Lógicas Não Monótonas

Ano Lectivo 2010/2011

Resumo

- ♢ Limitações da lógica clássica na representação do conhecimento
- ♢ Lógica por omissão
- ♢ Lógica autoepistémica
- ♢ Semânticas de Programação em Lógica

Limitações da lógica clássica

Considere-se o conjunto de cláusulas

$$ave(X) \Rightarrow voa(X)$$
$$pinguim(X) \Rightarrow ave(X)$$
$$pinguim(X) \Rightarrow \neg voa(X)$$

O que se deriva da teoria anterior juntamente com

1. $ave(b1)$?
2. $\neg voa(b1)$?
3. $ave(b1)$ e $\neg voa(b1)$?
4. $pinguim(fred)$?

A lógica clássica é monótona

O comportamento ilustrado anteriormente é justificado pela propriedade da monotonia da lógica clássica.

> Se algo é concluído a partir do que sei agora, então continuará a ser concluído no futuro.

Formalmente,

Se $T \models \phi$ então $T \cup F \models \phi$, para qualquer teoria T e F e fórmula ϕ.

O mundo aberto

Considere-se a pequena base de dados representada em lógica:

$$BD_1 = \left\{ \begin{array}{lll} cadeira(iia) & cadeira(sw) & docente(123, \text{'ABC'}) \\ lecciona(iia, t, 123) & lecciona(sw, t, 123) & docente(456, \text{'DEF'}) \\ lecciona(iia, p, 456) & & \end{array} \right\}$$

Quais as cadeiras só com aulas teóricas?

$\neg lecc(iia, p, 123)?$	$\neg lecc(sw, p, 123)?$
$\neg lecc(iia, p, 456)?$	$\neg lecc(sw, p, 456)?$
$\neg lecc(iia, p, 123) \wedge \neg lecc(iia, p, 456)?$	$\neg lecc(sw, p, 123) \wedge \neg lecc(sw, p, 456)?$
$lecc(iia, p, 123)?$	$lecc(sw, p, 123)?$
$lecc(iia, p, 456)?$	$lecc(sw, p, 456)?$
$lecc(iia, p, 123) \vee lecc(iia, p, 456)?$	$lecc(sw, p, 123) \vee lecc(sw, p, 456)?$

O mundo aberto

Considere-se a pequena base de dados representada em lógica:

$$BD_1 = \left\{ \begin{array}{lll} cadeira(iia) & cadeira(sw) & docente(123, \text{'ABC'}) \\ lecciona(iia, t, 123) & lecciona(sw, t, 123) & docente(456, \text{'DEF'}) \\ lecciona(iia, p, 456) & & \end{array} \right\}$$

Quais as cadeiras só com aulas teóricas?

$$\begin{array}{rr} \neg lecc(iia, p, 123) \times & \neg lecc(sw, p, 123) \times \\ \neg lecc(iia, p, 456) \times & \neg lecc(sw, p, 456) \times \\ \neg lecc(iia, p, 123) \wedge \neg lecc(iia, p, 456) \times & \neg lecc(sw, p, 123) \wedge \neg lecc(sw, p, 456) \times \\ lecc(iia, p, 123) \times & lecc(sw, p, 123) \times \\ lecc(iia, p, 456) \surd & lecc(sw, p, 456) \times \\ lecc(iia, p, 123) \vee lecc(iia, p, 456) \surd & lecc(sw, p, 123) \vee lecc(sw, p, 456) \times \end{array}$$

Hipótese do mundo fechado

Nas bases de dados usualmente adoptam-se as duas seguintes hipóteses simplificadoras:

♢ Hipótese do Mundo Fechado (CWA - Closed World Assumption)

♢ Hipótese do Domínio Fechado (asssume-se apenas a existência dos objectos mencionados na teoria)

O que se conclui por hipótese do mundo fechado é obtido com:

$$Cn_{CWA}(\Sigma) = \{\phi \mid \Sigma \cup \Sigma_{CWA} \models \phi\}$$

onde ϕ é um literal positivo e

$$\Sigma_{CWA} = \{\neg\phi \mid \Sigma \not\models \phi\}$$

Assim, já obtemos $\neg lecciona(sw, p, 123) \wedge \neg lecciona(sw, p, 456) \in Cn_{CWA}(BD_1)$!

O que acontece se adicionarmos a BD_1 a informação $lecciona(sw, p, 123)$?

Problemas da CWA

Considere-se a teoria

$$praia \vee cinema \qquad praia \qquad \neg praia \wedge \neg cinema \Rightarrow aborrecido$$

Será que se pode concluir $aborrecido$?

E de ?

$$praia \vee cinema \qquad \neg praia \wedge \neg cinema \Rightarrow aborrecido$$

Problemas da CWA

Considere-se a teoria

$$praia \vee cinema \qquad praia \qquad \neg praia \wedge \neg cinema \Rightarrow aborrecido$$

Será que se pode concluir $aborrecido$? Não, como se deseja.

E de ?

$$praia \vee cinema \qquad \neg praia \wedge \neg cinema \Rightarrow aborrecido$$

Sim! A teoria fica inconsistente, pois como nem praia nem cinema so concluídos da teoria original, então quer $\neg praia$ quer $\neg cinema$ são adicionados!

Mais um exemplo ilustrando que o raciocínio com CWA é não monótono.

Lógica por Omissão (Reiter)

Uma teoria de lógica por omissão (Default Logic) é um par

$$\langle D, W \rangle$$

em que W é um conjunto de fórmulas de LPO e D é um conjunto de regras por omissão (defaults) com a forma

$$\frac{\varphi \quad : \quad \psi_1, \ldots, \psi_n}{\chi}$$

Com a leitura "Se φ e for consistente assumir $\psi_1, \ldots, \psi_n$ então χ", em que:

- φ é o pré-requisito
- $\psi_1, \ldots, \psi_n$ é a justificação
- χ é a conclusão

Os passaritos novamente...

Normalmente as aves voam (Regra por defeito):

$$\frac{ave(X) \quad : \quad voa(X)}{voa(X)}$$

E o conhecimento imutável (teoria W)

$\forall_X (pinguim(X) \Rightarrow ave(X)) \wedge \forall_X (pinguim(X) \Rightarrow \neg voa(X)) \wedge ave(tweety)$

O que se pode concluir da teoria por omissão anterior?

E se juntarmos $pinguim(tweety)$?

Cálculo de extensões

E é uma extensão sse

$$
\begin{array}{l}
E_0 = W \\
E_{i+1} = Cn(E_i) \cup \left\{ \chi \;\mid\; \frac{\varphi \,:\, \psi_1,\ldots,\psi_n}{\chi} \text{ e } \varphi \in E_i \text{ e } \neg\psi_1 \notin E, \ldots, \neg\psi_n \notin E \right\} \\
E = \cup_{i=0}^{\infty} \quad E_i
\end{array}
$$

Em geral, o cálculo de extensões é indecidível para teorias de lógica de primeira ordem. Mas para o caso proposicional o problema é decidível.

Outro exemplo

Normalmente, os cisnes são brancos.

$$\frac{cisne(X) \quad : \quad branco(X)}{branco(X)}$$

Normalmente, os cisnes australianos são pretos.

$$\frac{cisne(X) \wedge australiano(X) \quad : \quad preto(X)}{preto(X)}$$

Bruce é um cisne australiano.

$$cisne(bruce) \wedge australiano(bruce)$$

Nada é preto e branco simultaneamente

$$\forall_X \neg (branco(X) \wedge preto(X))$$

Extensões para o problema dos cisnes

Para o exemplo dos cisnes temos duas extensões:

1. Uma extensão E_1 que contém
   $\{cisne(bruce), australiano(bruce), branco(bruce), \neg preto(bruce)\}$
2. Uma extensão E_2 que contém
   $\{cisne(bruce), australiano(bruce), preto(bruce), \neg branco(bruce)\}$

As regras por omissão anteriores são normais, pois têm a forma:

$$\frac{\varphi \quad : \quad \chi}{\chi}$$

Se as regras numa teoria por omissão forem todas normais, então garante-se a existência de pelo menos uma extensão.

Exercício

Quais são extensões da teoria

$$D = \left\{ \begin{array}{c} \frac{adulto \quad : \quad empregado}{empregado} \\ \frac{estudante \quad : \quad \neg empregado}{\neg empregado} \\ \frac{estudante \quad : \quad adulto}{adulto} \end{array} \right\}$$

com

$$W = \{estudante\}$$

?

Lógica autoepistémica (Robert Moore)

Imagine-se a seguinte pergunta:

♢ O Papa está hoje na FCT ?

Qual o argumento ?

Lógica autoepistémica (Robert Moore)

Imagine-se a seguinte pergunta:

♢ O Papa está hoje na FCT ?

Qual o argumento ?

Não está. Se estivesse eu teria conhecimento desse facto! Formalmente:

$$papaNaFCT \Rightarrow \mathbf{L}papaNaFCT$$

A fórmula $\mathbf{L}\varphi$ significa "Eu acredito em φ" ou "Eu sei φ".

Lógica autoepistémica (Robert Moore)

Expansão:

$$E = Cn\left(T \cup \{L\varphi \mid \varphi \in E\} \cup \{\neg L\varphi \mid \varphi \notin E\}\right)$$

Para o nosso exemplo é óbvio que não temos maneira de concluir $papaNaFCT$ logo na expansão deverei ter também $\neg \mathbf{L} papaNaFCT$.

Por contrapositiva, conclui-se que $\neg papaNaFCT$. E que mais ?

Lógica autoepistémica (Robert Moore)

Expansão:

$$E = Cn\left(T \cup \{L\varphi \mid \varphi \in E\} \cup \{\neg L\varphi \mid \varphi \notin E\}\right)$$

Para o nosso exemplo é óbvio que não temos maneira de concluir $papaNaFCT$ logo na expansão deverei ter também $\neg \mathbf{L} papaNaFCT$.

Por contrapositiva, conclui-se que $\neg papaNaFCT$. E que mais ?

$$\mathbf{L}\neg papaNaFCT \qquad \mathbf{L}\neg\mathbf{L} papaNaFCT$$
$$\vdots \qquad\qquad\qquad \vdots$$

Programação em Lógica

Programa em lógica definido ou positivo é um conjunto de regras:

$$A :- B_1, \ldots, B_m.$$

em que A, $B_1, \ldots, B_m$ são átomos da Lógica de Primeira Ordem. Se $n = 0$ temos um facto representado por A.

Uma regra lê-se "A se B_1 e ... e B_m", correspondendo à implicação

$$\forall \left(B_1 \wedge \ldots \wedge B_m \Rightarrow A\right)$$

ou seja, é um conjunto de cláusulas de Horn.

Algumas variantes da programação em lógica admitem restrições de integridade com a forma:

$$:- B_1, \ldots, B_m.$$

Os programas definidos datalog são aqueles que não utilizam símbolos de função nos seus termos, i.e. os termos ou são variáveis ou constantes.

Exemplos de Programas em Lógica

Programa definido datalog

$$
\begin{aligned}
&shutdown : -overheat.\\
&shutdown : -leak.\\
&leak : -valve\_closed, pressure\_loss.\\
&valve\_closed : -signal(1).\\
&pressure\_loss : -signal(2).\\
&overheat : -signal(3).\\
&alarm : -signal(X).\\
&signal(1).\\
&signal(2).
\end{aligned}
$$

Um programa definido mas não datalog:

$$
\begin{aligned}
&natural(0). && soma(0, X, X) : -natural(X).\\
&natural(s(X)) : -natural(X). && soma(s(X), Y, s(Z)) : -soma(X, Y, Z).
\end{aligned}
$$

Datalog (database logic)

Os programas datalog estão relacionados com as bases de dados, podendo ser utilizados para expressar regras e consultas a bases de dados dedutivas.

Os predicados num programa datalog normalmente classificam-se em:

♢ Predicados extensionais – formados apenas por factos contendo a a informação das relações da base de dados.

♢ Predicados intensionais – definidos através de 1 ou mais regras datalog.

♢ Predicados aritméticos – predicados cuja cuja interpretação se encontra previamente fixada e inalterável.

Uma base de dados de filmes

Considere-se o predicado extensional

$$movie(Title, Year, Length, InColor, StudioName)$$

com a seguinte informação:

```
movie('Star Wars' , 1977 , 124 , true, 'Fox').
movie('Mighty Ducks', 1991 , 104 , true, 'Disney').
movie('Wayne's World', 1992 , 95 , true, 'Paramount').
```

O predicado intensional

```
longMovie(T,Y) :- movie(T,Y,L,C,S), L >= 100.
```

corresponde à expressão de álgebra relacional

$$longMovie = \Pi_{Title,Year}\ \sigma_{Length>=100}\,(movie)$$

Tradução dos operadores de álgebra relacional

♢ Projecção $\Pi_{X_{i_1},\ldots,X_{i_k}}(r)$, com $k \leq m$

$$p(X_{i_1},\ldots,X_{i_k}) : -r(X_1,\ldots,X_m).$$

♢ Intersecção (caso $m = n$)

$$i(X_1,\ldots,X_m) : -r(X_1,\ldots,X_m), s(X_1,\ldots,X_m).$$

♢ União (caso $m = n$)

$$u(X_1,\ldots,X_m) : -r(X_1,\ldots,X_m).$$
$$u(X_1,\ldots,X_m) : -s(X_1,\ldots,X_m).$$

♢ Produto cartesiano

$$prod(X_1,\ldots,X_m,Y_1,\ldots,Y_n) : -r(X_1,\ldots,X_m), s(Y_1,\ldots,Y_n).$$

♢ Junção natural de $r/m+k$ com $s/k+n$:

$$j(X_1,\ldots,X_m,Z_1,\ldots,Z_k,Y_1,\ldots,Y_n) : -$$
$$r(X_1,\ldots,X_m,Z_1,\ldots,Z_k), s(Z_1,\ldots,Z_k,Y_1,\ldots,Y_n).$$

Tradução dos operadores de álgebra relacional

A operação de selecção é mais complicada, por causa do predicado de selecção.

A maneira mais simples consiste em passar o predicado de selecção para a forma normal disjuntiva (OR de ANDs). Por exemplo,

$$\sigma_{NOT\ (Length \geq 100\ AND\ StudioName = 'Fox')\ AND\ InColor = true}(movie)$$

Pode ser traduzido para

```
q(T,Y,L,I,S) :- movie(T,Y,L,I,S), L < 100, InColor='true'.
q(T,Y,L,I,S) :- movie(T,Y,L,I,S), S != 'Fox', InColor='true'.
```

E o operador de diferença ?

Consultas recursivas

A linguagem Datalog permite expressar consultas recursivas.

Seja o predicado extensional $sequelOf(X, Y)$ utilizado para indicar que o filme Y é uma sequela do filme X.

O predicado intensional $followOn(X, Y)$ indica que o filme Y segue o X:

```
followOn(X,Y) :- sequelOf(X,Y).
followOn(X,Y) :- sequelOf(X,Z), followOn(Z,Y).

sequelOf('Empire Strikes Back','Star Wars').
sequelOf('Return of Jedi','Empire Strikes Back').
```

A álgebra relacional não é suficientemente expressiva para representar este tipo de consultas!

Operador de Diferença

O operador de diferença da álgebra relacional necessita da negação por falha para ser traduzido correctamente (assumindo adicionalmente a hipótese do mundo fechado).

Semântica de Programas em Lógica Definidos

♢ Universo de Herbrand: conjunto de todos os termos formados por combinação das constantes e símbolos de função que ocorrem no programa.

Para o exemplo anterior: $\{0, s(0), s(s(0)), \ldots\}$.
Que aconteceria caso se juntasse o facto $xpto(1, g(0, X))$?

♢ Base de Herbrand: conjunto de todos os átomos básicos cujos argumentos são termos do Universo de Herbrand.

Para o exemplo:

$$\left\{ \begin{array}{l} natural(0), natural(s(0)), natural(s(s(0)), \ldots \\ soma(0,0,0), soma(0,0,s(0)), soma(0,s(0),0), \ldots, soma(s(0),s(0),s(0)), \\ soma(s(s(0)),0,0), soma(s(s(0)),s(0),0), \ldots, soma(s(s(0)),s(s(0)),s(s(0))), \\ \vdots \end{array} \right\}$$

Semântica de Programas em Lógica Definidos

♢ Programa instanciado $ground(P)$ é construído substituindo as variáveis por termos do Universo de Herbrand.

Para o exemplo, algumas das regras serão:

$natural(0).$
$natural(s(0)) : -natural(0).$
$natural(s(s(0)) : -natural(s(0)).$
$\vdots$

$soma(0,0,0) : -natural(0).$
$soma(0,s(0),s(0)) : -natural(s(0)).$
$soma(s(0),0,s(0)) : -soma(0,0,0).$
$soma(s(0),s(0),s(0)) : -soma(0,s(0),0).$
$\vdots$

♢ Interpretações são subconjuntos da Base de Herbrand. Uma interpretação M é modelo de um programa P sse

$$\forall_{A:-B_1,\ldots,B_m \in ground(P)} \text{ se } B_1,\ldots,B_m \in M \text{ então } A \in M$$

♢ Um programa em lógica definido tem sempre um modelo mínimo (único!).

Cálculo do modelo mínimo

♢ O modelo mínimo de um programa em lógica definido é obtido por iteração do operador de consequências imediatas T_P:

$$T_P(I) = \{A \mid A :- B_1, \ldots, B_m \in ground(P) \text{ tal que } B_1, \ldots, B_m \in I\}$$

♢ O operador T_P é monótono: se $I_1 \subseteq I_2$ então $T_P(I_1) \subseteq T_P(I_2)$.

♢ Tem-se ainda que M é modelo de um programa em lógica definido se e somente se $T_P(M) \subseteq M$.

♢ O modelo mínimo é dado por $least(P) = T_P \uparrow \omega$:

$$\begin{aligned} T_P \uparrow^0 &= \{\} \\ T_P \uparrow^{i+1} &= T_P(T_P \uparrow^i) \\ T_P \uparrow^\omega &= \cup_{0 \leq j < \omega} T_P \uparrow^j \end{aligned}$$

Programas em Lógica Normais

♢ Um programa em lógica normal é um conjunto de regras:

$$A : -B_1, \ldots, B_m, not\, C_1, \ldots, not\, C_n.$$

♢ A semântica do not da linguagem PROLOG é definida por mecanismos operacionais com uma série de problemas práticos e teóricos

♢ Dado um programa normal P completamente instanciado (ground), define-se a divisão de P por uma interpretação I como o programa definido:

$$\frac{P}{I} = \{A : -B_1, \ldots, B_m \mid A : -B_1, \ldots, B_m, not\, C_1, \ldots, not\, C_n \in P \text{ e } C_1, \ldots, C_n \notin I\}$$

♢ O operador de Gelfond-Lifschitz $\Gamma(I) = least(\frac{P}{I}) = T_{\frac{P}{I}} \uparrow^{\omega}$ obtém o modelo mínimo de Herbrand do programa dividido.

A interpretação M é um modelo estável sse $\Gamma(M) = M$

Exemplo

```
laugh :- joke.
joke.
hear_joke :- joke, not deaf.
understand_joke :- hear_joke, not stupid.
laugh :- understand_joke.
laugh :- stupid, not joke.
```

Qual o modelo estável deste programa?

As aves

O seguinte programa também só tem um modelo estável do qual

$$
\begin{array}{l}
voa(X) : -ave(X), not\, anormal(X). \\
ave(X) : -pinguim(X). \\
anormal(X) : -pinguim(X). \\
ave(tweety). \\
pinguim(joe).
\end{array}
$$

se conclui que $voa(tweety)$ e que $not\, voa(joe)$. Se juntarmos o facto $pinguim(tweety)$ deixaremos de concluir $voa(tweety)$.

Os cisnes outra vez

Quais os modelos estáveis para:

$$preto : -australiano, cisne, not\, branco.$$
$$branco : -cisne, not\, preto.$$
$$cisne.$$
$$australiano.$$

Uma regra de um programa em lógica normal pode ser traduzida para o default:

$$\frac{B_1 \wedge \ldots \wedge B_m \quad : \neg C_1, \ldots, \neg C_n}{A}$$

As extensões estão em correspondência 1-1 com os modelos estáveis.

Modelo bem-fundado

Considere-se o programa (não tendo modelos estáveis):

$$
\begin{aligned}
&vence(X) : -jogada(X, Y), not\, vence(Y).\\
&jogada(a, b).\\
&jogada(b, c).\\
&jogada(b, f).\\
&jogada(c, d).\\
&jogada(c, e).\\
&jogada(d, d).\\
&jogada(e, c).
\end{aligned}
$$

O par $\langle T, F\rangle$ é um modelo estável parcial (a 3 valores) sse

$$T = \Gamma\Gamma(T) \text{ e } F = H_P - \Gamma(T)$$

O menor deles pode ser obtido iterando o operador $\Gamma\Gamma$ a partir de $\{\}$. Esse é designado por modelo bem-fundado e pode ser obtido em tempo polinomial.

Cálculo do Modelo Bem-fundado

Considerem-se apenas os átomos da forma vence/1. O modelo bem fundado pode ser obtido iterando-se $\Gamma\Gamma$ a partir de $\{\}$. Esquematicamente:

n	$\Gamma^{2n}(\{\})$ (Verdadeiro)	$\Gamma\left(\Gamma^{2n}(\{\})\right)$ (Não falso)
0	$\{\}$	$\{v(a), v(b), v(c), v(d), v(e)\}$
1	$\{v(b)\}$	$\{v(b), v(c), v(d), v(e)\}$
2	$\{v(b)\}$	$\{v(b), v(c), v(d), v(e)\}$

Como temos $\Gamma^4\{\} = \Gamma^2\{\}$ terminamos o cálculo.

$\diamondsuit$ b é vencedora pois $v(b)$ pertence a $\Gamma^4\{\}$: $v(b)$ é verdadeiro.

$\diamondsuit$ a é perdedora pois $v(a)$ **não** pertence a $\Gamma(\Gamma^4\{\})$: $not\, v(a)$ é verdadeiro.

$\diamondsuit$ As posições c, d, e estão empatados pois, por exemplo, $v(c)$ pertence a $\Gamma(\Gamma^4\{\})$ mas não a $\Gamma^4\{\}$: $v(c)$ e $not\, v(c)$ estão indefinidos.

O que acontece se juntarmos $jogada(c, f)$?

Comportamento das semânticas

Para programas datalog:

♢ PROLOG não termina para ciclos positivos e negativos.

♢ Semântica de modelos estáveis e modelo bem-fundado coincidem para programas estratificados. Em particular, todos os ciclos positivos (sem negação) são falsos.

♢ Na semântica de modelo bem-fundado átomos envolvidos em ciclos positivos são falsos, enquanto que átomos em ciclos negativos são todos indefinidos.

♢ Na semântica de modelos estáveis átomos envolvidos em ciclos positivos são falsos, enquanto que átomos em ciclos negativos pares geram modelos alternativos. Contudo, a existência de ciclos ímpares negativos resulta na inexistência de modelos estáveis.

Complexidade da progr. em lógica proposicional

Complexidade para responder a $P \models A$

	complexidade
definidos	P-complete
não recursivos estratificados	P-complete
estratificados	P-complete
Bem-fundado	P-complete
Estável	coNP-complete

A existência de modelos estáveis é um problema NP-completo

Complexidade datalog

Definem-se várias medidas de complexidade para o raciocínio com programas em lógica datalog

♢ data complexity – fixa-se o programa, varia-se a base de dados (predicados extensionais) e a consulta.

♢ program complexity – fixa-se a base de dados, varia-se a base de dados intensional (programa) e a consulta.

♢ combined complexity – quer os predicados intensionais, quer os predicados extensionais e a consulta fazem parte do input.

Pode-se demonstrar que para o caso datalog a "combined complexity" coincide com a "program complexity". A noção de "combined complexity" corresponde àquela do transparente anterior.

Complexidade datalog

	data complexity	(combined) complexity
definidos	P-complete	EXPTIME-complete
não recursivos estratificados	polinomial	PSPACE-complete
estratificados	P-complete	EXPTIME-complete
Bem-fundado	P-complete	EXPTIME-complete
Estável	coNP-complete	co-NEXPTIME

Complexidade da progr. em lógica

	complexidade
definidos (sem recursão)	NEXPTIME-complete
definidos (com recursão)	RE-complete
estratificados (com n estratos)	Σ^0_{n+1}-complete
Bem-fundado	Π^1_1-complete
Estável	Π^1_1-complete

Para ver o significado da notação ver por exemplo:

- `http://en.wikipedia.org/wiki/Recursively_enumerable`
- `http://en.wikipedia.org/wiki/Arithmetical_hierarchy`
- `http://en.wikipedia.org/wiki/Analytical_hierarchy`