Umjetna inteligencija

Ekspertni sustavi

Marko Čupić

IZDAVAČ

JAVNO DOSTUPNO NA WEB STRANICI JAVA.ZEMRIS.FER.HR/NASTAVA/UI

Ovo je popratni materijal za kolegij Umjetna inteligencija na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu. Tekst je usklađen s prezentacijama koje se koriste na tom kolegiju i okvirno ih prati. Uz tekst je pripremljena implementacija jednostavne ljuske ekspertnog sustava: *ŠES - Školski Ekspertni Sustav* zajedno s primjerima koji su dani u službenoj prezentaciji na kolegiju Umjetna inteligencija. Čitatelj se upućuje da istu skine te isproba primjere koji se obrađuju u tekstu.

Prvo izdanje, travanj 2020.

Sadržaj

1 Uvod 5

2 Građa ekspertnog sustava 9

2.1 Zaključivanje u ekspertnom sustavu **11**

2.1.1 Ulančavanje unaprijed 11

2.1.2 Ulančavanje unatrag 14

2.1.3 Primjer 1: voće 18

2.1.4 Primjer 2 22

Bibliografija 29

Knjige **29**

Članci **29**

Konferencijski radovi i ostalo **29**

1. Uvod

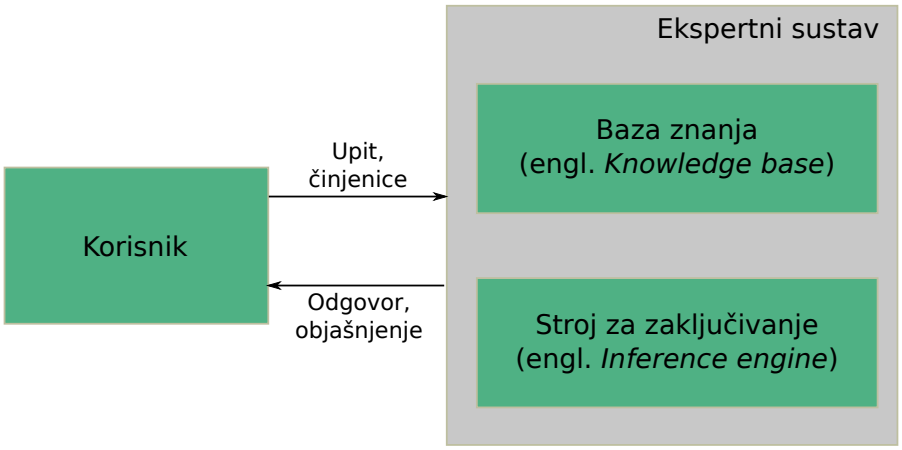
Ekspertni sustavi su programski sustavi koji pomažu ljudima u rješavanju složenih problema, a rad im se zasniva na emulaciji načina na koji ljudski eksperti donose odluke pri rješavanju tih problema. Ekspertni sustavi nastali su relativno rano (60. i 70. godine prošlog stoljeća) kao pokušaj ljudi da izgrade programske sustave koji se ponašaju inteligentno. Ideja je bila izgraditi programske sustave koji će iskazivati svojstva općenite inteligencije i biti primjenjivi na vrlo širok skup problema. To se, međutim, do danas nije ostvarilo jer se pokazalo da takve sustave još uvijek ne znamo graditi. Niša u kojoj su se ekspertni sustavi pokazali izuzetno uspješnima su situacije u kojima je problemska domena zatvorena i usko specijalizirana: u takvom okruženju ne javljaju se problemi koje susrećemo pri pokušaju izgradnje općenito inteligentnih sustava te je stoga moguće izgraditi programske sustave koji su vrlo korisni.

Pojednostavljen prikaz strukture ekspertnog sustava dan je na slici 1.1. Korisnik sustavu postavlja osnovno pitanje ili nudi početni niz poznatih činjenica i na zahtjev ekspertnog sustava može odgovarati na dodatna pitanja. Sam ekspertni sustav sastoji se od **baze znanja** (engl. *knowlegde base*) te **stroja za zaključivanje** (engl. *inference engine*).

Ekspertni sustavi temeljeni na pravilima još se nazivaju i *produkcijski ekspertni sustavi*. Takvi sustavi temeljeni su na nizu AKO-ONDA pravila te mehanizmu zaključivanja. njihova baza znanja sastavljena je od baze pravila te radne memorije u kojoj su pohranjene činjenice. Primjerice, zamislimo jedan produkcijski sustav izgrađen od sljedeća dva pravila:

```
[1] AKO je B istinit TADA je C istinit
[2] AKO je A istinit TADA je B istinit
```

`A`, `B` i `C` nazivamo **činjenice** i ovom primjeru to su logičke varijable (vrijednost im može biti istina ili laž). Korisnik bi ovaj ekspertni sustav mogao pitati "je li `C` istinit"? Ekspertni sustav mogao bi analizom pravila vidjeti da postoji pravilo `[1]` koje mu omogućava dokazivanje `C`-a, te da je prema tom pravilu `C` istinit ako je `B` istinit. Stoga bi sustav dalje trebao pokušati odrediti je li `B` istinit. Prema pravilu `[2]` on je istinit ako je `A` istinit pa je to sljedeći korak koji treba istražiti. Kako u bazi nema pravila koje bi omogućilo dokazivanje istinitosti od `A`, sustav sada može interaktivno pitati korisnika je li `A` istinit. Ako korisnik kaže da je, sustav dalje može zaključiti da je prema `[2]` i `B` istinit pa na temelju toga i pravila `[1]` da je i `C` istinit.



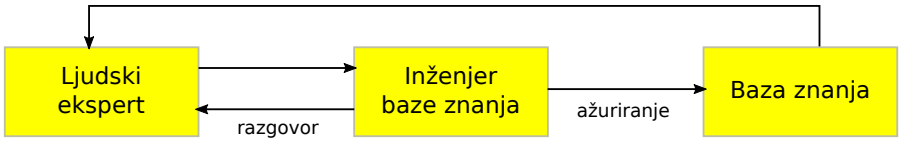
Slika 1.1: Pojednostavljen prikaz strukture ekspertnog sustava.

Uočite u prethodnom scenariju kako sustav na temelju postojećih pravila ista može kombinirati kako bi dokazao određenu činjenicu, odnosno kada za nešto nema pravilo, može interaktivno pitati korisnika da nešto provjeri. Također primijetite da sustav razlikuje dvije bitne situacije:

- situaciju u kojoj sustav ne zna istinitost neke činjenice (npr. `A`) te
- situaciju u kojoj sustav zna istinitost činjenice, a ta onda može biti (u našem primjeru) "istinita je" ili "nije istinita".

Spomenimo da razvoj ekspertnog sustava nije jednostavan zadatak, posebice za korisnike koji nisu posebno obrazovani za takve ili slične zadatke. Kako bi se olakšao prodor ekspertnih sustava u različita područja, prvi iskorak napravljen je kroz razvoj takozvanih ljuski ekspertnih sustava. **Ljuska ekspertnog sustava** je ekspertni sustav s praznom bazom znanja; radi se o čitavoj programskoj implementaciji koja je spremna za uporabu čim se napravi specijalizacija za određenu problemsku domenu punjenjem baze znanja.

Kako do nedavno uporaba računala nije bila raširena u mnogim granama ljudske djelatnosti, čak i prilagodba ljuski ekspertnih sustava pripremom baze znanja često je predstavljala nepremostivu prepreku. Stoga je uobičajena situacija da taj posao radi **inženjer baze znanja** - kako je ilustrirano na slici 1.2. Inženjer baze znanja kroz razgovor i interakciju s ljudskim ekspertom za problemsku domenu prikuplja relevantno znanje i oblikuje ga kao AKO-ONDA pravila koja upisuje u bazu znanja. Ljudski ekspert uvidom u trenutno stanje baze znanje kroz interakciju s inženjerom baze znanja osigurava razvoj ekspertnog sustava koji u konačnici može biti primijenjen na konkretnu problemsku domenu.



Slika 1.2: Razvoj baze znanja.

Spomenimo u nastavku i tri povijesno interesantnijih ekspertnih sustava redoslijedom kojim su nastajali.

1965. godine na sveučilištu Stanford je napravljen sustav DENDRAL. Bio je to ekspertni sustav čija je zadaća bila otkriti strukturu (građu) molekula na temelju informacija o atomima koji su prisutni te spektrografskim podatcima.

Ranih sedamdesetih razvijen je i sustav PROSPECTOR (*Stanford Research Institute*). Zadaća mu je bila pomoći geolozima u istraživanju mineralnih ruda. Ideja je bila predvidjeti izglednost da će neko geografsko područje sadržavati mineralne rude kako bi se procijenilo isplati li se uložiti u bušenje na tom mjestu.

Ranih sedamdesetih na sveučilištu Stanford je napravljen sustav MYCIN[1] - ekspertni sustav koji je pomagao u bakterijskih infektivnih bolesti te bolesti grušanja krvi. Sustav je imao bazu s oko šestotinjak pravila te je kroz postavljanje pitanja operateru na kraju davao niz hipoteza (o kojoj se bolesti radi), objašnjenje za te hipoteze (zašto sustav misli da se radi o toj bolesti) te pouzdanost zaključka (koliko je sustav siguran u tu hipotezu). Pravila su definirala zaključke uz određenu razinu pouzdanosti koja je bila definirana faktorom pouzdanosti (engl. *certainty factor*). Pozitivan faktor pouzdanosti (primjerice, 0.9) išao je u prilog zaključku dok je negativan faktor sigurnosti (primjerice, -0.7) osporavao zaključak. Ako je neko pravilo definiralo određen zaključak (primjerice, bakterija je "E. Coli") s faktorom pouzdanosti X, pa je u nekom kasnijem trenutku neko drugo pravilo definiralo isti zaključak s faktorom pouzdanosti Y, konačan faktor pouzdanosti zaključka bio je kombinacija tih faktora prema izrazu:

$$CF(X,Y) = \begin{cases} X+Y-X\cdot Y & \text{ako su } X,Y>0, \\ X+Y+X\cdot Y & \text{ako su } X,Y<0, \\ \frac{X+Y}{1-\min(|X|,|Y|)} & \text{inače.} \end{cases}$$

Ekspertni sustavi općenito spadaju u nemonotone sustave te mogu raditi s nepreciznim znanjem. Pojam nemonoton ovdje se odnosi na činjenicu da takve sustave možemo opisati nemonotonom logikom. *Monotona logika* je logika kod koje zaključivanje ne može narušiti valjanost prethodno izvedenih zaključaka; postupkom zaključivanja količina izvedenog znanja sve više i više raste (pa je u tom smislu monotona). *Nemonotone logike* su logike kod kojih izvedeno pravilo može poništiti zaključke prethodno izvedenih pravila. Izgradnja takvih sustava nije jednostavna i zahtjeva opremanje ekspertnog sustava dodatnim podsustavom za održavanje istinitosti (engl. *Truth Maintanance System*) koji će se pobrinuti da iz memorije ukloni i one činjenice koje su bile izvedene na temelju činjenice koju je pravilo uklonilo.

Kada kažemo da ekspertni sustavi mogu raditi s nepreciznim znanjem, tu imamo nekoliko smjerova u kojima možemo ići, a spomenut ćemo tri. Prva mogućnost je da je sustav vjerojatnosni, pa može pratiti kolika je vjerojatnost da neka činjenica vrijedi (ili ima određenu vrijednost). Druga mogućnost bi bila da pratimo u kojoj smo mjeri sigurni u zaključke (kao što je bio slučaj kod prethodno spomenutog sustava MYCIN koji je koristio faktore pozdanosti). Treća mogućnost bila bi da radimo u okruženju u kojem činjenice u određenoj mjeri poprimaju određene vrijednosti, pa govorimo o uporabi neizrazite logike (engl. *fuzzy logic*).

[1] Zainteresiranog se čitatelja dalje upućuje na `http://people.dbmi.columbia.edu/~ehs7001/Buchanan-Shortliffe-1984/MYCIN%20Book.htm`

2. Građa ekspertnog sustava

Ekspertni sustavi koje ćemo razmatrati, kao što smo već najavili u uvodu, bit će produkcijski ekspertni sustavi koji su temeljeni na AKO-ONDA pravilima. U formalnoj logici, AKO-ONDA pravilo je zapravo implikacija ($A \rightarrow B$), pa se izvođenje znanja u produkcijskim sustavima temelji na zaključivanju *Modus Ponens*:

A

$A \rightarrow B$

B

Produkcijski ekspertni sustav stoga sadrži dvije vrste znanja: **činjenično znanje** (tipa: vrijedi A) koje se čuva u **radnoj memoriji** te **uvjetno znanje** (tipa: $A \rightarrow B$) koje se čuva u **bazi pravila**. Sadržaj radne memorije zajedno sa bazom pravila nazivamo **bazom znanja** produkcijskog sustava.

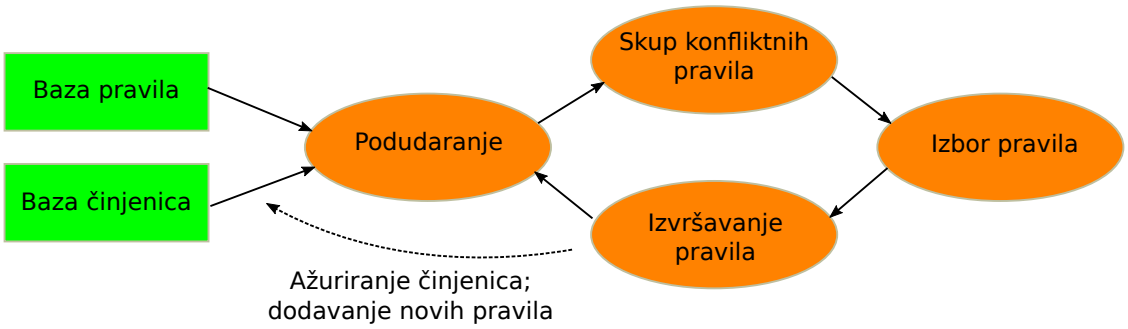
Produkcijski sustav sastoji se od tri dijela:

1. *baze pravila* koja čuva skup produkcijskih pravila,
2. *baze činjenica* odnosno radne memorije u kojoj se nalaze sve činjenice izvedene u postupku zaključivanja ili prikupljene od korisnika tijekom provedbe zaključivanja te
3. *upravljačkog mehanizma* (zovemo ga još *ciklus podudaranje-djelovanje*) koji definira način na koji se provodi zaključivanje, a temelji se na pronalaženju svih pravila koja su u skladu s činjenicama u bazi činjenica (skup tih pravila nazivamo konfliktni skup), izboru jednog od konfliktnih pravila te njegovom izvršavanju.

Njihova međusobna interakcija ilustrirana je na slici 2.1.

Sadržaj radne memorije podudara se s antecedentima pravila iz baze pravila te se gradi konfliktni skup. Ovdje pojam "konfliktni" ne znači da se radi o pravilima koja su u konfliktu sa trenutno izvedenim znanjem. Baš suprotno, to su pravila koja su u skladu s trenutnim spoznajama, a ako takvih pravila ima više, ona se natječu da ih odaberemo za izvođenje - pa od tuda pojam "konfliktni skup".

Izborom jednog od pravila iz konfliktnog skupa i njegovim izvođenjem generira se novo znanje pa se ažurira baza činjenica, a možda i baza pravila. Naime, ovisno o ekspresivnosti ljuske, ljuska bi mogla dopustiti da se izvođenjem nekog pravila u bazu pravila dodaju i nova pravila. Mi to u



Slika 2.1: Ciklus podudaranje-izvođenje

ovom tekstu nećemo razmatrati. Nakon toga, čitav se ciklus ponavlja.

U bazi činjenica, činjenice su vrlo često organizirane kao uređeni parovi (naziv,vrijednost). Ljuske ekspertnih sustava pri tome će obično ponuditi definiranje tipova podataka koji će ograničiti vrijednosti koje činjenica može poprimiti, te potom omogućiti da se činjenici definira njezin tip podataka. Osim imena i vrijednosti, mogu se pamtiti i dodatne informacije. Primjerice, ako ekspertni sustav radi s faktorima pouzdanosti, tada će se činjenice pamtiti kao uređene trojke (naziv,vrijednost,faktorPouzdanosti).

U Školskom ekspertnom sustavu koji je pripremljen uz prezentaciju i ovaj tekst, korisnički se tipovi podataka definiraju ključnom riječi `typedef` nakon čega slijedi naziv tipa, dvotočka, specifikacija tipa te točka-zarez. Specifikacija tipa može biti `Boolean`, ograničena cjelobrojna vrijednost ili pobrojana vrijednost (enumeracija). Evo primjera koji ilustrira posljednja dva.

```
typedef TBoja: enum zelena, žuta, žuto-smeđa, crvena, plava, narančasta;
typedef TBrojSjemenki: integer(1,);
```

Prethodni isječak definira tip imena `TBoja` koja može biti jedna od navedenih riječi nakon ključne riječi `enum`. Prethodni isječak definira i tip imena `TBrojSjemenki` koji je prirodni broj (odnosno integer čija je minimalna vrijednost 1, a maksimalna nije zadana.

U Školskom ekspertnom sustavu potom se definiraju sve činjenice koje mogu postojati, uporabom ključne riječi `var`.

```
var Boja: TBoja;
var Broj_sjemenki: TBrojSjemenki;
```

Primijetite da se ovime ne definira i vrijednost činjenice: samo se navodi naziv činjenice te ograničenje njezinog tipa, kako bi ekspertni sustav znao što može očekivati kao vrijednosti koje ta činjenica može poprimiti.

Konačno, u Školskom ekspertnom sustavu pravila se definiraju konstruktom AKO-ONDA. Evo primjera koji to ilustrira.

```
AKO Oblik = izdužen & Boja = (zelena | žuta) TADA Voće = banana;
AKO Oblik = (okrugli | zaobljen) & Promjer > 10 TADA Vrsta_voćke = loza;
```

Ovdje su `Oblik`, `Boja`, `Voće`, `Promjer` i `Vrsta_voćke` činjenice. Radi se jednom od primjera koji ćemo obraditi u nastavku pa će tada biti dana i njegova cjelokupna specifikacija sastavljena od svih tipova podataka, deklaracija činjenica i baze pravila. Antecedent pravila, kao što je prikazano u prethodnom isječku može sadržavati više činjenica čije vrijednosti ispituje i rezultati ispitivanja mogu biti povezani logičkim operatorima. Vrijednost koja je pridružena činjenici može se ispitivati na jednakost ili pak odnos (primjerice, veće ili manje). Ako je tip činjenice enumeracija,

u pravilu operator jednakosti s desne strane može dobiti popis kandidata (pogledajte prvo pravilo: ono ispituje ima li činjenica `Boja` vrijednost `zelena` ili pak `žuta`).

2.1 Zaključivanje u ekspertnom sustavu

Prilikom izvođenja znanja, ekspertni sustav temeljen na pravilima odgovaranje na korisnikov upit može raditi na dva dijametralno suprotna načina. Primijetite da je, pojednostavljeno govoreći, zadaća ekspertnog sustava na temelju činjenica koje je dao korisnik i niza implikacija (što su pojedina pravila) doći do odgovora na pitanje koje je postavio korisnik (odnosno "dokazati" činjenicu za koju je korisnik pitao da li vrijedi). U tom smislu kažemo da ekspertni sustav gradi lanac činjenica i zaključaka kojim se od početnih činjenica dolazi do traženog zaključka. Imajući u vidu ovakvu interpretaciju pojma "lanac", mehanizme zaključivanja nazivamo *ulančavanjima* pa su tako spomenute krajnosti:

- zaključivanje unaprijed te
- zaključivanje enatrag.

Zaključivanje unaprijed je postupak zaključivanja koji kreće od početnog danog skupa činjenica. Traže se sva pravila koja s obzirom na trenutne vrijednosti činjenica u radnoj memoriji imaju ispunjen antecedent. Ta se pravila izvršavaju i novodefinirane činjenice dodaju se u bazu činjenica. Sada se postupak ponavlja: traže se sva pravila čiji je antecedent ispunjen, ona se izvršavaju, itd. Primijetite da opisani postupak zapravo odgovara postupku pretraživanja u širinu: izvodi se sve što se može na temelju trenutnih činjenica (prva razina stabla) čime se nešto novo dodaje u bazu činjenica. Tada se ponovno izvodi sve što se može na temelju (sada) trenutnih činjenica, čime dobivamo drugu razinu stabla. Postupak se iterativno ponavlja i staje ili kad smo izveli vrijednost za činjenicu koju je korisnik tražio, ili kada se više ništa novoga ne može izvesti.

Zaključivanje unatrag je postupak zaključivanja koji kreće od ciljne činjenice; inicijalno, korisnik ne treba zadati ništa više od naziva ciljne činjenice. Sustav će potražiti sva pravila koja izvode vrijednost te činjenice (drugim riječima, pravila kojima je ta činjenica u konsekventu). Pogledat će antecedente tih pravila i činjenice koje se tamo spominju. Za takve činjenice čija vrijednost još nije poznata, rekurzivno će se potražiti pravila koja njih izvode. Ako pravilo u antecedentu ima činjenicu za koju nema pravila koje je izvodi, sustav će interaktivno pitati korisnika da unese vrijednost te činjenice. Zaključivanje unatrag odgovara postupku predraživanja u dubinu.

Pogledajmo obje vrste ulančavanja na konkretnom primjeru. Potom ćemo obraditi još nekoliko primjera. Prepišite sljedeću specifikaciju u datoteku imena `demo.txt`:

```
var A: Boolean;
var B: Boolean;
var C: Boolean;
var X: Boolean;
var Y: Boolean;
var Z: Boolean;

AKO A & B ONDA C;
AKO B ONDA X;
AKO C & X ONDA Y;
AKO Y ONDA Z;
```

2.1.1 Ulančavanje unaprijed

Uz pretpostavku da ste pripremili datoteku sa specifikacijom baze pravila i definicijom činjenica kako je prethodno opisano, pokrenite Školski ekspertni sustav uz ulančavanje unaprijed:

```
java -cp ses.jar hr.fer.zemris.ses.main.FCES demo.txt
```

Pokrenut će se program uz pozdravnu poruku:

```
*****************************************************
* Školski ekspertni sustav V1.0 - autor: Marko Čupić *
*****************************************************

Unesite činjenice, jednu po retku, u formatu IME = vrijednost.
Kada ste gotovi, unesite 'start' ili 'start NAZIV-CILJA'.
>
```

Unijet ćemo redom `A=true` i `B=true` (umjesto `true` mogli smo pisati `1`; umjesto `false` možemo pisati i `0`). Nakon svakog unosa, sustav će prikazati aktualno stanje baze činjenica. Za svaku činjenicu prikazuje njezin tip, dodijeljenu vrijednost (i u zagradi ponavlja informaciju da varijabla ima definiran sadržaj):

```
> A=true
OK. Stanje memorije je:
var A: Boolean = true (defined=true);
> B=true
OK. Stanje memorije je:
var A: Boolean = true (defined=true);
var B: Boolean = true (defined=true);
>
```

Sada možemo zatražiti pokretanje zaključivanja. To se radi naredbom `start` uz navođenje ciljne činjenice koja nas zanima. Zatražit ćemo izvođenje činjenice `Z`. Sustav će kao odgovor najprije ispisati sva pravila (i njihove redne brojeve na koje će se kasnije pozivati) te početno stanje u memoriji.

```
> start Z


Radimo sa sljedećim pravilima:
[1] AKO A & B ONDA C;
[2] AKO B ONDA X;
[3] AKO C & X ONDA Y;
[4] AKO Y ONDA Z;

Početno stanje u memoriji:
var A: Boolean = true (defined=true);
var B: Boolean = true (defined=true);
```

Nakon ovoga slijedi niz koraka. U prvom koraku, s činjenicama `A=true`, `B=true` i `C=true` podudaraju se samo pravila 1 i 2. Stoga ona čine konfliktni skup; niti jedno od tih pravila već nismo izveli, pa je skup blokiranih pravila prazan; ovo je u nastavku prikazano u prvom retku. Čitav korak prikazan je u nastavku.

```
Novi korak
============================================================
Skup konfliktnih pravila: [1, 2] \ []
Odabrano pravilo: 1
```

```
[1] AKO A & B ONDA C;

Stanje u memoriji:
var A: Boolean = true (defined=true);
var B: Boolean = true (defined=true);
var C: Boolean = true (defined=true);
```

Između pravila 1 i 2, odabrano je ono koje ima manji redni broj: 1. Stoga je pravilo 1 izvršeno, i u bazu činjenica je dodana nova činjenica: `C=true` što je vidljivo na kraju prethodnog ispisa. Pravilo 1 dodaje se u skup blokiranih pravila jer smo ga izveli. U sljedećem koraku i dalje se sa stanjem u memoriji podudaraju samo pravila 1 i 2; međutim, kako je 1 blokirano, sustav za izvođenje bira pravilo 2 te u bazu činjenica upisuje `X=true` te dodaje 2 u blokirana pravila. Ovo je prikazano u nastavku.

```
Novi korak
============================================================
Skup konfliktnih pravila: [1, 2] \ [1]
Odabrano pravilo: 2
[2] AKO B ONDA X;

Stanje u memoriji:
var A: Boolean = true (defined=true);
var B: Boolean = true (defined=true);
var C: Boolean = true (defined=true);
var X: Boolean = true (defined=true);
```

Sada se sa stanjem u memoriji podudaraju pravila 1, 2 i 3, pri čemu su 1 i 2 blokirana. Stoga sustav bira i izvodi pravilo 3 čime u bazu činjenica upisuje `Y=true` i dodaje 3 u blokirana pravila.

```
Novi korak
============================================================
Skup konfliktnih pravila: [1, 2, 3] \ [1, 2]
Odabrano pravilo: 3
[3] AKO C & X ONDA Y;

Stanje u memoriji:
var A: Boolean = true (defined=true);
var B: Boolean = true (defined=true);
var C: Boolean = true (defined=true);
var X: Boolean = true (defined=true);
var Y: Boolean = true (defined=true);
```

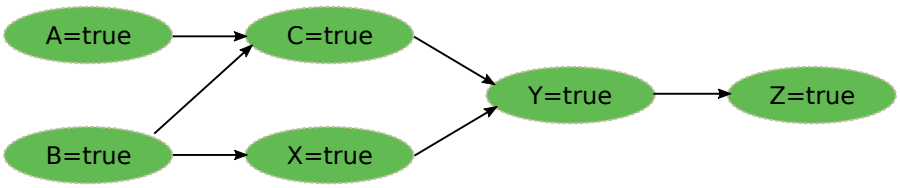
Sada se sa stanjem u memoriji podudaraju pravila 1, 2, 3 i 4, pri čemu su 1, 2 i 3 blokirana. Stoga sustav bira i izvodi pravilo 4 čime u bazu činjenica upisuje `Z=true` i dodaje 4 u blokirana pravila.

```
Novi korak
============================================================
Skup konfliktnih pravila: [1, 2, 3, 4] \ [1, 2, 3]
Odabrano pravilo: 4
[4] AKO Y ONDA Z;

Stanje u memoriji:
```

```
var A: Boolean = true (defined=true);
var B: Boolean = true (defined=true);
var C: Boolean = true (defined=true);
var X: Boolean = true (defined=true);
var Y: Boolean = true (defined=true);
var Z: Boolean = true (defined=true);
```

U ovom trenutku postupak staje jer je izvedena vrijednost za `C`.

Provedeni postupak zaključivanja izgradio je lanac prikazan na slici u nastavku.



2.1.2 Ulančavanje unatrag

Kod ulančavanja unatrag radimo pretraživanje u dubinu. Uobičajena izvedba je uporabom stoga. Ovaj algoritam stog koristi kao mjesto za pohranu činjenica čije vrijednosti treba dokazati. Postupak kreće tako da na stog gurnemo cilj koji nas zanima.

Pokrenite Školski ekspertni sustav uz ulančavanje unaprijed:

```
java -cp ses.jar hr.fer.zemris.ses.main.BCES demo.txt
```

Pokrenut će se program uz pozdravnu poruku:

```
*****************************************************
* Školski ekspertni sustav V1.0 - autor: Marko Čupić *
*****************************************************

Unesite činjenice, jednu po retku, u formatu IME = vrijednost.
Kada ste gotovi, unesite 'start' ili 'start NAZIV-CILJA'.
>
```

Sustavu nećemo unaprijed davati niti jednu činjenicu već ćemo odmah zatražiti da pokuša izvesti vrijednost za `Z` zadavanjem naredbe `start Z`.

```
> start Z
```

Sustav će ispisati bazu pravila i bazu činjenica:

```
Radimo sa sljedećim pravilima:
[1] AKO A & B ONDA C;
[2] AKO B ONDA X;
[3] AKO C & X ONDA Y;
[4] AKO Y ONDA Z;

Početno stanje u memoriji:
```

pa vidimo da je baza činjenica trenutno prazna. Prethodna naredba je na stog pohranila cilj `Z`. Stoga sustav razmatra postoji li pravilo koje izvodi `Z` i pronalazi da je to pravilo 4:

```
KORAK
==================================================
Stanje u memoriji:

Sadržaj stoga:
[Z]

Gledam pravila koja izvode cilj: Z
Skup pravila: [4] = [4] \ []
Razmatram pravilo [4] AKO Y ONDA Z;: NEDOSTAJU PODATCI!
Zakazujem novi cilj Y.
```

Pravilo 4 može dokazati `Z` preko `Y`. Stoga je sustav provjerio je li `Y` činjenica čiju vrijednost izvode neka druga pravila i utvrdio da je to istina. Stoga je na stog pohranio `Y` kao privremeno novi cilj koji treba dokazati i time završio trenutni korak.

Sada se ponovno razmatra vrh stoga (trenutno, to je `Y`; u ispisu je vrh stoga najdesniji ispisani element) i traže pravila koja ga mogu dokazati. Pronalazi se da je to pravilo 3. Bira se to pravilo i njegov antecedent te se uviđa da su potrebne vrijednosti za `C` i `X`. Provjerava se je li `C` činjenica koju izvode pravila; utvrđuje se da je, pa se `C` gura na vrh stoga kao novi privremeni cilj i time završava korak:

```
KORAK
==================================================
Stanje u memoriji:

Sadržaj stoga:
[Z, Y]

Gledam pravila koja izvode cilj: Y
Skup pravila: [3] = [3] \ []
Razmatram pravilo [3] AKO C & X ONDA Y;: NEDOSTAJU PODATCI!
Zakazujem novi cilj C.
```

Sada je `C` na vrhu stoga. Traže se pravila koje ga izvode; pronalazi se samo pravilo 1 i ono se uzima. Gleda se antecedent tog pravila, s lijeva u desno. Antecedent najprije provjerava vrijednost od `A`; stoga sustav provjerava je li to činjenica koju izvode pravila. Ovdje smo po prvi puta u situaciji da to nije istina: ne postoji niti jedno pravilo koje bi izvelo vrijednost od `A`; stoga sustav pita korisnika da unese vrijednost činjenice `A`:

```
KORAK
==================================================
Stanje u memoriji:

Sadržaj stoga:
[Z, Y, C]

Gledam pravila koja izvode cilj: C
Skup pravila: [1] = [1] \ []
Razmatram pravilo [1] AKO A & B ONDA C;: NEDOSTAJU PODATCI!
Idem pitat korisnika za A.
Unesite vrijednost za [A] koja je tipa Boolean: Boolean.
>
```

Unijet ćemo `true`. Kako unesena vrijednost zadovoljava ispitivanje u antecedentu trenutno razmatranog pravila 1, sustav utvrđuje da mu treba vrijednost i činjenice `B`. Provjera se postoji li pravilo koje izvodi tu činjenicu; kako je odgovor negativan, sustav pita korisnika da unese vrijednost za `B`:

```
> true
Idem pitat korisnika za B.
Unesite vrijednost za [B] koja je tipa Boolean: Boolean.
>
```

Unijet ćemo `true`.

```
> true
Pravilo pali. Cilj C je izveden.
```

Time se u bazi činjenica nalaze činjenice `A=true` i `B=true`. Stoga sustav utvrđuje da pravilo 1 "pali" (engl. *fires*); sustav ga izvodi čime u bazu činjenica upisuje `C=true` i pravilo dodaje u blokirana pravila. Također, `C` se skida sa stoga, tako da je sada na vrhu ponovno `Y`.

U novom koraku ponovno traže pravila koja izvode `Y`; to je pravilo 3. Gleda se njegov antecedent: vrijednost za `C` postoji u bazi činjenica i kako je `C=true` podudarno s pravilom, gleda se ostatak antecedenta i utvrđuje da je potreba vrijednost činjenice `X`. Kako za `X` postoji pravilo koje ga izvodi, `X` se gura na stog i postaje novi privremeni cilj:

```
KORAK
==================================================
Stanje u memoriji:
var A: Boolean = true (defined=true);
var B: Boolean = true (defined=true);
var C: Boolean = true (defined=true);

Sadržaj stoga:
[Z, Y]

Gledam pravila koja izvode cilj: Y
Skup pravila: [3] = [3] \ []
Razmatram pravilo [3] AKO C & X ONDA Y;: NEDOSTAJU PODATCI!
Zakazujem novi cilj X.
```

Sada se traže pravila koja izvode vrijednost činjenice `X`; pronalazi se da je to pravilo 2. Antecedent tog pravila provjerava je li `B=true`; kako radna memorija sadrži vrijednost za `B` i ona je doista `true`, to pravilo pali.

```
KORAK
==================================================
Stanje u memoriji:
var A: Boolean = true (defined=true);
var B: Boolean = true (defined=true);
var C: Boolean = true (defined=true);

Sadržaj stoga:
[Z, Y, X]
```

```
Gledam pravila koja izvode cilj: X
Skup pravila: [2] = [2] \ []
Razmatram pravilo [2] AKO B ONDA X;: PRAVILO PALI!
```

Stoga se u bazu činjenica upisuje `X=true`, zatim se `X` skida s vrha stoga i 2 se dodaje u blokirana pravila.

Sada je na vrhu stoga cilj `Y`. Traže se pravila koja ga izvode i pronalazi da se radi o pravilu 3. To pravilo u antecedentu provjerava je li `C=true` i `X=true`. Kako je to u skladu sa stanjem u radnoj memoriji, pravilo pali; u bazu činjenica upisuje se `Y=true`, s vrha stoga se skida cilj `Y` i 3 se dodaje u blokirana pravila.

```
KORAK
==================================================
Stanje u memoriji:
var A: Boolean = true (defined=true);
var B: Boolean = true (defined=true);
var C: Boolean = true (defined=true);
var X: Boolean = true (defined=true);

Sadržaj stoga:
[Z, Y]

Gledam pravila koja izvode cilj: Y
Skup pravila: [3] = [3] \ []
Razmatram pravilo [3] AKO C & X ONDA Y;: PRAVILO PALI!
```

Sada je na vrhu stoga `Z` pa se traže pravila koja ga izvode. Pronalazi se da je to pravilo 4 koje provjerava je li `Y=true`. Kako je to u skladu sa stanjem u memoriji, pravilo pali; u bazu činjenica upisuje se `Z=true`, `Z` se skida sa stoga i 4 se dodaje u blokirana pravila.

```
KORAK
==================================================
Stanje u memoriji:
var A: Boolean = true (defined=true);
var B: Boolean = true (defined=true);
var C: Boolean = true (defined=true);
var X: Boolean = true (defined=true);
var Y: Boolean = true (defined=true);

Sadržaj stoga:
[Z]

Gledam pravila koja izvode cilj: Z
Skup pravila: [4] = [4] \ []
Razmatram pravilo [4] AKO Y ONDA Z;: PRAVILO PALI!
```

Ovime smo došli do situacije da je stog prazan: nema više ciljeva koje treba dokazati. Stoga je postupak gotov. Sustav ispisuje završni izvještaj čitajući vrijednost činjenice `Z` iz radne memorije.

```
ZAVRŠNI IZVJEŠTAJ
=================
Izveden je sljedeći zaključak: var Z: Boolean = true (defined=true)
```

Konceptualno, postupak ulančavanja unatrag relativno je jednostavan, no treba paziti na nekoliko detalja. Uporabom stoga kao mjesta za pohranu trenutnog cilja postupak postaje iterativan, a kreće postavljanjem ciljne činjenice na vrh stoga. U svakoj iteraciji traže se sva neblokirana pravila koja izvode cilj koji je na vrhu stoga (ako takvih nema, cilj se skida sa stoga i ide se u sljedeću iteraciju s novim ciljem koji je bio "ispod"). Za svako pronađeno pravilo razmatra se antecent; ako su sve korištene činjenice poznate (imaju vrijednost u radnoj memoriji), tada ako antecedent nije zadovoljen, pravilo se blokira i ide na sljedeće; ako antecedent je zadovoljen, pravilo pali: u radnu memoriju se upisuje činjenica i njezina vrijednost koju definira konsekvent pravila, pravilo se blokira, trenutni cilj se skida sa stoga i ide se u novu iteraciju. Ako smo prošli kroz sva pravila za trenutni cilj i niti jedno nije palilo, trenutni cilj se skida sa stoga i ide se u novu iteraciju. Ako ništa od ovoga nije bilo zadvoljeno, to znači da smo za trenutni cilj naišli na pravilo koje u antecedentu ima činjenicu čiju vrijednost nemamo u radnoj memoriji. Stoga se provjerava postoji li neko pravilo koje izvodi tu činjenicu. Ako je odgovor potvrdan, ona se postavlja kao novi cilj na vrh stoga i ide se u novu iteraciju. Ako to nije slučaj, tada se pita korisnika da unese vrijednost činjenice. Nakon što je to učinjeno, na isti se način analiziraju preostale činjenice antecedenta trenutnog pravila. U slučaju da je pravilo konjunkcija, unosom vrijednosti činjenice odmah se može provjeriti zadovoljava li činjenica pravilo, pa ako ne, postupa se kao u slučaju kad smo naišli na pravilo čiji antecedent nije bio zadovoljen.

Čitav se postupak prekida kad se uđe u iteraciju u kojoj je stog prazan. To se moglo dogoditi u dva scenarija:

- uspjeli smo izvesti vrijednost za zadani cilj pa je isti uklonjen nakon paljenja prigodnog pravila ili
- cilj je maknut jer ga nismo uspjeli izvesti.

Stoga jednom kad je postupak završen treba provjeriti bazu činjenica i pogledati imamo li izvedenu vrijednost za traženu činjenicu ili istu nije bilo moguće izvesti.

2.1.3 Primjer 1: voće

U službenoj prezentaciji uz ovu temu (od slidea 42) dan je primjer ekspertnog sustava koji na temelju karakteristika ploda voćke utvrđuje o kojoj se voćki radi. Baza pravila sastoji se 14 pravila. Taj je primjer dostupan uz Školski ekspertni sustav u datoteci `pravila1.txt` koju prenosimo i ovdje.

```
typedef TOblik: enum izdužen, okrugli, zaobljen;
typedef TPovršina: enum glatka, hrapava;
typedef TBoja: enum zelena, žuta, žuto-smeđa, crvena, plava,
                    narančasta;
typedef TBrojSjemenki: integer(1,);
typedef TVrstaSjemenke: enum višestruke, koštunjasta;
typedef TPromjer: integer(1,);
typedef TVrstaVoćke: enum loza, stablo;
typedef TVoće: enum banana, lubenica, dinja, kanalupe, jabuka,
                    marelica, višnja, breskva, šljiva, naranča;

var Oblik: TOblik;
var Površina: TPovršina;
var Boja: TBoja;
var Broj_sjemenki: TBrojSjemenki;
var Vrsta_sjemenke: TVrstaSjemenke;
var Promjer: TPromjer;
var Vrsta_voćke: TVrstaVoćke;
```

```
var Voće: TVoće;

AKO Oblik = izdužen & Boja = (zelena | žuta) ONDA Voće = banana;
AKO Oblik = (okrugli | zaobljen) & Promjer > 10
      ONDA Vrsta_voćke = loza;
AKO Oblik = okrugli & Promjer <= 10 ONDA Vrsta_voćke = stablo;
AKO Broj_sjemenki = 1 ONDA Vrsta_sjemenke = koštunjasta;
AKO Broj_sjemenki > 1 ONDA Vrsta_sjemenke = višestruke;
AKO Vrsta_voćke = loza & Boja = zelena ONDA Voće = lubenica;
AKO Vrsta_voćke = loza & Površina = glatka & Boja = žuta
      ONDA Voće = dinja;
AKO Vrsta_voćke = loza & Površina = hrapava & Boja = žuto-smeđa
      ONDA Voće = kantalupe;
AKO Vrsta_voćke = stablo & Boja = narančasta &
      Vrsta_sjemenke = koštunjasta ONDA Voće = marelica;
AKO Vrsta_voćke = stablo & Boja = narančasta &
      Vrsta_sjemenke = višestruke ONDA Voće = naranča;
AKO Vrsta_voćke = stablo & Boja = crvena &
      Vrsta_sjemenke = koštunjasta ONDA Voće = višnja;
AKO Vrsta_voćke = stablo & Boja = narančasta &
      Vrsta_sjemenke = koštunjasta ONDA Voće = breskva;
AKO Vrsta_voćke = stablo & Boja = (žuta | zelena | crvena) &
      Vrsta_sjemenke = višestruke ONDA Voće = jabuka;
AKO Vrsta_voćke = stablo & Boja = plava &
      Vrsta_sjemenke = koštunjasta ONDA Voće = šljiva;
```

Pretpostavimo da se radi o voću analiziranom u prezentaciji uz ovu temu i za koje znamo sljedeće činjenice:
`Promjer=2`, `Oblik=okrugli`, `Boja=crvena` i `Broj_sjemenki=1`.
Postupak ulančavanja unaprijed uz dane činjenice možemo pokrenuti naredbom:

```
java -cp ses.jar hr.fer.zemris.ses.main.FCES pravila1.txt
```

Pokrenut će se program. Unosi korisnika kao i ispis ekspertnog sustava dan je u nastavku.

```
*****************************************************
* Školski ekspertni sustav V1.0 - autor: Marko Čupić *
*****************************************************

Unesite činjenice, jednu po retku, u formatu IME = vrijednost. Kada ste gotovi, unesite 'start' ili 'start NAZIV-CILJA'.
> Promjer=2
OK. Stanje memorije je:
var Promjer: TPromjer = 2 (defined=true);
> Oblik=okrugli
OK. Stanje memorije je:
var Promjer: TPromjer = 2 (defined=true);
var Oblik: TOblik = okrugli (defined=true);
> Boja=crvena
OK. Stanje memorije je:
var Promjer: TPromjer = 2 (defined=true);
var Oblik: TOblik = okrugli (defined=true);
var Boja: TBoja = crvena (defined=true);
> Broj_sjemenki=1
OK. Stanje memorije je:
var Promjer: TPromjer = 2 (defined=true);
var Oblik: TOblik = okrugli (defined=true);
var Boja: TBoja = crvena (defined=true);
var Broj_sjemenki: TBrojSjemenki = 1 (defined=true);
> start Voće


Radimo sa sljedećim pravilima:
[1] AKO Oblik = izdužen & Boja = (zelena | žuta) ONDA Voće = banana;
[2] AKO Oblik = (okrugli | zaobljen) & Promjer > 10 ONDA Vrsta_voćke = loza;
[3] AKO Oblik = okrugli & Promjer <= 10 ONDA Vrsta_voćke = stablo;
[4] AKO Broj_sjemenki = 1 ONDA Vrsta_sjemenke = koštunjasta;
[5] AKO Broj_sjemenki > 1 ONDA Vrsta_sjemenke = višestruke;
[6] AKO Vrsta_voćke = loza & Boja = zelena ONDA Voće = lubenica;
```

```
[7] AKO Vrsta_voćke = loza & Površina = glatka & Boja = žuta ONDA Voće = dinja;
[8] AKO Vrsta_voćke = loza & Površina = hrapava & Boja = žuto-smeđa ONDA Voće = kantalupe;
[9] AKO Vrsta_voćke = stablo & Boja = narančasta & Vrsta_sjemenke = koštunjasta ONDA Voće = marelica;
[10] AKO Vrsta_voćke = stablo & Boja = narančasta & Vrsta_sjemenke = višestruke ONDA Voće = naranča;
[11] AKO Vrsta_voćke = stablo & Boja = crvena & Vrsta_sjemenke = koštunjasta ONDA Voće = višnja;
[12] AKO Vrsta_voćke = stablo & Boja = narančasta & Vrsta_sjemenke = koštunjasta ONDA Voće = breskva;
[13] AKO Vrsta_voćke = stablo & Boja = (žuta | zelena | crvena) & Vrsta_sjemenke = višestruke ONDA Voće = jabuka;
[14] AKO Vrsta_voćke = stablo & Boja = plava & Vrsta_sjemenke = koštunjasta ONDA Voće = šljiva;

Početno stanje u memoriji:
var Promjer: TPromjer = 2 (defined=true);
var Oblik: TOblik = okrugli (defined=true);
var Boja: TBoja = crvena (defined=true);
var Broj_sjemenki: TBrojSjemenki = 1 (defined=true);

Novi korak
============================================================
Skup konfliktnih pravila: [3, 4] \ []
Odabrano pravilo: 3
[3] AKO Oblik = okrugli & Promjer <= 10 ONDA Vrsta_voćke = stablo;

Stanje u memoriji:
var Promjer: TPromjer = 2 (defined=true);
var Oblik: TOblik = okrugli (defined=true);
var Boja: TBoja = crvena (defined=true);
var Broj_sjemenki: TBrojSjemenki = 1 (defined=true);
var Vrsta_voćke: TVrstaVoćke = stablo (defined=true);

Novi korak
============================================================
Skup konfliktnih pravila: [3, 4] \ [3]
Odabrano pravilo: 4
[4] AKO Broj_sjemenki = 1 ONDA Vrsta_sjemenke = koštunjasta;

Stanje u memoriji:
var Promjer: TPromjer = 2 (defined=true);
var Oblik: TOblik = okrugli (defined=true);
var Boja: TBoja = crvena (defined=true);
var Broj_sjemenki: TBrojSjemenki = 1 (defined=true);
var Vrsta_voćke: TVrstaVoćke = stablo (defined=true);
var Vrsta_sjemenke: TVrstaSjemenke = koštunjasta (defined=true);

Novi korak
============================================================
Skup konfliktnih pravila: [3, 4, 11] \ [3, 4]
Odabrano pravilo: 11
[11] AKO Vrsta_voćke = stablo & Boja = crvena & Vrsta_sjemenke = koštunjasta ONDA Voće = višnja;

Stanje u memoriji:
var Promjer: TPromjer = 2 (defined=true);
var Oblik: TOblik = okrugli (defined=true);
var Boja: TBoja = crvena (defined=true);
var Broj_sjemenki: TBrojSjemenki = 1 (defined=true);
var Vrsta_voćke: TVrstaVoćke = stablo (defined=true);
var Vrsta_sjemenke: TVrstaSjemenke = koštunjasta (defined=true);
var Voće: TVoće = višnja (defined=true);
```

Postupak ulančavanja unatrag uz dane činjenice možemo pokrenuti naredbom:

```
java -cp ses.jar hr.fer.zemris.ses.main.BCES pravila1.txt
```

Pokrenut će se program. Unosi korisnika kao i ispis ekspertnog sustava dan je u nastavku.

```
*****************************************************
* Školski ekspertni sustav V1.0 - autor: Marko Čupić *
*****************************************************

Unesite činjenice, jednu po retku, u formatu IME = vrijednost. Kada ste gotovi, unesite 'start NAZIV-CILJA'.
> start Voće


Radimo sa sljedećim pravilima:
[1] AKO Oblik = izdužen & Boja = (zelena | žuta) ONDA Voće = banana;
[2] AKO Oblik = (okrugli | zaobljen) & Promjer > 10 ONDA Vrsta_voćke = loza;
[3] AKO Oblik = okrugli & Promjer <= 10 ONDA Vrsta_voćke = stablo;
[4] AKO Broj_sjemenki = 1 ONDA Vrsta_sjemenke = koštunjasta;
[5] AKO Broj_sjemenki > 1 ONDA Vrsta_sjemenke = višestruke;
[6] AKO Vrsta_voćke = loza & Boja = zelena ONDA Voće = lubenica;
[7] AKO Vrsta_voćke = loza & Površina = glatka & Boja = žuta ONDA Voće = dinja;
[8] AKO Vrsta_voćke = loza & Površina = hrapava & Boja = žuto-smeđa ONDA Voće = kantalupe;
[9] AKO Vrsta_voćke = stablo & Boja = narančasta & Vrsta_sjemenke = koštunjasta ONDA Voće = marelica;
[10] AKO Vrsta_voćke = stablo & Boja = narančasta & Vrsta_sjemenke = višestruke ONDA Voće = naranča;
[11] AKO Vrsta_voćke = stablo & Boja = crvena & Vrsta_sjemenke = koštunjasta ONDA Voće = višnja;
[12] AKO Vrsta_voćke = stablo & Boja = narančasta & Vrsta_sjemenke = koštunjasta ONDA Voće = breskva;
[13] AKO Vrsta_voćke = stablo & Boja = (žuta | zelena | crvena) & Vrsta_sjemenke = višestruke ONDA Voće = jabuka;
[14] AKO Vrsta_voćke = stablo & Boja = plava & Vrsta_sjemenke = koštunjasta ONDA Voće = šljiva;

Početno stanje u memoriji:


KORAK
=====================================================
Stanje u memoriji:
```

```
Sadržaj stoga:
[Voće]

Gledam pravila koja izvode cilj: Voće
Skup pravila: [1, 6, 7, 8, 9, 10, 11, 12, 13, 14] = [1, 6, 7, 8, 9, 10, 11, 12, 13, 14] \ []
Razmatram pravilo [1] AKO Oblik = izdužen & Boja = (zelena | žuta) ONDA Voće = banana;: NEDOSTAJU PODATCI!
Idem pitat korisnika za Oblik.
Unesite vrijednost za [Oblik] koja je tipa TOblik: enum izdužen, okrugli, zaobljen.
> okrugli
PRAVILO NE PALI!
Razmatram pravilo [6] AKO Vrsta_voćke = loza & Boja = zelena ONDA Voće = lubenica;: NEDOSTAJU PODATCI!
Zakazujem novi cilj Vrsta_voćke.

KORAK
=====================================================
Stanje u memoriji:
var Oblik: TOblik = okrugli (defined=true);


Sadržaj stoga:
[Voće, Vrsta_voćke]

Gledam pravila koja izvode cilj: Vrsta_voćke
Skup pravila: [2, 3] = [2, 3] \ [1]
Razmatram pravilo [2] AKO Oblik = (okrugli | zaobljen) & Promjer > 10 ONDA Vrsta_voćke = loza;: NEDOSTAJU PODATCI!
Idem pitat korisnika za Promjer.
Unesite vrijednost za [Promjer] koja je tipa TPromjer: Integer(1,).
> 2
PRAVILO NE PALI!
Razmatram pravilo [3] AKO Oblik = okrugli & Promjer <= 10 ONDA Vrsta_voćke = stablo;: PRAVILO PALI!

KORAK
=====================================================
Stanje u memoriji:
var Oblik: TOblik = okrugli (defined=true);
var Promjer: TPromjer = 2 (defined=true);
var Vrsta_voćke: TVrstaVoćke = stablo (defined=true);


Sadržaj stoga:
[Voće]

Gledam pravila koja izvode cilj: Voće
Skup pravila: [6, 7, 8, 9, 10, 11, 12, 13, 14] = [1, 6, 7, 8, 9, 10, 11, 12, 13, 14] \ [1, 2]
Razmatram pravilo [6] AKO Vrsta_voćke = loza & Boja = zelena ONDA Voće = lubenica;: PRAVILO NE PALI!
Razmatram pravilo [7] AKO Vrsta_voćke = loza & Površina = glatka & Boja = žuta ONDA Voće = dinja;: PRAVILO NE PALI!
Razmatram pravilo [8] AKO Vrsta_voćke = loza & Površina = hrapava & Boja = žuto-smeđa ONDA Voće = kantalupe;: PRAVILO NE PALI!
Razmatram pravilo [9] AKO Vrsta_voćke = stablo & Boja = narančasta & Vrsta_sjemenke = koštunjasta ONDA Voće = marelica;: NEDOSTAJU PODATCI!
Idem pitat korisnika za Boja.
Unesite vrijednost za [Boja] koja je tipa TBoja: enum zelena, žuta, žuto-smeđa, crvena, plava, narančasta.
> crvena
PRAVILO NE PALI!
Razmatram pravilo [10] AKO Vrsta_voćke = stablo & Boja = narančasta & Vrsta_sjemenke = višestruke ONDA Voće = naranča;: PRAVILO NE PALI!
Razmatram pravilo [11] AKO Vrsta_voćke = stablo & Boja = crvena & Vrsta_sjemenke = koštunjasta ONDA Voće = višnja;: NEDOSTAJU PODATCI!
Zakazujem novi cilj Vrsta_sjemenke.

KORAK
=====================================================
Stanje u memoriji:
var Oblik: TOblik = okrugli (defined=true);
var Promjer: TPromjer = 2 (defined=true);
var Vrsta_voćke: TVrstaVoćke = stablo (defined=true);
var Boja: TBoja = crvena (defined=true);


Sadržaj stoga:
[Voće, Vrsta_sjemenke]

Gledam pravila koja izvode cilj: Vrsta_sjemenke
Skup pravila: [4, 5] = [4, 5] \ [1, 2, 6, 7, 8, 9, 10]
Razmatram pravilo [4] AKO Broj_sjemenki = 1 ONDA Vrsta_sjemenke = koštunjasta;: NEDOSTAJU PODATCI!
Idem pitat korisnika za Broj_sjemenki.
Unesite vrijednost za [Broj_sjemenki] koja je tipa TBrojSjemenki: Integer(1,).
> 1
Pravilo pali. Cilj Vrsta_sjemenke je izveden.

KORAK
=====================================================
Stanje u memoriji:
var Oblik: TOblik = okrugli (defined=true);
var Promjer: TPromjer = 2 (defined=true);
var Vrsta_voćke: TVrstaVoćke = stablo (defined=true);
var Boja: TBoja = crvena (defined=true);
var Broj_sjemenki: TBrojSjemenki = 1 (defined=true);
var Vrsta_sjemenke: TVrstaSjemenke = koštunjasta (defined=true);


Sadržaj stoga:
[Voće]

Gledam pravila koja izvode cilj: Voće
Skup pravila: [11, 12, 13, 14] = [1, 6, 7, 8, 9, 10, 11, 12, 13, 14] \ [1, 2, 6, 7, 8, 9, 10]
Razmatram pravilo [11] AKO Vrsta_voćke = stablo & Boja = crvena & Vrsta_sjemenke = koštunjasta ONDA Voće = višnja;: PRAVILO PALI!
```

```
ZAVRŠNI IZVJEŠTAJ
=================
Izveden je sljedeći zaključak: var Voće: TVoće = višnja (defined=true)
```

2.1.4 Primjer 2

U službenoj prezentaciji uz ovu temu (od slidea 54) dan je još jedan primjer ulančavanja unaprijed i unatrag. Baza pravila sastoji se 6 pravila. Taj je primjer dostupan uz Školski ekspertni sustav u datoteci `pravila3.txt` koju prenosimo i ovdje.

```
var p: Boolean;
var q: Boolean;
var r: Boolean;
var s: Boolean;
var w: Boolean;
var t: Boolean;
var u: Boolean;
var v: Boolean;
var cilj: Boolean;
var početak: Boolean;

AKO p & q ONDA cilj;
AKO r & s ONDA p;
AKO w & r ONDA q;
AKO t & u ONDA q;
AKO v ONDA s;
AKO početak ONDA v & r & q;
```

Postupak ulančavanja unaprijed uz dane činjenice možemo pokrenuti naredbom:

```
java -cp ses.jar hr.fer.zemris.ses.main.FCES pravila3.txt
```

Pokrenut će se program. Unosi korisnika kao i ispis ekspertnog sustava dan je u nastavku.

```
*****************************************************
* Školski ekspertni sustav V1.0 - autor: Marko Čupić *
*****************************************************

Unesite činjenice, jednu po retku, u formatu IME = vrijednost.
Kada ste gotovi, unesite 'start' ili 'start NAZIV-CILJA'.
> početak=true
OK. Stanje memorije je:
var početak: Boolean = true (defined=true);
> start


Radimo sa sljedećim pravilima:
[1] AKO p & q ONDA cilj;
[2] AKO r & s ONDA p;
[3] AKO w & r ONDA q;
[4] AKO t & u ONDA q;
[5] AKO v ONDA s;
[6] AKO početak ONDA v & r & q;
```

```
Početno stanje u memoriji:
var početak: Boolean = true (defined=true);

Novi korak
==========================================================
Skup konfliktnih pravila: [6] \ []
Odabrano pravilo: 6
[6] AKO početak ONDA v & r & q;

Stanje u memoriji:
var početak: Boolean = true (defined=true);
var v: Boolean = true (defined=true);
var r: Boolean = true (defined=true);
var q: Boolean = true (defined=true);

Novi korak
==========================================================
Skup konfliktnih pravila: [5, 6] \ [6]
Odabrano pravilo: 5
[5] AKO v ONDA s;

Stanje u memoriji:
var početak: Boolean = true (defined=true);
var v: Boolean = true (defined=true);
var r: Boolean = true (defined=true);
var q: Boolean = true (defined=true);
var s: Boolean = true (defined=true);

Novi korak
==========================================================
Skup konfliktnih pravila: [2, 5, 6] \ [6, 5]
Odabrano pravilo: 2
[2] AKO r & s ONDA p;

Stanje u memoriji:
var početak: Boolean = true (defined=true);
var v: Boolean = true (defined=true);
var r: Boolean = true (defined=true);
var q: Boolean = true (defined=true);
var s: Boolean = true (defined=true);
var p: Boolean = true (defined=true);

Novi korak
==========================================================
Skup konfliktnih pravila: [1, 2, 5, 6] \ [6, 5, 2]
Odabrano pravilo: 1
[1] AKO p & q ONDA cilj;

Stanje u memoriji:
```

```
var početak: Boolean = true (defined=true);
var v: Boolean = true (defined=true);
var r: Boolean = true (defined=true);
var q: Boolean = true (defined=true);
var s: Boolean = true (defined=true);
var p: Boolean = true (defined=true);
var cilj: Boolean = true (defined=true);

Novi korak
============================================================
Skup konfliktnih pravila: [1, 2, 5, 6] \ [6, 5, 2, 1]
Gotovi smo. Izveli smo sve što se može izvesti.
```

Postupak ulančavanja unatrag uz dane činjenice možemo pokrenuti naredbom:

```
java -cp ses.jar hr.fer.zemris.ses.main.FCES pravila3.txt
```

Pokrenut će se program. Unosi korisnika kao i ispis ekspertnog sustava dan je u nastavku.

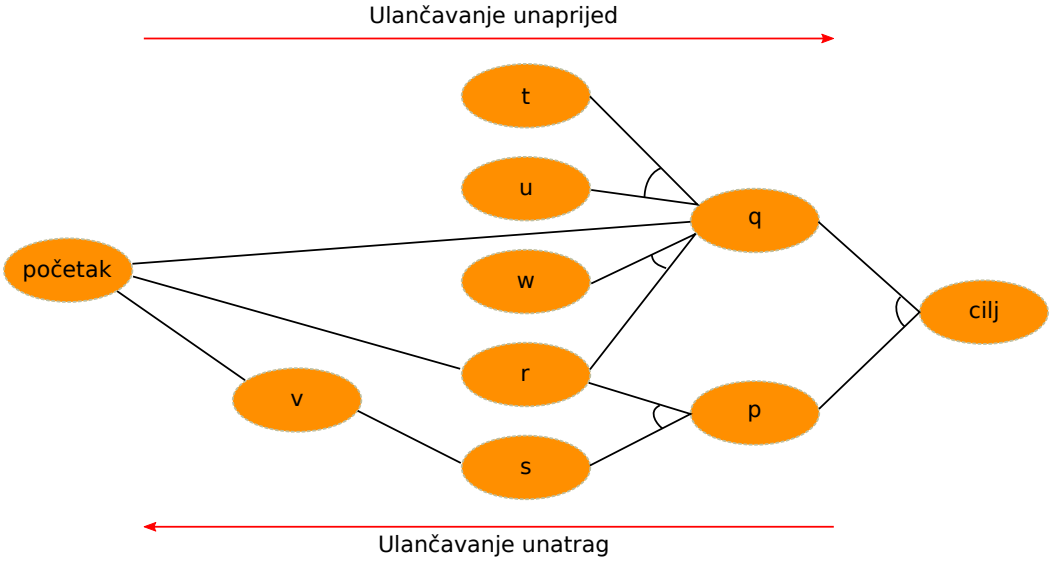
```
*****************************************************
* Školski ekspertni sustav V1.0 - autor: Marko Čupić *
*****************************************************

Unesite činjenice, jednu po retku, u formatu IME = vrijednost.
Kada ste gotovi, unesite 'start NAZIV-CILJA'.
> start cilj


Radimo sa sljedećim pravilima:
[1] AKO p & q ONDA cilj;
[2] AKO r & s ONDA p;
[3] AKO w & r ONDA q;
[4] AKO t & u ONDA q;
[5] AKO v ONDA s;
[6] AKO početak ONDA v & r & q;

Početno stanje u memoriji:


KORAK
=====================================================
Stanje u memoriji:


Sadržaj stoga:
[cilj]

Gledam pravila koja izvode cilj: cilj
Skup pravila: [1] = [1] \ []
Razmatram pravilo [1] AKO p & q ONDA cilj;: NEDOSTAJU PODATCI!
Zakazujem novi cilj p.
```

```
KORAK
==================================================
Stanje u memoriji:


Sadržaj stoga:
[cilj, p]

Gledam pravila koja izvode cilj: p
Skup pravila: [2] = [2] \ []
Razmatram pravilo [2] AKO r & s ONDA p;: NEDOSTAJU PODATCI!
Zakazujem novi cilj r.

KORAK
==================================================
Stanje u memoriji:


Sadržaj stoga:
[cilj, p, r]

Gledam pravila koja izvode cilj: r
Skup pravila: [6] = [6] \ []
Razmatram pravilo [6] AKO početak ONDA v & r & q;: NEDOSTAJU PODATCI!
Idem pitat korisnika za početak.
Unesite vrijednost za [početak] koja je tipa Boolean: Boolean.
> true
Pravilo pali. Cilj r je izveden.

KORAK
==================================================
Stanje u memoriji:
var početak: Boolean = true (defined=true);
var v: Boolean = true (defined=true);
var r: Boolean = true (defined=true);
var q: Boolean = true (defined=true);


Sadržaj stoga:
[cilj, p]

Gledam pravila koja izvode cilj: p
Skup pravila: [2] = [2] \ []
Razmatram pravilo [2] AKO r & s ONDA p;: NEDOSTAJU PODATCI!
Zakazujem novi cilj s.

KORAK
==================================================
Stanje u memoriji:
```

```
var početak: Boolean = true (defined=true);
var v: Boolean = true (defined=true);
var r: Boolean = true (defined=true);
var q: Boolean = true (defined=true);


Sadržaj stoga:
[cilj, p, s]

Gledam pravila koja izvode cilj: s
Skup pravila: [5] = [5] \ []
Razmatram pravilo [5] AKO v ONDA s;: PRAVILO PALI!

KORAK
=====================================================
Stanje u memoriji:
var početak: Boolean = true (defined=true);
var v: Boolean = true (defined=true);
var r: Boolean = true (defined=true);
var q: Boolean = true (defined=true);
var s: Boolean = true (defined=true);


Sadržaj stoga:
[cilj, p]

Gledam pravila koja izvode cilj: p
Skup pravila: [2] = [2] \ []
Razmatram pravilo [2] AKO r & s ONDA p;: PRAVILO PALI!

KORAK
=====================================================
Stanje u memoriji:
var početak: Boolean = true (defined=true);
var v: Boolean = true (defined=true);
var r: Boolean = true (defined=true);
var q: Boolean = true (defined=true);
var s: Boolean = true (defined=true);
var p: Boolean = true (defined=true);


Sadržaj stoga:
[cilj]

Gledam pravila koja izvode cilj: cilj
Skup pravila: [1] = [1] \ []
Razmatram pravilo [1] AKO p & q ONDA cilj;: PRAVILO PALI!

ZAVRŠNI IZVJEŠTAJ
```

```
=================
Izveden je sljedeći zaključak: var cilj: Boolean = true (defined=true)
```

Na ovom posljednjem primjeru možemo još jednom istaknuti razliku između ulančavanja unaprijed i ulančavanja unatrag. Lanac koji povezuje činjenice `početak` i `cilj` prikazan je na slici u nastavku.



Kod ulančavanja unaprijed, kreće se od poznatih činjenica i redom izvodi svo znanje koje slijedi iz tih činjenica, pa svo znanje koje dalje slijedi iz tih i izvedenih činjenica, pa ... sve dok se ne dođe do ciljne činjenice. Ulančavanje unaprijed odgovara pretraživanju u širinu.

Kod ulančavanja unatrag kreće se od ciljne činjenice i traži potpora za nju; zatim se traže potpore za te potpore i tako malo po malo sve do početka. Ulančavanje unatrag odgovara pretraživanju u dubinu.

Bibliografija

Knjige

Članci

Konferencijski radovi i ostalo