Técnicas de Inferência em BCs LPO

Construção de BCs: Princípios

Sumário

Técnicas de Inferência em BCs LPO:

- ♢ Provas
- ♢ Unificação
- ♢ Modus Ponens Generalizado
- ♢ Encadeamentos *forward* e *backward*
- ♢ Completeza
- ♢ Inferência por Resolução

Construção de BCs: Princípios:

- ♢ Propriedades de BCs
- ♢ A Engenharia do Conhecimento

Provas

Correção: achar apenas α tal que $BC \models \alpha$

Completeza: obter uma prova para qualquer α tal que $BC \models \alpha$

O processo de prova é uma busca, os operadores são as regras de inferência.

E.g., "operador" Modus Ponens (MP)

$$\frac{\alpha, \quad \alpha \Rightarrow \beta}{\beta} \qquad \frac{At(Marcos, ITA) \quad At(Marcos, ITA) \Rightarrow Happy(Marcos)}{Happy(Marcos)}$$

E.g., "operador" Introdução-AND (IA)

$$\frac{\alpha \quad \beta}{\alpha \wedge \beta} \qquad \frac{Happy(Marcos) \quad Engenharia(Marcos)}{Happy(Marcos) \wedge Engenharia(Marcos)}$$

Valem também para LPO. Adicionaremos mais algumas ...

Inferência envolvendo Quantificadores

Eliminação Universal (EU)

$$\frac{\forall x \;\; \alpha}{\alpha\{x/\tau\}} \qquad \frac{\forall x \;\; At(x, ITA) \;\Rightarrow\; Happy(x)}{At(Ana, ITA) \;\Rightarrow\; Happy(Ana)}$$

τ deve ser um termo $ground$ (i.e., sem variáveis)

Eliminação Existencial (EE)

$$\frac{\exists x \;\; \alpha}{\alpha\{x/\tau\}} \qquad \frac{\exists x \;\; At(x, ITA) \;\Rightarrow\; Happy(x)}{At(M, ITA) \;\Rightarrow\; Happy(M)}$$

τ deve ser um termo $ground$ **que não aparece na BC**.

Introdução Existencial (IE)

$$\frac{\alpha}{\exists x \;\; \alpha\{\tau/x\}} \qquad \frac{At(Ana, ITA) \;\Rightarrow\; Happy(Ana)}{\exists x \;\; At(x, ITA) \;\Rightarrow\; Happy(x)}$$

x deve ser uma variável **que não aparece na BC**.

Exemplo de prova

Bob é um gnu	1. $Gnu(Bob)$
Pat é um caracol	2. $Snail(Pat)$
Gnus correm mais que caracóis	3. $\forall x, y \;\; Gnu(x) \wedge Snail(y) \;\Rightarrow\; Faster(x, y)$
Bob corre mais que Pat	

Bob é um gnu	1. $Gnu(Bob)$
Pat é um caracol	2. $Snail(Pat)$
Gnus correm mais que caracóis	3. $\forall x, y \;\; Gnu(x) \wedge Snail(y) \;\Rightarrow\; Faster(x, y)$
Bob corre mais que Pat	
IA 1 & 2	4. $Gnu(Bob) \wedge Snail(Pat)$

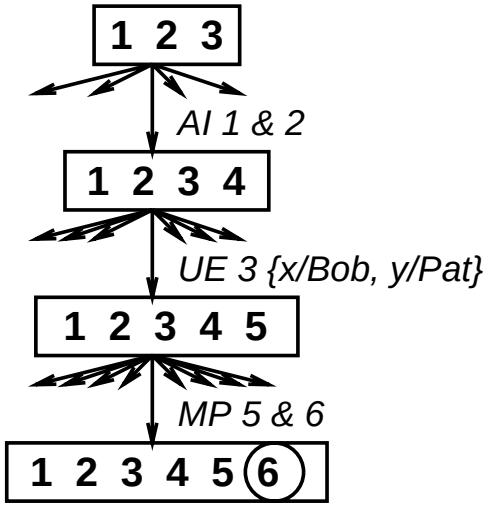
Exemplo de prova

Bob é um gnu	1. $Gnu(Bob)$
Pat é um caracol	2. $Snail(Pat)$
Gnus correm mais que caracóis	3. $\forall x, y \;\; Gnu(x) \wedge Snail(y) \;\Rightarrow\; Faster(x, y)$
Bob corre mais que Pat	
IA 1 & 2	4. $Gnu(Bob) \wedge Snail(Pat)$
EU 3, $\{x/Bob, y/Pat\}$	5. $Gnu(Bob) \wedge Snail(Pat) \;\Rightarrow\; Faster(Bob, Pat)$

Bob é um gnu	1. $Gnu(Bob)$
Pat é um caracol	2. $Snail(Pat)$
Gnus correm mais que caracóis	3. $\forall x, y \;\; Gnu(x) \wedge Snail(y) \;\Rightarrow\; Faster(x, y)$
Bob corre mais que Pat	
IA 1 & 2	4. $Gnu(Bob) \wedge Snail(Pat)$
EU 3, $\{x/Bob, y/Pat\}$	5. $Gnu(Bob) \wedge Snail(Pat) \;\Rightarrow\; Faster(Bob, Pat)$
MP 4 & 5	6. $Faster(Bob, Pat)$

Busca simples com regras de inferência

Operadores são regras de inferência
Estados são conjuntos de sentenças
Teste de objetivo checa estado para ver se este contém a sentença da pergunta feita à BC.



IA, EU, MP é padrão de inferência comum

Problema: fator de ramificação alto (esp. EU)

Solução: Melhorar espaço de busca com uma nova regra de inferência que execute vários passos em um só: Modus Ponens Generalizado (MPG)
Idéia de MPG: ache uma substituição que faz premissa de regra casar com alguns fatos conhecidos da BC

Unificação

Uma substituição σ unifica sentenças atômicas p e q se $\underline{\underline{p\sigma = q\sigma}}$

p	q	σ
$Knows(Jorge, x)$	$Knows(Jorge, Jane)$	
$Knows(Jorge, x)$	$Knows(y, Antonio)$	
$Knows(Jorge, x)$	$Knows(y, Mother(y)$	

Unificação

.

p	q	σ
$Knows(Jorge, x)$	$Knows(Jorge, Jane)$	$\{x/Jane\}$
$Knows(Jorge, x)$	$Knows(y, Antonio)$	$\{y/Jorge, x/Antonio\}$
$Knows(Jorge, x)$	$Knows(y, Mother(y)$	$\{y/Jorge, x/Mother(Jorge)\}$

Idéia: Unifique premissas da regra com fatos conhecidos, aplique unificador à conclusão

Por exemplo, se temos $Knows(Jorge, x) \Rightarrow Likes(Jorge, x)$

então concluímos $Likes(Jorge, Jane)$

$Likes(Jorge, Antonio)$

$Likes(Jorge, Mother(Jorge))$

Modus Ponens Generalizado (MPG)

$$\frac{p_1',\ p_2',\ \ldots,\ p_n',\ \ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\sigma}$$

onde $p_i'\sigma = p_i\sigma$ para todo i

E.g. p_1' = Faster(Oto,Ana)

p_2' = Faster(Ana,Marcos)

$p_1 \wedge p_2 \Rightarrow q = Faster(x,y) \wedge Faster(y,z) \Rightarrow Faster(x,z)$

$\sigma = \{x/Oto, y/Ana, z/Marcos\}$

$q\sigma = Faster(Oto, Marcos)$

MPG é usado com BC de cláusulas Horn:
uma única sentença atômica ou
(conjunção de sentenças atômicas) $\Rightarrow$ (sentença atômica)

Todas as variáveis assumidas universalmente quantificadas (preguiça de escrever $\forall\, x$ toda hora)

Correção de MPG

Precisamos mostrar que

$$p_1', \ \ldots, \ p_n', \ \ (p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models q\sigma$$

dado que $p_i'\sigma = p_i\sigma$ para todo i

Lema: para qualquer cláusula definida p, temos $p \models p\sigma$ por EU

1. $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models (p_1 \wedge \ldots \wedge p_n \Rightarrow q)\sigma = (p_1\sigma \wedge \ldots \wedge p_n\sigma \Rightarrow q\sigma)$
2. $p_1', \ \ldots, \ p_n' \models p_1' \wedge \ldots \wedge p_n' \models p_1'\sigma \wedge \ldots \wedge p_n'\sigma$
3. De 1 e 2, $q\sigma$ segue por MP simples

Encadeamento *forward*

Quando um fato novo p é adicionado à BC

para cada regra tal que p unifica com uma premissa

se as outras premissas são conhecidas

então adicione a conclusão à BC e continue encadeamento

Encadeamento *forward* é *data-driven*:

por exemplo, deduz propriedades e categorias à partir de percepções

Por outro lado, encadeamento *forward* pode gerar muitas conclusões irrelevantes

Encadeamento *forward*: exemplo

Adicione fatos 1, 2, 3, 4, 5, 7 um por um.
Número em [ ] = unificação literal; $\surd$ indica disparo de regra

1. $Gnu(x) \wedge Snail(y) \Rightarrow Faster(x, y)$
2. $Snail(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
3. $Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$
4. $Gnu(Bob)$ [1a,$\times$]
5. $Snail(Pat)$ [1b,$\surd$] $\rightarrow$ 6. $Faster(Bob, Pat)$ [3a,$\times$], [3b,$\times$]
   [2a,$\times$]
7. $Slug(Steve)$ [2b,$\surd$]
   $\rightarrow$8. $Faster(Pat, Steve)$ [3a,$\times$], [3b,$\surd$]
   $\rightarrow$9. $Faster(Bob, Steve)$ [3a,$\times$], [3b,$\times$]

Encadeamento *backward*

Quando uma questão q é perguntada

se um fato casado q' é conhecido, retorne o unificador

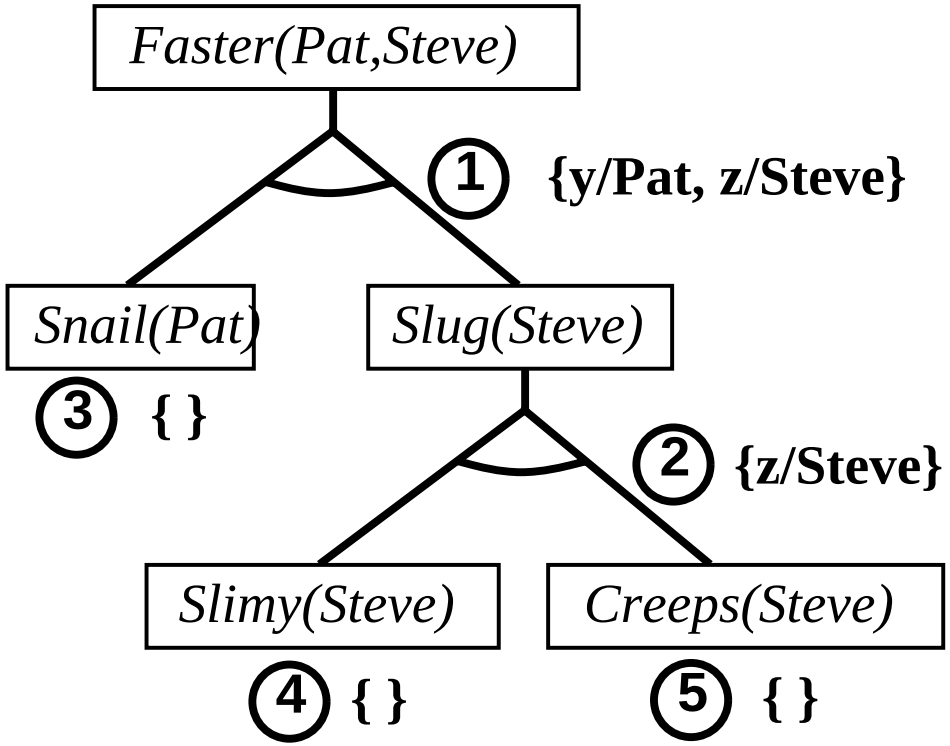
para cada regra cujo consequente q' casa com q

tente provar cada premissa da regra encadeando para trás

Duas versões: ache qualquer solução, ache todas as soluções.

Encadeamento *backward*: exemplo

1. $Snail(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
2. $Slimy(z) \wedge Creeps(z) \Rightarrow Slug(z)$
3. $Snail(Pat)$ 4. $Slimy(Steve)$ 5. $Creeps(Steve)$



Completeza em LPO

Relembrando: procedimento i é completo se e só se

$$BC \vdash_i \alpha \quad \text{sempre que} \quad BC \models \alpha$$

Encadeamentos *forward* e *backward* são completos para BCs Horn, mas incompletos para LPO em geral

Por exemplo, de

$$PhD(x) \Rightarrow HighlyQualified(x)$$
$$\neg PhD(x) \Rightarrow EarlyEarnings(x)$$
$$HighlyQualified(x) \Rightarrow Rich(x)$$
$$EarlyEarnings(x) \Rightarrow Rich(x)$$

deveria poder deduzir $Rich(Professor)$, mas EF/EB não fará isto. Por quê?

Existe um algoritmo completo para LPO?

Pequena História

450A.C.	Estóicos	lógica proposicional, inferência (talvez)
322A.C.	o Aristóteles	"silogismos" (regras de inferência), quantificadores
1565	Cardano	teoria de probabilidade (lógica proposicional + incerteza)
1847	Boole	lógica proposicional (de novo . . .)
1879	Frege	lógica de primeiro-ordem
1922	Wittgenstein	prova através de tabelas-verdade
1930	Gödel	$\exists$ algoritmo completo para LPO !
1930	Herbrand	algoritmo completo para LPO
1931	Gödel	$\neg\exists$ algoritmo completo para aritmética . . .
1960	Davis/Putnam	"algoritmo prático" para lógica proposicional
1965	Robinson	" algoritmo prático" para LPO — Resolução

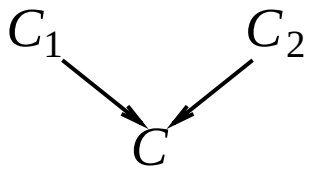
Resolução

Implicação em lógica de primeiro-ordem é apenas **semidecidível**: Posso achar uma prova de α se $BC \models \alpha$, mas nem sempre posso provar que $BC \not\models \alpha$
Problema de parada: procedimento de prova pode estar a ponto de terminar com sucesso ou fracasso, ou pode continuar para todo o sempre, amém . . .

Um procedimento de inferência completo usando resolução é refutação:
Para provar $BC \models \alpha$, mostre que $BC \wedge \neg\alpha$ é não-satisfatível

Resolução usaBC , $\neg\alpha$ em CNF ($Forma\ Normal\ Conjuntiva$)

Regra de inferência de resolução combina duas cláusulas e faz uma nova:



Inferência continua até que uma cláusula vazia seja derivada (contradição)

Resolução: Regra de Inferência

Versão proposicional básica:

$$\frac{\alpha \vee \beta, \ \ \neg\beta \vee \gamma}{\alpha \vee \gamma} \qquad \text{ou equivalentemente} \qquad \frac{\neg\alpha \ \Rightarrow \ \beta, \ \ \beta \ \Rightarrow \ \gamma}{\neg\alpha \ \Rightarrow \ \gamma}$$

Versão LPO completa:

$$\frac{\begin{array}{c} p_1 \vee \ldots \ p_j \ \ \ldots \vee p_m, \\ q_1 \vee \ldots \ q_k \ \ \ldots \vee q_n \end{array}}{(p_1 \vee \ldots \ p_{j-1} \vee p_{j+1} \ \ \ldots p_m \vee q_1 \ldots \ q_{k-1} \vee q_{k+1} \ \ \ldots \vee q_n)\sigma}$$

onde $p_j\sigma = \neg q_k\sigma$

Por exemplo,

$$\frac{\begin{array}{l} \neg Rich(x) \vee Unhappy(x) \\ Rich(Eu) \end{array}}{Unhappy(Eu)}$$

com $\sigma = \{x/Eu\}$

A Forma CNF

Literal = oração atômica (possivelmente negada), por exemplo, $\neg Rico(Eu)$

Cláusula = disjunção de literais, por exemplo, $\neg Rico(Eu) \vee Infeliz(Eu)$

A BC é uma conjunção de cláusulas

Qualquer BC LPO pode ser convertida para CNF como segue:

1. Substitua $P \Rightarrow Q$ por $\neg P \vee Q$
2. Mova $\neg$ para dentro, por exemplo, $\neg \forall x\, P$ se torna $\exists x\, \neg P$
3. Unifique variáveis separadamente, por exemplo, $\forall x\, P \vee \exists x\, Q$ se torna $\forall x\, P \vee \exists y\, Q$
4. Mova quantificadores para a esquerda, em ordem, por exemplo, $\forall x\, P \vee \exists x\, Q$ se torna $\forall x \exists y\, P \vee Q$
5. Elimine $\exists$ por Skolemização (próximo slide)
6. Elimine quantificadores universais
7. Distribua $\wedge$ entre os $\vee$, por exemplo, $(P \wedge Q) \vee R$ se torna $(P \vee Q) \wedge (P \vee R)$
8. Mais um passo (p/ converter para a forma normal implicativa): converter disjunções para implicações

   (literais negativos $\rightarrow$ literais positivos)

Skolemização

Nome estranho vem de Thoralf Skolem, que inventou o processo em 1928.

$\exists x\ Rico(x)$ se torna $Rico(G1)$ one $G1$ é uma nova "constante Skolem"

$\exists\, k\ \ \frac{d}{dy}(k^y) = k^y$ se torna $\frac{d}{dy}(e^y) = e^y$

Mais complicado se $\exists$ está dentro de $\forall$

E.g., "Todo mundo tem um coração"

$$\forall x\ \ Person(x) \Rightarrow \exists y\ \ Heart(y) \wedge Has(x, y)$$

Incorreto:

$$\forall x\ \ Person(x) \Rightarrow Heart(H1) \wedge Has(x, H1)$$

Correto:

$$\forall x\ \ Person(x) \Rightarrow Heart(H(x)) \wedge Has(x, H(x))$$

onde H é um novo símbolo ("função de Skolem")

Argumentos da função de Skolem: todas as variáveis universalmente quantificadas fora do quantificador existencial em questão.

Prova com Resolução

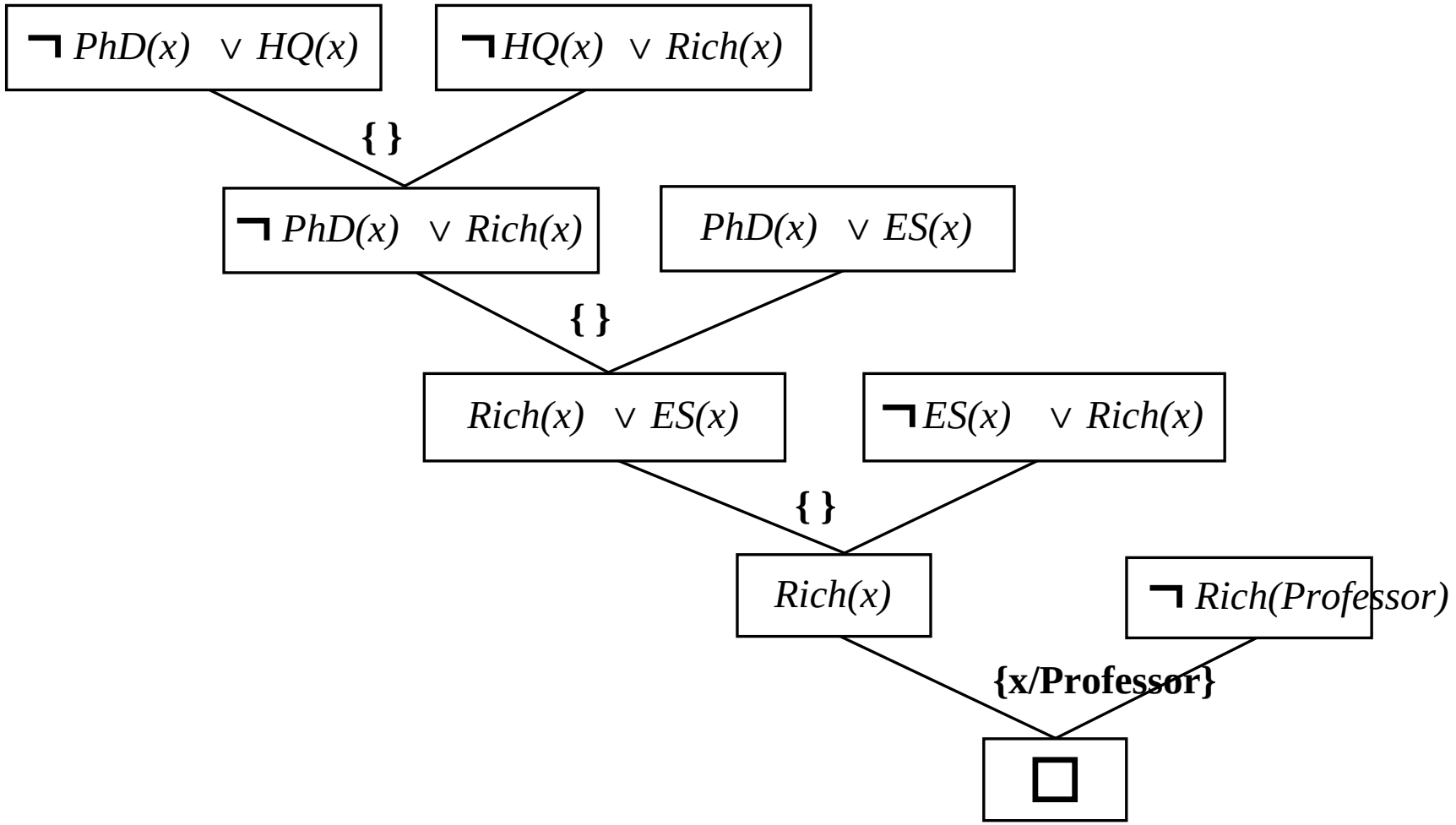
Para provar α:

- negue α
- converta para CNF
- adicione à BC CNF
- infira contradição

Por exemplo, para provar $Rich(Professor)$, adicione $\neg Rich(Professor)$ à BC CNF

$$\neg PhD(x) \vee HighlyQualified(x)$$
$$PhD(x) \vee EarlyEarnings(x)$$
$$\neg HighlyQualified(x) \vee Rich(x)$$
$$\neg EarlyEarnings(x) \vee Rich(x)$$

Prova com Resolução



Estratégias de Resolução

- Prefeência por cláusulas unitárias. Idéia: gerar sentenças mais curtas.
- Conjunto de suporte. Toda resolução usa pelo menos uma sentença do conjunto de suporte. Idéia: diminuir espaço de busca.
- Resolução sobre entrada. Produzir inferência usando sempre uma sentença da entrada, ou aquelas sentenças produzidas.
- *Subsumption*. Eliminar sentenças mais específicas do que aquelas presentemente na BC.

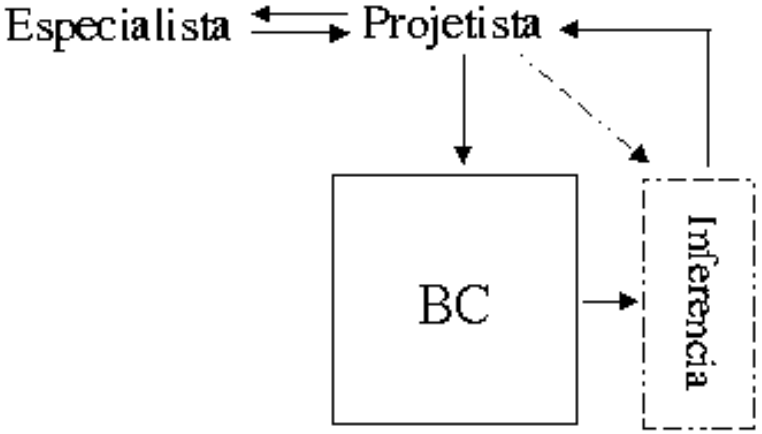
Resolução é completa e normalmente necessária em Matemática (refutação usada em provas por absurdo).

Construção de BCs e Eng. do Conhecimento

Vimos que LPO é uma linguagem razoavelmente flexível. Precisamos analisar agora como construir uma BC sobre a qual LPO irá atuar.

A construção de BCs para um dado domínio é tarefa complexa. O objetivo da *Engenharia do Conhecimento*. é definir BC que contenha os conceitos do domínio, e que admita representação sobre a qual um M.I. possa atuar.

Aquisição do conhecimento: entrevistas com especialistas, construção da representação formal. Projetista não deve se tornar um especialista!



Conceitos Básicos

♢ Clareza e Correção: Linguagem escolhida deve ser expressiva, concisa e não-ambígua. BC deve conter tudo o queé relevante, e não conter nada que seja irrelevante.

♢ Eficiência: Nem sempre é possível separar projeto de BC e projeto de motor de inferência. Projetista precisa entender como inferência é feita.

♢ Experiência é fundamental. Não existem metodologias "fechadas".

♢ BC é para inferência artificial, e não para inferência humana !

Especificidade da Representação

Computadores são mais burros do que humanos. . . Suponha que queiramos inferir se o urso Pooh, que tem um cérebro diminuto, é de fato estúpido.

$$\forall x \;\; BearofVerySmallBrain(b) \Rightarrow Silly(b)$$

Muito específico. Não consigo inferir nem se b é um urso. Para humanos, inferência é implícita: lemos $BearofVerySmallBrain$ e sabemos que isto faz referência a um tipo de urso.

Representação do conhecimento deve evitar especificidade excessiva.

Especificidade da Representação

Melhor seria algo como:

$$Bear(Pooh)$$
$$\forall b \;\; Bear(b) \Rightarrow Animal(b)$$
$$\forall b \;\; Animal(b) \Rightarrow PhysicalThing(b)$$

que categoriza ursos e os relaciona com contextos mais amplos.

A idéia de tamanho é relativa a um padrão. Podemos escrever:

$$RelativeSize(BrainOf(Pooh), BrainOf(TypicalBear)) = Very(Small)$$

etc, etc.

Princípios para Sentenças

Ao escrever sentenças:

- Verifique a veracidade. É possível reescrever a sentença em função de fatos mais básicos?
- BC é para várias sentenças. Procure usar predicados e funções aplicáveis a muitos objetos.
- Evite criar novos predicados 'a toa. A classe de objetos que satisfaz o predicado é uma subclasse ou superclasse de outras classes?

Uma Metodologia para EC

1. Entendimento básico do domínio: o que é o problema?
2. Escolha do vocabulário: predicados, funções e constantes (ontologia do problema).
3. Representar conhecimento geral do domínio: axiomas e sentenças lógicas.
4. Representar conhecimento específico do problema.
5. Preparar perguntas ao mecanismo de inferência e avaliar respostas.

Isto é mais fácil de falar do que fazer...

"IA se aprende pelos dedos"

Entendimento do Domínio

Circuitos eletrônicos = fios + portas lógicas.

Sinais circulam através dos fios.

Portas lógicas recebem sinais nos terminais de entrada e produzem sinais no terminal de saída.

Circuitos também possuem terminais de entrada e terminais de saída.

Portas lógicas que nos interessam são AND, OR, XOR e NOT.

Observe que esta é uma ontologia bem limitada, mas suficiente pro nosso domínio.

Escolha do Vocabulário

Distinguindo portas: $X1, X2$, etc.

Definindo tipos de portas: $Type(X1) = XOR$, etc. Note que $Type$ é uma função, o que dispensa escrever axiomas que dizem que uma porta só pode ter um tipo.

Identificando terminais: $X1In1, X1In2, X1Out$ é muito específico. Melhor algo como $In(1, X1), In(2, X1), Out(1, X1)$. Predicados In e Out servem pra qualquer porta (e talvez pra outros circuitos).

Conectividade entre terminais: Predicado $Connected(Out(1, X1), In(2, A2))$.

Identificando sinal (1 ou 0): Função $Signal(.)$, argumento sendo um terminal e imagem sendo objetos On ou Off. Isto é melhor do que um predicado $On(Terminal)$ devolvendo $Verdadeiro$ ou $Falso$. Por quê?

Codificando Regras Gerais

$\forall\, t_1, t_2 \;\; Connected(t_1, t_2) \;\Rightarrow\; Signal(t_1) = Signal(t_2)$

$\forall\, t \;\; Signal(t) = On \vee Signal(t) = Off$

$\forall\, t_1, t_2 \;\; Connected(t_1, t_2) \;\Leftrightarrow\; Connected(t_2, t_1)$

$\forall\, g \;\; Type(g) = OR \;\Rightarrow\; Signal(Out(1, g)) = On \;\Leftrightarrow\; \exists\, n \;\; Signal(In(n, g)) = On$

$\forall\, g \;\; Type(g) = AND \;\Rightarrow\; Signal(Out(1, g)) = Off \;\Leftrightarrow\; \exists\, n \;\; Signal(In(n, g)) = Off$

$\forall\, g \;\; Type(g) = XOR \;\Rightarrow\; Signal(Out(1, g)) = Off \;\Leftrightarrow\; Signal(In(1, g)) = Signal(In(2, g))$

$\forall\, g \;\; Type(g) = NOT \;\Rightarrow\; Signal(Out(1, g)) \neq Signal(In(1, g))$

Codificando Conhecimento Específico

$Type(X1) = XOR$, $Type(X2) = XOR$, $Type(A1) = AND$, $Type(A2) = AND$, $Type(O1) = OR$

$$Connected(Out(1, X1), In(1, X2)) \quad Connected(In(1, C1), In(1, X1))$$
$$Connected(Out(1, X1), In(2, A2)) \quad Connected(In(1, C1), In(1, A1))$$
$$Connected(Out(1, A2), In(1, O1)) \quad Connected(In(2, C1), In(2, X1))$$
$$Connected(Out(1, A1), In(2, O1)) \quad Connected(In(2, C1), In(2, A1))$$
$$Connected(Out(1, X2), Out(1, C1)) \quad Connected(In(3, C1), In(2, X2))$$
$$Connected(Out(1, O1), Out(2, C1)) \quad Connected(In(3, C1), In(1, A2))$$

testando

Exemplo 1:

$\textsc{Ask}(BC, \exists \quad i_1, i_2, i_3\, Signal(In(1, C1)) = i_1 \wedge Signal(In(2, C1)) = i_2 \wedge Signal(In(3, C1)) = i_3 \wedge Signal(Out(1, C1)) = Off \wedge Signal(Out(2, C1)) = On$

Exemplo 2 (verificação do circuito):

$\textsc{Ask}(BC, \exists \quad i_1, i_2, i_3, o_1, o_2\, Signal(In(1, C1)) = i_1 \wedge Signal(In(2, C1)) = i_2 \wedge Signal(In(3, C1)) = i_3 \wedge Signal(Out(1, C1)) = o_1 \wedge Signal(Out(2, C1)) = o_2$

Verificação do circuito. Note que posso usar a mesma BC (com um conjunto de sentenças de conhecimento específico modificado) para fazer verificação em circuitos digitais muito mais complexos.