Agentes Lógicos

Inteligência Artificial
PCS3438

Anna Helena Reali Costa
Escola Politécnica da USP
Engenharia de Computação (PCS)

CONVERSÃO DE FOL PARA CLÁUSULAS

Transformação para Forma Clausal

1. Substituir $\alpha \leftrightarrow \beta$ por $(\alpha \rightarrow \beta) \land (\beta \rightarrow \alpha)$ e substituir $\alpha \rightarrow \beta$ por $\neg\alpha \vee \beta$
2. Colar as negações nos átomos, utilizando as equivalências $\neg(\neg\alpha) \equiv \alpha$, $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$, $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$, $\neg\exists x\ \alpha \equiv \forall x\ \neg\alpha$, e $\neg\ \forall\ x\ \alpha \equiv \exists x\ \neg\alpha$.
3. Padronizar as variáveis, trocando os nomes quando estas aparecem no escopo de quantificadores diferentes
4. Remover os quantificadores existenciais utilizando variáveis e funções de Skolem
5. Remover os quantificadores universais
6. Distribuir as disjunções pelas conjunções, utilizando $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$

Transformação para Forma Clausal

Prova-se que qualquer fbf do cálculo de predicados pode ser transformada em um conjunto de cláusulas equivalente, através de uma sequência definida de passos.

Exemplo: "Todo aquele que ama todos os animais é amado por alguém"

Como seria a representação disto em lógica de predicados?

$$\forall x\ (\forall y\ (Animal(y) \rightarrow Loves(x,y))) \rightarrow (\exists y\ (Loves(y, x)))$$

Transformação para Forma Clausal

$\forall x\ (\forall y\ Animal(y) \rightarrow Loves(x,y)) \rightarrow (\exists y\ Loves(y, x))$

1. Substituir $\alpha \leftrightarrow \beta$ por $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$ e substituir $\alpha \rightarrow \beta$ por $\neg\alpha \vee \beta$

$\forall x \neg\ (\forall y\ \neg Animal(y) \vee Loves(x,y)) \vee (\exists y\ Loves(y, x))$

2. Colar as negações nos átomos, utilizando as equivalências $\neg(\neg\alpha) \equiv \alpha$, $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$, $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$, $\neg\exists x\ \alpha \equiv \forall x\ \neg\alpha$, e $\neg\forall x\ \alpha \equiv \exists x\ \neg\alpha$.

$\forall x(\exists\ y \neg\ (\neg Animal(y) \vee Loves(x,y))) \vee (\exists y\ Loves(y, x))$

$\forall x(\exists\ y\ \neg\neg Animal(y) \wedge \neg Loves(x,y)) \vee (\exists y\ Loves(y, x))$

$\forall x(\exists\ y\ Animal(y) \wedge \neg Loves(x,y)) \vee (\exists y\ Loves(y, x))$

Transformação para Forma Clausal

$\forall x(\exists y\ Animal(y) \wedge \neg Loves(x,y)) \vee (\exists y\ Loves(y, x))$

3. Padronizar as variáveis, trocando os nomes quando estas aparecem no escopo de quantificadores diferentes

$\forall x(\exists y\ Animal(y) \wedge \neg Loves(x,y)) \vee (\exists z\ Loves(z, x))$

4. Remover os quantificadores existenciais utilizando variáveis e funções de Skolem

$\forall x(Animal(A) \wedge \neg Loves(x,A)) \vee Loves(B, x)$ ***ERRO!!!***

$\forall x(Animal(F(x)) \wedge \neg Loves(x,F(x))) \vee Loves(G(x), x)$

5. Remover os quantificadores universais

$(Animal(F(x)) \wedge \neg Loves(x,F(x))) \vee Loves(G(x), x)$

Transformação para Forma Clausal

$(Animal(F(x)) \land \neg Loves(x,F(x))) \lor Loves(G(x), x)$

6. Distribuir as disjunções pelas conjunções, utilizando $\alpha \lor (\beta \land \gamma) \equiv (\alpha \lor \beta) \land (\alpha \lor \gamma)$

$(Animal(F(x)) \lor Loves(G(x), x)) \land$

$(\neg Loves(x,F(x)) \lor Loves(G(x), x))$

Esta sentença gerou 2 cláusulas na KB

Mais um exemplo: função de Skolem

- Quando um quantificador existencial estiver aninhado internamente a um quantificador universal, devemos usar funções de Skolem (caso contrário, basta substituir $\exists x\ P(x)$ por $P(A)$, sendo A uma constante que ainda não apareceu em lugar algum da KB):

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

- **ERRO**: $\forall x\ Person(x) \rightarrow Heart(H) \wedge Has(x,H)$ pois indica que <u>todos</u> têm o <u>mesmo</u> coração H!
- **CORRETO**: $\forall x\ Person(x) \rightarrow Heart(H(x)) \wedge Has(x,H(x))$

Regras de Inferência para Cálculo de Predicados

Resolução

$$p1 \vee \ldots \vee pi \vee \ldots \vee pn, \quad m1 \vee \ldots \vee mi \vee \ldots mn$$

$$\models$$

$$(p1 \vee \ldots \vee pi\text{-}1 \vee pi{+}1 \vee \ldots \vee pn \vee$$
$$m1 \vee \ldots \vee mi\text{-}1 \vee mi{+}1 \vee \ldots \vee mn)\ \sigma$$

onde $pi\ \sigma = \neg mi\ \sigma$

Regras de Inferência para Cálculo de Predicados

Exemplo:

$Animal(F(x)) \vee Loves(G(x), x)$ $\neg Loves(u, v) \vee \neg Kills(u, v)$

unificam com a substituição $\{u/G(x), v/x\}$ e geram o resolvente:

$Animal(F(x)) \vee \neg Kills(G(x), x)$

Outro Exemplo

- All Romans who know Marcus either hate Caesar or think that anyone who hates anyone is crazy

- $\forall x$, [Roman(x) $\wedge$ know(x, Marcus)] $\rightarrow$
  [hate(x, Caesar) $\vee$
  ($\forall y$, $\exists z$, hate(y, z) $\rightarrow$ thinkCrazy(x, y))]

- Use o fato que $x \rightarrow y$ é equivalente a $\neg x \vee y$

- $\forall x$, [Roman(x) $\wedge$ know(x, Marcus)] $\rightarrow$
  [hate(x, Caesar) $\vee$
  ($\forall y$, $\exists z$, hate(y, z) $\rightarrow$ thinkCrazy(x, y))]

- $\forall x$, $\neg$[Roman(x) $\wedge$ know(x, Marcus)] $\vee$
  [hate(x, Caesar) $\vee$
  ($\forall y$, $\neg$($\exists z$, hate(y, z) $\vee$ thinkCrazy(x, y))]

- Reduza o escopo da negação para um único termo, usando:
  - $\neg(\neg p) \equiv p$
  - $\neg(a \wedge b) \equiv (\neg a \vee \neg b)$
  - $\neg(a \vee b) \equiv (\neg a \wedge \neg b)$
  - $\neg\forall x,\ p(x) \quad \equiv \quad \exists x,\ \neg p(x)$
  - $\neg\exists x,\ p(x) \quad \equiv \quad \forall x,\ \neg p(x)$

- $\forall x,\ \neg[\ Roman(x) \wedge know(x, Marcus)\ ] \vee$
  $[hate(x, Caesar) \vee$
  $(\forall y,\ \neg(\exists z,\ hate(y, z) \vee thinkCrazy(x, y))]$

- $\forall x,\ [\ \neg Roman(x) \vee \neg know(x, Marcus)\ ] \vee$
  $[hate(x, Caesar) \vee$
  $(\forall y,\ \forall z,\ \neg hate(y, z) \vee thinkCrazy(x, y))]$

- Padronizar variáveis separadamente:

$\forall x, P(x) \lor \forall x, Q(x)$

becomes

$\forall x, P(x) \lor \forall y, Q(y)$

- Isso é apenas para evitar que os escopos das variáveis fiquem confusos
- *Não necessário no nosso exemplo*

- Mova todos quantificadores para a esquerda, sem trocar suas posições relativas:

- $\forall x, [\neg Roman(x) \vee \neg know(x, Marcus)] \vee [hate(x, Caesar) \vee (\forall y, \forall z, \neg hate(y, z) \vee thinkCrazy(x, y)]$

- $\forall x, \forall y, \forall z, [\neg Roman(x) \vee \neg know(x, Marcus)] \vee [hate(x, Caesar) \vee (\neg hate(y, z) \vee thinkCrazy(x, y))]$

- Eliminar quantificadores existenciais:
- Fazemos isso com as funções de Skolem:
  - Se $\exists x, p(x)$ então só pegue um, nomeando-o x'
  - Se o quantificador existencial estiver sob controle de um quantificador universal, o valor escolhido deverá ser uma função da variável quantificada universalmente:
    - Se $\forall x, \exists y, p(x, y)$ então $\forall x, p(x, y(x))$

- *Não necessário no nosso exemplo*

- $\forall x, \forall y, \forall z, [\neg Roman(x) \lor \neg know(x, Marcus)] \lor [hate(x, Caesar) \lor (\neg hate(y, z) \lor thinkCrazy(x, y))]$

- Neste ponto, todos os quantificadores são quantificadores universais
- Assumimos que todas as variáveis são quantificadas universalmente e eliminamos os quantificadores
- $[\neg Roman(x) \lor \neg know(x, Marcus)] \lor [hate(x, Caesar) \lor (\neg hate(y, z) \lor thinkCrazy(x, y))]$

- Criar uma conjunção de disjunções:

- $[\ \neg Roman(x) \lor \neg know(x, Marcus)\ ] \lor [hate(x, Caesar) \lor (\neg hate(y, z) \lor thinkCrazy(x, y))]$

  fica:

  $\neg Roman(x) \lor \neg know(x, Marcus) \lor hate(x, Caesar) \lor \neg hate(y, z) \lor thinkCrazy(x, y)$

- Em todo lugar que houver um $\wedge$, separamos nossa expressão em partes distintas
  - *Não necessário no nosso exemplo*

- Renomeie variáveis para que não haja duas cláusulas com a mesma variável
  - *Não necessário no nosso exemplo*

- Resultado final:
  $\neg$Roman(x) $\vee$ $\neg$know(x, Marcus) $\vee$
  hate(x, Caesar) $\vee$ $\neg$hate(y, z) $\vee$ thinkCrazy(x, y)

- *É isso! É um processo longo, mas fácil de fazer mecanicamente*