P — program w logice
G — klauzula negatywna: $\leftarrow A_1, \ldots, A_k$,
θ — podstawienie dla zmiennych z G

θ — **odpowiedź poprawna** na $P \cup \{G\}$, jeśli $\forall(A_1 \wedge \ldots \wedge A_k)\theta$ jest logiczną konsekwencją programu P

odpowiedź "Nie" jest poprawna, jeśli $P \cup \{G\}$ jest spełnialny ($\neg G$ nie jest logiczną konsekwencją P)

P — program w logice

Zbiór sukcesów programu P — zbiór wszystkich ustalonych formuł atomowych A takich, że dla $P \cup \{\leftarrow A\}$ istnieje SLD-dowód.

Wniosek
Zbiór sukcesów programu P jest podzbiorem najmniejszego modelu Herbranda tego programu (M_P).

dowód:
teza wynika z poprawności SLD-rezolucji.

Twierdzenie (K.Apt, M. van Emden)
(Słabe) twierdzenie o pełności SLD-rezolucji.

Zbiór sukcesów programu jest równy najmniejszemu modelowi Herbranda tego programu.

Twierdzenie (Robert F. Stärk, 1990)
Silne twierdzenie o pełności SLD-rezolucji i niezależności od funkcji wyboru.

P — program w logice, $\leftarrow Q$ — klauzula negatywna,
R — funkcja wyboru.

Jeśli

$$P \models \forall(Q\sigma),$$

to dla P $\cup$ $\{\leftarrow Q\}$ istnieje SLD-dowód uzyskany za pomocą funkcji R, odpowiedź obliczona θ oraz podstawienie γ takie, że

$$Q\sigma = Q\theta\gamma.$$

Wniosek.
Nie każda poprawna odpowiedź może być obliczona za pomocą SLD-rezolucji.

Przykład.

P: p(f(y)) $\leftarrow$ G: $\leftarrow$ p(x)
q(a) $\leftarrow$
q(b) $\leftarrow$

poprawne odpowiedzi: $\{x/f(a)\}, \{x/f(f(a))\}, \{x/f(b)\}$

jedyna odpowiedź obliczona: $\{x/f(y)\}$

SLD-rezolucja — najbardziej ogólny wynik obliczenia:

$$\forall y\ p(f(y))$$

Uwagi, uzupełnienia, wnioski

1. Pełność SLD-rezolucji

Nie każda poprawna odpowiedź może być obliczona za pomocą SLD-rezolucji.

Pytanie: czy każda odpowiedź obliczona jest bardziej ogólna od odpowiedzi poprawnej?

Odpowiedź: nie.

Przykład.

P: $\quad p(f(y, z)) \leftarrow$

niech $Q = p(x)$

a $\theta = \{x/f(a, a)\}$ — odpowiedź poprawna

(P $\models Q\theta$, czyli P $\models p(f(a, a)))$

jedyna odpowiedź obliczona to $\sigma = \{x/f(x1, x2)\}$

lecz podstawienie σ nie jest bardziej ogólne od θ, gdyż nie istnieje podstawienie γ takie, że $\theta = \sigma\gamma$

nie istnieje γ takie, że $\quad \{x/f(x1, x2)\}\,\gamma = \{x/f(a, a)\}$

Zachodzi:
θ — odpowiedź poprawna, σ — odpowiedź obliczona dla $\leftarrow Q$
istnieje γ takie, że: $\quad Q\theta = Q\sigma\gamma$

Np. $\qquad Q\theta = p(f(a, a)) \qquad Q\sigma = p(f(x1, x2))$

$$p(f(a, a)) = p(f(x1, x2))\,\{x1/a, x2/a\}$$

2. Poprawność SLD-rezolucji a *occur check*

Konsekwencje braku sprawdzania występowania zmiennej w termie.

Przykład 1.

P: test ← p(x, x)
p(x, f(x)) ←

odpowiedź obliczona dla P ∪ {← test } to ϵ

lecz P $\not\models$ test

Przykład 2.

P: f_constructed(f(t, y), t) ←
bizarre(x) ← f_constructed(x, x)
crazy(x) ← bizarre(f(y, x))

nie istnieją termy dziwaczne, a czy istnieją szalone termy?

SLD-dowód:

G_0: ← crazy(x)
G_1: ← bizarre(f(y_0, x_0))
G_2: ← f_constructed(f(y_0, x_0), f(y_0, x_0))
G_3: □ (bez *occur check*)

obliczona odpowiedź: { x/y_0 }, czyli wszystkie termy są szalone!

2. Reguła obliczeniowa, przeszukiwanie SLD-drzewa

Twierdzenie
Każde SLD-drzewo dla P ∪ {G} ma albo nieskończoną liczbę gałęzi sukcesu albo każde SLD-drzewo ma taką samą skończoną liczbę gałęzi sukcesu.

SLD-drzewo nieskończone →
— nieskończone przeszukiwanie drzewa,
— ew. brak znalezienia niektórych odpowiedzi,
— tylko strategia przeszukiwania wszerz jest uczciwa (każda gałąź sukcesu zostanie znaleziona)

Pytanie:
czy strategia w głąb przy stałym uporządkowaniu klauzul programu i dowolnej funkcji wyboru zapewnia znalezienie gałęzi sukcesu?

Odpowiedź: NIE!

Przykład.

P: (1) p(a, b) ← G: ← p(a, c)
(2) p(c, b) ←
(3) p(x, z) ← p(x, y), p(y, z)
(4) p(x, y) ← p(y, x)

istnieje SLD-dowód dla P ∪ {G} i wszystkie klauzule są niezbędne lecz nigdy nie uzyskamy tego dowodu

przyczyna: klauzule (3) i (4) zawsze można zastosować;
jeśli (3) jest wcześniej, to system nigdy nie użyje (4) i odwrotnie;

dla każdej funkcji wyboru i dowolnego ustalonego porządku klauzul — skrajnie lewa gałąź jest nieskończona

Podsumowanie

klauzule programu:

$A \leftarrow B_1, \ldots, B_n$ czyli $\forall\,(A \vee \neg B_1 \vee \ldots \vee \neg B_n)$

program w logice — skończony zbiór klauzul

klauzule negatywne:

$\leftarrow B_1, \ldots, B_n$ czyli $\neg\,\exists\,(B_1 \wedge \ldots \wedge B_n)$

sposoby interpretacji programów w logice

- deklaratywna — **znaczenie** programu, **co** program oblicza
- proceduralna (operacyjna) — **jak** program oblicza

pełność — zgodność tych dwóch interpretacji

semantyka deklaratywna		**semantyka proceduralna**
interpretacje, modele Herbranda		SLD-rezolucja
najmniejszy model Herbranda (zbiór log. konsekw. programu; dziedzina: uniwersum Herbranda)	$\longleftrightarrow$	zbiór sukcesów programu (zbiór ustalonych atomów mających SLD-dowód)
konstrukcja najmniejszego modelu Herbranda (operator T_P)		
poprawna odpowiedź (podstawienie t.że formuła jest logiczną konsekwencją programu)	$\longleftrightarrow$	obliczona odpowiedź (podst. uzgadniające (mgu) uzyskane z SLD-dowodu)

Sterowanie

- porządek klauzul (aksjomatów)
- porządek atomów w treści klauzuli (funkcja wyboru)

Teoria a praktyka

- brak sprawdzania occur check — utrata poprawności SLD-rezolucji
- strategia przeszukiwania w głąb — utrata pełności
- odcięcie — możliwość utraty pełności

Odcięcie

narzędzie do sterowania
bardzo kontrowersyjne lecz szeroko stosowane

założenie: reguła obliczeniowa — wybór lewego atomu

odcięcie — **!**
– pozalogiczna anotacja programu (sterowanie)
– atom, który zawsze skutkuje

semantyka programów z odcięciem nie zmienia się

znaczenie odcięcia: zmniejszenie przestrzeni przeszukiwań

SLD-drzewo,
rezolwenta — wybrany atom: klauzula programu zawiera odcięcie
$\Longrightarrow$ pominięcie części poddrzewa, której korzeniem jest ta rezolwenta

rodzaje odcięć
– bezpieczne — pominięte poddrzewo nie zawiera gałęzi sukcesu
– niebezpieczne — istnieje gałąź sukcesu, czyli utrata pełności
(ang. green cut, red cut)

Przykład

$A \leftarrow B, C$

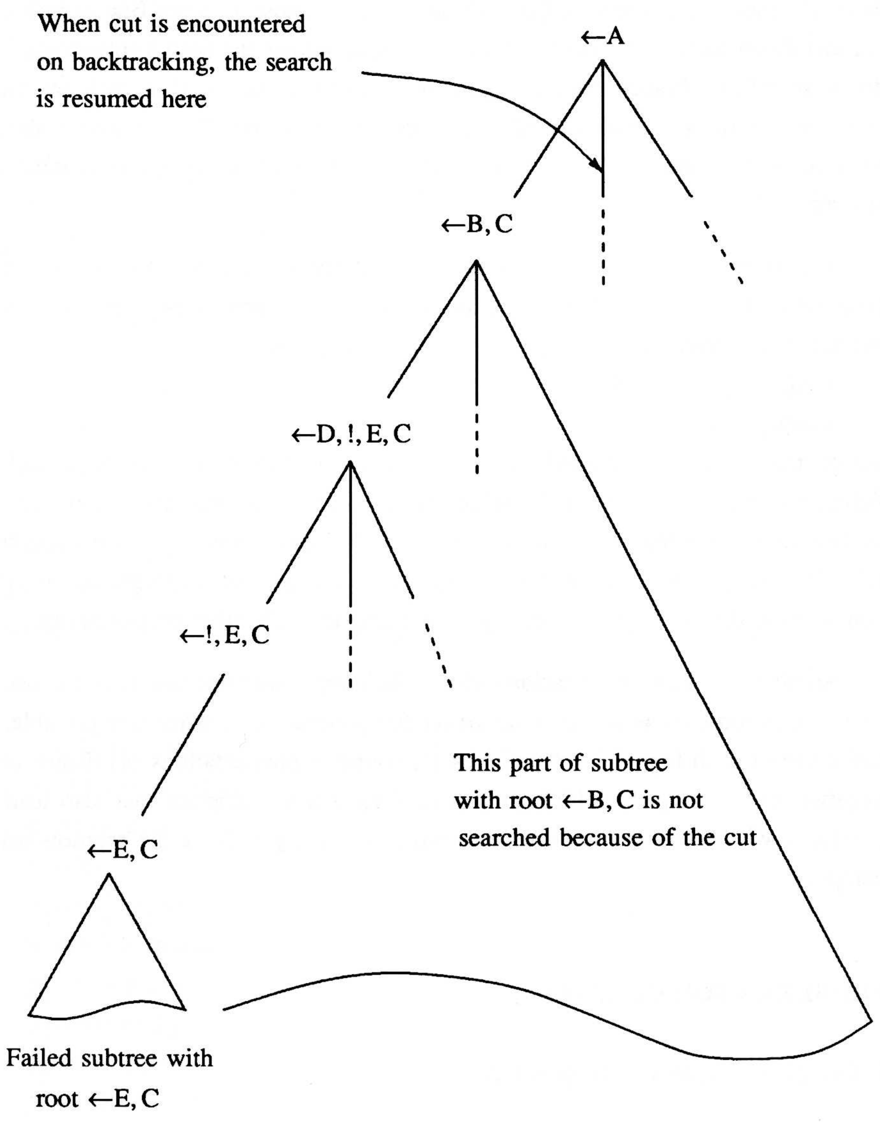
....

$B \leftarrow D, !, E$

....

$D \leftarrow$

....



(Rysunek pochodzi z książki Johna Lloyda ,,Foundations of Logic Programming".)

bezpieczne odcięcie — wzrost sprawności

niebezpieczne odcięcie — utrata pełności
(pominięcie odpowiedzi lub odpowiedź „Nie")

co gorsza: programy niepoprawne deklaratywnie

Przykład

$\max(x, y, y) \leftarrow x \leq y, !$
$\max(x, y, x) \leftarrow$

deklaratywnie — nie jest to relacja max

proceduralnie — uwzględniając strategię przeszukiwania, regułę obliczeniową i porządek klauzul: (prawie) max

czy prawdą jest, że $\leftarrow \max(3, 5, 3)$?
odpowiedź: TAK!

w takich sytuacjach:
— inne mechanizmy (np. konstrukcja if-then-else)
— przeniesienie uzgadniania z nagłówka do treści klauzuli
(np. $\max(x, y, z) \leftarrow x \leq y, !, z = y$)

Używanie odcięcia:
– nie używać, o ile nie jest to konieczne
– stosować rozsądnie (!)
– dodawać po napisaniu programu
– miejsce umieszczenia: bezpośrednio po stwierdzeniu wyboru właściwej klauzuli

Moc obliczeniowa programów w logice

Czy programy w logice mają tę samą moc obliczeniową co funkcje rekurencyjne?

Odpowiedź: TAK

Twierdzenie (Andréka, Németi)
Dla każdej rekurencyjnej funkcji f istnieje program P obliczający f przy użyciu relacji p_f.