# Cours Logique et Calculabilité

L3 Informatique 2013/2014

Texte par Séverine Fratani, avec addenda par Luigi Santocanale
Version du 21 janvier 2014

# Table des matières

# Chapitre 1

# Introduction

## 1.1 Pourquoi la logique ?

### 1.1.1 La formalisation du langage

Le mot *logique* provient du grec *logos* (raison, discours), et signifie "science de la raison". Cette science a pour objets d'étude le discours et le raisonnement. Ceux-ci dépendent bien entendu du langage utilisé. Si on prend le langage courant, on se rend compte facilement :

- qu'il contient de nombreuses ambiguïtés : nous ne sommes pas toujours sûrs de la sémantique d'un énoncé ou d'une phrase. Par exemple : « nous avons des jumelles à la maison ». (Deux filles ou des lunettes optiques ?) Ou « il a trouvé un avocat » (le professionnel ou le fruit ?)
- qu'il est difficile de connaître la véracité d'un énoncé : « il pleuvra demain », « Jean est laid », « la logique c'est dur ».
- qu'il permet d'énoncer des choses paradoxales :
  1. « Je mens » : comme pour tout paradoxe de ce type, on aboutit à la conclusion que si c'est vrai alors c'est faux... et inversement.
  2. « Je suis certain qu'il n'y a rien de certain »
  3. Un arrêté enjoint au barbier (masculin) d'un village de raser tous les hommes du village qui ne se rasent pas eux-mêmes et seulement ceux-ci. Le barbier n'a pas pu respecter cette règle car :
     - s'il se rase lui-même, il enfreint la règle, car le barbier ne peut raser que les hommes qui ne se rasent pas eux-mêmes ;
     - s'il ne se rase pas lui-même (qu'il se fasse raser ou qu'il conserve la barbe), il est en tort également, car il a la charge de raser les hommes qui ne se rasent pas eux-mêmes.
  4. Paradoxe de Russel : c'est la version mathématique du paradoxe du barbier : soit $a$ l'ensemble des ensembles qui ne se contiennent pas eux-même. Cet ensemble n'existe pas car on peut vérifier que $a \in a$ ssi $a \notin a$.

Tout ceci fait que les langues naturelles ne sont pas adaptées au raisonnement formel. C'est pourquoi par exemple, on vous a appris un langage spécifique pour faire des preuves en mathématique. Une preuve mathématique ne peut être faite en utilisant tout le vocabulaire de la langue naturelle car les énoncés et les preuves deviendraient alors ambiguës. Le langage utilisé en mathématique est celui de la logique classique (en réalité, c'est un langage un peu plus souple, entre la logique classique et la langue naturelle).

### 1.1.2 La formalisation du raisonnement

Une fois le langage formalisé, ce qui intéresse les logiciens c'est le raisonnement, et en particulier, la définition de systèmes formels permettant de mécaniser le raisonnement. Au début du 20ième siècle, le rêve du logicien est de faire de la logique un calcul et de mécaniser le raisonnement et par suite toutes les mathématiques. En 1930, Kurt Gödel met fin à cette utopie en présentant son résultat d'incomplétude : il existe des énoncés d'arithmétique qui ne sont pas prouvable par un

système formel de preuve. Il n'existe donc pas d'algorithme qui permette de savoir si un énoncé mathématique est vrai.

## 1.2 Logique et informatique

Malgré ce résultat négatif, l'arrivée de l'informatique à partir des années 30 marque l'essor de la logique.

Elle est présente dans quasiment tous les domaines de l'informatique :

— vous verrez par exemple en cours d'architecture que votre ordinateur est formé de circuits logiques.

— la programmation n'est au fond que de la logique. Dans les année 60, la correspondance de Curry-Howard, établie une correspondance preuve/programme : une relation entre les démonstrations formelles d'un système logique et les programmes d'un modèle de calcul.

— le traitement automatique des langues,

— l'intelligence artificielle,

— la logique apparait également dans toutes les questions de sureté.
On demande maintenant de plus en plus de prouver la sureté des programmes et des protocoles. Pour cela on modélise les exécutions des programmes, on exprime les propriétés de sûreté par une formule logique, puis on vérifie que les modèles satisfont bien la formule.
A cette effet, d'innombrables logiques ont étés développées, comme les logiques temporelles qui permettent de raisonner sur l'évolution de certains systèmes au cours du temps. Il existe même des logiques pour formaliser les règles des pare-feu, afin d'éviter d'avoir des systèmes de règle incohérents.

— et plein d'autres que j'oublie.

Il existe également des logiciels qui permettent de prouver des formules logique (automatiquement ou semi-automatiquement). En particulier on a des logiciels permettant de générer du code vérifié : on entre une abstraction du programme a réaliser, on prouve sur cette abstraction de façon plus ou moins automatique mais sûre, les propriétés de sûreté souhaitées ; et le logiciel produit du code certifié.

L'informatique est donc indissociable de la logique. Heureusement tout bon informaticien n'est pas obligé d'être un bon théoricien de la logique, mais il doit être capable maitriser son utilisation.

## 1.3 Contenu du cours

On s'intéressera principalement à (des fragments de) la logique classique, qui est la logique utilisée pour les mathématiques, et forme la base de presque toutes les autres logiques.

Nous allons nous intéresser aux fragment suivants :

— la logique propositionnelle ;

— la logique du premier ordre.

Pour chacune de ces logiques, nous nous poserons principalement les questions suivantes :

— Quelle est sa syntaxe ? I.e., comment écrire une phrase dans le langage de la logique considéré.

— Quel est sa sémantique ? C'est-à-dire, étant une phrase, savoir lui attribuer un sens. Cette question ouvre à une autre qui est "de quel forme sont les modèles d'une formule", c'est à dire : mon langage parle d'objets, qui se placent dans un univers précis : quel est cet univers ?

— La logique est elle-décidable ? Etant donné une phrase (formule) du langage, existe-il une procédure effective permettant d'évaluer cette formule.

— Existe-il un système formel de calcul permettant de "prouver" qu'un énoncé est vrai ou faux.

D'autres questions se posent évidement mais se sont principalement celles-ci qui intéressent l'informaticien.

# Chapitre 2

# Calcul propositionnel

## 2.1 Introduction

Les formules (ou phrases, ou énoncés) du calcul propositionnel sont de deux types : ou bien une formule est une *proposition atomique*, ou bien elle est composée à partir d'autres formules à l'aide des connecteurs logiques $\wedge, \neg, \vee, \Rightarrow$ (*et*, *non*, *ou*, *implique*, que l'on appelle *connecteurs propositionnels*).

Considérons, par exemple, l'énoncé arithmétique « 2+2 = 4 *ou* 3+3 = 5 ». Cet énoncé peut se considérer comme construit des propositions atomiques « 2+2 = 4 » et « 3+3 = 5 », via le connecteur propositionnel *ou*. Une analyse similaire peut se faire pour les énoncés du langage naturel. On considère l'énoncé « s'il pleut, alors le soleil se cache », que l'on reconnaîtra être équivalent à « il pleut implique que le soleil se cache », comme obtenu des deux propositions atomiques « s'il pleut » et « le soleil se cache » via le connecteur propositionnel *implique*.

Une proposition atomique est un énoncé simple, ne pouvant prendre que les valeurs "vrai" ou "faux ", et ce de façon non ambiguë ; elle donne donc une information sur un état de chose. De plus une proposition atomique est indécomposable : « le ciel est bleu et l'herbe est verte » n'est pas une proposition atomique mais la composition de deux propositions atomiques. Dans l'analyse du langage naturel, on ne peut pas considérer comme des propositions : les souhaits, les phrases impératives ou les interrogations.

Nous avons déjà vu des exemples de formules composées. Considérons maintenant l'énoncé « s'il neige, alors le soleil se cache et il fait froid ». C'est une formule composée, via le connecteur *implique*, depuis la formule atomique « il neige » et la formule composée « le soleil se cache et il fait froid ». On peut donc composer des formules à partir d'autres formules composées. La valeur de vérité d'une formule composée se calcule comme une fonction des formules dont elle est composée.

Le calcul des propositions est la première étape dans la définition de la logique et du raisonnement. Il définit les règles de déduction qui relient les phrases entre elles, sans en examiner le contenu ; il est ainsi une première étape dans la construction du calcul des prédicats, qui lui s'intéresse au contenu des propositions.

Nous partirons donc en général de faits : "p est vrai, q est faux" et essaierons de déterminer si une affirmation particulière est vraie.

## 2.2 Syntaxe du calcul propositionnel : les formules

Le *langage* du calcul propositionnel est formé de :
— symboles propositionnels $\textsc{Prop} = \{p_1, p_2, \ldots\}$ ;
— connecteurs logiques $\{\neg, \wedge, \vee, \Rightarrow\}$ ;
— symboles auxiliaires : parenthèses et espace.

**Remarque 2.1.** Dans la littérature logique on utilise plusieurs synonymes pour symbole propositionnel ; ainsi *variable propositionnelle*, *proposition atomique*, *formule atomique*, ou encore *atome* sont tous des synonymes de *symbole propositionnel*.

L'ensemble $\mathcal{F}_{\text{cp}}$ des *formules* du calcul propositionnel est le plus petit ensemble tel que :
— tout symbole propositionnel est une formule ;
— si $\varphi$ est une formule alors $\neg\varphi$ est une formule ;
— si $\varphi, \psi$ sont des formules alors $\varphi \vee \psi$, $\varphi \wedge \psi$ et $\varphi \Rightarrow \psi$ sont des formules.

Les symboles auxiliaires ne sont utilisés que pour lever les ambiguïtés possibles : par exemple, la formule $p \vee q \wedge r$ est ambiguë, car elle peut se lire de deux façons différentes, $((p \vee q) \wedge r)$ ou bien $(p \vee (q \wedge r))$.

**Exemple 2.2.** $p$, $p \Rightarrow (q \vee r)$ et $p \vee q$ sont des formules propositionnelles ; $\neg(\vee q)$ et $f(x) \Rightarrow g(x)$ n'en sont pas.

A cause de la structure inductive de la définition, une formule peut-être vue comme un arbre dont les feuilles sont étiquetés par des symboles propositionnels et les noeuds par des connecteurs. Par exemple, la formule $p \Rightarrow (\neg q \wedge r)$ correspond à l'arbre représenté Figure 2.2.

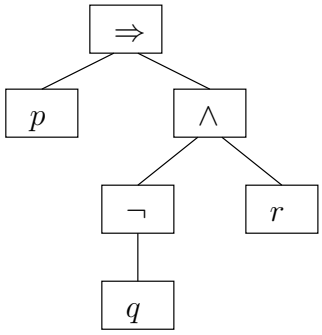



FIGURE 2.1 – Représentation arborescente de la formule $p \Rightarrow (\neg q \wedge r)$

**Notation 2.3.** On utilise souvent en plus le connecteur binaire $\Leftrightarrow$ comme abréviation : $\varphi \Leftrightarrow \psi$ est l'abréviation de $(\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$. De la même façon, on ajoute le symbole $\bot$ qui correspond à *Faux* et le symbole $\top$ qui correspond à *Vrai*. Ces deux symboles sont aussi des abréviations, ils ne sont pas indispensables au langage. (Par exemple $\bot$ peut être utilisé à la place de $p \wedge \neg p$ et $\top$ à la place de $p \vee \neg p$.)

**Définition 2.4** (Sous-formule)**.** L'ensemble $SF(\varphi)$ des sous-formules d'une formules $\varphi$ est défini par induction de la façon suivante.
— $SF(p) = \{p\}$ ;
— $SF(\neg\varphi) = \{\neg\varphi\} \cup SF(\varphi)$ ;
— $SF(\varphi \circ \psi) = \{\varphi \circ \psi\} \cup SF(\varphi) \cup SF(\psi)$ (où $\circ$ désigne un des symboles $\wedge$, $\vee$, $\Rightarrow$).

Par exemple, $SF(p \Rightarrow (\neg q \wedge r)) = \{p, q, r, \neg q, \neg q \wedge r, p \Rightarrow (\neg q \wedge r)\}$. Quand on voit une formule comme un arbre, une sous-formule est simplement un sous-arbre (voir Figure 2.2).

**Définition 2.5** (Sous-formule stricte)**.** $\psi$ est une sous-formule stricte de $\varphi$ si $\psi$ est une sous-formule de $\varphi$ qui n'est pas $\varphi$.

## 2.3 Sémantique du calcul propositionnel

Il faut maintenant un moyen de déterminer si une formule est vraie ou fausse. La première étape est de donner une valeur de vérité aux propositions atomiques. L'évaluation d'une formule, dépend donc des valeurs choisies pour les symboles propositionnels. Ces valeurs sont données par une **valuation**.

**Définition 2.6** (Valuation)**.** Une valuation est une application de PROP dans $\{0, 1\}$. La valeur 0 désigne le "faux" et la valeur 1 désigne le "vrai".

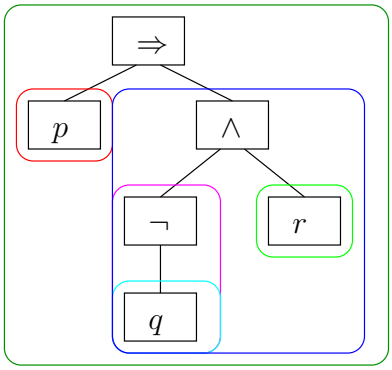



FIGURE 2.2 – Représentation arborescente des sous-formules de $p \Rightarrow (\neg q \wedge r)$

Une valuation sera souvent donnée sous forme d'un tableau. Par exemple, si PROP $= \{p, q\}$ alors la valuation $v : p \mapsto 1,\ q \mapsto 0$ s'écrit plus simplement $v$ :

| $p$ | $q$ |
|---|---|
| 1 | 0 |

Une fois la valuation $v$ choisie, la valeur de la formule se détermine de façon naturelle, par extension de la valuation $v$ aux formules de la façon suivante :

**Définition 2.7** (Valeur d'une formule)**.**
- $v(\neg\varphi) = 1$ ssi $v(\varphi) = 0$ ;
- $v(\varphi \vee \psi) = 1$ ssi $v(\varphi) = 1$ ou $v(\psi) = 1$ ;
- $v(\varphi \wedge \psi) = 1$ ssi $v(\varphi) = 1$ et $v(\psi) = 1$ ;
- $v(\varphi \Rightarrow \psi) = 0$ ssi $v(\varphi) = 1$ et $v(\psi) = 0$.

La définition précédente peut apparaître trompeuse car circulaire : afin d'expliquer la logique, nous somme en train de l'utiliser (*ssi*, *ou*, *et* ...). Par ailleurs, on peut se servir de la définition suivante qui est en effet équivalente à la Définition 2.7 :

**Définition 2.8** (Valeur d'une formule (bis))**.**
- $v(\neg\varphi) = 1 - v(\varphi)$ ;
- $v(\varphi \vee \psi) = \max(v(\varphi), v(\psi))$ ;
- $v(\varphi \wedge \psi) = \min(v(\varphi), v(\psi))$ ;
- $v(\varphi \Rightarrow \psi) = v(\neg\varphi \vee \psi)$.

Cette dernière définition est purement combinatoire car elle repose sur la structure de l'ensemble ordonné *fini* $\{\, 0 < 1 \,\}$ ; nous supposons en fait que cette structure est évidente et claire par soi-même qu'il n'y a pas besoin de la justifier par d'autres moyens.

**Exercice 2.9.** Proposez un algorithme qui, étant donné une formule $\varphi$ du calcul propositionnel et une valuation $v$, calcule $v(\varphi)$. Quel type de structure de données utiliser pour coder les formules ? Quel type de structure de données utiliser pour coder les valuations ?

Notez que la définition 2.8 correspond aux tables de vérité des connecteurs logiques (dont vous avez sûrement entendu parler) :

| $p$ | $\neg p$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

| $p$ | $q$ | $p \wedge q$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| $p$ | $q$ | $p \vee q$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| $p$ | $q$ | $p \Rightarrow q$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Remarque 2.10** (Langage naturel et langage formel)**.** Remarquez la définition particulière de l'implication : on l'entend en général comme un "si ..., alors ...", on voit ici que l'énoncé "si 1+1=1, alors la capitale de la France est Marseille" est vrai, puisque toute phrase $\varphi \Rightarrow \psi$ est vraie dès

lors que $\varphi$ est évaluée à faux. Ceci est peu naturel, car dans le langage courant, on ne s'intéresse à la vérité d'un tel énoncé que lorsque la condition est vraie : "s'il fait beau je vais à la pêche" n'a d'intérêt pratique que s'il fait beau... Attribuer la valeur vrai dans le cas ou la prémisse est fausse correspond a peu près à l'usage du si .. alors dans la phrase suivante : "Si Pierre obtient sa Licence, alors je suis Einstein" : c'est à dire que partant d'une hypothèse fausse, alors je peux démontrer des choses fausses (ou vraies). Par contre, il n'est pas possible de démontrer quelque chose de faux partant d'une hypothèse vraie.

D'autres exemples où il est difficile de coder le langage naturel via le langage formel :

— comment coderiez vous, en langage formel, l'énoncé français « Soit il est frais, soit il est chaud » ?

— et comment coderiez vous l'énoncé anglais « Either I cannot understand French, or my professor doesn't know how to speak it » ?

On peut ajouter la définition de la valeur de l'abréviation $\Leftrightarrow$ : $v(\varphi \Leftrightarrow \psi) = 1$ ssi $v(\varphi) = v(\psi)$. Ce qui correspond à la table de vérité suivante :

| $p$ | $q$ | $p \Leftrightarrow q$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Exercice 2.11.** Définissons l'ensemble $\textsc{Prop}(\varphi)$, des variables propositionnelles contenues dans $\varphi \in \mathcal{F}_{\text{cp}}$, par induction comme suit :

$$\begin{aligned} \textsc{Prop}(p) &= \{\, p \,\}, \\ \textsc{Prop}(\neg\varphi) &= \textsc{Prop}(\varphi), \\ \textsc{Prop}(\varphi \circ \psi) &= \textsc{Prop}(\varphi) \cup \textsc{Prop}(\psi), \qquad\qquad \circ \in \{\, \wedge, \vee, \Rightarrow \,\}. \end{aligned}$$

Montrez que :

— $\textsc{Prop}(\varphi) = \textsc{Prop} \cap SF(\varphi)$, pour tout $\varphi \in \mathcal{F}_{\text{cp}}$ ;

— si $v(p) = v'(p)$ pour tout $p \in \textsc{Prop}(\varphi)$, alors $v(\varphi) = v'(\varphi)$.

### 2.3.1 Modèles d'une formule

**Définition 2.12.** L'ensemble des **valuations** d'un ensemble de variables propositionnelles Prop est noté $\mathsf{Val}(\textsc{Prop})$ (ou juste $\mathsf{Val}$ lorsqu'il n'y a pas d'ambiguité sur Prop). $\mathsf{Val}(\textsc{Prop})$ est donc l'ensemble des fonctions de Prop dans $\{0, 1\}$.

Par exemple, si $\textsc{Prop} = \{p, q, r\}$, alors $\mathsf{Val}$ est représenté par la Table 2.3.1, dans lequel chaque ligne est une valuation de Prop :

| $p$ | $q$ | $r$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| 1 | 1 | 1 |

Table 2.1 – L'ensemble $\mathsf{Val}(\textsc{Prop})$ des valuations de $\textsc{Prop} = \{p, q, r\}$

**Définition 2.13** (Modèle d'une formule)**.** Un **modèle** de $\varphi$ est une valuation $v$ telle que $v(\varphi) = 1$. On note $\mathsf{mod}(\varphi)$ l'ensemble des modèles de $\varphi$.

**Exemple 2.14.** Si $\textsc{Prop} = \{p, q, r\}$ et $\varphi = (p \vee q) \wedge (p \vee \neg r)$ alors l'ensemble des modèles de $\varphi$ est

$$\mathsf{mod}(\varphi) = \begin{array}{|c|c|c|}\hline p & q & r \\ \hline 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ \hline \end{array}$$

**Définition 2.15** (Satisfaisabilité)**.** Une formule $\varphi$ est **satisfaisable** (ou **consistante**, ou encore **cohérente**) si elle admet un modèle (*i.e.*, s'il existe une valuation $v$ telle que $v(\varphi) = 1$, *i.e.* si $\mathsf{mod}(\varphi) \neq \varnothing$).

**Définition 2.16** (Insatisfaisabilité)**.** Une formule $\varphi$ est **insatisfaisable** (ou **inconsistante**, ou **incohérente**) si elle n'admet aucun modèle (*i.e.*, si pour toute valuation $v$, $v(\varphi) = 0$, *i.e.*, si $\mathsf{mod}(\varphi) = \varnothing$)

**Définition 2.17** (Tautologie)**.** Une formule $\varphi$ est une **tautologie** (ou **valide**) si $v(\varphi) = 1$ pour toute valuation $v$ (*i.e.*, si $\mathsf{mod}(\varphi) = \mathsf{Val}$). On note $\models \varphi$ pour dire que $\varphi$ est une tautologie.

Un exemple de tautologie est $\varphi \vee \neg\varphi$, c'est à dire le *tiers exclus*.

**Exercice 2.18.** Montrez que les formules suivantes sont des tautologies :

$$p \Rightarrow p\,, \qquad p \Rightarrow (q \Rightarrow p)\,, \qquad (p \Rightarrow (q \Rightarrow r)) \Rightarrow ((p \Rightarrow q) \Rightarrow (p \Rightarrow r))\,, \qquad ((p \Rightarrow q) \Rightarrow p) \Rightarrow p\,.$$

**Définition 2.19** (Equivalence)**.** On dit que $\varphi$ **est équivalente à** $\psi$ si les deux formules ont les mêmes modèles (*i.e.* si $\mathsf{mod}(\varphi) = \mathsf{mod}(\psi)$). On note alors $\varphi \equiv \psi$.

**Exemple 2.20.** Les opérateurs $\wedge$, $\vee$ sont associatifs-commutatifs. Deux formules identiques à associativité-commutativité près sont équivalentes. Remplacer une sous-formule $\psi$ d'une formule $\varphi$ par une formule équivalente $\psi'$ donne une formule notée $\varphi[\psi \leftarrow \psi']$. Cette substitution préserve les modèles, *i.e.*, $\mathsf{mod}(\varphi) = \mathsf{mod}(\varphi[\psi \leftarrow \psi'])$.

**Exercice 2.21.** Prouvez les équivalences suivantes :

$$\begin{array}{lll}
\varphi \vee \bot \equiv \varphi & \varphi \wedge \bot \equiv \bot & (\varphi \wedge \psi) \wedge \theta \equiv \varphi \wedge (\psi \wedge \theta) \\
\varphi \vee \top \equiv \top & \varphi \wedge \top \equiv \varphi & (\varphi \wedge \psi) \vee \theta \equiv (\varphi \wedge \theta) \vee (\psi \wedge \theta) \\
\varphi \vee \psi \equiv \psi \vee \varphi & \varphi \wedge \psi \equiv \psi \wedge \varphi & (\varphi \vee \psi) \wedge \theta \equiv (\varphi \wedge \theta) \vee (\varphi \wedge \theta) \\
\varphi \vee \varphi \equiv \varphi & \neg(\varphi \vee \psi) \equiv (\neg\varphi) \wedge (\neg\psi) & (\varphi \vee \psi) \vee \theta \equiv \varphi \vee (\psi \vee \theta) \\
\neg\neg\varphi \equiv \varphi & \varphi \wedge \varphi \equiv \varphi & \neg(\varphi \wedge \psi) \equiv (\neg\varphi) \vee (\neg\psi)\,.
\end{array}$$

**Proposition 2.22.** *Soient $\varphi$ et $\psi$ deux formules, on a :*

1. $\mathsf{mod}(\neg\varphi) = \mathsf{Val} - \mathsf{mod}(\varphi)$ *;*
2. $\mathsf{mod}(\varphi \vee \psi) = \mathsf{mod}(\psi) \cup \mathsf{mod}(\psi)$ *;*
3. $\mathsf{mod}(\varphi \wedge \psi) = \mathsf{mod}(\psi) \cap \mathsf{mod}(\psi)$ *;*
4. $\models \varphi \Rightarrow \psi$ *ssi* $\mathsf{mod}(\varphi) \subseteq \mathsf{mod}(\psi)$.

*Démonstration.* 1. pour toute valuation $v \in \mathsf{Val}$,

$$\begin{array}{rll}
v \in \mathsf{mod}(\neg\varphi) & \text{ssi} & v(\neg\varphi) = 1 \\
& \text{ssi} & v(\varphi) = 0 \\
& \text{ssi} & v \notin \mathsf{mod}(\varphi) \\
& \text{ssi} & \mathsf{Val} - \mathsf{mod}(\varphi)
\end{array}$$

2. pour toute valuation $v \in \mathsf{Val}$,

$$\begin{array}{rll}
v \in \mathsf{mod}(\varphi \vee \psi) & \text{ssi} & v(\varphi) = 1 \text{ ou } v(\psi) = 1 \\
& \text{ssi} & v \in \mathsf{mod}(\varphi) \text{ ou } v \in \mathsf{mod}(\psi) \\
& \text{ssi} & \mathsf{mod}(\varphi) \cup \mathsf{mod}(\psi)
\end{array}$$

3. pour toute valuation $v \in \mathsf{Val}$,

$$\begin{array}{lll} v \in \mathsf{mod}(\varphi \wedge \psi) & \text{ssi} & v(\varphi) = 1 \text{ et } v(\psi) = 1 \\ & \text{ssi} & v \in \mathsf{mod}(\varphi) \text{ et } v \in \mathsf{mod}(\psi) \\ & \text{ssi} & \mathsf{mod}(\varphi) \cap \mathsf{mod}(\psi) \end{array}$$

4.

$$\begin{array}{lll} \models \varphi \Rightarrow \psi & \text{ssi} & \text{pour toute valuation } v \in \mathsf{Val}, v(\varphi \Rightarrow \psi) = 1 \\ & \text{ssi} & \text{pour toute valuation } v \in \mathsf{Val}, v(\neg\varphi \vee \psi) = 1 \\ & \text{ssi} & \text{pour toute valuation } v \in \mathsf{Val}, v(\varphi) = 0 \text{ ou } v(\psi) = 1 \\ & \text{ssi} & \text{pour toute valuation } v \in \mathsf{Val}, v(\varphi) \leq v(\psi) \\ & \text{ssi} & \mathsf{mod}(\varphi) \subseteq \mathsf{mod}(\psi) \quad \square \end{array}$$

**Définition 2.23** (Conséquence logique)**.** Une formule $\psi$ est **conséquence logique** d'une formule $\varphi$ si tout modèle de $\varphi$ est un modèle de $\psi$ (*i.e.*, si $\mathsf{mod}(\varphi) \subseteq \mathsf{mod}(\psi)$). On note alors $\varphi \models \psi$.

**Remarque 2.24.** Attention à la confusion dans les deux notations !
— $v \models \varphi$ où $v$ est une valuation, i.e., l'assignation d'une valeur aux propositions atomiques de la formule ; c'est un raccourcis assez fréquent pour $v(\varphi) = 1$ ;
— $\psi \models \varphi$ où $\psi$ est une formule.

**Proposition 2.25.** *Soient $\varphi$ et $\psi$ deux formules propositionnelles.*

1. *$\varphi \models \psi$ si et seulement si $\models \varphi \Rightarrow \psi$.*
2. *$\varphi \equiv \psi$ si et seulement si $\models \varphi \Leftrightarrow \psi$.*

*Démonstration.* 1. Conséquence directe du point 4 de la Proposition 2.22

2. $\models \varphi \Leftrightarrow \psi$ ssi $\forall v \in \mathsf{Val}$, $v(\varphi \Leftrightarrow \psi) = 1$ (par la définition de tautologie) ssi $\forall v \in \mathsf{Val}$, $v(\varphi) = v(\psi)$ (par la table de verité de $\Leftrightarrow$) ssi $\forall v \in \mathsf{Val}$, $v \in \mathsf{mod}(\varphi)$ ssi $v \in \mathsf{mod}(\psi)$ ssi $\mathsf{mod}(\varphi) = \mathsf{mod}(\psi)$ ssi $\psi \equiv \varphi$. $\square$

### 2.3.2 La conséquence logique (d'un ensemble de formules)

Les formules propositionnelles peuvent être vues comme des contraintes sur les propositions atomiques. Par exemple, $p \wedge q$ contraint $p$ et $q$ à être vraies, où $p \Rightarrow q$ contraint $q$ à être vraie toute fois que $p$ est vraie. Il est donc très courant de considérer des ensembles de formules propositionnelles pour modéliser des problèmes de satisfaction de contraintes. Une valuation satisfaisant toute formule de l'ensemble pourra donc se considérer comme une solution du problème.

On étend les définitions vues précédemment aux ensembles de formules.

**Définition 2.26** (Modèle)**.** Un **modèle** d'un ensemble de formules $\Gamma$ est une valuation $v$ telle que $v(\varphi) = 1$ pour tout $\varphi \in \Gamma$. On note $\mathsf{mod}(\Gamma)$ l'ensemble des modèles de $\Gamma$.

Cet ensemble de modèles est donc l'ensemble des valuations qu'on peut attribuer aux variables si on veut respecter toutes les contraintes de $\Gamma$.

**Définition 2.27** (Satisfaisabilité/Consistance, Insatisfaisabilité/Contradiction)**.** Un ensemble de formules $\Gamma$ est
— **satisfaisable** (ou **consistant**, ou **cohérent**) s'il admet au moins un modèle (*i.e.*, si $\mathsf{mod}(\Gamma) \neq \varnothing$) ;
— **insatisfaisable** (ou **contradictoire**, ou **inconsistant**, ou encore **incohérent**) s'il n'admet aucun modèle ( *i.e.*, si $\mathsf{mod}(\Gamma) = \varnothing$), on note alors $\Gamma \models \bot$.

Un ensemble $\Gamma$ contradictoire ne peut être satisfait : par exemple l'ensemble $\Gamma = \{p, \neg p\}$ est insatisfaisable.

**Définition 2.28** (Conséquence logique)**.** Une formule $\varphi$ est conséquence logique de $\Gamma$ si et seulement si toute valuation qui donne 1 à toutes les formules de $\Gamma$ donne 1 à $\varphi$ (*i.e.*, si $\mathsf{mod}(\Gamma) \subseteq \mathsf{mod}(\varphi)$), on note alors $\Gamma \models \varphi$. On note cons($\Gamma$) l'ensemble des conséquences logiques de $\Gamma$.

**Remarque 2.29.** Attention aux deux notations :
— $\psi \models \varphi$ où $\psi$ est une formule ;
— $\Gamma \models \varphi$ où $\Gamma$ est un ensemble de formule.
Remarquez, par ailleurs que $\varphi \models \psi$ si, et seulement si, $\{\varphi\} \models \psi$ ; les deux dernières notations sont donc cohérentes entre elles.

Voici des relations élémentaires entre les relations que nous venons de présenter.

**Proposition 2.30.** $\Gamma \models \varphi$ *ssi* $\Gamma \cup \{\neg\varphi\}$ *est contradictoire.*

*Démonstration.* $\Gamma \models \varphi$ ssi
pour toute valuation $v$
— soit $v$ est un modèle de $\Gamma$ et $v(\varphi) = 1$
— soit $v$ n'est pas un modèle de $\Gamma$
ssi pour toute valuation $v$
— soit $v$ est un modèle de $\Gamma$ et $v(\neg\varphi) = 0$
— soit $v$ n'est pas un modèle de $\Gamma$
ssi pour toute valuation $v$, $v$ n'est pas un modèle de $\Gamma \cup \{\neg\varphi\}$
ssi $\Gamma \cup \{\neg\varphi\}$ est contradictoire. □

**Proposition 2.31.** *Pour tous ensembles de formules* $\Sigma$, $\Gamma$,

$$\mathit{mod}(\Sigma \cup \Gamma) = \mathit{mod}(\Sigma) \cap \mathit{mod}(\Gamma).$$

*En particulier, si* $\Sigma \subseteq \Gamma$ *alors* $\mathit{mod}(\Gamma) \subseteq \mathit{mod}(\Sigma)$.

Cette proposition se comprend bien si on voit un ensemble de formules comme un ensemble de contraintes sur les variables propositionnelles. Plus on ajoute de contraintes, et moins il reste de possibilités pour résoudre ces contraintes.

*Démonstration de la Proposition 2.31.* Pour toute valuation $v$ :

$$\begin{aligned} v \in \mathsf{mod}(\Sigma \cup \Gamma) \text{ ssi } & \text{ pour tout } \varphi \in \Sigma \cup \Gamma, v(\varphi) = 1 \\ \text{ssi } & \text{ pour tout } \varphi \in \Sigma, v(\varphi) = 1 \text{ et pour tout } \psi \in \Gamma,\ v(\psi) = 1 \\ \text{ssi } & v \in \mathsf{mod}(\Sigma) \text{ et } v \in \mathsf{mod}(\Gamma) \\ \text{ssi } & v \in \mathsf{mod}(\Sigma) \cap \mathsf{mod}(\Gamma). \end{aligned}$$

□

La preuve de la Proposition suivante est laissée en exercice.

**Proposition 2.32.** *Si* $\Gamma' \subseteq \Gamma$ *et* $\Gamma' \models \varphi$, *alors* $\Gamma \models \varphi$.

**Proposition 2.33.** $\{\varphi_1, \ldots \varphi_n\} \models \psi$ *ssi* $\models (\varphi_1 \wedge \ldots \wedge \varphi_n) \Rightarrow \psi$

Cette proposition exprime le fait qu'un ensemble **fini** de formules propositionnelles peut toujours être vu comme une seule formule formée de la conjonction des formules de l'ensemble. Une formule étant un objet fini, ce résultat ne se généralise pas au cas des ensembles de taille non bornée. Dans le cas où $\Gamma$ est infini, il faudra utiliser le théorème de compacité (Théorème 2.39) qui permet de ramener les problèmes de satisfaisabilité et de contradiction d'un ensemble de taille quelconque à celle d'ensemble finis.

*Démonstration.* Remarquons que

$$\begin{aligned} \mathsf{mod}(\{\varphi_1, \ldots, \varphi_n\}) &= \mathsf{mod}(\{\varphi_1\} \cup \ldots \cup \{\varphi_n\}) \\ &= \mathsf{mod}(\{\varphi_1\}) \cap \ldots \cap \mathsf{mod}(\{\varphi_n\}) && \text{(par la Proposition 2.31)} \\ &= \mathsf{mod}(\varphi_1) \cap \ldots \cap \mathsf{mod}(\varphi_n) && \text{(par la Remarque 2.29)} \\ &= \mathsf{mod}(\varphi_1 \wedge \ldots \wedge \varphi_n). && \text{(par la Propositon 2.22)} \end{aligned}$$

Donc, on a que $\mathsf{mod}(\{\varphi_1, \ldots, \varphi_n\}) \subseteq \mathsf{mod}(\psi)$ si et seulement si $\mathsf{mod}(\varphi_1 \wedge \ldots \wedge \varphi_n) \subseteq \mathsf{mod}(\psi)$ et, par la Proposition 2.22, la dernière relation est vraie ssi $(\varphi_1 \wedge \ldots \wedge \varphi_n) \Rightarrow \psi$ est une tautologie. □

**Proposition 2.34.** $\Gamma \models \varphi$ *si, et seulement si,* $\mathsf{mod}(\Gamma) = \mathsf{mod}(\Gamma \cup \{\varphi\})$.

La proposition peut se comprendre comme suit. Une conséquence logique $\varphi$ d'un ensemble $\Gamma$ est une nouvelle contrainte déduite directement de $\Gamma$. Puisqu'elle découle de $\Gamma$, elle ne peut pas apporter des "vraies" contraintes supplémentaires que celles apportées par $\Gamma$. Cela signifie que les modèles de $\Gamma$ et ceux de $\Gamma \cup \{\varphi\}$ sont exactement les mêmes.

*Démonstration de la Proposition 2.34.* La proposition découle du fait que $\mathsf{mod}(\Gamma \cup \{\varphi\}) = \mathsf{mod}(\Gamma) \cap \mathsf{mod}(\{\varphi\})$, et que la relation $\mathsf{mod}(\Gamma) \subseteq \mathsf{mod}(\varphi)$ est équivalente à $\mathsf{mod}(\Gamma) \cap \mathsf{mod}(\{\varphi\}) = \mathsf{mod}(\Gamma)$. □

**Exemple 2.35.** L'ensemble $\Gamma = \{(p \Rightarrow s) \vee q, \neg q\}$ possède comme conséquence logique $p \Rightarrow s$. Bien entendu, les modèles de $\Gamma$ sont exactement les modèles de $\Gamma \cup \{p \Rightarrow s\}$.

La Proposition 2.34 implique également une méthode de simplification d'un ensemble de formules : si $\Gamma$ contient une formule $\varphi$ conséquence logique de $\Gamma - \{\varphi\}$, alors $\varphi$ peut être retirée de l'ensemble de contraintes $\Gamma$ sans en modifier la sémantique, $\mathsf{mod}(\Gamma) = \mathsf{mod}(\Gamma - \{\varphi\})$. L'exemple suivant éclaircit cette méthode.

**Exemple 2.36.** Avec cet exemple, nous allons tirer avantage des propositions et remarques précédentes pour résoudre un ensemble de contraintes ayant une certaine complexité.

On dispose de 4 variables propositionnelles, $p_A, p_B, p_C, p_D$, qui obéissent aux contraintes suivantes :

$\varphi_1 : p_B \wedge \neg p_C$
$\varphi_2 : p_A \Rightarrow (p_C \vee p_D)$
$\varphi_3 : \neg p_C \wedge (p_B \vee p_A)$

Soit $\Gamma_1 = \{\varphi_1, \varphi_2, \varphi_3\}$, cet ensemble forme l'ensemble des prémisses à partir desquelles nous allons essayer de déduire les valeurs que les variables propositionnelles peuvent prendre.

On se pose les questions suivantes :

1. *Peut-on simplifier l'ensemble* $\Gamma_1$ *de façon à ne pas changer l'ensemble de ses modèles, et donc de ses conséquences ?*
   (a) On remarque que la contrainte $\varphi_3$ est une **conséquence logique** de la contrainte $\varphi_1$.
   En effet, pour toute valuation $v$,
   — $v$ satisfait $\varphi_1$ ssi $v(p_C) = 0$ et $v(p_B) = 1$,
   — $v$ satisfait $\varphi_3$ ssi $v(p_C) = 0$ et ($v(p_A) = 1$ ou $v(p_B) = 1$).
   D'où, $\mathsf{mod}(\varphi_1) \subseteq \mathsf{mod}(\varphi_3)$.
   (b) Soit $\Gamma_2 = \{\varphi_1, \varphi_2\}$, on a $\mathsf{mod}(\Gamma_2) = \mathsf{mod}(\Gamma_1)$. En effet, par définition,
   $\mathsf{mod}(\Gamma_1) = \mathsf{mod}(\varphi_1) \cap \mathsf{mod}(\varphi_2) \cap \mathsf{mod}(\varphi_3) = \mathsf{mod}(\varphi_1) \cap \mathsf{mod}(\varphi_2) = \mathsf{mod}(\Gamma_2)$.
   Donc les conséquences de $\Gamma_1$ et $\Gamma_2$ sont les mêmes et on peut alors simplifier $\Gamma_1$ par $\Gamma_2$.
2. *Quel est l'ensemble des modèles de* $\Gamma_2$ *?*

Modèles de $\varphi_1$ :

| $p_A$ | $p_B$ | $p_C$ | $p_D$ |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 |

Modèles de $\varphi_2$ :

| $p_A$ | $p_B$ | $p_C$ | $p_D$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 |

L'ensemble $M$ des modèles de $\Gamma_2$ est l'intersections des modèles de $\varphi_1$ et $\varphi_2$ :

| $p_A$ | $p_B$ | $p_C$ | $p_D$ |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 |

3. $\Gamma_2$ *est-il consistant ? contradictoire ?*
   $\Gamma_2$ admet un modèle, il est donc consistant et non contradictoire.
4. *Quelles conséquences logiques pouvons nous tirer de l'ensemble* $\Gamma_2$ *?*
   Les conséquences logiques de $\Gamma_2$ sont toutes les formules dont l'ensemble des modèles contient $M$. On a donc entre autres :
   $\neg p_C$, $p_B$, $p_B \wedge \neg p_C \in \text{cons}(\Gamma)$
5. Ajoutons maintenant une nouvelle contrainte : $\Gamma_3 = \{\neg p_B \wedge \neg p_D\} \cup \Gamma_2$. On a $\mathsf{mod}(\Gamma_3) = \mathsf{mod}(\Gamma_2) \cap \mathsf{mod}(\neg p_B \wedge \neg p_D) = \varnothing$. Donc $\mathsf{mod}(\Gamma_3) = \varnothing$ et $\Gamma_3$ est contradictoire.

#### 2.3.2.1 Compacité

Le théorème de compacité sert à caractériser la conséquence logique dans les cas où l'ensemble des formules est infini en ne considérant que des sous-ensembles finis. Par ailleurs, ce théorème jouera un rôle cruciale plus tard, dans le cadre de la preuve de complètude pour la calcul de la résolution (Théorème 3.66).

**Le théorème de compacité** Pour commencer, nous avons besoin du lemme suivant appelé Lemme de König.

**Lemme 2.37** (König)**.** *Tout arbre infini à branchement fini possède une branche infinie.*

*Démonstration.* Supposons que $T$, qui est à branchement fini, soit infini. On définit une branche infinie dans $T$, ce qui mènera à la conclusion. Pour construire cette branche, on montre, par induction sur les entiers, que la propriété suivante :

$$P(n) ::= \text{ il existe une branche } e_0, \ldots, e_n \text{ telle que le sous arbre issu de } e_n \text{ est infini}$$

est vraie de tout entier $n \geq 0$.

***(Base de l'induction).*** Pour $n = 0$, on choisit $e_0 = r$ (la racine de l'arbre) qui par hypothèse est la racine d'un arbre infini.

***(Étape inductive).*** On suppose $P(n)$ vraie et on montre $P(n+1)$. Par hypothèse, il existe une branche $e_0, \ldots, e_n$ telle que le sous arbre issu de $e_n$ est infini. Considérons les successeurs immédiats de $e_n$, disons $e_{n_1}, \ldots, e_{n_k}$ : si tous étaient racines de sous-arbres finis, disons de cardinaux $p_1, \ldots, p_k$, alors il en serait de même de $e_n$ (avec un cardinal d'au plus $p_1 + \ldots + p_{k+1}$), contradiction. Donc l'un d'entre eux est le $e_{n+1}$ recherché. □

Nous aurons besoin du Lemme dans la forme suivante :

**Lemme 2.38.** *Tout arbre a branchement fini et dont toutes les branches sont finies, est fini.*

**Théorème 2.39** (Compacité)**.** *Un ensemble de formules propositionnelles* $\Gamma$ *est satisfaisable ssi tout sous-ensemble fini de* $\Gamma$ *est satisfaisable.*

Par contraposée, le Théorème de compacité peut s'énoncer de la façon suivante :

**Théorème 2.40** (Compacité)**.** *Un ensemble de formules propositionnelles* $\Gamma$ *est contradictoire si, et seulement si, il existe un sous-ensemble fini de* $\Gamma$ *contradictoire.*

Remarquons que l'implication « si un sous-ensemble fini de $\Gamma$ est contradictoire, alors $\Gamma$ est contradictoire » est trivialement vraie. Nous nous limiterons à prouver l'implication inverse.

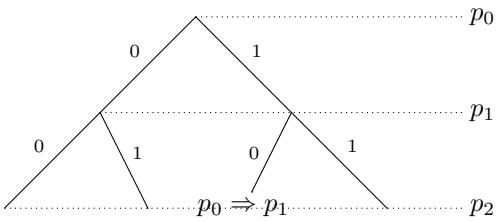



FIGURE 2.3 – Début de l'arbre sémantique avec le noeud d'échec 10 étiqueté

*Démonstration.* On fait la preuve dans le cas où $\text{PROP} = \{p_0, p_1, p_2, \ldots, p_n, \ldots\}$ est un ensemble dénombrable.

Nous avons besoin d'une construction importante appelée « arbre sémantique » ou « arbre de Herbrand ». L'arbre sémantique associé est un arbre binaire infini dont toutes les arêtes à gauches sont étiquetées par 0 (le « faux ») et celles à droites sont étiquetées par 1 (le « vrai »). Chaque niveau de l'arbre est associé à une proposition. La racine (le niveau 0) est associé à $p_0$ : chaque fois que l'on descend d'un noeud de niveau $i$, ceci revient à poser $p_i$ faux si l'on descend à gauche, et $p_i$ vrai si l'on descend à droite. Remarquons que :

1. chaque chemin infini partant de la racine correspond à une valuation de l'ensemble des propositions ;
2. chaque noeud $e$ à profondeur $n$ correspond à une valuation $v_e$ des variables $\{p_0, \ldots p_{n-1}\}$.

Nous appelons un noeud $e$ de l'árbre *noeud d'échec* (par rapport à $\Gamma$) s'il existe une formule $\varphi_e \in \Gamma$ telle que $\text{PROP}(\varphi_e) \subseteq \{ p_0, \ldots p_{n-1} \}$ et $v_e(\varphi_e) = 0$, où $n$ est la profondeur du noeud $e$.

On suppose que $\Gamma$ est inconsistante et on montre qu'il existe un sous-ensemble $\Gamma_0 \subseteq \Gamma$ fini et inconsistant.

On commence par remarquer que chaque branche contient un noeud d'échec. En effet, si une branche n'en contient pas, elle définit un modèle de $\Gamma$, ce qui est contradictoire à l'hypothèse.

On peut donc faire la construction suivante : prenons le premier noeud d'échec de chaque branche et étiquetons ce noeud par une formule de $\Gamma$ fausse sur ce noeud, puis coupons l'arbre au niveau du noeud d'echec. (Plus formellement, si $\pi$ est une branche de l'arbre sémantique, dénotons par $e(\pi)$ le premier noeud d'échec sur cette branche, et choisissons $\varphi_{e(\pi)} \in \Gamma$ tel que $v_{e(\pi)}(\varphi_{e(\pi)}) = 0$ ; ensuite, coupons l'arbre de façon que les noeuds $e(\pi)$ deviennent des feuilles de l'arbre.)

L'arbre obtenu en tronquant ainsi toutes les branches est un arbre à branchement fini, ses branches sont finies, et donc il est fini par le Lemme de König. Le sous-ensemble $\Gamma_0 = \{ \varphi_{e(\pi)} \mid \pi$ une branche de l'arbre sémantique $\}$ des formules de $\Gamma$ étiquetant les feuilles de l'arbre est donc fini. Or toutes les feuilles de l'arbre sont des noeuds d'échec et donc chacune des valuations rend fausse au moins une des formules de $\Gamma_0$. (Si $v \in \mathsf{Val}$, alors $v = v_\pi$ pour une branche $\pi$ de l'arbre, et donc $v(\varphi_{e(\pi)}) = v_{e(\pi)}(\varphi_{e(\pi)}) = 0$.) L'ensemble $\Gamma_0 \subseteq \Gamma$ est donc fini et inconsistant. □

**Exemple 2.41.** Supposons que $\Gamma$ soit de la forme $\{ p_0 \wedge \neg p_1, p_0 \Rightarrow p_1, \ldots \}$. Alors le noeud $e = 10$ de l'arbre sémantique est un noeud d'échec par rapport à cet ensemble $\Gamma$, car $p_0 \Rightarrow p_1 \in \Gamma$ and $v_e(p_0 \Rightarrow p_1) = 0$. (Le debut de) l'arbre sémantique, avec le noeud 10 étiqueté par la formule témoignant son échec, est représenté en Figure 2.3.

**Remarque 2.42.** On peut donner une preuve plus simple du théorème de compacité en utilisant les propriétés des systèmes de preuves. Nous la donnerons par la suite, lorsque nous aurons les outils nécessaires.

Corollaire du théorème de compacité :

**Corollaire 2.43.** *Une formule $\varphi$ est conséquence d'un ensemble de formules $\Gamma$ si et seulement s'il existe un sous-ensemble fini $\Gamma_{fini}$ de $\Gamma$ tel que $\Gamma_{fini} \models \varphi$.*

*Démonstration.*

$\Gamma \models \varphi$ ssi $\Gamma \cup \{\neg\varphi\}$ est contradictoire — par la Proposition 2.30
ssi il existe $\Gamma_f \subseteq \Gamma \cup \{\neg\varphi\}$ fini et contradictoire — par le Théorème de compacité
ssi il existe $\Gamma_f \subseteq \Gamma$ fini tel que $\Gamma_f \cup \{\neg\varphi\}$ est contradictoire
ssi il existe un sous-ensemble fini $\Gamma_f \subseteq \Gamma$ tel que $\Gamma_f \models \varphi$. □

### 2.3.3 Décidabilité du calcul propositionnel

Une logique est décidable s'il existe un algorithme (calcul réalisable sur un ordinateur qui termine toujours pour toute donnée) qui permet de savoir pour chaque formule si elle est une tautologie (i.e. si $\models \varphi$) ou pas.

**Théorème 2.44.** *Le calcul propositionnel est décidable.*

*Démonstration.* Méthode des tables de vérité : calculer la table de vérité prenant en argument les symboles propositionnels de $\varphi$ et calculer pour chaque valuation possible la valeur de $\varphi$.
Coût : $O(2^n)$ avec $n$ la taille de $\varphi$ (nombre de propositions). □

Nous verrons par la suite qu'il y a de meilleurs algorithmes, mais qu'il ont tous un coût exponentiel. La plupart de ces algorithmes débutent par une première phase de normalisation de la formule, c'est à dire qu'on modifie la syntaxe de la formule de manière à la mettre sous une forme normalisée, tout en conservant la sémantique de la formule, c'est-à-dire l'ensemble de ses modèles.

## 2.4 Equivalence entre formules

Il est courant de souhaiter modifier une formule, de façon à rendre son expression plus simple, ou plus facile à manipuler, et ceci en gardant bien sûr la sémantique de la formule, c'est-à-dire, sans modifier l'ensemble de ses modèles.

### 2.4.1 La substitution

La substitution d'une formule $\psi$ par une formule $\psi'$ dans une troisième formule $\varphi$ (notée $\varphi_{[\psi \leftarrow \psi']}$) consiste à remplacer chaque occurrence de $\psi$ dans $\varphi$ par $\psi'$.

Prenons par exemple $\varphi = (\neg p \vee q) \wedge (\neg p \vee \neg r)$, $\psi = \neg p$ et $\psi' = q \Rightarrow p$. Alors $\varphi_{[\psi \leftarrow \psi']} = ((q \Rightarrow p) \vee q) \wedge ((q \Rightarrow p) \vee \neg r)$.

Plus formellement, la substitution est définie de la façon suivante :

**Définition 2.45** (Substitution)**.** Soient $\varphi$, $\psi$, et $\psi'$ trois formules du calcul propositionnel,
- si $\psi$ n'est pas une sous-formule de $\varphi$, alors $\varphi_{[\psi \leftarrow \psi']} = \varphi$
- sinon si $\varphi = \psi$ alors $\varphi_{[\psi \leftarrow \psi']} = \psi'$
- sinon
  - si $\varphi = \neg\varphi'$ alors $\varphi_{[\psi \leftarrow \psi']} = \neg(\varphi'_{[\psi \leftarrow \psi']})$
  - si $\varphi = \varphi_1 \circ \varphi_2$ (où $\circ$ est un connecteurs $\wedge$, $\vee$, $\Rightarrow$) alors $\varphi_{[\psi \leftarrow \psi']} = \varphi_{1[\psi \leftarrow \psi']} \circ \varphi_{2[\psi \leftarrow \psi']}$.

**Proposition 2.46.** *Soient $\varphi$, $\psi$, et $\psi'$ trois formules du calcul propositionnel, si $\psi \equiv \psi'$ alors $\varphi \equiv \varphi_{[\psi \leftarrow \psi']}$.*

*Démonstration.* Voir TD 3. □

Attention, dans le cas général, la substitution ne conserve pas la sémantique de la formule. Par exemple, $p$ et $q \wedge \neg q$ ne sont pas équivalentes ; ainsi, les formules $p$ et $p[p \leftarrow q \wedge \neg q]$ ne sont pas équivalentes.

### 2.4.2 Equivalences classiques

Nous avons vu que remplacer une sous-formule $\psi$ d'une formule $\varphi$ par une formule équivalente $\psi'$ donne une formule notée $\varphi[\psi \leftarrow \psi']$ équivalente à $\varphi$. C'est-à-dire que cette substitution préserve les modèles, *i.e.*, $\mathsf{mod}(\varphi) = \mathsf{mod}(\varphi[\psi \leftarrow \psi'])$.

Voici quelques règles d'équivalences courantes, qui permettent de telles substitutions.

| | | |
|---|---|---|
| $\varphi \wedge \psi \equiv \psi \wedge \varphi$ | $\varphi \vee \psi \equiv \psi \vee \varphi$ | (Commutativité) |
| $\varphi \wedge (\psi_1 \wedge \psi_2) \equiv (\varphi \wedge \psi_1) \wedge \psi_2$ | $\varphi \vee (\psi_1 \vee \psi_2) \equiv (\varphi \vee \psi_1) \vee \psi_2$ | (Associativité) |
| $\top \wedge \varphi \equiv \varphi \wedge \top \equiv \varphi$ | $\bot \vee\varphi \equiv \varphi\vee \bot\equiv \varphi$ | (Élements neutres) |
| $\varphi \wedge \varphi \equiv \varphi$ | $\varphi \vee \varphi \equiv \varphi$ | (Idempotence) |
| $\varphi \wedge (\varphi \vee \psi) \equiv \varphi$ | $\varphi \vee (\varphi \wedge \psi) \equiv \varphi$ | (Absorption) |
| $\varphi\wedge \bot \equiv\bot \wedge\varphi \equiv\bot$ | $\varphi \vee \top \equiv \top \vee \varphi \equiv \top$ | (Elément absorbant) |
| $\varphi \wedge (\psi_1 \vee \psi_2) \equiv (\varphi \wedge \psi_1) \vee (\varphi \wedge \psi_2)$ | $\varphi \vee (\psi_1 \wedge \psi_2) \equiv (\varphi \vee \psi_1) \wedge (\varphi \vee \psi_2)$ | (Distributivité) |
| $\varphi \wedge \neg\varphi \equiv \neg\varphi \wedge \varphi \equiv\bot$ | $\varphi \vee \neg\varphi \equiv \neg\varphi \vee \varphi \equiv \top$ | (Complément) |
| $\neg\neg\varphi \equiv \varphi$ | | (Involution) |
| $\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$ | $\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$ | (Lois de De Morgan) |
| $\varphi \Rightarrow \psi \equiv \neg\varphi \vee \psi \equiv \neg(\varphi \wedge \neg\psi)$ | | (Implication matérielle) |
| $\varphi \Rightarrow \psi \equiv \neg\psi \Rightarrow \neg\varphi$ | | (Contraposition) |
| $\varphi_1 \Rightarrow (\varphi_2 \Rightarrow \varphi_3) \equiv (\varphi_1 \wedge \varphi_2) \Rightarrow \varphi_3$ | | (Curryfication) |

Nous observons ici que la formule suivante :

$$[(\varphi_1 \Rightarrow \varphi_2)\wedge(\varphi_2 \Rightarrow \varphi_3)] \Rightarrow [\varphi_1 \Rightarrow \varphi_3] \qquad \text{(Transitivité)}$$

est une tautologie, et que $\varphi \equiv \top$ si et seulement si $\varphi$ est une tautologie.

### 2.4.3 Formes normales

La mise sous forme normale transforme une formule en une formule équivalente (que l'on dit « normalisée ») plus adaptée au traitement algorithmique.

**Notations.** Remarquons que, à cause des lois d'associativité et commutativité, toutes les possibles formules construites à partir des formules $\varphi_1, \ldots, \varphi_n$ via l'application du connecteur logique $\wedge$, sont équivalentes. Par exemple

$$((\varphi_1 \wedge \varphi_2) \wedge \varphi_3) \wedge \varphi_4\,, \quad \varphi_1 \wedge ((\varphi_2 \wedge \varphi_3) \wedge \varphi_4)\,, \quad \varphi_1 \wedge (\varphi_2 \wedge (\varphi_3 \wedge \varphi_4))\,, \quad (\varphi_2 \wedge \varphi_4) \wedge (\varphi_3 \wedge \varphi_1)\,,$$

sont des formules équivalentes. Une remarque analogue vaut pour le connecteur logique $\vee$.

Ainsi, si $\varphi_1, \ldots, \varphi_n$ sont des formules, nous pourrons utiliser les notations :

$$\bigwedge_{i=1}^{n} \varphi_i = \varphi_1 \wedge \ldots \wedge \varphi_n \qquad \text{et} \qquad \bigvee_{i=1}^{n} \varphi_i = \varphi_1 \vee \ldots \vee \varphi_n$$

pour dénoter un parenthésage arbitraire de $\varphi_1 \wedge \ldots \wedge \varphi_n$ (resp. $\varphi_1 \vee \ldots \vee \varphi_n$). Les choix de l'ordre et du parenthésage ne sont donc pas significatifs, au moins du point de vue sémantique.

Nous pouvons même étendre nous considérations plus loin : si $\psi$ possède plus que deux occurrence dans la liste $\varphi_1, \ldots, \psi_n$, nous pouvons effacer les doublons de cette liste pour obtenir des formules équivalentes. Par exemple, les formules

$$\varphi_1 \wedge \varphi_2 \wedge \varphi_1 \wedge \varphi_3\,, \quad \varphi_1 \wedge \varphi_2 \wedge \varphi_3\,,$$

sont équivalents. Moralité : si ni l'ordre, ni le parenthésage, ni la multiplicité comptes, ce qui compte dans une telle formule est sa structure d'ensemble.

Nous allons donc considérer des conjonctions et disjonctions de liste de formules (avec ou sans répétions) ; $n$, la longuer de la liste pourra aussi prendre les valeurs 0 et 1, en posant :

$$\bigwedge_{i=1}^{0} \varphi_i := \top\,, \qquad \bigvee_{i=1}^{0} \varphi_i := \bot\,, \qquad \text{pour } n = 0,$$

$$\bigwedge_{i=1}^{1} \varphi_i := \varphi_1\,, \qquad \bigvee_{i=1}^{1} \varphi_i := \varphi_1\,, \qquad \text{pour } n = 1.$$

**Définition 2.47** (Littéral)**.** Un **littéral** est une formule atomique ou la négation d'une formule atomique. Autrement dit, c'est une formule $\ell$ de la forme $p$ ou $\neg p$, où $p$ est un symbole propositionnel.

**Définition 2.48** (Clause)**.** Une **clause disjonctive** est une disjonction de littéraux : $\bigvee_{i=1}^{n} \ell_i$ où les $\ell_i$ sont des littéraux. Une **clause conjonctive** est une conjonction de littéraux : $\bigwedge_{i=1}^{n} \ell_i$ où les $\ell_i$ sont des littéraux.

**Définition 2.49** (Forme normale conjonctive)**.** Une **formule conjonctive** (ou formule sous forme normale conjonctive (FNC), ou sous forme clausale) est une conjonction de clauses disjonctives : $\bigwedge_{j=1}^{m} C_j$ où les $C_j$ sont des clauses disjonctives, ou encore $\bigwedge_{j=1}^{m} \bigvee_{i=1}^{n} \ell_i^j$ les $\ell_i^j$ sont des littéraux.

**Exemple 2.50.** La formule suivante est sous forme clausale :

$$(\neg p \vee q \vee r) \wedge (\neg q \vee p) \wedge s$$

**Définition 2.51** (Forme normale disjonctive)**.** Une **formule disjonctive** (ou formule sous forme normale disjonctive (FND)) est une disjonction de clauses conjonctives : $\bigvee_{j=1}^{m} C_j$ où les $C_j$ sont des clauses, ou encore $\bigvee_{j=1}^{m} \bigwedge_{i=1}^{n} \ell_i^j$ les $\ell_i^j$ sont des littéraux.

**Exemple 2.52.** La formule suivante est sous forme normale disjonctive :

$$(\neg p \wedge q \wedge r) \vee (\neg q \wedge p) \vee s$$

La forme normale conjonctive est en général la plus adaptée lorsqu'on cherche un modèle d'une formule car il faut chercher une valuation satisfaisant chacune des clauses de la formule. Dans l'exemple 2.50, on voit que si $v$ est un modèle, alors forcément $v(s) = 1$ ; pour satisfaire la deuxième clause, il faut que $v(p) = 1$ ou $v(q) = 0$, mais alors la seule façon de satisfaire la première clause est $r = 1$. Il y a donc deux modèles.

**Mise sous forme normale conjonctive (ou forme clausale).** L'algorithme est le suivant :

**Etape 1 (élimination de l'implication).** Appliquer, tant que possible, la substitution suivante :

$$\varphi \Rightarrow \psi \leftarrow \neg\varphi \vee \psi \ .$$

**Etape 2 (pousser la négation vers les symboles propositionnels).** Appliquer tant que possible les substitutions suivantes (en remplaçant le membre gauche par le membre droit) :

$$\neg\neg\varphi \leftarrow \varphi \ , \qquad \neg(\varphi \wedge \psi) \leftarrow \neg\varphi \vee \neg\psi \ , \qquad \neg(\varphi \vee \psi) \leftarrow \neg\varphi \wedge \neg\psi \ .$$

**Etape 3 (pousser la disjonction vers les littéraux).** Appliquer tant que possible les substitutions suivantes (en remplaçant le membre gauche par le membre droit) :

$$\varphi \vee (\psi_1 \wedge \psi_2) \leftarrow (\varphi \vee \psi_1) \wedge (\varphi \vee \psi_2) \ .$$

**Exemple 2.53.** Considérons $\varphi = (\neg(p \wedge (\neg q \vee (r \vee s)))) \wedge (p \vee q)$.

— Etape 1 : rien à faire

— Etape 2 :

$\varphi = (\neg p \vee \neg(\neg q \vee (r \vee s))) \wedge (p \vee q)$
$\varphi = (\neg p \vee (\neg\neg q \wedge \neg(r \vee s))) \wedge (p \vee q)$
$\varphi = (\neg p \vee (q \wedge \neg(r \vee s))) \wedge (p \vee q)$
$\varphi = (\neg p \vee (q \wedge \neg r \wedge \neg s)) \wedge (p \vee q)$

— Etape 3 :

$\varphi = (\neg p \vee q) \wedge (\neg p \vee \neg r) \wedge (\neg p \vee \neg s) \wedge (p \vee q)$

**Proposition 2.54.** *Le calcul précédent termine et donne une formule en forme clausale équivalente à la formule initiale.*

**Remarque 2.55.** Il n'y a pas unicité de la forme clausale.

## 2.5 Le problème SAT

### 2.5.1 Définition du problème

**Définition 2.56** (Problème SAT)**.** Le problème SAT est le problème de décision qui consiste à déterminer si $\varphi \in \mathcal{F}_{\text{cp}}$ donnée en entrée admet, ou non, un modèle.

Le plus souvent, on suppose que la formule $\varphi$ en entrée est en forme normale conjonctive.

La plupart des algorithmes de résolution de SAT ne se contentent pas de répondre par oui ou par non, il peuvent fournir aussi un modèle, ou même l'ensemble des modèles.

**Exemple 2.57.** Donnée : Une formule booléenne mise sous forme FNC :

$$\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_1) \, .$$

Question : Est-ce que la formule $\varphi$ admet au moins un modèle ?

Réponse : Pour cet exemple, la réponse est oui : la valuation $v(x_1) = 0$, $v(x_2) = 1$, $v(x_3) = 1$ satisfait la formule $\varphi$, c'est-à-dire $v \in \mathsf{mod}(\varphi)$.

### 2.5.2 Un problème NP-complet

**Théorème 2.58.** *Le problème SAT est décidable.*

*Démonstration.* **Algorithme :** Etant donné une formule $\varphi$ ayant $n$ variables propositionnelles. Calculer les $2^n$ interprétations possibles. Pour chacune d'entre-elles, calculer la valeur de vérité de $\varphi$. Si au moins une est vraie, alors $\varphi$ est satisfaisable. □

Il existe des algorithmes plus performants, mais que ces améliorations ne changent pas fondamentalement la difficulté du problème. On est devant la situation suivante. Étant donnée une formule $\varphi$, on se demande si $\varphi$ admet un modèle ou non, i.e., s'il existe des valeurs de vérité attribuables aux variables propositionnelles qui satisferait $\varphi$ :

— une recherche exhaustive comme dans l'algorithme précédent peut demander jusqu'à $2^n$ vérifications si $\varphi$ possède $n$ variables propositionnelles. Cette démarche est dite **déterministe**, mais son temps de calcul est exponentiel.

— d'un autre côté, si $\varphi$ est satisfiable, il suffit d'une vérification à faire, à savoir tester précisément la configuration qui satisfait $\varphi$. Cette vérification demande un simple calcul booléen, qui se fait en temps polynomial (essentiellement linéaire en fait). Le temps de calcul cesse donc d'être exponentiel, à condition de savoir quelle configuration tester. Celle-ci pourrait par exemple être donnée par un être omniscient auquel on ne ferait pas totalement confiance. Une telle démarche est dite **non déterministe**.

La question de la satisfiabilité de $\varphi$, ainsi que tous les problèmes qui se résolvent suivant la méthode que nous venons d'esquisser, sont dits NP (pour polynomial non déterministe). Tester la satisfaisabilité de $\varphi$ équivaut par des calculs très simples (en temps polynomial) à tester la satisfaisabilité de sa négation.

Le problème SAT joue un rôle fondamental en théorie de la complexité, puisqu'on peut montrer que la découverte d'un algorithme déterministe en temps polynomial pour ce problème permettrait d'en déduire des algorithmes déterministes en temps polynomial pour tous les problèmes de type NP (théorème de Cook). On dit que SAT (et donc également le problème de la non-démontrabilité d'une proposition) est un problème NP-complet.

### 2.5.3 Modélisation - Réduction à SAT

Bien que le problème soit très difficile, nous verrons que nous disposons de logiciels très performants permettant de résoudre le problème de satisfaction d'une formule. Il est donc important pour tout informaticien de savoir profiter de ces outils. L'étape préalable est la suivante : étant donné un problème qui peut-être apparemment complètement dissocié de la logique, réduire la résolution de ce problème à la satisfaction d'une formule du calcul propositionnel. Il est bien sûr nécessaire que la réduction elle-même soit réalisable en un temps raisonnable (en temps polynomial).

#### 2.5.3.1 Comment modéliser

La plupart du temps, les problèmes sont énoncés en français (dans un fragment du français plus ou moins ambigu). La première étape est d'identifier les **propositions** d'un énoncé, il s'agit des plus petites briques de l'énoncé qui soient indécomposables et qui peuvent prendre la valeur vraie ou faux.

Il faut ensuite construire une formule traduisant les énoncés à partir des propositions, en utilisant les connecteurs booléens. On utilise pour cela la table de correspondance suivante

| | |
|---|---|
| et, mais | $\wedge$ |
| ou | $\vee$ |
| ne pas, non | $\neg$ |
| il n'est pas vrai que | $\neg$ |
| si $p$ alors $q$, $q$ seulement si $p$ | $p \Rightarrow q$ |
| $p$ si et seulement si $q$ | $p \Leftrightarrow q$ |

**Exemple 2.59** (suite)**.**

**Si et seulement si :** Considérons l'énoncé

« Le triangle est équilatéral si, et seulement si, il a trois coté égaux. »

et posons :
- *trois* : le triangle a trois côtés égaux
- *equi* : le triangle est équilatéral

L'enoncé se traduit par $trois \Leftrightarrow equi$. En effet, décomposons l'énoncé :
- « Le triangle est équilatéral si il a trois coté égaux » se traduit par $trois \Rightarrow equi$.
- « Le triangle est équilatéral seulement si il a trois coté égaux » signifie que si le triangle est équilatéral, alors il a forcément trois cotés égaux et se traduit donc par $equi \Rightarrow trois$.

On a donc $trois \Rightarrow equi \wedge equi \Rightarrow trois$, ce qui est équivalent à $equi \Leftrightarrow trois$.

**Implication versus équivalence** Il est d'usage, lors de la définition de concepts mathématiques, d'utiliser "si" à la place de "si et seulement si". Par exemple la définition du triangle équilatéral a plus souvent la forme suivante :

« Un triangle est équilatéral s'il a trois coté égaux. »

Pourtant il faut entendre un "si et seulement si". Il s'agit d'une convention malheureuse à laquelle vous devez vous habituer.

**Condition nécessaire et suffisante :** Cette expression est une autre version du "si et seulement si". Posons :
- *note* : avoir une bonne note
- *travail* : travailler

et considérons l'énoncé :

« Pour avoir une bonne note, il faut et il suffit de travailler »

ou de façon équivalente :

« Pour avoir une bonne note, il est nécessaire et suffisant de travailler »

Décomposons l'énoncé :

— $P_1$ = « Pour avoir une bonne note il faut travailler » signifie qu'il faut nécessairement travailler pour avoir une bonne note : on a donc la table suivante :

| *note* | *travail* | $P_1$ |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 1 |

Donc $P_1 \equiv note \Rightarrow travail$.

— $P_2$ = « Pour avoir une bonne note, il suffit de travailler » signifie que si on travaille, on a une bonne note :

| *note* | *travail* | $P_1$ |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |

Donc $P_2 \equiv travail \Rightarrow note$.

#### 2.5.3.2 Formalisation du problème du Sudoku pour la réduction à SAT

Une grille de Sudoku est composée de $n = \ell^2$ cases, et cette grille est elle-même divisée en $\ell$ sous-grilles. Généralement, on prend $n = 9$ et $\ell = 3$). Au départ certaines cases sont remplies par des chiffres et d'autres sont vides. Le but du jeu est de remplir les cases vides en respectant les règles suivantes :

— On ne doit pas avoir deux chiffres identiques sur une même ligne
— On ne doit pas avoir deux chiffres identiques sur une même colonne
— Il ne doit pas y avoir deux chiffres identiques dans l'une des sous-grilles de taille $\ell * \ell$.

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

Pour être un Sudoku, **la grille doit accepter une et une seule solution**.

**Formalisation.** Pour représenter le problème sous forme de problème SAT, nous avons besoin de beaucoup de variables propositionnelles. En effet il faut $n^3$ variables $V_{xyz}$ avec $x$, $y$ et $z$ *in* $[1..n]$. La variable $V_{xyz}$ sera vraie si et seulement si la case $(x, y)$ contient la valeur $z$. Par exemple, si $V_{135}$ est vraie, alors la case $[1, 3]$ contient 5. Si on prend $n = 4$, il faudra 64 variables, pour $n = 9$ il en faut 729. Voici la formulation des règles de remplissage d'un Sudoku.

« Chaque case contient un et un seul chiffre » : pour tout $i \in [1, n]$, pour tout $j \in [1, n]$, il existe $k \in [1, n]$ tel que la case $(i, j)$ contient $k$ et pour tout $k' \neq k$ dans $[1, k]$ : *la case* $(i, j)$ *ne contient pas* $k'$.

$$A = \bigwedge_{i\in[1,n]} \bigwedge_{j\in[1,n]} \bigvee_{k\in[1,n]} \left( V_{ijk} \wedge \bigwedge_{\substack{k' \in [1,k] \\ k' \neq k}} \neg V_{i,j,k'} \right)$$

« On ne doit pas avoir deux chiffres identiques sur une même colonne » : pour toute colonne $i \in [1, n]$, pour toutes lignes $j \neq j'$ dans $[1, n]$, pour toute valeur $k \in [1, n]$ : *si la case* $(i, j)$

*contient $k$, alors la case $(i, j')$ ne contient pas $k$.*

$$B = \bigwedge_{i\in[1,n]} \bigwedge_{j\in[1,n]} \bigwedge_{\substack{j' \in [1,n] \\ j' \neq j}} \bigwedge_{k\in[1,n]} (V_{i,j,k} \Rightarrow \neg V_{i,j',k})$$

« On ne doit pas avoir deux chiffres identiques sur une même ligne » : pour toute ligne $j \in [1, n]$, pour toutes colonnes $i \neq i'$ dans $[1, n]$, pour toute valeur $k \in [1, n]$ : *si la case $(i, j)$ contient $k$, alors la case $(i', j)$ ne contient pas $k$.*

$$C = \bigwedge_{j\in[1,n]} \bigwedge_{i\in[1,n]} \bigwedge_{\substack{i' \in [1,n] \\ i' \neq i}} \bigwedge_{k\in[1,n]} (V_{i,j,k} \Rightarrow \neg V_{i',j,k})$$

« On ne doit pas y avoir deux chiffres identiques dans l'une des sous-grilles de taille $\ell * \ell$ » : pour tous $x, x' \in [1, \ell]$,

$$D = \bigwedge_{x,y\in[0,\ell-1]} \bigwedge_{\substack{i,i' \in [1+\ell x, \ell+\ell x] \\ i \neq i'}} \bigwedge_{\substack{j,j' \in [1+\ell y, \ell+\ell y] \\ j \neq j'}} \bigwedge_{k\in[1,n]} (V_{i,j,k} \Rightarrow \neg V_{i',j',k})$$

Pour assurer qu'une grille donnée est un Sudoku, il faut indiquer les chiffres déjà inscrits et vérifier qu'il y a une et une seule solution au problème. Par exemple, pour la grille donnée en exemple, on crée la formule :

$$E = V_{1,9,5} \wedge V_{2,9,3} \wedge V_{5,9,7} \wedge \dots$$

On cherche alors les modèles de la formule $A \wedge B \wedge C \wedge D \wedge E$. Si il n'y a qu'un seul modèle, alors la grille est un Sudoku dont la solution est donnée par ce modèle.

### 2.5.4 Algorithmes de résolution de SAT

De nombreux algorithmes ont été proposés pour résoudre SAT, nous en présentons quelques-uns.

**Remarque 2.60.** Avant chercher un (ou plusieurs) modèle(s) d'une formule sous forme clausale, nous pouvons optimiser les calculs qui suivront en appliquant les règles suivantes :

1. Les clauses comportant deux littéraux opposés (par ex. $p \vee q \vee \neg r \vee \neg q$) sont valides (par le tiers-exclu) et peuvent donc être supprimées.
2. On peut supprimer les répétitions d'un littéral au sein d'une même clause (par ex. $\neg p \vee q \vee \neg r \vee \neg p$ équivaut à $\neg p \vee q \vee \neg r$).
3. Si, dans une formule clausale, une clause $C_i$ est incluse dans une clause $C_j$ (c'est-à-dire, tout littéral apparaissant dans $C_i$ apparaît aussi dans $C_j$), alors la clause $C_j$ peut être supprimée (la valeur de la conjonction des deux clauses ne dépend que de la valeur de $C_i$). Par exemple $C_i = p \vee q \vee r$ est incluse dans $C_j = p \vee \neg s \vee t \vee q \vee r$, de façon que l'on peut supprimer $C_i$ d'une formule clausale de la forme $C_1 \wedge \dots C_i \wedge \dots C_j \wedge \dots C_n$.

#### 2.5.4.1 Algorithmes de Quine et Davis-Putnam-Logemann-Loveland

La méthode de Quine s'applique à des formules mises au préalable sous forme normale conjonctive, assimilées à un ensemble de clauses.

**Exemple 2.61.** Si $\varphi = (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_4)$ (donc $\varphi$ est en FNC), alors son ensemble de clauses associé est

$$\mathcal{C}_\varphi = \{\neg x_1 \vee x_2; \neg x_2 \vee x_3; \neg x_3 \vee x_4; \neg x_1 \vee \neg x_4\}.$$

La méthode ne fait rien d'autre que parcourir l'arbre de toutes les solutions (l'arbre dont les branches complètes sont les valuations). Chaque fois qu'une variable est affectée à la valeur $x$ (avec $x \in \{0, 1\}$), le calcul se fait recursivement avec $\mathcal{C}[p \leftarrow f(x)]$, avec $f(0) = \bot$, et $f(1) = \top$.

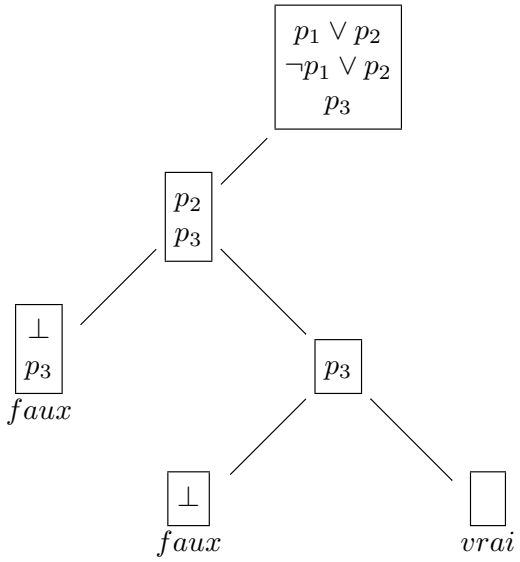
Comme la formule est sous forme clausale, le calcul de la subtitution $\mathcal{C}[p \leftarrow f(x)]$ avec $f(x) \in \{\bot, \top\}$ est donnée par une procédure purement syntaxique :

— pour substituer $p$ par $\top$ dans une formule clausale, il suffit de supprimer toutes les clauses contenant $p$ et de supprimer $\neg p$ de toutes les clauses contenant $\neg p$.
— de même, pour substituer $p$ par $\bot$ dans une formule clausale, il suffit de supprimer toutes les clauses contenant $\neg p$ et de supprimer $p$ de toutes les clauses contenant le littéral $p$.

On peut également appliquer les règles de simplification évidentes à $\mathcal{C}[p \leftarrow x]$, telles que la fusion ($\varphi \vee \psi \vee \psi \equiv \varphi \vee \psi$) et la subsumption (si on a deux clauses $C_1$ et $C_2$ telles que $C_2 = C_1 \vee C_1'$, alors $C_2$ peut-être supprimée).

| **Algorithme de Quine** |
|---|
| entrée : un ensemble de clauses $\mathcal{C}$<br>sortie : $vrai$ si $\mathcal{C}$ est satisfaisable ou $faux$ sinon |
| simplifier l'ensemble de clauses (cf. Remarque 2.60) ;<br>si $\mathcal{C} = \varnothing$ retourner $vrai$<br>si $\mathcal{C}$ contient la clause $\bot$ retourner $faux$<br>choisir le prochain $p \in$ Prop apparaissant dans une clause<br>si $Quine(\mathcal{C}[p \leftarrow \bot]) = vrai$ alors retourner vrai<br>sinon retourner $Quine(\mathcal{C}[p \leftarrow \top])$ |

**Exemple 2.62.** Considerons $\{ p_1 \vee p_2, \neg p_1 \vee p_2, p_3 \}$. L'algorithme explore l'arbre de Herbrand selon un parcours en profondeur gauche comme suit :




Observons dans l'exemple précédent que nous sommes contraintes par l'algorithme de Quine à essayer d'abord $p_1$, ensuite $p_2$, et puis $p_3$ ; de façon similaire, il en-force l'évaluation d'un symbole propositionnel d'abord à faux, et puis à vrai. Cette stratégie nous amène le plus souvent à des calculs inutiles : par exemple, dans l'exemple précédent, nous construisons des noeuds suite aux évaluations de $p_2$ et $p_3$ à faux, quand il est tout à fait évident que ces évaluations nous amènerons à un échec.

En général, nous ne sommes pas contraint à suivre un ordre fixé au debut. Nous allons donc chrecher d'améliorer l'algorithme de Quine pour trouver à la volée un ordre d'exploration des symboles propositionnels, qui soit optimisé pour l'ensemble des clauses passé en entrée.

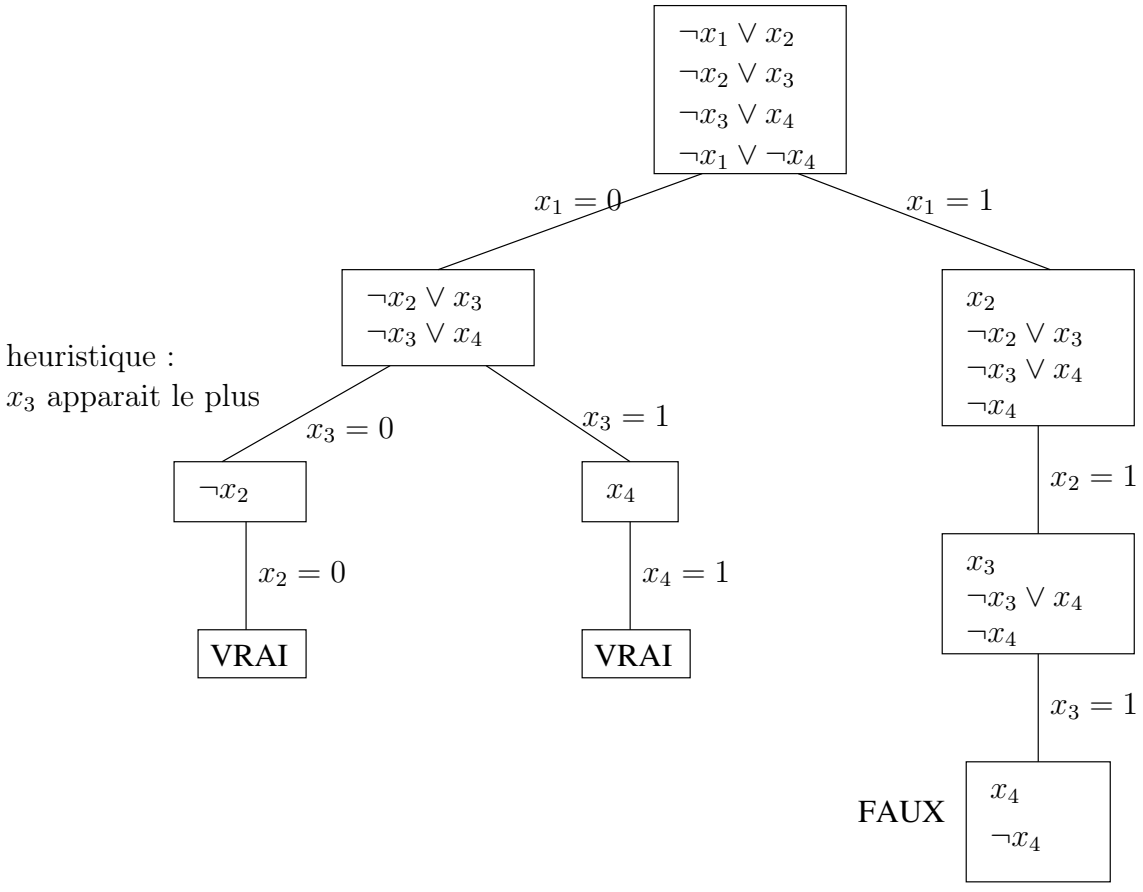
L'algorithme DPLL est considéré à ce jour comme l'une des méthodes les plus efficaces parmi celles permettant de résoudre le problème SAT. Il peut être vu comme un raffinement de la méthode de Quine ; l'amélioration principale qu'elle apporte est l'utilisation d'heuristiques pour accélérer le parcours des solutions.

| **Algorithme** $DPLL$ |
|---|
| entrée : un ensemble de clauses $\mathcal{C}$<br>sortie : $vrai$ si $\mathcal{C}$ est satisfaisable ou $faux$ sinon |
| simplifier l'ensemble de clauses (cf. Remarque 2.60) ;<br>si $\mathcal{C} = \varnothing$ retourner $vrai$<br>si $\mathcal{C}$ contient la clause $\bot$ retourner $faux$<br>si $\mathcal{C}$ contient la clause $p$ retourner $DPLL(\mathcal{C}[p \leftarrow \top])$<br>si $\mathcal{C}$ contient la clause $\neg p$ retourner $DPLL(\mathcal{C}[p \leftarrow \bot])$<br>choisir $p$ apparaissant dans une clause et un booléen $b$, **avec la bonne heuristique !**<br>si $DPLL(\mathcal{C}[p \leftarrow b]) = vrai$ alors retourner vrai<br>sinon retourner $DPLL(\mathcal{C}[p \leftarrow 1 - b])$ |

**Heuristiques.** Les heuristiques sont très importantes car elles permettent de réduire rapidement la taille de l'arbre de recherche. Parmi les heuristiques possibles :

- Choisir les variables qui apparaissent le plus dans les clauses les plus courtes
- Choisir les littéraux apparaissant dans les clauses les plus courtes.
- Choisir les littéraux qui n'apparaissent que positivement (resp. négativement), et parmi ceux-ci ceux qui sont présents le plus souvent.

**Exemple** Considérons l'ensemble de clauses $\mathcal{C} = \{\neg x_1 \vee x_2; \neg x_2 \vee x_3; \neg x_3 \vee x_4; \neg x_1 \vee \neg x_4\}$.




Les modèles de cette formule sont donc les suivants :