Systèmes à base de règles en logique des propositions

Mécanismes fondamentaux

ML Mugnier

Contexte : systèmes à base de connaissances

Base de connaissances ⟷ Services de raisonnement

- **Connaissances générales** du domaine d'application

  ex : diagnostic médical
  ex : fonctionnement système complexe

- **Connaissances factuelles (« faits »)** (observations, situations particulières, descriptions d'objets précis, ...)

  ex : symptômes d'un patient précis
  ex : données fournies par des capteurs

- **Déductif** :

Quelles conclusions peut-on tirer de la base de connaissances ?

⇒ services de haut niveau :

ex: faire le diagnostic médical d'un patient

ex: déterminer s'il y a des anomalies dans le système

Les connaissances sont exprimées dans un langage *basé sur* la logique et les raisonnements correspondent à des inférences logiques

Systèmes à base de règles

- Base de connaissances = (Base de faits, Base de règles)
- Forme générale d'une **règle** : SI *<condition>* ALORS *<conclusion>*
  « condition » appelée aussi « prémisses » ou « hypothèse »

Exemples (règles « certaines » : pas de probabilités)

$\text{VoitureSport} \wedge \text{JeuneConducteur} \rightarrow \text{TarifAssuranceElevé}$
(logique des propositions, ou logique d'ordre 0)

$\text{VoitureSport} = \text{vrai} \wedge \text{Age} < 20 \rightarrow \text{TarifAssurance} = 4$
(logique d'ordre 0+)

$\forall x.\ \text{VoitureSport}(x) \rightarrow \text{VehiculeARisque}(x)$
$\forall x.\ \forall y.\ \text{AgeDe}(x,y) \wedge y < 20 \rightarrow \text{ProfilArisque}(x)$
$\forall x.\ \forall y.\ \text{VehiculeARisque}(x) \wedge \text{ConducteurDe}(x,y) \wedge \text{ProfilARisque}(y) \rightarrow \text{TarifAssurance}(y,x,4)$
(logique du premier ordre)

- En programmation logique (Prolog, Datalog, ASP...) on dit *<tête>* SI *<corps>*
  TarifAssurance(y,x,4) **:-** VehiculeARisque(x), ConducteurDe(x,y), ProfilARisque(y)

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 : disjonction de littéraux

$$H \rightarrow C$$
$$\equiv \neg H \vee C$$

- 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, un fait est un littéral

- 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 formule logique associée à K est la conjonction des faits et des règles

Exemple de base de connaissances (réunion d'amis)

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

R_1 : Benoît $\wedge$ Djamel $\wedge$ Emma $\rightarrow$ Félix
R_2 : Gaëlle $\wedge$ Djamel $\rightarrow$ Amandine
R_3 : Cloé $\wedge$ Félix $\rightarrow$ Amandine
R_4 : Benoît $\rightarrow$ Xéna
R_5 : Xéna $\wedge$ Amandine $\rightarrow$ Habiba
R_6 : Cloé $\rightarrow$ Djamel
R_7 : Xéna $\wedge$ Cloé $\rightarrow$ Amandine
R_8 : Xéna $\wedge$ Benoît $\rightarrow$ Djamel

D'un point de vue logique, la base de connaissances correspond à la formule :
Benoît $\wedge$ Cloé $\wedge$ R_1 $\wedge$ ... $\wedge$ R_8

Application des règles (positives)

- Une règle R : H → C est applicable à BF si H ⊆ BF

  Ici on voit H *comme un ensemble d'atomes*

- Cette application est utile si C ∉ 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 ∪ {E, D}

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

$BF^* = BF \cup \{E, D\}$

- 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 chaînage avant (réunion d'amis)

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

R_1 : Benoît $\wedge$ Djamel $\wedge$ Emma $\rightarrow$ Félix
R_2 : Gaëlle $\wedge$ Djamel $\rightarrow$ Amandine
R_3 : Cloé $\wedge$ Félix $\rightarrow$ Amandine
R_4 : Benoît $\rightarrow$ Xéna
R_5 : Xéna $\wedge$ Amandine $\rightarrow$ Habiba
R_6 : Cloé $\rightarrow$ Djamel
R_7 : Xéna $\wedge$ Cloé $\rightarrow$ Amandine
R_8 : Xéna $\wedge$ Benoît $\rightarrow$ Djamel

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

BF* = ?

{ B, C, X, D, A, H }

Algorithme de chaînage avant (version naïve)

```
Algorithme ForwardChaining (K)        // Données : K = (BF, BR)
Début                                 // Résultat : BF* = BF saturée par BR
Fin ← faux
BF* ← BF
Pour toute règle R ∈ BR
    appliquée(R) ← faux     // Remarque : une règle s'applique au plus une fois
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*
Retourner BF*
Fin
```

|BR| +
|BR| x taille(BR) x (|BF| + |BR|)

O(taille(K)^3)

FC a t-il une complexité polynomiale ?
Borner « grossièrement » sa complexité

Algorithme de chaînage avant (avec compteurs)

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

```
    àTraiter ← BF
    BF* ← 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*                 // l'application de R est utile       (2)
                    Ajouter C à àTraiter
                    Ajouter C à BF*

    Retourner BF*
Fin
```

Exemple : dérouler FC2 sur la base « réunion d'amis »

FC2 sur l'exemple « réunion d'amis »

```
àTraiter ← BF
BF* ← BF
Pour toute règle R ∈ BR
    compteur(R) ← |hypothèse(R)|
Tant que àTraiter ≠ ∅
    Retirer F de àTraiter
    Pour toute règle (R : H → C) ∈ BR telle que F ∈ H
        Décrémenter compteur(R)
        Si compteur(R) = 0
            Si C ∉ BF*
                Ajouter C à àTraiter
                Ajouter C à BF*
Retourner BF*
```

BF = { Benoît, Cloé }
BR = $\{R_1 \dots R_8\}$

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

Algorithme de chaînage avant (avec compteurs)

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

àTraiter ← BF
BF* ← 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* *// l'application de R est utile* (2)
Ajouter C à àTraiter
Ajouter C à BF*

Retourner BF*
Fin

Quelle est la complexité de FC2 si :

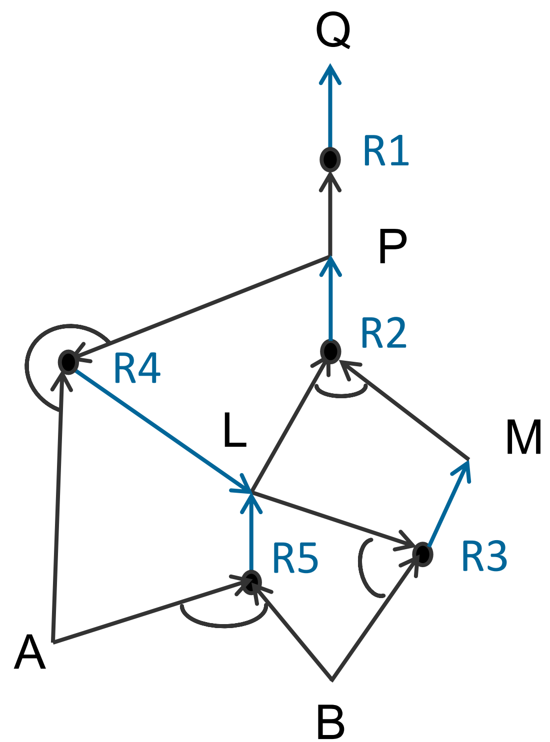
- en (1) accès **direct** à l'ensemble des règles telles que **F ∈ H**
- en (2) test C ∉ BF* **en temps constant** ?

Si chaque règle H → C est accédée au plus |H| fois
FC2 peut s'implémenter en *O*(K) : voir TD

Graphe « ET-OU » associé à la base de règles

BF = {A, B}

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



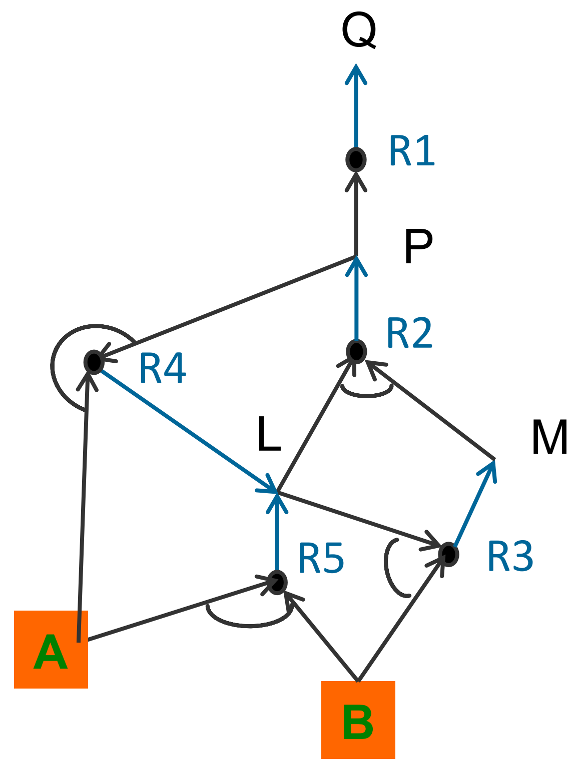
Sommets ET (règles)
« Pour utiliser R2,
il faut L **et** M »

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

Chaînage avant sur le graphe ET-OU

BF = {A, B}

1. $P \rightarrow Q$
2. $L \wedge M \rightarrow P$
3. $B \wedge L \rightarrow M$
4. $A \wedge P \rightarrow L$
5. $A \wedge B \rightarrow L$



Sommets ET (règles)
« Pour utiliser R2,
il faut L **et** M »

Descente dans le graphe ET-OU :

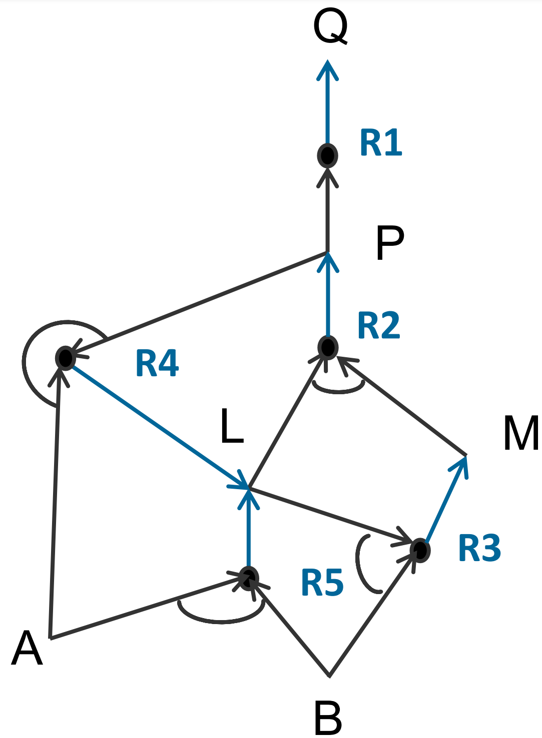
- Marquer les faits
- Un sommet règle peut être franchi quand tous ses prédécesseurs sont marqués
  On marque alors son successeur

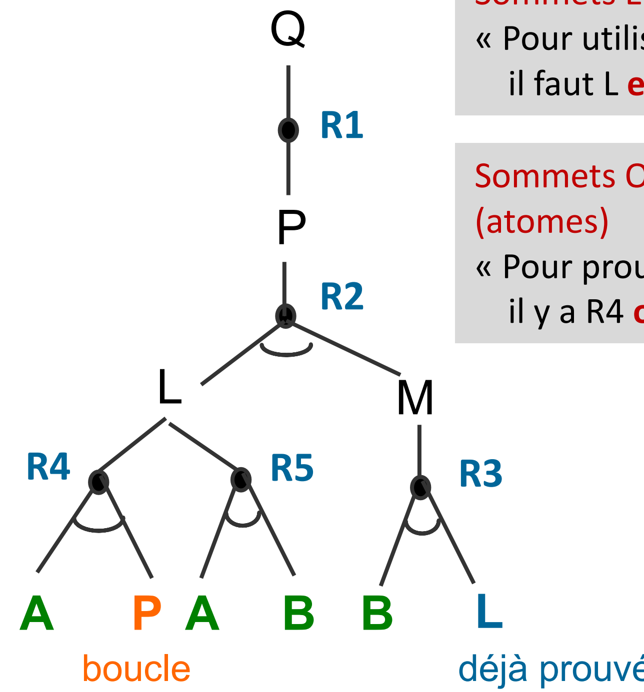
→ Base de faits **saturée** = ensemble des **sommets marqués**

Chaînage arrière sur le graphe ET-OU

BF = {A, B}

1. $P \rightarrow Q$
2. $L \wedge M \rightarrow P$
3. $B \wedge L \rightarrow M$
4. $A \wedge P \rightarrow L$
5. $A \wedge B \rightarrow L$





Sommets ET (règles)
« Pour utiliser R2, il faut L **et** M »

Sommets OU (atomes)
« Pour prouver L, il y a R4 **ou** R5»

Partant d'un but à prouver, 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 de ce noeud 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**

Algorithme de chaînage arrière (version 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

BC sur l'exemple

Algorithme BC(K,Q)

Début

Si $Q = \emptyset$, retourner vrai

Soit C = premier(Q)

Pour toute règle $R = H1 \wedge \ldots \wedge Hn \rightarrow C$ de BR **et BF**

Q' ⟵ Concaténer [H1, … , Hn] et reste(Q)

Si BC(K,Q')

retourner vrai

Retourner faux

Fin

BF = {A, B}

1. $P \rightarrow Q$
2. $L \wedge M \rightarrow P$
3. $B \wedge L \rightarrow M$
4. $A \wedge P \rightarrow L$
5. $A \wedge B \rightarrow L$

Q
P
L M
A P M (par R_4)
P M
L M
……….

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, donc Q aussi*

Retourner faux

Fin

Cet algorithme légèrement différent peut lui aussi boucler

BC2 sur l'example

Algorithme **BC2(K,Q)** *// ici Q est un littéral*

Début

Si Q ∈ **BF**, retourner vrai

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

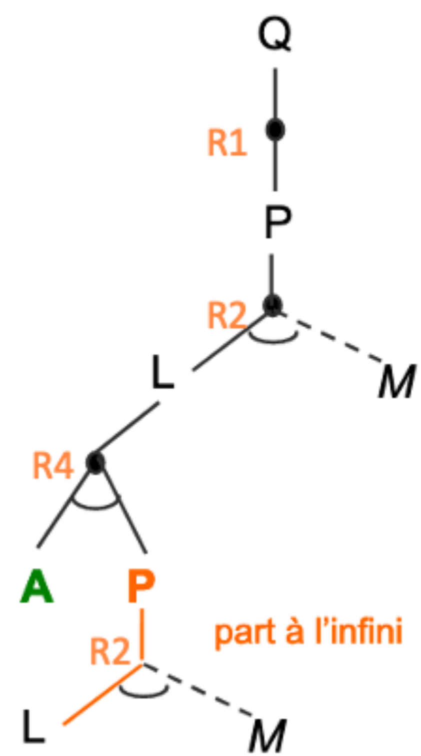
    i ⟵ 1
    Tant que i ≤ n et BC2(K,Hi)
        incrémenter i
    Si i > n, retourner vrai

Retourner faux

Fin

BF = {A, B}

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



Algorithme de chaînage arrière (qui s'arrête)

Algorithme BC3(K,Q,Lc) *// Données : K = (BF,BR), Q un atome*
*// **Lc** un ensemble d'atomes* **(chemin de la racine à Q)**
// 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 Lc *// sinon on va boucler*

i ← 1
Tant que i ≤ n et BC3(K, Hi, Lc ∪ {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*

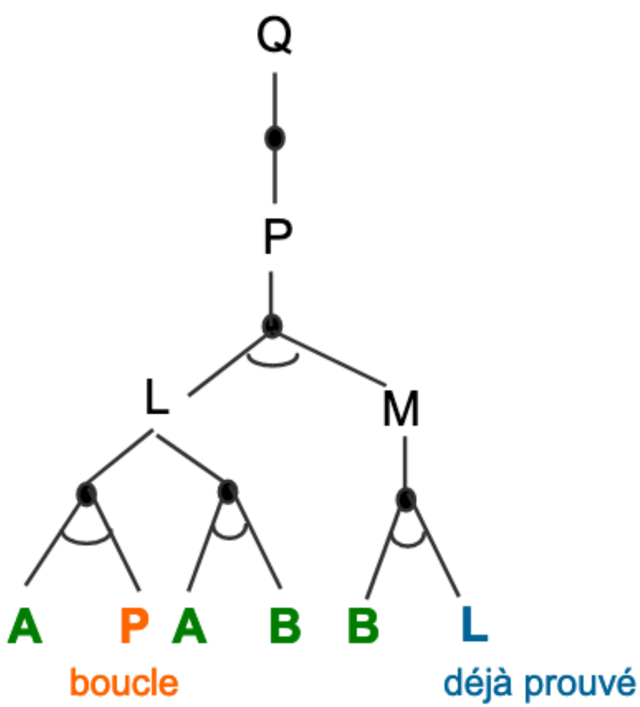
Fin

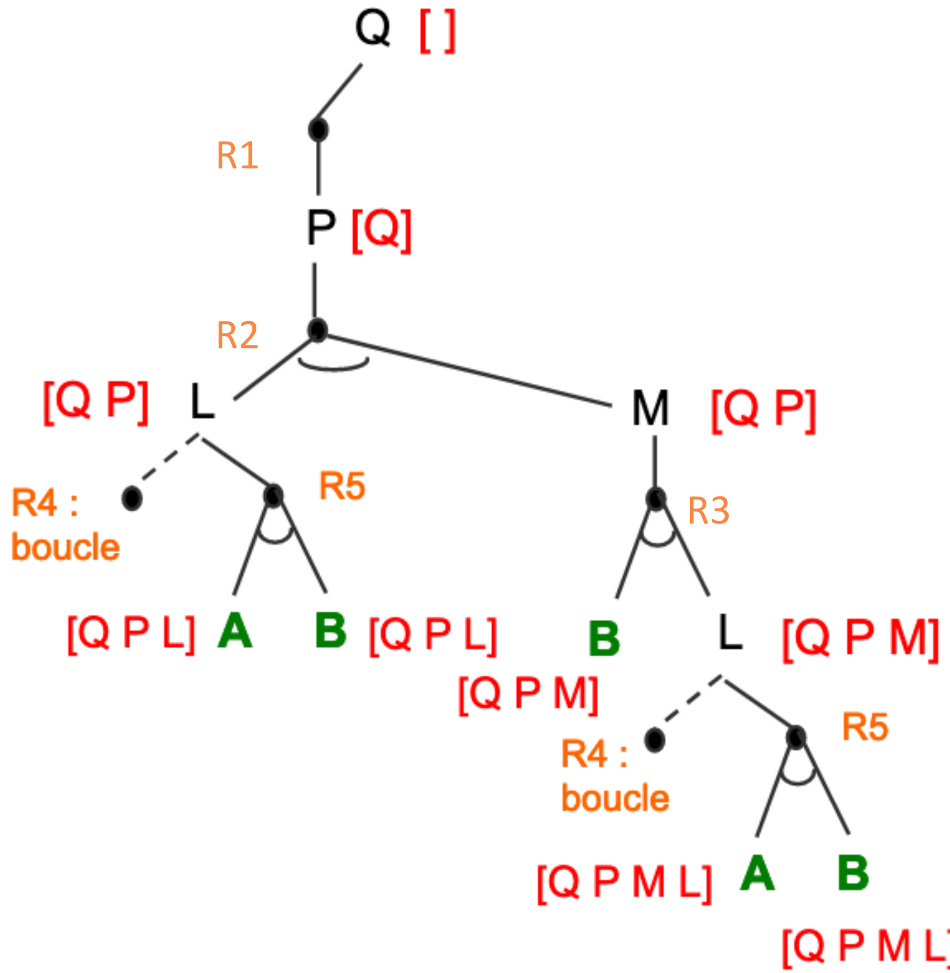
(Reste à « éviter de refaire le même travail »)

BC3 sur l’exemple

BF = {A, B}

1. $P \rightarrow Q$
2. $L \wedge M \rightarrow P$
3. $B \wedge L \rightarrow M$
4. $A \wedge P \rightarrow L$
5. $A \wedge B \rightarrow L$





Déjà prouvé ?

Synthèse

- **Règles conjonctives en logique des propositions**
  - *conjonction d'atomes → atome* (cas positif)
  - *conjonction de littéraux → littéral*
- **Base de connaissances** K = (BF, BR)

- **Chaînage avant** : dirigé par les faits

  applique les règles sur les faits jusqu'à obtention d'un point fixe
  => base de faits saturée : BF*

- **Chaînage arrière** : dirigé par un but

  cherche à prouver ce but en « remontant » les règles jusqu'aux faits

> Pour tout littéral L,
> **L s'obtient en chaînage avant** (L ∈ BF*)
> **si et seulement si L est prouvable en chaînage arrière**