# Projet Rocq

Nicolas Margulies nicolas.margulies@lmf.cnrs.fr
Théo Vignon theo.vignon@lmf.cnrs.fr

Le projet Rocq consiste à coder un SAT solveur vérifié en Rocq. Il s'agit donc d'écrire une fonction qui prend une formule en entrée et renvoie soit une affectation des variables qui rend cette formule valide, soit dit que la formule n'est pas satisfaisable. Ensuite, on prouvera que cette fonction est correcte. L'un des avantages de Rocq est que le langage de programmation et de preuve est le même, ce qui permet facilement de raisonner directement sur le programme.

Le projet est moins guidé que les tutoriels, il sera nécessaire d'énoncer et de prouver des lemmes intermédiaires avant de pouvoir prouver certains des résultats qui sont énoncés dans le fichier de départ. Il sera aussi utile de définir des fonctions auxiliaires afin de simplifier le code de vos algorithmes. Ceci est utile non seulement pour pouvoir prouver des résultats directement sur les fonctions auxiliaires, et car la difficulté pour prouver la correction d'une fonction est corrélée à son implémentation, et de mauvais choix pourront se révéler gênants plus tard.

## 1. Formules

D'abord, il nous faut une représentation des formules booléennes en Rocq, et une notion de satisfaisabilité.

**Question 1 — Formules booléennes**

Complétez l'inductif `form` et les notations en-dessous pour ajouter les connecteurs or, not, implication, true et false. Notez qu'ici les variables sont indexées par les entiers, et non des chaînes de caractères.

**Question 2 — Exemples de formules**

Complétez les définitions des `term*` avec les formules suivantes:

$$(x \vee \neg y) \wedge (\neg x \vee y) \qquad \neg y \Rightarrow (x \vee y) \qquad x \wedge \neg x \wedge \top \qquad x \Rightarrow \bot \qquad (x \vee y) \wedge (x \vee \neg y) \wedge \neg x$$

**Question 3 — Valuations et interprétation**

Le type des valuations est défini comme le type des fonctions des entiers (représentant les variables) dans les booléens. Définissez la valuation vide (toutes les variables sont assignées à $\bot$) et une fonction qui prend une valuation $v$, une variable $x$ et une valeur de vérité $b$ et renvoie une valuation $v'$ telle que $v'(y) = \begin{cases} b \text{ si } x{=}y \\ v(y) \text{ sinon} \end{cases}$.

Définissez la fonction `interp` qui renvoie la valeur booléenne d'une formule en sachant la valuation des variables.

**Question 4 — Satisfaisabilité et exemples**

Montrez que la satisfaisabilité (ou non) des formules données à la question 2.

## 2. Test de satisfaisabilité version 1

Une première façon de trouver si une formule est satisfaisable est de tester toutes les valuations possibles et de vérifier si l'une interprète la formule à $\top$. Pour garder un espace de recherche fini, on ne s'intéresse qu'à la valeur des variables apparaissant dans la formule concernée.

**Question 5 — `find_val`**

Remplissez la définition de `find_val f`, qui renvoie `Some val` tel que `interp val f = true` si possible, `None` sinon.

**Question 6 — Correction**

Prouvez la correction (soundness en anglais) de votre algorithme, c'est-à-dire que la formule est bien satisfaisable si votre algorithme renvoie `true`.

**Question 7 — Complétude**

Prouvez la complétude de votre algorithme, c'est-à-dire que si la formule est satisfaisable, alors votre algorithme renverra bien `true`.

Attention, comme bien souvent, la complétude est beaucoup plus dure à prouver que la correction, notamment car la valuation pour laquelle votre formule est évaluée à `true` donnée par le prédicat `sat` n'est pas nécessairement celle que renverra votre algorithme. Ceci est l'une des question les plus dures du projet.

Le solveur que l'on a codé est peu performant (il fonctionnera sur les exemples donnés ici, mais prendra très longtemps sur les exemples vus en partie 1). Pour améliorer ses performances, on commencera par exprimer les formules sous une forme plus facile à manipuler la forme normale conjonctive.

## 3. Vers la forme normale conjonctive

Une formule en forme normale conjonctive (abrégée CNF en anglais) se présente sous la forme : $\bigwedge_{i=0}^{n} \bigvee_{j=0}^{m} l_{i,j}$ où les $l_{i,j}$ sont chacun soit une variable, soit la négation d'une variable (on appelle ceci un littéral). Une première étape est de passer en forme normale négative, c'est-à-dire composée uniquement de littéraux et conjonction ou disjonction de formes normales négatives.

**Question 8 — Forme normale négative**

Définissez le type inductif `form_neg` des formes normales négatives, ainsi que ses notations et que l'interprétation desdites formules.

**Question 9 — Traduction vers la FNN**

Complétez la fonction `to_neg` qui traduit une formule vers une formule équivalente en forme normale négative. Prouvez ensuite que les formules sont effectivement équivalentes (i.e. qu'elles s'évaluent toujours de la même façon).

**Question 10 —— Forme normale conjonctive**

Sont définies dans le sujet les notions de littéraux, de clauses qui sont des listes de littéraux que l'on interprète comme leur disjonction et de formules en CNF qui sont des listes de clauses que l'in intermrète comme la conjonction de ces clauses.

Des lemmes auxiliaires concernant les littéraux sont énoncés dans le sujet, prouvez-les quand vous en verrez l'utilité.

Définissez l'interprétation d'une CNF et la traduction d'une FNN vers une CNF, prouvez qu'elle préserve l'interprétation.

Pour des raisons techniques, il est nécessaire de redéfinir la notion de satisfaisabilité pour les CNF, et de prouver que notre traduction la préserve bien.

## 4. DPLL

Maintenant que nos formules sont dans un format pratique d'utilisation, on peut s'intéresser à un autre algorithme pour décider la satisfaisabilité: l'algorithme DPLL (Davis-Putnam-Logemann-Loveland du nom de ses auteurs). L'idée principale de l'algorithme est qu'après avoir affecté une valeur à une variable, les clauses peuvent être modifiées en prenant en compte la valeur de la variable, et on peut même en déduire d'autres affectations de variables (par exemple, si on affecte $x$ à $\top$, et que l'une des clauses est $\neg x \vee \neg y$, alors $y$ doit nécessairement être affecté à $\bot$). Ainsi, l'algorithme alterne les phases de choix d'une valeur de variable $x$ avec la propagation de l'information donnée par ce choix, avant de terminer si le choix est correct, ou de revenir en arrière pour faire l'autre choix de valeur pour $x$ si les choix sont incohérents (i.e. l'une des clauses est vide).

**Question 11 —— implémentation de l'algorithme DPLL**

Pour implémenter l'algorithme DPLL, on va d'abord s'intéresser à la fonction `propagate`, qui étant donné un littéral `l` et une CNF `c`, simplifie `c` sous l'hypothèse que `l` est vrai. Ceci consiste à supprimer les clauses où `l` apparaît, et à retirer `~ l` des autres clauses.

Ensuite, on définit la notion de taille d'une CNF (la somme des tailles de ses clauses), et on montre que la propagation réduit nécessairement la taille de la CNF considérée.

Enfin, on peut utiliser `Function` pour définir la procédure de `dpll` en indiquant comme critère de terminaison que la taille de la CNF va décroître.

**Question 12 —— Correction**

Comme précédemment, prouvez que votre algorithme est correct.

**Question 13 —— Complétude**

Comme précédemment, prouvez que votre algorithme est complet.

De même que précédemment, la complétude est la partie difficile.

**Rendu 1 —— Vendredi 18 avril 23h59**

Envoyer vos fichers par email sous la forme d'une archive `nom_prénom.tar.gz` contenant l'intégralité de vos fichers ainsi qu'un readme expliquant les choix que vous avez effectués et les difficultés que vous avez rencontrées.