Języki i Paradygmaty Programowania

Prolog i programowanie w logice 1

Marcin Benke *

24 kwietnia 2017

1 Programowanie w logice

Prolog

- PROgrammation en LOGique.
- Colmerauer, Roussel; Marseille 1971.
- Metoda rezolucji: Robinson 1965, Kowalski 1971.
- Pierwsza implementacja: 1972 w ALGOL-W.
- Zaprojektowany dla przetwarzania języka naturalnego; wciąż uzywany w tej i innych dziedzinach AI
- Język deklaratywny
  - program opisuje *jak ma wyglądać rozwiązanie* a nie jak je uzyskać.

Prolog dzisiaj

- NASA 2005 — Clarissa: interfejs głosowy Międzynarodowej Stacji Kosmicznej
- IBM 2011 — Projekt Watson
- Erlang (na bazie Prologu i początkowo napisany w Prologu)
- InFlow —analiza sieci społecznościowych
- Gerrit — serwer git ze wsparciem dla przeglądów kodu

*Cześciowo w oparciu o notatki prof. Urzyczyna

Programowanie deklaratywne

Robert Kowalski (1972):

algorytm = logika + sterowanie

Programowanie imperatywne

- instrukcje opisujace kolejne kroki algorytmu

Programowanie deklaratywne

- opis problemu (cech pożądanego rozwiązania)

- funkcyjne — równania (funkcje)
- w logice — formuły logiczne (relacje)

`http://goo.gl/EPNFJ`

Wnioskowanie — fakty i implikacje

Fakt: $0 \in N$.

Reguła: $x \in N \Rightarrow S(x) \in N$.

$even(0)$. $even(x) \Rightarrow even(S(S(x)))$.

Istnieje $x \in N$ takie, że $even(S(x))$?

NB w Prologu zmienne z wielkiej litery, stałe z małej

```
nat(0).
nat(s(X)) :- nat(X).
even(0).
even(s(s(X)) :- even(X).

?- nat(X), even(s(X)).
```

Wnioskowanie w Prologu

Program — opis relacji w logice predykatów:

```
nat(0).
nat(s(X)) :- nat(X).
even(0).
even(s(s(X)) :- even(X).
```

Zapytanie: czy istnieje $x \in N$ takie, że $even(S(x))$?

```
?- nat(X), even(s(X)).
```

Odpowiedzi:

```
X = s(z) ? ;
X = s(s(s(z))) ? ;
X = s(s(s(s(s(z))))) ?
...
```

Opis świata przy pomocy termów i relacji

```
tree(empty).
tree(node(L,_,R)) :- tree(L),tree(R).

isoTree(empty,empty).
isoTree(node(L1,X,R1),node(L2,X,R2))
   :- isoTree(L1,L2), isoTree(R1,R2).
isoTree(node(L1,X,R1),node(L2,X,R2))
   :- isoTree(L1,R2), isoTree(L2,R1).
```

`isoTree` opisuje *relację* (zbiór par termów).

```
?- isoTree(node(node(empty,1,empty),2,empty),X).
X = node(node(empty, 1, empty), 2, empty) ;
X = node(node(empty, 1, empty), 2, empty) ;
X = node(empty, 2, node(empty, 1, empty)) ;
X = node(empty, 2, node(empty, 1, empty)).
```

2 Logika pierwszego rzędu

Logika — sygnatury i struktury

Sygnatura to pewien (zwykle skończony) zbiór symboli relacyjnych i funkcyjnych, każdy z ustaloną liczbą argumentów (arnością).

$$\Sigma = \bigcup_{n \in N} \Sigma_n \cup \bigcup_{n \in N} \Sigma_n^R$$

Fakt, że $f \in \Sigma_n$, zapisujemy czasem przez $ar(f) = n$.

W programowaniu w logice przyjęty jest także zapis `f/n`, oznaczający funkcję lub relację arności n.

Termy

Przez *termy* sygnatury Σ rozumiemy najmniejszy zbiór napisów T_Σ spełniający warunki:

- $V \subseteq T_\Sigma$; (zmienne)
- jeśli $f \in \Sigma_n$ oraz $t_1, \ldots, t_n \in T_\Sigma$ to "$f(t_1, \ldots, t_n)$" $\in T_\Sigma$.

$FV(t)$ — zbiór zmiennych w termie t

Jesli $FV(t) = \emptyset$, to t jest *termem ustalonym* (ground term).

Przykład

Wyrażenie "$f(g(g(x_2, f(c)), x_1))$" jest termem sygnatury Σ,

gdzie $g \in \Sigma_2$, $f \in \Sigma_1$ oraz $c \in \Sigma_0$.

$$FV("f(g(g(x_2, f(c)), x_1)") = \{x_1, x_2\}$$

$g(c, f(c))$ jest termem ustalonym

Modele

Model sygnatury Σ to niepusty zbiór (nośnik) wraz z interpretacją symboli jako funkcji i relacji o odpowiedniej liczbie argumentów:

$$\mathcal{A} = \langle A, f_1^{\mathcal{A}}, \dots, f_n^{\mathcal{A}}, r_1^{\mathcal{A}}, \dots, r_m^{\mathcal{A}} \rangle,$$

gdzie dla dowolnego i:

- $f_i^{\mathcal{A}} : A^k \to A$, jeśli $ar(f_i) = k$ (w szczególności $f_i^{\mathcal{A}} \in A$, gdy $k = 0$);
- $r_i^{\mathcal{A}} \subseteq A^k$, gdy $ar(r_i) = k$.

Model termowy

nośnikiem jest zbiór termów ustalonych nad Σ,

symbole funkcyjne są interpretowane jako funkcje budujące termy:

$$f^A("t_1", \dots, "t_n") = "f(t_1, \dots t_n)"$$

W programowaniu w logice ograniczamy się do modeli termowych.

Formuły atomowe

Formuły atomowe (atomy) sygnatury Σ:

- symbol "$\perp$" (na oznaczenie fałszu);
- napisy postaci "$r(t_1, \dots, t_n)$", gdzie $r \in \Sigma_n^R$ oraz $t_1, \dots, t_n \in T_\Sigma$;
- napisy postaci "$t_1 = t_2$", gdzie $t_1, t_2 \in T_\Sigma$.

Formuły logiki predykatów
Zbiór formuł $\mathcal{F}_\Sigma$:

- formuły atomowe należą do $\mathcal{F}_\Sigma$;
- jeśli $\varphi, \psi \in \mathcal{F}_\Sigma$ to także $(\varphi \to \psi)$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $\neg\psi \in \mathcal{F}_\Sigma$, ;
- jeśli $\varphi \in \mathcal{F}_\Sigma$ i $x \in V$ to także $(\forall x.\varphi) \in \mathcal{F}_\Sigma$, $(\exists x.\varphi) \in \mathcal{F}_\Sigma$.

$FV(\varphi)$ — zbiór zmiennych wolnych (nie związanych kwantyfikatorem) w formule φ

$$FV(\exists x.g(x, y) = c) = \{y\}$$

Wartościowanie zmiennych
Niech Σ będzie sygnaturą, zaś $\mathcal{A}$ jej modelem.
Wartościowanie (zmiennych) to funkcja $v : V \to A$.

Przez v_x^a oznaczamy wartościowanie określone tak:

$$v_x^a(y) = \begin{cases} a, & \text{gdy } y = x; \\ v(y), & \text{wpp.} \end{cases}$$

Znaczenie formuł
Wartość formuły φ przy wartościowaniu v:

- $v(\perp) = 0$;
- $v(r(t_1, \ldots, t_n)) = 1$, gdy $\langle v(t_1) \ldots, v(t_n)\rangle \in r^{\mathcal{A}}$;
- $v(\forall x.\varphi) = \min\{v_x^a(\varphi) \mid a \in |\mathcal{A}|\}$;
- $v(\varphi \to \psi) = 0$, gdy $v(\varphi) = 1$ i $v(\psi) = 0$;
- $v(\varphi \to \psi) = 1$, w przeciwnym przypadku;
- itd dla spójników logicznych.

Wartościowanie $V \to T_\Sigma$ w modelu termowym nazywamy też podstawieniem.

Prawdziwość i spełnialność

Formuła φ jest *spełniona* w $\mathcal{A}$ przez wartościowanie v, gdy $v(\varphi) = 1$. Piszemy wtedy $\mathcal{A}, v \models \varphi$.

Formuła jest *spełnialna* jeśli jest spełnione przez pewne wartościowanie.

Formuła jest *prawdziwa* (ozn $\models \varphi$), jeśli jest spełniona dla każdego wartościowania.

Formuła $A \models \varphi$ jest *konsekwencją* zbioru formuł A jeśli jest prawdziwa w każdym modelu w którym prawdziwe są A.

Das Entscheidungsproblem

Hilbert, 1928
Poszukiwany algorytm, który potrafi rozstrzygać

czy dana formuła pierwszego rzędu φ jest prawdziwa?

Równoważnie: czy jest konsekwencją zbioru A?

Church, Turing 1935–37: **nierozstrzygalne**

Klauzule

Jesli φ atom, to φ — **literał pozytywny**, $\neg\varphi$ — **literał negatywny**.

Klauzula – formuła postaci $\forall\vec{x}(l_1 \vee \ldots \vee l_m)$ gdzie l_i są literałami.
Grupując literały pozytywne i negatywne otrzymamy postać

$$\forall\vec{x}.(A_1 \vee ... \vee A_k) \vee \neg(B_1 \wedge \ldots \wedge B_n)$$

Albo, równoważnie

$$\forall\vec{x}.(B_1 \wedge \ldots \wedge B_n) \to (A_1 \vee \ldots \vee A_k)$$

W programowaniu w logice dla takiej postaci klauzuli mamy zapis

$$A_1, \ldots, A_k \leftarrow B_1, \ldots, B_n$$

Klauzule Horna

Ciekawym fragmentem jest przypadek klauzul z $\leqslant 1$ literałem pozytywnym (Horn 1951).

Klauzula Horna

$$\forall\vec{x}.A \vee \neg(B_1 \wedge \ldots \wedge B_n)$$

Albo, równoważnie

$$\forall\vec{x}.(B_1 \wedge \ldots \wedge B_n) \to A$$

W programowaniu w logice dla takiej postaci klauzuli mamy zapis

$$A_1, \dots, A_k \leftarrow B_1, \dots, B_n$$

Bez literału pozytywnego:

$$\forall \vec{x}.(B_1 \wedge \dots \wedge B_n) \to \bot$$

Nadal nierozstrzygalne, ale istnieją dobre heurystyki.

Przykład
Zapis

$$wnuk(x, y) \leftarrow \mathsf{syn}(x, z), \mathsf{dziecko}(z, y)$$

oznacza formułę (klauzulę)

$$\forall x, y, z.\mathsf{syn}(x, z) \wedge \mathsf{dziecko}(z, y) \to wnuk(x, y)$$

jest ona równoważna

$$\forall x, y.(\exists z.\mathsf{syn}(x, z) \wedge \mathsf{dziecko}(z, y)) \to wnuk(x, y)$$

W ogólności zauważmy, że jeśli $z \notin FV(\psi)$, to

$$\forall z.(\varphi \to \psi) \Leftrightarrow (\exists z.\varphi) \to \psi$$

Szczególne przypadki klauzul
Klauzula unarna (fakt)

$$A \leftarrow$$

Klauzula programu

$$A \leftarrow B_1, \dots B_n$$

Klauzula negatywna (zapytanie)

$$\leftarrow B_1, \dots B_n$$

Przykłady

```
nat(0).
nat(s(X)) :- nat(X).

?- nat(X), even(s(X)).
```

Interpretacja klauzul programu

Klauzula programu

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

może być odczytana na różne sposoby:

- interpretacja deklaratywna (prawdziwość): "A jest prawdziwe jeśli wszystkie B_i są prawdziwe"
- interpretacja deklaratywna (wartościowanie): "rozwiązanie (wartościowanie) które spełnia wszystkie B_i, jest rozwiązaniem A"
- interpretacja proceduralna "aby rozwiązać problem A, rozwiąż problemy $B_1, \ldots, B_n$"

Interpretacja zapytań

Załóżmy, ze mamy pewien zbiór aksjomatów (klauzul programu) Φ.

Klauzula negatywna

$$\leftarrow B(\vec{x})$$

reprezentuje pytanie

> "Dla jakich wartości $\vec{x}$, w każdym modelu spełniającym aksjomaty Φ będzie zachodzić B?"

Jest to równoważne znalezieniu kontrprzykładu (wartościowania) dla zbioru formuł $\Phi \cup \{\neg B\}$

Moze być wiele rozwiązań

Kwestię jak jest znajdowane takie wartościowanie omówimy później.

3 Prolog

Termy w Prologu

- stałe
  - literały całkowite: 7, -123
  - literały zmiennoprzecinkowe: -0.12e+8
- atomy: `a void =   'Algol-68'`
- zmienne (muszą się zaczynać od wielkiej litery lub podkreślenia)
- termy złożone: $f(t_1, \ldots, t_n)$
- infiksowe symbole funkcyjne też budują termy; 1+2 nie oznacza 3
- specjalna notacja dla list (lukier syntaktyczny)

Listy

- Lista pusta: `[]`
- lista o głowie t i ogonie u: `[t|u]` lub `.(t,u)`
- podobnie jak w Haskellu lukier syntaktyczny: `[1,2,3]` oznacza `[1|[2|[3|[]]]]`
- **Uwaga** na różnicę między [1,2|L] a [1,2,L]
- w Prologu jest jeden typ: zbiór termów ustalonych
- listy moga zawierac dowolne elementy ,np. `[5, "ala", 1.23, 'a', [], [2,[jasio]], f([a]`

Klauzule w Prologu

1. Klauzule unarne (fakty)

   ```
   even(0).
   ```

2. Klauzule ogólne (implikacje)

   ```
   even( s(s(X)) ) :- even(X).
   ```

   Klauzule programu wpisujemy w programie (pliku). Możemy je potem wczytać do systemu

   ```
   | ?- [pomysly].
   % compiling pomysly.pl...  compiled pomysly.pl
      in module user,
     0 msec 2632 bytes
   yes
   ```

3. Klauzule negatywne (zapytania)

Wczytywanie — SICStus Prolog

```
[ben@students ~]$ sicstus
SICStus 4.3.5 (x86_64-linux-glibc2.17): Tue Dec  6 10:41:06 PST 2016
| ?- ['lab1.pl'].
% compiling /home/staff/iinf/ben/Zajecia/Jpp/Prolog/lab1.pl...
yes
| ?- consult(lab1).
% consulting ...
```

- domyślnym rozszerzeniem plików jest `.pl`
- apostrofy potrzebne jesli ścieżka do pliku zawiera znaki niealfanumeryczne np "/").

Przykład — arytmetyka Peano
Sygnatura: z/0 s/1 nat/1 dodaj/3

```
nat(z).
nat(s(N)) :- nat(N)
dodaj(z,Y,Y) :- nat(Y).
dodaj(s(X),Y,s(T)) :- dodaj(X,Y,T).
```

X,Y,Z są zmiennymi; `dodaj/3` jest symbolem relacyjnym; `s/2` i `z/0` są symbolami funkcyjnymi (atomami).

```
| ?- nat(s(s(z))).
yes
| ?- nat(omega).
no
| ?- dodaj(s(s(z)),s(s(z)),R).
R = s(s(s(s(z))))
| ?- dodaj(s(z),Y,s(s(s(s(z))))).
Y = s(s(s(z)))
```

Wiele wyników
Program (problem) może mieć wiele wyników (rozwiązań).
Na przykład "$X + Y = 3$"

```
| ?- dodaj(X,Y,s(s(s(z)))).
X = z,
Y = s(s(s(z))) ? ;
X = s(z),
Y = s(s(z)) ? ;
X = s(s(z)),
Y = s(z) ? ;
X = s(s(s(z))),
Y = z ? n
no
```

Program w Prologu definiuje relacje, nie funkcje.

Luźne definicje relacji

Większość programow prologowych nie zawiera sprawdzeń typu **nat(Y)** ze względu na koszt. Dopuszcza to modele niestandardowe, np.

```
dodajn(z,Y,Y).
dodajn(s(X),Y,s(T)) :- dodajn(X,Y,T).

?- dodajn(s(z),s(omega),R).
R = s(s(omega)).
```

Interpretacja zapytań

Załóżmy, ze mamy pewien zbiór aksjomatów (klauzul programu) Φ.

Klauzula negatywna

$$\leftarrow B(\vec{x})$$

reprezentuje pytanie

> "Dla jakich wartości $\vec{x}$, w każdym modelu spełniającym aksjomaty Φ zachodzi B?"

Jak można znaleźć takie wartościowanie?

Dowody przez sprzeczność

Mamy pewną teorię (zbiór formuł) Φ i formułę B.

Chcemy się przekonać, czy $\Phi \models B$ (albo: dla jakich wartościowań $\Phi, \theta \models B$)

Fakt 1: $\Phi \models B \Leftrightarrow \Phi \cup \neg B \models \bot$

Fakt 2: Jeśli $\vdash$ jest poprawnym systemem dowodzenia, to

$$\Phi \cup \neg B \vdash \bot \Rightarrow \Phi \cup \neg B \models \bot$$

Dowód przez sprzeczność:

- Zacznij od $\Phi \cup \neg B$
- Stosuj reguły dowodzenia aż uzyskasz sprzeczność
- Jeśli się udało, to wiemy, że $\Phi \models B$

Dokładniej, w trakcie dowodu będziemy konstruować podstawienie θ takie, że $\Phi \models \theta(B)$ (czyli $\Phi, \theta \models B$)

4 Podstawienia

Podstawienia

Wartościowanie w algebrze termów nazywamy *podstawieniem*.

Używamy notacji

$$t[x_1 := t_1, \ldots, x_n := t_n]$$

lub

$$t[t_1/x_1, \ldots, t_n/x_n]$$

(zmienne, na których podstawienie jest identycznością, zwykle pomijamy).

Uwaga: $t[x := s, y := u]$ to zwykle nie to samo, co $t[x := s][y := u]$.

Podstawienia

Podstawienie jest operacją syntaktyczną. Term $t\theta$ powstaje z termu t przez wstawienie termu $\theta(x)$ na miejsce każdej zmiennej x.

Przykłady:

$$f(x, y)[x := f(y, x)] = f(f(y, x), y)$$

$$f(x, y)[x := y, y := x] = f(y, x)$$

Podstawienia są homomorfizmami w algebrze termów

Składanie podstawień

Jeśli θ_1, θ_2 są podstawieniami, to przez ich złożenie $\theta_1\theta_2$ rozumiemy podstawienie θ takie, że dla każdej zmiennej $x \in V$

$$x\theta = (x\theta_1)\theta_2$$

Uwaga: kolejność jest tu inna niż przy składaniu funkcji, tzn.

$$\theta_1\theta_2 = \theta_2 \circ \theta_1$$

Składanie podstawień — przykłady

$$\theta_1 = [x := f(y, g(z))], \quad \theta_2 = [y := g(t)]$$
$$\theta_1\theta_2 = [x := f(g(t),\ g(z)),\ y := g(t)]$$

$$\theta = \{x := f(y), y := z, u := g(u)\}$$
$$\sigma = \{x := a,\ y := b,\ z := y,\ u := c\}$$
$$\theta\sigma = \{x := f(b),\ u := g(c),\ z := y\}$$

Przemianowania, warianty

Podstawienie, które tylko zmienia nazwy zmiennych (jest permutacją V) nazywamy *przemianowaniem*.

Jeśli η jest przemianowaniem, to η^{-1} też.

Mówimy, że θ' jest wariantem θ jesli istnieje przemianowanie η takie, że $\theta' = \theta\eta$ (zatem $\theta = \theta'\eta^{-1}$)

Analogicznie mówimy o wariantach klauzul, etc.

Przemianowania, warianty

Podstawienie, które tylko zmienia nazwy zmiennych nazywamy *przemianowaniem*.

Mówimy, że θ' jest wariantem θ jesli istnieje przemianowanie η takie, że $\theta' = \theta\eta$.

Przykłady Niech

$$\eta = [z := y, y := z]$$
$$\theta = [x := f(y, g(z))]$$
$$\theta' = [x := f(z, g(y)), y := z, z := y]$$

wtedy

- η jest przemianowaniem,
- θ i θ' są wariantami, gdyż $\theta' = \theta\eta$

5 Uzgadnianie

Uzgadnianie (unifikacja)

Rozwiązaniem równania „$t = u$" jest podstawienie θ takie, że $t\theta = u\theta$.

Mówimy wtedy, że θ jest *unifikatorem* (podstawieniem uzgadniającym) termów t i u, lub że je *uzgadnia*.

$$f(X, b) = f(a, Y)$$
$$\theta = [X := a, Y := b]$$
$$f(X, b)\theta = f(a, b) = f(a, Y)\theta$$

Rozwiązanie główne

Równania mają zwykle wiele rozwiązań, ale niektóre są specjalne

Rozwiązanie główne (ang. most general unifier, mgu): takie θ,
że jeśli θ' jest rozwiązaniem, to $\theta' = \theta\eta$ dla pewnego podstawienia η.
(każde inne rozwiązanie można z niego otrzymać na drodze podstawienia)

Rozwiązanie główne jest jednoznaczne z dokładnością do przemianowania zmiennych.

Uzgadnianie — przykłady

Załóżmy, że $ar(\mathsf{f}) = ar(\mathsf{g}) = 2$ oraz $ar(\mathsf{c}) = ar(\mathsf{d}) = 0$.

- Rozwiązaniem równania

  $$\mathsf{f}(\mathsf{g}(X, Y), X) = \mathsf{f}(Z, \mathsf{g}(Y, \mathsf{c}))$$

  jest m. in. podstawienie

  $$\theta = [X := \mathsf{g}(Y, \mathsf{c}),\ Y := Y,\ Z = \mathsf{g}(\mathsf{g}(Y, \mathsf{c}), Y)]$$

  $$\mathrm{LHS}\theta = \mathsf{f}(\mathsf{g}(X, Y), X)\theta = \mathsf{f}(\mathsf{g}(\mathsf{g}(Y, \mathsf{c}), Y), \mathsf{g}(Y, \mathsf{c})) \tag{1}$$
  $$\mathrm{RHS}\theta = \mathsf{f}(Z, \mathsf{g}(Y, \mathsf{c}))\theta = \mathsf{f}(\mathsf{g}(\mathsf{g}(Y, \mathsf{c}), Y), \mathsf{g}(Y, \mathsf{c})) \tag{2}$$

- Równanie $\mathsf{f}(\mathsf{g}(Y, \mathsf{c}), Y) = \mathsf{f}(X, \mathsf{g}(X, \mathsf{d}))$ nie ma rozwiązania. Gdyby θ było rozwiązaniem, oraz $X\theta = \mathsf{t}$, to term t musiałby być postaci $\mathsf{g}(\mathsf{g}(\mathsf{t}, \mathsf{d}), \mathsf{c})$, czyli byłby dłuższy od siebie samego.

- Równanie $\mathsf{f}(X, \mathsf{f}(Y, \mathsf{c})) = \mathsf{f}(\mathsf{g}(Y, \mathsf{c}), X)$ też nie ma rozwiązania. Gdyby θ było rozwiązaniem, oraz $Y\theta = \mathsf{t}$, to termy $\mathsf{g}(\mathsf{t}, \mathsf{c})$ i $\mathsf{f}(\mathsf{t}, \mathsf{c})$ musiałyby być identyczne, a nie zgadzają się ich symbole początkowe.