Démonstration automatique

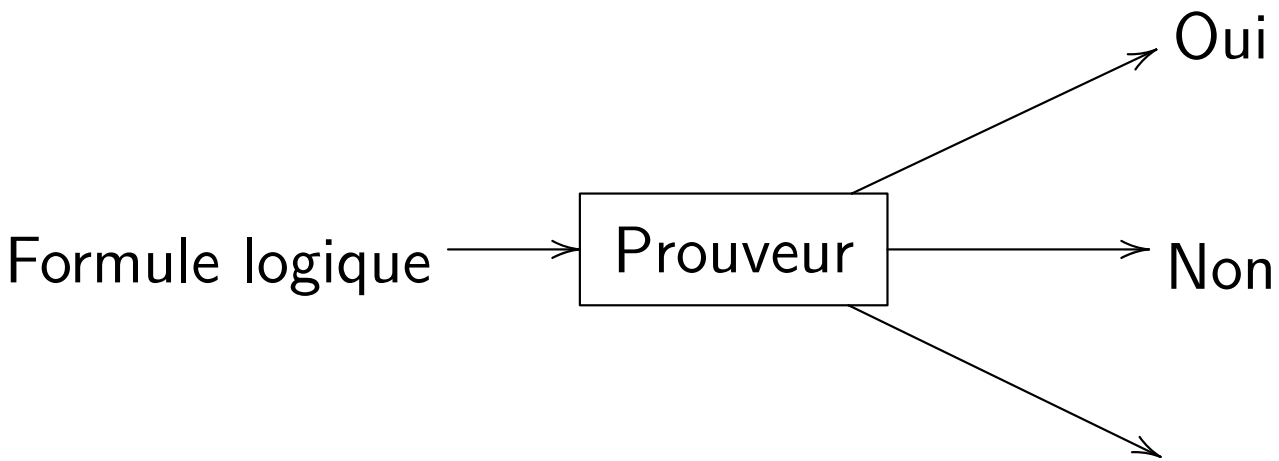
Option Programmation raisonnée 1

ENSIIE

Semestre 5 — 2016–17

Introduction générale

Preuve automatique



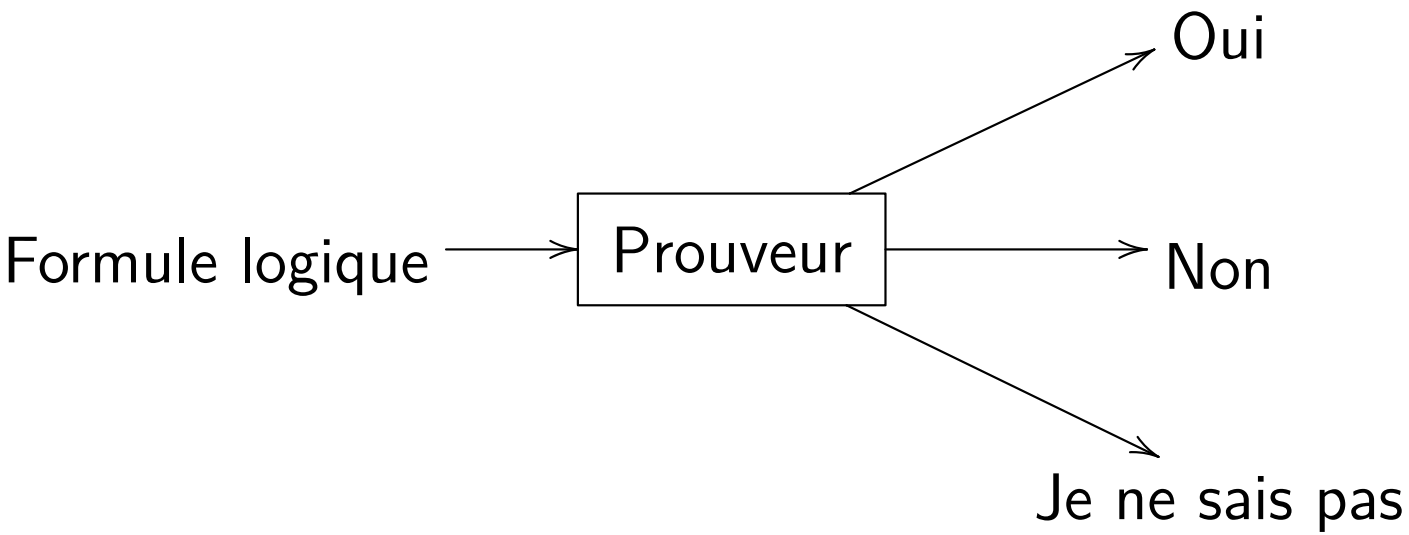
Mieux :

- Preuve
- Modèle (contre exemple)

Encore mieux :

- Preuve dans format vérifiable (e.g. Coq)

Preuve automatique



Mieux :

- Preuve
- Modèle (contre exemple)

Encore mieux :

- Preuve dans format vérifiable (e.g. Coq)

Mais...

Le problème :

- Entrée : une formule F de la logique du premier ordre
- Décide si F est valide ou non

n'est pas décidable (seulement récursivement énumérable)

Pire : Gödel [1931] :

Théorème

La question de savoir si une formule arithmétique est valide dans le modèle standard est indécidable (même pas récursivement énumérable).

Validité, satisfiabilité

F valide :

- F est vraie dans tous les modèles

F satisfiable :

- F est vraie dans un modèle

F valide ssi $\neg F$ est insatisfiable

En logique classique

F	$\neg F$
valide	insatisfiable
preuve	incohérence
non valide	satisfiable
contre-exemple	modèle

Preuve par réfutation

Pour montrer F

- on part de $\neg F$
- on essaie d'en déduire une contradiction

Remarque :

- Si $F = (H_1 \wedge \cdots \wedge H_n) \Rightarrow G$ alors
  $\neg F = H_1 \wedge \cdots \wedge H_n \wedge \neg G$

Logiques

Différents niveaux de logiques
$\Rightarrow$ différentes catégories de prouveurs

- Propositionelle
- Premier ordre (prédicats)
- Ordre supérieur
  - HOL
  - CIC
  - ...

+ Théories
  - Égalité
  - Arithmétiques
  - Tableaux
  - Bitvectors
  - ...

Correction et complétude

Méthode de preuve $\vdash$ validité logique $\models$

$\vdash$ correcte pour $\models$:

- pour toute formule F, si $\vdash F$ alors $\models F$

$\vdash$ complète pour $\models$:

- pour toute formule F, si $\models F$ alors $\vdash F$

Exemples

Prouveur toujours correct :

- Répond toujours “Je ne sais pas”

mais incomplet

Prouveur toujours complet :

- Répond toujours “Oui”

mais en général incorrect (sauf si logique incohérente. . .)

Méthode correcte et complète

Si $\models F$ alors $\vdash F$ (“Oui”)

Si $\not\models F$ alors

- “Non”
- “Je ne sais pas”
- ne termine pas

Clauses

Pour l'instant, logique propositionnelle
Variables propositionnelles $a, b, c, \ldots$
Littéral : variable ou négation de variable a, $\neg a$
Clause : ensemble de littéraux

- Peut être vue comme disjonction de littéraux

Mise en forme normale clausale

Forme normale clausale :

- Conjonction de disjonctions de littéraux
- correspond à un ensemble de clauses

$$\Gamma \wedge (C \vee (A \wedge B)) \rightarrow \Gamma \wedge (C \vee A) \wedge (C \vee B)$$
$$\Gamma \wedge (C \vee (A \Rightarrow B)) \rightarrow \Gamma \wedge (C \vee \neg A \vee B)$$
$$\Gamma \wedge (C \vee \neg(A \wedge B)) \rightarrow \Gamma \wedge (C \vee \neg A \vee \neg B)$$
$$\Gamma \wedge (C \vee \neg(A \vee B)) \rightarrow \Gamma \wedge (C \vee \neg A) \wedge (C \vee \neg B)$$
$$\Gamma \wedge (C \vee \neg(A \Rightarrow B)) \rightarrow \Gamma \wedge (C \vee A) \wedge (C \vee \neg B)$$
$$\Gamma \wedge (C \vee \neg\neg A) \rightarrow \Gamma \wedge (C \vee A)$$

Exemple

$$\neg \left(\begin{array}{l} \quad (z \Rightarrow ((x \Rightarrow a) \wedge b)) \\ \wedge (z \vee (a \wedge b)) \\ \wedge (y \vee a \vee b) \\ \wedge (y \Rightarrow x) \\ \Rightarrow (a \wedge b) \end{array} \right)$$

Attention

Traduction naïve :

- taille potentiellement exponentielle

On peut faire mieux en introduisant nouvelles variables

Mais :

- Conversion vers CNF : NP difficile

Résolution

$$\text{Resolution}\ \frac{\{a\} \cup C \qquad \{\neg a\} \vee D}{C \cup D}$$

On applique cette règle tant que l'on peut

- Si $\emptyset = \bot$ est produite, ensemble insatisfiable
- Sinon ensemble satisifiable

Résolution unitaire

En particulier :

$$\text{Resolution}\ \frac{\{a\} \qquad \{\neg a\} \vee D}{D}$$

On n'a plus besoin de $\{\neg a\} \vee D$ ensuite (redondante)

SAT

Satisfiabilité en logique propositionnelle

SAT

- un ensemble de clauses Γ
- décide si Γ est satisfiable ou non

Idéalement :

- Si satisfiable, retourne un modèle $=$ valuation 0 ou 1 des variables propositionnelles
- Sinon, retourne une preuve (par résolution par exemple)

Difficulté théorique

Problème NP-complet

- si algo polynomial, alors P=NP

Même si les clauses sont toutes de taille au plus trois

Mais...

Utilisations multiples

Depuis les années 90 (GRASP et Chaff), efficace en pratique
$\Rightarrow$ applications industrielles

Big data

Solution (négative) au problème des triplets pythagoriciens :

- Séparer $\mathbb{N}$ en deux ensembles disjoints N_1 et N_2
- tel que ni N_1 ni N_2 ne contient un triplet pythagoricien (a, b, c) tel que $a^2 + b^2 = c^2$

preuve utilisant un SAT solver : 200 TB ! ! !

Algorithme glouton

On construit le modèle petit à petit, en revenant en arrière si on s'est trompé.

Tant que toutes les variables ne sont pas assignées :

- Décider une valuation pour une variable
- Si conflit (une clause non satisfiable avec l'assignement courant)
  backtrack (revient sur la dernière décision)

Si toutes les variables sont assignées $\rightsquigarrow$ SAT
Si on a essayé en vain les deux possibilités à la racine $\rightsquigarrow$ UNSAT

DPLL [1962]

Davis, Putnam, Logemann, Loveland

Propagation unitaire :

- si tous les littéraux d'une clause sauf un sont assignés à 0, alors on doit nécessairement assigner ce dernier à 1

Tant que toutes les variables ne sont pas assignées :

- Décider une valuation pour une variable
- Faire toutes les propagations unitaires possibles
- Si conflit (une clause non satisfiable avec l'assignement courant)
  backtrack (revient sur la dernière décision)

CDCL [1996]

Clause-Driven Conflict Learning
Dans DPLL, on retombe sur les mêmes conflits dans des branches différentes

- Ajouter une clause qui permet d'éviter d'arriver au même conflit
  - explication du conflit
  - déductible par résolution depuis les clauses initiales
- Après ajout de la clause de conflit, on peut revenir à la dernière décision qui ne rend pas la clause fausse
  - on remonte plus haut (backjumping)
  - reste complet grâce à clause apprise

Graphe d'implication

Nœuds : assignement d'une variable à un niveau
Arêtes : propagations unitaires, étiquetées par la clause responsable

Si conflit dans graphe d'implication, on peut remonter pour obtenir la clause d'apprentissage (= les raisons du conflit)

On peut backjumper au niveau de l'assignement de niveau le plus grand qui ne soit pas le niveau courant

Utilisation des solveurs SAT

- Encodage du problème CNF
  - Représente le problème comme une instance de SAT
- Utilisation du SAT solveur comme oracle
  - boîte noire
- Intégration du SAT solveur
  - boîte blanche
  - utilisation des assignements partiels
  - dialogue

Glucose

Gilles Audemard (Lens) et Laurent Simon (Bordeaux)

Basé sur Minisat

Libre

Satisfiabilité modulo théories

Au-delà de SAT

Besoin de raisonner à un niveau d'abstraction supérieur

Preuve de programme :

- arithmétique
- structures de données
- ...

$\Rightarrow$ logique du premier ordre

Rappel : logique du premier ordre

Terme :

- constante
- variable
- symbole de fonction appliqué à des termes

Proposition atomique :

- symbole de prédicat appliqué à des termes

$\wedge, \vee, \Rightarrow, \neg$

$\forall x.\ F$, $\exists x.\ F$

Satisfiabilité modulo théorie

Problème : logique du premier ordre indécidable

On se restreint à la satisfiabilité dans une théorie donnée de formules sans quantificateurs

$$T \models F \qquad\qquad \vdash_{\equiv T} F$$

Décidable ou non en fonction de la théorie

Exemple

Théorie : arithmétique

$$
\begin{aligned}
& x \leq 3 \\
\wedge & (2x > 8 \vee x + y = 3) \\
\wedge & (y < -2 \vee x + y \neq 3)
\end{aligned}
$$

$$x_1 \wedge (\neg x_2 \vee x_3) \wedge (\neg x_4 \vee \neg x_3)$$

Coopération

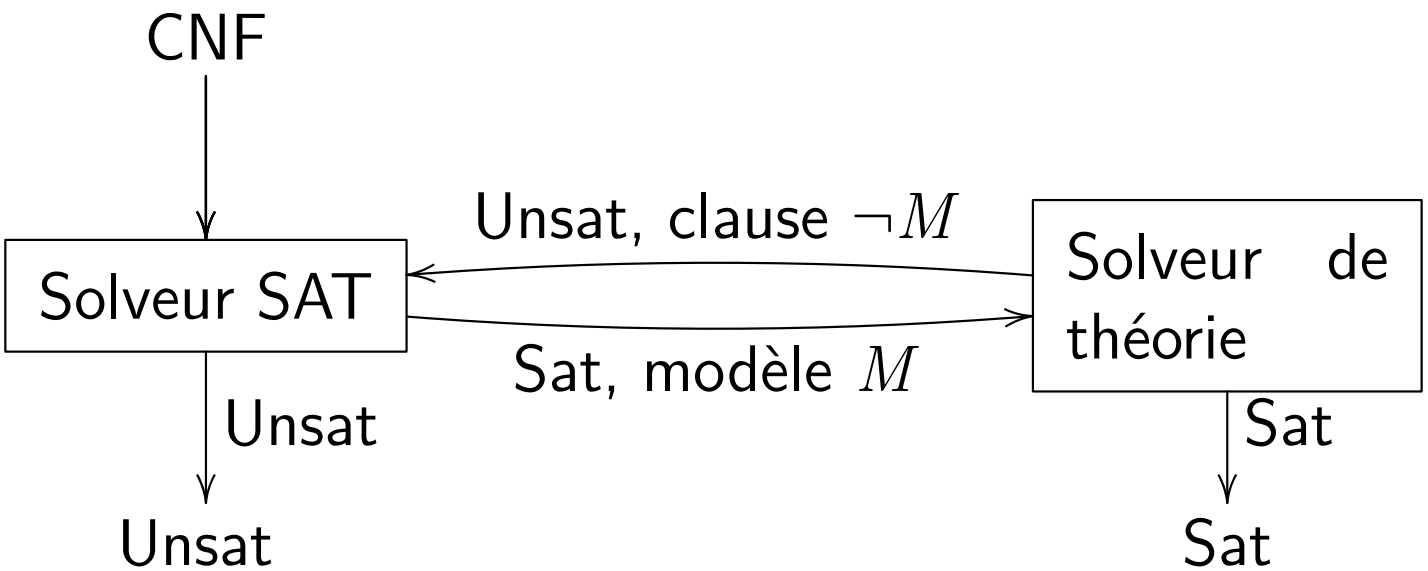
Solveur SAT

- gère la partie « logique » de la preuve

Solveur de théorie

- Entrée : conjonction de littéraux satisfaits (= modèle partiel)
- Sortie : satisfiable ou non pour la théorie

DPLL(T) offline



DPLL(T) online

Fait une requête au solveur de théorie dès qu'un nouvel assignement est effectué

Le solveur de théorie renvoie une clause de conflit (plus petite que $\neg M$)

Quelles théories ?

Théorie = ensemble des formules qui restreignent les modèles à considérer

DPLL(T) marche pour n'importe quelle théorie

En pratique, certaines théories ont un intérêt, notamment pour la vérification de programme

Égalité avec fonctions non interprétées QF_UF

Prédicat : $=$

Symboles de fonction quelconques

Axiomes : ceux de l'égalité
(réflexivité, symétrie, transitivité, congruence)
$x = y \Rightarrow f(\dots, x, \dots) = f(\dots, y, \dots)$

Exemple :

$$g(a) = c \wedge (f(g(a)) \neq f(c) \vee g(a) = d) \wedge (c \neq d \vee g(a) \neq d)$$

Arithmétique entière linéaire (QF_)LIA

Prédicats : $=$, $\leq$

Symboles de fonction : 0, 1, $+$, -

Axiomes : formules vraie dans $\mathbb{Z}$

Arithmétique de Presburger, décidable

Exemple :

$$x \leq 3 \wedge (2x > 8 \vee x + y = 3) \wedge (y < -2 \vee x + y \neq 3)$$

Arithmétique entière non linéaire (QF_)NIA

Prédicats : $=$, $\leq$

Symboles de fonction : 0, 1, $+$, -, $\times$

Axiomes : formules vraie dans $\mathbb{Z}$

Arithmétique de Peano, indécidable, même la satisfiabilité des formules sans quantificateurs

Exemple :

$$x \leq 3 \wedge (x^2 > 30 \vee xy = 3) \wedge (y < -2 \vee xy + x \neq 3)$$

Arithmétique réelle (QF_)NRA

Prédicats : $=$, $\leq$

Symboles de fonction : 0, 1, $+$, -, $\times$

Axiomes : formules vraie dans $\mathbb{R}$

Décidable (simplex)

Exemple :

$$x \leq 3 \wedge (x^2 > 30 \vee xy = 3) \wedge (y < -2 \vee xy + x \neq 3)$$

Tableaux QF_AX

Prédicats : $=$

Symboles de fonction : select, store

Axiomes :

$$\forall a\ i\ v.\ \mathsf{select}(\mathsf{store}(a, i, v), i) = v$$
$$\forall a\ i\ v\ j.\ i \neq j \Rightarrow \mathsf{select}(\mathsf{store}(a, i, v), i) = \mathsf{select}(a, j)$$
$$\forall a\ b.\ (\forall i.\ \mathsf{select}(a, i) = \mathsf{select}(b, i)) \Rightarrow a = b$$

Exemple :

$$\mathsf{select}(\mathsf{store}(\mathsf{store}(a, i, v), j, w), i) \neq w \quad \vee \quad i \neq j \quad \vee \quad v \neq w$$

Bitvectors QF_BV

Variables = vecteur de bits de taille fixe
Prédicats : $=$, $\leq$

Symboles de fonction : 0 : [1], 1 : [1],
$\_ \circ \_$: [n]$\times$[m]$\rightarrow$[n+m]
$+$ opération bits à bits (et, ou, plus, shift left, etc.)

Décidable

Utile pour vérification matériel et logiciel

Combinaison de théories

En pratique, pas une seule théories, mais une combinaison de plusieurs

Exemple :

$$1 \leq x \wedge x \leq 2 \wedge f(x+3) \neq f(5) \wedge f(x+4) \neq f(5)$$

Comment combiner des solveurs de deux théories ?

Nelson-Oppen

Conditions :

- Signatures disjointes (à part $=$)
- Théories stablement infinies :
  - si satisfiable, alors satisfiable dans modèle de cardinalité infinie
- Conjonctions de littéraux sans quantificateurs

Purification

Si un terme t_1 de la théorie T_1 apparaît sous un symbole de fonction/prédicat f de la théorie T_2

- nouvelle variable z
- on met z sous f
- on ajoute $z = t_1$

En itérant ce processus, on peut séparer la formule en deux formules, chacune étant uniquement dans une théorie

$$F \qquad \rightsquigarrow \qquad F_1 \wedge F_2$$

Combinaison

Théorie T_1 : SAT_1 Théorie T_2 : SAT_2

On peut lancer SAT_1 sur F_1 et SAT_2 sur F_2

- Si l'un des deux répond Unsat $\rightsquigarrow$ Unsat
- Mais sinon, on doit regarder variables partagées

Pour toutes les possibilités de rendre les variables égales ou non e.g. $e = (x_1 = x_2 \wedge x_2 \neq x_3)$

- On teste chaque SAT_i sur $F_i \wedge e$
- Si les deux répondent Sat $\rightsquigarrow$ Sat

Si on a tout testé $\rightsquigarrow$ Unsat