2, t_3, \dots, t_n)$

#### Exercice 4.9

- On peut changer l'ordre des motifs, les exécuter en parallèle lorsque :  $\text{inst}(p_i) \cap \text{inst}(p_j) = \emptyset$
- Tous les cas sont couverts si  $\text{inst}(p_1) \cup \dots \cup \text{inst}(p_n) = \mathcal{T}(\mathcal{F})$
- Le motif  $p_j$  est inutile si  $\text{inst}(p_j) \subseteq \text{inst}(p_1) \cup \dots \cup \text{inst}(p_{j-1})$

**Exercice 4.10** Pour prouver  $(\exists x, \neg P(x)), (\forall x, (P(x) \vee Q(x))) \models \exists x, Q(x)$  en utilisant la résolution, on procède par étapes :

1. Mise en forme clausale des hypothèses et de la négation de la conclusion  $\neg \exists x, Q(x) \equiv \forall x, \neg Q(x)$

$$\begin{aligned} C_1 &\stackrel{\text{def}}{=} \neg P(a) \\ C_2 &\stackrel{\text{def}}{=} P(x) \vee Q(x) \\ C_3 &\stackrel{\text{def}}{=} \neg Q(y) \end{aligned}$$

2. Résolution :

$$\begin{array}{c} [x \leftarrow a] \frac{C_1 \quad C_2}{Q(a)} \\ [y \leftarrow a] \frac{\quad C_3}{\perp} \end{array}$$

3. Conclusion : on a montré que l'ensemble  $(\exists x, \neg P(x)), (\forall x, (P(x) \vee Q(x))), \neg \exists x, Q(x)$  était insatisfiable, ce qui est équivalent à  $(\exists x, \neg P(x)), (\forall x, (P(x) \vee Q(x))) \models \exists x, Q(x)$

#### Exercice 4.11

1. Si  $\mathcal{E}$  est un ensemble de clauses tel que chaque clause contient au moins un littéral  $R(t_1, \dots, t_n)$  alors en choisissant pour interprétation de  $R$  la relation toujours vraie, chaque clause vaut vraie et donc l'ensemble des clauses  $\mathcal{E}$  est satisfiable.
2. Si  $\mathcal{E}$  est un ensemble de clauses, qui ne contient aucun littéral négatif  $\neg R(u_1, \dots, u_n)$ , alors  $\mathcal{E}$  est satisfiable ssi l'ensemble  $\{C \in \mathcal{E} \mid C \text{ ne contient pas de littéral } R(t_1, \dots, t_n)\}$  est satisfiable.  
Si  $\mathcal{E}$  est satisfiable alors a fortiori tout sous ensemble de  $\mathcal{E}$  l'est.  
Si  $\{C \in \mathcal{E} \mid C \text{ ne contient pas de littéral } R(t_1, \dots, t_n)\}$  est satisfiable alors comme il n'y a pas non plus de littéral  $\neg R(u_1, \dots, u_n)$  dans cette ensemble, l'interprétation qui rend vrai cet ensemble de clause ne dépend pas de l'interprétation de  $R$ , il suffit donc de l'étendre avec pour interprétation de  $R$  la relation toujours vraie pour avoir une interprétation qui rend vraie l'ensemble de clauses  $\mathcal{E}$ .