

Université de Sherbrooke
Département d'informatique

Logique et mathématiques discrètes
MAT115

Notes de cours
Version du 2025-02-07

Marc Frappier, Ph.D.
professeur

Notes de cours (très) synthétiques complétées par les présentations en classe.

Contenu

1	Introduction à la logique propositionnelle et la logique du premier ordre	2
1.1	Logique propositionnelle	4
1.1.1	Syntaxe	4
1.1.2	Tables de vérité des connecteurs logiques	7
1.1.3	Expression de faits en logique propositionnelle	8
1.2	Logique du premier ordre	12
1.2.1	Syntaxe	12
1.2.2	Interprétation des quantificateurs	16
1.2.3	TARSKIUNDES	17
1.2.4	Formes typiques des formules quantifiées	25
1.2.5	Formalisation de texte en langage naturel	27
1.3	Lois de la logique	28
1.3.1	Lois de la logique propositionnelle (tautologies)	28
1.3.2	Lois de la logique du premier ordre	31
1.4	Preuve en logique propositionnelle	32
1.4.1	Preuve par équivalence	32
1.4.2	Déduction	33
1.4.3	Règles d'inférence de la déduction naturelle	36
1.4.4	Cohérence, conséquence et complétude	39
1.4.5	Utilisation de la logique	42
1.5	Preuve en logique du premier ordre	44
1.5.1	Règles d'inférence	44
1.6	Formes normales	44
1.7	Exercices	48
2	Ensemble, relation et fonction	49
2.1	Conventions	50
2.2	La logique du premier ordre en B	50
2.3	Les ensembles	50
2.3.1	Définition par extension	50
2.3.2	Définition par compréhension	51
2.3.3	Appartenance	52
2.3.4	Inclusion	53
2.3.5	Les ensembles en B	54
2.3.6	Opérations sur les ensembles en B	55
2.4	Les relations	56
2.5	Les fonctions	61

2.5.1	Fonction	62
2.5.2	Fonction totale	64
2.5.3	Fonction injective	65
2.5.4	Fonction injective totale	66
2.5.5	Fonction surjective	67
2.5.6	Fonction surjective totale	68
2.5.7	Fonction bijective	69
2.5.8	Fonction bijective totale	70
2.5.9	Modèle conceptuel de données	71
2.6	Propriétés des relations	73
2.6.1	Relation réflexive	74
2.6.2	Relation irreflexive	75
2.6.3	Relation transitive	76
2.6.4	Relation symétrique	77
2.6.5	Relation asymétrique	78
2.6.6	Relation antisymétrique	79
2.6.7	Relation d'équivalence	80
2.6.8	Relation d'ordre	80
2.6.9	Relation d'ordre strict	81
2.6.10	Relation bien fondée	82
2.6.11	Relation acyclique	83
2.7	Les nombres	84
2.8	Diverses lois	85
2.9	Les suites	85
2.10	Combinatoire	87
2.11	Utilisation de ProB	89
2.12	Exercices	93
3	Types de preuve	98
3.1	Preuve par induction	98
3.1.1	Preuve par induction sur les naturels	98
3.1.2	Règle générale de preuve par induction	105
3.1.3	Application de la règle générale d'induction aux naturels	106
3.1.4	Plusieurs cas de base sur les naturels	108
3.1.5	Preuve par induction sur d'autres structures	110
3.1.6	Preuve de la règle générale de preuve par induction	113
3.2	Exprimer une preuve en langage naturel	115
3.2.1	$\mathcal{A} \Rightarrow \mathcal{B}$ - Implication	115
3.2.2	$\mathcal{A} \Leftrightarrow \mathcal{B}$ - Équivalence	115
3.2.3	$\mathcal{A} \wedge \mathcal{B}$ - Conjonction	115
3.2.4	Preuve par contradiction	116
3.2.5	Preuve par contraposition (contraposée)	116
3.2.6	Preuve par cas	116
3.2.7	\forall - Quantification universelle	116
3.2.8	\exists - Quantification existentielle	117
3.2.9	Illustration d'une preuve en langage naturel	117
3.3	Preuve dans le style équationnel	118
3.4	La preuve de correction de programmes séquentiels	121

3.4.1	La spécification d'un programme	121
3.4.2	Les axiomes du wp-calcul	122
3.4.3	Affectation	122
3.4.4	La séquence	123
3.4.5	La conditionnelle	123
3.4.6	La boucle	124
3.4.7	Preuve de correction d'un programme	126
3.5	Exercices	128
4	Automate	130
4.1	Automate fini déterministe	130
4.2	Automate fini non-déterministe	133
4.3	Déterminisation d'un AFND	134
4.4	Minimisation d'un AFD	137
4.5	Exercices	142
	Bibliographie	147
	Index	148

Chapitre 1

Introduction à la logique propositionnelle et la logique du premier ordre

La logique est au centre du développement des sciences et du bon fonctionnement de la société en général. L’humain est capable de raisonnement, c’est-à-dire de déduire des faits ou de nouvelles connaissances, à partir d’autres faits. On qualifie d’esprit “logique” une personne capable d’agir avec cohérence et rigueur, de *raisonner* correctement. Les mathématiques constituent le langage commun des sciences, et la logique est le fondement des mathématiques. L’informatique a été fondée dans les années 30 en tentant de résoudre un problème fondamental de la logique, proposé par David Hilbert et Wilhelm Ackermann en 1928, soit de déterminer si une formule quelconque est un théorème (*Entscheidungsproblem* en allemand). Ce problème a été résolu par Alonzo Church et Alan Turing, de manière indépendante, en 1935 et 1936. Ils ont démontré que cela était impossible en général. Ces travaux ont nécessité le développement de la notion d’*algorithme* et d’*ordinateur*, sous la forme de la machine de Turing¹ (par Turing) et du λ -calcul (par Church et Kleene). Le λ -calcul est le fondement des langages de programmation fonctionnelle comme Lisp, F#, Haskell et Scala. La logique date d’Aristote (384—322 avant J.-C.), mais sa version moderne date de la fin du 19^{ième} au début du 20^{ième} siècle. Ses pionniers furent Boole, De Morgan, Frege, Peano, Peirce, Whitehead, Russell, Hilbert, Ackermann, Gödel, ainsi que plusieurs autres.

La logique permet principalement deux choses:

1. *exprimer* de manière *formelle* des faits;
2. *déduire* de manière *formelle* de nouveaux faits.

L’adjectif *formel* utilisé ici signifie “sans ambiguïté”, “avec rigueur”, “objectif” (par opposition à “subjectif”), “calculable”, “analysable par une machine”. Les faits sont exprimés comme des *formules*. Un

¹Alan Turing (1912—1954) est devenu “célèbre” en 2015 suite au très populaire film *The imitation game* qui raconte le développement par Alan Turing d’une machine pouvant déchiffrer le code Enigma utilisé par les allemands durant la deuxième guerre mondiale, et qui se mérita l’Oscar du meilleur scénario adapté. L’équivalent du prix Nobel en informatique, le *ACM Turing Award*, est nommé en l’honneur d’Alan Turing, qui est souvent considéré comme le “père” de l’informatique. Il est décerné chaque année depuis 1966 à un informaticien ayant effectué une contribution exceptionnelle à l’informatique. Cinq canadiens se sont mérités ce prix. Stephen Arthur Cook (né aux États-Unis), professeur à l’Université de Toronto, le reçut en 1982 pour ses travaux en théorie de la complexité des algorithmes. William Morton Kahan le reçut en 1986 pour ses travaux en analyse numérique. Yoshua Bengio (né en France), de l’Université de Montréal, et Geoffrey Hinton (né en Angleterre), de l’Université de Toronto, le reçurent en 2018 pour leurs contributions au développement des réseaux de neurones profonds (conjointement avec Yann LeCun de l’Université de New-York). Alfred Vaino Aho (né au Canada), professeur à l’Université Columbia aux USA, le reçut en 2020 pour ses travaux en compilation, algorithmes et structures de données. Google offre maintenant un prix de 1 000 000 \$ au récipiendaire du prix Alan Turing.

ensemble de formules logiques peut être analysé pour déterminer sa cohérence, c'est-à-dire l'absence de *contradiction* entre les formules. Les règles utilisées pour faire de la déduction peuvent aussi être analysées pour déterminer leur cohérence, c'est-à-dire l'impossibilité de déduire une formule fautive à partir de formules considérées comme vraies. Une *preuve*, au sens de la logique formelle, est une suite de déductions. Les déductions sont effectuées en appliquant des *règles d'inférence*.

Les formules de logique sont énoncées à l'aide d'une syntaxe formelle (c'est-à-dire précise et sans ambiguïté). Plusieurs types de logiques ont été proposées. La logique *propositionnelle* constitue la base de tous les types de logique. Une formule est une représentation formelle d'un fait, d'une connaissance à propos d'un concept d'intérêt pour l'humain. La première compétence que vous devrez acquérir sera de *représenter* des faits, des connaissances, sous forme de formules logiques. La deuxième sera de raisonner sur ces formules. Le verbe "représenter" a une importance capitale: une formule n'est rien d'autre qu'une suite de symboles. Il faut aussi associer ces symboles à des objets, des concepts, des faits du "monde" représenté par la formule. Le "monde" dénote ce dont la formule traite, sur quoi porte la formule. On appelle *interprétation* le lien entre les symboles d'une formule et les objets du monde qu'ils représentent. La beauté de la logique est que le raisonnement et les calculs effectués sont indépendants du "monde" en question.

La logique est utilisée dans tous les domaines de l'informatique. Tous les langages de programmation utilisent les connecteurs (c'est-à-dire les opérateurs) de la logique propositionnelle. Les méthodes les plus avancées pour déterminer la correction d'un logiciel (c'est-à-dire vérifier qu'un logiciel fait bien ce qu'il est supposé faire, vérifier qu'un logiciel est correct, vérifier qu'un logiciel ne contient pas de faute (*bug*)) sont fondées sur la logique. Les logiciels contrôlent maintenant une foule d'objets comme des trains, des avions, des autos, des centrales nucléaires, des stimulateurs cardiaques, des appareils de radiologie. Une erreur dans ces logiciels peut entraîner des conséquences dramatiques pour les humains et l'environnement. L'étude de leur correction est primordiale. Cela ne serait possible sans la logique. Les opérations les plus élémentaires d'un ordinateur (opérations arithmétiques) sont exprimées en logique propositionnelle. Le fonctionnement de base d'un ordinateur est fondé sur l'algèbre de Boole, qui est essentiellement la même chose que la logique propositionnelle. Une algèbre permet de faire des calculs, c'est-à-dire appliquer des opérateurs à des opérandes. La logique permet aussi de faire des calculs, comme déterminer si une formule est vraie ou fautive pour une interprétation donnée, mais aussi de déduire de nouvelles formules.

Un des objectifs de ce cours est aussi de vous apprendre à développer une familiarité avec la notation mathématique. L'enseignement des mathématiques au niveau du primaire, du secondaire et du collégial a souvent tendance à vouloir beaucoup schématiser les concepts mathématiques, ce qui est utile pour favoriser la compréhension. Toutefois, un schéma demeure une *approximation* du concept mathématique; il ne permet pas de toujours bien saisir toutes les nuances, et il porte parfois (trop souvent!) à confusion, car la même image peut signifier des choses différentes pour deux personnes. On dit qu'une image vaut mille mots, mais une définition mathématique vaut une *infinité* de mots. Apprendre les mathématiques par l'image et les schémas, c'est un peu comme apprendre la musique à l'oreille: c'est bien, mais ça demeure limité. Il faut être capable de lire une partition pour devenir un bon musicien et jouer n'importe quelle pièce, et communiquer n'importe quelle pièce. Un schéma permet de développer l'intuition; il est un outil très important dans le travail des mathématiciens. Mais on ne définit pas un concept de manière précise avec un schéma; on fait simplement l'illustrer. Il faut utiliser des formules pour être précis. En spécification de système, ce sera la même chose; un modèle graphique d'un système, c'est bien, mais c'est insuffisant. Une formule mathématique permet de décrire exactement ce que l'on veut, sans ambiguïté.

Dans les cycles pré-universitaires, on enseigne souvent les mathématiques comme l'application d'une "recette". On apprend par coeur la recette, on découvre les cas où elle s'applique, et on l'applique sans réfléchir beaucoup à la recette elle-même (d'où vient-elle, pourquoi elle marche, y

a-t-il une meilleure recette, etc). Dans un cours de logique, on vise, avant tout, à développer votre capacité à réfléchir. Il n’y a pas de recette pour trouver une preuve ou pour créer un modèle d’un problème. En tant qu’informaticien, vous devrez *développer* des recettes. L’essentiel des tâches d’un informaticien peut se résumer ainsi : analyser, spécifier, concevoir, implémenter, valider et vérifier. Toutes ces tâches demandent de réfléchir, de raisonner, de créer et de tester. Cela demande de la créativité; il ne s’agit pas simplement d’appliquer des recettes.

1.1 Logique propositionnelle

1.1.1 Syntaxe

Une formule de la logique propositionnelle est construite à l’aide de *connecteurs logiques*, de *variables propositionnelles* et des *constantes vrai* (parfois représentée par “1” ou “ \top ”) et *faux* (parfois représentée par “0” ou “ \perp ”). Les connecteurs logiques usuels sont les suivants:

- \neg : la *négation*,
- \wedge : la *conjonction*, aussi appelé le “et” logique,
- \vee : la *disjonction*, aussi appelé le “ou” logique,
- \oplus : la *disjonction exclusive*, aussi appelé le “ou exclusif”,
- \Rightarrow : l’*implication*, aussi noté “ \rightarrow ” dans certains textes et logiciels,
- \Leftarrow : l’*implication inverse*, aussi noté “ \leftarrow ” dans certains textes et logiciels,
- \Leftrightarrow : l’*équivalence*, aussi noté “ \leftrightarrow ” dans certains textes et logiciels.

Une variable propositionnelle peut prendre la valeur *vrai* ou la valeur *faux*. Dans les exemples et définitions, nous utiliserons, par convention, X pour dénoter une variable propositionnelle (possiblement indexée par un nombre, comme X_0, X_1, \dots , si plusieurs variables sont nécessaires). Nous utiliserons $\mathcal{A}, \mathcal{B}, \mathcal{C}, \dots$ pour désigner une formule propositionnelle.

Définition 1 Une *formule propositionnelle* est une suite de symboles construite seulement à partir des règles suivantes:

1. *vrai* et *faux* sont des formules propositionnelles;
2. si X est une variable propositionnelle, alors X est une formule propositionnelle;
3. si \mathcal{A} est une formule propositionnelle, alors $\neg\mathcal{A}$ est une formule propositionnelle;
4. si \mathcal{A} et \mathcal{B} sont des formules propositionnelles, alors
 - $\mathcal{A} \wedge \mathcal{B}$
 - $\mathcal{A} \vee \mathcal{B}$
 - $\mathcal{A} \oplus \mathcal{B}$
 - $\mathcal{A} \Rightarrow \mathcal{B}$
 - $\mathcal{A} \Leftarrow \mathcal{B}$
 - $\mathcal{A} \Leftrightarrow \mathcal{B}$

sont des formules propositionnelles;

5. si \mathcal{A} est une formule propositionnelle, alors (\mathcal{A}) est une formule propositionnelle.

□

Les parenthèses servent à définir un ordre dans l'évaluation des connecteurs logiques, exactement comme dans les expressions arithmétiques.

Remarque 1 Dans le cadre du cours, par souci de lisibilité, les connecteurs sont parenthésés selon les priorités suivantes, de la plus forte à la plus faible; les opérateurs sur une même ligne ont la même priorité:

1. \neg ,
2. \wedge ,
3. \vee, \oplus
4. \Rightarrow, \Leftarrow
5. \Leftrightarrow .

Attention : Il n'existe pas de convention universelle pour la priorité en logique. Elle peut varier d'un livre à l'autre et d'un outil à l'autre. Il vous incombe de prendre connaissance de la priorité utilisée selon le livre ou l'outil.

En arithmétique, il est usuel de considérer que la multiplication a préséance sur l'addition, ce qui fait que lorsqu'on écrit $a + b * c$, on veut dire $a + (b * c)$. De la même manière, si on écrit

$$\mathcal{A} \wedge \mathcal{B} \vee \mathcal{C}$$

cela signifie, à cause de la préséance de \wedge sur \vee ,

$$(\mathcal{A} \wedge \mathcal{B}) \vee \mathcal{C}$$

Les parenthèses peuvent aussi être omise pour les opérateurs associatifs. Par exemple, l'expression $a + b + c$ peut aussi être écrite $(a + b) + c$, ou bien $a + (b + c)$; cela n'a pas d'importance, car l'addition est associative; on omet généralement les parenthèses dans ce cas. On procède de manière similaire en logique avec les **connecteurs associatifs** de la logique, qui sont les suivants:

1. \wedge ,
2. \vee
3. \oplus
4. \Leftrightarrow .

On remarque que seule l'implication n'est pas associative. Il est donc **ambigu** d'écrire

$$\mathcal{A} \Rightarrow \mathcal{B} \Rightarrow \mathcal{C}$$

Cela peut être interprété comme $(\mathcal{A} \Rightarrow \mathcal{B}) \Rightarrow \mathcal{C}$, ou bien comme $\mathcal{A} \Rightarrow (\mathcal{B} \Rightarrow \mathcal{C})$, qui ne sont pas équivalentes. Pour choisir entre les deux, il faudrait définir si on associe d'abord à gauche, ou bien

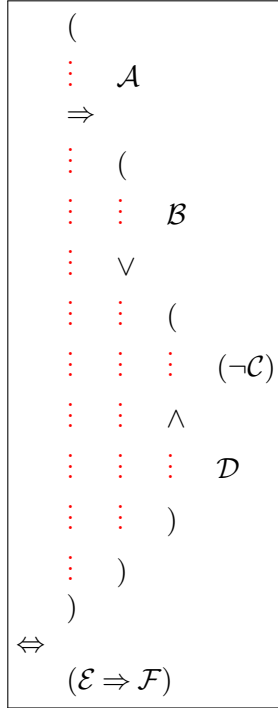


Figure 1.1: Représentation indentée de la formule (1.1)

à droite; en général, on évite ce type de règle de priorité sur les associations, car elles ne sont pas très intuitives.

Pour illustrer la priorité des connecteurs, considérez la formule suivante, qui ne contient aucune parenthèse:

$$\mathcal{A} \Rightarrow \mathcal{B} \vee \neg \mathcal{C} \wedge \mathcal{D} \Leftrightarrow \mathcal{E} \Rightarrow \mathcal{F} \quad (1.1)$$

Elle est équivalente, après introduction des parenthèses qui représentent la priorité des opérateurs, à la formule suivante

$$\left(\mathcal{A} \Rightarrow \left(\mathcal{B} \vee \left((\neg \mathcal{C}) \wedge \mathcal{D} \right) \right) \right) \Leftrightarrow (\mathcal{E} \Rightarrow \mathcal{F})$$

Le connecteur \Leftrightarrow de cette formule est le moins prioritaire; il s'applique donc en dernier, et nous avons entouré les deux formules à gauche et à droite avec les parenthèses les plus externes. À gauche de \Leftrightarrow , le connecteur le moins prioritaire est \Rightarrow ; nous avons entouré son opérande à droite de parenthèses, et ainsi de suite, en procédant toujours du moins prioritaire au plus prioritaire.

L'élimination des parenthèses en utilisant la priorité des opérateurs vise à simplifier la lecture des formules. Par exemple, en arithmétique, on écrit $a + b * c$ au lieu de $a + (b * c)$, parce que c'est plus simple à lire. Toutefois, comme on peut le voir avec la formule (1.1), l'élimination des parenthèses rend parfois une formule complexe plus difficile à lire. Il s'agit d'utiliser son jugement pour choisir l'une ou l'autre des représentations. Parfois, nous écrirons des formules assez longues, et nous les écrirons sur plusieurs lignes, avec une *indentation*, un peu comme dans un programme, afin d'en faciliter la lecture. Ainsi, la formule (1.1) peut s'écrire sur plusieurs lignes avec indentation, tel qu'illustré dans la figure 1.1. Les opérandes d'un connecteur logique sont indentées à droite et les parenthèses correspondantes sont alignées sur la même colonne, tel qu'illustré par les décorations “:” que nous avons ajoutées à titre illustratif. On peut voir plus facilement que le connecteur principal de cette formule est une équivalence, qui s'applique à une implication à gauche, et une implication

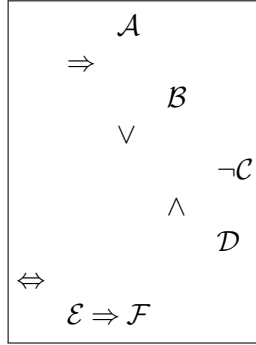


Figure 1.2: Représentation indentée, sans parenthèse, de la formule (1.1)

à droite, et ainsi de suite. On peut aussi omettre les parenthèses, ce qui donne une formule plus simple à lire, tel qu’illustré dans la figure 1.2; les parenthèses ne sont pas nécessaires, à cause de la priorité des opérateurs. Notons que certains langages de programmation comme Python utilisent l’indentation au lieu de parenthèse; dans ce cas, l’indentation *définit* la priorité. Ce n’est pas le cas en logique dans la plupart des outils : l’indentation ne définit pas la priorité des opérateurs; elle est utilisée à titre informatif.

1.1.2 Tables de vérité des connecteurs logiques

La valeur de vérité d’une formule, qui est soit **vrai**, soit **faux**, peut-être calculée à l’aide des tables de vérité suivantes. Elles se lisent de la même manière que les tables de calculs des opérateurs arithmétiques que l’on vous a enseignées dès l’école primaire, comme pour les additions et les multiplications. Par soucis de concision, on utilise 0 et 1 pour les valeurs **faux** et **vrai**, respectivement. Les lignes indiquent les valeurs de l’opérande de gauche, alors que les colonnes indiquent les valeurs de l’opérande de droite (cela a une importance pour les connecteurs \Rightarrow et \Leftarrow , qui ne sont pas commutatifs, alors que les autres opérateurs le sont).

\neg	0	1	\wedge	0	1	\vee	0	1	\Rightarrow	0	1	\Leftarrow	0	1	\Leftrightarrow	0	1	\oplus	0	1
1	0	1	0	0	0	0	0	1	0	1	1	0	0	1	0	1	0	0	0	1
0	1	0	1	1	1	1	1	1	1	0	1	1	1	0	1	0	1	1	1	0

Voici une représentation équivalente et plus compacte des tables de vérité des opérateurs binaires.

X_1	X_2	\wedge	\vee	\Rightarrow	\Leftarrow	\Leftrightarrow	\oplus
0	0	0	0	1	1	1	0
0	1	0	1	1	0	0	1
1	0	0	1	0	1	0	1
1	1	1	1	1	1	1	0

La valeur de vérité d’une formule propositionnelle peut être calculée si la valeur de chaque variable propositionnelle qu’elle contient est connue. Nous noterons $X := 0$ le fait que la valeur de la variable X est 0. Notons que $X := 0$ n’est pas une formule propositionnelle, car le symbole “:=” ne fait pas partie de la syntaxe de la logique propositionnelle. La valeur de la formule $X_1 \vee (X_2 \wedge X_3)$ est 0 pour $X_1 := 0, X_2 := 1, X_3 := 0$. On peut aussi calculer la valeur d’une formule propositionnelle pour toutes les combinaisons des valeurs de ses variables. Voici le calcul pour la formule $X_1 \vee (X_2 \wedge X_3)$.

Cela est plutôt fastidieux, car le nombre de combinaisons est égal à 2^n pour une formule contenant n variables propositionnelles.

no	X_1	X_2	X_3	$X_2 \wedge X_3$	$X_1 \vee (X_2 \wedge X_3)$
1	0	0	0	0	0
2	0	0	1	0	0
3	0	1	0	0	0
4	0	1	1	1	1
5	1	0	0	0	1
6	1	0	1	0	1
7	1	1	0	0	1
8	1	1	1	1	1

Le problème de la satisfaction d'une formule propositionnelle est un problème classique en algorithmique. Il s'agit de trouver une combinaison de valeur des variables d'une formule de sorte que la formule est vrai. Dans l'exemple précédent, les lignes 4 à 8 sont des solutions pour la satisfaction de la formule $X_1 \vee (X_2 \wedge X_3)$. Ce problème de la satisfaction d'une formule propositionnelle est très important pour les outils de vérification automatique des logiciels (appelés en anglais *model checking tools*) et la preuve automatique de théorèmes. C'est un problème dit NP-complet (ce qui veut dire, pour simplifier, que le meilleur algorithme connu actuellement exige un temps de calcul qui augmente de manière exponentielle en fonction de la taille de la formule, qu'une solution peut-être vérifiée en temps polynomial, et que si un jour on trouve un algorithme polynomial pour calculer une solution, cet algorithme pourra aussi résoudre tous les autres problèmes NP-complet). Ce théorème de Stephen A. Cook, considéré comme un théorème fondamental en théorie de la complexité, lui mérita (avec d'autres contributions) le prix Alan Turing en 1982. Il existe une compétition annuelle en informatique pour déterminer les meilleurs algorithmes de satisfaction de formules propositionnelles.

1.1.3 Expression de faits en logique propositionnelle

La logique a été inventée pour exprimer des faits, des connaissances avec plus de précision que le *langage naturel* (par exemple, le français, l'anglais, etc). L'humain utilise le langage naturel pour communiquer, mais le langage naturel comporte souvent des ambiguïtés, c'est-à-dire qu'il permet des interprétations différentes d'un ensemble d'énoncés, en fonction de la personne qui les considère. La logique permet d'éviter toute ambiguïté et de raisonner de manière rigoureuse. Il est difficile, voire impossible, de proposer une traduction systématique d'un langage naturel vers des formules de logique, à cause du trop grand nombre de façons d'exprimer des faits en langage naturel et de l'ambiguïté omniprésente dans le langage naturel. On peut toutefois identifier quelques correspondances classiques. Nous vous présenterons un résumé très simple de quelques correspondances en logique d'énoncés en langage naturel. Nous utilisons la police suivante pour identifier les variables propositionnelles: *variablePropositionnelle*. Par exemple, supposons qu'on s'intéresse au fait qu'une personne ait la grippe. On peut utiliser la variable *aLaGrippe*; si cette variable vaut *faux*, alors la personne n'a pas la grippe; si cette variable vaut *vrai*, alors on considère que la personne a la grippe. Puisqu'une variable propositionnelle est aussi une formule propositionnelle, alors énoncer le fait (c'est-à-dire la formule)

$$aLaGrippe \tag{1.2}$$

signifie que la personne a la grippe. Énoncer le fait

$$\neg aLaGrippe \tag{1.3}$$

signifie que la personne n'a pas la grippe. Dans un langage de programmation comme le C ou le Java, on aurait plutôt écrit "*aLaGrippe* == 1" et "*aLaGrippe* == 0". On voit que la logique propositionnelle a une notation distincte de la programmation. Bien sûr, on pourrait aussi écrire en logique propositionnelle quelque chose qui ressemble à la programmation, en utilisant l'équivalence, qui est analogue à l'égalité:

$$aLaGrippe \Leftrightarrow \text{vrai} \quad (1.4)$$

et

$$aLaGrippe \Leftrightarrow \text{faux} \quad (1.5)$$

(1.2) est équivalent à (1.4), et (1.3) est équivalent à (1.5), mais elles sont plus compliquées que nécessaire. Les sous-section suivantes présentent brièvement quelques traductions pour chaque connecteur.

1.1.3.1 Négation (\neg)

C'est la négation au sens usuel en langage naturel. La formule (1.3) est un exemple de négation.

1.1.3.2 Conjonction (\wedge)

La conjonction est généralement, mais pas toujours, équivalente au "et" du langage naturel. La phrase suivante

$$\text{"L'étudiant a réussi les cours IFT001 et IFT002."} \quad (1.6)$$

se traduit comme suit:

$$IFT001Réussi \wedge IFT002Réussi$$

Toutefois, la phrase

$$\text{"Dans la table d'hôte, vous avez le choix entre la soupe **et** le dessert."} \quad (1.7)$$

ne signifie pas que la soupe et le dessert sont inclus dans le forfait de la table d'hôte. Il s'agit de l'un ou l'autre. Cela serait plutôt représenté par un "ou exclusif" \oplus .

$$\text{soupe} \oplus \text{dessert}$$

1.1.3.3 Disjonction (\vee)

La disjonction est généralement, mais pas toujours, équivalente au "ou" du langage naturel. La phrase suivante illustre un cas classique de disjonction, au sens où les deux options peuvent être vraies en même temps.

$$\text{"Pour s'inscrire au cours IFT003, il faut avoir réussi IFT001 ou IFT002."} \quad (1.8)$$

La partie "il faut avoir réussi ..." se traduit par une disjonction.

$$IFT001Réussi \vee IFT002Réussi$$

Une personne qui a réussi IFT001 seulement, IFT002 seulement, ou bien les deux, peut s'inscrire à IFT003.

La phrase suivante, qui est presque identique à la phrase (1.7), mais avec un "ou" au lieu d'un "et", sera aussi traduite avec un "ou exclusif".

$$\text{"Dans la table d'hôte, vous avez le choix entre la soupe **ou** le dessert."} \quad (1.9)$$

On constate que les ambiguïtés surgissent très rapidement en langage naturel.

1.1.3.4 Implication (\Rightarrow)

L'implication correspond typiquement à la forme “si ... alors ...”, mais elle est aussi représentée sous une multitude d'autres formes. Par exemple,

$$\text{“Une personne qui a la grippe fait de la fièvre.”} \quad (1.10)$$

Cette phrase est traduite comme suit:

$$aLaGrippe \Rightarrow faitDeLaFièvre$$

Le connecteur \Rightarrow est souvent décrit en terme de *condition nécessaire* ou *condition suffisante*. Ainsi, dans une formule de la forme $\mathcal{A} \Rightarrow \mathcal{B}$, on dit que \mathcal{B} est une *condition nécessaire* pour \mathcal{A} , et, de manière duale, que \mathcal{A} est une *condition suffisante* pour \mathcal{B} . Par exemple, on dit que *faitDeLaFièvre* est une *condition nécessaire* pour *aLaGrippe*; faire de la fièvre n'est pas suffisant pour diagnostiquer une grippe; on peut faire de la fièvre pour plusieurs autres raisons (par exemple, pour une infection bactérienne, alors que la grippe est une infection virale, ou à la suite d'une insolation). De manière duale, on dit que *aLaGrippe* est une *condition suffisante* pour *faitDeLaFièvre*, parce que on est sûr qu'une personne ayant la grippe fait de la fièvre.

Les mathématiciens utilisent aussi la formulation “seulement si” pour représenter les conditions nécessaires et suffisantes. La phrase “ A seulement si B ” indique que B est une condition nécessaire pour A et, de manière duale, que A est une condition suffisante pour B . Par exemple, la phrase suivante est équivalente à la phrase (1.10).

$$\text{“Une personne a la grippe seulement si elle fait de la fièvre.”} \quad (1.11)$$

La phrase (1.8) se traduit en entier par une implication.

$$IFT001Réussi \vee IFT002Réussi \Rightarrow PeutSinscrireIFT003$$

Grâce aux tautologies présentées dans la prochaine section (Section 1.3.1), cette phrase peut aussi s'écrire, de manière équivalente, comme suit:

$$(IFT001Réussi \Rightarrow PeutSinscrireIFT003) \wedge (IFT002Réussi \Rightarrow PeutSinscrireIFT003)$$

On dit que *IFT001Réussi* est une condition suffisante pour *PeutSinscrireIFT003* (c'est-à-dire, que réussir IFT001 est une condition suffisante pour s'inscrire à IFT003). *IFT002Réussi* est aussi une condition suffisante pour *PeutSinscrireIFT003*. On remarque que réussir un des cours IFT001 et IFT002 est suffisant pour s'inscrire à IFT003. On remarque que le dual, “*PeutSinscrireIFT003* est une condition nécessaire pour *IFT001Réussi*”, semble un peu bizarre. Cela est techniquement correct d'un point de vue mathématique, mais pas usuel dans le langage courant, car cela porterait à confusion; personne ne dira que “pouvoir s'inscrire à un cours” est une condition nécessaire pour la réussite d'un cours préalable! La formule suivante, qui n'a pas le même sens, permet de mieux visualiser la distinction entre condition nécessaire et suffisante pour ce contexte.

$$IFT003Réussi \Rightarrow (IFT001Réussi \vee IFT002Réussi)$$

On voit que “réussir IFT001 ou réussir IFT002” est une condition nécessaire pour réussir IFT003.

La formule suivante permet aussi de voir la distinction entre une conjonction et une disjonction.

$$IFT001Réussi \wedge IFT002Réussi \Rightarrow PeutSinscrireIFT003 \quad (1.12)$$

Cette formule indique que si on a réussi les deux cours IFT001 et IFT002, on peut s'inscrire à IFT003.

On pourrait se poser la question suivante: y a-t-il d'autres conditions qui permettent de s'inscrire à IFT003? Une formule contenant une implication ne nous le dit pas. La formule peut être vraie, avec *PeutSinscrireIFT003* vraie et *IFT001Réussi* et *IFT002Réussi* fausses, comme le montre la table de vérité suivante, où X_1 , X_2 et X_3 représentent respectivement *IFT001Réussi*, *IFT002Réussi* et *PeutSinscrireIFT003*.

no	X_1	X_2	X_3	$X_1 \wedge X_2$	$(X_1 \wedge X_2) \Rightarrow X_3$
1	0	0	0	0	1
2	0	0	1	0	1
3	0	1	0	0	1
4	0	1	1	0	1
5	1	0	0	0	1
6	1	0	1	0	1
7	1	1	0	1	0
8	1	1	1	1	1

On remarque que la formule (1.12) est vraie dans tous les cas, sauf pour la ligne 7, où *IFT001Réussi* et *IFT002Réussi* sont vraies, et *PeutSinscrireIFT003* est fausse. Donc, si l'université affirme que la formule (1.12) est vraie, alors on est dans les cas 1 à 6 et 8. Si un étudiant a réussi IFT001 et IFT002, alors l'université doit lui permettre de s'inscrire à IFT003, si elle veut respecter sa formule, car sinon la formule est fausse. On remarque aussi que cette formule indique aussi que l'université peut permettre à l'étudiant de s'inscrire à IFT003 même s'il n'a pas réussi IFT001 ou IFT002; ce cas est représenté par les lignes 1 à 6. Si on veut exclure la possibilité d'inscrire un étudiant à IFT003 sans avoir réussi IFT001 et IFT002, alors on utilisera plutôt une équivalence, au lieu d'une implication.

1.1.3.5 Équivalence (\Leftrightarrow)

L'équivalence correspond typiquement à la forme "...si, et seulement si, ...", que l'on représente souvent par l'abréviation *ssi* dans les textes mathématiques. Par exemple, la phrase

“On est admis au bac en informatique ssi on a réussi les cours collégiaux suivants: *MAT103*, *MAT105* et *MAT203*.” (1.13)

$$\text{admisBaInfo} \Leftrightarrow (\text{MAT103Réussi} \wedge \text{MAT105Réussi} \wedge \text{MAT203Réussi}) \quad (1.14)$$

Pour une formule de la forme $\mathcal{A} \Leftrightarrow \mathcal{B}$, on dit que \mathcal{A} est une *condition nécessaire et suffisante* pour \mathcal{B} , et vice-versa, car l'équivalence est commutative (donc, \mathcal{B} est aussi une condition nécessaire et suffisante pour \mathcal{A}). Dans l'exemple (1.14), on dit que

$$\text{MAT103Réussi} \wedge \text{MAT105Réussi} \wedge \text{MAT203Réussi}$$

est une condition nécessaire et suffisante pour être admis au baccalauréat en informatique.

Si on veut indiquer que la seule façon de s'inscrire à IFT003 est d'avoir réussi IFT001 et IFT002, on écrit alors:

$$(\text{IFT001Réussi} \wedge \text{IFT002Réussi}) \Leftrightarrow \text{PeutSinscrireIFT003}$$

La phrase suivante est aussi un exemple d'équivalence, mais sans utiliser la forme “si et seulement si”. Il s'agit d'une définition d'un terme; les définitions sont typiquement représentées par une

équivalence.

“Les mammifères sont caractérisés essentiellement par l’allaitement des jeunes, un cœur à quatre cavités, un système nerveux et encéphalique développé, une homéothermie et une respiration pulmonaire.” (1.15)

se traduit comme suit:

$$\begin{aligned} \text{mammifère} \Leftrightarrow & \text{allaitement} \wedge \text{cœur} \wedge \text{Cavités} \wedge \\ & \text{systèmeNerveuxEncéphaliqueDéveloppé} \wedge \text{homéothermie} \wedge \\ & \text{respirationPulmonaire} \end{aligned}$$

1.2 Logique du premier ordre

La logique du premier ordre est plus riche que la logique propositionnelle. Elle permet d’utiliser des variables d’autres types que les variables booléennes, comme par exemple les naturels, les entiers, les ensembles, les relations, les fonctions, . . . Elle permet aussi d’utiliser des fonctions booléennes, que l’on appelle des *prédicats*.

1.2.1 Syntaxe

Les formules de la logique du premier ordre sont construites en appliquant des prédicats à des termes pour obtenir des formules atomiques, et en combinant des formules atomiques à l’aide de connecteurs logiques et de quantificateurs.

Définition 2 Un *terme* est une suite de symboles construite à partir des règles suivantes:

1. si x est une variable, alors x est un terme;
2. si c est une constante, alors c est un terme;
3. si f est un symbole de fonction d’arité (i.e., nb de paramètres) $n > 0$ et que t_1, \dots, t_n sont des termes, alors $f(t_1, \dots, t_n)$ est un terme.

□

Un terme est une expression qui représente un objet du monde que l’on veut décrire. En arithmétique, les expressions suivantes sont des exemples de terme :

$$\begin{array}{ccc} 0 & 0 + x & 1 + (2 * x) - y \\ \log(x + 4) & \sqrt{x + y * 2} & \sin x^2 \end{array}$$

Dans la définition de la syntaxe des termes, f représente un opérateur comme $+$, $-$, $*$, $\sqrt{\quad}$, \log , \sin , . . . ; pour simplifier la définition de la syntaxe, on utilise la forme générale $f(t_1, \dots, t_n)$, alors que la syntaxe usuelle est souvent en format *infixe*, comme $x + y$, au lieu de $+(x, y)$.

Remarque 2 Par souci de généralité, certains auteurs classifient les constantes comme des symboles de fonction d’arité 0, c’est-à-dire qu’une constante est une fonction qui ne prend pas de paramètre et qui retourne toujours la même valeur.

Définition 3 Une *formule atomique* est une suite de symboles construite à partir des règles suivantes:

1. **vrai** et **faux** sont des formules atomiques;
2. si r est un prédicat (i.e., une fonction booléenne) d'arité $n > 0$ et que t_1, \dots, t_n sont des termes, alors $r(t_1, \dots, t_n)$ est une formule atomique.

□

Un prédicat est une fonction qui retourne **vrai** ou **faux**. En arithmétique, les opérateurs suivants sont des exemples de prédicats :

$$= \quad < \quad > \quad \geq \quad \leq$$

Les expressions suivantes sont des formules atomiques en arithmétique.

$$0 = 0 \quad 1 + x > x \quad \log(x + 1) \leq x + 1$$

Définition 4 Une *formule* est une suite de symboles construite à partir des règles suivantes:

1. une formule atomique est une formule;
2. si \mathcal{A} est une formule, alors $\neg\mathcal{A}$ est une formule.
3. si \mathcal{A} et \mathcal{B} sont des formules, alors

- $\mathcal{A} \wedge \mathcal{B}$
- $\mathcal{A} \vee \mathcal{B}$
- $\mathcal{A} \oplus \mathcal{B}$
- $\mathcal{A} \Rightarrow \mathcal{B}$
- $\mathcal{A} \Leftarrow \mathcal{B}$
- $\mathcal{A} \Leftrightarrow \mathcal{B}$

sont des formules;

4. si \mathcal{A} est une formule, alors (\mathcal{A}) est une formule
5. si x est une variable et \mathcal{A} est une formule, alors

- $\forall x \cdot \mathcal{A}$
- $\exists x \cdot \mathcal{A}$

sont des formules.

□

Les symboles \exists, \forall sont appelés des *quantificateurs* (*existentiel* pour \exists et *universel* pour \forall). Une variable en logique du premier ordre représente un objet du monde que l'on désire décrire. Dans le langage du logiciel TARSKIODES, une variable désigne un objet (c'est-à-dire un cube, un tétraèdre ou un dodécaèdre). En arithmétique, une variable désigne un nombre. La variable en logique du premier ordre joue donc un rôle différent de la variable en logique propositionnelle, qui peut seulement désigner une valeur de vérité (**vrai** et **faux**). Nous utilisons le symbole " \equiv " pour indiquer une équivalence syntaxique, c'est-à-dire que $e_1 \equiv e_2$ signifie que e_1 est une abréviation pour e_2 . Il s'agit d'une forme d'égalité.

Remarque 3 Soit T un ensemble. Certains auteurs utilisent les abréviations suivantes:

$$\begin{aligned}
\exists x : T \cdot \mathcal{A} &\equiv \exists x \cdot x \in T \wedge \mathcal{A} \\
\exists x : T \mid \mathcal{A} \cdot \mathcal{B} &\equiv \exists x \cdot x \in T \wedge \mathcal{A} \wedge \mathcal{B} \\
\forall x : T \cdot \mathcal{A} &\equiv \forall x \cdot x \in T \Rightarrow \mathcal{A} \\
\forall x : T \mid \mathcal{A} \cdot \mathcal{B} &\equiv \forall x \cdot x \in T \wedge \mathcal{A} \Rightarrow \mathcal{B} \\
\forall x, y \cdot \mathcal{A} &\equiv \forall x \cdot \forall y \cdot \mathcal{A} \\
\forall x, y : T \cdot \mathcal{A} &\equiv \forall x \cdot \forall y \cdot x \in T \wedge y \in T \Rightarrow \mathcal{A} \\
\exists x, y \cdot \mathcal{A} &\equiv \exists x \cdot \forall y \cdot \mathcal{A} \\
\exists x, y : T \cdot \mathcal{A} &\equiv \exists x \cdot \forall y \cdot x \in T \wedge y \in T \Rightarrow \mathcal{A}
\end{aligned}$$

Remarque 4 Dans ces notes de cours, les quantificateurs \forall, \exists ont la même priorité, et leur priorité est plus faible que les connecteurs logiques, ce qui donne les priorités suivantes, 1 étant la plus forte priorité, et les opérateurs sur une même ligne ayant la même priorité:

1. \neg ,
2. \wedge ,
3. \vee, \oplus
4. \Rightarrow, \Leftarrow
5. \Leftrightarrow
6. \forall, \exists

Par exemple, la formule suivante

$$\forall x \cdot \mathcal{A} \Rightarrow \exists y \cdot \mathcal{A} \wedge \neg \mathcal{B} \vee \mathcal{C}$$

est équivalente, après introduction des parenthèses qui représentent la priorité des opérateurs, à la formule suivante :

$$\forall x \cdot \left(\mathcal{A} \Rightarrow \exists y \cdot \left((\mathcal{A} \wedge \neg \mathcal{B}) \vee \mathcal{C} \right) \right)$$

Définition 5 Dans les formules $\forall x \cdot \mathcal{A}$ et $\exists x \cdot \mathcal{A}$, les occurrences de x dans \mathcal{A} sont dites *liées* aux quantificateurs $\forall x$ et $\exists x$, respectivement. Si une occurrence est sous la portée de plusieurs quantificateurs, elle est alors liée au quantificateur le plus près (i.e., le plus imbriqué). \square

Définition 6 Dans une formule \mathcal{A} , une occurrences de x est dite *libre* si elle n'est pas liée à un quantificateur. \square

Définition 7 Une formule est dites *fermée* ssi elle ne contient aucune variable libre. \square

Dans la formule suivante, la variable x est liée et la variable y est libre. Cette formule n'est pas fermée, à cause de y .

$$\forall x \cdot x > 0 \wedge y = 1$$

Dans la formule suivante, la première occurrence de la variable x est liée (i.e., $x > 0$), mais la deuxième est libre (i.e., $x = 1$), car elle est hors de la portée du quantificateur \forall . Donc, la variable x est à la fois libre et liée dans cette formule. Cette formule n'est pas fermée, à cause de l'occurrence libre de x .

$$(\forall x \cdot x > 0) \wedge x = 1$$

Les deux occurrences de x ne désignent donc pas la même chose.

Dans l'exemple suivant, les variables x et y sont liées de plusieurs manières, indiquées par un indice en rouge. Cette formule est fermée, car toutes les variables sont liées.

$$\forall x_1, y_1 \cdot \text{pere}(x_1, y_1) \Rightarrow \left(\text{parent}(x_1, y_1) \wedge (\forall x_2, y_2 \cdot \text{mere}(x_2, y_2) \Rightarrow \text{parent}(x_2, y_2)) \wedge \text{homme}(x_1) \right) \quad (1.16)$$

On peut considérer les occurrences de x identifiées par x_1 comme distinctes de celles identifiées par x_2 . On verra, avec les lois de la logique et de la substitution, qu'il est plus clair d'utiliser la formule équivalente suivante, où les occurrences de x sont regroupées

$$\left(\forall x_1, y_1 \cdot \text{pere}(x_1, y_1) \Rightarrow (\text{parent}(x_1, y_1) \wedge \text{homme}(x_1)) \right) \wedge \left(\forall x_2, y_2 \cdot \text{mere}(x_2, y_2) \Rightarrow \text{parent}(x_2, y_2) \right) \quad (1.17)$$

Il est très fréquent d'utiliser la même variable plusieurs fois avec des quantificateurs différents. En pratique, chaque quantificateur définit une "nouvelle" variable.

La notion de variable libre et liée est très importante. Quand une formule contient une variable libre, il n'est pas possible d'évaluer la valeur de vérité de cette formule. Nous verrons plus loin dans ce chapitre que le logiciel TarskiUdeS ne peut évaluer une formule si elle contient une variable libre.

Définition 8 L'expression $\mathcal{A}[x := t]$, appelée la *substitution* de x par t dans \mathcal{A} , où \mathcal{A} est une formule, x est une variable et t est un terme, dénote la substitution de toutes les occurrences libres de x dans \mathcal{A} par t . Le terme t **peut** contenir la variable x . La substitution est dite *valide* si t ne contient pas de variables qui deviennent liées dans le résultat de la substitution.

Si une substitution est invalide, on peut toujours renommer dans \mathcal{A} les variables des quantificateurs en choisissant une variable qui n'apparaît pas dans \mathcal{A} . On peut renommer une variable quantifiée en utilisant la substitution. Soit x' une variable n'apparaissant pas dans la formule \mathcal{A} . Alors, la formule $\forall x \cdot \mathcal{A}$ est équivalente à la formule $\forall x' \cdot (\mathcal{A}[x := x'])$, et la formule $\exists x \cdot \mathcal{A}$ est équivalente à la formule $\exists x' \cdot (\mathcal{A}[x := x'])$. \square

La substitution a une priorité plus forte que tous les connecteurs logiques. Voici le problème engendré par une application **incorrecte** d'une substitution². La formule suivante est valide :

$$n \in \mathbb{N} \Rightarrow \neg \forall m \cdot (m \in \mathbb{N} \Rightarrow m = n)$$

On peut lui appliquer une substitution pour l'utiliser pour $n := m$

$$(n \in \mathbb{N} \Rightarrow \neg \forall m \cdot (m \in \mathbb{N} \Rightarrow m = n))[n := m]$$

Ce qui donne comme résultat

$$m \in \mathbb{N} \Rightarrow \neg \forall m \cdot (m \in \mathbb{N} \Rightarrow m = m)$$

ce qui cause un problème : l'occurrence libre de n devient liée et la formule résultante est fausse. Pour rendre la substitution valide, on peut remplacer $\forall m$ par $\forall z$

$$\begin{aligned} & (n \in \mathbb{N} \Rightarrow \neg \forall z \cdot (m \in \mathbb{N} \Rightarrow m = n))[m := z][n := m] \\ \equiv & (n \in \mathbb{N} \Rightarrow \neg \forall z \cdot (z \in \mathbb{N} \Rightarrow z = n))[n := m] \\ \equiv & m \in \mathbb{N} \Rightarrow \neg \forall z \cdot (z \in \mathbb{N} \Rightarrow z = m) \end{aligned}$$

Remarque 5 Certains auteurs utilisent les notations alternatives suivantes pour la substitution : $\mathcal{A}[x \setminus t]$, et $\mathcal{A}[t/x]$.

²Source: Jean-Raymond Abrial, présentation du langage B

1.2.2 Interprétation des quantificateurs

Une formule de la forme “ $\forall x \cdot \mathcal{A}$ ” signifie que la formule \mathcal{A} est vraie pour toutes les valeurs possibles de x , c’est-à-dire tous les objets du monde. Supposons que les objets du monde sont représentés par l’ensemble $\{o_1, \dots, o_n\}$. La valeur de vérité de $\forall x \cdot \mathcal{A}$ est donné par la formule suivante:

$$(\forall x \cdot \mathcal{A}) \Leftrightarrow (\mathcal{A}[x := o_1] \wedge \dots \wedge \mathcal{A}[x := o_n])$$

Il peut y avoir une infinité d’objets dans un monde, comme par exemple en arithmétique. L’ensemble des nombres naturels $\mathbb{N} = \{0, 1, 2, \dots\}$ peut être utilisé dans des formules quantifiées, et on obtient alors une conjonction infinie, d’où l’intérêt d’utiliser le symbole \forall pour désigner une conjonction infinie.

$$(\forall x \cdot x + 0 = x) \Leftrightarrow ((x + 0 = x)[x := 0] \wedge (x + 0 = x)[x := 1] \wedge (x + 0 = x)[x := 2] \wedge \dots)$$

En mathématiques, il arrive souvent que l’on donne une formule sans utiliser de quantification. Par exemple, la loi suivante de l’arithmétique

$$x + 0 = x \tag{1.18}$$

ne contient aucun quantificateur, mais, par convention, on sait que cela s’applique à tout nombre x , donc que la formule réelle est plutôt $\forall x \cdot x + 0 = x$. La convention usuelle en mathématiques est que les variables libres d’une formule sont implicitement quantifiées universellement.

Supposons que l’on vous donne deux formules :

$$x + y = 4 \tag{1.19}$$

$$x = y \tag{1.20}$$

Il y a maintenant une ambiguïté ici : on ne sait pas si les variables x et y des formules (1.19) et (1.20) denotent le même nombre dans chaque formule, i.e., si la valeur de x dans la première formule doit être la même que celle de la deuxième formule. Autrement dit, est-ce que x dénote le même nombre dans les deux formules. Si on prend en compte la convention précédente, qui indique qu’une variable sans quantificateur est implicitement quantifiée universellement, alors on obtient les formules suivantes :

$$\forall x \cdot \forall y \cdot x + y = 4 \tag{1.21}$$

$$\forall x \cdot \forall y \cdot x = y \tag{1.22}$$

Les formules (1.21) et (1.22) sont fausses en arithmétique (i.e., la somme de deux nombres quelconques ne donne pas 4, et deux nombres ne sont pas toujours égaux). Les formules (1.19) et (1.20) peuvent être satisfaites pour $x := 2$ et $y := 2$. Donc, quand on donne des formules, il faut bien expliciter le contexte de travail. Dans la formule (1.18), on donne une loi de l’arithmétique, et, implicitement, la variable x est quantifiée universellement. Les formules (1.19) et 1.20 seraient plutôt utilisée dans un contexte où on *cherche* les valeurs de x et y qui satisfont les deux formules en même temps. Lorsqu’on effectue des preuves, on doit gérer les quantificateurs avec beaucoup de rigueur. Si une variable est libre dans plusieurs hypothèses d’une preuve, c’est qu’il s’agit de la même variable et qu’elle désigne le même objet.

Une formule de la forme “ $\exists x \cdot \mathcal{A}$ ” signifie que la formule \mathcal{A} est vrai pour *au moins une* valeur de x , donc au moins un objet du monde. La valeur de vérité de $\exists x \cdot \mathcal{A}$ est donné par la formule suivante:

$$(\exists x \cdot \mathcal{A}) \Leftrightarrow (\mathcal{A}[x := o_1] \vee \dots \vee \mathcal{A}[x := o_n])$$

Nous allons illustrer les quantificateurs en utilisant le logiciel TARSKIUDÉS.

Prédicat	Signification
$\text{Triangle}(x)$	x est un triangle
$\text{Square}(x)$	x est un carré
$\text{Pentagon}(x)$	x est un pentagone
$\text{Small}(x)$	x est petit
$\text{Medium}(x)$	x est moyen
$\text{Large}(x)$	x est grand
$\text{Smaller}(x, y)$	x est strictement plus petit que y
$\text{SameSize}(x, y)$	x est de même taille que y
$\text{LeftOf}(x, y)$	x est à gauche de y
$\text{SameRow}(x, y)$	x est sur la même ligne que y
$\text{SameCol}(x, y)$	x est sur la même colonne que y
$\text{Between}(x, y, z)$	x est entre y et z sur une même ligne, colonne ou diagonale
$x = y$	x et y représentent le même objet
$x \neq y$	x et y représentent des objets distincts

Table 1.1: Prédicats du langage TARSKIUNDES

1.2.3 TARSKIUNDES

Le logiciel TARSKIUNDES permet de se familiariser avec la logique du premier ordre. Il a été développé à l'Université de Sherbrooke par Tristen Bronson et Laurent Beauchemin dans le cadre d'un projet informatique, à partir d'une version initiale développée par Robert Stärk. Il est inspiré du logiciel Tarski's World [2] développé à Stanford par les logiciens Jon Barwise et John Etchemendy³. TARSKIUNDES comporte plusieurs améliorations qui facilite son utilisation et la vérification d'une formule. Il est nommé ainsi en l'honneur du mathématicien et logicien Alfred Tarski⁴

1.2.3.1 Syntaxe du langage TARSKIUNDES

Le langage porte sur un monde composé d'objets dans un plan. Il y a trois types d'objets: triangle, carré et pentagone. Chaque objet a une taille: petit, moyen et grand. On peut associer une lettre à un objet, en double-cliquant l'objet, ce qui permet de référer à cet objet dans une formule. Le langage contient les prédicats décrits dans le tableau 1.1, où les variables x, y, z dénotent des objets. On utilise la syntaxe donnée au tableau 1.2 pour représenter les connecteurs logiques dans TARSKIUNDES, avec leur priorité, la priorité 1 étant la plus forte. Les formules quantifiées doivent être parenthésées.

$!x. (\text{Large}(x) \ \& \ \text{Square}(x))$

³En plus d'être logicien et philosophe, John Etchemendy fut aussi recteur de l'Université Stanford de 2010 à 2017, comme quoi la logique mène à tout!

⁴Alfred Tarski (1901–1983) fut l'un des logiciens les plus importants du vingtième siècle, et un mathématicien prolifique. Juif polonais, il fut sauvé bien malgré lui de l'extermination systématique des juifs polonais durant la deuxième guerre mondiale, en prenant le dernier bateau qui quitta la Pologne pour les États-Unis en août 1939, étant invité à prononcer une conférence au Symposium de l'Unité de la science à Harvard. La Pologne fut envahie par l'Allemagne et la Russie en septembre 1939, ce qui marqua le début de la deuxième guerre mondiale. Tarski devint professeur à Berkeley en 1942 et y demeura jusqu'à sa mort, continuant à superviser des étudiants au doctorat même après sa retraite. Il dirigea au doctorat un nombre important de femmes, ce qui est remarquable pour l'époque. Tarski fit d'importantes contributions en logique, en algèbre et en théorie des ensembles, entre autres.

Symbole mathématique	TARSKIUDES	Priorité
$\forall x \cdot$	<code>!x.(...)</code>	0
$\exists x \cdot$	<code>#x.(...)</code>	0
\neg	<code>not</code>	1
\wedge	<code>&</code>	2
\vee	<code>or</code>	3
\Rightarrow	<code>=></code>	4
\Leftrightarrow	<code><=></code>	5

Table 1.2: Syntaxe du langage TARSKIUDES

ce qui représente la formule mathématique suivante dans la notation usuelle des notes de cours:

$$\forall x \cdot \text{Large}(x) \wedge \text{Square}(x)$$

Si les parenthèses étaient omises, comme dans l'exemple ci-dessous, une erreur de syntaxe serait générée (*x is not defined*).

Large(x)

La variable x est libre dans la formule `Large(x)`, car il n'y a pas de quantificateur qui introduit la variable x . TARSKIUDES ne peut pas évaluer les formules non fermées, c'est-à-dire les formules qui contiennent une variable libre (voir définition 6). Les autres opérateurs utilisent la précedence usuelle des notes de cours. Par exemple, la formule

`!x.(Square(x) & Large(x) => not Triangle(x) or Small(x))`

est parenthésée implicitement comme suit, à cause la priorité des opérateurs

`!x.((Square(x) & Large(x)) => ((not Triangle(x)) or Small(x)))`




Afin de simplifier la lecture des formules, vous pouvez utiliser le formatage automatique des formules, avec la commande CTRL-SHIFT-F. Par exemple, la formule ci-dessus sera indentée comme suit:

```
!x.
(
  Square(x)
  &
  Large(x)
=>
  not
  Triangle(x)
or
  Small(x)
)
```

On utilise des parenthèses pour imposer un ordre d'évaluation des opérateurs, comme on le fait en arithmétique. La figure 1.3 illustre un monde où tous les types d'objets et de tailles sont illustrés.

1.2.3.2 Vérifier votre compréhension des formules

Tarski UdeS vous permet de vérifier votre compréhension d'une formule et de la tester. Pour tester, on veut explorer différents cas; en particulier, il faut explorer des cas où la formule est vraie, mais aussi des cas où la formule est fausse. Par exemple, si on vous demande de formaliser la phrase "Tous les objets sont grands", et que vous produisez la formule $\forall x \cdot x = x$ en réponse, alors effectivement la formule est vraie dans un monde où tous les objets sont grands, mais elle est aussi vraie dans n'importe quel monde, entre autres ceux où au moins un objet n'est pas grand. Pour vérifier une formule, il faut donc la tester sur des mondes où elle devrait être vraie et sur des mondes où elle devrait être fausse, afin de s'assurer que la formule représente exactement les mondes où la phrase devrait être vraie.

Pour chaque monde, vous devez indiquer si la formule devrait être vraie ou fausse, avec le bouton "Formula is", qui indique votre objectif de test avec le monde. Si vous pensez que la formule est fausse, vous sélectionnez alors **false**, sinon vous sélectionnez **true**. Le triangle vert () permet ensuite d'évaluer la formule sur le monde; TARSKIUDES indique alors si votre test est satisfait () ou insatisfait (). La formule de la figure 1.3 est une tautologie, donc elle est vraie peu importe le monde (car si un objet est un grand carré, alors ce n'est pas un triangle). C'est donc une formule peu intéressante.

Le nombre de cas de test est difficile à déterminer en général en informatique. On ne peut pas tester exhaustivement tous les cas possibles pour un programme, même le plus simple, car ils sont trop nombreux. Il y a 10^{64} tests possibles pour une formule en TarskiUdeS. Bien sûr, plusieurs cas se ressemblent et ne testent pas quelque chose de différent d'un autre. Il faut donc trouver des cas de tests "représentatifs", et on voit dans le cours IGL601 des critères et des algorithmes pour générer des cas de test selon une approche systématique. Ce sujet est trop avancé pour MAT115. Dans le cadre de MAT115, nous nous contenterons d'identifier de manière informelle les cas de tests, en analysant la forme de la formule et la phrase à formaliser. Pour chaque opérateur, on teste les cas où il est vrai et où il est faux, en regroupant les opérateurs similaires (par exemple, on teste une suite de conjonctions seulement 2 fois). La combinaison des opérateurs multiplie le nombre de tests à faire. On se limite à un nombre raisonnable de tests en fonction du temps disponible.

Notons que les tests ne garantissent pas l'absence d'erreurs dans un logiciel. Seule une preuve mathématique permet de conclure qu'un logiciel est conforme à sa spécification et garantit l'absence d'erreurs. C'est l'une des applications les plus importantes de la logique en informatique.

Les exemples de la section suivante illustrent quelques exemples de formules.

Formula: tests

- Formula-0
- World-0
- World-1
- Formula-1
- World-0

Formula-1 / World-0

Formule ayant peu d'intérêt, sauf pour illustrer la priorité des opérateurs

```

1 !x.
2 (
3     Square(x)
4     &
5     Large(x)
6 =>
7     not
8     Triangle(x)
9     or
10    Small(x)
11 )
12

```

Small									
Medium									
Large									

Formula is False True

Figure 1.3: Un exemple de monde dans TARSKIUNDES

1.2.3.3 Exemples

Voici quelques exemples de phrases représentées par des formules qui sont vraies dans le monde de la figure 1.3. Pour chaque formule, nous donnons une version écrite en syntaxe mathématique traditionnelle avec le nombre minimum de parenthèses selon la précedence définie à la remarque 4 page 14, et l'autre en utilisant la syntaxe du langage TARSKIUDES.

1. “L’objet a est un triangle de grande taille.”

$$\text{Triangle}(a) \wedge \text{Large}(a)$$

$$\text{Triangle}(a) \ \& \ \text{Large}(a)$$

Dans cet exemple, la formule porte sur un objet particulier, le triangle a . Le symbole a est donc une constante du langage, tel que décrit à la définition 2; ce n’est pas une variable. Dans TARSKIUDES, si on veut définir une variable, il faut l’introduire avec un quantificateur \forall ou \exists .

2. “Le carré b est situé entre les objets a et c .”

$$\text{Square}(b) \wedge \text{Between}(b, a, c)$$

$$\text{Square}(b) \ \& \ \text{Between}(b, a, c)$$

Ici aussi, les symboles a, b, c sont des constantes qui dénotent des objets particuliers.

3. “Tous les objets de grande taille sont sur la même ligne.”

$$\forall x \cdot \forall y \cdot \text{Large}(x) \wedge \text{Large}(y) \Rightarrow \text{SameRow}(x, y)$$

$$\!x. (\!y. (\text{Large}(x) \ \& \ \text{Large}(y) \Rightarrow \text{SameRow}(x, y)))$$

Cette phrase ne vise pas un objet en particulier, mais des objets quelconques du monde. Il faut donc utiliser des variables, x, y , introduites par un quantificateur. C’est ce qui distingue une variable d’une constante dans TARSKIUDES: une variable doit être introduite par un quantificateur universel ou existentiel; une constante est déclarée dans le monde en l’associant à une objet lors de sa création ou modification. Pour formaliser cette phrase, il suffit d’utiliser deux variables. Ces variables permettront de comparer chaque paire d’objets. Ainsi, la formule quantifiée est traduite par la formule suivante, qui est une conjonction de tous les cas possibles de valeurs pour x et y (il y en a $9 * 9 = 81$).

$$\begin{array}{l} 1 \quad (\text{Large}(a) \wedge \text{Large}(a) \Rightarrow \text{SameRow}(a, a)) \\ 2 \quad \wedge \quad (\text{Large}(a) \wedge \text{Large}(b) \Rightarrow \text{SameRow}(a, b)) \\ \quad \wedge \quad \dots \\ 9 \quad \wedge \quad (\text{Large}(a) \wedge \text{Large}(j) \Rightarrow \text{SameRow}(a, j)) \\ 10 \quad \wedge \quad (\text{Large}(b) \wedge \text{Large}(a) \Rightarrow \text{SameRow}(b, a)) \\ 11 \quad \wedge \quad (\text{Large}(b) \wedge \text{Large}(b) \Rightarrow \text{SameRow}(b, b)) \\ \quad \wedge \quad \dots \\ 81 \quad \wedge \quad (\text{Large}(j) \wedge \text{Large}(j) \Rightarrow \text{SameRow}(j, j)) \end{array}$$

Examinons le cas no 9. L’opérande de gauche de l’implication (\Rightarrow) est une conjonction dont la valeur est fausse, car l’objet j n’est pas de grande taille (il est de taille petite). L’implication

est donc vraie. Le cas no 9 traite un objet grand et un objet petit, donc, ces objets n'ont pas besoin d'être sur la même ligne. Il est très fréquent d'utiliser une implication à l'intérieur d'une quantification universelle, car une phrase porte généralement sur un sous-ensemble d'objets. On utilise l'opérande de gauche de " \Rightarrow " pour indiquer à quels objets on s'intéresse dans la phrase à formaliser. On note aussi que les cas où $x = y$ sont aussi traités systématiquement. Par exemple, pour le cas 1, la formule est vraie aussi, car a est grand et sur la même ligne que lui-même.

Puisque la formule est vraie dans tous les cas, la formule quantifiée originale est donc vraie.

4. "Il y a au moins un objet de grande taille pour chaque type d'objet."

$$\begin{aligned} & \exists x \cdot (\text{Large}(x) \wedge \text{Pentagon}(x)) \\ \wedge & \exists x \cdot (\text{Large}(x) \wedge \text{Square}(x)) \\ \wedge & \exists x \cdot (\text{Large}(x) \wedge \text{Triangle}(x)) \\ \\ & \#x. (\text{Large}(x) \ \& \ \text{Pentagon}(x)) \\ \& & \#x. (\text{Large}(x) \ \& \ \text{Square}(x)) \\ \& & \#x. (\text{Large}(x) \ \& \ \text{Triangle}(x)) \end{aligned}$$

On utilise une conjonction de trois quantifications existentielles, chacune traitant un type d'objet. Cette formule se traduit de la manière suivante pour éliminer le \exists . Chaque occurrence du quantificateur \exists est remplacée par une disjonction des 9 cas possibles, c'est-à-dire un cas pour chaque objet du monde.

$$\begin{aligned} & (\quad (\text{Large}(a) \wedge \text{Pentagon}(a)) \\ & \quad \vee \dots \\ & \quad \vee (\text{Large}(j) \wedge \text{Pentagon}(j)) \\ &) \\ \wedge & (\quad (\text{Large}(a) \wedge \text{Square}(a)) \\ & \quad \vee \dots \\ & \quad \vee (\text{Large}(j) \wedge \text{Square}(j)) \\ &) \\ \wedge & (\quad (\text{Large}(a) \wedge \text{Triangle}(a)) \\ & \quad \vee \dots \\ & \quad \vee (\text{Large}(j) \wedge \text{Triangle}(j)) \\ &) \end{aligned}$$

5. "Tous les carrés sont situés entre un pentagone à gauche et un triangle à droite, de même taille, sur la même ligne."

$$\forall x \cdot$$

$$\left(\text{Square}(x) \right.$$

$$\Rightarrow \exists y \cdot \exists z \cdot$$

$$\left(\text{Pentagon}(y) \right.$$

$$\wedge \text{Triangle}(z)$$

$$\wedge \text{Between}(x, y, z)$$

$$\wedge \text{LeftOf}(y, z)$$

$$\wedge \text{SameRow}(y, z)$$

$$\wedge \text{SameSize}(x, y)$$

$$\wedge \text{SameSize}(y, z)$$

$$\left. \right)$$

$$\left. \right)$$

$$\exists x \cdot$$

$$\left(\text{Square}(x) \right.$$

$$\Rightarrow \#y \cdot (\#z \cdot$$

$$\left(\text{Pentagon}(y) \right.$$

$$\& \text{Triangle}(z)$$

$$\& \text{Between}(x, y, z)$$

$$\& \text{SameSize}(x, y)$$

$$\& \text{SameSize}(y, z)$$

$$\left. \right))$$

$$\left. \right)$$

Ce dernier exemple démontre l'importance de l'ordre des quantificateurs. Le quantificateur \forall doit précéder les quantificateurs \exists , car cela permet de “choisir” pour chaque valeur de x , une nouvelle valeur pour y et z . L'exemple suivant illustre le cas inverse.

6. “Il existe un carré situé à gauche de tous les pentagones”

Cette phrase est fausse dans le monde de la figure 1.3, mais elle est vraie dans le monde ci-dessous. Elle permet d'illustrer l'ordre des quantificateurs.



Voici sa formalisation.

$$\exists x \cdot (\text{Square}(x) \wedge (\forall y \cdot (\text{Pentagon}(y) \Rightarrow \text{LeftOf}(x, y))))$$

$$\#x \cdot (\text{Square}(x) \& (!y \cdot (\text{Pentagon}(y) \Rightarrow \text{LeftOf}(x, y))))$$

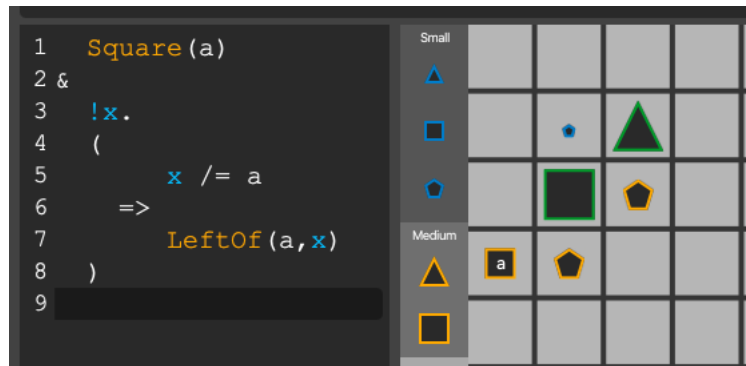
Et voici sa traduction. Il y a $4 * 4 = 16$ cas à considérer.

1	(Square(a) ∧ ((Pentagon(a) ⇒ LeftOf(a, a))	1.1
		∧ (Pentagon(b) ⇒ LeftOf(a, b))	1.2
		∧ (Pentagon(c) ⇒ LeftOf(a, c))	1.3
		∧ (Pentagon(d) ⇒ LeftOf(a, d))	1.4
))	
2	∨ (Square(b) ∧ ((Pentagon(a) ⇒ LeftOf(b, a))	2.1
		∧ (Pentagon(b) ⇒ LeftOf(b, b))	2.2
		∧ (Pentagon(c) ⇒ LeftOf(b, c))	2.3
		∧ (Pentagon(d) ⇒ LeftOf(b, d))	2.4
))	
3	∨ (Square(c) ∧ ((Pentagon(a) ⇒ LeftOf(c, a))	3.1
		∧ (Pentagon(b) ⇒ LeftOf(c, b))	3.2
		∧ (Pentagon(c) ⇒ LeftOf(c, c))	3.3
		∧ (Pentagon(d) ⇒ LeftOf(c, d))	3.4
))	
4	∨ (Square(d) ∧ ((Pentagon(a) ⇒ LeftOf(d, a))	4.1
		∧ (Pentagon(b) ⇒ LeftOf(d, b))	4.2
		∧ (Pentagon(c) ⇒ LeftOf(d, c))	4.3
		∧ (Pentagon(d) ⇒ LeftOf(d, d))	4.4
))	

Le prédicat **Square** est vrai seulement dans les cas 1 et 2. Dans le cas 1, toutes les sous-formules sont vraies. La formule est donc vraie. Dans le cas 2, la sous-formule 2.3 est fausse car **Pentagon(c)** est vrai mais **LeftOf(b, c)** est faux, donc la formule **Pentagon(c) ⇒ LeftOf(b, c)** est fausse.

7. “Le carré *a* est situé à la gauche de tous les autres carrés”

Cette phrase est fausse dans le monde de la figure 1.3, mais elle est vraie dans le monde ci-dessous. Elle permet d’illustrer l’inégalité, afin de ne pas comparer *a* avec lui-même dans la quantification.



Voici sa formalisation.

$$\text{Square}(a) \wedge \forall x \cdot \text{Square}(x) \wedge x \neq a \Rightarrow \text{LeftOf}(a, x)$$

$$\text{Square}(a) \ \& \ !x. (\text{Square}(x) \ \& \ x \neq a \Rightarrow \text{LeftOf}(a, x))$$

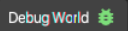
Notons qu'une erreur typique pour la formalisation de ce type phrase est justement d'oublier l'inégalité. Ainsi, la formule suivante est fausse, car le $\forall x \cdot \dots$ compare a avec lui-même.

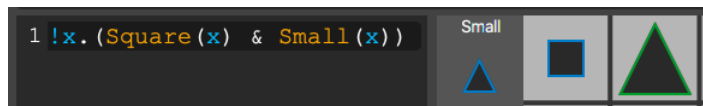
$$\text{Square}(a) \wedge \forall x \cdot \text{Square}(x) \Rightarrow \text{LeftOf}(a, x)$$

1.2.3.4 Erreurs

TARSKIUNDES vérifie la syntaxe de votre formule au fur et à mesure qu'elle est tapée. Les parties en erreur sont soulignées dans la formule. La console *Parser Log* contient les messages d'erreurs. Lors de l'évaluation, TARSKIUNDES peut être incapable d'évaluer une formule si un symbole n'est pas introduit par un quantificateur et si aucun objet n'est identifié par ce symbole. Cette erreur est affichée dans la console *Eval Log*

```
[17:38:00] [Tarski] - Starting evaluator task...
[17:38:00] [Formula-3 / World-0] - x is not defined
[17:38:00] [Tarski] - Evaluator task finished.
```

Vous pouvez utiliser le bouton  pour vous aider à corriger une formule lorsque son évaluation ne retourne pas la valeur que vous attendiez dans un monde. Par exemple, si vous pensiez que la formule suivante est vraie,



et que l'évaluation retourne faux, vous appuyez sur le bouton *Debug World*, et TARSKIUNDES vous indique alors pourquoi la formule est fausse, en vous montrant la valeur de chaque sous-formule.

Formula debugger	
Formula	Result
!x (x)	false
& (x=SMALL_SQUARE)	true
Square('SMALL SQUARE')	true
Small('SMALL SQUARE')	true
& (x=LARGE_TRIANGLE)	false
Square('LARGE TRIANGLE')	false
Small('LARGE TRIANGLE')	false

La formule est vraie pour le premier objet, qui est un petit carré, mais elle est fausse pour le deuxième, qui est un grand triangle.

1.2.4 Formes typiques des formules quantifiées

On remarque deux formes typiques pour les formules quantifiées.

1.2.4.1 Quantificateur universel

Pour un quantificateur universel, on a:

$$\forall x \cdot \mathcal{A} \Rightarrow \mathcal{B}$$

Cette forme correspond à des phrases de la forme suivante:

Tous les \mathcal{A} satisfont la propriété \mathcal{B} .

Voici quelques exemples concrets de cette forme avec leur formalisation.

1. Tous les carrés sont petits.

Un carré est petit.

Les carrés sont petits.

Ce qui est carré est petit.

$$\forall x \cdot \text{Square}(x) \Rightarrow \text{Small}(x)$$

2. Tous les petits carrés sont sur la même ligne.

Les petits carrés sont sur la même ligne.

Si deux carrés sont petits, alors ils sont sur la même ligne.

$$\forall x \cdot \forall y \cdot \text{Square}(x) \wedge \text{Small}(x) \wedge \text{Square}(y) \wedge \text{Small}(y) \Rightarrow \text{SameRow}(x, y)$$

On remarque que le prédicat **Small** apparaît dans les deux exemples, mais pas à la même place dans la formalisation. Dans le premier exemple, “petit” dénote une propriété des carrés, alors que dans le deuxième, on a deux propriétés pour les carrés. Être sur la même ligne dépend d’être petit, c-à-d que si un objet est un petit carré, alors il doit être sur la même ligne que tous les autres petits carré. Si un objet n’est pas un petit carré, alors la phrase ne lui impose aucune contrainte. On retrouve donc “petit” à gauche plutôt qu’à droite dans le deuxième exemple. Nous verrons la loi de la logique (LP-37) qui nous permet de conclure que la formule suivante est une formalisation équivalente du deuxième exemple.

$$\forall x \cdot \forall y \cdot \text{Square}(x) \Rightarrow (\text{Small}(x) \Rightarrow (\text{Square}(y) \Rightarrow (\text{Small}(y) \Rightarrow \text{SameRow}(x, y))))$$

Par soucis de lisibilité, on utilise plutôt la première forme.

Il n’est pas toujours nécessaire d’utiliser une implication avec un quantificateur universel. Par exemple, la phrase suivante

Tous les objets sont des petits carrés.

est formalisée comme suit.

$$\forall x \cdot \text{Square}(x) \wedge \text{Small}(x)$$

Elle ne requière pas d’implication, car la propriété “petit carré” s’applique à tous les objets.

1.2.4.2 Quantificateur existentiel

Pour un quantificateur existentiel, la forme typique est la suivante

$$\exists x \cdot \mathcal{A} \wedge \mathcal{B}$$

Voici quelques exemples concrets de cette forme avec leur formalisation.

1. Il existe un petit carré.

Il y a au moins un petit carré.

$$\exists x \cdot \text{Square}(x) \wedge \text{Small}(x)$$

2. Il existe un petit carré ayant à sa droite un grand triangle.

$$\exists x \cdot \text{Square}(x) \wedge \text{Small}(x) \wedge \exists y \cdot \text{Triangle}(y) \wedge \text{LeftOf}(x, y)$$

La forme

$$\exists x \cdot \mathcal{A} \Rightarrow \mathcal{B}$$

n’est quasiment jamais utilisée, car elle est très facilement satisfaisable. En effet, il suffit de trouver un objet qui ne satisfait pas \mathcal{A} , et la formule est satisfaite, car une implication est vraie si son opérande de gauche est fausse.

1.2.5 Formalisation de texte en langage naturel

Voici un petit exemple de formalisation de texte en logique du premier ordre. Considérons cet extrait du règlement des études, que nous allons formaliser:

“ Une activité pédagogique, en lien avec une autre, est :

- **préalable**, si elle doit être réussie avant l’inscription à une autre;
- **antérieure**, si elle doit être complétée avant une autre, sans exigence de réussite;
- **concomitante**, si elle doit être suivie en même temps qu’une autre, à moins d’avoir été réussie. ”

Pour simplifier le jargon universitaire, nous utiliserons le terme *cours* au lieu du terme *activité pédagogique* dans la suite.

Pour formaliser ce texte, nous avons besoin de définir des *prédicats* et d’utiliser des variables.

1. $\text{peutSInscrire}(e, c)$: retourne **vrai** si, et seulement si, l’étudiant e peut s’inscrire au cours c .
2. $\text{réussi}(e, c)$: retourne **vrai** si, et seulement si, l’étudiant e a réussi le cours c .
3. $\text{suivi}(e, c)$: retourne **vrai** si, et seulement si, l’étudiant e a suivi le cours c . Notons que cela ne signifie pas que l’étudiant a réussi le cours.
4. $\text{préalable}(c_1, c_2)$: retourne **vrai** si, et seulement si, le cours c_1 est préalable au cours c_2 .
5. $\text{antérieur}(c_1, c_2)$: retourne **vrai** si, et seulement si, le cours c_1 est antérieur au cours c_2 .
6. $\text{concomitant}(c_1, c_2)$: retourne **vrai** si, et seulement si, le cours c_1 est concomitant au cours c_2 .

Préalable est une condition nécessaire pour s’inscrire à un cours, mais elle n’est pas une condition suffisante. Il existe d’autres conditions nécessaires pour pouvoir s’inscrire à un cours, soit d’avoir payé ses frais de scolarité des sessions antérieures, avoir payés ses amendes à la bibliothèque, etc. Voici une formalisation d’une propriété de la notion de *préalable*.

$$\forall e, c_2 \cdot \text{peutSInscrire}(e, c_2) \Rightarrow (\forall c_1 \cdot \text{préalable}(c_1, c_2) \Rightarrow \text{réussi}(e, c_1)) \quad (1.23)$$

Notons que cette formalisation utilise une implication, au lieu d’une équivalence. Elle indique que préalable est une condition nécessaire pour s’inscrire, mais elle n’est pas suffisante. Dans le contexte universitaire, le prédicat *préalable* est défini par énumération de tous les préalables. Pour connaître l’impact de ce prédicat sur le reste des règlements universitaire, on donne une formule qui énumère une propriété de préalable. Donc, avoir réussi les préalables est une condition nécessaire pour pouvoir s’inscrire à un cours, ce qui correspond à la forme suivante, équivalente à la forme (1.24), grâce aux lois (LP-38) et (LPO-26).

$$\forall e, c_1, c_2 \cdot \text{préalable}(c_1, c_2) \Rightarrow (\text{peutSInscrire}(e, c_2) \Rightarrow \text{réussi}(e, c_1)) \quad (1.24)$$

Par souci de lisibilité, on peut aussi écrire la formule (1.24) en l’indentant comme suit:

$$\begin{array}{l} \forall e, c_1, c_2 \cdot \\ \quad \text{préalable}(c_1, c_2) \\ \Rightarrow \\ \quad (\text{peutSInscrire}(e, c_2) \\ \quad \Rightarrow \\ \quad \quad \text{réussi}(e, c_1) \\ \quad) \end{array}$$

Voici une formalisation de *antérieur*

$$\forall e, c_1, c_2 \cdot \text{antérieur}(c_1, c_2) \Rightarrow (\text{peutSInscrire}(e, c_2) \Rightarrow \text{suivi}(e, c_1))$$

$$\begin{aligned} &\forall e, c_1, c_2 \cdot \\ &\quad \text{antérieur}(c_1, c_2) \\ &\quad \Rightarrow \\ &\quad (\text{peutSInscrire}(e, c_2) \\ &\quad \Rightarrow \\ &\quad \quad \text{suivi}(e, c_1) \\ &\quad) \end{aligned}$$

Voici une formalisation de *concomittant*

$$\forall e, c_1, c_2 \cdot \text{concomittant}(c_1, c_2) \Rightarrow (\text{peutSInscrire}(e, c_2) \Rightarrow \text{réussi}(e, c_1) \vee \text{peutSInscrire}(e, c_1))$$

$$\begin{aligned} &\forall e, c_1, c_2 \cdot \\ &\quad \text{concomittant}(c_1, c_2) \\ &\quad \Rightarrow \\ &\quad (\text{peutSInscrire}(e, c_2) \\ &\quad \Rightarrow \\ &\quad \quad \text{réussi}(e, c_1) \\ &\quad \quad \vee \\ &\quad \quad \text{peutSInscrire}(e, c_1) \\ &\quad) \end{aligned}$$

1.3 Lois de la logique

1.3.1 Lois de la logique propositionnelle (tautologies)

Une tautologie est une formule de logique propositionnelle dont la valeur est vrai pour toutes les combinaisons possibles des valeurs des variables propositionnelles. Autrement dit, une tautologie est une formule qui est toujours vraie. Par exemple, voici deux tautologies très connues, identifiées par Augustus De Morgan⁵, et communément appelées *Lois de De Morgan*.

$$\neg(\mathcal{A} \wedge \mathcal{B}) \Leftrightarrow \neg\mathcal{A} \vee \neg\mathcal{B} \quad (\text{LP-17})$$

$$\neg(\mathcal{A} \vee \mathcal{B}) \Leftrightarrow \neg\mathcal{A} \wedge \neg\mathcal{B} \quad (\text{LP-18})$$

Voici le calcul de la valeur de vérité de la formule (LP-17), en utilisant une forme encore plus compacte. La valeur de vérité de chaque partie de la formule est donnée sous le connecteur principal de cette partie. La valeur de vérité de la formule est donc donnée sous la colonne (en rouge) du connecteur “ \Leftrightarrow ” pour cet exemple.

no	\mathcal{A}	\mathcal{B}	$\neg(\mathcal{A}$	\wedge	$\mathcal{B})$	\Leftrightarrow	\neg	\mathcal{A}	\vee	\neg	\mathcal{B}
1	0	0	1	0	0	1	1	0	1	1	0
2	0	1	1	0	0	1	1	0	1	0	1
3	1	0	1	1	0	1	0	1	1	1	0
4	1	1	0	1	1	1	0	1	0	0	1

⁵Augustus De Morgan (1806–1871) est un mathématicien et logicien britannique, né en Inde. Il est un des fondateurs de la logique moderne; en plus des lois (LP-17) et (LP-18), il a aussi formalisé le concept d’induction mathématique, que nous verrons au chapitre 3.

Les tautologies sont importantes, car elles sont à la source de lois de la logique propositionnelle. Entre autres, les tautologies de la forme $\mathcal{A} \Leftrightarrow \mathcal{B}$ nous permettent de remplacer la formule \mathcal{A} par la formule \mathcal{B} , et vice-versa, car les deux formules sont équivalentes, au sens où la valeur de vérité de \mathcal{A} est la même que celle de \mathcal{B} , peu importe la valeur des variables propositionnelles. On peut remplacer \mathcal{A} par \mathcal{B} lorsque \mathcal{A} est une sous-formule d'une formule, exactement comme pour l'égalité entre les nombres. Les tautologies de la forme $\mathcal{A} \Rightarrow \mathcal{B}$ nous permettent de déduire la formule \mathcal{B} de la formule \mathcal{A} si on sait que \mathcal{A} est vraie, car si \mathcal{A} est vraie, alors \mathcal{B} est vraie, selon la table de vérité de " \Rightarrow ". Les tables 1.3, 1.4, 1.5, 1.6 donnent une liste des principales tautologies.

<i>élément absorbant et</i>	$\mathcal{A} \wedge \text{faux} \Leftrightarrow \text{faux}$	(LP-1)
<i>élément absorbant ou</i>	$\mathcal{A} \vee \text{vrai} \Leftrightarrow \text{vrai}$	(LP-2)
<i>élément neutre et</i>	$\mathcal{A} \wedge \text{vrai} \Leftrightarrow \mathcal{A}$	(LP-3)
<i>élément neutre ou</i>	$\mathcal{A} \vee \text{faux} \Leftrightarrow \mathcal{A}$	(LP-4)
<i>idempotence et</i>	$\mathcal{A} \wedge \mathcal{A} \Leftrightarrow \mathcal{A}$	(LP-5)
<i>idempotence ou</i>	$\mathcal{A} \vee \mathcal{A} \Leftrightarrow \mathcal{A}$	(LP-6)
<i>commutativité et</i>	$\mathcal{A} \wedge \mathcal{B} \Leftrightarrow \mathcal{B} \wedge \mathcal{A}$	(LP-7)
<i>commutativité ou</i>	$\mathcal{A} \vee \mathcal{B} \Leftrightarrow \mathcal{B} \vee \mathcal{A}$	(LP-8)
<i>associativité et</i>	$\mathcal{A} \wedge (\mathcal{B} \wedge \mathcal{C}) \Leftrightarrow (\mathcal{A} \wedge \mathcal{B}) \wedge \mathcal{C}$	(LP-9)
<i>associativité ou</i>	$\mathcal{A} \vee (\mathcal{B} \vee \mathcal{C}) \Leftrightarrow (\mathcal{A} \vee \mathcal{B}) \vee \mathcal{C}$	(LP-10)
<i>distributivité et sur ou</i>	$\mathcal{A} \wedge (\mathcal{B} \vee \mathcal{C}) \Leftrightarrow (\mathcal{A} \wedge \mathcal{B}) \vee (\mathcal{A} \wedge \mathcal{C})$	(LP-11)
<i>distributivité ou sur et</i>	$\mathcal{A} \vee (\mathcal{B} \wedge \mathcal{C}) \Leftrightarrow (\mathcal{A} \vee \mathcal{B}) \wedge (\mathcal{A} \vee \mathcal{C})$	(LP-12)
<i>absorption et sur ou</i>	$\mathcal{A} \wedge (\mathcal{A} \vee \mathcal{B}) \Leftrightarrow \mathcal{A}$	(LP-13)
<i>absorption ou sur et</i>	$\mathcal{A} \vee (\mathcal{A} \wedge \mathcal{B}) \Leftrightarrow \mathcal{A}$	(LP-14)
<i>absorption et avec négation</i>	$\mathcal{A} \wedge (\neg \mathcal{A} \vee \mathcal{B}) \Leftrightarrow \mathcal{A} \wedge \mathcal{B}$	(LP-15)
<i>absorption ou avec négation</i>	$\mathcal{A} \vee (\neg \mathcal{A} \wedge \mathcal{B}) \Leftrightarrow \mathcal{A} \vee \mathcal{B}$	(LP-16)
<i>De Morgan négation et</i>	$\neg(\mathcal{A} \wedge \mathcal{B}) \Leftrightarrow \neg \mathcal{A} \vee \neg \mathcal{B}$	(LP-17)
<i>De Morgan négation ou</i>	$\neg(\mathcal{A} \vee \mathcal{B}) \Leftrightarrow \neg \mathcal{A} \wedge \neg \mathcal{B}$	(LP-18)
<i>contradiction</i>	$\mathcal{A} \wedge \neg \mathcal{A} \Leftrightarrow \text{faux}$	(LP-19)
<i>tiers exclu</i>	$\mathcal{A} \vee \neg \mathcal{A} \Leftrightarrow \text{vrai}$	(LP-20)
<i>involution</i>	$\neg \neg \mathcal{A} \Leftrightarrow \mathcal{A}$	(LP-21)

Table 1.3: Lois de \wedge , \vee et \neg

<i>implication comme disjonction</i>	$\mathcal{A} \Rightarrow \mathcal{B} \Leftrightarrow \neg \mathcal{A} \vee \mathcal{B}$	(LP-22)
<i>implication comme tiers exclu</i>	$\mathcal{A} \Rightarrow \mathcal{A} \Leftrightarrow \text{vrai}$	(LP-23)
<i>implication comme négation et</i>	$\mathcal{A} \Rightarrow \mathcal{B} \Leftrightarrow \neg(\mathcal{A} \wedge \neg \mathcal{B})$	(LP-24)
<i>négation implication</i>	$\neg(\mathcal{A} \Rightarrow \mathcal{B}) \Leftrightarrow \mathcal{A} \wedge \neg \mathcal{B}$	(LP-25)
<i>Contraposée</i>	$\mathcal{A} \Rightarrow \mathcal{B} \Leftrightarrow \neg \mathcal{B} \Rightarrow \neg \mathcal{A}$	(LP-26)
<i>implication divers 1</i>	$\mathcal{A} \Rightarrow \text{vrai} \Leftrightarrow \text{vrai}$	(LP-27)
<i>implication divers 2</i>	$\text{vrai} \Rightarrow \mathcal{A} \Leftrightarrow \mathcal{A}$	(LP-28)
<i>implication divers 3</i>	$\mathcal{A} \Rightarrow \text{faux} \Leftrightarrow \neg \mathcal{A}$	(LP-29)
<i>implication divers 4</i>	$\text{faux} \Rightarrow \mathcal{A} \Leftrightarrow \text{vrai}$	(LP-30)
<i>implication divers 5</i>	$\mathcal{A} \Rightarrow \neg \mathcal{A} \Leftrightarrow \neg \mathcal{A}$	(LP-31)
<i>implication divers 6</i>	$\neg \mathcal{A} \Rightarrow \mathcal{A} \Leftrightarrow \mathcal{A}$	(LP-32)
<i>distributivité implication et gauche</i>	$\mathcal{C} \Rightarrow (\mathcal{A} \wedge \mathcal{B}) \Leftrightarrow (\mathcal{C} \Rightarrow \mathcal{A}) \wedge (\mathcal{C} \Rightarrow \mathcal{B})$	(LP-33)
<i>distributivité implication ou gauche</i>	$\mathcal{C} \Rightarrow (\mathcal{A} \vee \mathcal{B}) \Leftrightarrow (\mathcal{C} \Rightarrow \mathcal{A}) \vee (\mathcal{C} \Rightarrow \mathcal{B})$	(LP-34)
<i>distributivité implication et droite</i>	$(\mathcal{A} \wedge \mathcal{B}) \Rightarrow \mathcal{C} \Leftrightarrow (\mathcal{A} \Rightarrow \mathcal{C}) \wedge (\mathcal{B} \Rightarrow \mathcal{C})$	(LP-35)
<i>distributivité implication ou droite</i>	$(\mathcal{A} \vee \mathcal{B}) \Rightarrow \mathcal{C} \Leftrightarrow (\mathcal{A} \Rightarrow \mathcal{C}) \vee (\mathcal{B} \Rightarrow \mathcal{C})$	(LP-36)
<i>implication conjonction</i>	$\mathcal{A} \Rightarrow (\mathcal{B} \Rightarrow \mathcal{C}) \Leftrightarrow (\mathcal{A} \wedge \mathcal{B}) \Rightarrow \mathcal{C}$	(LP-37)
<i>implication conjonction</i>	$\mathcal{A} \Rightarrow (\mathcal{B} \Rightarrow \mathcal{C}) \Leftrightarrow \mathcal{B} \Rightarrow (\mathcal{A} \Rightarrow \mathcal{C})$	(LP-38)
<i>définition par cas</i>	$(\mathcal{A} \Rightarrow \mathcal{B}) \wedge (\neg \mathcal{A} \Rightarrow \mathcal{C}) \Leftrightarrow (\mathcal{A} \wedge \mathcal{B}) \vee (\neg \mathcal{A} \wedge \mathcal{C})$	(LP-39)

Table 1.4: Lois de \Rightarrow

<i>équivalence comme implication</i>	$(\mathcal{A} \Leftrightarrow \mathcal{B}) \Leftrightarrow (\mathcal{A} \Rightarrow \mathcal{B}) \wedge (\mathcal{B} \Rightarrow \mathcal{A})$	(LP-40)
<i>équivalence comme ou</i>	$(\mathcal{A} \Leftrightarrow \mathcal{B}) \Leftrightarrow (\mathcal{A} \wedge \mathcal{B}) \vee \neg(\mathcal{A} \vee \mathcal{B})$	(LP-41)
<i>équivalence comme négation</i>	$(\mathcal{A} \Leftrightarrow \mathcal{B}) \Leftrightarrow (\neg \mathcal{A} \Leftrightarrow \neg \mathcal{B})$	(LP-42)
<i>commutativité de l'équivalence</i>	$(\mathcal{A} \Leftrightarrow \mathcal{B}) \Leftrightarrow (\mathcal{B} \Leftrightarrow \mathcal{A})$	(LP-43)
<i>associativité de l'équivalence</i>	$(\mathcal{A} \Leftrightarrow (\mathcal{B} \Leftrightarrow \mathcal{C})) \Leftrightarrow ((\mathcal{A} \Leftrightarrow \mathcal{B}) \Leftrightarrow \mathcal{C})$	(LP-44)
<i>équivalence divers 1</i>	$(\mathcal{A} \Leftrightarrow \mathcal{A}) \Leftrightarrow \text{vrai}$	(LP-45)
<i>équivalence divers 2</i>	$(\mathcal{A} \Leftrightarrow \neg \mathcal{A}) \Leftrightarrow \text{faux}$	(LP-46)
<i>équivalence divers 3</i>	$(\mathcal{A} \Leftrightarrow \text{vrai}) \Leftrightarrow \mathcal{A}$	(LP-47)
<i>équivalence divers 4</i>	$(\mathcal{A} \Leftrightarrow \text{faux}) \Leftrightarrow \neg \mathcal{A}$	(LP-48)
<i>équivalence divers 5</i>	$(\mathcal{A} \Rightarrow \mathcal{B}) \Leftrightarrow (\mathcal{A} \Leftrightarrow (\mathcal{A} \wedge \mathcal{B}))$	(LP-49)
<i>équivalence divers 6</i>	$(\mathcal{A} \Rightarrow \mathcal{B}) \Leftrightarrow (\mathcal{B} \Leftrightarrow (\mathcal{A} \vee \mathcal{B}))$	(LP-50)
<i>équivalence divers 7</i>	$(\mathcal{A} \vee (\mathcal{B} \Leftrightarrow \mathcal{C})) \Leftrightarrow ((\mathcal{A} \vee \mathcal{B}) \Leftrightarrow (\mathcal{A} \vee \mathcal{C}))$	(LP-51)
<i>équivalence négation</i>	$\neg(\mathcal{A} \Leftrightarrow \mathcal{B}) \Leftrightarrow (\neg \mathcal{A} \Leftrightarrow \mathcal{B})$	(LP-52)

Table 1.5: Lois de \Leftrightarrow

<i>ou exclusif 2</i>	$\mathcal{A} \oplus \mathcal{B} \Leftrightarrow \neg(\mathcal{A} \Leftrightarrow \mathcal{B})$	(LP-53)
<i>ou exclusif 3</i>	$\mathcal{A} \oplus \mathcal{B} \Leftrightarrow (\mathcal{A} \vee \mathcal{B}) \wedge \neg(\mathcal{A} \wedge \mathcal{B})$	(LP-54)
<i>ou exclusif 4</i>	$\mathcal{A} \oplus \mathcal{B} \Leftrightarrow (\mathcal{A} \wedge \neg \mathcal{B}) \vee (\neg \mathcal{A} \wedge \mathcal{B})$	(LP-55)

Table 1.6: Lois de \oplus

<i>loi du plus fort</i>	$(\mathcal{A} \Rightarrow \mathcal{B}) \Rightarrow (\mathcal{A} \wedge \mathcal{B} \Leftrightarrow \mathcal{A})$	(LP-56)
-------------------------	--	---------

Table 1.7: Lois diverses en logique propositionnelle

1.3.2 Lois de la logique du premier ordre

On utilise aussi $\mathcal{A}, \mathcal{B}, \dots$ pour désigner des formules de logique du premier ordre. On utilise \mathcal{N} pour désigner une formule où x n'est pas libre (c'est-à-dire, il n'y a aucune occurrence libre de x dans \mathcal{N}). Les lois de la logique propositionnelle s'appliquent aussi à la logique du premier ordre.

<i>point univ</i>	$(\forall x \cdot x = t \Rightarrow \mathcal{A}) \Leftrightarrow \mathcal{A}[x := t]$	(LPO-1)
<i>point exist</i>	$(\exists x \cdot x = t \wedge \mathcal{A}) \Leftrightarrow \mathcal{A}[x := t]$	(LPO-2)
<i>point univ ensemble</i>	$(\forall x \cdot x \in \{t_1, \dots, t_n\} \Rightarrow \mathcal{A}) \Leftrightarrow \mathcal{A}[x := t_1] \wedge \dots \wedge \mathcal{A}[x := t_n]$	(LPO-3)
<i>point existe ensemble</i>	$(\exists x \cdot x \in \{t_1, \dots, t_n\} \wedge \mathcal{A}) \Leftrightarrow \mathcal{A}[x := t_1] \vee \dots \vee \mathcal{A}[x := t_n]$	(LPO-4)
<i>idempotence univ</i>	$(\forall x \cdot \forall x \cdot \mathcal{A}) \Leftrightarrow \forall x \cdot \mathcal{A}$	(LPO-5)
<i>idempotence exist</i>	$(\exists x \cdot \exists x \cdot \mathcal{A}) \Leftrightarrow \exists x \cdot \mathcal{A}$	(LPO-6)
<i>permutation univ</i>	$(\forall x \cdot \forall y \cdot \mathcal{A}) \Leftrightarrow \forall y \cdot \forall x \cdot \mathcal{A}$	(LPO-7)
<i>permutation exist</i>	$(\exists x \cdot \exists y \cdot \mathcal{A}) \Leftrightarrow \exists y \cdot \exists x \cdot \mathcal{A}$	(LPO-8)
<i>univ en exist</i>	$(\forall x \cdot \mathcal{A}) \Leftrightarrow \neg \exists x \cdot \neg \mathcal{A}$	(LPO-9)
<i>exist en univ</i>	$(\exists x \cdot \mathcal{A}) \Leftrightarrow \neg \forall x \cdot \neg \mathcal{A}$	(LPO-10)
<i>De Morgan univ</i>	$(\neg \forall x \cdot \mathcal{A}) \Leftrightarrow \exists x \cdot \neg \mathcal{A}$	(LPO-11)
<i>De Morgan exist</i>	$(\neg \exists x \cdot \mathcal{A}) \Leftrightarrow \forall x \cdot \neg \mathcal{A}$	(LPO-12)
<i>distribution univ</i>	$(\forall x \cdot \mathcal{A} \wedge \mathcal{B}) \Leftrightarrow (\forall x \cdot \mathcal{A}) \wedge (\forall x \cdot \mathcal{B})$	(LPO-13)
<i>distribution exist</i>	$(\exists x \cdot \mathcal{A} \vee \mathcal{B}) \Leftrightarrow (\exists x \cdot \mathcal{A}) \vee (\exists x \cdot \mathcal{B})$	(LPO-14)
<i>exist avec implication</i>	$(\exists x \cdot \mathcal{A} \Rightarrow \mathcal{B}) \Leftrightarrow (\forall x \cdot \mathcal{A}) \Rightarrow (\exists x \cdot \mathcal{B})$	(LPO-15)

Table 1.8: Lois d'équivalence des formules du premier ordre

<i>univ exist</i>	$(\forall x \cdot \mathcal{A}) \Rightarrow \exists x \cdot \mathcal{A}$	(LPO-16)
<i>univ ou</i>	$(\forall x \cdot \mathcal{A}) \vee (\forall x \cdot \mathcal{B}) \Rightarrow \forall x \cdot \mathcal{A} \vee \mathcal{B}$	(LPO-17)
<i>univ avec implication</i>	$(\forall x \cdot \mathcal{A} \Rightarrow \mathcal{B}) \Rightarrow (\forall x \cdot \mathcal{A}) \Rightarrow (\forall x \cdot \mathcal{B})$	(LPO-18)
<i>exist avec et</i>	$(\exists x \cdot \mathcal{A} \wedge \mathcal{B}) \Rightarrow (\exists x \cdot \mathcal{A}) \wedge (\exists x \cdot \mathcal{B})$	(LPO-19)
<i>exist avec ou</i>	$(\exists x \cdot \mathcal{A}) \Rightarrow (\exists x \cdot \mathcal{B}) \Rightarrow \exists x \cdot \mathcal{A} \Rightarrow \mathcal{B}$	(LPO-20)
<i>exist sur univ</i>	$(\exists x \cdot \forall y \cdot \mathcal{A}) \Rightarrow \forall y \cdot \exists x \cdot \mathcal{A}$	(LPO-21)
<i>univ point</i>	$(\forall x \cdot \mathcal{A}) \Rightarrow \mathcal{A}[x := t]$	(LPO-22)
<i>univ point</i>	$\mathcal{A}[x := t] \Rightarrow \exists x \cdot \mathcal{A}$	(LPO-23)

Table 1.9: Lois d'implication des formules du premier ordre

déplacement univ et	$(\forall x \cdot \mathcal{N} \wedge \mathcal{A}) \Leftrightarrow \mathcal{N} \wedge \forall x \cdot \mathcal{A}$	(LPO-24)
déplacement univ ou	$(\forall x \cdot \mathcal{N} \vee \mathcal{A}) \Leftrightarrow \mathcal{N} \vee \forall x \cdot \mathcal{A}$	(LPO-25)
déplacement univ implication 1	$(\forall x \cdot \mathcal{N} \Rightarrow \mathcal{A}) \Leftrightarrow \mathcal{N} \Rightarrow \forall x \cdot \mathcal{A}$	(LPO-26)
déplacement univ implication 2	$(\forall x \cdot \mathcal{A} \Rightarrow \mathcal{N}) \Leftrightarrow (\exists x \cdot \mathcal{A}) \Rightarrow \mathcal{N}$	(LPO-27)
déplacement exist et	$(\exists x \cdot \mathcal{N} \wedge \mathcal{A}) \Leftrightarrow \mathcal{N} \wedge \exists x \cdot \mathcal{A}$	(LPO-28)
déplacement exist ou	$(\exists x \cdot \mathcal{N} \vee \mathcal{A}) \Leftrightarrow \mathcal{N} \vee \exists x \cdot \mathcal{A}$	(LPO-29)
déplacement exist implication 1	$(\exists x \cdot \mathcal{N} \Rightarrow \mathcal{A}) \Leftrightarrow \mathcal{N} \Rightarrow \exists x \cdot \mathcal{A}$	(LPO-30)
déplacement exist implication 2	$(\exists x \cdot \mathcal{A} \Rightarrow \mathcal{N}) \Leftrightarrow (\forall x \cdot \mathcal{A}) \Rightarrow \mathcal{N}$	(LPO-31)
absorption exist	$(\exists x \cdot \mathcal{N}) \Leftrightarrow \mathcal{N}$	(LPO-32)
absorption univ	$(\forall x \cdot \mathcal{N}) \Leftrightarrow \mathcal{N}$	(LPO-33)

Table 1.10: Lois de déplacement des quantificateurs

1.4 Preuve en logique propositionnelle

1.4.1 Preuve par équivalence

Nous utiliserons la convention suivante pour faciliter la lecture de certaines preuves, en identifiant ce qui change d'une ligne à l'autre à l'aide des symboles $\lfloor \rfloor$ et $\lceil \rceil$.

$$\begin{aligned}
& \mathcal{A}_1 \dots \lfloor \mathcal{A}_2 \rfloor \dots \mathcal{A}_3 && \langle \text{étape 1} \rangle \\
\Leftrightarrow & \mathcal{A}_1 \dots \lceil \mathcal{A}_4 \rceil \dots \lfloor \mathcal{A}_3 \rfloor && \\
\Leftrightarrow & \mathcal{A}_1 \dots \mathcal{A}_4 \dots \lceil \mathcal{A}_5 \rceil && \langle \text{étape 2} \rangle
\end{aligned}$$

Une formule entourée des symboles $\lfloor \rfloor$ est réécrite sur la ligne suivante dans la formule $\lceil \rceil$. Dans l'exemple ci-dessus, \mathcal{A}_2 est réécrit en \mathcal{A}_4 à l'étape 1, et \mathcal{A}_3 est réécrite en \mathcal{A}_5 à l'étape 2. Si les symboles $\lceil \rceil$ n'apparaissent pas dans la ligne en-dessous, cela veut dire que l'expression entourée de $\lfloor \rfloor$ de la ligne au-dessus est supprimée par la réécriture dans l'étape de preuve. Voici un exemple

$$\begin{aligned}
& \mathcal{A}_1 \wedge \lfloor \mathcal{A}_1 \rfloor && \\
\Leftrightarrow & \mathcal{A}_1 && \langle \text{(LP-5)} \rangle
\end{aligned}$$

Les lois des tableaux 1.3, 1.4, 1.5, 1.6 nous permettent de faire des preuves, c'est-à-dire de déduire une formule à partir d'autres formules. Par exemple, on peut déduire la formule (LP-33) à partir des formules (LP-12) et (LP-22). Voici une manière assez naturelle de représenter cette preuve.

$$\begin{aligned}
& \mathcal{C} \Rightarrow (\mathcal{A} \wedge \mathcal{B}) && \\
\Leftrightarrow & \neg \mathcal{C} \vee (\mathcal{A} \wedge \mathcal{B}) && \langle \text{(LP-22)} \rangle \\
\Leftrightarrow & (\lfloor \neg \mathcal{C} \vee \mathcal{A} \rfloor) \wedge (\lfloor \neg \mathcal{C} \vee \mathcal{B} \rfloor) && \langle \text{(LP-12)} \rangle \\
\Leftrightarrow & (\lceil \mathcal{C} \Rightarrow \mathcal{A} \rceil) \wedge (\lceil \mathcal{C} \Rightarrow \mathcal{B} \rceil) && \langle \text{(LP-22)} \rangle
\end{aligned}$$

Chaque étape de raisonnement de cette preuve est de la forme

$$\begin{array}{l} \mathcal{D} \\ \Leftrightarrow \\ \mathcal{E} \end{array} \qquad \langle \text{ justification } \rangle$$

Le connecteur \Leftrightarrow est *transitif*, c'est-à-dire que si on a les formules $\mathcal{A} \Leftrightarrow \mathcal{B}$ et $\mathcal{B} \Leftrightarrow \mathcal{C}$ que l'on considère vraies, alors on peut déduire que $\mathcal{A} \Leftrightarrow \mathcal{C}$ est vraie; le connecteur " \Leftrightarrow " est similaire à l'égalité "=" sur les nombres. On utilisant la transitivité, la preuve ci-dessus permet de conclure que la première formule est équivalente à la dernière, donc

$$\mathcal{C} \Rightarrow (\mathcal{A} \wedge \mathcal{B}) \Leftrightarrow (\mathcal{C} \Rightarrow \mathcal{A}) \wedge (\mathcal{C} \Rightarrow \mathcal{B})$$

Dans cette preuve, nous avons utilisé deux lois, (LP-12) et (LP-22), pour déduire (LP-33). Nous allons exprimer cela de manière formelle en utilisant la notation suivante:

$$(LP-12), (LP-22) \vdash (LP-33)$$

Voici une preuve assez complexe de la loi (LP-39) en utilisant seulement les lois de (LP-1) à (LP-22).

$$\begin{array}{l} (\llcorner \mathcal{A} \Rightarrow \mathcal{B} \lrcorner) \wedge (\llcorner \neg \mathcal{A} \Rightarrow \mathcal{C} \lrcorner) \\ \Leftrightarrow \langle (LP-22) \text{ deux fois } \rangle \\ (\lceil \neg \mathcal{A} \vee \mathcal{B} \rceil) \wedge (\lceil \llcorner \neg \neg \mathcal{A} \lrcorner \vee \mathcal{C} \rceil) \\ \Leftrightarrow \langle (LP-21) \rangle \\ (\neg \mathcal{A} \vee \mathcal{B}) \wedge (\lceil \mathcal{A} \rceil \vee \mathcal{C}) \\ \Leftrightarrow \langle (LP-11) \text{ deux fois } \rangle \\ \llcorner (\neg \mathcal{A} \wedge \mathcal{A}) \lrcorner \vee (\neg \mathcal{A} \wedge \mathcal{C}) \vee (\mathcal{B} \wedge \mathcal{A}) \vee (\mathcal{B} \wedge \mathcal{C}) \\ \Leftrightarrow \langle (LP-19) \rangle \\ \llcorner \lceil \text{faux} \rceil \vee (\neg \mathcal{A} \wedge \mathcal{C}) \vee (\mathcal{B} \wedge \mathcal{A}) \vee (\mathcal{B} \wedge \mathcal{C}) \lrcorner \\ \Leftrightarrow \langle (LP-4) \rangle \\ (\neg \mathcal{A} \wedge \mathcal{C}) \vee (\mathcal{B} \wedge \mathcal{A}) \vee (\mathcal{B} \wedge \mathcal{C}) \\ \Leftrightarrow \langle (LP-3) \rangle \\ (\neg \mathcal{A} \wedge \mathcal{C}) \vee (\mathcal{B} \wedge \mathcal{A}) \vee ((\mathcal{B} \wedge \mathcal{C}) \wedge \llcorner \text{vrai} \lrcorner) \\ \Leftrightarrow \langle (LP-20) \rangle \\ (\neg \mathcal{A} \wedge \mathcal{C}) \vee (\mathcal{B} \wedge \mathcal{A}) \vee (\llcorner (\mathcal{B} \wedge \mathcal{C}) \wedge (\lceil \mathcal{A} \vee \neg \mathcal{A} \rceil) \lrcorner) \\ \Leftrightarrow \langle (LP-11) \rangle \\ (\neg \mathcal{A} \wedge \mathcal{C}) \vee (\mathcal{B} \wedge \mathcal{A}) \vee \lceil (\mathcal{B} \wedge \mathcal{C} \wedge \mathcal{A}) \vee (\mathcal{B} \wedge \mathcal{C} \wedge \neg \mathcal{A}) \rceil \\ \Leftrightarrow \langle (LP-7), (LP-8) \rangle \\ \llcorner (\mathcal{A} \wedge \mathcal{B}) \vee (\mathcal{A} \wedge \mathcal{B} \wedge \mathcal{C}) \lrcorner \vee \llcorner (\neg \mathcal{A} \wedge \mathcal{C}) \vee (\neg \mathcal{A} \wedge \mathcal{C} \wedge \mathcal{B}) \lrcorner \\ \Leftrightarrow \langle (LP-14) \text{ deux fois } \rangle \\ \lceil (\mathcal{A} \wedge \mathcal{B}) \rceil \vee \lceil (\neg \mathcal{A} \wedge \mathcal{C}) \rceil \end{array}$$

Ce style de preuve basé sur une suite d'équivalences est assez courant. Toutefois, nous allons maintenant remonter aux racines des mathématiques et définir de manière formelle et générale ce qu'est une preuve, et définir quelles sont les déductions valides que l'on peut faire dans une preuve à l'aide de règles d'inférence.

1.4.2 Dédution

Définition 9 On dénote par $\mathcal{A}_1, \dots, \mathcal{A}_n \vdash \mathcal{B}$ la *relation de déduction* entre les formules de la logique propositionnelle (et aussi celle des prédicats du premier ordre). On dit alors que l'on peut *déduire* la formule \mathcal{B} , appelée la *conclusion*, à partir des formules $\mathcal{A}_1, \dots, \mathcal{A}_n$, appelées les *hypothèses*. \square

Par convention, nous utiliserons le symbole Γ pour désigner un ensemble de formules $\mathcal{A}_1, \dots, \mathcal{A}_n$. Selon ce qui est le plus approprié dans un contexte donné, nous utiliserons $\mathcal{A}_1, \dots, \mathcal{A}_n \vdash \mathcal{B}$ ou bien $\Gamma \vdash \mathcal{B}$.

L'expression $\mathcal{A}_1, \dots, \mathcal{A}_n \vdash \mathcal{B}$ signifie que si les formules $\mathcal{A}_1, \dots, \mathcal{A}_n$ sont vraies, alors la formule \mathcal{B} est vraie aussi. L'expression $\mathcal{A}_1, \dots, \mathcal{A}_n \vdash \mathcal{B}$ est aussi appelée un *séquent*; il s'agit d'une relation, représentée par “ \vdash ”, entre les formules; elle fut proposée par le logicien Gerhard Gentzen⁶. On confond souvent la formule $\mathcal{A} \Rightarrow \mathcal{B}$ avec le séquent $\mathcal{A} \vdash \mathcal{B}$. Un séquent **n'est pas** une formule; c'est une *relation* entre des formules (une relation de déduction). On peut calculer cette relation \vdash à l'aide de règles d'inférence de la forme

$$\frac{\mathcal{A}_1 \quad \dots \quad \mathcal{A}_k}{\mathcal{B}}$$

où $\mathcal{A}_1, \dots, \mathcal{A}_k$ sont appelées les *prémisses* et \mathcal{B} la *conclusion*. Cette règle se lit comme suit: si les hypothèses $\mathcal{A}_1, \dots, \mathcal{A}_k$ sont vraies, alors on peut conclure que \mathcal{B} est vraie. On peut composer les règles pour obtenir une preuve. Une *preuve* est donc une suite de déductions obtenues par application de règles d'inférence. Par exemple, les déductions suivantes

$$\frac{\frac{\mathcal{A}_1 \quad \mathcal{A}_2}{\mathcal{A}_5} \quad \frac{\mathcal{A}_3 \quad \mathcal{A}_4}{\mathcal{A}_6}}{\mathcal{A}_7}$$

permettent de conclure $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4 \vdash \mathcal{A}_7$, c'est-à-dire que \mathcal{A}_7 est vraie si $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4$ sont vraies.

La composition des règles forme une preuve, que l'on représente par un *arbre*. Les feuilles de l'arbre, soient $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4$, sont les hypothèses, alors que la racine est la conclusion, soit \mathcal{A}_7 . Les formules \mathcal{A}_5 et \mathcal{A}_6 sont des résultats intermédiaires dans la déduction de \mathcal{A}_7 .

Certaines preuves utilisent la notion de *déchargement* d'hypothèse. C'est typiquement le cas pour la preuve d'une formule de la forme $\mathcal{A} \Rightarrow \mathcal{B}$. Pour faire cette preuve, on procède généralement comme suit: on suppose que \mathcal{A} est vraie et on déduit que \mathcal{B} est vraie. Le séquent $\mathcal{A} \vdash \mathcal{B}$ résultant de cette preuve permet de conclure, grâce à la table de vérité de “ \Rightarrow ”, que le séquent $\vdash \mathcal{A} \Rightarrow \mathcal{B}$ est aussi vrai; ce séquent ne contient aucune hypothèse. L'arbre de preuve final aura la forme suivante.

$$\frac{\frac{[\mathcal{A}]^{[i]}}{\vdots} \quad \mathcal{B}}{\mathcal{A} \Rightarrow \mathcal{B}}^{[i]}}$$

On dit que l'hypothèse \mathcal{A} est déchargée, et que le séquent représenté par cet arbre de preuve est $\vdash \mathcal{A} \Rightarrow \mathcal{B}$ (c'est-à-dire que \mathcal{A} n'apparaît pas dans les hypothèses du séquent). Par exemple, dans l'arbre suivant,

$$\frac{\mathcal{A}_1 \quad \frac{[\mathcal{A}_2]^{[i]}}{\mathcal{A}_5}}{\mathcal{A}_7} \quad \frac{\mathcal{A}_3 \quad \mathcal{A}_4}{\mathcal{A}_6}^{[R]^{[i]}}}{\mathcal{A}_7}$$

⁶Gerhard Karl Erich Gentzen (24 novembre 1909 - 4 août 1945) est un mathématicien et logicien allemand. Il a apporté des contributions majeures aux fondements des mathématiques, à la théorie de la preuve, notamment sur la déduction naturelle et le calcul séquentiel. Il est mort de faim dans un camp de prisonniers soviétique à Prague en 1945, après avoir été interné en tant que ressortissant allemand après la Seconde Guerre mondiale.

la conclusion de la déduction est $\mathcal{A}_1, \mathcal{A}_3, \mathcal{A}_4 \vdash \mathcal{A}_7$, car \mathcal{A}_2 a été déchargée à la dernière étape à cause de l'application de la règle d'inférence R . On notera une règle qui décharge une hypothèse de la manière suivante:

$$\frac{\begin{array}{c} \lceil \mathcal{A} \rceil^{[i]} \\ \vdots \\ \mathcal{B} \end{array}}{\mathcal{C}}^{[i]}$$

Les “ $\lceil \cdot \rceil$ ” indiquent que \mathcal{A} apparaît comme une feuille de l'arbre (c'est-à-dire une hypothèse).

Pour illustrer ce concept, voici un théorème en algèbre linéaire que nous allons prouver en utilisant tout d'abord une approche classique, et ensuite une approche très formelle.

Théorème 1 Si une matrice carrée E possède un inverse à gauche F et un inverse à droite G , alors $F = G$. □

Dans ce théorème, on a les hypothèses suivantes:

1. $\mathcal{A}_1 \equiv$ “ E est une matrice carrée”
2. $\mathcal{A}_2 \equiv$ “ F est l'inverse à gauche de E ”
3. $\mathcal{A}_3 \equiv$ “ G est l'inverse à droite de E ”

La conclusion (le théorème) est que $F = G$. Lorsque ces trois hypothèses sont vraies, alors la conclusion est vraie. Ce théorème peut s'énoncer de manière formelle comme suit. Tout d'abord, il faut rappeler quelques définitions d'algèbre linéaire. Soit I la matrice identité. On dit qu'une matrice F est un inverse à gauche d'une matrice E ssi $F * E = I$. De manière duale, on dit qu'une matrice G est un inverse à droite d'une matrice E ssi $E * G = I$. Finalement, une matrice M est dite carrée, notée Carrée(M), ssi son nombre de lignes est égal à son nombre de colonnes. On peut formaliser ce théorème de deux manières. Dans la première, on met les hypothèses à gauche du symbole \vdash :

$$\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3 \vdash F = G$$

c'est-à-dire,

$$\text{Carrée}(E), F * E = I, E * G = I \vdash F = G$$

Dans la deuxième forme, on met toutes les hypothèses dans la formule à prouver :

$$\vdash \mathcal{A}_1 \wedge \mathcal{A}_2 \wedge \mathcal{A}_3 \Rightarrow F = G$$

c'est-à-dire,

$$\vdash \text{Carrée}(E) \wedge F * E = I \wedge E * G = I \Rightarrow F = G$$

Les deux formes sont équivalentes. Dans la deuxième forme, il n'y a aucune hypothèse; tout est dans la formule à prouver (i.e., le théorème). La preuve de ce théorème est généralement écrite de manière concise comme suit dans un bouquin d'algèbre linéaire.

$$F = F * I = F * (E * G) = (F * E) * G = I * G = G$$

On peut justifier chaque étape de manière plus rigoureuse comme suit.

$$\begin{aligned}
& F \\
= & \quad \langle \text{Loi de l'identité : pour toute matrice carrée } M, \text{ on a } M * I = M \rangle \\
& F * I \\
= & \quad \langle \text{hypothèse } \mathcal{A}_3 \rangle \\
& F * (E * G) \\
= & \quad \langle \text{Associativité du produit matriciel : } M_1 * (M_2 * M_3) = (M_1 * M_2) * M_3 \rangle \\
& (F * E) * G \\
= & \quad \langle \text{hypothèse } \mathcal{A}_2 \rangle \\
& I * G \\
= & \quad \langle \text{Loi de l'identité} \rangle \\
& G
\end{aligned}$$

Certaines lois de l'algèbre linéaire qui sont utilisées dans cette preuve n'ont pas été déclarées dans le séquent initial. Pour être complètement formel, il faudrait les inclure aussi dans les hypothèses du séquent. On a aussi utilisé de manière implicite les lois suivantes de l'égalité.

$$\frac{x = y \quad y = z}{x = z} \text{ (LE-1)} \qquad \frac{x = y}{f(x) = f(y)} \text{ (LE-2)}$$

Voici un arbre de preuve très détaillé de cette preuve en utilisant la première forme.

$$\frac{\frac{\frac{\forall M \cdot M = M * I}{F = F * I} \text{ E}_{\forall} [M := F]}{F = F * (E * G)} \text{ (LE-1)} \quad \frac{\frac{I = E * G}{F * I = F * (E * G)} \text{ (LE-2)}}{F = (F * E) * G} \text{ (LE-1)} \quad \frac{\text{associativité } *}{F * (E * G) = (F * E) * G} \text{ tr. } =$$

Par manque d'espace, nous continuons cette preuve en prenant la conclusion de l'arbre précédent et en la mettant comme hypothèse de l'arbre ci-dessous (en rouge).

$$\frac{\frac{\vdots}{F = (F * E) * G} \quad \frac{\frac{F * E = I}{(F * E) * G = I * G} \text{ (LE-2)}}{F = I * G} \text{ tr. } = \quad \frac{\frac{\forall M \cdot I * M = M}{I * G = G} \text{ E}_{\forall} [M := G]}{F = G} \text{ tr. } =$$

L'hypothèse “**associativité ***” est l'application de la règle de l'associativité du produit matriciel.

$$\forall M_1, M_2, M_3 \cdot (M_1 * M_2) * M_3 = M_1 * (M_2 * M_3) \text{ avec } [M_1 := F][M_2 := E][M_3 := G]$$

L'hypothèse \mathcal{A}_1 indiquant que la matrice est carrée a été implicitement utilisée pour s'assurer que le produit matriciel de chaque étape est bien défini, c'est-à-dire que les dimensions des matrices sont compatibles pour le produit matriciel. Nous avons aussi utilisé la commutativité de l'égalité de manière implicite. Dans cette preuve, il n'y a aucune hypothèse déchargée.

On constate que si on veut détailler une preuve a priori très simple et énumérer toutes les hypothèses requises, il faut énumérer un nombre considérable de lois du domaine en question. Pour simplifier et illustrer facilement le concept de preuve, nous allons utiliser seulement un ensemble restreint de règles d'inférence de la logique propositionnelle, appelé *la déduction naturelle*, et prouver des tautologies de la logique propositionnelle.

1.4.3 Règles d'inférence de la déduction naturelle

Voici un système de règles appelé *déduction naturelle* pour la logique propositionnelle. Nous utiliserons le logiciel Panda pour appliquer ces règles et faire des preuves. Panda offre une interface graphique simple pour vous permettre de construire des preuves et les vérifier.

$$\begin{array}{c}
\frac{\mathcal{A} \wedge \mathcal{B}}{\mathcal{A}} [E_{\wedge 1}] \quad \frac{\mathcal{A} \wedge \mathcal{B}}{\mathcal{B}} [E_{\wedge 2}] \quad \frac{\mathcal{A} \quad \mathcal{B}}{\mathcal{A} \wedge \mathcal{B}} [I_{\wedge}] \\
\frac{\mathcal{A} \vee \mathcal{B} \quad \begin{array}{c} [\mathcal{A}]^{[i]} \\ \vdots \\ \mathcal{C} \end{array} \quad \begin{array}{c} [\mathcal{B}]^{[j]} \\ \vdots \\ \mathcal{C} \end{array}}{\mathcal{C}} [E_{\vee}]^{[i,j]} \quad \frac{\mathcal{A}}{\mathcal{A} \vee \mathcal{B}} [I_{\vee 1}] \quad \frac{\mathcal{B}}{\mathcal{A} \vee \mathcal{B}} [I_{\vee 2}] \\
\frac{\mathcal{A} \quad \mathcal{A} \Rightarrow \mathcal{B}}{\mathcal{B}} [E_{\Rightarrow}] \quad \frac{\begin{array}{c} [\mathcal{A}]^{[i]} \\ \vdots \\ \mathcal{B} \end{array}}{\mathcal{A} \Rightarrow \mathcal{B}} [I_{\Rightarrow}]^{[i]} \\
\frac{\mathcal{A} \Leftrightarrow \mathcal{B}}{\mathcal{A} \Rightarrow \mathcal{B}} [E_{\Leftrightarrow 1}] \quad \frac{\mathcal{A} \Leftrightarrow \mathcal{B}}{\mathcal{B} \Rightarrow \mathcal{A}} [E_{\Leftrightarrow 2}] \quad \frac{\mathcal{A} \Rightarrow \mathcal{B} \quad \mathcal{B} \Rightarrow \mathcal{A}}{\mathcal{A} \Leftrightarrow \mathcal{B}} [I_{\Leftrightarrow}] \\
\frac{\neg \neg \mathcal{A}}{\mathcal{A}} [E_{\neg}] \quad \frac{\begin{array}{c} [\mathcal{A}]^{[i]} \\ \vdots \\ \perp \end{array}}{\neg \mathcal{A}} [I_{\neg}]^{[i]} \quad \frac{\perp}{\mathcal{A}} [E_{\perp}] \quad \frac{\mathcal{A} \quad \neg \mathcal{A}}{\perp} [I_{\perp}]
\end{array}$$

Le symbole “ \perp ” est un synonyme de faux et il est utilisé par Panda. La règle ConDict est parfois utilisée à la place de la règle I_{\neg} chez certains auteurs. Elle est obtenue en combinant les règles I_{\neg} et E_{\neg} .

$$\frac{\begin{array}{c} [\neg \mathcal{A}]^{[i]} \\ \vdots \\ \perp \end{array}}{\mathcal{A}} [\text{ConDict}]^{[i]}$$

Voici quelques exemples de preuve, où p, q, r dénotent des formules quelconques.

$$1. \vdash (p \Rightarrow (q \Rightarrow r)) \Rightarrow ((p \wedge q) \Rightarrow r)$$

$$\frac{\frac{\frac{[p \wedge q]^{[1]}}{p} [E_{\wedge 1}] \quad \frac{[p \Rightarrow (q \Rightarrow r)]^{[2]}}{q \Rightarrow r} [E_{\Rightarrow}] \quad \frac{[p \wedge q]^{[1]}}{q} [E_{\wedge 2}]}{r} [I_{\Rightarrow}]^{[1]}}{(p \wedge q) \Rightarrow r} [I_{\Rightarrow}]^{[1]}}{(p \Rightarrow (q \Rightarrow r)) \Rightarrow ((p \wedge q) \Rightarrow r)} [I_{\Rightarrow}]^{[2]}$$

$$2. \vdash (p \Rightarrow (q \wedge r)) \Rightarrow ((p \Rightarrow q) \wedge (p \Rightarrow r))$$

$$\frac{\frac{\frac{[p \Rightarrow (q \wedge r)]^{[1]} \quad [p]^{[2]}}{q \wedge r} [E_{\wedge}] \quad \frac{[p \Rightarrow (q \wedge r)]^{[1]} \quad [p]^{[3]}}{q \wedge r} [E_{\wedge}]}{\frac{q}{p \Rightarrow q} [I_{\Rightarrow}]^{[2]} \quad \frac{r}{p \Rightarrow r} [I_{\Rightarrow}]^{[3]}} [I_{\wedge}]}{\frac{(p \Rightarrow q) \wedge (p \Rightarrow r)}{(p \Rightarrow (q \wedge r)) \Rightarrow ((p \Rightarrow q) \wedge (p \Rightarrow r))} [I_{\Rightarrow}]^{[1]}}$$

3. $\vdash ((p \Rightarrow q) \wedge (p \Rightarrow r)) \Rightarrow (p \Rightarrow (q \wedge r))$

$$\frac{\frac{\frac{[(p \Rightarrow q) \wedge (p \Rightarrow r)]^{[1]}}{p \Rightarrow q} [E_{\wedge 1}] \quad [p]^{[2]} [E_{\Rightarrow}] \quad \frac{[(p \Rightarrow q) \wedge (p \Rightarrow r)]^{[1]} \quad [p]^{[2]} [E_{\wedge 2}]}{p \Rightarrow r} [E_{\wedge 2}] \quad [p]^{[2]} [E_{\Rightarrow}]}{q \quad r} [I_{\wedge}]}{\frac{q \wedge r}{p \Rightarrow (q \wedge r)} [I_{\Rightarrow}]^{[2]}} [I_{\Rightarrow}]^{[1]}$$

4. $\vdash p \vee (q \wedge r) \Rightarrow (p \vee q) \wedge (p \vee r)$

$$\frac{\frac{[p \vee (q \wedge r)]^{[1]} \quad \frac{\frac{[p]^{[2]}}{p \vee q} [I_{\vee 1}] \quad \frac{[p]^{[2]}}{p \vee r} [I_{\vee 1}]}{(p \vee q) \wedge (p \vee r)} [I_{\wedge}] \quad \frac{\frac{[q \wedge r]^{[3]}}{q} [E_{\wedge 1}] \quad \frac{[q \wedge r]^{[3]}}{r} [E_{\wedge 2}]}{p \vee q \quad p \vee r} [I_{\wedge}] \quad \frac{[q \wedge r]^{[3]}}{(p \vee q) \wedge (p \vee r)} [E_{\vee}]^{[2,3]}}{(p \vee q) \wedge (p \vee r)} [I_{\wedge}]^{[1]}}{p \vee (q \wedge r) \Rightarrow (p \vee q) \wedge (p \vee r)} [I_{\Rightarrow}]^{[1]}$$

5. $\vdash \neg(p \vee q) \Rightarrow \neg p \wedge \neg q$

$$\frac{\frac{\frac{[\neg(p \vee q)]^{[1]} \quad \frac{[p]^{[2]}}{p \vee q} [I_{\vee 1}]}{\neg p} [I_{\neg}]^{[2]} \quad \frac{[\neg(p \vee q)]^{[1]} \quad \frac{[q]^{[3]}}{p \vee q} [I_{\vee 2}]}{\neg q} [I_{\neg}]^{[3]}}{\neg p \wedge \neg q} [I_{\wedge}]}}{\neg(p \vee q) \Rightarrow \neg p \wedge \neg q} [I_{\Rightarrow}]^{[1]}$$

6. $\vdash p \wedge (q \vee r) \Rightarrow (p \wedge q) \vee (p \wedge r)$

$$\frac{\frac{\frac{[p \wedge (q \vee r)]^{[1]} \quad [q]^{[2]} [I_{\wedge 1}]}{p \wedge q} [E_{\wedge 1}] \quad \frac{[p \wedge (q \vee r)]^{[1]} \quad [r]^{[3]} [I_{\wedge 2}]}{p \wedge r} [E_{\wedge 2}]}{(p \wedge q) \vee (p \wedge r)} [I_{\vee 1}] \quad \frac{[p \wedge (q \vee r)]^{[1]} \quad [r]^{[3]} [I_{\wedge 1}]}{p \wedge r} [E_{\wedge 1}] \quad \frac{[p \wedge (q \vee r)]^{[1]} \quad [q]^{[2]} [I_{\wedge 2}]}{(p \wedge q) \vee (p \wedge r)} [I_{\vee 2}]}{\frac{(p \wedge q) \vee (p \wedge r)}{p \wedge (q \vee r) \Rightarrow (p \wedge q) \vee (p \wedge r)} [I_{\Rightarrow}]^{[1]}}$$

7. $\vdash (p \Rightarrow (q \Rightarrow r)) \Rightarrow (q \Rightarrow (p \Rightarrow r))$

$$\frac{\frac{\frac{[p \Rightarrow (q \Rightarrow r)]^{[1]} \quad [p]^{[2]}}{q \Rightarrow r} [E_{\Rightarrow}] \quad [q]^{[3]}}{\frac{r}{p \Rightarrow r} [I_{\Rightarrow}]^{[2]}} [E_{\Rightarrow}] \quad \frac{\frac{r}{p \Rightarrow r} [I_{\Rightarrow}]^{[2]}}{q \Rightarrow (p \Rightarrow r)} [I_{\Rightarrow}]^{[3]}}{(p \Rightarrow (q \Rightarrow r)) \Rightarrow (q \Rightarrow (p \Rightarrow r))} [I_{\Rightarrow}]^{[1]}$$

8. $\vdash \neg(p \wedge q) \Rightarrow \neg p \vee \neg q$

$$\frac{\frac{\frac{[\neg p]^{[1]}}{\neg p \vee \neg q} [I_{\vee 1}] \quad [\neg(\neg p \vee \neg q)]^{[2]}}{\perp} [I_{\perp}] \quad \frac{\frac{[\neg q]^{[3]}}{\neg p \vee \neg q} [I_{\vee 2}] \quad [\neg(\neg p \vee \neg q)]^{[2]}}{\perp} [I_{\perp}] \quad \frac{\frac{\perp}{\neg \neg p} [I_{\neg}]^{[1]} \quad \frac{\perp}{\neg \neg q} [I_{\neg}]^{[3]}}{p} [E_{\neg}] \quad \frac{\frac{\perp}{q} [I_{\wedge}] \quad [\neg(p \wedge q)]^{[4]}}{p \wedge q} [I_{\wedge}] \quad \frac{\frac{\perp}{\neg \neg(\neg p \vee \neg q)} [I_{\neg}]^{[2]} \quad \frac{\perp}{\neg p \vee \neg q} [E_{\neg}] \quad \frac{\perp}{\neg(p \wedge q) \Rightarrow \neg p \vee \neg q} [I_{\Rightarrow}]^{[4]}}{\neg(p \wedge q) \Rightarrow \neg p \vee \neg q} [I_{\perp}]^{[4]}}$$

9. $\vdash p \wedge (\neg p \vee q) \Rightarrow p \wedge q$

$$\frac{\frac{\frac{[p \wedge (\neg p \vee q)]^{[1]}}{p} [E_{\wedge 1}] \quad [\neg p]^{[2]}}{\perp} [I_{\perp}] \quad \frac{[p \wedge (\neg p \vee q)]^{[1]} \quad [q]^{[3]}}{p} [E_{\wedge 1}] \quad \frac{[p \wedge (\neg p \vee q)]^{[1]}}{\neg p \vee q} [E_{\wedge 2}] \quad \frac{\perp}{p \wedge q} [E_{\perp}] \quad \frac{p \wedge q}{p \wedge q} [E_{\vee}]^{[2,3]}}{\frac{p \wedge q}{p \wedge (\neg p \vee q) \Rightarrow p \wedge q} [I_{\Rightarrow}]^{[1]}}$$

10. $\vdash p \vee (\neg p \wedge q) \Rightarrow p \vee q$

$$\frac{\frac{[p \vee (\neg p \wedge q)]^{[1]} \quad \frac{[p]^{[2]}}{p \vee q} [I_{\vee 1}] \quad \frac{[\neg p \wedge q]^{[3]}}{q} [E_{\wedge 2}]}{\frac{q}{p \vee q} [I_{\vee 2}]} [E_{\vee}]^{[2,3]} \quad \frac{p \vee q}{p \vee (\neg p \wedge q) \Rightarrow p \vee q} [I_{\Rightarrow}]^{[1]}}$$

1.4.4 Cohérence, conséquence et complétude

Définition 10 Une *valuation* est une affectation de valeurs à des variables propositionnelles; elle est notée $[X_1 := v_1, \dots, X_n := v_n]$, où $v_i \in \{0, 1\}$. Certains auteurs utilisent le terme *interprétation* comme synonyme de valuation. La valeur de vérité d'une formule propositionnelle \mathcal{A} pour une valuation V est notée \mathcal{AV} . Cette valeur de vérité est calculée en remplaçant les variables de la formule par leur valeur donnée par la valuation, et en calculant le résultant à l'aide des tables de vérité. \square

Exemple 1 Par exemple, la valeur de vérité de la formule $X_1 \wedge \neg X_2$ sous la valuation $[X_1 := 1, X_2 := 0]$, notée $(X_1 \wedge \neg X_2)[X_1 := 1, X_2 := 0]$, est 1. On peut la calculer ainsi à l'aide des tables de vérité.

$$\begin{aligned}
& (X_1 \wedge \neg X_2)[X_1 := 1, X_2 := 0] \\
= & \hspace{10em} \langle \text{substitution des variables par leur valeur} \rangle \\
& 1 \wedge \neg 0 \\
= & \hspace{15em} \langle \text{calcul de } \neg 0 = 1 \rangle \\
& 1 \wedge 1 \\
= & \hspace{15em} \langle \text{calcul de } 1 \wedge 1 = 1 \rangle \\
& 1
\end{aligned}$$

□

Définition 11 On dit qu'une valuation V *satisfait* une formule \mathcal{A} , notée $V \models \mathcal{A}$, ssi $\mathcal{A}V = 1$. Lorsque $V \models \mathcal{A}$, on dit que V est un *modèle* de \mathcal{A} , et que \mathcal{A} est *satisfaisable*. Une formule \mathcal{A} est dite *valide*, notée $\models \mathcal{A}$, ssi toute valuation V est un modèle de \mathcal{A} . Une tautologie est donc un synonyme de formule valide. Soit $\Gamma = \{\mathcal{A}_1, \dots, \mathcal{A}_n\}$ un ensemble de formules propositionnelles et V une valuation; on dit que V est un modèle de Γ , noté $V \models \Gamma$, ssi V est un modèle pour chaque formule de Γ . □

Dans l'exemple 1, la valuation $[X_1 := 1, X_2 := 0]$ est un modèle de la formule $(X_1 \wedge \neg X_2)$

On utilise la notation $\{e_1, \dots, e_n\}$ pour dénoter un *ensemble* formé des éléments e_1, \dots, e_n . La notion d'ensemble sera définie formellement plus tard dans le cours.

Définition 12 Un ensemble de formules propositionnelles Γ est dit *cohérent* ssi il existe un modèle V pour Γ . □

Par exemple, les trois formules suivantes ne sont pas tautologiquement cohérentes.

$$aLaGrippe \Rightarrow faitDeLaFièvre \tag{1.25}$$

$$aLaGrippe \tag{1.26}$$

$$\neg faitDeLaFièvre \tag{1.27}$$

Si quelqu'un affirme que ces trois formules sont vraies, alors on peut dire qu'il est incohérent, car il n'existe pas de valuation permettant de satisfaire les trois formules en même temps. La seule valuation pouvant satisfaire la formule (1.26) contient $aLaGrippe := 1$. La seule valuation pouvant satisfaire la formule (1.27) contient $faitDeLaFièvre := 0$. La valuation

$$[aLaGrippe := 1, faitDeLaFièvre := 0]$$

ne satisfait pas la formule (1.25). On peut donc systématiquement vérifier si un discours ou un document de spécification est cohérent en calculant toutes les valuations possibles. Il peut exister plusieurs valuations satisfaisant l'ensemble des formules considérées; l'important est qu'il en existe au moins une.

Définition 13 On dit que \mathcal{B} est une *conséquence logique* de Γ , notée $\Gamma \models \mathcal{B}$, ssi tout modèle de Γ est aussi un modèle de \mathcal{B} . □

La relation de conséquence logique “ \models ” est donc calculée avec les tables de vérité des opérateurs logiques, alors de la relation de déduction “ \vdash ” est calculée avec les règles d’inférence. Si une formule comprend un grand nombre de variables, utiliser les règles d’inférence est beaucoup plus concis (mais pas automatique), car la longueur d’une table de vérité d’une formule augmente de façon exponentielle avec le nombre de variables. Si une formule de logique du premier ordre comprend une variable x représentant un nombre naturel, alors il devient impossible de calculer la table de vérité d’une formule, car le nombre de valeurs possibles pour x est infini. C’est pourquoi on utilise les règles d’inférence en mathématique pour prouver des théorèmes, mais ce n’est pas un processus automatique. Le calcul des tables de vérité pour une formule propositionnelle peut être entièrement automatique, mais cela prend un temps exponentiel en fonction du nombre de variables propositionnelles.

Exemple 2 Dans le tableau ci-dessous, on montre que la formule X_3 est une conséquence logique des formules $X_1 \wedge X_2$ et $(X_1 \wedge X_2) \Rightarrow X_3$, c’est-à-dire

$$\{ X_1 \wedge X_2, \quad (X_1 \wedge X_2) \Rightarrow X_3 \} \models X_3$$

no	X_1	X_2	X_3	$X_1 \wedge X_2$	$(X_1 \wedge X_2) \Rightarrow X_3$	X_3
1	0	0	0	0	1	0
2	0	0	1	0	1	1
3	0	1	0	0	1	0
4	0	1	1	0	1	1
5	1	0	0	0	1	0
6	1	0	1	0	1	1
7	1	1	0	1	0	0
8	1	1	1	1	1	1

Il n’y qu’un seul modèle des formules à gauche du symbole \models , soit la ligne 8. Il faut donc vérifier que la formule X_3 est vraie aussi pour ce modèle, et c’est bien le cas. \square

Exemple 3 Dans le tableau ci-dessous, on montre que la formule X_3 n’est pas une conséquence logique des formules $X_1 \vee X_2$ et $(X_1 \wedge X_2) \Rightarrow X_3$.

$$\{ X_1 \vee X_2, \quad (X_1 \wedge X_2) \Rightarrow X_3 \} \not\models X_3$$

no	X_1	X_2	X_3	$X_1 \vee X_2$	$(X_1 \wedge X_2) \Rightarrow X_3$	X_3
1	0	0	0	0	1	0
2	0	0	1	0	1	1
3	0	1	0	1	1	0
4	0	1	1	1	1	1
5	1	0	0	1	1	0
6	1	0	1	1	1	1
7	1	1	0	1	0	0
8	1	1	1	1	1	1

Les formules à gauche du symbole $\not\models$ ont cinq modèles, soit les lignes 3, 4, 5, 6 et 8. Il faut donc vérifier que la formule X_3 est vraie pour chacun de ces modèles, et ce n’est pas le cas pour les lignes 3 et 5. \square

Une autre façon de définir la cohérence est d'utiliser la relation déduction. On dit qu'un ensemble de formule $\mathcal{A}_1, \dots, \mathcal{A}_n$ est cohérent ssi il n'est pas possible d'obtenir $\mathcal{A}_1, \dots, \mathcal{A}_n \vdash \mathcal{B}$ et $\mathcal{A}_1, \dots, \mathcal{A}_n \vdash \neg\mathcal{B}$. On peut montrer que les formules (1.25), (1.26), (1.27) sont incohérentes, à l'aide de la déduction suivante.

$$\frac{aLaGrippe \quad aLaGrippe \Rightarrow faitDeLaFièvre}{faitDeLaFièvre} [E\Rightarrow]$$

On peut donc déduire la formule *faitDeLaFièvre*, ce qui contredit la formule (1.27). Cette approche est moins facile à automatiser. La plupart des approches de vérification de la cohérence utilisent la recherche d'un modèle à l'aide des formes normales, qui sont l'objet de la prochaine section.

Les règles d'inférence de la déduction naturelle sont dites *cohérentes*, car si $\Gamma \vdash \mathcal{B}$, alors $\Gamma \models \mathcal{B}$, c'est-à-dire tout théorème \mathcal{B} est aussi une conséquence logique. Ces règles sont aussi dites *complètes*, car si $\Gamma \models \mathcal{B}$, alors $\Gamma \vdash \mathcal{B}$, c'est-à-dire toute conséquence logique est aussi un théorème.

Théorème 2 $\Gamma \models \mathcal{B}$ ssi $\Gamma \vdash \mathcal{B}$. □

Ainsi, on peut calculer si une formule est vraie en utilisant soit les tables de vérité, soit les règles d'inférences. C'est une propriété fondamentale de la logique. Notons que \vdash et \models sont deux relations distinctes sur les formules propositionnelles; la première est calculée avec les règles d'inférence; la seconde est calculée avec les tables de vérités, c'est-à-dire sur les modèles. Pour calculer $\Gamma \models \mathcal{B}$ en utilisant la définition 13, il faut évaluer 2^n valuations, où n est le nombre de variables propositionnelles apparaissant dans les formules de Γ et \mathcal{B} .

Les outils utilisant de vérification en logique plutôt le théorème suivant, qui réduit le problème de conséquence logique au problème de satisfaction d'un ensemble de formules.

Théorème 3 $\Gamma \models \mathcal{B}$ ssi l'ensemble des formules $\Gamma \cup \{\neg\mathcal{B}\}$ est incohérent. □

Le symbole \cup est l'opérateur usuel d'union sur les ensembles. Finalement, la satisfaction d'un ensemble de formules peut être réduit au problème de la satisfaction d'une formule, qui est NP-complet.

Théorème 4 Un ensemble de formules $\{\mathcal{A}_1, \dots, \mathcal{A}_n\}$ est cohérent ssi la formule $\mathcal{A}_1 \wedge \dots \wedge \mathcal{A}_n$ est satisfaisable. □

Le théorème 2 est un théorème dit de *complétude* en logique. Cette propriété de complétude est très importante. Quand on définit un ensemble de règles d'inférence pour faire de la preuve, on cherche souvent à démontrer que le système d'inférence est *complet*, c'est-à-dire qu'il permet de démontrer tout ce qu'on voudrait pouvoir démontrer, comme dans le théorème 2. Une théorie peut être complète sans pour autant être décidable; c'est le cas de l'arithmétique avec l'addition et la multiplication. Gödel a démontré en 1929 que la logique du premier ordre est complète, c'est-à-dire que pour un ensemble de formules Γ , une formule est vraie dans tous les modèles de Γ ssi il existe une preuve de cette formule. Un ensemble de règles d'inférence est dit cohérent si tout ce qu'il permet de déduire est vrai dans tous les modèles.

1.4.5 Utilisation de la logique

1.4.5.1 Vérification de systèmes

Les logiciels sont généralement spécifiés avec la logique du premier ordre, qui est l'objet de la prochaine section. Toute formule représentant la correction d'un logiciel peut être traduite en une formule propositionnelle (souvent gigantesque). Quand on désire faire la preuve de correction d'un

logiciel, on a deux options. Soit on utilise les algorithmes de satisfaction d'une formule propositionnelle (c'est-à-dire, la relation " \models ") si le nombre de variables est relativement petit (quelques dizaines de million de variables!). Si le nombre de variables est trop grand, on utilisera les règles d'inférence et la déduction (donc, la relation " \vdash "), car le calcul d'une preuve est indépendant du nombre de variables propositionnelles. Malheureusement, la plupart des logiciels nécessitent un nombre astronomique (très très grand!) de variables propositionnelles et la relation \models n'est pas calculable en un temps et un espace mémoire acceptables. D'autre part, Gödel⁷ a démontré que l'arithmétique est *indécidable*, au sens où il existe des formules qui ne peuvent être ni prouvées ni réfutées (cf, le théorème d'incomplétude de l'arithmétique de 1931). En 1936, Turing et Church ont montré qu'il existe des problèmes indécidables, c'est-à-dire qu'il n'existe pas d'algorithme pouvant déterminer s'il existe une preuve d'un séquent $\Gamma \vdash \mathcal{B}$ pour n'importe quel Γ et \mathcal{B} . C'est pour ce problème que Turing a défini le concept de machine de Turing, le premier modèle mathématique abstrait d'un ordinateur, afin de définir formellement la notion d'algorithme. Sa preuve montre qu'il n'existe pas de machine qui peut lire la description d'une machine et déterminer si elle termine (i.e., communément appelé le problème de l'arrêt). À la même époque, Church a défini le λ -calcul pour montrer qu'il n'existe pas de fonction récursive permettant de dire si deux λ -expressions sont équivalentes. Le λ -calcul constitue la deuxième définition de la notion d'algorithme.

Les théorèmes de Gödel, Church et Turing établissent les limites de la déduction en logique. Bien sûr, cela n'empêche pas de faire de la preuve automatisée. En pratique, les prouveurs automatisés arrivent très souvent à trouver une preuve pour les séquents; ils utilisent des heuristiques pour le faire, ou bien ils sont guidés par un humain dans la recherche d'une preuve. Si on n'arrive pas à trouver une preuve d'un séquent $\Gamma \vdash \mathcal{B}$, on peut alors essayer de prouver $\Gamma \vdash \neg\mathcal{B}$, ou bien de satisfaire $\Gamma \cup \neg\mathcal{B}$. Si on trouve un modèle pour $\Gamma \cup \neg\mathcal{B}$, alors on sait que $\Gamma \vdash \mathcal{B}$ est faux, donc que $\Gamma \vdash \neg\mathcal{B}$ est vrai. Bien sûr, il faut aussi s'assurer que Γ est cohérent, car si Γ est incohérent, alors on peut prouver à la fois $\Gamma \vdash \neg\mathcal{B}$ et $\Gamma \vdash \mathcal{B}$. Si Γ est incohérent, cela signifie que la spécification du logiciel est mal construite.

1.4.5.2 Théorie et modélisation

En mathématique, on cherche généralement à trouver un ensemble minimal de formules Γ à partir desquels on peut prouver les théorèmes intéressants (pertinents) d'un domaine. On appelle alors les éléments de Γ des *axiomes*, et l'ensemble des théorèmes que l'on peut déduire à partir de ces axiomes est appelée une *théorie*. Par exemple, on a la théorie des nombres naturels, des nombre entiers, etc. L'algèbre est une discipline des mathématiques qui s'intéresse à l'étude de théories (ex: l'algèbre linéaire s'intéresse aux résultats que l'on peut prouver sur les matrices à partir d'un nombre restreint d'axiomes). L'algèbre s'intéresse aussi à généraliser les propriétés d'un grand nombre de théories particulières; par exemple, la théorie des *monoïdes* s'intéresse à prouver des théorèmes pour n'importe quel ensemble d'objets où on a une opération associative (ex: $+$ sur les nombres naturels) et un élément neutre (ex: 0 dans les nombres naturels). Cela donne les deux axiomes suivants:

$$\begin{array}{ll} x + (y + z) = (x + y) + z & \text{associativité} \\ x + 0 = x & \text{élément neutre} \end{array}$$

Les nombres naturels avec $+$ et 0 sont un monoïde. Les nombres naturels avec $*$ et 1 forment aussi un monoïde. Les matrices avec la multiplication et la matrice identité forment aussi un monoïde. Tout théorème prouvé à partir de ces deux axiomes sera un théorème dans n'importe quel monoïde.

⁷Kurt Gödel, né le 28 avril 1906 en Autriche et mort le 14 janvier 1978 à Princeton (New Jersey), est un logicien et mathématicien autrichien naturalisé américain.

L'algèbre est à l'origine de la création du concept de classe en programmation orientée objets. C'est ce qui inspira Ole-Johan Dahl⁸ et Kristen Nygaard⁹ pour définir le premier langage de programmation orienté objets, appelé Simula, dans les années 1960. Ils reçurent le prix Alan Turing en 2001 pour cette contribution. Les mathématiques sont une grande source d'inspiration pour plusieurs concepts fondamentaux en informatique. Notons qu'il fallut plus de 30 ans avant que la programmation orientée objets deviennent une pratique courante en informatique. Alan Kay¹⁰, qui travailla au centre de recherche de Xerox à Palo Alto en Californie, et ensuite chez Apple, développa le langage Smalltalk, inspiré de Simula, qui fut l'un des premiers langage orienté objets largement utilisé et popularisé par le MacIntosh d'Apple dans les années 1980. Il est aussi l'un des concepteurs de la notion d'interface graphique pour interagir avec un ordinateur; avant, les interactions se faisaient en ligne de commande sur un terminal. Il reçut le prix Alan Turing en 2003 pour ces contributions.

Pour la vérification de logiciels critiques, on utilise des logiciels, appelés prouveurs de théorèmes, pour vérifier la correction des systèmes (ex: COQ¹¹, HOL¹², Isabelle¹³, PVS¹⁴, Why3¹⁵, Z3¹⁶). Ces prouveurs permettent de définir des théories et des règles d'inférences. Le langage B, qui est utilisé dans ce cours, est supporté par plusieurs de ces prouveurs.

1.5 Preuve en logique du premier ordre

1.5.1 Règles d'inférence

Les règles d'inférence de la logique propositionnelle s'appliquent aussi à la logique du premier ordre, car elle contient aussi les connecteurs propositionnels. Il reste à ajouter des règles pour les quantificateurs existentiel (\exists) et universel (\forall).

$$\frac{\forall x \cdot \mathcal{A}}{\mathcal{A}[x := t]} \text{ [E}_{\forall}\text{]} \qquad \frac{\mathcal{A}}{\forall x \cdot \mathcal{A}} \text{ [I}_{\forall}\text{]} \text{ où } x \text{ n'est pas libre dans les hypothèses de } \mathcal{A}$$

$$\frac{\mathcal{A}[x := t]}{\exists x \cdot \mathcal{A}} \text{ [I}_{\exists}\text{]} \qquad \frac{\begin{array}{c} [\mathcal{A}]^{[i]} \\ \vdots \\ \exists x \cdot \mathcal{A} \quad \mathcal{B} \end{array}}{\mathcal{B}} \text{ [E}_{\exists}\text{]}^{[i]} \text{ où } x \text{ n'est pas libre dans } \mathcal{B} \text{ ni dans les hypothèses de } \mathcal{B}$$

1.6 Formes normales

Plusieurs travaux en logique, comme l'analyse de complexité algorithmique de la satisfaction d'une formule et la preuve automatisée de théorèmes, nécessitent de transformer une formule en une autre

⁸Ole-Johan Dahl (12 octobre 1931 – 29 juin 2002) est un informaticien norvégien.

⁹Kristen Nygaard (27 août 1926 - 10 août 2002) était un mathématicien et informaticien norvégien.

¹⁰Alan C. Kay (17 mai 1940), est un informaticien américain.

¹¹coq.inria.fr

¹²hol-theorem-prover.org

¹³isabelle.in.tum.de

¹⁴pvs.csl.sri.com

¹⁵why3.lri.fr

¹⁶z3prover.github.io

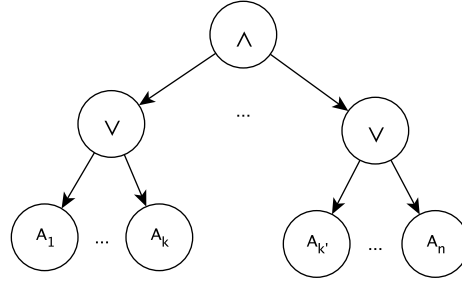


Figure 1.4: La représentation sous forme d'arbre de la FNC

formule équivalente, mais sous une forme appelée “normale”. Il y a deux principales formes normales, soit conjonctive et disjonctive.

Définition 14 Une formule est dite en *forme normale conjonctive* (FNC) ssi elle est de la forme

$$(\mathcal{A}_1 \vee \dots \vee \mathcal{A}_k) \wedge \dots \wedge (\mathcal{A}_{k'} \vee \dots \vee \mathcal{A}_n)$$

où chaque formule constituante \mathcal{A}_i est soit une variable propositionnelle (X_i), soit la négation d'une variable propositionnelle ($\neg X_i$). \square

Ainsi, une formule en FNC est une “conjonction de disjonctions”, et les éléments des disjonctions sont des variables ou des négation de variables. Puisque les connecteurs “ \wedge ” et “ \vee ” sont associatifs et commutatifs, il n'est pas nécessaire de parenthéser la conjonction principale et les disjonctions à l'intérieur de la conjonction principale. La figure 1.4 illustre la forme générale d'une FNC. On y représente la conjonction avec un seul opérateur, et chaque disjonction est aussi représentée par un seul opérateur.

Exemple 4 Les formules suivantes sont en FNC

$$(X_1 \vee \neg X_2) \wedge (X_1 \vee X_4) \tag{1.28}$$

$$(X_1 \vee \neg X_2) \wedge \neg X_3 \wedge (X_1 \vee X_4 \vee X_5) \tag{1.29}$$

$$X_1 \vee \neg X_3 \tag{1.30}$$

$$X_1 \wedge \neg X_3 \tag{1.31}$$

$$X_1 \tag{1.32}$$

$$\neg X_1 \tag{1.33}$$

La formule (1.29) comporte une particularité, car elle contient une disjonction ($\neg X_3$) de longueur 1, c'est-à-dire que cette disjonction est formée d'une seule formule, donc le connecteur “ \vee ” n'a pas besoin d'y apparaître. La figure 1.5 représente cette formule sous forme d'un arbre, en faisant apparaître le connecteur “ \vee ” pour cette disjonction de longueur 1. La version textuelle de cette formule omet le connecteur “ \vee ”, afin qu'elle soit syntaxiquement correcte. La formule (1.30) est aussi en FNC, même si elle ne contient pas de conjonction. Dans ce cas, on considère qu'il s'agit d'une conjonction de longueur 1, c'est-à-dire que la conjonction ne contient qu'une seule formule, et donc le connecteur “ \wedge ” n'a pas besoin d'y apparaître. On notera plus tard que cette formule est

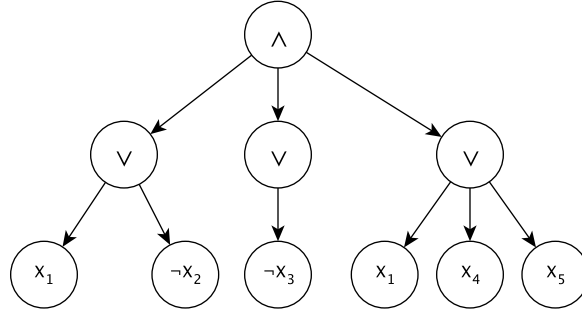


Figure 1.5: La représentation sous forme d'arbre de la formule (1.29), qui est en FNC

aussi en forme normale disjonctive. La formule (1.31) est en FNC : toutes ses disjonctions sont de longueur 1; elle ne contient donc pas de connecteurs de disjonction. Finalement, la formule (1.32) est une conjonction de longueur 1, dont la seule disjonction est aussi de longueur 1. Idem pour la formule (1.33).

Les formules suivantes ne sont pas en FNC. Les parties qui font en sorte que la définition est violée sont en rouge.

$$(X_1 \vee X_4) \wedge \neg(X_2 \vee \neg X_3) \quad (1.34)$$

$$X_1 \vee (X_2 \wedge \neg X_3) \quad (1.35)$$

$$X_1 \wedge (X_2 \vee (X_3 \wedge X_4)) \quad (1.36)$$

La formule (1.34) n'est pas en FNC, car la négation s'applique à une formule; elle ne peut s'appliquer qu'à une variable. La formule (1.35) n'est pas en FNC, car la disjonction s'applique à une conjonction; elle ne peut s'appliquer qu'à une variable ou à la négation d'une variable. Idem pour la formule (1.36). □

Définition 15 Une formule est dite en *forme normale disjonctive* (FND) ssi elle est de la forme

$$(\mathcal{A}_1 \wedge \dots \wedge \mathcal{A}_k) \vee \dots \vee (\mathcal{A}_{k'} \wedge \dots \wedge \mathcal{A}_n)$$

où chaque formule constituante \mathcal{A}_i est soit une variable propositionnelle (X_i), soit la négation d'une variable propositionnelle ($\neg X_i$). □

Exemple 5 Les formules suivantes sont en FND

$$(X_1 \wedge \neg X_2) \vee \neg X_3 \vee (X_1 \wedge X_4 \wedge X_5)$$

$$X_1 \vee \neg X_3$$

$$X_1 \wedge \neg X_3$$

$$X_1$$

$$\neg X_1$$

Les formules suivantes ne sont pas en FND. Les parties qui font en sorte que la définition est violée sont en rouge.

$$(X_1 \wedge X_4) \vee \neg(X_2 \wedge \neg X_3)$$

$$X_1 \wedge (X_2 \vee \neg X_3)$$

$$X_1 \vee (X_2 \wedge (X_3 \vee X_4))$$

□

Toute formule propositionnelle peut être transformée en une formule équivalente en FNC à l'aide des lois (LP-12), (LP-17), (LP-18), (LP-21), (LP-22) et (LP-40), ou en FND, avec les mêmes lois, mais en utilisant (LP-11) au lieu de (LP-12). Ces lois permettent de ré-écrire une formule pour déplacer les connecteurs vers l'intérieur de la formule et remplacer les connecteurs \Leftrightarrow et \Rightarrow .

Voici un exemple de transformation d'une formule en FNC.

$$\neg(X_1 \vee \neg X_2) \vee X_3$$

$$\Leftrightarrow \langle \text{Déplacement de la négation vers l'intérieur (LP-18)} \rangle$$

$$(\neg X_1 \wedge \neg \neg X_2) \vee X_3$$

$$\Leftrightarrow \langle \text{Élimination de la double négation (LP-21)} \rangle$$

$$(\neg X_1 \wedge X_2) \vee X_3$$

$$\Leftrightarrow \langle \text{Commutativité (afin d'appliquer (LP-10) à l'étape suivante) (LP-8)} \rangle$$

$$X_3 \vee (\neg X_1 \wedge X_2)$$

$$\Leftrightarrow \langle \text{Déplacement de la disjonction vers l'intérieur (LP-10)} \rangle$$

$$(X_3 \vee \neg X_1) \wedge (X_3 \vee X_2)$$

La FND est calculée de manière similaire.

$$\neg(X_1 \vee \neg X_2) \vee X_3$$

$$\Leftrightarrow \langle \text{Déplacement de la négation vers l'intérieur (LP-18)} \rangle$$

$$(\neg X_1 \wedge \neg \neg X_2) \vee X_3$$

$$\Leftrightarrow \langle \text{Élimination de la double négation (LP-21)} \rangle$$

$$(\neg X_1 \wedge X_2) \vee X_3$$

1.7 Exercices

1. Prouvez les formules suivantes en utilisant seulement les règles d'inférence de la déduction naturelle. Indiquez pour chaque étape de preuve, la règle utilisée et les hypothèses déchargées (s'il y a déchargement avec cette règle). Toutes les hypothèses doivent être déchargées, puisque chaque séquent ci-dessous ne contient aucune hypothèse (c'est-à-dire, chaque séquent est de la forme $\vdash \mathcal{A}$).

(a) $\vdash (p \vee q) \wedge (p \vee r) \Rightarrow p \vee (q \wedge r)$

(b) $\vdash (p \wedge q) \vee (p \wedge r) \Rightarrow p \wedge (q \vee r)$

(c) $\vdash p \wedge q \Rightarrow p \wedge (\neg p \vee q)$

(d) $\vdash \neg\neg p \Rightarrow p$

(e) $\vdash p \Rightarrow \neg\neg p$

(f) $\vdash (p \Rightarrow q) \Rightarrow (\neg q \Rightarrow \neg p)$

(g) $\vdash (\neg q \Rightarrow \neg p) \Rightarrow (p \Rightarrow q)$

(h) $\vdash \neg p \wedge \neg q \Rightarrow \neg(p \vee q)$

(i) $\vdash \neg p \vee \neg q \Rightarrow \neg(p \wedge q)$

2. Prouvez les formules suivantes avec les règles d'inférences de la déduction naturelle.

(a) $\vdash a \vee \neg a$ (loi du tiers exclu, c'est-à-dire la loi LP-20)

(b) $\vdash a \vee b \Rightarrow a \vee (\neg a \wedge b)$

(c) $\vdash (a \Rightarrow b) \Leftrightarrow (\neg a \vee b)$

3. Prouvez les lois (LP-23) à (LP-43) en utilisant le style équationnel et les lois (LP-1) à (LP-22). Il n'est pas nécessaire de toutes les prouver pour se préparer à l'examen. Plusieurs ont déjà été prouvées en exercices ou en devoir.

4. Transformez les formules suivantes en FNC et en FND.

(a) $\neg(X_1 \Rightarrow \neg(X_2 \vee X_3))$

(b) $\neg(X_1 \vee \neg((X_2 \wedge X_3) \Rightarrow \neg X_4))$

5. Déterminez si $X_1 \Rightarrow \neg(X_2 \vee X_3)$, $X_1 \wedge X_2 \models \neg X_3$

6. Traduisez les phrases données dans les exemples disponibles dans le répertoire ci-dessous en logique avec le langage de Tarski UdeS.

<https://marcfrappier.espaceweb.usherbrooke.ca/mat115/exercices/chap1/Tarski/>

Chapitre 2

Ensemble, relation et fonction

Les mathématiques discrètes comprennent un éventail assez large d'objets mathématiques. Dans le cadre de ce cours, nous nous limiterons aux fondements des mathématiques discrètes, soit la théorie des ensembles, les relations, les fonctions et les suites. Les structures de données et les bases de données relationnelles sont inspirées des mathématiques discrètes. Peter Codd¹ se mérita le prix Alan Turing en 1981 pour sa contribution à la définition des bases de données relationnelles fondées sur les relations (i.e., l'algèbre des relations). Les ensembles, les relations et les fonctions constituent aussi les fondements des langages de spécification formel de systèmes comme ASM, Alloy, B, TLA, VDM et Z. Ces langages sont utilisés pour concevoir des systèmes critiques et prouver leur correction et leur sûreté de fonctionnement.

Nous utilisons le langage de spécification B [1] pour illustrer les mathématiques discrètes, ainsi que l'outil ProB [8, 9], conçu par Michael Leuschel. Consultez le fichier

<http://info.usherbrooke.ca/mfrappier/mat115/ref/resume-ens-rel-fonction-abrial.pdf>

pour des exemples des opérateurs du langage B. Ce document a été produit par Jean-Raymond Abrial², auteur de la méthode B.

¹Edgar Frank « Ted » Codd (23 août 1923 - 18 avril 2003) est un informaticien britannique. Il est considéré comme l'inventeur du modèle relationnel en base de données durant son séjour chez IBM dans les années 1960. Il reçut le prix Alan Turing en 1981 pour cette contribution. Il participa à la définition de plusieurs formes normales, qui sont utilisées comme critère de qualité et de conception pour un modèle relationnel de données. IBM ne commercialisa pas ses travaux immédiatement, préférant continuer d'exploiter son SGBD hiérarchique IMS, ce qui permit à Larry Ellison de fonder la société Oracle en utilisant le résultat des travaux de Codd. Oracle connut un succès commercial important grâce à son SGBD relationnel, qui supplanta les SGBD hiérarchiques comme IMS d'IBM.

²Jean-Raymond Abrial (1938) est un informaticien français, docteur *honoris causa* de l'Université de Sherbrooke, diplômé de l'École Polytechnique en France et de l'Université de Stanford aux États-Unis. Il a débuté sa carrière d'informaticien dans les années 60, en concevant l'un des tout premiers systèmes de gestion de bases de données de type réseau, permettant aux développeurs d'applications de gestion de l'époque de faire abstraction de la complexité de la représentation interne des données. Il participa aussi à la conception du langage de programmation Ada en 1978-79 au sein de l'équipe CII-Honeywell-Bull qui remporta le concours international organisé par le Ministère de la défense américain. Il y fut responsable des aspects concurrentiels et notamment de la notion de rendez-vous, qui est à la base de la programmation concurrente. Dès le début des années 70, il a amorcé ses travaux sur les modèles sémantiques de données et sur la spécification formelle des systèmes, avec la méthode Z, la méthode B et finalement Event-B, qui constituent sans aucun doute une des plus grandes contributions en génie logiciel. Il a réalisé le défi colossal de mener à terme une idée, de son concept théorique initial jusqu'à son application industrielle à grande échelle, une performance encore plus remarquable qu'il fut le premier à le faire au niveau des méthodes formelles, plusieurs grands chercheurs s'y étant attaqué sans succès auparavant.

2.1 Conventions

Nous utilisons les conventions suivantes

Symbole	Description
X	variable propositionnelle
$\mathcal{A}, \mathcal{B}, \mathcal{C}$	formule
t, u, v, w	terme
x	variable (terme)
\vec{x}	x_1, \dots, x_n (liste de variables)
S, T, U	ensemble (terme)
f, g, h	fonction (terme)
r	relation (terme)
i, j, k, m, n	nombre (terme)
s	suite

2.2 La logique du premier ordre en B

Le langage B a une syntaxe légèrement différente de la syntaxe traditionnelle. Voici la syntaxe des formules en B.

Description	Expression	Syntaxe ASCII B	Notes
négation	$\neg \mathcal{A}$	not (\mathcal{A})	
conjonction	$\mathcal{A} \wedge \mathcal{B}$	$\mathcal{A} \ \& \ \mathcal{B}$	
disjonction	$\mathcal{A} \vee \mathcal{B}$	$\mathcal{A} \ \text{or} \ \mathcal{B}$	
implication	$\mathcal{A} \Rightarrow \mathcal{B}$	$\mathcal{A} \ \Rightarrow \ \mathcal{B}$	
équivalence	$\mathcal{A} \Leftrightarrow \mathcal{B}$	$\mathcal{A} \ \Leftrightarrow \ \mathcal{B}$	
pour tout	$\forall(\vec{x}) \cdot (\mathcal{A} \Rightarrow \mathcal{C})$	$!(\vec{x}) \cdot (\mathcal{A} \Rightarrow \mathcal{B})$	\mathcal{A} doit typer chaque x_i
il existe	$\exists(\vec{x}) \cdot (\mathcal{A} \wedge \mathcal{B})$	$\#(\vec{x}) \cdot (\mathcal{A} \wedge \mathcal{B})$	\mathcal{A} doit typer chaque x_i
égalité	$t_1 = t_2$	$t_1 = t_2$	
inégalité	$t_1 \neq t_2$	$t_1 \ / = \ t_2$	

Table 2.1: Syntaxe des formules de logique du premier ordre en B

2.3 Les ensembles

Un *ensemble* est une collection d'objets, sans ordre particulier. Les ensembles servent, entre autres, à donner un *type* aux variables d'une formule de logique du premier ordre. Il existe plusieurs méthodes pour définir un ensemble; nous en utiliserons deux, soit la définition par *extension* (aussi appelée par *énumération*) et la définition par *compréhension*.

2.3.1 Définition par extension

La définition par extension consiste à énumérer les éléments de l'ensemble:

$$\{e_1, \dots, e_n\}$$

où e_1, \dots, e_n représentent les éléments de l'ensemble. Par exemple, voici l'ensemble `CouleurPrimaire`, qui contient 3 couleurs.

$$\text{CouleurPrimaire} = \{\text{rouge}, \text{vert}, \text{bleu}\}$$

Vu que l'ordre d'énumération des éléments n'a pas d'importance, alors l'ensemble

$$\text{CouleurPrimaire2} = \{\text{vert}, \text{rouge}, \text{bleu}\}$$

est égal à l'ensemble `CouleurPrimaire`, i.e.,

$$\text{CouleurPrimaire2} = \text{CouleurPrimaire}$$

Un ensemble ne peut pas contenir de répétitions pour un élément; par exemple, on a

$$\{a, a, b\} = \{a, b\}$$

Un cas particulier d'ensemble est l'ensemble *vide*, noté $\{\}$, ou bien \emptyset ; il ne contient aucun élément. Si un ensemble comprend un grand nombre d'éléments, ou bien une infinité d'éléments, on utilise parfois les "...":

$$\mathbb{N} = \{0, 1, 2, \dots\}$$

$$\text{MultipleTrois} = \{0, 3, 6, 9, \dots\}$$

Toutefois, cela n'est pas formel. Pour obtenir une définition formelle, on utilise alors la deuxième méthode, c'est-à-dire par compréhension.

2.3.2 Définition par compréhension

Soit x une variable et \mathcal{A} une formule de logique du premier ordre; la forme générale d'une définition d'un ensemble par compréhension est la suivante:

$$\{x \mid \mathcal{A}\}$$

La variable x représente un élément quelconque de l'ensemble et la formule \mathcal{A} représente le critère d'appartenance d'un élément x à l'ensemble. La barre verticale " \mid " sert simplement de séparateur dans la définition. On peut choisir la variable que l'on veut pour définir un ensemble par compréhension. Cette variable devient liée par cette définition, un peu comme la variable x est liée dans une formule $\forall x \cdot \mathcal{A}$. Par exemple, l'ensemble suivant contient tous les multiples de 3:

$$\text{MultipleTrois} = \{x \mid x \in \mathbb{N} \wedge \exists y \cdot y \in \mathbb{N} \wedge x = y * 3\}$$

On peut lire cette définition comme suit.

Un multiple de 3, représenté ici par la variable x , est un nombre naturel ($x \in \mathbb{N}$) tel qu'il existe un nombre naturel y tel que $x = y * 3$.

On suppose ici que la définition des nombres naturels \mathbb{N} est déjà donnée. La définition de \mathbb{N} n'est pas si triviale à formuler. Elle n'utilise pas la notion de compréhension; elle est basée sur les axiomes de Giuseppe Peano³, proposés en 1889! Un axiome est une formule que l'on suppose vraie, et qui sert à définir une structure mathématique. Dans ce cours, nous n'aborderons pas la définition de \mathbb{N} ;

³Giuseppe Peano, (1858–1932) mathématicien et linguiste italien, pionnier de l'approche formaliste des mathématiques, il développa, parallèlement à l'Allemand Richard Dedekind, une axiomatisation de l'arithmétique (1889), incluant le concept d'induction mathématique

nous supposons qu'elle existe. La formulation de Zermelo⁴ est particulièrement simple et élégante: il s'agit d'un emboîtement d'ensemble vide, comme suit:

1. $0 = \{\}$
2. $1 = \{\{\}\}$
3. $2 = \{\{\{\}\}\}$
4. ...

La fonction successeur $s(x) = x + 1$, est représentée comme suit:

$$s(x) = \{x\}$$

À partir de successeur, on peut définir l'addition, la multiplication, etc.

2.3.3 Appartenance

L'appartenance d'un élément e à un ensemble E est notée

$$e \in E$$

Par exemple, on a

$$\text{rouge} \in \text{CouleurPrimaire}$$

On note que e est un élément, et E est un ensemble. L'appartenance à un ensemble défini par extension est définie comme suit:

$$y \in \{e_1, \dots, e_n\} \Leftrightarrow y = e_1 \vee \dots \vee y = e_n \quad (2.1)$$

L'appartenance d'un élément à un ensemble défini par compréhension est définie comme suit.

$$y \in \{x \mid \mathcal{A}\} \Leftrightarrow \mathcal{A}[x := y] \quad (2.2)$$

Pour vérifier qu'un nombre n appartient à `MultipleTrois`, il suffit de vérifier si n satisfait la formule définissant `MultipleTrois`, c'est-à-dire vérifier $\mathcal{A}[x := n]$. Voici le calcul pour $n = 6$.

$$\begin{aligned}
 & 6 \in \text{MultipleTrois} \\
 \Leftrightarrow & \hspace{20em} \langle (2.2) \rangle \\
 & (x \in \mathbb{N} \wedge \exists y \cdot y \in \mathbb{N} \wedge x = y * 3)[x := 6] \\
 \Leftrightarrow & \hspace{10em} \langle \text{application de la substitution} \rangle \\
 & 6 \in \mathbb{N} \wedge \exists y \cdot y \in \mathbb{N} \wedge 6 = y * 3 \\
 \Leftarrow & \hspace{15em} \langle \text{règle } I_{\exists} \text{ avec } y := 2 \rangle \\
 & 6 \in \mathbb{N} \wedge ((y \in \mathbb{N} \wedge 6 = y * 3)[y := 2]) \\
 \Leftrightarrow & \hspace{10em} \langle \text{application de la substitution} \rangle \\
 & 6 \in \mathbb{N} \wedge 2 \in \mathbb{N} \wedge 6 = 2 * 3 \\
 \Leftarrow & \hspace{15em} \langle \text{arithmétique} \rangle \\
 & \text{Lois de l'arithmétique}
 \end{aligned}$$

⁴Ernst Zermelo, (1871–1953) mathématicien allemand, qui s'est intéressé aux fondations de la théorie des ensembles et fut un des précurseurs de la théorie des jeux, maintenant largement utilisée en finance, en économie et en intelligence artificielle.

Une définition par compréhension peut mener à une contradiction. Russell⁵ l'a illustré par la définition suivante:

$$y = \{x \mid x \notin x\}$$

On peut alors effectuer les déductions suivantes.

$$\begin{aligned} & y \in y \\ \Leftrightarrow & & & \langle (2.2) \rangle \\ & (x \notin x)[x := y] \\ \Leftrightarrow & & & \langle \text{application de la substitution} \rangle \\ & y \notin y \end{aligned}$$

On obtient donc $y \in y \Leftrightarrow y \notin y$, ce qui est une contradiction. De la même manière, "l'ensemble de tous les ensembles" n'existe pas, car si on essaie de le définir, on obtient aussi une contradiction.

Pour éviter ces contradictions, on s'assure que chaque ensemble est défini à partir d'un ensemble S qui est lui-même bien défini, comme par exemple \mathbb{N} , ou bien un ensemble défini par énumération. Pour éviter ces paradoxes, on utilise la forme générale suivante, où S est un ensemble bien défini.

$$E = \{x \mid x \in S \wedge \mathcal{A}\}$$

De cette façon, l'ensemble E est un sous-ensemble de l'ensemble bien défini S . Ce type de définition ne peut mener à une contradiction comme celle illustrée par le paradoxe de Russell. Les mécanismes utilisés pour produire des ensembles bien définis sont hors de la portée de ce cours; il s'agit, entre autres, de la théorie des types.

2.3.4 Inclusion

On dit que qu'un ensemble E_1 est un sous-ensemble de E_2 , noté $E_1 \subseteq E_2$, ssi tous les éléments de E_1 sont aussi des éléments de E_2 . Voici la définition formelle du prédicat \subseteq .

$$E_1 \subseteq E_2 \Leftrightarrow \forall x \cdot x \in E_1 \Rightarrow x \in E_2 \quad (2.3)$$

Finalement, l'ensemble vide ne contient aucun élément, et il est donc inclus dans tous les ensembles. Soit S un ensemble.

$$x \in \{\} \Leftrightarrow \text{faux} \quad (2.4)$$

Soit S un ensemble quelconque, on a donc, par (2.3) et (2.4):

$$\{\} \subseteq S \quad (2.5)$$

Il arrive fréquemment que les étudiants confondent l'appartenance et l'inclusion. Dans une appartenance $x \in y$, x est un **élément** et y un ensemble; dans une inclusion $x \subseteq y$, x et y sont des ensembles. Le tableau 2.2 illustre la distinction entre les deux.

⁵Bertrand Russell (1872–1970) est un mathématicien, logicien, philosophe, épistémologue, homme politique et moraliste britannique. Il est considéré, avec Frege, comme l'un des fondateurs de la logique contemporaine (avec son livre *Principia Mathematica*). Il s'engage dans de nombreuses polémiques : il défend des idées proches du socialisme de tendance libertaire et milite également contre toutes les formes de religion, considérant qu'elles sont des systèmes de cruauté inspirés par la peur et l'ignorance. Il organise le tribunal Sartre-Russell contre les crimes commis pendant la guerre du Viêt Nam. Son œuvre, qui comprend également des romans et des nouvelles, est couronnée par le prix Nobel de littérature en 1950.

Formule	Valeur	Commentaire
$a \in \{a, b\}$	vrai	
$\{a\} \in \{a, b\}$	faux	Formule mal typée; $\{a\}$ n'est pas un élément
$\{a\} \subseteq \{a, b\}$	vrai	
$a \subseteq \{a, b\}$	faux	Formule mal typée; a n'est pas un ensemble
$\{\} \in \{a, b\}$	faux	Formule mal typée; $\{\}$ n'est pas un élément
$\{\} \subseteq \{a, b\}$	vrai	

Table 2.2: Distinction entre appartenance et inclusion

2.3.5 Les ensembles en B

Le tableau 2.3 donne la syntaxe des principaux ensembles de base et des constructeurs d'ensemble du langage B.

Description	Expression	Syntaxe ASCII B	Notes
ensemble vide	$\{\}$	<code>{}</code>	$x \in \{\} \Leftrightarrow \text{faux}$
extension (énumération)	$\{t_1, \dots, t_n\}$	<code>{ t₁, ..., t_n }</code>	$x \in \{t_1, \dots, t_n\} \Leftrightarrow x = t_1 \vee \dots \vee x = t_n$
compréhension	$\{x \mid \mathcal{A}\}$	<code>{ x A }</code>	$y \in \{x \mid \mathcal{A}\} \Leftrightarrow \mathcal{A}[x := y]$ \mathcal{A} doit donner un type à x
naturels	\mathbb{N}	<code>NATURAL</code>	$\{0, 1, 2, \dots\}$
naturels non nuls	\mathbb{N}_1	<code>NATURAL1</code>	$\{1, 2, \dots\}$
entiers	\mathbb{Z}	<code>INTEGER</code>	$\{\dots, -2, -1, 0, 1, 2, \dots\}$
intervalle d'entiers	$i..j$	<code>i..j</code>	$\{x \mid x \in \mathbb{Z} \wedge i \leq x \wedge x \leq j\}$, où $i \in \mathbb{Z}$ et $j \in \mathbb{Z}$
plus petit entier implémentable	<code>MININT</code>	<code>MININT</code>	valeur dépend du processeur
plus grand entier implémentable	<code>MAXINT</code>	<code>MAXINT</code>	valeur dépend du processeur
naturels implémentables	<code>NAT</code>	<code>NAT</code>	<code>0..MAXINT</code>
nat. impl. non nuls	<code>NAT1</code>	<code>NAT1</code>	<code>1..MAXINT</code>
entiers implémentables	<code>INT</code>	<code>INT</code>	<code>MININT..MAXINT</code>
chaîne de caractères	<code>STRING</code>	<code>STRING</code>	
booléens	<code>BOOL</code>	<code>BOOL</code>	<code>{TRUE, FALSE}</code>
bool	<code>bool(A)</code>	<code>bool(A)</code>	retourne le booléen de \mathcal{A} i.e., sa valeur de vérité

Table 2.3: Constructeurs d'ensemble en B

Le tableau 2.4 résume les prédicats sur les ensembles en B.

Description	Expression	Syntaxe ASCII B	Définition
appartenance	$x \in S$	$x : S$	x est un élément de S
négation appartenance	$x \notin S$	$x /: S$	$\neg(x \in S)$
inclusion	$S \subseteq T$	$S <: T$	$\forall x \cdot x \in S \Rightarrow x \in T$
négation inclusion	$S \not\subseteq T$	$S /<: T$	$\neg(S \subseteq T)$
inclusion stricte	$S \subset T$	$S <<: T$	$S \subseteq T \wedge S \neq T$
négation inclusion stricte	$S \not\subset T$	$S /<<: T$	$\neg(S \subset T)$
fini	$\text{finite}(S)$	N/A	S est fini

Table 2.4: Prédicat sur les ensembles en B

2.3.6 Opérations sur les ensembles en B

Le tableau 2.5 donne les opérations sur les ensembles dans le langage B. Ces opérations sont illustrées dans le document suivant:

<http://info.usherbrooke.ca/mfrappier/mat115/ref/resume-ens-rel-fonction-abrial.pdf>

Description	Expression	Syntaxe ASCII B	Définition
union	$S \cup T$	$S \vee T$	$\{x \mid x \in S \vee x \in T\}$
intersection	$S \cap T$	$S \wedge T$	$\{x \mid x \in S \wedge x \in T\}$
différence	$S - T$	$S - T$	$\{x \mid x \in S \wedge x \notin T\}$
ens. des parties (ens. des sous-ens.) (ens. de puissance)	$\mathbb{P}(S)$	$\text{POW}(S)$	$\{T \mid T \subseteq S\}$
ens. des parties non vides	$\mathbb{P}_1(S)$	$\text{POW}_1(S)$	$\mathbb{P}(S) - \{\{\}\}$
ens. des parties finies	$\mathbb{F}(S)$	$\text{FIN}(S)$	$\{T \mid T \subseteq S \wedge \text{finite}(T)\}$
ens. des parties finies non vides	$\mathbb{F}_1(S)$	$\text{FIN}_1(S)$	$\mathbb{F}(S) - \{\{\}\}$
union généralisée	$\text{union}(S)$	$\text{union}(S)$	$\{x \mid \exists T \cdot T \in S \wedge x \in T\}$
intersection généralisée	$\text{inter}(S)$	$\text{inter}(S)$	$\{x \mid \forall T \cdot T \in S \Rightarrow x \in T\}$
union quantifiée	$\bigcup(\vec{x}).(\mathcal{A} \mid S)$	$\text{UNION}(\vec{x}).(\mathcal{A} \mid S)$	$\{y \mid \exists(\vec{x}) \cdot (\mathcal{A} \wedge y \in S)\}$
intersection quantifiée	$\bigcap(\vec{x}).(\mathcal{A} \mid S)$	$\text{INTER}(\vec{x}).(\mathcal{A} \mid S)$	$\{y \mid \forall(\vec{x}) \cdot (\mathcal{A} \Rightarrow y \in S)\}$
cardinalité	$\text{card}(S)$	$\text{card}(S)$	nb. d'éléments de S (si $\text{finite}(S)$)

Table 2.5: Opérations sur les ensembles en B

Il arrive fréquemment que les étudiants confondent le typage des résultats de l'opérateur $\mathbb{P}(S)$, qui retourne l'ensemble des sous-ensembles de S . Le tableau 2.6 illustre la distinction entre les deux. Notons la valeur de l'expression suivante:

$$\mathbb{P}(\{a, b\}) = \{ \{\}, \{a\}, \{b\}, \{a, b\} \} \quad (2.6)$$

Formule	Valeur	Commentaire
$a \in \mathbb{P}(\{a, b\})$	faux	a n'est pas du bon type
$a \subseteq \mathbb{P}(\{a, b\})$	faux	a n'est pas du bon type
$\{a\} \in \mathbb{P}(\{a, b\})$	vrai	
$\{a\} \subseteq \mathbb{P}(\{a, b\})$	faux	$\{a\}$ n'est pas du bon type
$\{\{a\}\} \subseteq \mathbb{P}(\{a, b\})$	vrai	
$\{\} \in \mathbb{P}(\{a, b\})$	vrai	$\{\}$ est bien un élément; voir 2.6
$\{\} \subseteq \mathbb{P}(\{a, b\})$	vrai	$\{\}$ est un sous-ensemble de n'importe quel autre ensemble

Table 2.6: Distinction entre appartenance et inclusion

2.4 Les relations

Soit S et T deux ensembles; soit x un élément de S et y un élément de T . On note par $x \mapsto y$ le *couple* formé des éléments x et y ; on appelle x un antécédent et y une image. Plusieurs auteurs utilisent aussi la notation (x, y) pour dénoter un couple. Le produit cartésien de S par T , noté $S \times T$, est l'ensemble de tous les couples formés à partir des éléments de S et de T . Voici la définition formelle.

$$S \times T = \{x \mapsto y \mid x \in S \wedge y \in T\} \quad (2.7)$$

Notons que nous utilisons ici une version “abrégée” de la définition par compréhension; en effet, selon la syntaxe introduite à la section précédente, on devrait retrouver une seule variable à la gauche du séparateur “|”, alors qu'on retrouve ici un couple de variable $x \mapsto y$. La version non-abrégée, qui est beaucoup moins lisible, est la suivante:

$$S \times T = \{z \mid \exists x, y. z = x \mapsto y \wedge x \in S \wedge y \in T\}$$

Une *relation* r de S vers T est un sous-ensemble de $S \times T$. On a les équivalences suivantes:

$$\begin{aligned} r &\subseteq S \times T \\ \Leftrightarrow \\ r &\in \mathbb{P}(S \times T) \\ \Leftrightarrow \\ r &\in S \leftrightarrow T \end{aligned}$$

En B, une relation définie par compréhension se note comme suit:

$$\{x, y \mid \mathcal{A}\}$$

Malheureusement, on ne peut pas utiliser les formes usuelles $\{(x, y) \mid \mathcal{A}\}$ et $\{x \mapsto y \mid \mathcal{A}\}$.

Description	Expression	Syntaxe ASCII B	Définition
couple (élément d'une relation)	$x \mapsto y$	$x \mid \rightarrow y$	parfois aussi noté (x, y)
couple (notation alternative)	(x, y)	(x, y)	$(x, y) = x \mapsto y$
n -uplet	(x_1, \dots, x_n)	(x_1, \dots, x_n)	$(x_1, \dots, x_n) = ((x_1 \mapsto x_2) \mapsto \dots x_n)$
relation par compréhension	$\{x, y \mid \mathcal{A}\}$	$\{ x, y \mid \mathcal{A} \}$	$z_1 \mapsto z_2 \in \{x, y \mid \mathcal{A}\} \Leftrightarrow \mathcal{A}[x, y := z_1, z_2]$
produit cartésien	$S \times T$	$S * T$	$\{x \mapsto y \mid x \in S \wedge y \in T\}$
ensemble de relations	$S \leftrightarrow T$	$S \leftrightarrow T$	$\mathbb{P}(S \times T)$
identité	$\text{id}(S)$	$\text{id}(S)$	$\{x \mapsto y \mid x \in S \wedge x = y\}$
domaine d'une relation	$\text{dom}(r)$	$\text{dom}(r)$	$\{x \mid \exists y \cdot x \mapsto y \in r\}$
codomaine d'une relation	$\text{ran}(r)$	$\text{ran}(r)$	$\{y \mid \exists x \cdot x \mapsto y \in r\}$
composition (produit)	$(r_1 ; r_2)$	$(r_1 ; r_2)$	$\{x \mapsto y \mid \exists z \cdot x \mapsto z \in r_1 \wedge z \mapsto y \in r_2\}$
attention: il faut toujours entourer une composition avec des parenthèses			
produit direct	$r_1 \otimes r_2$	$r_1 \times r_2$	$\{x \mapsto (y \mapsto z) \mid x \mapsto y \in r_1 \wedge x \mapsto z \in r_2\}$
restriction du domaine	$S \triangleleft r$	$S \triangleleft r$	$\{x \mapsto y \mid x \in S \wedge x \mapsto y \in r\}$
restriction du codomaine	$r \triangleright S$	$r \triangleright S$	$\{x \mapsto y \mid y \in S \wedge x \mapsto y \in r\}$
antirestriction du domaine	$S \triangleleft r$	$S \triangleleft r$	$\{x \mapsto y \mid x \notin S \wedge x \mapsto y \in r\}$
antirestriction du codomaine	$r \triangleright S$	$r \triangleright S$	$\{x \mapsto y \mid y \notin S \wedge x \mapsto y \in r\}$
surcharge	$r_1 \triangleleft r_2$	$r_1 \triangleleft r_2$	$(\text{dom}(r_2) \triangleleft r_1) \cup r_2$
inverse	r^{-1}	$r \sim$	$\{x \mapsto y \mid y \mapsto x \in r\}$
image	$r[S]$	$r[S]$	$\text{ran}(S \triangleleft r)$
itération	r^n	$\text{iterate}(r, n)$	$r^0 = \text{id}(S), r^n = r ; r^{n-1}$, où $r \in S \leftrightarrow S$ si S n'est pas donné, par convention, on utilise $S = \text{dom}(r) \cup \text{ran}(r)$
fermeture réflexive et transitive	r^*	$\text{closure}(r)$	$\bigcup (n). (n \geq 0 \mid r^n)$
fermeture transitive	r^+	$\text{closure1}(r)$	$\bigcup (n). (n \geq 1 \mid r^n)$

Table 2.7: Opérations sur les relations

La figure 2.1 illustre la composition $(r_1 ; r_2)$ de deux relations $r_1 \in A \leftrightarrow B$ et $r_2 \in B \leftrightarrow C$. Le résultat est une relation de $A \leftrightarrow C$. La composition relationnelle est similaire à l'opérateur de jointure entre deux tables en SQL. On a

$$r_1 ; r_2 = \text{SELECT } A, C \text{ FROM } r_1 \text{ JOIN } r_2 \text{ ON } B$$

La figure 2.2 représente une relation $r \in S \leftrightarrow S$, où

$$S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$$

La relation r a la valeur suivante :

$$r = \{(s_1, s_2), (s_2, s_3), (s_3, s_4), (s_5, s_6), (s_6, s_7), (s_7, s_8), (s_8, s_5)\}$$

Lorsqu'une relation est définie sur un espace *homogène* (i.e., le même ensemble est utilisé pour le domaine et le codomaine), on peut représenter une relation par un graphe sur S . Un couple (s_1, s_2) est représenté par une flèche de s_1 vers s_2 dans le graphe. La relation r^2 , notée r^2 dans

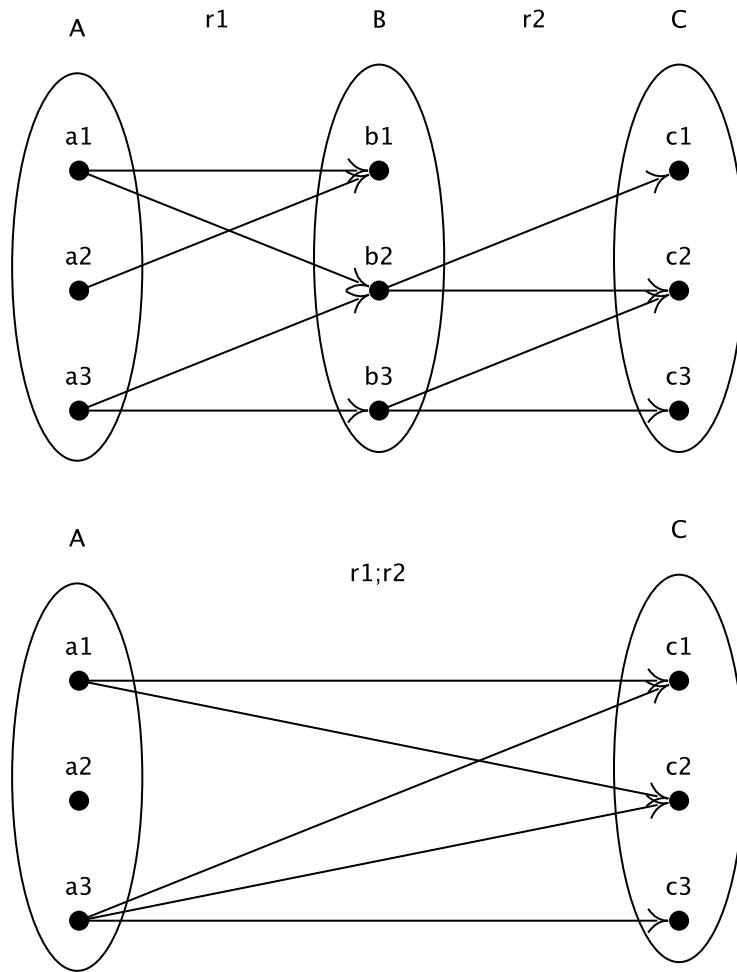


Figure 2.1: La composition de deux relations

la figure 2.2, est définie par $(r ; r)$; elle représente les chemins de longueur 2 dans le graphe de la relation r , en donnant le point de départ (la source) et le point d'arrivée (la destination). Il est donc assez facile de visualiser le résultat d'une composition relationnelle. De manière générale, la relation r^n représente les chemins de longueur n dans le graphe de r . La fermeture transitive de r , noté $r^+ = \bigcup_{n \geq 1} r^n$ est illustrée dans la figure 2.3. Elle représente tous les chemins de longueur supérieure ou égale à 1 dans r . On trouve aussi dans cette figure la relation identité $\text{id}(S)$ et la fermeture réflexive et transitive $r^* = r^+ \cup \text{id}(S)$. La relation identité représente les chemins de longueur 0 dans le graphe de r , c'est-à-dire que l'on ne se déplace pas dans le graphe, et ainsi la source est égale à la destination.

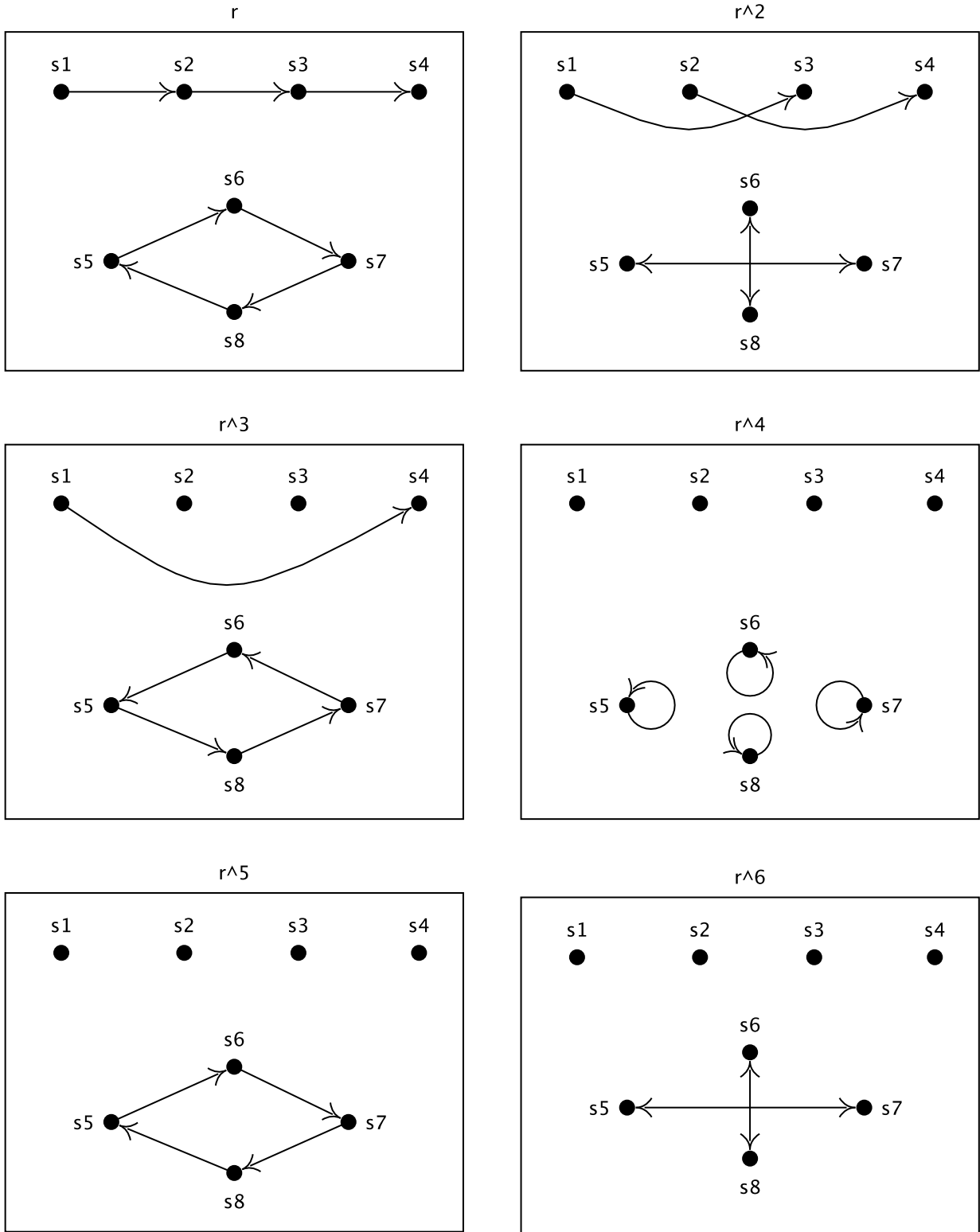


Figure 2.2: Le graphe d'une relation homogène r , et les relations r^2, r^3, r^4, r^5, r^6

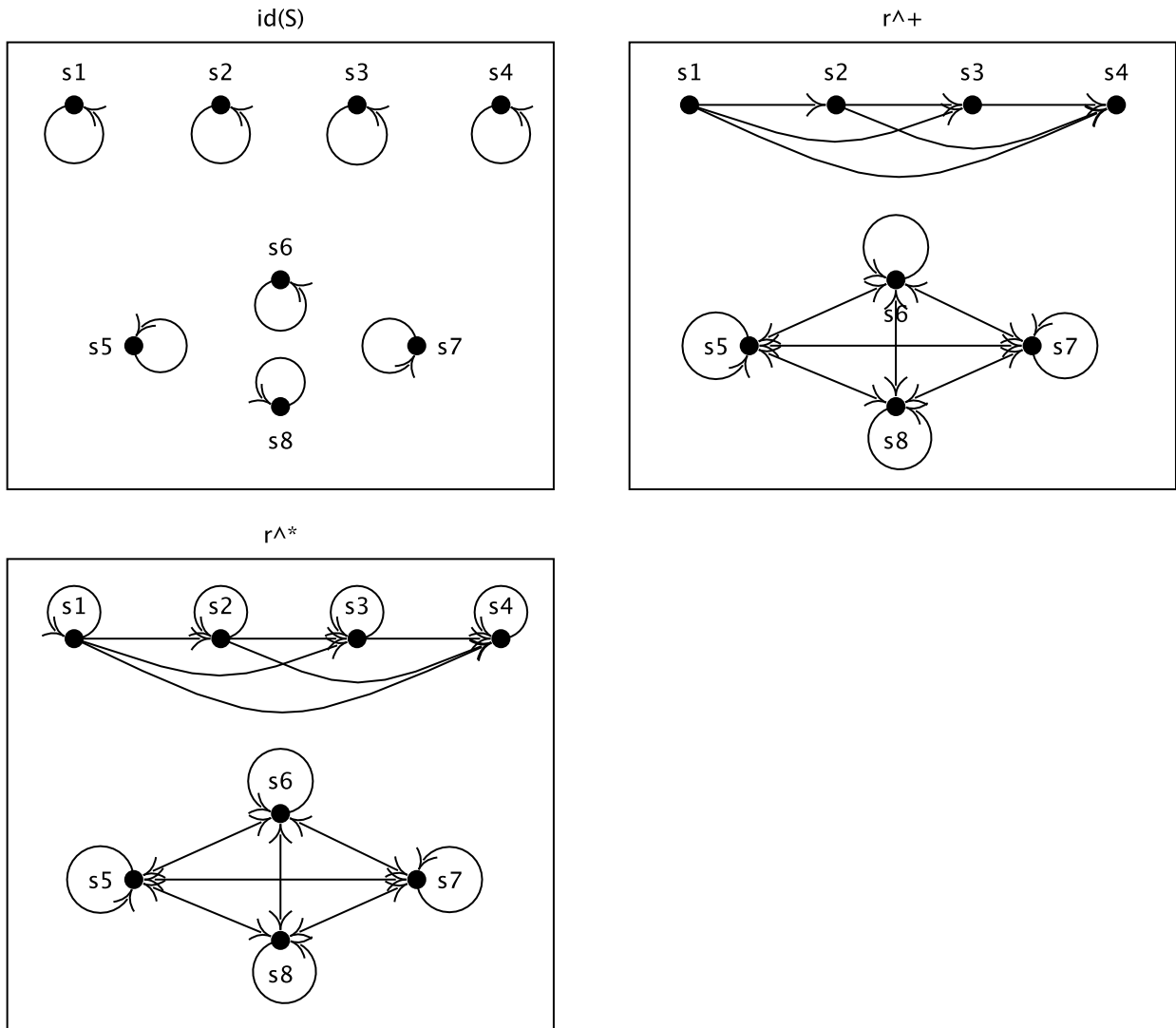


Figure 2.3: Les relations $id(S)$, r^+ et r^* calculées à partir de S et de la relation r de la figure 2.2

2.5 Les fonctions

Une fonction est un cas particulier de relation. Dans cette section, nous introduisons huit classes de fonction, la plus générale étant la classe des fonctions. La figure 2.4 illustre les classes de fonctions. Le tableau 2.8 résume les classes de fonctions.

Description	Expression	Syntaxe ASCII B	Définition
fonctions	$S \leftrightarrow T$	$S \leftrightarrow T$	$\{f \mid f \in S \leftrightarrow T \wedge f^{-1} ; f \subseteq \text{id}(T)\}$
fonctions totales	$S \rightarrow T$	$S \rightarrow T$	$\{f \mid f \in S \leftrightarrow T \wedge \text{dom}(f) = S\}$
injections	$S \mapsto T$	$S \mapsto T$	$\{f \mid f \in S \leftrightarrow T \wedge f^{-1} \in T \mapsto S\}$
injections totales	$S \mapsto T$	$S \mapsto T$	$\{f \mid f \in S \mapsto T \wedge \text{dom}(f) = S\}$
surjections	$S \twoheadrightarrow T$	$S \twoheadrightarrow T$	$\{f \mid f \in S \leftrightarrow T \wedge \text{ran}(f) = T\}$
surjections totales	$S \twoheadrightarrow T$	$S \twoheadrightarrow T$	$\{f \mid f \in S \twoheadrightarrow T \wedge \text{dom}(f) = S\}$
bijections	$S \leftrightarrow T$	$S \leftrightarrow T$	$\{f \mid f \in S \leftrightarrow T \wedge f \in S \leftrightarrow T\}$
bijections totales	$S \leftrightarrow T$	$S \leftrightarrow T$	$\{f \mid f \in S \leftrightarrow T \wedge f \in S \leftrightarrow T\}$
lambda expression	$\lambda x.(\mathcal{A} \mid t)$	$\%x.(\mathcal{A} \mid t)$	$\{x \mapsto t \mid \mathcal{A}\}$

Table 2.8: Classes et constructeur de fonctions

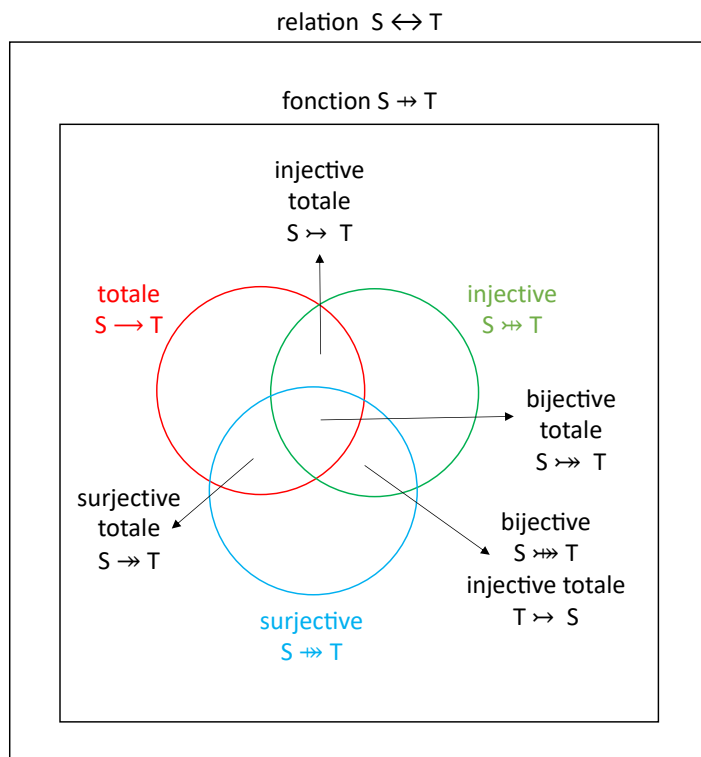


Figure 2.4: Les classes de fonctions

2.5.1 Fonction

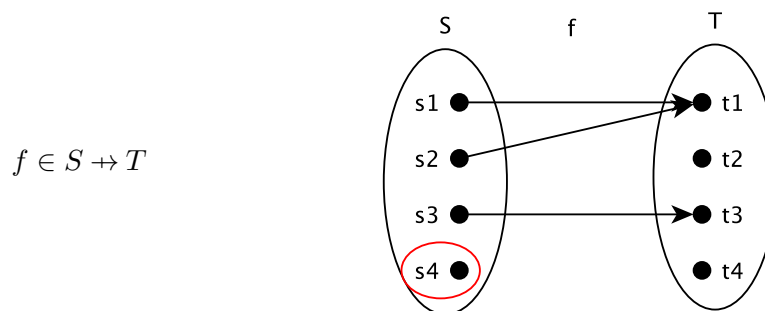
Une *fonction* f d'un ensemble S vers un ensemble T , notée $f \in S \rightarrow T$, est une relation telle que chaque élément de $\text{dom}(f)$ est associé à exactement un élément de T . De manière équivalente, on peut aussi dire que chaque élément de S est associé via f à au plus un élément de T (i.e., les éléments de S qui ne sont pas dans le domaine de f ne sont pas associés à un élément de T , bien entendu). Ce qui donne les définitions suivantes, qui sont toutes équivalentes.

$$\begin{aligned}
 & f \in S \rightarrow T \\
 \Leftrightarrow & \\
 & f \in S \leftrightarrow T \\
 & \wedge \forall x, y, z \cdot x \mapsto y \in f \wedge x \mapsto z \in f \Rightarrow y = z \\
 \Leftrightarrow & \\
 & f \in S \leftrightarrow T \\
 & \wedge f^{-1} ; f \subseteq \text{id}(T)
 \end{aligned}$$

On a les propriétés suivantes.

$$\begin{aligned}
 & f \in S \rightarrow T \\
 \Rightarrow & \\
 & \forall x \cdot x \in \text{dom}(f) \Rightarrow \text{card}(f[\{x\}]) = 1 \\
 & \wedge \\
 & \forall x \cdot x \in S \Rightarrow \text{card}(f[\{x\}]) \leq 1 \\
 & \wedge \\
 & \forall x \cdot x \in S \Rightarrow \text{card}(f[\{x\}]) \in 0..1 \qquad \text{** lien avec diag. UML **} \\
 & \wedge \\
 & \forall y \cdot y \in T \Rightarrow \text{card}(f^{-1}[\{y\}]) \in 0..\text{card}(S) \qquad \text{** lien avec diag. UML **}
 \end{aligned}$$

Ce qui donne ceci comme exemple typique, avec la partie en rouge qui montre la caractéristique distinctive d'une fonction par rapport aux autres classes de fonctions, c'est-à-dire qu'un élément de S peut ne pas être relié à un élément de T par la fonction f .



Les fonctions sont très utiles dans la modélisation des systèmes. Par exemple, les attributs *scalaires* (i.e., ayant une seule valeur) d'une entité dans un modèle de données sont représentés par des fonctions. Voici quelques exemples pour un livre et un membre.

$$\begin{aligned}
 \textit{titre} & \in \textit{Livre} \rightarrow \textit{String} \\
 \textit{emprunteur} & \in \textit{Livre} \rightarrow \textit{Membre} \\
 \textit{dateDePublication} & \in \textit{Livre} \rightarrow \textit{Année} \\
 \textit{nom} & \in \textit{Membre} \rightarrow \textit{String}
 \end{aligned}$$

Certains attributs d'un livre ne peuvent être représentés par une fonction, car ils ont plusieurs valeurs. Par exemple, un livre ayant potentiellement plusieurs auteurs, l'attribut *auteurs* est représenté par une relation.

$$auteurs \in Livre \leftrightarrow String$$

Ainsi, si le livre b_1 représente le livre [1], on a les valeurs suivantes:

$$\begin{aligned} titre(b_1) &= \text{“The B-book: Assigning Programs to Meanings”} \\ emprunteur(b_1) &= m \\ dateDePublication(b_1) &= 1996 \\ nom(m) &= \text{“Marc Frappier”} \end{aligned}$$

Vu que *auteurs* est une relation, on peut utiliser les expressions suivantes, qui sont équivalentes, pour représenter certaines de ses valeurs:

$$\begin{aligned} auteurs[\{b_1\}] &= \{\text{“Jean-Raymond Abrial”}\} \\ b_1 \mapsto \text{“Jean-Raymond Abrial”} &\in auteurs \end{aligned}$$

Si le livre $b_2 = [5]$, alors on a plusieurs auteurs. On a donc les expressions suivantes, qui sont équivalentes:

$$\begin{aligned} auteurs[\{b_2\}] &= \{\text{“David Gries”, “Fred B. Schneider”}\} \\ \{b_2 \mapsto \text{“David Gries”, } b_2 \mapsto \text{“Fred B. Schneider”}\} &\subseteq auteurs \end{aligned}$$

2.5.2 Fonction totale

Une fonction totale (aussi appelée *application* en mathématiques) f d'un ensemble S vers un ensemble T , notée $f \in S \rightarrow T$, est une relation telle que chaque élément de S est associé à exactement un élément de T . Ce qui donne les définitions alternatives suivantes, qui sont toutes équivalentes.

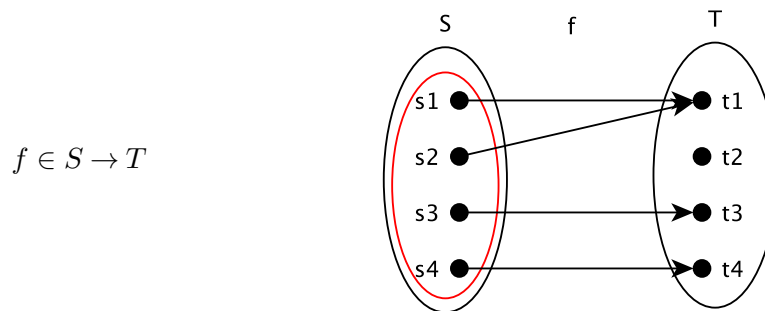
$$\begin{aligned}
 & f \in S \rightarrow T \\
 \Leftrightarrow & \\
 & f \in S \rightarrow T \\
 & \wedge \text{dom}(f) = S \\
 \Leftrightarrow & \\
 & f^{-1} ; f \subseteq \text{id}(T) \\
 & \wedge \text{id}(S) \subseteq f ; f^{-1}
 \end{aligned}$$

On a les propriétés suivantes.

$$\begin{aligned}
 & f \in S \rightarrow T \\
 \Rightarrow & \\
 & \forall x \cdot x \in S \Rightarrow \text{card}(f[\{x\}]) = 1 \\
 \wedge & \\
 & \forall x \cdot x \in S \Rightarrow \text{card}(f[\{x\}]) \in 1..1
 \end{aligned}$$

** lien avec diag. UML **

Ce qui donne ceci comme exemple typique, avec la partie en rouge qui montre la caractéristique distinctive d'une fonction totale, c'est-à-dire que tous les éléments de S sont reliés à un élément de T par la fonction f .



On note aussi que

$$S \rightarrow T \subseteq S \rightarrow T$$

ce qui signifie qu'une fonction totale est un cas particulier de fonction.

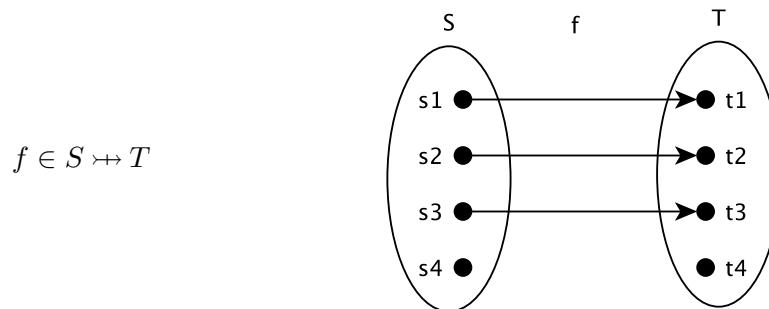
2.5.3 Fonction injective

Une fonction injective (aussi appelée *injection*) f d'un ensemble S vers un ensemble T , notée $f \in S \mapsto T$, est une fonction telle que f^{-1} est aussi une fonction. Cela signifie que chaque élément de T est associé à au plus un élément de S . Ce qui donne les définitions suivantes, qui sont toutes équivalentes.

$$\begin{aligned}
 & f \in S \mapsto T \\
 \Leftrightarrow & \\
 & f \in S \mapsto T \\
 & \wedge \forall x, y, z \cdot x \mapsto z \in f \wedge y \mapsto z \in f \Rightarrow x = y \\
 \Leftrightarrow & \\
 & f \in S \mapsto T \\
 & \wedge f^{-1} \in T \mapsto S \\
 \Leftrightarrow & \\
 & f ; f^{-1} \subseteq \text{id}(S) \\
 & \wedge f^{-1} ; f \subseteq \text{id}(T) \\
 \Leftrightarrow & \\
 & f \in S \mapsto T \\
 & \wedge \forall x \cdot x \in S \Rightarrow \text{card}(f[\{x\}]) \in 0..1 \\
 & \wedge \forall y \cdot y \in T \Rightarrow \text{card}(f^{-1}[\{y\}]) \in 0..1
 \end{aligned}$$

** lien avec diag. UML **
 ** lien avec diag. UML **

Ce qui donne ceci comme exemple typique.



2.5.4 Fonction injective totale

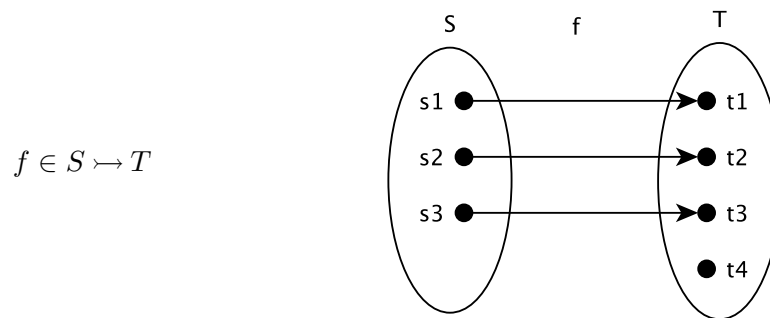
Une fonction injective totale f d'un ensemble S vers un ensemble T est notée $f \in S \twoheadrightarrow T$. Ce qui donne les définitions suivantes, qui sont toutes équivalentes.

$$\begin{aligned}
 & f \in S \twoheadrightarrow T \\
 \Leftrightarrow & \\
 & f \in S \twoheadrightarrow T \\
 & \wedge \text{dom}(f) = S \\
 \Leftrightarrow & \\
 & f ; f^{-1} = \text{id}(S) \\
 & \wedge f^{-1} ; f \subseteq \text{id}(T) \\
 \Leftrightarrow & \\
 & f \in S \twoheadrightarrow T \\
 & \wedge \forall x \cdot x \in S \Rightarrow \text{card}(f[\{x\}]) \in 1..1 \\
 & \wedge \forall y \cdot y \in T \Rightarrow \text{card}(f^{-1}[\{y\}]) \in 0..1
 \end{aligned}$$

** lien avec diag. UML **

** lien avec diag. UML **

Ce qui donne ceci comme exemple typique.



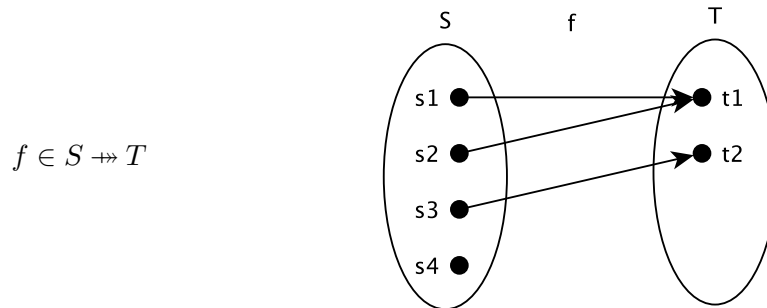
2.5.5 Fonction surjective

Une fonction surjective (aussi appelée *surjection*) f d'un ensemble S vers un ensemble T , notée $f \in S \twoheadrightarrow T$, est une fonction telle que chaque élément de T est une image d'un élément de S . Ce qui donne les définitions suivantes, qui sont toutes équivalentes.

$$\begin{aligned}
 & f \in S \twoheadrightarrow T \\
 \Leftrightarrow & \\
 & f \in S \twoheadrightarrow T \\
 & \wedge \forall y \cdot y \in T \Rightarrow \exists x \cdot x \in \text{dom}(f) \wedge f(x) = y \\
 \Leftrightarrow & \\
 & f \in S \twoheadrightarrow T \\
 & \wedge \text{ran}(f) = T \\
 \Leftrightarrow & \\
 & f \in S \twoheadrightarrow T \\
 & \wedge f^{-1} ; f = \text{id}(T) \\
 \Leftrightarrow & \\
 & f \in S \twoheadrightarrow T \\
 & \wedge \forall x \cdot x \in S \Rightarrow \text{card}(f[\{x\}]) \in 0..1 \\
 & \wedge \forall y \cdot y \in T \Rightarrow \text{card}(f^{-1}[\{y\}]) \in 1.. \text{card}(S)
 \end{aligned}$$

** lien avec diag. UML **
 ** lien avec diag. UML **

Ce qui donne ceci comme exemple typique.



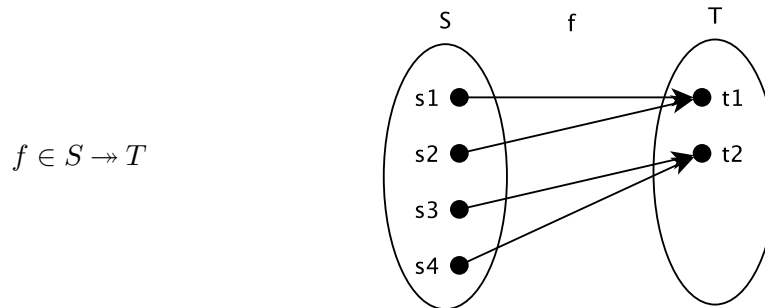
2.5.6 Fonction surjective totale

Une fonction surjective totale f d'un ensemble S vers un ensemble T est notée $f \in S \twoheadrightarrow T$. Ce qui donne les définitions suivantes, qui sont toutes équivalentes.

$$\begin{aligned} & f \in S \twoheadrightarrow T \\ \Leftrightarrow & \\ & f \in S \twoheadrightarrow T \\ & \wedge f \in S \rightarrow T \\ \Leftrightarrow & \\ & f \in S \rightarrow T \\ & \wedge \text{ran}(f) = T \\ \Leftrightarrow & \\ & f \in S \twoheadrightarrow T \\ & \wedge f^{-1}; f = \text{id}(T) \\ & \wedge \text{id}(S) \subseteq f; f^{-1} \\ \Leftrightarrow & \\ & f \in S \twoheadrightarrow T \\ & \wedge \forall x \cdot x \in S \Rightarrow \text{card}(f[\{x\}]) \in 1..1 \\ & \wedge \forall y \cdot y \in T \Rightarrow \text{card}(f^{-1}[\{y\}]) \in 1..\text{card}(S) \end{aligned}$$

** lien avec diag. UML **

Ce qui donne ceci comme exemple typique.



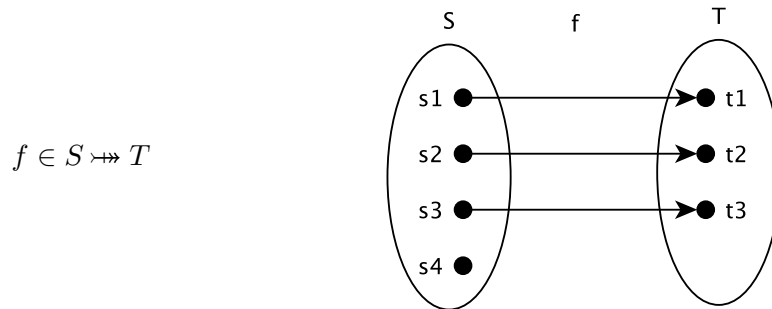
2.5.7 Fonction bijective

Une fonction bijective f d'un ensemble S vers un ensemble T , notée $f \in S \rightsquigarrow T$, est une fonction injective et surjective. Ce qui donne les définitions suivantes, qui sont toutes équivalentes.

$$\begin{aligned} & f \in S \rightsquigarrow T \\ \Leftrightarrow & \\ & f \in S \rightarrow T \\ & \wedge f \in S \twoheadrightarrow T \\ \Leftrightarrow & \\ & f \in S \rightarrow T \\ & \wedge \forall x \cdot x \in S \Rightarrow \text{card}(f[\{x\}]) \in 0..1 \\ & \wedge \forall y \cdot y \in T \Rightarrow \text{card}(f^{-1}[\{y\}]) \in 1..1 \\ \Leftrightarrow & \\ & f^{-1} \in T \rightarrow S \end{aligned}$$

** lien avec diag. UML **

Ce qui donne ceci comme exemple typique.



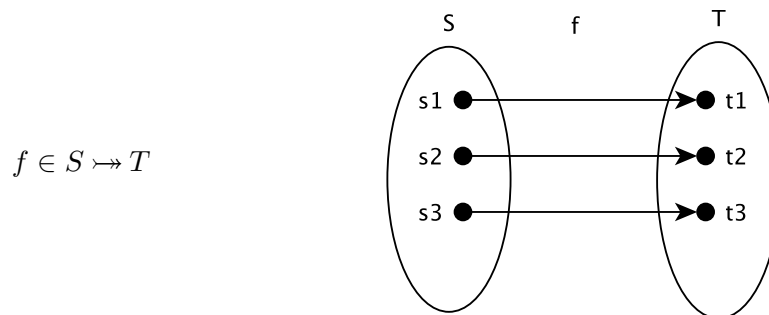
Vu que l'inverse d'une bijection est une injection totale, les bijections ne sont pas utilisées en pratique; on utilise au lieu les injections totales.

2.5.8 Fonction bijective totale

Une fonction bijective totale f d'un ensemble S vers un ensemble T , notée $f \in S \twoheadrightarrow T$, est une fonction totale, injective et surjective. Ce qui donne les définitions suivantes, qui sont toutes équivalentes.

$$\begin{aligned}
 & f \in S \twoheadrightarrow T \\
 \Leftrightarrow & \\
 & f \in S \rightarrow T \\
 \wedge & f \in S \rightarrow T \\
 \Leftrightarrow & \\
 & f ; f^{-1} = \text{id}(S) \\
 \wedge & f^{-1} ; f = \text{id}(T) \\
 \Leftrightarrow & \\
 & f \in S \leftrightarrow T \\
 \wedge & \forall x \cdot x \in S \Rightarrow \text{card}(f[\{x\}]) \in 1..1 && \text{** lien avec diag. UML **} \\
 \wedge & \forall y \cdot y \in T \Rightarrow \text{card}(f^{-1}[\{y\}]) \in 1..1 && \text{** lien avec diag. UML **}
 \end{aligned}$$

Ce qui donne ceci comme exemple typique.



Dans le langage courant, les bijections totales sont simplement appelées des bijections.

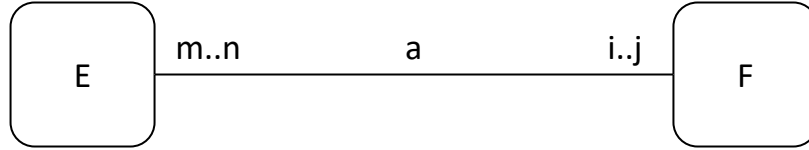


Figure 2.5: Modèle conceptuel de données

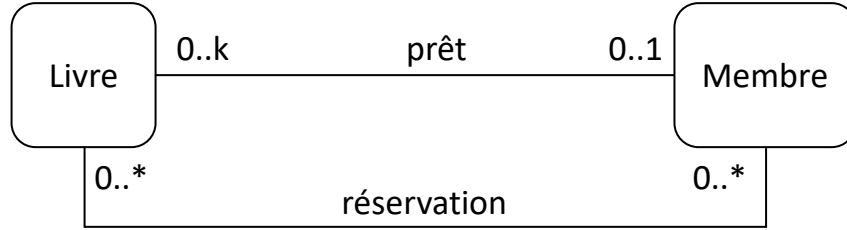


Figure 2.6: Modèle conceptuel d'une bibliothèque

2.5.9 Modèle conceptuel de données

Un modèle conceptuel de données permet de représenter les liens entre les éléments d'un système. La notation la plus usitée est la notation de Chen [3], aussi reprise en UML pour les diagrammes de classes. La figure 2.5 donne un exemple typique de modèle conceptuel. On y retrouve deux entités, E et F , reliées par une association a . Les annotations $i..j$ et $m..n$ sont appelées des multiplicités, ou bien des cardinalités. Une entité dénote un ensemble d'objets; une association dénote une relation. Une multiplicité dénote une contrainte sur le nombre d'images d'un élément d'une entité par l'association a . Voici la signification (sémantique) associée aux multiplicités.

$$i \in \mathbb{N} \wedge j \in \mathbb{N} \Rightarrow (\forall x \cdot x \in E \Rightarrow \text{card}(a[\{x\}]) \in i..j) \quad (2.8)$$

$$m \in \mathbb{N} \wedge n \in \mathbb{N} \Rightarrow (\forall x \cdot x \in F \Rightarrow \text{card}(a^{-1}[\{x\}]) \in m..n) \quad (2.9)$$

L'équation 2.8 signifie qu'un élément de l'entité E est associé à au moins i et au plus j éléments de l'entité F via l'association a . Dualelement, l'équation 2.9 signifie qu'un élément de l'entité F est associé à au moins m et au plus n éléments de l'entité E . De plus, on a que i, j, m, n sont des nombres naturels. On peut aussi utiliser la valeur spéciale "*" pour j et n , ce qui signifie qu'il n'y a pas de borne supérieure pour le nombre d'éléments associés, ce qui donne la sémantique suivante.

$$i \in \mathbb{N} \wedge j = "*" \Rightarrow (\forall x \cdot x \in E \Rightarrow \text{card}(a[\{x\}]) \geq i) \quad (2.10)$$

$$m \in \mathbb{N} \wedge n = "*" \Rightarrow (\forall x \cdot x \in F \Rightarrow \text{card}(a^{-1}[\{x\}]) \geq m) \quad (2.11)$$

La figure 2.6 illustre un modèle conceptuel de données pour un système de gestion de bibliothèque, où on doit gérer les prêts et les réservations de livres pour des membres. Elle indique qu'un livre est emprunté par au plus un membre à la fois (i.e., $0..1$), et qu'un membre peut emprunter au plus k livre à la fois (i.e., $0..k$). Un membre peut réserver un nombre arbitraire de livres (i.e., $0..*$), et un livre peut être réservé par un nombre arbitraire de membres (i.e., $0..*$). Cela donne les contraintes suivantes.

$$\text{pret} \in \text{Livre} \leftrightarrow \text{Membre} \wedge \forall m \cdot m \in \text{Membre} \Rightarrow \text{card}(\text{pret}^{-1}[\{m\}]) \leq k$$

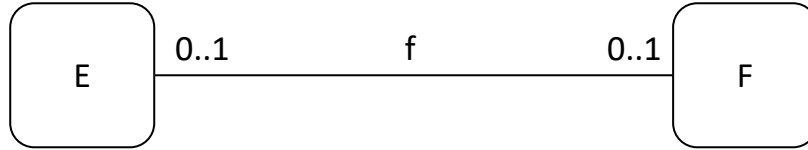


Figure 2.7: Modèle conceptuel correspondant à une fonction injective

$$\textit{reservation} \in \textit{Livre} \leftrightarrow \textit{Membre}$$

L'association prêt est donc une *fonction*, alors que l'association réservation est une *relation*. Chaque classe de fonctions peut être représentée par des valeurs particulières des multiplicités dans un modèle conceptuel de données. Par exemple, les fonctions injectives $f \in E \mapsto F$ sont représentées par le diagramme de la figure 2.7.

2.6 Propriétés des relations

Définition 16 Soit $r \in S \leftrightarrow S$. On définit les propriétés suivantes pour r .

Propriété	Définition en logique	Définition relationnelle	Exemples
réflexive	$\forall x \cdot x \in S \Rightarrow x \mapsto x \in r$	$\text{id}(S) \subseteq r$	$=, \leq, \subseteq$
irréflexive (syn. antiréflexive)	$\forall x \cdot x \in S \Rightarrow x \mapsto x \notin r$	$\text{id}(S) \cap r = \{\}$	$\neq, <, \subset$
transitive	$\forall(x, y, z) \cdot$ $x \mapsto y \in r \wedge y \mapsto z \in r$ \Rightarrow $x \mapsto z \in r$	$r ; r \subseteq r$	$=, <, \leq, \subset, \subseteq$
symétrique	$\forall(x, y) \cdot$ $x \mapsto y \in r$ \Rightarrow $y \mapsto x \in r$	$r^{-1} = r$	$=$
asymétrique (syn. antisymétrique forte)	$\forall(x, y) \cdot$ $x \mapsto y \in r$ \Rightarrow $y \mapsto x \notin r$	$r \cap r^{-1} = \{\}$	$<, \subset$
antisymétrique	$\forall(x, y) \cdot$ $x \mapsto y \in r \wedge y \mapsto x \in r$ \Rightarrow $x = y$	$r \cap r^{-1} \subseteq \text{id}(S)$	$=, <, \leq, \subseteq$
totale	$\forall x \cdot x \in S \Rightarrow \exists y \cdot x \mapsto y \in r$	$\text{dom}(r) = S$	$=, <, \leq, \subseteq$
surjective	$\forall x \cdot x \in S \Rightarrow \exists y \cdot y \mapsto x \in r$	$\text{ran}(r) = S$	$=, <, \leq, \subseteq$
pré-ordre	réflexive et transitive		$=, \leq, \subseteq$
équivalence	réflexive, transitive, symétrique		$=$
ordre	réflexive, transitive et antisymétrique		$=, \leq, \subseteq$
ordre strict	irréflexive et transitive (par déduction, elle est aussi asymétrique)		$<, \subset$
bien fondée	$\forall T \cdot T \in \mathbb{P}_1(S) \Rightarrow$ $\exists s_1 \cdot s_1 \in T \wedge \forall s_2 \cdot s_2 \mapsto s_1 \notin r$ i.e., il existe un élément minimal dans T . Ou bien, de manière équivalente, il n'existe pas de suite infinie x_0, x_1, \dots telle que $x_{n+1} \mapsto x_n \in r$ i.e., $\dots < x_2 < x_1 < x_0$		$<$ sur \mathbb{N}, \subset
acyclique	il n'existe pas de suite x_0, \dots, x_n telle que $x_n \mapsto x_0 \in r$ et $\forall i \cdot i \in 0..(n-1) \Rightarrow x_i \mapsto x_{i+1} \in r$	$r^+ \cap \text{id}(S) = \{\}$	$<$ sur $\mathbb{N}, \mathbb{Z}, \mathbb{R}, \subset$

□

2.6.1 Relation réflexive

Propriété	Définition en logique	Définition relationnelle	Exemples
réflexive	$\forall x \cdot x \in S \Rightarrow x \mapsto x \in r$	$\text{id}(S) \subseteq r$	$=, \leq, \subseteq$

Voici quelques autres exemples.

1. Soit $T = \{a, b\}$. Il existe 4 relations réflexives dans $T \leftrightarrow T$.

- (a) $\{(a \mapsto a), (b \mapsto b)\}$
- (b) $\{(a \mapsto a), (b \mapsto b), (a \mapsto b)\}$
- (c) $\{(a \mapsto a), (b \mapsto b), (b \mapsto a)\}$
- (d) $\{(a \mapsto a), (b \mapsto b), (a \mapsto b), (b \mapsto a)\}$

Pour être réflexive, une relation doit contenir $\text{id}(T) = \{(a \mapsto a), (b \mapsto b)\}$.

2. Soit T un ensemble de pièces chez un détaillant de pièces d'auto. Une pièce (par exemple un phare, un pare-brise, des freins, un pneu) peut être produite par différents fournisseurs. Le détaillant désire rechercher une pièce dans son inventaire et obtenir en sortie toutes les pièces disponibles qui lui sont équivalentes. On peut alors définir la relation **PièceÉquivalente** $\in T \leftrightarrow T$ telle que

$$x \mapsto y \in \text{PièceÉquivalente} \Leftrightarrow \text{“}x \text{ est une pièce équivalente à } y\text{”}$$

Cette relation est réflexive, car toute pièce est équivalente à elle-même. Quand le détaillant fait une recherche sur les pièces équivalentes à x qu'il détient en inventaire, il veut obtenir x et aussi toutes les autres pièces considérées équivalentes à x .

2.6.2 Relation irréflexive

Propriété	Définition en logique	Définition relationnelle	Exemples
irréflexive (syn. antiréflexive)	$\forall x \cdot x \in S \Rightarrow x \mapsto x \notin r$	$\text{id}(S) \cap r = \{\}$	$\neq, <, \subset$

Voici quelques autres exemples.

1. Soit $T = \{a, b\}$. Il existe 4 relations irréflexives dans $T \leftrightarrow T$.

- (a) $\{\}$
- (b) $\{(a \mapsto b)\}$
- (c) $\{(a \mapsto b), (b \mapsto a)\}$
- (d) $\{(b \mapsto a)\}$

Pour qu'une relation soit irréflexive, elle ne doit contenir aucun couple de l'identité $\text{id}(T)$.

2. Si une relation *n'est pas* réflexive, cela ne veut pas dire qu'elle est nécessairement irréflexive. Autrement dit, irréflexive n'est pas le contraire de réflexive, et vice-versa (une relation qui n'est pas irréflexive n'est pas nécessairement réflexive). Ainsi, pour $T = \{a, b\}$, il y a 8 relations qui ne sont *ni réflexives, ni irréflexives*. En voici 4 exemples.

- (a) $\{(a \mapsto a)\}$
- (b) $\{(a \mapsto a), (a \mapsto b)\}$
- (c) $\{(a \mapsto a), (b \mapsto a)\}$
- (d) $\{(a \mapsto a), (a \mapsto b), (b \mapsto a)\}$

Leur caractéristique commune est qu'elles contiennent un sous-ensemble *strict* de $\text{id}(T)$.

3. Les relations familiales suivantes sont irréflexives (ainsi que plusieurs autres) : Parent, Grand-Parent, Fraterie, Cousin, Oncle, Ancêtre.

2.6.3 Relation transitive

Propriété	Définition en logique	Définition relationnelle	Exemples
transitive	$\forall(x, y, z) \cdot$ $x \mapsto y \in r \wedge y \mapsto z \in r$ \Rightarrow $x \mapsto z \in r$	$r ; r \subseteq r$	$=, <, \leq, \subset, \subseteq$

Voici quelques autres exemples.

1. Soit $T = \{a, b, c, d\}$. La relation suivante n'est pas transitive

$$\{a \mapsto b, b \mapsto c, c \mapsto d\}$$

Il lui manque les 3 couples suivants

- (a) $a \mapsto c$, puisque $a \mapsto b \in r$ et $b \mapsto c \in r$
- (b) $a \mapsto d$, puisque $a \mapsto c$ devrait être dans r et $c \mapsto d \in r$
- (c) $b \mapsto d$, puisque $b \mapsto c \in r$ et $c \mapsto d \in r$

La relation suivante est transitive

$$\{a \mapsto b, a \mapsto c, a \mapsto d, b \mapsto c, b \mapsto d, c \mapsto d\}$$

Si une relation r est transitive, alors $r = r^+ = \bigcup_{n \geq 1} r^n$. Voyons pourquoi. Par définition de transitivité, on a $r ; r \subseteq r$. Donc $r^2 \subseteq r$. Pour prouver que $n > 0 \Rightarrow r^n \subseteq r$, nous allons utiliser la propriété suivante, soit la monotonie de la composition relationnelle.

$$r_1 \subseteq r_2 \Rightarrow r_1 ; r_3 \subseteq r_2 ; r_3$$

Montrons que $r^3 \subseteq r$.

$$\begin{aligned} & r^2 \subseteq r \\ \Rightarrow & r^2 ; r \subseteq r ; r && \langle \text{monotonie de } ; \rangle \\ \Leftrightarrow & r^3 \subseteq r^2 && \langle \text{def. de } r^n \rangle \\ \Rightarrow & r^3 \subseteq r && \langle r^2 \subseteq r \text{ et transitivité de } \subseteq \rangle \end{aligned}$$

On pourrait continuer ainsi pour chaque valeur de $n > 3$. Pour prouver complètement, nous pourrions effectuer une preuve par induction, ce qui est l'objet du chapitre 3.

2. Les relations familiales suivantes sont transitives : **Ancêtre**.
3. La relation **Préalable** sur les cours n'est pas transitive. La relation **Préalable**⁺ est transitive.
4. La relation **PièceÉquivalente** est transitive.

2.6.4 Relation symétrique

Propriété	Définition en logique	Définition relationnelle	Exemples
symétrique	$\forall(x, y) \cdot$ $x \mapsto y \in r$ \Rightarrow $y \mapsto x \in r$	$r^{-1} = r$	=

Voici quelques exemples.

1. Soit $T = \{a, b, c, d\}$. La relation suivante n'est pas symétrique

$$\{a \mapsto b, b \mapsto c, c \mapsto d\}$$

Il lui manque les couples suivants pour être symétrique.

$$\{b \mapsto a, c \mapsto b, d \mapsto c\}$$

La relation suivante est symétrique:

$$\{a \mapsto b, b \mapsto a, b \mapsto c, c \mapsto b, c \mapsto d, d \mapsto c\}$$

2. Les relations familiales suivantes sont symétriques : Fraterie, Cousin.
3. La relation PièceÉquivalente est symétrique.

2.6.5 Relation asymétrique

Propriété	Définition en logique	Définition relationnelle	Exemples
asymétrique (syn. antisymétrique forte)	$\forall(x, y) \cdot$ $x \mapsto y \in r$ \Rightarrow $y \mapsto x \notin r$	$r \cap r^{-1} = \{\}$	$<, \subset$

Voici quelques exemples.

1. Soit $T = \{a, b, c, d\}$. La relation suivante n'est pas asymétrique

$$r_1 = \{a \mapsto a, a \mapsto b, b \mapsto a, b \mapsto c, c \mapsto d\}$$

Il faut lui retirer les couples suivants pour être asymétrique.

- (a) $a \mapsto a$, par définition de asymétrique ($x \mapsto y \in r \Rightarrow y \mapsto x \notin r$); dans ce cas, on a $x = y$ pour le couple $a \mapsto a$, donc il ne peut y avoir de couple de la forme $x \mapsto x$ dans une relation asymétrique.
- (b) l'un des deux couples suivants: $a \mapsto b$, ou bien $b \mapsto a$ La relation suivante est asymétrique:

$$\{b \mapsto a, b \mapsto c, c \mapsto d\}$$

2. Une relation qui n'est pas symétrique n'est pas nécessairement asymétrique. La relation r_1 ci-dessus n'est ni symétrique, ni asymétrique. Elle n'est pas symétrique parce qu'il lui manque les couples $c \mapsto b$ et $d \mapsto c$.
3. Une relation asymétrique est toujours irréflexive.

$$r \text{ est asymétrique} \Rightarrow r \text{ est irréflexive}$$

4. Les relations familiales suivantes sont asymétriques : Parent, GrandParent.

2.6.6 Relation antisymétrique

Propriété	Définition en logique	Définition relationnelle	Exemples
antisymétrique	$\forall(x, y) \cdot$ $x \mapsto y \in r \wedge y \mapsto x \in r$ \Rightarrow $x = y$	$r \cap r^{-1} \subseteq \text{id}(S)$	$=, <, \leq, >, \geq, \subset, \subseteq$

Voici quelques exemples.

1. Soit $T = \{a, b, c, d\}$. La relation suivante n'est pas antisymétrique

$$r = \{a \mapsto a, a \mapsto b, b \mapsto a, b \mapsto c, c \mapsto d\}$$

Il faut la modifier de la manière suivante pour qu'elle devienne antisymétrique.

- (a) l'un des deux couples suivant: $a \mapsto b$, ou bien $b \mapsto a$
2. Une relation asymétrique est aussi antisymétrique
 3. Une relation antisymétrique n'est pas nécessairement asymétrique, car les couples de la forme $x \mapsto x$ ne sont pas permis dans une relation asymétrique, mais ils le sont dans une relation antisymétrique.
 4. Une relation antisymétrique et irréflexive est asymétrique.

$$r \text{ asymétrique} \Rightarrow r \text{ antisymétrique}$$

5. Une relation antisymétrique n'est pas nécessairement réflexive (e.g., $<$).

2.6.7 Relation d'équivalence

Propriété	Définition en logique	Définition relationnelle	Exemples
équivalence	réflexive, transitive, symétrique		=

Voici quelques exemples.

1. Soit $T = \{a, b, c\}$. La relation suivante n'est pas une relation d'équivalence

$$r = \{a \mapsto a, a \mapsto b, b \mapsto a, b \mapsto c\}$$

Il faut lui ajouter les couples suivants pour en faire une relation d'équivalence.

$$\{b \mapsto b, c \mapsto c, c \mapsto b, a \mapsto c, c \mapsto a\}$$

2. La relation PièceÉquivalente est une relation d'équivalence.
3. La relation Fraterie n'est pas une relation d'équivalence, car elle n'est pas réflexive.

2.6.8 Relation d'ordre

Propriété	Définition en logique	Définition relationnelle	Exemples
ordre	réflexive, transitive et antisymétrique		=, ≤, ⊆

Voici quelques exemples.

1. Soit $T = \{a, b, c\}$. La relation suivante n'est pas une relation d'ordre.

$$\{a \mapsto a, a \mapsto b, b \mapsto a, b \mapsto c\}$$

On peut la transformer en relation d'ordre en retirant le couple $b \mapsto a$, et en lui ajoutant les couples suivants.

$$\{b \mapsto b, c \mapsto c, a \mapsto c\}$$

Il y a bien sûr d'autres façons de la transformer en relation d'ordre. Par exemple, en retirant $a \mapsto b$ et en ajoutant

$$\{b \mapsto b, c \mapsto c\}$$

La relation suivante est une relation d'ordre

$$\{a \mapsto a, a \mapsto b, a \mapsto c, b \mapsto b, b \mapsto c, c \mapsto c\}$$

2.6.9 Relation d'ordre strict

Propriété	Définition en logique	Définition relationnelle	Exemples
ordre strict	irréflexive et transitive		$<, \subset$

Voici quelques exemples.

1. Soit $T = \{a, b, c\}$. La relation suivante n'est pas une relation d'ordre strict.

$$r = \{a \mapsto a, a \mapsto b, b \mapsto a, b \mapsto c\}$$

On peut la transformer en relation d'ordre en retirant les couples $a \mapsto a$ et $b \mapsto a$, et en lui ajoutant le couple $a \mapsto c$. Il y a bien sûr d'autres façons de la transformer en relation d'ordre strict. Par exemple, en retirant $a \mapsto a$ et $a \mapsto b$. La relation suivante est une relation d'ordre strict

$$\{a \mapsto b, a \mapsto c, b \mapsto c\}$$

2. Une relation d'ordre strict est aussi asymétrique.

$$r \text{ est un ordre strict} \Rightarrow r \text{ est asymétrique}$$

2.6.10 Relation bien fondée

Propriété	Définition en logique	Définition relationnelle	Exemples
bien fondée	$\forall T \cdot T \in \mathbb{P}_1(S) \Rightarrow$ $\exists s_1 \cdot s_1 \in T \wedge \forall s_2 \cdot s_2 \mapsto s_1 \notin r$ i.e., il existe un élément minimal dans T . Ou bien, de manière équivalente, il n'existe pas de suite infinie x_0, x_1, \dots telle que $x_{n+1} \mapsto x_n \in r$ i.e., $\dots \mapsto x_2 \mapsto x_1 \mapsto x_0$		$<$ sur \mathbb{N}, \subset

Voici quelques exemples.

1. Une relation bien fondée ne permet pas de remonter les flèches infiniment; il faut que la relation s'arrête quand on remonte les flèches, i.e., la relation suivante n'est pas bien fondée:

$$\dots \mapsto x_2 \mapsto x_1 \mapsto x_0$$

2. La relation $<$ sur les naturels (\mathbb{N}) est bien fondée, car on s'arrête à 0 quand on remonte les flèches.

$$0 \mapsto 1 \mapsto 2 \mapsto 3 \mapsto \dots$$

3. La relation $<$ sur \mathbb{N}_1 est aussi bien fondée.

4. La relation $<$ sur les entiers (\mathbb{Z}) n'est pas bien fondée, à cause de la suite suivante

$$0, -1, -2, \dots$$

c'est à dire

$$\dots \mapsto -3 \mapsto -2 \mapsto -1 \mapsto 0$$

5. La relation $<$ sur les réels (\mathbb{R}) n'est pas bien fondée.
6. La relation \subset sur les ensembles est bien fondée.
7. Les relations Parent, Ancêtre et Préalable sont bien fondées.
8. Une relation d'ordre strict n'est pas nécessairement bien fondée (e.g., $<$ sur \mathbb{Z}).
9. Une relation bien fondée n'est pas nécessairement une relation d'ordre strict, car une relation bien fondée n'est pas nécessairement transitive.
10. Une relation bien fondée est asymétrique et irréflexive.

$$r \text{ est bien fondée} \Rightarrow r \text{ est asymétrique et irréflexive}$$

2.6.11 Relation acyclique

Propriété	Définition en logique	Définition relationnelle	Exemples
acyclique	il n'existe pas de suite x_0, \dots, x_n telle que $x_n \mapsto x_0 \in r$ et $\forall i \cdot i \in 0..(n-1) \Rightarrow x_i \mapsto x_{i+1} \in r$	$r^+ \cap \text{id}(S) = \{\}$	$<$ sur $\mathbb{N}, \mathbb{Z}, \mathbb{R}, \mathbb{C}$

Voici quelques exemples.

1. Une relation acyclique ne contient pas de cycle

$$x_0 \mapsto \dots \mapsto x_0$$

2. Une relation bien fondée est aussi acyclique.

$$r \text{ est bien fondée} \Rightarrow r \text{ est acyclique}$$

3. Une relation acyclique n'est pas nécessairement bien fondée; par exemple, la relation $<$ sur les entiers (\mathbb{Z}) ne contient pas de cycle, mais on peut remonter les flèches infiniment, i.e., vers la gauche, dans la l'exemple ci-dessous.

$$\dots \mapsto -3 \mapsto -2 \mapsto -1 \mapsto 0$$

4. Les relations Parent, Oncle et Préalable sont acycliques.
5. Les relations Fraterie et Cousin ne sont pas acycliques.

2.7 Les nombres

Description	Expression	Syntaxe ASCII B	Exemple
successeur	succ	succ	$\text{succ} = \lambda x.(x \in \mathbb{Z} \mid x + 1)$
addition	$m + n$	$m + n$	$1 + 1 = 2$
soustraction	$m - n$	$m - n$	$2 - 1 = 1$
multiplication	$m * n$	$m * n$	$2 * 2 = 4$
puissance	m^n	$m ** n$	$3^2 = 9$
division entière	m/n	m/n	$5/2 = 2$
modulo	$m \bmod n$	$m \bmod n$	$5 \bmod 2 = 1$
maximum	$\max(S)$	$\max(S)$	$\max(\{1, 2, 3\}) = 3$
minimum	$\min(S)$	$\min(S)$	$\min(\{1, 2, 3\}) = 1$
somme quantifiée	$\Sigma(x).(\mathcal{A} \mid t)$	$\text{SIGMA}(x).(\mathcal{A} \mid t)$	$\Sigma(x).(x \in 1..3 \mid x * 2) =$ $1 * 2 + 2 * 2 + 3 * 2 = 12$
produit quantifié	$\Pi(x).(\mathcal{A} \mid t)$	$\text{PI}(x).(\mathcal{A} \mid t)$	$\Pi(x).(x \in 1..3 \mid x + 1) =$ $(1 + 1) * (2 + 1) * (3 + 1) = 24$
plus petit	$x < y$	$x < y$	$1 < 2$
plus petit ou égal	$x \leq y$	$x <= y$	$1 \leq 1$
plus grand	$x > y$	$x > y$	$2 > 1$
plus grand ou égal	$x \geq y$	$x >= y$	$2 \geq 2$

Table 2.9: Opérations et prédicats sur les nombres

2.8 Diverses lois

Soit A, B, C, S, T des ensembles et r, r_1, r_2 des relations sur S .

<i>Égalité transitive</i>	$x = y \wedge y = z \Rightarrow x = z$	(LE-1)
<i>Leibniz</i>	$x \in S \wedge y \in S \wedge f \in S \rightarrow T \Rightarrow$ $(x = y \Rightarrow f(x) = f(y))$	(LE-2)
<i>inclusion prédicat</i>	$\mathcal{A} \Rightarrow \mathcal{B} \Rightarrow \{x \mid \mathcal{A}\} \subseteq \{x \mid \mathcal{B}\}$	(LE-3)
<i>appartenance 1</i>	$x \in \{y \mid \mathcal{A}\} \Leftrightarrow \mathcal{A}[y := x]$	(LE-4)
<i>appartenance 2</i>	$x \in \{x \mid \mathcal{A}\} \Leftrightarrow \mathcal{A}$	(LE-5)
<i>égalité ensemble 1</i>	$S = T \Leftrightarrow \forall x \cdot x \in S \Leftrightarrow x \in T$	(LE-6)
<i>inclusion ensemble</i>	$S \subseteq T \Leftrightarrow \forall x \cdot x \in S \Rightarrow x \in T$	(LE-7)
<i>égalité ensemble 2</i>	$S = T \Leftrightarrow S \subseteq T \wedge T \subseteq S$	(LE-8)
<i>monotonie union</i>	$A \subseteq B \Rightarrow A \cup C \subseteq B \cup C$	(LE-9)
<i>monotonie intersection</i>	$A \subseteq B \Rightarrow A \cap C \subseteq B \cap C$	(LE-10)
<i>monotonie différence</i>	$A \subseteq B \Rightarrow A - C \subseteq B - C$	(LE-11)
<i>antitonicité différence</i>	$A \subseteq B \Rightarrow C - A \supseteq C - B$	(LE-12)
<i>monotonie cardinalité</i>	$A \subseteq B \Rightarrow \text{card}(A) \leq \text{card}(B)$	(LE-13)
<i>intersection inclusion</i>	$A \cap B \subseteq A$	(LE-14)
<i>union inclusion</i>	$A \subseteq A \cup B$	(LE-15)
<i>monotonie produit</i>	$r_1 \subseteq r_2 \Rightarrow$ $r_1 ; r_3 \subseteq r_2 ; r_3$ $\wedge r_3 ; r_1 \subseteq r_3 ; r_2$	(LE-16)
<i>monotonie inverse</i>	$r_1 \subseteq r_2 \Rightarrow r_1^{-1} \subseteq r_2^{-1}$	(LE-17)
<i>monotonie addition</i>	$i < j \Rightarrow i + k < j + k$	(LE-18)
<i>monotonie soustraction</i>	$i < j \Rightarrow i - k < j - k$	(LE-19)
<i>monotonie multiplication</i>	$k > 0 \Rightarrow (i < j \Rightarrow i * k < j * k)$	(LE-20)
<i>antitonicité multiplication</i>	$k < 0 \Rightarrow (i < j \Rightarrow i * k > j * k)$	(LE-21)
<i>évaluation fonction</i>	si $x \in S \wedge f \in S \rightarrow T \wedge g \in S \rightarrow T$ alors $f = g \Rightarrow f(x) = g(x)$	(LE-22)
<i>composition fonction</i>	si $x \in S \wedge f \in S \rightarrow T \wedge g \in T \rightarrow U$ alors $(f ; g)(x) = g(f(x))$	(LE-23)
<i>égalité substitution</i>	si $x = y$ alors $t = u \Rightarrow (t = u)[x := y]$	(LE-24)
<i>identité élément neutre</i>	$\text{id}(S) ; r = r$	(LE-25)
<i>identité élément neutre</i>	$r ; \text{id}(S) = r$	(LE-26)
<i>typage composition</i>	$f \in S \rightarrow T \wedge g \in T \rightarrow U \Rightarrow f ; g \in S \rightarrow U$	(LE-27)
<i>typage fonction</i>	$f \in S \rightarrow T \wedge x \in \text{dom}(f) \Rightarrow f(x) \in T$	(LE-28)
<i>typage fonction totale</i>	$f \in S \rightarrow T \wedge x \in S \Rightarrow f(x) \in T$	(LE-29)

Table 2.10: Diverses lois

2.9 Les suites

- $w = [\sigma_1, \dots, \sigma_n]$ dénote une *suite* formée des symboles $\sigma_1, \dots, \sigma_n$.

En B , une suite w formée d'éléments de S est une fonction de $\mathbb{N} \rightarrow S$ telle que

$$\text{dom}(w) = 1.. \text{card}(w)$$

On écrit donc $w(i)$ en B pour dénoter σ_i . Le tableau 2.11 présente les opérateurs du langage B permettant de manipuler des suites.

Plusieurs synonymes du terme *suite* sont utilisés dans la littérature:

- mot,
- chaîne de caractères (inspiré de l'anglais *character string*),
- séquence de symboles.

Une suite se distingue d'un ensemble par les caractéristiques suivantes:

- l'ordre d'énumération des éléments de la suite est important, alors que l'ordre d'énumération des éléments d'un ensemble n'est pas important;
- un élément peut apparaître plusieurs fois dans une suite, et chaque occurrence compte, alors qu'un élément peut apparaître une seule fois dans un ensemble.

Par exemple, on a les propriétés suivantes.

$$\begin{array}{ll} [a, b] \neq [b, a] & \{a, b\} = \{b, a\} \\ [a, a] \neq [a] & \{a, a\} = \{a\} \end{array}$$

- une suite *injective* ne contient pas de doublons, car elle est définie sur l'espace $\mathbb{N} \mapsto S$.

Le tableau 2.11 décrit les principales opérations sur les suites.

Description	Expression	Syntaxe ASCII B	Définition/Exemple
suite vide	$[]$	$[]$	
suite par extension	$[t_1, \dots, t_n]$	$[t_1, \dots, t_n]$	
suites sur S	$\text{seq}(S)$	$\text{seq}(S)$	$\{f \mid f \in \mathbb{N} \mapsto S \wedge \text{finite}(f) \wedge \text{dom}(f) = 1.. \text{card}(f)\}$
suites non-vide sur S	$\text{seq}_1(S)$	$\text{seq}_1(S)$	$\text{seq}(S) - []$
suites injective sur S	$\text{iseq}(S)$	$\text{iseq}(S)$	$\text{seq}(S) \cap \mathbb{N} \mapsto S$
suites inj. non-vide sur S	$\text{iseq}_1(S)$	$\text{iseq}_1(S)$	$\text{iseq}_1(S) - []$
concaténation	$s_1 \hat{\ } s_2$	$s_1 s_2$	$[a, b] \hat{\ } [c, d] = [a, b, c, d]$ $s_1 \hat{\ } s_2 = s_1 \cup ((\text{succ}^{-1})^{\text{card}(s_1)} ; s_2)$
premier élément	$\text{first}(s)$	$\text{first}(s)$	$s \neq []$, $\text{first}([a, b, c]) = a$ $s \neq [] \Rightarrow \text{first}(s) = s(1)$
sauf premier élément	$\text{tail}(s)$	$\text{tail}(s)$	$s \neq []$, $\text{tail}([a, b, c]) = [b, c]$ $s \neq [] \Rightarrow \text{tail}(s) = \text{succ} ; (\{1\} \triangleleft s)$
dernier élément	$\text{last}(s)$	$\text{last}(s)$	$s \neq []$, $\text{last}([a, b, c]) = c$ $s \neq [] \Rightarrow \text{last}(s) = s(\text{card}(s))$
sauf dernier élément	$\text{front}(s)$	$\text{front}(s)$	$s \neq []$, $\text{front}([a, b, c]) = [a, b]$ $s \neq [] \Rightarrow \text{front}(s) = \{\text{card}(s)\} \triangleleft s$
inverse	$\text{rev}(s)$	$\text{rev}(s)$	$\text{rev}([a, b, c]) = [c, b, a]$ $\text{rev}(s) = (\lambda x. (x \in \text{dom}(s) \mid \text{card}(s) - x + 1)) ; s$
ajout de e au début de s	$e \rightarrow s$	$e \rightarrow s$	$c \rightarrow [a, b] = [c, a, b]$ $e \rightarrow s = [e] \hat{\ } s$
ajout de e à la fin de s	$s \leftarrow e$	$s \leftarrow e$	$[a, b] \leftarrow c = [a, b, c]$ $s \leftarrow e = s \hat{\ } [e]$

Table 2.11: Opérations et prédicats sur les suites

2.10 Combinatoire

Voici quelques résultats de combinatoire fréquemment utilisés et associés aux éléments de ce chapitre.

- $\text{card}(\mathbb{P}(S)) = 2^{\text{card}(S)}$
- $\text{card}(S \times T) = \text{card}(S) * \text{card}(T)$
- $\text{card}(S \rightarrow T) = \text{card}(T)^{\text{card}(S)}$
- $\text{card}(S \mapsto T) = \frac{\text{card}(T)!}{(\text{card}(T) - \text{card}(S))!}$
 - Lorsque $S = 1..k$, alors $s \in S \mapsto T$ est une séquence injective, que l'on appelle aussi un *arrangement* en combinatoire. Si $n = \text{card}(T)$, alors on note typiquement

$$\text{card}(1..k \mapsto T) = A_k^n$$

- $1 \leq k \leq \text{card}(S) \Rightarrow \text{card}(\{x \mid x \in \mathbb{P}(S) \wedge \text{card}(x) = k\}) = \frac{\text{card}(S)!}{(\text{card}(S) - k)! * k!}$

- Si $n = \text{card}(S)$, alors cette expression est aussi appelée le *coefficient binomial*, ou bien le nombre de *combinaisons*, et dénotée

$$C_k^n = \binom{n}{k} = \frac{n!}{(n-k)!k!}$$

- Elle dénote le nombre de choix possibles de k éléments de S .

2.11 Utilisation de ProB

Nous utilisons le langage B et l'outil ProB pour tester des formules de logique du premier ordre ainsi que des expressions sur des structures discrètes (ensemble, relation et fonction). Une spécification B a la forme suivante:

```
MACHINE nomMachine

SETS
    nomEnsemble = {element, ..., element}
; ...
; nomEnsemble = {element, ..., element}

CONSTANTS
    nomConstante, ..., nomConstante

PROPERTIES
     $\mathcal{A}$ 
END
```

La clause SETS permet de déclarer des ensemble par énumération de leurs éléments. Les déclarations des ensembles sont séparées par des “;”. Ces ensembles peuvent ensuite être utilisés pour définir d'autres ensembles dans la clause CONSTANTS. La clause CONSTANTS permet de déclarer des *noms* de constante, séparées par des “;”. La valeur de ces constantes est déterminée par la clause PROPERTIES. La clause PROPERTIES permet de déclarer *une* (et une seule) *formule* qui détermine la valeur des constantes. Il s'agit généralement d'une conjonction, que l'on écrit avec l'indentation suivante, pour en faciliter la lecture.

```
     $\mathcal{A}_1$ 
     $\wedge \mathcal{A}_2$ 
     $\wedge \dots$ 
     $\wedge \mathcal{A}_n$ 
```

Voici un exemple de machine qui définit des relations familiales.

```
MACHINE Exemple

/* Voici un commentaire sur
   plusieurs lignes
*/

// Voici un autre commentaire

SETS

    Personne = {p0, p1, p2, p3, p4, p5, p6}

CONSTANTS
```

```

    Homme
  , Femme
  , Parent      /* (x,y) : Parent ssi x est un parent de y */
  , OncleTante /* (x,y) : OncleTante ssi x est un oncle ou une tante de y */
  , OncleTante_alt /* Définition alternative de OncleTante,
                    utilisant seulement des opérations sur les relations */
  , Oncle      /* (x,y) : Oncle ssi x est un oncle de y */
  , Tante      /* (x,y) : Tante ssi x est une tante de y */
  , EnsOncle   /* ensemble des personnes qui sont des oncles */

PROPERTIES

    Homme={p0, p1}
& Femme=Personne-Homme
& Parent={
    (p0,p1)
    , (p0,p2)
    , (p1,p3)
    , (p1,p4)
    , (p2,p5)
    , (p2,p6)
    }
& OncleTante =
    { x,y |
      #(z1,z2).
      (
        z2 /= x
        & (z1,x) : Parent
        & (z1,z2) : Parent
        & (z2,y) : Parent
      )
    }
& OncleTante_alt = (((Parent~ ; Parent) - id(Personne)) ; Parent)

& Oncle = Homme <| OncleTante
& Tante = Femme <| OncleTante

& EnsOncle = dom(OncleTante) /\ Homme
END

```

La figure 2.8 représente l'écran principal de ProB. La partie supérieure est un éditeur de spécification. La partie inférieure est divisée en trois parties. À droite, sous la section **State Properties**, on retrouve la définition des symboles. Au départ, seuls les symboles prédéfinis (ex., MAXINT et MININT) y apparaissent, ainsi que les ensembles de la clause SETS. Ces informations sont affichées lorsqu'on ouvre une spécification. Si on modifie une spécification, il faut la ré-ouvrir (menu **File**→**Reopen**, ou bien le raccourci CTRL-R) pour la recompiler. Si la spécification contient des erreurs de syntaxe ou si elle est insatisfaisable, un message d'erreur approprié (mais pas toujours très intuitif) est affiché. Pour afficher les constantes de la spécification, il faut double-cliquer sur **SETUP _CONSTANTS** dans

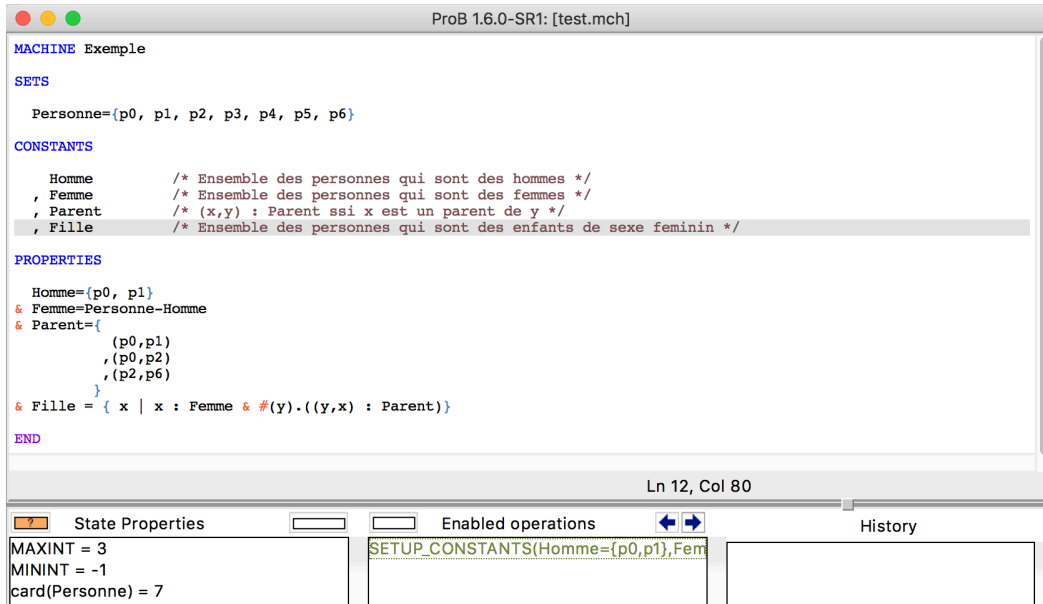


Figure 2.8: L'écran principal de ProB

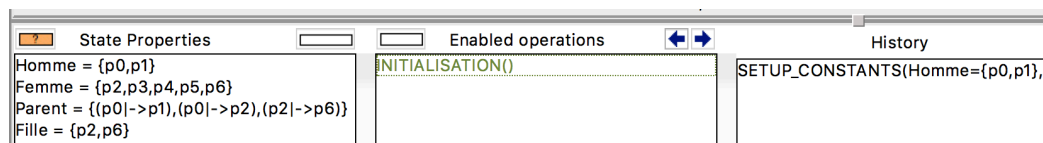


Figure 2.9: Affichage de la valeur des constantes

la partie centrale intitulée **Enabled Operations**. ProB affiche alors les valeurs des constantes qu'il a calculées pour satisfaire la formule de la clause **PROPERTIES** (voir figure 2.9). La figure 2.10 illustre un message d'erreur quand la spécification est insatisfaisable. On peut aussi appeler l'évaluateur de ProB, avec le (menu **Analyse**→**Eval...**, ou bien en double-cliquant n'importe où dans la fenêtre **State Properties**). L'évaluateur peut évaluer n'importe quelle formule ou expression du langage en utilisant la valeur actuelle des constantes (voir figure 2.11).

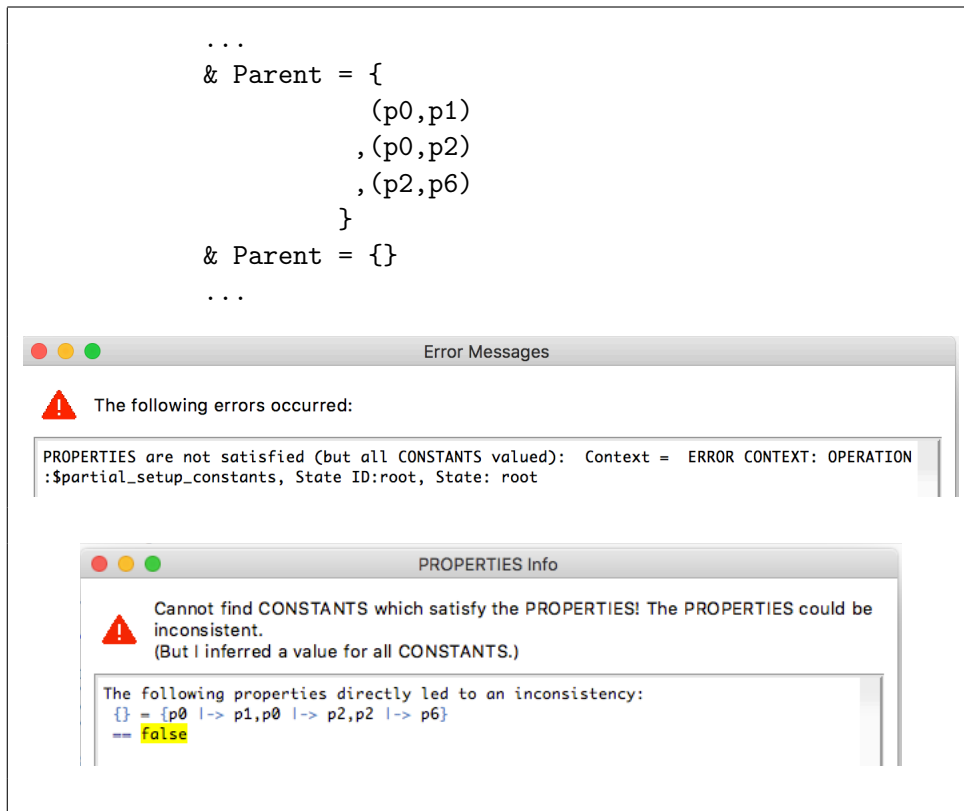


Figure 2.10: Spécification insatisfaisable

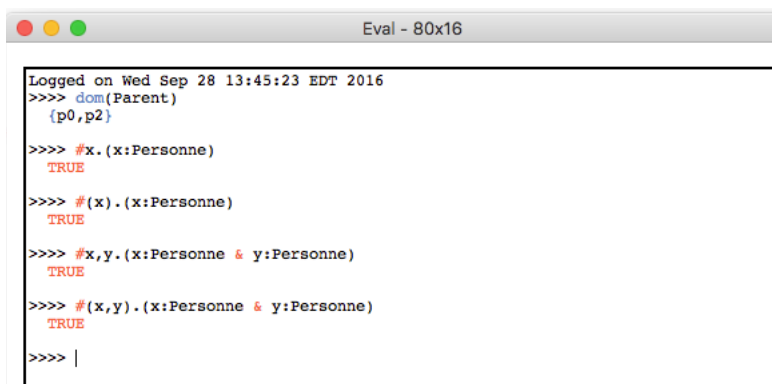


Figure 2.11: L'évaluateur d'expressions et de formules de ProB

2.12 Exercices

1. Soit

- $A = \{0, 1\}$,
- $B = \{a, b\}$,
- $C = B \cup \{c, d\}$,
- $D = \{0 \mapsto a, 1 \mapsto b, 2 \mapsto c, 3 \mapsto d\}$,
- $E = \{0 \mapsto c, 1 \mapsto d\}$,
- $F = \{c \mapsto 0, c \mapsto 1, d \mapsto 1\}$

Évaluez les expressions suivantes.

(a) Calculez $A \times B$.

(b) Déterminez si les énoncés suivants sont vrais. Justifiez votre réponse

- i. $\emptyset \in \mathbb{P}(\emptyset)$
- ii. $\emptyset \subseteq \mathbb{P}(\emptyset)$
- iii. $\emptyset \in A \times B$
- iv. $\emptyset \subseteq A \times B$
- v. $\emptyset \in \emptyset$
- vi. $\emptyset \subseteq \emptyset$
- vii. $\emptyset \subseteq \{\emptyset\}$
- viii. $\emptyset \in \{\emptyset\}$
- ix. $0 \mapsto a \in A \times B$
- x. $(0, a) \in A \times B$
- xi. $\{0 \mapsto a, 1 \mapsto b\} \in A \times B$
- xii. $\{0 \mapsto a, 1 \mapsto b\} \subseteq A \times B$
- xiii. $\{0 \mapsto a, 1 \mapsto b\} \in A \leftrightarrow B$
- xiv. $\{0 \mapsto a, 1 \mapsto b\} \subseteq A \leftrightarrow B$
- xv. $\{0 \mapsto a, 1 \mapsto b\} \in A \leftrightarrow B$
- xvi. $\{0 \mapsto a, 1 \mapsto b\} \in A \rightarrow B$
- xvii. $\{0 \mapsto a\} \in A \rightarrow B$
- xviii. $\{0 \mapsto a, 1 \mapsto b\} \in A \rightsquigarrow B$
- xix. $\{0 \mapsto a\} \in A \rightsquigarrow B$
- xx. $\{0 \mapsto a, 1 \mapsto a\} \in A \rightsquigarrow B$
- xxi. $\{0 \mapsto a, 1 \mapsto a\} \in A \rightsquigarrow B$
- xxii. $\{0 \mapsto a, 1 \mapsto b\} \in A \rightsquigarrow B$
- xxiii. $\{0 \mapsto a, 1 \mapsto b\} \in A \rightsquigarrow B$
- xxiv. $\{0 \mapsto a\} \in A \rightsquigarrow B$
- xxv. $\{0 \mapsto a, 1 \mapsto b\} \in A \rightsquigarrow B$
- xxvi. $\{0 \mapsto a, 1 \mapsto b\} \in A \rightsquigarrow B$
- xxvii. $\{0 \mapsto a, 1 \mapsto b\} \in A \rightsquigarrow B$

(c) Calculez les expressions suivantes (*ie*, en donnant les éléments de l'ensemble résultant)

- i. $\{x \mid x \in \mathbb{P}(A) \wedge 0 \in x\}$
- ii. $A \triangleleft D$
- iii. $\text{dom}(A \triangleleft D)$
- iv. $\text{ran}(A \triangleleft D)$
- v. $A \triangleleft D$
- vi. $D \triangleright B$
- vii. $D ; E$
- viii. $(D ; F)$
- ix. $((E ; F)^{-1} ; D)$
- x. $(\text{id}(A) ; D)$
- xi. $D \triangleleft E$
- xii. $F[\{c\}]$
- xiii. Soit $r \in 0..4 \leftrightarrow 0..4$ et $r = \{(x, y) \mid x \in 0..3 \wedge y = x + 1\}$; calculez r^* .
- xiv. $\{(x, y) \mid x \in 0..3 \wedge y \in 0..3 \wedge y = x + 1\}^+$

2. Soient les définitions suivantes:

MACHINE ExerciceMathDiscrete_1

SETS

Personne={h1,h2,h3,h4,h5,f1,f2,f3,f4,f5,f6}

CONSTANTS NombrePair,Homme,Femme,Pere,Mere

PROPERTIES

NombrePair = {x | x : 1..12 & x mod 2 = 0}
 & Homme={h1,h2,h3,h4,h5}
 & Femme={f1,f2,f3,f4,f5,f6}
 & Pere={h1|->h2,h1|->f2, h2|->h3}
 & Mere={f1|->h2,f1|->f2, f2|->h4, f4|->f6, f5|->f6}

END

- (a) Définissez l'ensemble **Facteur** qui contient les facteurs du nombre 12 (ie, x est un facteur de y ssi il existe un nombre z tel que $y = x * z$, ou, de manière équivalente, $y \bmod x = 0$).
- (b) Définissez l'ensemble **Premier** qui contient l'ensemble des nombres premiers entre 2 et 12 (ie, un nombre x est premier ssi il est supérieur à 1 et ses seuls facteurs sont 1 et x).
- (c) Définissez la constante s qui est égale à la somme des éléments de **Premier**.
- (d) Définissez l'ensemble **FacteurPremier** qui contient seulement les facteurs premiers de 12.
- (e) Définissez par compréhension l'ensemble **Papa** qui contient les personnes qui sont des pères.
- (f) Définissez l'ensemble **Papa2** qui contient les mêmes éléments que **Papa**, mais en utilisant seulement des opérations sur les relations.
- (g) Définissez par compréhension l'ensemble **Fils** qui contient les personnes qui sont des enfants de sexe masculin.

- (h) Définissez par compréhension la relation `GrandParent` qui contient les couples $x \mapsto y$ telles que x est un grand-parent de y .
- (i) Définissez la relation `GrandParent2` qui contient les mêmes éléments que `GrandParent`, mais en utilisant seulement des opérations sur les relations.
- (j) Définissez par compréhension la relation `Conjoint` qui contient des couples de personnes qui sont des conjoints. On dit que x est un conjoint de y ssi x et y ont eu un enfant ensemble.
- (k) Définissez la relation `Conjoint2` qui contient les mêmes éléments que `Conjoint`, mais en utilisant seulement des opérations sur les relations.
- (l) Définissez par compréhension la relation `ConjointAsymetrique` qui contient des couples de personnes qui sont des conjoints, mais où une paire de personnes x et y apparaît une seule fois, càd que soit le couple (x, y) , soit le couple (y, x) apparaît, mais pas les deux.
- (m) Définissez la relation `Fraterie`, dont les éléments sont des frères et sœurs (ie, des personnes qui ont un parent en commun).

3. Utilisez le fichier `prop-relation.mch` pour répondre aux questions suivantes

- (a) Examinez les définitions des propriétés dans `prop-relation.mch`. Pour chaque propriété, choisissez des valeurs pour r et vérifiez que votre relation r satisfait la propriété.
- (b) Vérifiez avec ProB les questions suivantes
 - i. Existe-t-il une relation qui n'est ni réflexive, ni irreflexive?
 - ii. Une relation asymétrique est-elle aussi une relation antisymétrique?
 - iii. Pour un ensemble fini S , est-ce que les propriétés bien fondée et acyclique sont équivalentes?
 - iv. Pour un ensemble infini S , est-ce que les propriétés bien fondée et acyclique sont équivalentes? (Vous ne pouvez tester cette question dans ProB, puisque S est infini; ProB ne peut traiter "complètement" que des ensembles finis).
 - v. Existe-t-il une relation r qui est à la fois une relation d'ordre et une relation d'équivalence?
 - vi. En général, est-ce qu'une relation d'ordre est aussi une relation d'équivalence?
 - vii. Une relation peut-elle être à la fois symétrique et antisymétrique?

4. Soit

- \mathbb{R} , l'ensemble des nombres réels.
- $\mathbb{R}^+ = \{x \mid x \in \mathbb{R} \wedge x > 0\}$

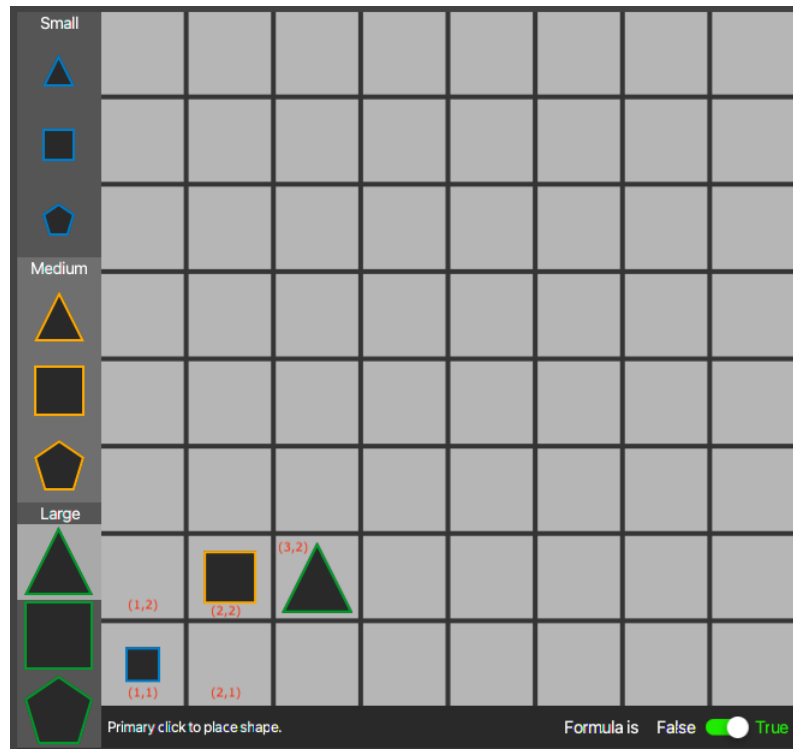
Déterminez à quelle classe chaque fonction ci-dessous appartient.

- (a) `sin`
- (b) `log`
- (c) $\lambda x.(x \in \mathbb{R} \mid x^2)$
- (d) $\{x \mapsto y \mid x \in \mathbb{R} \wedge y \in \mathbb{R} \wedge y = \sqrt{x}\}$
- (e) `succ`
- (f) $\lambda x.(x \in \mathbb{N} \mid succ(x))$

5. Considérez le monde Tarski ci-dessous et le fichier suivant qui représente ce monde en utilisant des relations.

<https://marcfrappier.espaceweb.usherbrooke.ca/mat115/exercices/chap2/q5/Tarski.mch>

Complétez les définitions demandées. Un objet est représenté par sa coordonnée dans plan, c'est-à-dire un couple (x, y) , où x est le numéro de colonne et y le numéro de ligne. La coordonnée $(0, 0)$ dénote la case située en bas à droite dans le monde. La relation *Square* contient les coordonnées des carrés du monde; les autres types d'objets sont représentés de la même manière. De même, la relation *Small* contient les objets qui sont petits.



6. Complétez la modélisation en B des prédicats de Tarski à partir du modèle ci-dessous. Codez les formules de la question 5 dans ce modèle. Vous pouvez aussi coder n'importe quelle formule du devoir 1 ou des exercices (1.7) du chapitre 1.

<https://marcfrappier.espaceweb.usherbrooke.ca/mat115/exercices/chap2/q6/q6.mch>

7. Soit les ensembles suivants:

- *Sigle* : l'ensemble des sigles de cours de l'université

$$\text{Sigle} = \{\text{MAT115}, \text{IFT187}, \dots\}$$

- *GroupeCours* : l'ensemble des groupes cours donnée à l'UdeS durant une session.

$$\text{GroupeCours} \in \text{Cours} \leftrightarrow \mathbb{N}$$

$$\text{GroupeCours} = \{\text{MAT115} \mapsto 1, \text{MAT115} \mapsto 2, \text{IFT187} \mapsto 1, \dots\}$$

- *Local* : l'ensemble des locaux de l'UdeS

$$\text{Local} = \{\text{D4-1021}, \text{D6-2034}, \text{D3-2018}, \dots\}$$

- *Etudiant* : l'ensemble des étudiants inscrits durant une session.

$$\text{Etudiant} = \{\mathbf{e1}, \mathbf{e2}, \mathbf{e3}, \dots\}$$

- *Professeur* : l'ensemble des professeurs de l'UdeS.

$$\text{Professeur} = \{\mathbf{e1}, \mathbf{e2}, \mathbf{e3}, \dots\}$$

- *Plage* : l'ensemble des plages horaires dans une semaine; on suppose que ces plages horaires sont disjointes (ex: $\mathbf{h1} = \text{LU-8h30}\hat{\text{a}}\text{9h20}$, $\mathbf{h2} = \text{LU-9h30}\hat{\text{a}}\text{10h20}$, etc).

$$\text{Plage} = \{\mathbf{h1}, \mathbf{h2}, \mathbf{h3}, \dots\}$$

Donner la classe la plus spécifique pour chacune des relations suivantes. Si la relation n'est pas une fonction, représentez les contraintes additionnelles en logique du premier ordre pour bien spécifier chaque relation.

- $r_1 \in \text{GroupeCours} \leftrightarrow \text{Local}$: Donne le local associé à un groupe cours. On suppose qu'un groupe cours se donne dans un seul local.
- $r_2 \in \text{GroupeCours} \leftrightarrow \text{Professeur}$: Donne le professeur qui enseigne à un groupe cours. Un professeur peut enseigner à plusieurs groupes cours durant une session. Tous les professeurs doivent enseigner à au moins un groupe cours dans une session.
- $r_3 \in \text{GroupeCours} \leftrightarrow \text{Etudiants}$: Donne les étudiants d'un groupe cours. Un étudiant ne peut suivre plus de 5 cours dans une session. Un étudiant doit suivre au moins un cours dans une session, vu qu'il est inscrit.
- $r_4 \in \text{Sigle} \leftrightarrow \text{Professeur}$: Donne les professeurs qui enseignent les cours durant une session. Il se peut qu'un cours ne soit pas donné.
- $r_5 \in \text{GroupeCours} \leftrightarrow \text{Plage}$: Donne la plage horaire d'un groupe cours. On suppose qu'un groupe cours se donne sur une seule plage horaire.
- $r_6 \in \text{GroupeCours} \leftrightarrow (\text{Plage} \times \text{Local})$: Donne la plage horaire et le local d'un groupe cours.
- Soit e un étudiant et $r_7 = r_3^{-1}[\{e\}]$: r_7 donne les cours (sigles) suivis par l'étudiant e dans une session.
- Soit p un professeur et $r_8 = (r_1^{-1}; r_6)[\{p\}]$. r_8 donne les plages horaires et locaux d'un professeur dans une session. Un professeur ne peut pas enseigner deux groupes cours en même temps.

Chapitre 3

Types de preuve

Dans ce chapitre, nous abordons différents types de preuves. Nous présentons tout d’abord les preuves par induction (aussi appelée *preuve par récurrence*), qui permettent de prouver une formule de la forme $\forall x \cdot x \in S \Rightarrow \mathcal{A}$, en utilisant une relation bien fondée sur S . Elle sont typiquement utilisées pour les nombres naturels, mais aussi pour les ensembles et les structures de données utilisées fréquemment en informatique, comme les arbres, les listes, ou tout autre objet, en utilisant leur structure syntaxique, qui est naturellement inductive, comme relation bien fondée.

Nous illustrons ensuite le style classique de rédaction d’une preuve en langage naturel, qui est une abstraction compacte d’un arbre de preuve tel que vu au chapitre 1. Les preuves écrites en langage naturel sont fréquemment utilisées en mathématiques. Nous présentons ensuite le style de preuve équationnel, qui consiste à formuler une suite d’équivalences, d’implications, d’égalités ou d’inclusions, en utilisant à chaque étape des lois ou les définitions des objets manipulés.

3.1 Preuve par induction

La preuve par induction est une des règles les plus utilisées en mathématique. Elle fut formalisée par Augustus De Morgan, Giuseppe Peano, Charles Sanders¹ et Richard Dedekind².

3.1.1 Preuve par induction sur les naturels

Soit \mathbb{N}_k les nombres naturels supérieurs ou égal à k , i.e., $\mathbb{N}_k = \{y \mid y \in \mathbb{N} \wedge y \geq k\}$ pour $k \in \mathbb{N}$. Soit à prouver la formule

$$\forall x \cdot x \in \mathbb{N}_k \Rightarrow \mathcal{A}$$

par induction.

1. Preuve du cas de base. On fait la preuve pour la valeur minimale k de \mathbb{N}_k (par exemple, $\min(\mathbb{N}_1) = 1$). Il faut donc prouver la formule suivante

$$\mathcal{A}[x := k]$$

¹Charles Sanders Peirce (1839–1914) est un philosophe, logicien et mathématicien américain. Dès 1886, il indique que les opérations booléennes, qui sont à la base du fonctionnement des ordinateurs modernes, pourraient être calculées par des circuits électriques.

²Julius Wilhelm Richard Dedekind (1831–1916) est un mathématicien allemand. Pionnier de l’axiomatisation de l’arithmétique, il a proposé une définition axiomatique de l’ensemble des nombres entiers ainsi qu’une construction rigoureuse des nombres réels à partir des nombres rationnels.

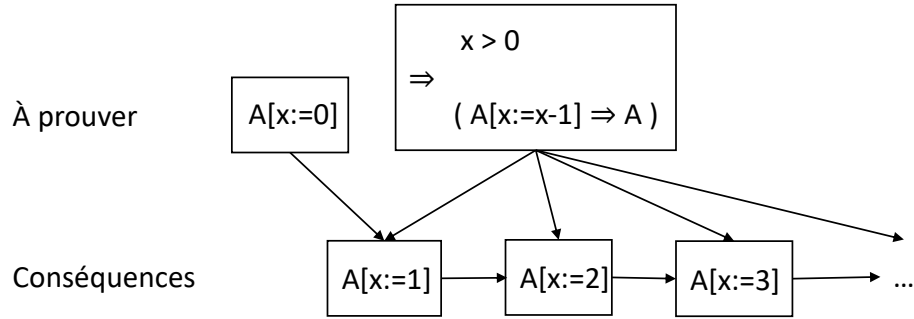


Figure 3.1: Le principe de preuve par induction

2. Preuve du cas d'induction ($x > k$), où on utilise l'hypothèse d'induction

$$\mathcal{A}[x := x - 1] \quad (\text{HI})$$

pour prouver \mathcal{A} . Pour ce faire, on cherche à ré-écrire \mathcal{A} pour faire apparaître $\mathcal{A}[x := x - 1]$ et utiliser l'hypothèse d'induction.

La figure 3.1 illustre le principe de preuve par induction.

Pour illustrer les preuves par induction, nous allons débiter par des théorèmes utilisant l'opérateur de sommation " \sum ". De manière informelle, on a

$$\begin{aligned} & \sum_{i=1}^n t \\ = & t[i := 1] + t[i := 2] + \dots + t[i := n] \end{aligned}$$

Dans une sommation $\sum_{i=m}^n t$, l'opérateur de sommation \sum déclare la variable i , et i est une variable quantifiée, exactement comme dans une quantification universelle; m est la borne inférieure de la sommation, et n est la borne supérieure. Le symbole t est un terme où la variable i apparaît; c'est le terme quantifié par la sommation. Par exemple, soit $t = 2i - 1$. On a

$$\begin{aligned} & \sum_{i=1}^n 2i - 1 \\ = & (2i - 1)[i := 1] + (2i - 1)[i := 2] + \dots + (2i - 1)[i := n] \\ = & 1 + 3 + \dots + 2n - 1 \end{aligned}$$

Voici la définition formelle d'une sommation.

$$n > m \Rightarrow \sum_{i=m}^n t = \left(\sum_{i=m}^{n-1} t \right) + (t[i := n]) \quad (3.1)$$

$$n = m \Rightarrow \sum_{i=m}^n t = (t[i := m]) \quad (3.2)$$

$$n < m \Rightarrow \sum_{i=m}^n t = 0 \quad (3.3)$$

Voici un exemple d'illustration d'une sommation.

$$\begin{aligned}
 & \sum_{i=1}^3 i^2 \\
 = & \\
 & i^2[i := 1] + i^2[i := 2] + i^2[i := 3] \\
 = & \\
 & 1^2 + 2^2 + 3^2 \\
 = & \\
 & 14
 \end{aligned}$$

Voici un premier exemple de preuve par induction. Nous allons prouver le théorème suivant par induction.

Théorème 5

$$\forall x \cdot x \in \mathbb{N} \Rightarrow \sum_{i=0}^x i = \frac{x(x+1)}{2}$$

PREUVE. Par induction. Soit

$$A \equiv \sum_{i=0}^x i = \frac{x(x+1)}{2}$$

Puisque $S = \mathbb{N}$, $min = 0$.

Cas de base $x = 0$.

Montrons

$$\mathcal{A}[x := 0] \equiv \sum_{i=0}^0 i = \frac{0(0+1)}{2}$$

$$\begin{aligned}
 & \sum_{i=0}^0 i \\
 = & \\
 & 0 \qquad \qquad \qquad \langle (3.2) \rangle \\
 = & \\
 & \frac{0(0+1)}{2} \qquad \qquad \qquad \langle \text{arithmétique} \rangle
 \end{aligned}$$

Cas $x > 0$, en utilisant l'hypothèse d'induction

$$\mathcal{A}[x := x - 1] \equiv \sum_{i=0}^{x-1} i = \frac{(x-1)((x-1)+1)}{2} = \frac{(x-1)x}{2} \quad (\text{HI}) \quad (3.4)$$

et prouvons \mathcal{A} .

$$\begin{aligned}
 & \sum_{i=0}^x i \\
 = & \qquad \qquad \langle (3.1). \text{ Cette étape permet de faire apparaître le terme de gauche de (3.4), soit } \sum_{i=0}^{x-1} i \rangle
 \end{aligned}$$

$$\begin{aligned}
& \lfloor \left(\sum_{i=0}^{x-1} i \right) \rfloor + x \\
= & \qquad \qquad \qquad \langle \text{Hypothèse d'induction (HI)} \rangle \\
& \lceil \frac{(x-1)x}{2} \rceil + x \\
= & \qquad \qquad \qquad \langle \text{arithmétique: dénominateur commun} \rangle \\
& \frac{x^2 - x}{2} + \frac{2x}{2} \\
= & \qquad \qquad \qquad \langle \text{arithmétique} \rangle \\
& \frac{x^2 - x + 2x}{2} \\
= & \qquad \qquad \qquad \langle \text{arithmétique} \rangle \\
& \frac{x^2 + x}{2} \\
= & \qquad \qquad \qquad \langle \text{arithmétique} \rangle \\
& \frac{x(x+1)}{2}
\end{aligned}$$

□

Voici un autre exemple pour \mathbb{N}_1

Théorème 6

$$\forall x \cdot x \in \mathbb{N}_1 \Rightarrow \sum_{i=1}^x (2i - 1) = x^2$$

PREUVE. Par induction. Soit

$$\mathcal{A} \equiv \sum_{i=1}^x (2i - 1) = x^2$$

Cas de base $x = 1$ (car le nombre 1 est l'élément minimal de \mathbb{N}_1). Il faut donc prouver

$$\mathcal{A}[x := 1] \equiv \sum_{i=1}^1 (2i - 1) = 1^2$$

Ce qui se prouve comme suit:

$$\begin{aligned}
& \sum_{i=1}^1 (2i - 1) \\
= & \qquad \qquad \qquad \langle (3.2) \rangle \\
& 2 - 1 \\
= & \qquad \qquad \qquad \langle \text{arithmétique} \rangle \\
& 1 \\
= & \qquad \qquad \qquad \langle \text{arithmétique} \rangle \\
& 1^2
\end{aligned}$$

Cas $x > 1$. Hypothèse d'induction:

$$A[x := x - 1] \equiv \sum_{i=1}^{x-1} (2i - 1) = (x - 1)^2 \quad (\text{HI}) \quad (3.5)$$

On prouve \mathcal{A} comme suit:

$$\begin{aligned} & \sum_{i=1}^x (2i - 1) \\ = & \quad \quad \quad \langle (3.1) \rangle \\ & \llbracket \left(\sum_{i=1}^{x-1} (2i - 1) \right) \rrbracket + (2x - 1) \\ = & \quad \quad \quad \langle \text{Hypothèse d'induction (3.5)} \rangle \\ & \llbracket (x - 1)^2 \rrbracket + (2x - 1) \\ = & \quad \quad \quad \langle \text{arithmétique} \rangle \\ & \llbracket (x^2 - 2x + 1) \rrbracket + (2x - 1) \\ = & \quad \quad \quad \langle \text{arithmétique} \rangle \\ & x^2 \end{aligned}$$

□

Voici une autre exemple à propos des exposants. Considérons les définitions suivantes de la notion d'exposant sur les nombres naturels.

$$b^0 = 1 \quad (3.6)$$

$$n \in \mathbb{N}_1 \Rightarrow b^n = b^{n-1}b \quad (3.7)$$

Théorème 7

$$\forall x, y \cdot x \in \mathbb{N} \wedge y \in \mathbb{N} \Rightarrow b^{x+y} = b^x b^y$$

Cette formule contient deux variables quantifiées. On peut réécrire cette formule à l'aide des lois (LP-37) et (LPO-26), comme suit.

$$\begin{aligned} & \forall x, y \cdot x \in \mathbb{N} \wedge y \in \mathbb{N} \Rightarrow b^{x+y} = b^x b^y \\ \Leftrightarrow & \quad \quad \quad \langle (\text{LP-37}) \rangle \\ & \forall x, y \cdot x \in \mathbb{N} \Rightarrow (y \in \mathbb{N} \Rightarrow b^{x+y} = b^x b^y) \\ \Leftrightarrow & \quad \quad \quad \langle (\text{LPO-26}) \rangle \\ & \forall x \cdot x \in \mathbb{N} \Rightarrow (\forall y \cdot y \in \mathbb{N} \Rightarrow b^{x+y} = b^x b^y) \end{aligned}$$

On peut donc faire une induction sur cette formule pour x . La quantification sur y fera partie de \mathcal{A} .

PREUVE. Par induction sur x seulement. Soit

$$\mathcal{A} \equiv \forall y \cdot y \in \mathbb{N} \Rightarrow b^{x+y} = b^x b^y$$

Cas de base $x = 0$ (car le nombre 0 est l'élément minimal de \mathbb{N}). Il faut donc prouver

$$\mathcal{A}[x := 0] \equiv \forall y \cdot y \in \mathbb{N} \Rightarrow b^{0+y} = b^0 b^y$$

Cette formule contient un quantificateur. On peut la prouver comme suit, avec les règles I_\forall (voir page 44) et I_\Rightarrow (voir page 36).

$$\frac{\frac{\frac{[y \in \mathbb{N}]^{[1]}}{\vdots}}{b^{0+y} = b^0 b^y}}{y \in \mathbb{N} \Rightarrow b^{0+y} = b^0 b^y} \text{ [I}\Rightarrow\text{]}^{[1]}}{\forall y \cdot y \in \mathbb{N} \Rightarrow b^{0+y} = b^0 b^y} \text{ I}\forall$$

Il reste donc à prouver ce qui est en rouge, que nous prouvons comme suit.

$$\begin{aligned} & b^0 b^y \\ = & & \langle (3.6) \rangle \\ & 1 b^y \\ = & & \langle \text{arithmétique} \rangle \\ & b^y \\ = & & \langle y = 0 + y \rangle \\ & b^{0+y} \end{aligned}$$

Cas $x > 0$. Hypothèse d'induction:

$$A[x := x - 1] \equiv \forall y \cdot y \in \mathbb{N} \Rightarrow b^{(x-1)+y} = b^{x-1} b^y \quad (\text{HI}) \quad (3.8)$$

Pour prouver \mathcal{A} , qui contient aussi un quantificateur comme dans le cas de base, on utilise les règles $\text{I}\forall$ et $\text{I}\Rightarrow$, on suppose $y \in \mathbb{N}$, et il reste à prouver

$$b^{x+y} = b^x b^y$$

Ce qui se prouve comme suit.

$$\begin{aligned} & b^{x+y} \\ = & & \langle \text{arithmétique: } x + y = (x - 1) + (y + 1) \rangle \\ & b^{(x-1)+(y+1)} \end{aligned}$$

On peut utiliser ici l'hypothèse d'induction (3.8). Pour ce faire, il suffit d'instancier (3.8) avec $y := y + 1$, grâce à la règle d'inférence $\text{E}\forall$ de la page 44 et la règle $\text{E}\Rightarrow$

$$\frac{\frac{y \in \mathbb{N}}{y + 1 \in \mathbb{N}} \text{ Arith.} \quad \frac{(\text{HI}) : (3.8)}{y + 1 \in \mathbb{N} \Rightarrow b^{(x-1)+(y+1)} = b^{x-1} b^{y+1}}{\text{E}\forall \text{ avec } [y := y + 1]}}{b^{(x-1)+(y+1)} = b^{x-1} b^{y+1}} \text{ [E}\Rightarrow\text{]}$$

et obtenir la déduction suivante

$$b^{(x-1)+(y+1)} = b^{x-1} b^{y+1} \quad (3.9)$$

de notre hypothèse d'induction, et de poursuivre notre preuve pour le cas $x > 0$.

$$\begin{aligned} & b^{(x-1)+(y+1)} \\ = & & \langle (3.9) \rangle \\ & b^{x-1} b^{y+1} \\ = & & \langle \text{puisque } y + 1 \in \mathbb{N}_1, (3.7) \text{ avec } n := y + 1 \text{ donne } b^{y+1} = b^y b \rangle \\ & b^{x-1} b^y b \\ = & & \langle \text{commutativité multiplication} \rangle \\ & b^{x-1} b b^y \\ = & & \langle (3.7) \text{ et hypothèse } x > 0 \rangle \\ & b^x b^y \end{aligned}$$

□

Voici un autre exemple avec les nombres de Fibonacci. Soient les définitions suivantes des nombres de Fibonacci, désignés ici par \mathcal{F}_x avec $x \in \mathbb{N}$.

$$\mathcal{F}_0 = 0 \tag{3.10}$$

$$\mathcal{F}_1 = 1 \tag{3.11}$$

$$n > 1 \Rightarrow \mathcal{F}_n = \mathcal{F}_{n-1} + \mathcal{F}_{n-2} \tag{3.12}$$

Théorème 8

$$\forall x, y \cdot x \in \mathbb{N} \wedge y \in \mathbb{N}_1 \Rightarrow \mathcal{F}_{x+y} = \mathcal{F}_{x+1}\mathcal{F}_y + \mathcal{F}_x\mathcal{F}_{y-1}$$

Comme pour le théorème précédent, nous avons deux variables quantifiées. On peut réécrire cette formule à l'aide des lois (LP-37) et (LPO-26), ce qui donne à prouver la formule suivante:

$$\forall x \cdot x \in \mathbb{N} \Rightarrow (\forall y \cdot y \in \mathbb{N}_1 \Rightarrow \mathcal{F}_{x+y} = \mathcal{F}_{x+1}\mathcal{F}_y + \mathcal{F}_x\mathcal{F}_{y-1})$$

PREUVE. Par induction. Soit

$$\mathcal{A} \equiv \forall y \cdot y \in \mathbb{N}_1 \Rightarrow \mathcal{F}_{x+y} = \mathcal{F}_{x+1}\mathcal{F}_y + \mathcal{F}_x\mathcal{F}_{y-1}$$

Cas $x = 0$ (car le nombre 0 est l'élément minimal de \mathbb{N}). Il faut donc prouver

$$\mathcal{A}[x := 0] \equiv \forall y \cdot y \in \mathbb{N}_1 \Rightarrow \mathcal{F}_{0+y} = \mathcal{F}_{0+1}\mathcal{F}_y + \mathcal{F}_0\mathcal{F}_{y-1}$$

Ce qui se prouve comme suit. Avec les règles I_\forall et I_\Rightarrow , on suppose $y \in \mathbb{N}_1$ et on prouve

$$\mathcal{F}_{0+y} = \mathcal{F}_{0+1}\mathcal{F}_y + \mathcal{F}_0\mathcal{F}_{y-1}.$$

Ce qui se prouve comme suit.

$$\begin{aligned} & \llbracket \mathcal{F}_{0+1} \rrbracket \mathcal{F}_y + \llbracket \mathcal{F}_0 \rrbracket \mathcal{F}_{y-1} && \langle (3.10) \text{ et } (3.11) \rangle \\ = & \llbracket 1 \rrbracket \mathcal{F}_y + \llbracket 0 \rrbracket \mathcal{F}_{y-1} && \langle \text{arithmétique} \rangle \\ = & \mathcal{F}_y && \langle \text{arithmétique: } y = 0 + y \rangle \\ = & \mathcal{F}_{0+y} \end{aligned}$$

Cas $x > 0$. Hypothèse d'induction:

$$\mathcal{A}[x := x - 1] \equiv \forall y \cdot y \in \mathbb{N}_1 \Rightarrow \mathcal{F}_{(x-1)+y} = \mathcal{F}_{(x-1)+1}\mathcal{F}_y + \mathcal{F}_{(x-1)}\mathcal{F}_{y-1} \quad (\text{HI}) \tag{3.13}$$

Pour prouver \mathcal{A} , on utilise les règles I_\forall et I_\Rightarrow comme suit:

$$\frac{\frac{\frac{\vdots}{\mathcal{F}_{x+y} = \mathcal{F}_{x+1}\mathcal{F}_y + \mathcal{F}_x\mathcal{F}_{y-1}}{y \in \mathbb{N}_1 \Rightarrow \mathcal{F}_{x+y} = \mathcal{F}_{x+1}\mathcal{F}_y + \mathcal{F}_x\mathcal{F}_{y-1}} [I_\Rightarrow]^{[1]}}{\forall y \cdot y \in \mathbb{N}_1 \Rightarrow \mathcal{F}_{x+y} = \mathcal{F}_{x+1}\mathcal{F}_y + \mathcal{F}_x\mathcal{F}_{y-1}} I_\forall}{[y \in \mathbb{N}_1]^{[1]}}$$

Il reste donc à prouver ce qui est en rouge:

$$\mathcal{F}_{x+y} = \mathcal{F}_{x+1}\mathcal{F}_y + \mathcal{F}_x\mathcal{F}_{y-1}.$$

à partir de l'hypothèse $y \in \mathbb{N}_1$, ce qui se prouve comme suit. Typiquement, on commence avec le terme le plus complexe et on essaie de le réécrire jusqu'à obtenir le terme le plus simple. En suivant cette stratégie, nous devrions commencer avec le terme de droite (i.e., $\mathcal{F}_{x+1}\mathcal{F}_y + \mathcal{F}_x\mathcal{F}_{y-1}$) et essayer d'y faire apparaître l'hypothèse d'induction. Toutefois, cette stratégie ne fonctionne pas ici (essayez-là pour le constater). Nous allons donc débiter avec le terme de gauche (i.e., \mathcal{F}_{x+y}) et essayer d'y faire apparaître un des termes de l'hypothèse d'induction.

$$\begin{aligned} & \mathcal{F}_{x+y} \\ = & \hspace{15em} \langle \text{arithmétique: } x + y = (x - 1) + (y + 1) \rangle \\ & \mathcal{F}_{(x-1)+(y+1)} \end{aligned}$$

Remarquons que ce terme apparaît presque dans (3.13). Pour l'obtenir, il suffit d'instancier (HI) (soit (3.13)) avec $y := y + 1$, grâce à la règle d'inférence E_{\vee} de la page 44. Puisque nous avons supposé que $y \in \mathbb{N}_1$, on a donc $y + 1 \in \mathbb{N}_1$. On peut donc utiliser la règle E_{\Rightarrow} comme suit:

$$\frac{\frac{y \in \mathbb{N}_1}{y + 1 \in \mathbb{N}_1} \text{ Arith.} \quad \frac{\text{(HI) : (3.13)}}{y + 1 \in \mathbb{N}_1 \Rightarrow \mathcal{F}_{(x-1)+(y+1)} = \mathcal{F}_{(x-1)+1}\mathcal{F}_{y+1} + \mathcal{F}_{(x-1)}\mathcal{F}_{(y+1)-1}}{E_{\vee} \text{ avec } [y := y + 1]} \quad \frac{}{E_{\Rightarrow}}}{\mathcal{F}_{(x-1)+(y+1)} = \mathcal{F}_{(x-1)+1}\mathcal{F}_{y+1} + \mathcal{F}_{(x-1)}\mathcal{F}_{(y+1)-1}}$$

et obtenir la déduction suivante, après simplification des indices.

$$\mathcal{F}_{(x-1)+(y+1)} = \mathcal{F}_x\mathcal{F}_{y+1} + \mathcal{F}_{(x-1)}\mathcal{F}_y \tag{3.14}$$

Poursuivons donc la preuve du cas d'induction, en utilisant (3.14).

$$\begin{aligned} & \llbracket \mathcal{F}_{(x-1)+(y+1)} \rrbracket \\ = & \hspace{15em} \langle (3.14) \rangle \\ & \lceil \mathcal{F}_x \llbracket \mathcal{F}_{y+1} \rrbracket + \mathcal{F}_{x-1} \mathcal{F}_y \rceil \\ = & \hspace{15em} \langle (3.12) \rangle \\ & \llbracket \mathcal{F}_x \lceil (\mathcal{F}_y + \mathcal{F}_{y-1}) \rceil \rrbracket + \mathcal{F}_{x-1} \mathcal{F}_y \\ = & \hspace{10em} \langle \text{distributivité multiplication sur addition} \rangle \\ & \lceil \mathcal{F}_x \mathcal{F}_y + \llbracket \mathcal{F}_x \mathcal{F}_{y-1} \rrbracket + \mathcal{F}_{x-1} \mathcal{F}_y \rceil \\ = & \hspace{15em} \langle \text{commutativité addition} \rangle \\ & \llbracket \mathcal{F}_x \mathcal{F}_y + \lceil \mathcal{F}_{x-1} \mathcal{F}_y \rceil + \mathcal{F}_x \mathcal{F}_{y-1} \rrbracket \\ = & \hspace{10em} \langle \text{distributivité multiplication sur addition} \rangle \\ & \lceil \llbracket (\mathcal{F}_x + \mathcal{F}_{x-1}) \rrbracket \mathcal{F}_y \rceil + \mathcal{F}_x \mathcal{F}_{y-1} \\ = & \hspace{10em} \langle \text{hyp. } x > 0, \text{ donc } x + 1 > 1, (3.12) \text{ avec } n := x + 1 \text{ donne } \mathcal{F}_{x+1} = \mathcal{F}_x + \mathcal{F}_{x-1} \rangle \\ & \lceil \mathcal{F}_{x+1} \mathcal{F}_y + \mathcal{F}_x \mathcal{F}_{y-1} \rceil \end{aligned}$$

□

3.1.2 Règle générale de preuve par induction

Voici la règle générale de preuve par induction. Soit S un ensemble et $\prec \in S \leftrightarrow S$ une relation bien fondée.

$$(\forall x \cdot x \in S \Rightarrow \mathcal{A}) \Leftrightarrow (\forall x \cdot x \in S \Rightarrow ((\forall y \cdot y \in S \wedge y \prec x \Rightarrow \mathcal{A}[x := y]) \Rightarrow \mathcal{A})) \tag{3.15}$$

Application aux nombres naturels avec $\prec := <$.

$$(\forall x \cdot x \in \mathbb{N} \Rightarrow \mathcal{A}) \Leftrightarrow (\forall x \cdot x \in \mathbb{N} \Rightarrow ((\forall y \cdot y \in \mathbb{N} \wedge y < x \Rightarrow \mathcal{A}[x := y]) \Rightarrow \mathcal{A})) \quad (3.16)$$

Application aux sous-ensembles finis d'un ensemble S avec $\prec := \subset$.

$$(\forall x \cdot x \in \mathbb{F}(S) \Rightarrow \mathcal{A}) \Leftrightarrow (\forall x \cdot x \in \mathbb{F}(S) \Rightarrow ((\forall y \cdot y \in \mathbb{F}(S) \wedge y \subset x \Rightarrow \mathcal{A}[x := y]) \Rightarrow \mathcal{A})) \quad (3.17)$$

Notons que l'on peut aussi utiliser $S \prec S' \Leftrightarrow \text{card}(S) < \text{card}(S')$ pour les ensembles. Ce ne change pas la difficulté de la preuve lorsqu'elle est faite à la main. Toutefois, les prouveurs automatisés, comme ceux du langage B, travaillent plus facilement avec \subset qu'avec $\text{card}(S) < \text{card}(S')$.

3.1.3 Application de la règle générale d'induction aux naturels

Examinons comment cette règle peut être utilisée pour \mathbb{N} . Supposons que nous voulons prouver une formule de la forme

$$\forall x \cdot x \in \mathbb{N} \Rightarrow \mathcal{A}$$

La formule (3.16) nous indique qu'on peut la prouver en utilisant son côté droit, soit

$$\forall x \cdot x \in \mathbb{N} \Rightarrow ((\forall y \cdot y \in \mathbb{N} \wedge y < x \Rightarrow \mathcal{A}[x := y]) \Rightarrow \mathcal{A})$$

Soit

$$\mathcal{B} \equiv y \in \mathbb{N} \wedge y < x$$

En utilisant la règle I_{\forall} , il reste à prouver

$$x \in \mathbb{N} \Rightarrow ((\forall y \cdot \mathcal{B} \Rightarrow \mathcal{A}[x := y]) \Rightarrow \mathcal{A})$$

En utilisant la règle d'introduction de l'implication (I_{\Rightarrow}), on pose comme hypothèse

$$x \in \mathbb{N} \quad (3.18)$$

et on doit prouver

$$(\forall y \cdot \mathcal{B} \Rightarrow \mathcal{A}[x := y]) \Rightarrow \mathcal{A} \quad (3.19)$$

En utilisant la règle d'introduction de l'implication (I_{\Rightarrow}), on pose comme hypothèse

$$\forall y \cdot \mathcal{B} \Rightarrow \mathcal{A}[x := y] \quad (\text{HIG}) \quad (3.20)$$

et on prouve

$$\mathcal{A}$$

La formule (3.20) est appelée l'*hypothèse d'induction générale*. Pour pouvoir utiliser cette hypothèse afin de prouver \mathcal{A} , il faut prouver \mathcal{B} . Pour $x = 0$, l'hypothèse d'induction ne peut pas être utilisée, car $\mathcal{B}[x := 0] \equiv y \in \mathbb{N} \wedge y < 0$ est fautive : il n'y a pas de valeur pour y qui puissent satisfaire $y \in \mathbb{N} \wedge y < 0$, le nombre 0 n'ayant pas de prédécesseur dans \mathbb{N} pour la relation " $<$ ". Il faut donc prouver la formule \mathcal{A} pour le cas $x = 0$, et on peut prouver les autres cas ($x > 0$) avec l'hypothèse d'induction. Cela justifie les deux cas que l'on retrouve dans une preuve par induction sur les naturels. Nous utilisons donc une preuve par cas (E_{\vee}) avec la formule $x = 0 \vee x > 0$, qui se déduit facilement de l'hypothèse (3.18) ($x \in \mathbb{N}$).

Voici un arbre de preuve qui illustre ces déductions. Ce qui reste à faire dans cette preuve est en rouge. Ce sont les deux étapes de la preuve par d'induction, soit le cas de base avec $x = 0$ en hypothèse, et le cas d'induction, avec $\mathcal{A}[x := x - 1]$ et $x > 0$ en hypothèse.

$$\begin{array}{c}
\frac{[x \in \mathbb{N}]^{[1]} \quad [x > 0]^{[3]} \quad \text{Arith.} \quad \frac{[\text{HIG}]^{[4]}}{\mathcal{B}[y := x - 1] \Rightarrow \mathcal{A}[x := x - 1]} \text{E}_{\forall} \ y := x - 1}{\mathcal{B}[y := x - 1]} \quad \frac{\mathcal{A}[x := x - 1]}{[\text{E}_{\Rightarrow}]} \\
\frac{[x \in \mathbb{N}]^{[1]} \quad \frac{[x = 0]^{[2]} \quad \frac{\vdots}{\mathcal{A}}}{\mathcal{A}}}{x = 0 \vee x > 0} \quad \frac{\vdots}{\mathcal{A}} \quad \frac{\vdots}{\mathcal{A}} \quad \text{[E}_{\forall}]^{[2,3]} \\
\frac{\frac{\frac{\mathcal{A}}{\text{HIG} \Rightarrow \mathcal{A}} \text{[I}_{\Rightarrow}]^{[4]}}{x \in \mathbb{N} \Rightarrow (\text{HIG} \Rightarrow \mathcal{A})} \text{[I}_{\Rightarrow}]^{[1]}}{\forall x \cdot x \in \mathbb{N} \Rightarrow (\text{HIG} \Rightarrow \mathcal{A})} \text{[I}_{\forall}]}}{\forall x \cdot x \in \mathbb{N} \Rightarrow \mathcal{A}} \text{(3.16)}
\end{array}$$

Illustrons cela sur la preuve du théorème 5.

Cas $x = 0$.

Puisque $x = 0$, on peut remplacer x par 0 dans notre formule à prouver (i.e., $\mathcal{A}[x := 0]$), il faut donc prouver

$$\sum_{i=0}^0 i = \frac{0(0+1)}{2}.$$

Ce qui se prouve comme suit:

$$\begin{array}{l}
\sum_{i=0}^0 i \\
= \hspace{15em} \langle \text{Définition de } \sum \rangle \\
0 \\
= \hspace{15em} \langle \text{arithmétique} \rangle \\
\frac{0(0+1)}{2}
\end{array}$$

Cas $x > 0$. Il faut prouver

$$\sum_{i=0}^x i = \frac{x(x+1)}{2} \quad (3.21)$$

en utilisant l'hypothèse d'induction générale (3.20). Nous l'utilisons en l'instanciant avec $y := x - 1$ avec la règle E_{\forall}

$$\frac{\left(\forall y \cdot y \in \mathbb{N} \wedge y < x \Rightarrow \sum_{i=0}^y i = \frac{y(y+1)}{2} \right)}{\left(x - 1 \in \mathbb{N} \wedge x - 1 < x \Rightarrow \sum_{i=0}^{x-1} i = \frac{(x-1)((x-1)+1)}{2} \right)} \text{[E}_{\forall}] \text{ avec } y := x - 1$$

Puisque que nous avons $x > 0$, alors $x - 1 \in \mathbb{N} \wedge x - 1 < x$ est vrai. Avec la règle E_{\Rightarrow} , on peut donc conclure

$$\sum_{i=0}^{x-1} i = \frac{(x-1)((x-1)+1)}{2} \quad (\text{HI}) \quad (3.22)$$

ce qui est notre hypothèse d'induction générale (3.20) instanciée avec $y := x - 1$, que nous identifions par (HI). Nous avons maintenant ce qu'il faut pour faire la preuve de (3.21).

$$\begin{aligned}
& \sum_{i=0}^x i \\
= & \hspace{20em} \langle (3.1) \rangle \\
& \left(\sum_{i=0}^{x-1} i \right) + x \\
= & \hspace{10em} \langle \text{Hypothèse d'induction (3.22)} \rangle \\
& \frac{(x-1)((x-1)+1)}{2} + x \\
= & \hspace{15em} \langle \text{arithmétique} \rangle \\
& \dots \\
= & \hspace{15em} \langle \text{arithmétique} \rangle \\
& \frac{x(x+1)}{2}
\end{aligned}$$

Ce qui complète la preuve. La règle de preuve donnée à la section 3.1.1 est donc une application particulière de la règle générale de preuve par induction. \square

3.1.4 Plusieurs cas de base sur les naturels

Voici un dernier théorème sur les nombres de Fibonacci. Soit

$$\phi = \frac{1 + \sqrt{5}}{2} \tag{3.23}$$

communément appelé le *nombre d'or*. Ce nombre satisfait la propriété suivante:

$$\phi^2 = \phi + 1 \tag{3.24}$$

De plus, il est relié aux nombres de Fibonacci par le théorème suivant.

Théorème 9

$$\forall x \cdot x \in \mathbb{N}_1 \Rightarrow \mathcal{F}_x \leq \phi^{x-1}$$

On peut prouver par induction ce théorème, mais on observera une particularité par rapport aux preuves précédentes : il faudra prouver deux cas de base plutôt qu'un seul.

PREUVE. Par induction. Soit

$$\mathcal{A} \equiv \mathcal{F}_x \leq \phi^{x-1}$$

Cas $x = 1$ (car le nombre 1 est l'élément minimal de \mathbb{N}_1). Il faut donc prouver

$$\mathcal{A}[x := 1] \equiv \mathcal{F}_1 \leq \phi^{1-1}$$

ce qui se prouve comme suit:

$$\begin{aligned}
& \mathcal{F}_1 \\
= & \hspace{20em} \langle (3.11) \rangle \\
& 1 \\
= & \hspace{15em} \langle \phi^0 = 1 \rangle \\
& \phi^{1-1}
\end{aligned}$$

Cas $x > 1$. Hypothèse d'induction:

$$\mathcal{A}[x := x - 1] \equiv \mathcal{F}_{x-1} \leq \phi^{x-2} \quad (\text{HI}) \quad (3.25)$$

Preuve de \mathcal{A}

$$\begin{aligned} & \mathcal{F}_x \\ = & \mathcal{F}_{x-1} + \mathcal{F}_{x-2} && \langle \text{hyp. } x > 1, (3.12) \rangle \\ \leq & \phi^{x-2} + \mathcal{F}_{x-2} && \langle (3.25) \rangle \end{aligned}$$

Nous avons instancié l'hypothèse d'induction avec $x := x - 1$, ce qui nous permet de l'appliquer au premier terme \mathcal{F}_{x-1} . Par contre, nous ne pouvons l'utiliser pour le deuxième terme \mathcal{F}_{x-2} , car $\mathcal{B}[x := x - 2]$ est fautive pour $x = 2$ dans (3.20), car nous n'avons que $x > 1$ en hypothèse. Nous allons donc faire une sous-preuve par cas, avec $x = 2 \vee x > 2$.

Cas $x = 2$. Il faut prouver

$$\mathcal{A}[x := 2] \equiv \mathcal{F}_2 \leq \phi^{2-1}$$

Ce qui se prouve comme suit:

$$\begin{aligned} & \mathcal{F}_2 \\ = & \mathcal{F}_0 + \mathcal{F}_1 && \langle (3.12) \rangle \\ = & 0 + 1 && \langle (3.10) \text{ et } (3.11) \rangle \\ < & \phi && \langle \text{par } (3.23), \phi = 1, 6180339887499 \rangle \\ = & \phi^{2-1} && \langle \text{arithmétique} \rangle \end{aligned}$$

Cas $x > 2$. On peut instancier l'hypothèse d'induction générale (3.20) avec $y := x - 2$, car $x > 2$ entraîne que $x - 2 \in \mathbb{N}_1 \wedge x - 2 < x$, ce qui donne comme deuxième application de l'hypothèse d'induction

$$\mathcal{A}[x := x - 2] \equiv \mathcal{F}_{x-2} \leq \phi^{x-3} \quad (\text{HI}) \quad (3.26)$$

On prouve \mathcal{A} comme suit.

$$\begin{aligned} & \mathcal{F}_x \\ = & \mathcal{F}_{x-1} + \mathcal{F}_{x-2} && \langle \text{hyp. } x > 2, (3.12) \rangle \\ \leq & \phi^{x-2} + \phi^{x-3} && \langle (3.25) \text{ et } (3.26) \rangle \\ = & \phi^{x-3} \phi + \phi^{x-3} && \langle \text{hyp. } x > 2, (3.7) \text{ entraîne } \phi^{x-2} = \phi^{x-3} \phi \rangle \\ = & \phi^{x-3}(\phi + 1) && \langle \text{distributivité} \rangle \\ = & \phi^{x-3} \phi^2 && \langle (3.24): \phi + 1 = \phi^2 \rangle \\ = & \phi^{x-1} && \langle \text{théorème 7} \rangle \end{aligned}$$

□

3.1.4.1 Schéma général de preuve par induction sur les naturels

On constate avec le théorème 9 que si on veut instancier l'hypothèse d'induction générale avec $x := x - 2$, il faut ajouter un cas de base avec $x := k + 1$ pour une preuve sur \mathbb{N}_k . Voici un schéma de preuve plus général sur les naturels, mais qui est moins fréquemment utilisé.

Soit la formule $\forall x \cdot x \in \mathbb{N}_k \Rightarrow \mathcal{A}$ à prouver par induction.

1. Déterminer le cas d'induction ($x > j$)
2. Prouver les cas de bases, c'est-à-dire les éléments de $k..j$; c-à-d prouver $\mathcal{A}[x := i]$ pour chaque $i \in k..j$.
3. Pour chaque $i \in k..j$, instancier l'hypothèse d'induction

$$\mathcal{A}[x := x - 1 - (i - k)]$$

et prouver \mathcal{A} pour le cas $x > j$.

3.1.5 Preuve par induction sur d'autres structures

3.1.5.1 Preuve par induction sur les ensembles

La règle générale de preuve par induction peut être utilisée pour prouver des propriétés sur les ensembles finis, lorsqu'on l'instancie avec $\prec := \subset$, ce qui donne la règle (3.17). Pour l'illustrer, nous prouvons le théorème suivant.

Théorème 10 Soit S un ensemble.

$$\forall x \cdot x \in \mathbb{F}(S) \Rightarrow \text{card}(\mathbb{P}(x)) = 2^{\text{card}(x)}$$

PREUVE. Par induction. Soit

$$\mathcal{A} \equiv \text{card}(\mathbb{P}(x)) = 2^{\text{card}(x)}$$

Cas de base : $x = \{\}$

Il faut prouver

$$\mathcal{A}[x := \{\}] \equiv \text{card}(\mathbb{P}(\{\})) = 2^{\text{card}(\{\})}$$

$$\begin{aligned} & \text{card}(\mathbb{P}(\{\})) \\ = & \\ & \text{card}(\{\{\}\}) \\ = & \\ & 1 \\ = & \\ & 2^0 \\ = & \\ & 2^{\text{card}(\{\})} \end{aligned}$$

Cas d'induction : $x \neq \{\}$

On pose comme hypothèse d'induction l'hypothèse d'induction générale (HIG)

$$\forall y \cdot y \in \mathbb{P}(S) \wedge y \subset x \Rightarrow \mathcal{A}[x := y] \quad (3.27)$$

Puisque $x \neq \{\}$, soit z un élément quelconque de x . Soit

$$x_0 = x - \{z\}.$$

On a donc $x_0 \subset x$. Puisque $x_0 \subset x$, on peut instancier (3.27) avec $y := x_0$, ce qui donne notre hypothèse d'induction instanciée.

$$\mathcal{A}[x := x_0] \equiv \text{card}(\mathbb{P}(x_0)) = 2^{\text{card}(x_0)} \quad (\text{HI})$$

Soit T l'ensemble des sous-ensembles de x qui ne contiennent pas z .

$$T = \{y \mid y \subseteq x \wedge z \notin y\} = \mathbb{P}(x_0) \quad (3.28)$$

Soit U l'ensemble des sous-ensembles de x qui contiennent z .

$$U = \{y \mid y \subseteq x \wedge z \in y\}$$

Par définition de T et U , on a

$$\mathbb{P}(x) = T \cup U \quad (3.29)$$

et

$$T \cap U = \{\} \quad (3.30)$$

On note que

$$\text{card}(x_0) = \text{card}(x) - 1 \quad (3.31)$$

Considérons les fonctions f et g suivantes.

$$f \in T \rightarrow U \wedge g \in U \rightarrow T$$

$$f(t) = t \cup \{z\} \wedge g(u) = u - \{z\}$$

Puisque $f ; g = \text{id}(T)$ et $g ; f = \text{id}(U)$, on a que $f \in T \rightsquigarrow U$ et $g = f^{-1}$ (vous devrez prouver cette loi en devoir). La loi suivante établit un lien entre bijection et cardinalité.

$$(\exists f \cdot f \in A \rightsquigarrow B) \Rightarrow \text{card}(A) = \text{card}(B) \quad (3.32)$$

Donc,

$$\text{card}(T) = \text{card}(U) \quad (3.33)$$

Finalement, la loi suivante sera utile pour calculer la cardinalité d'une union.

$$\text{card}(A \cup B) = \text{card}(A) + \text{card}(B) - \text{card}(A \cap B) \quad (3.34)$$

Nous avons maintenant tout ce qu'il faut pour prouver le cas d'induction.

$$\begin{aligned}
& \text{card}(\mathbb{P}(x)) && \langle (3.29) \rangle \\
= & \text{card}(T \cup U) && \langle (3.34) \rangle \\
= & \text{card}(T) + \text{card}(U) - \text{card}(T \cap U) && \langle (3.30) \rangle \\
= & \text{card}(T) + \text{card}(U) - \text{card}(\{\}) && \langle \text{card}(\{\}) = 0 \rangle \\
= & \text{card}(T) + \text{card}(U) && \langle (3.33) \rangle \\
= & \text{card}(T) + \text{card}(T) && \langle \text{arithmétique} \rangle \\
= & 2 \text{card}(T) && \langle (3.28) \rangle \\
= & 2 \text{card}(\mathbb{P}(x_0)) && \langle (H1) \rangle \\
= & 2 \cdot 2^{\text{card}(x_0)} && \langle (3.31) \rangle \\
= & 2 \cdot 2^{\text{card}(x)-1} && \langle \text{Théorème 7} \rangle \\
= & 2^{\text{card}(x)} &&
\end{aligned}$$

□

3.1.5.2 Preuve par induction sur les arbres

On peut aussi utiliser la preuve par induction pour prouver des propriétés sur des structures définies inductivement. Considérez les définitions suivantes de l'ensemble \mathcal{T} des arbres binaires. Soit E un ensemble (habituellement appelé l'ensembles des étiquettes des noeuds de l'arbre).

1. $\emptyset \in \mathcal{T}$ est un arbre binaire (il dénote l'arbre vide).
2. Si $e \in E$, $g \in \mathcal{T}$ et $d \in \mathcal{T}$, alors $\langle e, g, d \rangle \in \mathcal{T}$ est un arbre d'étiquette e , avec g comme sous-arbre de gauche et d comme sous-arbre de droite.

La hauteur $h(t)$ d'un arbre t est définie comme suit.

$$h(\emptyset) = 0 \tag{3.35}$$

$$h(\langle e, g, d \rangle) = 1 + \max(\{h(g), h(d)\}) \tag{3.36}$$

On définit la relation $<$ sur les arbres, où $t_1 < t_2$ ssi t_1 est un sous-arbre propre de t_2 , i.e.,

$$g < \langle e, g, d \rangle \wedge d < \langle e, g, d \rangle \tag{3.37}$$

Finalement, le nombre de noeuds $\text{card}(t)$ d'un arbre t est défini comme suit:

$$\text{card}(\emptyset) = 0 \tag{3.38}$$

$$\text{card}(\langle e, g, d \rangle) = \text{card}(g) + \text{card}(d) + 1 \tag{3.39}$$

Théorème 11

$$\forall x \cdot x \in \mathcal{T} \Rightarrow \text{card}(x) \leq 2^{h(x)} - 1$$

PREUVE. Par induction. Soit

$$\mathcal{A} \equiv \text{card}(x) = 2^{h(x)} - 1$$

Cas de base : $x = \emptyset$

$$\text{card}(\emptyset) = 0 = 2^0 - 1 = 2^{h(\emptyset)} - 1$$

Cas d'induction: $x = \langle e, g, d \rangle$. On pose comme hypothèse d'induction

$$\forall y \cdot y \in \mathcal{T} \wedge y < x \Rightarrow \mathcal{A}[x := y] \quad (\text{HI}) \quad (3.40)$$

Puisque $g < x$ et $d < x$, on peut instancier (3.40) comme suit

$$\mathcal{A}[x := g] \equiv \text{card}(g) \leq 2^{h(g)} - 1 \quad (3.41)$$

$$\mathcal{A}[x := d] \equiv \text{card}(d) \leq 2^{h(d)} - 1 \quad (3.42)$$

$$\begin{aligned} & \text{card}(\langle e, g, d \rangle) \\ = & \text{card}(g) + \text{card}(d) + 1 && \langle (3.39) \rangle \\ \leq & 2^{h(g)} - 1 + 2^{h(d)} - 1 + 1 && \langle (3.41), (3.42) \rangle \\ \leq & 2^i - 1 + 2^i - 1 + 1 && \langle \text{soit } i = \max(\{h(d), h(g)\}) \rangle \\ = & 2^{h(x)-1} - 1 + 2^{h(x)-1} - 1 + 1 && \langle i = h(x) - 1 \rangle \\ = & 2(2^{h(x)-1}) - 1 && \langle \text{Arithmétique} \rangle \\ = & 2^{h(x)} - 1 && \langle \text{Théorème 7} \rangle \end{aligned}$$

□

Comme pour les ensembles, on peut utiliser la relation bien-fondée $x \prec y \Leftrightarrow h(x) < h(y)$ sur les arbres. Toutefois, comme indiqué pour les ensembles, la relation x est un sous-arbre propre de y est plus facile à utiliser avec les prouveurs automatisés.

3.1.6 Preuve de la règle générale de preuve par induction

Rappelons la règle générale de preuve par induction (3.15). Soit S un ensemble et $\prec \in S \leftrightarrow S$ une relation bien fondée.

$$(\forall x \cdot x \in S \Rightarrow \mathcal{A}) \Leftrightarrow (\forall x \cdot x \in S \Rightarrow ((\forall y \cdot y \in S \wedge y \prec x \Rightarrow \mathcal{A}[x := y]) \Rightarrow \mathcal{A})) \quad (3.15)$$

Cette formule est de la forme $\mathcal{C} \Leftrightarrow \mathcal{D}$. De par la règle d'inférence I_{\Leftrightarrow} , il faut prouver $\mathcal{C} \Rightarrow \mathcal{D}$ et $\mathcal{D} \Rightarrow \mathcal{C}$. La preuve de $\mathcal{C} \Rightarrow \mathcal{D}$ est triviale, puisque \mathcal{A} est en hypothèse. La preuve de $\mathcal{D} \Rightarrow \mathcal{C}$ représente la principale difficulté. La preuve se fait par contradiction comme suit. Soit

$$\mathcal{B} \equiv \forall y \cdot y \in S \wedge y \prec x \Rightarrow \mathcal{A}[x := y]$$

$$\begin{aligned}\mathcal{C} &\equiv \forall x \cdot x \in S \Rightarrow \mathcal{A} \\ \mathcal{D} &\equiv \forall x \cdot x \in S \Rightarrow (\mathcal{B} \Rightarrow \mathcal{A})\end{aligned}$$

La preuve par contradiction a la structure suivante:

$$\frac{[\mathcal{D}]^{[1]} \quad [\neg \mathcal{C}]^{[2]}}{\vdots} \frac{\perp \quad [\text{ConDict}]^{[2]}}{\mathcal{C}} \frac{[\text{I}\Rightarrow]^{[1]}}{\mathcal{D} \Rightarrow \mathcal{C}}$$

Ré-écrivons $\neg \mathcal{C}$

$$\begin{aligned} &\neg \mathcal{C} \\ \Leftrightarrow &\exists x \cdot x \in S \wedge \neg \mathcal{A} \end{aligned} \quad \langle \text{(LPO-11)}, \text{(LP-25)} \rangle$$

De $\neg \mathcal{C}$, on peut déduire qu'il existe des valeurs de x pour lesquelles \mathcal{A} est fausse. Soit T l'ensemble de ces valeurs.

$$T = \{x \mid x \in S \wedge \neg \mathcal{A}\}$$

Puisque T est un sous-ensemble non-vide de S et que \prec est bien fondée, alors, par définition de relation bien fondée, il existe au moins une valeur minimale x_0 dans T et $\neg \mathcal{A}[x := x_0]$. La preuve par contradiction se fera avec la formule $\mathcal{A}[x := x_0]$.

$$\frac{\frac{[\neg \mathcal{C}]^{[1]}}{\vdots} \quad \frac{[\mathcal{D}]^{[2]} \quad [\neg \mathcal{C}]^{[1]}}{\vdots}}{\neg \mathcal{A}[x := x_0] \quad \mathcal{A}[x := x_0]}}{\frac{\perp \quad [\text{ConDict}]^{[1]}}{\mathcal{C}} \quad [\text{I}\Rightarrow]^{[2]}}{\mathcal{D} \Rightarrow \mathcal{C}}}$$

Voici l'arbre de preuve (simplifié) de $\neg \mathcal{A}[x := x_0]$.

$$\frac{\frac{\frac{\neg \mathcal{C}}{\exists x \cdot x \in S \wedge \neg \mathcal{A}} \text{ (LPO-11), (LP-25)}}{T \neq \{\}} \text{ définition de } T}}{\frac{x_0 \in \min(T)}{\neg \mathcal{A}[x := x_0]} \text{ définition de } T} \prec \text{ est bien fondée}$$

Voici l'arbre de preuve (simplifié) de $\mathcal{A}[x := x_0]$, afin de contredire $\neg \mathcal{A}[x := x_0]$.

$$\frac{\frac{\text{Voir arbre } \mathcal{P}_1}{\mathcal{B}[x := x_0] \Rightarrow \mathcal{A}[x := x_0]} \quad \frac{\text{Voir arbre } \mathcal{P}_2}{\mathcal{B}[x := x_0]}}{\mathcal{A}[x := x_0]}$$

Arbre de preuve \mathcal{P}_1

$$\frac{\frac{\frac{\neg \mathcal{C}}{T \neq \{\}}}{x_0 \in \min(T)} \quad \frac{\mathcal{D}}{\mathcal{D}[x := x_0]} \text{ [E}_\forall\text{]}}{\frac{x_0 \in S}{x_0 \in S \Rightarrow (\mathcal{B}[x := x_0] \Rightarrow \mathcal{A}[x := x_0])}}{\mathcal{B}[x := x_0] \Rightarrow \mathcal{A}[x := x_0]}$$

Arbre de preuve \mathcal{P}_2

$$\frac{\frac{\frac{\neg \mathcal{C}}{T \neq \{\}} \text{ déf. } \mathcal{C} \text{ et } T}{x_0 \in \min(T)} \prec \text{ bien fondée} \quad \frac{\frac{[y \in S \wedge y \prec x_0]^{[1]}}{y \prec x_0} [E_{\wedge 2}] \quad \frac{[\neg \mathcal{A}[x := y]]^{[2]}}{y \in T} \text{ déf. } T}{x_0 \notin \min(T)} \text{ déf. } T}{\perp} [E_{\neg}] \quad \frac{\perp}{\mathcal{A}[x := y]} [I_{\neg}]^{[2]} \quad \frac{\frac{y \in S \wedge y \prec x_0 \Rightarrow \mathcal{A}[x := y]}{\forall y \cdot y \in S \wedge y \prec x_0 \Rightarrow \mathcal{A}[x := y]} [I_{\Rightarrow}]^{[1]} \quad \frac{\forall y \cdot y \in S \wedge y \prec x_0 \Rightarrow \mathcal{A}[x := y]}{\mathcal{B}[x := x_0]} [I_{\forall}] \text{ déf. } \mathcal{B}}{\mathcal{B}[x := x_0]} [I_{\Rightarrow}]^{[1]} [I_{\forall}] \text{ déf. } \mathcal{B}$$

□

3.2 Exprimer une preuve en langage naturel

Les arbres de preuves sont très utiles pour comprendre la structure d'une preuve et la justifier, mais ils sont pénibles à écrire, tant à la main qu'en utilisant un traitement de texte. Les mathématiciens utilisent généralement le langage naturel pour décrire un arbre de preuve. Voici les différents cas de figure pour écrire une preuve "en français", tout en étant très rigoureux.

3.2.1 $\mathcal{A} \Rightarrow \mathcal{B}$ - Implication

Soit à prouver une formule de la forme

$$\mathcal{A} \Rightarrow \mathcal{B}$$

On décrira cette preuve comme suit:

Supposons \mathcal{A} et montrons \mathcal{B} (*comment prouver \mathcal{B}*) ...

Cela découle de la règle de preuve I_{\Rightarrow} .

3.2.2 $\mathcal{A} \Leftrightarrow \mathcal{B}$ - Équivalence

Soit à prouver une formule de la forme

$$\mathcal{A} \Leftrightarrow \mathcal{B}$$

On décrira cette preuve comme suit:

Montrons $\mathcal{A} \Rightarrow \mathcal{B}$ et ensuite montrons $\mathcal{B} \Rightarrow \mathcal{A}$

Cela découle de la règle de preuve I_{\Leftrightarrow} .

3.2.3 $\mathcal{A} \wedge \mathcal{B}$ - Conjonction

Soit à prouver une formule de la forme

$$\mathcal{A} \wedge \mathcal{B}$$

On décrira cette preuve comme suit:

Montrons \mathcal{A} et montrons ensuite \mathcal{B} .

... (*comment prouver \mathcal{A}*) ...

... (*comment prouver \mathcal{B}*) ...

Cela découle de la règle de preuve I_{\wedge} .

3.2.4 Preuve par contradiction

Soit à prouver une formule \mathcal{A} (ou bien $\neg\mathcal{A}$). On décrira cette preuve comme suit:

Preuve par contradiction. Supposons $\neg\mathcal{A}$ (ou bien \mathcal{A}). ... (*comment obtenir une contradiction*) ...

Cela découle des règles de preuve I_{\neg} et E_{\neg} , et la contradiction se prouve avec la règle I_{\perp} .

3.2.5 Preuve par contraposition (contraposée)

Soit à prouver une formule $\mathcal{A} \Rightarrow \mathcal{B}$. On décrira cette preuve comme suit:

Preuve par contraposition. Supposons $\neg\mathcal{B}$. Montrons $\neg\mathcal{A}$ (*comment prouver $\neg\mathcal{A}$*) ...

Cela découle de la loi (LP-26).

3.2.6 Preuve par cas

Soit à prouver une formule \mathcal{C} en utilisant la règle E_{\vee} sur une formule $\mathcal{A} \vee \mathcal{B}$ disponible en hypothèse, ou bien déduite des hypothèses. On décrira cette preuve comme suit:

Preuve de \mathcal{C} par cas sur $\mathcal{A} \vee \mathcal{B}$.
Cas \mathcal{A} (*comment prouver \mathcal{C}*) ...
Cas \mathcal{B} (*comment prouver \mathcal{C}*) ...

On utilise souvent la loi du tiers exclu (LP-20) pour distinguer les cas.

3.2.7 \forall - Quantification universelle

Soit à prouver une formule de la forme

$$\forall x \cdot \mathcal{A}$$

On décrira cette preuve comme suit:

Soit x ... (*comment prouver \mathcal{A}*) ...

Tel qu'indiqué dans la règle de preuve I_{\forall} (voir page 44), la preuve de \mathcal{A} ne doit rien supposer à propos de x ; toutes les hypothèses qui mentionnent x proviendront de la preuve de \mathcal{A} et seront déchargées.

Les quantifications universelles sont généralement de la forme

$$\forall x \cdot x \in S \Rightarrow \mathcal{A}$$

On décrira cette preuve comme suit:

Soit $x \in S$... (*comment prouver \mathcal{A}*) ...

Cela se justifie par la règle d'introduction de l'implication I_{\Rightarrow} qui permet de mettre $x \in S$ en hypothèse, et de décharger cette hypothèse. Il y a parfois plusieurs variables quantifiées, par exemple,

$$\forall(x, y, z) \cdot x \in S \wedge y \in S \wedge z \in S \Rightarrow \mathcal{A}$$

On décrira cette preuve comme suit:

Soit $x, y, z \in S$... (*comment prouver \mathcal{A}*) ...

3.2.8 \exists - Quantification existentielle

Soit à prouver une formule de la forme

$$\exists x \cdot \mathcal{A}$$

On décrira cette preuve comme suit:

Montrons $\mathcal{A}[x := t]$ (*comment prouver $\mathcal{A}[x := t]$*) ...

où t est un terme représentant une valeur de x . Tel qu'indiqué dans la règle de preuve I_{\exists} (voir page 44), la preuve de $\mathcal{A}[x := t]$ ne doit rien supposer à propos de x . Aucune hypothèse de la preuve de $\mathcal{A}[x := t]$ ne peut mentionner x ; les hypothèses doivent porter sur t . Entre autres, t ne peut contenir x .

3.2.9 Illustration d'une preuve en langage naturel

Voici un théorème à prouver.

Théorème 12 Soit A, B des ensembles.

$$f \in A \rightarrow B \wedge g \in B \rightarrow A \wedge f ; g = \text{id}(A) \wedge g ; f = \text{id}(B) \quad (3.43)$$

$$\Rightarrow \quad (3.44)$$

$$f \in A \rightsquigarrow B \quad (3.45)$$

Nous utiliserons les définitions suivantes d'injection et de surjection, pour simplifier les preuves et pour illustrer différents cas de figure.

$$f \in A \rightsquigarrow B \Leftrightarrow \text{dom}(f) = A \wedge \forall(x, y) \cdot (x \in A \wedge y \in A \Rightarrow (f(x) = f(y) \Rightarrow x = y)) \quad (3.46)$$

$$f \in A \rightarrow B \Leftrightarrow \forall x \cdot (x \in B \Rightarrow \exists y \cdot y \in A \wedge f(y) = x) \quad (3.47)$$

Voici les arbres de preuves de ce théorème.

Soit $\mathcal{C} \equiv (3.43)$.

$$\frac{\frac{\mathcal{P}_1}{f \in A \rightsquigarrow B} \quad \frac{\mathcal{P}_2}{f \in A \rightarrow B}}{\frac{f \in A \rightsquigarrow B}{\mathcal{C} \Rightarrow f \in A \rightsquigarrow B}} \text{ def. } \rightsquigarrow$$

Voici l'arbre de preuve \mathcal{P}_1 .

$$\frac{\frac{\frac{\frac{[x \in A \wedge y \in A]^{[1]} \quad [f(x) = f(y)]^{[2]} \quad [\mathcal{C}]^{[3]}}{\vdots}}{x = y}}{f(x) = f(y) \Rightarrow x = y}^{[I_{\Rightarrow}]^{[2]}}}{x \in A \wedge y \in A \Rightarrow (f(x) = f(y) \Rightarrow x = y)}^{[I_{\Rightarrow}]^{[1]}}}{\forall(x, y) \cdot (x \in A \wedge y \in A \Rightarrow (f(x) = f(y) \Rightarrow x = y))}^{[I_{\forall}]^{(3.46)}}}{f \in A \rightsquigarrow B} \mathcal{P}_1$$

Voici l'arbre de preuve \mathcal{P}_2 .

$$\begin{array}{c}
\frac{\dots}{\dots \in A} \quad \frac{\dots}{f(\dots) = x} \\
\frac{\dots \in A \wedge f(\dots) = x}{\dots \in A \wedge f(\dots) = x} \text{ [I}\wedge\text{]} \\
\frac{\dots \in A \wedge f(\dots) = x}{\exists y \cdot y \in A \wedge f(y) = x} \text{ I}\exists[y := \dots]\text{]} \\
\frac{\exists y \cdot y \in A \wedge f(y) = x}{x \in B \Rightarrow \exists y \cdot y \in A \wedge f(y) = x} \text{ [I}\Rightarrow\text{]} \\
\frac{x \in B \Rightarrow \exists y \cdot y \in A \wedge f(y) = x}{\forall x \cdot x \in B \Rightarrow \exists y \cdot y \in A \wedge f(y) = x} \text{ [I}\forall\text{]} \\
\frac{\forall x \cdot x \in B \Rightarrow \exists y \cdot y \in A \wedge f(y) = x}{f \in A \rightarrow B} \text{ (3.47)} \\
\frac{f \in A \rightarrow B}{\mathcal{P}_2}
\end{array}$$

Cette preuve se décrit en langue naturelle comme suit. Elle sera à compléter en devoir ou en exercices, d'où la présence de parties à compléter, identifiées par les "...".

Supposons (3.43). Montrons $f \in A \rightarrow B$ à partir de (3.46) et (3.47).

Preuve du côté droit de (3.46). Puisque $f \in A \rightarrow B$, alors $\text{dom}(f) = A$ par définition de $A \rightarrow B$. Soit $x, y \in A$ et supposons $f(x) = f(y)$, montrons $x = y$.

$$\begin{array}{l}
f(x) = f(y) \\
\Rightarrow \langle \dots \rangle \\
\dots \\
\Rightarrow \langle \dots \rangle \\
x = y
\end{array}$$

Preuve du côté droit de (3.47). Soit $x \in B$. Posons $y := \dots$ et montrons $y \in A \wedge f(y) = x$.

$$\begin{array}{l}
(3.43) \\
\Rightarrow \langle \dots \rangle \\
\dots \\
\Rightarrow \langle \dots \rangle \\
\dots \in A \\
(3.43) \\
\Rightarrow \langle \dots \rangle \\
\dots \\
\Rightarrow \langle \dots \rangle \\
f(\dots) = x
\end{array}$$

□

3.3 Preuve dans le style équationnel

Plusieurs théorèmes se prouvent en utilisant simplement les définitions et les lois de la logique. Par exemple, un théorème de la forme $n_1 = n_2$, ou bien $n_1 \leq n_2$, se prouve souvent facilement par une suite d'égalités ou d'inégalités où on applique les lois de l'arithmétique, dans le style suivant:

$$\begin{array}{l}
n_1 \\
= \\
\dots \\
\dots \quad \text{ou bien} \\
= \\
n_2
\end{array}
\qquad
\begin{array}{l}
n_1 \\
= \\
\dots \\
\leq \\
\dots \\
= \\
n_2
\end{array}$$

La même stratégie s'applique pour des égalités ou des inclusions sur les ensembles, par exemple $E_1 = E_2$ ou $E_1 \subseteq E_2$. Une stratégie de preuve typique pour les mathématiques discrètes consiste à ré-écrire l'ensemble E_1 sous forme d'un ensemble par compréhension en utilisant les définitions des opérations apparaissant dans E_1 et E_2 . Voici un premier exemple de théorème à prouver en utilisant les définitions.

Théorème 13 Soit $r_1, r_2, r_3 \in S \leftrightarrow S$ des relations sur un ensemble S .

$$r_1 ; (r_2 \cap r_3) \subseteq (r_1 ; r_2) \cap (r_1 ; r_3) \quad (3.48)$$

PREUVE.

Montrons que la partie de gauche est incluse dans la partie de droite.

$$\begin{aligned} & \llbracket r_1 ; (r_2 \cap r_3) \rrbracket \\ = & \llbracket \{x \mapsto y \mid \exists z \cdot x \mapsto z \in r_1 \wedge z \mapsto y \in \llbracket r_2 \cap r_3 \rrbracket\} \rrbracket \end{aligned} \quad \langle \text{Déf. " ;" } \rangle$$

Dans cette étape, nous avons simplement appliqué la définition de ";" telle que donnée au tableau 2.7. La prochaine étape consiste à ré-écrire $r_2 \cap r_3$ en utilisant la définition de " \cap " du tableau 2.5.

$$\begin{aligned} & = \llbracket \{x \mapsto y \mid \exists z \cdot x \mapsto z \in r_1 \wedge \llbracket z \mapsto y \in \llbracket \{z \mapsto y \mid z \mapsto y \in r_2 \wedge z \mapsto y \in r_3\} \rrbracket \rrbracket \rrbracket \quad \langle \text{déf. } \cap \rangle \\ = & \llbracket \{x \mapsto y \mid \exists z \cdot x \mapsto z \in r_1 \wedge \llbracket z \mapsto y \in r_2 \wedge z \mapsto y \in r_3 \rrbracket \rrbracket \rrbracket \quad \langle \text{(LE-5)} \rangle \end{aligned}$$

Ces deux étapes survenant fréquemment et étant particulièrement fastidieuses, nous les combinerons en une seule étape dans les preuves. Comme (LE-5) est toujours utilisé dans ces étapes où on remplace un opérateur par sa définition à l'intérieur d'un autre ensemble, on peut l'omettre dans la justification, par souci de simplicité.

$$= \llbracket \{x \mapsto y \mid \exists z \cdot \llbracket x \mapsto z \in r_1 \rrbracket \wedge z \mapsto y \in r_2 \wedge z \mapsto y \in r_3 \rrbracket \rrbracket \quad \langle \text{déf. } \cap \rangle$$

Poursuivons.

$$\begin{aligned} & = \llbracket \{x \mapsto y \mid \exists z \cdot \llbracket x \mapsto z \in r_1 \wedge \llbracket x \mapsto z \in r_1 \rrbracket \wedge z \mapsto y \in r_2 \wedge z \mapsto y \in r_3 \rrbracket \rrbracket \quad \langle \text{(LP-5)} \rangle \\ = & \llbracket \{x \mapsto y \mid \llbracket \exists z \cdot \llbracket x \mapsto z \in r_1 \wedge \llbracket z \mapsto y \in r_2 \wedge x \mapsto z \in r_1 \rrbracket \wedge z \mapsto y \in r_3 \rrbracket \rrbracket \rrbracket \quad \langle \text{(LP-7)} \rangle \\ \subseteq & \llbracket \{x \mapsto y \mid \llbracket \exists z \cdot \llbracket x \mapsto z \in r_1 \wedge \llbracket z \mapsto y \in r_2 \wedge x \mapsto z \in r_1 \rrbracket \wedge z \mapsto y \in r_3 \rrbracket \rrbracket \rrbracket \quad \langle \text{(LPO-19),(LE-3)} \rangle \\ = & \llbracket \{x \mapsto y \mid \llbracket \llbracket \exists z \cdot \llbracket x \mapsto z \in r_1 \wedge z \mapsto y \in r_2 \rrbracket \rrbracket \wedge \llbracket \exists z \cdot \llbracket x \mapsto z \in r_1 \wedge z \mapsto y \in r_3 \rrbracket \rrbracket \rrbracket \rrbracket \quad \langle \text{déf. " ;" } \rangle \\ = & \llbracket \{x \mapsto y \mid \llbracket x \mapsto y \in r_1 ; r_2 \rrbracket \wedge \llbracket x \mapsto y \in r_1 ; r_3 \rrbracket \rrbracket \rrbracket \quad \langle \text{déf. " ;" } \rangle \\ = & \llbracket r_1 ; r_2 \cap r_1 ; r_3 \rrbracket \quad \langle \text{déf. } \cap \rangle \end{aligned}$$

□

On peut aussi faire cette preuve en utilisant la définition de l'inclusion, c'est-à-dire

$$S \subseteq T \Leftrightarrow \forall x \cdot (x \in S \Rightarrow x \in T) \quad (3.49)$$

Pour prouver l'inclusion $S \subseteq T$, il suffit de prouver $x \in S \Rightarrow x \in T$ pour un x quelconque. Voici la preuve du théorème 13 en utilisant (3.49).

$$\begin{aligned}
& \llbracket x \mapsto y \in r_1 ; (r_2 \cap r_3) \rrbracket && \langle \text{Déf. " ;" } \rangle \\
\Leftrightarrow & \llbracket \exists z \cdot x \mapsto z \in r_1 \wedge \llbracket z \mapsto y \in (r_2 \cap r_3) \rrbracket \rrbracket && \langle \text{déf. } \cap \rangle \\
\Leftrightarrow & \exists z \cdot \llbracket x \mapsto z \in r_1 \rrbracket \wedge \llbracket z \mapsto y \in r_2 \wedge z \mapsto y \in r_3 \rrbracket && \langle \text{(LP-5)} \rangle \\
\Leftrightarrow & \exists z \cdot \llbracket x \mapsto z \in r_1 \wedge \llbracket x \mapsto z \in r_1 \rrbracket \wedge z \mapsto y \in r_2 \rrbracket \wedge z \mapsto y \in r_3 && \langle \text{(LP-7)} \rangle \\
\Leftrightarrow & \llbracket \exists z \rrbracket \cdot x \mapsto z \in r_1 \wedge \llbracket z \mapsto y \in r_2 \wedge x \mapsto z \in r_1 \rrbracket \wedge z \mapsto y \in r_3 && \langle \text{(LPO-19)} \rangle \\
\Rightarrow & \llbracket (\llbracket \exists z \rrbracket \cdot x \mapsto z \in r_1 \wedge z \mapsto y \in r_2) \rrbracket \wedge \llbracket (\llbracket \exists \rrbracket z \cdot x \mapsto z \in r_1 \wedge z \mapsto y \in r_3) \rrbracket \rrbracket && \langle \text{déf. " ;" } \rangle \\
\Leftrightarrow & \llbracket x \mapsto y \in r_1 ; r_2 \rrbracket \wedge \llbracket x \mapsto y \in r_1 ; r_3 \rrbracket && \langle \text{déf. } \cap \rangle \\
\Leftrightarrow & \llbracket x \mapsto y \in r_1 ; r_2 \cap r_1 ; r_3 \rrbracket && \langle \text{déf. } \cap \rangle
\end{aligned}$$

Cette preuve comprend le même nombre d'étapes que la preuve précédente, mais les lignes sont plus courtes, vu qu'on utilise seulement les prédicat définissant les ensembles de la preuve précédente; la partie $\{x \mapsto y \mid \dots\}$ est absente.

Voici un autre exemple utilisant l'opérateur ran , qui extrait un ensemble d'une relation. Voici un premier exemple de théorème à prouver en utilisant les définitions.

Théorème 14 Soit $r_1, r_2 \in S \leftrightarrow S$ des relations sur un ensemble S .

$$\text{ran}(r_1 ; r_2) \subseteq \text{ran}(r_2) \quad (3.50)$$

PREUVE.

Montrons que la partie de gauche est incluse dans la partie de droite.

$$\begin{aligned}
& \llbracket \text{ran}(r_1 ; r_2) \rrbracket && \langle \text{Déf. } \text{ran} \rangle \\
= & \llbracket \{x \mid \exists y \cdot \llbracket y \mapsto x \in r_1 ; r_2 \rrbracket\} \rrbracket && \langle \text{Déf. " ;" } \rangle \\
= & \{x \mid \exists y \cdot \llbracket \exists z \cdot \llbracket y \mapsto z \in r_1 \rrbracket \wedge z \mapsto x \in r_2 \rrbracket \rrbracket && \langle E_{\wedge 1}, \text{(LE-3)} \rangle \\
\subseteq & \{x \mid \llbracket \exists y \rrbracket \cdot \exists z \cdot z \mapsto x \in r_2\} && \langle \text{(LPO-32)} \rangle \\
= & \{x \mid \exists z \cdot z \mapsto x \in r_2\} && \langle \text{Déf. } \text{ran} \rangle \\
= & \text{ran}(r_2) && \square
\end{aligned}$$

Théorème 15 Soit $r_1, r_2 \in S \leftrightarrow S$ des relations sur un ensemble S et $T \subset S$.

$$(T \triangleleft r_1) ; r_2 = T \triangleleft (r_1 ; r_2) \quad (3.51)$$

PREUVE.

Montrons que la partie de gauche est égale à la partie de droite.

$$\begin{aligned}
& (T \triangleleft r_1); r_2 && \langle \text{Déf. “;”} \rangle \\
= & \{x \mapsto y \mid \exists z \cdot x \mapsto z \in T \triangleleft r_1 \wedge z \mapsto y \in r_2\} && \langle \text{Déf. } \triangleleft, (\text{LE-5}) \rangle \\
= & \{x \mapsto y \mid \exists z \cdot x \in T \wedge x \mapsto z \in r_1 \wedge z \mapsto y \in r_2\} && \langle (\text{LPO-28}) \rangle \\
= & \{x \mapsto y \mid x \in T \wedge \exists z \cdot x \mapsto z \in r_1 \wedge z \mapsto y \in r_2\} && \langle \text{Déf. “;”} \rangle \\
= & \{x \mapsto y \mid x \in T \wedge x \mapsto y \in r_1; r_2\} && \langle \text{Déf. } \triangleleft \rangle \\
= & T \triangleleft (r_1; r_2) &&
\end{aligned}$$

□

3.4 La preuve de correction de programmes séquentiels

Dans cette section, nous illustrons très simplement la preuve de correction de programmes séquentiels. Nous utilisons le langage de programmation élémentaire suivant, qui suffit pour illustrer les concepts.

- **skip** : l’instruction qui ne fait rien, c’est-à-dire quelle ne modifie aucune variable du programme. Elle est utilisée pour les conditionnelles sans **else**.
- $x := E$: l’affectation de la valeur E à la variable x .
- $P_1; P_2$: la séquence, soit l’exécution de P_1 suivie de P_2 .
- **if** C **then** P_1 **else** P_2 **end** : la conditionnelle.
- **while** C **do** P **end** : la boucle.

3.4.1 La spécification d’un programme

Une *spécification* est une paire de formules *pre* et *post*, respectivement appelées la précondition et la postcondition. Elle est notée

$$\text{Spec}(pre, post)$$

L’expression

$$\text{Spec}(pre, post) \sqsubseteq P$$

indique que le programme P est *correct* par rapport à la spécification $\text{Spec}(pre, post)$. Cela signifie que si le programme P démarre dans un état où *pre* est vrai, alors il termine dans un état où *post* est vrai. Elle est définie comme suit:

$$\text{Spec}(pre, post) \sqsubseteq P \Leftrightarrow (pre \Rightarrow wp(P, post)) \quad (3.52)$$

Cette définition utilise l’opérateur $wp(P, \mathcal{A})$, qui dénote la precondition la plus faible telle que l’exécution de P termine dans un état final satisfaisant \mathcal{A} . Cet opérateur est appelé *weakest precondition*. Il a été défini par E.W.D. Dijkstra en 1975, en étendant les travaux de C.A.R. Hoare effectués en 1968. La figure 3.2 illustre ces concepts.

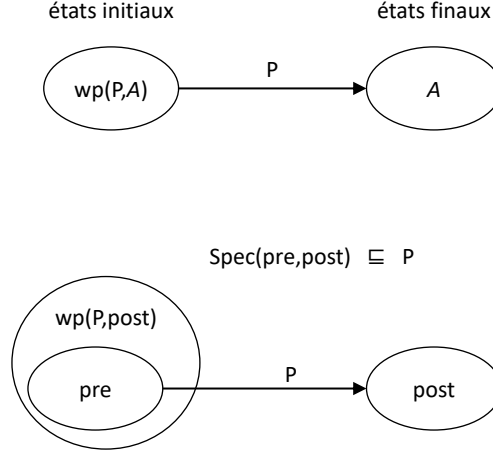


Figure 3.2: Illustration de wp et de correction

3.4.2 Les axiomes du wp-calcul

Soit les notations suivantes.

- \vec{x} l'ensemble des variables d'un programme,
- I une formule appelée un *invariant*,
- V un terme de type entier (i.e., $V \in \mathbb{Z}$) appelé un *variant*,
- n une variable *fraîche* (i.e., n n'apparaît pas dans \vec{x}).

L'expression $wp(P, \mathcal{A})$ est définie de manière inductive sur la structure des programmes.

$$wp(\mathbf{skip}, \mathcal{A}) \Leftrightarrow \mathcal{A} \quad (3.53)$$

$$wp(x := E, \mathcal{A}) \Leftrightarrow \mathcal{A}[x := E] \quad (3.54)$$

$$wp(P_1; P_2, \mathcal{A}) \Leftrightarrow wp(P_1, wp(P_2, \mathcal{A})) \quad (3.55)$$

$$wp(\mathbf{if } C \mathbf{ then } P_1 \mathbf{ else } P_2 \mathbf{ end}, \mathcal{A}) \Leftrightarrow (C \Rightarrow wp(P_1, \mathcal{A})) \wedge (\neg C \Rightarrow wp(P_2, \mathcal{A})) \quad (3.56)$$

$$wp(\mathbf{while } C \mathbf{ do } P \mathbf{ end}, \mathcal{A}) \Leftrightarrow I \quad (3.57)$$

$$\wedge (\forall \vec{x} \cdot \neg C \wedge I \Rightarrow \mathcal{A}) \quad (W1)$$

$$\wedge (\forall \vec{x} \cdot C \wedge I \Rightarrow wp(P, I)) \quad (W2)$$

$$\wedge (\forall \vec{x}, n \cdot C \wedge I \wedge n = V \Rightarrow wp(P, V < n)) \quad (W3)$$

$$\wedge (\forall \vec{x} \cdot C \wedge I \Rightarrow V \geq 0) \quad (W4)$$

3.4.3 Affectation

La figure 3.3 illustre l'application du wp-calcul à une affectation $x := E$. Si le programme

$$x := x + 1$$

démontre dans un état satisfaisant

$$x > -1$$

alors il termine dans un état satisfaisant

$$x > 0.$$

En voici le calcul:

$$\begin{aligned}
 & wp(x := x + 1, x > 0) \\
 \Leftrightarrow & \hspace{20em} \langle \text{def. (3.54)} \rangle \\
 & x > 0[x := x + 1] \\
 \Leftrightarrow & \hspace{10em} \langle \text{application de la substitution} \rangle \\
 & x + 1 > 0 \\
 \Leftrightarrow & \hspace{15em} \langle \text{lois de l'arithmétique} \rangle \\
 & x > -1
 \end{aligned}$$

3.4.4 La séquence

La figure 3.4 illustre l'application du wp-calcul à une séquence $P_1; P_2$. Si le programme

$$x := x + 1; y := x + 1$$

démarre dans un état satisfaisant

$$x > -2,$$

alors il termine dans un état satisfaisant

$$y > 0.$$

En voici le calcul:

$$\begin{aligned}
 & wp(x := x + 1; y := x + 1, y > 0) \\
 \Leftrightarrow & \hspace{20em} \langle \text{def. (3.55)} \rangle \\
 & wp(x := x + 1, wp(y := x + 1, y > 0)) \\
 \Leftrightarrow & \hspace{20em} \langle \text{def. (3.54)} \rangle \\
 & wp(x := x + 1, y > 0[y := x + 1]) \\
 \Leftrightarrow & \hspace{10em} \langle \text{application de la substitution} \rangle \\
 & wp(x := x + 1, x + 1 > 0) \\
 \Leftrightarrow & \hspace{20em} \langle \text{def. (3.54)} \rangle \\
 & x + 1 > 0[x := x + 1] \\
 \Leftrightarrow & \hspace{10em} \langle \text{application de la substitution} \rangle \\
 & (x + 1) + 1 > 0 \\
 \Leftrightarrow & \hspace{15em} \langle \text{lois de l'arithmétique} \rangle \\
 & x > -2
 \end{aligned}$$

3.4.5 La conditionnelle

La figure 3.5 ci-dessous illustre l'application du wp-calcul à une conditionnelle **if** C **then** P_1 **else** P_2 **end**. Si le programme

$$\text{if } x > 0 \text{ then skip else } x := -x$$

démarre dans un état satisfaisant

$$x \neq 0,$$

alors il termine dans un état satisfaisant

$$x > 0.$$

En voici le calcul:

$$\begin{aligned}
& wp(\mathbf{if } x > 0 \mathbf{ then skip else } x := -x \mathbf{ end}, x > 0) \\
\Leftrightarrow & \hspace{20em} \langle \text{def. (3.56)} \rangle \\
& x > 0 \Rightarrow wp(\mathbf{skip}, x > 0) \\
& \wedge \\
& \neg(x > 0) \Rightarrow wp(x := -x, x > 0) \\
\Leftrightarrow & \hspace{15em} \langle \text{def. (3.54) and (3.53)} \rangle \\
& x > 0 \Rightarrow x > 0 \\
& \wedge \\
& \neg(x > 0) \Rightarrow -x > 0 \\
\Leftrightarrow & \hspace{15em} \langle \text{(LP-23),(LP-22),(LP-21)} \rangle \\
& \mathbf{vrai} \\
& \wedge \\
& x > 0 \vee -x > 0 \\
\Leftrightarrow & \hspace{20em} \langle \text{lois de l'arithmétique} \rangle \\
& x > 0 \vee x < 0 \\
\Leftrightarrow & \hspace{20em} \langle \text{lois de l'arithmétique} \rangle \\
& x \neq 0
\end{aligned}$$

3.4.6 La boucle

La figure 3.6 ci-dessous illustre l'application du wp-calcul à une boucle

while $x < k$ **do** $x := x + 1; s := s + a[x]$ **end**

On suppose que $x, s, k : \mathbb{N}$ et $a : \mathbf{array}[1..k]$ of \mathbb{N} , ce qui se représente en mathématiques par $a \in 1..k \rightarrow \mathbb{N}$. Si cette boucle démarre dans un état satisfaisant

$$s = \sum_{i=1}^x a[i] \wedge x \in 0..k \quad (3.58)$$

alors elle termine dans un état satisfaisant

$$s = \sum_{i=1}^k a[i] \quad (3.59)$$

Nous allons faire ce calcul avec l'équation (3.57), en prouvant séparément chaque élément de la conjonction de (3.57) (i.e., en prouvant W1, W2, W3 et W4). Le choix de I et V est bien sûr déterminant. Ici, nous choisissons

$$I \triangleq (3.58)$$

et

$$V \triangleq k - x.$$

La condition de la boucle est

$$C \triangleq x < k.$$

Le corps de la boucle est

$$P \triangleq x := x + 1; s := s + a[x].$$

Preuve de (W1): supposons $\neg C \wedge I$. Il faut prouver \mathcal{A} , qui est dans ce cas-ci (3.59).

$$\begin{aligned}
& (3.59) \\
& \Leftrightarrow s = \sum_{i=1}^k a[i] \\
& \Leftrightarrow \langle \text{hyp. } I : s = \sum_{i=1}^x a[i] \rangle \\
& \quad \sum_{i=1}^x a[i] = \sum_{i=1}^k a[i] \\
& \Leftrightarrow \langle \text{hyp. } I : x \in 0..k \text{ et hyp. } \neg C : x \geq k \text{ entraînent } x = k \rangle \\
& \quad \sum_{i=1}^k a[i] = \sum_{i=1}^k a[i] \\
& \Leftarrow \\
& \text{Lois de l'arithmétique}
\end{aligned}$$

Preuve de (W2): supposons $C \wedge I$ et montrons $wp(P, I)$.

$$\begin{aligned}
& wp(P, I) \\
& \Leftrightarrow \langle \text{déf. } P \rangle \\
& \quad wp(x := x + 1; s := s + a[x], s = \sum_{i=1}^x a[i] \wedge x \in 0..k) \\
& \Leftrightarrow \langle \text{def. (3.54) et (3.55)} \rangle \\
& \quad s + a[x + 1] = \sum_{i=1}^{x+1} a[i] \wedge x + 1 \in 0..k \\
& \Leftrightarrow \langle \text{hyp. } I : s = \sum_{i=1}^x a[i] \rangle \\
& \quad (\sum_{i=1}^x a[i]) + a[x + 1] = \sum_{i=1}^{x+1} a[i] \wedge x + 1 \in 0..k
\end{aligned}$$

La formule $(\sum_{i=1}^x a[i]) + a[x + 1] = \sum_{i=1}^{x+1} a[i]$ est une instanciation de (3.2). La formule $x + 1 \in 0..k$ se déduit comme suit: l'hypothèse C implique $x + 1 \leq k$; l'hypothèse I implique $x + 1 \geq 0$; donc, par définition de $0..k$, on a $x + 1 \in 0..k$.

Preuve de (W3): : supposons $C \wedge I \wedge n = V$ et montrons $wp(P, V < n)$.

$$\begin{aligned}
& wp(P, V < n) \\
& \Leftrightarrow \\
& \quad wp(x := x + 1; s := s + a[x], k - x < n) \\
& \Leftrightarrow \langle \text{def. (3.54) et (3.55)} \rangle \\
& \quad k - (x + 1) < n \\
& \Leftrightarrow \langle \text{hyp. } n = V \rangle \\
& \quad k - (x + 1) < k - x \\
& \Leftrightarrow \langle \text{lois arithmétiques} \rangle \\
& \quad k - x - 1 < k - x \\
& \Leftarrow \\
& \quad x \in 0..k \\
& \Leftarrow \\
& \quad \text{hyp. } I
\end{aligned}$$

Preuve de (W4): : supposons $C \wedge I$ et montrons $V \geq 0$.

$$\begin{aligned}
& V \geq 0 \\
& \Leftrightarrow \\
& \quad k - x \geq 0 \\
& \Leftarrow \\
& \quad x \in 0..k \\
& \Leftarrow \\
& \quad \text{hyp. } I
\end{aligned}$$

Puisque W1, W2, W3 et W4 sont vraies, alors on peut conclure que

$$\begin{aligned}
& wp(\mathbf{while} \ x < k \ \mathbf{do} \ x := x + 1; s := s + a[x] \ \mathbf{end}, \ s = \sum_{i=1}^k a[i]) \\
\Leftrightarrow & \\
& s = \sum_{i=1}^x a[i] \ \wedge \ x \in 0..k
\end{aligned} \tag{3.60}$$

3.4.7 Preuve de correction d'un programme

Par exemple, montrons que le programme suivant

```

x := 0;
s := 0;
while x < k do x := x + 1; s := s + a[x] end

```

satisfait la spécification $\text{Spec}(\text{vrai}, s = \sum_{i=1}^k a[i])$. Soit W la boucle de ce programme. Il faut prouver (3.52), soit

$$\text{vrai} \Rightarrow wp(x := 0; s := 0; W, s = \sum_{i=1}^k a[i])$$

Voici la preuve.

$$\begin{aligned}
& wp(x := 0; s := 0; W, s = \sum_{i=1}^k a[i]) \\
\Leftrightarrow & \tag{3.55} \\
& wp(x := 0, wp(s := 0, wp(W, s = \sum_{i=1}^k a[i]))) \\
\Leftarrow & \tag{3.60} \\
& wp(x := 0, wp(s := 0, s = \sum_{i=1}^x a[i] \ \wedge \ x \in 0..k)) \\
\Leftarrow & \tag{3.54} \\
& wp(x := 0, 0 = \sum_{i=1}^x a[i] \ \wedge \ x \in 0..k) \\
\Leftarrow & \tag{3.54} \\
& 0 = \sum_{i=1}^0 a[i] \ \wedge \ 0 \in 0..k \\
\Leftarrow & \\
& \text{Lois de l'arithmétique}
\end{aligned}$$

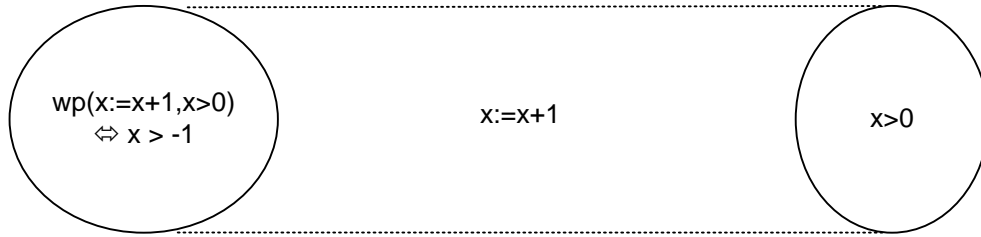


Figure 3.3: Le calcul d'un wp pour une affectation $x := E$

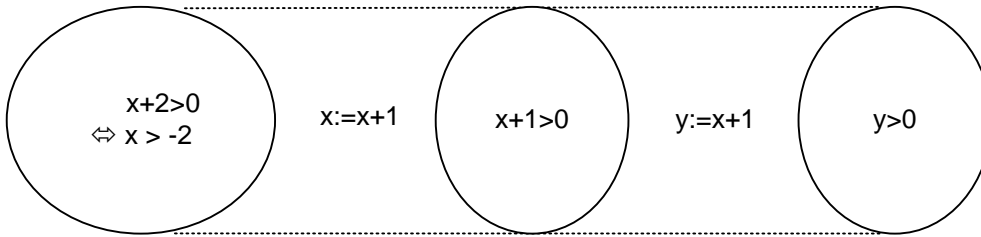
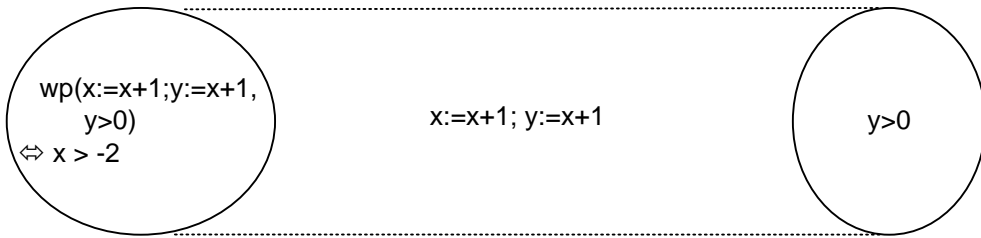


Figure 3.4: Le calcul d'un wp pour une sequence $P_1; P_2$

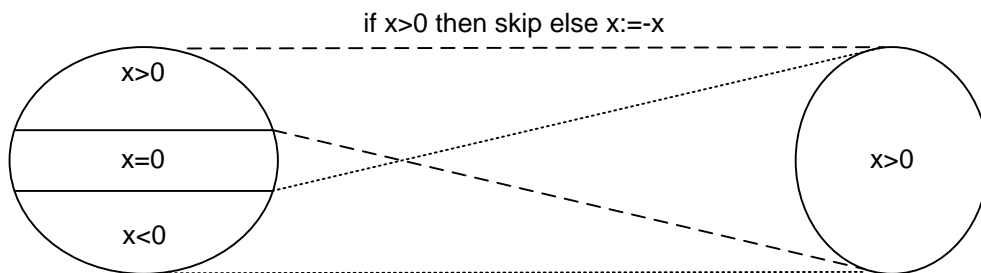


Figure 3.5: Le calcul d'un wp pour une conditionnelle

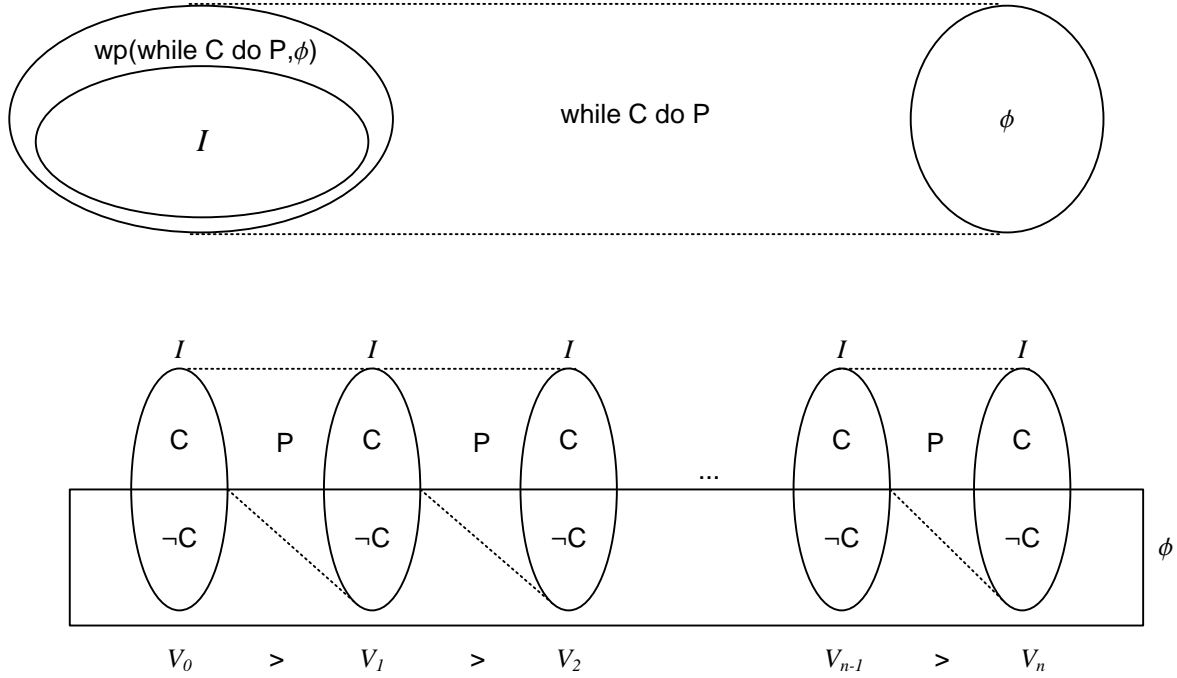


Figure 3.6: Le calcul d'un wp pour une boucle

3.5 Exercices

1. Prouvez par induction les formules suivantes.

- (a) $\forall x \cdot x \in \mathbb{N} \Rightarrow \sum_{i=0}^x 2^i = 2^{x+1} - 1$
- (b) $\forall x \cdot x \in \mathbb{N}_1 \Rightarrow \sum_{i=1}^x 5^{i-1} = (5^x - 1)/4$
- (c) $\forall x \cdot x \in \mathbb{N}_3 \Rightarrow 2x + 1 < 2^x$
- (d) $\forall x \cdot x \in \mathbb{N}_1 \Rightarrow \phi^{x-2} \leq \mathcal{F}_x$
- (e) $\forall x \cdot x \in \mathbb{N} \Rightarrow \mathcal{F}_x < 2^x$

2. Prouvez les formules suivantes: soit $r, r_1, r_2, r_3 \in S \leftrightarrow S$ et $T \subseteq S$

- (a) $(r_1 ; r_2)^{-1} = r_2^{-1} ; r_1^{-1}$
- (b) $(r_1 \cup r_2)^{-1} = r_1^{-1} \cup r_2^{-1}$
- (c) $\text{id}(T) ; r = (T \times S) \cap r$
- (d) $\text{id}(T) ; r = T \triangleleft r$
- (e) $\text{id}(T) ; (r_1 \cap r_2) = (\text{id}(T) ; r_1) \cap (\text{id}(T) ; r_2)$
- (f) $(T \times S) ; (T \times S) = (T \times S)$
- (g) $r_1 \subseteq r_2 \Leftrightarrow r_1^{-1} \subseteq r_2^{-1}$
- (h) $r_1 \in S \leftrightarrow S \Rightarrow r_1 ; (r_2 \cap r_3) = (r_1 ; r_2) \cap (r_1 ; r_3)$

Indice: Utilisez le lemme suivant, que vous prouvez ensuite.

$$r \in S \leftrightarrow S \Rightarrow ((x, y) \in r \wedge (x, z) \in r \Rightarrow y = z)$$

$$(i) \quad f \in A \rightarrow B \wedge g \in B \rightarrow A \wedge f; g = \text{id}(A) \wedge g; f = \text{id}(B)$$

$$\Rightarrow f = g^{-1} \wedge f \in A \rightsquigarrow B \wedge g \in B \rightsquigarrow A$$

$$(j) \quad f \in S \rightsquigarrow T$$

$$\Leftrightarrow \forall(x, y). (x \in S \wedge y \in S \Rightarrow (f(x) = f(y) \Rightarrow x = y)) \wedge \text{dom}(f) = S$$

$$(k) \quad r \in S \leftrightarrow S$$

$$\Rightarrow \text{id}(S) \subseteq r; r^{-1} \Leftrightarrow \text{dom}(r) \subseteq S$$

$$(l) \text{ Soit } r \in S \leftrightarrow S, \text{ montrez que } r; \text{id}(S) = r$$

3. Déterminez si les formules suivantes sont vraies ou fausses. Donnez une preuve si elles sont vraies. Donnez un contre-exemple si elles sont fausses.

Soit A, B, C, D des ensembles.

$$(a) \quad (A \cap B) \times (C \cap D) = (A \times C) \cap (B \times D)$$

$$(b) \quad (A \cup B) \times (C \cup D) = (A \times C) \cup (B \times D)$$

Chapitre 4

Automate

Les automates sont largement utilisés en informatique. Ils permettent de reconnaître les mots d'un langage, et définissent la classe des langages dits *réguliers*. On les retrouve dans les compilateurs, les spécifications de systèmes, la vérification de systèmes, entre autres. Ils constituent le modèle le plus simple de représentation du calcul dans un ordinateur. En effet, un ordinateur n'est rien d'autre qu'une machine qui dispose d'une mémoire finie, d'un état (i.e., la valeur courante de sa mémoire), et qui effectue une opération qui change son état. Les automates sont étroitement reliés aux expressions régulières et aux grammaires régulières, qui représentent tous la même classe de langages, soit les langages dits réguliers. Les expressions régulières et les grammaires sont présentées dans le cours IFT313. Il existe plusieurs variantes d'automates (machine de Mealy et de Moore, automate à pile, diagramme états-transitions, machine de Turing). Dans MAT115, on s'intéresse seulement aux automates finis déterministes et non-déterministes, qui sont équivalents. Les autres formes d'automates sont traitées dans les cours suivants : IFT313 – Introduction aux langages formels, IFT232 – Méthodes de conception orientée objet, IGL301 – Spécification et vérification des exigences, IFT580 – Compilation et interprétation des langages, IFT503 – Théorie du calcul et IGL501 – Méthodes formelles en génie logiciel.

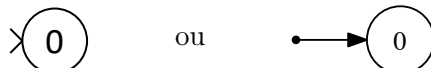
4.1 Automate fini déterministe

Définition 17 Un *automate fini déterministe* (AFD) est un quintuplet $\langle Q, \Sigma, \delta, q_0, F \rangle$ tel que

- Q est l'ensemble fini des états de l'automate.
- Σ est un ensemble fini de symboles; on appelle cet ensemble l'*alphabet* de l'automate.
- $\delta \in Q \times \Sigma \rightarrow Q$ est la fonction de transition de l'automate.
- $q_0 \in Q$ est l'état initial de l'automate.
- $F \subseteq Q$ est l'ensemble des états finaux de l'automate.

□

La figure 4.1 illustre deux AFD. L'état initial d'un AFD est identifié par l'un ou l'autre des pictogrammes ci-dessous.



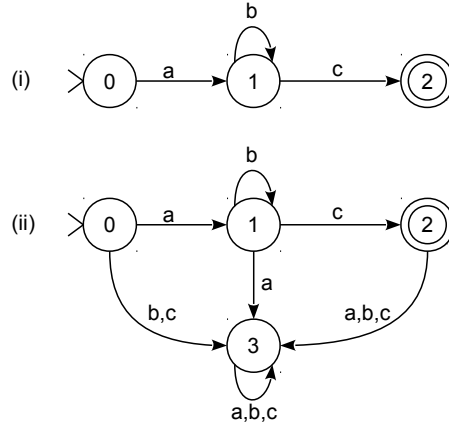
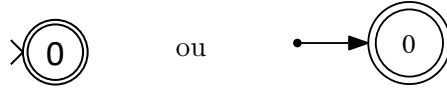


Figure 4.1: AFD incomplet (i), complété en (ii)

Un état final est représenté par un double cercle.



Un état initial peut aussi être final; il est représenté comme suit.



L'AFD (i) est dit *incomplet*, car sa fonction de transition n'est pas totale. Pour l'état 0, les transitions sur *b* et *c* ne sont pas définies. Idem pour l'état 1 avec *a*, et l'état 2 avec *a,b,c*. On utilise souvent la convention suivante, par soucis de lisibilité: les transitions non définies sont implicitement des transitions vers un état appelé *puits*; ce nom signifie qu'on ne peut plus sortir de cet état après y être entré. L'AFD (ii) représente l'AFD (i) avec son état 3, qui est un état puits, ajouté afin que la fonction de transition δ soit totale. Voici la définition formelle de l'automate (ii).

- $Q = \{0, 1, 2, 3\}$
- $\Sigma = \{a, b, c\}$
- $\delta = \{(0, a, 1), (0, b, 3), (0, c, 3),$
 $(1, a, 3), (1, b, 1), (1, c, 2),$
 $(2, a, 3), (2, b, 3), (2, c, 3),$
 $(3, a, 3), (3, b, 3), (3, c, 3)\}$
- $q_0 = 0$
- $F = \{2\}$

Chaque triplet de δ représente une transition dans l'automate. Voici quelques autres définitions utiles.

- Nous utiliserons généralement le terme “mot” au lieu de “suite de symboles” dans le reste de ce chapitre, comme cela est l'appellation typique dans la théorie des automates et des langages formels.

- Σ^* est l'ensemble de tous les mots formés d'éléments de Σ .
 Σ^* est noté $\text{seq}(\Sigma)$ dans le langage B.

- $[]$ dénote le mot vide (i.e., il ne contient aucun symbole).
 $[]$ est parfois notée ϵ dans plusieurs ouvrages sur les automates.
 $[]$ est l'élément neutre de la concaténation:

$$[] \wedge w = w \wedge [] = w$$

- $\Sigma^+ = \Sigma^* - []$
 Σ^+ est noté $\text{seq}_1(\Sigma)$ dans le langage B.
- Pour simplifier la représentation d'une transition de q_1 à q_2 sur σ , c'est-à-dire le triplet

$$(q_1, \sigma, q_2) \in \delta$$

on utilise à la place la notation suivante, qui rappelle une transition dans l'automate.

$$q_1 \xrightarrow{\sigma} q_2$$

- $\widehat{\delta} \in Q \times \Sigma^* \rightarrow Q$ est l'extension de δ aux mots. Elle est définie comme suit:

$$\widehat{\delta}(q, []) = q \tag{4.1}$$

$$\widehat{\delta}(q, w \wedge [\sigma]) = \delta(\widehat{\delta}(q, w), \sigma) \tag{4.2}$$

- $L(M) = \{w \mid w \in \Sigma^* \wedge \widehat{\delta}(q_0, w) \in F\}$ dénote le langage accepté par l'AFD M .

En termes plus simples, on dit qu'un automate accepte le mot $w = \sigma_1 \dots \sigma_n$ si la séquence de transitions qui lit le mot w termine sur $q_n \in F$.

$$q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} \dots \xrightarrow{\sigma_n} q_n \text{ et } q_n \in F$$

Rappelons que q_0 dénote l'état initial de l'automate. Voici un exemple d'acceptation du mot abc par l'automate de la Figure 4.1(ii)

$$0 \xrightarrow{a} 1 \xrightarrow{b} 1 \xrightarrow{b} 1 \xrightarrow{c} 2 \in F$$

Le mot bb est refusé par le même automate, car l'automate termine dans l'état 3 après avoir lu tous les symboles, et cet état n'est pas final.

$$0 \xrightarrow{b} 3 \xrightarrow{b} 3 \notin F$$

La fonction $\widehat{\delta}$ permet de représenter formellement la séquence de transitions qui permet de lire un mot dans un automate. Pour le voir, représentons cette séquence de transitions en utilisant δ , en remplaçant q_1 par $\delta(q_0, \sigma_1)$, q_2 par $\delta(\delta(q_0, \sigma_1), \sigma_2)$, et ainsi de suite.

$$q_0 \xrightarrow{\sigma_1} \delta(q_0, \sigma_1) \xrightarrow{\sigma_2} \delta(\delta(q_0, \sigma_1), \sigma_2) \dots \xrightarrow{\sigma_n} \delta(\delta(\dots), \sigma_{n-1}), \sigma_n)$$

La fonction $\widehat{\delta}$ permet de calculer le terme $\delta(\delta(\dots), \sigma_{n-1}), \sigma_n$. À titre illustratif, appliquons la définition de $\widehat{\delta}$ à l'automate de la Figure 4.1(ii) pour le mot abc .

$$\begin{aligned}
& \llbracket \widehat{\delta}(0, abbc) \rrbracket \\
= & \llbracket \delta(\llbracket \widehat{\delta}(0, abb) \rrbracket, c) \rrbracket && \langle \text{d\'ef. de } \widehat{\delta} \rangle \\
= & \delta(\llbracket \delta(\llbracket \widehat{\delta}(0, ab) \rrbracket, b) \rrbracket, c) && \langle \text{d\'ef. de } \widehat{\delta} \rangle \\
= & \delta(\delta(\llbracket \delta(\llbracket \widehat{\delta}(0, a) \rrbracket, b) \rrbracket, b), c) && \langle \text{d\'ef. de } \widehat{\delta} \rangle \\
= & \delta(\delta(\delta(\llbracket \delta(\llbracket \widehat{\delta}(0, []) \rrbracket, a) \rrbracket, b), b), c) && \langle \text{d\'ef. de } \widehat{\delta} \rangle \\
= & \delta(\delta(\delta(\llbracket \delta(\llbracket 0 \rrbracket, a) \rrbracket, b), b), c) && \langle \text{d\'ef. de } \widehat{\delta} \rangle \\
= & \delta(\delta(\llbracket \delta(\llbracket 1 \rrbracket, b) \rrbracket, b), c) && \langle \text{d\'ef. de } \delta \rangle \\
= & \llbracket \delta(\llbracket 1 \rrbracket, c) \rrbracket && \langle \text{d\'ef. de } \delta \rangle \\
= & \llbracket 2 \rrbracket && \langle \text{d\'ef. de } \delta \rangle
\end{aligned}$$

4.2 Automate fini non-d\'eterministe

D\'efinition 18 Un *automate fini non-d\'eterministe* (AFND) est un quintuplet $\langle Q, \Sigma, \delta, q_0, F \rangle$ tel que

- Q, Σ, q_0, F sont de m\^eme nature que Q, Σ, q_0, F dans un AFD.
- $\delta \subseteq Q \times (\Sigma \cup \{\lambda\}) \times Q$ est la *relation* de transition de l'automate.

□

Les diff\'erences entre un AFND et un AFD sont les suivantes:

- Les transitions d'un AFND peuvent \^etre \^etiquetees avec λ , un symbole particulier qui d\'enote un changement d'\^etat sans lecture d'un symbole dans la suite \^a traiter par un automate. Autrement dit, λ ne fait pas partie de l'alphabet du langage accept\'e par l'AFND.
- δ est une *fonction totale* dans un AFD, alors que δ est une *relation* dans un AFND. Il n'est donc pas n\'ecessaire de faire un \^etat *puits* dans un AFND, puisque la relation δ peut \^etre partielle.

La figure 4.2 illustre deux AFND, soit (i) et (ii), qui acceptent le m\^eme langage, ainsi qu'un AFD (iii) qui accepte le m\^eme langage que (i) et (ii). Le non-d\'eterminisme est pr\'esent dans (i) pour l'\^etat 0, car il y a deux transitions possibles pour a. Le non-d\'eterminisme est pr\'esent dans (ii) pour l'\^etat 0, car il y a une transition sur λ .

Les d\'efinitions suivantes seront utiles.

- La fonction λ -filtre $\in (\Sigma \cup \{\lambda\})^* \rightarrow \Sigma^*$ supprime le symbole λ d'un mot.

$$\lambda\text{-filtre}([]) = [] \tag{4.3}$$

$$\sigma \in \Sigma \wedge w \in (\Sigma \cup \{\lambda\})^* \Rightarrow \lambda\text{-filtre}([\sigma] \wedge w) = [\sigma] \wedge \lambda\text{-filtre}(w) \tag{4.4}$$

$$w \in (\Sigma \cup \{\lambda\})^* \Rightarrow \lambda\text{-filtre}([\lambda] \wedge w) = \lambda\text{-filtre}(w) \tag{4.5}$$

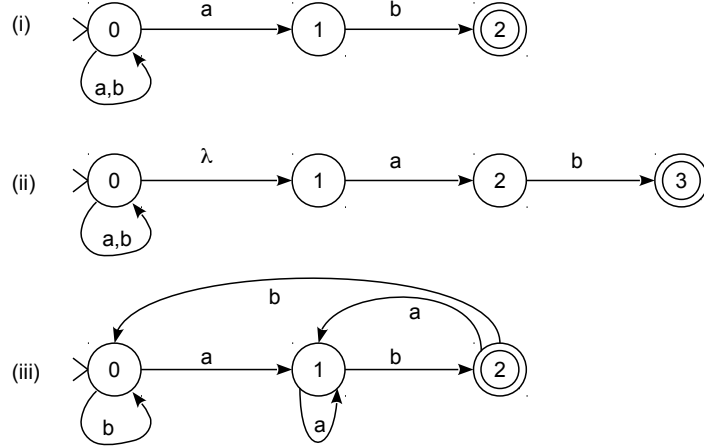


Figure 4.2: Les AFND (i) et (ii) acceptent le même langage que l'AFD (iii)

- Le langage accepté par un AFND $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ est défini comme suit:

$$\begin{aligned}
 L(M) = \{w_1 \mid & w_1 \in \Sigma^* \wedge \\
 & \exists w_2 \cdot w_2 \in (\Sigma \cup \{\lambda\})^* \wedge \lambda\text{-filtre}(w_2) = w_1 \wedge \\
 & \exists z \cdot z \in Q^+ \wedge \text{card}(z) = \text{card}(w_2) + 1 \wedge \\
 & \text{first}(z) = q_0 \wedge \text{last}(z) \in F \wedge \\
 & \forall i \cdot i \in \text{dom}(w_2) \Rightarrow (z(i), w_2(i), z(i+1)) \in \delta \\
 & \}
 \end{aligned}$$

Autrement dit, $w_1 \in L(M)$ ssi il existe w_2 tel que $w_1 = \lambda\text{-filtre}(w_2)$, et il existe z tel que $q_0 = z(1)$ et $\text{last}(z) \in F$ et

$$z(1) \xrightarrow{w_2(1)} z(2) \xrightarrow{w_2(2)} \dots \xrightarrow{w_2(n)} z(n+1)$$

Dans le cas où $w_1 = w_2 = []$, on a $\text{dom}(w_2) = \{\}$, et $w_1 \in L(M) \Leftrightarrow z = [q_0]$ et $q_0 \in F$.

4.3 Détermination d'un AFND

Cette section décrit comment déterminer un AFND M , c'est-à-dire construire un AFD M_D tel que $L(M) = L(M_D)$. Les définitions suivantes seront utiles.

- La relation λ -closure contient les couples d'états (q_1, q_2) telles qu'il existe, dans M , une suite (possiblement vide) de transitions λ menant de q_1 à q_2 , i.e.,

$$q_1 \xrightarrow{\lambda} \dots \xrightarrow{\lambda} q_2$$

On peut définir λ -closure en utilisant la fermeture réflexive et transitive.

$$\lambda\text{-closure} = \{(q_1, q_2) \mid (q_1, \lambda, q_2) \in \delta\}^*$$

Voici la valeur de λ -closure pour l'AFND (ii) de la figure 4.2:

$$\lambda\text{-closure} = \{(0, 0), (0, 1), (1, 1), (2, 2), (3, 3)\}$$

- La relation $t \in Q \times \Sigma \times Q$ contient les triplets (q_1, σ, q_2) tels qu'il existe, dans M ,
 - une suite (possiblement vide) de transitions λ menant de q_1 à q'_1 ,
 - puis une transition sur σ de q'_1 à q'_2 ,
 - et finalement une suite (possiblement vide) de transitions λ menant de q'_2 à q_2 .

Ce que l'on peut résumer comme suit:

$$q_1 \xrightarrow{\lambda} \dots \xrightarrow{\lambda} q'_1 \xrightarrow{\sigma} q'_2 \xrightarrow{\lambda} \dots \xrightarrow{\lambda} q_2$$

Intuitivement, on aimerait simplement définir t comme suit:

$$t = \lambda\text{-closure} ; \delta ; \lambda\text{-closure}$$

mais la syntaxe du langage B ne le permet pas, à cause d'un problème de typage¹. Elle est définie formellement comme suit:

$$t = \{(q_1, \sigma, q_2) \mid q_1 \in Q \wedge \sigma \in \Sigma \wedge q_2 \in Q \wedge \exists q_3 \cdot (q_1, q_3) \in \lambda\text{-closure} \wedge (q_3, \sigma, q_2) \in \delta ; \lambda\text{-closure}\}$$

Voici la valeur de t pour l'AFND (ii) de la figure 4.2:

$$t = \{(0, a, 0), (0, a, 1), (0, a, 2), (0, b, 0), (0, b, 1), (1, a, 2), (2, b, 3)\}$$

Soit un AFND $M = \langle Q, \Sigma, \delta, q_0, F \rangle$. Il existe un algorithme calculant un AFD $M_D = \langle Q_D, \Sigma, \delta_D, q_{0_D}, F_D \rangle$, tel que $L(M) = L(M_D)$. Les composantes de M_D sont typées comme suit.

- $Q_D \subseteq \mathbb{P}(Q)$, c'est-à-dire que les états de Q_D sont des sous-ensembles d'états de Q .
- $\delta_D \in Q_D \times \Sigma \rightarrow Q_D$
- $q_{0_D} \in Q_D$
- $F_D \subseteq Q_D$

Cet algorithme est donné à la figure 4.4. Il utilise le prédicat $transitionADefinir(X, \sigma)$, défini comme suit:

$$transitionADefinir(X, \sigma) \Leftrightarrow X \in Q_D \wedge (X, \sigma) \notin \text{dom}(\delta_D)$$

Cet algorithme utilise également l'opérateur **ANY**, dont la forme générale est la suivante:

ANY \vec{x} WHERE \mathcal{A} THEN p END

Cet opérateur choisit de manière non-déterministe une valeur de \vec{x} satisfaisant \mathcal{A} et exécute ensuite la partie p .

Un état de M_D est un sous-ensemble de Q . L'algorithme simule le parcours de tous les chemins permettant d'accepter un mot. L'état initial de M_D est $\lambda\text{-closure}[\{q_0\}]$, i.e., tous les états de M accessibles de l'état q_0 par une suite (possiblement vide) de transitions λ . L'algorithme utilise t pour calculer tous les états accessibles en lisant σ en effectuant des transitions λ avant et/ou après σ . Une transition $(X, \sigma, Y) \in \delta_D$ signifie qu'il existe, à partir d'un état de X , une suite (possiblement

¹Le langage Alloy nous le permettrait, car les relations y sont n -aires au lieu de binaires comme en B, et la composition relationnelle est généralisée aux relation n -aires.

```

 $q_{0_D} := \lambda\text{-closure}[\{q_0\}];$ 
 $Q_D := \{q_{0_D}\};$ 
 $\delta_D := \{\};$ 
WHILE  $\text{dom}(\delta_D) \neq (Q_D \times \Sigma)$  DO
  ANY  $X, \sigma, Y$  WHERE
    transitionADefinir( $X, \sigma$ )
     $\wedge Y = \bigcup_{q_2 \in X} t\{(q_2, \sigma)\}$ 
  THEN
     $Q_D := Q_D \cup \{Y\};$ 
     $\delta_D := \delta_D \cup \{(X, \sigma, Y)\}$ 
  END
END;
 $F_D := \{X \mid X \in Q_D \wedge X \cap F \neq \{\}\}$ 

```

Figure 4.4: Algorithme de détermination d'un AFND

vide) de transitions λ , suivie d'une transition sur σ et suivie d'une suite (possiblement vide) de transitions λ , menant à un des états de Y . Ainsi, Y est constitué de tous les états de M accessibles à partir d'un état de X en lisant σ . La figure 4.3 illustre l'AFD résultant de la détermination de l'AFND (ii) de la figure 4.2.

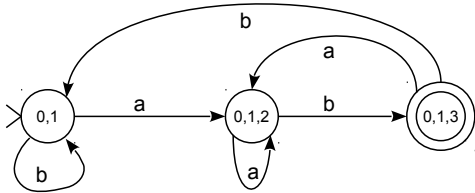


Figure 4.3: L'AFD résultant de la détermination de l'AFND (ii) de la figure 4.2

4.4 Minimisation d'un AFD

Étant donné un AFD M_1 , il est possible de calculer un AFD M_2 tel que $L(M_1) = L(M_2)$ et le nombre d'états de M_2 est minimal, c'est-à-dire, il n'existe pas d'autre AFD M_3 tel $L(M_3) = L(M_1)$ et le nombre d'états de M_3 est plus petit que le nombre d'états de M_2 . La minimisation repose sur le principe suivant: deux états q_1, q_2 sont *équivalents* ssi

$$\forall w \cdot w \in \Sigma^* \Rightarrow (\widehat{\delta}(q_1, w) \in F \Leftrightarrow \widehat{\delta}(q_2, w) \in F)$$

Quand deux états sont équivalents, on dit, de manière équivalente, qu'ils sont *indistinguables*. S'ils ne sont pas équivalents, on dit qu'ils sont *distinguables*, c'est-à-dire,

$$\exists w \cdot w \in \Sigma^* \wedge \neg(\widehat{\delta}(q_1, w) \in F \Leftrightarrow \widehat{\delta}(q_2, w) \in F)$$

Deux états équivalents peuvent être fusionnés, puisque tout mot acceptée par un le sera par l'autre aussi. On peut déterminer les états équivalents d'un AFD en identifiant les états qui sont distinguables, et en propageant cette distinguabilité aux états qui sont reliés aux états distinguables par une transition de l'AFD sur un même symbole. Après avoir parcouru toutes les paires d'états, ceux qui ne sont pas distinguables sont alors considérés comme étant équivalents, et ils peuvent être fusionnés. Au départ, on sait qu'un état final $q_1 \in F$ se distingue d'un état non final $q_2 \notin F$, parce que $\widehat{\delta}(q_1, []) \in F$ et $\widehat{\delta}(q_2, []) \notin F$. Ensuite, si $\delta(x_1, \sigma) = y_1$ et $\delta(x_2, \sigma) = y_2$, et si les états y_1 et y_2 sont distinguables, alors x_1 et x_2 sont aussi distinguables, car l'un permet de terminer après avoir accepté σ , alors que l'autre ne le permet pas. En itérant sur ce principe, on peut trouver tous les états distinguables, et, par conséquent, les états équivalents. L'algorithme de minimisation applique ce principe comme suit:

- La variable D contient les *paires* d'états $\{q_1, q_2\}$ tels que q_1 et q_2 sont distinguables. On utilise un ensemble au lieu d'un couple $q_1 \mapsto q_2$, car si q_1 et q_2 sont distinguables, alors q_2 et q_1 sont aussi distinguables. Utiliser une paire $\{q_1, q_2\}$ évite d'utiliser deux couples $q_1 \mapsto q_2$ et $q_2 \mapsto q_1$. Il n'est donc pas nécessaire de conserver les deux couples, et un ensemble suffit. La distinguabilité est une relation d'équivalence, c'est-à-dire qu'elle est symétrique, transitive et réflexive. On utilise le prédicat *paire*(x, y, Q) pour indiquer que x et y forment une paire de Q .

$$\text{paire}(x, y, Q) \Leftrightarrow x \in Q \wedge y \in Q \wedge x \neq y$$

- La variable r est une relation sur les paires d'états. Si on découvre que $\{x_2, y_2\}$ sont distinguables, alors toutes transitions $\delta(x_1, \sigma) = x_2$ et $\delta(y_1, \sigma) = y_2$ entraînent que x_1 et y_1 sont distinguables; on ajoute alors le couple $\{x_2, y_2\} \mapsto \{x_1, y_1\}$ à r . On itère sur r (i.e., r^*) pour propager la distinguabilité lorsqu'on détermine que x_2 et y_2 sont distinguables.
- La variable V contient les paires d'états $\{q_1, q_2\}$ pour lesquels la distinguabilité reste à déterminer.
- La variable E contient, à la fin de l'algorithme, les paires d'états $\{q_1, q_2\}$ tels que q_1 et q_2 sont équivalents. Il suffit alors de regrouper en un seul état les états qui sont équivalents entre eux, c'est-à-dire les classes d'équivalences. On note $[q]$ la classe d'équivalence de q . Donc, on forme les ensembles suivants, en considérant chaque état $q_1 \in Q$.

$$[q_1] = \{q_1\} \cup \{q_2 \mid \{q_1, q_2\} \in E\}$$

On voit que si $\{q_1, q_2\} \in E$, alors $[q_1] = [q_2]$.

La figure 4.5 illustre le calcul de D et de r . Au départ, on sait que la paire $\{1, 2\} \in D$, car $1 \in F$ et $2 \notin F$. Ensuite, en traitant la paire $\{3, 4\}$, on déduit que $\{3, 4\} \in D$, car $\{1, 2\} \in D$ et $\delta(3, a_1) = 1$ et $\delta(4, a_1) = 2$. La paire $\{5, 6\}$ est traitée de manière similaire. Pour la paire $\{7, 8\}$, on ne trouve pas de transition menant à une paire d'états $\{x, y\} \in D$. Toutefois, comme il existe une paire $\{9, 10\}$ dont on ne connaît encore pas la distinguabilité, et que $\delta(7, a_3) = 9$ et $\delta(8, a_3) = 10$, on ajoute alors le couple $\{9, 10\} \mapsto \{7, 8\}$ à r , car si on déduit subséquemment que $\{9, 10\} \in D$, alors on ajoutera aussi $\{7, 8\}$ à D . À la fin du traitement de toutes les paires d'états, si on n'a pu distinguer $\{7, 8\}$, alors on ajoute $\{7, 8\}$ à E , et ils seront considérés comme équivalents.

L'algorithme de minimisation est présenté à la figure 4.6. Il est issu des travaux de Edward F. Moore, un des auteurs de la théorie des automates, qui est centrale en informatique. John E. Hopcroft a aussi proposé un algorithme, plus performant que celui inspiré des travaux de Moore; la complexité de l'algorithme de Hopcroft est de $O(ns \log n)$, où $n = \text{card}(Q)$ et $s = \text{card}(\Sigma)$, alors que celle de Moore est de l'ordre de $O(n^2s)$. Hopcroft a reçu le prix Alan Turing en 1987 pour ses travaux en théorie des compilateurs, en architecture des grands systèmes, et pour l'invention des architectures RISC pour les processeurs. La figure 4.7 illustre un AFD et sa minimisation. La figure 4.8 illustre le tableau de sa minimisation. Les états équivalents sont ceux qui n'apparaissent pas dans D , c'est-à-dire $\{1, 3\}$ et $\{2, 4\}$; donc $E = \{\{1, 3\}, \{2, 4\}\}$. Les états équivalents sont regroupés dans l'AFD (iii) de la figure 4.7. Donc, étant donné un AFD $\langle Q, \Sigma, \delta, q_0, F \rangle$, on calcule l'automate minimisé

$$M_{min} = \langle Q_{min}, \Sigma, \delta_{min}, q_{0_{min}}, F_{min} \rangle$$

où

- $Q_{min} \subseteq \mathbb{P}(Q)$
- $\delta_{min} \in (\mathbb{P}(Q) \times \Sigma) \rightarrow \mathbb{P}(Q)$
- $F_{min} \subseteq \mathbb{P}(Q)$

comme suit:

- $Q_{min} = \{X \mid \exists x \cdot x \in Q \wedge X = [x]\}$
- $\delta_{min} = \{(X, u, Y) \mid \exists x, y \cdot x \in X \wedge y \in Y \wedge (x, u, y) \in \delta\}$
- $q_{0_{min}} = [q_0]$
- $F_{min} = \{X \mid \exists x \cdot x \in F \wedge X = [x]\}$

Donc, les états de l'automate minimal (Q_{min}) sont les classes d'équivalence des états de l'automate d'origine. Les états finaux (F_{min}) sont les classes d'équivalences des états finaux de l'automate d'origine. Il y a une transition de X sur σ vers Y s'il y a une transition entre un élément de X sur σ vers un élément de Y dans δ .

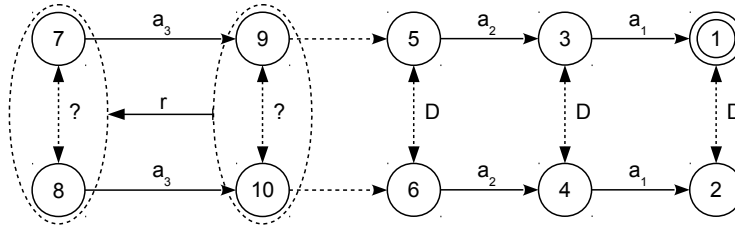


Figure 4.5: Calcul de D et r

```

 $D := \bigcup (x, y).(\text{paire}(x, y, Q) \wedge (x \in F \Leftrightarrow y \notin F) \mid \{\{x, y\}\})$ 
 $V := \bigcup (x, y).(\text{paire}(x, y, Q) \wedge (x \in F \Leftrightarrow y \in F) \mid \{\{x, y\}\})$ 
 $r := \{\}$ 
WHILE  $V \neq \{\}$  DO
  ANY  $x_1, y_1$  WHERE
     $\{x_1, y_1\} \in V$ 
  THEN
    IF  $\exists (u, x_2, y_2). (u \in \Sigma \wedge x_2 = \delta(x_1, u) \wedge y_2 = \delta(y_1, u) \wedge x_2 \neq y_2 \wedge \{x_2, y_2\} \in D)$ 
      THEN
         $D := D \cup r^*[\{\{x_1, y_1\}\}]$ 
      ELSE
         $r := r \cup$ 
           $\bigcup (u, x_2, y_2). (u \in \Sigma \wedge x_2 = \delta(x_1, u) \wedge y_2 = \delta(y_1, u) \wedge x_2 \neq y_2 \wedge$ 
             $\{x_1, y_1\} \neq \{x_2, y_2\}$ 
             $\mid \{\{x_2, y_2\} \mapsto \{x_1, y_1\}\})$ 
        END;
         $V := V - \{\{x_1, y_1\}\}$ 
      END
    END;
  END
 $E := \bigcup (x, y).(\text{paire}(x, y, Q) \mid \{\{x, y\}\}) - D$ 

```

Figure 4.6: Algorithme de minimisation d'un AFD

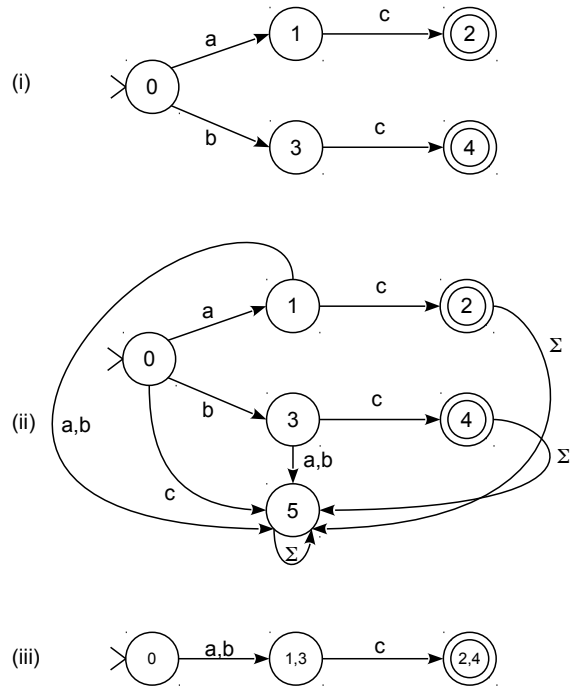
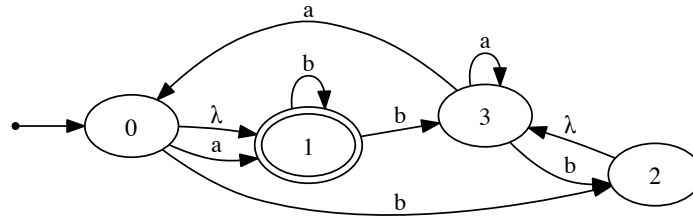


Figure 4.7: Minimisation de l'automate incomplet (i), complété en (ii), et minimisé en (iii)

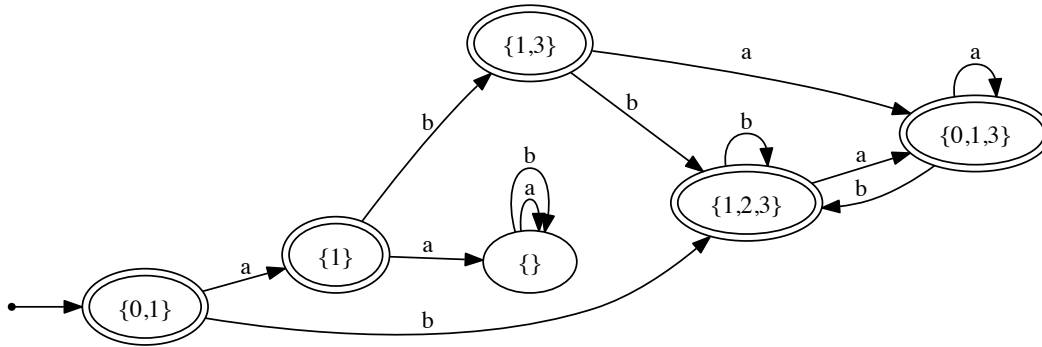
Paire	D	r	E
{0,2}	X		
{0,4}	X		
{1,2}	X		
{1,4}	X		
{2,3}	X		
{2,5}	X		
{3,4}	X		
{4,5}	X		
{0,1}	X		
{0,3}	X		
{0,5}	X (r[{{1,5}}])		
{1,3}			X
{1,5}	X	{0,5}	
{2,4}		{1,3}	X
{3,5}	X	{0,5}	

Figure 4.8: Tableau du calcul pour la minimisation de l'automate de la figure 4.7

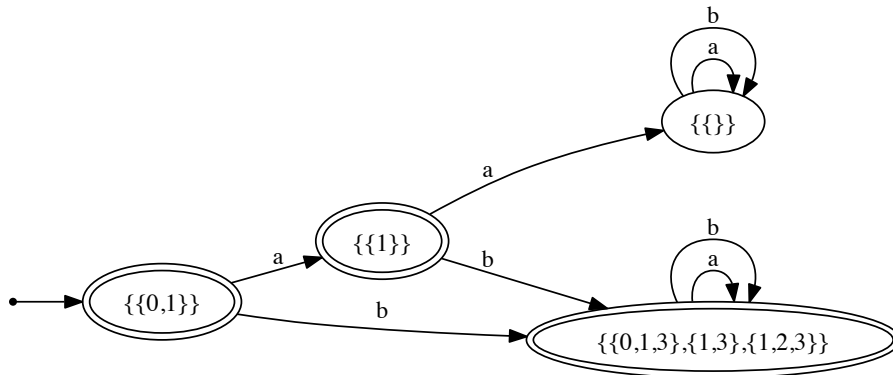
L'AFD résultant de la détermination d'un AFND n'est pas nécessairement minimal. On doit le minimiser. Considérons l'AFND suivant.



Voici l'AFD résultant de sa détermination.



Voici l'AFD résultant de sa minimisation.

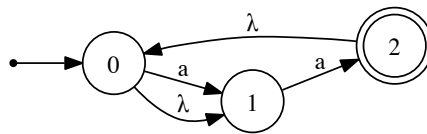


4.5 Exercices

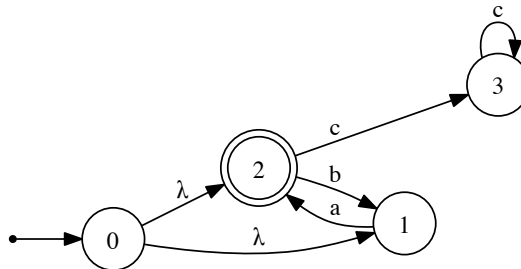
1. Définissez un automate déterministe qui accepte une suite de symboles représentant un entier dans un langage de programmation comme Java ou C++. Le signe (+ ou -) apparaît au début du nombre et il est facultatif. L'entier doit comprendre au moins un chiffre.
 $\Sigma = 0..9 \cup \{+, -\}$.
2. Définissez un automate déterministe qui accepte une suite de symboles représentant un nombre en point flottant dans un langage de programmation comme Java ou C++. Le nombre peut débuter par un “.”. L'exposant, introduit par le symbole e ou le symbole E, est optionnel; il est donné par un entier facultativement signé.
 $\Sigma = 0..9 \cup \{+, -, e, E, .\}$.
3. Définissez un automate déterministe qui décrit le comportement d'une pile de taille maximale 2. L'état final est une pile vide. On peut empiler les éléments de l'ensemble $\{0, 1\}$.
 $\Sigma = (\{\text{push}, \text{top}\} \times \{0, 1\}) \cup \{\text{pop}\}$.
 - L'opération (push, x) ajoute l'élément x au sommet de la pile, avec $x \in \{0, 1\}$.
 - L'opération (top, x) indique que l'élément x est sommet de la pile, avec $x \in \{0, 1\}$. Elle ne modifie pas l'état de la pile.
 - L'opération **pop** supprime l'élément au sommet de la pile. Elle n'a pas de paramètre, car il n'est pas nécessaire vu que c'est élément au sommet de la pile qui est enlevé.
4. Définissez un automate déterministe qui décrit le comportement d'une file de taille maximale 2. Pour simplifier, on suppose qu'une file ne peut contenir deux fois le même élément. L'état final est une file vide. On peut enfiler les éléments de l'ensemble $\{0, 1\}$.
 $\Sigma = (\{\text{enfiler}, \text{tête}\} \times \{0, 1\}) \cup \{\text{défiler}\}$.
 - L'opération $(\text{enfiler}, x)$ ajoute l'élément x à la queue de la file, avec $x \in \{0, 1\}$.
 - L'opération $(\text{tête}, x)$ indique que l'élément x est en tête de la file, avec $x \in \{0, 1\}$. Elle ne modifie pas l'état de la pile.
 - L'opération **défiler** supprime l'élément en tête de la file. Elle n'a pas de paramètre, car il n'est pas nécessaire vu que c'est élément en tête de la file qui est enlevé.
5. Définissez un automate déterministe qui décrit le cycle de vie d'un livre dans un système qui gère une bibliothèque; supposez qu'il n'existe que deux membres, identifiés par l'ensemble $M = \{0, 1\}$. L'état final représente un livre supprimé.
 $\Sigma = \{\text{créer}, \text{supprimer}, \text{retourner}\} \cup (\{\text{emprunter}, \text{réserver}, \text{annuler}\} \times M)$. On suppose qu'il faut réserver un livre avant de l'emprunter.
 - L'opération **créer** crée le livre.
 - L'opération **supprimer** supprime le livre. Avant de supprimer un livre, il faut avoir terminé son prêt et ses réservations.
 - L'opération $(\text{réserver}, x)$ réserve le livre pour le membre x , avec $x \in \{0, 1\}$. Un membre ne peut avoir deux réservations actives pour un même livre.
 - L'opération $(\text{emprunter}, x)$ prête le livre au membre x , avec $x \in \{0, 1\}$. Le membre doit être en tête de la file de réservation. Le membre doit avoir réservé le livre au préalable.
 - L'opération $(\text{annuler}, x)$ annule la réservation du membre x , avec $x \in \{0, 1\}$. Un membre ne peut avoir deux réservations actives pour un livre.

- L'opération **retourner** retourne le livre prêté; elle n'a pas de paramètre, car on sait que le livre est prêté.
- Définissez un automate déterministe qui accepte le paiement d'un café avec des pièces de 5 et 10 cents. L'alphabet contient les éléments suivants: $\Sigma = \{5, 10, \text{café}, (\text{rendre}, 5)\}$, où 5, 10 signifient accepter une pièce de 5 ou 10 cents, **café** veut dire prendre le café, et **(rendre, 5)** signifie rendre 5 cents en monnaie. La suite de pièces est acceptée ssi la somme des pièces donne au moins le coût d'un café, soit 25 cents et au maximum 30 cents (afin de rendre au plus 5 cents). On peut recommencer le processus de paiement, donc l'état final est seulement l'état initial, où il n'y a aucune pièce entrée.
 - Soit $\Sigma = \{a, b, c, d\}$. Définissez un automate déterministe pour chaque sous-question suivante. L'automate doit accepter seulement les suites décrites, et refuser toutes les autres suites.
 - une suite acceptée commence par la sous-suite $[a, b]$.
 - une suite acceptée contient la sous-suite $[a, b]$.
 - une suite acceptée contient la sous-suite $[a, b]$ suivie (pas nécessairement immédiatement) de la sous-suite $[c, d]$.
 - une suite acceptée termine par la sous-suite $[a, b]$.
 - une suite acceptée contient la sous-suite $[a, b]$ ou la sous-suite $[c, d]$.
 - une suite acceptée contient la sous-suite $[a, b]$ et la sous-suite $[c, d]$.
 - une suite acceptée contient un nombre pair de a et un nombre impair de b .
 - Est-il possible de définir un automate qui accepte seulement les suites de la forme $a^n b^n$, c'est-à-dire que la suite est formée d'une suite de a suivie d'une suite de b , et qu'il y a exactement le même nombre de a que de b ?
 - Déterminez les automates suivants:

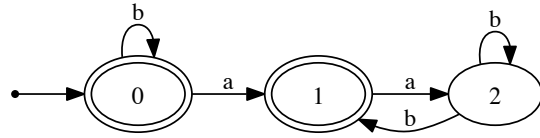
(a)



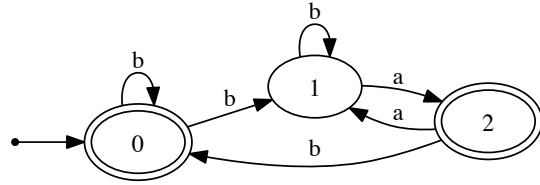
(b)



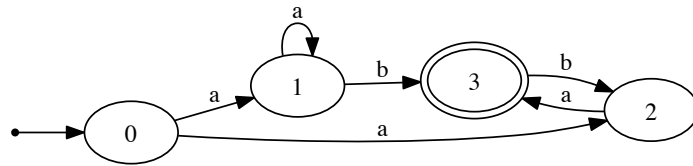
(c)



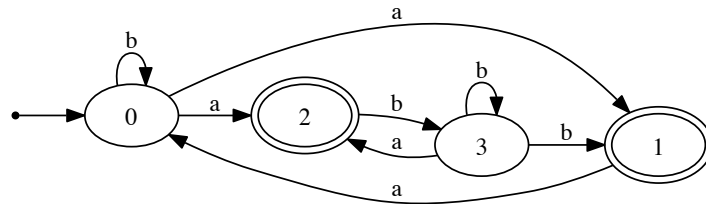
(d)



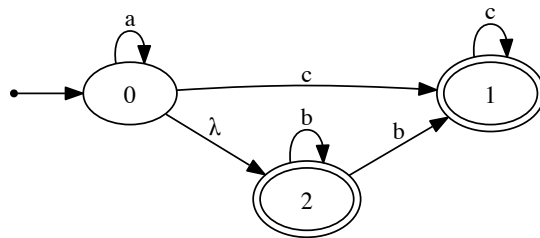
(e)



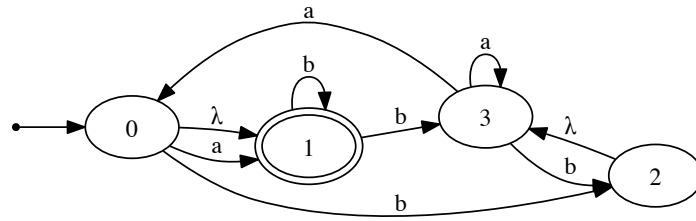
(f)



(g)

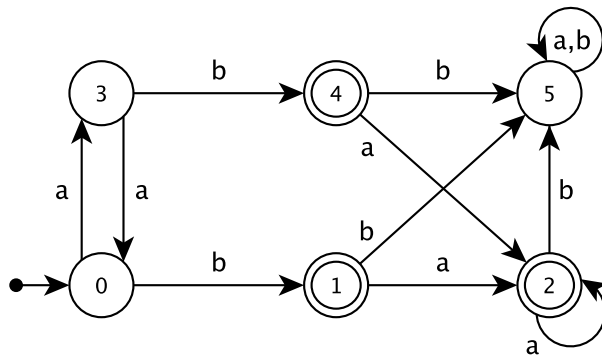


(h)

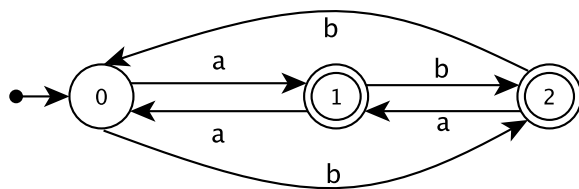


10. Minimisez les automates suivants:

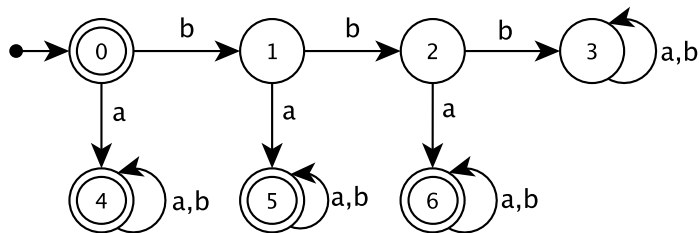
(a)



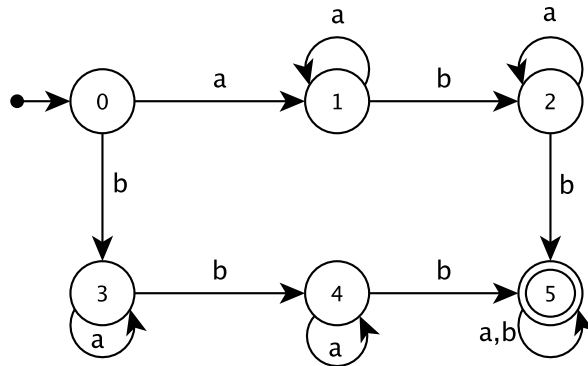
(b)



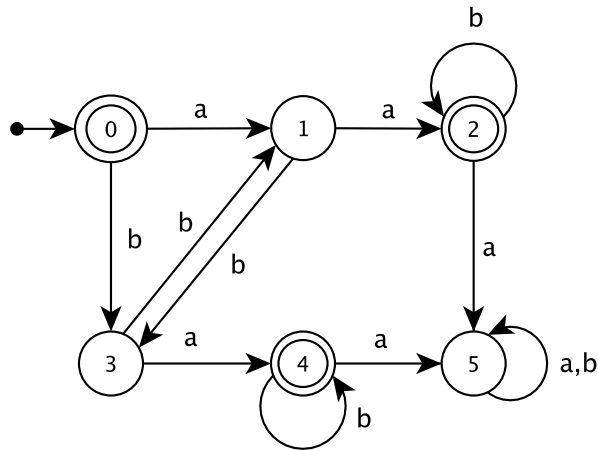
(c)



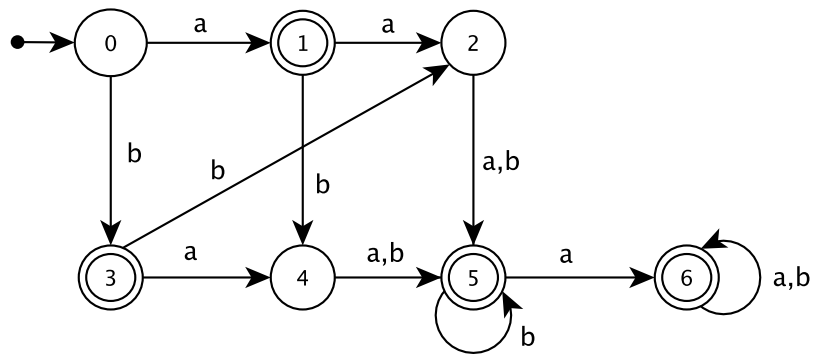
(d)



(e)



(f)



Bibliographie

- [1] J.-R. Abrial. *The B-book: Assigning Programs to Meanings*. Cambridge University Press, New York, NY, USA, 1996. Manuel disponible à la bibliothèque.
- [2] D. Barker-Plummer, J. Barwise, and J. Etchemendy. <https://ggweb.gradegrinder.net/tarskisworld>.
- [3] Peter Chen. The entity-relationship model - toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.
- [4] Olivier Gasquet, François Schwarzentruher, and Martin Strecker. *Panda: A Proof Assistant in Natural Deduction for All. A Gentzen Style Proof Assistant for Undergraduate Students*, pages 85–92. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
- [5] D. Gries and F. B. Schneider. *A Logical Approach to Discrete Math*. Springer-Verlag New York, Inc., New York, NY, USA, 1993. Manuel disponible à la bibliothèque.
- [6] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2012. Manuel disponible à la bibliothèque.
- [7] R. Lalement. *Logique, réduction, résolution*. Masson, Paris, 1990. Manuel disponible à la bibliothèque.
- [8] Michael Leuschel and Michael J. Butler. ProB: an automated analysis toolset for the B method. *STTT*, 10(2):185–203, 2008.
- [9] Michael Leuschel and *et al.* https://www3.hhu.de/stups/prob/index.php/The_ProB_Animator_and_Model_Checker.
- [10] K. H. Rosen. *Discrete Mathematics and Its Applications, Fourth Edition*. McGraw-Hill, 1999. Manuel disponible à la bibliothèque.
- [11] R. Stärk. <https://courses.cs.washington.edu/courses/cse590d/03sp/tarski/tarski.html>.
- [12] T. A. Sudkamp. *Languages and Machines: An Introduction to the Theory of Computer Science, Third Edition*. Addison Wesley, 2005. Manuel disponible à la bibliothèque.

Index

- $S - T$, 55
- $S \cap T$, 55
- $S \cup T$, 55
- $S \times T$, 56
- $[x := t]$, 15
- \Leftarrow , 4
- \models , 40
- \perp , 4
- \equiv , 13
- \in , 52
- $\mathbb{F}(S)$, 55
- $\mathbb{F}_1(S)$, 55
- $\mathbb{P}(S)$, 55
- $\mathbb{P}_1(S)$, 55
- $\text{card}(S)$, 55
- $\text{inter}(S)$, 55
- $\text{union}(S)$, 55
- \mathbb{N}_1 , 54
- \mathbb{N} , 54
- \mathbb{N}_k , 98
- \mathbb{Z} , 54
- \Leftrightarrow , 4
- \Rightarrow , 4
- \vDash , 40
- \mathbb{N} , 51
- \neg , 4
- \top , 4
- \vdash , 33
- \vee , 4
- \wedge , 4
- \oplus , 4
- $\{x \mid \mathcal{A}\}$, 51
- $\{\}$, 54
- $i..j$, 54
- $:=$, 7

- Abrial, Jean-Raymond, 49
- Ackermann, Wilhelm, 2

- AFD, 130
- AFND, 133
- Aho, Alfred Vaino, 2
- antécédent, 56
- application, 64
- automate
 - déterminisation, 134
 - fini déterministe, 130
 - fini non-déterministe, 133
 - minimisation, 137
- axiome, 43

- Bengio, Yoshua, 2

- Church, Alonzo, 2
- cohérence
 - règle d'inférence, 42
- cohérent, 40
- complétude, règle déduction, 42
- compréhension, 50
- conclusion, 34
- condition
 - nécessaire, 10
 - nécessaire et suffisante, 11
 - suffisante, 10
- conjonction, 4
- connecteur logique, 4
- constantes, 4
- conséquence logique, 40
- Cook, Stephen A., 2

- Dahl, Ole-Johan, 44
- De Morgan, Augustus, 28
- Dedekind, Richard, 98
- Dijkstra, E.W.D., 121
- disjonction, 4
- disjonction exclusive, 4
- déchargement, 34
- déduction naturelle, 36

- ensemble, 40, 50
- énumération, 50
- équivalence, 4
- extension, 50

- faux, 4
- fonction, 62
- fonction booléenne, 13
- fonction totale, 64
- forme normale, 45
 - conjonctive, 45
 - disjonctive, 46
- formule
 - atomique, 13
 - fermée, 14
 - propositionnelle, 4

- Hilbert, David, 2
- Hinton, Geoffrey, 2
- Hoare, C.A.R., 121
- Hopcroft, John E., 138
- hypothèses, 33, 34

- image, 56
- implication inverse, 4
- interprétation, 39

- Kahan, William M., 2
- Kay, Alan C., 44

- Leuschel, Michael, 49

- modèle, 40
- Moore, Edward F., 138

- NP-complet, 8
- Nygaard, Kristen, 44
- négation, 4

- Peano, Giuseppe, 51
- Peirce, Charles Sanders, 98
- preuve, 34
- preuve par récurrence, 98
- priorité, 5
- prédicat, 13
- prémises, 34

- quantificateur
 - existentiel, 13
 - universel, 13

- relation de déduction, 33
- renommage variable, 15
- Russell, Bertrand, 53
- règle d'inférence
 - cohérence, 42
- règle déduction, complétude, 42

- satisfaction, 40
- satisfaction, formule propositionnelle, 8
- satisfaisable, 40
- ssi, 11
- substitution, 15
- substitution valide, 15
- symbole de fonction, 12
- séquent, 34

- tables de vérité, 7
- Tarski, Alfred, 17
- TarskiUdeS, 17
- tautologie, 28
- terme, 12
- théorie, 43
- Turing, Alan, 2

- valeur de vérité, 7
- valide, 40
- valuation, 39
- variable
 - libre, 14
 - liée, 14
 - propositionnelle, 4
- vrai, 4