Cours : Logique

Table des matières

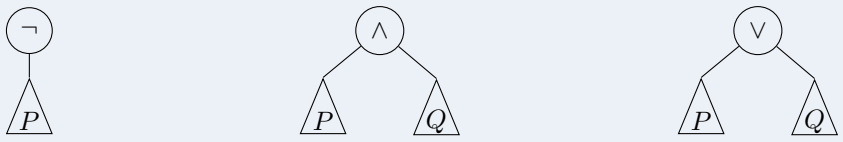
1 Syntaxe des formules propositionnelles

1.1 Formule propositionnelle

Définition 1.1

Soit $\mathcal{V}$ un ensemble infini dénombrable dont les éléments sont appelés *variables propositionnelles*.
Soient les symboles suivants :
- $\neg$ (*non* - négation) : connecteur unaire.
- $\wedge$ (*et* - conjonction) : connecteur binaire.
- $\vee$ (*ou* - disjonction) : connecteur binaire.
- $\top, \bot$ (*vrai*, *faux*) : constantes.

L'ensemble $\mathcal{P}$ des *formules propositionnelles* est défini de manière inductive par :
- $\top, \bot \in \mathcal{P}$.
- $\mathcal{V} \subset \mathcal{P}$.
- si $P, Q \in \mathcal{P}$ alors les arbres suivants sont dans $\mathcal{P}$:



Remarques

⇨ Les constantes $\top$ et $\bot$ ne feront pas systématiquement partie de la syntaxe.

⇨ Les *variables propositionnelles* (éléments de $\mathcal{V}$) seront le plus souvent notées $x, y \ldots$ ou x_i. Les *formules propositionnelles* seront notées $P, Q \ldots$.

Le type correspondant en OCaml :

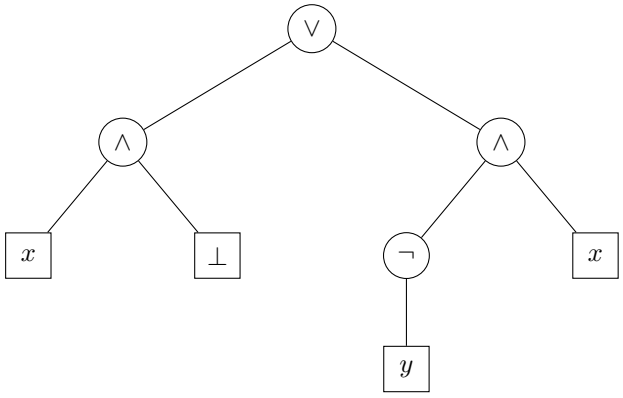
```
type prop =
  | Const of bool
  | Var of int
  | Non of prop
  | Et of prop * prop
  | Ou of prop * prop
```

Définition 1.2

Étant donnée une formule propositionnelle P :
- la *hauteur* de P est la hauteur de l'arbre correspondant (zéro si P est réduite à une variable ou une constante) ;
- la *taille* de P est le nombre de nœuds (internes ou non) de l'arbre ;
- une *sous-formule* de P est un sous-arbre, c'est-à-dire l'arbre enraciné en l'un des nœuds de P.

Exemple

⇨ On peut par exemple obtenir l'arbre



qui serait représenté en OCaml par `Ou(Et(Var 0, Const false), Et(Non(Var 1), Var 0))`.

1.2 Notation infixe

Même si une formule est fondamentalement un arbre, il est souvent plus pratique d'utiliser une représentation « à plat » (et nous avons en fait déjà commencé à le faire). Ces notations correspondent aux différents ordres associés au parcours en profondeur de l'arbre (préfixe, infixe, suffixe). Quelques remarques :
- si l'on utilise une notation traditionnelle (infixe), il faut des parenthèses. Pour en limiter le nombre, on se donne les règles de priorité suivantes :
  - $\neg$ est prioritaire sur $\wedge$ et $\vee$: $\neg A \wedge B$ signifie $(\neg A) \wedge B$;
  - **éventuellement**, $\wedge$ est prioritaire sur $\vee$ (et donc $A \wedge B \vee C$ signifie $(A \wedge B) \vee C$). En pratique, je vous conseille fortement de mettre les parenthèses pour éviter toute ambiguïté ;
- comme $\wedge$ et $\vee$ sont associatifs, on peut (par exemple) écrire $\bigwedge_{i=1}^{n} A_i$.

1.3 Égalité syntaxique

D'après ce qui précède, deux formules sont (syntaxiquement) *égales* si est seulement si les arbres correspondants sont égaux. Cependant, il peut y avoir quelques subtilités :
- les formules $A \wedge \neg A$ et $\bot$ ne sont clairement pas égales : les arbres sont totalement différents ;
- de même par exemple pour $\neg(A \wedge B)$ et $\neg A \vee \neg B$;
- en revanche, on peut vouloir considérer que $A \wedge B$ et $B \wedge A$ dénotent la même formule (c'est-à-dire faire rentrer la commutativité du « et » dans la syntaxe des formules). Il suffit pour cela de ne plus mettre d'ordre sur les deux fils du nœud $\wedge$ (il n'y a plus de fils gauche et de fils droit) ;
- il est également possible de rendre les connecteurs $\wedge$ et $\vee$ « syntaxiquement associatifs ». Dans ce cas, les nœuds étiquetés $\wedge$ et $\wedge$ ne seront plus binaires mais d'arité quelconque.

Remarque

⇨ Si l'on choisit comme notion d'égalité syntaxique la version « modulo commutativité et associativité », décider (informatiquement) l'égalité de deux formules n'est plus trivial !

1.4 Notation concise

On note parfois :
- $\overline{x}$ pour $\neg x$;
- $x \cdot y$ ou simplement xy pour $x \wedge y$;
- $x + y$ pour $x \vee y$.

Quand on utilise ces notations, on considère systématiquement que $\cdot$ est prioritaire sur $+$. Ainsi, $x\overline{y}z + y\overline{z}$ correspond à $(x \wedge \neg y \wedge z) \vee (y \wedge \neg z)$. On réserve le plus souvent cette notation au cas où $x, y \ldots$ sont des *variables*.

1.5 Substitution

Définition 1.3

Si P et Q sont des formules propositionnelles et si x est une variable, la *substitution de x par Q dans P* est la formule obtenue en remplaçant dans P toutes les occurrences de x par Q. On la note $P[Q/x]$. Formellement, on a :
- $\bot[Q/x] = \bot$;
- $\top[Q/x] = \top$;
- $x[Q/x] = Q$;
- $y[Q/x] = y$ si $y \neq x$;
- $(P_1 \wedge P_2)[Q/x] = P_1[Q/x] \wedge P_2[Q/x]$;
- $(P_1 \vee P_2)[Q/x] = P_1[Q/x] \vee P_2[Q/x]$;
- $(\neg P')[Q/x] = \neg(P'[Q/x])$.

On définit également la *substitution simultanée* de x_1 par Q_1, …, x_n par Q_n (où les x_i sont distinctes) de la même manière, en modifiant uniquement les troisième et quatrième points :
- $x_i[Q_1/x_1, \ldots, Q_n/x_n] = Q_i$
- $y[Q_1/x_1, \ldots, Q_n/x_n] = y$ si $y \notin \{x_1, \ldots, x_n\}$.

Remarques

⇨ Autrement dit, on remplace dans P (vue comme un arbre) toutes les feuilles étiquetée x par une copie de l'arbre de Q.

⇨ Attention, on n'a pas en général $P[x/Q][y/R] = P[x/Q, y/R]$. Dans quel cas est-ce vrai ?

Exercice 1

⇨ Écrire une fonction `substitue` telle que `substitue p x q` doit renvoyer $P[Q/x]$.

```
substitue : prop -> int -> prop -> prop
```

2 Sémantique des formules propositionnelles

2.1 Évaluation d'une proposition logique

Définition 2.1

Notons $\mathbb{B} = \{F, V\}$ (ou $\{0, 1\}$). On définit les fonctions Et, Ou, et Non par leur table de vérité :

x	y	Et(x, y)	Ou(x, y)	Non(x)
V	V	V	V	F
V	F	F	V	F
F	V	F	V	V
F	F	F	F	V

Définition 2.2: Valuation

Une *valuation* est une application de l'ensemble $\mathcal{V}$ des variables propositionnelles dans $\mathbb{B}$.

Remarques

⇨ On parle aussi de *distribution de vérité* ou d'*interprétation*.

⇨ Choisir une valuation, c'est donc choisir pour chacune des variables si elle est vraie ou fausse.

Définition 2.3: Évaluation d'une formule logique

Soit v une valuation. On définit l'*évaluation d'une formule P pour la valuation v par* :

- $\overline{v}(\top) = \top$
- $\overline{v}(\bot) = \bot$
- $\overline{v}(x) = v(x)$ si $x \in \mathcal{V}$
- $\overline{v}(\neg P) = \text{Non}(\overline{v}(P))$
- $\overline{v}(P \wedge Q) = \text{Et}(\overline{v}(P), \overline{v}(Q))$
- $\overline{v}(P \vee Q) = \text{Ou}(\overline{v}(P), \overline{v}(Q))$

Remarques

⇨ On parle aussi d'évaluation de P *dans le contexte* v.

⇨ On notera aussi $eval_v$ ou simplement e_v pour $\overline{v}$.

⇨ $\mathcal{V}$ est un ensemble infini, ce qui pose problème si l'on souhaite programmer cette évaluation. Cependant, pour évaluer une proposition P, il suffit de définir la valuation v sur l'ensemble $\mathcal{V}(P)$ des variables présentes dans P. Cet ensemble est bien évidemment fini.

⇨ Une manière souvent utile de voir les choses : si l'on prend $\mathbb{B} = \{0, 1\}$ (ou si l'on considère que $F < V$), alors $\wedge$ est un min et $\vee$ un max.

Exercice 2

⇨ Écrire une fonction `eval` prenant en entrée :
- un tableau de booléens `v` de taille n codant une valuation v (`v.(i)` correspond à $v(x_i)$) ;
- une formule propositionnelle `f` dont les variables sont supposées incluses dans $\{x_0, \ldots, x_{n-1}\}$

et renvoyant $\overline{v}(f)$.

```
eval : bool array -> prop -> bool
```

Définition 2.4

On dit qu'une valuation v *satisfait* une formule P si $\overline{v}(P) = V$.

Exercice 3

⇨
1. Dresser la table de vérité de la proposition $P = x \wedge (\neg y \vee z)$.
2. Comment s'interprète la ligne correspondant à $x = 0, y = 1, z = 1$ de cette table de vérité en termes d'évaluation de la formule P dans un contexte ?
3. Combien y a-t-il de tables de vérité différentes avec 2 variables ? et avec n variables ?

2.2 Conséquence logique, équivalence logique

Définition 2.5: Conséquence logique

Soient P et Q deux formules propositionnelles.
- On dit que Q est une *conséquence logique* de P, et l'on note $P \models Q$, si toute valuation satisfaisant P satisfait également Q.
- Si Γ est un ensemble de formules propositionnelles, on dit que $\Gamma \models Q$ si toute valuation satisfaisant l'ensemble des formules de Γ satisfait également Q.

Exemple

⇨ On a par exemple :

- $x \models x \vee y$
- $x, y \models x \wedge y$
- $x, y \models x$
- $x, \neg y \models (x \vee \neg z) \wedge (z \vee \neg y)$

Définition 2.6: Équivalence sémantique

Deux formules propositionnelles P et Q sont dites *sémantiquement équivalentes* (ou *logiquement équivalentes*, ou *tautologiquement équivalentes*) si $P \models Q$ et $Q \models P$.
On notera alors $P \equiv Q$.

Remarques

⇨ $P \equiv Q$ signifie donc que $\overline{v}(P) = \overline{v}(Q)$ pour toute valuation v.

⇨ Attention, deux formules sémantiquement équivalentes ne sont pas en général égales ! On a $\neg x \wedge \neg y \equiv \neg(x \vee y)$, mais les arbres correspondant ne sont pas du tout les mêmes.

Proposition 2.7: equivalence-substitution

Si $P \equiv Q$, alors pour toute variable x et toute formule R, on a $P[R/x] \equiv Q[R/x]$.

Remarque

⇨ Le résultat s'étend immédiatement à une substitution simultanée : si $P \equiv Q$, alors $P[x_1/R_1, \ldots, x_n/R_n] \equiv Q[x_1/R_1, \ldots x_n/R_n]$. La démonstration n'est que très légèrement modifiée.

Proposition 2.8

Si $P \equiv P'$ et $Q \equiv Q'$, alors $P \wedge Q \equiv P' \wedge Q'$, $P \vee Q \equiv P' \vee Q'$ et $\neg P \equiv \neg P'$.

2.3 Tautologies, satisfiabilité

Définition 2.9

On appelle *tautologie* une proposition P telle que $\overline{v}(P) = V$ pour toute valuation v. On pourra dans ce cas noter $\models P$.

Remarques

⇨ Cette notation est bien cohérente avec notre notation pour la conséquence logique, avec $\Gamma = \varnothing$.

⇨ P est une tautologie si et seulement si $P \equiv \top$.

⇨ On parle aussi de formule *universellement valide*.

Définition 2.10

Une proposition P est dite *satisfiable* s'il existe une valuation v telle que $\overline{v}(P) = V$. Une telle valuation v est alors appelée *témoin de satisfiabilité* de P.

Remarques

⇨ Une formule est satisfiable si et seulement si l'une au moins des lignes de sa table de vérité donne V, c'est une tautologie si et seulement si toutes les lignes de sa table de vérité donnent V.

⇨ On parle parfois d'*antilogie* pour une proposition non satisfiable (ou *insatisfiable*).

⇨ P est une antilogie si et seulement si $P \equiv \bot$.

Proposition 2.11: Règles usuelles de raisonnement

— $P \wedge P \equiv P$	**Idempotence de $\wedge$**
— $P \vee P \equiv P$	**Idempotence de $\vee$**
— $\neg\neg P \equiv P$	**Élimination de la double négation**
— $P \vee \neg P \equiv \top$	**Tiers exclu**
— $P \wedge \neg P \equiv \bot$	**Absurdité**
— $\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$	**De Morgan**
— $\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$	**De Morgan**
— $P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$	**Distributivité de $\wedge$ sur $\vee$**
— $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$	**Distributivité de $\vee$ sur $\wedge$**

Remarques

⇨ Aucune de ces règles ne correspond à une *égalité* (syntaxique) des formules !

⇨ On a bien d'autres règles valides, comme par exemple $P \wedge \top \equiv P$, ou bien $\bot \wedge P = \bot$: on pourra les utiliser sans justification (tant qu'elles sont valides et très simples).

Proposition 2.12

Si P et Q sont des propositions, on a $P \equiv Q$ ssi $\models ((P \wedge Q) \vee (\neg P \wedge \neg Q))$.

Remarque

⇨ Autrement dit, en anticipant un peu, $P \equiv Q$ ssi $\models P \leftrightarrow Q$.

2.4 Autres connecteurs

Un connecteur binaire est fondamentalement une application de $\mathbb{B}^2$, de cardinal 4, dans $\mathbb{B}$ qui est de cardinal 2. Il en existe donc $2^4 = 16$. Cependant, un certain nombre de ces connecteurs sont soit constants, soient de la forme $f \circ p_1$ ou $f \circ p_2$ où p_1 et p_2 représentent les projections (autrement dit, ils ne dépendent que d'un de leurs paramètres). Presque tous les autres (hors $\wedge$ et $\vee$) sont donnés ici :

P	Q	$P \leftarrow Q$	$P \to Q$	$P \leftrightarrow Q$	P NAND Q	P NOR Q	P XOR Q
1	1	1	1	1	0	0	0
1	0	1	0	0	1	0	1
0	1	0	1	0	1	0	1
0	0	1	1	1	1	1	0

Exercice 4

⇨ 1. Écrire tous ces connecteurs à partir de $\wedge$, $\vee$ et $\neg$.
2. Combien manque-t-il de connecteurs dans la table ci-dessus (en oubliant ceux qui sont constants ou qui ne dépendent que d'un paramètre) ? Les exprimer à l'aide de $\wedge$, $\vee$ et $\neg$.

2.5 Fonctions booléennes

Définition 2.13

On appelle *fonction booléenne* (d'arité n) une application de $\mathbb{B}^n$ dans $\mathbb{B}$.

Remarque

⇨ n peut éventuellement être nul, la fonction étant alors constante.

Définition 2.14

Soit P une proposition et $\mathcal{V}(P) = \{x_1, \dots, x_n\}$ l'ensemble de ses variables. La *fonction booléenne associée à* P est l'application

$$\begin{array}{rccc} \varphi_P : & \mathbb{B}^n & \longrightarrow & \mathbb{B} \\ & (v_1, \dots, v_n) & \longmapsto & e_d(P) \text{ où } d : x_i \to v_i \end{array}$$

Exercice 5

⇨ Déterminer la fonction booléenne associée à $x_1 \wedge \neg(\neg x_2 \vee (x_1 \wedge x_3))$.

Théorème 2.15: completude-fonctions-booleennes

Pour toute fonction booléenne f, il existe une proposition P telle que $f = \varphi_P$.

Remarques

⇨ Se référer au théorème **??** pour une autre démonstration, plus intuitive et plus parlante.

⇨ Comme $x \wedge \neg x \equiv \bot$ et $x \vee \neg x \equiv \top$, on peut en fait se passer des constantes.

Exercice 6

⇨ Trouver P telle que $\varphi_P = f$.

x	y	z	$f(x,y,z)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Définition 2.16

Un ensemble de connecteurs logiques est dit *système complet de connecteurs* si toute fonction booléenne est associée à une formule n'utilisant que ces connecteurs.

Remarque

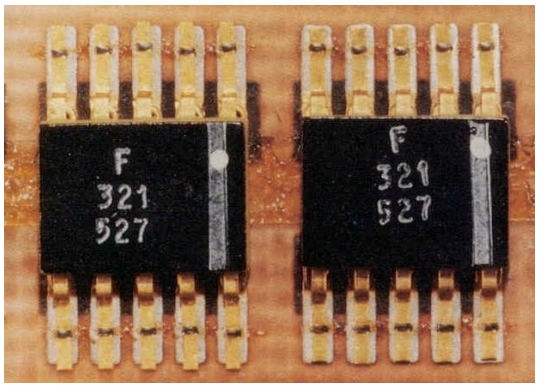
⇨ Dans le théorème **??**, les propositions sont (implicitement) construites avec $\neg, \wedge$ et $\vee$, qui constituent donc un système complet de connecteurs.

Proposition 2.17: complet

— $\{\neg, \wedge\}$ et $\{\neg, \vee\}$ sont des systèmes complets de connecteurs.
— $\{NAND\}$ et $\{NOR\}$ sont des systèmes complets de connecteurs.

Remarque

⇨ $\{NAND\}$ et $\{NOR\}$ sont les seuls systèmes complets constitués d'un seul connecteur. Certains circuits électroniques tirent parti de cette propriété en n'utilisant que des portes NAND (ou que des portes NOR). L'ordinateur de bord du module Apollo (AGC), par exemple, n'utilisait que deux types de circuits intégrés : des *sense amplifiers* pour amplifier le signal de la *core memory* et 2800 doubles portes NOR à trois entrées pour toute la partie logique et calcul.



Deux *dual three input NOR gates* (dimensions approximatives 70mm × 50mm).

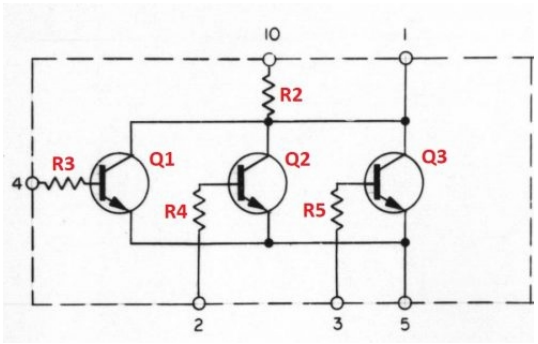


Schéma électronique équivalent. La patte 5 est V_0, 10 est V_+, 1 est le NOR de 2, 3 et 4. Le schéma est le même pour les pattes 6 à 9.

Exercice 7

⇨ Démontrer la propriété **??**.

3 Formes normales

Définition 3.1

On appelle :
— *littéral* une formule constituée uniquement d'une variable ou de la négation d'une variable.
— *min-terme* de k variables une conjonction de littéraux dans laquelle chacune des k variables apparaît une et une seule fois (soit telle quelle, soit niée).
— *max-terme* de k variables une disjonction de littéraux dans laquelle chacune des k variables apparaît une et une seule fois.
— *clause* un min-terme ou un max-terme.

Remarque

⇨ On peut former 2^k min-termes (ou max-termes) différents sur k variables : pour chaque variable, on choisit si elle apparaît positivement ou négativement.

Exemple

⇨ Pour deux variables, on peut former 4 min-termes et 4 max-termes :
—min-termes : $a \wedge b$, $a \wedge \neg b$, $\neg a \wedge b$, $\neg a \wedge \neg b$.
—max-termes : $a \vee b$, $a \vee \neg b$, $\neg a \vee b$, $\neg a \vee \neg b$.

Définition 3.2

On appelle *forme normale disjonctive* (FND), respectivement *forme normale conjonctive*, FNC, d'une proposition P toute disjonction de min-termes (respectivement conjonction de max-termes) logiquement équivalente à P.
Cette forme normale sera dite *canonique* si chacun des min-termes (respectivement des max-termes) contient les n variables de la formule initiale.

Remarques

⇨ Une formule est sous forme normale conjonctive si et seulement si c'est une conjonction de disjonctions de littéraux.

⇨ Une formule est sous forme normale disjonctive si et seulement si c'est une disjonction de conjonctions de littéraux.

⇨ Par convention, une disjonction vide est égale à $\bot$, donc $\bot$ est l'unique formule en FND constituée de zéro min-terme.

⇨ De même, une conjonction vide est égale à $\top$, donc $\top$ est l'unique formule en FNC constituée de zéro max-terme.

⇨ De manière générale, une formule en FND ayant n variables contient entre zéro et 2^n min-termes.

Exercices 8

⇨ Pour chacune des formules suivantes, dire si elle est en FND, en FNC et si elle est ou non canonique (en restreignant les variables à celles qui apparaissent dans la formule). On précisera aussi, le cas échéant, combien de clauses elle contient.

1. $P_1 = a \wedge b \wedge c$
2. $P_2 = (a \wedge b) \vee (b \wedge c)$
3. $P_3 = \bot$
4. $P_4 = \top$
5. $P_5 = \neg(a \wedge b)$
6. $P_6 = (a \wedge (b \vee c)) \vee (b \wedge c)$
7. $P_7 = (a \wedge \neg a) \vee (b \wedge c)$
8. $P_8 = (a \wedge b \wedge c) \vee (\neg b \wedge c)$

⇨ 1. On considère une formule en FND. Peut-on immédiatement déterminer s'il s'agit d'une tautologie ? si elle est satisfiable ?
2. Mêmes questions pour une formule en FNC.

Théorème 3.3

Toute formule propositionnelle admet une unique FND canonique et une unique FNC canonique à l'ordre (et l'associativité) près.

Remarque

⇨ Il y a deux manières d'obtenir une FND pour une formule donnée :
— passer par la table de vérité : on obtient naturellement une FND canonique ;
— utiliser les lois de De Morgan et la distributivité : on obtient une forme qui n'est pas *a priori* canonique.

Exercice 9

⇨ 1. Déterminer la FND et la FNC (canoniques) de $(a \wedge b) \vee (a \wedge \neg c)$.
2. De même pour $a \Longrightarrow b$.
3. Donner une FND et une FNC (non nécessairement canoniques) pour $x \wedge (y \vee z) \wedge (\neg x \vee y)$.

Une formule en FND ou FNC est très « simple » dans le sens où son arbre [1] est de hauteur au plus 3. Cependant, elle peut être très « grosse » dans le sens où la taille de son arbre peut être exponentielle en le nombre de variables. De plus, une formule peut avoir une FND concise (n min-termes) mais une FNC très longue (2^n max-termes), ou inversement.

Exercice 10

⇨ Mettre sous forme normale conjonctive la formule $(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee \cdots \vee (x_{2n-1} \wedge x_{2n})$.

4 Problème SAT

Définition 4.1

Le problème SAT consiste à déterminer, étant donnée une formule propositionnelle en forme normale conjonctive, si elle est satisfiable.
Le problème k-SAT est la restriction de SAT aux formules dont chaque clause contient au plus k littéraux.
Le problème MAX-SAT consiste à déterminer le nombre maximal de clauses que l'on peut simultanément satisfaire (dans une formule en FNC).

1. Dans la version n-aire gérant l'associativité.

Remarques

⇨ Pour SAT, on ne demande pas toujours que la formule soit en FNC : cela ne change rien, le problème restreint aux FNC est aussi difficile que le cas général.

⇨ SAT peut être vu comme une version « plus simple » de MAX-SAT : on ne demande pas combien de clauses on peut satisfaire simultanément, mais seulement si l'on peut toutes les satisfaire simultanément.

⇨ Si l'on restreint SAT aux formules en FND (au lieu de le restreindre aux FNC) le problème devient bien sûr trivial...

Nous reviendrons bien plus en détail sur ces problèmes l'année prochaine, mais on peut dès maintenant retenir quelques points :

— SAT est un problème absolument central en informatique, parce que d'innombrables problèmes se réduisent naturellement à lui ;
— SAT et k-SAT pour $k \geqslant 3$ sont des problèmes « difficiles » (NP-complets) pour lesquels on ne connaît pas d'algorithme en temps polynomial ;
— 1-SAT et 2-SAT, en revanche, peuvent être résolus en temps linéaire ;
— de même, un certain nombre de versions restreintes de SAT peuvent être résolues en temps polynomial.

Exercice 11

⇨ On considère le problème de la k-coloration d'un graphe : étant donné un graphe G, est-il possible de colorier ses sommets en utilisant au plus k couleurs ? Montrer qu'à partir d'un graphe G à n sommets, on peut calculer en temps polynomial en n une formule propositionnelle $P_k(G)$, de taille polynomiale en n, telle que $P_k(G)$ est satisfiable si et seulement si G est k-coloriable.

L'algorithme en « pure force brute » pour SAT consiste, pour une formule faisant intervenir n variables, à tester les 2^n valuations possibles : un tel algorithme n'est clairement pas utilisable pour n plus grand que cent. La conjecture P $\neq$ NP affirme plus ou moins que, dans le pire cas, on ne peut pas faire mieux. Cependant, un énorme effort à été fourni depuis des dizaines d'années pour écrire des *SAT solvers* capables de résoudre en temps raisonnable des instances « typiques » contenant des centaines de milliers de variables, voire davantage. Nous étudierons certains de ces algorithmes en TP.

5 Introduction au calcul des prédicats

Le *calcul des prédicats* (ou *logique du premier ordre*) remplace les variables propositionnelles de la logique propositionnelle par des formules atomiques, pouvant

5.1 Syntaxe

Définition 5.1: Signature

La *signature d'une théorie* est la donnée :
— d'un nombre fini de *symboles de fonctions*, doté chacun d'une arité (éventuellement nulle, on parle alors de *symbole de constante*) ;
— d'un nombre fini de *symboles de relation* (ou *symboles de prédicats*), doté chacun d'une arité strictement positive.

Sauf exception, on considérera que l'on dispose toujours d'un symbole de relation $=$ d'arité 2.

Exemple

⇨ *Signature de la théorie des groupes* Pour la théorie des groupes, on aurait la signature suivante :
—le symbole d'égalité ;
—un symbole de fonction $\star$ d'arité 2 (que l'on peut choisir de noter de manière infixe) ;
—un symbole de fonction d'arité 1, que l'on note $^{-1}$ de manière postfixe (*i.e.* x^{-1}) ;
—un symbole de constante (fonction d'arité 0) e.

Définition 5.2: Terme

On se dote d'un ensemble infini dénombrable $\mathcal{X}$ de variables [a]. Un *terme* (sur une certaine signature) est un arbre construit à partir de variables et de symboles de fonctions, en respectant leur arité.

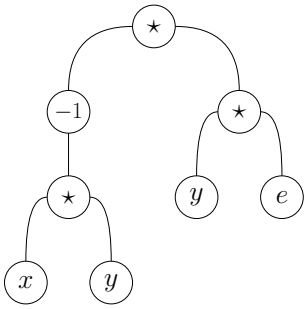
a. Qui ne sont pas des variables propositionnelles !

Remarque

⇨ Les feuilles de l'arbre seront donc étiquetées soit par des variables soit par des symboles de constantes, et les nœuds internes par des symboles de fonctions d'arité strictement positive.

Exemple

⇨ Avec la signature donnée pour la théorie des groupes, on peut par exemple construire les termes e, x (variable), $x \star y$, $(x \star y)^{-1} \star (y \star e) \dots$



Un terme en théorie des groupes.

Définition 5.3: Formule atomique

Une *formule atomique* (sur une certaine signature) est un arbre de la forme $R(t_1, \dots, t_k)$ où r est un symbole de relation d'arité k et $t_1, \dots, t_k$ sont des termes.

Exemple

⇨ Toujours avec la même signature, $x \star e = x$ et $x \star y = y \star x$ sont des formules atomiques.

Définition 5.4

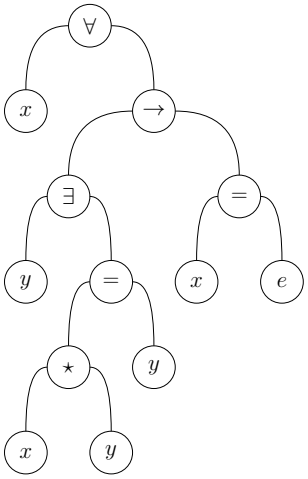
Une *formule* du calcul des prédicats est :
- soit une formule atomique ;
- soit de la forme $\exists x \varphi$ où x est une variable et φ une formule ;
- soit de la forme $\forall x \varphi$;
- soit de la forme $\neg\varphi$, $\varphi \wedge \varphi'$, $\varphi \vee \varphi'$ ou $\varphi \to \varphi'$, où φ et φ' sont des formules.

Remarque

⇨ Ces formules sont bien sûr des arbres, que l'on note partiellement en préfixe (quantificateurs, symboles de fonctions…) et partiellement en infixe (connecteurs logiques binaires, égalité…), voire partiellement en postfixe (symbole de fonction $^{-1}$ pour notre signature de théorie des groupes).

Exemple

⇨ L'arbre de formule ci-dessous sera noté $\forall x(\exists y\ x \star y = y) \to x = e$ avec les conventions usuelles de priorité. On peut rajouter des parenthèses si on le souhaite : $\forall x\big((\exists y\ x \star y = y) \to x = e\big)$.



Définition 5.5: Occurrence libre, occurrence liée

- Une *occurrence* d'une variable x dans une formule φ est un nœud de φ étiqueté par x.
- Une occurrence de x est dite *libre* si le chemin reliant l'occurrence à la racine ne contient pas de nœud $\forall x$ (ou plus précisément, pas de nœud $\forall$ dont le fils gauche est x), ni de nœud $\exists x$.
- Dans le cas contraire, l'occurrence est dite *liée*. Le point de liaison de l'occurrence est le premier nœud $\forall x$ ou $\exists x$ rencontré en remontant de l'occurrence vers la racine.

Remarque

⇨ La portée d'un quantificateur $\forall x$ (ou $\exists x$) est le sous-arbre enraciné en ce quantificateur. Toutes les occurrences de x dans ce sous-arbre sont liées, mais leur point de liaison n'est pas nécessairement la racine du sous-arbre.

Exercice 12

⇨ Représenter l'arbre de la formule $\forall x\big((\exists y\ \neg(x \star y = x)) \vee \forall x\ x = e\big)$ et donner le point de liaison de toutes les occurrences de variables.

,

Définition 5.6

Une variable est dite libre dans une formule φ si elle y a au moins une occurrence libre, liée si elle y a au moins une occurrence liée. On note $\mathcal{V}(\varphi)$ l'ensemble des variables qui apparaissent dans φ, $\mathcal{F}(\varphi)$ l'ensemble de ses variables libres, $\mathbb{B}(\varphi)$ l'ensemble de ses variables liées.

Remarque

⇨ Une variable peut être libre et liée dans une même formule : c'est par exemple le cas de x dans $x = e \wedge \forall x\ x \star x = e$. En règle générale, on évite d'écrire de telles formules.

Exercice 13

⇨ Donner des relations permettant de calculer $\mathcal{F}(\varphi)$ et $\mathbb{B})\varphi)$ pour une formule quelconque φ.

Définition 5.7: Substitution

Soit t un terme, φ une formule et x une variable. Si La *substitution* $\varphi[t/x]$ de x par t dans φ est la formule obtenue en remplaçant chaque occurrence libre de x dans φ par le terme t.

Exercice 14

⇨ On considère la formule $\varphi = \exists x\ \neg x = y$ et le terme $t = x$. Que vaut $\varphi[t/x]$? En quoi est-ce un problème ? Comment peut-on le régler ?

5.2 Notions de sémantique

Définition 5.8: Structure d'interprétation

Une *structure d'interprétation* S pour une signature L est la donnée :
- d'un ensemble non vide $\mathcal{U}$ appelé *univers* ;
- pour chaque symbole de constante c, d'un élément c^S de $\mathcal{U}$;
- pour chaque symbole de fonction f d'arité $a \geqslant 1$, d'une fonction $f^S : \mathcal{U}^a \to \mathcal{U}$;
- pour chaque symbole de relation R d'arité a, d'une relation a-aire sur $\mathcal{U}$, c'est-à-dire d'un ensemble $\mathbb{R}^S \subset \mathcal{U}^a$.

De plus, le symbole d'égalité est toujours interprété comme l'égalité entre éléments de $\mathcal{U}$.

Informellement, une telle structure permet de donner une signification aux formules closes, en interprétant les formules atomiques à partir de l'interprétation des symboles de fonctions et de relations et les quantificateurs comme portant sur l'univers $\mathcal{U}$. Il n'est pas très difficile de formaliser cette interprétation, mais cela demande de traiter les formules non closes : nous ne le ferons pas.

Définition 5.9: Modèle, conséquence logique

On suppose une signature L fixée.
- Une structure S est un *modèle* de la formule close φ si l'interprétation de φ dans S est un énoncé vrai. On note alors $S \models \varphi$.
- Si φ et φ' sont deux formules closes, on dit que φ' est une *conséquence logique* de φ si tout modèle de φ est également un modèle de φ'. On note alors $\varphi \models \varphi'$.

Exercice 15

⇨ On considère la signature $e, {}^{-1}, \star$ définie plus haut, et l'on définit les formules suivantes :
- $Ass : \forall x \forall y \forall z\ (x \star y) \star z = x \star (y \star z)$
- $Neu : \forall x\ x \star e = x \wedge e \star x = x$
- $Inv : \forall x\ x \star x^{-1} = x^{-1} \star x$
- $Gr : Ass \wedge Neu \wedge Inv$

1. Quelles sont les structures S vérifiant $S \models Gr$?
2. A-t-on $Gr \models \forall x\ (x^{-1})^{-1} = x$?
3. A-t-on $Gr \models \forall x \forall y\ x \star y = y \star x$?

4. A-t-on $Gr \models \neg(\forall x \forall y\ x \star y = y \star x)$?