Éléments de Logique Formelle et du Raisonnement Mathématique

Algorithme de Davis-Putnam-Logemann-Loveland

Stéphane Devismes

Université de Picardie Jules Verne

23 octobre 2026



Au dernier cours

- Résolution

Jean, Pierre et Marie par résolution

Hypothèses :

- (H1) : Si Pierre est grand, alors Jean n'est pas le fils de Pierre
- (H2) : Si Pierre n'est pas grand, alors Jean est le fils de Pierre
- (H3) : Si Jean est le fils de Pierre alors Marie est la soeur de Jean

Conclusion (C) : Marie est la soeur de Jean ou Pierre est grand ou les deux à la fois.

Question : Peut-on déduire C de H1, H2 et H3 ? C'est-à-dire, H1, H2, H3 $\models$ C ?.

D'après la propriété 1.3, la réponse est OUI ssi la formule ci-dessous est une contradiction :

$$(p \Rightarrow \neg j) \wedge (\neg p \Rightarrow j) \wedge (j \Rightarrow m) \wedge \neg(m \vee p)$$

ou de manière équivalente :

$$\{p \Rightarrow \neg j, \neg p \Rightarrow j, j \Rightarrow m, \neg(m \vee p)\} \models \bot$$

- (H1) : $p \Rightarrow \neg j \equiv \neg p \vee \neg j$
- (H2) : $\neg p \Rightarrow j \equiv p \vee j$
- (H3) : $j \Rightarrow m \equiv \neg j \vee m$

Plan

1. Principes du DPLL
2. Simplifications
3. Code de l'algorithme
4. Correction de l'algorithme
   - Terminaison de l'algorithme
   - Correction partielle de l'algorithme
5. Heuristiques de branchement
6. Autres améliorations

But

Répondre à la question :

Une formule propositionnelle (en forme normale conjonctive) est-elle satisfaisable ?

Si oui, en donner une preuve.

Principe I

Deux types de transformations de formules :

1. **préservant le sens** : transformant une formule en une formule équivalente
   - réduction
2. **préservant seulement la satisfaisabilité** : transformant une formule satisfaisable en une formule satisfaisable.
   - suppression des clauses qui ont des littéraux isolés
   - résolution unitaire

`DPLL` est efficace car il utilise ces 2 transformations.

En bref

Algorithme de Davis, Putnam, Logemann et Loveland

- Inventé par Martin Davis et Hilary Putnam en 1960, puis amélioré par Martin Davis, George Logemann et Donald Loveland en 1962
- Permet de savoir **si un ensemble de clauses Γ** (équivalent à une fnc) **est satisfaisable**.
- Parcours « intelligent » des assignations possibles de Γ
- Base de SAT-solveurs complets comme **chaff, zchaff et satz**.
  Outils génériques pour résoudre de nombreux problèmes :
  - Planification
  - Emploi du temps
  - Placement de circuits
  - Problèmes de graphes
  - Vérification de contraintes
  - ...
- **Exponentiel** en temps dans le pire des cas.

Principe II

« Branchement/Retour-arrière »

- **Branchement** : Après simplification, affecter à **vrai** une variable choisit heuristiquement.
- Continuer récursivement l'algorithme.
- **Retour-arrière** : Si on arrive à une contradiction, on retourne au dernier choix, et on « branche » en affectant **faux** à la variable choisie.

Réduction d'un ensemble de clauses

Comment réduire ?

Enlever les clauses valides et les clauses contenant une autre clause de l'ensemble.

Rappel : Une clause C est incluse dans une clause C' si l'ensemble de littéraux de C est inclus dans celui de C'

Un ensemble de clauses est réduit s'il n'est plus réductible.

Ensemble de clauses réduit

Définition 4.1

Un ensemble de clauses est **réduit** s'il ne contient aucune clause valide et aucune clause n'est incluse dans une autre clause.

Exemple 4.1

La réduction de l'ensemble de clauses $\{p+q+\overline{p}, p+r, p+r+\overline{s}, r+q\}$ donne l'ensemble réduit :

Justification

Propriété 4.1

Un ensemble de clauses E est équivalent à l'ensemble de clauses réduit obtenu à partir de E.

Preuve.

Idée de la preuve :

- Enlever une clause valide donc valant 1 : $x.1 = x$
- Enlever une clause incluant une autre clause : $x.(x+y) = x$

□

Suppression des clauses qui ont des littéraux isolés.

Définition 4.2

Un littéral L est **isolé** si aucune clause de Γ ne comporte son complémentaire L^c.

Lemme 4.1

Supprimer des clauses avec un littéral isolé préserve la satisfaisabilité.

Preuve.

La preuve est demandée en exercice de TD. □

Exemple

Soit Γ l'ensemble de clauses

(1) $p+q+r$

(2) $\bar{q}+\bar{r}$

(3) $q+s$

(4) $\bar{s}+t$

Simplifiez Γ en supprimant des clauses qui ont des littéraux isolés.

Les littéraux p et t sont isolés.
Donc on obtient

(2) $\bar{q}+\bar{r}$

(3) $q+s$

Les littéraux $\bar{r}$ et s sont isolés.
On obtient l'ensemble vide.
D'après le lemme 4.1, Γ a un modèle.
Mais il existe des contre-modèles, par exemple : $p=0, q=0, r=0$! ! !

Résolution unitaire

Définition 4.3

Une **clause unitaire** est une clause qui ne comporte qu'un littéral.

Lemme 4.2

Soit L l'ensemble des littéraux des clauses unitaires de Γ. Soit Θ l'ensemble de clauses ainsi obtenu à partir de Γ

- si L comporte deux littéraux complémentaires, alors $\Theta = \{\bot\}$.
- sinon Θ est obtenue ainsi
  - enlever les clauses qui comportent un élément de L
  - à l'intérieur des clauses restantes enlever les littéraux complémentaires des éléments de L

Γ a un modèle si et seulement si Θ en a un.

Preuve.

La preuve est demandée dans en exercice de TD. □

Exemple

Simplifiez les ensembles de clauses suivants par résolution unitaire :

- **Soit Γ l'ensemble de clauses :** $p+q, \bar{p}, \bar{q}$
  En enlevant p et q de $p+q$ par la résolution unitaire, on obtient la clause vide $\bot$, donc Γ n'a pas de modèle.
- **Soit Γ l'ensemble de clauses :** $a+b+\bar{d}, \bar{a}+c+\bar{d}, \bar{b}, d, \bar{c}$**.**
  1. $a, \bar{a}$
  2. $\bot$

  donc Γ n'a pas de modèle.
- **Soit Γ' l'ensemble de clauses :** $p, q, p+r, \bar{p}+r, q+\bar{r}, \bar{q}+s$**.**
  Par résolution unitaire, on obtient : r, s.
  Cet ensemble de clauses a un modèle donc Γ' en a un.

Suppression de clauses valides

Lemme 4.3

Soit Θ l'ensemble de clauses obtenu en supprimant les clauses valides de Γ.

Γ a un modèle ssi Θ en a un.

Preuve.

- Supposons que Γ a un modèle v, comme Θ est un sous-ensemble des clauses de Γ, v est aussi modèle de Θ. Donc, Θ a un modèle.
- Supposons que Θ a un modèle v. Soit v' une assignation de Γ telle que $v'(x) = v(x)$ pour toute variable x présente à la fois dans Γ et Θ. Soit C une clause de Γ. Si C est aussi une clause de Θ, alors v' est un modèle de C car v et v' donnent la même valeur à C. Si C n'est pas une clause de Θ, alors C est valide, en conséquence toute assignation, v' en particulier, est modèle de C. D'où, Γ a un modèle : v'.

□

Rappel : $\Gamma[L := 1]$

Définition 3.8

Soient Γ un ensemble de clauses et L un littéral.

$\Gamma[L := 1]$ est l'ensemble de clauses obtenu en supprimant les clauses dont L est élément et en enlevant L^c des autres clauses.

Nous posons $\Gamma[L := 0] = \Gamma[L^c := 1]$.

Propriété 3.3

Soient Γ un ensemble de clauses et L un littéral. Γ a un modèle si et seulement si $\Gamma[L := 1]$ ou $\Gamma[L := 0]$ a un modèle.

Exemples

Exemple 3.8

Soit Γ l'ensemble de clauses $\overline{p}+q, \overline{q}+r, p+q, p+r$. Nous avons :

- $\Gamma[p := 1] = \{q, \overline{q}+r\}$.
- $\Gamma[p := 0] = \{\overline{q}+r, q, r\}$.

Observons que :

- $(\overline{1}+q)(\overline{q}+r)(1+q)(1+r) = q(\overline{q}+r) = \Gamma[p := 1]$.
- $(\overline{0}+q)(\overline{q}+r)(0+q)(0+r) = (\overline{q}+r)qr = \Gamma[p := 0]$.

L'algorithme DPLL

bool fonction Algo_DPLL(Γ : ensemble de clauses)
0 Supprimer les clauses valides de Γ.
Si $\Gamma = \emptyset$, retourner (vrai).
Sinon retourner (DPLL(Γ))

bool fonction DPLL(Γ : ensemble de clauses non valides)
La fonction retourne vrai si et seulement si Γ est satisfaisable
1 **Si** $\bot \in \Gamma$, retourner(faux).
Si $\Gamma = \emptyset$, retourner (vrai).
2 Réduire Γ : il suffit d'enlever toute clause contenant une *autre* clause.
3 Enlever de Γ les clauses comportant des littéraux isolés.
Si l'ensemble Γ a été modifié, aller en 1.
4 Appliquer à Γ la résolution unitaire.
Si l'ensemble Γ a été modifié, aller en 1.
5 Choisir x une variable quelconque de Γ
retourner (DPLL($\Gamma[x := 0]$) ou alors DPLL($\Gamma[x := 1]$))

Trace

Pour obtenir une trace de l'algorithme,

- nous supprimons les clauses valides, en abrégé **VAL**,

puis nous dessinons l'arbre des appels avec l'argument de la fonction ainsi que les ensembles obtenus par les étapes

- 2 (réduction, en abrégé **RE**),
- 3 (enlèvement des clauses ayant des littéraux isolés, en abrégé **ELI**) et
- 4 (résolution unitaire, abrégé en **RU**).

À cause du « ou alors », il est inutile de poursuivre la construction de cet arbre dès que la réponse vrai (attachée à un ensemble vide) a été obtenue.

Dans ce cas, un modèle peut être exhibé en remontant la récursion, en tenant compte des simplifications : le modèle doit rendre vrai chaque littéral isolé ou clause unitaire trouvé.

Exemple 1

Soit Γ l'ensemble de clauses : $\bar{a}+\bar{b}, a+b, \bar{a}+\bar{c}, a+c, \bar{b}+\bar{c}, b+c$.

Exemple 2

Soit Γ l'ensemble de clauses : $\overline{p}+\overline{q}, \overline{p}+s, p+q, \overline{p}+\overline{s}$.

Exemple 3

Soit Γ l'ensemble de clauses : $\bar{a}+b, \bar{b}+c, \bar{c}+\bar{d}, a+c+d, c+\bar{e}$

Rappel : correction d'un algorithme

Correction (totale) = Terminaison + Correction Partielle

1. Terminaison : Algo_DPLL(Γ) termine, pour tout ensemble fini de clauses Γ
2. Correction Partielle : Si Algo_DPLL(Γ) termine, alors sont résultat est correct, c'est-à-dire

   Algo_DPLL(Γ) retourne vrai si et seulement si Γ est satisfaisable

Roadmap

1. Principes du DPLL
2. Simplifications
3. Code de l'algorithme
4. Correction de l'algorithme
   - Terminaison de l'algorithme
   - Correction partielle de l'algorithme
5. Heuristiques de branchement
6. Autres améliorations

Terminaison

Théorème 4.1

L'algorithme Algo_DPLL termine.

Preuve.

- Le pas 0 n'est exécuté qu'une seule fois.
- Itération en 1 : le nombre de clauses diminue strictement.
- Récursivité en 5 : le nombre de variables diminue strictement.

D'où, terminaison.

□

Roadmap

1. Principes du DPLL
2. Simplifications
3. Code de l'algorithme
4. Correction de l'algorithme
   - Terminaison de l'algorithme
   - Correction partielle de l'algorithme
5. Heuristiques de branchement
6. Autres améliorations

Correction partielle

Propriété 3.3 (rappel) : Γ a un modèle ssi $\Gamma[x := 0]$ est satisfaisable ou $\Gamma[x := 1]$ est satisfaisable.

Théorème 4.2

L'algorithme Algo_DPLL est partiellement correct.

Preuve.

Invariant : la valeur courante de Γ a un modèle ssi Γ a un modèle.

Vérifié au pas 0, 1 et 5, en particulier réponses en 0 ou 1 correctes. Supposons les appels récursifs corrects :

- si DPLL($\Gamma[x := 0]$) est vrai, alors par récurrence $\Gamma[x := 0]$ est satisfaisable donc Γ est satisfaisable, d'après la propriété 3.3. ce qui correspond à la valeur vrai de DPLL(Γ).
- si DPLL($\Gamma[x := 0]$) est faux, alors par récurrence $\Gamma[x := 0]$ est insatisfaisable. Dans ce cas, DPLL(Γ) vaut DPLL($\Gamma[x := 1]$) :
  - Supposons que DPLL($\Gamma[x := 1]$) est vrai, alors par récurrence $\Gamma[x := 1]$ est satisfaisable donc Γ est satisfaisable, d'après la propriété 3.3, ce qui correspond à la valeur vrai de DPLL(Γ).
  - Supposons que DPLL($\Gamma[x := 1]$) est faux, alors par récurrence $\Gamma[x := 1]$ est insatisfaisable. D'après la propriété 3.3, Γ est insatisfaisable, ce qui correspond à la valeur faux de DPLL(Γ).

□

Conclusion

Corollaire 4.1

L'algorithme Algo_DPLL est correct.

Remarque 4.1

`DPLL` reste correcte si on oublie la réduction (2), enlèvement des littéraux isolés (3) et/ou réduction unitaire (4).

MOMS

Maximum Occurences in Clauses of Minimum Size

Sélectionner **la variable ayant le plus d'occurences dans les clauses les plus courtes**.

Les choix effectués par cette heuristique favorisent la propagation unitaire.

Principe

Le choix de la prochaine variable à affecter a un impact important sur la taille de l'arbre de recherche développé par `DPLL`.

Ainsi, le choix de l'heuristique de branchement est déterminant dans les algorithmes fondés sur `DPLL`.

Les heuristiques de branchement sont souvent basées sur des arguments syntaxiques tels que la taille des clauses, le nombre d'occurences, *etc*.

JW

Jeroslow-Wang

Utilise également la longueur des clauses.

Un poids est affecté à chaque littéral en fonction de la taille des clauses où il apparaît.

La possibilité qu'une variable soit sélectionnée est **inversement proportionnelle à la somme des tailles des clauses où elle apparaît**.

UP

Unit Propagation

Pas syntaxique

Essaie de sélectionner une variable en prévoyant à l'avance son effet sur la recherche.

Elle consiste à calculer un poids pour chaque variable **en fonction des simplifications qu'elle génère par propagation unitaire.**

Analyse des conflits et retour-arrière non chronologique

Lorsque l'algorithme `DPLL` atteint une contradiction (un conflit), il revient (chronologiquement) au dernier branchement de variable effectué.

Or, ce branchement n'est pas nécessairement l'une des décisions à l'origine de la contradiction.

L'analyse des conflits remédie à ce problème en tenant compte des décisions à l'origine du conflit.

Cette analyse permet ainsi d'effectuer un **retour-arrière non-chronologique**, c'est-à-dire un retour direct au niveau, plus haut dans l'arbre, qui est à l'origine du conflit sans tester les niveaux intermédiaires.

Ajout de clauses

Ajouter des clauses, en pré-traitement, à l'ensemble initial de clauses.

Le nouvel ensemble de clauses reste équivalent à l'ensemble initial.

Par exemple, la résolution restreinte ajoute des résolvants de taille bornée. [1]

But : réduire la taille de l'arbre recherche **en amenant aux contradictions plus tôt** dans la recherche.

1. Exemple de résolvant : $a + \bar{b}, b + \bar{c} \models a + \bar{c}$

Apprentissage

L'apprentissage consiste à analyser et apprendre les raisons qui ont amené à une contradiction : lorsqu'une assignation partielle amène à une contradiction, il est possible d'isoler un sous-ensemble d'assignations et un sous-ensemble de clauses, qui sont responsables du conflit.

À partir de ces assignations, il est possible de construire (d'apprendre) une **clause qui est impliquée** par ces clauses.

Cette nouvelle clause est ajoutée à l'ensemble de clauses du problème.

Les assignations pertinentes sont déterminées par un graphe de dépendance entre les clauses et les assignations, appelé graphe d'implication.

Les clauses apprises permettent de ne pas refaire plusieurs fois les mêmes « erreurs » dans l'arbre de recherche.

Redémarrage

La plupart des solveurs SAT utilisent une stratégie de redémarrage : redémarrer la recherche en prenant en compte de nouvelles clauses apprises durant la recherche précédente.

Un redémarrage est effectué après qu'un seuil de clauses apprises soit atteint.

À chaque redémarrage, le parcours de l'espace de recherche change car l'ensemble de clauses est modifié.

La plupart des solveurs effacent régulièrement les clauses apprises.

Ainsi, le mixage des redémarrages et des oublis peut dans certains cas mettre en cause la complétude des solveurs.

Un paramétrage des redémarrages est donc nécessaire pour garder cette dernière propriété.

Le solveur `Chaff`, par exemple, augmente à chaque redémarrage le seuil au-delà duquel une clause est oubliée. Ainsi, de plus en plus de clauses sont gardées, ce qui assure la complétude.

Structures de données paresseuses

Dans les solveurs SAT modernes, 80% du temps de calcul est consommé par la propagation unitaire.

Ainsi, des structures de données dites « paresseuses » ont été introduites dans le but d'optimiser la propagation unitaire.

Conclusion

C'est la fin !



Merci de votre attention.