Semântica de Programação em Lógica

José Júlio Alferes

Dep. Matemática, Universidade de Évora
7000 Évora, Portugal

Escola de Verão
Fundamentos Matemáticos da Computação

Coimbra Setembro 1997

Introdução

A Programação em Lógica (LP) trouxe para a Informática o importante conceito de programação *declarativa* – por oposição à tradicional programação procedimental. Idealmente, o programador deveria apenas preocupar-se com o significado declarativo do seu programa. Os aspectos procedimentais da execução seriam tratados automaticamente.

Dada a sua natureza declarativa, rapidamente se tornou num candidato para uso em representação do conhecimento. A sua adequação tornou-se mais notória nos anos 80, quando foram estabelecidas pontes com as bases de dados dedutivas.

Para se ter a tal especificação declarativa, a cada programa tem que ser associado um significado (semântica) preciso.

A definição de tal semântica tem sido reconhecido como um dos mais importantes e difíceis problemas em LP.

Programa

- Motivação
- Linguagem, interpretações e modelos.
- Semântica de programas normais:
  - Modelos mínimos
  - Completação de Clark
  - Programas estratificados e Modelos perfeitos
  - Modelos estáveis
  - Semântica bem-fundada.

Programa (cont.)

- Raciocínio Não Monótono e Programação em Lógica
  - Default Logic
  - Auto-epistemic Logics
  - Auto-epistemic Logics of beliefs
- Motivação para negação explícita
- Semântica de programas estendidos:
  - Answer-sets
  - Semântica bem-fundada com negação explícita

Linguagem

Um programa em lógica normal é um conjunto finito de regras da forma:

$H \leftarrow A_1, \ldots, A_n, not\ B_1, \ldots, not\ B_m \quad (n, m \geq 0)$

onde H, A_i, e B_j são átomos.

Literais $not\ B_j$ são chamados de *default literals*.

Um programa normal P é dito *definido* se nenhuma das suas regras tem default literals.

A base de Herbrand $\mathcal{H}$ de um programa P é o conjunto de todos os átomos instanciados de P.

Consideraremos que um programa é um conjunto, possivelmente infinito, de regras instanciadas.

Programação declarativa

LP permite que o programa seja a especificação do problema:

$$
\begin{array}{l}
member(X, [X|Y]) \\
member(X, [Y|L]) \leftarrow member(X, L)
\end{array}
$$

Mais fácil e rápido de desenvolver.

O código é mais compacto.

Sendo executável, facilita na construção de protótipos.

Tendo uma implementação eficiente, porque não usar para programar.

BD dedutivas

Numa base de dados, as tabelas podem ser vistas como factos:

voa	de	$para$
	$lisboa$	$porto$
	$lisboa$	$londres$
...	...	...

$\Rightarrow$

$voa(lisboa, porto)$
$voa(lisboa, londres)$
...

Outras relações podem ser descritas como regras de LP:

$$\begin{aligned} ligacao(A,B) &\leftarrow voa(A,B) \\ ligacao(A,B) &\leftarrow voa(A,C), ligacao(C,B) \end{aligned}$$

$$outraComp(A,B) \leftarrow not\ ligacao(A,B)$$

LP permite ter BDs que para além das relações, armazenam formas de deduzir novas relações (Bases de Dados Dedutivas).

Notar que not não é negação clássica.

Regras supletivas

Em representação do conhecimento é necessário ter regras supletivas (por "default") da forma:

Todas as aves voam

A não-monotonicadade de not permite a LP representar estas regras:

$$\begin{aligned} voa(A) &\leftarrow ave(A), not\ anormal(A) \\ ave(P) &\leftarrow pinguim(P) \\ anormal(P) &\leftarrow pinguim(P) \end{aligned}$$

$$\begin{aligned} ave(t) \\ pinguim(p) \end{aligned}$$

Necessidade de Semântica

Em todos estes casos é necessária uma semântica bem definida, e diferente da lógica clássica.

A lógica clássica:

- no $1^{\underline{o}}$ caso não deriva $not\ member(3, [1, 2])$;
- no $2^{\underline{o}}$ caso nunca permite conclui que se deve escolher outra companhia;
- no $3^{\underline{o}}$ caso exigia que se qualificassem todas as excepções.

A definição de tal semântica tem sido reconhecido como um dos mais importantes e dificeis problemas em LP. Segue de perto as áreas de bases de dados dedutivas, e de representação do conhecimento.

Interpretações

Uma interpretação a 2-valores I é um subconjunto de $\mathcal{H}$.

- A é *verdadeiro* em I ($I(A) = 1$) se $A \in I$.
- Caso contrário, A é *falso* ($I(A) = 0$).

Vendo interpretações como "mundos possíveis" que representam estados possíveis do conhecimento, e podendo este ser incompleto, é necessário um conceito de interpretação onde alguns átomos não são verdadeiros nem falsos.

Uma interpretação a 3-valores I é um conjunto $T \cup not\ F$, onde T e F são subconjuntos dijuntos de $\mathcal{H}$.

- A é *verdadeiro* em I se $A \in T$.
- A é *falso* se $A \in F$.
- Caso contrário A é *indefinido* ($I(A) = \frac{1}{2}$).

As interpretações a 2-valores são um caso especial das a 3-valores, onde $\mathcal{H} = T \cup F$.

Modelos

Podem-se definir modelos, recorrendo a uma função de avaliação $\hat{I}$ associada a cada interpretação I.

- Se A é um átomo, então $\hat{I}(A) = I(A)$.
- Sendo S uma formula, $\hat{I}(not\ S) = 1 - \hat{I}(S)$.
- Sendo S e V formulas
  - $\hat{I}((S, V)) = min(\hat{I}(S), \hat{I}(V))$.
  - $\hat{I}(V \leftarrow S) = 1$ se $\hat{I}(S) \leq \hat{I}(V)$, e 0 caso contrário.

Uma interpretação I diz-se um modelo de P sse, para toda a regra de P, $\hat{I}(H \leftarrow B) = 1$.

Modelos Mínimos

A primeira tentativa de dar semântica a LPs deve-se a [EK76]. A ideia é que se deve minimizar a informação positiva, limitando-a ao que é implicado pelo programa. Tudo o resto será falso.

$$\begin{aligned} able\_mathematician(X) &\leftarrow physicist(X) \\ physicist(einstein) & \\ president(sampaio) & \end{aligned}$$

$\{pr(s), pr(e), ph(s), ph(e), a\_m(s), a\_m(e)\}$ é um modelo.

A falta de informação sobre se Sampaio é físico, deveria indicar que não o é.

O modelo mínimo é $\{pr(s), ph(e), a\_m(e)\}$.

A existência de um único modelo mínimo, para programas definidos, levou à definição da *semântica de modelo mínimo* para esses programas.

Semântica de Modelo Mínimo

Def. 1 (Ordem clássica) *Uma interpretação I é menor que uma J ($I \leq J$) sse para todo o átomo A, $I(A) \leq I(J)$, i.e. $T_I \subseteq T_J$ e $F_I \supseteq F_J$.*

Def. 2 (T_P) *Seja I uma interpretação e P um programa definido.*

$$T_P(I) = \{H : H \leftarrow Body \in P,\ and\ Body \subseteq I\}$$

Th. 1 *Sendo P um programa definido, T_P é monótono e continuo. Assim sendo, tem um ponto fixo mínimo, que se pode construir:*

$$\begin{aligned} I_0 &= \{\} \\ I_n &= T_P(I_{n-1}) \end{aligned}$$

O ponto fixo mínimo de T_P é o modelo mínimo (wrt ordem clássica) de P.

Def. 3 (Sem. Modelo Mínimo) *Um átomo A é verdadeiro em P sse A pertence ao modelo mínimo de P, i.e. $A \in T_P^{\uparrow\omega}$, i.e. A pertence a todos os modelos de P. Caso contrário A é falso.*

Procedimentos de prova

Nota: Apresenta-se apenas a ideia de base, e bastante simplificada (em particular não considera a necessidade de unificação para programas não instanciados). Não é objectivo deste curso entrar pelas matérias de métodos de derivação.

Derivações SLD

Para provar L, há que escolher uma regra $L \leftarrow L_1, \ldots L_n$ e provar $L_1, \ldots L_n$.
Caso haja um facto para L (i.e. $L \leftarrow true$) L é provado.

Em Prolog é escolhida a primeira regra para L. Caso não seja possível provar o seu corpo, é escolhida a regra seguinte em alternativa (backtracking).
Se não houver mais regras então L não se prova, $not\ L$ é verdadeiro (diz-se que L falha).
Para provar o corpo de uma regra, provam-se os seus átomos, da esquerda para a direita.

Completação de Clark

A semântica de modelo mínimo não se aplica a programas com not :
$\{p \leftarrow not\ q\}$ tem dois modelos minimais, $\{p\}$ e $\{q\}$. Não existe modelo mínimo.

Ideia [Clark78]: Usa-se *"se"* para dizer *"sse"*.

$$\begin{array}{rcl} natural\_n(0) & & \\ natural\_n(succ(X)) & \leftarrow & natural\_n(X) \end{array}$$

Este programa não implica que -1 não é natural (nem que uma letra não é um nº natural!). O que se quer dizer é:

$$n\_n(X) \Leftrightarrow \\ (X = 0 \vee (\exists Y \mid X = succ(Y) \wedge n\_n(Y)))$$

Com base nesta ideia, Clark definiu a completação de um programa P, sendo a semântica de P determinada pelos modelos a 2-valores da sua completação.

Def. Completação

Def. 4 (Completação de P) *É a teoria obtida de P por aplicação sucessiva de:*

1. *Transformar $p(t) \leftarrow L$ em $p(X) \leftarrow X = t \wedge L$.*

2. *Transformar $p(X) \leftarrow F$ em $p(X) \leftarrow \exists Y \ F$, sendo Y as variáveis originais da regra.*

3. *Sejam $p(X) \leftarrow F_1; \ \ldots; p(X) \leftarrow F_n$ todas as regras com $p(X)$ na cabeça. Substitui-las por $p(X) \leftarrow F_1 \vee \ldots \vee F_n$.*

4. *Para todo o predicado q que não ocorre na cabeça de nenhuma regra, adicionar $q(X) \leftarrow \bot$.*

5. *Transformar toda a regra obtida $p(X) \leftarrow F$ em $\forall X(p(X) \leftrightarrow F)$.*

Semântica de completação

Seja $Comp(P)$ a teoria obtida da completação do programa normal P, onde not é interpretado como negação clássica. Pela semântica de completação de Clark:

- A é *verdadeiro* em P sse $Comp(P) \models A$.
- A é *falso* em P sse $Comp(P) \models not\ A$.

Clark mostrou que, com a semântica de completação, not corresponde a "negação por falha finita", i.e. o método SLDNF (SLD com Negação por Falha) é correcto relativamente à semântica de completação. SLDNF é o método usado nas implementações de Prolog.

A semântica de completação tem assim uma implementação eficaz. Embora tenha um grande aparato a nível teórico, a ideia de base da semântica é simples. Define uma inferência não clássica que é coerente e completa relativamente à consequência clássica de uma teoria derivada do programa.

Mas...

Ideia de SLDNF

No método de derivação SLDNF procede-se da mesma forma que no método SLD, sempre que se trate da prova de átomos.

Para provar um literal $not\ L$, começa-se por tentar provar L (numa derivação subsidiaria).

Se se conseguir provar L, então $not\ L$ falha (i.e. é falso - não se consegue provar).

Se, após um nº finito de passos, se conclui que L falha, então $not\ L$ está provado.

Considere:

$$\begin{array}{lcl} p & \leftarrow & p \\ a & \leftarrow & not\ p \\ b & \leftarrow & not\ c \end{array}$$

Qualquer tentativa de provar c falha. Logo consegue-se provar $not\ c$ e b.

Embora não exista prova para p, não se conclui que p falha após um nº finito de passos. Logo não se prova $not\ p$ nem a.

Problemas da completação

- Pode tornar inconsistentes programas consistentes:
  A completação de $\{p \leftarrow not\ p\}$ é $\{p \Leftrightarrow not\ p\}$.
  O 1º é consistente e o 2º não.

- Não trata correctamente fechos transitivos:

$$\begin{array}{rcl} edge(a,b) \quad edge(c,d) \quad edge(d,c) & & \\ reachable(a) & & \\ reachable(X) & \leftarrow & reachable(Y), edge(X,Y) \end{array}$$

  Com completação não se conclui que c e d não se alcançam a partir de a. Aqui a dificuldade resulta de haverem regras simétricas para $edge(c,d)$ e $edge(d,c)$.

- Tem problemas quando P tem equivalências:

$$\begin{array}{rcl} bird(tweety) & & \\ fly(X) & \leftarrow & bird(X), not\ abnormal(X) \\ abnormal(X) & \leftarrow & irregular(X) \\ irregular(X) & \leftarrow & abnormal(X) \end{array}$$

  Da completação não se segue que tweety voa.

  Uma explicação seria dizer que as 2 últimas regras originam um loop.
  Mas esta explicação é procedimental!
  Como a ideia era ter uma programação declarativa, esta explicação tem que ser rejeitada.

Programas estratificados

Seguiram-se tentativas de definir semânticas que resolvessem os problemas da completação, mesmo que para isso tivessem que restringir a classe de programas a que se podiam aplicar. O caso mais notável é o de semânticas para programas estratificados.

A ideia foi voltar aos modelos mínimos, que não tinham os "problemas com loops". Mas só se aplicava a programas definidos. Como fazer?

Dividir o programas em estratos:

- num 1º estrato não haveria not s nos corpos, e a semântica seria a do modelo mínimo;

- em estratos seguintes eliminavam-se os literais definidos em estratos anteriores com base nos resultados desses estratos, e calculava-se o modelo mínimo do restante.

Exemplo de estratificação

$$P : \begin{cases} P_1 : \begin{cases} p \leftarrow p \\ a \leftarrow b \\ b \end{cases} \\ P_2 : \begin{cases} c \leftarrow not\ p \\ d \leftarrow c, not\ a \end{cases} \\ P_3 : \begin{cases} e \leftarrow a, not\ d \\ f \leftarrow not\ c \end{cases} \end{cases}$$

O modelo mínimo de P_1 é $\{a, b, not\ p\}$.

Sabendo isto, pode-se "processar" esta informação em P_2, que fica:

$$\begin{array}{rcl} c & \leftarrow & true \\ d & \leftarrow & c, false \end{array}$$

O seu modelo mínimo é (junto com o de P_1):

$$\{a, b, c, not\ d, not\ p\}$$

Fazendo o mesmo para P_3, obtem-se a semântica de P:

$$\{a, b, c, not\ d, e, not\ f, not\ p\}$$

Estratificação

Def. 5 (Estratificação) *Seja $\mathcal{H}$ a base de Herbrand de P. Seja $S_1; \ldots; S_n$, tal que os S_is são dijuntos, $S_1 \cup \ldots \cup S_n = \mathcal{H}$ e para toda a regra de P:*

$$A \leftarrow B_1, \ldots, B_m, not\ C_1, \ldots, not\ C_k$$

se $A \in S_i$ então:

- $\{B_1, \ldots, B_m\} \subseteq \bigcup_{j=1}^{i} S_j$ *e*
- $\{C_1, \ldots, C_k\} \subseteq \bigcup_{j=1}^{i-1} S_j$

Seja P_i o programa contendo todas as regras de P cuja cabeça pertence a S_i.

$P_1; \ldots; P_n$ é uma estratificação de P.

Estratificação (cont)

Um programa pode ter várias estratificações:

$$P : \left\{ \begin{array}{l} P_1 : \left\{ \begin{array}{l} a \\ b \leftarrow a \end{array} \right. \\ P_2 : \left\{ \begin{array}{l} c \leftarrow not\ a \end{array} \right. \end{array} \right. \qquad P_1 : \left\{ \begin{array}{l} a \\ b \leftarrow a \end{array} \right. \quad P_2 : \left\{ \begin{array}{l} c \leftarrow not\ a \end{array} \right.$$

Ou nenhuma:

$$\begin{array}{l} a \leftarrow not\ b \\ b \leftarrow not\ a \end{array}$$

Def. 6 (Programa Estratificado) *Sse admite alguma estratificação.*

Semântica de LPs Estr.

Seja $I|R$ a restrição da interpretação I aos átomos de R.
Seja $P_1; \ldots; P_n$ uma estratificação de P (originada por $S_1; \ldots; S_n$).
Defina-se a sequência de interpretações:

- $M_1 = least(P_1)$.

- M_{i+1} é o menor modelo de P_{i+1} tal que:

$$M_{i+1}|\left(\bigcup_{j=1}^{i} S_j\right) = M_i$$

M_n diz-se o modelo "standard" de P.

A é verdadeiro em P sse $A \in M_n$. Caso contrário A é falso.

Propriedades

Seja M_P o modelo "standard" do programa estratificado P.

- M_P é único, i.e. havendo várias, M_P não depende da estratificação de P.
- M_P é um modelo minimal de P.
- M_P é um modelo suportado de P.

Def. 7 (Modelo suportado) *Um modelo M de P diz-se suportado sse:*

$$A \in M \quad \Rightarrow \quad \exists A \leftarrow Body \in P \mid Body \subseteq M$$

I.e. os átomos do modelo "não caiem do céu".

Modelos perfeitos

Em rigor, a definição (de Apt, Blair e Walker) de programas estratificados era feita em programas não instanciados e referia-se a nomes de predicados e não a átomos. Assim, se $p(a)$ estava num estrato, $p(b)$ estava necessariamente no mesmo estrato.

A definição que se apresenta aqui é mais geral e, a menos de se restringir a um nº finito de estratos, é equivalente à de *programa localmente estratificado* de Przymusinski.

Retirando a restrição de finitude dos estratos obtem-se a semântica de Modelos Perfeitos. As propriedades apresentadas mantêm-se, alargando o escôpo de aplicação.

O programa:

$$\begin{array}{rcl} par(0) & & \\ par(s(X)) & \leftarrow & not\ par(X) \end{array}$$

não é estratificado pois $par/1$ depende negativamente de $par/1$. Mas é localmente estratificado:

$$\begin{array}{lcl} P_1 & = & \{par(0)\}; \\ P_2 & = & \{par(1) \leftarrow not\ par(0)\}; \\ P_3 & = & \{par(2) \leftarrow not\ par(1)\}; \\ \ldots & & \end{array}$$

O seu modelo perfeito é:

$$M_P = \{par(0), not\ par(1), par(2), not\ par(3), \ldots\}$$

Problemas com estratificação

A semântica de modelos perfeito é aceite como a adequada a programas estratificados. Todas as semânticas então definidas são generalizações da de modelos perfeitos (coincidem em programas estratificados).

Mas há programas uteis que não são estratificados:

$$\begin{array}{ll} par(X) \leftarrow zero(X) & zero(0) \\ par(Y) \leftarrow suc(X,Y), not\ par(X) & suc(X, s(X)) \end{array}$$

Esta pequena variação do programa anterior não é estratificado pois, e.g:

$$par(0) \leftarrow suc(0,0), not\ par(0) \ \in \ P$$

i.e. $par(0)$ depende negativamente de $par(0)$.

Não é estratificado um programa que contenha:

$$\begin{array}{l} pacifista(X) \leftarrow not\ hawk(X) \\ hawk(X) \leftarrow not\ pacifista(X) \end{array}$$

Este tipo de construção é bastante usado em representação de conhecimento.

Significa que "X é pacifista se não se puder assumir que é hawk" e vice-versa. Se não se disser mais nada sobre X, deve ficar indefinido se X é hawk ou pacifista.

Sobre procedimentos

Com modelos perfeitos not pode ser visto como negação por falha (mesmo que infinita).

Está definido um método teórico para modelos perfeitos (SLS), que se baseia em noções de falha possivelmente infinita.

Não podem haver implementações completas de SLS (como detectar um falha infinita, no caso geral??). Mas há aproximações correctas:

- Baseadas em detecção de loops entre atomos numa derivação. Se se está a tentar provar A num contexto em que A já foi chamado, A deve falhar.

- Baseadas em técnicas de tabulação. Para além da árvore de prova existe uma tabela onde se vão guardando literais já provados.

  Salienta-se aqui a implementação XSB de Warren et al.

Modelos Estáveis

A ideia por detrás dos Modelos Estáveis de Gelfond e Lifschitz [GL88] vem dos formalismos de raciocínio não-monótono.

Se assumirmos verdadeiros alguns nots, e falsos todos os outros, seguem-se certas consequências.
Se as consequências corroboram completamente as *hipóteses* feitas, então elas formam um modelo estável.

A semântica é definida pela intersecção de todos os modelos estáveis.

$$\begin{array}{lcl} a & \leftarrow & not\ b \\ b & \leftarrow & not\ a \\ c & \leftarrow & a \\ c & \leftarrow & b \end{array} \qquad \begin{array}{lcl} p & \leftarrow & not\ q \\ q & \leftarrow & not\ r \\ r & \leftarrow & \end{array}$$

Os modelos estáveis são

$$\{a, not\ b, c, r, p, not\ q\}$$
$$\{not\ a, b, c, r, p, not\ q\}$$

Assim conclui-se que c, r e p são verdadeiros, e que q é falso.

Modelos estáveis (defs.)

Def. 8 (Operador Γ) *Seja P um LP e I uma interpretação a 2-valores. O programa $\frac{P}{I}$ é o obtido de P:*

- *apagando todas as regras contendo no corpo $not\ A$, em que $A \in I$;*
- *apagando todos os restantes default literals.*

Como $\frac{P}{I}$ é um programa definido, tem um modelo mínimo J. Define-se $\Gamma(I) = J$.

Def. 9 (Modelos estáveis) *M é um modelo estável de P sse $\Gamma(M) = M$.*

Pela semântica de modelos estáveis, um átomo A é verdadeiro sse pertence a todos os modelos estáveis de P; é falso sse $not\ A$ pertence a todos os modelos estáveis de P.

Propriedades

Seja P um programa normal.

- Todo o modelo estável de P é modelo minimal de P.
- Todo o modelo estável é suportado.
- Se P é localmente estratificado, então tem um único modelo estável que coincide com o modelo perfeito de P.

Comparando com a semântica de modelos perfeitos, a de modelos estáveis não depende de restrições sintáticas, e atribui significado a (alguns) programas não estratificados.

Ambos os programas atrás não estratificados têm modelos estáveis.

Importância de Mod. Estáveis

Esta semântica foi um marco importante no estudo do significado declarativo de LPs:

- Introduziu a noção de negação supletiva (por "default") por oposição à anterior negação por falha.
- Permitiu estabelecer pontes importantes entre LP e raciocínio não-monótono (NMR): Deu início à área de investigação de LP e NMR.
- Permitiu uma melhor compreensão e uso para LP em representação do conhecimento.

Por tudo isto esta semântica é, ainda hoje, uma das mais usadas. É defendida como a "melhor" semântica para LP por uma parte da comunidade científica.

Mas...

Cumulatividade e relevância

Vejamos agora algumas propriedade desejáveis para semânticas de LPs. (Estas propriedades vêm dos formalismos de NMR.)

Def. 10 (Cumulatividade) *Sem é cumulativa sse para todo P:*

$$se\ A \in Sem(P)\ e\ B \in Sem(P)\ entao\ B \in Sem(P \cup \{A\})$$

I.e. todo o átomo derivado pode ser adicionado ao programa como um facto (sem alterar o seu significado). Esta propriedade também é importante por razões de implementação: permite guardar lemas intermédios. Sem ela não é possível usar técnicas de tabulação.

Diz-se que A depende directamente de B se ocorre no corpo de alguma regra de B. Diz-se que A depende de B se depende directamente de B ou se existe C tal que A depende directamente de C e C depende de B.

Def. 11 (Relevância) *Sem é relevante sse para todo o programa P e átomo A:*

$$A \in Sem(P)\ sse\ A \in Sem(Rel(P))$$

onde $Rel(P) \subseteq P$ é formado por todas as regras para A e para todos os átomos dos quais A depende.

Só esta propriedade garante a execução (clássica) top-down da programação em lógica.

Problemas de Mod. Estáveis

- Não dão semântica a todos os programas: $p \leftarrow not\ p$ não tem modelos estáveis.

- É não-cumulativa e não-relevante:

$$\begin{array}{llll} a & \leftarrow & not\ b & \quad c \leftarrow not\ a \\ b & \leftarrow & not\ a & \quad c \leftarrow not\ c \end{array}$$

  O único ME é $\{c, b\}$.
  $P \cup \{c\}$ não deriva b.
  b não é obtido das regras de que depende.

- A sua computação é NP-completo.

- A semântica não é suportada:

$$\begin{array}{llll} a & \leftarrow & not\ b & \quad c \leftarrow a \\ b & \leftarrow & not\ a & \quad c \leftarrow b \end{array}$$

  Tem dois MEs $\{c, b\}$ e $\{c, a\}$. c não é suportado pois nem a nem b são verdadeiros.

A semântica de modelos perfeitos é cumulativa, relevante e a sua computação é polinomial.

Semântica bem-fundada

Definida por Van Gelder et al. [GRS90] é uma generalização para 3-valores dos modelos estáveis, resolve os problemas e dá semântica a *todos* os LPs

Observação: Há programas que não têm pontos fixos de Γ, mas têm sempre pontos fixos de Γ^2.

Seja $P = \{a \leftarrow not\ a\}$.
$\{\}$ não é estável pois $\Gamma(\{\}) = \{a\}$.
$\{a\}$ também não é pois $\Gamma(\{a\}) = \{\}$.
Mas $\Gamma^2(\{\}) = \{\}$ e $\Gamma^2(\{a\}) = \{a\}$.

Mod. estáveis parciais

Def. 12 (ME parcial) *Uma interpretação $T \cup not\ F$ a 3-valores diz-se um modelo estável parcial de P sse:*

- $T = \Gamma^2(T)$
- $T \subseteq \Gamma(T)$
- $F = \mathcal{H} - \Gamma(T)$.

A 2ª condição garante que nenhum átomo pode ser simultaneamente verdadeiro e falso (I.e. que $T \cup F = \{\}$).

$P = \{a \leftarrow not\ a\}$ tem um MEP $M = \{\}$, i.e. onde a é indefinido.

O programa da pág. 34 tem 3 MPEs:

$$\begin{array}{lcl} M_1 & = & \{\} \\ M_2 & = & \{a, not\ b\} \\ M_3 & = & \{c, b, not\ a\} \end{array}$$

M_3 corresponde ao único modelo etável.

Semântica bem-fundada

Th. 2 (Modelo bem-fundado) *Todo o P tem um ME parcial mínimo M_P (wrt $\subseteq$), que pode ser obtido pela sequência transfinita de conjuntos de átomos:*

- $S_0 = \{\}$;
- $S_{i+1} = \Gamma^2(S_i)$;
- $S_\delta = \bigcup_{\alpha<\delta} S_\alpha$ *para ordinais limite* δ.

Sendo T o ponto fixo mínimo obtido pela sequência:

$$M_P = T \cup not\ (\mathcal{H} - \Gamma T)$$

A M_P chama-se o modelo bem-fundado (WFM) de P e determina a semântica bem-fundada (WFS).

Notar que aqui mínimo **não** se refere à ordem clássica.

Propriedades

- Dá semântica a todos os programas.

- Todo o MPE estende um ME. Logo se o WFM de P é total, coincide com o único ME de P.

- Se A é verdadeiro (resp. falso) no WFM, então é verdadeiro (resp. falso) pela semântica de ME.

- A WFS coincide com a de modelos perfeitos em programas localmente estratificados, e com a de modelo mínimo em definidos.

Mais propriedades

- O WFM é suportado
- A WFS é cumulativa e relevante.
- A computação do WFM é polinomial (no nº de regras instanciadas do programa).
- Existem procedimentos de prova top-down, e implementações coerentes.

Por tudo isto, a WFS é cada vez mais
a semântica de programas normais.

Procedimentos para WFS

Por a WFS não ter problemas com loops, têm que haver um mecanismo de detecção: por detecção em "ancestors" (não é geral); ou por tabulação.

Por ser a 3-valores, deixa de haver relação directa entre sucesso e verdade (e falha e falsidade). **Soluções:**

SLX (Alferes et al.) Definir 2 formas de derivação: para veracidade (T) e para não falsidade (TU). Para provar $not\ A$ numa prova T (resp. TU) há que falhar A numa prova TU (resp T).

XSB (Warren et al.) Associar etiquetas de verdade, falsidade e indefinição aos literais.

Ambos os sistemas se encontram implementados, embora apenas o último o esteja a baixo nível (recorrendo à definição de uma nova WAM).

Não-Monotonicidade

A lógica clássica é monotónica:

$$Se\ T \models A\ entao\ T \cup B \models A$$

(Tudo o que se deriva agora é válido para sempre.)

É adequado para uso em matemática, mas não para representação do conhecimento e raciocínio de senso comum.

- Lidamos com informação incompleta, mas avançamos para conclusões. Nova informação pode invalidar conclusões anteriores.

- Não permite regras supletivas. Dado o conhecimento ser incompleto, usamos essas regras:

  $$passaro(X) \Rightarrow voa(X)$$

  E se é um pinguim? Ou um recem nascido? ...

  $$\ldots \wedge \neg pinguim(X) \wedge rn(X) \wedge passaro(X) \Rightarrow voa(X)$$

  Será possível determinar essa lista de excepções? ("qualification problem"). E mesmo se fosse? Seria prático representar assim? Nós representamos assim?

Lógicas Não-Monotónicas

Para representação do conhecimento desenvolveram-se lógicas não monotónicas.

Veremos:

- Default Logic
- Auto-epistemic Logics
- Auto-epistemic Logics of beliefs

Todas as semânticas que vimos são não-monotónicas.

A programação em lógica pode ser vista como uma formalismo para raciocínio não-monotónico. Estudamos a sua relação com os outros.

Default Logic

Para além das fórmulas da lógica clássica, para lidar com informação incompleta tem regras supletivas:

$$\frac{\phi : \psi}{\gamma}$$

onde ϕ, ψ e γ são fórmulas clássicas.

O significado é: *Se ϕ é verdade, e for consistente assumir ψ, então γ*

$\frac{passaro(X):voa(X)}{voa(X)}$	Se X é pássaro, e for consistente assumir que X voa, então X voa
$\frac{:\neg voo(A,B)}{\neg voo(A,B)}$	Se for consistente assumir que não há voo entre A e B, então é porque não há mesmo.

Extensão de default

Def. 13 (Teoria de Default) *é um par* (D, W) *onde* W *é um conjunto de fórmulas proposicionais e* D *é um conjunto de regras supletivas.*

Def. 14 (Operador de Reiter) *Seja* $\Delta = (D, W)$ *uma teoria e* E *um conjunto de literais.* $\Gamma_\Delta(E)$ *é o menor conjunto que:*

- *contem todas as consequências de* W*;*
- *é fechado sobre todas as regras* $\frac{\phi}{\gamma}$*, onde* $\frac{\phi:\psi}{\gamma} \in D$ *e* $W \cup E \not\models \neg\psi$*.*

Def. 15 (Default extension) *É um conjunto de literais* E *tal que* $\Gamma_\Delta(E) = E$

LP e defaults

Seja Δ_P a teoria obtida de P transformando as regras:

$$H \leftarrow B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m$$

nos defaults:

$$\frac{B_1, \ldots, B_n \ : \ \neg C_1, \ldots, \neg C_m}{H}$$

e onde $W = \{\}$.

Th. 3 *Existe uma correspondencia de 1 para 1 entre os modelos estáveis de P e as extensões de Δ_P.*

Th. 4 *Um literal L é consequência de todas as extensões de Δ_P sse pertence à semântica de modelos estáveis de P.*

Th. 5 *Se L pertence ao WFM de P então é consequência de todas as extensões de Δ_P.*

LP como defaults

Os programas em lógica podem ser vistos como conjuntos de regras supletivas.

A negação por default pode ser vista como hipóteses que se podem assumir se tal for consistente (agora).

Se depois se adicionar mais informação que a torne inconsistente, então passa a não se assumir (não-monotonicidade).

$$voa(A) \leftarrow ave(A), \; not\ anormal(A) \qquad \frac{ave(A) \ : \ \neg anormal(A)}{voa(A)}$$

$$ave(P) \leftarrow pinguim(P) \qquad \frac{pinguim(P)}{ave(P)}$$

$$anormal(P) \leftarrow pinguim(P) \qquad \frac{pinguim(P)}{anormal(P)}$$

$$ave(t)$$

Como é consistente assumir $\neg anormal(t)$, assume-se e conclui-se $voa(t)$.

Se depois adicionar $pinguim(t)$, deixa de se concluir $voa(t)$.

Auto-epistemic logic

Adiciona à lógica clássica o operador modal $\mathcal{L}$ de conhecimento. A linguagem permite falar sobre o seu próprio conhecimento.

O significa de, e.g.,

$$\mathcal{L}\, A \wedge B \wedge \neg \mathcal{L}\, C \Rightarrow D$$

é: *Se sei que A é verdade, B é verdade e não sei se C é verdade, então D.*

$passaro(X) \wedge \neg \mathcal{L}\, \neg voa(X) \Rightarrow voa(X)$
Se X é pássaro, e não sei que X não voa, então X voa.

$\neg \mathcal{L}\, voo(A, B) \Rightarrow \neg voo(A, B)$
Se não sei se há voo entre A e B, então é porque não há mesmo.

AEL Expansion

Uma teoria auto-epistémica é uma teoria lógica com o operador modal $\mathcal{L}$.

Def. 16 (Expansão de Moore) *Uma teoria consistente T^* é uma expansão da teoria T sse:*

$$\begin{aligned} T^* \;=\; T \cup & \quad \{\mathcal{L}\, F : T^* \models F\} \\ & \cup \{\neg\mathcal{L}\, F : T^* \not\models F\} \end{aligned}$$

Informalmente:

- Sei aquilo que derivo (i.e. que está em todos os modelos).
- Não sei aquilo que não consigo derivar (i.e. que, podendo estar em algum dos meus modelos, não está em todos).

LP e AEL

Seja T_P a teoria AEL obtida de P transformando as regras:

$$H \leftarrow B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m$$

em:

$$B_1 \wedge \ldots \wedge B_n \wedge \neg\mathcal{L}\, C_1 \wedge \ldots \wedge \neg\mathcal{L}\, C_m \Rightarrow H$$

Th. 6 *Existe uma correspondencia de 1 para 1 entre os modelos estáveis de P e as expansões de T_P.*

Th. 7 *Um literal L é consequência de todas as expansões de T_P sse pertence à semântica de modelos estáveis de P.*

Th. 8 *Se L pertence ao WFM de P então é consequência de todas as expansões de T_P.*

LP e conhecimento

Os programas em lógica podem ser vistos como teorias que falam sobre o que elas próprias sabem (ou derivam).

A negação por default de A pode ser vista como não conhecimento de A.

Informação adicional pode permitir que se passe a saber A (não-monotonicidade).

$$\begin{array}{rcl} voa(A) & \leftarrow & ave(A), not\ anormal(A) \\ (ave(A) \wedge \neg \mathcal{L}\, anormal(A) \Rightarrow voa(A)) & & \end{array}$$

$$\begin{array}{rcl} ave(P) & \leftarrow & pinguim(P) \\ & & (pinguim(P) \Rightarrow ave(P)) \end{array}$$

$$\begin{array}{rcl} anormal(P) & \leftarrow & pinguim(P) \\ & & (pinguim(P) \Rightarrow anormal(P)) \\ ave(t) & & \end{array}$$

Como não sei $anormal(t)$ (há modelos com e modelos sem) conclui-se $voa(t)$.

Se depois adicionar $pinguim(t)$ passo a saber $anormal(t)$ e deixo de concluir $voa(t)$.

AEL of belief

Przymusinski definiu uma lógica auto-epistémica que em vez de *conhecimento* tem a noção mais frace de *crença*. Adiciona à lógica clássica o operador modal $\mathcal{B}$ de crença.

Uma teoria, que modela um agente racional, pode falar das crenças do próprio agente.

Para definição da lógica, assumem-se os axiomas:

- Consistencia: $\neg\mathcal{B}\,\bot$;
- Axioma K: $\mathcal{B}\,(F \Rightarrow G) \Rightarrow (\mathcal{B}\,F \Rightarrow \mathcal{B}\,G)$;

e que as teorias são fechadas sob:

$$\frac{F}{\mathcal{B}\,F}$$

Com o axioma (K), o da consistencia fica equivalente a:

$$\mathcal{B}\,F \Rightarrow \neg\mathcal{B}\,\neg F$$

Static expansions

Def. 17 (Modelos minimais) *Um modelo M é minimal se não exite nenhum menor M'* que coincida com os beliefs *de M.*
Se F pertence a todos o modelos minimais de T, $T \models_{min} F$.

Def. 18 (Static expansion) *Uma teoria consistente T^* é uma expansão da teoria T sse:*

$$T^* = T \cup \{\mathcal{B}\, F : T^* \models_{min} F\}$$

Acredito no que pertence a todos os meus modelos preferidos (i.e. minimais).

Versus: Sei o que está em todos os meus modelos, preferidos ou não.

LP e AEB

Seja T_P a teoria AEB obtida de P transformando as regras:

$$H \leftarrow B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m$$

em:

$$B_1 \wedge \ldots \wedge B_n \wedge \mathcal{B}\,\neg C_1 \wedge \ldots \wedge \mathcal{B}\,\neg C_m \Rightarrow H$$

Th. 9 *Existe uma correspondencia de 1 para 1 entre os modelos estáveis parciais de P e as expansões estáticas de T_P.*

Th. 10 *Um literal L é consequência de todas as expansões estáticas de T_P sse pertence ao WFM de P.*

Na WFS, a negação por default de A pode ser vista como a crença de que A é falso.

LP estendida

Em programas normais a informação negativa está implicita. Não é possível explicitar falsidade. Embora em certos casos se pretenda isso (e.g. o da base de dados de voos) nem sempre é assim.

A negação explícita é importante para representação do conhecimento:

"Os pinguins **não** voam" pode ser representado por $naoVoa(X) \leftarrow pinguim(X)$.
Mas isto não liga $naoVoa(X)$ à propriedade de voar.

$$\begin{aligned} voa(X) &\leftarrow passaro(X) \\ naoVoa(X) &\leftarrow pinguim(X) \end{aligned}$$

Não há ligação entre voa e $naoVoa$.

A introdução de $\neg$ em LP estabelece tais ligações e torna a representação do conhecimento mais clara.

LP estendida(cont)

• $\neg$ também é necessária em corpos:
"O réu é acusado se **não** *estiver inocente*

Se representar por $acusado(X) \leftarrow not\ inocente(X)$
é-se acusado se não houver prova de que se é inocente.
O que se quer é: $acusado(X) \leftarrow \neg inocente(X)$.

A diferença entre $not\ p$ e $\neg p$ é essencial sempre que não se pode assumir que a informação sobre p é completa, i.e. que não se pode assumir que a falta de informação de p indica a sua falsidade.

• $\neg$ permite maior expressividade:
"Se não se tem a certeza de que o réu não está inocente, deve-se investigar mais sobre ele
é representado por:
$investigar(X) \leftarrow not\ \neg inocente(X)$.

• $\neg$ alarga a relação com NMR.
E.g. permite regras supletivas com conclusões, pré-requisitos e justificações negativas.

Neg. explícita vs. clássica

- A negação clássica obedece ao princípio do 3º excluído, i.e. $F \vee \neg F$ é uma tautologia. Embora isto esteja correcto em lógica matemática, faz pouco sentido em raciocínio de senso comum.

  $\neg A$ representa o oposto de A (e.g. $\neg bom$ representa mau). O 3º excluído não deixa espaço para incertezas:

  $$\begin{aligned} contratar(X) &\leftarrow qualificado(X) \\ rejeitar(X) &\leftarrow \neg qualificado(X) \end{aligned}$$

  pelo 3º excluído, X é sempre contratado ou rejeitado. Não há margem para tratar aqueles de que é necessária mais informação.

- A negação clássica não é tratada de forma simétrica pela negação por default. E.g., sendo $\neg$ clássica,

  $$\begin{aligned} acusado(a) & \\ condenado(X) &\leftarrow culpado(X), acusado(X) \end{aligned}$$

  não deriva $condenado(a)$. Mas substituindo por:

  $$condenado(X) \leftarrow \neg inocente(X), acusado(X)$$

  já deriva (a negação clássica implica a por default).

- A negação clássica não é suportada.

Linguagem

Um programa em lógica estendida é um conjunto de regras da forma:

$L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n \ \ (n, m \geq 0)$

onde os L_is são literais objectivos.

Um literal objectivo é um átomo A ou a sua negação explícita $\neg A$.

Uma interpretação é um conjunto $T \cup not\ F$, onde T e F são conjuntos dijuntos de literais objectivos e:

$$\neg L \in T \ \Rightarrow \ L \in F$$

i.e. se L é explicitamente falso então tem de ser assumido falso (Princípio da coerência).

Uma interpretação é total sse $T \cup F$ é o conjunto de todos os literais objectivos.

Em interpretações totais, o princípio da coerencia é satisfeito trivialmente.

Answer-sets

Foi a $1^{\underline{a}}$ semântica para programas estendidos (Gelfond e Lifschitz, 90).

Generalizam a semântica de modelos estáveis para ELPs.

Um answer-set de P é um modelo estável consistente do programa normal obtido substituindo os literais $\neg A$ por um novo átomo $\bar{A}$. Um tal modelo estável é consistente se não contem $\{A, \bar{A}\}$ para algum átomo A.

Um átomo é verdadeiro num answer-set S se $A \in S$; é falso se $\neg A \in S$, e desconhecido caso contrário.

Passam a haver programas inconsistentes. E.g. $\{a \leftarrow \quad \neg a \leftarrow\}$.

Answer-sets e NMR

Seja Δ_P a teoria default obtida de P transformando as regras:

$$L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n$$

nos defaults:

$$\frac{L_1, \ldots, L_m\ :\ \neg L_{m+1}, \ldots, \neg L_n}{L_0}$$

onde $\neg\neg\neg A$ é substituido por A.

Seja T_P a teoria AEL obtida de P transformando as regras em:

$L_1 \wedge \mathcal{L}L_1 \wedge \ldots \wedge L_m \wedge \mathcal{L}L_m \wedge \neg\mathcal{L}L_{m+1} \wedge \ldots \wedge \neg\mathcal{L}L_n \Rightarrow (L_0 \wedge \mathcal{L}L_0)$

Th. 11 *Existe uma correspondencia de 1 para 1 entre os answer-sets de P e as extensões de Δ_P.*

Th. 12 *Existe uma correspondencia de 1 para 1 entre os answer-sets de P e as expansões de T_P.*

Princípio da coerência

Generalizar a WFS da mesma forma não dá resultados intuitivos:

Considere:

$$\begin{aligned} pacifista(X) &\leftarrow not\ hawk(X) \\ hawk(X) &\leftarrow not\ pacifista(X) \\ \neg pacifista(a) & \end{aligned}$$

Usando o mesmo método tem-se: $WFM = \{\neg pac(a)\}$. Não é imposta a coerência.

Em answer-sets é imposta, pois sendo interpretações consistentes a 2 valores:

$$\neg A \in S \rightsquigarrow A \notin S \rightsquigarrow not\ A$$

Há que impôr a coerência.

WFSX

Def. 19 (Versão semi-normal de P) *Obtem-se adicionando $not\ \neg L$ a toda a regra com cabeça L.*

Def. 20 (Modelo estável parcial) *Uma interpretação $T \cup not\ F$ de P onde:*

- $T = \Gamma\Gamma_s(T)$
- $T \subseteq \Gamma_s(T)$
- $F = \mathcal{H} - \Gamma_s(T)$.

sendo Γ_s, o resultado de aplicar Γ à versão semi-normal de P.

Def. 21 (WFSX) *A semântica bem-fundada com negação explícita é o resultado do menor modelo estável parcial.*

Propriedades WFSX

- Obedece ao princípio da coerência.
- Coincide com a de modelo mínimo em programas definidos, com a de modelos perfeitos em localmente estratificados e com a WFS em programas normais.
- Se a WFSX é total, então coincide com o único answer-set.
- Se A é verdadeiro (resp. falso) na WFSX, então é verdadeiro (resp. falso) pela semântica de answer-sets.
- É suportada, cumulativa e relevante.
- A computação da WFSX é polinomial (no nº de regras instanciadas do programa).
- Existem procedimentos de prova top-down, e implementações coerentes.

Quem quiser saber mais

Pode começar por:

- Sobre semântica de programas normais:
  H. Przymusinska and T. Przymusinski. Semantic Issues in Deductive Databases and Logic Programs. In R. Banerji ed., *Formal Techniques in AI, a Sourcebook*, pages 321-367. North Holland, 1990.

- Sobre ligação com procedimentos de derivação:
  K. Apt and R. Bol. Logic Programming and Negation: A Survey. In *Journal of Logic Programming*, **19-20**, 1994.

- Sobre o uso em representação do conhecimento:
  C. Baral and M. Gelfond. Logic Programming and Knowledge Representation. In *Journal of Logic Programming*, **19-20**, 1994.

- Sobre ligação com NMR, e propriedade estruturais:
  G Brewka, J. Dix, and K. Konolige. *Nonmonotonic reasoning: An overview*. CSLI Publications, 1997.

- Sobre negação explícita, sua relação com NMR, e aplicações:
  J. J. Alferes and L. M. Pereira. *Reasoning with Logic Programming*. Springer LNAI 1111, 1996.