Sudoku u SWISH-u

Glazer, Karla

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Karla Glazer

SUDOKU U SWISH-U

ZAVRŠNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Karla Glazer

Matični broj: 0016143003

Studij: Informacijski sustavi

SUDOKU U SWISH-U

ZAVRŠNI RAD

Mentor/Mentorica:

Vlatka Sekovanić mag. educ. inf.

Varaždin, rujan 2023.

Karla Glazer

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristila drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autorica potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Cilj ovog rada je upoznavanje sa programskim jezicima za logičko programiranje, najvećim dijelom s Prologom i njegovim online okruženjem koje se naziva SWISH. SWISH je detaljno opisan i na jednostavnim primjerima objašnjen je način programiranja u navedenom jeziku. Kao uvod u praktični dio rada napravljeno je upoznavanje sa igrom „Sudoku“. Ukratko je navedena povijest te je objašnjena teorija i način rješavanja igre. Na primjeru Sudoku-a izrađenog u SWISHU prikazan je složeniji način programiranja. Uz primjere navedene igre kreirane su usporedbe programskih jezika za logičko programiranje.

Ključne riječi: Sudoku; logičko programiranje; Prolog; SWISH;

Sadržaj

1. Uvod

Tema ovog rada je izrada Sudoku-a u SWISH-u. SWISH je online okruženje za besplatnu verziju Prologa pod nazivom SWI-Prolog. Sudoku je problemska logička igra koja zahtjeva popunjavanje matrice (9x9) brojevima prema određenim pravilima.

U teorijskom dijelu rada ukratko je objašnjeno što je logičko programiranje. Opisani su najpoznatiji i najkorišteniji programski jezici za logičko programiranje. Prilikom opisivanja programskog jezika „Prolog“ prikazane su i uspoređene njegove verzije koje se koriste. U sljedećem poglavlju upoznajemo se sa SWISH verzijom Prologa. Na jednostavnim primjerima prikazani su svi ključni koraci logičkog programiranja u ovom online okruženju. Na kraju teorijskog dijela opisana je logička igra „Sudoku“. U tom se poglavlju prikazuje kratka povijest same igre te detaljno objašnjava logika na kojoj se igra temelji, pravila i mogući načini rješavanja dobivene zagonetke.

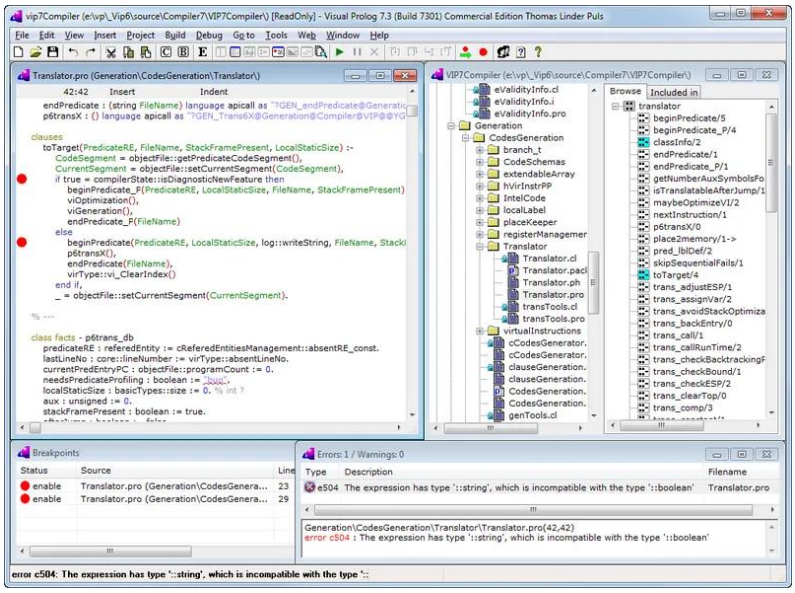
Praktični dio, odnosno izrada Sudoku-a u SWISH-u sadrži opisane korake kreiranja igre. Svaki korak popraćen je programskim kodom. U određenom koraku opisan je modul za Sudoku koji se koristi kako bi se generiralo rješenje zadane zagonetke. Na kraju rada prikazana je usporedba više različitih Sudoku-a kreiranih u različitim logičkim programskim jezicima.

2. Programski jezici za logičko programiranje

Logičko programiranje je vrsta računalnog programiranja u kojoj računalo, pomoću definiranih činjenica i pravila, donosi logičke odluke. Logički jezici relevantne podatke prikazuju pomoću upita koji mogu biti ručno ili automatski pokrenuti. Ovakav način programiranja tvrtkama i organizacijama može pomoći kroz procesiranje prirodnog jezika, upravljanje bazom podataka i prediktivnom analizom.[1] Neki od najpoznatijih jezika, koji su opisani u nastavku, su: Prolog, Mercury i Datalog.

2.1. Prolog

Prolog je, primarno, namijenjen za deklarativno programiranje. Ima veliku ulogu u umjetnoj inteligenciji i primijenjena logika je njegova srž. U Prologu, logika se definira preko relacija koje se nazivaju „Činjenice“ i „Pravila“. [2]



Slika 1: Primjer programskog koda u Prologu [3]

2.1.1. Povijest

Prvu verziju Prologa napravili su Alain Colmerauer i Phillipe Roussel u suradnji sa Robertom Kowalskim u ranim 70-ima. Colmerauerovo istraživanje je formaliziralo programski jezik, a Kowalski je doprinio teorijski okvir na kojem se jezik temelji. Ime Prolog je odabrao Phillipe Roussel kao skraćenicu za „programiranje u logici“ (franc. „programmation en

logique“). [4] Prolog probleme rješava na temelju dokazivanja teorema. Smatra se „razgovornim“ programskim jezikom jer zahtjeva da korisnik i računalo „komuniciraju“.

2.1.2. Struktura

Prolog se sastoji od baze znanja i interaktivnog dijela, u bazu znanja upisuju se činjenice i pravila, tj. klauzule, a u interaktivni dio se postavljaju upiti.

Najčešće korišteni elementi prilikom pisanja programskog koda u Prologu su: činjenice, pravila i upiti. Činjenice se pišu tako da se odabere naziv predikata i nakon naziva se argumenti pišu u zagradama, za njih se smatra da su istinite. Pravila su logičke izjave koje se sastoje od glave, vrata i tijela. Glava pravila je predikat koji u zagradama ima zaključak, vrat je „:-„, a tijelo pravila služi za postavljanje pretpostavke koja može biti lažna ili istinita. Varijabla služi za promjenjive vrijednosti, naziv mora započeti velikim slovom ili specijalnim znakom „_“. U jednoj klauzuli varijabla se može pojaviti više puta. Atom je niz znakova koji čine jedan podatak. U pravilu započinju malim slovom, ali ako započinje velikim slovom ili sadrži razmak mora biti pod jednostrukim navodnicima. Upiti služe kako bi se preko interaktivnog dijela dobilo rješenje problema. Pišu se na isti način kao i činjenice samo se na kraju doda specijalni znak „?“. [3]

2.1.3.Verzije Prologa

Postoje različite verzije, odnosno implementacije Prologa. Neke implementacije se znatno razlikuju od ostalih. Zbog ne prihvaćanja ISO-Prolog standarda za module većina implementacija nije kompatibilna s ostalima. To se događa jer neke implementacije koriste ograničenu aritmetiku, napredne numeričke tipove i/ili biblioteke koje druge implementacije ne mogu koristiti. U nastavku su ukratko opisane neke najkorištenije verzije Prologa.

2.1.3.1. SWI-Prolog

SWI-Prolog je besplatna implementacija Prologa koja se najčešće koristi za obrazovanje. Kompatibilan je s mnogim drugim programskim jezicima kao što su Java, C, C#, Python, itd. Ima proširene tipove podataka, što mu omogućava lakšu razmjenu podataka s drugim komponentama i/ili programskim jezicima. Ova implementacija Prologa omogućava distribuciju i instalaciju dodataka preko paketa koji korisnicima služe za lakšu razmjenu koda u zajednici. Opremljen je opsežnim okvirom za web poslužitelj, tako da se može koristiti za kreiranje usluga i aplikacija koje se temelje na HTML5, CSS i JavaScript jezicima. Prologovi motori se nazivaju „Penguines“ i oni omogućavaju postavljanje upita prema udaljenim klijentskim programima. [5]

SWISH je online verzija SWI-Prologa koja sadržava prijenosnu bilježnicu i integrirano razvojno okruženje (eng. IDE). Zbog svoje proširivosti, SWISH se može koristiti u različitim scenarijima. SWISH je detaljnije opisan u poglavlju „Uvod u SWISH“.

Primjer koda prikazuje čitanje korisničkog unosa i ispisivanje istog.[6] Ukoliko korisnik unese riječ „Stop“, petlja se prestaje izvršavati.

```
% Reading and writing
% ------------------

hello_world :-
    writeln('Hello World!'),
    sleep(1),
    hello_world.

read_and_write :-
    prompt(_, 'Type a term or \'stop\''),
    read(Something),
    (   Something == stop
    ->  true
    ;   writeln(Something),
        read_and_write
    ).

/** <examples>

?- hello_world.
?- read_and_write.

*/
```

2.1.3.2. B-Prolog

B-Prolog predstavljen je 1994. godine. Bio je visokoučinkovita implementacija Prologa koja je imala sve funkcionalnosti kao i standardni jezik uz dodatna proširenja koja su uključivala klauzule za podudaranje, nizove i hash tablice, deklarativne petlje i dr. B-Prolog je besplatan za korištenje u svrhe učenja i neprofitabilna istraživanja. Više se ne razvija ali služi kao osnova za Picat (programski jezik). [7]

U nastavku prikazan je primjer programskog koda napisanog u B-Prologu. Primjer prikazuje rješavanje problema pakiranja kutija. Autor programskog koda je Neng-Fa Zhou, a kod je izrađen 2002. godine.[8]

```
/*  Purpose: Solve the box packing problem.
    Author: Neng-Fa Zhou, 2002
```

```
    Platform: B-Prolog 6.1 or up
    Description:
    The dimensions of the boxes are give by the relation box/1
    and the layout area is defined by the predicate area/2.
    For each box, let W and H be the width and the height box,
    respectively, and AreaW and AreaH be the width and height
    of the layout area, respectively. The domain of positions
    for the box is:
         [(0,0),...,(AreaW-W,AreaH-H)]
*/
go:-
    findall(box(W,H,_),box(W,H),Boxes),  %box(Width,Height,PosVar)
    area(AreaW,AreaH),  % layout area
    createPosVars(Boxes,AreaW,AreaH,PosVars),
    writeln(PosVars),
    labeling_mix(PosVars,1000,Result),
    writeln(Boxes),
    writeln(Result).

createPosVars([],AreaW,AreaH,[]).
createPosVars([box(W,H,(PosX,PosY))|Boxes],AreaW,AreaH,
[PosX,PosY|PosVars]):-
    PosX :: 0..AreaW-W,
    PosY :: 0..AreaH-H,
    notOverlap(W,H,PosX,PosY,Boxes),
    createPosVars(Boxes,AreaW,AreaH,PosVars).

notOverlap(W,H,X,Y,[]).
notOverlap(W,H,X,Y,[box(W1,H1,(X1,Y1))|Boxes]):-
    (X+W #=< X1 #\/ X1+W1 #=< X #\/ Y+H #=< Y1 #\/ Y1+H1 #=< Y),
    notOverlap(W,H,X,Y,Boxes).

area(33,24).

box(3,9).
box(6,6).
box(9,6).
box(9,3).
box(12,3).
box(12,6).
box(3,12).
box(6,9).
box(6,12).
box(6,3).
box(21,3).
box(9,9).
box(3,18).
```

2.1.3.3. Visual Prolog

Visual Prolog se temelji na standardnom Prologu, ali nije namjenjen isključivo za logičko programiranje. Kombinacijom logičkih, funkcijskih i objektno orijentiranih paradigmi postao je vrlo moćan programski jezik. Prikladan je za rad sa složenim bazama znanja i omogućava laku izgradnju aplikacija za Microsoft Windows operacijski sustav. U prošlosti, Visual Prolog, bio je poznat kao PDC Prolog i Turbo Prolog. Za korištenje u komercijalne svrhe potrebno je platiti, ali verzija koja služi za osobno korištenje je besplatna. [9]

Ispod je prikazan primjer koda napisan u navedenom programskom jeziku. Kod prikazuje rješavanje zagonetke “Problem hanojskih tornjeva”.[10]

```
   implement main
       open core, std, stdio

   domains
       pole = a; b; c.
       solution =
           done;
           compound(solution SourceToTemp, pole Source, pole
Destination, solution TemoToDestination).

   class facts
       memoization : mapM{tuple{unsigned Size, pole Source, pole
Temp, pole Destination}, solution} := mapM_redBlack::new().

   class predicates
       hanoi : (unsigned Size, pole Source, pole Temp, pole
Destination) -> solution S.
   clauses
       hanoi(Size, Source, Temp, Destination) =
           memoization:get_defaultLazy(tuple(Size, Source, Temp,
Destination),
               {  =
                   if Size = 0 then
                       done
                   else
                       compound(hanoi(Size - 1, Source, Destination,
Temp), Source, Destination,
                           hanoi(Size - 1, Temp, Source,
Destination))
                   end if
               }).

   class predicates
```

```
        writeSolution : (solution Solution).
    clauses
        writeSolution(done).
        writeSolution(compound(First, Source, Destination, Last)) :-
            writeSolution(First),
            writef("    % -> %\n", Source, Destination),
            writeSolution(Last).

    class predicates
        profileSolve : (unsigned N).
    clauses
        profileSolve(N) :-
            T1 = time::now(),
            _ = hanoi(N, a, b, c),
            T2 = time::now(),
            D = duration::new(T1, T2),
            writef("Tower of hanoi size % in %p\n", N, D).

    clauses
        run() :-
            foreach I = std::fromTo(1, 4) do
                writef("Moving % discs from % to % using % as
temporary\n", I, a, b, c),
                writeSolution(hanoi(I, a, b, c)),
                nl
            end foreach,
            profileSolve(64),
            profileSolve(1000),
            profileSolve(10000).

    end implement main

    goal
        console::runUtf8(main::run).
```

2.1.3.4. Usporedba verzija

Preko primjera može se primjetiti kako SWI-Prolog i B-Prolog imaju jako sličnu sintaksu pisanja klauzula, dok Visual Prolog ima potpuno različitu sintaksu koja se više poklapa sa sintaksom programskog jezika C. Kako je ranije navedeno, zbog ne prihvaćanja ISO-Prolog standardnog jezika, sintakse se razlikuju i Visual Prolog nije kompatibilan s ostalim primjerima. Također, Visual Prolog ima puno više mogućnosti zbog kombiniranja s ostalim programskim paradigmama. Prednost SWI-Prologa i B-Prologa je u tome što je lakše programirati, to omogućava njihova jednostavnost pri pisanju sintakse.

2.2. Mercury

Mercury je funkcionalni logički programski jezik koji se temelji na čistoj deklarativnoj logici. Prva verzija objavljena je 1995. godine, a razvili su ju Fergus Henderson, Thomas Conway i Zoltan Smogyi. Ima visoko optimizirani algoritam koji je učinkovitiji od ostalih programskih jezika za logičko programiranje. Zbog toga je u Mercury-u moguće razvijati zahtjevnije programe. [11]

Čisto deklarativno programiranje znači da predikati i funkcije nemaju nelogične nuspojave jer prilikom korisničkog unosa jezik zahtjeva da je posljednje stanje sustava stvarno posljednje stanje. Na taj način se omogućava destruktivno ažuriranje stanja. Stroga tipizacija zahtjeva od programera točnu deklaraciju potrebnih tipova prema određenim pravilima, isto tako mora deklarirati i stanje predikata u vrijeme poziva i u vrijeme uspjeha. [12]

Pomoću svoje modularnosti Mercury omogućava implementaciju apstraktnih tipova podataka. Mercury ima mogućnost korištenja ekstenzija C programskog jezika i pomoću njih izgraditi kod koji je puno kvalitetniji od dosadašnjih kodova izrađenih u Prologu.

Primjer koda koji je naveden ispod prikazuje kako se pomoću Mercury-a može pozvati C# metoda Console.Writeline().[13]

```
   % This is a simple example of using the foreign language interface to call
the
   % C# method Console.WriteLine().

   % This source file is hereby placed in the public domain.

   :- module short_example.
   :- interface.
   :- import_module io.

   :- pred main(io::di, io::uo) is det.

   :- implementation.

   main(!IO) :-
       csharp_write_line("Hello, world", !IO).

   :- pragma foreign_decl("C#", "using System;").

   :- pred csharp_write_line(string::in, io::di, io::uo) is det.
   :- pragma foreign_proc("C#",
       csharp_write_line(S::in, _IO0::di, _IO::uo),
       [promise_pure, will_not_call_mercury],
```

```
"
    Console.WriteLine(S);
").
```

2.3. Datalog

Datalog je deklarativni programski jezik. Izveden je iz Prologa tako da se njegova logika temelji na logici prvog reda. Osmišljen je u kasnim 70.-ima prošlog stoljeća. Često se koristi u radu s bazama podataka jer omogućava kreiranje upita na bazu. Sintaksa je vrlo slična kao i kod Prologa i zbog toga programeri vole raditi u Datalogu. Prilikom korištenja Dataloga važno je pripremiti se na mogućnost pojave neobrađenih skupova podataka jer današnje baze mogu sadržavati apstraktne informacije. [14][15]

Pogodan je za korištenje u aplikacijama umjetne inteligencije jer omogućuje upravljanje podacima i učinkovitu obradu upita. Još se može koristiti i u rudarenju podataka i programskom inženjerstvu. Prednost Dataloga je u tome što se može izvoditi na bilo kojem Prolog interpreteru.

U nastavku je prikazan primjer upita napravljenog na bazu. Primjer je kreiran u Datomicu koji koristi Datalog za kreiranje upita na bazu.[16]

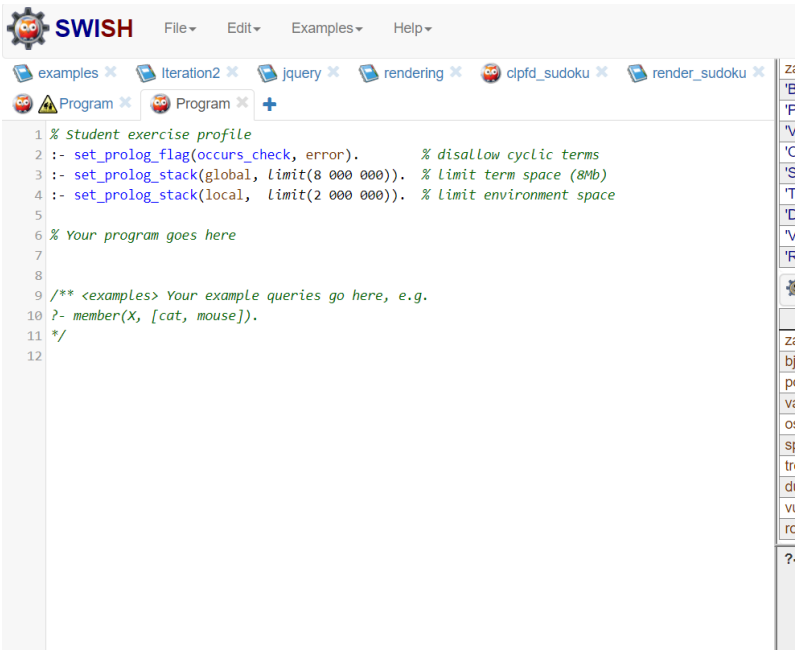
```
%baza n-torki
[[sally :age 21]
 [fred :age 42]
 [ethel :age 42]
 [fred :likes pizza]
 [sally :likes opera]
 [ethel :likes sushi]]
%upit
[:find ?e
 :where [?e :age 42]]
```

3. Uvod u SWISH

SWISH je online okruženje napravljeno za SWI-Prolog implementaciju. Služi kao platforma za međusobno dijeljenje koda u zajednici, po čemu je i dobio naziv kao skraćenica od „SWI-Prolog for Sharing“. Prethodno je ukratko opisana struktura Prologa, a sada se detaljnije upoznajemo s načino programiranja u Prologu. [5]

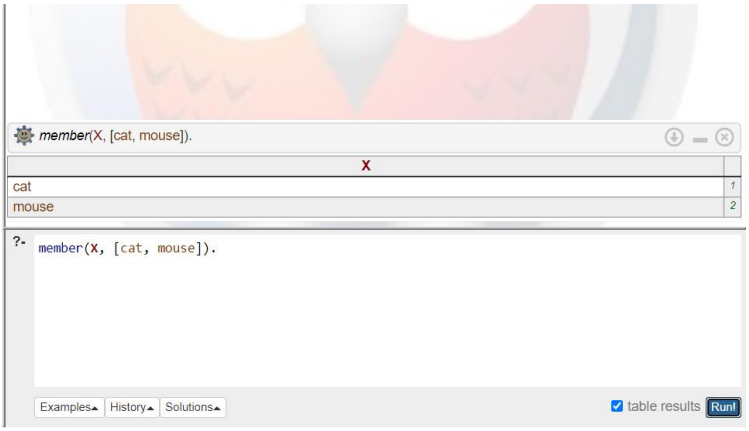
Prolog sadrži tri osnovne konstrukcije: činjenice, pravila i upiti. Skup činjenica i pravila čini bazu znanja, a upiti se postavljaju u interaktivni dio, tj. upiti se postavljaju kao pitanja o bazi znanja. Jednostavno rečeno, programiranje u Prologu znači izgradnja baze znanja.

S obzirom da se kroz primjere najbolje uči, na njima je objašnjeno programiranje u SWISH-u. Na slici „Slika 2“ prikazana je baza znanja, u kojoj se piše programski kod, u SWISH-u.



Slika 2: Baza znanja u SWISH-u [autorski rad]

Slika 3 prikazuje interaktivni dio u koji se upisuju upiti na koje Prolog vraća rješenje.



Slika 3: Interaktivni dio SWISH-a [autorski rad]

3.1. Kreiranje baze znanja

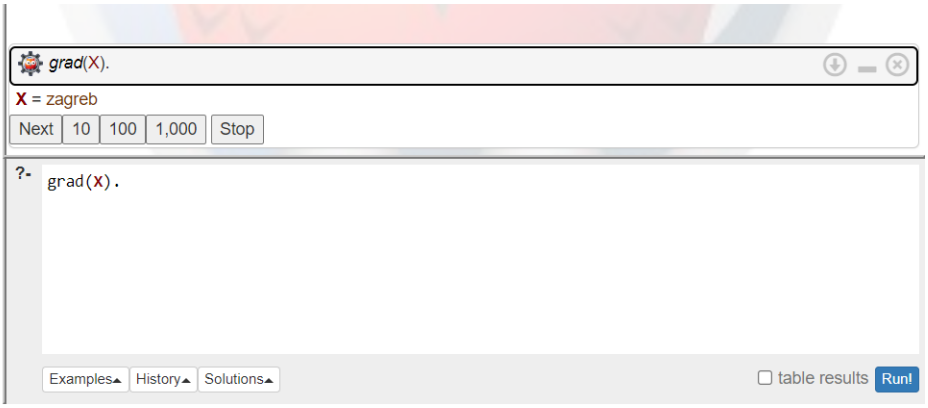
Kroz nekoliko primjera prolazimo kroz sve važne korake kako bi se mogao kreirati program u SWISH-u. U ovim primjerima objašnjavamo samo činjenica i pravila. Kasnije su opisane kompleksnije stvari.

3.1.1. Primjer 1

Za početak kreiramo jednostavnu bazu znanja koja se sastoji samo od činjenica. Za činjenice se smatra da su istinite. Činjenica je predikat koji se sastoji samo od glave, bez vrata i tijela. Na primjer, definirali smo neke gradove u Hrvatskoj. Predikat smo nazvali „grad“, nazivi gradova pisani su malim slovom jer se atomi pišu malim slovom ili velikim slovima unutar jednostrukih navodnika.

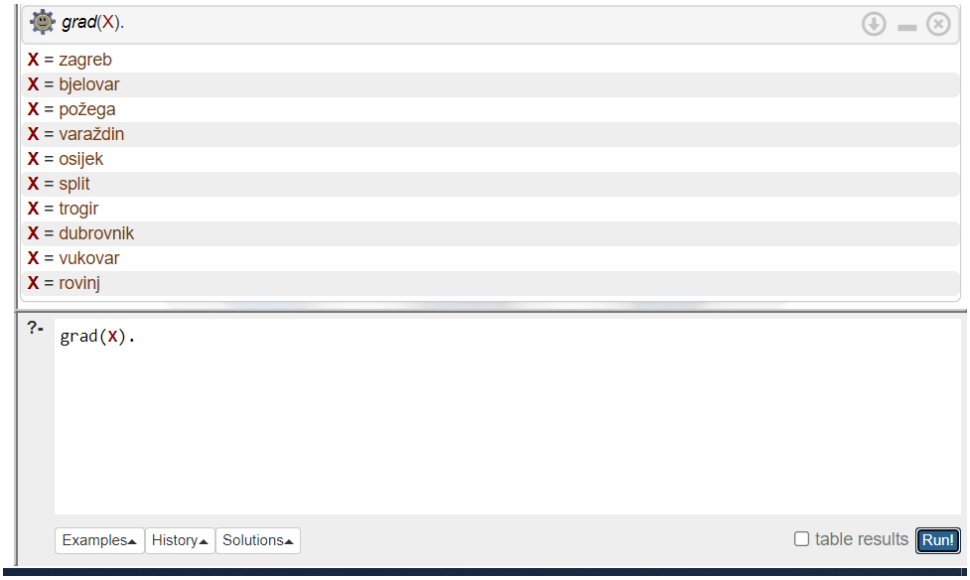
```
grad(zagreb).
grad(bjelovar).
grad(požega).
grad(varaždin).
grad(osijek).
grad(split).
grad(trogir).
grad(dubrovnik).
grad(vukovar).
grad(rovinj).
```

Ako u interaktivni dio postavimo upit `?- grad(X).` Prolog nam vraća naziv prvog grada kojeg smo unijeli u bazu znanja. Nudi nam mogućnost ispisivanja sljedećeg grada, sljedećih 10, 100 ili 1000, ili zaustavljanje programa. Slika 4 prikazuje naziv prvog grada uz ostale mogućnosti odabira.



Slika 4: Primjer upita 1 [autorski rad]

Kako imamo točno 10 gradova upisanih u bazu znanja, odabirom na ponuđeni gumb „10“, upit nam vraća sve definirane gradove. Na slici 5 prikazano je vraćeno rješenje našeg drugog upita.



Slika 5: Primjer upita 2 [autorski rad]

Kada bi u upitu zamijenili X sa zagreb (?- grad(zagreb).), rezultat upita bio bi true jer u bazi znanja postoji takva činjenica. Zamjenimo li zagreb sa npr. rijeka, rezultat bi bio false.

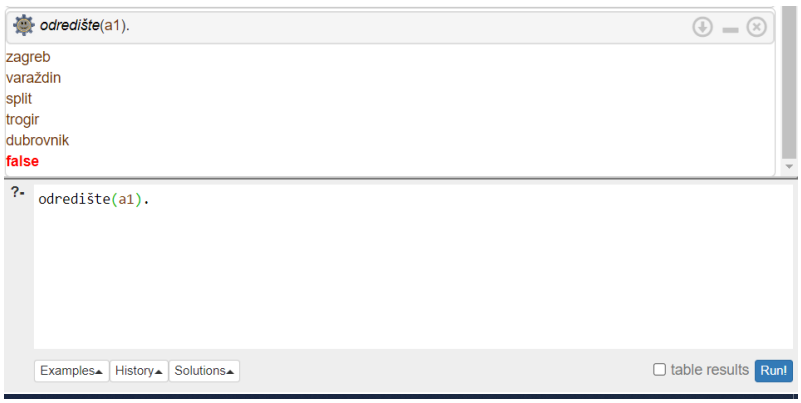
3.1.2. Primjer 2

U ovom primjeru nadograđujemo našu bazu znanja sa još jednim predikatom kojeg nazivamo cesta. Uz to moramo definirati i predikat pod nazivom put koji ima kratnost 2 i povezuje cestu sa gradom. Na kraju moramo definirati pravilo, nazovimo ga odredište, koje nam provjerava do kojeg grada možemo doći s odabranom cestom. Pravilo je predikat koji se sastoji od glave, vrata i tijela. Može se reći da glava pravila uspijeva ako tijelo pravila uspijeva.

Sada naša baza znanja izgleda ovako.

```
grad(zagreb).
grad(bjelovar).
grad(požega).
grad(varaždin).
grad(osijek).
grad(split).
grad(trogir).
grad(dubrovnik).
grad(vukovar).
grad(rovinj).

cesta(a1).
cesta(a2).
cesta(a3).

put(a1,zagreb).
put(a1,split).
put(a1,varaždin).
put(a1,trogir).
put(a1,dubrovnik).
put(a2,zagreb).
put(a2,vukovar).
put(a2,rovinj).
put(a3,zagreb).
put(a3,bjelovar).
put(a3,osijek).
put(a3,požega).

odredište(X):-grad(Y),put(X,Y),write(Y),nl,fail.
```

Postavljanjem upita `?- odredište(a1).` Prolog nam vraća popis gradova do kojih možemo doći. Upit je prikaza na slici „Slika 6“.



Slika 6: Primjer upita 3 [autorski rad]

Pravilo se na razgovorni jezik može pročitati kao ako je X put do grada Y onda ispiši grad Y. U ovom pravilu pojavile su nam se neki novi predikati koje nismo definirali u bazi znanja, to su `write/1`, `nl` i `fail`. Ovi predikati nazivaju se ugrađeni predikati i definirani su u samom jeziku Prolog. Usklađivanje predikata `write/1` uvijek uspijeva pri pozivanju, a služi za ispisivanje terma, koji mu je argument, na zaslon. Predikati `nl` i `fail` imaju kratnost 0 što znači da ne primaju nikakve argument, nl na zaslonu prelazi u novi red, vrlo često se koristi prilikom ispisivanja listi ili nizova. Postoji još jedan ugrađeni predikat koji objedinjuje `write` i `nl`, a zove se `writeln/1`. Taj predikat odmah ispisuje svaki term u novi red. I za kraj ostaje nam predikat `fail`, usklađivanje ovog predikata nikada ne uspijeva.

Problem je što nam upit uvijek vraća false. Zbog toga dodajemo, odmah ispod predikata „odredište“, još jednu klauzulu kojoj je argument anonimna varijabla. Anonimna varijabla služi za definiranje varijable koja se pojavljuje samo jednom u klauzuli, a označava se specijalnim znakom „_“.

4. Sudoku

Sudoku je logička igra koja zahtjeva popunjavanje matrice brojevima i/ili znakovima u intervalu koji je ograničen veličinom matrice. Danas postoje razne vrste dizajna Sudoku-a, ali klasični Sudoku sastoji se od velike matrice (9x9) u kojoj se nalazi devet malih matrica (3x3) koje se popunjavaju znamenkama od 1 do 9. Svaka znamenka smije se pojaviti jednom u redu, stupcu i maloj matrici. Kreator Sudoku-a određuje koje znamenke prikazuje, a koje igrač treba popuniti. Važno je da matrica ima jedinstveno rješenje.

Ispod, na Slici 7, prikazan je nepopunjeni klasični Sudoku minimalnog dizajna.

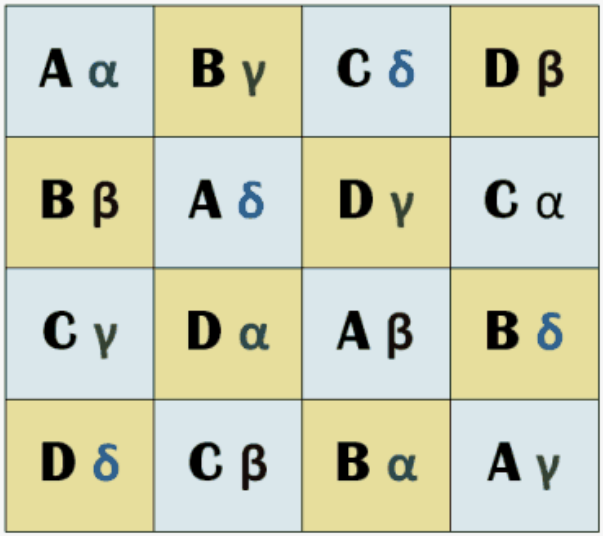
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Slika 7: Klasični Sudoku [17]

4.1. Povijest

1783. godine, švicarski matematičar, Leonhard Euler osmislio je mrežu u kojoj se svaki broj ili simbol, u svakom stupcu ili retku smije ponoviti jednom. Tu mrežu nazvao je „Latin Squares“. „Latin Squares“ osmišljeni su kao sustav za korištenje u statističkoj analizi. [18]

Slika prikazuje primjer latinskih kvadrata u kojima se koriste grčka slova α, β, γ i δ u kombinaciji s latinskim slovima A, B, C i D.



Slika 8. "Latin Squares"[19]

Prvi primjer modernog Sudoku-a pojavio se, 1979. godine, u New Yorku pod nazivom „Number Place“. Vjeruje se da je izumitelj modernog Sudoku-a Howard Garns. „Number Place“ je matrica, veličine 9x9, koja zahtjeva od igrača postavljanje brojeva od 1 do 9, bez ponavljanja broja u stupcu ili retku. [18]

1984. godine, izdavač Nikoli je predstavio modificiranu verziju „Number Place“ igre u Japanu. Dodavanjem dva nova pravila napravio je, danas poznati, Sudoku. Nova pravila su određivala da početno prikazani brojevi moraju stvarati različite uzorke i ograničio je maksimalan broj početnih brojeva na 32. U Japanu se igra brzo proširila. [20]

Igra se proširila u ostatak svijeta zbog Wayne Goulda koji je napisao računalni program za generiranje Sudoku-a. The London Times je, 2004. godine, objavio Gouldove zagonetke koje su potaknule i ostale novine i časopise da objavljuju Sudoku. [20]

4.2. Logika

Postoji mnogo različitih načina na koje se Sudoku može riješiti. Svaka metoda zapravo sadržava neku verziju vraćanja unatrag (eng. Backtracing).

Backtracing metoda, odnosno metoda vraćanja unatrag. Igrač (rješavač) zapisuje sve brojeve koji prema poznatim ograničenjima mogu biti smješteni u svaku od ćelija zadane matrice. Kada je neka od ćelija sužena na točno jedan broj, igrač unosi taj „točan“ odgovor i ponavlja postupak. Svaki točno upisani broj daje nova ograničenja za ostale ćelije u redu, stupcu ili podmatrici. Ovo je najosnovnija strategija koja može bespotrebno produžiti vrijeme potrebno za rješavanje dobivene zagonetke. [21][22]

Prethodno navedeni način može se ubrzati tako da korisnik odabere red, stupac ili podmatricu koju ispunjava sa svim mogućim brojevima. Ove dvije strategije mogu riješiti lagane zagonetke, ali često je za rješavanje potrebno kompliciranije analiziranje. Nekada je čak potrebno i pogađati te, ukoliko to rezultira sukobom, vraćanje na prethodne ćelije.

Jedna od kompliciranijih strategija je promatranje trojki u stupcu ili redu. Za ovu strategiju igrač treba uzeti sva tri stupca ili retka iz podmatrice i analizirati odabrani broj i njegove mogućnosti popunjavanja. Ovaj način može pomoći pri eliminaciji mogućnosti u svim povezanim ćelijama ukoliko igrač primjeti da se broj može pojaviti na određenim mjestima.[22]

Pogađanje brojeva najbolje je koristiti tek kada stvarno više nema ćelija u koja uz sva ograničenja nema jedinstveni broj koji se može zapisati. Kod pogađanja treba uzeti ćeliju u kojoj je najmanji broj mogućnosti jer, ako i dođe do konflikta među znamenkama, nju je najlakše poništiti i ispraviti u najmanjem broju koraka.

4.2.1. Rješavanje Sudoku-a

S obzirom da svaka osoba ima svoj način razmišljanja i rješavanja, ne može se reći koja je najbolja strategija prilikom rješavanja. U ovom dijelu, u koracima, prikazan je jedan od načina razmišljanja prilikom rješavanja klasičnog Sudoku-a prikazanog na Slici 7.

Tablica 1 prikazuje početno stanje zagonetke koju je potrebno riješiti. Tablica je kreirana prema Slici 7.

5	**3**			**7**				
6			**1**	**9**	**5**			
	9	**8**					**6**	
8				**6**				**3**
4			**8**		**3**			**1**
7				**2**				**6**
	6					**2**	**8**	
			4	**1**	**9**			**5**
				8			**7**	**9**

Tablica 1: Početna zagonetka

U prvom koraku prolazimo redom kroz brojeve i provjeravamo postoji li red, stupac ili podmatrica koja samo na jednom mjestu može sadržavati određeni broj. Uzmimo npr. broj 1,

odmah možemo vidjeti kako u devetoj podmatrici postoji samo jedna ćelija koja nema konflikte s ostalima. U tablici „Tablica 2“ vizualno je prikazano navedeno zapažanje.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Tablica 2: Vizualni prikaz mogućeg unosa 1

Nakon unošenja broja 9 u podmatricu 9, možemo primjetiti da nam se u trećoj podmatrici, prikazano u Tablici 3, isto tako stvorilo jedinstveno mjesto za broj 1. Taj korak ponavljamo za svaki broj, dok god postoji mogućnost takvog unosa brojeva.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8		*1*	7	9

Tablica 3: Vizualni prikaz mogućeg unosa 2

Nakon prolaženja kroz sve brojeve, u Tablici 4, uočavamo kako nam je u devetoj podmatrici ostala samo jedna prazna ćelija. Nju popunimo s preostalim brojem i tako smo rješili prvu podmatricu. S obzirom da smo ispunili sve podmatrice s brojem 9 njega smo isto tako riješili.

5	**3**			**7**		*9*	*1*	
6			**1**	**9**	**5**			
1	**9**	**8**				*5*	**6**	
8		*9*		**6**				**3**
4		*6*	**8**		**3**		*9*	**1**
7		*3*	*9*	**2**		*8*		**6**
9	**6**	*1*				**2**	**8**	*4*
	8		**4**	**1**	**9**	*6*		**5**
				8		*1*	**7**	**9**

Tablica 4: Prikaz dovršenog prvog koraka rješavanja

U drugom koraku rješavanja najbolje je opet proći po strategiji prvog jer postoji mogućnost da nam se, zbog popunjavanja ćelija u podmatricama, opet stvorilo jedinstveno mjesto za određeni broj.

Ovaj primjer bio je lagana verzija Sudoku-a koja se mogla riješiti samo pomoću ponavljanja istog načina. U svakom trenutku pojavljivala se jedinstvena ćelija za svaki broj. Tablica 5 prikazuje ispunjenu matricu, odnoso rješenje Sudoku zagonetke.

5	**3**	*4*	*6*	**7**	*8*	*9*	*1*	*2*
6	*7*	*2*	**1**	**9**	**5**	*3*	*4*	*8*
1	**9**	**8**	*3*	*4*	*2*	*5*	**6**	*7*
8	*5*	*9*	*7*	**6**	*1*	*4*	*2*	**3**
4	*2*	*6*	**8**	*5*	**3**	*7*	*9*	**1**
7	*1*	*3*	*9*	**2**	*4*	*8*	*5*	**6**
9	**6**	*1*	*5*	*3*	*7*	**2**	**8**	*4*
2	*8*	*7*	**4**	**1**	**9**	*6*	*3*	**5**
3	*4*	*5*	*2*	**8**	*6*	*1*	**7**	**9**

Tablica 5: Rješenje

5. Izrada Sudoku-a u SWISH-u

Ovo poglavlje posvećeno je izradi logičke igre Sudoku u SWISH-u. Kako bi se najbolje objasnio način programiranja igre, koraci izrade redom su prikazani uz odgovarajući isječak programskog koda. Na kraju je cijeli kod prikazan odjednom zbog usporedbe s drugim primjerima kreiranim u programskom jeziku Mercury i B-Prolog.

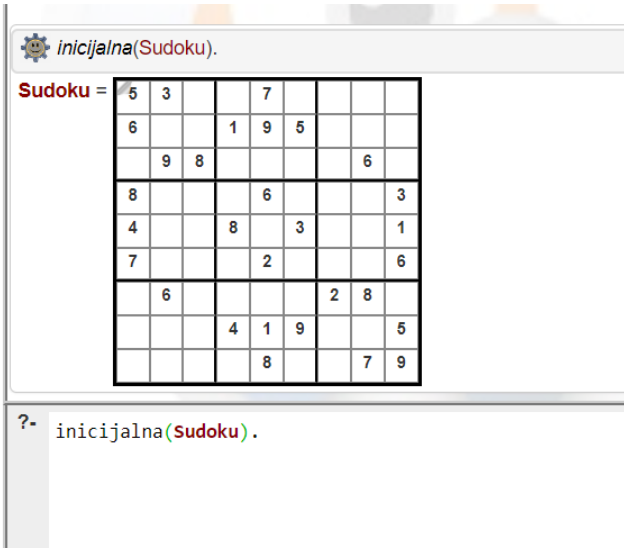
Kako već postoji primjer programskog koda za rješavanje Sudoku-a u SWISH-u, taj dio je preuzet sa interneta i implementiran u ostatak koda.

5.1. Izrada programskog koda

Prvi korak bio je kreirati matricu 9x9 koja se popunjava inicijalnim brojevima i tako kreira Sudoku. Kao početnu zagonetku odabrali smo primjer koji je, u prethodnom poglavlju, riješen. Vizualni prikaz Sudoku-a moguć je uz korištenje Prologovog dodatka use_rendering(sudoku). Ispod je prikazana baza znanja koja se sastoji od uključivanja dodatka za vizualni prikaz i klauzule kojoj je argument lista od devet elemenata od kojih je svaki element posebna lista od isto devet elemenata. Na taj način dobili smo matricu koja je kompatibilna sa uključenim dodatkom.

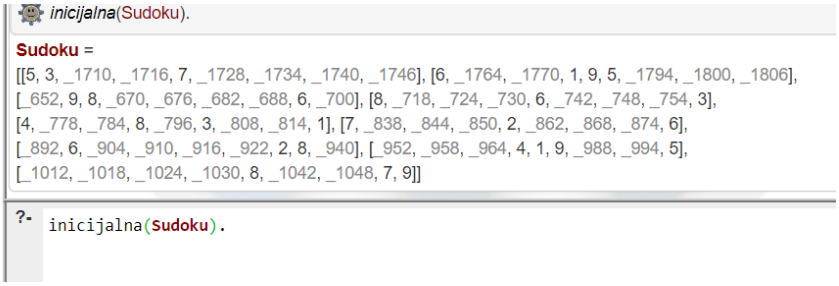
```
:- use_rendering(sudoku).

inicijalna([
    [_,_,_, _,_,_, _,_,_],
    [_,_,_, _,_,3, _,8,5],
    [_,_,1, _,2,_, _,_,_],

    [_,_,_, 5,_,7, _,_,_],
    [_,_,4, _,_,_, 1,_,_],
    [_,9,_, _,_,_, _,_,_],

    [5,_,_, _,_,_, _,7,3],
    [_,_,2, _,1,_, _,_,_],
    [_,_,_, _,4,_, _,_,9]
]).
```

U matrici prazna polja označavaju se sa specijalnim znakom „_“ jer se po takvom načinu označavanja renderira prikaz. Kada u interaktivni dio unesemo `?-inicijalna(Sudoku).` Prolog nam vrati lijepo strukturirani prikaz Sudoku-a, prikazano je na slici „Slika 9“.



Slika 9: Primjer upita 4 [autorski rad]

Nažalost, renderiranje se ne može koristiti ako se koriste predikati `write` ili `writeln` i zbog toga se u nastavku više ne može vidjeti ovakav prikaz. Zbog tog problema trebalo je naći način kako na sličan način prikazati Sudoku matricu. Prikaz matrice bez ikakve dorade prikazan je na slici 10.



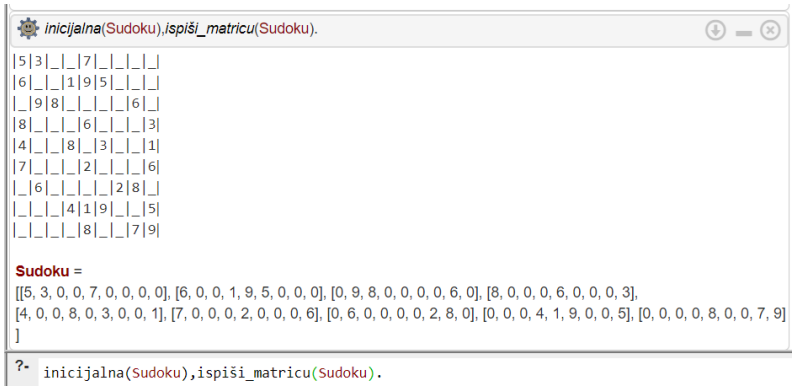
Slika 10: Prikaz nestrukturirane matrice [autorski rad]

Sljedeći isječak programskog koda prikazuje najsličniji prikaz matrice do kojeg smo uspjeli doći uz korištenje čiste sintakse Prologa.

```
ispiši_red([]) :- writeln("|").
ispiši_red([Celija|Ostalo]) :-
    (Celija = 0 -> PrikazCelije = '_'; PrikazCelije = Celija),
    format("|~w", [PrikazCelije]),
    ispiši_red(Ostalo).

ispiši_matricu([]) :- nl.
ispiši_matricu([Red|Ostalo]) :-
    ispiši_red(Red),
    ispiši_matricu(Ostalo).
```

Ovaj dio programskog koda sastoji se od rekurzivnih funkcija, klauzula `ispiši_matricu([Red|Ostalo])` strukturirana je na način da matricu, koja se šalje kao argument, rastavi na prvi red i ostatak, nakon izvršavanja klauzule `ispiši_red`, ponovno se poziva `ispiši_matricu`, ali sada za drugi red jer se prosljeđuje samo ostatak. U npr. programskom jeziku C# to bi se izvodilo kroz foreach petlju. Klauzula `ispiši_red([Celija|Ostalo])` funkcionira na isti način, samo se u ovom slučaju ne prosljeđuje dalje nego se ispisuje na ekran sa ugrađenim predikatom format. Ovdje je kreiran primjer korištenja If-Then-Else kontrolnu strukturu. `(Celija = 0 -> PrikazCelije = '_'; PrikazCelije = Celija)`, na razgovornom jeziku bi ovo bilo „Ako je vrijednost u ćeliji jednaka nuli, prikaži specijalni znak _, u suprotnom prikaži stvarnu vrijednost ćelije.“ U ovom slučaju nule nisu poželjne jer se, u nastavku, dobiva osnovna povratna informacija o mogućnosti ispunjavanja zagonetke.

Upit za testiranje ovog koda smatra se složenim upitom jer se ne postavlja samo jedna klauzula, već je potrebno prvo dohvatiti matricu koja se proslijeđuje kao argument u sljedeću klauzulu. Slika 11 prikazuje rezultat upita.



Slika 11: Primjer složenog upita [autorski rad]

Ugrađeni predikat `format/2`, prije svake ćelije ispisuje znak | i nakon toga vrijednost ćelije. Oznaka ~w znači da sljedeći argument, u ovom slučaju vrijednost ćelije, stavlja u predikat `write`.

Ovim dijelom koda završeno je početno postavljanje prikaza i matrice. Sljedeći korak je omogućavanje korisniku unos broja na željeno mjesto u matrici. Taj dio odrađen je sa ugrađenim predikatom read/1. Nakon odabira reda, stupca i broja koji se unosi, u pozadini se u, dinamičku klauzulu, nova_matrica unosi broj na definirano mjesto. Kako bi to bilo moguće bazu znanja trebalo je nadograditi sa više klauzula. U nastavku je isječak koda koji to omogućava.

```
zamijeni(X, Y, Z) :-
    nova_matrica(L),
    retractall(nova_matrica(_)),
    same_length(L, R),
    append(RowPfx, [Red|RowSfx], L),
    length(RowPfx, X),
    append(ColPfx, [_|ColSfx], Red),
    length(ColPfx, Y),
    (Z = '_' -> NoviBroj = 0; NoviBroj = Z),
    append(ColPfx, [NoviBroj|ColSfx], NoviRed),
    append(RowPfx, [NoviRed|RowSfx], R),
    assertz(nova_matrica(R)),
    ispiši_matricu(R),
    riješi.

dodaj :-
    inicijalna(Sudoku),
    ispiši_matricu(Sudoku),
    repeat,
    write('Red: '),
    read(Red),
    write('Stupac: '),
    read(Stupac),
    write('Broj: '),
    read(Broj),
    NoviRed is Red - 1,
    NoviStupac is Stupac - 1,
    zamijeni(NoviRed, NoviStupac, Broj),
    nl,
    kraj(Red).

kraj(kraj) :- writeln('Gotovo').

start :-
    inicijalna(List),
    assertz(nova_matrica(List)),dodaj.
```

U klauzuli `dodaj` korišten je još jedan ugrađeni predikat pod nazivom `repeat`. Taj predikat ponovno izvršava sve što se nalazi nakon njega. To omogućava neprekidan korisnički unos. Nakon korisničkog unosa poziva se klauzula zamijeni koja kao argumente prima unesene indekse i broj.

Prilikom zamijene praznih ćelija prvo se uzima postojeća matrica u varijablu L, nakon toga se ta postojeća matrica obriše, ugrađenim predikatom `retractall`, kako bi se kasnije mogla spremiti ažurirana matrica pod istu klauzulu bez dupliciranja redova. Sa `same_length` kreira se nova varijabla R koja je iste veličine kao i varijabla L. U nastavku, uz pomoć predikata

`append/3` uzimaju se dijelovi liste koji se mijenjaju, ovisno o X, Y i Z varijablama. X predstavlja odabrani red, Y je odabrani stupac, a Z je broj koji se postavlja na to mjesto. Predikat `assertz/1` služi za dodavanje klauzule u bazu znanja, na posljednje mjesto. Kako smo već ranije očistili novu matricu, sa ovim predikatom dodajemo novu matricu koja se nalazi u varijabli R i ta matrica, do sljedećeg unosa, ostaje zapisana u bazi znanja.

Naredbom `start` inicijalna matrica se kopira u klauzulu `nova_matrica` i tako omogućava rješavanje ove igre, nakon toga korisniku se, na ekranu, ispisuje Sudoku.

Kako bi se moglo provjeriti je li korisnik unio ispravan broj na ispravno mjesto poziva se klauzula `riješi` koja je prikazana u sljedećem isječku koda.

```
riješi :-
    nova_matrica(Puzzle),
    (sudoku(Puzzle) ->
        writeln('Moguće je riješiti zagonetku!');
        writeln('Nije moguće riješiti zagonetku.')
    ).
%programski kod nakon komentara preuzet je sa interneta

sudoku(Rows) :-
    length(Rows, 9), maplist(same_length(Rows), Rows),
    append(Rows, Vs),
    Vs ins 1..9,
    maplist(all_distinct, Rows),
    transpose(Rows, Columns),
    maplist(all_distinct, Columns),
    Rows = [A,B,C,D,E,F,G,H,I],
    blocks(A, B, C), blocks(D, E, F), blocks(G, H, I).

blocks([], [], []).
blocks([A,B,C|Bs1], [D,E,F|Bs2], [G,H,I|Bs3]) :-
    all_distinct([A,B,C,D,E,F,G,H,I]),
    blocks(Bs1, Bs2, Bs3).
```

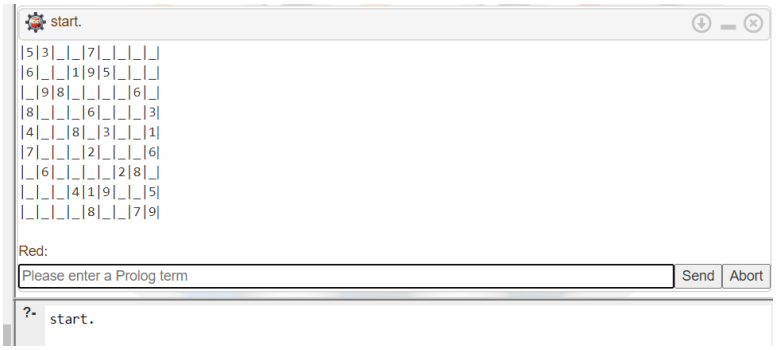
U ovom dijelu provjerava se mogućnost rješavanja nakon unosa novog broja. Ako je korisnik unio ispravan broj, ispisuje se „Moguće je riješiti zagonetku!“, u suprotnom se ispisuje „Nije moguće riješiti zagonetku.“. Kod koji je preuzet s interneta nalazi se u SWISH-u.

Klauzula `sudoku(Rows)` služi za generiranje rješenja zadane zagonetke. Izvršava se na način da prvo uzme tri grupe od po tri reda, nakon toga transponira matricu i sada su stupci zapravo postali redovi tako da opet ponovi isto grupiranje. S obzirom da ne može uzimati prva

tri elementa stupca, transponiranje matrice je idealno rješenje koje to omogućava. Uz pomoć predikata `all_distinct/1` ograničava red na jedinstvenost brojeva.

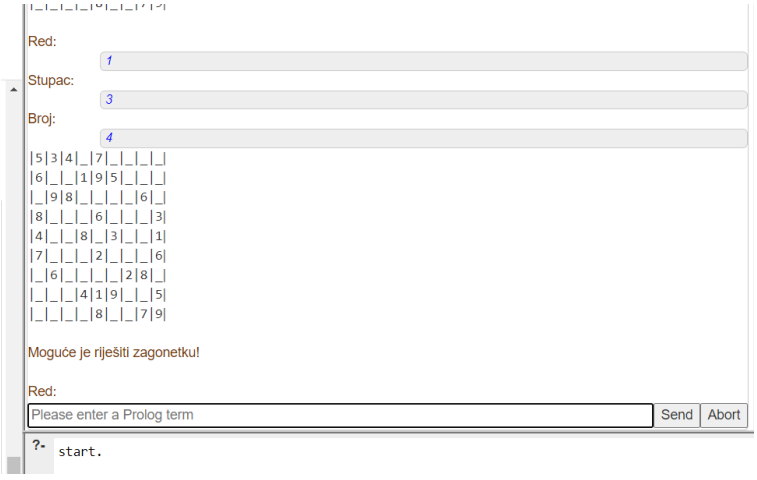
5.2. Način rješavanja zagonetke

Za pokretanje Sudoku-a potrebno je u interaktivni dio unijeti upit `?-start.`. Slika 12 prikazuje Prologov odgovor na navedeni upit.



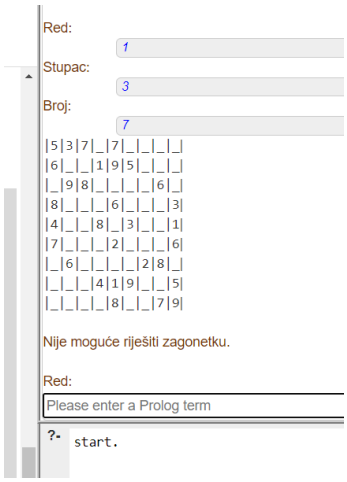
Slika 12: Primjer početka rješavanja zagonetke [autorski rad]

Zatim Prolog omogućava korisniku odabir retka u kojem želi promijeniti vrijednost. Od korisnika se očekuje da unese broj retka i pritisne gumb „Send“. Sljedeća stvar koju korisnik treba unijeti je broj stupca koji želi izmjenjeniti i za kraj treba odabrati broj koji želi unijeti na tu određenu poziciju. Ako je korisnik unio ispravan broj ispisuje mu se da je moguće riješiti zagonetku. Na slici 13 prikazan je korisnikov unos broja 4, na poziciju 1-3.



Slika 13: Primjer korisničkog unosa [autorski rad]

Ukoliko korisnik unese krivi broj na neku poziciju program mu dopušta unos novih brojeva, ali ispisuje se da je zagonetka nije rješiva. Taj primjer prikazan je na slici 14.



Slika 14: Primjer unosa krivog broja [autorski rad]

5.3. Usporedba vlastitog primjera s drugim primjerom

Programski kod ispod prikazuje vlastiti primjer Sudoku-a koji je ranije opisan.

```
:- use_rendering(sudoku).
:- use_module(library(clpfd)).

:- dynamic nova_matrica/1.

inicijalna([
    [5,3,_, _,7,_, _,_,_],
    [6,_,_, 1,9,5, _,_,_],
    [_,9,8, _,_,_, _,6,_],

    [8,_,_, _,6,_, _,_,3],
    [4,_,_, 8,_,3, _,_,1],
    [7,_,_, _,2,_, _,_,6],

    [_,6,_, _,_,_, 2,8,_],
    [_,_,_, 4,1,9, _,_,5],
    [_,_,_, _,8,_, _,7,9]
]).

zamijeni(X, Y, Z) :-
    nova_matrica(L),
    retractall(nova_matrica(_)),
    same_length(L, R),
    append(RowPfx, [Red|RowSfx], L),
    length(RowPfx, X),
    append(ColPfx, [_|ColSfx], Red),
    length(ColPfx, Y),
```

```
    (Z = '_' -> NoviBroj = 0; NoviBroj = Z),
    append(ColPfx, [NoviBroj|ColSfx], NoviRed),
    append(RowPfx, [NoviRed|RowSfx], R),
    assertz(nova_matrica(R)),
    ispiši_matricu(R),
    riješi.

dodaj :-
    inicijalna(Sudoku),
    ispiši_matricu(Sudoku),
    repeat,
    write('Red: '),
    read(Red),
    write('Stupac: '),
    read(Stupac),
    write('Broj: '),
    read(Broj),
    NoviRed is Red - 1,
    NoviStupac is Stupac - 1,
    zamijeni(NoviRed, NoviStupac, Broj),
    nl,
    kraj(Red).

kraj(kraj) :- writeln('Gotovo'),!.

start :-
    inicijalna(List),
    assertz(nova_matrica(List)),
    dodaj.

ispiši_red([]) :- writeln("|").
ispiši_red([Celija|Ostalo]) :-
    (Celija = 0 -> PrikazCelije = '_'; PrikazCelije = Celija),
    format("|~w", [PrikazCelije]),
    ispiši_red(Ostalo).

ispiši_matricu([]) :- nl.
ispiši_matricu([Red|Ostalo]) :-
    ispiši_red(Red),
    ispiši_matricu(Ostalo).

riješi :-
    nova_matrica(Puzzle),
    (sudoku(Puzzle) ->
        writeln('Moguće je riješiti zagonetku!');
        writeln('Nije moguće riješiti zagonetku.')
    ).
%programski kod nakon komentara preuzet je sa interneta
```

```
sudoku(Rows) :-
    length(Rows, 9), maplist(same_length(Rows), Rows),
    append(Rows, Vs),
    Vs ins 1..9,
    maplist(all_distinct, Rows),
    transpose(Rows, Columns),
    maplist(all_distinct, Columns),
    Rows = [A,B,C,D,E,F,G,H,I],
    blocks(A, B, C), blocks(D, E, F), blocks(G, H, I).

blocks([], [], []).
blocks([A,B,C|Bs1], [D,E,F|Bs2], [G,H,I|Bs3]) :-
    all_distinct([A,B,C,D,E,F,G,H,I]),
    blocks(Bs1, Bs2, Bs3).
```

Primjer Sudoku-a izrađen u programskom jeziku Mercury prikazan je u nastavku. Autor primjera je Ralph Becket.[23]

```
%---------------------------------------------------------------------
---------%
% vim: ft=mercury ts=4 sw=4 et wm=0 tw=0
%---------------------------------------------------------------------
---------%
%
% Author: Ralph Becket
%
% Sudoku test case for the eqneq type.
%
%---------------------------------------------------------------------
---------%

:- module sudoku.
:- interface.

:- import_module io.

:- pred main(io::di, io::uo) is cc_multi.

%---------------------------------------------------------------------
---------%
%---------------------------------------------------------------------
---------%

:- implementation.

:- import_module eqneq.
:- import_module int.
:- import_module list.
```

```
:- import_module string.

:- pragma require_feature_set([trailing]).

%---------------------------------------------------------------------
---------%

main(!IO) :-
    io.command_line_arguments(Args, !IO),
    ( if Args = [InputFileName] then
        io.open_input(InputFileName, OpenResult, !IO),
        (
            OpenResult = ok(InputFile),
            io.read_file_as_string(InputFile, ReadResult, !IO),
            (
                ReadResult = ok(StartString),
                Start = string.words(StartString),
                ( if solve_sudoku(Start, Solution) then
                    write_solution(9, 1, Solution, !IO)
                else
                    io.write_string("No solution.\n", !IO)
                )
            ;
                ReadResult = error(_, Error),
                io.write_string(io.error_message(Error), !IO),
                io.nl(!IO),
                io.set_exit_status(1, !IO)
            )
        ;
            OpenResult = error(Error),
            io.write_string(io.error_message(Error), !IO),
            io.nl(!IO),
            io.set_exit_status(1, !IO)
        )
    else
        io.write_string("usage: sudoku <filename>\n", !IO),
        io.set_exit_status(1, !IO)
    ).

%---------------------------------------------------------------------
---------%

    % Solve a sudoku problem.
    %
:- pred solve_sudoku(list(string)::in, list(int)::out) is nondet.

solve_sudoku(Start, Solution) :-
    % The following if-then-else is used to ensure that the call to
```

```
        % label_board/2 occurs after the constraints have been posted.
        % (In principle the compiler could reorder the code so that the
call
        % to label_board/2 occurs before the constraints have been posted.)
        ( if
            % Set up the board.
            init_board(Start, Board),

            Board = [X11, X12, X13,  X14, X15, X16,  X17, X18, X19,
                     X21, X22, X23,  X24, X25, X26,  X27, X28, X29,
                     X31, X32, X33,  X34, X35, X36,  X37, X38, X39,

                     X41, X42, X43,  X44, X45, X46,  X47, X48, X49,
                     X51, X52, X53,  X54, X55, X56,  X57, X58, X59,
                     X61, X62, X63,  X64, X65, X66,  X67, X68, X69,

                     X71, X72, X73,  X74, X75, X76,  X77, X78, X79,
                     X81, X82, X83,  X84, X85, X86,  X87, X88, X89,
                     X91, X92, X93,  X94, X95, X96,  X97, X98, X99],

            % The digits in each row must be different.
            eqneq.all_different([X11, X12, X13, X14, X15, X16, X17, X18,
X19]),
            eqneq.all_different([X21, X22, X23, X24, X25, X26, X27, X28,
X29]),
            eqneq.all_different([X31, X32, X33, X34, X35, X36, X37, X38,
X39]),
            eqneq.all_different([X41, X42, X43, X44, X45, X46, X47, X48,
X49]),
            eqneq.all_different([X51, X52, X53, X54, X55, X56, X57, X58,
X59]),
            eqneq.all_different([X61, X62, X63, X64, X65, X66, X67, X68,
X69]),
            eqneq.all_different([X71, X72, X73, X74, X75, X76, X77, X78,
X79]),
            eqneq.all_different([X81, X82, X83, X84, X85, X86, X87, X88,
X89]),
            eqneq.all_different([X91, X92, X93, X94, X95, X96, X97, X98,
X99]),

            % The digits in each column must be different.
            eqneq.all_different([X11, X21, X31, X41, X51, X61, X71, X81,
X91]),
            eqneq.all_different([X12, X22, X32, X42, X52, X62, X72, X82,
X92]),
            eqneq.all_different([X13, X23, X33, X43, X53, X63, X73, X83,
X93]),
            eqneq.all_different([X14, X24, X34, X44, X54, X64, X74, X84,
X94]),
            eqneq.all_different([X15, X25, X35, X45, X55, X65, X75, X85,
X95]),
```

```
            eqneq.all_different([X16, X26, X36, X46, X56, X66, X76, X86,
X96]),
            eqneq.all_different([X17, X27, X37, X47, X57, X67, X77, X87,
X97]),
            eqneq.all_different([X18, X28, X38, X48, X58, X68, X78, X88,
X98]),
            eqneq.all_different([X19, X29, X39, X49, X59, X69, X79, X89,
X99]),

            % The digits in each subsquare must be different.
            eqneq.all_different([X11, X12, X13, X21, X22, X23, X31, X32,
X33]),
            eqneq.all_different([X14, X15, X16, X24, X25, X26, X34, X35,
X36]),
            eqneq.all_different([X17, X18, X19, X27, X28, X29, X37, X38,
X39]),
            eqneq.all_different([X41, X42, X43, X51, X52, X53, X61, X62,
X63]),
            eqneq.all_different([X44, X45, X46, X54, X55, X56, X64, X65,
X66]),
            eqneq.all_different([X47, X48, X49, X57, X58, X59, X67, X68,
X69]),
            eqneq.all_different([X71, X72, X73, X81, X82, X83, X91, X92,
X93]),
            eqneq.all_different([X74, X75, X76, X84, X85, X86, X94, X95,
X96]),
            eqneq.all_different([X77, X78, X79, X87, X88, X89, X97, X98,
X99])
        then
            % Assign a digit to each square on the board.
            label_board(Board, Solution)
        else
            false
        ).

    %-------------------------------------------------------------------
---------%

        % Convert a board description into a board representation. Each
"word" in
        % the board description is either a digit, in which case we fix
that board
        % entry in the representation, or a non-digit, in which case we
leave that
        % board entry unconstrained.
        %
    :- pred init_board(list(string)::in, list(eqneq(int))::oa) is semidet.

    init_board([], []).
    init_board([Start | Starts], [EqNeq | EqNeqs]) :-
        eqneq.new(EqNeq),
```

```
        ( if string.to_int(Start, X) then
            eqneq.bind(EqNeq, X)
        else
            true
        ),
        init_board(Starts, EqNeqs).

    %-----------------------------------------------------------------------
---------%

        % Assign a digit to each square on the board.
        %
    :- pred label_board(list(eqneq(int))::ia, list(int)::out) is nondet.

    label_board([], []).
    label_board([EqNeq | EqNeqs], [X | Xs]) :-
        ( X = 1 ; X = 2 ; X = 3
        ; X = 4 ; X = 5 ; X = 6
        ; X = 7 ; X = 8 ; X = 9
        ),
        eqneq.bind(EqNeq, X),
        label_board(EqNeqs, Xs).

    %-----------------------------------------------------------------------
---------%

        % Pretty-print a solution.
        %
    :- pred write_solution(int::in, int::in, list(int)::in, io::di, io::uo)
is det.

    write_solution(_, _, [], !IO) :-
        io.nl(!IO).
    write_solution(N, R, [X | Xs], !IO) :-
        ( if N = 0 then
            io.nl(!IO),
            ( if (R mod 3) = 0 then
                io.nl(!IO)
            else
                true
            ),
            write_solution(9, R + 1, [X | Xs], !IO)
        else
            io.write_int(X, !IO),
            io.write_char(' ', !IO),
            ( if (N mod 3) = 1 then
                io.write_char(' ', !IO)
            else
```

```
                true
            ),
            write_solution(N - 1, R + 1, Xs, !IO)
        ).

    %-----------------------------------------------------------------------
---------%
    :- end_module sudoku.
    %-----------------------------------------------------------------------
---------%
```

Treći primjer prikazuje rješavanje Sudoku-a u B-Prologu. Kod je kreiran 1996. godine, a njegov autor je Neng-Fa Zhou.[8]

```
    %   File   : sudoku81.pl (in B-Prolog)
    %   Author : Neng-Fa ZHOU
    %   Date   : 1996
    %   Purpose: solve a Japanese arithmetic puzzle (9*9)

    top:-
        vars(Vars),
        labeling(Vars),
        display_board(Vars).


    go:-
        statistics(runtime,[Start|_]),
        top,
        statistics(runtime,[End|_]),
        T is End-Start,
        write('execution time is '),write(T), write(milliseconds),nl.

    vars(Vars):-
        Vars=[A11,A12,A13,B11,B12,B13,C11,C12,C13,
            A21,A22,A23,B21,B22,B23,C21,C22,C23,
            A31,A32,A33,B31,B32,B33,C31,C32,C33,
            D11,D12,D13,E11,E12,E13,F11,F12,F13,
            D21,D22,D23,E21,E22,E23,F21,F22,F23,
            D31,D32,D33,E31,E32,E33,F31,F32,F33,
            G11,G12,G13,H11,H12,H13,I11,I12,I13,
            G21,G22,G23,H21,H22,H23,I21,I22,I23,
            G31,G32,G33,H31,H32,H33,I31,I32,I33],
        Vars in 1..9,
        A12=6,B11=2,B13=4,C12=5,
        A21=4,A22=7,B22=6,C22=8,C23=3,
        A33=5,
        B32=7,
```

```
    C31=1,
    D11=9,E11=1,E13=3,F13=2,
    D22=1,D23=2,F23=9,
    D31=6,E31=7,E33=9,F33=8,
    G13=6,H12=8,I11=7,
    G21=1,G22=4,H22=9,I22=2,I23=5,
    G32=8,H31=3,H33=5,I32=9,
    % block
    alldifferent([A11,A12,A13,A21,A22,A23,A31,A32,A33]),
    alldifferent([B11,B12,B13,B21,B22,B23,B31,B32,B33]),
    alldifferent([C11,C12,C13,C21,C22,C23,C31,C32,C33]),
    alldifferent([D11,D12,D13,D21,D22,D23,D31,D32,D33]),
    alldifferent([E11,E12,E13,E21,E22,E23,E31,E32,E33]),
    alldifferent([F11,F12,F13,F21,F22,F23,F31,F32,F33]),
    alldifferent([G11,G12,G13,G21,G22,G23,G31,G32,G33]),
    alldifferent([H11,H12,H13,H21,H22,H23,H31,H32,H33]),
    alldifferent([I11,I12,I13,I21,I22,I23,I31,I32,I33]),
    % horizontal
    alldifferent([A11,A12,A13,B11,B12,B13,C11,C12,C13]),
    alldifferent([A21,A22,A23,B21,B22,B23,C21,C22,C23]),
    alldifferent([A31,A32,A33,B31,B32,B33,C31,C32,C33]),
    alldifferent([D11,D12,D13,E11,E12,E13,F11,F12,F13]),
    alldifferent([D21,D22,D23,E21,E22,E23,F21,F22,F23]),
    alldifferent([D31,D32,D33,E31,E32,E33,F31,F32,F33]),
    alldifferent([G11,G12,G13,H11,H12,H13,I11,I12,I13]),
    alldifferent([G21,G22,G23,H21,H22,H23,I21,I22,I23]),
    alldifferent([G31,G32,G33,H31,H32,H33,I31,I32,I33]),
    % vertical
    alldifferent([A11,A21,A31,D11,D21,D31,G11,G21,G31]),
    alldifferent([A12,A22,A32,D12,D22,D32,G12,G22,G32]),
    alldifferent([A13,A23,A33,D13,D23,D33,G13,G23,G33]),
    alldifferent([B11,B21,B31,E11,E21,E31,H11,H21,H31]),
    alldifferent([B12,B22,B32,E12,E22,E32,H12,H22,H32]),
    alldifferent([B13,B23,B33,E13,E23,E33,H13,H23,H33]),
    alldifferent([C11,C21,C31,F11,F21,F31,I11,I21,I31]),
    alldifferent([C12,C22,C32,F12,F22,F32,I12,I22,I32]),
    alldifferent([C13,C23,C33,F13,F23,F33,I13,I23,I33]).


display_board([A11,A12,A13,B11,B12,B13,C11,C12,C13,
               A21,A22,A23,B21,B22,B23,C21,C22,C23,
               A31,A32,A33,B31,B32,B33,C31,C32,C33,
               D11,D12,D13,E11,E12,E13,F11,F12,F13,
               D21,D22,D23,E21,E22,E23,F21,F22,F23,
               D31,D32,D33,E31,E32,E33,F31,F32,F33,
               G11,G12,G13,H11,H12,H13,I11,I12,I13,
               G21,G22,G23,H21,H22,H23,I21,I22,I23,
               G31,G32,G33,H31,H32,H33,I31,I32,I33]):-
```

```
write([A11,A12,A13,B11,B12,B13,C11,C12,C13]),nl,
write([A21,A22,A23,B21,B22,B23,C21,C22,C23]),nl,
write([A31,A32,A33,B31,B32,B33,C31,C32,C33]),nl,
write([D11,D12,D13,E11,E12,E13,F11,F12,F13]),nl,
write([D21,D22,D23,E21,E22,E23,F21,F22,F23]),nl,
write([D31,D32,D33,E31,E32,E33,F31,F32,F33]),nl,
write([G11,G12,G13,H11,H12,H13,I11,I12,I13]),nl,
write([G21,G22,G23,H21,H22,H23,I21,I22,I23]),nl,
write([G31,G32,G33,H31,H32,H33,I31,I32,I33]),nl.
```

	SWISH	**Mercury**	**B-Prolog**
Kompleksnost koda	Jednostavan	Složeniji	Jednostavan
Čistoća koda	Dobra	Dobra	Loša
Dodaci	use-rendering, library(clpfd)	io, eqneq, int, list, string	Nema
Definiranje ograničenja	Preko petlje	Zasebno po redu	Zasebno po redu
Ispis	write, writeln	write_string, write_int, write_char	write

Tablica 6: Usporedba primjera [autorski rad]

Ova tri primjera znatno se razlikuju, ne samo po sintaksi, već i po načinu pisanja. Ukoliko samo pogledamo kompleksnost, odnosno složenost koda, možemo primijetiti da su SWISH i B-Prolog primjeri vrlo jednostavni, u odnosu na primjer napisan u Mercury-u. S obzirom da Mercury koristi mnoge dodatke, važno je da programer zna koji dodatak može ili treba iskoristiti. U B-Prolog primjeru ne koriste se nikakvi dodaci i zbog toga se programski kod lako čita.

O čistoći koda moglo bi se napraviti poseban rad, ali mi se ovdje dotičemo samo nekih malih dijelova. Kod napisan u Mercury-u ima dosta komentara što je poželjno ukoliko je kod promjenjiv, ali taj primjer nema najčišće definirana ograničenja. Primjer u SWISH-u ima ograničenja definirana u petlji, što je puno čitljivije i urednije u odnosu na zasebno definirana ograničenja. Primjer iz B-Prologa nije baš reprezentativni primjer čistog koda. Ograničenja su bespotrebno zakomplicirana.

U Becketovom primjeru, napisanom u jeziku Mercury-u, postoje posebni predikati za ispisivanje stringova, cijelih brojeva i znakova. Prolog sve ispisuje preko predikata `write` ili `writeln`, što se može vidjeti u oba primjera pisana u Prologu. U suštini sva tri programa imaju istu namjenu.

6. Zaključak

SWISH okruženje najčešće se koristi za edukacije. S obzirom da se temelji na SWI-Prolog implementaciji, sadrži većinu njegovih mogućnosti. SWI-Prolog je preuzeo ISO-Prolog standardni jezik i zbog toga je kompatibilan s nekim drugim implementacijama Prologa. Na primjer, B-Prolog se također temelji na ISO-Prolog-u i ove dvije implementacije su kompatibilne, dok s druge strane Visual Prolog nije preuzeo ISO-Prolog standardni jezik i nije kompatibilan s ostalim navedenim implementacijama.

Logička igra Sudoku ima bogatu povijest u kojoj je postojalo više različitih oblika matrice. Od početaka, kada je dimenzija matrice bila 4x4, pa sve do današnjih primjera u kojima se povezuje više matrica u jednu zagonetku.

Izrada Sudoku-a u SWISH-u zahtijevala je dosta istraživanja. Najviše vremena oduzelo je isprobavanje raznih načina prikaza od kojih je samo trenutni prikaz zadovoljavajući, ali ne i idealan. Kroz pisanje programskog koda moglo se primjetiti kako SWISH ima nekih nedostataka, ali kada se sve zbroji i oduzme, jako je kvalitetan alat za učenje Prolog-a. Usporedbom s drugim programskim jezicima dobije se bolji dojam koliko slični jezici mogu biti različiti. Mercury je zanimljiv programski jezik koji može stvarati kompleksne programe. Prolog je dovoljan kao samostalan jezik, ali ukoliko se želi kreirati kompliciraniji program sa grafičkim sučeljem, potrebno je koristiti ekstenzije i povezivati ga sa drugim programskim jezicima.

Vjerujem da se, uz daljnje razvijanje, ovaj primjer Sudoku-a u SWISH-u može dovesti do visoke razine kvalitete.

Popis literature

[1] J. Novotny, „A Guide to Understanding Logic Programming“, 4.travnja 2023. Dostupno: https://www.linode.com/docs/guides/logic-programming-languages/ [pristupano 19.8.2023.].

[2] A.I. for anyone, „Prolog“ (bez dat.) Dostupno: https://www.aiforanyone.org/glossary/prolog [pristupano 18.8.2023.].

[3] C. Calapini, „Introduction to Prolog: A Programming Language for Artificial Intelligence“, 3.prosinca 2022. Dostupno: https://blog.devgenius.io/introduction-to-prolog-a-programming-language-for-artificial-intelligence-320b75455381 [pristupano 21.8.2023.].

[4] Wikipedia, „Prolog“, (bez dat.) Dostupno: https://en.wikipedia.org/wiki/Prolog [pristupano 15.8.2023.].

[5] SWI-Prolog, „SWI-Prolog's features“, (bez dat.) Dostupno: https://www.swi-prolog.org/features.html [pristupano 20.8.2023.]

[6] SWISH, „Read and write“, (bez dat.) Dostupno: https://swish.swi-prolog.org/example/examples.swinb [pristupano 21.8.2023.]

[7] Wikipedia, „B-Prolog“, (bez dat.) Dostupno: https://en.wikipedia.org/wiki/B-Prolog [pristupano 20.8.2023.].

[8] „B-Prolog“, (bez dat.) Dostupno: https://www.picat-lang.org/bprolog/ [pristupano 25.8.2023.]

[9] Visual Prolog, „Visual Prolog“, (bez dat.) Dostupno: https://www.visual-prolog.com/ [pristupano 26.8.2023.]

[10] Visual Prolog, „Tower of Hanoi (Dynamic Programming)“, (bez dat.) Dostupno: https://wiki.visual-prolog.com/index.php?title=Tower_of_Hanoi_(Dynamic_Programming) [pristupano 26.8.2023.]

[11] Wikipedia, „Mercury (programming language)“, (bez dat.) Dostupno: https://en.wikipedia.org/wiki/Mercury_(programming_language) [pristupano 16.8.2023.]

[12] Mercury, „About Mercury“, (bez dat.) Dostupno: https://www.mercurylang.org/about.html [pristupano 16.8.2023.]

[13] juliensf, [primjer koda], (bez dat.) Dostupno: https://github.com/Mercury-Language/mercury/blob/master/samples/csharp_interface/short_example.m [pristupano 16.8.2023.]

[14] F. Alberi, „An introduction to Datalog“, (23. prosinca 2022.) Dostupno: https://blogit.michelin.io/an-introduction-to-datalog/ [pristupano 16.8.2023.]

[15] M. Rouse, „Datalog“, (11. svibnja 2015.) Dostupno: https://www.techopedia.com/definition/3915/datalog [pristupano 16.8.2023.]

[16] „Datomic Queries and Rules“, (bez dat.) Dostupno: https://docs.datomic.com/pro/query/query.html [pristupano 17.8.2023.]

[17] T. Stellmach, „Sudoku puzzle with a nice layout“, [Slika] (bez dat.) Dostupno: https://en.wikipedia.org/wiki/Sudoku#/media/File:Sudoku_Puzzle_by_L2G-20050714_standardized_layout.svg [pristupano 27.8.2023.]

[18] „History of Sudoku“, (bez dat.) Dostupno: https://www.sudokuonline.io/tips/history-of-sudoku#:~:text=The%20history%20of%20Sudoku%20began,be%20used%20in%20statistical%20analysis. [pristupano 27.8.2023.]

[19] „Latin Squares – *History of Sudoku*“ [Slika] (bez dat.) Dostupno: https://www.sudokudragon.com/sudokuhistory.htm [pristupano 27.8.2023.]

[20] „The History of Sudoku“, (bez dat.) Dostupno: https://sudoku.com/how-to-play/the-history-of-sudoku/ [pristupano 27.8.2023.]

[21] J. Delahaye, „The Science Behind SudoKu“, (2006.) Dostupno: https://www.cs.virginia.edu/~robins/The_Science_Behind_SudoKu.pdf [pristupano 28.8.2023.]

[22] J. Calmes i sur., „Sudoku“, (bez dat.) Dostupno: https://brilliant.org/wiki/sudoku/#:~:text=Sudoku%20is%20a%20logic%2Dbased,in%20relation%20to%20one%20another. [pristupano 28.8.2023.]

[23] R. Becket, „Sudoku test case for the eqneq type“, (bez dat.) Dostupno: https://github.com/Mercury-Language/mercury/blob/master/samples/solver_types/sudoku.m [pristupano 29.8.2023.]

Popis slika

Popis tablica