Programme

- Introduction
- Le langage de la LP (syntaxe)
- La sémantique de la LP
- Équivalence logique et Substitution
- Conséquence logique
- Formes normales et clausale
- **Méthode de résolution**
- Méthode des tableaux
- Méthode de Davis et Putnam
- Initiation à la logique des prédicats

Origine

- Méthode simple constituée d'une règle (dite règle de résolution) permettant de produire une clause à partir de clauses existantes. Cette règle est appliquée itérativement jusqu'à tomber sur la clause vide
- Cette méthode permet de déterminer si une forme clausale est insatisfiable, elle permet donc de savoir si
  - Une fbf est insatisfiable (en passant à sa forme clausale)
  - Une fbf C est la conséquence logique d'un ensemble de fbf $\{H_1, \ldots H_k\}$ (en passant à la forme clausale de $H_1 \wedge \ldots \wedge H_k \wedge \neg C$)
- Cette méthode généralisée à la logique des prédicats et restreinte aux clauses de Horn est à la base du langage Prolog

Règle de résolution

- **Définition**

  « Soit $C = \{p, L_1, \ldots, L_k\}$ et $C' = \{\neg p, M_1, \ldots, M_m\}$ deux clauses ayant des littéraux opposés, la ***résolvante*** *de C et C' selon p est la clause $res(C,C',p)=\{L_1,\ldots,L_k,M_1\ldots,M_m\}$ obtenue par union des littéraux restants »*

- *Exemple :*
  - $res(\{\neg p,q,r\},\{p,q,\neg s\},p) = \{q,r,\neg s\}$
  - $res(\{\neg p,q,r\},\{p,\neg q,\neg s\},p) = \{q,\neg q,r,\neg s\} \equiv \mathrm{T}$
  - $res(\{\neg p,q,r\},\{p,\neg q,\neg s\},q) = \{p,\neg p,r,\neg s\} \equiv \mathrm{T}$
  - $res(\{\neg p\},\{p\},p) = \varnothing \equiv \perp$

Propriétés

- *Si plusieurs résolvantes peuvent être calculées à partir de deux clauses C et C' alors ces résolvantes sont logiquement équivalentes et valides*
  - On se contentera de noter res(C,C') la résolvante de deux clauses
- *La règle de résolution est une règle d'inférence (i.e {C, C'} |= Res(C,C'))*

Définition d'une séquence de résolutions

Soit F une forme clausale et C une clause. On a ***F*** $\boldsymbol{|\!\!-\!\!-_{res}}$ ***C*** *ssi il existe une séquence finie de clauses* $(C_1, C_2, ..., C_r)$ *avec :*

- $C_r = C$
- *et pour tout* $i=1,...,r$ *:*
  - C_i *est dans F*
  - *ou* C_i *est une résolvante de deux clauses avant* C_i *dans la séquence (i.e. il existe* $l < i$ *et* $k < i$ C_i *étant une résolvante de* C_l *et* C_k*)*

Théorème

Une forme clausale F est insatisfiable ssi F $\vdash_{res}$ Ø

- Exemple
  - C1={a,d} C4={a,e}
  - C2={c,¬d} C5={¬c,¬e}
  - C3={¬a,e} C6={¬a,d}
- Dérivation 1
  - C1,C2,{a,c},C5,{a,¬e},C4,{a},C6,{d},C3,{e},{¬c},{¬d},∅
- Dérivation 2
  - C1,C6,{d},C3,C4,{e},C2,C5,{¬d,¬e},{¬e}, ∅

Théorème

Une forme clausale F est insatisfiable ssi F $|\!-\!-_{res}$ Ø

- Preuve
  - ←Facile : la règle de résolution est une règle d'inférence donc $F \models \perp$ et donc $F \wedge \neg\perp \equiv F$ insatisfiable
  - →Plus difficile :
    - Lemme :
    - *« Soit L un littéral d'une forme clausale F et soit F^L la forme clausale obtenue en supprimant les clauses contenant le littéral L et en supprimant le littéral opposé à L dans les autres clauses de F, on a si F est insatisfiable alors F^L est insatisfiable »*
    - F n'est pas vide car une forme clausale vide est valide
    - Par récurrence sur le nombre n de littéraux de F

La méthode de résolution en pratique…

- Il est inutile de calculer les résolvantes des clauses contenant plusieurs littéraux opposés car cela produit des clauses valides que l'on peut supprimer
  - On peut refaire les démonstrations en imposant dans la définition de la règle de résolution de ne pas utiliser de clauses valides
- L'idée est d'essayer de produire des clauses de plus en plus petites car la clause vide ne peut être produite que par des clauses singleton
- Dès que la forme clausale est finie, alors on peut exhiber un algorithme qui produit toutes les résolvantes possibles à partir de cette forme clausale
  - Ainsi la méthode de résolution fournie une **procédure de décision** pour la logique des propositions
  - La complexité d'un tel algorithme est exponentielle en nombre de littéraux
  - Il existe un algorithme polynomial pour les formes clausales dont les clauses sont réduites à 2 littéraux (2-SAT)
  - Le problème de la satisfiabilité d'une proposition est NP-complet dès 3-SAT
- Différentes stratégies adaptées à des Formes Clausales particulières :
  - Largeur
  - SLD Résolution (celle de Prolog est incomplète : pb du choix de la règle à unifier avec le but !!!)
  - Unit résolution complète sur les clauses de Horn

Application

- Soit le problème :

$$u, (w \rightarrow u), (w \rightarrow v), (t \rightarrow v), (u \rightarrow (w \vee t)) \models v \ ?$$

ssi $u \wedge (w \rightarrow u) \wedge (w \rightarrow v) \wedge (t \rightarrow v) \wedge (u \rightarrow (w \vee t)) \wedge \neg v$ insatisfiable ?

ssi $\{\{u\},\{u, \neg w\},\{v,\neg w\},\{\neg t,v\},\{t,\neg u,w\},\{\neg v\}\}$ insatisfiable ?

ssi $\{\{u\},\{v,\neg w\},\{\neg t,v\},\{t,\neg u,w\},\{\neg v\}\}$ insatisfiable ?

ssi $\{\{u\},\{v,\neg w\},\{\neg t,v\},\{t,\neg u,w\},\{\neg v\}\} \vdash_{res} \emptyset$?

- Résolution
  - On a la dérivation : C1,C4,$\{t,w\}$,C3,$\{v,w\}$,C2,$\{v\}$,C5,$\emptyset$
  - Donc : $u, (w \rightarrow u), (w \rightarrow v), (t \rightarrow v), (u \rightarrow (w \vee t)) \models v$

Application (suite)

- Soit le problème :

$$u,(w \rightarrow u),(w \rightarrow w),(t \rightarrow v) \models v \quad ?$$

ssi $u \wedge (w \rightarrow u) \wedge (w \rightarrow w) \wedge (t \rightarrow v) \wedge \neg v$ insatisfiable ?

ssi $\{\{u\},\{u, \neg w\},\{w,\neg w\},\{\neg t,v\},\{\neg v\}\}$ insatisfiable ?

ssi $\{\{u\},\{\neg t,v\},\{\neg v\}\}$ insatisfiable ?

ssi $\{\{u\},\{\neg t,v\},\{\neg v\}\} \mid\!\!-\!\!-_{res} \varnothing$?

- Résolution
  - On calcule l'ensemble de **toutes les clauses** dérivables par res : $Res(F)=\{\{u\},\{\neg t,v\},\{\neg v\},\{\neg t\}\}$
  - $\varnothing \notin Res(F)$ donc : $u,(w \rightarrow u),(w \rightarrow w),(t \rightarrow v) \not\models v$

Implémentation de la méthode

- Filtrage initial : on élimine les clauses tautologiques et les clauses subsumées
- On se dote d'une implémentation de la règle de résolution

  résolvable(c,c') vrai si deux clauses sont résolvables

  résolvante(c,c') qui retourne la clause résolvante de deux clauses résolvables
- On met en œuvre une stratégie en largeur
  - Soit E_0 l'ensemble initial de clauses, on calcule E_1 l'ensemble de clauses produites à partir des clauses de E_0
  - Puis E_2 l'ensemble des clauses produites à partir d'une clause de E1 et d'une clause de $E_0 \cup E_1$ (inutile de refaire les clauses de E_0 entre-elles)
  - …
  - A chaque étape, E_i est produit à partir d'une clause de E_{i-1} et d'une clause de E_j avec $j<i$.
  - Quand aucune **nouvelle** clause n'est produite on s'arrête.

Stratégie Largeur

appel initial : *résolutionLargeur({},E_0)*

<u>Algorithme :</u> résolutionLargeur

<u>Données :</u> E et N deux ensembles de clauses. L'ensemble des résolvantes des clauses de E sont supposées appartenir à E ou N (et E et N sont disjoints)

<u>Résultat :</u> L'ensemble de clauses obtenues par résolution à partir des clauses de E et N (sans résoudre entre-elles les clauses de E)

```
Var P : ensemble de clauses;
  si N={} alors E
  sinon
    P ← {} ;
    pour tout c ∈ N faire
      pour tout c' ∈ E∪N faire
        si resolvable(c,c') alors
          r ← résolvante(c,c');
          si r ∉ E∪N∪P alors P ← P∪{r} finsi;
        finsi;
      finpour;
    finpour;
    résolution(E∪N, P);
  finsi;
```

Adaptation à la recherche de la clause vide

<u>Algorithme :</u> clauseVideParRésolution

<u>Données :</u> E et N deux ensembles de clauses. L'ensemble des résolvantes des clauses de E sont supposées appartenir à E ou N (et E et N sont disjoints)

<u>Résultat :</u> vrai si on peut produire la clause vide par résolution à partir des clauses de E et N (sans résoudre entre-elles les clauses de E), faux sinon.

```
Var P : ensemble de clauses;
 si N={} alors faux
 sinon
  | si ∅ ∈ N alors vrai
  | sinon
  |  | P ← {} ;
  |  | pour tout c ∈ N faire
  |  |  | pour tout c' ∈ E∪N faire
  |  |  |  | si resolvable(c,c') alors
  |  |  |  |  | r ← résolvante(c,c');
  |  |  |  |  | si r ∉ E∪N∪P alors P ← P∪{r} finsi;
  |  |  |  | finsi;
  |  |  | finpour;
  |  | finpour;
  |  | clauseVideParRésolution(E∪N, P);
  | finsi;
 finsi;
```

On peut optimiser en éliminant les clauses subsumées produites à chaque étape.