

Démonstration automatique

Option Programmation raisonnée 1

ENSIIE

Semestre 5 — 2022–23

Organisation de l'UE

SELP Sémantique des langages de programmation

- ▶ cet après-midi

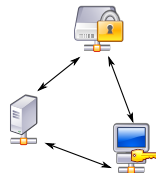
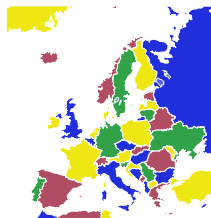
PRFM Preuve formelle mécanisée

Évaluation pour ce module :

- ▶ 1 projet (Coq) $\left(\frac{1}{2}\right)$
- ▶ examen final $\left(\frac{1}{2}\right)$

Introduction générale

Pourquoi des preuves mécanisée ?



- ▶ Mathématiques
- ▶ Vérification de programme
- ▶ Cryptographie
- ▶ ...

Quel type de preuve mécanisée ?

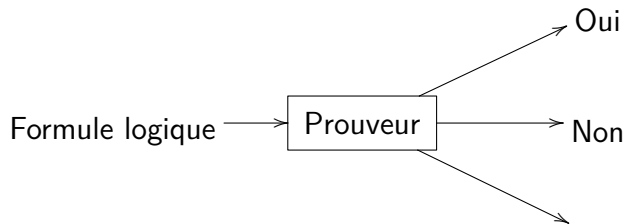
Assistants à la preuve :

- ▶ construction de la preuve guidée par l'utilisateur
- ▶ automatisation limitée
- ▶ Isabelle, Lean, HOL Light, Coq

Prouveurs automatiques :

- ▶ presse-bouton
- ▶ limites théoriques et pratiques
- ▶ plusieurs familles selon rapport expressivité/performance

Preuve automatique



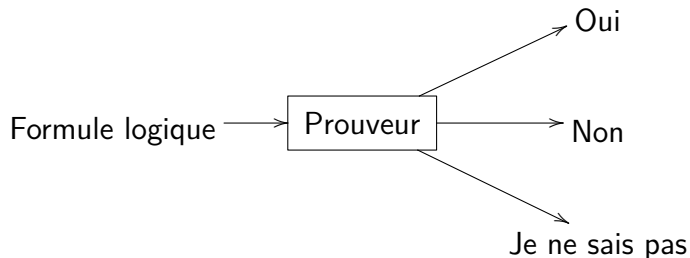
Mieux :

- ▶ 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)

Définition d'un prouveur automatique

“A program that you can stuff a mathematical specification [...], and which will then run forever, using up all of your machines resources.

Very occasionally it will find a proof for the conjecture and tell you so.”

(README du prouveur E)

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
preuve

insatisfiable
incohérence

non valide
contre-exemple

satisfiable
modèle

Preuve par réfutation

Pour montrer F

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

Remarque :

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

Logiques

Différents niveaux de logiques

⇒ différentes catégories de prouveurs

- ▶ Propositionnelle
- ▶ 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, \dots

Littéral : variable ou négation de variable $a, \neg a$

Clause : ensemble de littéraux

- ▶ Peut être vue comme disjonction de littéraux

Mise en forme normale clauseale

Forme normale clauseale :

- ▶ 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

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

Attention

Traduction naïve :

- ▶ taille potentiellement exponentielle

On peut faire mieux en introduisant nouvelles variables

Mais :

- ▶ Conversion vers CNF : NP difficile

Résolution

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

On applique cette règle tant que l'on peut

- ▶ Si $\emptyset = \perp$ est produite, ensemble insatisfiable
- ▶ Sinon ensemble satisfiable

Résolution unitaire

En particulier :

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

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
 ⇒ 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 pythagorien (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 : assignment 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'assignment 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
- ▶ ...

⇒ 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

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

Décidable ou non en fonction de la théorie

Exemple

Théorie : arithmétique

$$x \leq 3$$

$$\wedge(2x > 8 \vee x + y = 3)$$

$$\wedge(y < -2 \vee x + y \neq 3)$$

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

Coopération

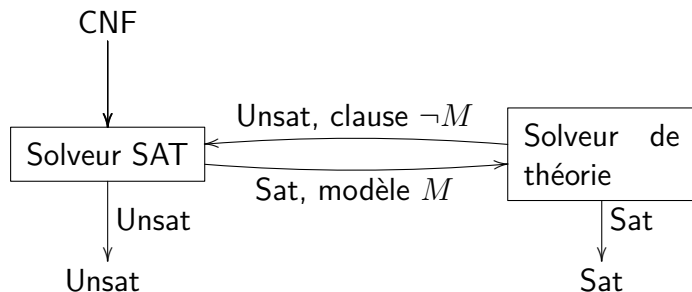
Solveur SAT

- ▶ 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 :

$$\forall a \ i \ v. \text{select}(\text{store}(a, i, v), i) = v$$

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

$$\forall a \ b. (\forall i. \text{select}(a, i) = \text{select}(b, i)) \Rightarrow a = b$$

Exemple :

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

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

Exemple :

$$(x = 4 * 2 \vee 1 < 0 * y) \wedge y * x = \text{sll}(y, 3)$$

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

$$F \rightsquigarrow F_1 \wedge F_2$$

Combinaison

Théorie T_1 : SAT₁

Théorie T_2 : SAT₂

On peut lancer SAT₁ sur F_1 et SAT₂ sur F_2

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

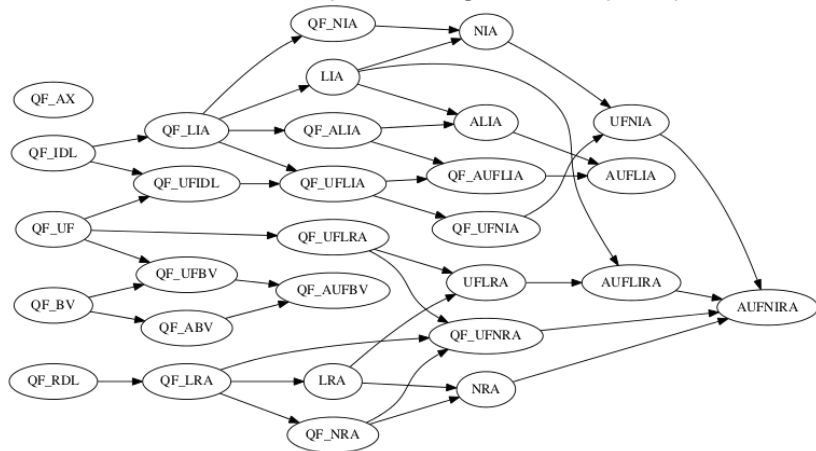
Pour toutes les possibilités de rendre les variables égales ou non e.g. $e = (x_1 = x_2 \wedge x_2 \neq x_3)$

- ▶ On teste chaque SAT _{i} sur $F_i \wedge e$
- ▶ Si les deux répondent Sat \rightsquigarrow Sat

Si on a tout testé \rightsquigarrow Unsat

Dans SMT-LIB v2

Toutes les combinaisons pas envisageables en pratique



Exemple : AUFNIRA

A : tableaux (mais restreints à Int ou $\text{Int}^2 \mapsto \text{Real}$)

UF : symboles de fonction non-interprétés

NIRA : arithmétique non-linéaire entière et réelle

$$\forall x y, x \leq y \Rightarrow t[x] \leq t[y]$$

$$t[f(r)] = r \wedge b \leq f(r) < e$$

$$m = \left\lfloor \frac{b + e}{2} \right\rfloor$$

$$\neg \left(\begin{array}{l} (t[m] < r \Rightarrow m + 1 \leq f(r) < e) \\ \wedge (t[m] > r \Rightarrow b \leq f(r) < m) \end{array} \right)$$

Prouveurs en logique du premier ordre

Limites de l'approche SMT

Possibilité de donner des axiomes quantifiés en SMT

- ▶ remplacés par des instances bien choisie

Mais peu efficace

Passer à la logique du premier ordre complète

Résolution

Résolution propositionnelle :

$$\text{Resolution } \frac{P \vee C \quad \neg P \vee D}{C \vee D}$$

À partir de l'ensemble de clauses de départ

- ▶ on essaie de dériver la clause vide \square
- ▶ auquel cas l'ensemble de départ était insatisfiable

Clauses au premier ordre

Forme prénexe :

- ▶ on fait remonter les quantifications en tête
- ▶ en faisant des renommages si besoin

$$(\forall x. A) \wedge B \rightsquigarrow \forall x. (A \wedge B) \quad \text{si } x \text{ n'est pas dans } B$$

$$(\forall x. A) \Rightarrow B \rightsquigarrow \exists x. (A \Rightarrow B) \quad \text{si } x \text{ n'est pas dans } B$$

...

Au final, la formule est de la forme :

$$Q_1 x_1. \dots Q_n x_n. A$$

avec $Q_i \in \{\forall, \exists\}$ et A sans quantificateur

Clauses au premier ordre (cont.)

Skolémisation :

- ▶ on supprime les quantifications existentielles
- ▶ en introduisant de nouveaux symboles de fonction

$$\begin{aligned} & \forall x_1. \dots \forall x_n. \exists y. A[x_1, \dots, x_n, y] \\ & \rightsquigarrow \forall x_1. \dots \forall x_n. A[x_1, \dots, x_n, f(x_1, \dots, x_n)] \end{aligned}$$

Au final, la formule est de la forme :

$$\forall x_1. \dots \forall x_n. A$$

avec A sans quantificateur

Exemple

$$\forall X Y Z. (r(X, Y) \wedge r(Y, Z)) \Rightarrow r(X, Z) \quad (\text{T})$$

$$\forall X. \exists Y. r(X, s(Y)) \quad (\text{S})$$

$$\forall X Y. r(X, Y) \Rightarrow r(s(X), s(Y)) \quad (\text{C})$$

$$\forall X. \exists Y. r(X, s(s(Y))) \quad (\text{G})$$

On veut montrer $(\text{T} \wedge \text{S} \wedge \text{C}) \Rightarrow \text{G}$

Les clauses associées à $\neg((\text{T} \wedge \text{S} \wedge \text{C}) \Rightarrow \text{G})$ sont

$$\neg r(X, Y) \vee \neg r(Y, Z) \vee r(X, Z)$$

$$r(X, s(sk_1(X)))$$

$$\neg r(X, Y) \vee r(s(X), s(Y))$$

$$\neg r(sk_2, s(s(Y)))$$

Passage au premier ordre

Règle d'instanciation :

$$\text{Instantiation } \frac{C}{\sigma C}$$

σ substitution des variables de C par des termes

Quand faire les instanciations ?

- ▶ le minimum nécessaire pour pouvoir appliquer la règle de résolution

Résolution au premier ordre

$$\text{Resolution} \frac{P \vee C \quad \neg Q \vee D}{\sigma(C \vee D)}$$

σ est l'unificateur le plus général de P et Q ,
c'est-à-dire la substitution la plus générale telle que $\sigma P = \sigma Q$

Résolution au premier ordre

$$\text{Resolution} \frac{P \vee C \quad \neg Q \vee D}{\sigma(C \vee D)}$$

σ est l'unificateur le plus général de P et Q ,
c'est-à-dire la substitution la plus générale telle que $\sigma P = \sigma Q$

On a aussi besoin de

$$\text{Factoring} \frac{P \vee Q \vee C}{\sigma(P \vee C)}$$

Complétude

Résolution réfutationnellement complète :

- ▶ si A provable
alors \square peut être obtenue à partir de $\text{CNF}(\neg A)$

Complétude

Résolution réfutationnellement complète :

- ▶ si A prouvable
alors \square peut être obtenue à partir de $\text{CNF}(\neg A)$

Si A non prouvable :

- ▶ Peut s'arrêter après avoir généré toutes les clauses possibles
 - l'ensemble de clauses était satisfiable
- ▶ Peut ne jamais s'arrêter

En pratique, besoin de restreindre les applications de
Resolution

Exemple

$$\neg r(X, Y) \vee \neg r(Y, Z) \vee r(X, Z) \quad (1)$$

$$r(V, s(sk_1(V))) \quad (2)$$

$$\neg r(I, J) \vee r(s(I), s(J)) \quad (3)$$

$$\neg r(sk_2, s(s(K))) \quad (4)$$

Exemple

$$\neg r(X, Y) \vee \neg r(Y, Z) \vee r(X, Z) \quad (1)$$

$$r(V, s(sk_1(V))) \quad (2)$$

$$\neg r(I, J) \vee r(s(I), s(J)) \quad (3)$$

$$\neg r(sk_2, s(s(K))) \quad (4)$$

$$\neg r(sk_2, Y) \vee \neg r(Y, s(s(K))) \quad \text{par (1) et (4)} \quad (5)$$

$$X \mapsto sk_2 \quad Z \mapsto s(s(K))$$

Exemple

$$\neg r(X, Y) \vee \neg r(Y, Z) \vee r(X, Z) \quad (1)$$

$$r(V, s(sk_1(V))) \quad (2)$$

$$\neg r(I, J) \vee r(s(I), s(J)) \quad (3)$$

$$\neg r(sk_2, s(s(K))) \quad (4)$$

$$\neg r(sk_2, Y) \vee \neg r(Y, s(s(K))) \quad \text{par (1) et (4)} \quad (5)$$

$$\neg r(s(sk_1(sk_2)), s(s(K))) \quad \text{par (2) et (5)} \quad (6)$$

$$V \mapsto sk_2 \quad Y \mapsto s(sk_1(sk_2))$$

Exemple

$$\neg r(X, Y) \vee \neg r(Y, Z) \vee r(X, Z) \quad (1)$$

$$r(V, s(sk_1(V))) \quad (2)$$

$$\neg r(I, J) \vee r(s(I), s(J)) \quad (3)$$

$$\neg r(sk_2, s(s(K))) \quad (4)$$

$$\neg r(sk_2, Y) \vee \neg r(Y, s(s(K))) \quad \text{par (1) et (4)} \quad (5)$$

$$\neg r(s(sk_1(sk_2)), s(s(K))) \quad \text{par (2) et (5)} \quad (6)$$

$$\neg r(sk_1(sk_2), s(K)) \quad \text{par (3) et (6)} \quad (7)$$

$$I \mapsto sk_1(sk_2) \quad J \mapsto s(K)$$

Exemple

$$\neg r(X, Y) \vee \neg r(Y, Z) \vee r(X, Z) \quad (1)$$

$$r(V, s(sk_1(V))) \quad (2)$$

$$\neg r(I, J) \vee r(s(I), s(J)) \quad (3)$$

$$\neg r(sk_2, s(s(K))) \quad (4)$$

$$\neg r(sk_2, Y) \vee \neg r(Y, s(s(K))) \quad \text{par (1) et (4)} \quad (5)$$

$$\neg r(s(sk_1(sk_2)), s(s(K))) \quad \text{par (2) et (5)} \quad (6)$$

$$\neg r(sk_1(sk_2), s(K)) \quad \text{par (3) et (6)} \quad (7)$$

$$\square \quad \text{par (2) et (7)}$$

$$V \mapsto sk_1(sk_2) \quad K \mapsto sk_1(sk_1(sk_2))$$

Traitement de l'égalité

$$\text{Equality Resolution } \frac{u \neq v \vee R}{\sigma(R)} \sigma = mgu(u, v)$$

$$\text{Neg. Superposition } \frac{s = t \vee S \quad u \neq v \vee R}{\sigma(u[t]_p \neq v \vee S \vee R)} 1$$

$$\text{Pos. Superposition } \frac{s = t \vee S \quad u = v \vee R}{\sigma(u[t]_p = v \vee S \vee R)} 1$$

$$\text{Eq. Factoring } \frac{s = t \vee u = v \vee R}{\sigma(t \neq v \vee u = v \vee R)} \sigma = mgu(s, u)$$

Avec des restrictions sur l'application des règles

1. $\sigma = mgu(u|_p, s)$

Format d'entrée

TPTP

Formula	syntaxe TPTP
$\neg p(X)$	$\sim p(X)$
$A \vee B$	$A \mid B$
$A \wedge B$	$A \& B$
$A \Rightarrow B$	$A \Rightarrow B$
$\forall x. A$	$! [X] : A$
$\exists x. A$	$? [X] : A$

fof(name, role, formula).

role : axiome, conjecture, lemme, etc.

Prouveur du premier ordre

- ▶ Vampire
 - U. de Manchester (Andrei Voronkov, Giles Regeer, etc.)
- ▶ E
 - DHBW Stuttgart (Stefan Schulz)