Učinkovito induktivno logično programiranje od spodaj navzgor

Miha Drole

doktorska disertacija

predana

Fakulteti za računalništvo in informatiko

kot del izpolnjevanja pogojev za pridobitev naziva

doktor znanosti

s področja

računalništva in informatike

Ljubljana, 2016

IZJAVA

Izjavljam, da sem avtor dela in da slednje ne vsebuje materiala, ki bi ga kdorkoli predhodno že objavil ali oddal v obravnavo za pridobitev naziva na univerzi ali na drugem visokošolskem zavodu, razen v primerih, kjer so navedeni viri.

— Miha Drole —

april 2016

Oddajo so odobrili

dr. Igor Kononenko
redni profesor za računalništvo in informatiko
MENTOR IN ČLAN OCENJEVALNE KOMISIJE

dr. Marko Robnik-Šikonja
redni profesor za računalništvo in informatiko
PREDSEDNIK OCENJEVALNE KOMISIJE

dr. Sašo Džeroski
redni profesor za računalništvo in informatiko
ZUNANJI ČLAN OCENJEVALNE KOMISIJE
Inštitut Jožefa Stefana

PREDHODNA OBJAVA

Izjavljam, da so bili rezultati obravnavane raziskave predhodno objavljeni/sprejeti za objavo v recenzirani reviji ali javno predstavljeni v naslednjih primerih:

[1] M. Drole in I. Kononenko. Closed world specialisation inside the induction process. *Intelligent Data Analysis*, 20(4), 2016.

Potrjujem, da sem pridobil pisna dovoljenja vseh lastnikov avtorskih pravic, ki mi dovoljujejo vključitev zgoraj navedenega materiala v pričujočo disertacijo. Potrjujem, da zgoraj navedeni material opisuje rezultate raziskav, izvedenih v času mojega podiplomskega študija na Univerzi v Ljubljani.

Neli,
s katero sem našel srečo tam,
kjer bi jo sam zaman iskal.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*
Miha Drole
Učinkovito induktivno logično programiranje od spodaj navzgor

POVZETEK

Induktivno logično programiranje (ILP) je področje strojnega učenja, za katerega je značilna uporaba logičnega programiranja za opisovanje tako vhodov kot izhodov. Ta značilnost postavlja ILP med pristope strojnega učenja z največjo izrazno močjo, hkrati pa morajo pristopi k ILP učinkovito preiskovati potencialno neskončen prostor logičnih programov, ki jih je moč sestaviti nad podlagi danega vhoda, zaradi česar so ti pristopi bodisi neprimerni za obravnavo velikih podatkovnih množic bodisi so primorani sprejeti omejitve glede svojega jezika hipotez in posledično izraznosti. V delu naslavljamo dve problematiki sistemov ILP, ki delujejo po principu od spodaj navzgor – njihovo nezmožnost uporabe negacije v hipotezah ter njihovo časovno zahtevnost.

Pristopi od spodaj navzgor temeljijo na izračunu nasičenj (spodnjih stavkov) posameznih primerov, ki pa navadno vključujejo zgolj veljavna dejstva, vezana na določen primer, kar onemogoča neposredno odkrivanje hipotez, ki vključujejo negirane literale. Eden izmed pristopov, ki takšnim sistemom omogoča uporabo negacije, je specializacija zaprtega sveta (ang. *closed world specialisation*, CWS). Metoda CWS temelji na ideji učenja pravil, ki opisujejo napačno pokrite primere (izjeme), nakar se negacija tega pravila doda v hipotezo. Na takšen način se omogoči uporaba negacije z učenjem samo na podlagi veljavnih dejstev. Dosedanje uporabe CWS ta postopek uporabljajo za dodatno specializacijo teorije, ki jo je sistem vrnil kot rešitev podanega problema. V tem primeru se dodatno specializirajo le stavki, ki se pojavijo v izhodni teoriji – stavki z najbolje ocenjenim pozitivnim delom. V našem delu prikažemo situacije, kjer takšna uporaba privede do neoptimalnih rešitev in predlagamo dve alternativni uporabi CWS znotraj procesa gradnje hipoteze. Rezultat sta sistema ILP ProGolemNot ter ProGolemNRNot, nadgradnji sistema ProGolem. Za izdelana sistema smo pokazali, da dosegata primerljive ali boljše rezultate od osnovnega sistema ProGolem oz. sistema ProGolem s klasično uporabo CWS. Eksperimentalna primerjava pokaže tudi, da

sta predlagana sistema med seboj po kvaliteti dobljenih rešitev ekvivalentna, vendar ProGolemNRNot za izdelavo rešitve potrebuje manj časa.

Sistemi ILP večino časa prebijejo v izračunu pokritja kandidatnih hipotez. Število kandidatnih hipotez je pri sistemih, ki delujejo od spodaj navzgor, med drugim odvisno tudi od števila literalov v stavku, ki služi kot spodnja meja preiskovanja v prostoru hipotez. V disertaciji razvijemo koncept parnih nasičenj, ki nudi varno odstranjevanje literalov iz spodnjega stavka enega primera ob predpostavki, da bo končna hipoteza pokrila tudi nek drug naključno izbran primer. Varno odstranjevanje litaralov poteka brez eksplicitnega izračuna pokritja in je zato hitrejše. Parna nasičenja smo s posplošitvijo na n-terna nasičenja implementirali kot sistem ProParGolem. Eksperimenti pokažejo, da so časovni prihranki sicer močno odvisni od strukture predznanja v podanem problemu, opaženi faktorji pohitritve pa znašajo do 1.44 brez izgube točnosti najdene rešitve.

Sistema ProParGolem ter ProGolemNRNot smo združili v enovit sistem ProParGolemNRNot, ki uporablja parna nasičenja in je zmožen odkrivati hipoteze, ki uporabljajo negacijo za odstranjevanje izjem iz pokritja. Sistem ProParGolemNRNot smo uporabili za učenje preprostih geometrijskih konceptov iz podatkov, pridobljenih iz simuliranih globinskih senzorjev. Pri tem smo sistemu omogočili, da že naučene koncepte uporablja pri učenju novih. Sistem je za podane mu naloge našel intuitivno smiselna pravila z visoko točnostjo.

Ključne besede: specializacija zaprtega sveta,negacija,nasičenje,spodnji stavek

University *of Ljubljana*
Faculty *of Computer and Information Science*
Miha Drole
Efficient bottom-up inductive logic programming

ABSTRACT

Inductive logic programming (ILP) is a subfield of machine learning that uses logic programming as its input and output language. While the language of logic programming places ILP as one of the most expressive approaches to machine learning, it also causes the space of candidate solutions to be potentially infinite. ILP systems therefore need to be able to efficiently search through a possibly infinite space, often imposing limits on the hypothesis language in order to be able to handle large problems. We address two problems in the domain of bottom-up ILP systems: their inability to use negation and their efficiency.

Bottom-up approaches to ILP rely on the concept of bottom clauses of examples. Bottom clause of a given example includes all known positive facts about it in the background knowledge, causing a bottom-up ILP system to be unable to reason with negation. One approach that enables such systems to use negation is the closed world specialisation (CWS). The method attempts to learn rules that hold for incorrectly covered negative examples, and then adds the negated rule to the hypothesis body. In this manner the use of negation is enabled using only positive facts. Existing uses of CWS use it to further specialise the output theory, which consists of clauses containing only positive literals that achieved the best scores. We show that such application of CWS is prone to lead to suboptimal solutions and provide two alternative uses of CWS inside of the hypothesis generation process. We implemented the two approaches as the ProGolemNot and ProGolemNRNot ILP systems, both based on the ProGolem system. We show that the two proposed systems both perform at least as well in terms of achieved accuracies as the base ProGolem system or its variant using CWS to further specialise the output hypothesis. Experimental comparison of the two systems also shows that they are equivalent in terms of the quality of their outputs, while ProGolemNRNot needs less time to derive the solution.

ILP systems tend to spend most of the time computing the coverage of candidate hypotheses. In bottom-up systems the quantity of candidate hypotheses to be tested also depends on the number of literals in the bottom-clause of a randomly chosen example that forms the lower bound of the search space. In the thesis we define the concept of pairwise saturations. Pairwise saturations allow us to safely remove literals from a given bottom clause under the assumption that the final hypothesis also covers some other randomly chosen example. Safe removal of these literals does not require explicit coverage testing and can be performed faster. We implemented pairwise saturations along with their generalisation to n-wise saturations in the ProParGolem system. Experiments show that the speedups obtained from using pairwise saturations are highly dependent on the background knowledge structure. We observed speedups of up to factor 1.44 without loss of accuracy.

We combine ProGolemNRNot with ProParGolem in ProParGolemNRNot – an ILP system that uses pairwise saturations and CWS. We use ProParGolemNRNot to learn simple geometric concepts using data obtained from simulated depth sensors. In the devised experiment the system can use previously learned concepts to describe new ones. The solutions found by the system are intuitively correct and achieve high accuracy on test data.

Key words: closed-world specialisation,negation,saturation,bottom clause

ZAHVALA

Mentorju Igorju za potrpežljivost, razumevanje in prepotrebno spodbudo pri nastajanju tega dela. Skozi ta leta me ni usmerjal le na moji raziskovalni poti, temveč tudi na poti moje osebne rasti.

Celotnemu plemenu LKM, ki me je sprejelo medse. V njihovi sredi sem našel prostor, kamor spadam.

Staršem in bratoma za neomajno podporo in vero v trenutkih, ko je sam nisem več premogel.

Neli, ker prežene moje strahove.

— Miha Drole, Ljubljana, april 2016.

KAZALO

SLOVARČEK

Θ-*vsebovanost* Θ-subsumption.

deklaracija načina mode declaration.

determiniranost determinacy.

koristen literal gainful literal.

najmanj splošna posplošitev least general generalisation.

nasičenje saturation.

ostritev refinement.

pokritje coverage.

pristranskost jezika language bias.

specializacija zaprtega sveta closed world specialisation.

spodnji stavek bottom clause.

stavek clause.

zamenjava substitution.

I

Uvod

Induktivno logično programiranje (ILP) je podpodročje strojnega učenja, ki se ukvarja z avtomatskim učenjem opisov konceptov. V naravi mnogokrat naletimo na probleme, ki jih ni mogoče podati v klasičnem atributnem zapisu. Takšni problemi za svojo obravnavo terjajo sposobnost uporabe relacij med elementi v problemski domeni. Tovrstne probleme je moč najti predvsem v podpodročjih biologije in kemije, biokemiji, sistemski biologiji in podobno [38].

Primernost ILP na teh področjih ne izhaja zgolj iz njegove sposobnosti učenja iz relacijskih podatkov. Drugi prednosti, ki ju ILP nudi pred drugimi pristopi, sta:

- Preprosto dodajanje ekspertnega znanja, ki ga sistem lahko nato uporabi pri učenju.
- Človeku razumljivi izhodi, ki jih lahko domenski ekspert interpretira in ovrednoti njihovo smiselnost.

Ti dve prednosti sta posebej pomembni pri obravnavi kompleksnih problemov. Prva omogoča usmerjeno povečevanje izraznosti sistema in uporabo domensko specifičnih znanj pri reševanju. Druga lastnost pa omogoča uporabo ILP za odkrivanje novega strokovnega znanja znotraj problemske domene. Pristopi ILP so bili tako v preteklosti uspešno uporabljeni za napovedovanje interakcij med beljakovinami [56], učenje kvalitativnih modelov [7], metode končnih elementov [12], napovedovanju strukture beljakovin [9, 57], napovedovanje aktivnosti iz strukture pirimidinov [18] in triazinov [19], sintezo zdravil [32], med bolj znana odkritja, h katerim je pripomogla uporaba ILP, pa spadajo funkcije genov kvasovk [24], do katerega je prišlo v okviru projekta The Robot Scientist.

Vendar se v praksi izkaže, da je ravno izrazna moč sistemov ILP tudi njihovo breme. Prostor, znotraj katerega leži rešitev podanega problema, je namreč potencialno neskončen. Iz tega vzroka se pristopi k ILP poslužujejo različnih pristopov, s katerimi omejijo časovno zahtevnost izračuna. Med te pristope sodijo tako različne hevristike, s katerimi sistemi vodijo preiskovanje, kot tudi razne omejitve jezika hipotez, denimo omejitev na uporabo zgolj determiniranih literalov, omejitev števila literalov v hipotezi ali njihove globine, največje dopustno število argumentov predikata ipd. Z uporabo hevristik in omejitev jezika sistemi iščejo ravnotežje med izraznostjo, optimalnostjo najdenih hipotez in časom izvajanja.

Iz navedenega sledi, da je pri snovanju sistemov ILP ključnega pomena njihova učinkovitost, pri čemer želimo njihovo učinkovitost povečati, ne da bi kakorkoli vplivali

na kvaliteto rešitev, ki jih sistemi generirajo.

V disertaciji se prednostno ukvarjamo s sistemi ILP, ki hipotezo gradijo v smeri od najbolj specifične proti bolj splošnim. Takšen pristop se je namreč izkazal za primernejšega pri reševanju problemov, ki terjajo izpeljavo dolgih in kompleksnih hipotez [39]. Osredotočimo se na dve njihovi pomanjkljivosti.

Prva je njihova nezmožnost uporabe negacije v hipotezah. V tem sklopu obiščemo in nadgradimo koncept specializacije zaprtega sveta [3] tako, da negacijo uporabljamo le, kadar je ta nujno potrebna, ter odpravimo v dosedanjih pristopih prisotno predpostavko, da je najboljšo hipotezo moč zgraditi z dodatno specializacijo najboljše najdene hipoteze, sestavljene zgolj iz pozitivnih literalov. V delu predlagamo dva takšna pristopa, za katera pokažemo, da sta med seboj po kvaliteti izhodnih hipotez, zgrajenih na realnih podatkih, neločljiva, vendar je eden za faktor 2 hitrejši od drugega. Izvedeni eksperimenti pokažejo, da premik izvajanja specializacije zaprtega sveta v proces indukcije izvajanje bistveno upočasni, vendar se s stališča kakovosti izhodne hipoteze, merjene s točnostjo, izplača.

Drugo pomanjkljivost si sistemi, ki delujejo po principu od spodaj navzgor, delijo s sistemi, ki hipotezo gradijo od zgoraj navzdol: vsem sistemom ILP je namreč skupna visoka časovna kompleksnost izračuna izhodne teorije. Časovna zahtevnost primarno izhaja iz dveh vzrokov: velikega prostora, znotraj katerega se nahaja rešitev, ter časovne zahtevnosti njihovega testiranja. V svojem delu se ukvarjamo s pohitritvijo sistema ProGolem [39], ki za izpeljavo hipoteze uporablja asimetrično relativno minimalno posplošitev. Z upoštevanjem lastnosti rezultata posplošitve lahko namreč vnaprej prepoznamo podmnožico literalov, ki se v njej ne morejo pojaviti. S tem premaknemo spodnjo mejo prostora, v katerem se posplošitev nahaja, bliže k rešitvi, s čimer je za njen izračun potrebnih manj korakov.

Prispevka k znanosti, ki ju podaja disertacija, sta:

1. Pristop k odkrivanju nemonotonih hipotez, ki negacijo uporablja le, kadar negativnih in pozitivnih primerov ni mogoče ločiti zgolj z uporabo pozitivnih literalov. Razviti pristop prav tako ne predpostavlja, da je optimalno rešitev mogoče dobiti s specializacijo najboljše hipoteze, ki ne uporablja negacije. [13]

2. Metoda dvojnih nasičenj, ki z uporabo analize spodnjih stavkov primerov iz njih predhodno odstrani literale, ki ne morejo biti del njihove posplošitve. Predlagani pristop privede do hitrejšega postopka indukcije. [14]

V drugem poglavju podamo pregled definicij in področja, ki je širše povezano z našim delom. Tretje poglavje poda pregled nekaterih nemonotonih sistemov ILP, podrobneje predstavi idejo specializacije zaprtega sveta in opiše prvi prispevek k znanosti skupaj z eksperimentalnimi rezultati in primerjavo z drugimi sistemi. Drugi prispevek k znanosti je opisan v četrtem poglavju skupaj z nekaterimi drugimi pristopi, katerih cilj je pohitritev postopka indukcije. To poglavje vsebuje tudi eksperimentalno ovrednotenje doseženih rezultatov. V petem poglavju združimo prispevka, opisana v tretjem in četrtem poglavju, v enovit sistem in ga uporabimo za učenje opisov primitivnih konceptov iz podatkov, pridobljenih iz simuliranih globinskih senzorjev. V zaključku ponovno izpostavimo glavne točke disertacije ter podamo nekatere smernice za nadaljnje delo.

2

Induktivno logično programiranje

Za ILP je značilna uporaba logičnih programov za opis tako vhodov v sistem ILP (množic pozitivnih in negativnih primerov ter predznanja), kot tudi izhodne hipoteze. Najprej definirajmo nekatere pojme iz področja logičnega programiranja.

Definicija 1 (Term):Term zaznamuje spremenljivko, konstanto ali funkcijo z argumenti, ki so prav tako termi.

Definicija 2 (Atom):Atom je sestavljen iz predikatnega simbola in termov, ki predstavljajo njegove argumente.

Definicija 3 (Literal):Literal zaznamuje atom ali negacijo atoma. Literal je lahko pozitiven (zaznamuje atom) ali negativen (zaznamuje negacijo atoma).

Definicija 4 (Stavek):Stavek je disjunkcija literalov. Stavek je lahko tudi prazen (tj. ne vsebuje nobenega literala).

V ILP se navadno omejimo na Hornove stavke.

Definicija 5 (Hornov stavek):
Hornov stavek je disjunkcija literalov, izmed katerih je največ en pozitiven.

Hornov stavek je torej disjunktno vezana množica literalov oblike $\neg A_1 \vee \neg A_2 \vee ... \neg A_n \vee B$, kjer so A_i in B atomi. Pozitivni literal v Hornovem stavku B imenujemo *glava* stavka, množico negativnih literalov $A_1, ... A_n$ pa *telo* stavka. V kontekstu logičnega programiranja Hornove stavke navadno zapišemo kot $B \leftarrow A_1 \wedge A_2 \wedge ... A_n$ oziroma v sintaksi programskega jezika Prolog kot

```
B :- A1, ..., An.
```

Hornove stavke delimo na tri podvrste stavkov glede na prisotnost oziroma odsotnost glave ali telesa stavka:

Definitni stavki vsebujejo tako glavo kot telo.

Dejstva sestojijo zgolj iz glave.

Povpraševanja predstavljajo stavki s prazno glavo.

V nadaljevanju poglavja opustimo eksplicitno navajanje, da gre za Hornove stavke, in implicitno smatramo, da so vsi stavki Hornovi. Nad stavki definiramo operacijo zamenjave.

Definicija 6 (Zamenjava (ang. substitution)):Zamenjava Θ je množica parov A_i/a_i, kjer je A_i spremenljivka in a_i poljuben term. Zamenjava $\Theta = \{A_1/a_1, \ldots A_n/a_n\}$ nad stavkom C (označeno s $C\Theta$) za vsak par A_i/a_i v Θ nadomesti vse pojavitve spremenljivke A_i v C s termom a_i.

Nad stavki je definirana relacija Θ-*vsebovanosti*, ki prek prostora hipotez napenja mrežo, matematično strukturo, v kateri ima vsak par elementov enolično določen infimum (največja spodnja meja) in supremum (najmanjša zgornja meja).

Definicija 7 (Θ-vsebovanost (ang, Θ-subsumption)):Naj bosta C_1 in C_2 stavka, podana kot množici literalov. Stavek C_1 Θ-vsebuje stavek C_2 (označeno s $C_1 \geq C_2$ oz. $C_2 \leq C_1$), če obstaja zamenjava Θ, za katero $C_1\Theta \subseteq C_2$. Stavek C_1 strogo Θ-vsebuje stavek C_2 (označeno s $C_1 > C_2$ oz. $C_2 < C_1$), kadar obstaja takšna zamenjava Θ, da $C_1\Theta \subset C_2$.

Stavek C_1 tako Θ-vsebuje C_2, kadar z uporabo neke zamenjave Θ povzročimo, da se vsak literal v C_1 pojavi tudi v C_2 – zamenjava Θ torej povzroči, da postane množica literalov v C_1 podmnožica literalov v C_2. Primer 1 podaja primer Θ-vsebovanosti.
Primer 1: Dana naj bosta stavka C in C':

```
C = :- spol(A, S), starš(A, B), starš(D, E).
C' = :- spol(a, m), starš(a, b).
```

Stavek C Θ-vsebuje C', saj postane po zamenjavi $\Theta = \{A/a, S/m, B/b, D/a, E/b\}$ enak

```
:- spol(a, m), starš(a, b), starš(a, b).
```

Vsak izmed literalov v $C\Theta$ se pojavi v C', zato $C \geq C'$. △

Definicija 8 (Infimum):Največja spodnja meja stavkov (infimum) C_1 *in* C_2 *je stavek* C*, za katerega velja:*

- $C \preceq C_1$,
- $C \preceq C_2$,
- $\forall C' : C' \preceq C_1, C_2 : C' \preceq C$.

Izračun infimuma dveh stavkov se izvede s pomočjo unifikacije in ustreza postopku, ki ga izvede programski jezik Prolog ob uporabi operatorja =. Primer 2 podaja primer infimuma.

Primer 2: Infimum stavkov

```
C = :- starš(A, B), spol(A, m).
C' = :- starš(C, d), spol(C, E).
```

je stavek

```
:- starš(A, d), spol(A, m).
```

△

Definicija 9 (Supremum):Najmanjša zgornja meja (supremum) stavkov C_1 *in* C_2 *je stavek* C*, za katerega velja:*

- $C_1 \preceq C$,
- $C_2 \preceq C$,
- $\forall C' : C_1, C_2 \preceq C' : C \preceq C'$.

Supremum dveh stavkov sovpada z njuno najmanj splošno posplošitvijo, ki jo podrobneje obravnavamo v podpoglavju 2.2.1. Primer supremuma podaja primer 3.

Primer 3: Supremum stavkov

```
C = :- starš(A, B), spol(A, m).
C' = :- starš(C, d), spol(C, E).
```

je stavek

```
:- starš(VAC, VBd), spol(VAC, VmE).                                    △
```

Definicija 10 (Problem ILP):Podana naj bo množica pozitivnih primerov E^+, množica negativnih primerov E^- in predznanje B. Rešitev problema ILP $\langle E^+, E^-, B\rangle$ je logični program T, za katerega velja, da je:

popoln (ang. complete) $(T \wedge B \vdash E^+)$ *– pokrije vse podane pozitivne primere,*

konsistenten (ang. consistent) $(\forall e^- \in E^- : T \wedge B \nvdash e^-)$ *– ne pokrije nobenega izmed negativnih primerov.*

V nadaljevanju logični program, ki ga sestavi sistem ILP, imenujemo tudi *teorija*. Posamezna teorija sestoji iz enega ali več stavkov, ki jih v kontekstu sistemov ILP imenujemo *hipoteze*.

Primer 4 prikazuje primer problema ILP in eno izmed njegovih možnih rešitev.

Primer 4: Dan naj bo sledeč problem ILP:

Predznanje:

```
spol(m1, m). spol(m2, m). spol(m3, m). spol(ž1, ž).

starš(m2, m1). starš(m3, m2). starš(m3, ž1).
```

Množica pozitivnih primerov: `sin(m1, m2). sin(m2, m3).`

Množica negativnih primerov: `sin(m1, m3). sin(ž1, m3).`

Pogojema popolnosti in konsistentnosti zadosti denimo teorija

```
sin(A, B) :- starš(B, A), spol(A, m).                                  △
```

Sistemi ILP izkoriščajo strukturiranost prostora hipotez za izvedbo usmerjenega preiskovanja. Večina sistemov ILP spada glede na smer preiskovanja prostora v eno izmed dveh kategorij:

od spodaj navzgor (ang. bottom-up) ti sistemi iščejo hipotezo v smeri od bolj specifičnih proti bolj splošnim,

od zgoraj navzdol (ang. top-down) ti sistemi preiskujejo prostor v smeri od bolj splošnih proti bolj specifičnim.

Preiskovanje se izvaja z uporabo ostritvenega operatorja. Ostritveni operator iz danega stavka zgradi množico njegovih ostritev, ki so bodisi bolj splošne bodisi bolj specifične od vhodnega.

Definicija 11 (Ostritveni operator (ang. refinement operator)):Kadar za nek ostritveni operator $\rho(H) = \{H_1, H_2, \ldots, H_m\}$ *velja* $\forall H_i \in \rho(H) : H \succ H_i$*, operator* ρ *imenujemo* specializacija. *Podobno v primeru, ko za vsak* $H_i \in \rho(H)$ *velja* $H_i \succ H$*,* ρ *imenujemo* generalizacija.

Sistemi ILP, ki delujejo od zgoraj navzdol, uporabljajo ostritvene operatorje specializacije, medtem ko sistemi od spodaj navzgor uporabljajo generalizacijo.

Do sedaj smo o jeziku hipotez pisali kar najbolj splošno – jezik, kot smo ga obravnavali do sedaj, je bil kar najbolj splošen, potencialno celo neskončen. V praksi se iz tega razloga navadno jezik hipotez omeji.

Definicija 12 (Deklaracije načina (ang. mode declarations)):Z uporabo deklaracij načina dodelimo argumentom predikatov v predznanju eno izmed treh vlog:

- *Vhodni argument (označen s +). Spremenljivke, ki nastopajo kot argumenti na vhodnih mestih, morajo biti ob klicu že opredeljene.*
- *Izhodni argument (označen z -). Spremenljivke na izhodnih mestih so lahko bodisi nove bodisi že opredeljene. Že opredeljene spremenljivke na izhodnih mestih igrajo podobno vlogo kot vhodne spremenljivke, s to razliko, da za izvedbo predikata niso nujne. Še neopredeljene spremenljivke imajo po izvršitvi klica dodeljeno vrednost (se opredelijo).*
- *Konstantni argument (označen z #). Na teh mestih lahko nastopajo zgolj konstante – na teh mestih se torej ne more pojaviti term, ki vsebuje spremenljivke.*

Ob podanih deklaracijah načina M definiramo jezik hipotez $\mathscr{L}(M)$, ki vsebuje vse hipoteze, ki so glede na M veljavno zgrajene. Hipoteza H je torej vsebovana v $\mathscr{L}(M)$, če

- Na vseh vhodnih (+) in izhodnih (-) mestih literalov v H nastopajo spremenljivke.

- Na vseh konstantnih (#) mestih literalov v H nastopajo konstante.
- Vsaka spremenljivka, ki se v i-tem literalu pojavi na vhodnem mestu, nastopa bodisi kot vhodni argument v glavi hipoteze bodisi kot izhodni argument j-tega literala ($j < i$).

Vsakemu stavku brez spremenljivk lahko ob upoštevanju deklaracij načina izračunamo njegovo najbolj specifično posplošeno ustreznico v jeziku $\mathscr{L}(M)$ s postopkom variabilizacije.

Definicija 13 (Variabilizacija):Variabilizacija stavka C *glede na jezik hipotez* $\mathscr{L}(M)$ *nadomesti vse pojavitve konstant na vhodnih in izhodnih mestih (določene v deklaracijah načina* M*) s spremenljivkami. Pri tem se posamezna konstanta vedno nadomesti z isto spremenljivko.*

Dodatno navadno jezik hipotez omejimo z največjo globino spremenljivk, ki nastopajo v njej.

Definicija 14 (Globina spremenljivk (ang. variable depth)):Naj bo v *spremenljivka v definitnem stavku* H*. Globina spremenljivke* v*,* $d(v)$*, je definirana kot*

$$d(v) = \begin{cases} 0 & v \text{ je vhodni argument glave} \\ \max_{u \in vhodi(L)} d(u) + 1 & v \text{ je izhodni argument } L \end{cases}$$

kjer $vhodi(L)$ *označuje množico spremenljivk na vhodnih mestih literala* L*.*

Primer stavka z globinami spremenljivk je podan kor primer 5.

Primer 5: Naj bo dan problem s sledečimi deklaracijami načina predikatov:

```
vzhodni(+). pred(+,-). zaprt(+).
```

Globine spremenljivk, nastopajočih v stavku:

```
vzhodni(A) :-
    pred(A, B), zaprt(B), pred(B, C).
```

so: $d(A) = 0$, $d(B) = 1$ in $d(C) = 2$. △

Jezik hipotez s podanimi deklaracijami načina M in omejitvijo največje globine spremenljivk i označimo z $\mathscr{L}_i(M)$.

Definicija 15 (Urejen stavek (ang. ordered clause)):Urejen stavek je zaporedje literalov $L_1 \vee ... L_n$. *Urejene stavke označimo z nadpisano puščico, npr.* $\vec{C}$.

V nasprotju z navadnimi definitnimi stavki, kjer vrstni red literalov ni pomemben in jih lahko obravnavamo kot množico literalov, pri urejenih stavkih to ne drži – stavka $L_1 \vee L_2$ in $L_2 \vee L_1$ sta različna (razen v primeru $L_1 = L_2$).

Nad urejenimi stavki definiramo relacijo podzaporedja:

Definicija 16 (Podzaporedje (ang. subsequence)):Urejen stavek $\vec{A} = L_{a1} \vee ... \vee L_{an}$ *je podzaporedje urejenega stavka* $\vec{B} = L_{b1} \vee ... \vee L_{bm}$, *če obstaja takšna preslikava indeksov* ai *v indekse* bj, *ki vsakemu izmed literalov* L_{ai} *priredi literal* L_{bj}, *da velja* $L_{ai} = L_{bj}$ *in* $ai \leq bj$. *Da je* $\vec{A}$ *podzaporedje* $\vec{B}$ *označimo kot* $\vec{A} \sqsubseteq \vec{B}$.

Iz te definicije je trivialno razvidno, da $\vec{A} \sqsubseteq \vec{B} \implies A \succeq B$.

Pri iskanju rešitve nekega problema ILP se navadno omejimo na hipoteze, ki so povezane z glavo.

Definicija 17 (Z glavo povezan stavek (ang. head-connected clause)):Dani urejen definitni stavek $H \leftarrow L_1, \wedge ... \wedge L_n$ *je ob podanih deklaracijah načina povezan z glavo, če za vsak literal* L_j $(1 \leq j \leq n)$ *velja, da se vse njegove vhodne vrednosti pojavijo bodisi kot vhodne vrednosti v glavi stavka bodisi kot izhodne vrednosti nekega predhodnega literala* L_i, $i < j$.

S tako definiranim jezikom hipotez lahko definiramo spodnji stavek (ang. bottom clause) ali nasičenje (ang. saturation) nekega primera.

Definicija 18 (Spodnji stavek (ang. bottom clause)):Spodnji stavek $\perp_i$ *primera* e_i *je najbolj specifičen stavek v podanem jeziku hipotez, ki ob podanem predznanju B pokrije primer* $e_i (\perp_i \wedge B \vdash e_i)$.

V [36] je podan algoritem, ki ob podanem predznanju BK, deklaracijah načina M, primeru e_i ter omejitvi globine spremenljivk d izračuna $\perp_i$, spodnji stavek primera e_i. Ta izračun je prikazan kot algoritem 1.

V nekaterih domenah ni mogoče najti splošne hipoteze, ki bi zadostila pogojema popolnosti in konsistentnosti. Do takšne situacije lahko pride zaradi prisotnosti šuma

Vhod : Primer $e = P(a_1, \ldots a_n)$,
deklaracije načina M,
predznanje BK,
omejitev globine d
Izhod : Nasičenje $\bot$

$\bot \leftarrow \{\}$;
termi $\leftarrow \{\}$;
globina $\leftarrow 0$;
$m \leftarrow M[P]$;
for $i \in 1 \ldots n$ *do*
  if $m[i]$ *vhodno mesto then*
    termi $\leftarrow$ *termi* $\cup\, a_i$;
  end
end
while globina $< d \wedge$ *sprememba množice termi do*
  for $m_i = P_i(+t_{1+} \ldots + t_{n_i+}, \#t_{1\#} \ldots \#t_{n_i\#}, t_{1-} \ldots t_{n_i-}) \in M$ *do*
    $I \leftarrow$ množica vhodnih mest v m;
    $X = P_i(A_{1+} \ldots + A_{n_i+}, \#A_{1\#} \ldots \#A_{n_i\#}, A_{1-} \ldots A_{n_i-})$ *for vhod* $\in$ *termi*$^{|I|}$
    do
      $\Theta \leftarrow \{A_{1+}/vhod[1], \ldots A_{n_i+}/vhod[n_i+]\}$;
      for $\Theta_i = \Theta \cup \{A_{1\#}/c_{1\#}, \ldots A_{n_i\#}/c_{n_i\#}, A_{1-}/d_{1-} \ldots A_{n_i-}/d_{n_i-}\}$ *do*
        if $X\Theta_i \notin \bot \wedge X\Theta_i$ *uspe then*
          $\bot \leftarrow \bot \cup X\Theta_i$;
          termi $\leftarrow$ *termi* $\cup \{d_{1-} \ldots d_{n_i-}\}$;
        end
      end
    end
  end
  globina $\leftarrow$ *globina* $+ 1$;
end
return $\bot$;

Algoritem 1: Izračun nasičenja oz. spodnjega stavka danega primera.

v podatkih (npr. nek primer je naveden kot pozitiven, čeprav je v resnici negativen, predznanje vsebuje napake ipd.), lastnosti domene same (hipoteza, ki bi razločevala pozitivne in negativne primere, sploh ne obstaja), pomanjkljivega predznanja (takšna hipoteza sicer obstaja, vendar predznanje ne vsebuje vseh potrebnih predikatov, da bi jo lahko zgradili) ali premajhnega števila učnih primerov (zaradi česar je iskanje regularnosti med njimi težavno) [40]. Zaradi takšnih primerov sistemi ILP navadno kvaliteto hipoteze ocenjujejo s pomočjo funkcije ocene $s(H, P, N)$, ki na podlagi hipoteze (H) ter njenega pokritja pozitivnih (P) in negativnih primerov (N) izračuna njeno kvaliteto. Najpogostejše funkcije ocene v ILP so:

pokritje

$$s(H, P, N) = P - N$$

Pokritje nagrajuje pozitivno pokritje in kaznuje negativno. Pri uporabi takšne ocene za vodenje iskanja se ocena izboljša tudi pri povečanju negativnega pokritja ob pogoju, da je povečanje pozitivnega pokritja večje.

točnost

$$s(H, P, N) = \frac{P}{P + N}$$

Podobno kot pri uporabi ocene pokritja, tudi točnost omogoča pokritje dela negativnih primerov. V primerjavi s pokritjem nam točnost pove delež pravilno pokritih primerov in je po vrednosti omejena na interval $[0, 1]$.

kompresija

$$s(H, P, N) = P - N - |H|$$

Kompresija poleg pozitivnega in negativnega pokritja upošteva tudi kompleksnost hipoteze same. Kot približek kompleksnosti se navadno uporablja kar njena dolžina, tj. število literalov v njej.[1]

Poleg teh se za vodenje iskanja ciljne hipoteze uporabljajo druge statistike, znane iz strojnega učenja:

laplace

$$s(H, P, N) = \frac{P + 1}{P + N + 2}$$

.

[1] Včasih tudi $s(H, P, N) = P - N - |H| + 1$, pri čemer se iz dolžine hipoteze izvzame njeno glavo.

m-ocena

$$s(H,P,N) = \frac{P + mp_a}{P + N + m},$$

kjer je m parameter in p_a relativna frekvenca pozitivnih primerov v množici učnih primerov $p_a = \frac{|E^+|}{|E^+|+|E^-|}$.

gini

$$s(H,P,N) = 2p(1-p),$$

kjer je p točnost hipoteze, $p = \frac{P}{P+N}$.

entropija

$$s(H,P,N) = p \log p + (1-p) \log 1 - p,$$

kjer p označuje točnost hipoteze $p = \frac{P}{P+N}$.

ReliefF Vse prej omenjene funkcije so kratkovidne in delujejo globalno nad vsemi primeri. Kononenko in sodelavci so hevristično funkcijo ReliefF prilagodili za uporabo v sistemih ILP [26]. Osnovna razlika med ReliefF in drugimi ocenami leži v tem, da ReliefF ocenjuje, kako dobro dodani literal razlikuje med podobnimi pozitivnimi in negativnimi primeri. (tj. primeri, ki pripadajo nasprotnima množicama, a ležijo blizu drug drugega v prostoru primerov).

2.1 Sistemi od zgoraj navzdol

Sistemi, ki delujejo od zgoraj navzdol običajno pričnejo iskanje s prazno hipotezo, ki uspe za čisto vsak nabor argumentov. Sistem trenutno hipotezo na vsakem koraku specializira (napravi bolj specifično) z ostritvenim operatorjem. Pri izpeljevanju hipoteze navadno uporabljajo prekrivni pristop, ki iz množice primerov tekom učenja odstranjuje že pokrite primere.

Shema algoritma, ki deluje po principu od zgoraj navzdol, je podana kot algoritem 2. Ti pristopi gradijo in dodajajo nove hipoteze v izhodno teorijo, dokler ta ne pokrije vseh pozitivnih primerov, ki so na voljo za učenje. Takšen sistem na vsakem koraku izračuna množico ostritev trenutne hipoteze in izmed njih hevristično izbere novo trenutno hipotezo na podlagi podane funkcije ocene. Gradnjo hipoteze ponavlja, dokler ni zadoščeno ustavitvenim pogojem (denimo, kadar hipoteza ne pokriva več nobenega negativnega primera, ali kadar nadaljnje ostritve ne izboljšajo ocene hipoteze). Na

Vhod : Množica pozitivnih primerov E^+, množica negativnih primerov E^-, funkcija ocene s, ostritveni operator ρ
Izhod : Teorija T
$T \leftarrow \emptyset$;
$E \leftarrow E^+$;
while $E \neq \emptyset$ *do*
  $H \leftarrow$ hipoteza s praznim telesom;
  while $\neg$ *ustavitveni pogoji do*
    $HC \leftarrow \rho(H)$;
    $H \leftarrow argmax_{H'}(s(H'); H' \in HC)$;
  end
  $E \leftarrow E \setminus pokritje(H, E)$;
  $T \leftarrow T \cup H$;
end
return T;

Algoritem 2: Shema algoritma ILP, ki deluje po principu od zgoraj navzdol.

tem mestu trenutni stavek doda v izhodno teorijo, odstrani z njim pokrite pozitivne primere in postopek ponovi, če še obstajajo nepokriti pozitivni primeri.

Iz same sheme delovanja teh sistemov izhaja težava, poimenovana "problem planote" (v področju strojnega učenja je ta pojav imenovan tudi kratkovidnost). Pri takšnem preiskovanju prostora namreč lahko pride do situacije, kjer enkratna uporaba ostritvenega operatorja ρ (npr. dodajanje enega literala) ne izboljša kvalitete hipoteze, vendar večkratna uporaba ρ privede do njenega izboljšanja. Primer takšnega obnašanja sistema prikazuje primer 6.

Primer 6: Naj bo podan problem ILP s predznanjem:

```
spol(m1, m). spol(m2, m). spol(m3, m). spol(m4, m).
spol(ž1, ž). spol(ž2, ž). spol(ž3, ž). spol(ž4, ž).

starš(m1, ž1). starš(m2, m3). starš(ž2, ž3). starš(ž4, m4).
```

Množico pozitivnih primerov:

```
ima_hčer(m1). ima_hčer(ž2).
```

Množico negativnih primerov:

```
ima_hčer(m2). ima_hčer(ž4).
```

Ko sistem prične svoje delovanje s prazno hipotezo `ima_hčer(A)`, noben dodan literal ne izboljša njegove ocene. Prazna hipoteza pokrije vse pozitivne in negativne primere. Če hipotezo ostrimo z dodajanjem literala `spol(A, m)` ali `spol(A, ž)` nova hipoteza pokrije en negativen in en pozitiven primer, če pa v telo dodamo literal `starš(A, B)` ostane pokritje hipoteze nespremenjeno (torej takšna hipoteza še vedno pokrije vse pozitivne in negativne primere). V takšnih primerih sistem ILP pade v slepo iskanje po prostoru ostritev, zaradi česar lahko zgreši ciljni opis koncepta. △

V nadaljevanju nekoliko podrobneje predstavimo dva bolj znana sistema, ki rešitev problema iščeta v smeri od bolj splošne proti bolj specifični.

2.1.1 FOIL

FOIL (First Order Inductive Learner) [45] išče rešitev problema na klasičen način od zgoraj navzdol. Zanj je značilno, da si pri preiskovanju neposredno ne pomaga s predznanjem, kot to počnejo drugi sistemi tipa od zgoraj navzdol. Kot drugi sistemi tudi FOIL uporablja prekrivni pristop k reševanju problema – gradi nove stavke in jih dodaja v teorijo, dokler ta ne pokrije vseh pozitivnih primerov. Pri tem vsak stavek gradi z ozirom na v trenutni iteraciji še nepokrite pozitivne primere, torej iz svoje učne množice sproti odstranjuje že pokrite primere.

FOILov ostritveni operator $\rho_{FOIL}(C)$ na desno stran stavka C doda nov literal L. Izbira literala L, ki bo dodan k C, se opravi na požrešen način – za vsak $L \in \rho_{FOIL}(C)$ FOIL izračuna oceno stavka (C, L) in iskanje nadaljuje z najbolje ocenjenim izmed teh stavkov. Literali, ki jih FOIL doda v C, so lahko koristni iz dveh vzrokov. Prvi tip teh literalov so koristni (ang. gainful) literali, ki iz pokritja trenutne hipoteze odstranijo napačno pokrite primere (t.j. negativne primere), drugi pa determinirani literali, ki sicer ne izboljšajo ocene trenutnega stavka, vendar vanj vnesejo nove spremenljivke. Uporaba teh spremenljivk v naslednjih iteracijah algoritma namreč lahko privede do izboljšanja ocene stavka, pri čemer determinirani literali služijo kot predpogoj za njihovo uvedbo.

Definicija 19 (Koristni literali (ang. gainful literals)):Literal L_m je koristen z ozirom na delno zgrajeno hipotezo $H = R(V_1, \ldots, V_k) \leftarrow L_1, \ldots L_{m-1}$, če ima hipoteza $H' = R(V_1, \ldots, V_k) \leftarrow L_1, \ldots L_{m-1}, L_m$ višjo oceno kot H.

Definicija 20 (Determinirani literali (ang. determinate literals)):Literal L_m je determiniran z ozirom na delno zgrajeno hipotezo $H = R(V_1, \ldots, V_k) \leftarrow L_1, \ldots L_{m-1}$, kadar:

- *L_m vsebuje vsaj eno novo spremenljivko,*
- *L_m uspe z natanko eno instanciacijo njegovih spremenljivk za vsak pokriti pozitivni primer,*
- *L_m na negativnih primerih bodisi uspe z natanko eno instanciacijo njegovih spremenljivk bodisi ne uspe.*

Z uporabo determiniranih literalov se FOIL delno izogne problemu planote. Ob preverjanju kandidatnih literalov za razširitev hipoteze si FOIL sproti zapomni, kateri izmed njih so determinirani. Kadar v trenutni iteraciji nobena razširitev hipoteze ne doseže zadostne izboljšave, se trenutna hipoteza razširi z *vsemi* najdenimi determiniranimi literali. Ti literali v hipotezo uvedejo nove spremenljivke in s tem povečajo prostor možnih razširitev trenutne hipoteze. Zaradi nekritičnega dodajanja determiniranih literalov v hipotezo je potrebno iz stavka, preden ga dodamo v izhodno teorijo, odstraniti odvečne literale – le redko bodo namreč vse spremenljivke, ki so dodane z uporabo determiniranih literalov, tudi dejansko uporabljene v nadaljnji izpeljavi.

Oris FOILovega algoritma je podan kot algoritem 3.

2.1.2 *Aleph in Progol*

Sistema ILP Aleph [53] in Progol [36] sta najširše uporabljana sistema, ki delujeta po principu od zgoraj navzdol. V primerjavi s sistemom FOIL oba sistema pristopata h gradnji hipoteze na bolj usmerjen način. Ker si delita mnoge značilnosti, ju na tem mestu obravnavamo skupaj.

Za usmerjanje izpeljave sistema uporabljata t.i. spodnji stavek oz. nasičenje primerov [36]. V začetku izpeljave se izbere naključen pozitiven primer e_i, katerega bo

Vhod :
Množica pozitivnih primerov E^+ (oblike $R(v_1, \ldots v_k)$),
množica negativnih primerov E^- (oblike $R(v_1, \ldots v_k)$),
predznanje BK
Izhod : Teorija T

---

function FOIL(E^+, E^-, BK) is
  $T \leftarrow \{\}$;
  $S \leftarrow E^+$;
  while $S \neq \emptyset$ *do*
    $C \leftarrow (R(V_1, \ldots V_k) \leftarrow)$;
    while C *pokriva negativne primere do*
      $C_{ext} = \rho(C)$;
      $C \leftarrow argmax_{C' \in C_{ext}} ocena(C')$;
    end
    iz C odstrani odvečne literale ;
    $S \leftarrow S \setminus pokritje(C, S)$;
    $T \leftarrow T \cup C$;
  end
  return T;
end

Algoritem 3: Osnovni algoritem sistema FOIL.

ustvarjena hipoteza gotovo pokrila. Za ta primer se izračuna njegovo nasičenje $\bot_i$, katerega variabilizacija ($var(\bot_i)$) v nadaljevanju predstavlja spodnjo mejo prostora kandidatnih hipotez. Nato sistem ustvari prazen stavek, ki ga sistem korakoma hevristično ostri z dodajanjem literalov, prisotnih v $var(\bot_i)$. Osnutek takšnega algoritma prikazuje algoritem 4.

Vhod : Množici pozitivnih in negativnih primerov E^+ ter E^-,
deklaracije načina M,
predznanje BK
Izhod : Teorija T

$T \leftarrow \emptyset$;
$S \leftarrow E^+$;
while $S \neq \emptyset$ *do*
  $C \leftarrow \{\}$;
  $e_i \leftarrow izberi(S)$;
  $\bot_i \leftarrow nasicenje(e_i, M, BK)$;
  while sprememba trenutnega stavka C *do*
    $L \leftarrow argmax_{L'}\{ocena(C \cup L'); L' \in \bot_i\}$;
    $\bot_i \leftarrow \bot_i \setminus L$;
    if $ocena(C) < ocena(C \cup L)$ *then*
      $C \leftarrow C \cup L$;
    end
  end
  $S \leftarrow S \setminus pokritje(C, S)$;
  $T \leftarrow T \cup C$;
end
return T ;

Algoritem 4: Psevdokoda algoritma za gradnjo hipoteze po principu od zgoraj navzdol glede na spodnji stavek naključno izbranega primera.

Takšen način izpeljave torej ne omogoča reševanja problemov, ki za svojo izpeljavo potrebujejo deljenje spremenljivk (ang. variable splitting). Standarden primer tovrstnega problema je bitno seštevanje s prenosom, prikazano kot primer 7, povzet po

[55].

Primer 7: Naj bo podan problem seštevanja dveh bitov `sestej(A, B, C, D)`, kjer sta `A` in `B` bita, ki ju seštevamo, `C` njun seštevek, ter `D` prenos po seštevanju. Sistemu naj bosta kot predznanje podani definiciji operacij logične konjunkcije in ekskluzivne disjunkcije: `and(A, B, C)` ($C = A \wedge B$) in `xor(A, B, C)` ($C = A \oplus B$). Množici pozitivnih in negativnih primerov, podani sistemu, naj bosta popolni. Posplošitev spodnjega stavka pozitivnega primera `sestej(1,0,1,0)` je (pojavitve `1` zamenjamo s spremenljivko `A`, pojavitve `0` pa z `B`)

```
sestej(A, B, A, B) :-
    and(A, A, A), and(A, B, B), and(B, A, B), and(B, B, B),
    xor(A, A, B), xor(A, B, A), xor(B, A, B), xor(B, B, B).
```

Hipoteza, ki predstavlja rešitev tega problema je

```
sestej(A, B, C, D) :-
    xor(A, B, C), and(A, B, D).
```

Takšne hipoteze s podanim načinom njihove gradnje ni mogoče sestaviti. Da bi takšen problem postal rešljiv, bi sistem moral ugotoviti, da nekatera sovpadanja vrednosti niso splošno veljavna za domeno (najočitnejši primer je že posplošitev glave hipoteze, kjer sta izhoda že enolično določena z vhodi) in uvesti nove spremenljivke, ki zahtevo po sovpadanju odstranijo. △

S tem primerom smo pokazali, da so sistemi, ki hipotezo gradijo na takšen način, za določene učne naloge neprimerni. To omejitev je sicer mogoče razrešiti s transformacijo variabilizacije spodnjega stavka, tako da vse večkratne pojavitve iste vrednosti v glavi ali na izhodnih mestih spremenljivke nadomesti z novimi spremenljivkami. Spodnji stavek se nato razširi z izrazi $A_i = A_j$, kjer sta A_i in A_j spremenljivki, ki ustrezata isti vrednosti, najdeni na različnih mestih v stavku.

2.2 Sistemi od spodaj navzgor

Sistemi, ki delujejo po principu od spodaj navzgor, prinašajo napram sistemom, ki delujejo od zgoraj navzdol, bolj usmerjen način preiskovanja prostora hipotez. Svoje iskanje pričnejo z izbiro množice pozitivnih primerov (navadno gre za par primerov) oz. semen, za katere z uporabo operatorja generalizacije izračunajo najbolj specifičen

stavek v jeziku hipotez, ki jih pokrije. Ti stavki so navadno pretirano specifični – so dolgi in imajo majhno pozitivno pokritje. Nadaljnje posploševanje hipoteze ti sistemi izvajajo na dva načina:

Pozitivna redukcija (ang. positive reduction) Pri uporabi pozitivne redukcije za posploševanje hipoteze se izbere naključen še nepokrit pozitiven primer. Ta primer se nato z uporabo posplošitvenega operatorja (navadno gre za uporabo enakega operatorja, kot je bil uporabljen pri izračunu posplošitve semen) privede v pokritje trenutne hipoteze.

Negativna redukcija (ang. negative reduction) Ta pristop nadaljnje posploševanje vodi z uporabo podanih negativnih primerov. Pri negativni redukciji se iz trenutne hipoteze odstranijo vsi literali, katerih odstranitev ne povzroči pokritja negativnih primerov. Kot vodilo se v nekaterih primerih uporablja blažji kriterij – odstrani se literale, katerih odstranitev izboljša oceno hipoteze.

Oris algoritmov, ki hipotezo gradijo od spodaj navzgor je podan kot algoritem 5.

Vhod : Množici pozitivnih in negativnih primerov E^+ in E^-,
predznanje BK
Izhod : Teorija T

---

function nauci(E^+, E^-, BK) *is*
  $T \leftarrow \{\}$;
  $S \leftarrow E^+$;
  while $S \neq \emptyset$ *do*
    $(e_1, e_2) \leftarrow$ *nakljucen_par*(S);
    $H \leftarrow$ *posplosi*(e_1, e_2, BK);
    $H \leftarrow$ *poz_red*($H, S \setminus \{e_1, e_2\}$);
    $H \leftarrow$ *neg_red*(H, E^-);
    $S \leftarrow S \setminus$ *pokritje*(H, S);
    $T \leftarrow T \cup H$;
  end
end

Algoritem 5: Generični sistem ILP, ki hipotezo gradi od spodaj navzgor.

Bolj znani sistemi, ki hipotezo gradijo na takšen način, so Golem [37], ProGolem [39] in LogAn-H [2].

2.2.1 Golem

Sistem Golem [37] temelji na ideji relativne najmanj splošne posplošitve (ang. relative least general generalisation, RLGG), kot jo je opisal Plotkin [44]. Najprej definirajmo najmanj splošno posplošitev para literalov.

Definicija 21 (Najmanj splošna posplošitev literalov in termov (ang. least general generalisation)):
Najmanj splošna posplošitev termov $t_i = f(r_1, \ldots, r_n)$ *in* $t_j = g(s_1, \ldots, s_n)$ *je*

$$lgg(t_i, t_j) = \begin{cases} g(s_1, \ldots, s_n) & \text{če } t_i = t_j \\ V_{ij} & \text{če } f \neq g \\ f(lgg(r_1, s_1), \ldots lgg(r_n, s_n)) & \text{sicer} \end{cases}$$

.
Najmanj splošna posplošitev literalov $a_i = p(r_1, \ldots, r_n)$ *in* $a_j = q(s_1, \ldots, s_n)$ *je*

$$lgg(a_i, a_j) = \begin{cases} \text{nedefiniran} & \text{če } p \neq q \\ p(lgg(r_1, s_1), \ldots lgg(r_n, s_n)) & \text{sicer} \end{cases}$$

Posplošitev dveh literalov je tako mogoče izračunati le, kadar imata enak predikatni simbol in predznak (tj. sta oba bodisi pozitivna bodisi negativna). Iz definicije posplošitve dveh termov izhaja tudi, da se dva terma vedno posplošita v isto spremenljivko. Primer posplošitve literalov podaja primer 8.

Primer 8: Najmanj splošna posplošitev literalov `P(a,b,c)` in `P(b,a,c)` se izračuna kot lgg`(P(a,b,c),P(b,a,c))` =`P(Vab, Vba, c)`. △

Z uporabo najmanj splošne posploštive lahko definiramo najmanj splošno posplošitev stavkov C_i in C_j.

Definicija 22 (Najmanj splošna posplošitev stavkov (ang. least general generalisation)):Naj bosta podana stavka $C_i = L_{i1}, \ldots L_{im}$ *ter* $C_j = L_{j1}, \ldots L_{jn}$. *Njuna najmanj splošna posplošitev je definirana kot* $lgg(C_i, C_j) = \{lgg(L, L'); L \in C_i, L' \in C_j\}$.

Posplošitev poljubnega števila stavkov $lgg(C_1, C_2, \ldots C_n)$ je mogoče izračunati s pomočjo izraza $lgg(C_1, C_2, \ldots C_n) = lgg(C_1, lgg(C2, \ldots C_n))$.

Izračun najmanj splošne posplošitve dveh stavkov je tako množica posplošitev vseh možnih parov literalov, ki nastopajo v njima. Iz tega sledi, da število literalov v posplošitvi stavkov narašča kot zmnožek dolžin stavkov, ki jih posplošujemo. Primer posplošitve dveh stavkov je podan kot primer 9.

Primer 9: Podana naj bosta stavka:

```
hči(b, a) :-
    moški(a), ženska(b), starš(a, b), starš(a, i), starš(b, c),
    ženska(i), moški(c).


hči(g, f) :-
    ženska(g), ženska(f), starš(f, g), starš(f, j), starš(g, h),
    ženska(h), moški(j).
```

Njuna najmanj splošna posplošitev je:

```
hči(Vbg, Vaf) :-
    moški(Vaj), ženska(Vbg), ženska(Vbf), ženska(Vbh),
    starš(Vaf, Vbg), starš(Vaf, Vbj), starš(Vag, Vbh),
    starš(Vaf, Vig), starš(Vaf, Vij), starš(Vag, Vih),
    starš(Vbf, Vcg), starš(Vbf, Vcj), starš(Vbg, Vch),
    ženska(Vig), ženska(Vif), ženska(Vih), moški(Vcj).
```

Znotraj izračunane posplošitve se nahaja tudi rešitev problema:

```
hči(Vbg, Vaf) :-
    ženska(Vbg), starš(Vaf, Vbg).
```

△

Golem posplošitev primerov računa tako, da zanje sestavi stavke, katerih glava je enaka primeru, telo stavka pa je enako konjunkciji vseh dejstev v predznanju. Te stavke nato posploši z uporabo najmanj splošne posplošitve.

Težave takšnega pristopa postanejo ob upoštevanju rasti dolžine posplošitve očitne. Dolžina posplošitve namreč narašča eksponentno s številom primerov, uporabljenih pri izračunu posplošitve.

Če naj izračun ostane učinkovit s stališča časovne zahtevnosti, je treba jezik hipotez omejiti na *ij*-determiniranost.

Definicija 23 (ij-determiniranost (ang. ij-determinacy)):Stavek C je ij-determiniran, kadar je determiniran, ima največja globino literala v njem omejeno z i ter je številost vseh literalov v njem največ j.

Zahteva po *ij*-determiniranosti napravi sistem Golem neprimeren za uporabo na mnogih realnih domenah.

2.2.2 ProGolem

Sistem ProGolem [39] je sistem, ki je soroden sistemoma Progol [36] in Golem [37] v smislu, da uporablja Progolov koncept spodnjih stavkov in iskalno strategijo sistema Golem, pri čemer ProGolem namesto posplošitvenega operatorja relativne najmanj splošne posplošitve uporablja njeno inačico asimetrično relativno minimalno posplošitev (ang. asymmetric relative minimal generalisation, ARMG).

Glavna motivacija za uporabo drugega operatorja posplošitve namesto RLGG leži v že omenjenem dejstvu, da velikost hipoteze pri uporabi RLGG narašča eksponentno s številom primerov, ki jih dodajamo v pokritje hipoteze. Zaradi časovne in prostorske zahtevnosti je Golem omejen na reševanje ij-determiniranih problemov, kar ga napravi neprimernega za uporabo na mnogih realnih domenah. Pri uporabi ARMG do eksponentne rasti ne pride, saj je dolžina $ARMG(e_2|e_1)$ navzgor omejena z dolžino spodnjega stavka $\bot_1$, s tem pa tudi odpade omejitev na ij-determinirane probleme.

Da bi formalno definirali ARMG, najprej definiramo asimetrično relativno skupno posplošitev (ang. asymmetric relative common generalisation, ARCG) [39].

Definicija 24 (Asimetrična relativna skupna posplošitev (ang. asymmetric relative common generalisation)):Naj bo podana množica primerov E ter predznanje BK. Naj bosta e_i in e_j primera iz množice E, ter $\bot_i$ nasičenje primera e_i. Asimetrična relativna skupna posplošitev primera e_j glede na e_i, označena z $ARCG(e_j|e_i)$, je takšen stavek $\vec{C}$, da $\vec{C} \geq \bot_i$ in $BK \wedge C \vdash e_j$.

Iz definicije sledi, da ARCG dveh primerov ni unikatna, ampak vsebuje množico mogočih rešitev. Prav tako je potrebno izpostaviti, kakor tudi sledi že iz njenega imena, da operacija ni simetrična – torej v splošnem ne velja, da je $ARCG(e_2|e_1) =$

$ARCG(e_1|e_2)$. S pomočjo ARCG lahko definiramo ARMG.

Definicija 25 (Asimetrična relativna minimalna posplošitev (ang. asymmetric relative minimal generalisation)):Podana naj bo množica primerov E, predznanje BK ter izbrana primera e_i in e_j iz množice E. $Z \perp_i$ označimo nasičenje primera e_i. $\vec{C}$ je asimetrična relativna minimalna posplošitev primera e_j glede na e_i, kadar je $\vec{C}$ njuna asimetrična relativna skupna posplošitev, ter za vsak drug urejen stavek $\vec{C}' \in ARCG(e_j|e_i)$, za katerega drži $\vec{C} \succeq \vec{C}'$, velja, da je ekvivalenten $\vec{C}$.

Primer 10 prikazuje asimetrično relativno skupno ter minimalno posplošitev.

Primer 10: Podana naj bosta primera e_1 = `hči(b,a)` in e_2 = `hči(g,f)` z njunima posplošitvama spodnjih stavkov pri deklaracijah načina `hči(+, +), starš(+, -), moški(+), ženska(+):`

```
hči(B, A) :-
    starš(A, B), starš(B, I), moški(A), ženska(B), moški(I).

hči(G, F) :-
    starš(F, G), starš(F, E), ženska(G), ženska(F), starš(E, J),
    moški(E), moški(J).
```

V množici asimetričnih relativnih skupnih posplošitev $ARCG(e_2|e_1)$ med drugim najdemo stavke

```
hči(V1, V2) :- starš(V2, V3), ženska(V3).
hči(V1, V2) :- starš(V2, V1).
hči(V1, V2) :- starš(V2, V1), starš(V1, V3).
```

Množica asimetričnih relativnih minimalnih posplošitev $ARMG(e_2|e_1)$ vsebuje

```
hči(V1, V2) :- starš(V2, V1), starš(V1, V3), ženska(V1).
hči(V1, V2) :- starš(V2, V3), starš(V3, V4), ženska(V1),
               moški(V4).
```

△

Čeprav smo ARCG ter ARMG definirali kot posplošitev enega primera glede na drugega, ju lahko na preprost način definiramo tudi za posplošitev stavka glede na

podan primer. Pri izračunu $ARMG(e_2|C)$, kjer je C nek stavek, tako ne iščemo ustrezne podmnožice literalov v spodnjem stavku primera $\perp_1$, kot bi to storili pri izračunu $ARMG(e_2|e_1)$, temveč podmnožico stavka C.

Prav tako kot ARCG je tudi ARMG asimetrična ter neunikatna, kar je razvidno iz primera 10. Algoritem, kakršnega za izračun ARMG uporablja sistem ProGolem, je podan kot algoritem 6.

Vhod : Primera e_i, e_j
nasičenje primera e_i $\perp_i$
Predznanje BK
Izhod : Asimetrična relativna minimalna posplošitev primera e_j glede na primer e_i.

---

function $ARMG(e_i, e_j, \perp_j, BK)$ *is*
  $\vec{C} \leftarrow \perp_i = (h \leftarrow l_1 \ldots l_n)$;
  while $\vec{C}$ *ne pokrije* e_j *do*
    poišči prvi literal l_k, ki pri izvajanju $\vec{C}(e_j)$ ne uspe ;
    $\vec{C} \leftarrow \vec{C} \setminus l_k$;
    iz $\vec{C}$ odstrani z glavo nepovezane literale ;
  end
  return $\vec{C}$
end

Algoritem 6: ProGolemov algoritem za izračun asimetrične relativne minimalne posplošitve dveh primerov.

Algoritem 6 ne generira celotne množice posplošitev. Podobno kot ostali sistemi, ki preiskovanje vodijo z nasičenji, je ProGolem nezmožen generirati hipoteze, ki uporabljajo deljenje spremenljivk (ang. variable splitting). V primeru 10 ta algoritem ni zmožen generirati drugega elementa v množici asimetričnih relativnih minimalnih posplošitev. Pri onemogočenem deljenju spremenljivk ima izračun ARMG unikatno rešitev. V nadaljevanju se ob sklicih na ARMG vedno nanašamo na ARMG, kot je uporabljena v ProGolemu, torej na njeno nepopolno, deterministično različico.

ProGolem uporablja klasični prekrivni pristop v zunanji zanki, v notranji zanki pa uporablja pristop, soroden Golemu [37][2]. Najprej izbere množico *stParov* parov pozi-

[2]ProGolem kot tak podpira več načinov izvajanja notranje zanke. Opisani pristop ustreza t.i. načinu parov

tivnih primerov in zanje izračuna njihove ARMG, izmed njih pa izbere b (širina snopa) najboljših po dani funkciji ocene (privzeto se uporablja kompresija), ki jih pošlje v nadaljnje posploševanje. Sistem nad njimi najprej izvede pozitivno redukcijo – v snopu v njihovo pokritje z uporabo ARMG uvaja nove naključno izbrane pozitivne primere, dokler se kvaliteta hipotez izboljšuje. Pri uvajanju naključno izbranih pozitivnih primerih sistem poskusi več možnosti – naključno izbere množico pozitivnih primerov velikosti *stPrimerov* in z vsakim izmed njih posploši hipoteze v snopu. Snop v naslednji iteraciji tvorijo najboljše razširjene hipoteze, ki so ocenjene bolje od vseh stavkov v prejšnjem snopu. Ob zaključku iskanja se nad najboljšo hipotezo v snopu izvede še negativna redukcija, katere rezultat je nato dodan v izhodno teorijo. Psevdokoda ProGolemovega algoritma je podana kot algoritem 7.

(ang. pairs mode).

Vhod : Množici pozitivnih in negativnih primerov E^+ in E^-,
predznanje BK,
deklaracije načina M
Izhod : Teorija T

```
function progolem(E+, E−, BK, M) is
    T ← {};
    S ← E+;
    while S ≠ ∅ do
        nakljucni_pari ← izberi_naključne_pare(S, stParov);
        kandidati ← {ARMG(e_i|e_j) : (e_i, e_j) ∈ nakljucni_pari};
        kandidati ← izberi_najboljse(trenutni_stavki, b);
        do
            snop ← izberi_najboljse(kandidati, b);
            naključni_primeri ← nakljucno_izberi(S, stPrimerov) ;
            kandidati ← {} ;
            for stavek C ∈ snop do
                for primer e ∈ naključni_primeri do
                    C′ ← ARMG(H, e) ;
                    if ocena(C′) > max_ocena(snop) then
                        kandidati ← kandidati ∪ C′;
                    end
                end
            end
        while kandidati ≠ ∅;
        C ← najboljši stavek v snopu;
        C ← neg_red(C, E+, E−);
        T ← T ∪ C ;
        S ← S \ pokritje(C, S);
    end
    return T;
end
```

Algoritem 7: ProGolem.

3

Specializacija zaprtega sveta

Večina sistemov ILP, opisanih v prejšnjem pogavju, se pri iskanju rešitev omejuje na uporabo pozitivnih literalov v telesih hipotez. V tem poglavju se osredotočimo na način izpeljave hipotez, ki vključujejo negacijo, poimenovan specializacija zaprtega sveta (ang. closed world specialisation). V prvih dveh podpoglavjih najprej predstavimo težavnost iskanja hipotez pri omogočeni uporabi negacije, ki izhaja iz nemonotonosti (Podpoglavje 3.1) in kombinatorični eksploziji števila dejstev, če mednje uvrstimo tudi "'neveljavna"' dejstva (Podpoglavje 3.2). Sledita pregled obstoječih sistemov ILP z zmožnostjo uporabe negacije v Podpoglavju 3.3 in podrobnejši opis specializacije zaprtega sveta, ki predstavlja osnovo za naše delo, v Podpoglavju 3.4. V naslednjih dveh podpoglavjih predstavimo dve varianti uporabe specializacije zaprtega sveta in sistema ProGolemNot ter ProGolemNRNot [13], ki razviti metodi uporabljata, ter posplošitev pristopa za odkrivanje hipotez z gnezdeno negacijo. Poglavje zaključuje analiza pristranskosti jezika opisanih pristopov ter eksperimentalni del, v katerem sistema ProGolemNot ter ProGolemNRNot primerjamo z uporabo specializacije zaprtega sveta pri sistemih ProGolem ter Aleph.

3.1 Predpostavka zaprtega sveta in nemonotonost

Do sedaj smo se omejevali na pristope, ki gradijo definitne stavke. Pri definitnih stavkih telo hipoteze vsebuje zgolj pozitivne literale. Narava nekaterih problemov pa terja tudi uporabo negiranih literalov v telesu hipoteze.

Najprej moramo definirati primere, v katerih negativni literal uspe. V ta namen se opremo na predpostavko zaprtega sveta (ang. closed world assumption). Predpostavka zaprtega sveta pravi, da v dani domeni držijo *le in samo* eksplicitno navedene trditve in trditve, izpeljive iz podanih izrazov.

Definicija 26 (Predpostavka zaprtega sveta (ang. closed world assumption)):Pod predpostavko zaprtega sveta negativni literal $\neg L$ *v sistemu* F *uspe natanko takrat, kadar* L *ni posledica* F.

S tem se premaknemo v območje *nemonotone logike*. V monotoni logiki se z dodajanjem znanja (npr. v obliki novih formul, dejstev ipd.) v sistem množica izpeljivih sklepov nikoli ne zmanjša. V primerih nemonotonih sistemov lahko pride do situacije, kjer dodajanje znanja povzroči, da nekatere izjave niso več izpeljive. Primer takšne situacije je prikazan kot primer 11.

Primer 11: Podana naj bo sledeča množica dejstev:

```
vagon(a). vagon(b). vagon(c). vagon(d).

dolg(a). dolg(b).
```

Podana naj bo tudi formula:

```
ciljni(X) :- \+ dolg(X).
```

Iz te formule je moč (poleg podanih dejstev) izpeljati `ciljni(c)` ter `ciljni(d)`.

Če med dejstva dodamo `dolg(c)`, `ciljni(c)` ni več izpeljiv. Množica izpeljivih dejstev se je z dodajanjem novega znanja zmanjšala. △

Do dodatnega zapleta privede tudi možnost gnezdenja negacije, kadar imamo opravka z negiranjem konjunkcij literalov. Čemu je obravnavanje takšnih primerov težavno, prikazuje primer 12.

Primer 12: Naj bo podana množica dejstev kot v primeru 11, dopolnjena s sledečimi dejstvi:

```
pokrit(a). pokrit(b).
```

Kot v primeru 11, `ciljni(X) :- \+ dolg(X)` uspe za zamenjavi X/c in X/d. Če v negirani del te formule dodamo `\+ pokrit(X)`, dobimo:

```
ciljni(X) :- \+ (dolg(X), \+ pokrit(X)).
```

Pod predpostavko zaprtega sveta je iz te formule moč izpeljati dejstva `ciljni(a)`, `ciljni(b)`, `ciljni(c)` in `ciljni(d)`. △

Iz primera 12 je razvidno, da v nemonotoni logiki dodajanje literalov v hipotezo lahko povzroči njeno generalizacijo z vidika pokritja primerov. Sistemi ILP navadno pri preiskovanju prostora predpostavljajo, da dodajanje novih literalov v hipotezo povzroči njeno specializacijo (tj. pokritje po dodajanju novega literala je kvečjemu enako pokritju pred dodajanjem), kar pa v nemonotonem okolju ne drži.

3.2 *Naivni pristop k odkrivanju negacije*

Preden preidemo na naprednejše načine uvajanja negacije, si poglejmo najočitnejši pristop k uvajanju negacije: eksplicitno dodajanje (posebej označenih) neveljavnih dejstev

v predznanje. Z uporabo predpostavke zaprtega sveta ugotovimo, za katere nabore argumentov predikati, prisotni v predznanju, ne uspejo in jih posebej označene dodamo v predznanje. Pri naborih argumentov se omejimo na konstante, prisotne v predznanju. Algoritem, ki ob predpostavki zaprtega sveta dopolni podano predznanje s posebej označenimi neveljavnimi dejstvi, je podan kot algoritem 8.

Vhod : Predznanje BK
Izhod : Z negativnimi dejstvi dopolnjeno predznanje BK

```
function razširi_z_neg(BK) is
    predikati ← množica predikatov v BK ;
    konstante ← množica konstant v BK ;
    for P ∈ predikati do
        ustvari nov predikat ¬P v BK;
        k ← številnost(P);
        for c_1 ... c_k ∈ konstante do
            if P(c_1 ... c_k) ne uspe then
                BK ← BK ∪ ¬P(c_1 ... c_k);
            end
        end
    end
    return BK;
end
```

Algoritem 8: Algoritem za razširjanje podanega predznanja z negativnimi dejstvi.

S takšnim pristopom k dopolnjevanju se število dejstev v predznanju ustrezno poveča. Naj bo podano predznanje BK, ki vsebuje predikat P številnosti k in n različnih konstant. V predznanju naj bo tudi p dejstev, katerih predikatni simbol je P. Število možnih instanciacij predikata P je enaka n^k (katerakoli izmed n konstant lahko stoji na kateremkoli izmed k mest), izmed katerih je p uspešnih, vse ostale pa so neuspešne. Po takšni razširitvi predznanja so v njem prisotne *vse* možne instanciacije predikata P – bodisi kot instance predikata P bodisi kot instance predikata $\neg P$.

Iz tega je razvidno, da takšen pristop povzroči eksponentno rast števila dejstev v predznanju. Za ilustracijo postopka služi primer 13.

Primer 13: Podano naj bo predznanje

```
moški(a). moški(c). moški(e). moški(i). moški(k). moški(n).

starš(a, b). starš(a, i). starš(b, c). starš(b, d).
starš(d, e). starš(d, m). starš(f, g). starš(f, j).
starš(g, h). starš(g, l). starš(j, k). starš(d, n).
```

Izračunajmo, kako veliko bi to predznanje postalo, če ga dopolnimo z neveljavnimi dejstvi. To predznanje vsebuje 14 konstant, en predikat številnosti 1, ter en predikat številnosti 2. Število možnih instanciacij predikata s številnostjo 1 je 14, predikata z dvema argumentoma pa $14 \cdot 14 = 196$. Izmed njih je zgolj 6 oziroma 12 takšnih, ki so glede na podano predznanje veljavne. Po dopolnitvah z neveljavnimi dejstvi poleg že prisotnih instanciacij dobimo 8 neveljavnih vnosov za predikat ¬`moški`, ter 184 za ¬`starš`. Velikost predznanja je tako narasla z 18 na 210. △

Naivni pristop k omogočanju uporabe negacije tako privede do eksponentne rasti velikosti predznanja. Poleg tega ta pristop ne omogoča odkrivanja negacij konjunkcije literalov (posledično tudi gnezdene negacije). S takšnim pristopom je tako nemogoče rešiti problem, predstavljen v primeru 14.

Primer 14: Imejmo podano množico oseb. Nad to množico naj bodo definirani predikati `moški`, `ženska`, `starš` ter `brat_ali_sestra`. Predikat, ki bi opisal lastnost, da je oseba `A` edini sin osebe `B`, je moč opisati kot

```
edini_sin(A, B) :- starš(B, A), moški(A),
                   \+ (brat_ali_sestra(A, C), moški(C)).
```

Za opis tega koncepta brez negiranja konjunkcije literalov bi moral sistem ILP sestaviti pomožni predikat `brat`, definiran kot

```
brat(A, B) :- brat_ali_sestra(A, B), moški(B).
```
△

Poglejmo si še, kaj se po takšnem dopolnjevanju predznanja zgodi s spodnjimi stavki primerov. Spodnji stavek določenega primera vsebuje vsa dejstva iz predznanja, ki so povezana z njim. Če predznanje dopolnimo z negativnimi dejstvi, ta ustvarijo vse povezave, ki prej niso obstajale – posledično spodnji stavek katerega koli primera po tem procesu zajame celotno predznanje. S tem sistemi, ki si pri preiskovanju pomagajo

s spodnjim stavkom (sem spadajo Progol, Aleph ter ProGolem), izgubijo svojo prednost, saj ta prostora preiskovanja več ne zmanjša. Poglejmo si še, kaj se po takšnem dopolnjevanju predznanja zgodi s spodnjimi stavki primerov. Spodnji stavek določenega primera vsebuje vsa dejstva iz predznanja, ki so povezana z njim. Če predznanje dopolnimo z negativnimi dejstvi, ta ustvarijo vse povezave, ki prej niso obstajale – posledično spodnji stavek katerega koli primera po tem procesu zajame celotno predznanje. S tem sistemi, ki si pri preiskovanju pomagajo s spodnjim stavkom (sem spadajo Progol, Aleph ter ProGolem), izgubijo svojo prednost, saj ta prostora preiskovanja več ne zmanjša.

Primerjavo spodnjih stavkov pred in po dopolnitvi z neveljavnimi dejstvi prikazuje primer 15.

Primer 15: Naj bo podano predznanje

```
moški(a). moški(i).
starš(a, b). starš(a, i). starš(d, e).
```

Spodnji stavek primera `hči(b,a)` je

```
hči(b, a) :-
    moški(a), starš(a, b), starš(a, i).
```

Po dopolnitvi predznanja z neveljavnimi dejstvi pa spodnji stavek postane (uvedeni predikati, ki odražajo neveljavna dejstva imajo predpono `n_`):

```
hči(b, a) :-
    moški(a), n_moški(b), starš(a, b), starš(a, i), n_starš(a, a),
    n_starš(a, d), n_starš(a, e), n_starš(b, a), n_starš(b, b),
    n_starš(b, i), n_starš(b, d), n_starš(b, e), moški(i),
    n_moški(d), n_moški(e), starš(d, e), n_starš(i, a),
    n_starš(i, b), n_starš(i, i), n_starš(i, d), n_starš(i, e),
    n_starš(d, a), n_starš(d, b), n_starš(d, i), n_starš(d, d),
    n_starš(e, a), n_starš(e, b), n_starš(e, i), n_starš(e, d),
    n_starš(e, e).
```

V spodnjem stavku postane po dopolnitvi z neveljavnimi dejstvi dosegljiv tudi pozitivni literal `starš(d, e)`, ki ga pred to dopolnitvijo v nasičenju primera ni bilo.

Hkrati lahko opazimo, da novi spodnji stavek resnično obsega celotno predznanje, dobljeno z dopolnitvijo z neveljavnimi dejstvi. △

3.3 Nemonotoni sistemi ILP

Najširše znan sistem ILP, ki deluje v nemonotonem načinu, je FOIL [45]. Uvedba negiranih literalov v telo hipoteze je zaradi samega načina generiranja kandidatnih hipotez preprosta. FOIL namreč na vsakem koraku poskuša trenutno hipotezo dodatno izboljšati z dodajanjem novega literala v hipotezo. Pri tem se uporaba negacije omogoči že zgolj s tem, da se za vsak preizkušan literal L preizkusi tudi hipotezo, razširjeno z njegovo negacijo $\neg L$. Takšen pristop ne omogoča odkrivanja hipotez z negiranimi konjunkcijami literalov (posledično tudi takšnih z gnezdeno negacijo), zaradi česar vsako dodajanje literala povzroči specializacijo trenutne hipoteze. Takšen pristop povzroči, da na vsakem koraku preverimo enkrat več potencialnih razširitev.

Med drugimi sistemi, ki omogočajo odkrivanje hipotez z negacijo, je $Tracy^{NOT}$ [5]. Glavni prispevek tega sistema leži v samem postopku izpeljave primera (tj. preverjanju, ali dana hipoteza pokrije nek primer). Izpeljava v $Tracy^{NOT}$ sledi klasični SLD-resoluciji za pozitivne literale. Do sprememb pride, ko naleti na negiran literal. Ta literal nato vedno uspe, njegovo nadaljnje obnašanje pa zavisi v odvisnosti od tega, ali gre za izpeljavo pozitivnega ali negativnega primera. Če izpeljujemo pozitiven primer, se literal doda v množico negativnih primerov, sicer pa v množico pozitivnih – po smislu, da mora za pozitivni primer njegova negacija uspeti, za negativen pa ne. Takšen način izpeljave omogoča sistemu, da zna ravnati s hipotezami, ki vsebujejo negirane literale, vendar ne nudi načina, kako takšne hipoteze generirati. $Tracy^{NOT}$ zato potrebuje tudi množico kandidatnih hipotez kot svoj vhod.

Razširitev sistema CLINT [46] na trovrednostno logiko v svojih hipotezah sicer neposredno ne uporablja negacije, vendar gre prav tako za neke vrste nemonoton sistem. CLINT se za podan problem ILP nauči dveh hipotez H_p in H_n: s H_p poskuša opisati pozitivne primere, s H_n pa negativne. Pri določanju, ali je nek nov primer pozitiven ali negativen, nad njim izvede obe naučeni hipotezi. Primer nato uvrsti v primerno skupino zgolj v primeru uspeha natanko ene izmed njiju (če uspe le H_p gre za pozitiven primer, za negativnega pa v primeru, ko uspe le H_n). Sicer (torej, kadar uspeta bodisi obe hipotezi bodisi nobena) CLINT vhodnemu primeru dodeli oznako "neznano". Nemonotonost je v takšnem pristopu mogoče opaziti, kadar novo dodano znanje povzroči, da poleg H_p uspe še H_n in primer kot posledica ne spada več med pozitivne,

temveč med neznane.

Še eno drugačno vrsto nemonotonosti je moč najti v sistemu ICL (Inductive Constraint Logic) [47]. ICL, podobno kot CLINT, eksplicitno uporablja negativne primere pri učenju. Za razliko od večine sistemov ILP, se ICL uči omejitvenih pravil (ang. constraint rules). Nek primer je tako v kontekstu ICL pokrit, kadar zanj uspejo vsa izpeljana omejitvena pravila. Posledično nek primer ni pokrit takoj, ko eno izmed pravil zanj ne drži. ICL uporablja prekrivni pristop nad negativnimi primeri – za vsako izpeljano pravilo mora torej držati, da uspe za vse pozitivne primere, ter da za nekatere negativne primere ne uspe. Takšen pristop je nemonoton v smislu, da se z dodajanjem novih pravil število pokritih primerov manjša.

Sistem ILP INDED [52] zna uporabljati negacijo v hipotezah, vendar zahteva eksplicitno prisotnost negativnih dejstev v predznanju. Pri odsotnosti ekspertnega znanja glede tega, katera neveljavna dejstva so v dani domeni pomembna, to povzroči eksponentno rast velikosti predznanja, kot smo opazili pri naivni metodi uvajanja negacije.

Sistem FOIDL (First-Order Induction of Decision Lists) [35] je razširitev sistema FOIL. Namesto množice pravil je njegov rezultat urejen seznam pravil, ki se na primerih izvajajo zaporedno, dokler bodisi ena izmed njih uspe bodisi nobeno izmed pravil za vhod ne uspe. Po vsakem dodanem pravilu FOIDL preveri, ali so v pokritju pravila prisotni tudi napačno pokriti primeri. Če takšni primeri obstajajo, zanje sestavi novo pravilo, s katerim jih ustrezno obravnava, ter ga doda pred trenutno izpeljano pravilo. Takšen pristop je v srži podoben specializaciji zaprtega sveta (ki je podrobneje obravnavana v podpoglavju 3.4) – ravno tako namreč sistem za vsako pravilo poišče njegovo napačno pokritje in ga poskusi ustrezno obravnavati z uporabo posebnega pravila.

3.4 Specializacija zaprtega sveta

Specializacijo zaprtega sveta (ang. closed world specialisation, CWS) sta predlagala Muggleton in Bain [3] kot način uvajanja negacije v hipoteze. Temeljna ideja leži v uporabi negacije za specializacijo obstoječe presplošne hipoteze.

Predpostavimo sledečo situacijo: poljuben sistem ILP je na svoj vhod prejel problem $\langle E^+, E^-, BK\rangle$ in kot rešitev sestavil hipotezo $H = P \leftarrow A_1, \ldots A_n$, ki pokriva množico pozitivnih primerov $E_c^+ \subseteq E^+$ in množico negativnih primerov $E_c^- \subseteq E^-$. Naj H ne bo konsistentna ($E_c^- \neq \emptyset$). Napačno pokrite negativne primere imenujmo izjeme. Uporaba negacije je en mogoč pristop za odstranjevanje izjem iz pokritja. Naj obstaja takšen literal L_a s predikatnim simbolom P_a, da se, če z njim razširimo telo hipoteze

H pokritje primerov spremeni na sledeč način:

1. $E_c^{+'} = \emptyset$: L_a ne uspe za nobenega izmed pokritih pozitivnih primerov,
2. $E_c^{-'} = E_c^-$: L_a uspe za vse izmed pokritih negativnih primerov.

Ob prisotnosti takšnega literala lahko hipotezo H napravimo konsistentno z vključitvijo $\neg L_a$ v telo hipoteze. Dodatek $\neg L_a$ na pozitivno pokritje H ne vpliva (saj L_a ne uspe za nobenega izmed njih, po principu negacije kot neuspeha torej $\neg L_a$ uspe za vse), hkrati pa odstrani iz pokritje vse primere iz E_c^- (saj L_a zanje uspe, torej po negaciji kot neuspehu $\neg L_a$ zanje ne uspe).

Seveda se na tem mestu postavi vprašanje, kako ustrezni literal L_a določiti. Specializacija zaprtega sveta v ta namen v predznanju definira nov predikat P_a z enako številnostjo in deklaracijami načina kot ciljni predikat in ga uporabi kot "detektor izjem". Naj bo dana nekonsistentna hipoteza

$$H = P(A_1 \ldots A_m) \leftarrow L_1(B_{11}, \ldots B_{1m_1}), \ldots L_n(B_{n1}, \ldots B_{nm_n})$$

K tej hipotezi dodamo negiran klic predikata P_a:

$$H' = P(A_1 \ldots A_m) \leftarrow L_1(B_{11}, \ldots B_{1m_1}), \ldots L_n(B_{n1}, \ldots B_{nm_n}), \neg P_a(A_1 \ldots A_m)$$

P_a prejme enak nabor argumentov, kot jih dobi H na svoj vhod. CWS torej predlaga uvedbo predikata, namenjenega prepoznavanju izjem.

Takšen predikat je mogoče ustvariti na trivialen način že s tem, da se izjem naučimo "na pamet" – predikat P_a definira ekstenzionalno (tj. z eksplicitnim navajanjem naborov argumentov, za katere ta predikat uspe) s tem, da za vsak primer $e_i^- = P(a_{i1}, \ldots a_{im})$ v množici E_c^- doda v predznanje dejstvo $P_a(a_{i1}, \ldots a_{im})$. S tem sicer dosežemo, da postane hipoteza H konsistentna, vendar takšen pristop ne posplošuje in je kot tak v praksi vprašljive uporabe.

V izogib prekomernemu prilagajanju učnim podatkom (čemur ekstenzionalno definiranje predikata P_a ustreza) je mogoče poskusiti z učenjem tega predikata. Za učenje lahko uporabimo kar sam sistem ILP, s katerim smo zgradili hipotezo H. Množici pozitivnih in negativnih primerov, ki služita za vhod pri učenju predikata P_a, ustrezata množici negativnih in pozitivnih primerov, ki jih pokriva H, torej so vloge pozitivnih in negativnih primerov zamenjane.

Takšen pristop denimo ubira sistem LELP [21], ki se poskuša naučiti intenzionalne definicije pomožnega predikata P_a. Vendar je LELP pri učenju omejen z zahtevo, da mora biti pokritje intenzionalne definicije identično pokritju ekstenzionalne. To vodi do pretiranega prilagajanja učnim podatkom v primerih prisotnosti šuma, pomanjkljivega predznanja in/ali prisotnosti neoznačenih primerov.

Da bi se pri učenju pomožnega predikata P_a izognili pretiranemu prilagajanju učnim primerom, se za ocenjevanje kvalitete pomožnih hipotez navadno uporablja mera kompresije [54] z utemeljitvijo, da se pretirano prilagajanje učnim podatkom odrazi kot neproporcionalno podaljšanje dolžine hipoteze v primerjavi s pridobljeno točnostjo zaradi podaljšanja. Ta mera se je pokazala kot odporna na pretirano prilagajanje kot tudi na prisotnost šuma v podatkih.

Zapis postopka specializacije zaprtega sveta v obliki algoritma je podan kot algoritem 9.

Vhod : Hipoteza H,
Množica pozitivnih primerov E^+,
Množica negativnih primerov E^-,
Predznanje BK
Izhod : Specializacija hipoteze H

---

function $cws(H, E^+, E^-, BK)$ *is*
  $pozitivno\_pokritje \longleftarrow pokritje(H, E^+)$;
  $negativno\_pokritje \longleftarrow pokritje(H, E^-)$;
  $pomozna\_h \longleftarrow nauci(negativno\_pokritje, pozitivno\_pokritje, BK)$;
  $H' \longleftarrow H, \neg pomozna\_h$;
  return H' ;
end

Algoritem 9: Specializacija zaprtega sveta.

Tradicionalno se uporaba specializacije zaprtega sveta opravi šele, ko sistem ILP sestavi dokončno teorijo zgolj z uporabo pozitivnih literalov – CWS se torej uporablja kot korak postprocesiranja. Takšno operacijo denimo nudi sistem Aleph [53], enak pristop pa je v [4] apliciran na sistem Golem pri učenju zmagovalnih šahovskih končnic KRK (kralj-trdnjava-kralj, ang. king-rook-king).

Primer 16 prikazuje, čemu uporaba CWS na tak način lahko povzroči, da sistem ILP zgreši ciljno hipotezo.

Primer 16: Naj bo podan problem ILP s predznanjem:

```
moški(a). moški(c). moški(e). moški(i). moški(k). moški(n).

starš(a, b). starš(a, i). starš(b, c). starš(b, d).
starš(d, e). starš(d, m). starš(f, g). starš(f, j).
starš(g, h). starš(g, l). starš(j, k). starš(d, n).
```

Množica pozitivnih primerov naj bo

```
hči(b, a). hči(d, b). hči(g, f). hči(j, f). hči(l, g). hči(m, d).
```

Množica negativnih primerov pa

```
hči(i, a). hči(c, b). hči(e, d). hči(k, j). hči(d, a). hči(h, j).
hči(b, i). hči(n, d).
```

Popolno in konsistentno rešitev tega problema predstavlja hipoteza:

```
hči(A, B) :-
    starš(B, A), \+ moški(A).
```

Da bi z uporabo CWS kot postprocesiranja izhodne hipoteze nek sistem ILP zgradil to hipotezo, bi se stavek `hči(A, B) :- starš(B, A)` moral nahajati med izhodnimi stavki, saj je ciljna hipoteza specializacija tega stavka. Izračunajmo kompresivnost tega stavka. Stavek pokrije vseh sedem pozitivnih primerov, vendar tudi pet negativnih (instance, kjer je `B` moškega spola). Pri dolžini telesa 1 je njegova kompresivnost zaradi pokritja petih negativnih primerov nizka – enaka je $7 - 5 - 1 = 1$.

Večina sistemov ILP (tako takšni, ki hipotezo gradijo v smeri proti splošnejšim, kot takšni, ki jo gradijo proti bolj specifičnim) pri takšnem vhodu izdelajo hipotezo

```
hči(A, B) :-
    starš(B, A), starš(A, C).
```

Ta hipoteza pokrije štiri pozitivne primere in je konsistentna (ne pokrije nobenega negativnega primera), njena kompresivnost pa znaša $4 - 0 - 2 = 2$, torej doseže višjo

oceno od tiste, ki bi nas privedla do želenega rezultata. Ker je ta hipoteza konsistentna, nad njo ni mogoče vršiti specializacije zaprtega sveta, oziroma je njen rezultat enak vhodni hipotezi. △

V takšnih primerih uporaba CWS kot postprocesiranja zgreši najboljšo hipotezo. V nekoliko splošnejšem opisu denimo, da imamo opraviti s problemom, za katerega je $H = P \leftarrow A_1 \ldots A_n, \neg(B_1 \ldots B_m)$ popolna in konsistentna rešitev. Prav tako naj bo presek spremenljivk v literalih $B_1 \ldots B_m$ in $A_1 \ldots A_n$ enak množici spremenljivk, ki nastopajo v glavi (tj. literali v $B_1 \ldots B_m$ vse vrednosti na svojih vhodnih mestih prejmejo neposredno iz glave). V takšnem primeru lahko hipotezo H zapišemo kot $H = P \leftarrow A_1 \ldots A_n, \neg L_a$, kjer ima L_a predikatni simbol P_a z enakimi argumenti kot P in je definiran kot $P_a \leftarrow B_1 \ldots B_m$. Hipotezo takšne oblike je mogoče najti z uporabo specializacije zaprtega sveta, če se v izhodni teoriji nahaja hipoteza $H' = P \leftarrow A_1 \ldots A_n$. Če naj bo ta stavek del izhodne teorije, mora njegova ocena biti dovolj visoka. Do težav pride, kadar je negirani del hipoteze ključen za njeno oceno.

Da bi omogočili sistemom ILP, da tudi v takšnih primerih odkrijejo ciljno hipotezo, predlagamo premik izvajanja specializacije zaprtega sveta v sam proces indukcije.

3.5 Specializacija zaprtega sveta kot del indukcije

V prejšnjem podpoglavju smo izpostavili slabosti specializacije zaprtega sveta, če jo vršimo le za izhodno teorijo, ki jo proizvede sistem ILP z uporabo zgolj pozitivnih literalov. Takšen pristop predpostavlja, da je najboljša rešitev podanega problema specializacija najboljše rešitve, ki sestoji zgolj iz pozitivnih literalov. Kot smo pokazali, temu ni nujno tako.

Negativne primere, pokrite z neko hipotezo, lahko razdelimo v dve vrsti:

neizogibni Zanje ni mogoče sestaviti (kompresivne) hipoteze, ki bi jih ločila od pozitivnih,

izogibni So v pokritju trenutne hipoteze zaradi njene pretirane splošnosti. S specializacijo hipoteze jih je mogoče odstraniti iz pokritja.

V tej točki se postavlja povsem praktično vprašanje, kako oba tipa pokritih negativnih primerov med seboj ločiti. Odgovor na to vprašanje nam s svojim načinom delovanja nudijo sistemi ILP, ki delujejo po principu od spodaj navzgor. Za njih je namreč značilno, da za podano množico pozitivnih primerov zgradijo *najbolj specifično*

hipotezo, ki jih še pokriva. Postavimo se torej v situacijo, ko smo za par primerov e_1 in e_2 izračunali njuno posplošitev (z uporabo enega izmed operatorjev posplošitve) H. Kadar na tak način dobljena H ni konsistentna in torej pokriva neko neprazno množico negativnih primerov E_c^-, vemo, da v jeziku hipotez sistema ILP *ne obstaja* hipoteza, ki bi pokrila primera e_1 in e_2 ter bila konsistentna. To dejstvo izhaja neposredno iz delovanja operatorjev, saj je H po definiciji najbolj specifična. V takšnih primerih so primeri v množici E_c^- neizogibni.

To lastnost lahko izkoristimo, da uporabljamo negacijo le za odstranjevanje neizogibno pokritih negativnih primerov.

3.6 ProGolemNot

Odstranjevanje neizogibnih negativnih primerov iz pokritja smo preizkusili na primeru sistema ILP ProGolem [39]. ProGolem predstavlja trenutni standard med sistemi, ki delujejo po principu od spodaj navzgor. Kot svoj operator posplošitve namreč uporablja asimetrično relativno minimalno generalizacijo, s čimer se izogne eksponentni rasti velikosti hipoteze z dodajanjem novih primerov v pokritje, kot se to zgodi v primeru sistemov, ki uporabljajo relativno najmanj splošno posplošitev (Golem [37]).

Da bi ločili neizogibne negativne primere od izogibnih, premaknemo izvedbo CWS neposredno za uporabo operatorja posplošitve, s čimer zagotovimo, da množica pokritih negativnih primerov vsebuje zgolj neizogibne izjeme. Ta algoritem smo poimenovali ProGolemNot in je podan kot algoritem 10.

Od takšne uporabe specializacije zaprtega sveta pričakujemo dve prednosti pred klasično uporabo kot koraka postprocesiranja:

1. Sistem se ne zanaša več na predpostavko, da je najboljšo rešitev problema moč dobiti s specializacijo najboljše hipoteze, zgrajene zgolj z uporabo pozitivnih literalov. Zaradi tega je sistem zmožen uspešno najti popolne in konsistentne rešitve, kjer standardni pristop odpove.

2. Sistem uporablja negacijo le kadar je to nujno potrebno. V takšnem režimu uporabe CWS ne uporabljamo negacije za odstranjevanje tistih negativnih primerov iz pokritja, ki so vanj prišli kot posledica pretirano splošne hipoteze.

Kljub tema prednostima pa takšen pristop sproža pomisleke v dveh pogledih. Prva potencialna težava izhaja iz dejstva, da se specializacija zaprtega sveta izvede nad vsako

Vhod : Množica pozitivnih primerov E^+,
množica negativnih primerov E^-,
predznanje BK
Izhod : Teorija T

---

function progolemnot(E^+, E^-, BK) *is*
  $T \leftarrow \{\}$;
  $S \leftarrow E^+$;
  while $S \neq \emptyset$ *do*
    $pari \leftarrow izberiPare(S, steviloParov)$;
    $kandidati \leftarrow \{\}$;
    for $(e_i, e_j) \in pari$ *do*
      $\bot_j \leftarrow nasicenje(e_j, BK)$;
      $armg_{ij} \leftarrow ARMG(e_i | \bot_j)$;
      $armg_{ij}^{cws} \leftarrow cws(armg_{ij}, E^+, E^-, BK)$;
      if $ocena(armg_{ij}) > ocena(armg_{ij}^{cws})$ *then*
        $kandidati \leftarrow kandidati \cup armg_{ij}$;
      else
        $kandidati \leftarrow kandidati \cup armg_{ij}^{cws}$;
      end
    end
    $najkandidati \leftarrow najboljsi(kandidati, velikostSnopa)$;
    $najkandidati \leftarrow \bigcup_{H \in najkandidati} poz\_red(H, E^+, E^-)$;
    $najhipoteza \leftarrow najboljsi(najkandidati, 1)$;
    $najhipoteza \leftarrow neg\_red(najhipoteza)$;
    $T \leftarrow T \cup najhipoteza$;
    $S \leftarrow S \setminus pokritje(najhipoteza, S)$;
  end
  return T;
end

Algoritem 10: ProGolemNot.

kandidatno hipotezo. Omejimo število hipotez, ki jih posamezen sistem ILP preizkusi, na k. Za vsako izmed teh k hipotez se ustvari nova učna naloga z novim naborom učnih primerov – v najslabšem primeru enake velikosti, le z obrnjenimi vlogami pozitivnih in negativnih primerov. Ta učna naloga ponovno izpelje največ k novih hipotez – število hipotez, ki jih naš sistem v tej obliki izpelje in oceni, je torej reda k^2. Pričakujemo torej znatno upočasnitev v primerjavi z osnovnim sistemom in sistemom s klasično uporabo specializacije zaprtega sveta. Pri klasični uporabi namreč sistem najprej zgradi in oceni k hipotez, izmed katerih se jih nato v izhodni teoriji pojavi $n_h \leq k$. Nad njimi se izvede specializacija zaprtega sveta, kar privede do generiranja $n_h k$ dodatnih hipotez, katerih skupno število je torej enako $(1 + n_h)k \leq k^2$ – do enakosti bi prišlo v primeru, če bi čisto vsaka izmed preizkušenih hipotez bila del izhodne teorije.

Da bi lahko izpostavili drugo potencialno težavo takšnega pristopa, si moramo pobliže pogledati, kako se ob prisotnosti specializacije zaprtega sveta obnaša ocena hipoteze. Zaradi splošnosti se na tem mestu ne bomo omejili zgolj na en sam predikat izjem, temveč bomo obravnavali splošno situacijo, kjer dovoljujemo tudi "izjeme k izjemam" – situacije, kjer se znotraj predikata izjem nahaja klic drugega predikata izjem. Do takšne situacije lahko pride, če nad pravilom, zgrajenim za prepoznavanje izjem, uporabimo specializacijo zaprtega sveta. Takšen scenarij je podrobneje obravnavan v podpoglavju 3.9.

Naj bo množica novouvedenih predikatov izjem enaka $AP = \{AP_1, \ldots AP_n\}$, kjer je vsak AP_i, $1 < i < n$, definiran kot $AP_i : -\beta_i, \backslash+ AP_{i+1}$, le AP_n je enak $AP_n : -\beta_n$, kjer β_i predstavlja konjunkcijo pozitivnih literalov in AP_i predikat za prepoznavanje izjem, ki je bil uveden za specializacijo definicije predikata AP_{i-1} (z izjemo AP_1, ki je predikat, uveden v namen specializacije osnovnega stavka). Nadaljnje za podano definicijo predikata $H = P : -\beta_0, \backslash+ AB_1$, ker je AB_1 literal s predikatnim simbolom AP_1, definiramo notacijo $\overline{H}$ in $\overline{AP_1}$ kot:

$$\overline{H} = \text{pozitivni del } H$$
$$\overline{AP_i} = \text{pozitivni del } AP_i$$

Primer 17 prikazuje vrednosti $\overline{H}$ in $\overline{AP_i}$ na konkretnem primeru.

Primer 17: Naj bodo podane sledeče definicije predikatov:

```
vzhodno(A) :-
    \+ AP1(A).
```

```
AP1(A) :-
    vagon(A, B), zaprt(B), \+ AP2(A).

AP2(A) :-
    vagon(A, B), kratek(B).
```

Izraz H ustreza definiciji predikata `vzhodno`. Vrednost $\overline{H}$ je enaka `vzhodno(A)`. Z izrazom $\overline{AP_1}$ označimo `AP1(A) :- vagon(A, B), zaprt(B).`, $\overline{AP_2}$ pa je kar enaka definiciji predikata AP_2. △

Najprej si oglejmo, kako z uporabo predlagane notacije izrazimo pokritje hipoteze H s predikati izjem AP_1 ... AP_N.

$$pokritje(H, E) = pokritje(\overline{H}, E) \setminus pokritje(AP_1, E)$$

$$pokritje(AP_i, E) = \begin{cases} pokritje(\overline{AP_i}, E) \setminus pokritje(AP_j, E) & ; j = i + 1, i < n, \\ pokritje(APi_) & ; i = n \end{cases}$$

Če v formulo pokritja hipoteze vstavimo formulo pokritja predikata izjem, dobimo

$$\begin{aligned} pokritje(H, E) &= pokritje(\overline{H}, E) \setminus (pokritje(\overline{AP_1}, E) \setminus pokritje(AP_2, E)) = \\ &= (pokritje(\overline{H}, E) \setminus (pokritje(\overline{AP_1}, E) \setminus (pokritje(\overline{AP_2}, E) \setminus \\ &(pokritje(\overline{AP_3}, E) \setminus (... \setminus pokritje(\overline{AP_n}, E)) ...)) \end{aligned}$$

Iz takšnega zapisa je razvidno, kakšno vlogo igrajo uvedeni predikati izjem glede na njihovo globino. Predikati izjem na lihih globinah skrbijo za odstranjevanje napačno pokritih negativnih primerov iz pokritja hipoteze, predikati izjem na sodih globinah pa v pokritje privedejo pozitivne primere, ki so iz pokritja izpadli zaradi splošnosti predikatov izjem na lihih globinah. Vsak predikat izjeme tako popravlja neželeno spremembo pokritja, do katere pride zaradi splošnosti predikata izjem na prejšnjem nivoju.

Sedaj pa z novouvedeno notacijo poskusimo natančneje izraziti kompresijo podane hipoteze H. Osnovni izračun kompresije je enak

$$kompresija(H) = |pokritje(H, E^+)| - |pokritje(H, E^-)| - dolzina(H)$$

Zapis kompresije lahko razširimo z uporabo uvedene notacije v

$$\begin{aligned} kompresija(H) = &|pokritje(\overline{H}, E^+) \setminus pokritje(AP_1, E^+)| - \\ &|pokritje(\overline{H}, E^-) \setminus pokritje(AP_1, E^-)| - \\ &dolzina(\overline{H}) - \sum_{i=1}^{n} dolzina(AP_i) \end{aligned} \tag{3.1}$$

Če iz enačbe odpravimo razliko množic, pridemo do

$$\begin{aligned} kompresija(H) = &|pokritje(\overline{H}, E^+)| - |pokritje(AP_1, E^+) \cap pokritje(\overline{H}, E^+)| - \\ &(|pokritje(\overline{H}, E^-)| - |pokritje(AP_1, E^-) \cap pokritj(\overline{H}, E^-))| - \\ &dolzina(\overline{H}) - \sum_{i=1}^{n} dolzina(AP_i) \end{aligned}$$

Iz česar po preureditvi členov dobimo

$$\begin{aligned} kompresija(H) = &(|pokritje(\overline{H}, E^+)| - |pokritje(\overline{H}, E^-)| - dolzina(\overline{H})) \\ &+ (|pokritje(AP_1, E^-) \cap pokritje(\overline{H}, E^-)| - \\ &|pokritje(AP_1, E^+) \cap pokritje(\overline{H}, E^+)| - \sum_{i=1}^{n} dolzina(AP_i)) \end{aligned} \tag{3.2}$$

Iz enačbe 3.1 sledi, da ima nanjo pozitiven vpliv:

- Pozitivno pokritje osnovne hipoteze, ki poveča vrednost člena $|pokritje(\overline{H}, E^+) \setminus pokritje(AP_1, E^+)|$.
- Negativno pokritje predikata izjem, povečanje katerega zmanjša vrednost člena $|pokritje(\overline{H}, E^-) \setminus pokritje(AP_1, E^-)|$.

Negativno pa nanjo vplivajo:

- Pozitivno pokritje predikata izjem, ki zmanjša vrednost člena $|pokritje(\overline{H}, E^+) \setminus pokritje(AP_1, E^+)|$.

- Negativno pokritje osnovne hipoteze, s katerim se povečuje tudi vrednost člena $|pokritje(\overline{H}, E^-) \setminus pokritje(AP_1, E^-)|$.
- Dolžina definicije hipoteze in predikatov izjem.

Takšen zapis kompresije hipoteze nas pripelje do dveh sklepov.

Iz enačbe 3.2 je razvidno, da je kompresija končne hipoteze vsota kompresije pozitivnega dela in kompresije negativnega dela z ozirom na primere, pokrite s strani pozitivnega dela, z zamenjanimi vlogami pozitivnih in negativnih primerov. V kolikor je povečanje ocene zaradi odstranjevanja napačno pokritih primerov iz pokritja znatno, bo sistem ILP takšno rešitev zgrešil, saj pozitivni del hipoteze sam po sebi prejme nizko oceno zaradi visokega pokritja negativnih primerov. V takšnih primerih sistemi ILP pogosto izberejo bolj specifično hipotezo z manjšim pokritjem negativnih primerov.

Vendar takšen zapis izpostavi tudi potencialno težavo predlaganega premika specializacije zaprtega sveta v sam proces indukcije. Pri učenju predikata izjem AP so sistemu na voljo namreč le primeri, ki jih pokriva najbolj specifična hipoteza, dobljena iz naključno izbranih primerov (semen). Takšna množica še ne vsebuje vseh pozitivnih primerov, za katere končna hipoteza $\overline{H}$ uspe, kar lahko privede do situacije, da je predikat izjem AP z ozirom nanje preveč splošen (pri njegovi konstrukciji namreč še niso bili pokriti s strani hipoteze, ki jo specializiramo) in so zaradi tega odstranjeni iz pokritja, saj AP zanje uspe. Dejstvo, da pri učenju predikata AP ne uporabimo vseh negativnih primerov, ki jih pokrije končna $\overline{H}$ (novi negativni primeri lahko namreč pridejo v pokritje tekom nadaljnjega posploševanja generalizacije semen), je v skladu z željo, da odstranjujemo zgolj tiste negativne primere, katerih pokritje je neizogibno.

3.7 ProGolemNRNot

V podpoglavju 3.6 smo izpostavili dve slabosti predlaganega sistema ProGolemNot:

1. Porast časa, potrebnega za reševanje problema.
2. Pretirano splošni predikati izjem.

S sistemom ProGolemNRNot poskušamo ti dve slabosti omiliti.

Da bi znižali časovno kompleksnost sistema ProGolemNot, se naslonimo na sam režim delovanja sistema ProGolem. ProGolem za preiskovanje prostora hipotez uporablja iskanje v snopu. Na vsakem koraku izostri vse hipoteze v trenutnem snopu,

dokler ne doseže meje, kjer se nobena hipoteza v snopu po svoji ostritvi ne izboljša. Na tej točki ProGolem izbere najbolje ocenjeno hipotezo iz snopa in jo doda v izhodno teorijo. Namesto vršenja specializacije zaprtega sveta takoj po izračunu generalizacije semen, predlagamo zamik te operacije do samega konca iskanja hipoteze – do točke, kjer ne pride več do izboljšanja nobene izmed hipotez v snopu. Na tej točki nad vsemi hipotezami v snopu izvedemo specializacijo zaprtega sveta in šele nato izberemo najboljšo izmed njih.

Specializacija zaprtega sveta se tako izvrši le nad b hipotezami, kjer je b velikost snopa. Vsaka izmed hipotez $H_1 \ldots H_b$ ima na tej točki svoje pokritje pozitivnih primerov E_i^+ in svoje pokritje negativnih primerov E_i^-. Na tem mestu lahko množica E_i^- vsebuje tudi izogibne negativne primere – primere, ki so v pokritje prišli šele z nadaljnjim posploševanjem hipoteze, izračunane iz semen. Ker želimo iz pokritja z uporabo negacije odstranjevati le primere, ki jih drugače ni mogoče, si moramo pokritje negativnih primerov osnovne hipoteze (tj. hipoteze izračunane iz semen, preden je bila dodatno posplošena) zapomniti ob njenem izračunu. Tako shranjeno množico negativnih primerov nato uporabimo kot vhod v specializacijo zaprtega sveta, s čimer zagotovimo, da odstranjujemo le neizogibne primere.

Ta premik pa nam poleg krajšega časa izvajanja nudi način, kako nasloviti drugo težavo sistema ProGolemNot. Ob zaključku iskanja v snopu vemo, da do njenih nadaljnjih generalizacij ne bo prišlo, zaradi česar se množica pokritih pozitivnih primerov ne bo več povečevala. Pri specializaciji zaprtega sveta tako lahko uporabimo množico pozitivnih primerov, pokritih s posplošeno hipotezo, s čimer se izognemo generiranju pretirano splošnih predikatov izjem.

S takšnim pristopom pričakujemo, da se bo znatno zmanjšal čas izvajanja v primerjavi s sistemom ProGolemNot, ter da se bomo izognili pretirano splošnim predikatom izjem. Potrebno pa se je zavedati, da smo s tem v sistem ponovno vnesli nekoliko pristranskosti, kateri smo se želeli izogniti s premikom specializacije zaprtega sveta v sam proces indukcije. V takšnem režimu delovanja se namreč specializira le b najboljših hipotez, sestavljenih izključno iz pozitivnih literalov.

Oris algoritma ProGolemNRNot je podan kot algoritem 11.

3.8 Omejitve specializacije zaprtega sveta

Predlagane rešitve si s klasičnim pristopom specializacije zaprtega sveta delijo omejitve pri naboru hipotez, ki jih je na takšen način mogoče izpeljati. Omejitev izhaja iz

Vhod : Množici pozitivnih in negativnih primerov E^+ ter E^-,
predznanje BK
Izhod : Teorija T

---

function ProGolemNRNot(E^+, E^-, BK) is
  $T \leftarrow \{\}$;
  $S \leftarrow E^+$;
  while $S \neq \emptyset$ *do*
    $pari \leftarrow izberiPare(S, stParov)$;
    $kandidati \leftarrow \{\}$;
    for $(e_i, e_j) \in pari$ *do*
      $\perp_j \leftarrow nasicenje(e_j, BK)$;
      $cws_{ij} \leftarrow ARMG(e_i | \perp_j)$;
      $cws_{ij}negPok \leftarrow pokritje(cws_{ij}, E^-)$;
      $kandidati \leftarrow kandidati \cup (cws_{ij}, cws_{ij}negPok)$;
    end
    $najkandidati \leftarrow najboljsi(kandidati, velikostSnopa)$;
    $najkandidati \leftarrow \bigcup_{(H,NC)\in najkandidati}(poz\_red(H, S), NC)$;
    $najkandidati \leftarrow \bigcup_{(H,NC)\in najkandidati}(neg\_red(H, E^+, E^-), NC)$;
    $najkandidati^{cws} \leftarrow \bigcup_{(H,NC)\in najkandidati} cws(H, E^+, NC, BK)$;
    $najhipoteza \leftarrow najboljsi(najkandidati, 1)$;
    $najhipoteza^{cws} \leftarrow najboljsi(najkandidati^{cws}, 1)$;
    if $ocena(najhipoteza) > ocena(najhipoteza^{cws})$ *then*
      $T \leftarrow T \cup najhipoteza$;
      $S \leftarrow S \setminus pokritje(najhipoteza, S)$;
    else
      $T \leftarrow T \cup najHipoteza^{cws}$;
      $S \leftarrow S \setminus pokritje(najhipoteza^{cws}, S)$;
    end
  end
  return T
end

Algoritem 11: ProGolemNRNot.

dejstva, da poteka učenje predikata izjem popolnoma neodvisno (če izvzamemo način določanja pozitivnih in negativnih primerov, uporabljenih pri njegovem učenju) od učenja definicije predikata, ki ga specializiramo. Poglejmo si denimo primer iz znane domene vlakov Michalskega. Naj bo opis ciljnega predikata enak

```
vzhodni(A) :-
    ima_vagon(A, B), zaprt(B), \+ kratek(B).
```

Z uporabo specializacije zaprtega sveta ni mogoče izpeljati ekvivalentnega opisa. Denimo, da nek sistem ILP izpelje želeni pozitivni del hipoteze

```
vzhodni(A) :- ima_vagon(A, B), zaprt(B)
```

Pozitivni del pokriva vse pozitivne primere E^+ (je popoln) in neko množico negativnih primerov E_c^-. Zaradi nekonsistentnosti to hipotezo dodatno specializiramo z uporabo specializacije zaprtega sveta – sistemu podamo novo učno nalogo: učenje predikata izjem AP z množico pozitivnih primerov E_c^- in množico negativnih primerov E^+. Pri učenju predikata izjem ima sistem dostop zgolj do spremenljivk, ki se nahajajo v glavi hipoteze, zaradi česar ni mogoče v telesu hipoteze zahtevati, da je vrednost neke nove spremenljivke enaka vrednosti, ki ji je dodeljena v pozitivnem delu hipoteze. Za ilustracijo si poglejmo eno izmed možnih rešitev, kot bi jo vrnil sistem:

```
vzhodni(A) :-
    ima_vagon(A, B), zaprt(B), \+ AP1(A).

AP1(A) :-
    ima_vagon(A, B), kratek(B).
```

Takšna rešitev ni ekvivalentna definiciji ciljnega predikata. Nimamo namreč zagotovila, da sta vrednosti spremenljivke `B` v definiciji predikata `vzhodni` in `AP1` enaki. V naravnem jeziku nam ciljni predikat pravi, da gre nek vlak proti vzhodu, če vsebuje zaprt vagon, ki ni kratek. Rešitev, ki jo vrne sistem, pa trdi, da gredo proti vzhodu tisti vlaki, ki imajo zaprt vagon in nobenega kratkega. Rešitvi sta ekvivalentni v primerih, kjer imajo vlaki po en sam vagon, v splošnem pa zaradi navedenega razloga nista ekvivalentni.

S specializacijo zaprtega sveta tako ni mogoče sestaviti hipotez, ki bi si delile spremenljivke izven glave hipoteze med svojim pozitivnim in negiranim delom.

3.9 Odkrivanje večnivojskih izjem

Nekateri problemi za svojo rešitev terjajo uporabo gnezdenih negacij. Vzemimo si kot primer Michalskijev problem vlakov brez podanih predikatov za dolge in odprte vagone, kjer ima zaradi omejitev specializacije zaprtega sveta (podpoglavje 3.8) vsak vlak en sam vagon. S takšnim predznanjem lahko ustreznico predikatu

```
vzhodni(A) :-
    ima_vagon(A, B), \+ (dolg(B), zaprt(B)).
```

izrazimo z uporabo gnezdenih negacij

```
vzhodni(A) :-
    \+ (ima_vagon(A, B), zaprt(B), \+ kratek(B)).
```

Srinivasan in drugi v [54] prilagodijo specializacijo zaprtega sveta za odkrivanje večnivojskih izjem tako, da v učenje predikata izjem vključijo rekurzivni klic specializacije zaprtega sveta. Ko se torej sistem nauči predikata izjem, preveri, če ta nepravilno iz pokritja odstrani katerega izmed pozitivnih primerov osnovne učne naloge. Kadar je ta množica neprazna, se nad predikatom izjem lahko ponovno uporabi specializacija zaprtega sveta, s katero poskušamo te primere ponovno privesti v pokritje.

Pri uporabi rekurzivnega pristopa je treba določiti, kdaj zaključiti z uvajanjem novih predikatov izjem v hipotezo. Na to vprašanje lahko odgovorimo na dva načina:

- S specializacijo zaključimo, ko postane hipoteza konsistentna. Do te situacije pride tudi, kadar kompresivne hipoteze za predikat izjem ni več mogoče najti.
- Uvedemo največjo dovoljeno globino gnezdenih negacij.

V drugem pristopu je potrebno vnaprej definirati omejitev nad globino rekurzije klicev specializacije in jo kot parameter podati sistemu. Prilagoditev algoritma specializacije zaprtega sveta, ki odkriva tudi hipoteze z gnezdenimi negacijami z omejeno globino, je podana kot algoritem 12.

Uporaba omejitve globine na n nam omogoča oceniti časovno zahtevnost takšnega pristopa. Omejimo število kandidatnih hipotez, ki jih zgenerira sistem, na k. Pri sistemu ProGolemNot se v primeru, ko dovolimo le en nivo izjem, za vsako izmed k hipotez ustvari k definicij kandidatnih izjem, kar nas privede do k^2 generiranih hipotez.

Vhod : Hipoteza H,
množica pozitivnih primerov E^+,
množica negativnih primerov E^-,
predznanje BK,
preostalo število dovoljenih gnezdenj *nivo*
Izhod : Specializacija hipoteze H

```
function cws_ml(H, E+, E−, BK, stNivojev) is
    poz_pokritje ⟵ pokritje(H, E+) ;
    neg_pokritje ⟵ pokritje(H, E−) ;
    if |neg_pokritje| < 2 then
        return H;
    end
    aux_h ⟵ nauci(neg_pokritje, poz_pokritje, BK) ;
    if nivo > 0 then
        pomozna_h ⟵
        cws_ml(pomozna_h, neg_pokritje, poz_pokritje, BK, nivo − 1);
    end
    H′ ⟵ H, ¬pomozna_h ;
    if ocena(H′) > ocena(H) then
        return H′ ;
    else
        return H;
    end
end
```

Algoritem 12: Specializacija zaprtega sveta z gnezdeno negacijo omejene globine.

Z vsakim dodatnim nivojem izjem se število generiranih hipotez poveča za faktor k, kar nas privede do sklepa, da je število generiranih hipotez pri omejitvi globine izjem na n enako k^{n+1}. Število hipotez tako narašča eksponentno z največjo dovoljeno globino izjem.

Rast števila generiranih hipotez je pri uporabi sistema ProGolemNRNot počasnejša, vendar še vedno eksponentna in je enaka kb^n, kjer je b velikost snopa.

Zaradi strme rasti časa, potrebnega za izračun hipoteze, z večanjem največje globine izjem, je uporabnost takšnega pristopa v praksi vprašljiva.

3.10 Pristranskost jezika pri uporabi specializacije zaprtega sveta

V tem poglavju si najprej podrobneje ogledamo pristranskost jezika sistemov, ki uporabljajo specializacijo zaprtega sveta, nakar se osredotočimo na sistema ProGolemNot ter ProGolemNRNot.

Sistemi ILP se med seboj razlikujejo tudi glede na svojo pristranskost jezika. Med bolj znane pristranskosti jezikov sodi *ij*-determiniranost, pri uporabi katere dolžina relativne najmanj splošne posplošitve raste polinomsko in ne eksponentno, kot sicer. *ij*-determiniranost je iz tega razloga pristranskost jezika sistema Golem [37].

Druga, trenutno bolj razširjena pristranskost, je omejitev prostora preiskovanja s spodnjim stavkom naključno izbranega primera. Definiranje pristranskosti jezika je s tem prepuščena uporabniku prek uporabe deklaracij načina. Z njimi uporabnik za vsak predikat v predznanju določi:

Vhodna mesta Spremenljivke, ki nastopajo na teh mestih, morajo ob izvedbi imeti že dodeljeno vrednost.

Izhodna mesta Spremenljivke na teh mestih bodo po izvedbi imele dodeljeno vrednost.

Konstantna mesta Na teh mestih ne more nastopati spremenljivka.

Priklic Največje število uspešnih instanciacij za en nabor vhodov.

Poleg deklaracij načina mora, zaradi obstoja neskončnih spodnjih stavkov, uporabnik navadno določiti tudi največjo dovoljeno globino literalov v spodnjem stavku.

S tem uporabnik vnese omejitev jezika hipotez – sistem namreč ne more generirati stavkov, bolj specifičnih od spodnjih stavkov, ki pa so podvrženi omejitvam globine literalov ter deklaracij načina.

Načina delovanja pri sistemih, ki svoj prostor hipotez navzdol omejujejo s spodnjimi stavki, določa še eno pristranskost jezika – sposobnost deljenja spremenljivk (ang. *variable splitting*). Sistemi, ki deljenja spremenljivk ne omogočajo, so namreč nezmožni rešiti nekatere probleme. Primer takšnega problema je podan v poglavju 2.1.2.

Podan naj bo poljuben sistem ILP I s pristranskim jezikom hipotez $\mathscr{L}$. Takšen sistem nadgradimo z uporabo specializacije zaprtega sveta. Ker se za izračun pomožne hipoteze (tj. definicije predikata, katerega klic bo kot negiran literal pripet k originalni hipotezi) uporablja nespremenjen osnovni sistem, je pristranskost jezika, v katerem se nahaja njena definicija, enak pristranskosti jezika, uporabljenega pri gradnji osnovne hipoteze.

Do edine spremembe glede na sistem I pride pri klicu pomožnega predikata. Jezik je v ta namen treba ustrezno razširiti tako, da ta klic omogoča. Poimenujmo tako prilagojen jezik $\mathscr{L}^{CWS}$.

$$\mathscr{L}^{CWS} = \mathscr{L} \cup \{\mathscr{L} \cdot \{\backslash + \texttt{pomožni\_klic}\}\}$$

Pri ciljnem predikatu `cilj` s številnostjo n (`cilj(A1, ... An)`) je `pomožni_klic` oblike `cilj_pom_i(A1, ... An)`, kjer je `i` oznaka, unikatna za vsako hipotezo iz jezika $\mathscr{L}^{CWS}$. Pomožni klic sprejme *identične* argumente, kot se ti pojavijo v glavi hipoteze. Posledice te omejitve smo razdelali v poglavju 3.8.

Posplošitev na poljubno globino dovoljenih negacij je trivialna. Pri posplošitvi na n negacij končno teorijo sestavlja množica največ n hipotez iz jezika $\mathscr{L}^{CWS}$ in natanko ena iz jezika $\mathscr{L}$.

Jezik, iz katerega izhaja grajena hipoteza, je pri sistemu ProGolem navzdol omejen s posplošitvijo spodnjega stavka naključno izbranega primera. Za izračun spodnjega stavka ProGolem uporablja deklaracije načina in omejitev globine literalov v njem. Posplošitev spodnjega stavka je dobljena z zamenjavo konstant s spremenljivkami, pri čemer se vsaka konstanta vedno zamenja z isto spremenljivko (mehanizem je podrobneje opisan v podpoglavju 2.1.2). Na takšen način dobljen jezik z omejitvijo globine i in deklaracijami načina M označimo z $\mathscr{L}_i(M)$. Za dano omejitev globine n bo ProGolem(NR)Not zgradil n hipotez iz jezika $\mathscr{L}_i^{CWS}(M)$ in eno iz jezika $\mathscr{L}_i(M)$.

Opozoriti velja, da se pristranskost jezika s premikom specializacije zaprtega sveta v sam proces indukcije ne spremeni. Ob pravem naboru parametrov sistema in primerov je mogoče z uporabo CWS kot korakom postprocesiranja generirati katerokoli hipotezo, proizvedeno s strani sistema z uporabo CWS znotraj procesa indukcije.

3.11 Eksperimentalno ovrednotenje

Pri ovrednotenju predlaganih sistemov smo najprej z uporabo manjših primerov z znanimi definicijami ciljnih predikatov preverili, če so ti sistemi resnično sposobni najti prave definicije ciljnih predikatov tudi v primerih, kjer jih sistemi s klasično uporabo specializacije zaprtega sveta zgrešijo.

V drugem sklopu eksperimentov primerjamo sistema ProGolemNot in ProGolemNRNot z osnovnim sistemom ProGolem in sistemom ProGolem z uporabo specializacije zaprtega sveta kot koraka postprocesiranja (označen kot ProGolem+CWS). Dodatno sisteme primerjamo tudi s sistemom Aleph tako brez kot tudi z uporabo specializacije zaprtega sveta kot postprocesiranja (Aleph+CWS).

3.11.1 Preverjanje delovanja koncepta

Osnovno delovanje pristopa smo preizkusili na primeru Michalskijevih vlakov. Vlake smo generirali z uporabo pravila:

```
vzhodni(A) :-
  \+ (ima_vagon(A, B), zaprt(B), \+ kratek(B)).
```

Zaradi omejitev delovanja specializacije zaprtega sveta (podpoglavje 3.8) smo se pri generiranju omejili zgolj na vlake, ki imajo en sam vagon. Le v takšnem primeru lahko namreč zgeneriramo ekvivalenten opis koncepta.

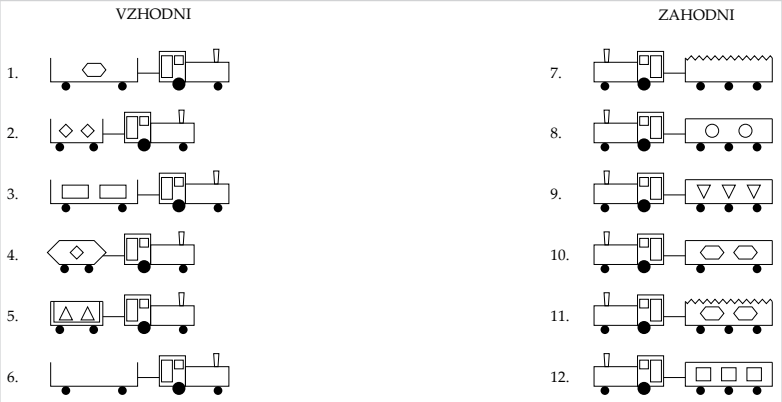
Da bi sistemu vsilili uporabo negacije, smo iz predznanja odstranili predikata `dolg` in `odprt`. Ustvarili smo skupno 192 učnih primerov, izmed katerih je polovica primerov pozitivnih in polovica negativnih. Del uporabljenih primerov prikazuje slika 3.1.

ProGolem in Aleph oba najdeta konsistentno, a nepopolno, rešitev problema:

```
vzhodni(A) :-
    ima_vagon(A, B), kratek(B).
```

Ker je ta rešitev konsistentna tudi varianti teh sistemov s specializacijo zaprtega sveta kot postprocesiranjem, ostaneta pri takšni izhodni teoriji.

Nasprotno pa tako ProGolemNot kot ProGolemNRNot najdeta konsistentno in popolno hipotezo oblike:



Slika 3.1

Del umetne podatkovne množice, uporabljene za testiranje delovanja koncepta.

```
vzhodni(A) :-
    \+ negPred17(A).

negPred17(A) :-
    \+ negPred2(A), ima_vagon(A, B), zaprt(B).

negPred2(A) :-
    ima_vagon(A, B), kratek(B).
```

Hipoteza, ki jo najdeta ProGolemNot in ProGolemNRNot v osnovi pokrije vse primere. Nato s predikatom `negPred17` iz pokritja odstrani vse vlake, ki imajo zaprt vagon. Iz pokritja `negPred17` se nato odstranijo še vsi vlaki s kratkim vagonom, s čimer je pokritje teorije za koncept `vzhodni` enako tistim vlakom, katerih vagon je bodisi dolg in odprt bodisi kratek.

3.11.2 Eksperimenti

Izvedene eksperimente razdelimo na dva sklopa. V prvem sklopu gre za testiranje opisanih sistemov na umetni podatkovni množici z znanim opisom ciljnega predikata. S tem smo želeli opazovati, kako se predlagane metode obnašajo ob prisotnosti šuma v podatkih. Pri uvajanju n% šuma v podatke smo n% primerom v podatkovnih množici naključno zamenjali pripadnost iz množice negativnih primerov v množico pozitivnih

primerov oz. obratno.

Drugi sklop sestavlja 19 realnih domen, ki se pogosto uporabljajo za primerjavo različnih sistemov ILP med seboj. Sisteme smo med seboj primerjali na podlagi desetih izvajanj desetkratnega prečnega preverjanja (z izjemo domen pyrimidines ter dsstox, kjer smo zaradi že določene razdelitve v 5 podmnožic takšno razdelitev ohranili in uporabili deset meritev petkratnega prečnega preverjanja). Desetkratno izvajanje je namenjeno zmanjšanju vpliva naključnega izbiranja primerov in oceni časa, potrebnega za izdelavo rešitve.

Sisteme, ki smo jih med seboj primerjali, smo označili z:

Aleph Osnovni sistem Aleph.

AlephCWS Sistem Aleph z uporabo specializacije zaprtega sveta kot koraka postprocesiranja.

ProGolem Osnovni sistem ProGolem v načinu parov primerov (ang. pairs mode).

ProGolemCWS Sistem ProGolem z uporabo specializacije zaprtega sveta na izhodni hipotezi.

ProGolemNot Naš sistem ProGolem, ki uporabi specializacijo zaprtega sveta nad vsako asimetrično relativno minimalno generalizacijo, ki jo izračuna.

ProGolemNRNot Naš sistem ProGolem, ki specializacijo zaprtega sveta uporabi nad hipotezami, ki se nahajajo v snopu ob zaključku preiskovanja.

Uporabljene umetne podatkovne množice so:

VlakiXX V teh domenah XX zaznamuje delež (v procentih) naključnih primerov, katerim je bila spremenjena pripadnost množici pozitivnih oz. negativnih primerov. Opis koncepta, ki je služil za generiranje teh podatkov je

```
vzhodni(A) :-
    \+ (ima_vagon(A, B), kratek(B), pravokoten(B)).
```

VlakiVecStavkov Ta podatkovna zbirka od sistema terja izgradnjo teorije, ki vsebuje dva stavka. Ciljni opis koncepta je enak:

```
vzhodni(A) :-
    \+ (ima_vagon(A, B), kratek(B)).

vzhodni(A) :-
    \+ (ima_vagon(A, B), pravokoten(B)).
```

VlakiVecNivojev Ta problem za svojo rešitev terja uporabo dveh nivojev negacije. Opis koncepta je enak, kot je bil uporabljen za testiranje delovanja koncepta v podpoglavju 3.11.1:

```
vzhodni(A) :-
    \+ (ima_vagon(A, B), zaprt(B), \+ kratek(B)).
```

Pričakovano se sistema ProGolemNot in ProGolemNRNot dobro izkažeta pri odkrivanju umetnega koncepta. Kot je razvidno iz doseženih točnosti v tabeli 3.1, povsod dosežeta boljše ali enakovredne rezultate kot Aleph, AlephCWS, ProGolem in ProGolemCWS. Pričakovano se čas indukcije (prikazan v tabeli 3.2) poveča, saj ProGolemNot in ProGolemNRNot oba preizkusita večje število hipotez.

Na realnih domenah rezultati niso tako enoznačni. Točnosti najdenih hipotez so prikazani v tabeli 3.3, časi potrebni za izpeljavo izhodne hipoteze pa v tabeli 3.4. Rezultati namigujejo, da se premik izvajanja specializacije zaprtega sveta v proces indukcije s stališča doseženih točnosti izplača – s premikom povsod dosežemo vsaj enako točne napovedi na še ne videnih primerih, kot jih dosega ProGolemCWS.

Statistične primerjave doseženih rezultatov smo opravili na dveh nivojih. Sisteme smo najprej primerjali med seboj na posameznih podatkovnih množicah z uporabo Welchevega t-testa (zaradi neenakih varianc) in Holmove korekcije za večkratne primerjave. V drugem koraku med seboj primerjamo vse preizkušene sisteme na vseh podatkovnih zbirkah hkrati z uporabo Friedmanovega testa z Nemenyijevim post-hoc testom [11].

Tabela 3.5 prikazuje p-vrednosti, dobljene s primerjavo sistema ProGolemNot z vsemi ostalimi sistemi. Najprej se omejimo na primerjavo sistema ProGolemNot z osnovnim sistemom ProGolem ter njegovo inačico z uporabo specializacije zaprtega sveta (ProGolemCWS). V 14 podatkovnih množicah doseže predlagani sistem signifikantno boljše rezultate od osnovne različice sistema ProGolem. Pri primerjanju s sistemom ProGolem, nadgrajenim s CWS, je naš sistem boljši v 13 primerih. V vseh

Tabela 3.1

Točnosti (v procentih), dosežene na umetnih podatkovnih množicah. Poleg točnosti je v oklepaju zapisan rang sistema na tej podatkovni množici.

	Aleph		AlephCWS		ProGolem	
Vlakioo	94.39 ± 5.70	(3.5)	94.39 ± 5.70	(3.5)	92.12 ± 7.94	(6)
Vlakio5	87.16 ± 9.16	(4.5)	87.16 ± 9.16	(4.5)	86.99 ± 10.01	(6)
Vlaki10	77.14 ± 15.37	(6)	81.18 ± 10.29	(3)	78.25 ± 13.86	(5)
Vlaki15	55.55 ± 13.07	(6)	57.69 ± 13.02	(5)	61.31 ± 13.47	(4)
Vlaki20	68.84 ± 20.52	(5)	66.20 ± 22.15	(6)	70.48 ± 13.82	(3)
Vlaki25	58.26 ± 14.25	(6)	64.80 ± 11.72	(3)	64.76 ± 13.49	(4)
VlakiVecStavkov	94.00 ± 6.89	(3)	92.66 ± 8.09	(4)	90.92 ± 8.58	(5.5)
VlakiVecNivojev	76.53 ± 12.08	(6)	80.26 ± 14.29	(5)	85.01 ± 7.68	(3.5)
	ProGolemCWS		ProGolemNot		ProGolemNRNot	
Vlakioo	92.61 ± 6.69	(5)	**99.70 ± 2.23**	(1)	96.53 ± 5.11	(2)
Vlakio5	89.33 ± 8.39	(3)	**95.79 ± 6.64**	(1)	91.66 ± 8.02	(2)
Vlaki10	80.54 ± 11.60	(4)	**85.99 ± 8.97**	(1)	84.18 ± 9.99	(2)
Vlaki15	61.88 ± 12.90	(3)	67.45 ± 11.14	(2)	**67.49 ± 9.39**	(1)
Vlaki20	**72.68 ± 13.10**	(1)	71.15 ± 10.59	(2)	69.73 ± 9.89	(4)
Vlaki25	**66.11 ± 13.99**	(1)	63.53 ± 12.96	(5)	65.76 ± 14.59	(2)
VlakiVecStavkov	90.92 ± 8.58	(5.5)	**98.03 ± 4.87**	(1)	96.96 ± 4.80	(2)
VlakiVecNivojev	85.01 ± 7.68	(3.5)	**100.00 ± 0.00**	(1)	95.51 ± 6.93	(2)

Tabela 3.2

Časi (v sekundah), potrebni za izračun rešitve na umetnih podatkovnih množicah.

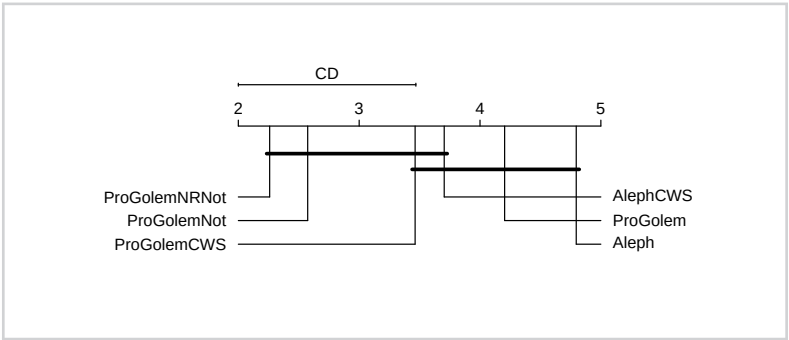
	Aleph	AlephCWS	ProGolem	ProGolemCWS	ProGolemNot	ProGolemNRNot
Vlaki00	1.19 ± 0.03	**0.85 ± 0.04**	3.05 ± 0.15	3.00 ± 0.06	10.85 ± 2.93	3.74 ± 0.23
Vlaki05	1.71 ± 0.04	**0.59 ± 0.05**	3.73 ± 0.10	3.84 ± 0.07	7.33 ± 1.59	3.39 ± 0.13
Vlaki10	3.23 ± 0.06	**0.38 ± 0.05n**	5.29 ± 0.17	6.44 ± 0.10	27.34 ± 2.48	10.07 ± 1.39
Vlaki15	**4.79 ± 0.16**	5.29 ± 0.14	6.81 ± 0.23	9.03 ± 0.15	87.95 ± 7.95	19.00 ± 0.83
Vlaki20	**3.27 ± 0.15**	3.73 ± 0.16	6.49 ± 0.34	7.67 ± 0.36	73.48 ± 5.55	19.91 ± 0.78
Vlaki25	2.13 ± 0.22	**0.69 ± 0.06**	3.95 ± 0.31	6.04 ± 0.14	115.92 ± 8.10	22.62 ± 0.93
VlakiVecStavkov	1.27 ± 0.07	**0.96 ± 0.05**	3.05 ± 0.05	3.11 ± 0.09	4.19 ± 0.75	3.03 ± 0.39
VlakiVecNivojev	4.17 ± 0.14	**2.36 ± 0.10**	3.64 ± 0.05	3.74 ± 0.16	42.14 ± 3.28	3.57 ± 0.30

ostalih primerih je ProGolemNot tema dvema sistemoma enakovreden oz. se njegove dosežene točnosti od njunih ne razlikujejo signifikantno. To nakazuje, da je predlagani sistem sposoben odkriti koristna pravila, do katerih z uporabo sistemov ProGolem ter ProGolemCWS ni mogoče priti. Hkrati lahko ugotovimo, da uporaba negacije znotraj procesa indukcije ne poslabša doseženih točnosti na nobeni izmed uporabljenih podatkovnih zbirk.

V naslednjem koraku v tabeli 3.5 med seboj primerjamo v tem poglavju predlagana sistema ProGolemNot ter ProGolemNRNot. Uporabljeni zamik izvajanja specializacije zaprtega sveta signifikantno poslabša rezultate le na štirih domenah (treh umetnih in eni realni), na ostalih pa med njunimi doseženimi točnostmi ne prihaja do statistično signifikantnih razlik. Do poslabšanja rezultatov sistema ProGolemNRNot v primerjavi s sistemom ProGolemNot pride zaradi uporabe hevristične izbire, nad katerimi hipotezami naj se izvrši specializacija zaprtega sveta.

V zadnji primerjavi nad posameznimi podatkovnimi zbirkami v tabeli 3.5 primerjamo sistem ProGolemNot s sistemoma Aleph ter AlephCWS. Sistem Aleph predstavlja trenutno najbolj razširjen sistem ILP in ima vgrajeno možnost uporabe specializacije zaprtega sveta. Z imenom Aleph označimo njegovo verzijo brez uporabe specializacije zaprtega sveta, z AlephCWS pa primer, ko je ta omogočena. Primerjava teh sistemov ni enoznačna. ProGolemNot je uspešnejši na umetnih domenah, kar nakazuje, da so tudi sistemi, ki delujejo po principu od zgoraj navzdol, z uporabo CWS podvrženi enakim slabostim kot sistemi, ki rešitev iščejo od spodaj navzgor. Pri realnih domenah pa naletimo tako na primere, kjer ProGolemNot daje signifikantno boljše rezultate kot Aleph ter AlephCWS (denimo vse Alzheimers domene), kot tudi primere, kjer se Aleph oz. AlephCWS obnese signifikantno bolje. Pri nekaterih domenah že sam sistem Aleph doseže višjo točnost od sistema ProGolemNot, kar nakazuje, da so nekatere podatkovne množice bolj primerne za reševanje s pristopom od zgoraj navzdol kot s pristopom od spodaj navzgor. Pri ostalih podatkovnih zbirkah, kjer ProGolemNot sicer doseže boljše rezultate kot Aleph, vendar slabše kot AlephCWS, tiči mogoč razlog za takšno obnašanje v dejstvu, da AlephCWS nima omejitve nad globino izjem – ProGolemNot pa vedno izpelje le en nivo izjem. Vsekakor pa na podlagi dobljenih rezultatov ne moremo trditi, da je kateri izmed teh sistemov v boljši od drugih – kvaliteta njihovih rešitev se namreč od domene do domene spreminja.

Obravnavane sisteme primerjamo še preko vseh uporabljenih podatkovnih množic z uporabo Friedmanovega testa. Povprečni rangi testiranih sistemov so prikani v tebeli



Slika 3.2

Slikovni prikaz rezultatov Nemenyijevega post-hoc testa iz tabele 3.7.

3.6. Friedmanov test vrne p-vrednost $9.235 \cdot 10^{-07}$, zato zavrnemo ničelno hipotezo, da med sistemi v rangiranju ni razlik. Za iskanje izvora zaznanih razlik uporabimo Nemenyijev post-hoc test, katerega rezultati so prikazni v tabeli 3.7 in sliki 3.2.

Vir razlik v zbirki sistemov najdemo v primerjavi predlaganih sistemov (ProGolemNot ter ProGolemNRNot) s sistemoma, ki ne uporabljata negacije (Aleph ter ProGolem). Ostale razlike v rangiranju sistemov so nesignifikantne. Čeprav smo pri primerjavi sistemov nad posameznimi podatkovnimi množicami opazili večje razlike med sistemi, rezultat dobljen s Friedmanovim testom ob upoštevanju njegovih specifik ni presenetljiv. Friedmanov test namreč prezre absolutne razlike v doseženih točnostih in se opira le na rangiranje sistemov. Pri podatkovnih zbirkah, kjer sistemi delujejo podobno dobro, je tako njihova razvrstitev naključna zaradi variance pri izbiri naključnih primerov – takšne naključne razvrstitve pa vplivajo na rezultat Friedmanovega testa. Rezultat Nemenyevega post-hoc testa tako pravi, da predlagana uporaba CWS ne nudi izboljšave glede na tradicionalno uporabo CWS. To opažanje se sklada z ostalimi opažanji – za neko novo, neznano podatkovno množico, ne moremo vnaprej trditi, ali bo vključitev CWS v proces indukcije privedla do bolj točnih hipotez ali ne. Uporabnost naših pristopov tako zavisi od problema, ki ga rešujemo.

Tabela 3.3

Točnosti (v procentih) in rangi, ki jih dosežejo posamezni sistemi na realnih podatkovnih množicah.

	Aleph		AlephCWS		ProGolem	
Krk_Illegal	99.65 ± 0.09	(6)	**100 ± 0.02**	(1)	99.79 ± 0.20	(5)
Kr_vs_kp	74.69 ± 2.28	(5)	81.69 ± 1.79	(4)	69.47 ± 4.47	(6)
Votes	92.08 ± 2.82	(4)	**94.09 ± 1.84**	(1)	89.55 ± 4.13	(6)
Alzheimer-Acetyl	62.72 ± 3.54	(6)	63.70 ± 3.40	(5)	64.04 ± 4.45	(4)
Alzheimer-Amine	75.13 ± 4.32	(5)	76.29 ± 4.14	(4)	68.05 ± 5.76	(6)
Alzheimer-Mem	67.53 ± 4.65	(3)	67.48 ± 4.32	(4)	59.65 ± 6.56	(6)
Alzheimer-Toxic	78.16 ± 4.19	(5)	80.13 ± 5.52	(3)	72.57 ± 6.67	(6)
Carcinogenics	**66.93 ± 7.51**	(1)	65.28 ± 8.09	(2)	58.26 ± 7.45	(6)
Dspg-Astrophorida	81.16 ± 4.44	(5.5)	81.16 ± 4.44	(5.5)	95.98 ± 3.13	(2)
Dspg-Axinellida	86.62 ± 5.27	(5.5)	86.62 ± 5.27	(5.5)	94.42 ± 3.12	(4)
Dspg-Dictyoceratida	98.01 ± 1.60	(6)	98.45 ± 1.91	(5)	**98.82 ± 1.36**	(1)
Dspg-Hadromerida	76.65 ± 4.49	(5.5)	76.65 ± 4.49	(5.5)	91.88 ± 4.40	(4)
Dspg-Halichondrida	92.72 ± 3.78	(5.5)	92.72 ± 3.78	(5.5)	**94.52 ± 3.54**	(1)
Dspg-Haplosclerida	91.66 ± 2.75	(5.5)	91.66 ± 2.75	(5.5)	96.61 ± 2.79	(2)
Dspg-Poecilosclerida	81.12 ± 7.36	(5.5)	81.12 ± 7.36	(5.5)	**92.38 ± 3.28**	(1)
Dsstox	71.13 ± 3.85	(2)	**71.34 ± 4.40**	(1)	68.89 ± 4.21	(5)
Metabolism	66.68 ± 10.10	(2.5)	**71.83 ± 11.77**	(1)	66.68 ± 10.10	(2.5)
Mutagenesis	66.03 ± 11.48	(6)	**78.25 ± 7.69**	(1)	72.68 ± 11.68	(3)
Pyrimidines	85.27 ± 1.23	(5)	**89.33 ± 1.68**	(1)	76.47 ± 4.19	(6)

	ProGolemCWS		ProGolemNot		ProGolemNRNot	
Krk_Illegal	99.87 ± 0.12	(3.5)	99.87 ± 0.12	(3.5)	99.90 ± 0.09	(2)
Kr_vs_kp	85.18 ± 4.84	(3)	**89.73 ± 4.33**	(1)	88.62 ± 4.58	(2)
Votes	92.00 ± 2.58	(5)	93.29 ± 2.39	(2)	92.50 ± 3.18	(3)
Alzheimer-Acetyl	73.25 ± 4.35	(3)	**76.74 ± 4.71**	(1)	75.97 ± 4.90	(2)
Alzheimer-Amine	80.53 ± 3.90	(3)	83.77 ± 3.46	(2)	**84.07 ± 3.70**	(1)
Alzheimer-Mem	63.50 ± 5.28	(5)	**68.81 ± 5.68**	(1)	68.07 ± 5.63	(2)
Alzheimer-Toxic	79.89 ± 5.36	(4)	**84.36 ± 4.03**	(1)	84.35 ± 4.20	(2)
Carcinogenics	59.20 ± 7.98	(4)	59.07 ± 7.57	(5)	60.86 ± 7.27	(3)
Dspg-Astrophorida	95.66 ± 3.34	(3)	95.52 ± 3.38	(4)	**96.51 ± 2.83**	(1)
Dspg-Axinellida	94.61 ± 3.47	(2)	94.56 ± 3.54	(3)	**95.02 ± 3.19**	(1)
Dspg-Dictyoceratida	98.71 ± 1.41	(3)	98.59 ± 1.64	(4)	98.76 ± 1.25	(2)
Dspg-Hadromerida	93.05 ± 4.26	(2)	92.85 ± 4.23	(3)	**93.08 ± 4.34**	(1)
Dspg-Halichondrida	94.42 ± 3.28	(3)	94.51 ± 3.49	(2)	93.92 ± 3.27	(4)
Dspg-Haplosclerida	96.35 ± 2.97	(4)	96.60 ± 2.85	(3)	**97.32 ± 2.93**	(1)
Dspg-Poecilosclerida	91.97 ± 3.57	(3)	91.52 ± 4.02	(4)	92.06 ± 3.62	(2)
Dsstox	69.19 ± 4.03	(3)	69.03 ± 4.33	(4)	68.51 ± 3.93	(6)
Metabolism	63.94 ± 10.37	(6)	65.61 ± 9.25	(5)	66.50 ± 8.62	(4)
Mutagenesis	71.44 ± 12.20	(4)	70.13 ± 11.62	(5)	73.94 ± 10.83	(2)
Pyrimidines	85.67 ± 2.21	(4)	88.34 ± 1.82	(2)	88.19 ± 1.79	(3)

Tabela 3.4

Časi (v sekundah), ki jih posamezni sistemi potrebujejo za izvedbo ene iteracije prečnega preverjanja.

	Aleph	AlephCWS	ProGolem
Krk_Illegal	158.05 ± 15.62	147.23 ± 14.66	**70.17** ± **5.13**
Kr_vs_kp	744.67 ± 31.09	703.78 ± 17.53	**34.78** ± **1.30**
Votes	6.01 ± 0.40	7.08 ± 0.77	**4.76** ± **0.20**
Alzheimer-Acetyl	1432.50 ± 498.38	1148.00 ± 305.54	**23.41** ± **1.23**
Alzheimer-Amine	286.11 ± 76.99	198.33 ± 54.47	**8.33** ± **0.42**
Alzheimer-Mem	374.86 ± 93.41	282.22 ± 70.51	**19.96** ± **0.71**
Alzheimer-Toxic	411.96 ± 100.00	316.76 ± 84.20	**22.35** ± **0.78**
Carcinogenics	1493.93 ± 75.37	1113.12 ± 12.36	**134.52** ± **13.59**
Dspg-Astrophorida	507.28 ± 27.11	480.38 ± 9.60	192.49 ± 16.98
Dspg-Axinellida	458.22 ± 8.61	444.72 ± 7.05	260.50 ± 8.21
Dspg-Dictyoceratida	**1.95** ± **0.08**	4.09 ± 0.11	77.7 ± 5.51
Dspg-Hadromerida	586.02 ± 14.07	538.07 ± 13.46	**353.37** ± **16.82**
Dspg-Halichondrida	305.40 ± 9.01	**294.75** ± **7.55**	317.28 ± 10.54
Dspg-Haplosclerida	**122.39** ± **4.02**	118.85 ± 4.22	182.77 ± 10.06
Dspg-Poecilosclerida	315.30 ± 60.00	**288.47** ± **5.82**	385.24 ± 16.55
Dsstox	810.74 ± 482.76	667.51 ± 182.54	**179.19** ± **10.64**
Metabolism	**60.05** ± **7.61**	61.35 ± 3.48	86.79 ± 3.24
Mutagenesis	**1.71** ± **0.10**	33.78 ± 0.30	683.08 ± 98.22
Pyrimidines	353.70 ± 86.25	340.16 ± 78.65	**37.30** ± **1.63**
	ProGolemCWS	ProGolemNot	ProGolemNRNot
Krk_Illegal	94.63 ± 10.62	140.55 ± 14.94	108.93 ± 7.61
Kr_vs_kp	59.00 ± 1.28	455.78 ± 46.34	219.45 ± 14.76
Votes	8.54 ± 0.31	40.72 ± 2.90	29.82 ± 2.84
Alzheimer-Acetyl	57.56 ± 2.84	888.60 ± 71.43	430.62 ± 21.30
Alzheimer-Amine	28.05 ± 1.27	494.52 ± 30.22	333.80 ± 21.81
Alzheimer-Mem	57.53 ± 3.61	728.39 ± 44.11	360.05 ± 19.64
Alzheimer-Toxic	53.28 ± 3.22	580.98 ± 28.74	403.71 ± 26.52
Carcinogenics	396.45 ± 12.97	7070.78 ± 474.59	1706.80 ± 121.83
Dspg-Astrophorida	202.65 ± 10.03	431.82 ± 51.80	**179.76** ± **8.48**
Dspg-Axinellida	262.85 ± 12.38	3162.56 ± 348.05	**248.00** ± **6.66**
Dspg-Dictyoceratida	75.54 ± 3.76	80.80 ± 7.75	78.37 ± 2.93
Dspg-Hadromerida	370.41 ± 12.53	3245.45 ± 180.44	359.74 ± 17.33
Dspg-Halichondrida	322.08 ± 13.59	4300.10 ± 292.09	313.23 ± 5.53
Dspg-Haplosclerida	185.86 ± 8.26	996.49 ± 82.87	157.48 ± 4.60
Dspg-Poecilosclerida	407.12 ± 12.76	3770.82 ± 261.08	366.78 ± 7.64
Dsstox	365.35 ± 22.73	3693.43 ± 474.59	308.04 ± 23.13
Metabolism	367.81 ± 15.30	16063.90 ± 968.93	4604.99 ± 201.52
Mutagenesis	1151.85 ± 141.12	18722.18 ± 2897.87	4588.43 ± 1123.16
Pyrimidines	104.98 ± 6.02	974.47 ± 50.99	509.42 ± 23.98

Tabela 3.5

Z Holmovo korekcijo popravljene p-vrednosti, dobljene z uporabo Welchevega t-testa pri primerjanju doseženih točnosti sistema ProGolemNot z ostalimi sistemi. Oznake ob p-vrednostih kažejo, ali ima sistem ProGolemNot na tej domeni signifikantno višjo (+), nižjo (−) ali nesignifikantno različno (=) točnost. Vrednosti so zaokrožene na dve decimalni mesti oz. na eno značilno mesto pri vrednostih, manjših od 0.01.

	Aleph	AlephCWS	ProGolem	ProGolemCWS	ProGolemNRNot
Vlaki00	4e−14(+)	4e−14(+)	8e−15(+)	6e−17(+)	8e−08(+)
Vlaki05	7e−12(+)	7e−12(+)	3e−10(+)	2e−08(+)	1e−04(+)
Vlaki10	8e−06(+)	1e−03(+)	2e−05(+)	8e−04(+)	0.18(=)
Vlaki15	3e−10(+)	2e−07(+)	2e−03(+)	3e−03(+)	0.97(=)
Vlaki20	1.00(=)	0.23(=)	1.00(=)	1.00(=)	1.00(=)
Vlaki25	0.03(+)	0.94(=)	0.94(=)	0.71(=)	0.77(=)
Vlaki-VecStavkov	7e−06(+)	2e−07(+)	1e−10(+)	1e−10(+)	0.12(=)
Vlaki-VecNivojev	5e−35(+)	2e−24(+)	5e−35(+)	5e−35(+)	3e−09(+)
Krk_Illegal	2e−33(+)	4e−17(−)	3e−03(+)	0.91(=)	0.12(=)
Kr_vs_kp	5e−66(+)	6e−35(+)	1e−80(+)	8e−11(+)	0.08(=)
Votes	4e−03(+)	0.02(−)	3e−12(+)	1e−03(+)	0.05(+)
Alzheimer-Acetyl	3e−57(+)	2e−53(+)	1e−47(+)	3e−07(+)	0.26(=)
Alzheimer-Amine	3e−35(+)	3e−30(+)	4e−53(+)	5e−09(+)	0.55(=)
Alzheimer-Mem	0.19(=)	0.19(=)	3e−20(+)	4e−10(+)	0.36(=)
Alzheimer-Toxic	1e−20(+)	8e−09(+)	4e−32(+)	9e−10(+)	0.99(=)
Carcinogenesis	9e−09(−)	5e−05(−)	0.07(=)	0.40(=)	0.75(=)
Dspg-Astrophorida	3e−62(+)	3e−62(+)	0.63(=)	0.77(=)	0.08(=)
Dspg-Axinellida	3e−25(+)	3e−25(+)	1.00(=)	1.00(=)	1.00(=)
Dspg-Dictyoceratida	0.06(=)	1.00(=)	1.00(=)	1.00(=)	1.00(=)
Dspg-Hadromerida	1e−65(+)	1e−65(+)	0.34(=)	1.00(=)	1.00(=)
Dspg-Halichondrida	3e−03(+)	3e−03(+)	1.00(=)	1.00(=)	0.65(=)
Dspg-Haplosclerida	4e−26(+)	4e−26(+)	1.00(=)	1.00(=)	0.24(=)
Dspg-Poecilosclerida	3e−24(+)	3e−24(+)	0.30(=)	0.64(=)	0.64(=)
Dsstox	2e−03(−)	2e−03(−)	1.00(=)	1.00(=)	1.00(=)
Metabolism	1.00(=)	2e−04(−)	1.00(=)	0.92(=)	1.00(=)
Mutagenesis	5e−04(+)	5e−05(−)	1.00(=)	1.00(=)	0.79(=)
Pyrimidines	1e−09(+)	2e−05(−)	1e−25(+)	2e−05(+)	0.12(=)

Tabela 3.6

Povprečni rangi testiranih sistemov.

	Aleph	AlephCWS	ProGolem
Umetne domene	5.00	4.25	4.63
Realne domene	4.71	3.47	4.03
Skupno	4.77	3.79	4.20
	ProGolemCWS	ProGolemNot	ProGolemNRNot
Umetne domene	3.25	**1.75**	2.13
Realne domene	3.55	2.92	**2.32**
Skupno	3.46	2.57	**2.26**

Tabela 3.7

Rezultati Nemenyijevega post-hoc testa. Primerjamo vse sisteme preko vseh podatkovnih množic hkrati.

	Aleph	AlephCWS	ProGolem	ProGolemCWS	ProGolemNot
Aleph+CWS	0.26373	-	-	- -	
ProGolem	0.85406	0.92382	-	-	-
ProGolem+CWS	0.09274	0.99708	0.69318	-	-
ProGolemNot	**0.00019**	0.22905	**0.01725**	0.50131	-
ProGolemNRNot	**0.00001**	0.05184	**0.00186**	0.16902	0.98974

4

Dvojna nasičenja

V tem poglavju opišemo koncept dvojnih nasičenj (ang. pairwise saturations) [14]. Dvojna nasičenja temeljijo na analizi spodnjih stavkov primerov, katerih posplošitev predstavlja spodnjo mejo prostora preiskovanja pri sistemih, ki rešitev danega problema iščejo od spodaj navzgor. V prvih dveh podpoglavjih predstavimo problem in sorodno delo. Sledi analiza omejitev, katerim morajo zadoščati literali v asimetrični relativni minimalni posplošitvi, če naj ta pokrije primera, na podlagi katerih je izračunana. Sam postopek izračuna je predstavljen v Podpoglavju 4.3, v Podpoglavju 4.4 pa je postopek posplošen s parov primerov (dvojna nasičenja) na poljubne n-terke primerov (n-terna nasičenja). Sledi analiza časovne zahtevnosti opisanega pristopa in eksperimentalni del, kjer sistem z dvojnimi in n-ternimi nasičenji (ProParGolem) primerjamo z osnovnim sistemom ProGolem.

4.1 Izračun pokritja hipotez

Kot ključno težavo pri uporabi sistemov ILP smo izpostavili njihovo časovno zahtevnost. Velik del časa sistemi ILP prebijejo v testiranju pokritja hipotez bodisi z uporabo resolucije bodisi s preverjanjem Θ-vsebovanosti hipoteze v spodnjih stavkih primerov, kar je znan NP-poln problem [23].

Pohitritev sistema ILP je v tem delu mogoče doseči na dva načina

1. Uporaba učinkovitih pristopov za izračun pokritja.
2. Manjšanje števila testiranj, ki se izvedejo.

Večina sistemov ILP za ugotavljanje pokritja privzeto uporablja Prologovo SLD-resolucijo z leve proti desni in najprej v globino. Uporaba SLD-resolucije je pri nedeterminiranem predznanju neučinkovita, zaradi česar se v takšnih primerih obrestuje uporaba pristopov, ki uporabljajo izračun Θ-vsebovanosti, ali pristopov, ki hevristično določajo naslednji korak SLD-resolucije [51].

SLD-resolucija (selektivna linearna definitna resolucija, ang. Selective Linear Definite Resolution) je standardni način izvajanja resolucije v Prologu.

Definicija 27 (Korak SLD-resolucije):Korak SLD-resolucije cilja `:- B1 ... Bk` *in stavka* `A :- A1 ... An` *se izračuna kot:*

1. *Združi* `B1` *in* `A` *z najbolj splošno zamenjavo* Θ*, za katero* $\mathtt{A}\Theta = \mathtt{B1}$.

2. *Nadomesti* `B1` *z* `A1 ... An`.

3. *Novi cilj je* `:- (A1 ... An, B2 ... Bk)`Θ.

Veljavnost posameznega cilja z uporabo SLD-resolucije preverimo tako, da jo apliciramo nanj, dokler ta ne postane prazen (kadar je cilj resničen) oz. izčrpamo vse možne poti, ki jih SLD-resolucija lahko ubere (kadar je cilj neresničen). Za Hornove stavke je SLD-resolucija zdrava (uspe le v primerih, ki držijo; ang. sound) in zavrnitveno popolna (zavrne vse primere, ki ne držijo; ang. refutation complete).

Med hevristike za pohitritev SLD-resolucije spadata izbor predikata z najmanjšo domeno (ang. smallest predicate domain) in izbor spremenljivke z najmanjšo domeno (ang. smallest variable domain). Pri izboru predikata z najmanjšo domeno se kot naslednji literal pri izvajanju SLD-resolucije izbere tisti z najmanj možnimi rešitvami. Pri uporabi hevristike spremenljivke z najmanjšo domeno se na vsakem koraku dodeli vrednost spremenljivki z najmanj različnimi veljavnimi vezavami. Empirično testiranje v [51] pokaže, da pohitritve zaradi uporabe hevristik zelo nihajo med različnimi podatkovnimi množicami. V nekaterih primerih uporaba hevristik privede tudi do desetkratne pohitritve, v drugih domenah pa lahko pride do upočasnitve (v nekaterih primerih tudi štirikratne). Dosežena pohitritev je odvisna od nedeterminiranosti domene. V determiniranih domenah se namreč izračun hevristike odraža zgolj kot dodatno delo, saj imajo vsi predikati ob podanem naboru vhodnih argumentov natanko eno instanciacijo svojih izhodnih spremenljivk.

Drugi pristop k pohitritvi SLD-resolucijo nadomesti z izračunom Θ-vsebovanosti hipoteze v spodnjem stavku primera, za katerega ugotavljamo, ali je s to hipotezo pokrit. Ta izračun je ekvivalenten SLD-resoluciji v primeru, ko je predznanje podano kot čisti Prologov program, v splošnem (tj. ob prisotnosti aritmetičnih operatorjev, rezov ipd.) pa ne [51]. Med pristope k izračunu Θ-vsebovanosti stavkov spadajo Django [34], Resumer2 [30] in Subsumer [50]. Vsi trije sistemi osnovo za svoje delovanje črpajo iz problema zadovoljevanja omejitev (ang. constraint satisfaction problem).

Blockeel in sodelavci [6] predlagajo uporabo skupinskih poizvedb (ang. query packs) za pohitritev sistemov ILP. Skupinske poizvedbe izkoriščajo dejstvo, da so si kandidatne hipoteze med iskanjem rešitve navadno podobne in se iz tega razloga izvajajo redundantni izračuni njihovih delov. Takšna situacija je najočitnejša pri sistemih ILP, ki hipotezo gradijo z dodajanjem novih literalov – kandidatne hipoteze se med seboj namreč razlikujejo zgolj v zadnjem literalu, pri izračunu pokritja pa se vezava spre-

menljivk, ki zadosti prvemu delu, računa vedno znova za vsako kandidatno hipotezo. Avtorji predlagajo ustreznico matematičnemu izpostavljanju skupnega faktorja – skupnemu delu hipoteze (tj. hipotezi, ki jo razširjamo) se pripišejo disjunktno vezane možne razširitve. S takšnim zapisom hipoteze odpadejo redundantni izračuni vezave, ki zadosti skupnemu delu hipoteze.

Do sedaj omenjeni pristopi vsi poskušajo pohitriti sam proces odločanja, ali je en izmed primerov pokrit s strani dane hipoteze ali ne. Druga možnost, s katero bi zmanjšali čas izvajanja sistemov ILP, pa je manjšanje števila hipotez, ki jih sistem ovrednoti. V sistemih, ki delujejo od spodaj navzgor, je število testiranj pokritja neposredno odvisno od števila literalov v spodnjem stavku – pri izvajanju negativne redukcije se tako preverja pokritje hipoteze po odstranitvi vsakega literala. Z zmanjšanjem literalov v spodnjem stavku bi se torej pokritje računalo manjkrat in posledično bi sistem prej prišel do rešitve danega problema.

Takšen pristop uporabljajo Kuželka s sodelavci [29]. V svojem delu uvedejo pojem *varne redukcije* primera, katerega rezultat je njegov "okrajšani zapis", ki pa je ob upoštevanju pristranskosti jezika hipotez od osnovnega primera neločljiv. Gre za posebno varianto Plotkinove Θ-redukcije [44], ki pa pri svojem izračunu upošteva pristranskost jezika. Redukcijo primerov avtorji izvedejo z uporabo jezika hipotez z omejeno drevesno širino stavka, kar omogoča izvedbo redukcije v polinomskem času – v nasprotju z uporabo Θ-redukcije, ki je v splošnem NP-poln problem. Takšen pristop so avtorji uporabili na sistemih Aleph [53] in nFOIL [31].

4.2 Analiza spodnjih stavkov pri vodenju indukcije

V nadaljevanju predstavimo pristop, soroden varnim redukcijam [29], ki pa deluje nad pari primerov in s pomočjo analize njunih spodnjih stavkov iz prostora preiskovanja izloča literale, ki ne morejo biti del hipoteze, ki bi pokrila oba primera, uporabljena pri redukciji.

Analiza spodnjih stavkov primerov se je v ILP že večkrat pokazala kot koristna pri naslavljanju različnih problemov. Znani so denimo pristopi relacijskega iskanja poti (ang. relational pathfinding), s katerim sta Richards in Mooney [49] premagovala problem planote in lokalnih maksimumov ocene pri sistemih, ki delujejo od zgoraj navzdol. Ong in sodelavci [43] so pristop prilagodili, da izkorišča dodatno znanje, prisotno v deklaracijah načina. Podoben pristop se uporablja tudi v področju rudarjenja relacijskih preopisov (ang. relational redescription mining) [17]. Analiza spodnjih

stavkov je tudi orodje, na katerem temeljijo nekateri pristopi za odkrivanje rekurzivno definiranih predikatov. Navadno ti pristopi temeljijo na odkrivanju struktur v obliki poti [20]. Takšen pristop denimo uporablja sistem MRI [16].

Idejo hkratne uporabe več spodnjih stavkov primerov pri iskanju rešitve je mogoče zaslediti predvsem pri sistemih za iskanje relacijskih povezovalnih pravil, denimo WARMR [10] in njegovih izpeljank FARMER [42] in PIX [22].

V svojem delu izhajamo iz ProGolemovega posplošitvenega operatorja, asimetrične relativne minimalno generalizacije dveh primerov $H = ARMG(e_2|e_1)$. Po definiciji je $ARMG(e_2|e_1)$ podmnožica variabiliziranega spodnjega stavka $\bot_1$ primera e_1 (tj. $ARMG(e_2|e_1) \subseteq var(\bot_1)$). Poleg tega za $ARMG(e_2|e_1)$ obstaja takšna zamenjava Θ_2, da je $glava(H\Theta_2) = e_2$ in $telo(H\Theta_2) \subseteq \bot_2$. Iz teh dejstev lahko izpeljemo določene omejitve glede tega, kateri literali iz $var(\bot_1)$ se lahko pojavijo kot del H.

Osnova za te omejitve izhaja iz sledečega premisleka. Če naj H pokrije primera e_1 in e_2, morata obstajati zamenjavi Θ_1 ter Θ_2, za kateri je $glava(H\Theta_1) = e_1$ in $glava(H\Theta_2) = e_2$, ter $telo(H\Theta_1) \subseteq \bot_1$ in $telo(H\Theta_2) \subseteq \bot_2$. Posledično za vsak literal L v telesu hipoteze H velja, da $L\Theta_1 \in \bot_1$ in $L\Theta_2 \in \bot_2$. Za določene literale L iz $var(\bot_1)$ *ne more* obstajati takšna hipoteza H, $L \in H$, da bi zanjo obstajala takšna zamenjava Θ_2, za katero $glava(H\Theta_2) = e_2$ in $telo(H\Theta_2) \subseteq \bot_2$. V namen odkrivanja takšnih literalov uvedemo omejitve, ki morajo držati za vsak literal, če naj bo ta del posplošitve podanih primerov.

4.2.1 Omejitve nad konstantami

Naj bo v predznanju definiran predikat P z deklaracijo načina enako $P(... \#A_k ...)$ – na njegovem k-tem mestu mora stati konstanta. Naj bo literal L_p del spodnjega stavka primera e_1 ($L_p \in \bot_1$) s predikatnim simbolom P. Če naj neka z glavo povezana hipoteza $H \subseteq var(\bot_1)$, ki vsebuje $var(L_p) = P(... c_k ...)$ (c_k je konstanta), pokrije primer e_2, mora obstajati zamenjava Θ_2, da je $H\Theta_2$ podmnožica $\bot_2$. Ker apliciranje zamenjave nima vpliva na vrednosti konstant, sledi, da se mora ob prisotnosti $P(... c_k ...)$ v H nahajati v $\bot_2$ nek literal oblike $P(... c_k ...)$. V nasprotnem primeru namreč nobena Θ_2 ne more ustvariti takšne zamenjave, da bi $P(... c_k ...)$ uspel.

Iz tega razloga lahko pri izračunu $ARMG(e_2|e_1)$ iz $\bot_1$ odstranimo tiste literale $P(... c_k ...)$, za katere v $\bot_2$ ni mogoče najti literala oblike $P(... c_k ...)$. Primer literalov, ki jih lahko varno odstranimo, je prikazan v primeru 18.

Primer 18: Imejmo par primerov $e_1 = partnerja(m1, z1)$, $e_2 = partnerja(m2, z2)$.

Deklaracije načina naj bodo enake:

```
partnerja(+oseba, +oseba).
poklic(+oseba, #poklic).
osebnost(+oseba, #tip).
```

Posplošena spodnja stavka pa:

```
partnerja(A, B) :- poklic(A, poslovnez), poklic(B, student),
       osebnost(A, introvert), osebnost(B, ekstravert).

partnerja(C, D) :- poklic(C, poslovnez), poklic(D, zdravnik),
       osebnost(C, ekstravert), osebnost(D, ekstravert).
```

Literala `poklic(B, student)` in `poklic(A, introvert)` se ne moreta pojaviti v nobeni hipotezi, ki bi pokrila oba navedena primera, saj se konstanti `student` in `introvert` v $\bot_2$ ne pojavita kot druga argumenta literalov s predikatnim simbolom `poklic` oziroma `osebnost`. △

Poudariti je potrebno, da zgornja argumentacija v prisotnosti predikatov, ki uporabljajo logično programiranje z omejitvami (ang. constraint logic programming, CLP) – denimo pri predikatih, ki v svojih definicijah uporabljajo primerjalna operatorja $<$ in $>$, ne drži. V njihovi prisotnosti za uspeh nekega literala ni nujno, da je konstantni operator identičen. Za demonstracijo si poglejmo primer 19.

Primer 19: Podane naj bodo deklaracije načina:

```
ekstrem(+vzorec).
vrednost(+vzorec, -vrednost).
manjsi_ali_enak(+vrednost, #celostevilo).
```

Dana naj bosta tudi dva primera s posplošenima nasičenjima:

```
ekstrem(A) :- vrednost(A, B), manjsi_ali_enak(B, 50).
ekstrem(C) :- vrednost(C, D), manjsi_ali_enak(D, 45).
```

Opazimo lahko, da se konstanti na drugem mestu literalov s predikatnim simbolom `manjsi_ali_enak` ne ujemata, vendar bi literal `manjsi_ali_enak(B, 50)` uspel tudi v drugem primeru (drži namreč, da je vrednost `D` iz drugega primera manjša od 45). △

Iz primera je razvidno, da je predikate, definirane z uporabo logičnega programiranja z omejitvami, treba predhodno identificirati in jih posebej označiti, saj zanje omejitve nad konstantami v splošnem ne držijo. Če jih namreč odstranimo iz posplošitve spodnjega stavka $\perp_1$, s tem izgubimo del prostora hipotez, v katerem se lahko nahaja rešitev.

4.2.2 Omejitve nad vhodnimi spremenljivkami

Za definicijo ostalih omejitev nad literali v posplošitvi dveh primerov si oglejmo, kako se instancirajo vhodni argumenti posameznih literalov. Vhodni argumenti literalov lahko dobijo vrednost bodisi kot vhodni argumenti glave hipoteze bodisi kot izhodni argumenti predhodnih literalov oz. literalov na nižjih nivojih.

Omejitve nad vhodnimi spremenljivkami izhajajo iz dejstva, da bo rezultat posplošitve pokril oba primera, podana na vhodu operatorja posplošitve. Ker je rezultat posplošitve povezan z glavo, za vsak literal v njem velja, da se vse spremenljivke na njegovih vhodnih mestih pojavijo bodisi v prejšnjih literalih kot izhodi bodisi kot vhodi v glavi. Hipoteza, ki je rezultat posplošitve, s svojo razporeditvijo spremenljivk določa, na kakšen način se spremenljivke instancirajo. Če naj rezultat posplošitve pokrije nek primer (po definiciji ga mora), mora spodnji stavek tega primera instanciacijo iz hipoteze "podpirati" v smislu, da vhodna mesta vsakega literala v njem lahko (načinov, kako se lahko vrednost spremenljivke instancira, je namreč lahko znotraj enega stavka več) dobijo vrednosti na enak način kot v hipotezi. Ker pa po definiciji hipoteza pokrije oba primera, lahko pri izračunu $ARMG(e_j|e_i)$ iz $\perp_i$ varno odstranimo tiste literale, katerih vhodne spremenljivke v $\perp_j$ ni mogoče instancirati na enak način kot v $\perp_i$.

Da bi preverili, če se vhodne spremenljivke nekega literala instancirajo na enak način, definiramo *množico instanciacij* vrednosti.

Definicija 28 (Množica instanciacij vrednosti (ang. instantiation set)):Množica instanciacij vrednosti v v stavku C je možica vseh njenih pojavitev: tako na vhodnih mestih glave stavka, kot tudi na izhodnih mestih literalov. Vsak element v množici se hrani kot terka $\langle P, k, n\rangle$, kjer je P predikatni simbol literala, v katerem se vrednost pojavi, k ustreza mestu vrednosti (tj. kot kateri argument nastopa ta vrednost), n pa zaznamuje globino, na kateri se nahaja literal.

Algoritem, ki zgradi množico instanciacij danemu spodnjem stavku za neko vre-

dnost, je podan kot algoritem 13.

Primer množice instanciacij podaja primer 20.

Primer 20: Za spodnji stavek primera e_1

```
vzhodni(A) :-
    pred(A, B), dolg(B), odprt(B), oblika(B, pravokoten),
    tovor(B, pravokotnik, 3), pred(B, C), zaprt(C),
    kratek(C), oblika(C, pravokoten), tovor(C, trikotnik, 1).
```

z deklaracijami načina

```
vzhodni(+), dolg(+), odprt(+), kratek(+), oblika(+, #),
pred(+, -), tovor(+, #, #)
```

so množice instanciacij enake:

$$MI(A) = \{\langle vzhodni/1/0\rangle\}$$
$$MI(B) = \{\langle pred/2/1\rangle\}$$
$$MI(C) = \{\langle pred/2/2\rangle\}$$

Množice instanciacij nam omogočajo, da za istoležni vhodni mesti literalov L in L' določimo, ali se lahko instancirata na enak način. To storimo z izračunom preseka množic instanciacij vrednosti na istoležnih vhodnih mestih. Če je njun presek neprazen, obstaja način instanciacije te vrednosti, ki uspe za oba primera. Primer 21 podaja primer, kako upoštevanje omejitev nad vhodnimi spremenljivkami literalov vpliva na spodnji stavek $\bot_i$ pri izračunu $ARMG(e_j|e_i)$.

Primer 21: Naj bodo podani spodnji stavek, deklaracije načina in množice instanciacij kot v primeru 20. Naj bo podan še drug primer e_2 s spodnjim stavkom

```
vzhodni(A') :-
    pred(A', B'), kratek(B'), pred(B', C'), dolg(C').
```

Množice instanciacij zanj so enake:

$$MI(A') = \{\langle vzhodni/1/0\rangle\}$$
$$MI(B') = \{\langle pred/2/1\rangle\}$$
$$MI(C') = \{\langle pred/2/2\rangle\}$$

Vhod : Stavek B, deklaracije načina M, vrednost V
Izhod : Množica instanciacij MI_V vrednosti V

Function mnozica_instanciacij(B, V , M) is
    $MI_V \leftarrow \{\}$;
    $H \leftarrow glava(B)$;
    $HP \leftarrow predikatni\_simbol(H)$;
    $M_{HP} \leftarrow M[HP]$;
    $HA \leftarrow argumenti(H)$
    for $i \in nastej(HA)$ *do*
        if $HA[i] == V \wedge M_{HP}[i]$ *označuje vhod then*
            $IS_V.dodaj(\langle HP, i, 0\rangle)$;
        end
    end
    for $L \in telo(B)$ *do*
        $P \leftarrow predikatni\_simbol(L)$;
        $M_P \leftarrow M[P]$;
        $A \leftarrow argumenti(L)$;
        for $i \in nastej(A)$ *do*
            if $A[i] == V \wedge M_P[i]$ *označuje izhod then*
                $IS_V.add(\langle HP, i, nivo(L)\rangle)$;
            end
        end
    end
    return IS_V;
end

Algoritem 13: Izgradnja množice instanciacij.

Pri izračunu $ARMG(e_2|e_1)$ lahko najprej iz $\perp_1$ odstranimo vse literale s predikatnimi simboli, ki se v $\perp_2$ ne nahajajo. Tako ostanejo zgolj literali, katerih predikatni simbol je eden izmed `pred`, `dolg` ter `kratek`. $\perp_1$ je po tem koraku enak

```
vzhodni(A) :-
    pred(A, B), dolg(B), pred(B, C), kratek(C).
```

Za vsakega izmed literalov preverimo še veljavnost omejitev nad vhodnimi spremenljivkami. Literal `pred(A, B)` ima vhodno spremenljivko `A`. $\perp_2$ vsebuje dva potencialno ustrezna literala: `pred(A', B')` ter `pred(B', C')`. Presek $MI(A) \cap MI(A')$ je neprazen, zato `pred(A, B)` ostaja v nasičenju. Literal `dolg(B)` ima eno samo ustreznico, `dolg(C')`, presek $MI(B) \cap MI(C')$ pa je prazen, zaradi česar `dolg(B)` ne more biti del $ARMG(e_2|e_1)$. Za literal `pred(B, C)` v $\perp_2$ najdemo ustreznico `pred(B', C')`, pri literalu `dolg(C)` pa ponovno naletimo na prazen presek množic instanciacij $MI(C) \cap MI(B')$. Po upoštevanju omejitev nad vhodi tako dobimo:

```
vzhodni(A) :-
    pred(A, B), pred(B, C).
```

△

4.3 Krajšanje spodnjih stavkov

S pomočjo prej definiranih omejitev in množice instanciacij definiramo pojem *kompatibilnosti literalov*.

Definicija 29 (Kompatibilna literala (ang. compatible literals)):
Literala $L = P(A_1, \ldots A_n$ *in* $L' = P'(B_1, \ldots B_m)$ *sta kompatibilna, kadar:*

1. *Imata enako številnost in predikatni simbol* ($P = P' \wedge m = n$).
2. *Se ujemata na konstantnih mestih.*
3. *Za vsako vhodno mesto* i *velja:* $MI[A_i] \cap MI[B_i] \neq \emptyset$, *kjer* $MI[X]$ *označuje množico instanciacij vrednosti* X.

Algoritem za preverjanje kompatibilnosti literalov L_1 in L_2 ob podanih množicah instanciacij je prikazan kot algoritem 14.

Vhod : Literala L_1, L_2, množici instanciacij $MI[e_1]$, $MI[e_2]$, deklaracije načina M.

Izhod : Boolova vrednost, ali sta L_1 in L_2 kompatibilna.

Function kompatibilna(L_1, L_2, $MI[e_1]$, $MI[e_2]$, M) *is*
    $P_1 \leftarrow$ *predikatni_simbol*(L_1);
    $P_2 \leftarrow$ *predikatni_simbol*(L_2);
    $arg_1 \leftarrow$ *argumenti*(L_1);
    $arg_2 \leftarrow$ *argumenti*(L_2);
    if $P_1 \neq P_2 \vee$ *dolzina*($args_1$) $\neq$ *dolzina*($args_2$) *then*
        return false;
    end
    nacin $\leftarrow M[P_1]$;
    for $i = 1 \ldots$ *dolzina*(arg_1) *do*
        if *nacin*$[i] = \#$ *then*
            if P_1 *je predikat CLP then*
                naslednja vrednost i;
            end
            else if $arg_1[i] \neq arg_2[i]$ *then*
                return false;
            end
        end
        else if *nacin*$[i] = +$ *then*
            if $MI[e_1][arg_1[i]] \cap MI[e_2][arg_2[i]] = \emptyset$ *then*
                return false;
            end
        end
    end
    return true;
end

Algoritem 14: Algoritem za preverjanje kompatibilnosti literalov.

Izrek 1: Če nek literal $L \in \perp_1$ nima kompatibilnega literala v $\perp_2$, se njegova generalizacija ne more pojaviti v $ARMG(e_2|e_1)$.

Dokaz 1: Literal L nima kompatibilnega literala v $\perp_2$, če za vsak $L' \in \perp_2$ vsaj en izmed pogojev iz definicije 29 ne drži. Trditev preverimo za vsakega izmed njih.

Pogoj 1: Naj bo $L = P(A_1, \ldots A_n)$ in naj za vsak literal $L' = P'(B_1, \ldots B_n) \in \perp_2$ drži bodisi $P' \neq P$ bodisi $m \neq n$. V takšnem primeru $var(L)$ ne more biti del posplošitve $H = ARMG(e_2|e_1)$. H mora namreč (po definiciji) pokrivati e_2 – obstajati mora torej zamenjava Θ_2, za katero $H\Theta_2 \subseteq \perp_2$. Ker zamenjave ne vplivajo na predikatne simbole in številnost literalov, takšna zamenjava Θ_2, da bi $L\Theta_2$ bil element $\perp_2$ ne more obstajati.

Pogoj 2: Naj bo $L = P(\ldots c_1, \ldots c_{n\#})$ z deklaracijo načina $P(\ldots \#_1, \ldots \#_{n\#})$ del spodnjega stavka $\perp_1$. Brez izgube splošnosti naj v $\perp_2$ obstaja en sam literal z enako številnostjo in enakim predikatnim simbolom kot L, $L' = P(\ldots c'_1, \ldots c'_{n\#}) \in \perp_2$. Ker zamenjave ne vplivajo na konstante, ne more obstajati nobena zamenjava Θ_2, za katero bi $H\Theta_2 \subseteq \perp_2$, če je $var(L) \in H$ in $c_i \neq c'_i$ za nek i.

Pogoj 3: Če naj bo presek množic instanciacij $MI[a]$ ter $MI[a']$ prazen, se mora vsak par instantiacij $\langle P/i/l\rangle$ in $\langle P'/i'/l'\rangle$ razlikovati vsaj v eni vrednosti (predikatnem simbolu (možnost 1), mestu (možnost 2) ali nivoju (možnost 3)). V nadaljevanju brez izgube splošnosti predpostavimo, da $MI[a]$ in $MI[a']$ vsebujeta zgolj en element – $MI[a] = \{\langle P/i/l\rangle\}$ in $MI[a'] = \{\langle P'/i'/l'\rangle\}$.

Naj se v $\perp_1$ nahaja literal $L = T(a_1, \ldots a_{n+}, b_1, \ldots b_{n-})$, kjer a_i zaznamujejo vhodna mesta, b_i pa izhodna. Prav tako naj bo $L' = T(a'_1, \ldots a'_{n+}, b'_1, \ldots b'_{n-})$ literal v $\perp_2$. Ponovno brez izgube splošnosti predpostavimo, da je L' edini literal v $\perp_2$, ki se z L ujema v pogojih 1 in 2.

Možnost 1: $\mathbf{P \neq P'}$ V tem primeru imamo opravka s situacijo, v kateri je $MI[a] = \{\langle P/i/l\rangle\}$ in $MI[a'] = \{\langle P'/i/l\rangle\}$ in $P \neq P'$. Po konstrukciji množic instanciacij vsak vnos v njej ustreza bodisi vhodni argument glave bodisi izhodni

argument literala v telesu. Spodnja stavka primerov sta tako oblike:

$$\begin{aligned}\bot_1 &= \ldots P(\ldots a_i \ldots), T(a_1, \ldots a_i \ldots a_{n+}, b_1, \ldots b_{n-}) \ldots\\ \bot_2 &= \ldots P'(\ldots a'_i \ldots), T(a'_1, \ldots a'_i \ldots a'_{n+}, b'_1, \ldots b'_{n-}) \ldots\\ var(\bot_1) &= \ldots P(\ldots A_i \ldots), T(A_1, \ldots A_i \ldots A_{n+}, B_1, \ldots B_{n-}) \ldots\end{aligned}$$

Če naj se $L_v = T(A_1, \ldots A_i \ldots A_{n+}, B_1, \ldots B_{n-})$ pojavi v $H = ARMG(e_2|e_1)$, se mora tudi $P(\ldots A_i \ldots)$, saj je edini literal, ki instancira vrednost A_i. Če se namreč ne pojavi v njem, hipoteza ni več povezana z glavo. Iz tega sledi, da mora zamenjava, za katero velja $H\Theta \subseteq \bot_2$, vsebovati A_i/a'_i, da prilagodi L_v literalu L'. Prisotnost takšne zamenjave pa vpliva tudi na literal $P(\ldots A_i \ldots)$, ki ob aplikaciji zamenjave postane $P(\ldots a'_i \ldots)$. Vendar takšen literal ni prisoten v $\bot_2$, saj bi ob njegovi prisotnosti množica instanciacij $MI[a']$ vsebovala element $\langle P/i/l' \rangle$. Odsotnost tega elementa v množici instanciacij nas privede do kontradikcije – predikatna simbola se morata ujemati.

Možnost 2: $\mathbf{i \neq i'}$ Podani naj bosta vrednosti a in a' s pripadajočima množicama instanciacij $MI[a] = \{\langle P/i/l \rangle\}$ in $MI[a'] = \{\langle P/i'/l \rangle\}$, kjer $i \neq i'$. Naj se a_i in $a'_{i'}$ pojavita kot istoležna vhodna argumenta v literalih L in L'. Brez izgube splošnosti predpostavimo $i < i'$.

Spodnja stavka danih primerov sta tako oblike:

$$\begin{aligned}\bot_1 &= \ldots P(\ldots a_i \ldots a_{i'} \ldots), T(a_1, \ldots a_i \ldots a_{n+}, b_1, \ldots b_{n-}) \ldots\\ \bot_2 &= \ldots P'(\ldots a'_i \ldots a'_{i'} \ldots), T(a'_1, \ldots a'_i \ldots a'_{n+}, b'_1, \ldots b'_{n-}) \ldots\\ var(\bot_1) &= \ldots P(\ldots A_i \ldots A_{i'} \ldots), T(A_1, \ldots A_i \ldots A_{n+}, B_1, \ldots B_{n-}) \ldots\end{aligned}$$

Sklepamo na podoben način, kot v prejšnjem primeru. Če se literal $L_v = T(A_1, \ldots A_i \ldots A_{n+}, B_1, \ldots B_{n-})$ pojavi v hipotezi $H = ARMG(e_2|e_1)$, mora obstajati zamenjava Θ, za katero je $H\Theta \subseteq \bot_2$, zaradi česar $L_v\Theta \in \bot_2$ in $A_i/a'_{i'} \in \Theta$. Prisotnost $A_i/a'_{i'}$ v Θ povzroči tudi $P(\ldots A_i \ldots A_{i'} \ldots)\Theta = P(\ldots a'_{i'}, \ldots A_{i'} \ldots)$. Vendar takšen literal v $\bot_2$ ne more obstajati, saj bi v tem

primeru množica instanciacij $a'_{i'}$ vsebovala tudi vnos $\langle P/i/l\rangle$. S tem smo dosegli kontradikcijo, saj mora veljati ali $i = i'$ ali pa mora množica instanciacij imeti več kot le en element. Iz tega sledi, da se morata vrednosti pojaviti na istem argumentnem mestu.

Možnost 3: $\mathbf{l} \neq \mathbf{l}'$ Z $\mathscr{L}_j(\bot_i)$ označimo množico literalov na nivoju j spodnjega stavka $\bot_i$. Brez izgube splošnosti predpostavljamo, da ima vsaka vrednost zgolj en element v svoji množici instanciacij. Vedno lahko namreč vrednost z več instancijacijami predstavimo z več ločenimi konstantami, ki jih med seboj povežemo s predikatom, ki pove, da dve konstanti označujeta isto vrednost.

Naj bodo vrednosti L, L' in L_v definirane kot prej. Naj se na i-tem argumentnem mestu literalov L in L' pojavita vrednosti a_i in a'_i z množicama instanciacij $MI[a_i] = \{\langle P_1/k/l_1\rangle\}$ in $MI[a'_i] = \{\langle P_1/k/l'_1\rangle\}$ in $l_1 \neq l'_1$.

V $\mathscr{L}_l(\bot_1)$ torej obstaja nek literal L_1 s predikatnim simbolom P_1, ki ima kot svoj k-ti argument vrednost a_i. Podobno se v $\mathscr{L}_{l'}(\bot_2)$ nahaja literal L'_1, ki ima a'_i kot svoj k-ti argument.

Kot poprej si oglejmo, kakšna mora biti zamenjava Θ, za katero pri dani $H = ARMG(e_2|e_1)$ velja $H\Theta \subseteq \bot_2$, če je L_v del H. Θ zagotovo vključuje zamenjavo A_i/a'_i.

Najprej ugotovimo, da mora za vključitev L_v v H biti v njej prisotna tudi variabilizacija $L_1 = P_1(a_{11}, \ldots A_{1n^+}, \ldots a_i \ldots)$, saj je to edini način za instanciranje vhodne vrednost A_i v L_v. Pri apliciranju zamenjave Θ postane k-ti argument literala $var(L_1)$ enak a'_i. Ker je L'_1 edini literal v $\bot_2$, ki instancira a'_i (njegova množica instanciacij ima namreč le en element), je posledično tudi edini literal, ki je lahko po izvedbi zamenjave Θ nad hipotezo H enak $var(L_1)\Theta$ (spomnimo, vsak literal v $H\Theta$ mora biti enak enemu izmed literalov v $\bot_2$, da $H\Theta \subseteq \bot_2$). Θ mora torej vsebovati tudi zamenjave za vse vhodne argumente $var(L_1)$: $A_{11}/a'_{11} \ldots A_{1n^+}/a'_{1n^+}$.

Ker $l_1 \neq l'_1$, drži bodisi $l_1 > l'_1$ bodisi $l'_1 > l_1$.

Možnost $l_1 > l'_1$ Ker se literal L_1 pojavi na nivoju l_1, se mora vsaj en izmed njegovih vhodnih argumentov pojaviti na nivoju $l_2 = l_1 - 1$. Naj bo

en izmed njih i-ti argument, ki ima vrednost a_{1i}. Njegova množica instanciacij mora biti enaka $MI[a_{1i}] = \{\langle P_2/k_2/l_2\rangle\}$. Istoležni argument v L'_1 je a'_{1i} z lastno množico instanciacij $MI[a'_{1i}] = \{\langle P_2/k_2/l'_2\rangle\}$, kjer je $l_2 > l'_2$, saj je $l_2 = l_1 - 1$, l'_2 pa je vsaj za 1 manjši od l'_1 (največja globina vhodnih argumentov L'_1 je namreč $l'_1 - 1$, sicer bi njegova globina bila večja od l'_1). Ponovno poiščemo ustrezna literala $L_2 = P_2(a_{21} \ldots a_{2n^+} \ldots a_{1i}) \in \mathscr{L}_{l_2}(\bot_1)$ in $L'_2 = P_2(a'_{21} \ldots a'_{2n^+} \ldots a'_{1i}) \in \mathscr{L}_{l'_2}(\bot_2)$. Postopek ponavljamo in vsakič v j-ti iteraciji izberemo mesto argumenta k_j tako, da se globina literala L_{j+1}, ki instancira vrednost a_{jk}, zmanjša za 1, globina njegove ustreznice L'_{j+1}, ki instancira a'_{jk}, pa vsaj za 1. Po največ $s \leq l'_1$ korakih se L'_s nahaja v $\mathscr{L}_0(\bot_2)$ in je tako enak glavi stavka. Vendar je L_s element $\mathscr{L}_{l_1-s}(\bot_1)$, kjer $l - s > 0 - var(L_s)$ je literal v telesu hipoteze in mora biti del nje, saj sicer hipoteza ni povezana z glavo. Vendar literala, ki bi ob takšni zamenjavi uspel, v $\bot_2$ ni, saj bi sicer množica instanciacij vrednosti a'_{sk} vsebovala vsaj dve vrednosti (tj. eno instanciacijo v glavi in drugo v telesu), kar nasprotuje dejstvu, da se vsaka vrednost instancira le na posameznem mestu.

Možnost $l'_1 > l_1$ Argumentacija sledi prejšnjemu postopku, le da tokrat na vsakem koraku izberemo pozicijo argumenta k, ki se instancira v literalu $L_j \in \mathscr{L}_{l'_1-j}(\bot_2)$. Po $s \leq l_1$ korakih se eden izmed vhodnih argumentov literala L'_s, a'_{sk} instancira na nivoju $s - 1 \neq 0$, istoležni argument L_s, a_{sk} pa v glavi $\bot_1$, kar nas privede do sklepa, da mora množica instanciacij a_{si} vsebovati dva elementa – enega kot vhodni argument glave, drugega kot izhodni argument nekega literala na nivoju $s-1$, kar pa ni mogoče, saj imajo vse množice instanciacij en sam element.

Ker smo v obeh primerih prišli do kontradikcije, mora veljati $l_1 = l'_1$.

Algoritem, ki z uporabo omejitev izračuna dvojno nasičenje primerov e_1 in e_2, je podan kot algoritem 15. Pri tem poudarimo, da je izračun spodnjih stavkov posameznih primerov in njihove pripadajoče množice instanciacij mogoče izračunati že med branjem problema in jih predpomniti, tako da ni potrebno njihovo vsakokratno raču-

nanje, s čimer dosežemo dodatne pohitritve.

Vhod : Par primerov e_1, e_2
deklaracije načina M
predznanje BK
Izhod : Dvojno nasičenje

Function dvojno(e_1, e_2, M, BK) is
  $nasicenje \leftarrow \{\}$;
  $MI = \{\}$;
  for $e_i \in \{e_1, e_2\}$ *do*
    $\bot_i \leftarrow nasicenje(e_i, M, BK)$;
    for $v \in vrednosti(\bot_i)$ *do*
      $MI[e_i][v] \leftarrow mnozica\_instanciacij(\bot_i, v, M)$;
    end
  end
  for literal $L \in \bot_1$ *do*
    if $\exists L' \in \bot_2 : kompatibilna(L, L', MI[e_1], MI[e_2], M)$ *then*
      $nasicenje \leftarrow nasicenje \cup L$;
    end
  end
  return nasicenje;
end

Algoritem 15: Izračun dvojnega nasičenja.

Iz algoritma je razvidno, da je operacija dvojnih nasičenj, prav tako kot operacija relativne najmanj splošne posplošitve, asimetrična. Asimetričnost izhaja iz dejstva, da odstranjujemo literale iz spodnjega stavka enega primera glede na to, ali je v spodnjem stavku drugega primera prisoten njemu kompatibilen literal – rezultat dvojnega nasičenja je tako podmnožica posplošitve spodnjega stavka prvega primera. Če vlogi primerov zamenjamo, bo rezultat podmnožica posplošitve spodnjega stavka drugega primera. Iz tega izhaja, da v splošnem ne velja $dvojno(e_1, e_2, M, BK) = dvojno(e_2, e_1, M, BK)$.

Treba je poudariti, da Izrek 1 temelji na predpostavki, da imamo na voljo celotna spodnja stavka primerov $\bot_1$ in $\bot_2$. Ker sta v praksi spodnja stavka podvržena ome-

jitvam maksimalne globine literalov ter največjega dovoljenega števila resolucij, lahko pride do razhajanj pri hipotezah, dobljenih s sistemom, ki uporablja dvojna nasičenja, in osnovnim sistemom. Primera 22 in 23 prikazujeta izvor teh razlik.

Primer 22: Podana naj bosta primera `pr(a1)` ter `pr(a2)` in predznanje s predikatom `l` z deklaracijo načina `l(+, -)`. Spodnja stavka podanih primerov naj bosta:

```
pr(a1) :- l(a1, b1), l(a1, c1), l(b1, c1), l(c1, d1).

pr(a2) :- l(a2, b2), l(b2, c2), l(c2, c2).
```

Pri izračunu teh spodnjih stavkov z omejitvijo globine 2, sta izračunana spodnja stavka enaka:

```
pr(a1) :- l(a1, b1), l(a1, c1), l(b1, c1), l(c1, d1).

pr(a2) :- l(a2, b2), l(b2, c2).
```

Literalu `l(c1, d1)` v spodnjem stavku primera `pr(a2)`, izračunanem z omejeno največjo globino literalov, ni mogoče najti kompatibilnega literala. Čeprav ta obstaja, ga je moč odkriti le pri zvišanju vrednosti največje dovoljene globine na najmanj 3. △

Primer 23: Imejmo podana primera `pr(a1)` ter `pr(a2)` in predznanje z deklaracijama načina `l(+,-)` ter `m(+,-)`. Spodnja stavka primerov `pr(a1)` ter `pr(a2)` naj bosta enaka:

```
pr(a1) :- l(a1, b1), m(b1, c1), m(b1, d1), l(d1, e1).

pr(a2) :- l(a2, b2), m(b2, c2), m(b2, d2), m(b2, e2), l(e2, f2).
```

Spodnja stavka teh primerov, izračunana pri omejitvi največjega števila resolucij na 2, sta:

```
pr(a1) :- l(a1, b1), m(b1, c1), m(b1, d1), l(d1, e1).

pr(a2) :- l(a2, b2), m(b2, c2), m(b2, d2).
```

Zaradi omejitve števila resolucij se v spodnjem stavku primera `pr(a2)` ne pojavi literal `m(b2, e2)`. Brez tega literala literal `l(e2, f2)` ni dosegljiv, saj se nikjer ne instancira konstanta na njegovem vhodnem mestu `e2`. Posledično literal `l(d1, e1)` iz

spodnjega stavka primera `pr(a1)` nima kompatibilnega literala v spodnjem stavku primera `pr(a2)`, izračunanem pri omejitvi največjega števila resolucij na 2. Pri zadostnem dovoljenem številu resolucij do takšnih odstopanj ne pride. △

4.4 N-terna nasičenja

Dvojna nasičenja je mogoče trivialno posplošiti na poljubno velike množice primerov.

Izrek 2: Če za literal $L \in \perp_1$ ne obstaja kompatibilen literal v enem izmed spodnjih stavkov $\perp_2 \ldots \perp_n$, njegova posplošitev ne more biti del posplošitve primerov $e_1, \ldots e_n$, $ARMG(e_2 \ldots e_n | e_1)$.

Dokaz 2: Za $n = 2$ smo izrek že dokazali. Če ta dokaz apliciramo na vse pare e_1, e_i $(2 \leq i \leq n)$, dobimo dokaz posplošenega izreka.

Pri uporabi n-ternih nasičenj pa se je potrebno zavedati naslednje nevarnosti. Ko namreč sistem izbere množico primerov, ki jih bo uporabil kot semena za trenutno hipotezo, predpostavlja, da so v "pravilni" rešitvi vsi ti primeri pokriti z istim stavkom. Kadar temu ni tako, bo dobljeni stavek presplošen, kar privede do neoptimalnih rešitev. V nadaljevanju podamo kratko analizo verjetnosti, da so naključno izbrani primeri v ciljni rešitvi resnično pokriti z istim stavkom.

Denimo, da imamo podanih ne^+ pozitivnih primerov, ki so v ciljni rešitvi pokriti s k stavki. Vsak stavek naj pokrije enako število pozitivnih primerov $\frac{ne^+}{k}$. Pri naključnem izboru n primerov je verjetnost, da so izbrani primeri resnično pokriti s strani istega stavka, enaka $\prod_{j=1}^{n-1} \frac{1}{ne^+}(\frac{ne^+}{k} - j)$. Z večanjem n se ta verjetnost zmanjšuje, zaradi česar pričakujemo, da večanje n poslabša kvaliteto zgrajenih rešitev.

Algoritem za izračun dvojnih nasičenj, posplošen na n-terna nasičenja, je podan kot algoritem 16.

4.5 Ocena časovne zahtevnosti izračuna dvojnih nasičenj

Prihranki, ki jih lahko dosežemo z uporabo dvojnih nasičenj, so neposredno odvisni od kompleksnosti njihovega izračuna. V tem podpoglavju si podrobneje ogledamo asimptotične časovne zahtevnosti posameznih korakov izračuna.

Prvi korak našega pristopa predstavlja izračun množic instanciacij za vse konstan-

Vhod : Množica primerov $E = \{e_1, ... e_n\}$,
deklaracije načina M,
predznanje BK
Izhod : n-terno nasičenje

Function n-terno($E = \{e_1, ... e_n\}$, M, BK) *is*
  $nasicenje \leftarrow \{\}$;
  $MI = \{\}$;
  for $e_i \in E$ *do*
    $\perp_i \leftarrow nasicenje(e_i, M, BK)$;
    for $v \in vrednosti(\perp_i)$ *do*
      $MI[e_i][v] \leftarrow mnozica\_nasicenj(\perp_i, v, M)$;
    end
  end
  for literal $L \in \perp_1$ *do*
    if $\forall i, 1 < i < |E| : \exists L' \in \perp_i : kompatibilen(L, L', MI[e_1], MI[e_i], M)$
    then
      $nasicenje \leftarrow nasicenje \cup L$;
    end
  end
  return nasicenje;
end

Algoritem 16: Izračun n-ternega nasičenja primerov.

te v vseh primerih. Postopek je prikazan kot algoritem 13. Algoritem se pri gradnji množice instanciacij vrednosti V za spodnji stavek B sprehodi čez argumente vseh literalov v stavku B in doda vanjo vse pojavitve vrednosti V. Dodajanje posameznega vnosa pri uporabi rdeče-črnega drevesa terja $O(\log n)$ časa, kjer je n število elementov v njem. Pri največjih dolžinah stavka C_L in številnosti literalov M_A za izgradnjo množice instanciacij tako potrebujemo $O(C_L M_A \log(C_L M_A))$ časa. Pomemben prihranek pri času lahko dosežemo z gradnjo množic instanciacij za vse vrednosti v nekem stavku hkrati. Pri takšni gradnji moramo ob vsaki vrednosti še izbrati množico instanciacij, ki ustreza tej vrednosti. Čas, potreben za to izbiro, zavisi od uporabljene podatkovne strukture – v našem primeru smo množice instanciacij hranili v rdeče-črnem drevesu, torej je čas izbora enak $O(\log n)$, kjer n predstavlja število elementov v drevesu. Največje število vrednosti v stavku je navzgor omejeno s $C_L M_A$, kar nas privede do končne zahtevnosti izgradnje vseh množic instanciacij za posamezen stavek $O(C_L M_A(\log(C_L M_A) + \log(C_L M_A)) = O(C_L M_A \log(C_L M_A))$.

Množice instanciacij se tekom reševanja konkretnega problema ne spreminjajo, zato jih lahko izračunamo že ob branju problema v sistem in predpomnimo. Pri prebranem problemu z n_e primeri tako izgradnja množic instanciacij terja enkratno izvedbo reda $O(n_e C_L M_A \log(C_L M_A))$.

Algoritem 14 za preverjanje kompatibilnosti literalov izračuna presek množic instanciacij za vse istoležne argumente literalov. Množice instanciacij so predstavljene kot rdeče-črna drevesa, kar omogoča izračun preseka v redu časovne zahtevnosti $O(n)$ kjer je n velikost največje izmed njiju – $O(M_A C_L)$. Ker presek izračunamo za vsak par istoležnih argumentov literalov, je končna časovna zahtevnost reda $O(M_A^2 C_L)$.

Za končen izračun dvojnega nasičenja, kot je zapisano v algoritmu 15, moramo v najslabšem primeru preveriti kompatibilnost vseh parov literalov v spodnjih stavkih primerov, katerih dvojno nasičenje računamo. Pri najdaljši dolžini spodnjega stavka C_L to navrže C_L^2 izračunov kompatibilnosti – za izračun dvojnega nasičenja potrebujemo $O(C_L^3 M_A^2)$ časa.

Kljub dodatnemu delu, katerega rast je kubična v velikosti spodnjih stavkov, pričakujemo, da bo korak izračuna dvojnega nasičenja pogosto koristen. Z vsakim odstranjenim literalom se namreč poenostavi problem izračuna asimetrične relativne minimalne posplošitve (ARMG). Njen izračun namreč sestoji iz postopka, kjer iz posplošitve spodnjega stavka prvega primera odstranimo literal ter preverimo, ali tako dobljena hipoteza pokrije drugi primer, ki služi kot vhod v ARMG – izračun pokritja pa spa-

da v množico NP-polnih problemov. Z opisanim postopkom tako zmanjšamo število pokritij, ki se ovrednotijo tekom izračuna ARMG, vsako izmed njih pa predstavlja reševanje NP-polnega problema.

Dodatno pohitritev opisanega postopka bi bilo mogoče doseči z zamenjavo rdečečrnih dreves z razpršenimi tabelami. Razpršene tabele namreč omogočajo vstavljanje in iskanje elementov v (povprečnem) konstantnem času. Pri tem bi lahko pričakovali prihranke zgolj v okviru gradnje podatkovnih struktur med branjem problema. Med samim izvajanjem je namreč edina uporabljena operacija izračun preseka, katere časovna zahtevnost je v obeh primerih linearna. [1] Z vidika prostorske zahtevnosti bi lahko namesto rdeče-črnih dreves uporabili Bloomov filter, pri čemer bi sprejeli tveganje, da se v dvojnem nasičenju pojavijo nekateri odvečni literali (tj. literali brez kompatibilnega para).

4.6 Eksperimenti

Opisani metodi dvojnih in n-ternih nasičenj smo implementirali kot del sistema ProGolem [39]. Ker osnovna implementacija imenovanega sistema ne uporablja predpomnenja spodnjih stavkov primerov, smo zaradi poštenosti primerjave rezultatov tudi njegovo osnovno implementacijo nadgradili s predpomnenjem. Takšna primerjava nam tudi omogoča ločiti med pohitritvijo, ki je posledica predpomnenja, in pohitritvijo, katere vzrok je uporaba dvojnih oz. n-ternih nasičenj.

Med seboj smo najprej primerjali osnovni sistem ProGolem, ProGolem z uporabo predpomnenja (imenovan ProGolemCaching), ter ProGolem z uporabo dvojnih nasičenj (imenovan ProParGolem). Te sisteme smo testirali na 11 realnih podatkovnih množicah in merili:

- doseženo točnost,
- čas, potreben za rešitev problema,
- velikost nasičenja, ki predstavlja osnovo za nadaljnje posploševanje.

Tabela 4.1 prikazuje dosežene čase in točnosti na izbranih podatkovnih množicah, ki ju dosežeta sistema ProGolem in ProParGolem. Vse razlike med doseženimi točnostmi so nesignifikantne, opazne pa so večje razlike v izvajalnih časih. Relativne razlike

[1] Preliminarni rezultati z uporabo razpršenih tabel, implementiranih v knjižnici *bhash* v YAP prologu, so pokazali, da se izvajanje pri uporabi razpršenih tabel upočasni za 5 do 10 odstotkov.

v časih izvajanja so podane v tabeli 4.2. Opazimo, da sistem ProGolem v primerjavi s sistemom ProParGolem v povprečju potrebuje 4.07 krat več časa, ProGolem s predpomnenjem pa potrebuje v povprečju zgolj 3% več časa. K takšni razliki med povprečjema največ prispevajo podatkovne množice iz zbirke "demospongiae", kjer celotna pohitritev izhaja iz uporabe predpomnenja. Pri ostalih domenah opazimo, da ProGolem s predpomnenjem potrebuje tudi do ranga 1.44 (metabolism) oz. 1.24 (carcinogenesis ter dsstox) več časa.

Iz tega izhaja, da je opažena pohitritev močno odvisna od podatkovne množice same. Velikosti stavkov, ki služijo kot spodnja meja prostora, znotraj katerega se išče rešitev, so podane v tabeli 4.3. Pričakovali smo, da bo pohitritev močno korelirala z razmerjem med osnovnim ter dvojnim nasičenjem, vendar je situacija kompleksnejša. Manjše število literalov v izračunanem spodnjem stavku sicer neposredno odgovarja manjšemu številu izračunov Θ-vsebovanosti pri izračunu ARMG dveh primerov, vendar so prihranki zelo odvisni tudi od determiniranosti predznanja. Pri visoko determiniranih problemih je namreč izračun Θ-vsebovanosti preprost (tj. sistemu ni potrebno preveriti velikega števila različnih možnih instanciacij) in je prihranek na času posledično ustrezno manjši oz. je odstranjevanje odvečnih literalov – čeprav z njegovo uporabo spodnji stavek zmanjšamo – časovno v absolutnem smislu bolj potratno. Očiten primer so domene "demospongiae", kjer so navkljub temu, da je dvojno nasičenje za tretjino manjše od navadnega, prihranki zelo majhni. Podobno lahko na primeru "mutagenesis" domene opazimo, da je zmanjšanje spodnjega stavka napram domenam "demospongiae" sicer manjše, pohitritev pa večja.

Uporaba predpomnenja sicer nekoliko podaljša čas branja problema, saj je med branjem potrebno naračunati spodnje stavke ter množice instanciacij za vse pozitivne primere v podatkovni množici. Časi branj z in brez vnaprejšnjega izračuna spodnjih stavkov ter množic instanciacij primerov so podani v tabeli 4.4. Opazimo, da se čas branja zaradi potrebnih dodatnih izračunov podaljša za približno 50%. Vendar se izkaže, da je čas branja in predpomnenja bistveno manjši od časa, potrebnega za iskanje hipoteze, zaradi česar se predhodni izračun v končnem seštevku časov izplača.

Ugotovili smo, da uporaba dvojnih nasičenj občutno pohitri delovanje sistema brez izgube točnosti najdenih rešitev. Na istih podatkovnih množicah smo eksperimentalno ovrednotili tudi uporabo n-ternih nasičenj za vrednosti n med 3 in 5 in jih primerjali s sistemom ProParGolem, ki uporablja dvojna nasičenja. Po pričakovanjih se z večanjem parametra n zniža kvaliteta najdenih hipotez, kot je razvidno iz slike 4.1. V skladu s

Tabela 4.1

Dosežene točnosti in potrebni časi na realnih podatkovnih množicah. Vse meritve so bile opravljene z uporabo 10-kratnega prečnega preverjanja z izjemo podatkovnih množic dsstox in pyrimidines (5-kratno prečno preverjanje) in proteins (16-kratno prečno preverjanje). Tabela ne vsebuje podatkov za ProGolemCaching, saj so zanj točnosti enake sistemu ProGolem, relativni časi zanj pa so podani v tabeli 4.2.

	ProGolem		ProParGolem	
	Čas [s]	Točnost [%]	Čas [s]	Točnost [%]
Alzheimer-Acetyl	40.61 ± 2.99	66.30 ± 3.26	30.92 ± 1.27	66.47 ± 3.85
Alzheimer-Amine	9.51 ± 0.67	70.57 ± 5.64	5.56 ± 0.52	70.57 ± 5.64
Alzheimer-Mem	35.09 ± 1.78	64.38 ± 6.43	24.91 ± 1.98	64.41 ± 6.26
Alzheimer-Toxic	150.08 ± 18.32	78.17 ± 6.71	94.78 ± 11.02	77.50 ± 6.42
Carcinogenesis	467.82 ± 18.44	62.20 ± 8.27	304.46 ± 17.78	62.98 ± 7.84
Dspg-Astrophorida	97.47 ± 6.06	94.26 ± 3.17	15.06 ± 0.95	94.84 ± 2.77
Dspg-Axinellida	270.22 ± 12.17	92.26 ± 4.78	30.59 ± 1.83	92.26 ± 4.78
Dspg-Dictyoceratida	45.02 ± 8.05	97.25 ± 2.05	7.10 ± 0.37	97.25 ± 1.96
Dspg-Hadromerida	273.88 ± 7.43	88.46 ± 5.78	26.47 ± 0.60	88.46 ± 5.78
Dspg-Halichondrida	380.23 ± 12.84	88.88 ± 6.86	41.09 ± 2.97	88.88 ± 6.86
Dspg-Haplosclerida	155.82 ± 11.80	94.16 ± 6.31	17.74 ± 0.97	94.16 ± 6.31
Dspg-Poecilosclerida	289.73 ± 12.76	84.89 ± 5.60	35.18 ± 1.68	84.89 ± 5.60
Dsstox	252.67 ± 17.17	69.71 ± 4.09	180.58 ± 9.04	69.22 ± 4.02
Metabolism	150.08 ± 18.32	66.46 ± 10.22	94.78 ± 11.02	66.64 ± 10.08
Mutagenesis	572.38 ± 64.08	78.58 ± 10.62	444.98 ± 70.38	76.95 ± 11.74
Proteins	9909.22 ± 355.97	59.65 ± 8.29	8688.55 ± 319.24	60.13 ± 8.69
Pyrimidines	35.09 ± 1.78	75.06 ± 4.67	24.91 ± 1.98	72.93 ± 4.71
Triazines	572.38 ± 64.08	68.88 ± 2.12	444.98 ± 70.38	68.73 ± 2.08

Tabela 4.2

Relativni časi, potrebni za izvedbo 10-kratnega prečnega preverjanja, glede na čas, ki ga potrebuje ProParGolem. Časi so povprečeni čez 10 iteracij 10-kratnega prečnega preverjanja (5-kratnega prečnega preverjanja v primeru podatkovnih množic pyrimidines in dsstox, ter 16-kratnega prečnega preverjanja pri proteins).

	ProGolem	ProGolemCaching	ProParGolem
Alzheimer-Acetyl	1.31	1.06	1.00
Alzheimer-Amine	1.71	0.97	1.00
Alzheimer-Mem	1.41	1.06	1.00
Alzheimer-Toxic	1.25	1.05	1.00
Dspg-Astrophorida	6.47	1.05	1.00
Dspg-Axinellida	8.83	1.01	1.00
Dspg-Dictyoceratida	6.34	1.02	1.00
Dspg-Hadromerida	10.35	1.02	1.00
Dspg-Halichondrida	9.25	1.05	1.00
Dspg-Haplosclerida	8.78	1.00	1.00
Dspg-Poecilosclerida	8.24	1.02	1.00
Carcinogenesis	1.54	1.24	1.00
Dsstox	1.40	1.24	1.00
Metabolism	1.58	1.44	1.00
Mutagenesis	1.29	1.15	1.00
Proteins	1.29	1.09	1.00
Pyrimidines	1.18	0.99	1.00
Triazines	1.12	1.05	1.00

Tabela 4.3

Povprečna dolžina (število literalov) stavkov, ki so uporabljeni kot spodnja meja preiskovanega prostora hipotez. Dolžine so povprečene čez 1000 naključno izbranih parov v vsaki podatkovni množici.

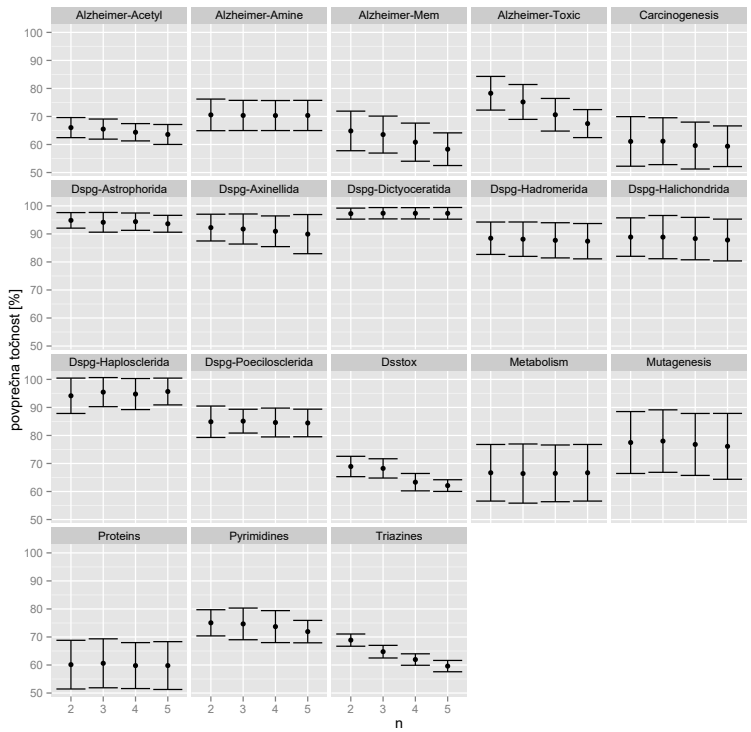
	ProGolem	ProParGolem
Alzheimer-Acetyl	38.0 ± 12.6	16.88 ± 10.5
Alzheimer-Amine	31.0 ± 8.9	14.81 ± 8.1
Alzheimer-Mem	36.8 ± 12.4	15.94 ± 10.1
Alzheimer-Toxic	37.5 ± 11.5	16.8 ± 11.0
Carcinogenesis	88.8 ± 26.8	38.8 ± 22.6
Dspg-Astrophorida	60.4 ± 9.9	37.1 ± 9.4
Dspg-Axinellida	50.7 ± 9.0	32.1 ± 8.2
Dspg-Dictyoceratida	51.3 ± 7.3	33.9 ± 5.9
Dspg-Hadromerida	48.7 ± 11.1	30.0 ± 6.3
Dspg-Halichondrida	53.9 ± 9.6	33.8 ± 8.8
Dspg-Haplosclerida	52.9 ± 8.8	34.8 ± 6.9
Dspg-Poecilosclerida	54.8 ± 11.4	33.3 ± 7.7
Dsstox	32.4 ± 7.3	15.4 ± 11.6
Metabolism	72.3 ± 92.6	44.0 ± 67.6
Mutagenesis	66.4 ± 5.4	53.0 ± 7.0
Proteins	292.7 ± 42.9	285.4 ± 42.5
Pyrimidines	49.9 ± 12.9	35.3 ± 12.6
Triazines	67.6 ± 12.0	62.8 ± 13.2

Tabela 4.4

Povprečen čas branja podanega problema v sekundah. Meritve so podane na podlagi desetih branj podatkovne množice.

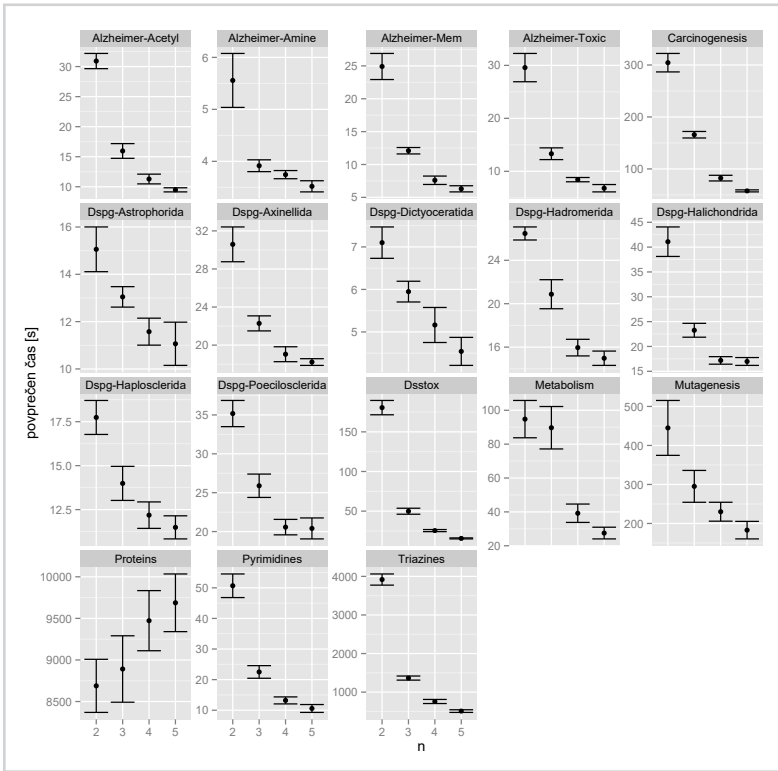
	ProGolem	ProParGolem
Alzheimer-Acetyl	1.75 ± 0.09	2.60 ± 0.10
Alzheimer-Amine	0.89 ± 0.04	1.23 ± 0.08
Alzheimer-Carcinogenesis	1.66 ± 0.26	1.74 ± 0.16
Alzheimer-Mem	0.82 ± 0.04	1.23 ± 0.07
Alzheimer-Toxic	1.17 ± 0.10	1.75 ± 0.08
Dspg-Astrophorida	60.47 ± 1.71	64.33 ± 1.57
Dspg-Axinellida	60.63 ± 0.64	62.28 ± 3.32
Dspg-Dictyoceratida	63.67 ± 2.69	63.64 ± 1.76
Dspg-Hadromerida	62.67 ± 1.29	62.78 ± 0.17
Dspg-Halichondrida	63.31 ± 2.85	62.66 ± 0.31
Dspg-Haplosclerida	58.65 ± 0.69	62.86 ± 0.39
Dspg-Poecilosclerida	58.59 ± 0.81	62.35 ± 0.22
Dsstox	0.69 ± 0.11	0.96 ± 0.14
Metabolism	0.49 ± 0.03	1.02 ± 0.09
Mutagenesis	0.55 ± 0.06	0.64 ± 0.07
Proteins	12.46 ± 0.15	18.06 ± 0.66
Pyrimidines	4.82 ± 0.10	6.45 ± 0.34
Triazines	58.74 ± 1.24	90.38 ± 1.56

pričakovanji z večanjem števila primerov, vključenih v n-terno nasičenje, se čas, potreben za izračun hipoteze, manjša. Tukaj izstopa podatkovna množica "proteins" – pri njej namreč opazimo rast časa izvajanja z večanjem parametra n. Razlog za takšno obnašanje je razviden iz slike 4.3 – pri tej podatkovni množici je trend upadanja velikosti n-ternega nasičenja izjemno počasen – kar pomeni, da večina literalov ostane v nasičenju (kar vodi do nizke pospešitve), zaradi česar se izračun n-ternega nasičenja odraža kot odvečno delo. Takšne situacije se pojavijo pri veliki regularnosti spodnjih stavkov: bodisi kadar imamo opravka s problemom atributne narave bodisi kadar so dani primeri podobno strukturirani.



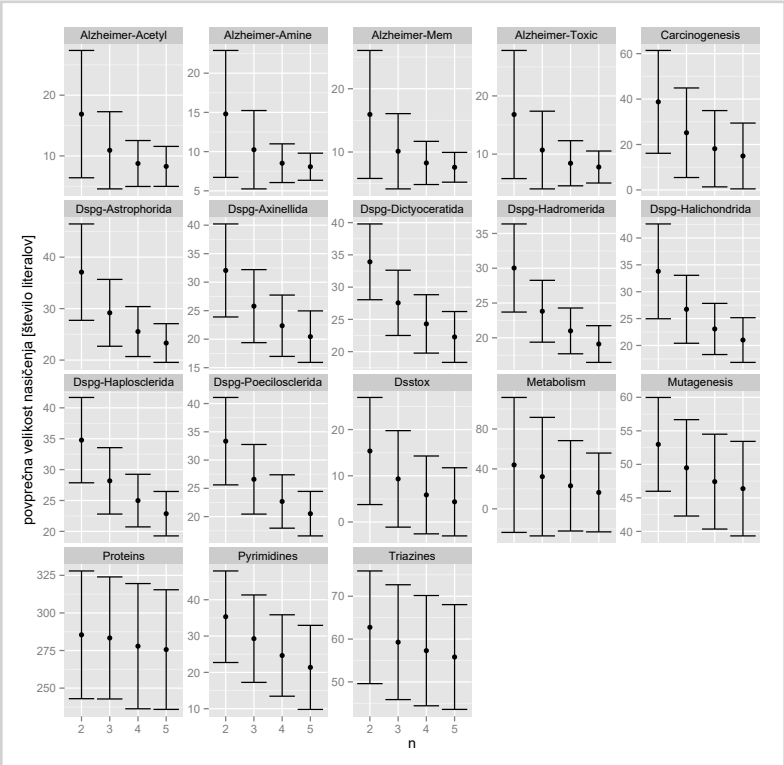
Slika 4.1

Točnosti, dosežene z uporabo n-ternih nasičenj, za vrednosti $2 \leq n \leq 5$.



Slika 4.2

Časi, potrebni za izgradnjo hipoteze pri uporabi n-ternih nasičenj, za vrednosti $2 \leq n \leq 5$.



Slika 4.3

Velikosti n-ternih nasičenj, ki služijo kot spodnja meja prostora preiskovanja, za vrednosti $2 \leq n \leq 5$.

5

Testiranje in uporaba združenega sistema ProParGolemNRnot

V tretjem poglavju smo kot ključno pomanjkljivost sistema ProGolemNot izpostavili njegovo časovno potratnost. Prvo pohitritev njegovega delovanja predstavlja sistem ProGolemNRNot, ki specializacije zaprtega sveta ne izvede nad čisto vsemi kandidatnimi hipotezami, temveč le nad najboljšimi hipotezami v snopu. V tem poglavju poskusimo ta sistem še dodatno pohitriti z uporabo dvojnih nasičenj, opisanih v četrtem poglavju. Kombinirani sistem imenujemo ProParGolemNRNot.

Sistem ProParGolemNRNot najprej primerjamo s sistemom ProGolemNRNot na standardnem naboru podatkovnih množic, nakar njegovo zmogljivost preverimo še na podatkih, pridobljenih iz simuliranih globinskih senzorjev. Sistem smo uporabili za učenje primitivnih geometrijskih konceptov iz njih [15]. Uporabili smo postopno učenje vedno bolj kompleksnih konceptov, pri čemer so vsi prej naučeni koncepti na voljo pri učenju novega [41].

5.1 *ProParGolemNRNot*

Čas, ki ga sistem ProGolemNRNot potrebuje, da sestavi odgovor na podani problem, poskusimo dodatno znižati z uporabo dvojnih nasičenj. Glede na rezultate, opisane v četrtem poglavju, smo uporabili le dvojna nasičenja, saj n-terna nasičenja lahko povzročijo slabšanje rezultata v smislu njegove točnosti.

Združeni sistem (ProParGolemNRNot) smo z njegovo osnovno inačico (ProGolemNRNot) primerjali na naboru realnih podatkovnih množic. Na podlagi opažanj, zapisanih v četrtem poglavju, pričakujemo odvisnost pohitritve od dane podatkovne možice. Tabeli 5.1 ter 5.2 prikazujeta meritve, ki zajemajo primerjavo teh dveh sistemov.

Statistično primerjavo dobljenih točnosti smo izvedli z uporabo t-testa. Nobena izmed razlik v doseženih točnostih ni statistično značilna, kar se sklada z ugotovitvami, opisanimi v četrtem poglavju.

Iz primerjave izstopajo predvsem meritve na podatkovnih množicah "demospongiae" (s predpono "Dspg") ter množici iz šahovske domene ("Kr_vs_kp" in "Krk_illegal"). Kot smo že opazili v četrtem poglavju, dvojna nasičenja bistveno pohitrijo izvajanje na množicah "demospongiae". V primerjavi sistemov ProParGolemNRNot ter ProGolemNRNot je na teh množicah opaziti, da z uporabo dvojnih nasičenj čas izvajanja zmanjšamo na desetino potrebnega pri uporabi sistema ProGolemNRNot.

Nasprotno sliko dobimo na obeh problemih iz šahovskih domen. Predznanje v njima je namreč zelo pravilno strukturirano – za vsako figuro na šahovnici so podane

Tabela 5.1

Tabela točnosti, dobljenih s sistemoma ProGolemNRNot ter ProParGolemNRNot.

	ProGolemNRNot	ProParGolemNRNot
Krk_Illegal	99.90 ± 0.09	99.90 ± 0.09
Kr_vs_kp	88.62 ± 4.58	88.64 ± 5.35
Alzheimer-Acetyl	76.09 ± 4.86	75.19 ± 5.10
Alzheimer-Amine	83.77 ± 3.64	83.95 ± 3.88
Alzheimer-Mem	68.67 ± 6.34	67.92 ± 6.15
Alzheimer-Toxic	85.17 ± 4.05	85.24 ± 3.92
Carcinogenesis	59.96 ± 7.21	61.23 ± 8.03
Dspg-Astrophorida	95.18 ± 5.02	94.82 ± 4.88
Dspg-Axinellida	89.91 ± 6.47	89.22 ± 6.96
Dspg-Dictyoceratida	97.16 ± 3.12	97.08 ± 3.18
Dspg-Hadromerida	88.99 ± 7.35	88.43 ± 7.63
Dspg-Halichondrida	87.64 ± 6.82	87.64 ± 6.82
Dspg-Haplosclerida	95.15 ± 6.49	95.22 ± 5.94
Dspg-Poecilosclerida	87.33 ± 6.47	87.31 ± 6.81
Dsstox	69.73 ± 4.29	69.25 ± 3.79
Metabolism	66.71 ± 12.05	66.52 ± 12.90
Mutagenesis	81.10 ± 9.44	81.31 ± 10.24
Pyrimidines	87.70 ± 2.10	87.77 ± 1.80

njene koordinate, spodnji stavek pa zajema prostorsko relacijo med polji na šahovnici. Izračun dvojnega nasičenja se v teh primerih izkaže kot dodatno delo, saj zaradi pravilne strukture ni mogoče iz spodnjih stavkov odstraniti nobenega literala.

Poudariti je treba, da prihranki pri uporabi ProParGolemNRNot izhajajo tako iz uporabe predpomnenja spodnjih stavkov kot tudi odstranjevanja odvečnih literalov iz njih. Kot smo opazili že v četrtem poglavju, velik delež pohitritve izhaja ravno iz uporabe predpomnenja, še posebej je to izrazito na domenah “demospongiae”.

5.2 Simulacija globinskih senzorjev

Zamišljeni vir podatkov je stereo naprava, sestavljena iz projektorja in kamere. Projektor na sceno projicira vzorec točk v obliki enakokrakega križa. Iz zajete slike je mogoče na podlagi popačenja in pozicije projiciranega vzorca izračunati oddaljenost objektov, na katere so se projicirale točke vzorca. Projicirani vzorec se med zajemanjem slik vrti, s čimer pokrije celotno površino kroga, orisanega z njegovimi skrajnimi točkami.

Tabela 5.2

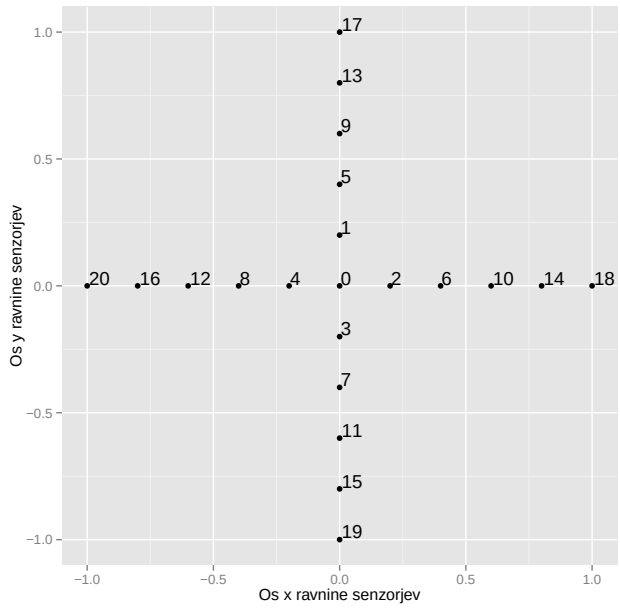
Tabela časov v sekundah, potrebnih za izgradnjo rešitve pri uporabi sistemov ProGolemNRNot ter ProParGolemNRNot.

	ProGolemNRNot	ProParGolemNRNot	Razmerje
Krk_Illegal	149.31 ± 7.85	232.15 ± 10.70	0.64
Kr_vs_kp	350.18 ± 22.35	384.08 ± 25.52	0.91
Alzheimer-Acetyl	487.42 ± 28.98	442.57 ± 21.11	1.10
Alzheimer-Amine	322.47 ± 20.09	262.84 ± 17.97	1.23
Alzheimer-Mem	381.14 ± 29.66	297.31 ± 15.44	1.28
Alzheimer-Toxic	447.54 ± 24.85	432.36 ± 23.93	1.04
Carcinogenesis	3599.18 ± 265.84	2479.14 ± 191.9	1.45
Dspg-Astrophorida	386.20 ± 31.73	35.55 ± 1.07	10.86
Dspg-Axinellida	416.09 ± 18.09	42.20 ± 2.60	9.86
Dspg-Dictyoceratida	119.00 ± 8.54	8.70 ± 0.28	13.68
Dspg-Hadromerida	1077.02 ± 86.35	73.72 ± 3.70	14.61
Dspg-Halichondrida	451.67 ± 19.24	53.59 ± 2.70	8.43
Dspg-Haplosclerida	230.56 ± 15.91	22.00 ± 1.36	10.48
Dspg-Poecilosclerida	990.69 ± 109.19	76.22 ± 4.00	13.00
Dsstox	426.37 ± 41.91	339.90 ± 35.95	1.25
Metabolism	5511.04 ± 408.84	4337.63 ± 393.22	1.27
Mutagenesis	3921.63 ± 505.18	3262.39 ± 582.21	1.20
Pyrimidines	423.22 ± 24.57	401.22 ± 19.84	1.05

V namen pridobivanja primernih podatkov smo napravo simulirali kot množico n globinskih senzorjev, enakomerno razporejenih v obliki enakokrakega križa. Senzorji na kraku so med seboj oddaljeni d enot. Križ s senzorji se enakomerno vrti s frekvenco ν_k, medtem ko senzorji svoje podatke berejo s frekvenco ν_s. Senzor i ob branju pri času t odda svoje branje v obliki trojke $\langle x_{it}, y_{it}, z_{it}\rangle$, kjer sta x_{it} in y_{it} koordinati senzorja i ob času t v ravnini senzorjev, ter z prebrana vrednost – oddaljenost zadete točke.

Branja znotraj četrtinskega obrata kroga združimo v en sam zajem. Zajemi globine znotraj četrtine obrata projiciranega vzorca pokrijejo celotno površino kroga, orisanega s skrajnimi točkami vzorca. Zajem j je tako predstavljen kot množica $\frac{\nu_k}{4} \cdot \nu_s \cdot n$ točk, pri čemer se središčna točka (na koordinatah $(0, 0)$ ravnine senzorjev) pojavi večkrat. Zanjo obdržimo najnižjo prebrano vrednost (tj. vrednost, ko je bilo branje najbližje senzorjem).

Zaradi netočnosti izračunov globine na podlagi popačenja vzorca smo prebrane vrednosti globine pošumili z normalno porazdeljenim šumom s standardno deviacijo 1.7, pozicije senzorjev v njihovi ravnini pa z normalno porazdeljenim šumom s standardno



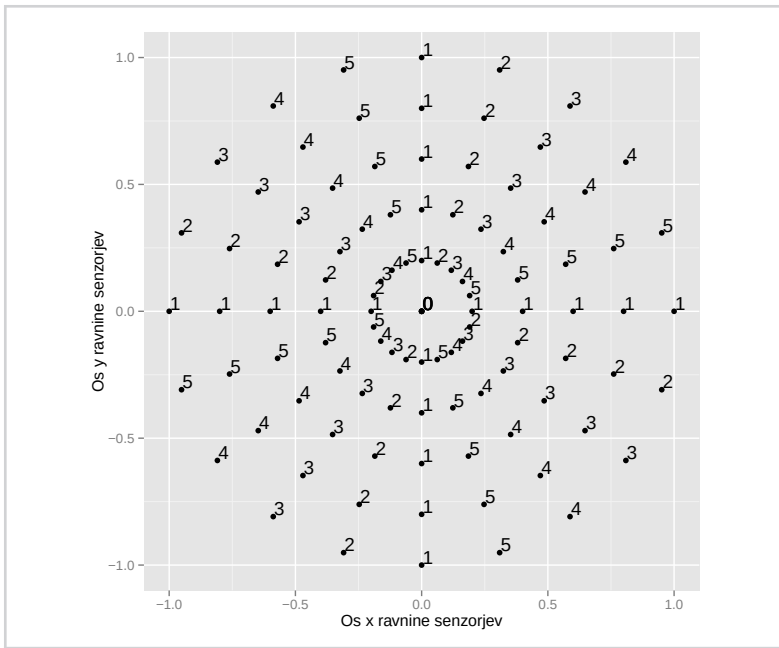
Slika 5.1

Prikaz uporabljene razporeditve in oštevilčenja senzorjev v njihovi ravnini.

deviacijo 0.1.

Simulirali smo projekcijo iz 21 točk, razporejenih v obliki enakokrakega križa (slika 5.1). Vsak krak sestoji iz petih točk, oddaljenih med seboj 0.2 enote. Frekvenca vrtenja ν_k je en obrat na sekundo, frekvenca zajema pa je enaka 20, torej se branje globine izvede vsake 0.05 sekunde. Število branj za konstrukcijo posameznega zajema (tj. število branj v četrtini obrata) je tako enaka $\frac{1}{4} \cdot 20 = 5$. Prikaz razporeditve točk pri posameznem zajemu je prikazan na sliki 5.2. Uporabljeni prizor vključuje dve krogli različnih radijev, ki gresta ena mimo druge. Končna podatkovna množica vsebuje štiri sekunde gibanja. Pri frekvenci branja vrednosti 20 to znese 80 branj vrednosti senzorjev oziroma $\frac{80}{5} = 16$ zajemov celotnega območja pod krogom.

Na podlagi tako zajetih podatkov smo se želeli naučiti razpoznavanja primitivnih konceptov v globinskih slikah prizora. Pri obravnavi in učenju iz podatkov, pridobljenih iz globinskih senzorjev, se navadno uporabljajo pristopi, izhajajoči iz področja razpoznave slik. Ti pristopi (z nekaterimi izjemami, denimo globokega učenja) pred-



Slika 5.2

Prikaz posameznega zajema, sestoječega iz vseh branj, opravljenih znotraj četrtinskega obrata senzorskega križa. Oštevilčenje točk odraža zaporedno številko branja znotraj četrtinskega obrata. Središčna točka je zajeta v vsakem branju, zato ji je prirejen ločen indeks branja 0.

hodno obdelajo slike na vhodu, s čimer iz slike izločijo značilke (npr. SIFT [33]), izvede segmentacija, prepoznavanje robov (npr. z uporabo Cannyevega algoritma [8]), ipd. Učenje se nato izvaja na podlagi tako izločenih značilk. Uporaba ILP nam omogoča, da se ne zanašamo na uporabo predhodno definiranih načinov izračuna značilk, temveč se tudi teh naučimo. Dodatno lahko z uporabo ILP neposredno pristopimo k učenju konceptov, ki zahtevajo upoštevanje zaporedja slik (denimo sledenje objektom) in ne zgolj posamezne slike.

5.3 Podano predznanje

Podatki, pridobljeni neposredno iz simuliranih senzorjev, so neprimerni za učenje s sistemi ILP, zato jih najprej pretvorimo v relacijsko obliko.

Podatki predstavimo kot instanciacije predikata `točka(T, S, X, Y, Z)`, kjer `T` podaja indeks zajema, `S` unikatno številko senzorja, `X` in `Y` njegove koordinate v ravnini senzorskega križa ob zajemu `T`, ter `Z` prebrano vrednost.

V predznanje dodamo predikat `naslednji(A, B)`, ki hrani sosledje branj v času. Ta predikat uspe, kadar `A` in `B` zaznamujeta zaporedna zajema podatkov.

Poleg samih prebranih točk hranimo tudi odnose sosednosti med senzorji. Senzorja sta lahko sosednja bodisi v prostoru ali v času: prostorsko sosednja senzorja ležita neposredno eden poleg drugega na istem kraku, časovno pa je vsak senzor soseden sam sebi – gre za branja, pridobljena iz istega senzorja v dveh zaporednih zajemih. Sosednost je podana s pomočjo predikata `sosed(S1, S2)`, ki uspe, če sta `S1` in `S2` sosedna senzorja.

Poleg teh predikatov, primarno namenjenih opisu vhodnih podatkov, smo v predznanju definirali še nekatere predikate, ki sistemu omogočajo sklepanje na osnovi preproste geometrije:

zvo(Z, c) uspe, kadar je vrednost spremenljivke `Z` večja od konstante `c`.

dvo(V1, V2) uspe, kadar je `V1` > `V2`.

deltaz(Z1, Z2, DZ) priredi razliko med `Z1` in `Z2` spremenljivki `DZ`.

razdalja(X1, Y1, X2, Y2, D) priredi spremenljivki `D` vrednost evklidske razdalje med točkama $(X1, Y1)$ in $(X2, Y2)$.

bliže(T1, S11, T2, S21, S22) uspe, če je razdalja med koordinatama (x, y) senzorja `S11` ob času `T1` in koordinatama (x, y) senorja `S21` ob času `T2` manjša, kot je razdalja med koordinatama senzorja `S11` ob času `T1` in koordinatama senzorja `S22` ob času `T2`.

5.4 Inkrementalno učenje osnovnih konceptov

K učenju konceptov iz opisanih podatkov smo pristopili na inkrementalen način. Najprej s sistemom ProParGolemNRNot zgradimo opis koncepta `rob`, ki opisuje pare točk, med katerimi leži rob predmeta v sceni. Koncept roba uporabimo pri učenju iskanja roba, ki je ob naslednjem zajemu podatkov podani točki najbliže. S pomočjo opisa tega koncepta se lahko naučimo preprostega opisa sledenja robovom med zaporednimi zajemi podatkov.

5.4.1 Učenje prepoznavanja robov

Predikat, ki se ga želimo naučiti, smo definirali kot `rob(A, B, C)`, kjer je `A` indeks zajema, `B` in `C` pa sta oznaki senzorjev, med katerima leži rob. Pri tem `B` vedno označuje točko, ki zadene bližji objekt. Iz simulatorja smo poleg točk za vsako izmed njih pridobili še podatek, na katerem objektu (ali ozadju) leži zadeta točka. Na podlagi tega podatka smo pozitivne primere generirali kot pare sosednjih točk, ki zadeneta različna objekta (oz. en zadene objekt in drugi ozadje). Na ta način smo generirali 218 pozitivnih primerov. Negativne primere predstavljajo instance, pri katerih obe točki ležita na istem objektu (ali ozadju), senzorja nista sosednja, ali pa prva točka leži dlje od druge. Izmed vseh možnih negativnih primerov smo naključno izbrali 2% takšnih, kjer točki nista sosednji, 5% tistih, kjer obe točki zadaneta isti objekt, ter 1% primerov, kjer prva točka sploh ne zadane objekta. Skupno število negativnih primerov znese 3607.

Če za učenje uporabimo celotno množico generiranih primerov, sistem najde sledečo rešitev:

```
rob(A, B, C) :-
   točka(A, C, D, E, F), točka(A, B, G, H, I), deltaz(F, I, J),
   zvo(J, 2), sosed(C, B).
```

V naravnem jeziku takšen opis ustreza iskanju sosednjih točk, katerih razlika v z koordinati je višja od neke konstante, v našem primeru 2. Naučena vrednost praga

je odvisna od konkretnega nabora učnih podatkov. Nanj vplivajo oblike objektov in njihova razporeditev v prostoru.

Najdena rešitev ne pokrije nobenega izmed 3607 negativnih primerov in pravilno najde rob v 216 izmed 218 pozitivnih primerov. Uspešnost učenja preverimo z uporabo desetkratnega prečnega preverjanja, pri katerem sistem doseže povprečno točnost 99.9% ± 0.1%.

Naučeno definicijo predikata `rob` dodamo v predznanje skupaj s pomožnim predikatom `robna_točka`, ki sistemu omogoča sklepati, ali leži podana točka ob robu objekta, ne da bi za to moral poznati tudi sosednjo točko, ki zadene drug objekt. Ta predikat je definiran kot

```
robna_točka(A, B) :-  rob(A, B, C).
```

5.4.2 Iskanje sledečega najbližjega roba

Naslednji koncept zajema iskanje najbližjega roba v naslednjem zajemu. Predikat, ki se ga želimo naučiti, je `naslednji_najblizji_rob(A, B, C)`, ki uspe, kadar so koordinate (x, y) senzorja C ob zajemu `A+1` najbližje koordinatam (x, y) senzorja `B` ob zajemu `A`.

Množico pozitivnih primerov smo generirali z naključnim vzorčenjem zajemov senzorjev, za katere smo nato v naslednjem zajemu poiskali njej najbližjo robno točko. Na ta način smo generirali 334 pozitivnih primerov. Negativne primere predstavljajo instance, kjer druga točka bodisi ni najbližja bodisi ni robna. Izmed obeh možnosti smo jih v množico negativnih primerov uvrstili 10%, kar nam da 311 negativnih primerov. Uporaba vseh primerov za učenje vrne sledečo definicijo:

```
naslednji_najblizji_rob(A, B, C) :-
  \+ negPred(A, B, C).

negPred(A, B, C) :-
  naslednji(A, D), rob(D, E, F), bliže(A, B, D, E, C).
```

V naravnem jeziku: za dani zajem `A` ter senzorja `B` in `C`, ne obstaja takšen senzor `E`, ki bi v naslednjem zajemu bil robna točka in bi ležal bliže, kot leži `C`.

Najdeni opis je konsistenten (ne pokrije nobenega izmed 311 negativnih primerov) in popoln (pokrije vseh 334 pozitivnih primerov). Z uporabo desetkratnega prečnega

preverjanja ugotovimo, da sistem občasno zgreši takšen opis – povprečna točnost namreč znaša 92.7% ± 8.8%. Kljub temu je takšna točnost velik napredek v primerjavi z osnovno točnostjo (56.7% ± 4.9%).

Naučeni predikat dodamo sistemu na razpolago kot del predznanja.

5.4.3 *Odkrivanje premikov*

Z na ta način obogatenim predznanjem se poskusimo naučiti sledenja robovom. Cilj je odkriti korespondenco med robnimi točkami v dveh zaporednih zajemih. Korespondenco opišemo s predikatom `premik(A, B, C)`, kjer je `B` senzor, ki je zadel nek rob v zajemu `A`, `C` pa je indeks senzorja, ki zadane isti rob v zajemu, ki sledi `A`.

Pozitivne primere smo generirali z vzorčenjem robnih točk. Njihovo ustreznico na naslednjem zajemu smo definirali kot najbližjo robno točko v njem, ki zadene isti objekt. Takšnih instanc je v naših podatkih 1002. Negativni primeri so instance, kjer točke niso robne, točka pod `C` zadene drugi objekt ali pa ni najbližji zadetek. Naključno smo izbrali 10% takšnih instanc in dobili 710 negativnih primerov za učenje.

Takšno generiranje primerov temelji na predpostavki, da je hitrost zajemanja bistveno večja od hitrosti gibanja objektov v sceni – v tem primeru so namreč njihovi premiki med zajemi majhni.

Pri uporabi celotne množice za učenje ProParGolemNRNot sestavi sledečo definicijo predikata:

```
premik(A, B, C) :-
  robna_točka(A, B), naslednji(A, D), robna_točka(D, C),
  naslednji_najblizji_rob(A, B, C).
```

Ta definicija pokrije vse (1002) pozitivne primere in nobenega izmed 710 negativnih. Sistem je pravilno odkril naš način generiranja primerov – išče najbližjo robno točko v naslednjem zajemu.

Pri uporabi desetkratnega prečnega preverjanja sistem doseže povprečno točnost 99.9% ± 0.5%.

Naučeni predikat je mogoče uporabiti za aproksimacijo gibanja objektov v sceni. S poznavanjem frekvence zajema podatkov lahko izračunamo ne le približno smer njihovega gibanja, temveč tudi njihovo hitrost. Te aproksimacije lahko tvorijo podlago za analitični izračun prihodnje lokacije objektov in s tem odkrivanje morebitnih opazovalcu nevarnih objektov.

5.5 Omejitve pristopa

Z uporabo inkrementalnega pristopa smo pokazali, da je z uporabo sistema ProPar-GolemNRNot mogoče pridobiti razumljive opise primitivnih konceptov iz globinskih podatkov, ki so intuitivno smiselni. Vsekakor pa gre pri problemu, kakršnega smo reševali v tem poglavju, za precejšnjo poenostavitev v primerjavi s podatki, pridobljenimi v realnem okolju. Število objetov v simulaciji je izjemno nizko in so vsi – čeprav različnih velikosti – enake oblike. Druga poenostavitev leži v samih zaznavah, pridobljenih iz simuliranih senzorjev. V simulaciji smo predpostavili, da so senzorji med seboj popolnoma vzporedni, zaradi česar se ločljivost zajetih podatkov z razdaljo ne zmanjšuje in razdalje v ravnini zajema neposredno odgovarjajo razdaljam med zadetimi točkami. Pri realni napravi takšna korespondenca ni prisotna, saj žarki med seboj niso popolnoma vzporedni, temveč se projicirajo v obliki stožca. S povečevanjem razdalje od ravnine zajema se povečujejo tudi razdalje med žarki.

Drugo pomanjkljivost bi lahko deloma odpravili z uporabo dodatnega predznanja, s katerim bi na podlagi oddaljenosti zadete točke lahko izračunali njene prave koordinate v ravnini senzorjev, medtem ko bi za odpravo prve poenostavitve potrebovali precej večje število učnih primerov s spremenljivim številom objektov različnih oblik v njih.

6

Zaključek in nadaljnje delo

V delu smo opisali dva prispevka k področju induktivnega logičnega programiranja: prenos CWS v proces indukcije in uporabo dvojnih nasičenj. Uporabnost pristopov smo prikazali z njihovo uporabo na problemu učenja konceptov iz podatkov, pridobljenih s simulacijo globinskih senzorjev.

6.1 Specializacija zaprtega sveta kot del indukcije

Najprej smo naslovili problematiko obstoječih pristopov k omogočanju uporabe negacije v hipotezah. Obstoječi pristopi imajo namreč eno izmed treh pomanjkljivosti. V prvem sklopu najdemo pristope, ki za svoje delovanje terjajo izračun celotne množice neveljavnih dejstev, za katero smo pokazali, da vodi k eksponentni rasti velikosti predznanja, zaradi česar je v večjih domenah časovno in prostorsko neprimerna. Za uporabo drugih pristopov je potrebno ekspertno predznanje, ki sistem usmerja z eksplicitnim zapisom predikatov, katerih negacija je lahko zanimiva. Za specializacijo zaprtega sveta (ang. closed world specialisation, CWS) – pristop, ki ga nadgrajujemo – pa smo v delu pokazali, da deluje pod predpostavko, da je najboljšo hipotezo mogoče zgraditi v dveh korakih: najprej se zgradi kar najboljša hipoteza zgolj z uporabo pozitivnih literalov, nakar se jo dodatno specializira z uporabo specializacije zaprtega sveta.

V svojem delu to problematiko naslovimo s premikom izvajanja specializacije zaprtega sveta pri sistemih, ki delujejo po principu od spodaj navzgor, v sam proces indukcije, takoj za izvedbo posplošitve semen. S tem se ne le izognemo predpostavki o kvaliteti pozitivnega dela hipoteze, temveč tudi uporabljamo negacije le za odstranjevanje tistih primerov iz pokritja, ki jih na drugačen način ni mogoče izločiti. Takšen način uporabe specializacije zaprtega sveta smo implementirali kot sistem ProGolemNot, ki temelji na sistemu ProGolem.Za takšen pristop se izkaže, da dosega vsaj tako dobre rezultate glede dosežene točnosti, kot klasična uporaba specializacije, na nekaterih podatkovnih množicah pa tudi značilno boljše.

V naslednjem koraku izpostavimo znatno upočasnitev delovanja sistema po izvedbi premika specializacije zaprtega sveta v proces indukcije. Postopek pohitrimo z zamikom izvedbe specializacije do zaključka preiskovanja v snopu. Nato specializacijo izvedemo nad celotnim snopom hipotez, pri čemer iz pokritja še vedno odstranjujemo zgolj primere, ki jih drugače ni mogoče odstraniti iz pokritja, in v teorijo dodamo najboljšo izmed njih. Na ta način delovanje sistema bistveno pohitrimo, vendar ponovno vnesemo pristranskost, saj takšen pristop favorizira hipoteze z boljšim pozitivnim

delom. Tako pohitreno verzijo sistema ProGolemNot smo implementirali v sistemu, poimenovanem ProGolemNRNot.

Eksperimenti pokažejo, da apliciranje specializacije na snopu izvajanje pohitri za faktor 2 v primerjavi s specializiranjem vseh hipotez, pri čemer se dosežene točnosti ne razlikujejo. Izkaže se, da premik izvajanja CWS na nekaterih domenah privede do izboljšanja točnosti za približno 4% in na nobeni domeni točnosti ne poslabša. Pristopa smo primerjali tudi s sistemom Aleph in sistemom Aleph z uporabo CWS. Rezultati eksperimentov kažejo, da na nekaterih domenah uporaba metode od zgoraj navzdol prinaša boljše rezultate, na drugih pa slabše. Tako se predlagani pristopi bolje odrežejo na petih izmed devetih podatkovnih množic, na preostalih štirih pa je pristop od zgoraj navzdol primernejši.

6.2 Dvojna nasičenja

V drugem delu se neposredno ukvarjamo s pohitritvijo sistemov, ki pri izpeljavi uporabljajo asimetrično relativno minimalno posplošitev (ang. asymmetrical relative minimal generalisation, ARMG). Pri uporabi ARMG smo pokazali, da je mogoče prostor, v katerem iskana posplošitev leži, zmanjšati z uporabo omejitev nad literali, prisotnimi v trenutni spodnji meji prostora (pri izračunu $ARMG(e_2|e_1)$ to spodnjo mejo predstavlja spodnji stavek primera e_1). Literale, ki ne zadostijo tem omejitvam, lahko varno odstranimo in spodnjo mejo približamo pravi rešitvi brez izgube kvalitete izhodne hipoteze. Spodnji stavek, dobljen po odstranitvi teh literalov, imenujemo dvojno nasičenje. Razvito metodo smo implementirali kot nadgradjno sistema ProGolem, poimenovano ProParGolem. ProParGolem uporablja dvojna nasičenja za pohitritev izračuna posplošitve dveh primerov. Sistem smo razširili še s sposobnostjo predpomnenja spodnjih stavkov. Spodnji stavki se tako vnaprej izračunajo že ob branju problema, kar dodatno pohitri izvajanje. Eksperimentalno preverjanje pokaže, da lahko uporaba dvojnih nasičenj privede tudi do pohitritve izvajanja za faktor 1.44, pri čemer pa je dejanska pohitritev močno odvisna od problema, ki ga sistem ILP rešuje. Uvedli smo tudi posplošitev dvojnih nasičenj na poljubno velike množice primerov, s čimer dobimo n-terna nasičenja. Eksperimenti pričakovano pokažejo, da z večanjem množice primerov, uporabljenih pri izračunu n-ternega nasičenja, faktor pohitritve narašča ob hkratnem slabšanju kvalitete izhodne teorije.

6.3 Nadaljnje delo

Pričujoče delo odpira več smernic za nadaljnje delo. Postavlja se vprašanje o uporabi dvojnih nasičenj pri sistemih, ki delujejo po principu od zgoraj navzdol. Opisani pristop lahko neposredno uporabimo pri sistemih, ki hipotezo gradijo relativno glede na spodnje stavke primerov in ne uporabljajo deljenja spremenljivk. V takšnih primerih imamo namreč opravka z enako pristranskostjo jezika hipotez, kot jo imamo pri sistemu ProGolem. Kuželka in sodelavci [28] so že pokazali, da uporaba okrajšanih spodnjih stavkov privede do znatnih pohitritev sistemov ILP Aleph [53] ter nFOIL [31].

V podpoglavju 2.1.2 smo izpostavili omejitve, s katerimi se soočajo sistemi, ki hipotezo gradijo relativno glede na spodnji stavek, zaradi nezmožnosti učinkovite uporabe deljenja spremenljivk. To pomanjkljivost si z njimi deli tudi sistem ProGolem. Postavlja se vprašanje, če je mogoče pri gradnji dvojnih nasičenj identificirati smiselne kandidatne vrednosti za deljenje spremenljivk – potrebo po deljenju bi lahko zaznali z uporabo primerjave množic instanciacij posameznih vrednosti.

Tretje vprašanje, ki se postavlja ob zaključku, je možnost uporabe metode, opisane v razdelku o dvojnih nasičenjih (poglavje 4), pri računanju asimetrične najmanj splošne posplošitve ne le pri izračunu posplošitve spodnjih stavkov dveh primerov, temveč tudi posplošitve že zgrajenega stavka z uporabo novega primera. Prilagoditev podane metode na takšne primere je sama po sebi očitna, postavlja pa se vprašanje učinkovitosti takšnih izračunov, saj imamo v primeru izračuna posplošitve dveh primerov opravka s statičnimi množicami instanciacij in spodnjimi stavki, pri uporabi opisane tehnike v primeru posploševanja stavka pa bi bilo treba potrebne strukture v vsaki iteraciji algoritma posodabljati.

Pri izračunu dvojnih nasičenj smo kot semena uporabljali naključno izbrane primere. Ker izbor semen močno vpliva na stavek, ki služi za osnovo nadaljnjih posplošitev, bi z vodenim izborom semen lahko postopek dodatno pohitrili. V ta namen bi se lahko zatekli k izbiranju primerov na podlagi njihove podobnosti oz. podobnosti njihovih nasičenj. Pri tem se postavlja tako vprašanje same definicije podobnosti, kot tudi smotrne izbire primerov. Izbira najbolj podobnih primerov bi zmanjšal vpliv uporabe dvojnih nasičenj, prevelika medsebojna oddaljenost semen pa bi lahko imela negativen vpliv na kvaliteto dobljenih rešitev. Na področju izračuna podobnosti oz. razdalje med primeri lahko uporabimo enega izmed več pristopov, ki omogočajo primerjanje

stavkov. Nekateri pristopi k izračunu razdalje primere najprej preslikajo v atributni prostor z uporabo propozicionalizacije [27] (npr. [1]), drugi pa delujejo neposredno nad stavki [25, 48].

Pri eksperimentalnem primerjanju sistemov ProGolemNot ter ProGolemNRNot s sistemoma Aleph ter AlephCWS (Aleph s specializacijo zaprtega sveta) smo ugotovili, da je pri reševanju nekaterih problemov pristop od zgoraj navzdol primernejši od pristopa od spodaj navzgor. Odprto ostaja vprašanje, katere značilnosti problema ali oblike rešitev so odločilnega pomena za to razliko. Odgovor na to vprašanje bi hkrati lahko usmeril nadaljnje raziskave, saj bi izpostavil vrsto problemov, s katerimi ima določen tip sistemov težave.

Pri obravnavi specializacije zaprtega sveta smo omenili, da ta ne omogoča uporabe spremenljivk, instanciranih v pozitivnem delu hipoteze. Ta nezmožnost se lahko kaže kot huda omejitev – npr. pri umetnih primerih z vlaki v tretjem poglavju smo se morali omejiti na vlake z enim samim vagonom. Nezmožnost navezovanja na spremenljivke iz pozitivnega dela izhaja iz načina delovanja, saj je učenje predikata izjem (tj. negiranega dela hipoteza) povsem samostojna učna naloga in poteka brez vednosti o že sestavljenem pozitivnem delu hipoteze. Trivialna, a neučinkovita rešitev je dodajanje vseh spremenljivk, uvedenih v telesu trenutne hipoteze, kot argumentov predikatu izjem. Pri takšni uporabi je potrebna pazljivost pri konstruiranju pozitivnih in negativnih primerov, uporabljenih za učenje predikata izjem – množici morata namreč vsebovati *vse* instanciacije dodatnih argumentov, za katere pozitivni del hipoteze uspe. S stališča učinkovitosti delovanja je takšen pristop neprimeren. Smiselno bi bilo preiskati različne možnosti, kako omejiti nabor spremenljivk, ki jih damo sistemu na razpolago pri učenju predikata izjem, denimo z uporabo različnih hevristik.

Uporabo razvitih metod na podatkih, pridobljenih iz simuliranih globinskih senzorjev, bi bilo mogoče nadaljevati z učenjem nadaljnjih konceptov. Korespondenca med robovi skozi čas nudi osnovo za učenje izračuna smeri in hitrosti gibanja posameznih objektov v sceni, kar naprej nudi priložnost za učenje detekcije opazovalcu nevarnih objektov. Drugo nadaljnje delo zaobsega učenje in testiranje na kompleksnejših prizorih – prizorih, ki vsebujejo večje število objektov različnih oblik in velikosti, katerih gibanje ni enakomerno.

LITERATURA

[1] Grant Anderson and Bernhard Pfahringer. Clustering relational data based on randomized propositionalization. In Hendrik Blockeel, Jan Ramon, Jude W. Shavlik, and Prasad Tadepalli, editors, *Inductive Logic Programming, 17th International Conference, ILP 2007, Corvallis, OR, USA, June 19-21, 2007, Revised Selected Papers*, volume 4894 of *Lecture Notes in Computer Science*, pages 39–48. Springer, 2007.

[2] Marta Arias, Roni Khardon, and Jérôme Maloberti. Learning horn expressions with LOGAN-H. *Journal of Machine Learning Research*, 8:549–587, 2007.

[3] M Bain and S Muggleton. Non-Monotonic Learning. In S Muggleton, editor, *Inductive Logic Programming*, pages 145–161. Academic Press, 1992.

[4] Michael Bain and Stephen Muggleton. Learning optimal chess strategies. In *Machine Intelligence 13*, pages 291–309, 1994.

[5] F. Bergadano, D. Gunetti, M. Nicosia, and G. Ruffo. Learning Logic Programs with Negation as Failure. In Luc de Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 33–52. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[6] Hendrik Blockeel, Luc Dehaspe, Bart Demoen, Gerda Janssens, Jan Ramon, and Henk Vandecasteele. Improving the efficiency of inductive logic programming through the use of query packs. *J. Artif. Intell. Res. (JAIR)*, 16:135–166, 2002.

[7] Ivan Bratko, Stephen Muggleton, and Alen Varsek. Learning qualitative models of dynamic systems. In Lawrence Birnbaum and Gregg Collins, editors, *Proceedings of the Eighth International Workshop (ML91), Northwestern University, Evanston, Illinois, USA*, pages 385–388. Morgan Kaufmann, 1991.

[8] John Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, June 1986.

[9] Jianzhong Chen, Stephen Muggleton, and José Carlos Almeida Santos. Learning probabilistic logic models from probabilistic examples. *Machine Learning*, 73(1):55–85, 2008.

[10] Luc Dehaspe and Luc De Raedt. Mining association rules in multiple relations. In Nada Lavrač and Sašo Džeroski, editors, *Inductive Logic Programming, 7th International Workshop, ILP-97, Prague, Czech Republic, September 17-20, 1997, Proceedings*, volume 1297 of *Lecture Notes in Computer Science*, pages 125–132. Springer, 1997.

[11] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[12] B. Dolšak and S. Muggleton. The application of inductive logic programming to finite-element mesh design. In S. Muggleton, editor, *Inductive Logic Programming*, pages 453–472. Academic Press, London, 1992.

[13] Miha Drole and Igor Kononenko. Closed world specialisation inside the induction process. *Intelligent Data Analysis*, 20(4), 2016.

[14] Miha Drole and Igor Kononenko. Pairwise saturations in inductive logic programming. *Submitted to Artificial Intelligence Review*, 2016.

[15] Miha Drole, Petar Vračar, Ivo Stančić, Josip Musić, Ante Panjkota, Igor Kononenko, and Matjaž Kukar. Learning from depth sensor data using inductive logic programming. In *XXV International Symposium on Information, Communication and Automation Technologies, ICAT 2015, Sarajevo, Bosnia and Herzegovina, October 29 - October 31, 2015*, pages 23–28. IEEE Computer Society, 2015.

[16] Mitsue Furusawa, Nobuhiro Inuzuka, Hirohisa Seki, and Hidenori Itoh. Induction of logic programs with more than one recursive clause by analyzing saturations. In *Proceedings of the 7th International Workshop on Inductive Logic Programming (ILP'97)*, volume 1297 of *LNCS*, pages 165–172. Springer-Verlag, 1997.

[17] Esther Galbrun and Angelika Kimmig. Finding relational redescriptions. *Machine Learning*, 96(3):225–248, 2014.

[18] Jonathan D. Hirst, Ross D. King, and Michael J. E. Sternberg. Quantitative structure-activity relationships by neural networks and inductive logic programming. i. the inhibition of dihydrofolate reductase by pyrimidines. *Journal of Computer-Aided Molecular Design*, 8(4):405–420, 1994.

[19] Jonathan D. Hirst, Ross D. King, and Michael J. E. Sternberg. Quantitative structure-activity relationships by neural networks and inductive logic programming. II. the inhibition of dihydrofolate reductase by triazines. *Journal of Computer-Aided Molecular Design*, 8(4):421–432, 1994.

[20] Peter Idestam-Almquist. Efficient induction of recursive definitions by structural analysis of saturations. In Luc De Raedt, editor, *Advances in Inductive Logic Programming*. IOS Press, 1996.

[21] Katsumi Inoue and Yoshimitsu Kudoh. Learning extended logic programs. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*, pages 176–181. Morgan Kaufmann, 1997.

[22] Nobuhiro Inuzuka, Jun-ichi Motoyama, and Tomofumi Nakano. Relational association mining based on structural analysis of saturation clauses. In Bogdan Gabrys, Robert J. Howlett, and Lakhmi C. Jain, editors, *Knowledge-Based Intelligent Information and Engineering Systems, 10th International Conference, KES 2006, Bournemouth, UK, October 9-11, 2006, Proceedings, Part II*, volume 4252 of *Lecture Notes in Computer Science*, pages 1162–1169. Springer, 2006.

[23] Deepak Kapur and Paliath Narendran. Np-completeness of the set unification and matching problems. In Jörg H. Siekmann, editor, *8th International Conference on Automated Deduction, Oxford, England, July 27 - August 1, 1986, Proceedings*, volume 230 of *Lecture Notes in Computer Science*, pages 489–495. Springer, 1986.

[24] Ross D. King, Jem Rowland, Stephen G. Oliver, Michael Young, Wayne Aubrey, Emma Byrne, Maria Liakata, Magdalena Markham, Pinar Pir, Larisa N. Soldatova, Andrew Sparkes, Kenneth E. Whelan, and Amanda Clare. The automation of science. *Science*, 324(5923):85–89, 2009.

[25] Mathias Kirsten, Stefan Wrabel, and Tamá Horváth. Distance based approaches to relational learning and clustering. In Sašo Džeroski, editor, *Relational Data Mining*, pages 213–230. Springer-Verlag New York, Inc., New York, NY, USA, 2000.

[26] I Kononenko, M Robnik-Šikonja, and U Pompe. ReliefF for estimation and discretization of attributes in classification, regression and ILP problems. In A. Ramsay, editor, *Artificial Intelligence: Methodology, Systems, Applications: Proceddings of AIMSA'96*, pages 31–40. IOS Press, 1996.

[27] Stefan Kramer, Nada Lavrač, and Peter Flach. Propositionalization approaches to relational data mining. In Sašo Džeroski, editor, *Relational Data Mining*, pages 262–286. Springer-Verlag New York, Inc., New York, NY, USA, 2000.

[28] Ondrej Kuželka, Andrea Szabóová, and Filip Železný. Bounded least general generalization. In Fabrizio Riguzzi and Filip Železný, editors, *Inductive Logic Programming - 22nd International Conference, ILP 2012, Dubrovnik, Croatia, September 17-19, 2012, Revised Selected Papers*, volume 7842 of *Lecture Notes in Computer Science*, pages 116–129. Springer, 2012.

[29] Ondrej Kuželka, Andrea Szabóová, and Filip Železný. A method for reduction of examples in relational learning. *J. Intell. Inf. Syst.*, 42(2):255–281, 2014.

[30] Ondrej Kuželka and Filip Zelezný. A restarted strategy for efficient subsumption testing. *Fundam. Inform.*, 89(1):95–109, 2008.

[31] Niels Landwehr, Kristian Kersting, and Luc De Raedt. Integrating naïve bayes and FOIL. *Journal of Machine Learning Research*, 8:481–507, 2007.

[32] Nada Lavrač, Dragan Gamberger, Hendrik Blockeel, and Ljupčo Todorovski, editors. *Knowledge Discovery in Databases: PKDD 2003, 7th European Conference on Principles and Practice of Knowledge Discovery in Databases, Cavtat-Dubrovnik, Croatia, September 22-26, 2003, Proceedings*, volume 2838 of *Lecture Notes in Computer Science*. Springer, 2003.

[33] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, Washington, DC, USA, 1999. IEEE Computer Society.

[34] Jérôme Maloberti and Michèle Sebag. Fast theta-subsumption with constraint satisfaction algorithms. *Machine Learning*, 55(2):137–174, 2004.

[35] Raymond J. Mooney and Mary Elaine Califf. Induction of first-order decision lists: Results on learning the past tense of english verbs. *J. Artif. Intell. Res. (JAIR)*, 3:1–24, 1995.

[36] S. Muggleton. Inverse Entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[37] S. Muggleton and C. Feng. Efficient induction of logic programs. In S. Muggleton, editor, *Inductive Logic Programming*, pages 281–298. Academic Press, London, 1992.

[38] Stephen Muggleton, Luc De Raedt, David Poole, Ivan Bratko, Peter A. Flach, Katsumi Inoue, and Ashwin Srinivasan. ILP turns 20 - biography and future challenges. *Machine Learning*, 86(1):3–23, 2012.

[39] Stephen Muggleton, José Carlos Almeida Santos, and Alireza Tamaddoni-Nezhad. Progolem: A system based on relative minimal generalisation. In Luc De Raedt, editor, *Inductive Logic Programming, 19th International Conference, ILP 2009, Leuven, Belgium, July 02-04, 2009. Revised Papers*, volume 5989 of *Lecture Notes in Computer Science*, pages 131–148. Springer, 2009.

[40] Lavrač N., S. Džeroski, and I. Bratko. Handling imperfect data in inductive logic programming. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 48–64. IOS Press, Amsterdam, 1996.

[41] S. Natarajan, G. Kunapuli, C. O'Reilly, R.Maclin, T. Walker, D. Page, and J.Shavlik. ILP for bootstrapped learning: A layered approach to automating the ilp setup problem. In *Presented at the Nineteenth Conference on Inductive Logic Programming*, Leuven, Belgium, 2009.

[42] Siegfried Nijssen and Joost N. Kok. Faster association rules for multiple relations. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, pages 891–896. Morgan Kaufmann, 2001.

[43] Irene M. Ong, Inês de Castro Dutra, David Page, and Vítor Santos Costa. Mode directed path finding. In João Gama, Rui Camacho, Pavel Brazdil, Alípio Jorge, and Luís Torgo, editors, *Machine Learning: ECML 2005, 16th European Conference on Machine Learning, Porto, Portugal, October 3-7, 2005, Proceedings*, volume 3720 of *Lecture Notes in Computer Science*, pages 673–681. Springer, 2005.

[44] G. D. Plotkin. A further note on inductive generalization, 1971.

[45] J.R. Quinlan and R.M. Cameron-Jones. Foil: A midterm report. In PavelB. Brazdil, editor, *Machine Learning: ECML-93*, volume 667 of *Lecture Notes in Computer Science*, pages 1–20. Springer Berlin Heidelberg, 1993.

[46] Luc De Raedt and Maurice Bruynooghe. On negation and three-valued logic in interactive concept-learning. In *ECAI*, pages 207–212, 1990.

[47] Luc De Raedt and Wim Van Laer. Inductive constraint logic. In Klaus P. Jantke, Takeshi Shinohara, and Thomas Zeugmann, editors, *Algorithmic Learning Theory, 6th International Conference, ALT '95, Fukuoka, Japan, October 18-20, 1995, Proceedings*, volume 997 of *Lecture Notes in Computer Science*, pages 80–94. Springer, 1995.

[48] Jan Ramon and Maurice Bruynooghe. A framework for defining distances between first-order logic objects. In David Page, editor, *Inductive Logic Programming, 8th International Workshop, ILP-98, Madison, Wisconsin, USA, July 22-24, 1998, Proceedings*, volume 1446 of *Lecture Notes in Computer Science*, pages 271–280. Springer, 1998.

[49] Bradley L. Richards and Raymond J. Mooney. Learning relations by pathfinding. In William R. Swartout, editor, *Proceedings of the 10th National Conference on Artificial Intelligence. San Jose, CA, July 12-16, 1992.*, pages 50–55. AAAI Press / The MIT Press, 1992.

[50] Jose Santos and Stephen Muggleton. Subsumer: A prolog theta-subsumption engine. In Manuel V. Hermenegildo and Torsten Schaub, editors, *Technical Communications of the 26th International Conference on Logic Programming, ICLP 2010, July 16-19, 2010, Edinburgh, Scotland, UK*, volume 7 of *LIPIcs*, pages 172–181. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[51] Jose Santos and Stephen Muggleton. When does it pay off to use sophisticated entailment engines in ilp? In Paolo Frasconi and Francesca A. Lisi, editors, *Inductive Logic Programming - 20th International Conference, ILP 2010, Florence, Italy, June 27-30, 2010. Revised Papers*, volume 6489 of *Lecture Notes in Computer Science*, pages 214–221. Springer, 2010.

[52] Jennifer Seitzer, James P. Buckley, and Yi Pan. INDED: A distributed knowledge-based learning system. *IEEE Intelligent Systems*, 15(5):38–46, 2000.

[53] A. Srinivasan. *The Aleph Manual*, 2004. http://www.comlab.ox.ac.uk/activities/machinelearning/Aleph/.

[54] A Srinivasan, S Muggleton, and M Bain. Distinguishing exceptions from noise in non-monotonic learning. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, pages 97–107. ICOT, 1992.

[55] Alireza Tamaddoni-Nezhad and Stephen Muggleton. The lattice structure and refinement operators for the hypothesis space bounded by a bottom clause. *Machine Learning*, 76(1):37–72, 2009.

[56] Tuan Nam Tran, Kenji Satou, and Tu Bao Ho. Using inductive logic programming for predicting protein-protein interactions from multiple genomic data. In Alípio Jorge, Luís Torgo, Pavel Brazdil, Rui Camacho, and João Gama, editors, *Knowledge Discovery in Databases: PKDD 2005, 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, Porto, Portugal, October 3-7, 2005, Proceedings*, volume 3721 of *Lecture Notes in Computer Science*, pages 321–330. Springer, 2005.

[57] Marcel Turcotte, Stephen Muggleton, and Michael J. E. Sternberg. Application of inductive logic programming to discover rules governing the three-dimensional topology of protein structure. In David Page, editor, *Inductive Logic Programming, 8th International Workshop, ILP-98, Madison, Wisconsin, USA, July 22-24, 1998, Proceedings*, volume 1446 of *Lecture Notes in Computer Science*, pages 53–64. Springer, 1998.