Démonstration automatique de théorèmes

Nicolas Daire

1. Syntaxe
   - Langage
   - Formules
   - Implémentation
2. Sémantique
   - Structures et modèles
   - Satisfiabilité
   - Compacité
3. Démonstration
   - Séquents
   - Conversion
   - Démonstration par coupure
   - Démonstration par résolution

Definition

Un langage L est un ensemble de symboles constitué :

- des parenthèses ( ), des connecteurs $\neg$ $\wedge$ $\vee$ $\Rightarrow$ $\Leftrightarrow$, et des quantificateurs $\forall$ $\exists$
- d'un ensemble $\mathfrak{V} = \{v_n\}$ infini dénombrable de variables
- d'un ensemble $\mathfrak{C}$ de symboles de constantes
- d'une réunion d'ensembles $\mathfrak{F}_n$ $(n \in \mathbb{N})$ de symboles de fonctions n-aires, où $\mathfrak{F}_0 = \mathfrak{C}$
- d'une réunion d'ensembles $\mathfrak{R}_n$ $(n \in \mathbb{N})$ de symboles de relations n-aires, où $\mathfrak{R}_0$ est l'ensemble des propositions atomiques qui contient parfois $\{\top, \bot\}$

Definition

On définit les termes, atomes et formules par induction structurelle :

- L'ensemble $\mathfrak{T}(L)$ des termes est le plus petit ensemble qui contient $\mathfrak{V}$ et qui est stable par $(t_1, \ldots, t_n) \mapsto f(t_1, \ldots, t_n)$ pour tout $f \in \mathfrak{F}_n$
- L'ensemble $\mathfrak{A}(L)$ des atomes est l'ensemble des $R(t_1, \ldots, t_n)$ où $R \in \mathfrak{R}_n$ et $(t_1, \ldots, t_n) \in \mathfrak{T}(L)^n$
- L'ensemble $\mathfrak{F}(L)$ des formules du premier ordre est le plus petit ensemble qui contient $\mathfrak{A}(L)$, et $\neg A$, $A \wedge B$, $A \vee B$, $A \Rightarrow B$, $A \Leftrightarrow B$, $\forall v_n\ A$, $\exists v_n\ A$ lorsqu'il contient A et B

Definition

Les occurences d'une variable v dans une formule F peuvent être liées ou libres :

- Si $F = \neg G$, $G \gamma H$ (γ *connecteur*), $\forall w\ G$, $\exists w\ G$ $(w \neq v)$ alors les occurences libres dans F sont les occurences libres dans G et H
- Si $F = \forall v\ G$, $\exists v\ G$ alors aucunes des occurences n'est liée

Les occurences non libres de v sont dites liées.
Une formule où aucune variable n'a d'occurence libre est dite close.
On écrit $F[v_1, \ldots, v_n]$ pour dire que les variables ayant une occurences libres sont parmi les v_i
La clôture universelle de $F[v_1, \ldots, v_n]$ est la formule close $\forall v_1 \ldots \forall v_n\ F$

Definition

On définit les substitutions dans les termes et les formules :

- Si t est un terme, on note $t_{u_1/v_1,\ldots,u_n/v_n}$ le terme où on a remplacé les occurences de v_i par le terme u_i
- Si F est une formule, on note $F_{u_1/v_1,\ldots,u_n/v_n}$ la formule où on a remplacé les occurences libres de v_i par le terme u_i.

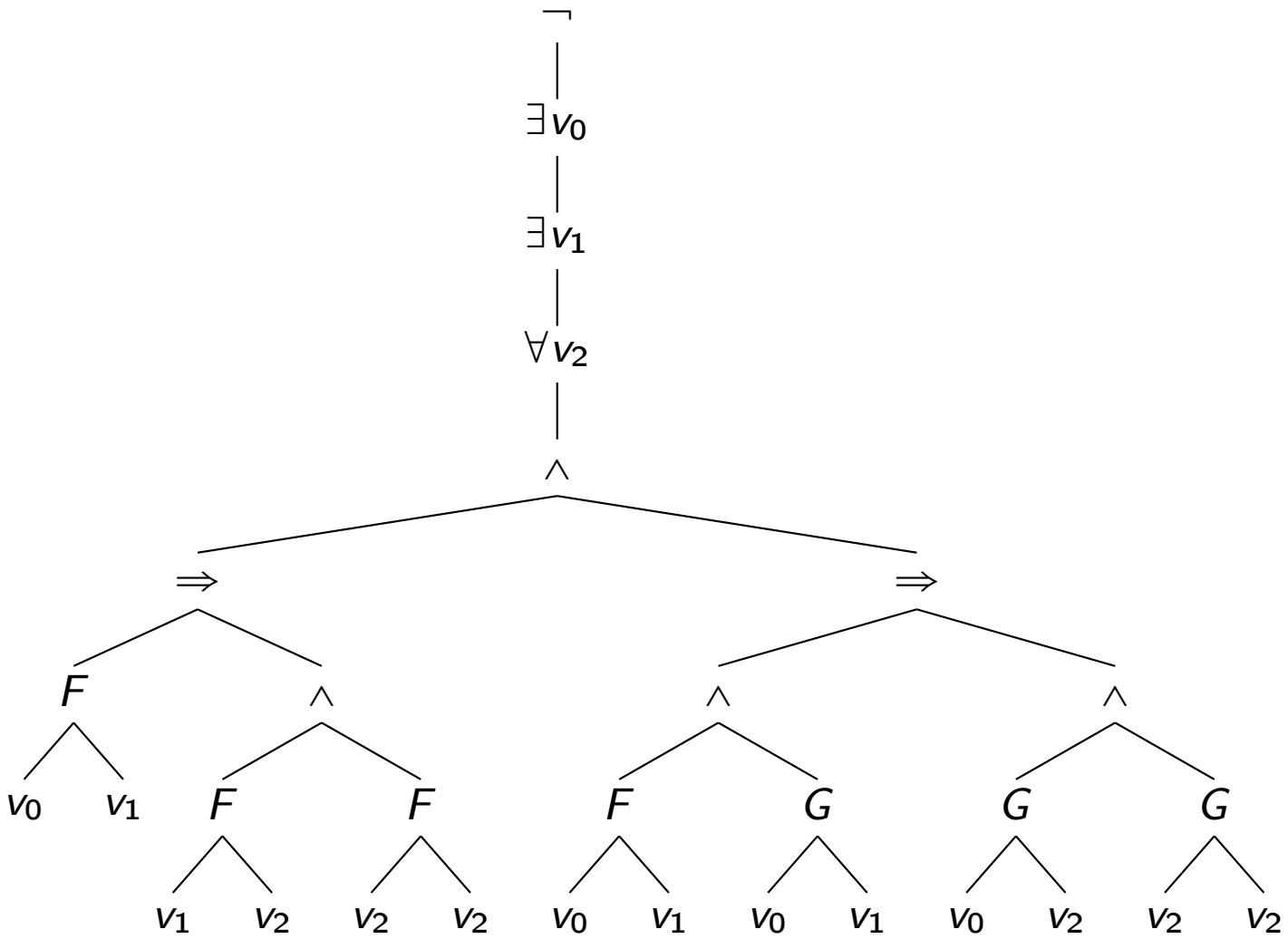
On pourra noter noter $[v_1 \longrightarrow u_1, \ldots, v_n \longrightarrow u_n]$ voire $[u_1, \ldots, u_n]$ s'il n'y a pas ambiguïté.

```
1  type term = Var of int
2            | Fn of string * term list
3
4  type atom = P of string * term list
5
6  type fol = False
7           | True
8           | Atom of atom
9           | Not of fol
10          | And of fol * fol
11          | Or of fol * fol
12          | Imp of fol * fol
13          | Iff of fol * fol
14          | Forall of int * fol
15          | Exists of int * fol
```

$$\exists x \exists y \forall z (F(x,y) \Rightarrow (F(y,z) \wedge F(z,z))) \wedge ((F(x,y) \Rightarrow G(x,y)) \Rightarrow (G(x,z) \wedge G(z,z)))$$

```
# let f = parse "~(exists x. exists y. forall z.
        (F(x,y) ==> (F(y,z) /\ F(z,z))) /\
        ((F(x,y) /\ G(x,y)) ==> (G(x,z) /\ G(z,z))))";;

  val f : Base.fol =
 Not
  (Exists (0,
    Exists (1,
     Forall (2,
      And
       (Imp (Atom (P ("F", [Var 0; Var 1])),
         And (Atom (P ("F", [Var 1; Var 2])),
          Atom (P ("F", [Var 2; Var 2])))),
        Imp
         (And (Atom (P ("F", [Var 0; Var 1])),
           Atom (P ("G", [Var 0; Var 1]))),
          And (Atom (P ("G", [Var 0; Var 2])), Atom (P ("G", [Var
               2; Var 2])))))))))
```


¬
∃ v_0
∃ v_1
∀ v_2
∧
⇒
⇒
F
∧
∧
∧
v_0
v_1
F
F
F
G
G
G
v_1
v_2
v_2
v_2
v_0
v_1
v_0
v_1
v_0
v_2
v_2
v_2

Definition

Une L-structure $\mathfrak{M}$ est constituée :

- d'un ensemble de base M
- d'éléments $\overline{c}^{\mathfrak{M}}$ interprétant les symboles de constantes $c \in \mathfrak{C}$
- de fonctions $\overline{f}^{\mathfrak{M}} : M^k \to M$ interprétant les symboles de fonctions $f \in \mathfrak{F}$
- de sous-ensembles $\overline{R}^{\mathfrak{M}} \subseteq M^k$ interprétant les symboles de relations $R \in \mathfrak{R}$

Definition

Un environnement e est une fonction $\mathfrak{V} \to M$.
La valeur d'un terme t dans un environnement e est défini par induction :

- $V(c) = \overline{c}$
- $V(v) = e(v)$
- $V(f(v_1, \ldots, v_n) = f(V(v_1, \ldots, V(v_n))$

La valeur d'une formule F dans un environnement e est une fonction à valeurs dans $\{0, 1\}$ définie par induction avec le sens courant que l'on donne aux symboles.

Definition

On dit que $\mathfrak{M}$ satisfait F dans l'environnement e, noté $\mathfrak{M}, e \models F$, lorsque $V_{\mathfrak{M}}(F, e) = 1$.
On dit que $\mathfrak{M}$ satisfait F, ou $\mathfrak{M}$ est un modèle de F, noté $\mathfrak{M} \models F$, lorsque $\mathfrak{M}$ satisfait F dans tout environnement e, où de manière équivalente que $\mathfrak{M}$ satisfait la clôture universelle de F.

Definition

- Une formule close F est universellement valide, noté $\models F$, si elle est satisfaite dans toute L-structure.
  Si $\models \neg F$, on dit que F est contradictoire.
- Deux formules F et G sont universellement équivalentes, noté $F \sim G$, si $\models F \Leftrightarrow G$.
- Une théorie est un ensemble de formules closes, souvent appelées axiomes.
- Une L-structure $\mathfrak{M}$ satisfait la théorie T, ou est un modèle de T, noté $\mathfrak{M} \models T$, lorsque $\mathfrak{M}$ satisfait toutes les formules de T.
- Une théorie T est consistente si elle admet un modèle, sinon elle est contradictoire.
- Une formule close F est conséquence sémantique de T si $\mathfrak{M} \models F$ pour tout modèle $\mathfrak{M}$ de T.
- Deux théories T_1 et T_2 sont équivalentes lorsque $\forall F \in T_1, T_2 \models F$ et inversement.

Theorem (Tychonov)

L'espace topologique produit d'une famille d'espaces compacts est compact.

Theorem (Compacité du calcul propositionnel)

Un ensemble T de formules du calcul propositionnel est satisfiable si et seulement si T est finiment satisfiable.

Theorem (Compacité du calcul des prédicats)

Une théorie T dans un langage du premier ordre est consistente si et seulement si T est finiment consistente.

Lemma

Soient $A_1, \ldots, A_n$ des variables propositionnelles, $J[A_1, \ldots, A_n]$ une formule propositionnelle, et $F_1, \ldots, F_n$ des formules du premier ordre. Si J est une tautologie, alors $\models J[F_1, \ldots, F_n]$.

Lemma

Si G est une sous-formule de F, et $G \sim G'$, alors $F \sim F'$ obtenue à partir de F en remplaçant une occurence de G par G'.

Theorem

Toute formule du premier ordre est universellement équivalente à une formule n'utilisant que les symboles de connecteur d'un système complet ainsi que l'un des symboles de quantification $\forall$ et $\exists$.

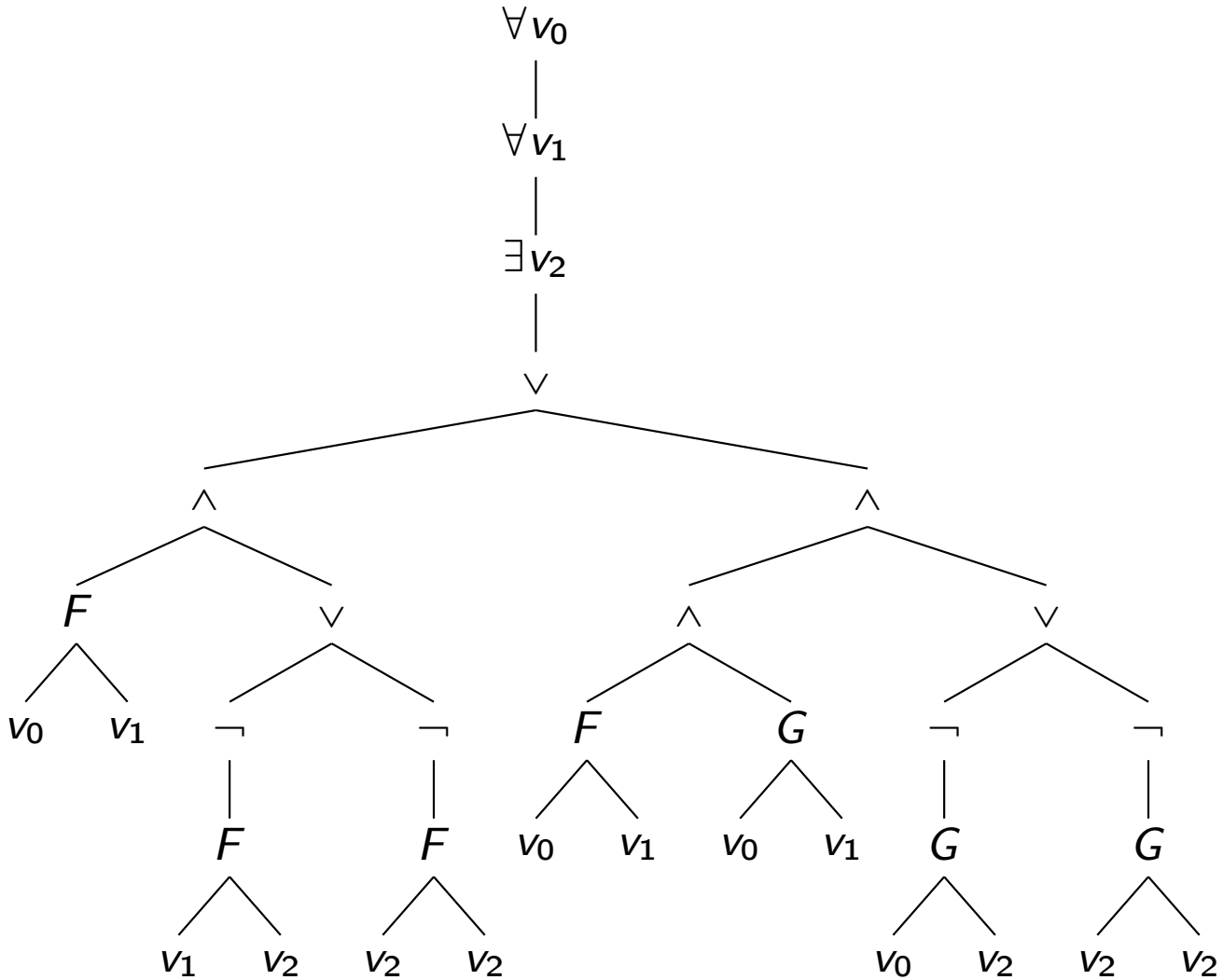
Une formule est sous forme prénexe si elle s'écrit $Q_1 v_1 \dots Q_n v_n F$ où $Q_1, \dots, Q_n$ sont des quantificateurs, $v_1, \dots, v_n$ sont des variables et F est une formule sans quantificateur. On appelle alors $Q_1 v_1 \dots Q_n v_n$ son préfixe.
Une formule sous forme prénexe est polie si les variables quantifiées sont deux à deux distinctes.
Une forme prénexe est universelle (resp. existentielle) si elle ne contient pas de quantificateur existentiel (resp. universel).

Theorem

Toute formule du premier ordre est universellement équivalente à une formule sous forme prénexe polie.

```
# prenex f;;

- : Base.fol =
Forall (0,
 Forall (1,
  Exists (2,
   Or
    (And (Atom (P ("F", [Var 0; Var 1])),
      Or (Not (Atom (P ("F", [Var 1; Var 2])))),
       Not (Atom (P ("F", [Var 2; Var 2]))))),
     And
      (And (Atom (P ("F", [Var 0; Var 1])), Atom (P ("G", [Var 0;
           Var 1])))),
      Or (Not (Atom (P ("G", [Var 0; Var 2])))),
       Not (Atom (P ("G", [Var 2; Var 2])))))))))
```


∀v0
∀v1
∃v2
∨
∧
∧
F
∨
∧
∨
v0
v1
¬
¬
F
G
¬
¬
F
F
v0
v1
v0
v1
G
G
v1
v2
v2
v2
v0
v2
v2
v2

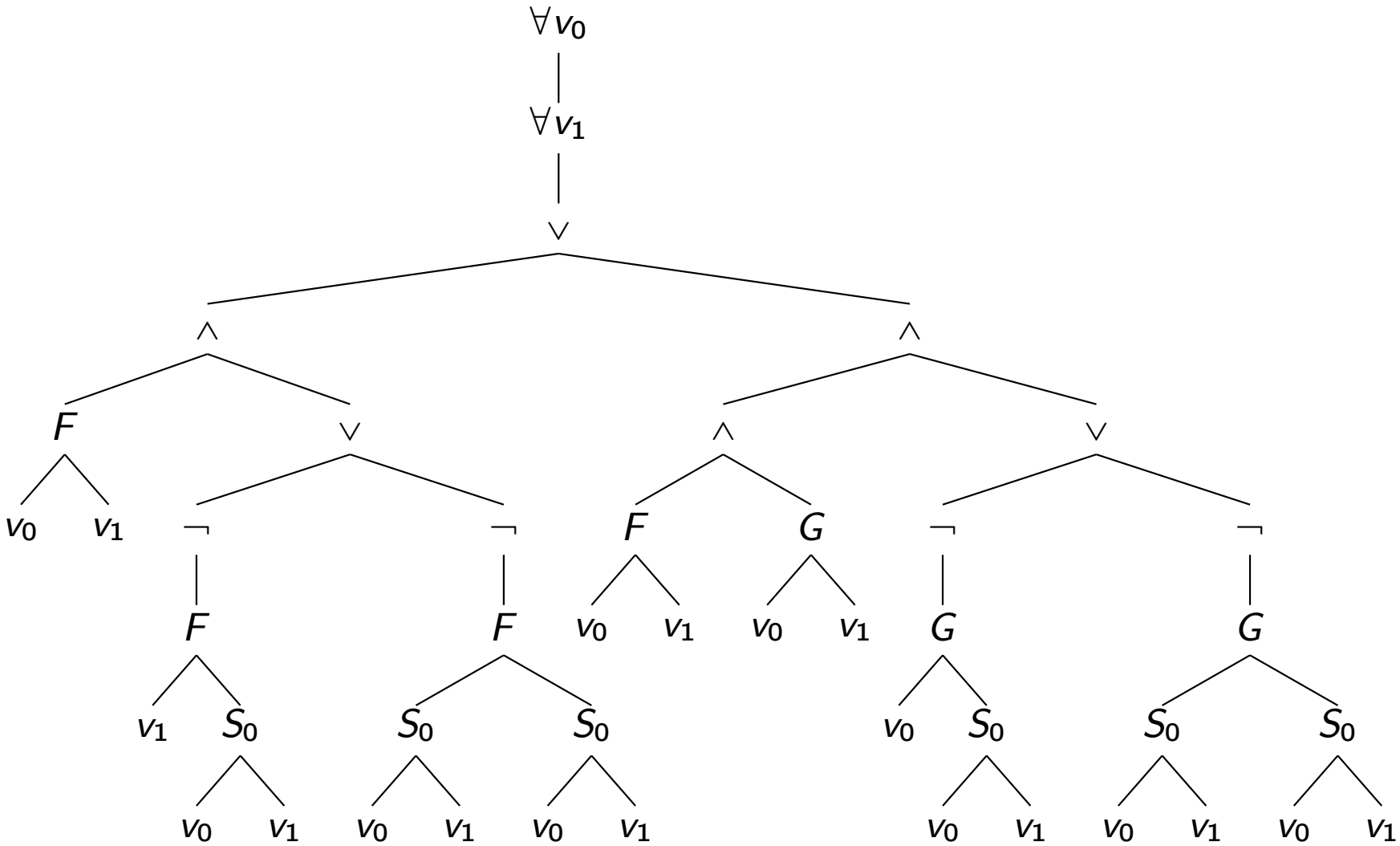
On appelle forme de Skolem de F la formule F_S obtenue à partir de F en remplaçant les variables quantifiées existentiellement par de nouveaux symboles de fonction appelés fonctions de Skolem.

Theorem

F et sa forme de Skolem F_S sont équisatisfiables.

```
# skolemization f;;

- : Base.fol =
Forall (0,
 Forall (1,
  Or
   (And (Atom (P ("F", [Var 0; Var 1])),
     Or (Not (Atom (P ("F", [Var 1; Fn ("S#0", [Var 0; Var 1])])
         )),
      Not
       (Atom
         (P ("F", [Fn ("S#0", [Var 0; Var 1]); Fn ("S#0", [Var
             0; Var 1])])))))),
   And (And (Atom (P ("F", [Var 0; Var 1])), Atom (P ("G", [Var
       0; Var 1])))),
    Or (Not (Atom (P ("G", [Var 0; Fn ("S#0", [Var 0; Var 1])]))
        ),
     Not
      (Atom
        (P ("G", [Fn ("S#0", [Var 0; Var 1]); Fn ("S#0", [Var 0;
            Var 1])]))))))))
```


∀v0
∀v1
∨
∧
∧
F
∨
∧
∨
v0
v1
¬
¬
F
G
¬
¬
F
F
v0
v1
v0
v1
G
G
v1
S0
S0
S0
v0
S0
S0
S0
v0
v1
v0
v1
v0
v1
v0
v1
v0
v1
v0
v1

Deux règles de déduction :

- Le modus ponens : $\dfrac{F,\ F \Rightarrow G}{G}$
- La règle de généralisation : $\dfrac{F}{\forall v\ F}$

Des axiomes logiques :

- Les tautologies
- Les axiomes de quantificateurs :
  - $\exists v\ F \Leftrightarrow \neg \forall v\ \neg F$
  - $\forall v\ (F \Rightarrow G) \Rightarrow (F \Rightarrow \forall v\ G)$ si v n'a pas d'occurence dans F
  - $\forall v\ F \Rightarrow F_{t/v}$ si les occurences libres de v ne sont pas dans le champ d'un quantificateur liant une variable de t

Definition

On appelle démonstration de F dans T une suite $\mathfrak{D} = (F_0, \ldots, F_n)$ avec $F_n = F$ vérifiant pour tout i une des conditions suivantes :

- $F_i \in T$
- F_i est un axiome logique
- F_i se déduit à partir de formules parmi $F_0, \ldots, F_{i-1}$ et d'une règle de déduction

S'il existe un démonstration de F dans T on dit que F est conséquence syntaxique de T, ou que T démontre F, noté $T \vdash F$.

Definition

- T est cohérente si on n'a jamais $T \vdash F$ et $T \vdash \neg F$, ou de manière équivalente $T \nvdash \bot$, ou encore il existe une formule F telle que $T \nvdash F$.
- T est complète si T est cohérente et pour toute formule close F, $T \vdash F$ ou $T \vdash \neg F$.

Theorem

Soit F une formule close. Si $T, F \vdash G$, alors $T \vdash F \Rightarrow G$.

Corollary

$T \vdash F$ si et seulement si $T \cup \{\neg F\}$ est incohérente.

Definition

Une preuve par réfutation de F dans T est la démonstration $T, F \vdash \bot$.

Theorem

Si $T \vdash F$, alors $T \models F'$ où F' est une clôture universelle de F.

Corollary

Si T est consistante, alors T est cohérente.

Theorem

Si T est cohérente, alors T est consistante.

Theorem (Complétude du calcul des prédicats)

Soient T une théorie et F une formule close, $T \vdash F$ si et seulement si $T \models F$.

Definition

- Un littéral L est un atome A ou sa négation $\neg A$.
- Une clause est une disjonction de littéraux $L_1 \vee \cdots \vee L_n$. On peut aussi écrire $A_1 \wedge \cdots \wedge A_m \Rightarrow B_1 \vee \cdots \vee B_n$ où $A1, \ldots, A_m, B_1, \ldots, B_n$ sont des atomes, auquel cas on appelle $A_1 \wedge \cdots \wedge A_m$ la prémisse et $B_1 \vee \cdots \vee B_n$ la conclusion de la clause.
- Une formule du premier ordre sous forme normale clausale est une conjonction de clauses $\bigwedge_i \bigvee_j L_{ij}$.

Theorem

Toute formule F admet une forme normale clausale équisatisfiable.

```
1  type literal = L of atom | NL of atom
2
3  type clause = literal list
4
5  type cnf = clause list
```

```
# let c = convert_to_cnf f;;

val c : Base.cnf =
  [[L (P ("F", [Var 0; Var 1])); L (P ("F", [Var 0; Var 1]))];
   [L (P ("F", [Var 0; Var 1])); L (P ("G", [Var 0; Var 1]))];
   [L (P ("F", [Var 0; Var 1]));
    NL (P ("G", [Var 0; Fn ("S#0", [Var 0; Var 1])]));
    NL (P ("G", [Fn ("S#0", [Var 0; Var 1]); Fn ("S#0", [Var 0;
        Var 1])]))];
   [NL (P ("F", [Var 1; Fn ("S#0", [Var 0; Var 1])]));
    NL (P ("F", [Fn ("S#0", [Var 0; Var 1]); Fn ("S#0", [Var 0;
        Var 1])]));
    L (P ("F", [Var 0; Var 1]))];
   [NL (P ("F", [Var 1; Fn ("S#0", [Var 0; Var 1])]));
    NL (P ("F", [Fn ("S#0", [Var 0; Var 1]); Fn ("S#0", [Var 0;
        Var 1])]));
    L (P ("G", [Var 0; Var 1]))];
   [NL (P ("F", [Var 1; Fn ("S#0", [Var 0; Var 1])]));
    NL (P ("F", [Fn ("S#0", [Var 0; Var 1]); Fn ("S#0", [Var 0;
        Var 1])]));
    NL (P ("G", [Var 0; Fn ("S#0", [Var 0; Var 1])]));
    NL (P ("G", [Fn ("S#0", [Var 0; Var 1]); Fn ("S#0", [Var 0;
        Var 1])]))]]
```

- La règle de simplification : $\dfrac{A \vee A \vee B_1 \vee \cdots \vee B_n}{B_1 \vee \cdots \vee B_n}$ où $A, B_1, \ldots, B_n$ sont des littéraux.
- La règle de coupure qui généralise le modus ponens : $\dfrac{A \vee B_1 \vee \cdots \vee B_m,\ \neg A \vee C_1 \vee \cdots \vee C_n}{B_1 \vee \ldots B_m \vee C_1 \vee \cdots \vee C_n}$ où $A, B_1, \ldots, B_m, C_1, \ldots, C_n$ sont des littéraux.

On peut combiner les deux règles :

$$\frac{A \vee \cdots \vee A \vee B_1 \vee \cdots \vee B_m,\ \neg A \vee \cdots \vee \neg A \vee C_1 \vee \cdots \vee C_n}{B_1 \vee \ldots B_m \vee C_1 \vee \cdots \vee C_n}$$

Theorem (Complétude de la démonstration par coupure)

Tout ensemble de clauses insatisfiable est réfutable par coupure.

Pour déterminer si un ensemble Γ est satisfiable, on élimine d'abord les tautologies et on simplifie les clauses, puis on applique exhaustivement la règle de coupure sur tous les couples de clauses. Le nombre de clauses simplifiées est fini donc la procédure termine, il suffit alors de regarder si on a pu dériver la clause vide.

Definition

Soit $S = \{(t_1, u_1), \ldots, (t_n, u_n)\}$ un ensemble de couples de termes. On appelle unificateur de S toute substitution σ telle que $\sigma(t_i) = \sigma(u_i)$ pour tout i.
On appelle unificateur principal de S tout unificateur π tel que si σ est un unificateur de S il existe une substitution τ telle que $\sigma = \tau \circ \pi$.

Theorem

Si S admet un unificateur, alors S admet un unificateur principal.

On introduit deux procédures :

- Réduction : si $(f(t_1, \ldots, t_n), f(u_1, \ldots, u_n)) \in S$, on remplace le couple dans S par les couples $(t_1, u_1), \ldots, (t_n, u_n)$.
- Elimination : si $(v, t) \in S$ où v est une variable, on applique la substitution $(v \Rightarrow t)$ à chaque membre des autres couples de S.

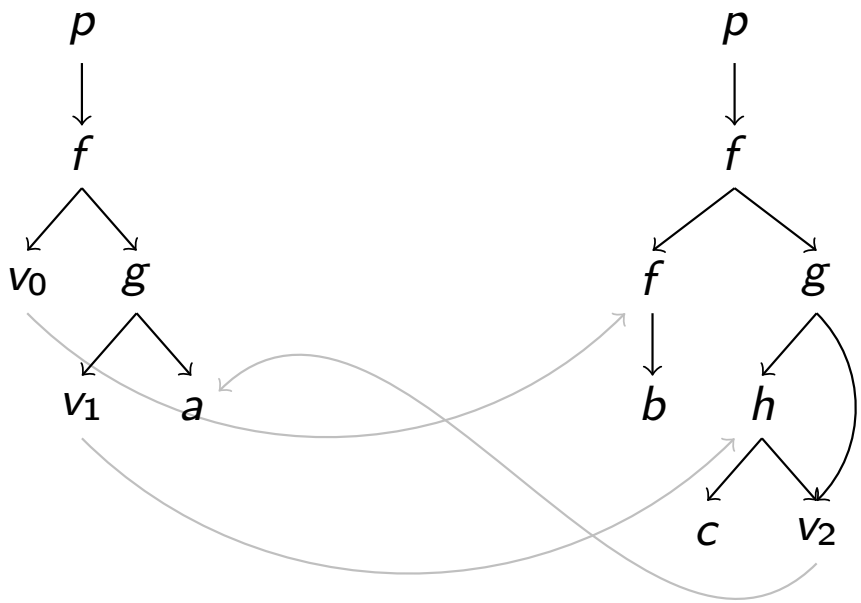
On exécute les procédures suivantes tant que possible :

- Si $(t, v) \in S$ où v est une variable mais pas t, on remplace le couple par (v, t).
- Si $(v, v) \in S$ où v est une variable, on retire le couple de S.
- Si $(t, u) \in S$ où t et u ne sont pas des variables, si t et u ne commencent pas par les mêmes symboles de fonction on termine avec échec, sinon on applique la réduction.
- Si $(v, t) \in S$ où v est une variable qui a une autre occurrence dans S et $v \neq t$, si v a une occurrence dans t on termine avec échec, sinon on applique l'élimination.

Unification du système $\{(f(v_0, g(v_1, a)), f(f(b), g(h(c, v_2), v_2))\}$, dont un unificateur principal est $[v_0 \rightarrow f(b), v_1 \rightarrow h(c, a), v_2 \rightarrow a]$.

```
1  # let t1 = Fn ("f", [Var 0; Fn ("g", [Var 1; Fn ("a", [])])]);;
2  # let t2 = Fn ("f", [Fn ("f", [Fn ("b", [])]); Fn ("g", [Fn ("h"
       , [Fn ("c", []); Var 2]); Var 2])]);;
3  # let g = global_make 16;;
4  # unify_terms g t1 t2;;
5  - : bool = true
6  # g;;
7  - : Base.global =
8  {graph =
9    [|{n = T (Fn ("f", [Fn ("b", [])])); p = 0; r = -2};
10     {n = T (Fn ("h", [Fn ("c", []); Fn ("a", [])])); p = 1; r =
           -2};
11     {n = T (Fn ("a", [])); p = 2; r = -2}; ... \];
12   max = -1; vars = [2; 1; 0]}
```

L'algorithme précédent a une complexité temporelle exponentielle. On peut obtenir un algorithme presque linéaire en utilisant la structure de graphe orienté acyclique pour représenter les termes et les substitutions, et la structure Union-Set pour calculer les affectations, ainsi le test occur-check n'est mené q'une seule fois à la fin pour déterminer si le graphe est bien acyclique.



Soient C et D deux clauses séparées. Soient $X \subseteq C^+$ et $Y \subseteq D^-$ non vides tels que $X \cup Y$ soit unifiable. Alors la règle de résolution s'écrit

$$\frac{C,\ D'}{\pi_{X \cup Y}(C \backslash X,\ D' \backslash Y)}$$

où π est un unificateur principal.

Theorem

Si E se déduit de C, D, tout modèle de C, D est aussi un modèle de E.

Definition

Soi Γ un ensemble de clause. On appelle réfutation de Γ toute suite de clause $(C_1, \ldots, C_n)$ se terminant par la clause vide telle que pour tout i, $C_i \in \Gamma$ ou C_i se déduit de deux clauses antérieures de la suite. On dit que Γ est réfutable s'il existe une réfutation de Γ.

Theorem (Complétude de la méthode de résolution)

Un ensemble de clauses Γ *est réfutable si et seulement si* Γ *est inconsistant.*

Méthode de Herbrand : on cherche un modèle dont l'ensemble de base est $\mathfrak{T}(L)$, et les symboles fonctions ont leur interprétation naturelle. Pour tout $\delta \in \{0,1\}^{\mathfrak{A}(L)}$ on définit une interprétation $\mathfrak{I}_\delta$ dans laquelle si R est un symbole de relation et $t_1, \ldots, t_n$ des termes, $(t_1, \ldots, t_n) \in R$ si et seulement si $\delta(R(t_1, \ldots, t_n)) = 1$.
On se ramène ainsi au calcul propositionnel sur lequel on a une procédure de réfutation complète, à partir de laquelle on peut déduire une réfutation du premier ordre.
Ainsi pour savoir si $\Gamma \models F$, on essaie de réfuter $\Gamma, \neg F$. On maintient deux listes correspondant aux clauses utilisées et non utilisée, et on défile la liste non utilisées en résolvant chaque clause avec toutes les clauses de la liste utilisées, jusqu'à déduire la clause vide ou vider la liste non utilisées. Si F est prouvable l'algorithme termine.

Definition

Une clause C subsume une clause D, noté $C \leqslant D$, s'il existe une substitution σ telle que $\sigma(C) \subseteq D$.

Theorem

Si $C \leqslant C'$, alors tout résolvant de C', D est subsumé par C ou un résolvant de C, D.

Corollary

Si $\Gamma \vdash C'$, alors il existe une démonstration $\Gamma \vdash C \leqslant C'$ où aucune clause n'est subsumée par une clause antérieure.

On peut donc ignorer les résolvants subsumés par des clauses antérieures, et remplacer les clauses subsumées par les résolvants dans la liste utilisées.

Theorem

Soit $\mathfrak{M}$ une L-structure. Si Γ est réfutable, il existe une réfutation qui ne résoud deux clauses que si l'une au moins n'est pas satisfaite dans $\mathfrak{M}$.

Si Γ satisfait les deux clauses la contradiction doit venir des autres clauses, donc on peut trouver une telle réfutation.
La résolution positive est un cas particulier où l'on choisit une interprétation dans laquelle toutes les formules atomiques sont fausses, les résolutions font donc toujours intervenir au moins une clause constituée uniquement de littéraux positifs.