Wykład monograficzny

Programowanie w języku logiki

mgr Mirosława Miłkowska

(e-mail: M.Milkowska@mimuw.edu.pl)

`http://rainbow.mimuw.edu.pl/LP-ZSI/`

Programowanie w logice — programowanie deklaratywne

Robert Kowalski (1972):

algorytm = logika + sterowanie

logika — definicja problemu (JAKI problem)
sterowanie — sposób rozwiązania (JAK)

oddzielenie logiki od sterowania:
- programista — specyfikacja logicznej części algorytmu
- system — dostarczenie mechanizmu sterowania

stan idealny:

czysto deklaratywne programowanie

twórcy – Robert Kowalski i Alain Colmerauer

1965 : Alan Robinson – zasada rezolucji

1972 : podstawowa idea – język logiki jako język programowania

1973 : powstanie języka Prolog (PROgramming in LOGic)
pierwszy interpreter Prologu, Marsylia (A. Colmerauer)

1977 : Edinburgh Prolog (D.H.D.Warren)

1986 : Constraint Logic Programming

Programowanie w logice a programowanie imperatywne

programowanie w logice – programowanie deklaratywne

program – opis problemu a nie sposób jego rozwiązania
CO JAK

	języki imperatywne	jęz. deklaratywne
paradygmat	JAK rozwiązać problem	JAKI problem
program	ciąg poleceń	zbiór zdań
przykłady	Pascal, C, Ada	Prolog, Haskell, ML
zalety	sprawne, wyspecjalizowane programy	ogólne, czytelne, poprawne programy

- program w języku logiki – aksjomatyzacja problemu
- obliczenie – dowód (twierdzenia, zapytania)

Dwa poziomy odczytania programu:

- deklaratywny – zbiór aksjomatów
- operacyjny – ciąg procedur

Te poziomy są (w znacznym stopniu) niezależne:
modyfikacja sterowania nie wpływa na poprawność programu.

Problem – zdefiniować relację bycia wnukiem

Opis deklaratywny:

> Wnuk x'a jest to dziecko jednego z dzieci x'a.

Opis imperatywny 1:

> Aby znaleźć wnuka x'a, znajdź najpierw dziecko x'a, a następnie dziecko tego dziecka.

Opis imperatywny 2:

> Aby znaleźć wnuka x'a, znajdź najpierw rodzica jakiemuś dziecku, a następnie sprawdź, czy ten rodzic jest dzieckiem x'a.

Program imperatywny

```
read(osoba);
for i:=1 to maxosoba do
  if rodzic[i] = osoba then   /* dziecko osoby */
    for j:=1 to maxosoba do
      if rodzic[j] = i then   /* mamy wnuka */
        writeln( nazwa[j] );
```

Formuła logiczna

$$\forall x\, \forall y\, (\, \exists z\; (dziecko(x,z)\ \wedge\ dziecko(z,y))\ \rightarrow\ wnuk(x,y))$$

Program w języku logiki

```
wnuk( X, Y ) :- dziecko( X, Z ),  dziecko( Z, Y ).
```

zapytania:

```
:- wnuk(jan, adam). % czy Jan jest wnukiem Adama?

:- wnuk(jan, X).    % czyim wnukiem jest Jan?

:- wnuk(X, adam).   % kto jest wnukiem Adama?

:- wnuk(X, Y).      % jakikolwiek wnuk/dziadek
```

Teorie pierwszego rzędu

Teoria pierwszego rzędu:

- alfabet
- zestaw formuł danej teorii
- zbiór aksjomatów (podzbiór zbioru formuł)
- zbiór reguł wnioskowania

Twierdzenia teorii – wyprowadzane na podstawie zbioru aksjomatów i reguł dowodzenia

Alfabet – klasy symboli:

- zmienne $x, y, z, \ldots$
- stałe $a, b, c, d, \ldots$
- symbole funkcyjne $f, g, \ldots$
- symbole predykatywne $p, q, r, \ldots$
- stałe logiczne $prawda, falsz$
- funktory zdaniotwórcze $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
- kwantyfikatory $\forall, \exists$
- znaki interpunkcyjne

Symbole logiczne:
funktory, kwantyfikatory, znaki interpunkcyjne, zmienne

Symbole pozalogiczne:
stałe, symbole funkcyjne i predykatywne
(jednoznacznie określają język pierwszego rzędu)

funkcje i predykaty – określona arność, tj. liczba argumentów
arność funkcji > 0, (stałe)
arność predykatów ≥ 0, (twierdzenia)

Język pierwszego rzędu

Term (definicja indukcyjna):
– zmienna jest termem
– stała jest termem
– jeśli f – n-argumentowy symbol funkcyjny,
$t_1, \ldots, t_n$ – termy, to $f(t_1, \ldots, t_n)$ jest termem

oznaczenia termów – s, t, u

Formuła:
– jeśli p – n-argumentowy predykat, $t_1, \ldots, t_n$ – termy, to $p(t_1, \ldots, t_n)$ jest formułą (formuła atomowa, atom)
– $prawda$ i $falsz$ są formułami
– jeśli F i G są formułami, to formułami są również:
$\neg F,\ F \wedge G,\ F \vee G,\ F \rightarrow G,\ F \leftrightarrow G$
– jeśli F jest formułą, a x jest zmienną, to formułami są:
$\forall x F, \quad \exists x F$

Język pierwszego rzędu – alfabet oraz zbiór wszystkich formuł utworzonych z symboli tego alfabetu

Oznaczenia notacyjne:

- zamiast $F \to G$ piszemy $G \leftarrow F$
- A, B — formuły atomowe, F, G — formuły
- F – formuła zawierająca zmienne wolne $x_1, \ldots, x_n$
  $\forall F$ oznacza $\forall x_1 \ldots \forall x_n F$

Klauzule

literał – atom lub negacja atomu
literał pozytywny (atom) oraz negatywny (negacja atomu)

Klauzula — formuła postaci $\forall (L_1 \vee \ldots \vee L_m)$,
gdzie L_i są literałami $(i = 1, \ldots, m)$

Przykład klauzuli: $\forall x \forall y \, (p(x, y) \vee q(f(x), h(y), a))$

Klauzula postaci: $\forall (A_1 \vee \ldots \vee A_k \vee \neg B_1 \vee \ldots \vee \neg B_n)$

jest równoważna: $\forall ((A_1 \vee \ldots \vee A_k) \vee \neg (B_1 \wedge \ldots \wedge B_n))$

jest równoważna: $\forall (B_1 \wedge \ldots \wedge B_n \to A_1 \vee \ldots \vee A_k)$

zapisujemy: $\underbrace{A_1, \ldots, A_k}_{\text{wnioski}} \leftarrow \underbrace{B_1, \ldots, B_n}_{\text{przesłanki}}$

przecinki w przesłance (poprzednik) – oznaczają koniunkcje,
przecinki we wniosku (następnik) – oznaczają alternatywy

Przykład:
formuła: $\forall x \forall y \, (p(x) \vee \neg A \vee \neg q(y) \vee B)$
postać klauzulowa: $p(x),\ B \leftarrow A,\ q(y)$

Szczególne przypadki klauzul:

- $A \leftarrow B_1, \ldots, B_n$ ($k = 1$, klauzula programu)
  (ang. definite clause)
- $A \leftarrow$ ($n = 0$, klauzula unarna)
- $\leftarrow B_1, \ldots, B_n$ ($k = 0$, klauzula negatywna, rezolwenta, zapytanie, ang. goal)
- $\Box$ (alternatywa pusta – fałsz, sprzeczność)

Programy w języku logiki

klauzula programu: $A \leftarrow B_1, \ldots, B_n$

$$\forall (A \vee \neg B_1 \vee \ldots \vee \neg B_n)$$

A – **nagłówek** klauzuli, $B_1, \ldots, B_n$ – **treść** klauzuli programu

definicja predykatu p/n:
zbiór wszystkich klauzul, w których nagłówku występuje p/n

Program (ang. definite clause program)
skończony (niepusty) zbiór klauzul programu

Program z negacją (ang. normal program)

klauzula programu $A \leftarrow L_1, \ldots, L_m$

L_i — literały ($i = 1, \ldots, m \quad m \geq 0$)

klauzula negatywna $\leftarrow L_1, \ldots, L_m \quad (m \geq 1)$

Klauzule Horna

klauzula zawierająca co najwyżej jeden literał pozytywny
inaczej
klauzula programu lub klauzula negatywna

Programy w języku logiki składają się wyłącznie z klauzul Horna

Interpretacja klauzul programu

$$A \leftarrow B_1, \ldots, B_n$$

- interpretacja deklaratywna
  A jest prawdziwe o ile $B_1, \ldots, B_n$ są prawdziwe
- interpretacja proceduralna (operacyjna)
  aby rozwiązać A — rozwiąż $B_1, \ldots, B_n$

interpretacja proceduralna – odróżnienie programowania w logice od logiki pierwszego rzędu

Przykład

Sortowanie ciągu liczb całkowitych nieujemnych.

Reprezentacja liczb: stała 0, symbol funkcyjny s (następnik)
$s(0), s(s(0)), \ldots$

Reprezentacja ciągu:
lista — binarny symbol funkcyjny '.', stała *nil*

Np. 17.(22.(6.(5.nil))) lub prościej 17.22.6.5.nil

Program **slowsort**

sort(x, y) ← perm(x, y), sorted(y)
sorted(nil) ←
sorted(x.nil) ←
sorted(x.y.z) ← x ≤ y, sorted(y.z)
perm(nil, nil) ←
perm(x.y, u.v) ← delete(u, x.y, z), perm(z, v)
delete(x, x.y, y) ←
delete(x, y.z, y.w) ← delete(x, z, w)
0 ≤ x ←
s(x) ≤ s(y) ← x ≤ y

Program zawiera pięć predykatów: sort, sorted, perm, delete, ≤

Nieformalne znaczenie predykatu sort:

jeśli x, y są listami i y jest permutacją listy x oraz
y jest listą posortowaną, to y jest posortowaną wersją listy x

klauzula negatywna (zapytanie): $\leftarrow sort(17.22.6.5.nil, y)$

równoważne $\forall y \neg sort(17.22.6.5.nil, y)$,

czyli: $\neg \exists y \, sort(17.22.6.5.nil, y)$

żądanie znalezienia listy y – dowód (konstruktywny) przez zaprzeczenie, że istnieje lista y o podanych właściwościach

Program wykładu

- teorie pierwszego rzędu, klauzule, programy w języku logiki
- algorytm uzgadniania (podstawienia, właściwości)
- reguły wnioskowania, SLD-rezolucja
- semantyka programów w logice (interpretacje, modele; interpretacje i modele Herbranda)
- poprawność i pełność SLD-rezolucji
- wyprowadzanie negatywnych informacji (reguły niemonotoniczne)
- programy z negacją, dopełnienie programu
- SLDNF-rezolucja
- różne semantyki programów z negacją
- przegląd dziedzin programowania w logice
  - programowanie w logice z więzami (CLP)
  - równoległość
  - deklaratywna lokalizacja błędów
  - transformacje programów w logice,
    synteza programów w logice
  - metaprogramowanie
- język programowania Prolog (inne języki programowania w logice, np. Gödel)

Literatura

1. Wprowadzenie do programowania w logice
   - K. R. Apt — From Logic Programming to Prolog. 1997.
   - J. W. Lloyd — Foundations of Logic Programming. Springer, Berlin, wyd. 2, 1987.
   - U. Nilsson, J. Małuszyński — Logic, Programming and Prolog. John Wiley & Sons, 1990.
     (`http://www.ida.liu.se/~ulfni/lpp/`)
   - K. R. Apt — Logic Programming. W: J. van Leeuwen, ed. Handbook of Theoretical Computer Science. 1990, vol. B.
   - K. Doets — From Logic to Logic Programming. MIT, 1994.

2. Podręczniki Prologu
   - F. Kluźniak, S. Szpakowicz — Prolog for Programmers. Academic Press, 1985.
   - L. Sterling, E. Shapiro — The Art of Prolog. MIT, wyd. 2, 1994.
   - W. F. Clocksin, C. S. Mellish — Programming in Prolog. Springer, 1994.
     (Prolog. Programowanie. Wyd. Helion, 2003).
   - R. O'Keefe — The Craft of Prolog. MIT, 1990.
   - D. Maier, D.S. Warren — Computing with Logic. Logic Programming with Prolog. Benjumin/Cummings, 1988.