

Programowanie w Logice - powtórzenie (v1.4)

Mikołaj Pietrek, Szymon Wróbel

korekta: Adam Friedensberg, Szymon Wróbel

Jak w poniedziałek mam sześć grup laboratoryjnych i wieczorem patrzę na Prologa, to się zastanawiam, co mi się w nim przez trzydzieści lat podobało.

dr Przemysław Kobyłański

1 Wprowadzenie¹

1.1 Termy

Term oznacza dowolne wyrażenie.

1.1.1 Termy proste (stałe, zmienne)

Stałe to liczby lub ciągi liter i cyfr rozpoczynające się małą literą: `null`, `p1`, `kwadrat`, `10`, `3.14`, `1e-6`.

Zmienne to ciągi liter i cyfr rozpoczynające się wielką literą: `X`, `WysProst`, `P123`.

Zmienna o nazwie `_` wyraża dowolną (nieistotną) wartość.

Ciągi znaków (np. `'string'`) i stałe niebędące liczbami są określane jako atomy.

1.1.2 Termy złożone

Termy złożone powstają z połączenia termów *funktorami*, które (jak stałe) są ciągami liter i cyfr rozpoczynającymi się od małej litery, np: `para(a, b)`, `para(a, para(b, c))`.

1.2 Predykaty wbudowane

Podstawowym predykatem jest unifikacja `A = B`. Dostępne są też m.in. predykaty sprawdzające „typ”.

- `var(T)` - zmienna (bez wartości)
- `nonvar(T)` - zmienna posiadająca wartość lub stała
- `atom(T)` - stała, ale nie liczba
- `number(T)` - liczba
- `compound(T)` - term złożony

¹gr. *prologos*

1.3 Definiowanie predykatów

Predykaty zapisuje się w postaci klauzul czyli zdań zakończonych kropką. Rodzaje predykatów:

- fakty, np. `rodzice(uranus, gaia, rhea).`
- reguły, np. `ojciec(X, Y) :- rodzice(X, _, Y).`

1.3.1 Alteratywa

W standardowych przypadkach (m.in. bez `freeze`) zamiast

```
rodzic(X, Y) :- ojciec(X, Y); matka(X, Y).
```

lepiej jest pisać

```
rodzic(X, Y) :- ojciec(X, Y).
```

```
rodzic(X, Y) :- matka(X, Y).
```

1.3.2 Warunki termów złożonych

Jeśli w programie są termy (złożone) określające pary, można zdefiniować predykat spełniony przez element pary.

```
element(X, Y) :-
```

```
    Y = para(X, _).
```

```
element(X, Y) :-
```

```
    Y = para(_, X).
```

Lepszy, krótszy zapis:

```
element(X, para(X, _)).
```

```
element(X, para(_, X)).
```

1.4 Listy

Szczególne przypadki termów złożonych to listy. Składają się z głowy (pierwszego elementu) i ogona (listy pozostałych). W klasycznej notacji funktor `.` (kropka) łączy głowę z ogonem. Natomiast wygodniejsze w użyciu jest zapisanie elementów między nawiasami kwadratowymi.

notacja tradycyjna	notacja uproszczona
<code>[]</code>	<code>[]</code>
<code>.(a, [])</code>	<code>[a]</code>
<code>.(a, .(b, []))</code>	<code>[a, b]</code>
<code>.(.(a, []), .(.(a, .(b, [])), []))</code>	<code>[[a], [a, b]]</code>

1.4.1 Rozdzielanie elementów

Tradycyjny zapis listy umożliwiałyby „naturalny”² wybór. Przykład dla drugiego elementu: `.(_, .(X, _))`

Natomiast w powszechnie stosowanej notacji z nawiasami potrzebny jest dodatkowy operator `|` (pionowa kreska), który oddziela początkowe elementy od pozostałych: `[_ , X | _]`.

1.4.2 Predykaty operujące na listach

- `member(X, L) → X ∈ L`
- `append(L1, L2, L3) → L1 + L2 = L3`

²ale niekoniecznie „humanitarny” oraz „posiadający jakiekolwiek praktyczne zastosowanie”

- $\text{select}(X, L1, L2) \rightarrow X + L1 = L2$

`select` i `append` działają w obie strony - mogą zarówno dodawać, jak też usuwać.

2 Jak działa Prolog?³

2.1 Reguły rozwiązywania równań

2.1.1 Decomposition

$$\frac{f(s_1, \dots, s_n) = f(t_1, \dots, t_n)}{s_1 = t_1, \dots, s_n = t_n}$$

Gdy funktory mają te same nazwy i tyle samo argumentów, Prolog unifikuje parami kolejne zmienne.

$$f(s_1, \dots, s_n) = f(t_1, \dots, t_n) \rightarrow (s_1 = t_1, \dots, s_n = t_n)$$

2.1.2 Deletion

$$\frac{X = X}{\top}$$

Zmienna jest zawsze równa samej sobie.

$$X = X \rightarrow \text{true}$$

2.1.3 Transposition

$$\frac{t = X}{X = t}$$

Jeśli t nie jest zmienną:

$$t = X \rightarrow X = t$$

2.1.4 Substitution

$$\frac{X = t, E}{X = t, EX/t}, X \notin \text{Var}(t), X \in \text{Var}(E)$$

Zmienna unifikowana ze stałą zostaje zastąpiona tą stałą w dalszej części równania.

$$(X = t, E) \rightarrow (X = t, E\{X/t\})$$

2.2 Łączenie reguł

Prolog pozostawia termy możliwie ogólnymi, żeby pasowały do możliwie dużej ilości innych termów.

Zadanie 1. $k(Z, f(X, b, Z)) = k(h(X), f(g(a), Y, Z))$

Rozwiązanie.

$$\begin{cases} Z = h(x), \\ f(X, b, Z) = f(g(a)), \\ Y, \\ Z \end{cases}$$

$$\begin{cases} Z = h(X), \\ X = g(a), \\ Y = b, \\ Z = Z \end{cases}$$

³oraz dlaczego zdecydowanie lepiej niż ANSI C

$$\begin{cases} Z = h(g(a)), \\ X = g(a), \\ Y = b \end{cases}$$

Zadanie 2. Czy istnieją takie termy s, t, u , że:

$s = t$,

$s = u$,

$t \bar{u}$.

Rozwiązanie.

$s = a$,

$t = b$,

$u = X$.

Rozwiązanie.

$s = f(X, X)$,

$t = f(a, b)$,

$u = f(X, Y)$.

2.3 Przypadki zakończenia

2.3.1 Failure 1

$$\frac{f(s_1, \dots, s_n) = g(t_1, \dots, t_k)}{\perp}, \text{ if } f \neq g \text{ or } n \neq k$$

Porównanie termów złożonych kończy się niepowodzeniem, jeśli różni się funktor lub liczba termów prostych.

$f(s_1, \dots, s_n) = g(t_1, \dots, t_n) \quad \text{false.}$

$f(s_1, s_2, s_3) = f(t_1) \quad \text{false.}$

2.3.2 Failure 2

$$\frac{X = t}{\perp}, X \in \text{Var}(t), X \neq t$$

Wartość każdej zmiennej (w ramach danej ścieżki rozwiązania) może zostać określona tylko jeden raz.

$X = a, X = b \quad \text{false.}$

2.4 Unifikacja i równość

$$s = t \mapsto X_1 = u_1, X_2 = u_2, \dots, X_n = u_n, \forall i, \forall j, X_i \notin \text{Var}(u_j)$$

- $A = B \rightarrow$ unifikacja A z B
- $A == B \rightarrow A$ jest dokładnie tym samym co B (czyli $X == X$, ale nie $X == Y$)
- $A \text{ is } B \rightarrow$ unifikacja wartości wyrażenia B z A
- $A ::= B \rightarrow$ równość (matematyczna) wartości wyrażenia B z wartością wyrażenia A

2.4.1 Porównania niezainicjalizowanych zmiennych

Wyjaśnienie bardzo *nieformalne*⁴: różnica między pojedynczą a podwójną równością podobna jak w C/C++.

⁴gramatyki są później

INPUT	OUTPUT
$X = X$	<code>true.</code>
$X == X$	<code>true.</code>
$X \text{ is } X$	<code>ERROR</code>
$X := X$	<code>ERROR</code>

(a) ta sama zmienna

INPUT	OUTPUT
$X = Y$	<code>true.</code>
$X == Y$	<code>false.</code>
$X \text{ is } Y$	<code>ERROR</code>
$X := Y$	<code>ERROR</code>

(b) różne zmienne

Tablica 1: Przykłady porównań zmiennych niezainicjalizowanych

Pojedyncza równość zadziała w obu przypadkach, ponieważ niezainicjalizowany X można zunifikować zarówno z X , jak też z Y . Natomiast podwójny znak równości zwróci `false` w drugim przypadku, ponieważ X nie jest dokładnie tym samym co Y . Pozostałe dwa zwrócą błąd (niezainicjalizowana zmienna nie może zostać obliczona).

Jeśli $A == B$ jeden raz zwróci `true`, to do końca rozwiązania wartość $A == B$ będzie równa `true`. Nie gwarantuje tego predykat $=$, ponieważ sprawdzenie warunku bez podstawienia jest możliwe za pomocą podwójnej negacji: $\neg \neg A = B$. Pierwsza negacja powoduje „zapomnienie” podstawienia, a druga przywraca wartość `true`.

2.4.2 Obliczanie wyrażeń arytmetycznych

Pojedyncza i podwójna równość są tu praktycznie nieprzydatne (traktują wyrażenie jako całość), operator `is` podstawia wartość wyrażenia z PRAWEJ strony (lewa może być nieznana), a operator `:=` oblicza obie strony.

INPUT	OUTPUT
$X = 1 + 2$	$X = 1+2.$
$1 + 2 = X$	$X = 1+2.$
$X == 1 + 2$	<code>false.</code>
$1 + 2 == X$	<code>false.</code>
$X \text{ is } 1 + 2$	$X = 3$
$1 + 2 \text{ is } X$	<code>ERROR.</code>
$X := 1 + 2$	<code>ERROR.</code>
$1 + 2 := X$	<code>ERROR.</code>

Tablica 2: Przykłady zastosowania predykatów do wyrażenia arytmetycznego

Zgodnie z regułą transpozycji, kolejność nie ma znaczenia przy znakach równości. Zachowanie analogiczne do powyższego, pojedynczy postawił całe wyrażenie, podwójny zwrócił fałsz. Natomiast poprawną wartość wyrażenia obliczył tylko predykat $X \text{ is } 1 + 2$. Ostatnie trzy przykłady zakończyły się błędem przez brak inicjalizacji.

2.4.3 Porównywanie wyrażeń arytmetycznych

INPUT	OUTPUT
$0 \text{ is } 0.0$	<code>false.</code>
$0 := 0.0$	<code>true.</code>

Tablica 3: Przykłady porównań

Predykat `is` oblicza tylko jedną stronę wyrażenia, natomiast samo porównanie odbywa się jak dla stałych Prologa, dlatego porównanie liczby rzeczywistej z całkowitą zwróci `false`. Predykat `:=` porównuje w sensie matematycznym.

2.4.4 Nierówności

Dostępne są predykaty $>$, $<$, $>=$, $=<$, $=\backslash=$. Ich działanie nie wymaga osobnego omówienia, zachowują się analogicznie jak $=$. Wskazówka do zapamiętania: operatory $>=$ oraz $=<$ mają być tak zapisane, aby nie tworzyły strzałki.

3 Gramatyki

Info od autora:

nawet nie próbuję udawać że coś o tym wiem, 4 semestr here. Jak ktoś ma czas i chęci, to wyślę link edycyjny.

Zadanie 1. Słowa postaci $a^n b^n$.

Rozwiązanie. Słowo puste spełnia warunki. Po obu stronach ma być tyle samo a oraz b .

$s \rightarrow []$.

$s \rightarrow [a], s, [b]$.

Zadanie 2. Język składa się z listy, środkowy element ma być a .

Rozwiązanie. Słowo a spełnia warunki. Po obu stronach ma być tyle samo dowolnych znaków.

$s \rightarrow [a]$.

$s \rightarrow [], s, []$.

Zadanie 3. Palindromy.

Rozwiązanie. Palindromem jest słowo puste, słowo składające się z jednego znaku oraz słowo powstałe poprzez „dołączenie” tego samego znaku po obu stronach palindromu.

$s \rightarrow []$

$s \rightarrow [_]$

$s \rightarrow [X], s, [X]$

4 Niedeterminizm

Niedeterminizm to wiele możliwych ścieżek osiągnięcia klauzuli pustej (czyli prawidłowego rozwiązania). Niedeterminizm nie oznacza losowości!

Zadanie 1. Opisz X :

$p(a, s(X))$.

$p(s(X), s(Y)) :- p(X, Y)$.

$?- p(X, s(s(s(a))))$.

Rozwiązanie. Są to termy izomorficzne z kolejnymi liczbami naturalnymi mniejszymi od 3:

$X = a$;

$X = s(a)$;

$X = s(s(a))$;

$false$.

Rozwiązanie. Wnioskowanie można zapisać również w postaci SLD-drzewa:

```
:- P(X, s(s(s(a))))
> { X = a, X1 = s(s(a)) }
  []
> { X = s(X1), Y1 = s(s(a)) }
  p(s(X1), s(Y1)) :- p(X1, Y1)
> { X1 = a, X2 = s(a) }
  []
```

```

> { X1 = s(X2), Y2 = s(a) }
p(s(X2), s(Y2)) :- p(X2, Y2)
> { X2 = a, X3 = a }
[]
> { X2 = s(X3), Y3 = a }
:- p(x3, a)

```

4.1 Predykaty niedeterministyczne

Deterministyczny predykat `true` jest prawdziwy na dokładnie jeden sposób. Odpowiednikiem (skrajnie) niedeterministycznym jest `repeat`, prawdziwy na nieskończenie wiele sposobów. Przykład wykorzystania:

`repeat`, czytaj(`X`), oblicz(`X`,`Y`), print(`Y`), fail.

Prolog odczytuje wartość `X`, wykonuje predykat `oblicz` i wypisuje wynik. Predykat `fail` powoduje przerwanie ścieżki, następuje nawrót do `repeat`, co umożliwia odczytanie dowolnej liczby wartości..

5 Programowanie współbieżne

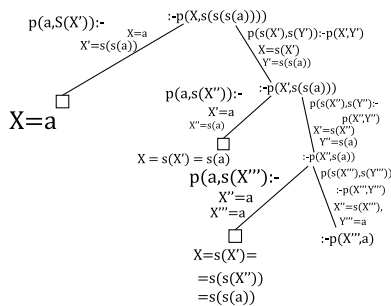
Metapredykat `freeze(term, cel)` pozwala na zamrożenie celu, aż będzie znana wartość termu. Cel zostanie wykonany „niezwłocznie” po określeniu wartości termu. Reguły z wykorzystaniem `freeze` muszą być deterministyczne oraz zapisane jako jeden predykat (info od doktora - tutaj użycie alternatywy jest dozwolone, a nawet konieczne).

Wskazówki przed kolokwium

- Ważniejsza jest znajomość działania Prologu, niż samo programowanie.
- Jeżeli pojawiają się zadania z gramatyki, wyżej będą punktowane proste rozwiązania.
- Zamiast pisania „predykatu który robi coś”, trzeba będzie określić, jakie odpowiedzi zostaną znalezione.
- Jeśli jest pytanie „podaj gramatykę, która akceptuje tylko listy złożone z termów...”, to nie kombinować.
- Raczej nie pojawi się analiza negacji⁵ oraz odcięcia.
- Oceny będą zróżnicowane, bo trzeba docenić tych, którzy potrafią zadziałać jak Prolog.

Dodatek graficzny⁶

SLD-drzewo



Powodzenia!

⁵nadal nie udało się zdefiniować „podnegacji”, koniecznej do zapisania podstawowego twierdzenia Programowania w Analizie

⁶o przepraszam, to nie ten kurs!