Systèmes à base de règles

en logique des propositions

Cours de HMIN107 (IA)

ML Mugnier

1

Systèmes à base de règles

- Base de connaissances
  - Base de faits : observations factuelles sur une situation précise
  - Base de règles : connaissances générales sur un domaine d'application
- Forme générale d'une règle : SI <*condition*> ALORS <*conclusion*>
  « condition » appelée aussi « prémisses » ou « hypothèse »

> Eau $\wedge$ TempératureSup90 $\rightarrow$ EauBout
> (logique des propositions, ou logique d'ordre 0)
>
> Température > 90 $\wedge$ Liquide = « Eau » $\rightarrow$ Etat = « Ebullition »
> (logique d'ordre 0+)
>
> $\forall x\, \forall y\, \forall z$ (Température(x,y) $\wedge$ PtEbullition(x,z) $\wedge \geq$ (y,z) $\rightarrow$ Etat(x,Ebullition))
> (logique du premier ordre)

- En programmation logique (Prolog, Datalog, ...) on dit plutôt :
  <*tête*> SI <*corps*> Etat(x,Ebullition) :- Température(x,y), PtEbullition(x,z), y ≥ z

2

Règles en logique des propositions

- Rappel : littéral positif : atome (ici : symbole propositionnel)
  littéral négatif : négation d'un atome
- Règle conjonctive : <conjonction de littéraux> → <littéral>
  Ces règles correspondent aux clauses
- Règle (conjonctive) positive : <conjonction d'atomes> → <atome>
  Ces règles correspondent aux « clauses définies » :
  clauses avec exactement un littéral positif
- Fait = règle avec une condition vide (« toujours vraie »)
  → si on considère des règles conjonctives positives, un fait est un atome
- Base de connaissances : K = (BF,BR)
  - BF : ensemble de faits vu comme la conjonction des faits
  - BR : ensemble de règles vu comme la conjonction des règles

→ la base de connaissances est vue comme la conjonction des faits et des règles

3

Application des règles

- On voit souvent l'hypothèse d'une règle comme un ensemble
- Une règle R : H → C est applicable à BF si $H \subseteq BF$
- Cette application est utile si $C \notin BF$
- Appliquer R à BF consiste à ajouter C dans BF
- BF est saturée (par rapport à BR)
  si aucune application d'une règle de BR à BF n'est utile

BF = {A, B, C}

BR = { $R_1 : A \wedge B \rightarrow E$
$R_2 : C \wedge E \rightarrow D$ }

BF saturée $BF^* = BF \cup \{E, D\}$

4

Mécanismes principaux sur les règles

- Chaînage avant :
  - principe : appliquer les règles sur les faits pour produire de nouveaux faits
  - la base de faits est saturée si on ne peut plus produire de nouveaux faits

BF = {A, B, C}

BR = { R_1 : A ∧ B → E
R_2 : C ∧ E → D }

BF* = BF ∪ {E, D}

- Chaînage arrière :
  - principe : prouver un but (atome /littéral) en «remontant» le long des règles
  - le but initial est prouvé lorsqu'on arrive à une liste de buts vide

	liste de buts
But initial : D ?	{D}
Avec R_2 :	{C, E}
Avec le fait C (vu comme → C) :	{E}
Avec R_1 :	{A,B}
Avec le fait A	{B}
Avec le fait B	{}

Exemple (réunion d'amis)

BF = { Benoît, Cloé }
BR = {R_1 ... R_8}

R_1 : Benoît ∧ Djamel ∧ Emma → Félix
R_2 : Gaëlle ∧ Djamel → Amandine
R_3 : Cloé ∧ Félix → Amandine
R_4 : Benoît → Xéna
R_5 : Xéna ∧ Amandine → Habiba
R_6 : Cloé → Djamel
R_7 : Xéna ∧ Cloé → Amandine
R_8 : Xéna ∧ Benoît → Djamel

Remarque : une règle s'applique (de façon utile) au plus une fois

BF* = ?

Algorithme de chaînage avant (version naïve)

Algorithme ForwardChaining (K) *// Données : K = (BF, BR)*
Début *// Résultat : BF saturée par BR*

```
Fin ← faux
Pour toute règle R ∈ BR
    appliquée(R) ← faux
Tant que non fin
    nouvFaits ← ∅      // ensemble des nouveaux faits obtenus à cette étape
    Pour toute règle R : H → C ∈ BR telle que appliquée(R) = faux
        Si H ⊆ BF      // R est applicable
            appliquée(R) ← vrai                 // que l'application soit utile ou pas
            Si C ∉ (BF ∪ nouvFaits)             // l'application de R est utile
                Ajouter C à nouvFaits
    Si nouvFaits = ∅
        Fin ← vrai
    Sinon Ajouter les éléments de nouvFaits à BF
Fin
```

FC a t-il une complexité polynomiale ?
Borner sa complexité

7

Algorithme de chaînage avant (avec compteurs)

Algorithme ForwardChaining2 (K) *// Données : K = (BF, BR)*
Début *// Résultat : BF saturée par BR*

```
    àTraiter ← BF
    Pour toute règle R ∈ BR
        compteur(R)← |hypothèse(R)|    // Nombre de littéraux de l'hypothèse de R
    Tant que àTraiter ≠ ∅
        Retirer F de àTraiter
        Pour toute règle R : H → C ∈ BR telle que F ∈ H              (1)
            Décrémenter compteur(R)
            Si compteur(R) = 0          // R est applicable
                Si C ∉ (BF ∪ àTraiter)  // l'application de R est utile  (2)
                    Ajouter C à àTraiter
                    Ajouter C à BF
Fin
```

Dérouler FC2 sur la base « réunion d'amis »

Que peut-il se passer si on ne fait pas le test C ∉ (BF ∪ àTraiter) en (2) ?

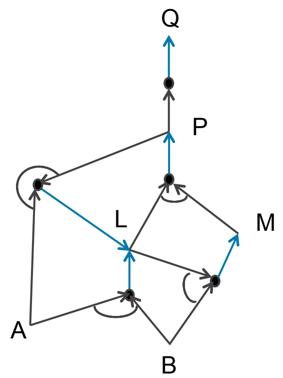
Quelle est la complexité de FC2 si : en (1) accès direct à l'ensemble des règles avec F ∈ H
en (2) test C ∉ (BF ∪ àTraiter) en temps constant ?

8

Graphe ET-OU

BF = {A, B}

1. P → Q
2. L ∧ M → P
3. B ∧ L → M
4. A ∧ P → L
5. A ∧ B → L



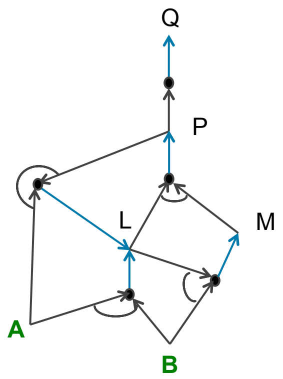
- 2 sortes de sommets :
  - atomes / littéraux (sommets « OU »)
  - règles (sommets « ET »)
- arcs pour chaque règle :
  - sommets symboles de l'hypothèse → sommet règle
  - sommet règle → sommet symbole de la conclusion

9

Chaînage avant sur le graphe ET-OU

BF = {A, B}

1. P → Q
2. L ∧ M → P
3. B ∧ L → M
4. A ∧ P → L
5. A ∧ B → L



Parcours du graphe ET-OU :
- Marquer les faits
- Un sommet règle peut être franchi quand tous ses prédécesseurs sont marqués
  On marque alors son successeur

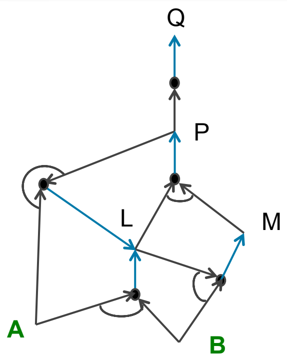
→ Base de faits saturée = ensemble des littéraux (des sommets) marqués

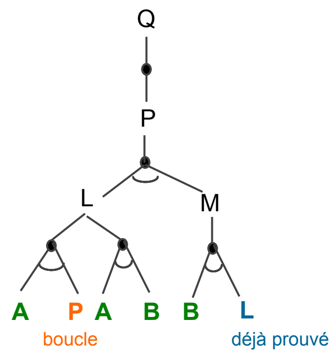
10

Chaînage arrière sur le graphe ET-OU

BF = {A, B}

1. P → Q
2. L ∧ M → P
3. B ∧ L → M
4. A ∧ P → L
5. A ∧ B → L





Partant d'un but, on construit un arbre en remontant le long des arcs
- Un fait est considéré comme *prouvé*
- Si tous les fils d'un noeud ET sont prouvés, alors le père du noeud ET est *prouvé*

Eviter les boucles : vérifier si le nouveau sous-but est déjà sur le chemin depuis la racine
Eviter de refaire le même travail : vérifier si le nouveau sous-but a déjà été prouvé ou si on a déjà échoué à le prouver

11

Algorithme de chaînage arrière (version récursive naïve)

Ici, les faits sont vus comme des règles à hypothèse vide

Algorithme BackwardChaining(K,Q)
// Données : K = (BF,BR) et Q une liste de littéraux
// Résultat (quand l'algorithme s'arrête) : vrai si Q prouvable, faux si Q non prouvable
Début
Si Q = ∅, retourner vrai
Soit C = premier(Q) // premier(Q) : *premier littéral de Q*
Pour toute règle R = H1 ∧ … ∧ Hn → C de BR **et BF**
  Q' ⟵ Concaténer [H1, … , Hn] et reste(Q) // reste(Q) : *Q privé de son 1er littéral*
  Si BC(K,Q')
    retourner vrai
Retourner faux
Fin

Problème : cet algorithme peut boucler

12

Algorithme de chaînage arrière (version alternative)

Algorithme BC2(K,Q) *// ici Q est un littéral (et pas une liste de littéraux)*

Début

Si Q ∈ **BF**, retourner vrai

Pour toute règle R = H1 ∧ … ∧ Hn → Q de **BR**

  i ⟵ 1
  Tant que i ≤ n et BC2(K,Hi)
    incrémenter i
  Si i > n, retourner vrai *// toute l'hypothèse de R prouvée*

Retourner faux

Fin

Cet algorithme légèrement différent peut lui aussi boucler

13

Algorithme de chaînage arrière (qui s'arrête)

Algorithme BC3(K,Q,L) *// Données : K = (BF,BR), Q un atome*
// L un ensemble d'atomes (à ne pas générer)
// Résultat : vrai ssi Q est prouvable

Appel BC3(K,Q,ø)

Début

Si Q ∈ BF, retourner vrai

Pour toute règle R = H1 ∧ … ∧ Hn → Q de BR

  Si aucun des H1 ... Hn n'appartient à L *// sinon on va boucler*
    i ← 1
    Tant Que i ≤ n et BC3(K, Hi, L ∪ {Q})
      incrémenter i
    Si i > n, retourner vrai *// hypothèse de R prouvée, donc Q aussi*

Retourner faux *// aucun des faits et aucune des règles ne permet de prouver Q*

Fin

14

Adéquation et complétude

- Un mécanisme de chaînage avant / arrière est
  - **adéquat** (ou **correct**) : s'il ne fait **que** des déductions
    Pour tout littéral A,
    si A ∈ BF* alors BF, BR ⊨ A (pour le chaînage avant)
    si A est prouvé alors BF, BR ⊨ A (pour le chaînage arrière)
  - **complet** : s'il fait **toutes** les déductions
    Pour tout littéral A,
    si BF, BR ⊨ A alors A ∈ BF* (pour le chaînage avant)
    si BF, BR ⊨ A alors A est prouvé (pour le chaînage arrière)
- BR = {règles conjonctives positives} : adéquation et complétude
- BR = {règles conjonctives (pas forcément positives)} : adéquation mais pas complétude

BF = ∅
BR = { A → B
¬A → B }

BF, BR ⊨ B

BF* = ? Prouver B ?

Adéquation du chaînage avant

Pour les règles conjonctives

Repose sur le modus ponens
Si on a H et (H → C) alors on a C

Ceci est logiquement correct car : H, H → C ⊨ C

- Chaque application de règle correspond à une application du modus ponens

- Adéquation du chaînage avant :
  Pour tout littéral A, si A ∈ BF* alors BF, BR ⊨ A
  Autrement dit : BF, BR ⊨ BF*

Peut se prouver par récurrence
sur le nombre d'applications de règles conduisant à BF*

Complétude du chaînage avant

Pour les règles conjonctives **positives**

- Complétude du chaînage avant :

  Pour tout atome A, si BF, BR ⊨ A alors A ∈ BF*

Preuve

Supposons que BF, BR ⊨ A
Ceci signifie que A est vrai dans **tout** modèle de BF et de BR

On va construire I un modèle particulier de BF et BR (qui rend donc A vrai)

- Soit l'interprétation I telle que : pour tout symbole s, I(s) = vrai ssi s ∈ BF*
- I est un modèle de BF car BF ⊆ BF*
- I est un modèle de BR :
  Si I n'est pas un modèle d'une règle R : H → C,
  c'est que I rend H vrai mais C faux
  donc R serait applicable mais non appliquée
  Ceci contredit le fait que BF* est la base de faits *saturée*

Puisque I est un modèle de BF et BR, on a I(A) = vrai par hypothèse

Or, par définition de I, I(A) = vrai ssi A ∈ BF*. Donc A ∈ BF*

17

Complexité du raisonnement en logique d'ordre 0

Problème « inférence d'un atome » :
Données : une formule $\mathcal{F}$, un atome A
Question : a-t-on $\mathcal{F} \models$ A ?

- Si $\mathcal{F}$ est une formule propositionnelle quelconque,
  ce problème est co-NP-complet ($\mathcal{F} \not\models$ A est NP-complet)

  Comment le résoudre ?
  Par exemple avec DPLL : $\mathcal{F} \models$ A ssi ($\mathcal{F} \wedge \neg$ A) est insatisfiable

- Si $\mathcal{F}$ est une base de connaissances avec des règles conjonctives (pas forcément positives), on n'a rien gagné

  car toute CNF (conjonction de clauses) peut se réécrire sous cette forme (avec éventuellement une BF vide)

- Si $\mathcal{F}$ est une base de connaissances avec des règles conjonctives positives :

  le chaînage avant (ou arrière) est adéquat et complet

  le problème devient polynomial, et même linéaire en la taille de $\mathcal{F}$

18