Skrypt do wykładu o Prologu (Paradygmaty)

matma6 (tech. Michał Gabor)

22 grudnia 2012

Przypisy oznaczają nieścisłości bądź dodatkowe informacje – ich zrozumienie nie jest wymagane.

1 Jak uruchomić SWI-Prolog

Należy zainstalować paczkę z repozytorium lub ze strony. Po instalacji: w Windowsie po kliknięciu na ikonę pojawi się REPL w innych systemach wywołujemy w terminalu `swipl`.

2 Paradygmat logiczny

Imperatywnie mówimy o liście instrukcji do wykonania. Weźmy dla przykładu `x += 1` z C. To oznacza wykonanie trzech operacji: wzięcia wartości x, dodania 1 i zapamiętania wyniku pod zmienną x. Innym przykładem może być powszechna w starszych językach instrukcja `goto`. Powoduje ona przeskok do innego miejsca programu. Podobnym zachowaniem są pętle – ciągi instrukcji wykonywane wielokrotnie.

Funkcyjne podejście[1] polega na matematycznym podejściu do funkcji. Nie ma tutaj instrukcji. Funkcje nie mają efektów ubocznych, dla tych samych argumentów dają te same wyniki, zaś po ciele funkcji nie można skakać. Mamy wyrażenia z wartością, więc ciężko mówić o skoku – jaki sens ma powiedzenie: proszę mi jeszcze raz obliczyć wartość x?

Paradygmat logiczny opiera się na relacjach. Jak wiemy z matematyki, każda funkcja jest relacją, ale nie każda relacja funkcją. Rozpatrzmy np.

[1]Nie jest to prawdą dla OCamla i Scali, ponieważ nie są to języki czysto funkcyjne. Przykładowo w OCamlu mamy pętle – jest to konstrukcja imperatywna, a nie funkcyjna. Prawdziwie funkcyjnym językiem jest Haskell – wtedy to, co zostało zapisane staje się prawdą. Efekty uboczne, co prawda, istnieją w Haskellu, ale mają czystofunkcyjną postać tzw. monad.

relację ρ z wykładu. Ile wynosi „wartość” ρ dla „argumentu” b, czy c? Tak więc relacja jest czymś szerszym niż funkcja.

3 Fakty

W Prologu zapis w kodzie postaci np. `ro(a, b).` nazywamy faktem. Możemy to rozumieć jako: istotnie a jest w relacji ρ z b. Inny przykład to `rodzice(kasia, jan, zosia).` – czytamy to jako: rodzicami Kasi są Jan i Zosia.

4 Komentarze, atomy i zmienne

Bardzo ważna w Prologu jest wielkość pierwszej litery. Mała litera oznacza atomy. Są to niepodzielne[2] symbole, reprezentujące konkretne obiekty, np. mojego psa, Wojtka (konkretną osobę) czy gatunek świń (niemniej jest to konkretny gatunek). Niepodzielność oznacza niemożliwość rozdzielenia, np. OCamlową listę `[1;2;3]` możemy rozbić na głowę i ogon, zaś listę pustą już nie.

Duża pierwsza litera oznacza zmienne, czyli „jakieś” obiekty, np.: jakiś pies, dowolna osoba, jakaś świnia, dowolny gatunek. Przykładowo x w OCamlowym `let f x = (+) List.length x;;` oznacza dowolną listę.

5 Źródło a REPL

Bardzo ważne jest zrozumienie różnicy pomiędzy kodem źródłowym, tj. plikiem z rozszerzeniem `.pl`[3], a tym, co zostało podane w konsoli (polecenie `swipl` lub ikona sowy w Windows). Dla przykładu, `lubi(jan, koty).` w kodzie źródłowym oznacza stwierdzenie faktu: Jan lubi koty. Jeżeli napiszemy to w REPLu[4], będzie to oznaczało pytanie: CZY Jan lubi koty, więc Prolog nam spróbuje na to pytanie odpowiedzieć.

Chcąc wywoływać cele (np. pytać) w kodzie źródłowym, poprzedzamy pytanie znakami `:-`[5]. Jeżeli chcemy dopisać w terminalu nowe fakty, najprościej jest napisać `[user].`, a gdy skończymy je podawać, nacisnąć ctrl+D.

[2]Prolog pozwala rozbić atom na pojedyncze znaki, lecz nie jest to rozbicie semantyczne.

[3]Czasami stosuje się .pro dla odróżnienia od skryptów Perla

[4]Skrót od Read-Evaluate-Print Loop. W OCamlu możemy wywołać REPLa poprzez wpisanie polecenia ocaml w terminalu.

[5]Należy pamiętać o tym, że Prolog będzie wtedy wypisywał mniej informacji, np. nie powie, jakie wartości przypisano zmiennym. Dodatkowo, niepowodzenie w takiej sytuacji oznacza dla Prologu błąd.

6 Zmienne

Próba przypisania nowej wartości do zmiennej musi zakończyć się niepowodzeniem. Ale co dokładnie się stało? Otóż Prolog powiązał zmienną X z atomem a. Potem próbuje dopasować X do b. X jest równe a, więc Prolog próbuje dopasować atom a do atomu b. Dwa różne atomy nigdy nie są ze sobą równe, więc zostaje ogłoszone niepowodzenie. Nota bene: niepowodzenie jest czymś innym niż błąd!

W następnym przykładzie nie wiemy nic o X ani o Y. Niemniej Prolog stwierdził, że X i Y to to samo. Więc wiążąc X z wartością 2, jednocześnie wiążemy Y z 2. W wielu innych językach, gdzie zmienna jest skrótem do wartości, nie możemy powiązać X i Y, bo ani X ani wartości nie mają.

Prolog dokonuje uzgadniania zmiennych, przez co, w każdym przypisaniu dochodzi do dopasowania do wzorca.

Ponieważ zmienne w Prologu są jednokrotnego przypisania, to nie muszą mieć ściśle przypisanej wartości. Dlatego zmienne mogą być częścią danych. Możemy np. mieć parę, w której pierwszy element jest ustalony, a drugi dowolny. Innym przykładem jest lista trzech dowolnych elementów. Możemy oczywiście powiązać te elementy z wartościami, ale gdy raz zwiążemy jakąś zmienną z wartością (choćby częściowo), to nie będziemy mogli tego zmienić. Innym przykładem jest złączenie listy określonej i dowolnej. Wynik to lista, której trzy pierwsze elementy znamy.

Prolog nie oblicza wartości wyrażeń, przez co `2+2` to nie 4. To pierwsze to wyrażenie: mamy tam znak dodawania łączący dwie liczby. Taki złożony twór nie jest liczbą 4. Jednak żeby dokonać ewaluacji możemy posłużyć się `is`.

7 Listy

Listy w Prologu zdefiniowano „tradycyjnie". Chcąc zdefiniować długość możemy napisać fakt i regułę. Reguła zawiera `:-`, co oznacza jeżeli. To, co stoi po lewej, nazywamy głową. Po prawej od tego znaku mamy ciało reguły złożone z kilku zapisów. Są one oddzielone przecinkami, co oznacza spójnik „i". Np. długość ogona to N i N1 to N + 1.

Możemy też tego dokonać optymalnie - rekursją ogonową. Długość listy pustej to wartość zgromadzona w akumulatorze. Faktem też jest, że długość całej listy, to długość ogona `T` aktualnie rozpatrywanego kawałka `[H|T]`, przy czym muszę dodać 1 do akumulatora. Warto by także powiedzieć, że długość listy, to długość listy dla akumulatora równego 0.

Złączenie list wygląda podobnie. Tutaj możemy jednak skorzystać z do-

wolnego kierunku obliczeń (nie ma `is`). Możemy zapytać, co z czym należy połączyć, by otrzymać określoną listę. Uzyskamy 4 odpowiedzi. Prolog wyświetla pierwszą odpowiedź, a po naciśnięciu spacji szuka następnej. I tak do momentu niepowodzenia lub znalezienia ostatniego rozwiązania.

8 Listy różnicowe

Tutaj mamy ciekawy przykład zmiennej jako danej. Z jednej strony nie wiemy, co odejmujemy od listy, więc mamy wolną zmienną, z drugiej natomiast wiemy, ile i jakich elementów ma listy. Tak więc pomimo wolnych zmiennych o obiekcie wiemy wszystko.

Możemy interpretować odejmowanie ogona jako wskaźnik nań. Mając przypisanie `X-Y = _-[a]` ukonkretniamy `Y = [a]`. Ale skoro `X = [1, 2 | Y]`, to mamy `X = [1, 2, a]`. Dzięki takiej konstrukcji udało nam się dostać do ogona listy.

Więc jeżeli `Y` w wyrażeniu `X-Y` jest dowolne, to uzgodnienie dowolnej zmiennej z czymkolwiek może dokonać się w czasie stałym. Zatem w czasie stałym elementy z listy `Y` pojawiają się na liście `X`.

9 Ciąg Fibonacciego

Modyfikujemy definicję poprzez dodanie wykrzyknika i `asserta/1`[6].

Po wyświetleniu pierwszej odpowiedzi Prolog oczekuje reakcji. Spacja spowoduje tzw. nawrót i znalezienie innego rozwiązania. My chcemy zapobiec nawrotom, więc mówimy `!`, co oznacza: znalazłeś już dobre rozwiązanie, nie szukaj więcej. Ten mechanizm nazywa się odcięciem.

Predykat `asserta/1` powoduje dodanie faktu do bazy. `asserta/1` dodaje na początek, zaś `assertz/1` na koniec (łatwo skojarzyć z alfabetem). Chcemy dodać fakt przed regułę, dlatego `asserta/1`. Gdybyśmy użyli `assertz/1`, reguła wykonałaby się najpierw, a odcięcie spowodowałoby, że nigdy nie dojdziemy do faktu na końcu.

10 Zagadka

Możemy traktować wszystkich rycerzy tak samo, zatem umówmy się, że zmienne oznaczą ludzi, zaś atomy, kim jest dany człowiek. Chcielibyśmy także powiedzieć „A jest rycerzem”. Zgodnie z umową jest to `A = rycerz`. Nie

[6]Ten zapis oznacza jednoargumentowy predykat.

mniej, jeżeli B skłamał, że A jest rycerzem, to nie możemy zrobić takiego przypisania. W tym celu mówimy, że rycerz jest rycerzem, a łotr łotrem.

Teraz możemy stosować wygodną składnię, np: `powiedzial(A, lotr(B)).`. I jeżeli rycerz coś powiedział, to to jest prawda. Jeżeli coś zostało powiedziane przez łotra, to musimy rozpatrzeć zdanie przeciwne.[7]

11 Więzy

Przed skorzystaniem z więzów potrzebujemy biblioteki:
`:- usemodule(library(clpfd)).`.

W zagadce na początku ustalamy listę zmiennych, potem mówimy, że te wszystkie zmienne są z przedziału 0..9. Następnie mówimy, że zmienne są różne. Potem określamy stosunki arytmetyczne między zmiennymi w stylu:

$$\overline{xyz} = 100x + 10y + z \tag{1}$$

czyli określamy wartość liczby złożonej z cyfr x, y i z. Po stwierdzeniu, że początkowe cyfry są różne od 0, możemy zadać pytanie. Po predykacie puzzle każemy określić wartości zmiennych zgodnie z ograniczeniami.

W sudoku mamy predykat, który dostaje listę wierszy. Każdy wiersz to lista 9 liczb, bądź wieloznaczników `_`, oznaczających puste miejsca w sudoku.

Najpierw stwierdzamy istnienie 9 wierszy. Potem operacją podobną do OCamlowego `List.map` stosujemy predykat `length_/2`. Innymi słowy, dla każdego wiersza W wywołujemy `length_(9, W)` czyli `length(W, 9)`, przez co każdy wiersz ma 9 zmiennych (cyfr, bądź zmiennych wolnych). Predykat `append/2` powoduje spłaszczenie listy. Podobnie jak OCamlowy `List.flatten` zamienia listę list zmiennych na listę zmiennych. Ta lista jest nam potrzebna do nałożenia więzów na sudoku - dowolna zmienna musi być liczbą z przedziału 0..9. Potem mówimy, że w każdym wierszu cyfry się różnią. Następnie po nałożeniu transpozycji możemy powiedzieć, że w kolumnie wszystkie cyfry również się różnią. Potem tworzymy zmienne dla wierszy: od A do I.

Chcąc powiedzieć, że w każdym kwadracie cyfry też mają być różne, możemy wziąć po trzy zmienne z każdym z trzech wierszy. Ustanawiamy więzy. Potem to samo robimy dla ogonów. Oczywiście poprawną sytuacją jest, kiedy ogony są puste, stąd fakt `blocks([], [], []).`.

Ładujemy nasze reguły. `[sudoku].` oznacza plik z bieżącego katalogu o nazwie `sudoku.pl`. Tworzymy zagadkę za pomocą reguł opisanych wcześniej. Nadajemy też numer 1.

[7] Niniejszy program działa tylko dla prostych przypadków. Bardziej uniwersalny i złożony program dostępny jest na stronie `http://www.im.pwr.wroc.pl/~przemko/prolog/blog-3/files/50c0703f5247aa2d62079a08c111bb13-0.html`

W pytaniu przywołujemy zagadkę numer 1, nakładamy więzy i mówimy: wypisz wszystkie wiersze (ze znakami nowej linii pomiędzy).

12 Funkcje anonimowe

Chciałem zaimplementować funkcje anonimowe, żeby móc napisać w Prologu coś na kształt OCamlowego `fun x -> x+1`. W tym celu chcę napisać coś takiego: `\(X->Y, Y is X+1)`, gdzie pierwszy element pary to *dana* → *wynik*, zaś drugi, to ciało, czyli kawałek kodu w Prologu.

Oczywiście mogę też zrobić tak[8] `Cdr = \([_|T] -> T, true).` Wtedy z listy wyciągam ogon. Wszystko obsługuje dopasowanie wzorca, więc muszę dać coś na kształt instrukcji pustej, która zwróci prawdę. Dlatego napisałem `true`.

Podstawianie jest proste: trzeba uzgodnić argument z wejściem, wynik z wyjściem i wykonać predykat P. Jednakże wyobraźmy sobie, że chcemy zrobić coś na kszałt OCamlowego `List.map`, wtedy na pierwszym elemencie X zostanie związane z jego wartością, więc np. `[1, 2, 3]`, wtedy `X = 1`, przechodzimy do elementu 2, ale wtedy mamy uzgodnić `X = 2`, ale skoro `X = 1`, to `1 = 2` zawiedzie. Wniosek: do funkcji można podstawić jeden argument. Przydałoby się jakoś ją skopiować, na szczęście do tego mamy wbudowany predykat `copy_term/2`.

[8] Nazwa funkcji pochodzi z języka Lisp i oznacza branie ogona (ściślej to drugiego elementu pary kropkowej, np. listy niepustej).