Dopełnienie programu (K.Clark) (ang. program completion)

intuicyjnie: przekształcenie implikacji w równoważność

P = { A ← B, A ← C }

dwie możliwości:

A ↔ B ∨ C lub (A ↔ B) ∧ (A ↔ C)

niech '=' nowy binarny predykat nie występujący w P
zapis s ≠ t oznacza ¬ (s = t)
niech $x_1, \ldots, x_n, \ldots$ — nowe zmienne (nie występujące w P)

Definicja **IFF(P)**:

Kolejne kroki transformacji programu P:

Krok 1 (usunięcie termów z nagłówków klauzul).

Przekształć każdą klauzulę: $p(t_1, \ldots, t_n) \leftarrow B_1, \ldots, B_m$

$$p(x_1, \ldots, x_n) \leftarrow (x_1 = t_1) \wedge \ldots \wedge (x_n = t_n) \wedge B_1 \wedge \ldots \wedge B_m$$

Krok 2 (wprowadzenie kwantyfikatorów egzystencjalnych).

Zapisz każdą formułę: $p(x_1, \ldots, x_n) \leftarrow F$

$$p(x_1, \ldots, x_n) \leftarrow \exists\, y_1 \ldots \exists\, y_d\; F$$

gdzie $y_1, \ldots, y_d$ — (wszystkie) zmienne oryginalnej klauzuli

Krok 3 (grupowanie klauzul).

Niech $p(x_1, \ldots, x_n) \leftarrow E_1$

...

$p(x_1, \ldots, x_n) \leftarrow E_k$

wszystkie formuły, w których nagłówkach występuje p
(każda formuła E_i jest powyższej postaci).

Zastępujemy je formułą:

$$p(x_1, \ldots, x_n) \leftarrow E_1 \vee \ldots \vee E_k$$

Jeśli $E_1 \vee \ldots \vee E_k$ jest puste, to zastąp przez **true**.

Krok 4 (obsługa 'niezdefiniowanych' predykatów).

Dla każdego n-argumentowego predykatu q nie występującego w nagłówku żadnej klauzuli P dodaj formułę:

$$q(x_1, \ldots, x_n) \leftarrow \mathbf{false}$$

Krok 5 (wprowadzenie kwantyfikatorów ogólnych).

Zastąp każdą formułę $p(x_1, \ldots, x_n) \leftarrow E$ przez

$$\forall\, x_1 \ldots \forall\, x_n\, (p(x_1, \ldots, x_n) \leftarrow E)$$

Krok 6 (wprowadzenie równoważności).

W każdej formule zastąp $\leftarrow$ przez $\leftrightarrow$.

□

Przykład

loves(x, y) ← mother(x), child_of(y, x)
loves(john, tom) ←
loves(mary, john) ←

dopełnienie predykatu loves:

$$\begin{aligned}
&\forall\, v\, \forall\, z\, (loves(v, z) \leftrightarrow \\
&\quad \exists\, x\, \exists\, y\, (v = x \wedge z = y \wedge mother(x) \wedge child_of(y, x)) \\
&\quad \vee (v = john \wedge z = tom) \\
&\quad \vee (v = mary \wedge z = john))
\end{aligned}$$

Czy teraz zachodzi IFF(P) $\models \neg$ loves(john, mary)?
Tak, o ile '=' jest odpowiednio zdefiniowane (identyczność).

następująca definicja:

$\models_I^\sigma t_1 = t_2$ wiw gdy $t_1\sigma$ i $t_2\sigma$ są tymi samymi elementami dziedziny interpretacji I

jest niewystarczająca, gdyż np. różnym stałym można przypisać ten sam element dziedziny interpretacji

Teoria równości Clarka (CET)

(zał. stałe — symbole funkcyjne 0-argumentowe)

1. x = x
2. $f(x_1, \ldots, x_n) \neq g(y_1, \ldots, y_m)$
   dla każdych symboli funkcyjnych f i g takich, że f $\neq$ g
3. $x_1 \neq y_1 \vee \ldots \vee x_n \neq y_n \rightarrow f(x_1, \ldots, x_n) \neq f(y_1, \ldots, y_n)$
   dla każdego symbolu funkcyjnego f
4. $x_1 = y_1 \wedge \ldots \wedge x_n = y_n \rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$
   dla każdego symbolu funkcyjnego f
5. $x_1 = y_1 \wedge \ldots \wedge x_n = y_n \rightarrow (p(x_1, \ldots, x_n) \rightarrow p(y_1, \ldots, y_n))$
   dla każdego predykatu p (łącznie z '=')
6. $x \neq t$ dla każdej zmiennej x oraz termu t takiego, że $x \not\equiv t$ i x występuje w t

□

dopełnienie programu P, **comp(P)**:

$$\text{comp(P)} \stackrel{df}{=} \text{IFF(P)} \cup \text{CET}$$

Charakterystyka dopełnienia programów w logice

Twierdzenie

$$\text{comp(P)} \models \text{P}$$

Wniosek

Jeśli P $\models$ F, to comp(P) $\models$ F.

Twierdzenie
P — program, $A_1, \ldots, A_m$ — atomy
Jeśli $\forall\ (A_1 \wedge \ldots \wedge A_m)$ jest logiczną konsekwencją comp(P), to jest również logiczną konsekwencją programu P.

Wniosek
Dla każdego programu w logice P — pozytywne informacje, które można wyprowadzić z dopełnienia P oraz z programu P są identyczne.

Twierdzenie (K.R. Apt, M. van Emden)

Dla dowolnego programu w logice P zachodzi:

(1) comp(P) ma model Herbranda

(2) dla dowolnego ustalonego atomu A:
comp(P) $\cup$ { A } ma model Herbranda wiw gdy
A $\in$ gfp(T_P)

Wniosek
Istnieje program P i atom ustalony A taki, że zbiór
comp(P) $\cup$ { A } ma model, ale nie ma modelu Herbranda.

Charakterystyka reguły negacji przez porażkę dla programów w logice

Twierdzenie (**Poprawność reguły NF**)

P — program w logice
$\leftarrow A_1, \ldots, A_n$ — klauzula negatywna

Jeśli istnieje skończone SLD-drzewo porażki
dla P oraz $\leftarrow A_1, \ldots, A_n$, to
formuła $\forall\neg(A_1 \wedge \ldots \wedge A_n)$ jest logiczną konsekwencją dopełnienia programu P.

dowód:
indukcja względem głębokości SLD-drzewa, z wykorzystaniem aksjomatów teorii równości

Twierdzenie (**Pełność reguły NF**)

P — program w logice
$\leftarrow A_1, \ldots, A_n$ — klauzula negatywna

Jeśli $\forall\neg(A_1 \wedge \ldots \wedge A_n)$ jest logiczną konsekwencją dopełnienia programu P, to każde sprawiedliwe SLD-drzewo dla programu P oraz $\leftarrow A_1, \ldots, A_n$ jest skończonym SLD-drzewem porażki.

Programy z negacją (ogólne programy w logice)

klauzula programu z negacją (ogólna klauzula programu):

$$A \leftarrow L_1, \ldots, L_n$$

gdzie A – atom, L_i – literały

program z negacją (ogólny program w logice):
skończony, niepusty zbiór klauzul programu z negacją

ogólna klauzula negatywna: $\leftarrow L_1, \ldots, L_n$

Przykład
dwa zbiory są różne:

different(x, y) ← member(z, x), ¬ member (z, y)
different(x, y) ← ¬ member(z, x), member (z, y)

dla ogólnych programów w logice:
nadal baza Herbranda jest modelem
czyli niemożność wyprowadzania negatywnych informacji

Definicje

P — program z negacją, G: $\leftarrow L_1, \ldots, L_n$

Odpowiedź na P ∪ { G } — podstawienie zawierające dowiązania zmiennych występujących w G.

θ — odpowiedź na P ∪ { G }
θ jest **poprawną odpowiedzią** na comp(P) ∪ { G },
jeśli $\forall\,(L_1 \wedge \ldots \wedge L_n)\theta$ jest logiczną konsekwencją dopełnienia P

Własności dopełnienia programów z negacją

1. Każdy program z negacją jest niesprzeczny, ale jego dopełnienie może być sprzeczne.

P: $p \leftarrow \neg p$

IFF(P): $p \leftrightarrow \neg p$

2. Dopełnienie programu z negacją zależy nie tylko od jego „logicznej zawartości".

Np. $p \leftarrow \neg q$ $\qquad$ $q \leftarrow \neg p$

są logicznie równoważne ($p \vee q$) a ich dopełnienia **nie** są

$p \leftrightarrow \neg q$ $\qquad$ $q \leftrightarrow \neg p$

$\neg q$ $\qquad$ $\neg p$

Reguła NF dla programów z negacją — problemy

P: $p(x) \leftarrow \neg q(x)$

$q(a) \leftarrow$

$\leftarrow \neg p(x)$

$\leftarrow p(x)$

$\leftarrow \neg q(x)$ $\qquad$ $\leftarrow q(x)$

sukces $\qquad$ porażka $\qquad$ $\square$ $(\{x/a\})$

$\leftarrow p(x)$ odnosi porażkę, czyli mamy dowód dla $\leftarrow \neg p(x)$,

ale $\quad comp(P) \not\models \forall x\, \neg p(x)$

czyli reguła NF **nie** jest poprawna dla programów z negacją (!)

Warunek poprawności:

literał $\neg p(\vec{x})$ można wybierać tylko wtedy, gdy $\vec{x}$ jest ustalony reguła obliczeniowa jest **bezpieczna** (ang. safe), jeśli każdy wybrany literał negatywny jest ustalony

SLDNF-rezolucja

ograniczenie: tylko bezpieczne reguły obliczeniowe (konieczne w celu zapewnienia poprawności)

P — program z negacją,
G — ogólna klauzula negatywna,
$\mathcal{R}$ — bezpieczna reguła obliczeniowa

SLDNF-rezolucja = SLD-rezolucja + reguła NF

Twierdzenie (**Poprawność SLDNF-rezolucji**)

Każda odpowiedź obliczona dla $P \cup \{G\}$ jest poprawną odpowiedzią dla comp(P) $\cup \{G\}$.

inaczej:

1. Jeśli istnieje SLDNF-dowód dla P $\cup \{\leftarrow Q\}$ z obliczoną odpowiedzią θ, to

$$\text{comp(P)} \models Q\theta$$

2. Jeśli istnieje skończone SLDNF-drzewo porażki dla $P \cup \{\leftarrow Q\}$, to

$$\text{comp(P)} \models \neg\, Q$$

Programy warstwowe i hierarchiczne

poziom predykatu — odwzorowanie przyporządkowujące każdemu predykatowi nieujemną liczbę całkowitą

program P jest **warstwowy**, jeśli istnieje odwzorowanie spełniające warunek, że dla każdej klauzuli P:

– jeśli p występuje w pozytywnym literale, to poziom p jest mniejszy lub równy od poziomu predykatu nagłówka,
– jeśli p występuje w negatywnym literale, to poziom p jest mniejszy od poziomu predykatu nagłówka

program P jest **hierarchiczny**, jeśli istnieje odwzorowanie spełniające warunek, że dla każdej klauzuli P:

poziom predykatu występującego w treści klauzuli jest mniejszy od poziomu predykatu nagłówka

P — program warstwowy

podział programu na warstwy: P $= P_1 \cup \ldots \cup P_n$,

gdzie P_i zawiera definicje pewnych predykatów P

zachodzą dwa warunki:

– jeśli p występuje w pozytywnym literale klauzuli należącej do P_i, to jego definicja znajduje się w zbiorze $P_j,\ j \leq i$

– jeśli p występuje w negatywnym literale klauzuli należącej do P_i, to jego definicja znajduje się w zbiorze $P_j,\ j < i$

zbiory P_i — **warstwy** programu P (warstwy są rozłączne)

Przykłady

$$P1 = \{ \mathrm{p} \leftarrow, \quad \mathrm{q} \leftarrow \mathrm{p}, \mathrm{r}, \quad \mathrm{r} \leftarrow \neg\, \mathrm{q} \}$$

P1 nie jest warstwowy — zależności między q oraz r

$$P2 = \{ \mathrm{p} \leftarrow, \quad \mathrm{q} \leftarrow \mathrm{p}, \quad \mathrm{r} \leftarrow \neg\, \mathrm{q} \}$$

P2 jest programem warstwowym:

$$P2 = \{p \leftarrow\} \cup \{q \leftarrow p\} \cup \{r \leftarrow \neg q\}$$

jak również: $P2 = \{p \leftarrow, q \leftarrow p\} \cup \{r \leftarrow \neg q\}$

Większość programów to programy warstwowe, ale

- istnieją programy nie warstwowe, których dopełnienia nie są sprzeczne

  odd(x) ← ¬ even(x)
  even(x) ← ¬ odd(x)

(program poprawny, ale nic z niego nie można wywnioskować)

- (niestety) istnieją użyteczne programy nie warstwowe

  odd(s(x)) ← ¬ odd(x)

Twierdzenie

Dopełnienie każdego programu warstwowego posiada minimalny rozszerzony model Herbranda.

Kwestia pełności SLDNF-rezolucji

Przykłady

P: q ← ¬ r   r ← p
p ← p   r ← ¬ p

P — program warstwowy

¬ q jest logiczną konsekwencją comp(P),
lecz nie istnieje SLDNF-drzewo porażki dla P ∪ $\{\leftarrow q\}$

P: r ← p
r ← ¬ p
p ← p

poprawną odpowiedzią na comp(P) ∪ $\{\leftarrow r\}$ jest ϵ,
ale nie uzyskamy jej stosując SLDNF-rezolucję

Ograniczona pełność SLDNF-rezolucji
(dla programów hierarchicznych).

Inne podejścia: semantyka dopełnieniowa 3-wartościowa.

Podsumowanie

Programy (bez negacji)
— poprawność i pełność SLD-rezolucji
— wyprowadzanie wyłącznie pozytywnych informacji

Programy z negacją
— comp(P) może być sprzeczne
— brak modelu najmniejszego, modele minimalne

Wyprowadzanie negatywnych informacji:
reguła negacji przez (skończoną) porażkę (NF)

Programy (bez negacji)
– poprawność reguły NF
– pełność reguły NF (sprawiedliwość)

Programy z negacją
– poprawność reguły NF (bezpieczna reguła obliczeniowa)
– pełność reguły NF: w zasadzie brak

Reguła NF — tylko test,
tzn. dla klauzuli postaci $\leftarrow A_1, \ldots, A_n$
możemy sprawdzić, czy $\quad \mathrm{comp(P)} \models \forall \neg(A_1 \wedge \ldots \wedge A_n)$
ale nie możemy obliczyć θ t.że: $\quad \mathrm{comp(P)} \models \forall \neg(A_1 \wedge \ldots \wedge A_n)\theta$

Programy z negacją: SLDNF-rezolucja

– tylko bezpieczne reguły obliczeniowe
– poprawność SLDNF-rezolucji
– pełność SLDNF-rezolucji:
tylko dla programów hierarchicznych (warunki)