Programme

- Introduction
- La syntaxe de la LP
- La sémantique de la LP
- Equivalence logique et Substitution
- Conséquence logique
- Modélisation
- Théorie de la preuve
- Méthode des séquents
- Formes normales et clausale
- **Méthode de résolution**
- Méthode de Davis et Putnam

Origine

- Méthode de démonstration constituée d'une seule 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 $H_1 \wedge \ldots \wedge H_k \wedge \neg C$)
  - Une fbf F est valide (en passant à la forme clausale de $\neg F$)

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

- *Exemple :*
  - $res(\{\neg p,q,r\},\{p,q,\neg s\},p) = \{q, r, \neg s\}$
  - $res(\{\neg p,q,r\},\{p,\neg q,\neg s\},p) = \{q,\neg q, r,\neg s\}$
  - $res(\{\neg p,q,r\},\{p,\neg q,\neg s\},q) = \{p, \neg p, r, \neg s\}$
  - $res(\{\neg p\},\{p\},p) = \emptyset$

Séquence de résolutions

Soit F une forme clausale et C une clause, on dit que C a été produit à partir de F par une séquence de résolution, et l'on note **$F \vdash_{res} C$**, *s'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 précédentes de la séquence (i.e. il existe l<i et k<i tels que* C_i *est une résolvante de* C_l *et* C_k*)*

Démonstration par résolution

- La méthode de résolution est un système de démonstration correct et complet pour les formes clausales :

  Soit F une forme clausale : F est insatisfiable ssi $F \vdash_{res} \emptyset$

- On peut présenter ***res*** comme un système formel de démonstration comportant un axiome et une règle :

$$\frac{}{\Delta, \perp \vdash}\ \text{clause vide} \qquad\qquad \frac{\Delta, D \vee p, D' \vee \neg p, D \vee D' \vdash}{\Delta, D \vee p, D' \vee \neg p \vdash}\ \text{résolution}$$

Où : Δ est un ensemble de clauses,
⊥ représente la clause vide,
D et D' sont des disjonctions (possiblement vides) de littéraux,
p est un symbole propositionnel

Exemple de production de la clause vide

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

- Soit F= $\{C_1,C_2,C_3,C_4,C_5,C_6\}$ avec :

$C_1=\{a,d\}$

$C_2=\{c,\neg d\}$

$C_3=\{\neg a,e\}$

$C_4=\{a,e\}$

$C_5=\{\neg c,\neg e\}$

$C_6=\{\neg a,d\}$

$C_7=res(C_1,C_2)=\{a,c\}$

$C_8=res(C_5,C_7)=\{a,\neg e\}$

$C_9=res(C_4,C_8)=\{a\}$

$C_{10}=res(C_6,C_9)=\{d\}$

$C_{11}=res(C_3,C_9)=\{e\}$

$C_{12}=res(C_5,C_{11})=\{\neg c\}$

$C_{13}=res(C_2,C_{12}\}=\{\neg d\}$

$C_{14}=res(C_{10},C_{13})=Ø$

Donc F est insatisfiable

Autre exemple de production de la clause vide

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

- Soit F= $\{C_1,C_2,C_3,C_4,C_5,C_6\}$ avec :

$C_1=\{a,d\}$

$C_2=\{c,\neg d\}$

$C_3=\{\neg a,e\}$

$C_4=\{a,e\}$

$C_5=\{\neg c,\neg e\}$

$C_6=\{\neg a,d\}$

$C_7=res(C_1,C_6)=\{d\}$

$C_8=res(C_2,C_5)=\{\neg d,\neg e\}$

$C_9=res(C_3,C_4)=\{e\}$

$C_{10}=res(C_7,C_8)=\{\neg e\}$

$C_{11}=res(C_9,C_{10})=\varnothing$

Optimisation

- 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 donc de noter **res(C,C')** la résolvante de deux clauses*

- Comme on a vu qu'une clause valide peut-être éliminer d'une forme clausale sans changer sa sémantique, on ne cherchera pas à produire de telles clauses par résolution

  *=> On limitera l'application de la règle de résolution à deux clauses ayant **un unique littéral opposé***

Propriété

- *Propriété : la règle de résolution produit une conséquence logique $\{C, C'\} \models Res(C,C')$*
- Idée de la preuve :
  1. C et C' ne peuvent pas être vide sinon on ne peut pas appliquer la règle de résolution
  2. Soit p le symbole de la résolution et C la clause ayant le littéral p : on a donc $res(C,C') = C\backslash\{p\} \vee C'\backslash\{\neg p\}$.
  3. Par cas montrons que pour tout I modèle de C et C', I est un modèle de res(C,C')
     - Si I(p)=1 alors val(¬p,I)=0 (sém. du ¬) ; comme val(C',I)=1 il faut (sém. du v) que val($C'\backslash\{\neg p\}$,I)=1 ; on a alors (sém. du ∨) val(res(C,C'),I)=1
     - Si I(p)=0 : comme val(C,I)=1 il faut (sém.du v) que val($C\backslash\{p\}$,I)=1 ; on a alors (sém. du ∨) val(res(C,C'),I)=1

Théorème de correction et complétude de la méthode de résolution

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

- Correction :
  - Lemme : *Si* F *contient deux clauses* C, C' *et si* F ∪ {res(C,C')} *est insatisfiable alors* F *est insatisfiable*
  - Preuve par récurrence sur la longueur de la démonstration (le nombre d'applications de la règle de résolution)

- Complétude :
  - 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 insatisfiable alors F[L] insatisfiable*
  - Preuve par induction sur le nombre n de symboles propositionnels de F

La méthode de résolution en pratique…

- Dès qu'une formule 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 complète sur les clauses de Horn contenant exactement un littéral positif
  - 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 $\{\{u\},\{v,\neg w\},\{\neg t,v\},\{t,\neg u,w\},\{\neg v\}\} \vdash_{res} \varnothing$?

- Résolution
  - On a la dérivation : $C_6=res(C_1,C_4)=\{t,w\}$, $C_7=res(C_3,C_6)=\{v,w\}$, $C_8=res(C_2,C_7)=\{v\}$, $C_9=res(C_5,C_8)=\varnothing$
  - Donc : $u,\ (w\Rightarrow u),\ (w\Rightarrow v),\ (t\Rightarrow v),\ (u\Rightarrow(w\vee t)) \models v$

Application (suite)

- Soit le problème :

$$u,(w\Rightarrow u),(w\Rightarrow w),(t\Rightarrow v) \models v \ \ ?$$

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 ? (supp. clauses valides ou ayant une incluse)

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

- 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 qui contiennent une clause incluse
- 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({},FC)*

Algorithme : résolutionLargeur

Données : 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)

Résultat : 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;
  | résolutionLargeur(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 produites à chaque étape qui ont une clause incluse.