AutoSat.

Nous proposons un algorithme pour SAT basé sur la théorie des automates finis.

1. L'idée.

Soit une formule F du calcul propositionnel en forme CNF sur des variables $x_1, x_2, \ldots$. Nous allons lui associer un automate A qui reconnait exactement le sous-langage de $\{Faux, Vrai\}^*$ des modèles de F. L'intuition est que la représentation en automate est assez compacte et permet de définir assez facilement une clause, la conjonction de formules et de faire des simplifications.

2. Les automates.

Nous allons représenter un automate à k états comme une liste de k couples $[e_0, e_1]$ où e_0 indique l'état suivant si la variable est à $Faux$, e_1 indique l'état suivant si la variable est à $Vrai$.

L'état 0 représente l'état rejetant et l'état 1 représente l'état initial.

Ainsi pour 3 variables, un automate qui accepte tous les modèles est $[[2,2],[3,3],[4,4],[4,4]]$. L'état 4 est un état acceptant puits qui accepte virtuellement toutes les valeurs des variables supérieures. Vous noterez que cet automate peut être simplifié en $[[1,1]]$. La clause $(x_1 \vee \neg x_2 \vee x_3)$ a comme automate $[[2,4],[4,3],[0,4],[4,4]]$. En effet, si $x_1 = Vrai$ alors on accepte en l'état 4, sinon, on va en 2 et si $x_2 = Faux$ alors on va en 4 sinon on va en 3 et si $x_3 = Vrai$ alors on va en 4 sinon on rejète en 0. L'état acceptant 4 accepte toutes les autres configurations. L'automate correspondant à la clause vide (représentant le $Faux$) est donc $[[0,0]]$.

Voici l'algorithme de construction de l'automate a correspondant à une clause c :

`a:=[ ]; si` $c = (\ )$ `alors` $i := 0$ `sinon` $i := 1$.

`Tant que` $c \neq (\ )$ `faire :`

  `Si` x_i `est la dernière variable de` c `alors` $j := 0$ `sinon` $j := i+1$.

  `Si` $x_i \in c$ `alors` $u := [j, Z]$ `sinon si` $\neg x_i \in c$ `alors` $u := [Z, j]$ `sinon` $u := [j, j]$

  `Ajouter` u `à la liste` a`, enlever l'éventuel` x_i `de` c `et faire` $i := i+1$.

`Ajouter` $[Z, Z]$ `à la liste` a `et remplacer dans` a `tous les` Z `par` i.

Ainsi l'automate correspondant à la clause $(\neg x_2 \vee x_5)$ est $[[2,2],[6,3],[4,4],[5,5],[0,6],[6,6]]$.

3. Conjonction des automates.

Nous allons construire à partir de deux automates a et b, l'automate c qui reconnait l'intersection des langages de a et de b. En termes de formules, c correspond donc à l'automate de la conjonction des formules de a et de b.

$c := [\ ]$; $ab := [[1,1]]$; $h := 1$; $k := 1$; `Tant que` $h \leq k$ `faire :`

  $\alpha := ab_h[1]$; $\beta := ab_h[2]$; $u := [0,0]$. `Pour` p `dans` $[1,2]$ `faire :`

    $i := a_\alpha[p]$; $j := b_\beta[p]$; `si` $i \neq 0$ `et` $j \neq 0$ `alors :`

      `si` $[i,j] \notin ab$ `alors ajouter` $[i,j]$ `à la liste` ab `et faire` $k := k+1$.

      `pour` q `la position de` $[i,j]$ `dans` ab `faire` $u[p] := q$.

  `ajouter` u `à la liste` c `et faire` $h := h+1$

L'automate $a = [[2,0],[2,2]]$ de la clause $(\neg x_1)$ et

l'automate $b = [[2,3],[0,3],[3,3]]$ de la clause $(x_1 \vee x_2)$ donnent

l'automate $c = [[2,0],[0,3],[3,3]]$ qui correspond bien à la formule $\neg x_1 \wedge x_2$.

De même, le produit des automates des clauses (x_5) et $(\neg x_5)$ donne $[[2,2],[3,3],[4,4],[5,5],[0,0]]$.

4. Simplification des automates.

Nous avons vu que certains automates pouvaient être simplifiés.

Voici la méthode pour simplifier un automate a.

Procédure $simp(i,k)$
  remplacer dans a tous les i par k
  supprimer a_i de la liste a
  remplacer dans a les i+1 par i, i+2 par i+1, i+3 par i+2...

Procédure principale
  répéter
    si $a_1 = [0,0]$ alors faire $a := [[0,0]]$
    si $a_i = [0,0]$ pour $i > 1$ alors $simp(i,0)$
    si $a_i = a_j$ pour $i > j$ alors $simp(i,j)$
    si $a_i = [i,i]$ et $a_j = [j,j]$ pour $i > j$ alors $simp(i,j)$
  jusqu'à ce que a soit stable

La simplification de l'automate $[[2,2],[3,3],[4,4],[5,5],[0,0]]$ donne $[[0,0]]$.

La simplification de l'automate $[[2,2],[3,3],[4,4],[5,5],[5,5]]$ donne $[[1,1]]$.

5. Algorithmes pour SAT.

Voici deux algorithmes de décision pour une formule F formée d'une conjonction de clauses.

1. Méthode séquentielle :
Commencer avec l'automate $a := [[1,1]]$.
Pour toutes les clauses c de F faire :
  mettre dans a la conjonction de a et de l'automate de la clause c
  simplifier a
retourner a

2. Procédure récursive $auto(F)$:
Si F a une seule clause alors retourner l'automate de cette clause
Si F a plus d'une clause alors
  décomposer F en deux parties non vides F_1 et F_2
  calculer $a := auto(F_1)$ et $b := auto(F_2)$.
  calculer c l'automate de la conjonction de a et b
  simplifier c et retourner c

Quelque soit la méthode, si elle donne $[[0,0]]$ alors F est insatisfaisable sinon F est satisfaisable.