Systèmes Experts

Boi Faltings

Laboratoire d'Intelligence Artificielle
boi.faltings@epfl.ch
http://moodle.epfl.ch/

Qu'est-ce qu'un système expert?

- Un système expert est un programme qui reproduit le raisonnement d'un expert.
- Expert = personne ayant des connaissances profondes d'un domaine particulier et étroit, pas un génie universel!
- Exemples:
  - interprétation de radiographies.
  - recommandation d'antibiotiques.
  - diagnostic de pannes.
- Défi: maîtriser la complexité de calcul.

General Problem Solver (GPS)

- Origine: A. Newell & H.Simon proposent une nouvelle méthodologie pour la *psychologie*: construire des programmes qui modélisent le comportement humain.
- $\Rightarrow$ un programme: Résolveur Général de Problèmes (General Problem Solver = GPS) est proposé comme théorie de la manière humaine de résoudre des problèmes.
- Le programme GPS, entre autre, a valu un prix Nobel à H. Simon.

L'analyse moyens-buts

Un domaine d'experts se caractérise pour sa grande complexité:

- il y a souvent un très grand nombre d'opérations possibles, ce qui rend une recherche très inefficace.
- une heuristique importante pour limiter la recherche est *l'analyse moyens-buts*: seulement les règles qui contribuent effectivement à la résolution du *but* du système sont utilisées.
- l'analyse moyens-buts a été inventé par Newell & Simon lors de leurs travaux sur GPS (General Problem Solver).
- elle s'est traduite par l'utilisation du chaînage arrière au lieu du chaînage avant.

Systèmes experts

Formulation de connaissances par règles
Inférence par modus ponens:

$$\mathtt{F}_1 \overset{\mathtt{R}_1}{\Rightarrow} \mathtt{F}_2 \overset{\mathtt{R}_2}{\Rightarrow} \ldots \overset{\mathtt{R}_n}{\Rightarrow} \mathtt{Solution}$$

mais en chaînage *arrière*:

- l'inférence commence par le(s) but(s).
- le système maintient un agenda de *buts*, et applique des règles dont les conclusions s'unifient avec un but.
- chaque application d'une règle réduit un but à un ou plusieurs (sous-)buts.
- un but peut aussi s'unifier avec une prémisse de la base de données.
- l'inférence s'arrête dès qu'il n'y a plus de buts à satisfaire.

GPS et la technologie des systèmes experts

L'inférence en chaînage arrière est similaire à GPS:

- formulation du problème par des règles. Les règles "logiques" transforment un état de connaissance en un autre.
- chaînage arrière $\Leftrightarrow$ génération de sous-buts
- le chaînage arrière est une forme simplifiée de l'analyse moyens-buts. Certains systèmes utilisent aussi des heuristiques qui suggèrent l'utilisation de certains règles dans certains contextes.

$\Rightarrow$ un formalisme logique bien fondé qui reprend les idées principales du modèle psychologique de GPS.
théorie psychologique $\Rightarrow$ paradigme informatique!

Environnements

Le même résultat peut souvent être obtenu par plusieurs chemins:

- `<-100-francs` $\vdash$ `hors-taxe`
- `vin` $\wedge$ `<-2-litres` $\vdash$ `hors-taxe`

En chaînage avant, ceci ne pose pas de problème: chaque inférence est valable en soi.
En chaînage arrière, il faut distinguer les environnements, car seul les sous-buts d'un même environnement sont obligatoires pour la solution:

montrer `<-100-Francs` **ou** `vin` $\wedge$ `<-2-litres`

Exemple

Premisses:

R1: père(?x,?y) $\wedge$ frère(?y,?z) $\Rightarrow$ père(?x,?z)

R2: père(?x,?y) $\wedge$ frère(?x,?z) $\Rightarrow$ oncle(?z,?y)

F1: père(Jacques,Charles),

F2: frère(Charles,Francois),

F3: frère(Jacques,Pierre)

But:

oncle(?x,Francois)

Solution en chaînage arrière

But, premisses $\rightarrow$ *ensembles* de *sous-buts*:
But, R2 $\rightarrow$ {SB1 = `père(?y,Francois)`,SB2 = `frère(?y,?x)`}

Solution en chaînage arrière

But, premisses → *ensembles* de *sous-buts*:
But, R2 → {SB1 = `père(?y,Francois)`,SB2 = `frère(?y,?x)`}

SB1, R1 → {SB3 = `frère(?w,Francois)`,SB4 = `père(?y,?w)`}

SB3 → {`?w=Charles`}
SB4 → {`?y=Jacques,?w=Charles`}

Solution en chaînage arrière

But, premisses → *ensembles* de *sous-buts*:
But, R2 → {SB1 = `père(?y,Francois)`,SB2 = `frère(?y,?x)`}

SB1, R1 → {SB3 = `frère(?w,Francois)`,SB4 = `père(?y,?w)`}

SB3 → {`?w=Charles`}
SB4 → {`?y=Jacques,?w=Charles`}

SB2 → E1: {`?y=Charles,?x=Francois`} ,
E2: {`?y=Jacques,?x=Pierre`}

Solution en chaînage arrière

But, premisses → *ensembles* de *sous-buts*:
But, R2 → {SB1 = `père(?y,Francois)`,SB2 = `frère(?y,?x)`}

SB1, R1 → {SB3 = `frère(?w,Francois)`,SB4 = `père(?y,?w)`}

SB3 → {`?w=Charles`}
SB4 → {`?y=Jacques,?w=Charles`}

SB2 → E1: {`?y=Charles,?x=Francois`} ,
E2: {`?y=Jacques,?x=Pierre`}

Résultat = deux *environnements*:
E1:
{`(?y=Charles,?x=Francois)`∧`(?w=Charles)`∧`(?y=Jacques,?w=Charles)`}

Solution en chaînage arrière

But, premisses → *ensembles* de *sous-buts*:
But, R2 → {SB1 = `père(?y,Francois)`,SB2 = `frère(?y,?x)`}

SB1, R1 → {SB3 = `frère(?w,Francois)`,SB4 = `père(?y,?w)`}

SB3 → {`?w=Charles`}
SB4 → {`?y=Jacques,?w=Charles`}

SB2 → E1: {`?y=Charles,?x=Francois`} ,
E2: {`?y=Jacques,?x=Pierre`}

Résultat = deux *environnements*:
E1:
{`(?y=Charles,?x=Francois)`∧`(?w=Charles)`∧`(?y=Jacques,?w=Charles)`}
E2:
{`(?y=Jacques,?x=Pierre)`∧`(?w=Charles)`∧`(?y=Jacques,?w=Charles)`}

Solution en chaînage arrière

But, premisses → *ensembles* de *sous-buts*:
But, R2 → {SB1 = père(?y,Francois),SB2 = frère(?y,?x)}

SB1, R1 → {SB3 = frère(?w,Francois),SB4 = père(?y,?w)}

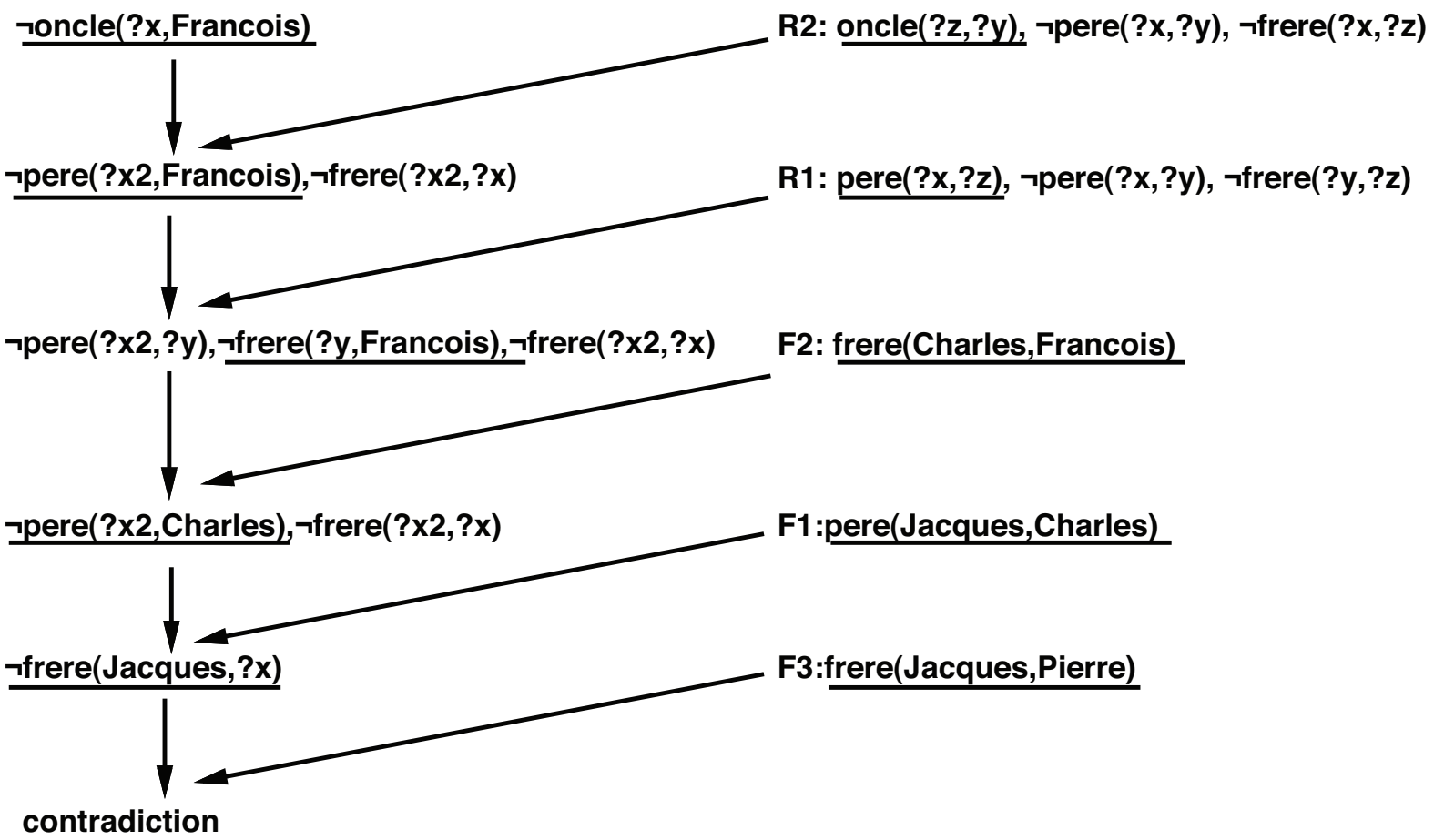
SB3 → {?w=Charles}
SB4 → {?y=Jacques,?w=Charles}

SB2 → E1: {?y=Charles,?x=Francois} ,
E2: {?y=Jacques,?x=Pierre}

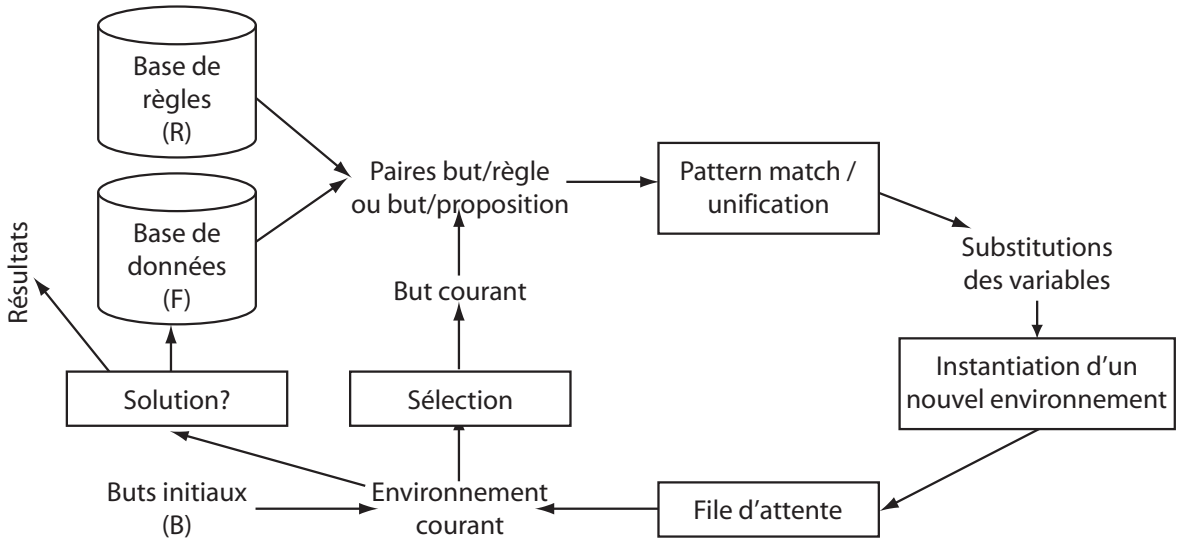
Résultat = deux *environnements*:
E1:
{(?y=Charles,?x=Francois)∧(?w=Charles)∧(?y=Jacques,?w=Charles)}
E2:
{(?y=Jacques,?x=Pierre)∧(?w=Charles)∧(?y=Jacques,?w=Charles)}

E1 est contradictoire pour ?y, donc ne peut être satisfait ⇒ ?x=Pierre

Chaînage arrière = preuve par contradiction



Architecture: chaînage arrière



Les environnements sont liés par une trace qui permet de retrouver le cheminement de la preuve.
Le chaînage arrière est plus difficile à implémenter car il faut maintenir des liens entre buts et environnements distincts!

Algorithme (chaînage arrière)

```
Procédure Chaînage-Arriére (R,F,B)
envs ← ({ buts-initiaux }) = (B)
repeat
  e ← first(envs), envs ← rest(envs)
  if tous les buts de e sont résolus then
   return (buts(e),unificateurs(e))
  for tout but non résolu b ∈ e do
   for toute proposition p ∈ F do
    U ← UNIFIER(p,b)
    if U ≠ ECHEC then
     envs ← append(envs, (e \ b) ∪ U)
   for toute règle r ∈ R do
    U ← UNIFIER(droit(r),b)
    if U ≠ ECHEC then
     nb ← instantiate(gauche(r),U)
     envs ← append(envs, (e \ b) ∪ nb ∪ U)
until envs = ()
```

Problèmes des variables

Les buts contiennent des variables $\Rightarrow$ 2 problèmes:

- Noms uniques:
  la variable `?x` dans deux règles n'est pas la même
  $\Rightarrow$ il faut générer des noms uniques
- Substitutions circulaires et leur mise à jour:
  `?x` ne peut pas s'unifier avec `f(?x)`
  $\Rightarrow$ l'unificateur doit en tenir compte

Questions à l'utilisateur

- L'utilisateur ne veut pas rentrer toutes les prémisses dès le départ.
- Exemple:
  un malade ne subira que les tests nécessaires.
  $\Rightarrow$ certaines règles posent des *questions*:
  `ask(pression sanguine = ?x)` $\Rightarrow$ `pression-sang(?x)`
- Chaque mesure a un coût $\Rightarrow$ des algorithmes heuristiques (A^*) peuvent minimiser le coût des mesures nécessaires pour trouver une conclusion.

Combinaison avant/arrière

Incompatibilité principale: le chaînage arrière construit des environnements distincts.
$\Rightarrow$ on ne peut pas intégrer le chaînage arrière dans un algorithme à chaînage avant, mais l'inverse est possible:

- classer toutes les règles en mode avant et arrière.
- appliquer les règles avant aux faits initiaux.
- commencer la recherche en mode chaînage arrière.
- appliquer le moteur à chaînage avant à chaque environnement généré en chaînage arrière, à condition que des nouveaux buts aient été satisfaits depuis la dernière application.

Critères de choix

En général, le chaînage arrière est plus efficace pour des problèmes bien ciblés tel que la planification ou le diagnostic. Par contre, le chaînage avant est plus adapté aux problèmes d'interprétation des données:

- si la règle peut produire beaucoup de résultats, par exemple dans un diagnostic, le chainage arrière est préférable.
- si la règle s'applique à beaucoup de données différentes, le chaînage avant est preferable.

Exemple: règle à chaînage avant

Détection de contours dans un image:

`ligne(?x,?y)` $\Rightarrow$ `contour(?x,?y)`

`contour(?x,?y)`$\wedge$`ligne(?y,?z)` $\Rightarrow$ `contour(?x,?z)`

- Utilisation en chaînage arrière: chaque point ?y est candidat
- 10'000 points $\Rightarrow$ 10'000 environnements!
- Utilisation en chaînage avant: seul les points liés sont considérés.

Exemple: règle à chaînage arrière

`le-plus-grand(?x)` $\Rightarrow$ `plus-grand(?x,?y)`

- Utilisation en chaînage avant: ?x est plus grand que tout ?y, donc il y a une infinitude de résultats!
- Utilisation en chaînage arrière: en général, ?y fait partie du but ou est retenu comme variable $\rightarrow$ inférence bien ciblée.

L'explication du raisonnement

Un grand atout pratique des systèmes experts est leur capacité de donner les raisons pour:

- les conclusions: pourquoi est-ce qu'on me recommande de faire cette opération risquée et coûteuse?
- les questions: pourquoi est-ce qu'il faut faire cette analyse compliquée?

Ces explications sortent de manière directe du raisonnement!

Les explications du résultat

- Toute conclusion donnée par le système est obtenue par une réduction successive du but à des questions posées à l'utilisateur.
- On peut donc expliquer *pourquoi* ce résultat a été trouvé par la trace des règles appliquées, par exemple:
  "La solution est `batterie-vide` *parce que:*
  - `phares-éteints`, *et*
  - `interrupteur-enclenché`

  et la règle:
  `phares-éteints` $\wedge$ `interrupteur-enclenché` $\Rightarrow$ `batterie-vide`"
- L'explication peut être continuée récursivement suivant la trace des règles appliquées.

Les explications des questions

- Chaque question est motivée par un sous-but, qui est lui-même motivé par un autre sous-but.
- Explication d'une question: le sous-but visé, et la règle et le but qui l'ont motivé.
- Par exemple:

  "Est-ce que le moteur démarre?"

  Pourquoi $\Rightarrow$

  "Ceci permettra de savoir si `moteur-ne-demarre-pas` *est vrai, et avec la règle*

  `moteur-ne-demarre-pas` $\Rightarrow$ `batterie-vide`

  je pourrais savoir si `batterie-vide` *est vrai."*

- L'explication peut être poursuivie récursivement.

La négation

- Considérez une condition négative:
  $\texttt{petite-quantite} \wedge \neg\, \texttt{frontalier} \Rightarrow \texttt{hors-taxe}$
- Quand est-ce qu'on peut déclencher cette règle?
- *Negation as failure*: Si le moteur d'inférence n'arrive pas à prouver $\texttt{p}$, alors supposer que $\neg\, \texttt{p}$ est vrai.
- Problème si:
  $\neg\, \texttt{pluie} \Rightarrow \texttt{soleil}$
  $\neg\, \texttt{soleil} \Rightarrow \texttt{pluie}$

  Le résultat dépend de l'ordre des règles!

Logique non-monotone

- En logique classique, toute inférence reste toujours valable: de nouvelles informations ne peuvent jamais infirmer les connaissances existantes. Les connaissances sont en croissance *monotone*.
- En pratique, on ne peut jamais garantir cette propriété: même si toutes les oiseaux volent, on peut découvrir plus tard qu'un certain oiseau a des défauts qui l'empêchent de voler.
- Solution: logique *non-monotone* qui permet de retirer des conclusions suite à de nouvelles informations.
- Cependant, la logique non-monotone pose des problèmes théoriques sévères (par ex. l'absence de point fixe).

Logique par défaut

- Vérifier toutes les conditions pour une inférence peut être très couteux:
  $\texttt{oiseau} \wedge \neg\texttt{autruche} \wedge \neg\texttt{ailes-coupées} \wedge \neg \ldots \Rightarrow \texttt{vole}$
- Souvent, les conditions se résument en l'absence d'anomalies:
  $\texttt{oiseau} \wedge \neg\texttt{anormal} \Rightarrow \texttt{vole}$
- On appelle ces cas des inférences *par défaut*.
- La logique par défaut permet l'inférence sur des systèmes ouverts, dont le modèle peut être incomplet.

Le traitement de l'incertitude

Raisons pour l'introduction de l'incertitude:

- imprécision de la modélisation.
- conclusions disjonctives non modélisable par des clauses de Horn.
- informations insuffisantes.

Principe du raisonnement incertain:

> *associer à chaque proposition une représentation numérique de l'incertitude*

Raisonnement *Bayesien*: les chiffres sont une estimation sur la base de l'ensemble des observations.

Formalismes pour l'incertitude

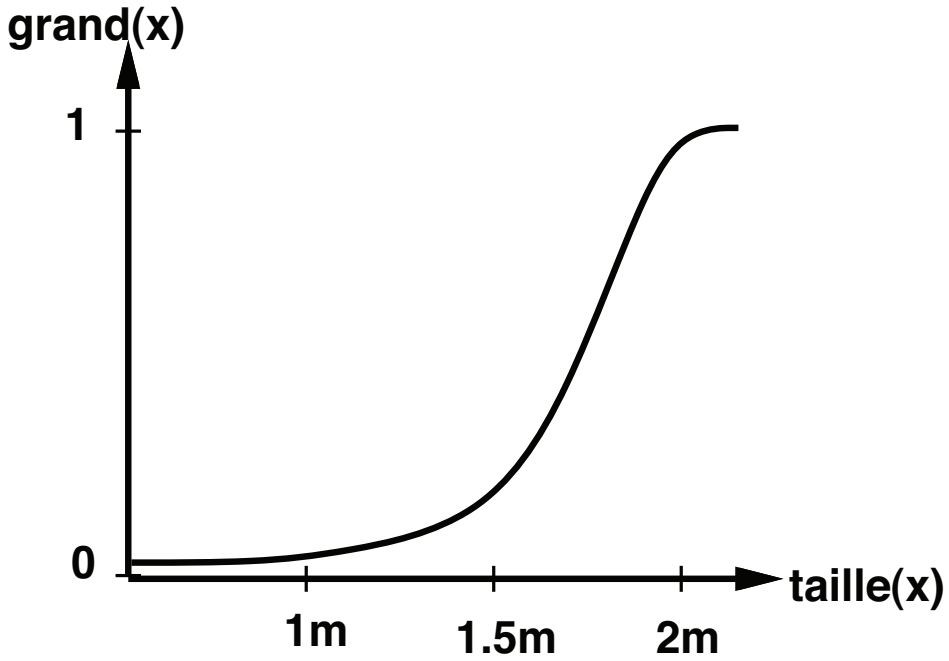
- Logique floue:
  faciles à appliquer, mais base théorique faible.
- Raisonnement probabiliste:
  très bien fondé, mais difficile à appliquer.
- Réseaux Bayesiens: faciles à appliquer et théoriquement bien fondées, mais applicables uniquement dans une interprétation *causale*.

Traduction langue ⇒ logique

- Connaissances exprimées en langue naturelle ⇒ imprécision.
- Exemple: est-ce que Jacques (1.80 mètres) est grand?
- Thailande: oui, Suède: non, Suisse: 50%(?)
- Comment modéliser?

La logique floue

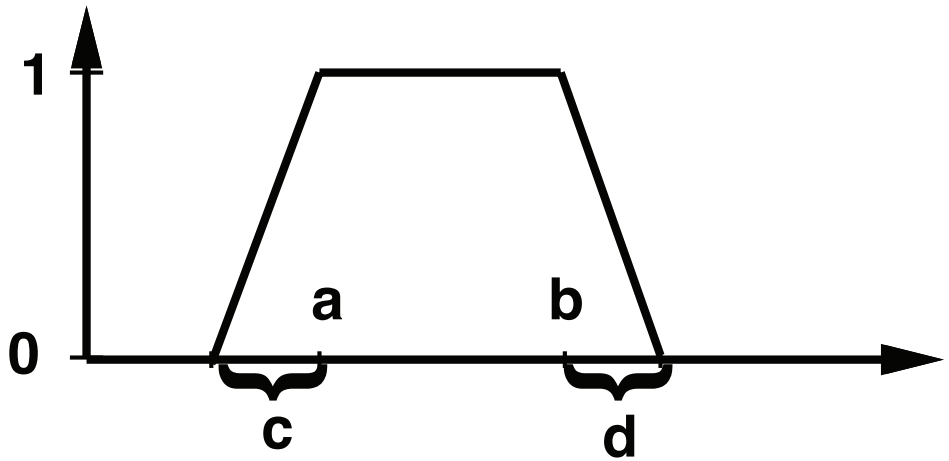
Idée principale: traduction de valeurs numériques en prédicats dont la validité est une distribution de possibilité:



Possibilité $\simeq$ pourcentage de gens qui diraient que la proposition est vraie.

Nombres floues

Les distributions à la base des nombres flous sont representées par 4 chiffres $[a, b, c, d]$:

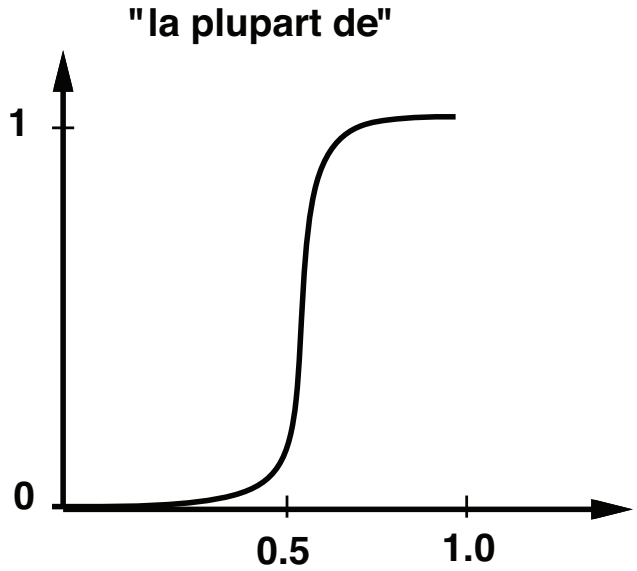


x	$< (a-c)$	$\in [a-c..a]$	$\in [a,b]$	$\in [b..b+d]$	$> (b+d)$
poss	0	$1-\frac{a-x}{c}$	1	$1-\frac{x-b}{d}$	0

$\Rightarrow$ permet le calcul de la possibilité de $x+y$ à partir des distributions pour x et y.

Les quantificateurs flous

Le calcul avec des distributions de possibilité permet également définir des quantificateurs flous:



- a(x) = poss(‘‘la-plupart-de’’)
- h = poss(‘‘personne-est-honnete’’), $h \geq 0$
- $\Rightarrow$ poss(‘‘la-plupart-des-personnes-sont-honnetes’’) = a(h)

Les facteurs de certitude

- Les connaissances d'un système expert correspondent à des expressions linguistiques imprécises.
- Imprécision est exprimé par un *facteur de certitude* (certainty factor, CF) associé à chaque fait et règle.
- Hybride entre logique floue et probabilité.
- Les CF sont des nombres réels entre -1.0 et 1.0:
  - -1.0: certainement faux
  - 0.0: inconnu
  - 0.735 (par exemple): probablement vrai
  - 1.0: certainement vrai

Propagation des facteurs de certitude

Application d'une règle:

$$\texttt{A} \wedge \texttt{B} \wedge \ldots \Rightarrow \texttt{X}$$

$\Rightarrow$ règle de combinaison en série:

$$CF(\texttt{X}) = CF(règle) * max(min(CF(A),CF(B),...)),0)$$

"maillon le plus faible" (relève de la logique floue)
jamais en dessous de 0

Combinaison de facteurs de certitude

2 chemins d'inférence pour la même proposition A:

1. chemin 1: $CF(\mathtt{A}) = x$
2. chemin 2: $CF(\mathtt{A}) = y$

$\Rightarrow$ règle de combinaison parallèle:

$$CF_{combine}(x,y) = \begin{cases} x + y - xy & \text{si } x \geq 0, y \geq 0 \\ x + y + xy & \text{si } x < 0, y < 0 \\ \frac{x+y}{1-min(|x|,|y|)} & \text{autrement} \end{cases}$$

Hypothèse: indépendance des composants.

Facteurs de certitude: exemple 1

Considérez:

P1(CF=0.8): client-solvable (CS)
P2(CF=0.7): client-regulier (CR)
P3(CF=0.9): petite-commande (PC)
R1(CF=0.9): CS $\rightarrow$ sans-accompte
R2(CF=0.8): CR $\wedge$ PC $\rightarrow$ sans-accompte

Inférence: P4(CF=??): sans-accompte

Facteurs de certitude de P4:

par la règle R1: CF(P4)= 0.8 * 0.9 = 0.72

par la règle R2: CF(P4)= max(min(0.7,0.9),0) * 0.8 = 0.56

combiner: CF(P4)= 0.72 + 0.56 - 0.72*0.56 = 0.88

$\Rightarrow$ les deux inférences se renforcent.

Facteurs de certitude: exemple 2

Considérez:

P1(CF=0.9): mauvais-payeur (MP)
P2(CF=0.7): client-regulier (CR)
P3(CF=0.99): petite-commande (PC)
R1(CF=-0.9): MP $\rightarrow$ sans-accompte
R2(CF=0.8): CR $\wedge$ PC $\rightarrow$ sans-accompte

Inférence: P4(CF=??): sans-accompte

Facteurs de certitude de P4:

par la règle R1: CF(P4)= 0.9 * -0.9 = -0.81

par la règle R2: CF(P4)= max(min(0.8,0.99),0) * 0.8 = 0.56

à combiner...

$\Rightarrow$ facteur de certitude définitif:

$$\begin{aligned} CF(P4) &= CF_{combine}(0.56, -0.81) \\ &= \frac{0.56 - 0.81}{1 - min(0.56, 0.81)} \\ &= -0.25/0.44 = -0.57 \end{aligned}$$

Donc, probablement plutot demander l'accompte.
Explication peut être plus complexe: dépendances entre mauvais-payeur et les autres critères.
Devrait être traité par un propre calcul probabiliste.

Les systèmes experts en pratique

Quelques-uns des premiers systèmes experts:

- Dendral: système d'analyse des spectroscopies de masse, le premier vrai système expert.
- Mycin: système diagnostiquant des infections et recommandant des antibiotiques, le SE le plus connu.
- Prospector: système d'interprétation de données géologiques pour détecter des gisements miniers, le SE qui a gagné le plus d'argent.
- CYC: outil pour la recherche documentaire intelligente.

Il existe des milliers de SE construits dans des entreprises, dont un très grand nombre fonctionne sur PC.

Le Boom des systèmes experts

L'Intelligence a connu 3 booms:

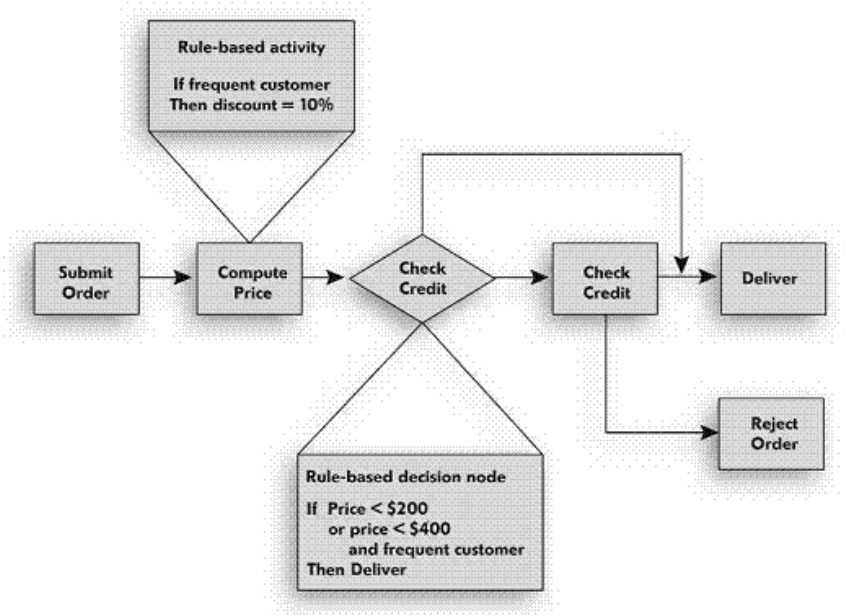
- 1965-1970: Planification (SHRDLU), Chatbots (ELIZA)
- 1980-1987: Systèmes Experts
- 2010 - aujourd'hui: vision, langue naturelle

En 1985, l'espoir était que la programmation serait remplacée par des systèmes experts:

- Programmation logique (PROLOG)
- Inférence parallèle (5e génération).
- Mais on oubliait la complexité de calcul, et la difficulté de mettre à jour les connaissances.
- Mêmes problèmes que aujourd'hui...

Business Rules

L'opération d'un entreprise peut être modélisée par des règles:



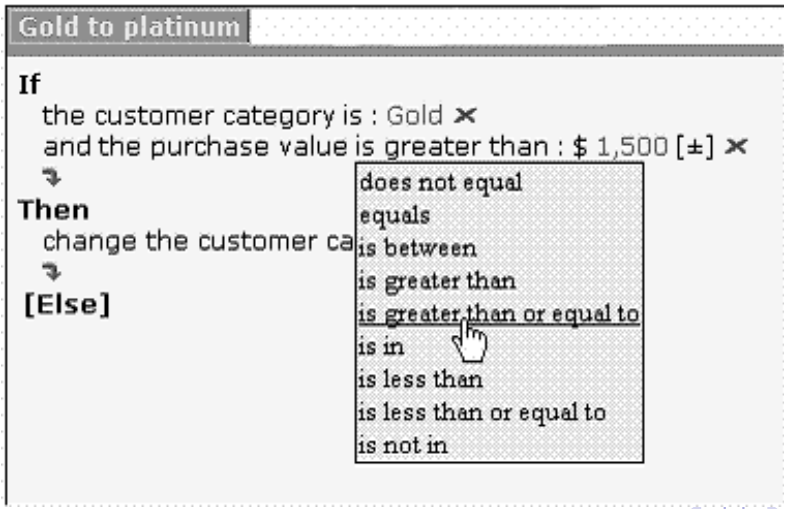
Buts:

- rendre les règles modifiables par les commerciaux
- faciliter l'intégration des règles entre partenaires commerciaux

Proprietés des Business Rules

- formulation en XML/RuleML
- divers moteurs d'inférence: ILOG Rules, Common Rules (IBM), etc.
- ordre de priorité entre règles pour eviter les problèmes de la negation as failure

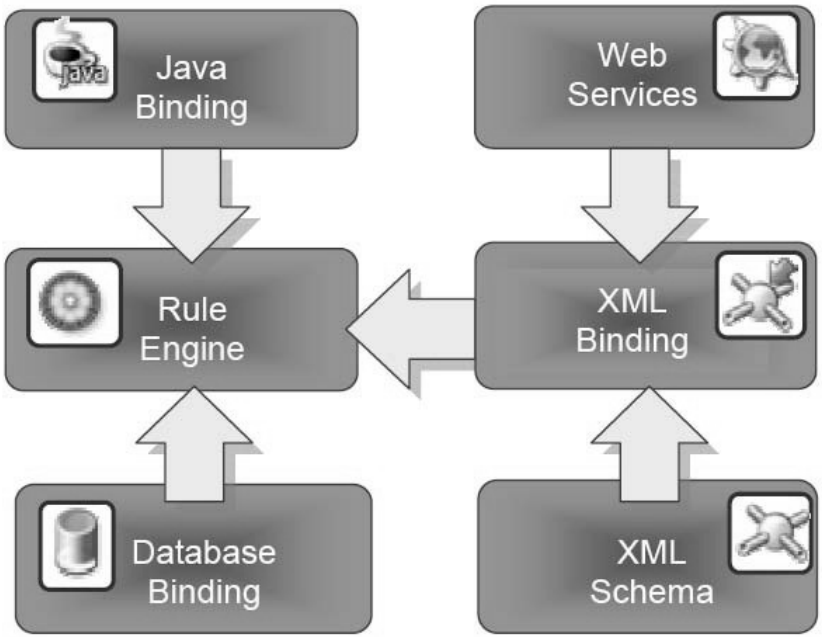
Editeur graphique:



Integration des Business Rules

Modèle: logiciel d'entreprise =

- services standardisées
- + règles de l'entreprise, écrites par les commerciaux mêmes



Application: Prédiction de toxicité

- Problème: savoir si une substance est toxique
- Très complexe et difficile même pour des experts
- Système expert DEREK très utilisé
- Résultat d'un longue développement, d'abord par un entreprise (Schering) et ensuite par une organisation non-profit (Lhasa) `http://www.lhasalimited.org`

Résumé

- Les systèmes experts sont des programmes capables de reproduire un raisonnement humain.
- Ils se basent sur la théorie psychologique établie par GPS (General Problem Solver).
- Eléments principaux des SE:
  - l'analyse moyens-buts
  - le chaînage arrière
  - les explications
- Logique non-monotone/défauts
- Logique floue
- Business rules