Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Aljaž Košmerlj

Avtonomno modeliranje robotskih akcij z odkrivanjem abstraktnih konceptov

doktorska disertacija

Ljubljana 2013

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Aljaž Košmerlj

Avtonomno modeliranje robotskih akcij z odkrivanjem abstraktnih konceptov

doktorska disertacija

Mentor: akad. prof. dr. Ivan Bratko

Ljubljana 2013

Povzetek

V disertaciji predstavimo nov pristop za avtonomno učenje robotskih modelov akcij oblike STRIPS, ki vsebujejo novoodkrite abstraktne koncepte. Naučeni koncepti so abstraktni, če niso eksplicitno izraženi v učnih podatkih (npr. premičnost, "nižje kot", obteženost ipd.).

Razvili smo metodo STRUDEL, ki z deljenjem robotskih akcij v skupine glede na podobnosti njihovih učinkov odkriva abstraktne koncepte. Abstraktni koncepti so izraženi kot pogoji, da ima posamezna akcija nek učinek. Z uporabo odkritih konceptov metoda STRUDEL nato zgradi končni model. Tako sam model akcij kot odkriti koncepti v njem so lahko izraženi rekurzivno.

Delitev v skupine STRUDEL opravi z novim algoritmom za razvrščanje relacijskih podatkov v skupine ACES, ki smo ga razvili v ta namen. Algoritem ACES razvršča logične opise učinkov akcij na podlagi strukture relacij v njih, natančneje na ujemanju grafov, ki jih priredimo literalom iz opisa učinka. Delovanje metod STRUDEL in ACES predstavimo s poskusi v več domenah.

Na koncu predstavimo še nov sistem induktivnega logičnega programiranja (ILP) HYPER/CA, razvit kot nadgradnja sistema HYPER, ki za razliko od slednjega deluje s šumnimi podatki. Sistem HYPER/CA smo razvili za učenje konceptov in modelov v metodi STRUDEL. Njegovo delovanje primerjamo z najnovejšimi sistemi ILP na več tipičnih učnih problemih ILP. Rezultati kažejo, da je sistem HYPER/CA sicer precej počasnejši od primerjanih algoritmov, a na testnih učnih problemih dosega primerljive točnosti in gradi manjše hipoteze.

Ključne besede: kognitivna robotika, strojno učenje, odkrivanje konceptov, induktivno logično programiranje, relacijsko učenje

Abstract

The thesis presents a novel approach to autonomous learning of STRIPS-like robot action models that contain newly discovered abstract concepts. A learnt concept is abstract if it is not explicitly expressed in the learning data (eg. movability, "lower than", weighted, etc.).

We developed STRUDEL, a method that clusters robot actions into groups and then discovers abstract concepts as conditions that classify actions to their respective effects. The definitions of these concepts are used to build the final action model. Both the action model and the concept definitions can be expressed recursively.

A new clustering algorithm ACES is used for clustering in STRUDEL. ACES clusters logical action descriptions by similarity of ther structure. Specifically it uses matching of graphs constructed from literals in the action effect descriptions as a distance measure. Performance of STRUDEL and ACES are demonstrated on several experimental domains.

Finally, we describe HYPER/CA, a new inductive logic programming (ILP) system, developed as an upgrade of the HYPER system. HYPER/CA was developed to handle learning from noisy data in STRUDEL. We compare it with state-of-the-art ILP systems on several typical ILP learning tasks. Results on the test problems show that HYPER/CA, though quite slower then the algorithms used in the comparison, can attain similar accuracy while building smaller hypotheses.

Keywords: cognitive robotics, machine learning, concept discovery, inductive logic programming, relational learning

Zahvala

Veliko ljudi je tako ali drugače pripomoglo k nastanku moje disertacije. Prva zahvala gre mojemu mentorju Ivanu Bratku, ki mi je omogočil, da sem po diplomi kot raziskovalec v njegovem laboratoriju ostal na fakulteti in opravljal doktorski študij. Delo z njim nikoli ni bilo dolgočasno in vedno znova me je presenečal z novimi idejami in vidiki, ki so moje delo vodili v zanimive smeri, ki jih sam najbrž ne bi videl. Od njega sem se naučil, kako svoje delo predstaviti jasno in suvereno, kar je med drugim bistveno vplivalo na moje pisanje te disertacije.

Laboratorij za umetno inteligenco na Fakulteti za računalništvo in informatiko, kjer sem delal tekom svojega doktorskega študija, je zelo sproščeno in dinamično delovno okolje. Na svoje tamkajšnje sodelavce sem vedno lahko računal, če je šlo za strokovno vprašanje, pomoč pri krmarjenju skozi motne vode fakultetne birokracije ali pa samo pogovor med pavzo. Jure Žabkar mi je izredno pomagal pri prvih raziskovalnih korakih v okviru projekta XPERO in pri pisanju in objavi prvega članka. Tadej Janež, s katerim sva delila študijsko pot že od srednje šole naprej, je bil zanesljiv kamerad na skupni poti do doktorata. Od Aleksandra Sadikova sem se učil pedagoških spretnosti, kot tudi neumorno iskrivega navdušenja nad raziskovanjem. Vida Groznik pa je vedno imela odgovore na vsa organizacijska vprašanja. Zahvala gre še mnogim drugim tako aktualnim kot nekdanjim sodelavcem kot so: Martin Možina, Matej Guid, Blaž Strle, Gregor Leban, Sašo Moškon, Simon Kozina, Timotej Lazar.

Profesorja Claude Sammut in Eduardo F. Morales sta s komentarji tekom svojih obiskov v laboratoriju nudila koristne uvide v moje delo in pomagala, da nisem pozabil na širšo sliko. Svoje raziskovalno delo sem začel v okviru evropskega projekta XPERO, ki ga je koordiniral profesor Erwin Prassler. Nenazadnje gre zahvala tudi članoma moje doktorske komisije Matjažu Kukarju in Bogdanu Filipiču za njun čas in komentarje pri branju moje disertacije.

Imel sem srečo, da sem tekom dodiplomskega in podiplomskega študija študiral v skupini odličnih kolegov, ki so nudili izzivalno in prizadevno okolje, v katerem sem lahko rasel kot računalničar. Če izpostavim le najtesnejše, so to (po abecednem vrstnem redu): Tadej Janež, Peter Nose, Ruben Sipoš, Lovro Šubelj, Mitja Trampuš, Janoš Vidali in Lan Žagar.

Zahvala tudi moji družini, očetu Stanetu, mami Zvonki in bratu Anžetu, za leta podpore in vzpodbude. Na koncu pa velika zahvala še Urški Bajec, moji življenjski sopotnici, za vso ljubezen in oporo.

Kazalo

Poglavje 1

Uvod

Disertacija predstavi nove metode umetne inteligence [87], s katerimi lahko avtonomen robot modelira učinke, ki jih imajo njegove akcije na svet okoli njega. Fokus našega pristopa je na razumljivosti ter izrazni moči modelov. S tem se oddaljujemo od pristopov klasične robotike, kjer je poudarek na metodah, ki omogočajo čim boljše in natančnejše delovanje robotov (npr. čim boljše motorične sposobnosti).

Struktura disertacije je sledeča. V tem poglavju začnemo z motivacijo in predstavitvijo problema (razdelek 1.1) ter naštejemo naše prispevke k znanosti. Naslednje poglavje (poglavje 2) vsebuje pregled področij, na katera spada disertacija, skupaj s predstavitvijo sorodnega dela. Poglavja 3-6 obsegajo jedro disertacije s poglobljenim opisom vseh znanstvenih prispevkov. Poglavje 3 vsebuje opis metode STRUDEL (*structural derivative learning*), s katero robot gradi relacijske modele svojih akcij. V poglavju 4 predstavimo algoritem ACES (*action clustering by effect similarity*), ki ga metoda STRUDEL uporabi za razvrščanje akcij v skupine glede na podobnost njihovih učinkov. Delovanje v disertaciji razvitih metod prikažemo v več poskusih, ki jih, skupaj z rezultati učenja, predstavimo v poglavju 5. V poglavju 6 opišemo sistem HYPER/CA (*hypothesis refiner/classification accuracy*), nov sistem induktivnega logičnega programiranja, ki smo ga uporabili za učenje v metodi STRUDEL. Disertacijo končamo s poglavjem 7, ki vsebuje zaključni komentar predstavljenih prispevkov in oriše nekaj možnosti za nadaljnje delo.

1.1 Problem, pristop in motivacija

V našem delu se ukvarjamo z avtonomnimi roboti. Tak robot deluje samostojno in z njim ne upravlja človek (npr. preko teleoperacije). Še natančneje se izmed avtonomnih robotov osredotočamo na robote, katerih delovanje je inteligentno. Tu imamo v mislih lastnosti in sposobnosti, ki jih kot inteligentne obravnava področje umetne inteligence: učenje, sklepanje, planiranje in zaznavanje. To iz naše tematike izključi tipične industrijske robote in nasploh vse tiste, ki le izvajajo fiksne procedure.

Vsak avtonomen robot za delovanje potrebuje nek interni model, ki modelira, kako se bo svet okoli robota odzval na akcije, ki jih robot izvede. To mu omogoča izbor ustreznih akcij in planiranje, sklepanje o stanju okolice ter podobne operacije. Ta model lahko robotu v celoti podamo vnaprej, vendar je v tem primeru statičen in se ne prilagaja spremembam v okolju. To je zelo omejujoče, saj je v splošnem nerealno pričakovati, da bomo ob snovanju tega modela predvideli vse, s čimer lahko robot pride v stik. Že najmanjša sprememba, ki ni v skladu z robotu podanim modelom, lahko popolnoma onemogoči delovanje robota.

Omejitve fiksnega modela lahko odpravimo tako, da robotu omogočimo, da se modela uči in ga s tem prilagaja svojim izkušnjam. Tipično robotu podamo nek jezik hipotez, v katerem lahko izrazi model sveta in svojega delovanja v njem, ter metodo, s katero gradi ta model sveta ter prireja njegove parametre. Tak jezik ali del njega so lahko na primer pravila, naučena s spodbujevanim učenjem, ali klasifikatorji, zgrajeni z metodami strojnega učenja, grafični modeli ipd.

Taka sposobnost učenja modela znatno izboljša robotovo uspešnost in prilagodljivost delovanja v neznanem ali delno neznanem okolju, vendar ima tudi ta pristop svoje omejitve. Večina metod strojnega učenja privzame fiksen jezik, v katerem robot zgradi svoj model. S tem, ko robot nabira novo znanje, model raste in prej ali slej postane nepraktično velik. Da se izognemo temu, da bi model postal okoren in neučinkovit, moramo omogočiti robotu, da tekom učenja razširja svoj jezik hipotez. Tako ga lahko dopolnjuje z novimi izrazi, s čimer lahko poenostavi svoj model in preprečuje, da bi zrasel preko praktičnih meja. Primer takega razširjanja jezika hipotez je izum predikata v induktivnem logičnem programiranju [105]. Osnovni cilj našega dela je razviti metodo, ki bo robotu omogočala učenje modelov svojih akcij z odkrivanjem in formulacijo novih konceptov.

Za opis akcij in njihovih učinkov je bilo, predvsem na področju planiranja, doslej razvitih več formalizmov. Primer sta recimo jezik PDDL [70] (*planning domain definition language*) in situacijski račun [66]. Mi smo se odločili za gradnjo modelov tipa STRIPS [32] (*Stanford research institute problem solver*). Obstajajo že pristopi, ki omogočajo avtomatsko učenje operatorjev STRIPS iz podatkov, in nekateri so tudi sposobni razširjati jezik hipotez (več o tem v okviru pregleda sorodnega dela v podrazdelku 2.2). Naš pristop se od teh razlikuje predvsem po *abstraktnosti* in *splošnosti* zgrajenih konceptov.

Z besedo *abstraktni* tu označujemo koncepte, ki niso eksplicitno razvidni iz robotovih opažanj in senzorjev. Primer tega je, ko robot, ki je sposoben iz kamere razbrati lokacije predmetov, odkrije koncept premičnosti (kot opisano v [60] in podrazdelku 5.2.1). Pri tem ne prejme nobenih primerov koncepta oz. praktičnega prikaza, temveč samostojno oblikuje opis premičnosti kot dinamike, kjer se lokacija predmeta spreminja. Primeri podobnih abstraktnih konceptov so 'stabilnost', 'ovira', 'orodje' itd. Pomembna lastnost zgrajenih modelov je, da so ti odkriti koncepti v njih predstavljeni eksplicitno in človeku razumljivo, kar omogoča njihovo interpretacijo.

S *splošnostjo* pa mislimo, da želimo imeti čim manj predpostavk o predznanju in končni obliki modela. Natančneje, metoda lahko privzame neko predznanje (predhodno naučeno ali podano), ne sme pa imeti za posamezno domeno specifičnih predpostavk. Zanima nas, kako lahko robot nove koncepte odkriva po nekem čim manjšem naboru osnovnih principov, kar je jedro filozofije, po kateri smo razvijali naše metode. Prav tako hočemo doseči čim večjo izraznost zgrajenih modelov. To med drugim pomeni, da lahko končni model akcij vsebuje učinek akcije ali odkriti koncept, ki je definiran rekurzivno. To dovoljuje elegantno modeliranje dinamike sveta – na primer, kako ima potisk prve kocke v vlakcu kock enak učinek na vsako naslednjo kocko, kot če bi jo potisnili neposredno. V nekaterih primerih s podanim jezikom hipotez in v okviru razumnih predpostavk (npr. o končnosti modela oz. spomina, v katerem ga hranimo) s tem sploh omogočimo splošen opis nekaterih domen (primer opisan v [10] in predstavljen v podrazdelku 5.2.2). Ni nam znan noben drug pristop, ki bi omogočal avtonomno robotsko učenje rekurzivnih modelov tipa STRIPS in rekurzivnih abstraktnih konceptov.

Na tej točki gre ponovno poudariti, da se z osredotočanjem na abstraktnost in splošnost konceptov oddaljujemo od tipičnih raziskav v robotiki, kjer je nava-

dno glavni cilj izboljšanje ali implementacija robotove sposobnosti izvajanja neke konkretne (fizične) naloge. V ta namen se uporabi poljubno, čim močnejšo učno metodo in poljubno obsežno predznanje, da le izboljša delovanje. Naš pristop spada v področje kognitivne robotike in je v nekaterih pogledih bližje računskim sistemom za odkrivanje (podrazdelek 2.1) kot klasični robotiki.

Naša osnovna predpostavka glede učenja je, da to poteka v tako imenovani *zanki poskusov* [8] (angl. *experimental loop*). Gre za potencialno neskončno zanko, v kateri poteka proces odkrivanja oz. učenja. Robot začne z neko (morda nično) začetno količino predznanja, nato pa ponavlja naslednji postopek[1]:

1. Izvedi poskuse in zberi opažanja.
2. Z metodami strojnega učenja iz zbranih podatkov zgradi nov model.
3. Oblikuj nove poskuse, ki bodo dali najbolj zanimive nove rezultate.
4. Splaniraj izvedbo novih poskusov na podlagi trenutnega modela.
5. Ponovi zanko od koraka 1.

Znotraj vsakega od korakov zanke poskusov se skrivajo zanimiva vprašanja in izzivi, ki so lahko podlaga za številne samostojne raziskave. V našem delu se ukvarjamo z 2. točko zanke poskusov, v kateri se na podlagi zbranih opažanj odkriva koncepte, ki sestavljajo nov model. Tu zanko poskusov omenjamo zaradi vpetosti v širšo sliko. V disertaciji se z ostalimi koraki ne ukvarjamo.

V grobem lahko naš pristop učenja modelov akcij strnemo v naslednje točke:

- Robot izvaja akcije na predmetih v svojem okolju ter pred vsako akcijo in po njej zbira vsa opažanja o stanju svojega okolja. Ta točka ustreza še prvemu koraku zanke poskusov in še ni del učenja.

- Za vsako akcijo lahko iz razlike med predhodnim in naslednjim stanjem izračunamo, kaj se v okolju zaradi akcije spremeni. Predpostavljamo, da so vse spremembe v okolju posledica robotovih akcij in drugih zunanjih vplivov (drugi agenti, ljudje, vpliv vremena ipd.) ni. Zaradi te predpostavke vemo, da izračunane spremembe predstavljajo primere učinkov robotovih akcij.

[1]Podan je splošni korak zanke. V praksi mora robot čisto na začetku ob prvem koraku, če nima predznanja, postopati drugače. Preprosta možnost je, da izvaja naključne akcije, s katerimi zbere podatke za prvo gradnjo modela.

- Akcije razvrstimo v skupine glede na razlike med njihovimi učinki. Ideja našega pristopa je, da je razlika v učinkih akcij posledica različnih lastnosti predmetov, ki nastopajo v akcijah. Lahko gre za inherentne lastnosti predmetov (npr. premičnost) ali neko trenutno relacijo, v kateri je ta predmet s svojo okolico (npr. najvišji od vseh). Induciran pogoj, ki loči med skupinami akcij, vsebuje definicijo te lastnosti in predstavlja odkriti abstrakten koncept. Na primer definicijo koncepta premičnosti, ki razlikuje med predmeti, ki se jim po potisku spremeni lokacija, in predmeti, ki se jim lokacija ne spremeni.

- Ko imamo enkrat definicijo koncepta (lastnosti), ki ločuje predmete, lahko zgradimo (induciramo) model oblike STRIPS, ki opisuje učinek akcije, glede na to, kakšne lastnosti ima predmet, na katerem akcijo izvedemo.

Postopek, opisan v gornjih točkah, smo implementirali v metodi STRUDEL, ki je v disertaciji opisana v poglavju 3.

Podatki, ki opisujejo robotov svet, njegovo predznanje in akcije, so po naravi relacijski. Vzemimo za primer preprosto akcijo potiska kocke. Ta ima glede na relativno pozicijo drugih kock v okolici lahko veliko različnih učinkov. Če se pri premiku kocka zadene v kako drugo kocko, se bo morda premaknila tudi ta; če je na kocko zložen stolp iz več kock, se te lahko podrejo in padejo po prostoru ipd. Zato, da lahko take podatke predstavimo, potrebujemo ustrezno izrazno močan jezik. V ta namen smo izbrali logično programiranje, paradigmo programiranja, ki temelji na logiki prvega reda. Natančneje, uporabili smo programski jezik prolog [11], ki nam omogoča zapis tako podatkov (logična dejstva), kot tudi modelov (predikati).

Ena ključnih točk našega pristopa je razvrščanje učinkov akcij v skupine. Učinki akcij so opisani z množico relacijskih podatkov in ne moremo uporabiti metod razvrščanja namenjenih za propozicionalne podatke. Obstaja nekaj metod, ki razvrščajo relacijske podatke v skupine (nekaj jih predstavimo v razdelku 2.3), vendar nobena ni popolnoma ustrezala našim namenom, zato smo razvili svojo metodo, ACES, ki jo podrobno predstavimo v poglavju 4.

Izbira predstavitve znanja gre z roko v roki z izbiro učne metode, saj mora biti slednja sposobna predelati vhodne podatke in zgraditi ustrezno bogat model. Za naše namene je primerno induktivno logično programiranje (ILP) [73], metoda strojnega učenja, katere izhod je logični program. V kontekstu učenja znotraj zanke poskusov je velika prednost, da je model izražen v istem jeziku kot

predznanje, ker z njim potem enostavneje razširimo predznanje in ga uporabimo v nadaljnjem učenju. Za metode ILP je tudi značilno, da lahko dobro izkoristijo raznoliko predznanje, ki ga podamo v obliki logičnih programov. Vse ta izrazna in učna moč seveda ne pride brez cene in največja težava metod ILP je velika računska zahtevnost, zaradi katere lahko učenje postane praktično neizvedljivo.

Obstaja več algoritmov za učenje po principih ILP, vendar smo po pregledu področja (razdelek 2.3) ugotovili, da so mnogi za naše namene neustrezni, saj zaradi svojega prekrivnega načina učenja slabo obravnavajo rekurzivne koncepte, kar je v nasprotju z našim načelom splošnosti konceptov. Za naše namene je v tem pogledu primeren sistem HYPER [7] (*hypothesis refiner*), ki inducira celotno hipotezo naenkrat in preiskuje prostor hipotez od zgoraj navzdol, zaradi česar je dobro sposoben inducirati rekurzivne hipoteze. Problem sistema HYPER je, da ne obravnava šumnih podatkov, ki so značilni za robotske domene. HYPER namreč vedno išče popolno in konsistentno hipotezo, torej tako s klasifikacijsko točnostjo enako 1. Če so v učnem naboru šumni primeri, ki jim hipoteze ne more prirediti, učenje konča z neuspehom in ne vrne ničesar. Po drugi strani sistemi ILP, ki šum obravnavajo, tipično zahtevajo, da uporabnik poda pričakovano stopnjo šuma kot vhodni parameter. Jasno je, da avtonomni robot pri izvajanju svojih poskusov ne more vnaprej poznati stopnje šuma, ki jo bodo imeli podatki. Za realizacijo učenja smo tako razvili nadgradnjo sistema HYPER imenovano HYPER/CA, ki je sposobna obravnavati šumne podatke in je podrobno opisana v poglavju 6. V istem poglavju predstavimo tudi rezultate poskusov s tipičnimi učnimi nalogami ILP, kjer sistem HYPER/CA primerjamo najnovejšimi sistemi ILP.

ILP ni edini možen pristop, s katerim bi lahko izvedli učenje. Lahko bi uporabili metode genetskega programiranja, kjer bi pogoje, ki ločijo med posameznimi učinki, poiskali kot avtomatsko definirane funkcije [53] (ADF). Genetsko programiranje je metodologija, ki izhaja iz področja evolucijskih algoritmov. Njen navdih tako prihaja iz evolucije v biološkem smislu kot procesa, v katerem se populacija bitij razvija preko več generacij z razmnoževanjem in mutacijami. V genetskem programiranju razvijamo populacijo programov, ki jih med seboj križamo (menjamo dele kode) in v njih vnašamo mutacije (naključne spremembe kode). Pravila za križanje in mutacije določi uporabnik in od njih je odvisna uspešnost evolucije. Po vsaki ponovitvi križanja in mutacij v naslednjo generacijo sprejmemo tiste potomce prejšnje generacije, ki najbolj ustrezajo neki kriterijski funkciji. Ta

kriterijska funkcija tipično temelji na uspešnosti opravljanja neke naloge. V kontekstu našega dela recimo, da program kot izhod vrne razred, v katerega spada vhodni primer. V tem primeru lahko kriterijska funkcija primerja klasifikacijske točnosti programov in izbira take, ki imajo čim višjo.

V kontekstu genetskega programiranja je avtomatsko definirana funkcija taka funkcija (procedura, modul itd.), ki se avtomatsko razvije tekom evolucije genetskega programa in jo lahko kliče glavni program. Uporaba teh funkcij temelji na razbitju problema na podprobleme. Izvajajo podnaloge, ki se v problemu večkrat ponovijo in na ta način omogočajo učinkovitejšo rešitev in krajši, elegantnejši program. Nanje lahko gledamo kot na evolucijski pristop k izumljanju predikatov [39]. Ker je pogoj, ki loči med akcijami z različnimi učinki, formuliran kot samostojen predikat, bi lahko njegovo učenje izvedli z ADF.

Temeljna razlika genetskega programiranja v primerjavi z ILP je, da gre za stohastično metodo. Genetsko programiranje na podlagi pravil za križanje in mutacije naključno preiskuje prostor programov, pri čemer ga vodi kriterijska funkcija. Metode ILP prav tako iščejo program z najboljšo vrednostjo kriterijske funkcije a prostor programov preiskujejo deterministično. Pri ILP je prostor programov omejen z determinističnimi pravili za razvoj programov in morebitnih drugih parametrov (npr. velikost programa). Ker je preiskovanje deterministično, imamo zagotovljeno, da bomo znotraj preiskanega prostora našli program, ki dosega optimalno vrednost kriterijske funkcije. Obstaja pa možnost, da bomo za to morali preiskati neperspektivne dele prostora, ki se jim genetsko programiranje s preferiranjem optimalnejših programov morda lahko izogne. Tako nam genetsko programiranje lahko predstavlja alternativo v primerih, ko je učenje z metodami ILP prezahtevno in smo se pripravljeni zadovoljiti s suboptimalnim programom.

Delovanje razvitih metod prikažemo v poglavju 5. Predstavljeno je učenje v štirih domenah, kjer se robot nauči modelov akcij tipa STRIPS. Med zgrajenimi modeli v rezultatih so tako taki z rekurzivno definiranimi pogoji, ki ločijo med učinki akcij, kot tudi taki z rekurzivno definiranim učinkom akcij. Vsi poskusi so bili izvedeni na umetnih podatkih, ki jih prilagamo v dodatku A. Robot v poskusih odkrije abstraktne koncepte: premičnost, “nižje kot”, “ne najvišje”, obteženost in prôstost.

1.2 Prispevki k znanosti

Znanstvene prispevke disertacije lahko strnemo v tri glavne točke:

- Metoda STRUDEL (poglavje 3), s katero avtonomen robot iz podatkov, ki jih sam zbere z poskusi, zgradi model tipa STRIPS, ki opisuje učinke njegovih akcij. Robot je pri gradnji sposoben odkrivati abstraktne koncepte, ki so sestavni del modela. Tako odkriti koncepti kot model sam so lahko definirani rekurzivno.

- Algoritem razvrščanja v skupine ACES (poglavje 4), ki akcije razvrsti v skupine glede na podobnosti njihovih učinkov. To nam omogoča učenje pogoja, ki loči med učinki akcij, v samostojnem koraku pred indukcijo modela akcij STRIPS.

- Sistem induktivnega logičnega programiranja HYPER/CA (poglavje 6), nadgrajena verzija sistema HYPER. Sistem HYPER/CA je sposoben obravnavati šumne podatke, pri čemer pričakovane stopnje šuma ne potrebuje podane kot parameter.

Poglavje 2

Pregled področja

Pregled področja in sorodnega dela smo organizirali v tri dele, ki ustrezajo trem ločenim podpodročjem, s katerimi se disertacija vsebinsko prekriva.

Začnemo z razdelkom 2.1, v katerem predstavimo pregled področja kognitivne robotike od teoretičnih začetkov do modernih pristopov. Na koncu razdelka na kratko omenimo tudi sorodno področje sistemov za odkrivanje.

Modeli, ki jih gradimo z našimi metodami, po obliki ustrezajo modelom tipa STRIPS, ki se prvenstveno uporabljajo za opis domen v planiranju. Znotraj področja planiranja že obstajajo pristopi, ki se učijo operatorjev za opis akcij iz podatkov. Pregled teh metod in njihovih podobnosti ter razlik z našim pristopom navedemo v razdelku 2.2.

Zadnji del tega poglavja, razdelek 2.3, vsebuje pregled induktivnega logičnega programiranja in umestitev sistema HYPER/CA med metode s tega področja.

2.1 Kognitivna robotika

Glavni namen kognitivne robotike [57, 61] je razvoj in preučevanje metod in formalizmov, ki omogočajo avtonomnemu robotu (fizičnemu agentu), da se v kompleksnem svetu uči, sklepa ter izpolnjuje zahtevne naloge. Področje leži v preseku med umetno inteligenco in robotiko [20, 88, 80] in se od slednje razlikuje predvsem v tem, da stremi k visokonivojskim procesom upravljanja robota [96]. Namen torej ni razviti namenskih rešitev za nek konkreten razred problemov, temveč sistem, ki je sposoben ustrezno krmiliti robota v kompleksnem, spreminjajočem se svetu,

skladno z nekim (trenutnim) prepričanjem o stanju tega sveta in visokonivojskimi cilji robota. Jasno je, da je za izpolnitev tega cilja potrebno neprestano učenje, da se lahko robot prilagaja novim, še nevidenim stanjem okolja.

Zaradi svoje splošnosti so cilji kognitivne robotike zelo zahtevni, saj gre v najbolj ambiciozni verziji za utelešeno močno umetno inteligenco (angl. strong AI). Nekatere osnovne principe tega področja lahko prepoznamo že v ideji o "strojih, ki razmišljajo", ki je temeljna v področju umetne inteligence ter je prisotna od njenih samih začetkov [69, 108]. Razvoj teoretične podlage tako sega daleč nazaj. Že v šestdesetih letih prejšnjega stoletja sta recimo McCarthy in Hayes razvila situacijski račun [68], logični formalizem za opis dinamičnih domen, ki je služil kot osnova nadaljnjemu razvoju simboličnih predstavitev znanja na tem področju [67, 86].

V zadnjih nekaj letih smo priča precejšnjemu razvoju in razširitvi robotske strojne opreme. Bolj znani primeri so recimo humanoidni roboti Asimo [42], NAO [41] in Atlas [29], štirinožni Bigdog [85] in avtomatski domači sesalnik Roomba [107]. Z izboljšavo strojne opreme je narasla potreba bo boljših kognitivnih sposobnostih robotov. Zanimiv indikator tega trenda so tekmovanja Robocup, kjer akademske ekipe z roboti tekmujejo v več disciplinah kot so robotski nogomet [49], reševalni roboti [50] ali robotski pomočniki [111]. Organizatorji namreč vsakič, ko tekmovalci uspejo dovolj izpopolniti svoje robote, zaostrijo pravila in otežijo nalogo tekmovanja. V zadnjih letih se tako dodaja vedno več kognitivno zahtevnih nalog. Podoben zanimiv projekt je Darpa Robotics Challenge [22], kjer je namen razviti humanoidnega robota, ki bo z ljudmi sodeloval pri reševalnih operacijah.

Začetki dela, predstavljenega v tej disertaciji, izvirajo iz evropskega projekta XPERO: Learning by experimentation [12], katerega namen je bil razviti robota, ki se samostojno uči iz lastnih poskusov. Zasnova problema in filozofija pristopa v disertaciji se v celoti ujemajo s tisto iz projekta XPERO. Znotraj tega projekta so že bili opravljeni poskusi, kjer se je robot iz podatkov o svojih akcijah učil abstraktnih konceptov, kot sta 'premičnost' in 'ovira' [60], a zgrajeni modeli niso tipa STRIPS in so inducirani direktno, medtem ko naš pristop indukcijo razdeli na več korakov, kot je opisano v razdelku 3.3. V nadaljnjih poskusih v okviru XPERO je z razširitvijo ideje učenja iz podatkov o eni akciji na učenje iz zaporedja akcij združenih v načrt za doseg nekega cilja robot odkril koncept orodja [55]. Odkrita definicija orodja je abstraktna: *Orodje je predmet uporabljen v neki akciji, ki ni*

cilj te akcije. Prvo verzijo metode STRUDEL [54], ki jo v disertaciji opišemo v razdelku 3.2, smo razvili tekom dela v projektu XPERO. Izkušnje zbrane znotraj tega projekta so bile podlaga za naš izbor učnih metod [8], kot tudi osnova za izhodišča o učenju abstraktnih konceptov [10, 9].

Iz projekta XPERO izhaja tudi sistem EVOLT (*EVOLving Theories*) [106, 79]. Ta je bil razvit na univerzi Bonn-Rhein-Sieg približno istočasno, kot je potekalo delo predstavljeno v tej disertaciji. Zasnova problema, ki ga rešuje sistem EVOLT, je zelo podobna zasnovi učnega problema metode STRUDEL. Sistem EVOLT se prav tako kot metoda STRUDEL uči definicije učinkov akcij iz opažanj, ki jih zbere avtonomen robot. Razen skupne zasnove problema pa sta pristopa zelo različna. Sistem za predstavitev modela uporablja nov formalizem imenovan teorija akcija–učinek (*angl. action–effect theory*). Trenutna oblika tega formalizma ne omogoča izražanja rekurzivnih definicij učinkov akcij. Modela se nauči z algoritmom ILP, ki s preiskovanjem s surovo silo gradi modele akcij. Učenje ne formulira splošnih novih predikatov, ki bi jih lahko uporabili kot predznanje v nadaljnjem učenju. Izrazna moč modelov, ki jih gradi sistem EVOLT je tako manjša kot izrazna moč modelov zgrajenih z našim pristopom.

Sama uporaba pristopov ILP v kognitivni robotiki ni novost. Sistem CAP [93, 44] se z metodami ILP uči splošnih opisov zaporedij akcij, ki mu jih je pokazal učitelj. Sistem CAP je celo sposoben zamenjati zaporedja iste ponovljene akcije s preprosto rekurzivno proceduro, a se ne uči rekurzivnih definicij konceptov. V sistemu TRAIL [2, 3] se agent avtonomno z izvajanjem planov ali iz prikazov uči "teleo-operatorjev", posplošitev operatorjev STRIPS na zvezne domene. Brown v svojem delu [13, 14] predstavi metodo za učenje uporabe predmetov kot orodij. Njegov robot z metodami ILP iz prikazov opravljanja neke naloge z rabo orodja in nato z lastnim poskušanjem poišče splošen opis primernega orodja za to nalogo. Klingspor, Morik in Rieger v [51] predstavijo sistem GRDT, s katerim se robot uči visokonivojskih konceptov iz nizkonivojskih senzorskih podatkov. Njihov robot se iz označenih primerov časovnih zaporedij sonarskih podatkov nauči opisa koncepta prehoda skozi vrata. Naj omenimo, da so si ti pristopi v nekaterih pogledih zelo podobni s tistimi iz učenja v planiranju in je včasih kako delo težko enolično klasificirati.

Veliko pristopov, ki se ukvarjajo z avtonomnim učenjem lastnosti robotovega okolja, uporablja princip *funkcijskih lastnosti.* Izraz *funkcijska lastnost* upora-

bljamo kot slovenski prevod angleškega izraza *affordance*, ki izhaja iz glagola *to afford* in ki ga v slovarju angleškega jezika ne najdemo. Teorija funkcijskih lastnosti prihaja iz psihologije. Izraz je leta 1977 uvedel Gibson [37] in označuje inherentne lastnosti predmetov in okolja, ki agentu (človeku, živali, robotu itd.) nudijo neke funkcije oziroma možnosti za izvajanje akcij. Na primer, ročaj skodelice človeku nudi to, da skodelico lahko prime, sama skodelica pa to, da iz nje pije. Po Gibsonovi teoriji naj bi predmete agent dojemal preko njihovih funkcijskih lastnosti pomembnih za njegove akcije.

Modeli, ki jih gradimo z našim pristopom, vsebujejo pogoje, zakaj ima neka akcija izvedena na izbranem predmetu nek učinek. To niso nujno funkcijske lastnosti, kot jih definira Gibson, vendar bi jih lahko reformulirali vanje, če bi želeli. Čeprav za naše namene to ni zanimivo in v tej smeri nismo raziskovali, gre gotovo za soroden princip, saj tudi funkcijske lastnosti opisujejo relacijo med lastnostmi predmetov in učinki akcij. Primerov učenja funkcijskih lastnosti v kognitivni robotiki je več. V skupini, ki jo vodi José Santos-Victor, so razvili pristop modeliranja in učenja funkcijskih lastnosti z Bayesovskimi mrežami preko imitacije [72, 71] in učenje pomenov besed preko asociacij z robotovimi akcijami in percepcijo [56, 91]. Fitzpatrick et al. uporabijo funkcijske lastnosti, da robota naučijo preko preproste fizične interakcije (potiski in potegi) prepoznavati predmete in oponašati akcije človeka [34]. Erol Şahin et al. uporabijo svojo formalizacijo funkcijskih lastnosti [90] za učenje ciljno orientiranega obnašanja iz primitivnih premikov [27] in prepoznavanje prehodnosti terena iz 3-D globinskih slik [109, 110].

Z našim pristopom robot pri gradnji modelov akcij odkriva abstraktne koncepte, ki lahko izražajo zelo splošne principe (premičnost, stabilnost itd.). Odkrivanje splošnih zakonitosti v različnih področjih znanosti je domena sistemov za računsko znanstveno odkrivanje (angl. computational scientific discovery). Gre sicer za dve različni področji. Kognitivna robotika se ukvarja z inteligenco fizičnih robotov, sistemi za odkrivanje pa pogosto nimajo fizične komponente in so izvedeni v celoti v programski opremi. Zgodnje delo na sistemih za odkrivanje [99] se je osredotočalo predvsem na ponovno odkrivanje že znanih zakonov. Primeri takih sistemov so sistemi GLAUBER [58], DALTON [58] za odkrivanje kvalitativnih in BACON [58], LAGRANGE [31] numeričnih zakonov. Kljub pozivom po pomembnosti avtonomije teh sistemov [117, 118, 119] so popolnoma avtonomni sistemi za odkrivanje relativno redki. Primera avtonomnih sistemov sta sistema

DIDO [95] in LIVE [98]. Langley v novejšem pregledu področja [59] ugotovi, da so bili računski sistemi za odkrivanje uporabljeni pri odkrivanju novih znanstvenih dognanj, objavljenih v njihovih matičnih znanstvenih področjih. Najbrž najbolj znan primer je Kingov robotski znanstvenik [47], ki je planiral poskuse in testiral lastne hipoteze na področju funkcionalne genomike. Langley poudarja, da ima pri teh sistemih človeški vpliv (formalizacija problema, priprava podatkov itd.) še vedno zelo pomembno vlogo pri uspehu učenja.

2.2 Učenje v planiranju

Planiranje [89] sodi med klasična področja umetne inteligence in se ukvarja s predstavitvijo akcij ter iskanjem zaporedij akcij, ki dosežejo želeni cilj. Učenje v planiranju [116] se tipično uporablja v tri namene:

- **pospešitev planiranja** – Detekcija neperspektivnih delov prostora možnih planov in usmerjanje preiskovanja v njem.
- **izboljšava kakovosti planov** – Usmerjanje iskanja k planom z želenimi lastnostmi ali učenje preferenc uporabnika pri planih.
- **učenje in izboljšava domenske teorije** – Učenje operatorjev, ki opisujejo akcije, ali njihovo prilagajanje v dinamičnih domenah.

Našemu delu sorodni so pristopi, ki sodijo v zadnjo skupino, saj se tudi naš pristop uči opisov akcij v obliki operatorjev STRIPS.

Schmill, Oates in Cohen v [94] predstavijo sistem učenja, kjer robot podatke o svojih akcijah najprej razvrsti v skupine glede na učinke in nato zgradi klasifikacijsko drevo, s katerim izvedejo planiranje podobno metodam planiranja z operatorji STRIPS. Učinke akcij obravnavajo na precej nizkem nivoju zveznih senzorskih podatkov in za razvrščanje uporabljajo metodo DTW (dynamic time warping).

Sistem LOPE [36] avtorjev García-Martínez in Borrajo omogoča integrirano učenje, planiranje in izvedbo planov mobilnega agenta. Predstavitev akcij, ki jo uporablja, je mešana med nizkim (senzorski podatki) in visokim (logika prvega reda). Modela se učijo s spodbujevanim učenjem in ga potem posplošijo z domensko specifičnimi hevristikami.

Sistem OBSERVER [113, 114] se uči operatorjev tipa STRIPS iz prikazov planov domenskega eksperta. Učenje je izvedeno s posploševanjem primerov podobnim pristopu s prostori verzij (version spaces).

Arhitektura PRODIGY [112], ki je bila zastavljena kot testna in razvojna platforma za algoritme planiranja in učenja v planiranju, vsebuje implementacije več različnih učnih pristopov, tako za učenje domenske teorije kot za izboljšavo planov. Sistem EXPO [38], ki bazira na arhitekturi PRODIGY omogoča dopolnjevanje in popravljanje nepopolnih in nepravilnih domenskih teorij.

Vsi do sedaj našteti pristopi učenja se sicer učijo modelov akcij, a je za to učenje uporabljen fiksen jezik in je prvenstveno usmerjeno v to, da so plani iz akcij čim učinkovitejši. Za nas je najbolj zanimivo delo avtorjev Pasula, Zettlemoyer in Kaelbling [82], kjer je predstavljena edina nam znana metoda učenja operatorjev za planiranje, ki uporablja predznanje in tekom učenja razširja svoj jezik hipotez. Njihova metoda se uči verjetnostnih modelov akcij v nedeterminističnih domenah z večnivojsko indukcijo, katere najvišji sloj konstruira nove predikate in jih dodaja v predznanje nižjih slojev, ki potem to predznanje uporabijo za konstrukcijo operatorjev za planiranje. Metoda, s katero so inducirani novi predikati in model akcij, je manj splošna kot v našem pristopu. Definicije novih predikatov in modela akcij recimo ne morejo biti rekurzivne. Tako kot pri vseh ostalih pristopih v tem področju je manj poudarka na splošnosti in abstraktnosti zgrajenega modela, saj je končni cilj učinkovito planiranje in ne razumevanje domene.

2.3 Induktivno logično programiranje

Induktivno logično programiranje (ILP) sodi v presek področij strojnega učenja in logičnega programiranja [63, 11]. Gre za metodo strojnega učenja, ki za predstavitev primerov, predznanja in hipoteze (zgrajenega modela) uporablja logično programiranje. Izraz je prvi uporabil Stephen Muggleton v svojem članku iz leta 1991 [73], vendar začetki področja z rezultati, kot je Plotkinova relativna najmanj splošna posplošitev (angl. relative least general generalization) [83] ter Muggletonova in Buntineova uporaba inverzne razrešitve (angl. inverse resolution) [76], segajo že v čas pred tem. Aktualna pregledna članka področja sta [77, 78].

Nadzorovano učenje z ILP

Učni problem, ki ga rešuje ILP, lahko v splošnem zapišemo v naslednji obliki. Podano imamo množico pozitivnih primerov, E_+, in množico negativnih primerov, E_-, ki sta navadno zapisani v logičnem programu kot množici logičnih dejstev. Podano imamo tudi predznanje, B, izraženo v obliki logičnih formul (predikatov in dejstev), tako da velja $B \not\models E_+$. Operator $\models$ izraža logično posledico. Če bi držal izraz $B \models E_+$, bi to pomenilo, da lahko vsak pozitivni primer $e \in E_+$ z logičnimi operacijami izpeljemo iz logičnih formul v B. Pogoj $B \not\models E_+$ preverja ali je učenje sploh potrebno, saj v nasprotnem primeru podano predznanje že reši učni problem. Poiskati moramo tako hipotezo H, kot množico logičnih formul, da bosta veljali formuli 2.1 in 2.2.

$$B \wedge H \models E_+ \tag{2.1}$$

$$B \wedge H \not\models E_- \tag{2.2}$$

Če za posamezen primer e velja $B \wedge H \models e$, rečemo, da hipoteza H pokrije e. Če velja enačba 2.1, pravimo, da je hipoteza *popolna*, tj. da pokrije vse pozitivne primere, če velja enačba 2.1, pa da je *konsistentna*, tj. da ne pokrije nobenega negativnega primera.

Navadno je predpostavka pri tej definiciji problema ILP, da so podatki pravilni in ne vsebujejo šuma. V nasprotnem primeru je namreč problem lahko nerešljiv. Pod to predpostavko deluje več sistemov ILP, kot so sistemi MIS [97], MARVIN [92], CIGOL [76] in CLINT [23]. Tudi sistem HYPER [7, 11], ki ga v disertaciji uporabimo za učenje in služi kot izhodišče našemu delu, deluje pod to predpostavko. Omeniti gre, da lahko težavo šumnih podatkov včasih rešimo z dodajanjem predikatov, ki poskrbijo za šum (npr. uvedejo nek tolerančni interval pri numeričnih vrednostih), v predznanje.

Zanimiv je tudi zelo nov sistem MC-TopLog [62], ki prav tako predpostavlja brezšumne podatke. Sistem MC-TopLog so razvili nekateri isti avtorji kot sistem Progol omenjen v naslednjem odstavku. Sistem Progol spada med najučinkovitejše moderne sisteme ILP, a ga sistem MC-TopLog v hitrosti prekaša na nekaterih zahtevnih domenah[1]. Poleg tega se je sistem MC-TopLog za razliko od sistema

[1] Parametri sistema Progol (pričakovana stopnja šuma v podatkih itd.) so nastavljeni ustrezno, da je primerjava poštena.

Progol sposoben učiti večstavčnih hipotez. Sistem MC-TopLog je preveč nov, da bi ga uporabili v tej disertaciji, tako da poskusi z njim ostajajo za nadaljnje delo.

Empirični podatki iz realnih domen seveda pogosto vsebujejo šum, zato veliko sistemov ILP popusti pri pogojih o popolni in konsistentni hipotezi ter išče tako, ki se podatkom najbolje prilega na podlagi neke kriterijske funkcije. Tak je recimo dobro znan Quinlanov sistem FOIL [84], prekrivni algoritem, ki na podlagi hevristike temelječe na informacijskem prispevku išče ustrezne stavke, ki jih dodaja hipotezi. V to skupino sodita sistema Progol [74] in Aleph [101], ki ju uporabimo za primerjavo v naši evaluaciji in sta bolj podrobno predstavljena v podrazdelku 6.2.1. Obstaja tudi nadgrajena verzija sistema HYPER, HYPER/N [81], ki sta jo razvila Oblak in Bratko ter je sposobna obravnavati šum. Pri teh sistemih je pogosto, da potrebujejo pričakovano stopnjo šuma podano kot parameter.

Še korak dlje gredo pristopi verjetnostnega logičnega učenja oziroma verjetnostnega ILP [24, 26, 28]. V teh področjih je jezik logike nadgrajen z verjetnostnimi parametri. Naučena hipoteza izraža verjetnostno porazdelitev v prostoru logičnih formul. Primer takih verjetnostnih relacijskih modelov so relacijske Bayesovske mreže [45] in stohastični logični programi [75], primer sistema ILP, ki gradi take modele, je Alchemy [52].

Nenadzorovano učenje z ILP

Vsi do sedaj našteti sistemi ILP sodijo na področje nadzorovanega učenja, ker prejmejo označene primere, na podlagi katerih gradijo hipotezo. Cilj njihovega učenja je ekvivalenten pravilni klasifikaciji primerov. Komplementarne temu so metode razvrščanja v skupine, ki delujejo na relacijskih podatkih. Te metode, tako kot propozicionalne metode nenadzorovanega učenja [4], prejmejo množico neoznačenih primerov E in je njihova naloga razvrstiti primere iz E v skupine primerov, ki so si med seboj znotraj skupine bolj podobni kot so podobni primerom izven skupine.

Pristope tu lahko delimo na dve skupini. Prva so metode za konceptualno razvrščanje v skupine, ki zgradijo skupine s simboličnimi opisi v danem predstavitvenem jeziku. Primera sta sistem KBG [5], ki zgradi hierarhijo simboličnih opisov konceptov z iterativnim posploševanjem opisov najpodobnejših skupin, in algoritem C 0.5 [25], ki za razvrščanje uporablja logična odločitvena drevesa iz sistema

TILDE [6].

Druga skupina metod pa le razvrsti primere v skupine, brez gradnje simboličnih opisov. Algoritem RDBC [48] uporablja mero razdalje iz sistem RIBL [43], ki bazira na številu sprememb potrebnih, da en logični zapis spremenimo v drugega, za aglomerativno razvrščanje v skupine. Model MMRC (mixed model relational clustering) [64] je verjetnostni pristop, ki gradi skupine z Monte Carlo EM (expectation maximization) algoritmom. Sem sodi tudi pristop najbližje našemu, predstavljenemu v disertaciji (poglavje 4). Gre za priredbo sistema SUBDUE [46], ki je sicer namenjen odkrivanju podskupin [18]. Njihova mera razdalje, tako kot naša, temelji na razdalji med grafi, ki ustrezajo logičnim izrazom. Bistvena razlika, kot pojasnimo v poglavju 4, je, da njihov pristop upošteva le razlike med grafi, naš pa tudi podobnosti. Podobnosti in razlike med grafi so pri obeh pristopih merjene z velikostjo delov grafov, ki se po strukturi ujemajo oz. ne ujemajo.

Poglavje 3

Metoda učenja modelov robotskih akcij STRUDEL

STRUDEL – *STRUctural DErivative Learning* (učenje strukturnih odvodov), je metoda, ki z deljenjem učne množice na podmnožice glede na strukturne razlike med primeri, omogoča učinkovito učenje ciljnega modela. To poglavje v razdelku 3.1 najprej predstavi osnovno motivacijo in ideje, na podlagi katerih STRUDEL deluje. V nadaljevanju je v razdelku 3.2 opisana začetna verzija te metode, ki jo bomo tu poimenovali kvalitativni STRUDEL. Poglavje konča razdelek 3.3, ki vsebuje opis metode STRUDEL, kot je uporabljena v poskusih opisanih v tem doktorskem delu.

3.1 Motivacija in zasnova

Ena največjih težav ILP [73] oziroma celotnega področja relacijskega učenja [30] je časovna zahtevnost učnih algoritmov, ki je posledica velike kombinatorične kompleksnosti prostora možnih modelov, ki ga morajo preiskati. Zaradi tega je učenje relacijskih modelov pogosto nepraktično ali pa celo neizvedljivo.

Najbolj preprost način, kako zmanjšati to časovno zahtevnost je, da dodamo več predznanja v učni problem. S tem lahko izboljšamo definicijo problema in razširimo množico logičnih predikatov, ki jih lahko uporabimo za opis modela. Seveda bo dodatno predznanje izboljšalo delovanje le, če dejansko bolje definira problem in nudi uporabne informacije za učenje. V nasprotnem primeru je dodatno

predznanje samo moteče in praviloma močno poveča kombinatorično zahtevnost učenja. Dodajanje predznanja v učne probleme ILP je preprosto, saj moramo le dopisati dodatna dejstva in predikate v logični opis problema. Dodatna prednost je, da nam ni potrebno spreminjati algoritmov, ki jih uporabljamo. Sistemi ILP po svoji zasnovi praviloma dobro izkoristijo podano predznanje [102], dokler je to relevantno za dani učni problem. Težava je, da dodatno oziroma boljše predznanje pogosto ni na voljo, ali pa ni očitno, kateri del znanja, ki ga imamo na voljo, je uporaben kot predznanje za naš učni problem, tako da te možnosti ne moremo vedno izkoristiti.

Sistemi ILP vsebujejo različne načine, kako izboljšati učinkovitost preiskovanja. Glavna pristopa, ki izhajata iz teorije preiskovalnih algoritmov, sta omejevanje prostora preiskovanja in uporaba hevrističnih funkcij. Primer omejevanja prostora je v Progolu [74] (podrazdelek 6.2.1) konstrukcija najbolj specifičnega stavka, ki od spodaj omeji preiskovalni prostor možnih stavkov, ki pokrijejo dani primer. Primer hevristike pa najdemo v sistemu HYPER [7, 11] (podrazdelek 6.1.1), kjer cenilna funkcija na podlagi točnosti in velikosti hipoteze omogoča preiskovanje od najboljšega proti najslabšemu kandidatu.

Kot posebno težavni se lahko izkažejo koncepti, definirani z več medsebojno odvisnimi stavki. Vzemimo kot primer definicijo premičnosti iz [60] zapisan v programu 1.

Model opisuje učinek akcije `move` z uporabo pomožnega predikata `p`, ki loči med različnima učinkoma `move`. Če hočemo odkriti ta model, moramo, preden lahko oblikujemo oba `move` stavka, najprej odkriti predikat `p`, saj ta nastopa v obeh stavkih. Brez učinkovite hevristike ali omejenega prostora preiskovanja bo v najslabšem primeru preiskovanje za vsako napačno obliko predikata `p` pregledalo vse možne oblike obeh `move` stavkov.

Ta model je mogoče zapisati tudi brez uporabe pomožnega predikata `p`. Namesto klicev predikata `p` lahko v oba `move` stavka prestavimo vsebino njegovega telesa in v prvem stavku, kjer je klic negiran, negiramo konjunkcijo literalov iz p. Ta verzija je sicer enako točna, vendar je model večji (tj. število literalov in spremenljivk v njegovem logičnem programu je večje). Prav tako se delno izgubita razumljivost in uporabnost modela. V predikatu `p` lahko prepoznamo definicijo koncepta premičnosti predmetov. Če je ta definiran v samostojnem stavku, ga lahko v nadaljnjem učenju uporabimo pri indukciji drugih modelov.

```
p(Obj) :-
  at(Obj, T1, Pos1),
  at(Obj, T2, Pos2),
  different(Pos1, Pos2).

move(Obj, Start, Dist, End) :-
  approxEqual(Start, End),
  not p(Obj).

move(Obj, Start, Dist, End) :-
  add(Start, Dist, End),
  p(Obj).
```

Program 1: Model akcije `move` iz [60].

V primeru, da bi `p` vseboval v več stavkih definiran rekurziven koncept, v prejšnjem odstavku predlagani alternativni zapis brez samostojne definicije sploh ne bi bil mogoč. Ne moremo namreč na enak način vložiti konjunkcije vsebine več rekurzivno definiranih stavkov v nek drug stavek. Že sama rakurzivnost zahteva samostojen stavek, čigar glavo lahko potem v njegovem telesu spet uporabimo. Dodatna težava je, da hevristike pri rekurzivnih konceptih pogosto delujejo slabo, saj delne oblike rekurzivnih definicij, ki jih razvijemo tekom preiskovanja, ponavadi ne delujejo obetavno (na učnih podatkih nimajo visoke točnosti). Hevristike tako preiskovanja ne vodijo učinkovito v smeri končne rekurzivne definicije.

Metoda STRUDEL gradi modele tako, da učno množico relacijskih podatkov razdeli na skupine primerov, ki so si med seboj strukturno različni. S strukturnimi razlikami tu govorimo o razliki v strukturi relacij, v katerih nastopajo posamezni primeri. V prejšnjem primeru to pomeni, da bi učinke akcije `move` razdelil na tiste, kjer se predmet premakne, in tiste, kjer ostane na istem mestu, saj sta relaciji med začetno in končno pozicijo v obeh primerih različni. Metoda nato v ločenem koraku najprej poišče definicije predikatov, ki ločijo med skupinami primerov. To pomeni, da bi v zgornjem primeru najprej definiral predikat `p`, ki deluje kot pogoj za to, kateri učinek ima akcija `move`. Te predikate nazadnje metoda uporabi za definicijo

končnega modela. Celoten postopek podrobneje opišemo v razdelku 3.3.

Pogoji v modelih, ki jih gradimo z metodo STRUDEL, kakršen je na primer predikat `p` iz primera, kot razložimo v prejšnjem odstavku, niso enaki predpogojem, kot se tipično uporabljajo v jezikih tipa STRIPS. V slednjih imamo navadno predpogoje ali akcijo lahko izvedemo, medtem ko pogoji, ki jih odkrivamo v metodi STRUDEL, povejo, ali bo izvedena akcija imela nek učinek. Predpogojev za izvedbo akcij se z metodo STRUDEL ne učimo. Možno je, da bi jih v nekaterih modelih zgrajenih z njo morda lahko pridobili iz pogojev za učinke, vendar teh možnosti v okviru disertacije nismo preučili. Na primer v modelu premičnosti v programu 8 (stran 55), bi lahko prazni učinek (tj. učinek, ko robot poskuša premakniti škatlo, ki ni premična), označili za neuspeh in pogoj `movable` uporabili kot predpogoj za to, da lahko uspešno izvedemo akcijo `move`.

Učenje predpogojev za izvedbo akcije je zahtevno, ker robot vedno vidi le pozitivne primere svojih akcij. Primera nemogoče akcije po definiciji ne mora empirično pridobiti. Ena obetavna možnost za rešitev tega problema je, da bi robot lahko o predpogojih sklepal na podlagi statistične analize stanj, ko je akcijo uspešno izvedel. Če neka dejstva v stanjih, ko je akcijo uspešno izvedel, vedno veljajo, je verjetno, da nekatera od njih predpogoj za izvedbo te akcije. Lahko bi sklepali, da je ta verjetnost še večja, če ta dejstva sicer v drugih stanjih ne veljajo velikokrat. Na podlagi tako določenih verjetnosti za predpogoje akcij bi robot lahko načrtoval nadaljnje poskuse in tako še natančneje določil predpogoje akcij. Poskusov s tem ali podobnimi pristopi nismo izvajali in raziskovanje v tej smeri ostaja odprto za nadaljnje delo.

Osnovna predpostavka pristopa STRUDEL je, da učno množico sestavljajo disjunktne podskupine strukturno različnih primerov, med katerimi mora ciljni model ločiti, da jo lahko (učinkovito) opiše. Te skupine lahko na podlagi strukturnih razlik poiščemo v ločenem koraku in s tem znižamo zahtevnost indukcije končnega modela. Seveda tega v splošnem ne moremo zagotoviti. Jasen protiprimer so problemi, kjer bi moral model ločiti primere na podlagi vrednosti atributov (npr. numeričnih vrednosti nekaterih argumentov v relacijah). Tega iz strukturnih razlik ne moremo razbrati. Naši poskusi (poglavje 5) kažejo, da se ta pristop obnese v robotskih domenah, kjer so učinki akcij odvisni od fizične postavitve predmetov v svetu. V teh domenah je struktura relacij, ki opisujejo primere, zelo podobna dejanski fizični strukturi predmetov, ki je razlog za različne učinke.

Pri avtonomnem učenju učinkov akcij v robotiki, kar je glavna tema tega dela, predpostavljamo, da so različni učinki robotskih akcij posledica razlik v lastnostih predmetov uporabljenih v teh akcijah. Pogoj za posamezni učinek izrazimo s predikatom, ki kot argument prejme predmet in čas. Primer pogoja iz modela zgrajenega v podrazdelku 5.2.4 je zapisan v programu 2. Operator `\+` uporabljen v primeru v jeziku prolog označuje negacijo[1].

```
clear(Box1, Time):-
  \+ in_state(Time, on(Box2, Box1)).
```

Program 2: Pogoj, da je škatla `Box1` ob času `Time` prosta, torej, da na njej ni druge škatle.

Tako lahko v pogojih upoštevamo tako inherentne lastnosti predmeta (npr. teža), kot tudi trenutne, spremenljive lastnosti predmeta (npr. trenutna lokacija). Ti pogoji so potencialno zanimivi koncepti in želimo, da bi bil robot sposoben odkriti njihove definicije, zato v ta namen uporabimo metodo STRUDEL.

3.2 Metoda kvalitativni STRUDEL

Prva verzija metode STRUDEL [54] je bila razvita v okviru projekta XPERO: Learning by Experimentation (evropskega projekta iz okvirnega programa evropske komisije, šifra: IST-29427). Osnovni cilj projekta je bil razviti kognitivni sistem, utelešen v robotu, ki bo sposoben izvajati poskuse v resničnem svetu, z namenom pridobivanja znanja o svojem okolju, razvoja in nadgradnje svojih sposobnosti ter izpopolnjevanja svojega delovanja na splošno. Ta verzija je bila uporabljena v predstavitvi, ki je na razstavi v okviru konference European Future Technologies Conference 2009 [17] dobila prvo nagrado po izboru občinstva.

Zaradi boljšega razumevanja bomo v tem delu to zgodnjo verzijo metode STRUDEL v tej disertaciji imenovali kvalitativni STRUDEL. Namen metode kvalitativni STRUDEL se od zdajšnje verzije ni razlikoval. Omogočal je učinkovito gradnjo relacijskih modelov robotskih akcij z delitvijo učne množice na disjunktne skupine

[1] Natančneje gre za negacijo kot neuspeh (*angl. negation as failure*), ki ni ekvivalentna logični negaciji, kot je definirana v formalni matematični logiki.

primerov učinkov akcij, ki so bili strukturno različni. Modeli akcij niso bili opisani v jeziku tipa STRIPS, vendar v obliki navadnih logičnih programov, tako kot v predhodnem delu v okviru projekta XPERO [60].

Metoda kvalitativni STRUDEL v poskusih opisanih v disertaciji ni uporabljena. Namen tega razdelka je navezava na predhodno delo in njegova umestitev v razvoj metod, ki so znanstveni prispevek disertacije. Metoda STRUDEL, ki jo uporabljamo v poskusih, si z metodo sicer do neke mere deli osnovno strukturo delovanja, a se v izvedbi posameznih korakov popolnoma razlikuje in gradi modele drugačnega tipa (STRIPS). Zaradi tega opis v tem razdelku ni zelo podroben. Natančnejši in s primeri bolje podprt opis zainteresiran bralec najde v [54].

3.2.1 Delovanje metode kvalitativni STRUDEL

Vhodni podatki v kvalitativni STRUDEL sestojijo iz zaporedja robotskih akcij, $A_1, \ldots, A_n$, in zaporedja množic dejstev, $S_1, \ldots, S_{n+1}$, ki opisujejo stanja sveta med akcijami. Poleg tega je potrebno podati še predznanje v obliki predikatov, ki so se lahko uporabili v modelu, in specifikacije tipov argumentov vseh predikatov in dejstev v učni množici in predznanju. Kvalitativni STRUDEL deluje po naslednjem postopku:

1. **Izračun sprememb:** Najprej se za vsak par zaporednih stanj, S_i in S_{i+1}, izračuna spremembo med njima, D_i, kot simetrično razliko[2] obeh množic: $D_i = S_i \Delta S_{i+1}$. D_i je vsebovala vsa dejstva, ki se spremenijo med zaporednimi stanji. Zato, da se da ugotoviti, katera dejstva pripadajo predhodnemu in katera naslednjemu stanju, imajo po predpostavki vsa dejstva čas kot enega od argumentov.

2. **Razvrščanje v skupine s posplošitvijo sprememb:** Vse spremembe, $D_i, \ldots, D_{i+1}$, se posploši tako, da se vse argumente dejstev v njih spremeni v spremenljivke. S tem, da se vsi argumenti iste vrednosti in tipa posplošijo v isto spremenljivko. To služi kot poenostavitev problema in je v splošnem lahko vir šuma. Vrednosti različnih argumentov so lahko enake tudi po

[2]Simetrična razlika dveh množic A in B, označena z $A \Delta B$, je množica vseh elementov, ki so v množicah A ali B a niso v njunem preseku, $A \cap B$.

naključju in ne gre za poseben primer spremembe. Primer take posplošitve množice dejstev je:

```
dejstvo1(1, a, b).        dejstvo1(A, B, C).
dejstvo1(2, a, c).   ⟶    dejstvo1(D, B, E).
dejstvo2(2, b).           dejstvo2(D, C).
```

Posplošene spremembe se nato razvrsti v ekvivalenčne skupine glede na ujemanje med njimi. Dve spremembi sta razvrščeni skupaj, če se razlikujeta kvečjemu v vrstnem redu dejstev in/ali imenih spremenljivk. Na primer naslednji spremembi sta ekvivalentni in bi bili zato razvrščeni v isto skupino:

```
dejstvo1(A, B, C).      dejstvo1(C, D, A).
dejstvo1(D, B, E).  ≡   dejstvo2(B, A).
dejstvo2(D, C).         dejstvo1(B, D, E).
```

Iz vsake skupine se izbere po ena posplošena sprememba in iz njih se v jeziku prolog tvorijo stavki, `f1`,...,`fk`, ki predstavljajo kvalitativne strukturne spremembe odkrite v podatkih oz. kvalitativne učinke akcij, $f_i, \ldots, f_k$. Glava stavka f_i je oblike `fi(A1,...,An)`, kjer so argumenti `A1`,...,`An` spremenljivke istega tipa in v istem vrstnem redu, kot so argumenti akcij. Telo stavka f_i sestoji iz literalov posplošene spremembe in dveh tipov dodatnih omejitev. Vsakemu od njih se v telo poleg literalov posplošene spremembe doda omejitev, ki pove, da gre za dejstva iz dveh zaporednih stanj – torej če sta spremenljivki tipa *čas*, ki nastopata v dejstvih, poimenovani `T1` in `T2`, se doda CLP(R) (*Constraint Logic Programming (Real)*, poglavje 7 v [11]) omejitev `{T2 = T1 + 1}`. Ta notacija pomeni numerično omejitev, da je `T2` enak `T1 + 1`, ki velja v celotnem logičnem programu. Prav tako se za vsak par različno poimenovanih spremenljivk različnega tipa, `Var1` in `Var2`, ki nastopata v argumentih literalov posplošene spremembe, v telo f_i doda neenakost `Var1 \== Var2`. Operator `≡` v prologu preveri ali imata spremenljivki isto vrednost oziroma ali sta bili vezani na isto vrednost.

3. **Indukcija pogojev:** Za vsak v prejšnjem koraku odkriti kvalitativni učinek f_i se s sistemom HYPER inducira predikat `pi`, ki loči akcije s tem učinkom od ostalih. Predikat `pi` opisuje pogoj p_i, ki pove ali ima neka akcija učinek f_i,

in ima iste argumente (tj. v glavi ima spremenljivke istega tipa) kot akcija, razen časa izvedbe akcije, ker je predpostavka, da gre za splošne lastnosti predmetov, ki nastopajo v akciji, in so tako neodvisne od časa.

Vsak predikat `pi` se inducira ločeno tako, da se izmed vseh izvedenih akcij $A_1, \ldots, A_n$ za vsako akcijo z učinkom f_i ustvari pozitiven primer in negativen primer za vsako drugo akcijo. Vsa dejstva iz vseh stanj, ki jih je robot zbral tekom poskusov, so podana kot predznanje pri indukciji skupaj z vsem drugim predznanjem, ki ga ima robot.

Če je indukcija vseh predikatov `p1`,...,`pk` uspešna, lahko nadaljujemo z gradnjo modela v zadnjem koraku. V primeru, da ni uspela indukcija samo enega od predikatov `pj`, ga lahko konstruiramo tako, da v enem stavku negiramo vse ostale pogoje – njegovo telo torej sestoji iz negiranih klicev `pg`, za vsak $g \in \{1, \ldots, k\}/\{j\}$. Kadar ne uspe indukcija več kot enega pogoja, se učenje s kvalitativnim STRUDEL konča neuspešno.

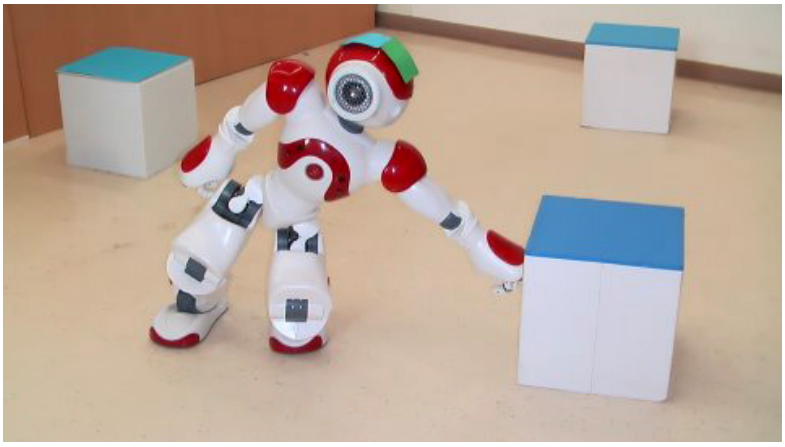
4. **Gradnja modela:** V zadnjem koraku kvalitativne učinke in njihove pogoje združimo v model učinkov akcij. Za vsak par učinek in pogoj, f_i in p_i, tako konstruiramo stavek:

```
action(Time, arguments):-
  pi(arguments),
  fi(Time, arguments).
```

kjer je `action` ime robotske akcije, ki jo modeliramo, in so `arguments` argumenti, ki jih ima akcija poleg časa `Time`, ko je bila izvedena. Ti stavki, skupaj s stavki, ki definirajo učinke in pogoje, sestavljajo končni rezultat učenja.

3.2.2 Poskusi z metodo kvalitativni STRUDEL

Metoda kvalitativni STRUDEL je bila preizkušena s poskusi na fizičnem robotu NAO [41]. Podroben opis izvedbe in rezultatov poskusov je v [54], tu podajamo le primera dveh modelov zgrajenih z njim, ki vsebujeta odkrita koncepta *premičnost* in *stabilnost*.



Slika 3.1: Odkrivanje koncepta premičnosti: robot potiska škatlo z akcijo `move`.

Premičnost: Zasnova za odkrivanje koncepta premičnosti je enaka kot v [60]. Robot je v prostoru z več škatlami, med katerimi na videz ne zazna nobenih razlik. Nekatere škatle so lahke in jih robot lahko fizično premakne, nekatere pa so obtežene in jih ne more premakniti. Robot zna izvajati akcijo `move(Time, Obj)`, s katero se premakne do predmeta (škatle) `Obj` in jo potisne (Slika: 3.1). Argument `Time` pove, ob katerem času (tj. v katerem stanju) je bila akcija izvedena. Robot je v poskusu v naključnem vrstnem redu izvedel akcijo na vseh škatlah in iz zbranih podatkov induciral model akcije `move` (program 3).

V pogoju `p1(Obj)` iz modela jasno prepoznamo enako definiran koncept premičnosti kot v [60] in ga lahko interpretiramo: *"Predmetpredmet Obj je premičen, če je kdaj bil na dveh različnih mestih."*

Stabilnost: Postavitev poskusa, v katerem robot odkrije koncept stabilnosti, je podoben tistemu za premičnost. Robot je v prostoru z več predmeti dveh različnih oblik: žoge in kocke. Robot je vse predmete sposoben pobrati in prenašati naokoli (slika 3.2). Z akcijo `put(Time, Obj1, Obj2)` robot pobere predmet `Obj1` in ga odloži na `Obj2`. Argument `Time` ima enako vlogo kot pri akciji `move`. Robot ne pozna učinka akcije. Če je spodnji predmet kocka, zgornji predmet obstane na njem (stolp iz dveh predmetov stoji), v primeru, da je spodnji predmet žoga, pa zgornji predmet pade dol (stolp se podre). Robot v poskusu v naključnem redu poskuša akcijo na naključno izbranih parih predmetov. Vsakega od predmetov v akciji ne glede na vlogo uporabi le enkrat. Slednja omejitev skrbi za poenostavitev

```
move(T, Obj):-
  p1(Obj),
  f1(T, Obj).

move(T, Obj):-
  not p1(Obj),
  f2(T, Obj).

f1(T1, Obj):-
  at(T1, Obj, Pos1),
  at(T2, Obj, Pos2),
  Pos1 \== Pos2,
  {T2 = T1+1}.

f2(T, Obj):-
  not f1(T, Obj).

p1(Obj):-
  at(T1, Obj, Pos1),
  at(T2, Obj, Pos2),
  Pos1 \== Pos2.
```

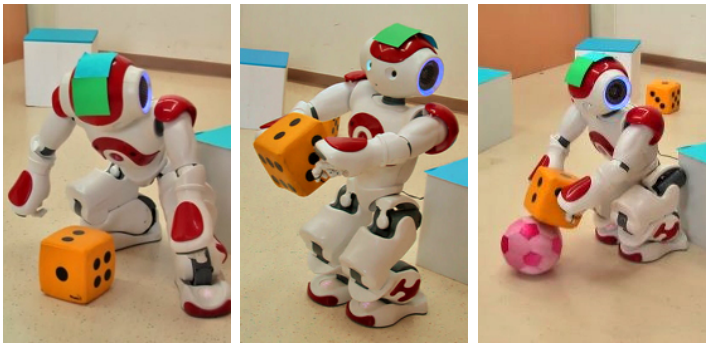
Program 3: Model akcije `move` z definicijo premičnosti.

scenarija, saj se s tem ognemo temu, da bi robot skušal pobirati ali odlagati na druge predmete, ki so že postavljeni v nekih stolpih. Robot, ko mu zmanjka predmetov na katerih lahko izvaja akcijo, inducira končni model (program 4).

Predikata `p1` in `p2` v modelu vsebujeta definiciji stabilnosti in nestabilnosti za to preprosto domeno kock in žog. Stolpi iz dveh predmetov, kjer je spodnji predmet škatla, so stabilni, taki, kjer je spodnji predmet žoga, pa nestabilni.

3.2.3 Prednosti in slabosti metode kvalitativni STRUDEL

Zaradi visokega nivoja abstrakcije se metoda kvalitativni STRUDEL dobro obnese v domenah, kjer je točen rezultat akcije težko napovedati. Primer take domene je odkrivanje koncepta stabilnosti predstavljene v podrazdelku 3.2.2. V primeru, ko robot postavi kocko na žogo, se kocka prevrne na tla in odkotali nekam po prostoru. Točno končno lokacijo, kjer bo kocka pristala, je skoraj nemogoče napovedati, tudi če bi robotu ponudili veliko več podatkov, kot jih ima na voljo. Ker metoda kvalitativni STRUDEL za vrednost posameznega argumenta literala v opisu stanja



Slika 3.2: Odkrivanje koncepta stabilnosti: robot z akcijo `put` prime en predmet in ga postavi na drugega.

loči le, če se je spremenila ali ostala enaka, v tej domeni loči le med tem, ali en predmet obstane na drugem, ali ne. Iz enakega razloga je ta pristop precej odporen na šum, saj abstrakcija odpravi velik del težav s točnostjo v podatkih. Dokler robot zazna spremembo, ko se ta pojavi, je vseeno, če ta ni natančno izmerjena.

Razvrščanje v skupine v metodi kvalitativni STRUDEL je računsko dokaj nezahtevno. V najslabšem primeru, ko so vse spremembe kvalitativno različne in vsaka spada v svojo skupino, moramo vsako spremembo primerjati z vsemi ostalimi. Vsako primerjavo lahko opravimo v $O(k \log k)$ časa, kjer je k število literalov v vsaki od sprememb (če število literalov v obeh spremembah ni enako, takoj vemo, da spremembi nista kvalitativno enaki). Zahtevnost celotnega razvrščanja je tako reda $O(n^2 k \log k)$, kjer je n število sprememb in k največje število literalov v posamezni spremembi.

Preprostost razvrščanja v skupine je hkrati tudi ena največjih pomanjkljivosti te metode. Vse spremembe se zmeraj popolnoma posplošijo in informacija o vrednostih argumentov se izgubi. To pomeni, da metoda kvalitativni STRUDEL ne razlikuje med spremembami, ki se med seboj razlikujejo le v vrednosti argumentov. To pomanjkljivost lahko odpravimo z uvedbo namenskih predikatov v predznanje ali z izboljšanjem robotovih sposobnosti opazovanja (da bo stanja opisoval z več in drugimi dejstvi), vendar z uvajanjem dodatnega predznanja lahko otežimo indukcijo pogojev.

Modeli, ki jih gradi metoda kvalitativni STRUDEL, so zaradi svoje kvalitativne narave zgolj opisni. Modela z definicijo premičnosti (program 3) robot ne more neposredno uporabiti za to, da bi splaniral, kako prestaviti predmet na neko želeno

```
put(T, Obj1, Obj2):-
  p1(Obj1, Obj2),
  f1(T, Obj1, Obj2).

f1(T1, Obj1, Obj2):-
  at(T1, Obj1, Pos1),
  at(T1, Obj2, Pos2),
  hasShape(T1, Obj2, S),
  at(T2, Obj1, Pos2),
  Obj1 \== Obj2,
  Pos1 \== Pos2,
  {T2 = T1+1}.

p1(Obj1, Obj2):-
  hasShape(T, Obj2, box).
```

```
put(T, Obj1, Obj2):-
  p2(Obj1, Obj2),
  f2(T, Obj1, Obj2).

f2(T1, Obj1, Obj2):-
  at(T1, Obj2, Pos1),
  at(T1, Obj1, Pos2),
  at(T2, Obj2, Pos3),
  at(T2, Obj1, Pos4),
  Obj1 \== Obj2,
  Pos1 \== Pos2,
  Pos1 \== Pos3,
  Pos1 \== Pos4,
  Pos2 \== Pos3,
  Pos2 \== Pos4,
  Pos3 \== Pos4,
 {T2 = T1+1}.

p2(Obj1, Obj2):-
  hasShape(T, Obj2, ball).
```

Program 4: Model akcije `put` z definicijo stabilnosti.

lokacijo. Možna je samo zelo neučinkovita raba. Lahko bi recimo naključno izvajal akcijo, ki spremeni lokacijo tega predmeta, in upal, da bo kdaj izpolnil želeni cilj. Še vedno pa to ne pomeni, da so ti kvalitativni modeli popolnoma neuporabni. Robot lahko z njihovo uporabo recimo ugotovi, s katerimi predmeti si lahko pomaga pri izpolnjevanju določene naloge. Ker bi bilo jalovo graditi plane, ki zahtevajo premikanje nepremičnih škatel, lahko na podlagi kvalitativnega modela take plane zavrže.

Vredno je izpostaviti, da smo z metodo kvalitativni STRUDEL uspeli odkriti informativne in razumljive koncepte, kot sta premičnost in stabilnost, čeprav deluje z zelo posplošeno obliko vhodnih podatkov. Celo tako visokonivojska obravnava

sprememb torej že zadošča, da lahko robot odkriva zanimive koncepte.

S tem zaključujemo pregled metode kvalitativni STRUDEL, ki predstavlja zgodnjo verzijo metode STRUDEL uporabljene v poskusih v tej disertaciji. V naslednjem razdelku opišemo delovanje aktualne metode STRUDEL, ki je uporabljena v poskusih.

3.3 Delovanje in implementacija metode STRUDEL

Metoda STRUDEL iz podatkov o robotskih akcijah in stanjih sveta med njimi zgradi model tipa STRIPS [32], ki opisuje učinke akcij in vsebuje novoodkrite abstraktne koncepte. V tem razdelku predstavimo delovanje metode in njeno implementacijo, kot je bila uporabljena v poskusih opisanih v poglavju 5. Posamezni deli tega razdelka ustrezajo posameznim stopnjam delovanja metode STRUDEL. Začnemo z opisom vhodnih podatkov v podrazdelku 3.3.1. V naslednjem podrazdelku, 3.3.2, opišemo, kako metoda STRUDEL razvrsti učinke akcij v skupine. Sledi opis učenja pogojev v podrazdelku 3.3.3 in zaključimo z razlago indukcije končnega modela v podrazdelku 3.3.4.

3.3.1 Vhodni podatki

Vhod v metodo STRUDEL sestoji iz zaporedja izvedenih akcij skupaj z njihovimi parametri, $A_1, \ldots, A_n$, in zaporedja stanj med njimi, $S_1, \ldots, S_{n+1}$, kjer je vsako stanje opisano z množico dejstev o robotovih opažanjih o svetu. Robot ima ob tem tudi predznanje B, podano kot množico logičnih formul.

Akcije so podane z dejstvi `action(Time, Action)`. Argument `Time` ne vsebuje časa v fizikalnem smislu, temveč zaporedno številko stanja v katerem je bila akcija izvedena. Argument `Action` vsebuje literal oblike `ime_akcije(...parametri akcije...)`, ki opisuje izvedeno akcijo.

Stanja so podana z dejstvi `state(Time, List_of_observations)`. Argument `Time` vsebuje zaporedno številko stanja v sosledju stanj po času. Argument `List_of_observations` pa je seznam dejstev, ki opisujejo opažanja o svetu. Katera dejstva so v seznamu opažanj, je odvisno od robotovih senzorjev in sposobnosti analize vhodnih podatkov.

Iz zaporedij akcij in stanj se v predprocesiranju konstruira zaporedje dejstev `effect(Time, Action, Adds, Dels)`, ki vsebujejo informacijo o učinkih posameznih akcij v obliki podobni pravilom za planiranje STRIPS. Za vsak par stanj, S_i in S_{i+1}, ter akcijo, A_i, oblikujemo eno dejstvo o učinku akcije. Argument `Time` ima vrednost i, torej številko stanja, ko je bila akcija `Action`, pobrana iz drugega argumenta A_i, izvedena. Argumenta `Adds` in `Dels` vsebujeta dejstva, ki jih akcija spremeni v stanju sveta. `Adds` vsebuje dejstva, ki jih akcija doda v stanje, $S_{i+1} \setminus S_i$, `Dels` pa dejstva, ki jih akcija odstrani iz stanja, $S_i \setminus S_{i+1}$.

3.3.2 Razvrščanje učinkov v skupine

Prvi korak metode STRUDEL je razvrstitev akcij v skupine. Splošno gledano lahko v tem koraku uporabimo poljubno razvrstitev, ki ustreza našemu namenu. Metoda bo v nadaljevanju poiskala pogoje, ki uvrščajo akcije v skupine, ki jih oblikujemo v tem koraku, glede na vrednosti argumentov akcij, in končni model bo za vsako skupino vseboval ločene stavke, ki bodo opisovali učinek akcij, ki sodijo v to skupino.

Naš pristop je, da učinke akcij razvrstimo na podlagi strukturne podobnosti učinkov akcij. V skupine razvrščamo dejstva `effect` konstruirana v pripravljalnih korakih metode. S strukturno podobnostjo mislimo podobnost v strukturi relacij, ki jih določajo dejstva v argumentih `Action`, `Adds` in `Dels` (natančno opisano v podrazdelku 4.2.1). Za razvrščanje uporabljamo algoritem ACES (*Action Clustering by Event Similarities*), ki je podrobno opisan v poglavju 4. Rezultat razvrščanja so množice $E_1, \ldots, E_k$, ki so disjunktno razbitje množice vseh učinkov akcij.

3.3.3 Učenje pogojev

V predhodnem koraku je metoda STRUDEL razvrstila učinke akcij v skupine E_1,...,E_k glede na strukturne podobnosti med njimi. Te konkretne skupine akcij iz učnih podatkov so edini rezultat razvrščanja. Do sedaj metoda še ni zgradila nobenega logičnega modela ali pravila, preko katerega bi bila sposobna nove, še nevidene primere akcij razvrščati v skupine. V ta namen v tem koraku inducira pogoje za posamezne učinke – predikate, ki bodo na podlagi argumentov akcije določili ali spada v posamezno skupino akcij z enakim učinkom. Natančneje, predpostavka metode STRUDEL je, da je razlika v učinku akcije posledica različnih

lastnosti predmetov uporabljenih v akciji. Lastnost je tu lahko dveh različnih tipov:

- inherentna, stalna lastnost predmeta, kot je recimo teža ali premičnost
- trenutna, spremenljiva lastnost, kot je lokacija predmeta ali pa neka relacija z ostalimi predmeti

Pogoji bodo tako odvisni samo od tistih argumentov akcije, ki ustrezajo uporabljenim predmetom, in od časa. Slednji je dodan, ker je lastnost predmeta, ki jo iščemo, lahko neka časovno spremenljiva relacija v kateri predmet trenutno nastopa.

Pogoji v poskusih predstavljenih v tej disertaciji se inducirajo z uporabo sistemov HYPER [7, 11] in HYPER/CA (poglavje 6). V splošnem bi lahko uporabili poljuben sistem ILP, vendar bi bilo potrebno ustrezno prirediti nastavitev učnega problema.

Ciljni predikat za pogoj, da akcija spada v skupino E_i je oblike:

```
pi(Time, ...objects from action arguments...)
```

Za vsako akcijo iz učne množice, ki pripada skupini E_i, ustvarimo pozitiven učni primer, ter za vsako, ki ne pripada tej skupini, negativen učni primer. Predznanje B za učenje pogoja sestoji iz predznanja, ki ga ima robot podanega vnaprej oziroma se ga je naučil predhodno (npr. aritmetika, fizikalni zakoni itd.), in podatkov o stanjih $S_1, \ldots, S_{n+1}$. Ker so stanja opisana znotraj dejstev `state` opisanih v podrazdelku 3.3.1, v predznanje B dodamo predikat `in_state` definiran v programu 5, ki omogoča, da se referenciramo na posamezna dejstva iz stanj.

```
in_state(Time, Observation):-
  state(Time, Observations),
  member(Observation, Observations).
```

Program 5: Definicija predikata `in_state`.

Naučeni predikati predstavljajo novo znanje, saj vsebujejo definicije lastnosti predmetov uporabljeni v akcijah, ki povejo, kakšen učinek bo imela akcija izvedena

z njimi. V poskusih (poglavje 5) predstavimo, kako se pri učenju oblikujejo pogoji, ki jih lahko interpretiramo kot definicije splošnih abstraktnih konceptov.

Če uspe učenje pogojev za vse skupine, lahko ta korak uspešno zaključimo. V primeru, da ne uspe učenje pogoja za eno od skupin, E_j, lahko pogoj `pj` zgradimo tako, da v njegovem telesu negiramo vse ostale inducirane pogoje. Če ni uspešno učenje več kot enega od pogojev se metoda neuspešno konča. Rezultat učenja pogojev so definicije predikatov `p1`,...,`pk`.

3.3.4 Indukcija končnega modela

V zadnjem koraku uporabimo rezultate predhodnih korakov za indukcijo končnega modela učinkov akcij oblike STRIPS. Za vsako skupino E_i induciramo predikata `adds` in `dels`, ki določata dejstva, ki jih akcija spremeni v stanju.

Ciljna predikata za učenje sta naslednje oblike:

```
adds(Action, Time, Adds)
dels(Action, Time, Dels)
```

Argumenti obeh teh predikatov so enaki kot v dejstvih `effect`. Teoretično ni razlogov, da ne bi mogli učinka akcije opisati v enem predikatu, ki bi določal tako dejstva, ki se dodajo, kot tista, ki se odstranijo. Za ločeno obliko smo se odločili iz dveh razlogov. V planiranju so *adds* in *dels* pravila praviloma ločena. Poleg tega se je v naših poskusih (poglavje 5) pokazalo, da telesi obeh predikatov pogosto nimata veliko skupnih literalov in je učinkoviteje inducirati dva manjša, bolj specializirana predikata, kot enega večjega.

Model tako induciramo v dveh ločenih fazah. V eni induciramo `adds` stavke za vse skupine ter v drugi `dels` stavke za vse skupine. Ker smo pogoje, ki ločijo med skupinami akcij, že inducirali v prejšnjem koraku, jih lahko vstavimo v začetno hipotezo za učenje. Primer, kako izgleda začetna oblika hipoteze za učenje `adds`, je prikazan v programu 6. Oblika za učenje `dels` je podobna.

Pozitivne primere za učenje predikatov `adds` in `dels` lahko konstruiramo iz dejstev `effect` tako da izpustimo četrti ali tretji argument. Težje je z negativnimi primeri. Če želimo modelirati vse možne učinke akcij, potem so negativni primeri nemogoči učinki, ki jih robot ne more v praksi izvesti in opazovati. Naš pristop bazira na pristopu iz [60] in rešuje to težavo tako, da tvori negativne primere

```
adds(Action, Time, Adds):-
  p1(Time, ...objects from action arguments...).

                    ...

adds(Action, Time, Adds):-
  pk(Time, ...objects from action arguments...).
```

Program 6: Začetna oblika hipoteze za učenje predikata `adds`.

na podlagi predpostavke, da je učinek akcije funkcija njenih vhodnih parametrov in je enolično določen. Generiranje negativnih primerov in učenje potekata po naslednjem iterativne postopku:

1. Najprej tvorimo množico naključnih negativnih primerov tako, da za vsak pozitiven primer ustvarimo enega z naključnim učinkom. Naključen učinek pomeni, da nastavimo naključne vrednosti vsem izhodnim argumentom akcije (tj. tistim argumentom, katerih vrednost je del učinka akcije – npr. končna lokacija predmeta pri akciji `move`) in naključno tvorimo seznama dejstev `Adds` ali `Dels`. Kateri argumenti akcije so izhodni je podano v predznanju. Seznam dejstev tvorimo tako, da sestavimo naključno dolg seznam literalov, ki ustrezajo vsem opažanjem, ki jih je robot sposoben zbirati v dani domeni. Maksimalna možna dolžina je parameter učenja in jo lahko nastavimo na največje število dejstev v katerikoli do sedaj videni spremembi. Vrednosti parametrov argumentov tako v akciji (izhodni argumenti) kot v opažanjih nastavimo naključno na neko do sedaj videno vrednost ustreznega tipa. Z uporabo teh naključnih negativnih vrednosti induciramo prvi model.

2. Začetni model bo zelo verjetno presplošen in bo poleg pravega učinka akcije dovolil tudi druge. Po predpostavki, da so učinki akcije enolično določeni, to pomeni, da je model potrebno bolje specificirati. V tem koraku preverimo, ali za vsak pozitiven primer model vrne le en, pravi učinek. V primeru, da je temu tako, zaključimo z učenjem in vrnemo trenutni model. V primeru,

da model generira še druge, neprave učinke, pa nadaljujemo na korak 3.

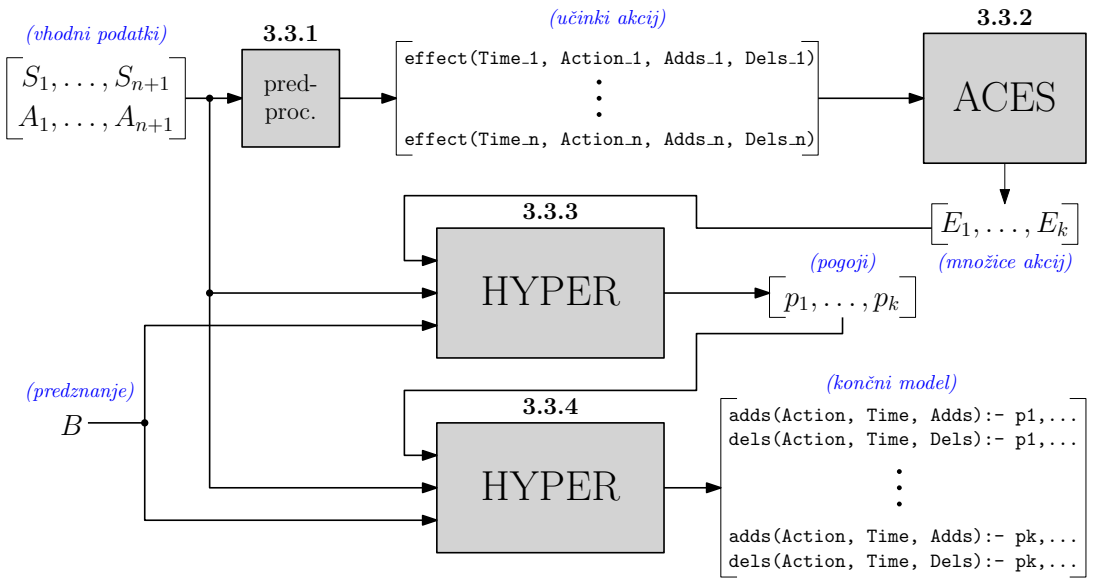
3. Če je model presplošen, je to posledica tega, da so naključno tvorjeni naključni primeri premalo specifični. Da preprečimo to obnašanje, za vsak pozitiven primer, kjer model generira tudi druge učinke poleg pravega, enega od nepravih učinkov dodamo med negativne primere. V primeru, da ima kateri od generiranih nepravih učinkov neopredeljene spremenljivke, jim določimo naključno izbrano vrednost ustreznega tipa iz učne množice. S to novo učno množico induciramo novi model in postopek ponovimo od koraka 2 naprej.

Predznanje za indukcijo v vseh korakih učenja modela je enako kot za učenje pogojev, torej vnaprej podano in predhodno naučeno predznanje B ter podatki o stanjih. Končni model, ki ga učenje vrne, sestoji iz vseh induciranih stavkov `adds` in `dels`, vseh pogojev `p1,...,pk` in predikatov iz predznanja, ki so uporabljeni v induciranih stavkih.

3.4 Povzetek opisa metode STRUDEL

Metoda STRUDEL iz podatkov o robotskih akcijah in stanjih sveta med njimi zgradi model, ki opisuje učinke teh akcij. Model vsebuje novoodkrite abstraktne koncepte, ki jih lahko uporabimo v nadaljnjem učenju. Vhodni podatki so neoznačeni in akcije metoda z algoritmom ACES najprej razvrsti v skupine glede na podobnosti njihovih učinkov. Za vsako skupino akcij nato inducira pogoj, ki loči akcije iz te skupine od ostalih. Ti pogoji vsebujejo definicije odkritih konceptov. Nazadnje inducira še končni model, ki sestoji iz `adds` in `dels` predikatov, ki opisujejo učinek akcij vsake skupine.

Shema celotne metode STRUDEL je grafično prikazana na sliki 3.3. Posamezni koraki so narisani kot sive škatle, ki imajo na vrhu številko podrazdelka v katerem so opisani. Škatla z imenom HYPER se pojavi dvakrat, ker gre za dve različni inducijski nalogi (indukcija pogojev in indukcija končnega modela). Podatki na vhodih/izhodih škatel so označeni s kratkimi imeni (modri napisi v poševni pisavi), ki razlagajo njihov pomen.



Slika 3.3: Grafična shema metode STRUDEL.

Kvalitativni STRUDEL je zgodnja verzija metode STRUDEL in slednja se od svoje predhodnice razlikuje v več lastnostih:

- Metoda STRUDEL gradi modele oblike STRIPS, metoda kvalitativni STRUDEL pa opisne modele.
- Modeli zgrajeni z metodo kvalitativni STRUDEL so kvalitativni, medtem ko modeli zgrajeni z metodo STRUDEL niso. Zaradi tega lahko slednje uporabimo za točno napoved naslednjega stanja.
- Metoda kvalitativni STRUDEL akcije razvrsti v skupine na podlagi preprostega ujemanja s posplošitvijo vseh spremenljivk. Metoda STRUDEL v ta namen uporablja algoritem ACES, ki je splošnejši.
- Končni model se pri metodi STRUDEL inducira z uporabo sistema ILP. Pri metodi kvalitativni STRUDEL pa se le oblikuje stavke iz pogojev in posplošenih literalov, ki opisujejo učinke akcij.

Razlika je velika in razen podobnosti v grobi zasnovi gre za skoraj popolnoma drugačno metodo.

Poglavje 4

Algoritem razvrščanja v skupine ACES

Eden ključnih korakov metode STRUDEL (poglavje 3) je razvrščanje učinkov akcij v skupine glede na podobnosti oz. razlike med njimi. To omogoči, da se pogojev, ki ločijo med različnimi učinki, naučimo vnaprej v ločenem koraku in ne tekom indukcije končnega modela. Da je to smotrno, mora biti razvrščanje v skupine učinkovitejše, kot če to delo prepustimo indukciji končnega modela, tako da želimo uporabiti pristop, ki bo znal razvrščanje opraviti brez preiskovanja vseh možnih delitev. Algoritem ACES – *Action Clustering by Effect Similarities* (razvrščanje akcij v skupine glede na podobnosti njihovih učinkov), ki ga predstavimo v tem poglavju, izkorišča strukturne podobnosti učinkov za izračun mere razdalje, ki jo uporabi za razvrščanje. Najprej v razdelku 4.1 podrobneje predstavimo učno nalogo, ki jo rešujemo z metodo ACES, nato pa v razdelku 4.2 opišemo, kako metoda deluje.

4.1 Učna naloga

Osnovna naloga metode ACES je razvrščanje akcij, ki jih je robot zbral s svojimi poskusi, v skupine glede na razlike v strukturi njihovih učinkov. Gre za nenadzorovano učno nalogo, ker avtonomen robot nima nobenega vira (učitelj, prerok (oracle) ipd.), ki bi mu označil, v katero skupino spadajo posamezne akcije. Skupine se oblikujejo na podlagi podobnosti v strukturi literalov, ki opisujejo učinke.

Ker s tem razvrsti v skupine tudi same učinke akcij, zaradi tega v disertaciji pri metodi ACES včasih govorimo o razvrščanju akcij in včasih o razvrščanju njihovih učinkov, saj gre za isti proces.

Učinke akcij razberemo iz razlike med opažanji, ki jih robot razbere v stanju pred akcijo in stanju po akciji. Pri tem moramo upoštevati še parametre akcij, saj je pomemben tudi namen robota. Na primer, če robot izvede akcijo `kick(a)`, s katero želi brcniti žogo `a`, ločimo, ali pri tem opazi, da se je premaknila žoga `a`, ali da se je premaknila neka druga žoga `b`. Pri drugem učinku so bile okoliščine akcije očitno drugačne (npr. žoga `b` je bila robotu pri brci v napoto).

Vhodni podatek za razvrščanje je množica literalov:

```
effect(Action, Time, Adds, Dels),
```

ki jih predhodno konstruira STRUDEL. Argument `Time` je za razvrščanje nepomemben in se v tej fazi ignorira. Izhod je disjunktno razbitje vhodne množice učinkov v podmnožice podobnih učinkov, $E_1, \ldots, E_k$.

4.2 Delovanje

Metoda ACES razvrsti literale `effect` na podlagi strukturnih podobnosti grafov, ki jih določajo množice literalov v argumentih `Action`, `Adds` in `Dels`. Kako tvorimo grafe iz množic literalov in na podlagi njih izračunamo razdaljo med primeri, opišemo v podrazdelku 4.2.1. Kako to funkcijo razdalje uporabimo za razvrščanje pa razložimo v podrazdelku 4.2.2.

4.2.1 Funkcija razdalje

Vsak učinek akcije in s tem primer, ki ga razvrščamo, je opisan z množico literalov, ki so podani kot argumenti v `effect` literalih. To so: literal, ki določa akcijo in njene parametre (`Action`), literali, ki jih akcija v stanje doda (`Adds`), in literali, ki jih akcija iz stanja izbriše (`Dels`). Gre za strukturirane, relacijske podatke, katerih strukturo lahko predstavimo z grafom, zgrajenim po navodilih opisanih v nadaljevanju.

Metoda ACES meri razdaljo med učinki na podlagi strukturnih razlik med grafi, ki jih predstavljajo. Ideja mere razdalje med grafi, ki jo opišemo v nadaljevanju razdelka, temelji na ti. preureditveni razdalji (*angl. edit distance*) [35]. Meri

torej število potrebnih operacij (dodajanje vozlišča, odstranjevanje vozlišča itd.), da en graf spremenimo v drugega. Ker želimo res primerjati le strukturo učinkov, metoda ACES loči med dvema tipoma argumentov: nominalni in številka. Vrednosti vseh argumentov, ne glede na tip, so v grafu predstavljene z vozlišči. Vrednost nominalnega argumenta v učinku vedno imenuje eno samo stvar (predmet, pojem itd.), ne glede na to, kolikokrat je uporabljena v literalih. Pri primerjavi strukture učinkov oz. njihovih grafov lahko torej preprosto štejemo, ali je vozlišče prisotno ali ne. Vrednost številskega argumenta pa izraža numerično vrednost in je ne moremo šteti na tak način. Lahko bi uvedli neko mero za razliko med dvema numeričnima argumentoma (na primer na nek način normirano absolutno razliko), vendar ta ne bi merila strukturne razlike med grafoma, temveč razliko med vrednostmi znotraj vozlišč. Ni očitno, kako bi to razliko v vrednosti argumentov v splošnem upoštevali v ustreznem sorazmerju s strukturno razliko, zato numeričnih argumentov ne upoštevamo pri oceni razlik in uporabljamo le "čisto" strukturno razliko. To v splošnem ni korektno in zmanjša učno moč metode ACES in posledično metode STRUDEL, saj se robot ne more naučiti konceptov, kjer so kritične numerične vrednosti posameznih argumentov. Ta pomanjkljivost žal ostaja nerešena in jo prepuščamo za nadaljnje delo v poglavju 7.

Pravila za gradnjo grafa iz množice literalov so sledeča:

1. Za vsak argument arg, ki se pojavi v kateremkoli od literalov, ustvarimo vozlišče v_{arg}. Pri tem ločimo dva primera glede na dva tipa podatkov, nominalni in številka. Pri nominalnih podatkih ustvarimo za posamezno vrednost le eno vozlišče, ne glede na to, kolikokrat se pojavi kot argument v literalih. Pri številkah za vsak argument ustvarimo novo vozlišče, ne glede na vrednost. Tipi argumentov posameznih literalov so del vnaprej podanega predznanja.

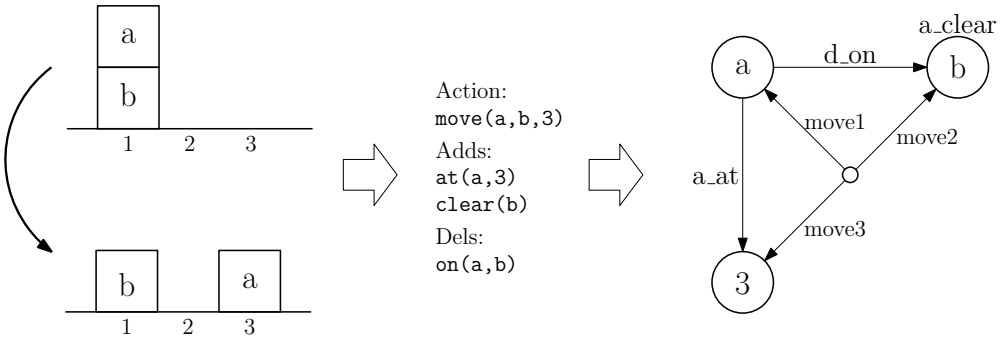
2. Za vsak literal z enim argumentom ustvarimo oznako vozlišča. V primeru, da vozlišče za argument literala še ne obstaja, ustvarimo tudi vozlišče. Če gre za literal iz `Adds`, dodamo oznaki predpono `a_`, in, če gre za literal iz `Dels`, dodamo predpono `d_`. Če gre za literal iz argumenta `Action`, predpona ni potrebna.

3. Za vsak literal z dvema argumentoma, $arg1$ in $arg2$, ustvarimo usmerjeno povezavo med vozliščema v_{arg1} in v_{arg2}. Usmerjenost je pomembna zaradi

vrstnega reda argumentov v literalu. Oznaka povezave je enaka imenu funktorja literala. Če gre za literal iz `Adds`, dodamo oznaki predpono `a_`, in, če gre za literal iz `Dels`, dodamo predpono `d_`. Če gre za literal iz argumenta `Action`, predpona ni potrebna.

4. Za vsak literal z mestnostjo več kot dva dodamo anonimno vozlišče s katerim so povezani vsi njegovi argumenti. Povezave so usmerjene stran od anonimnega vozlišča. Oznake teh povezav so sestavljene podobno kot pri pravilu 3, le da na konec oznake dodamo številko, ki ustreza zaporedni številki argumenta v literalu.

Primer gradnje grafa na podlagi množice literalov je na sliki 4.1. Naj opozorimo, da je lokacija škatle na tleh numeričen argument (enodimenzionalna koordinata), čeprav uporabljamo le cele vrednosti. Zato med literali tudi ni `clear` literalov za lokacije. V tem odstopamo od tipične konvencije v "blocks world" domenah, a podani primer tako bolje predstavi principe gradnje grafa po danih pravilih.



Slika 4.1: Primer gradnje grafa iz strukturiranih podatkov, ki opisujejo akcijo.

Razdaljo med dvema učinkoma akcij, e_i in e_j, izračunamo kot negativno vrednost podobnosti med grafoma g_i in g_j zgrajenima na podlagi literalov, ki ju opisujejo (enačba 4.1). Da se izognemo negativnim vrednostim razdalj, vsem podobnostim prištejemo maksimalno vrednost podobnosti izmed vseh parov učinkov v domeni. Naj na tem mestu opozorimo, da funkcija *dist* formalno ni metrika, tako da je naša raba izraza "razdalja" morda malo neustrezna. Za ta izraz smo

se vseeno odločili, ker je zaradi njegove ustaljenosti razumljivost danega besedila boljša.

$$dist(e_i, e_j) = \max_{x,y \in \{g_1,\ldots,g_n\}, x \neq y} sim(x, y) - sim(g_i, g_j) \tag{4.1}$$

Podobnost $sim(g_i, g_j)$ med grafoma se izračuna na podlagi največjega izomorfnega podgrafa g'_{max}, ki ga imata g_i in g_j skupnega, pri čemer upoštevamo, da se morajo ujemati tudi oznake povezav in vozlišč. Poiskati moramo torej tako prirejanje μ argumentov iz e_i in e_j, pri katerem je ujemanje grafov maksimalno (enačba 4.2). Prirejajo se le vrednosti nominalnih argumentov.

$$sim(g_i, g_j) = \max_{\mu \in \text{match}(\text{args}_{(e_i)}, \text{args}_{(e_j)})} \text{matchGraph}(\mu(g_i), g_j) \tag{4.2}$$

Stopnjo ujemanja grafov matchGraph($\mu(g_i)$,g_j) z danim prirejanjem μ izračunamo tako, da ujemanja štejemo pozitivno, elemente iz posameznega grafa, ki se ne ujemajo z elementi v drugem grafu pa negativno. Podrobneje:

a) Za vozlišča, ki ustrezajo nominalnim argumentom, za vsak par vozlišč $v \in \mu(g_i)$,$v' \in g_j$ in $v = v'$ stopnji ujemanja prištejemo 2 ter ji za vsako vozlišče iz $\mu(g_i)$ in g_j, ki v drugem grafu nima para z enako vrednostjo, odštejemo 1. Vozlišč, ki ustrezajo številkam ne štejemo.

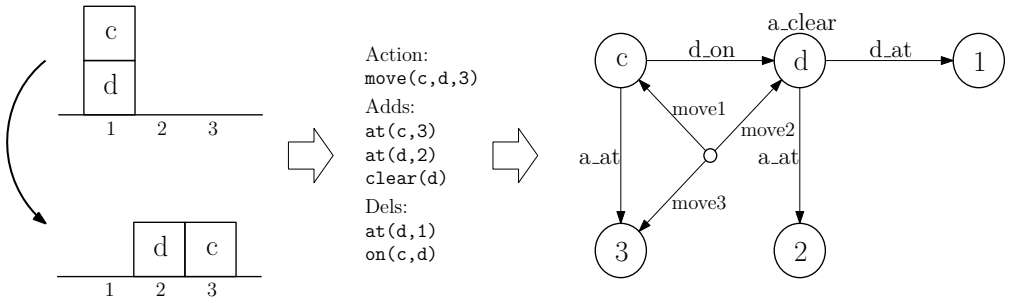
b) Za vsak par vozlišč $v \in \mu(g_i)$ in $v' \in g_j$ z enako oznako stopnji ujemanja prištejemo 2, za vsako vozlišče z oznako, ki mu v drugem grafu ni prirejeno vozlišče z enako oznako, pa ji odštejemo 1.

c) Za vsako par povezav $p(v_{arg_1}, v_{arg_2}) \in \mu(g_i)$ in $p'(v_{arg'_1}, v_{arg'_2}) \in g_j$, ki imata enaki oznaki, kjer nobeno od krajiščnih vozlišč ni anonimno in za oba para istoležnih vozlišč, v_{arg_h} in $v_{arg'_h}$, velja, da sta arg_h in arg'_h nominalna in $arg_h = arg'_h$ ali pa sta števili, stopnji ujemanja prištejemo 2. Pri ujemanju števil torej ne preverjamo vrednosti. Za vsako povezavo, ki v drugem grafu nima povezave, ki ustreza zgornjemu pogoju, stopnji ujemanja odštejemo 1.

d) Naj bo $P = \{p_1, \ldots, p_m\}$ množica vseh povezav iz grafa $\mu(g_i)$, ki imajo krajišče v istem anonimnem vozlišču in ustrezajo eni večmestni relaciji, ter naj bo $V_P = \{v_{arg_k}; \text{v } v_{arg_k} \text{ je krajišče } p_k, p_k \in P\}$ množica ne-anonimnih vozlišč, ki so krajišča povezav iz P. Če obstaja v g_j množica povezav $P' = \{p'_1, \ldots, p'_m\}$,

ki imajo eno krajišče v skupnem anonimnem vozlišču in se s povezavami iz P paroma ujemajo v oznakah, katerih krajišča so v ne-anonimnih vozliščih $V'_P = \{v_{arg'_k}; \text{v } v_{arg'_k} \text{ je krajišče } p'_k, p'_k \in P'\}$, pri čemer velja, da se vsak par arg'_k z arg_k ujema v nominalni vrednosti ali pa sta števili, stopnji ujemanja prištejemo 2. Za vsako množico povezav P, za katero v drugem grafu ne obstaja množica P', ki ustreza zgornjemu pogoju, stopnji ujemanja odštejemo 1. Večmestne relacije torej štejemo v celoti.

Opisana mera je sorodna tisti iz algoritma za odkrivanje podstruktur SUBDUE [18, 46] v tem, da temelji na ideji, da se razdaljo med dvema grafoma meri v številu operacij (dodajanje/odstranjevanje vozlišč, dodajanje/odstranjevanje povezav itd.), ki so potrebne, da en graf spremenimo v drugega. To je tako imenovana preureditvena razdalja (*angl. edit distance*). Vendar, obstaja razlika: preureditvena razdalja meri le razlike med grafoma (tj. le negativni del naše mere podobnosti), naša mera pa upošteva tudi del grafov, ki jima je skupen. Intuicija za tem je, da je enako velika absolutna razlika pri večjih grafih bolj zanemarljiva (morda celo posledica šuma), saj nam velik skupni del daje več zaupanja v podobnost. Gledano s stališča učinkov akcij je za dva "velika" učinka (tj. opisana z veliko literali), ki se razlikujeta v nekaj literalih, manj verjetno, da sta si podobna po naključju, kot za dva manjša učinka, ki se razlikujeta v enakem številu literalov.



Slika 4.2: Graf učinka podobne akcije, kot je prikazana na sliki 4.1, le da se spodnja kocka med akcijo premakne.

Recimo, da želimo izračunati podobnost grafa g_1 iz slike 4.1 in grafa g_2 iz slike 4.2, ki opisuje podobno akcijo, le da se spodnja kocka po nesreči premakne. Maksimalno stopnjo ujemanja grafov g_1 in g_2 dosežemo pri prirejanju $\mu = \{(a \to c), (b \to d)\}$. Če ovrednotimo najprej ujemanje grafov, štejemo:

- $+4$ po pravilu a) za para prirejenih vozlišč (a, c) in (b, d).
- $+2$ po pravilu b), ker imata prirejeni vozlišči b in d enako oznako `a_clear`.
- $+4$ po pravilu c). Od tega $+2$ za povezavi med vozliščema a in b iz g_1 in med vozliščema c in d iz g_2, ker so krajišča prirejena vozlišča in imata enako oznako. Drugih $+2$ pa za povezavi med vozliščema a in 3 iz g_1 in med vozliščema c in 3 iz g_2, ker sta en par njunih krajišč prirejeni vozlišči, drugi par krajišč pa oboje števili.
- $+2$ po pravilu d za množici povezav z oznakami `move1`, `move2` in `move3` iz obeh grafov, ker so njihova končna krajišča prirejena vozlišča ali števila in je njihovo skupno izvorno krajišče neimenovano vozlišče.

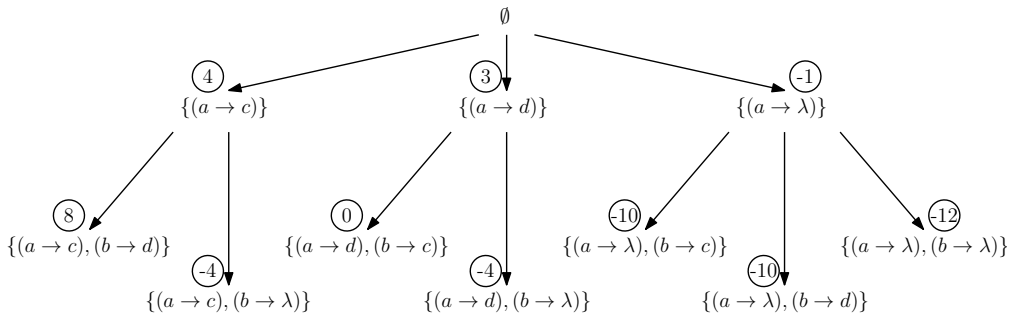
Za ovrednotenje neujemanja grafov, štejemo:

- -2 po pravilu c) za povezavi med vozliščema d in 1 ter vozliščema d in 2 iz grafa g_2, ker v grafu g_1 nimata ustreznih povezav.

Končna vrednost podobnosti grafov g_1 in g_2 je tako:

$$sim(g_1, g_2) = 4 + 2 + 4 + 2 - 2 = 12 - 2 = 10.$$

Formalno je prirejanje preslikava $\mu : args_1 \rightarrow args_2 \cup \{\lambda\}$, kjer sta $args_1$ in $args_2$ množici argumentov literalov, ki opisujejo primerjana učinka, ter λ označuje prirejanje z ničemer. Argumenti iz $args_1$, ki se jim v μ priredi λ, se ne preslikajo v nobenega iz $args_2$. Prirejanje argumentov μ, pri katerem je stopnja ujemanja grafov največja, poiščemo z izčrpnim preiskovanjem prostora možnih prirejanj. Primer prostora za grafa g_1 in g_2 je prikazan na sliki 4.3. Računska zahtevnost tega preiskovanja pri učinkih e_1 z n_{e_1} nominalnimi argumenti in e_2 z n_{e_2} nominalnimi argumenti (kjer je $n_{e_2} \geq n_{e_1}$) je v splošnem reda $O(n_{e_1}^{n_{e_2}+1})$. Če so nominalni argumenti različnih podtipov (npr. ločimo predmete glede na obliko), potem lahko to zahtevnost zmanjšamo s prirejanjem le tistih argumentov, ki se ujemajo v podtipu. Ker so učinki akcij v naših poskusih (poglavje 5) relativno "majhni", eksponentna računska zahtevnost ni bila problem. V primeru večjih učinkov bi lahko uporabili pristop razmeji in omeji (*angl. branch and bound*) kot v metodi SUBDUE ali kako drugo metodo za učinkovito preiskovanje. Obstaja tudi več aproksimativnih algoritmov za iskanje ujemanja grafov [1, 40], ki jih lahko uporabimo v primeru, da bi ta del postal ozko grlo algoritma.



Slika 4.3: Prostor preiskovanja možnih prirejanj. Poleg prirejanj je trenutna vrednost `matchGraph`, ki se pri notranjih vozliščih izračuna na podlagi podgrafa napetega na vozliščih, ki so že v trenutnem prirejanju.

4.2.2 Razvrščanje v skupine

Funkcijo razdalje opisano v podrazdelku 4.2.1 lahko uporabimo v poljubnem algoritmu za razvrščanje v skupine. V poskusih (poglavje 5) smo uporabili aglomerativno hierarhično razvrščanje v skupine, kjer se razdalja med skupinami izračuna kot povprečje razdalj elementov skupin (average linkage) [115].

Razlog za izbor te metode razvrščanja v skupine je boljša razumljivost modelov, saj nam hierarhičen model bolje razloži podobnost med posameznimi učinki. Ker moramo za nadaljnje učenje izbrati konkretno razdelitev na skupine, potrebujemo nek kriterij, kje odrezati zgrajeni hierarhičen model. Obstaja več načinov, kako določiti ta prag, lahko recimo uberemo pristop z maksimiranjem povprečne podobnosti znotraj skupin, kot ga predlagajo v sorodnih pristopih za razvrščanje relacijskih podatkov v skupine [48]. V poskusih predstavljenih v tej disertaciji se osredotočamo na binarne koncepte, tako da model vedno odrežemo na vrhu in razdelimo učinke na dve skupini.

Na tej točki gre omeniti še eno dobro posledico upoštevanja tudi podobnosti in ne le razlik grafov v meri razdalje (intuicijo za tem smo že razložili v podrazdelku 4.2.1). V primerih, ko je učinek akcije možno opisati rekurzivno, ta mera tipično loči robne primere od ostalih.

Vzemimo primer, ko je robot v prostoru z več stolpi iz kock in lahko izvede akcijo `push(Box)`, s katero potisne spodnjo kocko v stolpu, kar povzroči, da se kocke zložene na njej, podrejo na tla. Rekurziven model učinka te akcije je predstavljen

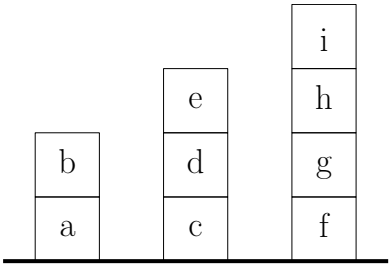
```
effect(push(Box1), Time,
       [on(Box2, floor)], [on(Box2, Box1)]):-
  p(Box1, Time),
  in_state(Time, on(Box2, Box1)).

effect(push(Box1), Time,
       Adds, Dels):-
  \+ p(Box1, Time),
  in_state(Time, on(Box2, Box1)),
  delete(on(Box2, Box1), Dels, Dels1),
  delete(on(Box2, floor), Adds, Adds1),
  effect(push(Box2), Time, Adds1, Dels1).

p(Box1, Time):-
  in_state(Time, on(Box2, Box1)),
  clear(Box2).
```

Program 7: Rekurziven model, ki opisuje učinke robotove akcije `push`. Če robot potisne spodnjo škatlo v stolpu, se vse zgornje kocke podrejo na tla.

v programu 7. Vidimo, da model različno obravnava robni primer in vse ostale, za kar skrbi pogoj `p(Box1, Time)`. Če hočemo ta pogoj odkriti vnaprej, nam mora razvrščanje v skupine vrniti to delitev primerov. Privzemimo, da je robot akcijo izvedel na treh različnih stolpih (slika 4.4).



Slika 4.4: Trije stolpi iz kock, ki jih robot podre s svojo akcijo `push`.

Učinki teh treh akcij so sledeči:

```
e1: effect( push(a), 1, [on(b,floor)],
                        [on(b,a)]).
e2: effect( push(c), 2, [on(d,floor), on(e,floor)],
                        [on(d,c), on(e,d)]).
e3: effect( push(f), 3, [on(g,floor), on(h,floor), on(i,floor)],
                        [on(g,f), on(h,g), on(i,h)]).
```

V tabeli 4.1 so podobnosti in razdalje med učinki `e1`,`e2` in `e3`. Vidimo, da je razdalja med učinkoma `e2` in `e3` najmanjša in bi ju v delitvi na dve skupini aglomerativno razvrščanje uvrstilo v eno skupino, učinek `e1` pa v drugo. Ta delitev je ravno tista, ki ustreza pogoju `p` iz modela. Do podobne delitve bi prišlo tudi, če bi robot izvedel tudi poskuse z višjimi stolpi.

sim:			
	e2	9	
	e3	6	15
		e1	e2

dist:			
	e2	6	
	e3	9	0
		e1	e2

Tabela 4.1: Tabeli podobnosti in razdalje med učinki `e1`,`e2` in `e3`.

4.3 Povzetek opisa algoritma ACES

Algoritem ACES razvrsti akcije v skupine glede na strukturne razlike in podobnosti med njihovimi učinki. Gre za nenadzorovano učno metodo, ki znotraj metode STRUDEL omogoča, da se robot uči samostojno in ne potrebuje označenih primerov akcij. Vsakemu učinku akcije algoritem ACES priredi usmerjen graf. Razdalja

med učinki in posledično akcijami se izračuna na podlagi ujemanja in razlike med temi prirejenimi grafi. Za samo razvrščanje v skupine na podlagi izračunanih razdalj se uporabi aglomerativno hierarhično razvrščanje v skupine.

Algoritem se od sorodnih metod ključno razlikuje v tem, da v funkciji razdalje upošteva tudi stopnjo ujemanja učinkov akcij in ne le njihovih razlik. To je v našem kontekstu bolj intuitivno kot alternativa, saj je smiselno, da ima enako velika razlika med učinki pri siceršnjem precejšnjem ujemanju manjšo težko, kot če je to ujemanje manjše. Pozitivna posledica tega je tudi ugodno obnašanje algoritma pri učinkih z rekurzivnim opisom, kjer ločuje robne primere od ostalih. Žal trenutna verzija funkcije razdalje ne upošteva numeričnih in ordinalnih atributov, ki nekateri sorodni pristopi jih.

Poglavje 5

Poskusi učenja modelov akcij z metodama STRUDEL in ACES

V tem poglavju predstavimo štiri domene v katerih z uporabo metode STRUDEL zgradimo modele akcij, ki vsebujejo abstraktne koncepte. Najprej v razdelku 5.1 opišemo uporabljeno metodologijo, ki je skupna vsem domenam, nato pa v razdelku 5.2 podrobno predstavimo še vsako posamezno domeno. Na koncu komentiramo rezultate učenja v razdelku 5.3.

5.1 Metodologija

Za gradnjo modelov akcij v domenah uporabimo metodo STRUDEL, kot je opisana v poglavju 3. Za razvrščanje v skupine v prvem koraku metode STRUDEL je uporabljena metoda ACES opisana v poglavju 4. Po predpostavki se osredotočamo na binarne koncepte, zato metoda ACES primere vedno razdeli v dve skupini. Obe metodi nimata nobenih za domeno specifičnih parametrov in sta v vseh domenah uporabljeni na enak način.

Za odkrivanje pogojev in indukcijo končnega modela v predzadnjem in zadnjem koraku metode STRUDEL, uporabimo sistem HYPER [7, 11]. Uporabljamo nadgrajeno različico programa HYPER [55] razvito v okviru projekta XPERO z nekaj manjšimi spremembami. Delovanje programa HYPER je opisano v razdelku 6.1.1 skupaj s podrobnostmi o nadgradnjah in spremembah v verziji, ki jo uporabljamo, v primerjavi s "čistim" programom HYPER. V zadnji domeni (podrazdelek 5.2.4)

je predstavljen tudi rezultat dobljen z metodo HYPER/CA, opisano v poglavju 6.

Sistem HYPER za delovanje potrebuje negativne primere. Kot opisano v podrazdelkih 3.3.3 in 3.3.4, jih pri odkrivanju pogojev enostavno dobimo iz skupin akcij, pri indukciji končnega modela pa jih moramo generirati. Pristop, ki ga uporabimo, temelji na pristopu iz [60]. Definicijo posameznega `adds` in `dels` predikata najprej induciramo z naborom, sestavljenim iz dveh tipov negativnih primerov opisanih v nadaljevanju. Primeri za učenje `adds` so oblike `adds(Action, Time, Adds)` in primeri za učenje `dels` oblike `dels(Action, Time, Dels)`. Negativne primere za oba predikata tvorimo tako, da vzamemo nek pozitiven primer in spremenimo vrednosti izhodnih argumentov akcije ter argumenta `Adds` oz. `Dels`. Izhodni argument akcije je tisti, katerega vrednost je del učinka akcije (npr. končna lokacija predmeta `Pos2` pri akciji `move(Box, Pos1, Dist, Pos2)`). Kateri argumenti akcij so izhodni, robot ve iz predznanja. Tvorimo:

- **popolnoma naključne primere**: Vsak popolnoma naključni primer se generira tako, da se vrednost izhodnih argumentov akcije in argumentov `Adds` oz. `Dels` določi naključno. Izhodnim argumentom se vrednost določi iz zaloge vrednosti glede na tip argumenta. Zaloga vrednosti se določi v predznanju (npr. razponi vrednosti za številske argumente, kot so koordinate škatel) ali pa jo sestavljajo vrednosti iz učne množice (npr. vse škatle v domeni). `Adds` in `Dels` se generirata tako, da se generira naključno število literalov, s katerimi robot opisuje svoja opažanja, a ne več kot je najdaljša dolžina `Adds` oz. `Dels` v učnih podatkih. Argumenti teh generiranih literalov (opažanj) se generirajo na enak način, kot izhodni argumenti akcij. Pri generaciji se preverja, da slučajno ne generiramo primera, ki je enak kateremu od pozitivnih primerov. Zaradi predpostavke, da je učinek akcije funkcija vhodnih parametrov akcije, vemo, da je generirani primer negativen. Teh primerov generiramo 100.

- **bližnje naključne primere**: Bližnji naključni primer je primer, ki se od nekega pozitivnega razlikuje v vrednosti le enega od izhodnih argumentov ali argumenta `Adds` oz. `Dels`. Natančneje, generiramo jih tako, da za vsak argument vsakega od izhodnih argumentov akcije in argument vsakega literala v `Adds` oz. `Dels` vsakega pozitivnega primera p_+ generiramo en negativen primer, ki ima vrednost tega argumenta določeno naključno, a drugačno kot

je v p_+, vrednost ostalih argumentov pa je enaka kot v p_+. Poleg tega generiramo še po en negativen primer za vsak literal iz `Adds` oz. `Dels`, kjer izbrišemo ta literal, ostale pa pustimo nespremenjene. Zadnji negativen primer generiramo tako, da `Adds` oz. `Dels` dodamo naključno generiran literal. Število bližnjih negativnih primerov je tako $|p_+| * (n_{args} + n_{lits} + 1)$, kjer je $|p_+|$ število pozitivnih primerov (število akcij v učnem naboru) in n_{args} število izhodnih argumentov akcije in argumentov literalov v `Adds` oz. `Dels` posameznega pozitivnega primera in je n_{lits} število literalov v `Adds` oz. `Dels`. Naključne vrednosti posameznih argumentov in naključno tvorjeni literali se določijo enako kot pri popolnoma negativnih primerih. Tudi tu na podlagi predpostavke, da je učinek akcije funkcija vhodnih parametrov akcije, vemo, da je generirani primer negativen.

Število začetnih naključnih primerov je po naših poskusih lahko precej manjše, pa bi bilo učenje še vedno uspešno (testov kakšen je minimalen nabor primerov nismo delali). Prav tako generiranje bližnjih naključnih primerov ni nujno. Ima pa ta začetni nabor vpliv na to, koliko iteracij učenja bo potrebnih (kolikokrat bomo generirali dodatne negativne primere), in zgoraj opisani nabor je bil izbran tako, da je obnašanje učenja med različnimi zagoni stabilno in enako. V vsaki iteraciji učenja, dokler je inducirani predikat presplošen, po postopku opisanem v podrazdelku 3.3.4 generiramo po en negativni primer za vsak pozitivni primer, kjer vrne model poleg pravega še napačne učinke. Generiranje naključnih primerov je računsko nezahtevno in nima bistvenega vpliva na zahtevnost celotnega postopka.

Naštete metode so sprogramirane v programskem jeziku prolog [11], natančneje v jeziku SICStus prolog [16] (verzija 4.2.0). Prolog je uporabljen tudi kot opisni jezik vhodnih podatkov in končnega modela.

Podatki v vseh domenah so umetni in ne vsebujejo šuma. Poskusov na fizičnem robotu nismo opravili. Vsi izračuni so bili izvedeni na prenosniku s procesorjem Intel Core 2 Duo 2.40 GHz in 4 GB rama.

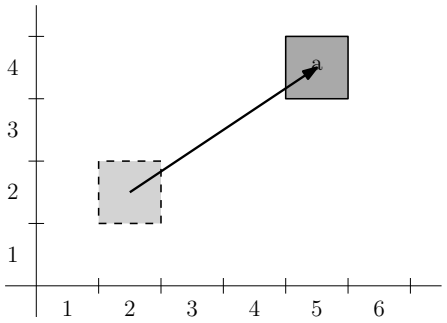
5.2 Domene poskusov

Štiri domene predstavljene v tem razdelku so poimenovane po abstraktnih konceptih, ki jih metoda STRUDEL odkrije pri gradnji modela akcij. Vsaka od domen

je opisana v svojem podrazdelku. Definicije domen za sistem HYPER in podatki uporabljeni za učenje modelov opisanih v tem poglavju so podani v dodatku A.

5.2.1 Učenje koncepta premičnosti

Ta domena je podobna, kot je opisana v razdelku o poskusih opravljenih z metodo kvalitativni STRUDEL (podrazdelek 3.2.2) in odkrivanje koncepta premičnosti je že bilo izvedeno v okviru projekta XPERO [60]. Robot je v prostoru z več škatlami razporejenimi po tleh. Nekatere škatle so lahke in jih robot lahko premakne, druge pa težje in jih ne more premakniti, vendar s svojimi senzorji ni sposoben neposredno ločiti težjih škatel od lažjih. Robot zna izvesti akcijo `move(Box, Pos1, Dist, Pos2)`, s katero škatlo `Box` premakne z začetne pozicije `Pos1` za razdaljo `Dist` na pozicijo `Pos2` (primer na sliki 5.1). Robot učinka akcije ne pozna in se ga z metodo STRUDEL nauči iz podatkov.



Slika 5.1: Primer učinka akcije `move(Box(a, [2,2], [3,2], [5,4]))`.

V poskusu je uporabljenih 8 škatel, 5 premičnih in 3 nepremične. Učni nabor obsega podatke o 10 akcijah in 11 stanjih. Vsaka od škatel je uporabljena v vsaj eni od akcij, ena premična in ena nepremična škatla v dveh. Vsako stanje vsebuje podatke o lokaciji vseh škatel zapisanih v dejstvih `at(Box, Pos)`. Učni podatki in programi z definicijami učnih problemov uporabljeni v tem poskusu so v razdelku A.1 na strani 103.

Metoda STRUDEL zgradi končni model zapisan v programu 8. Gradnja modela, ki vključuje razvrščanje v skupine z metodo ACES, indukcijo pogoja, ki ločuje med skupinama učinkov, in vse iteracije indukcije končnega modela (predikatov `adds` in `dels`), traja manj kot sekundo.

```
adds(move(Box,Pos1,Dist,Pos2),
     Time1, [at(Box,Pos2)]):-
  movable(Box),
  add(Pos1, Dist, Pos2).

adds(move(Box,Pos1,Dist,Pos1),
     Time1, []):-
  \+ movable(Box).

movable(Box):-
  in_state(Time1, at(Box, Pos1)),
  in_state(Time2, at(Box, Pos2)),
  different(Pos1, Pos2).
```

```
dels(move(Box,Pos1,Dist,Pos2),
     Time1, [at(Box,Pos1)]):-
  movable(Box),
  add(Pos1, Dist, Pos2).

dels(move(Box,Pos1,Dist,Pos1),
     Time1, []):-
  \+ movable(Box).
```

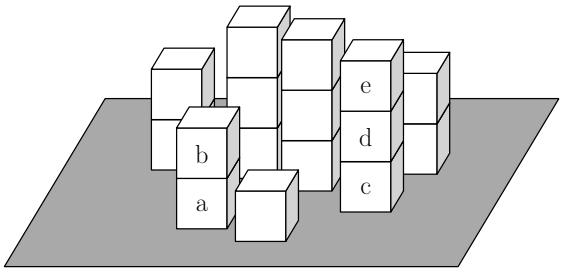
Program 8: Model akcije `move` z definicijo premičnosti v predikatu `movable`.

Končni model je ekvivalenten tistemu iz [60] (opisan v podrazdeleku 3.2.2), le da je v obliki STRIPS. V predikatu `movable` prepoznamo definicijo koncepta premičnosti: *"Škatla je premična, če je kdaj bila na dveh različnih mestih."* Metoda STRUDEL samega imena predikata ni izumila, temveč uporabi splošno ime `p`. V modelu smo ga z `movable` poimenovali mi zaradi boljše razumljivosti. Vidimo lahko še, da `movable` nima argumenta `Time`, za katerega smo rekli, da ga metoda STRUDEL doda v glavo ciljnega predikata pri učenju pogoja. Tudi pri učenju `movable` je bil ta argument dodan, vendar ga je metoda STRUDEL na koncu odstranila, saj v telesu predikata ni uporabljen. To pomeni, da gre za trajno lastnost predmeta, ki ni vezana na neko trenutno stanje sveta.

5.2.2 Učenje konceptov "ne najvišje" in "nižje kot"

Osnovna ideja te domene izvira iz članka [10]. Robot z akcijo `grab(Box1, Box2)` zvrha pobira škatle, ki so zložene v stolpe na ravni površini (slika 5.2). Argument `Box1` je škatla, ki jo robot hoče pobrati, `Box2` pa škatla, ki jo dejansko uspe pobrati ali `none`, če ne uspe zgrabiti nobene škatle. Robot skuša pobirati le vrhnje škatle

v stolpih. Ker je robotovo prijemalo v primerjavi s škatlami precej večji, lahko doseže le kocke, ki so od vseh najvišje – tj. vrhnje škatle na najvišjih stolpih. Stolpi škatel so zloženi tako tesno skupaj, da se sicer njegova roka fizično zatakne ob višje, ko skuša doseči spodnje, vendar pri tem ne premakne nobene od kock. Če pogledamo primer postavitve na sliki 5.2 (to ni postavitev uporabljena v poskusih), lahko robot na primer zgrabi kocko e, kocke b pa ne. Kocke, ki jih v akciji uspešno zgrabi, so odstranjene iz postavitve.



Slika 5.2: Škatle zložene v stolpe na ravni površini.

V poskusu so podatki o 13 škatlah zloženih v stolpe. Učni nabor vsebuje 12 akcij in 13 stanj. Postavitev kock je opisana s predikati `on(Box1, Box2)` in `on_floor(Box)`, odstranjene kocke pa naštejemo v dejstvih `removed(Box)`. Učni podatki in programi z definicijami učnih problemov uporabljeni v tem poskusu so v razdelku A.2 na strani 111.

Končni model, ki ga metoda STRUDEL zgradi, je zapisan v programu 9. Model je zgrajen v 68 s, kjer skoraj ves ta čas vzame indukcija pogoja (tj. predikata `not_highest`). Razvrščanje v skupine ter indukcija `adds` in `dels` skupaj trajajo reda eno sekundo.

S predikatom `not_highest`, ki je pogoj za to, da akcija nima učinka, je definiran koncept "ne najvišje": *"Škatla a trenutno ni najvišje, če obstaja neka druga škatla b, od katere je a nižje."* Rekurzivni predikat `lower_than`, uporabljen v pogoju, ni bil podan vnaprej, temveč je bil induciran hkrati z `not_highest`. Pri indukciji pogoja smo sistemu HYPER v predznanju z nastavkom ponudili možnost, da definira pomožni predikat, ki opisuje neko relacijo med dvema škatlama. Rezultat je definicija koncepta "nižje kot" v predikatu `lower_than`, ki se uporabi v definiciji pogoja. Podani nastavek za relacijo je seveda zelo ugoden za učenje v tej domeni in učenju tu z njim zelo pomagamo. V splošnem robot ne mora vna-

```
adds(grab(Box, none), Time,
     []):-
  not_highest(Box, Time).

adds(grab(Box, Box), Time,
     [removed(Box)]):-
  \+ not_highest(Box, Time).

not_highest(Box, Time):-
  lower_than(Box, Box1, Time).

lower_than(Box1, Box2, Time):-
  in_state(Time, on_floor(Box1)),
  in_state(Time, on(Box2, Box3)).

lower_than(Box1, Box2, Time):-
  in_state(Time, on(Box1, Box3)),
  lower_than(Box3, Box4, Time),
  in_state(Time, on(Box2, Box4)).
```

```
dels(grab(Box, none), Time,
     []):-
  not_highest(Box, Time).

dels(grab(Box1, Box1), Time,
     [on(Box1, Box2)]):-
  \+ not_highest(Box1, Time),
  in_state(Time, on(Box1, Box2)).
```

Program 9: Model akcije `grab` z definicijo "ne najvišje" v predikatu `not_highest` in "Nižje kot" v predikatu `lower_than`.

prej vedeti, kateri oziroma kakšen nastavek mu bo prišel prav. To je del širšega problema izbire relevantnega predznanja, ki ga opišemo v razpravi v poglavju 7. Vidimo, da sta oba inducirana predikata (in koncepta, ki ju definirata) odvisna od argumenta `Time`. To je smiselno, saj je ključna relativna pozicija škatel, ki se v času spreminja.

Človeku je negirana oblika koncepta "ne najvišje" najbrž neintuitivna in bi kot pogoj raje uporabil preprostejšo verzijo brez negacije, "najvišje". Sistem HYPER negirano obliko dosti lažje poišče, ker se pri ne-negirani verziji negacija prestavi v telo predikata pred predikat `lower_than`. Definicija "najvišje" bi se torej glasila:

"Škatla a je trenutno najvišja, če ne obstaja nobena druga škatla b, od katere je a nižje." Zaradi narave negacije v jeziku prolog – gre za ti. negacijo preko neuspeha (angl. negation by failure) – sistem HYPER sedaj dosti težje poišče definicijo pomožnega predikata `lower_than`, saj po njegovi cenilni funkciji tekom preiskovanja delujejo obetavne tudi popolnoma nesmiselne in nedelujoče oblike tega predikata.

5.2.3 Učenje koncepta obteženosti

Ta domena je dokaj podobna domeni v podrazdelku 5.2.1, v kateri robot odkrije koncept premičnosti. Tudi tu ima robot na voljo akcijo `move(Box, Pos1, Dist, Pos2)`, ki deluje enako kot prej. Glavna razlika je v tem, da so sedaj vse škatle sicer enako težke, vendar so zložene v stolpe. Postavitev je torej podobna tisti na sliki 5.2. Robot potiska le najnižje škatle v stolpih (na sliki 5.2 sta to recimo škatli a in c) in premik škatle ter s tem celotnega stolpa mu uspe le, če so na škatlo, ki jo potiska, zložene manj kot tri škatle. V primeru stolpov visokih štiri ali več škatel je skupna teža stolpa prevelika in robot ni dovolj močan, da bi ga lahko prestavil. Robot torej sedaj lahko na podlagi svojih opažanj o postavitvi kock loči premične kocke od nepremičnih.

V poskusu so podatki o 15 škatlah zloženih v stolpe. Učni nabor vsebuje 7 akcij in 8 stanj. Postavitev kock je opisana s predikati `on(Box1, Box2)` in `at(Box, Pos)`. Za kocke na tleh tako poznamo lokacijo, za višje kocke pa na kateri kocki stojijo. Učni podatki in programi z definicijami učnih problemov uporabljeni v tem poskusu so v razdelku A.3 na strani 119.

Metoda STRUDEL v času osem sekund zgradi model zapisan v programu 10. Tudi v tej domeni večino časa vzame indukcija pogoja (predikata `unmovable`), medtem ko preostale operacije skupaj trajajo reda eno sekundo.

V tej domeni metoda STRUDEL kot pogoj za učinek akcije definira nepremičnost škatel. Razlog za negirano obliko pogoja v primerjavi z domeno iz podrazdelka 5.2.1 je enak kot pri odkrivanju koncepta "ne najvišje" (podrazdelek 5.2.2). Nepremičnost je definirana z uporabo rekurzivnega predikata `weighted`, ki bi moral biti v definiciji premičnosti negiran, a bi ga potem sistem HYPER bistveno težje odkril. Predikat `weighted` definira koncept obteženosti kocke s tem, da prešteje kocke zložene nanjo. Samo štetje se izvede preko rekurzivnega dvigovanja po stolpu kock po relacijah `on` in hkratnega sprehoda po seznamu dolžine tri. Če

```
adds(move(Box,Pos1,Dist,Pos1),
  Time, []):-
  unmovable(Box, Time).

adds(move(Box,Pos1,Dist,Pos2),
  Time, [at(Box, Pos2)]):-
  \+ unmovable(Box, Time),
  add(Pos1, Dist, Pos2).

dels(move(Box,Pos1,Dist,Pos1),
  Time, []):-
  unmovable(Box, Time).

dels(move(Box,Pos1,Dist,Pos2),
  Time, [at(Box, Pos1)]):-
  \+ unmovable(Box, Time),
  add(Pos1, Dist, Pos2).

unmovable(Box, Time):-
  weighted(Box, [Box1,Box2,Box3], Time).

weighted(Box, [], Time).

weighted(Box, [Box1|Boxes], Time):-
  in_state(Time, on(Box2, Box)),
  weighted(Box2, Boxes, Time).
```

Program 10: Model akcije `move` z definicijo nepremičnosti v predikatu `unmovable` in obteženosti v predikatu `weighted`.

je kock v stolpu dovolj, da rekurzija doseže konec seznama, predikat `unmovable` uspe. Sistem HYPER `weighted` inducira kot pomožni predikat, ki opisuje relacijo med škatlo in množico drugih škatel. Tudi tu sistemu HYPER nastavek za relacijo med škatlo in množico škatel podamo mi, in mu s tem pomagamo podobno kot pri učenju "nižje kot".

Zelo zanimivo je, da `weighted` ne izraža relacije med škatlo iz akcije in neko množico konkretnih škatel, temveč spremenljivk iz seznama v glavi ne veže z nobeno konkretno vrednostjo (atomom). Če pogledamo kako sistem HYPER v modelu določi mejo, kateri stolpi so pretežki, v telesu `unmovable` vidimo, da zgradi seznam dolžine tri, s katerim določi mejo, od koder naprej so stolpi nepremični. Ker se spremenljivke iz tega seznama v predikatu `weighted` ne vežejo, to pomeni, da njihova vrednost ni pomembna, in je res pomembna le kardinalnost seznama.

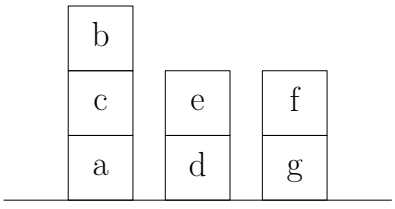
Efektivno robot torej izumi število, saj bi s prirejanjem s tem seznamom lahko preveril, ali poljuben drug seznam vsebuje tri elemente. `weighted(Box, X, Time)` šteje, koliko škatel je ob času `Time` zloženih na škatlo `Box` in nam v spremenljivki `X` vrne višino vseh (pod)stolpov zloženih na `Box`.

5.2.4 Učenje koncepta prôstosti

Robot je v tej domeni soočen s podobno situacijo kot pri odkrivanju "ne najvišje" (podrazdelek 5.2.2). Tudi tu ima množico kock postavljenih v stolpe in jih zvrha jemlje stran z akcijo `pickup(Box1, Box2)`. Prvi argument akcije, `Box1`, je škatla, ki jo hoče pobrati, drugi, `Box2`, pa škatla, ki jo dejansko uspe pobrati. Ključna razlika je, da tokrat lahko skuša pobrati poljubno kocko, vendar mu bo vedno uspelo zgrabiti le vrhnjo kocko iz stolpa v katerem je ciljna kocka. Na primer, če skuša na sliki 5.2 pobrati kocko c, bo v resnici uspel pobrati kocko e. Stolpi stojijo dovolj narazen, da njihova višina ne igra vloge.

V poskusu so podatki o 13 škatlah zloženih v stolpe. Učni nabor vsebuje 8 akcij in 9 stanj. Postavitev kock je opisana s predikati `on(Box1, Box2)` in `on_floor(Box)`, odstranjene kocke pa naštejemo v dejstvih `removed(Box)`. Učni podatki in programi z definicijami učnih problemov uporabljeni v tem poskusu so v razdelku A.4 na strani 127.

V tej domeni razvrščanje v skupine z metodo ACES ne vrne optimalnih skupin in posledično sistem HYPER, ki ne deluje s šumnimi podatki, ne uspe inducirati pogoja. Pri razvrščanju v dve skupini bi si želeli delitev na učinke, kjer je ciljna škatla na vrhu stolpa, in učinke, kjer ciljna škatla ni vrhnja. Po pregledu vhodnih podatkov vidimo, da ima v tem primeru druga skupina poseben primer, ko je ciljna škatla tik nad vrhnjo, ki ga metoda ACES ne razvrsti v želeno skupino. Vzemimo za primer enostavno situacijo na sliki 5.3.

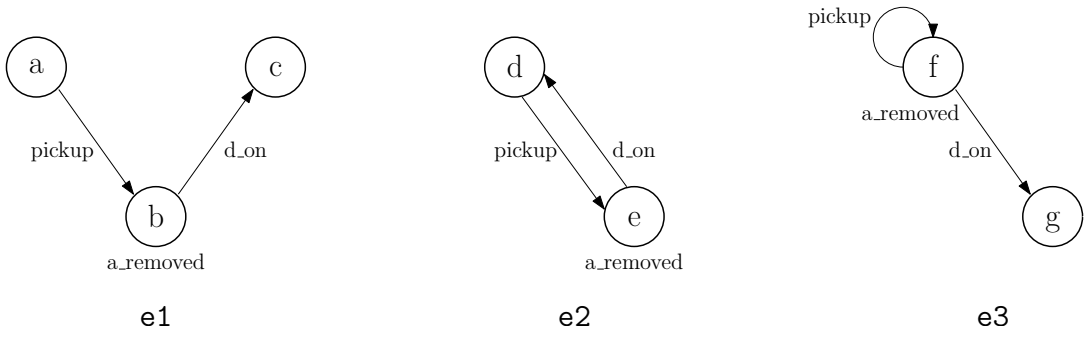


Slika 5.3: Preprosta situacija s škatlami postavljenimi v tri stolpe.

Trije različni učinki predstavljeni v okviru te postavitve so sledeče oblike:

```
e1: effect(pickup(a, b), _, [removed(b)], [on(b,c)]).
e2: effect(pickup(d, e), _, [removed(e)], [on(e,d)]).
e3: effect(pickup(f, f), _, [removed(f)], [on(f,g)]).
```

Iz grafov (slika 5.4), ki ustrezajo tem trem učinkom, postane razvidno, zakaj so učinki `e2` razvrščeni v isto skupino kot `e3` in ne v isto kot `e1`, kot bi želeli.V učinku `e1` nastopa ena škatla več kot v drugih dveh, zaradi česar sta si po meri razdalje metode ACES bližje (tabela 5.1).



Slika 5.4: Grafi treh različnih učinkov akcije `pickup`.

sim:			
	e2	5	
	e3	5	6
		e1	e2

dist:			
	e2	1	
	e3	1	0
		e1	e2

Tabela 5.1: Tabeli podobnosti in razdalje med učinki akcije `pickup`.

To težavo lahko rešimo na dva načina. Lahko spremenimo razvrščanje v skupine ali pa učni algoritem. Primer prvega je, da opustimo predpostavko o binarnih učinkih in razdelimo učinke na tri skupine ter jih ločeno obravnavamo. Mi smo izbrali drugo možnost in zamenjali sistem HYPER, ki ne obravnava šuma, z njegovo nadgrajeno različico sistemom HYPER/CA (poglavje 6), ki deluje tudi s šumnimi podatki. Če z njim induciramo pogoj, metoda STRUDEL v 43 s inducira model zapisan v programu 11, od česar 2 s traja indukcija `adds` stavkov in 40 s indukcija `dels` stavkov.

```
adds(pickup(Box, Box),
     Time, [removed(Box)]):-
  clear(Box, Time).

adds(pickup(Box1, Box2),
     Time, Adds):-
  \+ clear(Box1, Time),
  in_state(Time, on(Box3, Box1)),
  adds(pickup(Box3, Box2),
  Time, Adds).

clear(Box1, Time):-
  \+ in_state(Time, on(Box2, Box1)).
```

```
dels(pickup(Box1, Box1),
     Time, [on(Box1, Box2)]):-
  clear(Box1, Time),
  in_state(B, on(Box1, Box2)).

dels(pickup(Box1, Box2),
     Time, Dels):-
  \+ clear(Box1, Time),
  in_state(Time, on(Box3, Box1)),
  dels(pickup(Box3, Box2),
  Time, Dels).
```

Program 11: Model akcije `pickup` z definicijo prôstosti v predikatu `clear`.

Koncept prôstosti (predikat `clear`), ki ga metoda STRUDEL odkrije v tej domeni, je zelo preprost: *Kocka je prosta, če na njej ni nobene škatle.* Bolj zanimivi sta definiciji predikatov `adds` in `dels`, ki sta v tem modelu rekurzivni. Če ju interpretiramo, je okvirno delovanje pri obeh enako. Če je škatla, ki jo skušamo z akcijo `pickup` pobrati prosta, potem jo odstranimo iz postavitve. V primeru, da škatla, ki jo skušamo z akcijo `pickup` pobrati ni prosta, pa je učinek enak, kot če bi skušali pobrati škatlo nad njo. Prek rekurzije model tako pravilno napove, da vedno zgrabimo vrhnjo škatlo v stolpu.

5.3 Komentar rezultatov

V opisanih domenah smo prikazali učne sposobnosti metode STRUDEL. Vidimo, da lahko iz majhnega števila poskusov zgradi razumljiv relacijski model, ki vsebuje novoodkrite abstraktne koncepte. To je pomembna prednost, saj si v robotiki pogosto težko privoščimo veliko število poskusov. Robot mora namreč v praksi vse poskuse fizično izvesti, kar je lahko zamudno, poleg tega pa smo lahko omejeni še z dejavniki kot je omejen robotov vir energije (baterija).

Tudi dosežena hitrost učenja (tj. časovna zahtevnost) je zadovoljiva. Najdaljši čas učenja je 68 s v domeni "nižje kot", kar v področju učenja ILP, kjer imajo algoritmi praviloma eksponentno računsko zahtevnost, ni zelo dolg učni čas. Glavni del prihrankov pri računski zahtevnosti je posledica delitve učenja pogoja in celotnega modela na dve fazi. Če se direktno primerjamo s pristopom indukcije modela brez metode STRUDEL iz [60], vidimo, da je tam indukcija modela akcije `move` s konceptom premičnosti trajala 58 minut, nam pa uspe v manj kot 1 s. Seveda je potrebno biti pri tej primerjavi previden. Uspeh pristopa STRUDEL temelji na uspešnosti razvrščanja učinkov v skupine z metodo ACES v prvem koraku. Kot jasno pokaže učenje koncepta prôstosti, je razvrščanje lahko neuspešno ali celo zavajajoče že v domenah, kjer šum ni prisoten. V takih primerih lahko učenje z metodo STRUDEL postane neizvedljivo. Pristop s splošno indukcijo ni omejen z (ne)uspešnostjo razvrščanja v skupine in sam preko indukcije pogoje poišče ustrezno delitev učinkov.

Zelo pomemben dejavnik pri sposobnosti učenja in računski zahtevnosti je predznanje, ki ga podamo sistemu ILP uporabljenem v metodi STRUDEL. V predstavljenih domenah je predznanje omejeno na relevantne predikate in dejstva. Če bi sistem HYPER, ki je uporabljen za učenje, podali neustrezno ali preobsežno predznanje (npr. veliko število za domeno nerelevantnih predikatov) bi lahko učenje postalo neizvedljivo zaradi zahtevnosti. Gre za splošen problem ILP (in pravzaprav celotnega strojnega učenja). Težava je, da mora v splošnem izbor relevantnega predznanja opraviti robot sam. V tej disertaciji rešitve tega problema ne podamo in jo v zaključku (poglavje 7) pustimo odprto za nadaljnje delo.

Zgrajeni modeli so zaradi relacijske narave in logičnega jezika izrazno močni in preprosto interpretabilni. Kljub temu, da oblika zgrajenih predikatov ni vedno točno taka, kot bi jo spisal človek (npr. negirana oblike koncepta "ne najvišje"), odkritih konceptov ni težko razumeti in prepoznati. Metoda STRUDEL je sposobna graditi rekurzivne modele, pri čemer je rekurziven lahko tako pogoj (domeni "ne najvišje" in obteženost) kot sam opis učinka (domena prôstost). Rekurzivna oblika modela je splošnejša od nerekurzivne. Primer tega je rekurzivni opis učinka akcije `pickup`, ki velja za poljubno visoke stolpe kock. Brez rekurzije bi robot (brez dodatnega predznanja) moral v tej domeni za vsako različno število kock zloženih med ciljno in vrhnjo kocko zgraditi svoja predikata `adds` in `dels`.

Poglavje 6

Sistem induktivnega logičnega programiranja HYPER/CA

Robot za opis sveta in svojega znanja potrebuje visokonivojski relacijski jezik. Motivacija in razlogi za to so opisani v uvodu disertacije (poglavje 1). Mi v ta namen uporabimo logično programiranje, paradigmo deklarativnega programiranja, ki temelji na logiki prvega reda. Posledično potrebujemo metodo učenja, ki je sposobna graditi modele v tem jeziku. Za naše potrebe je še posebej ustrezno induktivno logično programiranje (ILP), ki je poleg tega sposobno dobro izkoristiti podano predznanje.

V metodi kvalitativni STRUDEL (razdelek 3.2) in v večini naših poskusov (poglavje 5), ki nimajo šumnih podatkov, smo uporabili sistem HYPER, ki ne obravnava šuma. Ker v realnih domenah ne moremo pričakovati brezšumnih podatkov in ker celo v umetnih domenah brez šuma razvrščanje v skupine ne deluje popolno (primer je odkrivanje koncepta prôstosti v podrazdelku 5.2.4), smo potrebovali še nek sistem, ki bi deloval na podatkih s šumom. Obstaja več sistemov ILP, ki imajo to sposobnost, a niso ustrezni iz drugih razlogov. Tipično potrebujejo vnaprej podano pričakovano stopnjo šuma, česar pa avtonomen robot sam ne more določiti. Da bi odpravili to pomanjkljivost in obdržali način delovanja, ki je čim bolj podoben sistemu HYPER, s katerim je bila metoda STRUDEL razvita, smo razvili sistem HYPER/CA nadgradnjo sistema HYPER, ki deluje na šumnih podatkih.

To poglavje je organizirano na sledeč način. V razdelku 6.1 najprej predstavimo

sistem HYPER, nato pa opišemo kako smo ga nadgradili v sistem HYPER/CA. Poglavje zaključimo z ovrednotenjem s poskusi v razdelku 6.2.

6.1 Sistem HYPER/CA

Sistem HYPER, iz katerega je bil razvit sistem HYPER/CA, je bil prvič predstavljen v [7] in njegova osnovna verzija je podrobno predstavljena v učbeniku prof. Bratka [11]. Tej verziji pravimo osnovna, ker obstaja že več spremenjenih in izboljšanih verzij, ki so bile uporabljene v različne namene. Za nas sta pomembni dve različni nadgrajeni verziji sistema HYPER. Prva je verzija razvita v okviru projekta XPERO [55], ki združuje več tehničnih izboljšav ter je verzija, ki smo jo uporabili v naših poskusih. Tu se bomo nanjo iz praktičnih razlogov sklicevali z imenom HYPER/X. Druga je sistem HYPER/N [81], verzija sistema HYPER, ki deluje s šumnimi podatki, a potrebuje pričakovano stopnjo šuma podano vnaprej. V tem razdelku najprej v podrazdelku 6.1.1 predstavimo, kako deluje osnovni sistem HYPER in tehnične nadgradnje iz sistema HYPER/X, nato pa v podrazdelku 6.1.2 opišemo naš prispevek z nadgradnjo v sistem HYPER/CA. Ker slednji vsebuje tudi izboljšave iz prej omenjenih nadgrajenih verzij, sproti izpostavimo, katere izboljšave iz prejšnjih verzij smo uporabili.

6.1.1 Izhodišče razvoja: sistem HYPER

Sistem HYPER je neprekriven algoritem ILP, ki s preiskovanjem drevesa izostritev glede na podane primere poišče popolno in konsistentno hipotezo zapisano kot program v jeziku prolog. Vhodni podatki za splošni učni problem ILP, ki ga rešuje, so:

- Množica pozitivnih učnih primerov, E_+, in množica negativnih učnih primerov, E_-.
- Predznanje, B, izraženo kot množica predikatov in dejstev. Da je indukcija sploh potrebna mora veljati: $B \not\models E_+$.

Za dane vhodne podatke sistem HYPER poišče hipotezo H, izraženo kot množica logičnih stavkov, za katero velja, da je:

- **popolna** – pokrije vse pozitivne primere: $B \wedge H \models E_+$
- **konsistentna** – ne pokrije nobenega negativnega primera: $B \wedge H \not\models E_-$

Končno hipotezo sistem HYPER poišče s preiskovanjem grafa izostritev. To je usmerjen acikličen graf, katerega koren je neka splošna začetna hipoteza, ki jo definira uporabnik. Ponavadi je to majhna množica (eden ali nekaj) preprostih stavkov s praznim telesom. Glave teh stavkov so vse oblike `p(A1,A2...)`, kjer je `p` ciljni predikat, argumenti `A1,A2...` pa različne spremenljivke. Vozlišča grafa ustrezajo različnim hipotezam, povezave pa trem izostritvenim operacijam:

1. prilagajanje dveh spremenljivk enakega tipa iz hipoteze
2. izostritev spremenljivke v izraz (seznam, konstanta itd.) glede na pravila podana v predznanju
3. dodajanje literala iz predznanja v hipotezo

Vsaka izostritev je specializacija trenutne hipoteze, tako da izostrena pokrije kvečjemu manj primerov. To pomeni, da lahko pri preiskovanju upoštevamo samo popolne hipoteze, saj nepopolne nikoli ne bomo izostrili v popolno. Tej optimizaciji navkljub je kombinatorična kompleksnost grafa še vedno velika.

Neprekriven pristop in gradnja celotne hipoteze naenkrat sicer pomenita večjo računsko zahtevnost v primerjavi z bolj razširjenimi prekrivnimi algoritmi. Vendar omogočata sistemu HYPER učenje rekurzivnih hipotez ali takih, kjer je potrebno inducirati več različnih, medsebojno odvisnih predikatov naenkrat.

Preiskovanje prostora poteka po načelu "najboljši najprej" (best-first). Cenilna funkcija, ki sistemu HYPER služi kot hevristika pri preiskovanju, je zapisana v enačbi 6.1. Funkcija upošteva tako velikost kot tudi točnost hipoteze.

$$cost(H) = \#vars(H) + 10 \cdot \#lits(H) + 10 \cdot \#negCover(H) \tag{6.1}$$

kjer je $\#vars(H)$ število različnih spremenljivk v hipotezi, $\#lits(H)$ število literalov v hipotezi in $\#negCover(H)$ število negativnih primerov, ki jih pokrije hipoteza. Preiskovanje se ustavi, ko najde popolno in konsistentno hipotezo, ali pa,

ko je izpolnjen nek drug ustavitveni pogoj, ki preprečuje preiskovanje v nedogled. Lahko omejimo število hipotez, ki jih lahko preiskovanje izostri, in/ali omejimo maksimalno dovoljeno velikost hipoteze.

Leban je v okviru projekta XPERO razvil več tehničnih nadgradenj sistema HYPER [55] in jih združil v verzijo, ki jo tu imenujemo HYPER/X. Ker smo v poskusih (poglavje 5) uporabili njegovo verzijo z manjšimi spremembami, tu naštevamo vse nadgradnje:

1. **Cenilna funkcija:** Pri cenilni funkciji originalnega sistema HYPER se je pri večjem številu negativnih primerov lahko zgodilo, da je zadnji del, ki šteje pokrite negativne dele, premočno vplival na oceno in kompleksnost hipoteze praktično ni imela vloge. Da bi to odpravili, se ta člen zamenja s $100 \cdot \#negCover(H)/\#allNeg$, kjer je $\#allNeg$ število vseh negativnih primerov. Konstanta 100 je določena glede na število vseh učnih primerov.

   Ta verzija tudi predpostavi, da je bolj verjetno, da bodo v hipotezi uporabljeni različni literali, zato dodatno kaznuje ponavljanje enakih literalov (tj. literalov z enakim funktorjem in enakim številom argumentov enakih tipov in v istem vrstnem redu).

   Poleg naštetih sprememb smo v cenilno funkcijo sistema HYPER/X dodali še en člen, ki ga v Lebanovi verziji ni: $(\#allVars(H) - \#vars(H))\,/\#allVars(H)$. Ta člen ceni doda delež ponovljenih spremenljivk izmed vseh spremenljivk. Namenjen je temu, da cenilna funkcija izmed hipotez, ki se razlikujejo le po prilagajanju spremenljivk, preferira tiste z manj prilagojenimi spremenljivkami. Tako skrbi, da sistem HYPER/X izbere najbolj splošno obliko hipoteze in ne prilagaja spremenljivk po nepotrebnem. Ker je vrednost tega člena v intervalu $[0, 1]$, vrednosti ostalih členov pa so cela števila, sicer ne vpliva na razvrstitev hipotez.

2. **Shranjevanje dokazov:** Sistem HYPER testira popolnost hipotez s svojim interpreterjem hipotez, ki preveri, ali je nek dani pozitivni oz. negativni primer možno izpeljati s trenutno hipotezo. Ker se interpreter za posamezno hipotezo kliče večkrat (enkrat za vsak primer), je smiselno shranjevati dele dokazov hipotez, da mu ni potrebno vedno znova dokazovati istih ciljev.

3. **Ovrednotenje izostrenih hipotez na podmnožicah negativnih pri-**

merov: Kadar hipotezo H izostrimo v H', bo slednja pokrila kvečjemu tiste primere, ki jih je pokrila H. Tako nam, ko računamo vrednost cenilne funkcije na H', ni potrebno preveriti vseh negativnih primerov, temveč le tiste, ki jih je pokrila H. Seveda to pomeni, da moramo voditi evidenco pokritih negativnih primerov za vsako hipotezo.

4. **Hramba generiranih hipotez v dejstvih v jeziku prolog:** To je edina od izboljšav, ki poveča zahtevnost sistema HYPER. Dodana je bila, ker je program HYPER zaradi težav s prologovim čiščenjem pomnilnika (*angl. garbage collection*) občasno prekinil preiskovanje in zavrgel ves napredek. Ker se vse generirane hipoteze preko `assert` klicev vpišejo v dejstva v jeziku prolog, lahko ponovno poženemo preiskovanje od točke, kjer je bilo prekinjeno.

5. **Preverjanje ekvivalence hipotez:** Če sistem HYPER tekom preiskovanja po več različnih poteh pride do iste hipoteze, tega ne zazna in jo razvije večkrat. Temu se ognemo s preverjanjem ali je bila trenutna hipoteza že kdaj razvita.

6. **Pogoji za prilagajanje spremenljivk:** V predznanju lahko pri opisu posameznih literalov določimo, da neke spremenljivke istega tipa ne smejo biti prilagojene. S tem lahko zmanjšamo kombinatorično kompleksnost prostora hipotez.

V tem razdelku smo opisali sistem HYPER in našteli tehnične spremembe ter nadgradnje, ki so bile implementirane v sistemu HYPER/X. V naslednjem podrazdelku opišemo na kakšen način smo nadgradili osnovni sistem HYPER v sistem HYPER/CA.

6.1.2 Nadgradnja v sistem HYPER/CA

V tem podrazdelku podamo opis sistema HYPER/CA. Ker je ta v svoji osnovi nadgradnja sistema HYPER, podrobno opišemo le spremembe, ki smo jih implementirali. Tu gre poudariti, da izhajamo iz osnovne verzije sistema HYPER (kot je opisan v [11]) in da v sistemu HYPER/CA niso uporabljene vse tehnične izboljšave iz verzije HYPER/X naštete v podrazdelku 6.1.1. Uporabljene izboljšave bomo navedli sproti, kjer pridejo v poštev.

Glavni namen naše nadgradnje je omogočiti sistemu HYPER delovanje s šumnimi podatki. Samo ime HYPER/CA pomeni: HYPER, ki optimizira klasifikacijsko točnost (CA; classification accuracy). V ta namen moramo najprej odpraviti omejitev, da se pri preiskovanju upoštevajo samo popolne hipoteze. Sistem HYPER/CA preišče celoten prostor hipotez in vrne vse hipoteze, ki minimizirajo vrednost njegove cenilne funkcije. Za razliko od sistema HYPER se ne ustavi, tudi če najde popolno in konsistentno hipotezo.

Jedro naše nadgradnje je nova cenilna funkcija (enačba 6.2).

$$cost(H) = \#misclassified(H) + cc \cdot search\_size(H) \tag{6.2}$$

Oblika te cenilne funkcije ni novost, saj ustreza klasičnemu nastavku, ki optimizira točnost hipoteze (prvi člen) in njeno velikost oz. kompleksnost (drugi člen). Temu nastavku navsezadnje ustreza tudi cenilna funkcija sistema HYPER. Točnost hipoteze naša cenilna funkcija oceni s členom $\#misclassified(H)$, ki označuje število vseh (tako pozitivnih kot negativnih) učnih primerov, ki jih H narobe klasificira. Velikost hipoteze drugi sistemi ILP tipično ocenjujejo kot neko uteženo število posameznih sestavnih delov hipoteze (literalov, spremenljivk in izrazov). Namesto velikosti hipoteze naša cenilna funkcija v členu $search\_size(H)$ oceni velikost prostora, ki jo je sistem HYPER/CA moral pregledati, da je odkril hipotezo H. Velikost tega prostora je enaka številu vseh hipotez, ki so po velikosti manjše ali enake H, in jih je moral sistem HYPER/CA pri preiskovanju obravnavati pred H (privzamemo, da je pred H pregledal vse enako velike hipoteze). Člen $search\_size(H)$ utežimo s parametrom cc, *koeficientom kompleksnosti*.

Za razliko od velikosti hipoteze, $search\_size(H)$ z izostritvami hipoteze ne narašča linearno, temveč eksponentno. Naj bosta H_1 in H_2 dve hipotezi, naj bo H_1 manjša kot H_2 in naj velja $cost(H_1) = cost(H_2)$. Če obema hipotezama dodamo enak literal in dobimo H_1' in H_2', se bo velikost obeh povečala za enako količino, vendar bo $search\_size(H_1') < search\_size(H_2')$. Za to, da bi veljalo $cost(H_1') = cost(H_2')$, bi se morala točnost H_2' v primerjavi z H_2 povečati bistveno bolj, kot točnost H_1' v primerjavi z H_1. S tem dosežemo, da je HYPER/CA z naraščanjem velikosti hipoteze vedno manj pripravljen preiskovati naprej oz. mora zato doseči večje izboljšave točnosti. Ni nam znan noben drug sistem ILP, ki bi v cenovni funkciji upošteval velikost preiskanega prostora.

Parameter cc uravnava, koliko smo pripravljeni preiskati za neko izboljšavo

točnosti H merjene na učnih podatkih. Manjši kot je *cc*, večji prostor smo pripravljeni preiskati. Vrednost *cc* lahko določi uporabnik ali pa se določi preko postopka podobnega iterativnemu poglabljanju. Uporabnik (ali robot) ve koliko časa ima oziroma je pripravljen nameniti učenju neke hipoteze H. Učenje s sistemom HYPER/CA najprej požene z neko (visoko) vrednostjo *cc*, ki naj bi se pričakovano končala hitro. Če se učenje konča in dosežena točnost ni zadovoljiva, potem učenje ponovi z nižjo vrednostjo *cc*. Če učenje pri neki vrednosti traja predolgo, ga lahko prekine in vzame trenutno najboljšo odkrito hipotezo.

Kot je razvidno iz formule 6.2, sistem HYPER/CA ne potrebuje pričakovane (oz. dovoljene) stopnje šuma, N_{ex}, ki je tipično vhodni parameter sistemov ILP. Parameter *cc* ni ekvivalenten N_{ex}. Če iščemo hipotezo, ki ustreza dani vrednosti N_{ex}, je preiskovanje grafa izostritev, dokler ne najde hipoteze, ki doseže stopnjo šuma manjšo ali enako N_{ex}, brez dodatnih omejitev pripravljeno povečevati velikost hipoteze, dokler se njena točnost še izboljšuje, ne glede na velikost teh izboljšav. Ker parameter *cc* utežuje člen $search\_size(H)$ to z našo cenilno funkcijo ni mogoče, saj $search\_size(H)$ zaradi eksponentne rasti prej ali slej preseže vrednost, ki jo z izboljšavo točnosti sploh lahko dosežemo. Kot opišemo v nadaljevanju, lahko to lastnost izkoristimo za rezanje prostora preiskovanja.

Velikost preiskanega prostora ocenimo iz velikosti H. K velikosti prostora prispevajo literali, ki jih dodajamo, izrazi, v katere izostrimo spremenljivke, in prilagajanja spremenljivk, ki jih opravimo. Pri hipotezah z več stavki se prispevki posameznih stavkov množijo, tako da oceno celotnega prostora dobimo po enačbi 6.3.

$$search\_size(H) = \prod_{clause\ c \in H} lit\_space(c) \cdot term\_space(c) \cdot var\_unif(c) \qquad (6.3)$$

Prispevke literalov, izrazov in prilagajanj spremenljivk znotraj posameznega stavka c izračunamo na podlagi njihovega števila v stavku in možnosti za njihove vrednosti iz predznanja. Celoten prostor preiskanih literalov je tako seštevek števil vseh možnih zaporedij literalov iz predznanja dolžin od 1 do števila vseh literalov v c. Izračun je v enačbi 6.4, kjer je $\#lits(c)$ število literalov v c in $\#backliterals$ število literalov v predznanju, izmed katerih se izbira literale, ki se jih dodaja v c.

$$lit\_space(c) = \sum_{i=1}^{\#lits(c)} \#backliterals^i \tag{6.4}$$

Velikost prostora preiskanih izrazov ocenimo na podoben način kot pri literalih, kot je razvidno iz enačbe 6.5, kjer je $\#terms(c)$ število izrazov v c in $\#term\_clauses$ število možnih izostritev spremenljivk v izraze. Tu gre za oceno zgornje meje, saj pri izostritvi v izraze upoštevamo tip spremenljivke in ne pridejo vedno vse možnosti v poštev. Število izrazov v c določimo tako, da preštejemo vse izraze (atome, števila, vgnezdene strukture) v c, ki so bili izostreni z uporabo pravil za izostritev spremenljivk v izraze. Za našo razpravo lahko privzamemo, da obstaja za izostritev posameznega stavka samo poti enake dolžine (take, ki se razlikujejo kvečjemu za vrstni red nekaterih izostritev), ker se pri nastavljanju učnih problemov ILP redundanci ponavadi izogibamo. Če bi obstajale poti za izostritev različne dolžine, potem bi vzeli najdaljšo od njih, ker nas zanima ocena zgornje meje velikosti preiskanega prostora.

$$term\_space(c) = \sum_{i=1}^{\#terms(c)} \#term\_clauses^i \tag{6.5}$$

Oceno prostora prilagajanj spremenljivk izračunamo za vsako množico spremenljivk istega tipa posebej, nato pa jih zmnožimo, da dobimo oceno za celoten stavek. Naj bodo $t_1, ..., t_k$ vsi tipi spremenljivk, ki se pojavijo v c. Označimo z v_j vse spremenljivke v c, ki so tipa t_j. Za spremenljivke istega tipa velja, da smo za vsako prilagajanje izbrali dve od takrat še neprilagojenih spremenljivk, tako da za oceno prostora zmnožimo vrednost $\binom{i}{2}$ za vse vrednosti i od števila neprilagojenih spremenljivk tega tipa v c, $\#ununif(v_j) + 1$, do števila vseh spremenljivk tega tipa v c, $\#occur(v_j)$. Če je $\#occur(v_j) = \#ununif(v_j)$ je vrednost člena za tip t_j enaka 1. Končna ocena prostora prilagajanj spremenljivk za cel stavek c, $var\_unifs(c)$, se tako izračuna po enačbah 6.6 in 6.7.

$$var\_unifs(c) = \prod_{j=1}^{k} var\_unifs(c, t_j) \tag{6.6}$$

$$var\_unifs(c, t_j) = \begin{cases} \prod_{i=\#ununif(c,v_j)+1}^{\#occur(c,v_j)} \binom{i}{2}; \text{ če } \#occur(c, v_j) = \#ununif(c, v_j) \\ 1; \text{ sicer} \end{cases} \quad (6.7)$$

Kot je razvidno iz enačb 6.2–6.7, velikost prostora narašča (super)eksponentno hitro, kar pomeni, da se pri manjših hipotezah neko povečanje, npr. dodajanje literala, ceni manj kot pri večjih hipotezah in bo morala biti izboljšava točnosti temu ustrezno večja, da bo hipoteza dobro ocenjena. To poskrbi, da je hevristika z večanjem hipoteze vedno manj pripravljena preiskovati naprej.

Za primer izračunajmo vrednost $search\_size(H)$ na konkretnem primeru. Vzemimo primer iz domene `member` (stran 82), ki ima v definiciji problema naslednja pravila za izostritve:

```
backliteral( member(X,L), [L:list], [X:item]).

term( list, [], []).
term( list, [X|L], [ X:item, L:list]).
```

Izračunajmo vrednost $search\_size(H)$ za hipotezo H:

```
member(A, [B|X]):-
  member(C, [A,D|X]).

member(A, X):-
  member(A, [B|X]),
  member(B, []).
```

Imenujmo z c_1 prvi stavek H in drugega z c_2. Iz pravil za izostritve določimo naslednje vrednosti, ki so skupne c_1 in c_2, število možnih izostritev literalov: $\#backlits = 1$, število možnih izostritev v izraze: $\#term\_clauses = 2$ in možni tipi spremenljivk: `item`, `list`.

Stavek c_1 vsebuje dva literala, enega v glavi in enega v telesu, torej je $\#lits(c_1) = 2$. Vrednost $lit\_space(c_1)$ je tako:

$$lit\_space(c_1) = \sum_{i=1}^{2} 1^i = 2$$

Pri razvoju c_1 smo trikrat uporabili pravilo za izostritev v izraz, enkrat v glavi, da smo izostrili seznam `[B|X]`, in dvakrat v telesu, da smo izostrili seznam `[A,D|X]`, torej je $\#terms(c_1) = 3$. Vrednost $term\_space(c_1)$ je tako:

$$term\_space(c_1) = \sum_{i=1}^{3} 2^i = 14$$

V stavku c_1 je pet spremenljivk tipa `item` (označimo jih z v_{item}), torej je vrednost $\#occur(c_1, v_{item}) = 5$, a se spremenljivka `A` pojavi dvakrat, torej nastopajo v c_1 le štiri različne spremenljivke in je $\#ununif(c_1, v_{item}) = 4$. Od spremenljivk tipa `list` (označimo jih z v_{list}) se pojavi le dvakrat spremenljivka `X`, torej je $\#occur(c_1, v_{list}) = 2$ in $\#ununif(c_1, v_{list}) = 1$. Vrednost $var\_unifs(c_1)$ je tako:

$$var\_unifs(c_1) = \prod_{i=5}^{5} \binom{i}{2} \cdot \prod_{i=2}^{2} \binom{i}{2} = 10$$

Za c_2 na enak način, kot pri c_1, določimo naslednje vrednosti: $\#lits(c_2) = 3$, $\#terms(c_2) = 2$, $\#occur(c_2, v_{item}) = 4$, $\#ununif(c_2, v_{item}) = 2$, $\#occur(c_2, v_{list}) = 2$, $\#ununif(c_2, v_{list}) = 1$ in izračunamo:

$$lit\_space(c_2) = \sum_{i=1}^{3} 1^i = 3$$

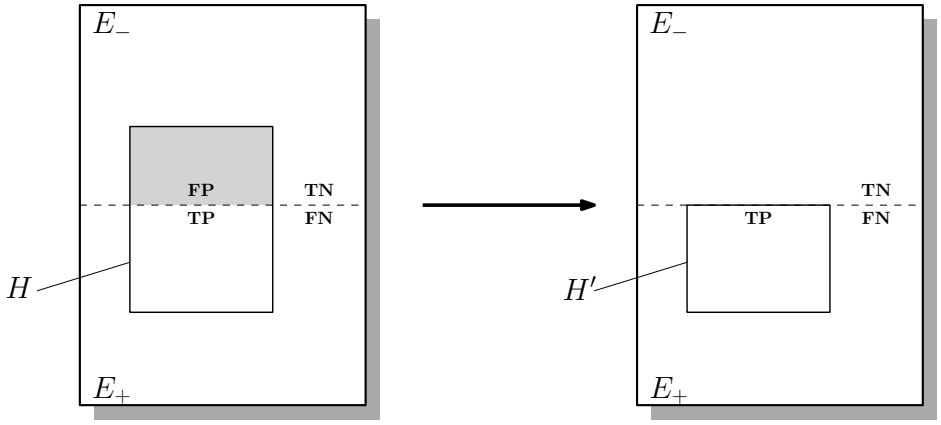
$$term\_space(c_2) = \sum_{i=1}^{2} 2^i = 6$$

$$var\_unifs(c_2) = \prod_{i=3}^{4} \binom{i}{2} \cdot \prod_{i=2}^{2} \binom{i}{2} = 18$$

Vrednost $search\_size(H)$ je torej:

$$search\_size(H) = 2 \cdot 14 \cdot 10 \ \cdot \ 3 \cdot 6 \cdot 18 = 90720$$

Ker je vsaka izostritev specializacija hipoteze in pokrije manj primerov, lahko to izkoristimo za rezanje neperspektivnih hipotez. Vemo, da pozitivnih primerov, ki jih hipoteza H ne pokrije, ne bo pokrila tudi po izostritvi. Edini način, da se izboljša točnost H, je, da po izostritvi ne pokriva več negativnih primerov, ki jih je prej pokrila. Najboljši rezultat, tj. hipoteza z najmanjšo ceno, ki ga lahko



Slika 6.1: Idealen primer, ko izostritev H v H' (en korak preiskovanja) iz pokritih primerov odstrani vse negativne in H' ne klasificira nobenega primera napačno pozitivno (false positive; na sliki **FP**).

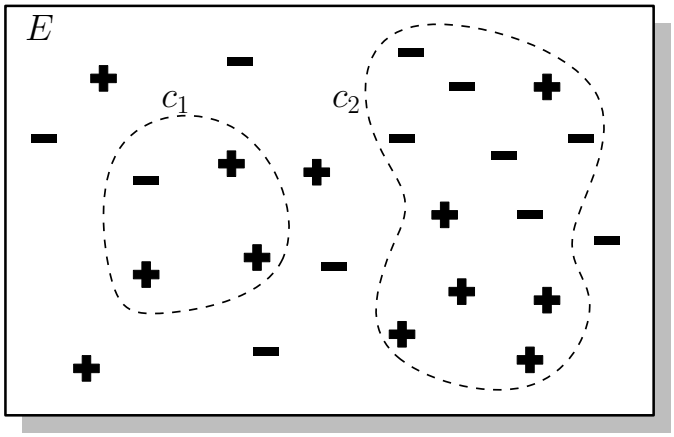
dosežemo iz H, je, če naslednja izostritev H' pokrije še vse tiste pozitivne primere, ki jih je pokrila H, in nobenega negativnega (slika 6.1). Torej da H' nima več napačno pozitivnih (angl. false positive) primerov.

Tekom preiskovanja za vsako generirano hipotezo H izračunamo najboljšo možno ceno hipoteze še dosegljive iz nje, $opt\_cost(H)$. Če je ta slabša (večja) od najboljše do sedaj najdene hipoteze (oz. hipotez), potem H odrežemo, tj. jo zavržemo in je ne razvijamo naprej. Ker se izračun $opt\_cost(H)$ od izračuna $cost(H)$ razlikuje samo v vrednosti $misclassified(H)$, je rezanje računsko nezahtevno. Ker prej ali slej pridemo do točke, ko ni več mogoče toliko izboljšati točnosti, da bi upravičilo dodatno povečevanje hipoteze, saj je število narobe klasificiranih primerov navzdol omejeno z 0, se preiskovanje zaradi rezanja vedno samo ustavi. Zaradi tega ne potrebujemo nujno dodatnega ustavitvenega pogoja in lahko zahtevnost preiskovanja uravnavamo samo s parametrom cc iz enačbe 6.2. Seveda nas to ne omejuje, da vseeno ne dodamo drugih omejitev ali ustavitvenih pogojev.

Da smo še dodatno odpravili nepotrebno preiskovanje, smo implementirali preverjanje ekvivalence hipotez, eno izmed tehničnih izboljšav iz sistema HYPER/X.

Zadnja implementirana sprememba se tiče le učenja hipotez z več stavki in smo jo povzeli iz sistema HYPER/N. Sistem HYPER za izostritev hipotez z več stavki izbere prvi stavek, ki sam, brez preostanka hipoteze, pokrije vsaj en negativen primer. V primeru, da iščemo popolno hipotezo v podatkih brez šuma, je to

smiselno, ker bo ta stavek prej ali slej potrebno izostriti. Problem je, da je v šumni domeni možno, da tega stavka nikoli ne bomo izostrili tako, da ne bi pokril vsaj enega negativnega primera. Vzemimo za primer situacijo na sliki 6.2. Recimo, da se c_1 ne da tako izostriti, da ne bi pokrival negativnega primera, ki ga trenutno pokriva. V takih primerih sistem HYPER za izostritev zmeraj izbere prvi stavek, ki pokriva kak negativen primer, torej bo tu izbral c_1 in se zataknil, medtem ko stavek c_2 očitno deluje bolj obetaven za izostritve, saj je od pokritih primerov večji delež negativnih.



Slika 6.2: Hipoteza z dvema stavkoma, c_1 in c_2, ki neodvisno pokrivata pozitivne (+) in negativne (−) primere. Sistem HYPER tu za izostritev izbere prvi stavek, c_1. Če se c_1 ne da tako izostriti, da ne bi pokril edinega negativnega primera, ki ga pokriva, se preiskovanje tu zatakne.

Da preprečimo, da se preiskovanje v takih primerih ne zatakne, moramo spremeniti vrstni red izbiranja stavkov. Sistem HYPER/CA tako kot sistem HYPER/N izostri vse stavke, ki imajo najvišji *clause_cost*, ki se izračuna po enačbi 6.8, kjer je p število pozitivnih primerov in n število negativnih primerov, ki jih pokriva c.

$$clause\_cost(c) = \frac{n}{p+n};\ e\ p+n>0,\ sicer\ clause\_cost(c)=0 \qquad (6.8)$$

Z naštetimi izboljšavami sistem HYPER/CA obdrži prednosti sistema HYPER (ne prekrivnost, dobro učenje rekurzivnih hipotez itd.), hkrati pa je sposoben delovati na šumnih podatkih. Z uporabo svoje cenilne funkcije iz enačbe 6.2 poišče

hipoteze, ki so kar najbolj točne (imajo najvišjo klasifikacijsko točnost), a še niso preveč obsežne.

Na koncu za boljšo predstavo orišemo grob pregled algoritma HYPER/CA:

HYPER/CA

Vhod:
- neprazni množici pozitivnih in negativnih primerov
- definicija učnega problema: množica začetnih hipotez, pravila za izostritve , predznanje, parametri učenja

Izhod:
- hipoteza iz prostora preiskovanja z najnižjo vrednostjo *cost*; prostor preiskovanja je omejen glede na parameter *cc*

Algoritem:

1. Inicializacija seznama generiranih hipotez *Hyps*, urejenega po naraščajoči vrednosti *cost*. Na začetku so notri samo začetne hipoteze. Inicializira se tudi spremenljivka *BestHyp*, ki hrani do sedaj najboljšo najdeno hipotezo.
2. Če $cost(Hyps[0]) < cost(BestHyp)$ zamenjaj vrednost *BestHyp* z $Hyp[0]$ in z rezanjem iz $Hyps[1:]$ odstrani vse hipoteze, ki so zaradi nove vrednosti $BestHyps$ postale neperspektivne.
3. Odstrani $Hyps[0]$ iz *Hyps* in generiraj *NewHyps*, seznam vseh izostritev $Hyps[0]$.
4. Odstrani iz *NewHyps* vse hipoteze, ki so že bile razvite.
5. Z rezanjem odstrani iz *NewHyps* vse neperspektivne hipoteze.
6. Uredi seznam *NewHyps* po naraščajoči vrednosti *cost* in ga zlij skupaj s starim *Hyps* v nov seznam *Hyps*.
7. Če je seznam *Hyps* prazen, končaj in vrni *BestHyp*, sicer ponovi postopek od 2. koraka.

6.2 Poskusi in primerjava z najnovejšimi sistemi ILP

Uspešnost sistema HYPER/CA smo testirali na več tipičnih učnih nalogah ILP in rezultate primerjali z rezultati dveh najnovejših sistemov ILP, sistemov Progol in Aleph. V tem razdelku najprej opišemo tehnične lastnosti sistemov Progol in Aleph v podrazdelku 6.2.1, nato pa v podrazdelku 6.2.2 predstavimo rezultate poskusov in jih komentiramo.

6.2.1 Sistema Progol in Aleph

Progol [74] je v zgodnjih devetdesetih letih razvil profesor Stephen Muggleton, eden začetnikov področja ILP, ki je v svojem članku iz leta 1991 [73] prvi uporabil izraz induktivno logično programiranje. Progol je danes eden najbolj razširjenih sistemov ILP in je bil uporabljen v vrsti različnih področij, med drugim biokemiji [104, 103], farmakologiji [33] in procesiranju naravnega jezika [21]. Verzija Progol4.4, ki smo jo uporabili v naših poskusih, je prosto dostopna na spletu[1].

Aleph [101] je sprva nastal kot implementacija istih idej, na katerih je temeljil razvoj Progola. Razvila sta ga Ashwin Srinivasan in Rui Camacho kot prototip za poskuse z različnimi pristopi ILP. Sčasoma je prerasel začetne okvirje in je danes[2] zelo prilagodljiv sistem, ki omogoča uporabo vrste različnih metod in tehnik ILP. Aplikacije Aleph so na podobnih področjih kot Progol [15, 100].

Zaradi velike podobnosti principov delovanja obeh sistemov zadošča, če predstavimo teoretično ozadje in osnovno strukturo Progola, in na koncu opozorimo kje se Aleph razlikuje od njega.

Glavna ideja, na kateri bazira Progol, je tako imenovana inverzna implikacija (angl. *inverse entailment*). Kot smo že omenili, je osnovni problem ILP

$$B \wedge H \models E,$$

kjer je B podano predznanje, H iskana hipoteza in so E primeri iz katerih se učimo. Če privzamemo, da sta E in H posamezna Hornova stavka, lahko izraz

[1] *http://www.doc.ic.ac.uk/∼shm/progol.html*

[2] *http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.pl*

preuredimo v

$$B \wedge \overline{E} \models \overline{H}.$$

Naj bo $\overline{\bot}$ (potencialno neskončna) konjunkcija popolnoma opredeljenih literalov (angl. ground atoms), ki drži v vseh modelih $B \wedge \overline{E}$. Ker mora tudi $\overline{H}$ veljati v vseh modelih $B \wedge \overline{E}$, mora vsebovati podmnožico literalov iz $\overline{\bot}$. Torej velja:

$$B \wedge \overline{E} \models \overline{\bot} \models \overline{H}$$

in posledično po negaciji:

$$H \models \bot.$$

Slednjo relacijo Progol izkoristi za omejitev prostora preiskovanja pri iskanju H.

Progolov učni algoritem lahko zapišemo v štirih korakih:

1. Izbere se pozitiven primer $e_i \in E$, ki ga bo novo pravilo pokrilo.
2. Glede na podano specifikacijo jezika hipotez se zgradi najbolj specifični stavek $\bot_i$, za katerega velja $B \wedge \bot_i \models e_i$. $\bot_i$ vsebuje predstavitev vsega znanja o primeru e_i, ki ga lahko zapišemo znotraj omejitev jezika hipotez in parametrov učenja.
3. Poišče se stavek C_i, za katerega velja $\square \preceq C_i \preceq \bot_i$, kjer je $\preceq$ relacija θ-subsumpcije in $\square$ stavek s praznim telesom. Prolog C_i poišče s preiskovanjem podmreže stavkov urejene po θ-subsumpciji. Sam algoritem preiskovanja je podoben A^*, kjer kriterijska funkcija optimizira Occamovo kompresijo (tj. število bitov potrebnih, da zakodiramo posamezen stavek – formalna definicija v [74]). Prostor preiskovanja je omejen od zgoraj z $\square$ ter od spodaj z $\bot_i$.
4. C_i se doda hipotezi H in iz E se odstrani vse primere, ki jih C_i pokrije. Če so v E še pozitivni primeri, se postopek ponovi od koraka 1, sicer se konča in vrne H.

Aleph deluje po podobnem postopku kot Progol. Glavna algoritmična razlika je v koraku 3, kjer Aleph izvede preiskovanje najboljši-najprej s svojo kriterijsko funkcijo, ki optimizira vrednost $P - N$, kjer je P število pozitivnih in N število negativnih primerov, ki jih pokrije stavek C_i. Poleg tega se razlikujeta še v vrsti tehničnih podrobnosti.

6.2.2 Poskusi in rezultati

Uspešnost sistemov HYPER/CA, Progol in Aleph smo testirali na več tipičnih učnih nalogah ILP. Definicije učnih problemov za vsak uporabljen sistem ILP posebej so podane v dodatku B. Rezultate vseh treh sistemov na danih problemih bi se dalo izboljšati, ali z izboljšanim predznanjem, ali z iskanjem boljše nastavitve parametrov, vendar ni bil naš namen preverjati najboljše možne uspešnosti teh sistemov. Želimo preveriti, kako dobro deluje sistem HYPER/CA v primerjavi s sistemoma Progol in Aleph v okviru nekih (smiselnih) primerljivih nastavitev.

Učni podatki so izbrani naključno iz nekega omejenega obsega. Vsak učni nabor je vseboval vsaj en pozitiven in vsaj en negativen primer. V testih uporabimo štiri različne velikosti učnih množic: 10, 50, 100, 200. V tabelah rezultatov je velikost učne množice navedena v stolpcu z imenom 'N'. Za vsako velikost podatkov smo teste izvedli pri treh stopnjah šuma v podatkih: 0; 0,2 in 0,5. V tabelah je stopnja šuma navedena v stolpcu z imenom 'data noise'. Šum smo dodali tako, da smo najprej generirali brezšumen nabor podatkov, nato pa naključno izbrali delež primerov enak stopnji šuma in jim naključno določili, ali gre za pozitiven ali negativen primer. Pri tem smo vzdrževali pogoj, da mora učni nabor vsebovati vsaj en pozitiven in vsaj en negativen primer. Ker so problemi ILP zelo občutljivi na izbor učnih podatkov smo v nekaterih domenah morali biti previdni pri naključni generaciji primerov, da smo dobili uporabne rezultate (tj. da se je katerikoli od sistemov sploh česa naučil). Morebitne posebnosti pri generiranju učnega nabora izpostavimo pri vsaki domeni posebej. Kot testno množico smo, kjer je bilo mogoče, uporabili celotno omejeno domeno, iz katere smo izbirali učne primere. Kjer je bila celotna domena prevelika, smo uporabili ustrezno vzorčenje. Izbor testnega nabora opišemo pri vsaki domeni posebej. Kjer navajamo število primerov v testni množici, je potrebno upoštevati, da kot testno množico pogosto uporabimo celotno domeno primerov, vendar z odstranjenimi učnimi primeri. Število dejansko uporabljenih primerov v testnih množicah je tako na koncu ponavadi malo manjše (za maksimalno 200 primerov).

Edini parameter, ki ga med testi spreminjamo pri sistemu HYPER/CA, je koeficient kompleksnosti (cc iz enačbe 6.2) in sicer testiramo vrednosti: 1; $0,1$; $0,01$; $0,001$; $0,0001$. V tabelah z rezultati sistema HYPER/CA je vrednost koeficienta kompleksnosti navedena v stolpcu z imenom 'cc'. Pri Progolu in Alephu spremi-

njamo parameter noise, ki pomeni pričakovano stopnjo šuma oz. natančneje, koliko procentov primerov, ki jih pokrije posamezni stavek, je lahko negativnih. Pri obeh testiramo štiri vrednosti parametra noise: 0, 20, 50, 100. Vrednost parametra noise je v tabelah rezultatov Progol in Aleph navedena v stolpcu z imenom 'noise'.

Pri vsakem zagonu merimo več značilk, ki jih tu navajamo po imenih stolpcev, v katerih so navedeni v tabelah:

- **runtime** – čas izvajanja v sekundah
- **CA** – klasifikacijska točnost kot delež pravilno klasificiranih primerov
- **TP, FP, FN in TN** – vrednosti matrike napak (angl. confusion matrix; TP – true positive, FP – false positive, FN – false negative, TN – true negative)
- #**literals** – število literalov v končni hipotezi
- #**vars** – število spremenljivk v končni hipotezi

Za vsako kombinacijo velikosti učnega nabora, stopnje šuma in vrednoti parametra *cc* oz. noise, smo pognali deset testov in v tabelah rezultatov navedli povprečja vseh desetih testov. Sledijo opisi vseh testnih domen ILP. Rezultati učenja v njih so podani v tabelah v dodatku C na strani 147.

`issorted` (neuravnoteženi učni nabor)

Učimo se definicije predikata `issorted(L)`, ki pove, ali je seznam `L` leksikografsko urejen. Ciljni predikat je zapisan v programu 12.

```
issorted([H]).
issorted([H1,H2|T]):-
  H1 @=< H2,
  issorted([H2|T]).
```

Program 12: Ciljna definicija predikata `issorted`. Operator `@=<` leksikografsko primerja oba argumenta.

V domeni so vsi seznami dolžine manj ali enako 5 in več ali enako 1, ki jih lahko sestavimo iz znakov `a`, `b`, `c`, `d` in `e`. Testna množica je sestavljena iz vseh seznamov v domeni in je zelo neuravnotežena. Sestoji iz 251 pozitivnih primerov (urejenih seznamov) in 3905 negativnih primerov (neurejenih seznamov). Popolnoma naključno generiranje učnih množic nebi skoraj nikoli generiralo pozitivnega primera, tako da so primeri za učne množice generirani enakomerno porazdeljeno po dolžinah. Isto velja za vse generirane sezname v vseh domenah. Pri majhnih seznamih je možnost, da se naključno generira urejen seznam večja, a še vedno so učni nabori precej neuravnoteženi (povprečno je pozitivnih primerov okoli 21%). Rezultati v tej domeni so v razdelku C.1 na strani 148.

`issorted` (uravnoteženi učni nabor)

Domena tega testa je popolnoma enaka prejšnji. Edina razlika je, da sedaj generiramo popolnoma uravnotežene učne nabore, torej take kjer je pol primerov pozitivnih in pol negativnih. Testna množica ostaja enaka. To domeno smo dodali zato, da preverimo, če na uspešnost učenja bolj pozitivno vpliva več informacije o pozitivnih primerih, kot škodi drugačna porazdelitev primerov med učno in testno množico. Rezultati v tej domeni so v razdelku C.2 na strani 152

`member`

V tem testu je ciljni predikat `member(E, L)` (program 13), ki preverja, ali je `E` element seznama `L`.

```
member(H, [H|T]).
member(H, [H1|T]):-
  member(H, T).
```

Program 13: Ciljna definicija predikata `member`

V domeni so vsi primeri, ki jih lahko generiramo z uporabo znakov `a`, `b`, `c`, `d` in `e`, kjer je seznam `L` lahko dolg 5 ali manj znakov. Testna množica sestoji iz vseh možnih primerov v domeni, kar pomeni 12705 pozitivnih in 6825 negativnih

primerov. Za učne nabore smo uporabili naključno generiranje, kjer so primeri generirani enakomerno po dolžini seznama `L`, kar da podobno uravnoteženo razmerje pozitivnih in negativnih primerov kot v testni množici. Rezultati v tej domeni so v razdelku C.3 na strani 156.

`oddeven`

V tem testu se učimo definicije predikata `oddeven(OE, L)`, ki v argumentu `OE` vrne `odd`, če je seznam `L` lihe dolžine, in `even`, če je seznam `L` sode dolžine. Definicija predikata, kot jo vidimo v programu 14, je malo nenavadna. Predikat je narejen po primeru iz knjige [11], ki je napisan v dveh ločenih predikatih `odd` in `even`, ki se kličeta drug drugega. V dano obliko smo ga preoblikovali, ker nam ni uspelo izvesti hkratnega učenja dveh različnih predikatov v Progolu in Alephu.

```
oddeven(odd, [H]).
oddeven(even, []).
oddeven(OE, [H1,H2|L]):-
  oddeven(OE, L).
```

Program 14: Ciljna definicija predikata `oddeven`

V domeni so vsi primeri, ki jih lahko generiramo z uporabo znakov `a`, `b` in `c`, kjer je seznam `L` lahko dolg 8 ali manj znakov. Argument `OE` ima seveda lahko vrednost `odd` ali `even`. Testna množica sestoji iz vseh možnih primerov v domeni, ker pomeni 9841 pozitivnih in 9841 negativnih primerov. Za učne nabore smo uporabili naključno generiranje, kjer so primeri generirani enakomerno po dolžini seznama `L`, ki da v povprečju popolnoma uravnoteženo učno množico. Rezultati v tej domeni so v razdelku C.4 na strani 159.

`concat`

V zadnjem testu je ciljni predikat `concat(L1, L2, L3)`, ki konkatenira seznama `L1` in `L2`, da tvori seznam `L3`. Definicija ciljnega predikata je zapisana v programu 15.

V domeni so vsi primeri, ki jih lahko generiramo z uporabo znakov `a`, `b` in `c`, kjer sta seznama `L1` in `L2` lahko dolga 3 ali manj znakov in je seznam `L3` lahko dolg

```
concat([],L,L).
concat([H|L1],L2,[H|L3]):-
  concat(L1, L2, L3).
```

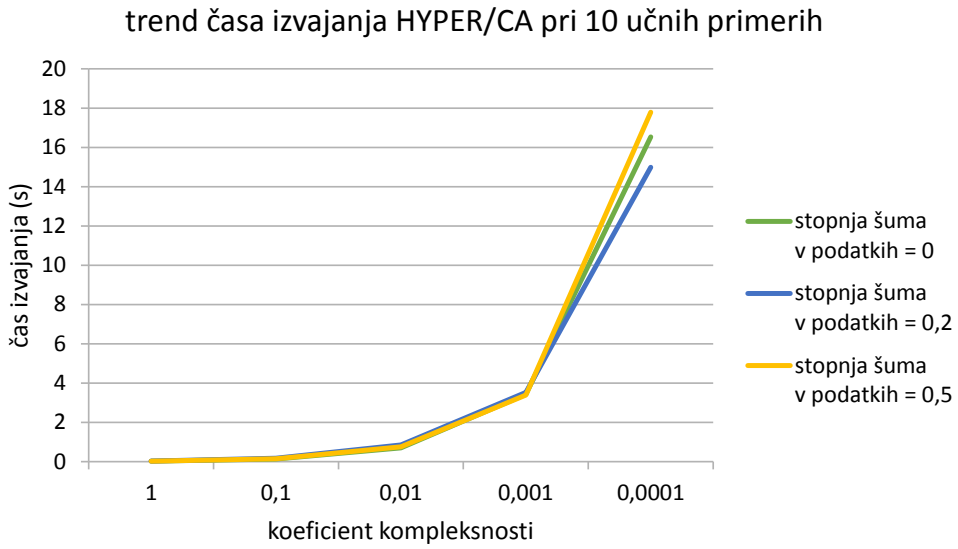
Program 15: Ciljna definicija predikata `concat`

6 ali manj znakov. Testna množica bi bila prevelika in preveč neuravnotežena, če bi uporabili celo domeno, tako da sestoji iz vseh pozitivnih primerov v domeni in dvakrat toliko negativnih primerov, torej 1600 pozitivnih primerov in 3200 negativnih primerov. Generirani učni nabori so popolnoma uravnoteženi, s tem, da so generirani negativni primeri ti. bližnji negativni primeri, ki se jih generira tako, da najprej generiramo pozitiven primer, nato pa naključno spremenimo vrednost enega od argumentov. Generiranje takih negativnih primerov je bilo potrebno, ker so bili pri popolnoma naključni generaciji vsi trije sistemi zelo neuspešni. Pri tej domeni je namreč zelo lahko dobiti uspešen model, če se upošteva le začetke seznamov. Pri popolnoma naključnih učnih naborih je ponavadi model, ki je le preveril, če sta prva elementa seznamov `L1` in `L3` enaka, imel 100% točnost. Rezultati v tej domeni so v razdelku C.5 na strani 162.

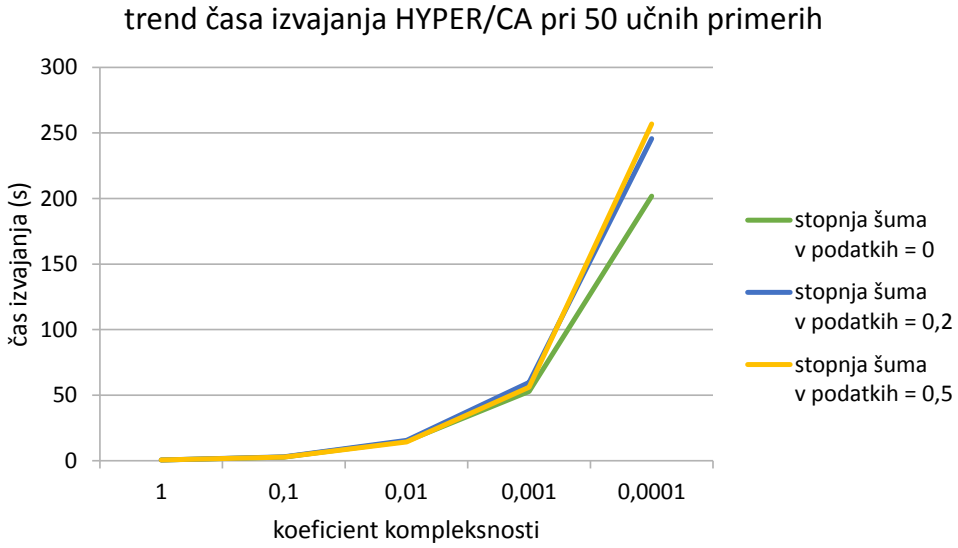
6.2.3 Razprava o rezultatih

Trendi v rezultatih sistema HYPER/CA so v skladu s pričakovanji. Grafi na slikah 6.3–6.3 jasno kažejo, da, manjši ko je parameter *cc*, dlje traja indukcija. Od tega trenda rezultati odstopajo samo v redkih primerih, kot je pri učenju predikata `issorted` pri 200 primerih z uravnoteženim učnim naborom, kjer učenje pri $cc = 0.0001$ traja manj kot pri $cc = 0.001$. Ta anomalija je posledica tega, da iskanje pri manjšem *cc* na neki točki odkrije dobro (v tem primeru popolno in konsistentno) hipotezo in potem rezanje poskrbi, da se iskanje prej ustavi. Uravnoteževanje učnega nabora pri učenju `issorted` sicer večinoma poslabša točnost končne hipoteze (porazdelitev primerov v učni množici ne ustreza porazdelitvi v testni množici), vendar sistem HYPER/CA pri dovolj majhni vrednosti *cc* in dovolj veliko podatki še vedno odkrije pravilno hipotezo (tj. tako s klasifikacijsko točnostjo enako 1).

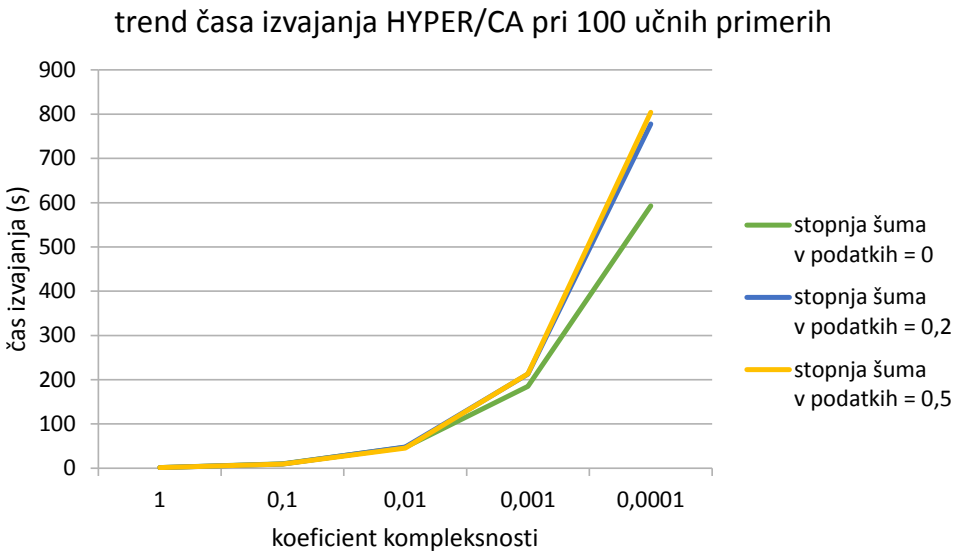
Sistema Aleph in Progol se obnašata podobno, kot smo tudi pričakovali, glede



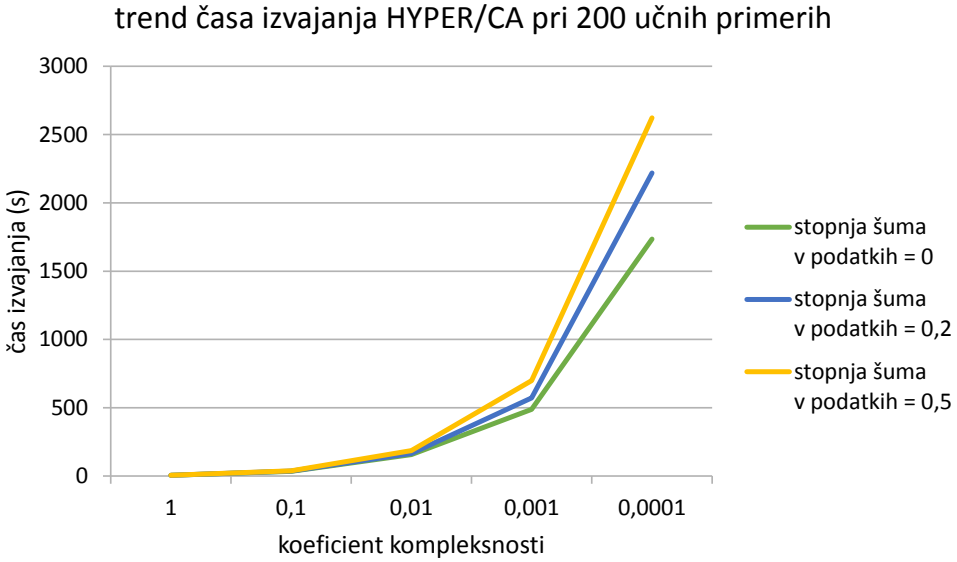
Slika 6.3: Za vsako stopnjo šuma posebej po vseh poskusih povprečen čas izvajanja dosežen s sistemom HYPER/CA pri različnih vrednostih koeficienta kompleksnosti. Velikost učne množice je 10 primerov.



Slika 6.4: Za vsako stopnjo šuma posebej po vseh poskusih povprečen čas izvajanja dosežen s sistemom HYPER/CA pri različnih vrednostih koeficienta kompleksnosti. Velikost učne množice je 50 primerov.



Slika 6.5: Za vsako stopnjo šuma posebej po vseh poskusih povprečen čas izvajanja dosežen s sistemom HYPER/CA pri različnih vrednostih koeficienta kompleksnosti. Velikost učne množice je 100 primerov.



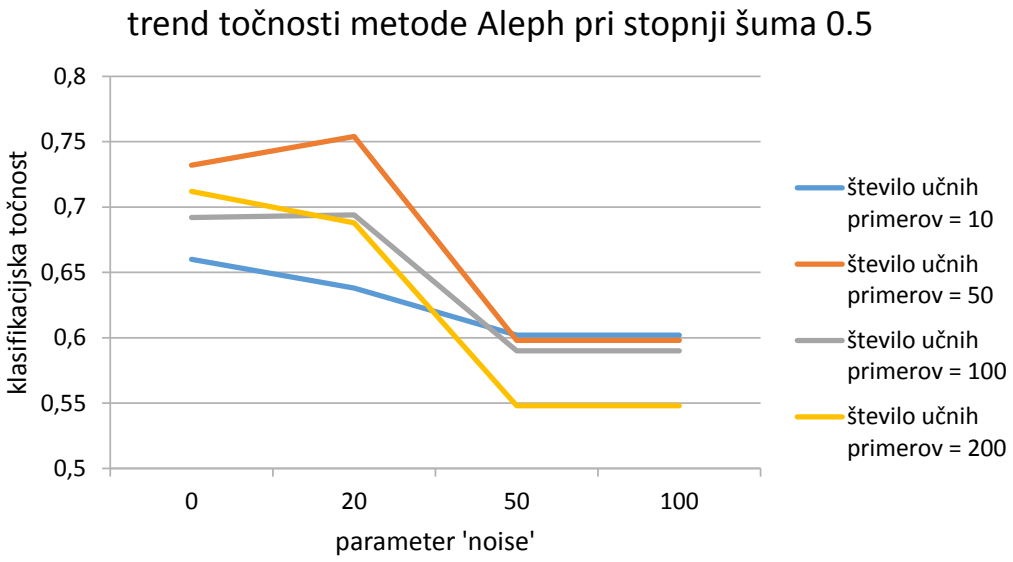
Slika 6.6: Za vsako stopnjo šuma posebej po vseh poskusih povprečen čas izvajanja dosežen s sistemom HYPER/CA pri različnih vrednostih koeficienta kompleksnosti. Velikost učne množice je 200 primerov.

na podobnost algoritmov. Pri obeh je klasifikacijska točnost na skoraj vseh naborih podatkov najvišja pri nastavitvi parametra 'noise' (pričakovane stopnje šuma v podatkih) na vrednosti 0 ali 20, vendar je treba biti pri tem previden. Oba sta namreč pri teh nastavitvah nagnjena k pretiranemu prilagajanju podatkom (overfitting). V tabelah to prepoznamo po daljšem času učenja, nizkih vrednostih TP in povprečnemu številu spremenljivk blizu ali celo pod 1. Slednje pomeni, da so bile ustvarjene hipoteze, ki vsebujejo le dejstva, torej, da se je sistem ILP popolnoma prilagodil podatkom in si zgolj zapomnil pozitivne primere. V domenah, kjer je testna množica uravnotežena, to pomeni tudi nizko točnost, tako da je na to potrebno biti pozoren predvsem v obeh domenah *issorted*. Tudi Aleph in Progol se uspešnost zmanjša, če uravnotežimo učno množico pri učenju `issorted`.

Presenetljivo je, da je uspešnost Progola in Alepha ponavadi največja pri nastavitvi parametra noise na vrednosti 0 ali 20 tudi pri množicah s stopnjo šuma 0.5, kot je razvidno iz grafov na slikah 6.7 in 6.8. Pri teh vrednostih bi pričakovali, da bi bila zaradi pretiranega prilagajanja uspešnost nižja, kot če recimo parameter noise nastavimo na (po pričakovanjih ustreznejšo) vrednost 50. Po pregledu zgrajenih hipotez smo ugotovili, da pri vrednostih parametra noise 50 ali 100 Progol in Aleph ponavadi zgradita preveč splošno hipotezo. To je razumljivo, če upoštevamo, da vrednost prametra noise 50 pomeni, da je lahko polovica primerov, ki jih pokrije posamezen stavek hipoteze, napačno klasificiranih. Pri vrednostih 50 in 100 posledično pri gradnji hipoteze nad točnostjo hipoteze prevlada njena kompleksnost. Pri vrednostih 0 in 20 pa pogosto dele prostora pozitivnih primerov pokrijeta z (delno) pravilnimi splošnejšimi stavki, primere, ki ostanejo, pa si preprosto zapomnita (tj. jih eksplicitno dodata v hipotezo). Izkaže se, da so take, vsaj delno pravilne hipoteze, na testnih podatkih pogosto točnejše, kot preveč splošne hipoteze, ki jih gradita pri nastavitvi noise na vrednosti 50 ali 100.

Programi 16, 17 in 18 predstavijo narave napak, ki jih pogosto delajo posamezni testirani sistemi ILP. Hipoteze, ki jih gradi sistem HYPER/CA, so pri previsokih nastavitvah parametra *cc* pogosto preveč enostavne, kot je primer v programu 16, in posledično pogosto tudi presplošne. Sistema Progol in Aleph se obnašata precej podobno in v programih 17 in 18 vidimo nagnjenost k pretiranem prilagajanju, ki smo ga opisali že v prejšnjem odstavku.

Za boljšo primerjavo med tremi pristopi, smo v tabelah C.16–C.20 (razdelek C.6 na strani 165) za vsako domeno in za vse tri metode eno ob drugi zbrali za

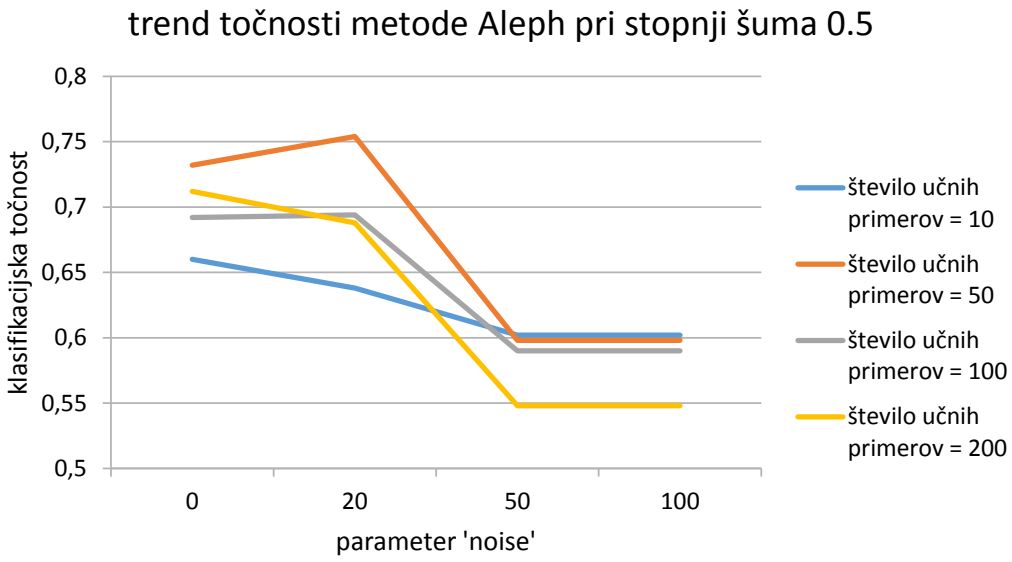


Slika 6.7: Po vseh poskusih s stopnjo šuma v učni množici 0.5 povprečena klasifikacijska točnost dosežena s sistemom Aleph za vse velikosti učne množice.

```
member(A, [B,A|C]).
member(A, [A|B]).
```

Program 16: Primer definicije predikata `member`, ki jo HYPER/CA zgradi pri 100 učnih primerih s stopnjo šuma 0.5 in parametrom $cc = 0.01$.

vsako velikost nabora podatkov najvišjo doseženo povprečno točnost, povprečen čas učenja ter povprečno število literalov in spremenljivk. Tam, kjer je imela posamezna metoda za isti učni nabor pri več nastavitvah parametrov isto uspešnost, smo vzeli tisto, kjer je bil čas učenja krajši. Tako so v teh tabelah uspešnosti, ki jih lahko dosežemo, če preizkusimo vse nabore parametrov iz naših poskusov in obdržimo najboljši model. Glede na to, da je, kot smo že omenili, pri nekaterih domenah testna množica precej neuravnotežena in sta Progol in Aleph nagnjena k pretiranem prilagajanju, bi klasifikacijska točnost lahko bila za primerjavo precej zavajajoča mera, vendar smo po pregledu ugotovili, da pride po našem izboru v primerjalne tabele zelo malo vrednosti, ki so posledice pretiranega prilagajanja. Ker te redke posamezne vrednosti ne vplivajo bistveno na splošno primerjavo



Slika 6.8: Po vseh poskusih s stopnjo šuma v učni množici 0.5 povprečena klasifikacijska točnost dosežena s sistemom Aleph za vse velikosti učne množice.

```
member(A,[B,A|C]).
member(A,[A|B]).
member(d,[b,a,d,d]).
member(d,[a,b,c,e,c]).
member(A,[B,B|C]).
member(e,[b,c,e]).
member(c,[a,b,b,c,a]).
member(a,[e,c,a,b]).
member(e,[d,a,e]).
member(a,[d,c,e,d,c]).
member(b,[d]).
member(a,[c,d,b,a]).
member(b,[c,a,b]).
member(c,[e,a]).
```

Program 17: Primer definicije predikata `member`, ki jo Aleph zgradi pri 100 učnih primerih s stopnjo šuma 0.5 in parametrom 'noise'= 20.

metod, nismo računali drugih mer.

Rezultati v vseh domenah nakazujejo, da je sistem HYPER/CA sposoben doseči primerljivo dobre točnosti kot Progol in Aleph (graf na sliki 6.9), a je od njiju bistveno počasnejši, s pogosto za več redov velikosti višjim časom učenja (graf na sliki 6.10). Najbolj je ta razlika očitna v domeni `oddeven`, kjer učenje s sistemom HYPER/CA na večjih učnih naborih traja tudi preko dveh ur, medtem ko Progol

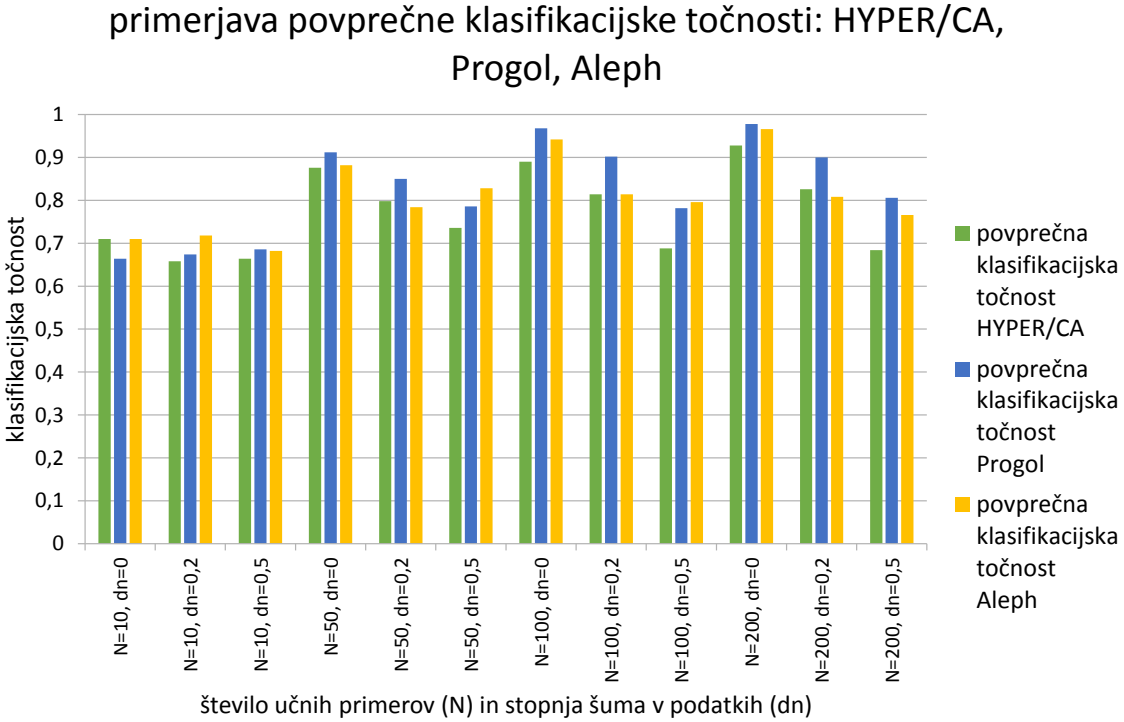
```
member(b,[b]).
member(c,[c,a]).
member(a,[d,a,e,c]).
member(d,[a,d,c]).
member(d,[b,a,d,d]).
member(a,[d,c,e,d,c]).
member(d,[a,b,c,e,c]).
member(c,[a,a,b,b,d]).
member(e,[c,e,c]).
member(e,[a,a,c]).
member(d,[b,d,b,b]).
member(c,[e,a]).
member(a,[a,c,d,c,b]).
member(d,[a,d,e,a,e]).
member(c,[e,c,c,e]).
member(a,[a]).
member(e,[d,e,c,c,c]).
member(d,[d,e,b]).
member(d,[e,d]).
member(e,[a,a,c,b]).
member(e,[d,a,e]).
member(d,[e,e]).
member(d,[d,b,c,b,c]).
member(d,[a,a,a,a,e]).
member(e,[b,c,e]).
member(b,[b,d,e]).
member(a,[e,e]).
member(c,[b,c,a,d]).
member(a,[a,c,a,a,e]).
member(a,[b,b]).
member(c,[c]).
member(c,[b,b]).
member(a,[c,c,d,e]).
member(b,[d]).
member(a,[e,c,a,b]).
member(d,[d,a,c,a,b]).
member(b,[b,d,a]).
member(e,[a,a,c,a,c]).
member(a,[a,b,c,d]).
member(e,[a,e,c]).
member(A,[B|C]) :-
  member(A,C).
```

Program 18: Primer definicije predikata `member`, ki jo Progol zgradi pri 100 učnih primerih s stopnjo šuma 0.5 in parametrom 'noise'= 20.

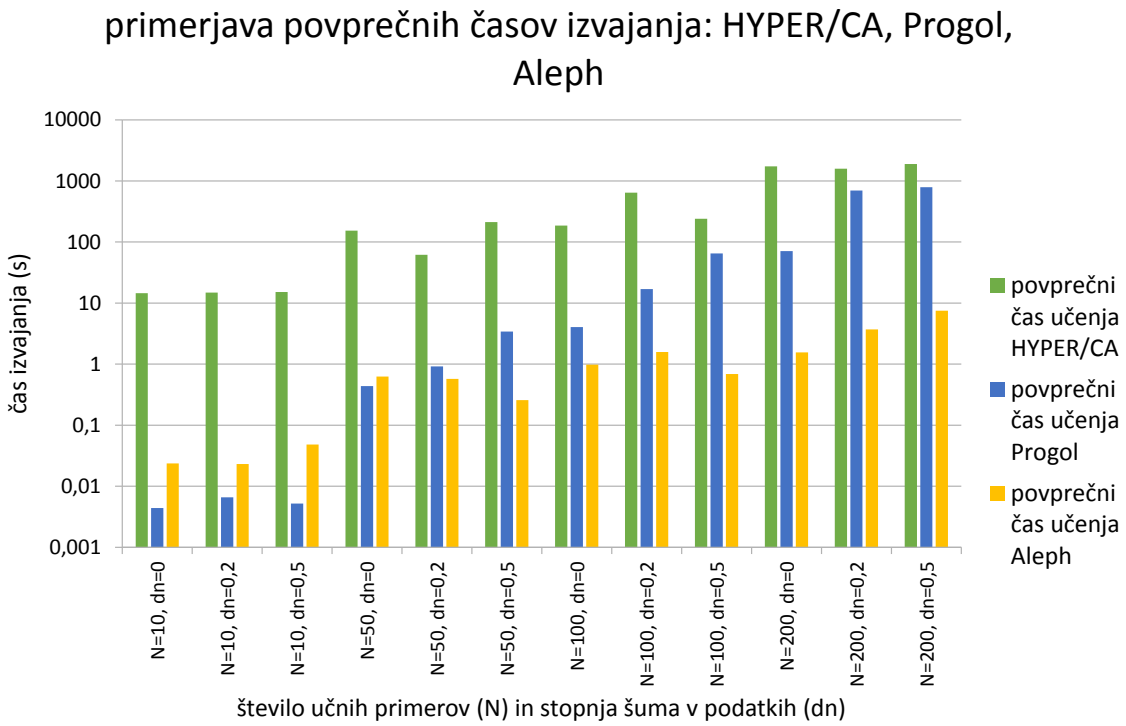
inducira hipotezo v pol minute, Aleph pa v manj kot sekundi. Se pa prednost Progola in Alepha v času izvajanja zmanjšuje z naraščanjem zahtevnosti naloge (z naraščanjem števila primerov in stopnje šuma v podatkih.

Druga očitna razlika pa je, da sistem HYPER/CA gradi bolj preproste hipoteze kot Progol in Aleph (grafa na slikah 6.11 in 6.12). Če primerjamo vrednosti povprečnega števila literalov in spremenljivk, vidimo, da Progol in Aleph za primerljivo točnost zgradita hipotezo s po več deset literali in spremenljivkami, medtem ko sistem HYPER/CA v povprečju nikoli ne preseže štirih literalov in šestih spremenljivk. Ta razlika je posledica v osnovi prekrivne narave algoritmov

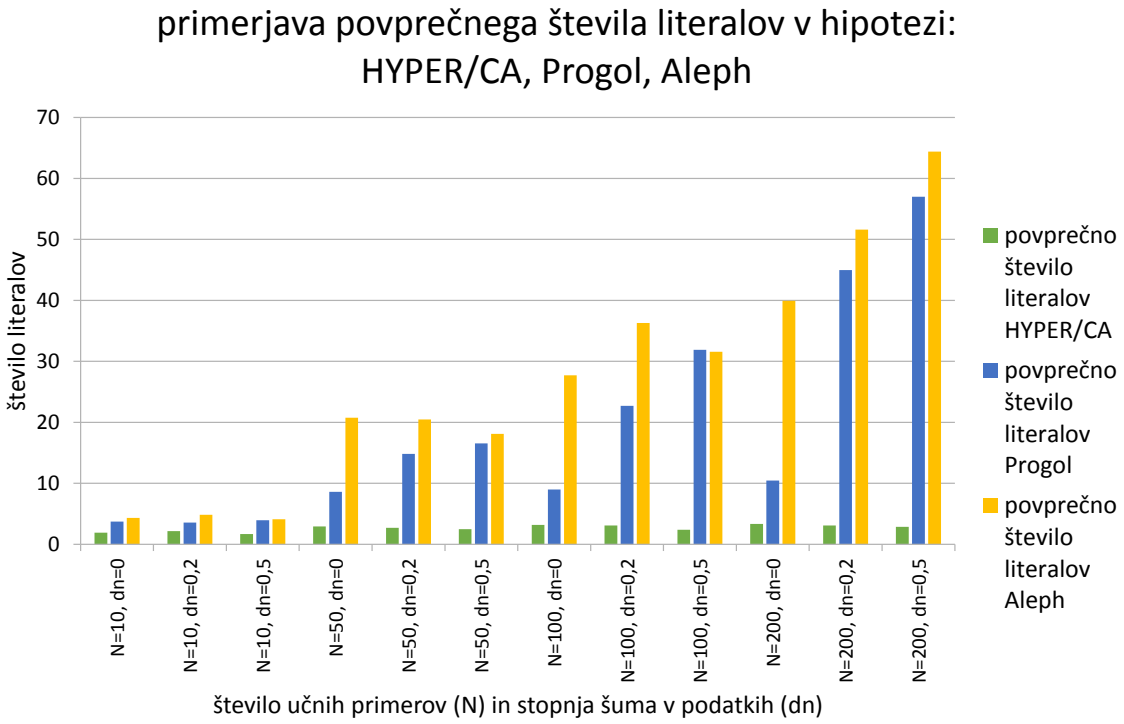
Progol in Aleph, kot smo ju opisali v podrazdelku 6.2.1.



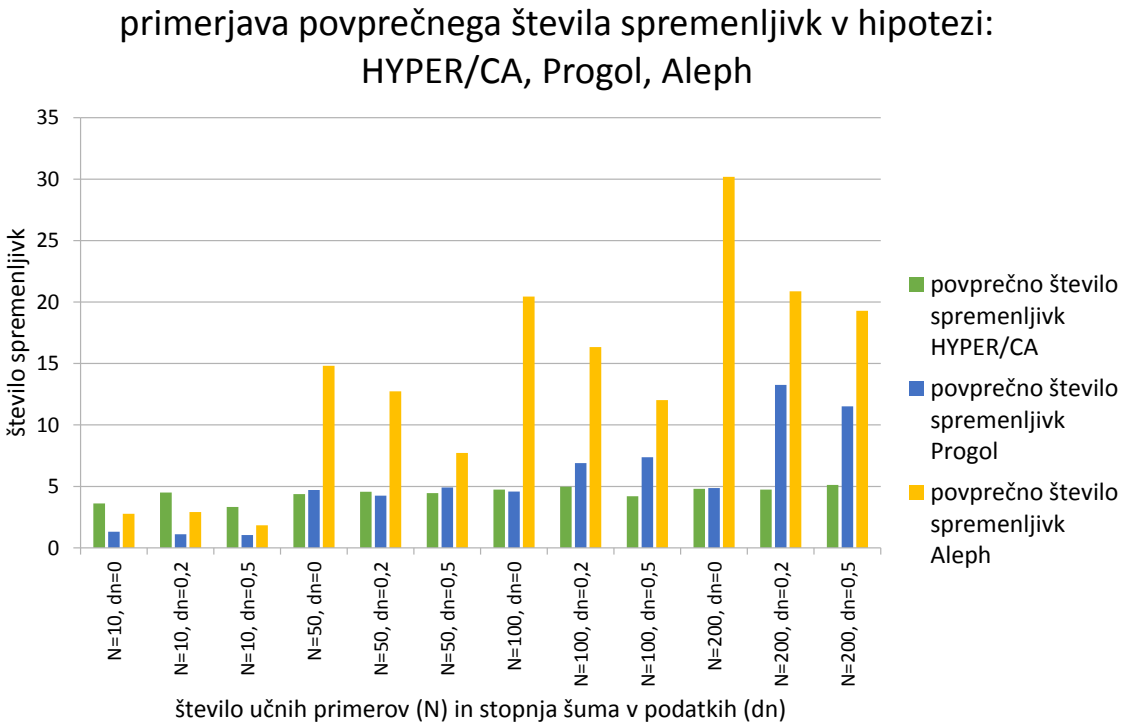
Slika 6.9: Primerjava po vseh poskusih povprečene klasifikacijske točnosti vseh treh testiranih sistemov ILP za vse možne parametre učne množice. Parametra učnega nabora na osi x označimo skrajšano.



Slika 6.10: Primerjava po vseh poskusih povprečenega časa učenja vseh treh testiranih sistemov ILP za vse možne parametre učne množice. Lestvica na navpični osi je logaritemska. Parametra učnega nabora na osi x označimo skrajšano.



Slika 6.11: Primerjava po vseh poskusih povprečenega števila literalov v hipotezi za vse tri testirane sisteme ILP pri vseh možnih parametrih učne množice. Parametra učnega nabora na osi x označimo skrajšano.



Slika 6.12: Primerjava po vseh poskusih povprečenega števila spremenljivk v hipotezi za vse tri testirane sisteme ILP pri vseh možnih parametrih učne množice. Parametra učnega nabora na osi x označimo skrajšano.

Poglavje 7

Zaključni komentar in nadaljnje delo

Povzetek dela

Razvili smo novo metodo za avtonomno modeliranje robotskih akcij imenovano STRUDEL, ki je sposobna kot del modelov odkriti nove abstraktne koncepte. Njeno delovanje smo podrobno opisali v poglavju 3. Metoda STRUDEL iz podatkov, ki jih robot sam zbere s poskusi, zgradi model tipa STRIPS, ki opisuje učinke robotovih akcij. Model je zgrajen z učnimi metodami ILP in je izražen v logiki prvega reda. Metoda zbrane primere akcij najprej razvrsti v skupine glede na podobnost med njihovimi učinki, nato z ILP inducira pogoj, ki loči med skupinami učinkov, in nazadnje inducira model akcij, v katerem uporabi prej inducirani pogoj. Pogoj, ki loči skupine akcij, je po predpostavki neka lastnost predmetov, ki so argument akcij, in definirajo abstraktne in splošne koncepte. Koncepti so abstraktni, ker pred odkritjem niso eksplicitno izraženi v domeni. Splošnost konceptov dosežemo s tem, da so odkriti s splošnim postopkom, brez predpostavk vezanih na domeno.

Pomemben korak metode STRUDEL je razvrščanje akcij v skupine glede na podobnosti njihovih učinkov. Za to nalogo smo razvili metodo ACES, algoritem za razvrščanje relacijskih podatkov v skupine. Kot je opisano v poglavju 4, metoda ACES podobnost učinkov akcij oz. razdaljo med njimi meri na podlagi najboljšega ujemanja grafov, ki jih določajo logične relacije iz opisa učinkov. To mero razdalje uporabi za gradnjo hierarhičnega modela z aglomerativnim razvrščanjem v skupine.

Učne sposobnosti metod STRUDEL in ACES prikažemo z več poskusi opisa-

nimi v poglavju 5. Metodi odkrijeta definicije več abstraktnih konceptov: *premičnost*, *"nižje kot"*, *"ne najvišje"*, *obteženost* in *prôstost*. Tako nekateri koncepti (*"ne najvišje"* in *obteženost*), kot tudi sami modeli akcij (v domeni *prôstost*) so definirani rekurzivno.

Kot jasno pokaže učenje v domeni *prôstost*, razvrščanje v skupine z metodo ACES lahko ne uspe, celo če v podatkih ni prisoten šum. Posledično ni uspešna metoda STRUDEL, če za indukcijo v njej uporabimo sistem HYPER, ker pri učenju ne obravnava šumnih podatkov. Da smo odpravili to pomanjkljivost, smo razvili sistem HYPER/CA, opisan v poglavju 6, ki je nadgrajena različica sistema HYPER, ki obravnava šumne podatke. Sistem HYPER/CA za razliko od sorodnih metod ILP ne potrebuje vnaprej podane pričakovanje stopnje šuma za delovanje. V disertaciji s poskusi pokažemo, da sistem HYPER/CA dosega podobno točnost hipotez, kot najnovejši sistemi ILP. Poleg tega ne njegova prednost tudi manjša nagnjenost k pretiranem prilagajanju in to, da gradi preprostejše hipoteze od sistemov, s katerimi smo ga primerjali. Žal je zaradi svoje zasnove precej počasnejši. Celo do te mere, da učenje lahko traja tako dolgo, da ni praktično izvedljivo.

Komentar računske zahtevnosti razvitih metod

Razvite metode omogočajo avtonomno učenje visokonivojskih in zelo izraznih modelov. Cena, ki jo plačamo za to izrazno moč, je visoka računska zahtevnost. Pri razvrščanju učinkov v skupine metoda ACES za mero razdalje računa ujemanje med grafi, ki ustrezajo posameznim učinkom. V ta namen mora poiskati največji izomorfen podgraf za vsak par grafov, kar je znan NP-poln problem [19]. Za manjše grafe, kot so grafi učinkov v naših domenah, to ni težavno, vendar bi z naraščajočimi grafi in večjim številom primerov razvrščanje lahko postalo računsko neizvedljivo. Izpostaviti velja, da za večji graf učinka ni nujno potrebno, da je fizični učinek robotove akcije zelo velik. Zadošča že, da je robot sposoben zaznati veliko število raznolikih opažanj in je posledično opis spremembe v njegovi predstavitvi znanja velik in kompleksen. Vseeno to ni tako huda omejitev, saj je veliko zanimivih in uporabnih učinkov (npr. premikanje predmetov v domeni *premičnost* ipd.) relativno majhnih. Kot smo že omenili v podrazdelku 4.2.1 pa lahko za iskanje izomorfnih podgrafov uporabimo tudi katerega od aproksimacijskih algoritmov in s tem omilimo težavnost tega koraka.

Drugi vir računske zahtevnosti je ILP, ki ga uporabljamo za učenje. Tudi tu bi z naraščanjem velikosti in kompleksnosti predznanja učenje prej ali slej postalo neučinkovito. Robot, ki bi ga pojmovali kot inteligentnega in razgledanega, bi moral biti sposoben hraniti in izkoristiti obširno predznanje. Primer takega nabora znanja so splošne ontologije, kot je (Open)Cyc [65]. Če bi želeli, da bi bila raba metode STRUDEL skalabilna do te mere, bi morali ustvariti nek mehanizem, ki bi na podlagi meta-znanja ocenil, kako relevanten je nek del predznanja za trenutno nalogo. Na primer, da znanje iz kemije najverjetneje ni uporabno pri poskusih s prestavljanjem kock. Poudariti gre, da je to nivo skalabilnosti, ki ga danes ni sposobna nobena znana metoda. Tako pri tem komentarju ne gre toliko za kritiko, kot za pogled k čemu stremeti v nadaljnjem delu.

Nadaljnje delo

Do sedaj smo v zaključnem poglavju že omenili nekaj možnosti za nadaljnje izboljšave naših metod. V tem podrazdelku strnemo še nekaj idej za nadaljnje raziskave, ki se jih tekom razprave še nismo dotaknili.

V disertaciji se pri učenju z metodo STRUDEL omejimo na koncepte, ki ločijo med dvema učinkoma akcij. Tako pri metodi ACES vemo, da moramo primere deliti na dve skupini. Metodo bi se dalo relativno enostavno posplošiti na učenje modelov akcij z več kot dvema različnima učinkoma, torej da bi moral model imeti več kot dva različna stavka adds in dels. Po gradnji hierarhičnega modela z metodo ACES bi morali oceniti, s katerega nivoja se splača vzeti delitev (tj., na koliko skupin je smiselno razdeliti primere). Kirsten in Wrobel v [48] predlagata, da se vzame delitev, ki maksimira povprečno podobnost znotraj skupin. V primeru, da imamo več kandidatov za delitev, lahko upoštevamo tudi kakovost končnega modela (npr. njegovo točnost in kompleksnost), vendar to poveča zahtevnost učenja, saj moramo za vsako od delitev zgraditi celoten model. Posplošitev zaenkrat prepuščamo za nadaljnje delo.

Ena največjih omejitev našega pristopa je slaba obravnava numeričnih vrednosti. Metoda ACES v trenutni različici ne računa razdalj med numeričnimi vrednostmi, saj ni očitno, kako jih upoštevati. Pri odkrivanju koncepta premičnosti s kockami recimo ni smiselno upoštevati dolžine premika kocke, saj za samo premičnost to ni pomembno. Če bi primerjali, kako se po potisku obnašata kocka in žoga,

pa bi to bil bistveno relevantnejši podatek, saj se žoge odkotalijo dlje kot kocke. Možno je, da bi to težavo do neke mere rešilo že ocenjevanje kakovosti delitev, kot smo jo opisali v prejšnjem odstavku. V vsakem primeru bi morali premisliti, kako upoštevati velikost razdalje med numeričnimi atributi v razdalji med grafi. Gre za drug tip razdalje, saj ta ne meri razlike v strukturi sprememb, temveč v vrednosti atributov, ki so del strukture. Poleg numeričnih atributov bi se dalo na tak način upoštevati tudi ordinalne atribute. Upoštevanje vrednosti numeričnih in ordinalnih atributov pri razvrščanju v skupine prepuščamo za nadaljnje delo.

Osnova ideja metode STRUDEL je, da lahko strukturne razlike v podatkih o robotskih akcijah izkoristimo za to, da razdelimo učenje na posamezne faze in ga s tem olajšamo. To odpira vprašanje, če in v kaki meri bi lahko isti princip uporabili v splošnem učenju ILP. Metoda STRUDEL je v robotskih domenah, ki smo jih predstavili, uspešna, ker opazuje strukturne razlike v podatkih in so učinki akcij odvisni od postavitve (strukture) predmetov v svetu. Ni očitno, če to velja širše v problemih ILP. Poleg tega metoda ACES za učinkovito razvrščanje v skupine potrebuje množice relacij, v problemih ILP pa so primeri ponavadi posamezna dejstva. Lahko bi generirali neke vrste okolico vsakega primera s tem, da bi iz predznanja generirali opredeljene literale, v katerih nastopajo atomi iz primera in potem to ponavljali (tj., generirali večjo okolico tako, da bi generirali opredeljene literale, v katerih nastopajo atomi iz prejšnje okolice), glede na to, kako veliko množico relacij želimo imeti. To in ostala podvprašanja iz te precej splošne teme ostaja predmet nadaljnjega dela.

Ponoven pregled in komentar prispevkov k znanosti

Za konec ponovimo znanstvene prispevke disertacije s komentarji:

STRUDEL: Razvili smo metodo STRUDEL, s katero lahko avtonomen robot iz podatkov, ki jih sam zbere s poskusi, zgradi model tipa STRIPS, ki opisuje učinke njegovih akcij. Principi delovanja metode STRUDEL so opisani v poglavju 3. V praksi smo njeno delovanje predstavili v poskusih v poglavju 5.

Metoda temelji na predpostavki, da lahko različne učinke med seboj ločimo na podlagi njihove strukture (tj. strukture relacij s katerimi so opisani). Z uporabo tega principa metoda STRUDEL razdeli učno nalogo v dele tako,

da najprej loči učinke glede na podobnosti v njihovih strukturah, potem poišče pogoje, ki ločijo med skupinami učinkov, in nazadnje inducira model akcije za vsak učinek posebej. Ta predpostavka ne velja vedno in ni očitno, kako splošno velja ter v kakšnih primerih. Odprtost vprašanja, kdaj točno lahko uporabimo metodo STRUDEL, je ena pomanjkljivosti te disertacije, ki bi jo bilo koristno teoretično in s poskusi raziskati v nadaljnjem delu. Domnevamo, da v domenah iz robotike, ki so fokus našega dela, metoda STRUDEL dobro deluje, ker struktura relacij, ki opisujejo učinke, dobro odraža fizično strukturo robotovega sveta, od katere so odvisni učinki akcij.

Delitev učinkov glede na njihovo strukturo opravi metoda STRUDEL z uporabo metode ACES opisane v poglavju 4 in je posledično omejena s pomanjkljivostmi te metode (računska zahtevnost, slaba obravnava numeričnih vrednosti). Implementacija metode STRUDEL ni neločljivo vezana na metodo ACES, tako da bi slednjo lahko nadomestili z neko morebitno boljšo metodo in s tem izboljšali delovanje metode STRUDEL.

Kot jasno pokaže poskus opisan v razdelku 5.2.4, je lahko razvrščanje učinkov v skupine z metodo ACES neuspešno tudi, če v domeni ni šuma. V tem primeru lahko za učenje uspešno uporabimo sistem ILP, ki je sposoben delati s šumnimi podatki, kot prikažemo v istem poskusu.

Pri gradnji modelov z metodo STRUDEL odkriti pogoji, ki ločijo med različnimi učinki, lahko definirajo nove abstraktne koncepte, s katerimi lahko robot razširi svoj jezik hipotez. Tako ti koncepti kot tudi celotni modeli učinkov akcij so lahko definirani rekurzivno. Ni nam znan noben drug pristop s področja kognitivne robotike ali učenja v planiranju, ki bi bil sposoben graditi tako splošne rekurzivne modele.

Nekatere omejitve metode STRUDEL so posledica uporabe metod ILP za učenje. Zaradi tega je metoda STRUDEL računsko precej zahtevna in občutljiva na podano predznanje, saj lahko majhne spremembe v njem pomenijo veliko povečanje računske zahtevnosti ali pa celo naredijo učenje neizvedljivo. Ker so podatki, ki opisujejo robotov svet, po naravi relacijski, uporaba propozicionalnih pristopov učenja ni enostavno možna. Vsaj ne brez žrtvovanja dela splošnosti in izrazne moči modelov, ki jih gradimo. Rešitve za problem zahtevnosti lahko iščemo v stohastičnih pristopih kot je v uvodu (poglavje

1) omenjeno genetsko programiranje. Na ta način morda ne bomo dobili optimalne hipoteze, a lahko zmanjšamo računsko zahtevnost učenja.

ACES: Razvili smo metodo za razvrščanje v skupine ACES, ki robotove akcije razvrsti v skupine glede na podobnosti strukture njihovih učinkov (tj. strukture relacij s katerimi so opisani). Metoda ACES vsaki množici relacij, ki opisuje učinek posamezne akcije, priredi graf. Na podlagi podobnosti strukture teh grafov izračuna razdalje med učinki.

Funkcija razdalje metode ACES mora za izračun razdalje med dvema grafoma poiskati njun največji izomorfen podgraf. To je znan NP-poln problem, tako da je računska zahtevnost v odvisnosti od velikosti grafov ena od omejitev metode ACES. V naših poskusih (poglavje 5) pokažemo, da se tudi z akcijami z majhnimi učinki lahko odkrije zanimive in uporabne koncepte. V primeru, da bi računska zahtevnost iskanja največjega izomorfnega podgrafa postala težavna, lahko v ta namen uporabimo katerega od aproksimativnih algoritmov za ta problem.

Metoda ACES je zasnovana tako, da primerja predvsem strukturo učinkov oziroma njim prirejenih grafov. Ker nismo uspeli razviti načina, da bi v funkciji razdalje ob tem uspešno upoštevali tudi razlike v vrednosti numeričnih argumentov v učinkih, so slednje skoraj popolnoma zanemarjene pri izračunu razdalje. Upošteva se le njihov tip pri iskanju izomorfnih podgrafov. Gre za hibo, saj to pomeni, da metoda ACES ne zna ločiti med učinki, ki se razlikujejo po nekih numeričnih vrednostih v učinkih (npr. dolžini premika). Problem, kako v metodi ACES smiselno upoštevati numerične vrednosti, ostaja odprt za nadaljnje delo.

Za razliko od podobnih metod, ki relacijske podatke razvrščajo v skupine na podlagi podobnosti grafov, ki jih določajo, funkcija razdalje metode ACES poleg velikosti razlike med grafoma upošteva tudi velikost dela, ki jima je skupen. Zaradi tega metoda ACES razvršča primere v skupine na način ugoden za učenje rekurzivnih hipotez, ker loči robne primere od rekurzivnih.

HYPER/CA: Razvili smo sistem HYPER/CA, ki je nadgrajena verzija sistema HYPER, ki je sposobna obravnavati šumne podatke. Zasnova in delovanje sistema HYPER/CA sta podrobno opisana v poglavju 6. V istem poglavju

je opisana tudi primerjava s poskusi s sistemoma Progol in Aleph.

Sistem HYPER/CA uporablja neprekriven pristop, da zgradi hipotezo zapisano v jeziku prolog, ki loči med pozitivnimi in negativnimi primeri v učni množici. Hipoteza se zgradi s preiskovanjem grafa izostritev od zgoraj navzdol. Cenilna funkcija, ki se pri tem uporablja, optimizira število napačno klasificiranih primerov in velikost prostora, ki smo ga morali preiskati, da smo našli trenutno hipotezo. Upoštevanje velikosti preiskanega prostora in ne velikosti hipoteze, poskrbi, da je sistem HYPER/CA za enako izboljšanje točnosti hipoteze pri večjih hipotezah manj pripravljen preiskovati naprej kot pri manjših hipotezah. Ni nam znan noben drug sistem ILP, ki bi v cenilni funkciji upošteval velikost preiskanega prostora.

Za delo s šumnimi podatki sistemi ILP pogosto potrebujejo pričakovano stopnjo šuma podano kot vhodni parameter. Sistem HYPER/CA pričakovane stopnje šuma za delovanje ne potrebuje.

Primerjava s poskusi s sistemoma Progol in Aleph kaže, da je sistem HYPER/CA sposoben graditi primerljivo točne hipoteze. Poleg tega rezultati kažejo, da je sistem HYPER/CA manj nagnjen k pretiranem prilagajanju (overfitting) in gradi bistveno manjše hipoteze, kot Progol in Aleph. Žal je pri tem dokaj počasnejši, s časi učenja, ki so pogosto za več redov velikosti večji. Ta počasnost je velika hiba, saj traja učenje s sistemom HYPER/CA pogosto zelo dolgo, in lahko zaradi tega postane praktično neizvedljivo. Ostajajo še možnosti, kako pospešiti delovanje sistema HYPER/CA. Lahko recimo implementiramo še ostale izboljšave iz Lebanovega sistem HYPER/X (podrazdelek 6.1.1), ki jih do sedaj še nismo.

Sklepna beseda

Vse opisane metode so novosti, ki širijo znanstvena področja na katera spadajo. Omogočajo avtonomno učenje splošnih abstraktnih konceptov, kakršnih se, kolikor je nam znano, do sedaj ni učil še noben znan pristop. Taki koncepti so med drugim pomemben element človeškega razumevanja sveta. Zaradi zahtevnosti in relativno majhne neposredne uporabnosti v praktičnih aplikacijah je v področju strojnega učenja v zadnjem času učenje tovrstnih konceptov dokaj slabo zasto-

pano. Verjamemo, da naše delo predstavlja zanimiv in uporaben prispevek v tem pomembnem, a morda malo zapostavljenem področju.

Dodatek A

Definicije eksperimentalnih domen

V tem dodatku podajamo definicijo vseh domen iz poglavja 5 skupaj z učnimi podatki, ki smo jih uporabili. Vse definicije so zapisane kot logični programi. Uporabljen jezik je SICStus prolog, a specifičnih programskih konstruktov vezanih na to verzijo Prologa ne uporabljamo, tako da bi morale delovati tudi v drugih verzijah (SWI, Yap ipd.). Vsi materiali iz tega dodatka so skupaj s kodo potrebno, da se požene učenje, dostopni tudi na elektronskem naslovu `http://www.ailab.si/aljaz/aljaz_kosmerlj_phd_code.zip`.

A.1 Učenje koncepta premičnosti

A.1.1 Učenje koncepta premičnosti: podatki

```
box(a).   % premična
box(b).   % premična
box(c).   % nepremična
box(d).   % premična
box(e).   % premična
box(f).   % nepremična
box(g).   % premična
box(h).   % nepremična

state(0, [
```

```
  at(a, [1,3]), at(b, [8,7]),
  at(c, [9,2]), at(d, [5,7]),
  at(e, [2,4]), at(f, [6,6]),
  at(g, [3,9]), at(h, [6,2])]).

action(0, move(a, [1,3], [1,-2], [2,1])).

state(1, [
  at(a, [2,1]), at(b, [8,7]),
  at(c, [9,2]), at(d, [5,7]),
  at(e, [2,4]), at(f, [6,6]),
  at(g, [3,9]), at(h, [6,2])]).

action(1, move(b, [8,7], [-3,2], [5,9])).

state(2, [
  at(a, [2,1]), at(b, [5,9]),
  at(c, [9,2]), at(d, [5,7]),
  at(e, [2,4]), at(f, [6,6]),
  at(g, [3,9]), at(h, [6,2])]).

action(2, move(c, [9,2], [1,5], [9,2])).

state(3, [
  at(a, [2,1]), at(b, [5,9]),
  at(c, [9,2]), at(d, [5,7]),
  at(e, [2,4]), at(f, [6,6]),
  at(g, [3,9]), at(h, [6,2])]).

action(3, move(d, [5,7], [0,-4], [5,3])).

state(4, [
  at(a, [2,1]), at(b, [5,9]),
  at(c, [9,2]), at(d, [5,3]),
  at(e, [2,4]), at(f, [6,6]),
  at(g, [3,9]), at(h, [6,2])]).
```

```
action(4, move(e, [2,4], [6,1], [8,5])).

state(5, [
  at(a, [2,1]), at(b, [5,9]),
  at(c, [9,2]), at(d, [5,3]),
  at(e, [8,5]), at(f, [6,6]),
  at(g, [3,9]), at(h, [6,2])]).

action(5, move(f, [6,6], [-3,-2], [6,6])).

state(6, [
  at(a, [2,1]), at(b, [5,9]),
  at(c, [9,2]), at(d, [5,3]),
  at(e, [8,5]), at(f, [6,6]),
  at(g, [3,9]), at(h, [6,2])]).

action(6,
    move(g, [3,9], [-2,0], [1,9])).

state(7, [
  at(a, [2,1]), at(b, [5,9]),
  at(c, [9,2]), at(d, [5,3]),
  at(e, [8,5]), at(f, [6,6]),
  at(g, [1,9]), at(h, [6,2])]).

action(7, move(h, [6,2], [4,4], [6,2])).

state(8, [
  at(a, [2,1]), at(b, [5,9]),
  at(c, [9,2]), at(d, [5,3]),
  at(e, [8,5]), at(f, [6,6]),
  at(g, [1,9]), at(h, [6,2])]).

action(8, move(b, [5,9], [2,-4], [7,5])).
```

```
state(9, [
  at(a, [2,1]), at(b, [7,5]),
  at(c, [9,2]), at(d, [5,3]),
  at(e, [8,5]), at(f, [6,6]),
  at(g, [1,9]), at(h, [6,2])]).

action(9, move(c, [9,2], [3,3], [9,2])).

state(10, [
  at(a, [2,1]), at(b, [7,5]),
  at(c, [9,2]), at(d, [5,3]),
  at(e, [8,5]), at(f, [6,6]),
  at(g, [1,9]), at(h, [6,2])]).
```

A.1.2 Učenje koncepta premičnosti: učenje pogoja

```
% BACKGROUND KNOWLEDGE
backliteral( in_state(T, at(Box, P)), [],
             [T:time, Box:box, P:position], []).

in_state(Time, Observation) :-
  state(Time, Observations),
  member(Observation, Observations).

backliteral( add(Pos1, Dist, Pos2), [Pos1:position, Dist:distance],
                                    [Pos2: position], []).

add([X1,Y1], [DX,DY], [X2,Y2]):-
  X2 is X1 + DX,
  Y2 is Y1 + DY.

backliteral( different( Pos1, Pos2), [Pos1:position, Pos2:position],
             [], []).
```

```
different([X1,Y1],[X2,Y2]) :-
    X1 \== X2
  ;
    Y1 \== Y2.

% ----------

% no term clauses
term( defaultterm, [], []).

% ----------

prolog_predicate( in_state(_, _)).
prolog_predicate( add(_, _, _)).
prolog_predicate( different(_, _)).

% ----------

start_clause( [ p(Box, T)] /  [Box:box, T:time]).
```

A.1.3 Učenje koncepta premičnosti: učenje `adds`

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

backliteral( in_state(T, at(Box, P)), [],
             [T:time, Box:box, P:position], []).

in_state(Time, Observation) :-
  state(Time, Observations),
  member(Observation, Observations).

backliteral( add(Pos1, Dist, Pos2), [Pos1:position, Dist:distance],
                                    [Pos2: position], []).

add([X1,Y1], [DX,DY], [X2,Y2]):-
  X2 #= X1 + DX,
  Y2 #= Y1 + DY.

backliteral( different( Pos1, Pos2), [Pos1:position, Pos2:position],
             [], []).

different([X1,Y1],[X2,Y2]) :-
    X1 \== X2
  ;
    Y1 \== Y2.

p(Box):-
  in_state(T1, at(Box, Pos1)),
  in_state(T2, at(Box, Pos2)),
  different(Pos1, Pos2).

% ----------

term( observation, at( Box, Pos), [Box:box, Pos:position]).
```

```
term( list_of_observations, [], []).
term( list_of_observations, [O|OL],
      [O:observation, OL:list_of_observations]).

% ----------

prolog_predicate( in_state(_, _)).
prolog_predicate( add(_, _, _)).
prolog_predicate( different(_, _)).
prolog_predicate( p(_)).
prolog_predicate( \+ _).

% ----------

start_clause( [ adds(move(Box, Pos1, Dist, Pos2), T, Dels),
                    p(Box)] /
                [Box:box, Dist:distance, Dels:list_of_observations,
                 T:time, Pos1:position, Pos2:position]).

start_clause( [ adds(move(Box, Pos1, Dist, Pos2), T, Dels),
                    \+ p(Box)] /
                [Box:box, Dist:distance, Dels:list_of_observations,
                 T:time, Pos1:position, Pos2:position]).
```

A.1.4 Učenje koncepta premičnosti: učenje `dels`

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

backliteral( in_state(T, at(Box, P)), [],
             [T:time, Box:box, P:position], []).

in_state(Time, Observation) :-
  state(Time, Observations),
```

```
  member(Observation, Observations).

backliteral( add(Pos1, Dist, Pos2), [Pos1:position, Dist:distance],
                                    [Pos2: position], []).

add([X1,Y1], [DX,DY], [X2,Y2]):-
  X2 #= X1 + DX,
  Y2 #= Y1 + DY.

backliteral( different( Pos1, Pos2), [Pos1:position, Pos2:position],
             [], []).

different([X1,Y1],[X2,Y2]) :-
    X1 \== X2
  ;
    Y1 \== Y2.

p(Box):-
  in_state(T1, at(Box, Pos1)),
  in_state(T2, at(Box, Pos2)),
  different(Pos1, Pos2).

% ----------

term( observation, at( Box, Pos), [Box:box, Pos:position]).

term( list_of_observations, [], []).
term( list_of_observations, [O|OL],
      [O:observation, OL:list_of_observations]).

% ----------

prolog_predicate( in_state(_, _)).
prolog_predicate( add(_, _, _)).
prolog_predicate( different(_, _)).
prolog_predicate( p(_)).
```

```
prolog_predicate( \+ _).

% ----------

start_clause( [ dels(move(Box, Pos1, Dist, Pos2), T, Dels),
                    p(Box)] /
                [Box:box, Dist:distance, Dels:list_of_observations,
                 T:time, Pos1:position, Pos2:position]).

start_clause( [ dels(move(Box, Pos1, Dist, Pos2), T, Dels),
                    \+ p(Box)] /
                [Box:box, Dist:distance, Dels:list_of_observations,
                 T:time, Pos1:position, Pos2:position]).
```

A.2 Učenje konceptov “ne najvišje” in “nižje kot”

A.2.1 Učenje konceptov “ne najvišje” in “nižje kot”: podatki

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

```
object(none).

state(0, [
  on(a,b), on(b,c), on(c,d), on_floor(d),
  on(e,f), on_floor(f),
  on_floor(g),
  on(h,i), on(i,j), on(j,k), on(k,l), on(l,m), on_floor(m)]).

action( 0, grab(e, none)).

state(1, [
  on(a,b), on(b,c), on(c,d), on_floor(d),
  on(e,f), on_floor(f),
  on_floor(g),
  on(h,i), on(i,j), on(j,k), on(k,l), on(l,m), on_floor(m)]).

action( 1, grab(g, none)).

state(2, [
  on(a,b), on(b,c), on(c,d), on_floor(d),
  on(e,f), on_floor(f),
  on_floor(g),
  on(h,i), on(i,j), on(j,k), on(k,l), on(l,m), on_floor(m)]).

action( 2, grab(a, none)).

state(3, [
  on(a,b), on(b,c), on(c,d), on_floor(d),
  on(e,f), on_floor(f),
  on_floor(g),
  on(h,i), on(i,j), on(j,k), on(k,l), on(l,m), on_floor(m)]).

action( 3, grab(h, h)).

state(4, [
  on(a,b), on(b,c), on(c,d), on_floor(d),
```

```
  on(e,f), on_floor(f),
  on_floor(g),
  on(i,j), on(j,k), on(k,l), on(l,m), on_floor(m),
  taken(h)]).

action( 4, grab(a, none)).

state(5, [
  on(a,b), on(b,c), on(c,d), on_floor(d),
  on(e,f), on_floor(f),
  on_floor(g),
  on(i,j), on(j,k), on(k,l), on(l,m), on_floor(m),
  taken(h)]).

action( 5, grab(e, none)).

state(6, [
  on(a,b), on(b,c), on(c,d), on_floor(d),
  on(e,f), on_floor(f),
  on_floor(g),
  on(i,j), on(j,k), on(k,l), on(l,m), on_floor(m),
  taken(h)]).

action(6 , grab(i, i)).

state(7, [
  on(a,b), on(b,c), on(c,d), on_floor(d),
  on(e,f), on_floor(f),
  on_floor(g),
  on(j,k), on(k,l), on(l,m), on_floor(m),
  taken(h),
  taken(i)]).

action(7 , grab(j, j)).

state(8, [
```

```
  on(a,b), on(b,c), on(c,d), on_floor(d),
  on(e,f), on_floor(f),
  on_floor(g),
  on(k,l), on(l,m), on_floor(m),
  taken(h), taken(i),
  taken(j)]).

action(8 , grab(k, none)).

state(9, [
  on(a,b), on(b,c), on(c,d), on_floor(d),
  on(e,f), on_floor(f),
  on_floor(g),
  on(k,l), on(l,m), on_floor(m),
  taken(h), taken(i),
  taken(j)]).

action(9 , grab(g, none)).

state(10, [
  on(a,b), on(b,c), on(c,d), on_floor(d),
  on(e,f), on_floor(f),
  on_floor(g),
  on(k,l), on(l,m), on_floor(m),
  taken(h), taken(i),
  taken(j)]).

action(10 , grab(a, a)).

state(11, [
  on(b,c), on(c,d), on_floor(d),
  on(e,f), on_floor(f),
  on_floor(g),
  on(k,l), on(l,m), on_floor(m),
  taken(h), taken(i),
  taken(j), taken(a)]).
```

```
action(11 , grab(b, b)).

state(12, [
  on(c,d), on_floor(d),
  on(e,f), on_floor(f),
  on_floor(g),
  on(k,l), on(l,m), on_floor(m),
  taken(h), taken(i),
  taken(j), taken(a),
  taken(b)]).

action(12 , grab(c, none)).

state(13, [
  on(c,d), on_floor(d),
  on(e,f), on_floor(f),
  on_floor(g),
  on(k,l), on(l,m), on_floor(m),
  taken(h), taken(i),
  taken(j), taken(a),
  taken(b)]).
```

A.2.2 Učenje konceptov “ne najvišje” in “nižje kot”: učenje pogoja

---

```
backliteral( in_state(T, on( Box1, Box2)),
             [T:time, Box1:box], [Box2:box], [Box1 \== Box2]).
backliteral( in_state(T, on_floor(Box)), [T:time, Box:box], [], []).

in_state(Time, Observation) :-
  state(Time, Observations),
  member(Observation, Observations).
```

```
backliteral( r(Box1, Box2, Time),
             [Box1:box, Time:time], [Box2:box], [Box1 \== Box2]).

% ----------

% no term clauses
term( defaultterm, [], []).

% ----------

prolog_predicate( in_state(_, _)).

% ----------

start_clause( [ p(Box, T)] /  [Box:box, T:time]).
start_clause( [ r(Box1, Box2, T)] /  [Box1:box, Box2:box, T:time]).
start_clause( [ r(Box1, Box2, T)] /  [Box1:box, Box2:box, T:time]).
```

A.2.3 Učenje konceptov "ne najvišje" in "nižje kot": učenje adds

---

```
backliteral( in_state(T, on( Box1, Box2)), [T:time, Box1:box],
             [Box2:box], [Box1 \== Box2]).

in_state(Time, Observation) :-
  state(Time, Observations),
  member(Observation, Observations).

p(Box, Time):-
    r(Box, Box1, Time).

r(Box1, Box2, Time):-
```

```
    in_state(Time, on_floor(Box1)),
    in_state(Time, on(Box2, Box3)).

r(Box1, Box2, Time):-
    in_state(Time, on(Box1, Box3)),
    r(Box3, Box4, Time),
    in_state(Time, on(Box2, Box4)).

% ----------

term( observation, on( Box1, Box2), [Box1:box, Box2:box]).
term( observation, on_floor(Box), [Box:box]).
term( observation, taken(Box), [Box:box]).

term( list_of_observations, [], []).
term( list_of_observations, [O], [O:observation]).

term( box, none, []).

% ----------

prolog_predicate( in_state(_, _)).
prolog_predicate( \+ _).
prolog_predicate( p(_, _)).

% ----------

start_clause( [ adds(grab(Box1, Box2), Time, Adds),
                    p(Box1, Time)] /
                [Box1:box, Box2:box, Adds:list_of_observations,
                 Time:time]).

start_clause( [ adds(grab(Box1, Box2), Time, Adds),
                    \+ p(Box1, Time)] /
                [Box1:box, Box2:box, Adds:list_of_observations,
                 Time:time]).
```

A.2.4 Učenje konceptov "ne najvišje" in "nižje kot": učenje dels

```
backliteral( in_state(T, on( Box1, Box2)), [T:time, Box1:box],
             [Box2:box], [Box1 \== Box2]).

in_state(Time, Observation) :-
  state(Time, Observations),
  member(Observation, Observations).

p(Box, Time):-
    r(Box, Box1, Time).

r(Box1, Box2, Time):-
    in_state(Time, on_floor(Box1)),
    in_state(Time, on(Box2, Box3)).

r(Box1, Box2, Time):-
    in_state(Time, on(Box1, Box3)),
    r(Box3, Box4, Time),
    in_state(Time, on(Box2, Box4)).

% ----------

term( observation, on( Box1, Box2), [Box1:box, Box2:box]).
term( observation, on_floor(Box), [Box:box]).
term( observation, taken(Box), [Box:box]).

term( list_of_observations, [], []).
term( list_of_observations, [O], [O:observation]).

term( box, none, []).
```

```
% ----------

prolog_predicate( in_state(_, _)).
prolog_predicate( \+ _).
prolog_predicate( p(_, _)).

% ----------

start_clause( [ dels(grab(Box1, Box2), Time, Dels),
                    p(Box1, Time)] /
                [Box1:box, Box2:box, Dels:list_of_observations,
                 Time:time]).

start_clause( [ dels(grab(Box1, Box2), Time, Dels),
                    \+ p(Box1, Time)] /
                [Box1:box, Box2:box, Dels:list_of_observations,
                 Time:time]).
```

A.3 Učenje Koncepta obteženosti

A.3.1 Učenje Koncepta obteženosti: podatki

```
object(a).
object(b).
object(c).
object(d).
object(e).
object(f).
object(g).
object(h).
object(i).
```

```
object(j).
object(k).
object(l).
object(m).
object(n).
object(o).

state(1, [
  at(a, [2,3]), on(b,a), on(c,b), on(d,c),
  at(e, [7,4]),
  at(f, [9,7]), on(g,f),
  at(h, [1,8]), on(i,h), on(j,i), on(k,j), on(l,k),
  at(m, [5,7]), on(n,m), on(o,n)]).

action(1, move(a, [2,3], [1,-1], [2,3])).

state(2, [
  at(a, [2,3]), on(b,a), on(c,b), on(d,c),
  at(e, [7,4]),
  at(f, [9,7]), on(g,f),
  at(h, [1,8]), on(i,h), on(j,i), on(k,j), on(l,k),
  at(m, [5,7]), on(n,m), on(o,n)]).

action(2, move(e, [7,4], [-3,4], [4,8])).

state(3, [
  at(a, [2,3]), on(b,a), on(c,b), on(d,c),
  at(e, [4,8]),
  at(f, [9,7]), on(g,f),
  at(h, [1,8]), on(i,h), on(j,i), on(k,j), on(l,k),
  at(m, [5,7]), on(n,m), on(o,n)]).

action(3, move(f, [9,7], [-2,2], [7,9])).

state(4, [
  at(a, [2,3]), on(b,a), on(c,b), on(d,c),
```

```
  at(e, [4,8]),
  at(f, [7,9]), on(g,f),
  at(h, [1,8]), on(i,h), on(j,i), on(k,j), on(l,k),
  at(m, [5,7]), on(n,m), on(o,n)
  ]).

action(4, move(h, [1,8], [6,-4], [1,8])).

state(5, [
  at(a, [2,3]), on(b,a), on(c,b), on(d,c),
  at(e, [4,8]),
  at(f, [7,9]), on(g,f),
  at(h, [1,8]), on(i,h), on(j,i), on(k,j), on(l,k),
  at(m, [5,7]), on(n,m), on(o,n)]).

action(5, move(m, [5,7], [3,-4], [8,3])).

state(6, [
  at(a, [2,3]), on(b,a), on(c,b), on(d,c),
  at(e, [4,8]),
  at(f, [7,9]), on(g,f),
  at(h, [1,8]), on(i,h), on(j,i), on(k,j), on(l,k),
  at(m, [8,3]), on(n,m), on(o,n)]).

action(6, move(a, [2,3], [-6,2], [2,3])).

state(7, [
  at(a, [2,3]), on(b,a), on(c,b), on(d,c),
  at(e, [4,8]),
  at(f, [7,9]), on(g,f),
  at(h, [1,8]), on(i,h), on(j,i), on(k,j), on(l,k),
  at(m, [8,3]), on(n,m), on(o,n)]).

action(7, move(f, [7,9], [0,-2], [7,7])).

state(8, [
```

```
  at(a, [2,3]), on(b,a), on(c,b), on(d,c),
  at(e, [4,8]),
  at(f, [7,7]), on(g,f),
  at(h, [1,8]), on(i,h), on(j,i), on(k,j), on(l,k),
  at(m, [8,3]), on(n,m), on(o,n)]).
```

A.3.2 Učenje Koncepta obteženosti: učenje pogoja

```
backliteral( in_state(T, at(Box, P)),
             [], [T:time, Box:box, P:position], []).
backliteral( in_state(T, on(Box1, Box2)),
             [T:time, Box1:box, Box2:box], [], [Box1 \== Box2]).
backliteral( in_state(T, on(Box1, Box2)),
             [T:time, Box2:box], [Box1:box], [Box1 \== Box2]).
backliteral( in_state(T, on(Box1, Box2)),
             [T:time, Box1:box], [Box2:box], [Box1 \== Box2]).

in_state(Time, Observation) :-
  state(Time, Observations),
  member(Observation, Observations).


backliteral( add(Pos1, Dist, Pos2), [Pos1:position, Dist:distance],
                                    [Pos2: position], []).

add([X1,Y1], [DX,DY], [X2,Y2]):-
  X2 is X1 + DX,
  Y2 is Y1 + DY.

backliteral( different( Pos1, Pos2), [Pos1:position, Pos2:position],
             [], []).

different([X1,Y1],[X2,Y2]) :-
```

```
    X1 \== X2
  ;
    Y1 \== Y2.

backliteral( r(Box, BL, T), [Box:box, T:time], [BL:box_list], []).
backliteral( r(Box, BL, T), [Box:box, BL:box_list2, T:time], [], []).

% ----------

term( box_list, [], []).
term( box_list, [B], [B:box]).
term( box_list, [B1,B2], [B1:box, B2:box]).
term( box_list, [B1,B2,B3], [B1:box, B2:box, B3:box]).

term( box_list1, [], []).
term( box_list1, [B|BL], [B:box, BL:box_list2]).

% ----------

prolog_predicate( in_state(_, _)).

% ----------

start_clause( [ p(Box, T)] /  [Box:box, T:time]).
start_clause( [ r(Box, BL, T)] /  [Box:box, BL:box_list1, T:time]).
start_clause( [ r(Box, BL, T)] /  [Box:box, BL:box_list1, T:time]).
```

A.3.3 Učenje Koncepta obteženosti: učenje adds

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

backliteral( in_state(T, at(Box, P)),
             [], [T:time, Box:box, P:position], []).
```

```
backliteral( in_state(T, on(Box1, Box2)),
             [T:time, Box1:box, Box2:box], [], [Box1 \== Box2]).

in_state(Time, Observation) :-
  state(Time, Observations),
  member(Observation, Observations).


backliteral( add(Pos1, Dist, Pos2), [Pos1:position, Dist:distance],
                                    [Pos2: position], []).

add([X1,Y1], [DX,DY], [X2,Y2]):-
  X2 #= X1 + DX,
  Y2 #= Y1 + DY.

backliteral( different( Pos1, Pos2), [Pos1:position, Pos2:position],
             [], []).

different([X1,Y1],[X2,Y2]) :-
    X1 \== X2
  ;
    Y1 \== Y2.

p(Box, Time):-
    r(Box, [C,D,E], Time).

r(Box, [], Time).
r(Box, [O|Os], Time):-
    in_state(Time, on(O, Box)),
    r(O, Os, Time).

% ----------

term( observation, at( Box, Pos), [Box:box, Pos:position]).
term( observation, on( Box1, Box2), [Box1:box, Box2:box]).
```

```
term( list_of_observations, [], []).
term( list_of_observations, [O|OL],
      [O:observation, OL:list_of_observations]).

% ----------

prolog_predicate( in_state(_, _)).
prolog_predicate( add(_, _, _)).
prolog_predicate( different(_, _)).
prolog_predicate( p(_,_)).
prolog_predicate( r(_,_,_)).
prolog_predicate( \+ _).

% ----------

start_clause( [ adds(move(Box, Pos1, Dist, Pos2), T, Adds),
                    p(Box, T)] /
                [Box:box, Dist:distance, Adds:list_of_observations,
                 T:time, Pos1:position, Pos2:position]).

start_clause( [ adds(move(Box, Pos1, Dist, Pos2), T, Adds),
                    \+ p(Box, T)] /
                [Box:box, Dist:distance, Adds:list_of_observations,
                 T:time, Pos1:position, Pos2:position]).
```

A.3.4 Učenje Koncepta obteženosti: učenje `dels`

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

backliteral( in_state(T, at(Obj, P)),
             [], [T:time, Obj:object, P:position], []).
backliteral( in_state(T, on(Obj1, Obj2)),
             [T:time, Obj1:object, Obj2:object], [],
```

```
            [Obj1 \== Obj2]).

in_state(Time, Observation) :-
  state(Time, Observations),
  member(Observation, Observations).

backliteral( add(Pos1, Dist, Pos2), [Pos1:position, Dist:distance],
                                    [Pos2: position], []).

add([X1,Y1], [DX,DY], [X2,Y2]):-
  X2 #= X1 + DX,
  Y2 #= Y1 + DY.

backliteral( different( Pos1, Pos2), [Pos1:position, Pos2:position],
            [], []).

different([X1,Y1],[X2,Y2]) :-
    X1 \== X2
  ;
    Y1 \== Y2.

p(Obj, Time):-
    r(Obj, [C,D,E], Time).

r(Obj, [], Time).
r(Obj, [O|Os], Time):-
    in_state(Time, on(O, Obj)),
    r(O, Os, Time).

% ----------

term( observation, at( Obj, Pos), [Obj:object, Pos:position]).
term( observation, on( Obj1, Obj2), [Obj1:object, Obj2:object]).

term( list_of_observations, [], []).
term( list_of_observations, [O|OL],
```

```
      [O:observation, OL:list_of_observations]).

% ----------

prolog_predicate( in_state(_, _)).
prolog_predicate( add(_, _, _)).
prolog_predicate( different(_, _)).
prolog_predicate( p(_,_)).
prolog_predicate( r(_,_,_)).
prolog_predicate( \+ _).

% ----------

start_clause( [ dels(move(Obj, Pos1, Dist, Pos2), T, Dels),
                    p(Obj, T)] /
                [Obj:object, Dist:distance,
                 Dels:list_of_observations, T:time,
                 Pos1:position, Pos2:position]).

start_clause( [ dels(move(Obj, Pos1, Dist, Pos2), T, Dels),
                    \+ p(Obj, T)] /
                [Obj:object, Dist:distance,
                 Dels:list_of_observations, T:time,
                 Pos1:position, Pos2:position]).
```

A.4 Učenje koncepta prôstosti

A.4.1 Učenje koncepta prôstosti: podatki

```
object(a).
object(b).
object(c).
```

```
object(d).
object(e).
object(f).
object(g).
object(h).
object(i).
object(j).
object(k).
object(l).
object(m).

state(1, [
  on(a,b), on(b,c), on(c,d), on_floor(d),
  on(e,f), on(f,g), on_floor(g),
  on(h,i), on(i,j), on(j,k), on(k,l), on(l,m), on_floor(m)]).

action( 1, pickup(k, h)).

state(2, [
  on(a,b), on(b,c), on(c,d), on_floor(d),
  on(e,f), on(f,g), on_floor(g),
  on(i,j), on(j,k), on(k,l), on(l,m), on_floor(m),
  taken(h)]).

action( 2, pickup(g, e)).

state(3, [
  on(a,b), on(b,c), on(c,d), on_floor(d),
  on(f,g), on_floor(g),
  on(i,j), on(j,k), on(k,l), on(l,m), on_floor(m),
  taken(h), taken(e)]).

action( 3, pickup(a, a)).

state(4, [
  on(b,c), on(c,d), on_floor(d),
```

```
  on(f,g), on_floor(g),
  on(i,j), on(j,k), on(k,l), on(l,m), on_floor(m),
  taken(h), taken(e),
  taken(a)]).

action( 4, pickup(j, i)).

state(5, [
  on(b,c), on(c,d), on_floor(d),
  on(f,g), on_floor(g),
  on(j,k), on(k,l), on(l,m), on_floor(m),
  taken(h), taken(e),
  taken(a), taken(i)]).

action( 5, pickup(b, b)).

state(6, [
  on(c,d), on_floor(d),
  on(f,g), on_floor(g),
  on(j,k), on(k,l), on(l,m), on_floor(m),
  taken(h), taken(e),
  taken(a), taken(i),
  taken(b)]).

action( 6, pickup(l, j)).

state(7, [
  on(c,d), on_floor(d),
  on(f,g), on_floor(g),
  on(k,l), on(l,m), on_floor(m),
  taken(h), taken(e),
  taken(a), taken(i),
  taken(b), taken(j)]).

action( 7, pickup(f, f)).
```

```
state(8, [
  on(c,d), on_floor(d),
  on_floor(g),
  on(k,l), on(l,m), on_floor(m),
  taken(h), taken(e),
  taken(a), taken(i),
  taken(b), taken(j),
  taken(f)]).

action( 8, pickup(k, k)).

state(9, [
  on(c,d), on_floor(d),
  on_floor(g),
  on(l,m), on_floor(m),
  taken(h), taken(e),
  taken(a), taken(i),
  taken(b), taken(j),
  taken(f), taken(k)]).
```

A.4.2 Učenje koncepta prôstosti: učenje pogoja

```
backliteral( in_state(T, on( Box1, Box2)), [T:time],
             [Box1:box, Box2:box], [Box1 \== Box2]).
backliteral( \+ in_state(T, on( Box1, Box2)), [T:time],
             [Box1:box, Box2:box], [Box1 \== Box2]).
backliteral( \+ in_state(T, on( Box1, Box2)), [T:time, Box1:box],
             [Box2:box], [Box1 \== Box2]).
backliteral( \+ in_state(T, on( Box1, Box2)), [T:time, Box2:box],
             [Box1:box], [Box1 \== Box2]).
backliteral( \+ in_state(T, on( Box1, Box2)),
             [T:time, Box1:box, Box2:box], [], [Box1 \== Box2]).
```

```
backliteral( in_state(T, on_floor(Box)),
             [T:time], [Box:box], []).
backliteral( \+ in_state(T, on_floor(Box)),
             [T:time], [Box:box], []).

in_state(Time, Observation) :-
  state(Time, Observations),
  member(Observation, Observations).

% ----------

% no term clauses
term( defaultterm, [], []).

% ----------

prolog_predicate( in_state(_, _)).
prolog_predicate( \+ _).

% ----------

start_clause( [ p(Box, T)] /  [Box:box, T:time]).
```

A.4.3 Učenje koncepta prôstosti: učenje `adds`

```
backliteral( in_state(T, on( Box1, Box2)), [T:time],
             [Box1:box, Box2:box], [Box1 \== Box2]).

backliteral( in_state(T, on_floor(Box)),
             [T:time], [Box:box], []).

in_state(Time, Observation) :-
  state(Time, Observations),
```

```
  member(Observation, Observations).

backliteral( adds(pickup(Box1, Box2), Time, Adds),
             [Box1:box, Box2:box, Time:time,
              Adds:list_of_observations],
             [], []).

p(Box, Time):-
    \+ in_state(Time, on(Box1, Box)).

% ----------

term( observation, on( Box1, Box2), [Box1:box, Box2:box]).
term( observation, on_floor(Box), [Box:box]).
term( observation, taken(Box), [Box:box]).

term( list_of_observations, [], []).
term( list_of_observations, [O], [O:observation]).

% ----------

prolog_predicate( in_state(_, _)).
prolog_predicate( \+ _).
prolog_predicate( p(_, _)).

% ----------

start_clause( [ adds(pickup(Box1, Box2), Time, Adds),
                    p(Box1, Time)] /
                [Box1:box, Box2:box, Adds:list_of_observations,
                 Time:time]).

start_clause( [ adds(pickup(Box1, Box2), Time, Adds),
                    \+ p(Box1, Time)] /
                [Box1:box, Box2:box, Adds:list_of_observations,
                 Time:time]).
```

A.4.4 Učenje koncepta prôstosti: učenje `dels`

```
backliteral( in_state(T, on( Box1, Box2)), [T:time],
             [Box1:box, Box2:box], [Box1 \== Box2]).

backliteral( in_state(T, on_floor(Box)), [T:time], [Box:box], []).

in_state(Time, Observation) :-
  state(Time, Observations),
  member(Observation, Observations).

backliteral( dels(pickup(Box1, Box2), Time, Dels),
             [Box1:box, Box2:box, Time:time,
              Dels:list_of_observations],
             [], []).

p(Box, Time):-
    \+ in_state(Time, on(Box1, Box)).

% ----------

term( observation, on( Box1, Box2), [Box1:box, Box2:box]).
term( observation, on_floor(Box), [Box:box]).
term( observation, taken(Box), [Box:box]).

term( list_of_observations, [], []).
term( list_of_observations, [O], [O:observation]).

% ----------

prolog_predicate( in_state(_, _)).
prolog_predicate( \+ _).
prolog_predicate( p(_, _)).
```

```
% ----------

start_clause( [ dels(pickup(Box1, Box2), Time, Dels),
                    p(Box1, Time)] /
                [Box1:box, Box2:box, Dels:list_of_observations,
                 Time:time]).

start_clause( [ dels(pickup(Box1, Box2), Time, Dels),
                    \+ p(Box1, Time)] /
                [Box1:box, Box2:box, Dels:list_of_observations,
                 Time:time]).
```

Dodatek B

Definicije testnih domen ILP

V tem dodatku podajamo definicijo vseh testnih domen ILP iz podrazdelka 6.2.2. Vse definicije so zapisane kot logični programi. Za vsako domeno podamo definicijo problema ILP za sisteme HYPER/CA, Progol in Aleph. Uporabljen jezik je v primeru sistema HYPER/CA SICStus prolog, v primeru Progola njegova interna implementacija logičnega programiranja in v primeru Alepha Yap prolog. Vsi materiali iz tega dodatka so skupaj s kodo potrebno, da se požene učenje, dostopni tudi na elektronskem naslovu `http://www.ailab.si/aljaz/aljaz_kosmerlj_phd_code.zip`.

B.1 Domena `issorted`

B.1.1 Domena `issorted` – HYPER/CA

```
% Static parameters - same for all domains
max_proof_length(10).
min_pos_covered(0.50).
clause_cost_modifier(0.05).
max_refined_hypotheses(1000000).

% ----------

% domain specific parametrs
max_clauses(2).
```

```
max_clause_length(3).

% ----------

% run specific parameters
complexity_coefficient(0.0001).

% ----------

backliteral( X @=< Y, [ X:item, Y:item ], []).
backliteral( issorted(L), [ L:list ], []).

term( list, [], []).
term( list, [X|L], [ X:item, L:list]).

prolog_predicate( X @=< Y ).

constraints(_).

start_clause( [ issorted(L) ] / [ L:list ] ).
```

B.1.2 Domena `issorted` – Aleph

```
% domain specific parametrs
:- set(i,3).
:- set(depth, 5).

% ----------

% run specific parameters
:- set(noise, 0).

% ----------

:- mode(1, issorted(+list)).
```

```
:- mode(1, +el @=< +el).
:- mode(1,((+list) = ([-el|-list]))).

:- determination(issorted/1,issorted/1).
:- determination(issorted/1, '@=<'/2).
:- determination(issorted/1, '='/2).
```

B.1.3 Domena issorted – Progol

```
% Static parameters - same for all domains
:- unset(cover)?
:- set(h,1000)?

% ----------

% domain specific parametrs
:- set(c,4)?
:- set(i,3)?

% ----------

% run specific parameters
% nodes always set to 1.6 * #training examples
%  as the Progol manual recommends
:- set(nodes, 16)?
:- set(noise, 0)?

% ----------

:- modeh(1, issorted(+list))?
:- modeb(1, issorted(+list))?
:- modeb(1, +el @=< +el)?
:- modeb(1, +list = [-el| -list])?

el(a).
```

```
el(b).
el(c).
el(d).
el(e).

list([]).
list([H|T]) :- el(H), list(T).
```

B.2 Domena member

B.2.1 Domena member – HYPER/CA

```
% Static parameters - same for all domains
max_proof_length(10).
min_pos_covered(0.50).
clause_cost_modifier(0.05).
max_refined_hypotheses(1000000).

% ----------

% domain specific parametrs
max_clauses(2).
max_clause_length(2).

% ----------

% run specific parameters
complexity_coefficient(0.0001).

% ----------

backliteral( member(X,L), [L:list], [X:item] ).

term( list, [], []).
```

```
term( list, [X|L], [ X:item, L:list]).

prolog_predicate( fail).

constraints(_).

start_clause( [ member(X,L) ] / [ X:item, L:list] ).
```

B.2.2 Domena member – Aleph

```
% domain specific parametrs
:- set(i,3).
:- set(depth, 5).

% ----------

% run specific parameters
:- set(noise, 0).

% ----------

:- mode(*, memb(+any, +list)).
:- mode(1,((+list) = ([-any|-list]))).

:- determination(memb/2,memb/2).
:- determination(memb/2,'='/2).
```

B.2.3 Domena member – Progol

```
% Static parameters - same for all domains
:- unset(cover)?
:- set(h,1000)?
```

```
% ----------

% domain specific parametrs
:- set(c,1)?
:- set(i,2)?

% ----------

% run specific parameters
% nodes always set to 1.6 * #training examples
%  as the Progol manual recommends
:- set(nodes, 16)?
:- set(noise, 0)?

% ----------

:- modeh(*, member(+el, +list))?
:- modeb(*, member(+el, +list))?
:- modeb(*, member(-el, +list))?
:- modeb(1, +list = [-el| -list])?

el(a).
el(b).
el(c).
el(d).
el(e).

list([]).
list([H|T]) :- el(H), list(T).
```

B.3 Domena oddeven

B.3.1 Domena oddeven – HYPER/CA

```
% Static parameters - same for all domains
max_proof_length(10).
min_pos_covered(0.50).
clause_cost_modifier(0.05).
max_refined_hypotheses(1000000).

% ----------

% domain specific parametrs
max_clauses(3).
max_clause_length(2).

% ----------

% run specific parameters
complexity_coefficient(0.0001).

% ----------

backliteral( oddeven( T, L), [ L:list], [T:type]).

term( list, [X|L], [ X:item, L:list]).
term( list, [], []).

term( type, odd, []).
term( type, even, []).

prolog_predicate( fail).

constraints(_).

start_clause([ oddeven( T, L) ] / [ T:type, L:list]).
```

B.3.2 Domena oddeven – Aleph

```
% domain specific parametrs
:- set(i,3).
:- set(depth, 8).
:- set(clauselength, 3).
:- set(nodes, 100).

% ----------

% run specific parameters
:- set(noise, 0).

% ----------

:- mode(1, oddeven(+type, +list)).
:- mode(1, (+type) = (#type)).
:- mode(1, +list = []).
:- mode(1, +list = [-el| -list]).

:- determination(oddeven/2, oddeven/2).
:- determination(oddeven/2, '='/2).
```

B.3.3 Domena oddeven – Progol

```
% Static parameters - same for all domains
:- unset(cover)?
:- set(h,1000)?

% ----------

% domain specific parametrs
:- set(c,4)?
```

```
:- set(i,5)?

% ----------

% run specific parameters
% nodes always set to 1.6 * #training examples
%  as the Progol manual recommends
:- set(nodes, 16)?
:- set(noise, 0)?

% ----------

:- modeh(100, oddeven(+type, +list))?
:- modeh(100, oddeven(-type, +list))?
:- modeb(100, oddeven(+type, +list))?
:- modeb(100, oddeven(-type, +list))?
:- modeb(1, +type = #type)?
:- modeb(1, +list = [-el| -list])?

el(a).
el(b).
el(c).

type(odd).
type(even).

list([]).
list([H|T]) :- el(H), list(T).
```

B.4 Domena concat

B.4.1 Domena concat – HYPER/CA

```
% Static parameters - same for all domains
```

```
max_proof_length(10).
min_pos_covered(0.50).
clause_cost_modifier(0.05).
max_refined_hypotheses(1000000).

% ----------

% domain specific parametrs
max_clauses(2).
max_clause_length(2).

% ----------

% run specific parameters
complexity_coefficient(0.0001).

% ----------


backliteral( concat(L1, L2, L3), [L1:list, L2:list, L3:list], [] ).

term( list, [], []).
term( list, [X|L], [ X:item, L:list]).

prolog_predicate( fail).

constraints(_).

start_clause( [ concat(L1, L2, L3) ] / [L1:list, L2:list, L3:list] ).
```

B.4.2 Domena concat – Aleph

```
% domain specific parametrs
:- set(i,2).
:- set(depth, 3).
```

```
% ----------

% run specific parameters
:- set(noise, 0).

% ----------

:- mode(1, concat(+list, +list, +list)).
:- mode(1, +list = [-el| -list]).

:- determination(concat/3, concat/3).
:- determination(concat/3, '='/2).
```

B.4.3 Domena concat – Progol

```
% Static parameters - same for all domains
:- unset(cover)?
:- set(h,1000)?

% ----------

% domain specific parametrs
:- set(c,4)?
:- set(i,2)?

% ----------

% run specific parameters
% nodes always set to 1.6 * #training examples
%  as the Progol manual recommends
:- set(nodes, 16)?
:- set(noise, 0)?

% ----------
```

```
:- modeh(1, concat(+list, +list, +list))?
:- modeb(1, concat(+list, +list, +list))?
:- modeb(1, +list = [-el| -list])?

el(a).
el(b).
el(c).
el(d).
el(e).

list([]).
list([H|T]) :- el(H), list(T).
```

Dodatek C

Tabele rezultatov testov sistemov ILP

Ta dodatek vsebuje tabele rezultatov sistemov HYPER/CA, Aleph in Progol na učnih nalogah opisanih v razdelku 6.2.2 na strani 80.

C.1 Rezultati v domeni `issorted` (neuravnoteženi učni nabor)

N	data noise	cc	runtime	CA	TP	FP	FN	TN	#literals	#vars
10	0	1	0,002	0,06	0,063	0,937	0,000	0,000	1,0	1,0
10	0	0,1	0,014	0,59	0,026	0,375	0,038	0,562	1,0	1,0
10	0	0,01	0,092	0,93	0,005	0,017	0,059	0,919	1,1	1,4
10	0	0,001	0,320	0,93	0,005	0,018	0,058	0,919	1,6	2,1
10	0	0,0001	0,609	0,92	0,006	0,020	0,057	0,916	1,8	2,4
10	0,2	1	0,011	0,06	0,064	0,936	0,000	0,000	1,0	1,0
10	0,2	0,1	0,025	0,55	0,035	0,425	0,029	0,511	1,0	1,3
10	0,2	0,01	0,114	0,63	0,029	0,331	0,035	0,605	1,0	1,4
10	0,2	0,001	0,389	0,81	0,017	0,145	0,046	0,792	1,6	2,0
10	0,2	0,0001	0,796	0,84	0,018	0,110	0,045	0,826	2,1	2,8
10	0,5	1	0,002	0,06	0,063	0,937	0,000	0,000	1,0	1,0
10	0,5	0,1	0,015	0,24	0,051	0,749	0,013	0,187	1,0	1,0
10	0,5	0,01	0,159	0,56	0,028	0,405	0,036	0,532	1,0	2,0
10	0,5	0,001	0,540	0,75	0,019	0,203	0,045	0,734	1,6	2,8
10	0,5	0,0001	1,194	0,74	0,021	0,220	0,042	0,717	1,6	2,9
50	0	1	0,056	0,06	0,061	0,939	0,000	0,000	1,0	1,0
50	0	0,1	0,484	0,15	0,055	0,845	0,006	0,094	1,0	1,1
50	0	0,01	1,752	0,84	0,023	0,119	0,039	0,820	1,9	2,9
50	0	0,001	3,903	0,91	0,009	0,035	0,053	0,904	2,3	2,9
50	0	0,0001	7,359	0,92	0,015	0,035	0,047	0,904	2,6	3,1
50	0,2	1	0,061	0,06	0,061	0,939	0,000	0,000	1,0	1,0
50	0,2	0,1	0,523	0,06	0,061	0,939	0,000	0,000	1,0	1,0
50	0,2	0,01	2,153	0,90	0,011	0,052	0,050	0,887	2,0	3,0
50	0,2	0,001	4,785	0,94	0,002	0,002	0,059	0,937	2,0	3,0
50	0,2	0,0001	9,017	0,95	0,008	0,001	0,053	0,938	2,5	3,1
50	0,5	1	0,063	0,06	0,061	0,939	0,000	0,000	1,0	1,0
50	0,5	0,1	0,521	0,06	0,061	0,939	0,000	0,000	1,0	1,0
50	0,5	0,01	2,618	0,67	0,028	0,293	0,033	0,646	1,8	2,6
50	0,5	0,001	6,359	0,84	0,017	0,119	0,044	0,821	2,2	3,5
50	0,5	0,0001	12,280	0,81	0,026	0,154	0,035	0,786	2,6	3,7
100	0	1	0,276	0,06	0,059	0,941	0,000	0,000	1,0	1,0
100	0	0,1	2,089	0,58	0,039	0,401	0,020	0,540	1,7	2,4
100	0	0,01	4,806	0,88	0,013	0,069	0,046	0,873	2,0	3,0
100	0	0,001	11,702	0,93	0,004	0,018	0,055	0,923	2,0	3,0
100	0	0,0001	15,879	1,00	0,059	0,000	0,000	0,941	4,0	4,0
100	0,2	1	0,276	0,06	0,059	0,941	0,000	0,000	1,0	1,0
100	0,2	0,1	2,173	0,06	0,059	0,941	0,000	0,000	1,0	1,0
100	0,2	0,01	6,360	0,72	0,029	0,248	0,030	0,693	2,1	3,1
100	0,2	0,001	15,088	0,82	0,019	0,144	0,040	0,797	2,1	3,4
100	0,2	0,0001	23,846	1,00	0,059	0,000	0,000	0,941	4,0	4,0
100	0,5	1	0,254	0,06	0,059	0,941	0,000	0,000	1,0	1,0
100	0,5	0,1	2,452	0,06	0,059	0,941	0,000	0,000	1,0	1,0
100	0,5	0,01	8,496	0,54	0,050	0,455	0,009	0,486	2,4	3,2
100	0,5	0,001	20,102	0,53	0,052	0,461	0,007	0,480	2,2	3,3
100	0,5	0,0001	35,939	0,75	0,053	0,241	0,007	0,700	3,4	4,2
200	0	1	0,998	0,06	0,057	0,943	0,000	0,000	1,0	1,0
200	0	0,1	6,697	0,75	0,025	0,217	0,032	0,726	2,5	3,3
200	0	0,01	15,112	0,84	0,019	0,116	0,037	0,827	2,1	3,4
200	0	0,001	34,676	0,87	0,017	0,090	0,040	0,854	2,2	3,5
200	0	0,0001	34,941	1,00	0,057	0,000	0,000	0,943	4,0	4,0
200	0,2	1	0,981	0,06	0,056	0,944	0,000	0,000	1,0	1,0
200	0,2	0,1	7,659	0,42	0,043	0,573	0,013	0,370	2,2	2,5
200	0,2	0,01	20,689	0,64	0,038	0,339	0,018	0,604	2,6	3,6
200	0,2	0,001	42,810	0,89	0,015	0,068	0,041	0,876	2,4	4,5
200	0,2	0,0001	63,413	1,00	0,056	0,000	0,000	0,944	4,0	4,0
200	0,5	1	0,843	0,06	0,056	0,944	0,000	0,000	1,0	1,0
200	0,5	0,1	8,051	0,06	0,053	0,938	0,003	0,006	1,0	2,6
200	0,5	0,01	24,710	0,46	0,055	0,536	0,001	0,408	2,3	3,3
200	0,5	0,001	54,366	0,48	0,049	0,513	0,007	0,431	2,2	3,9
200	0,5	0,0001	104,487	0,58	0,050	0,410	0,006	0,534	3,2	4,5

Tabela C.1: Rezultati HYPER/CA na neuravnoteženi domeni `issorted`.

N	data noise	noise	runtime	CA	TP	FP	FN	TN	#literals	#vars
10	0	0	0,004	0,89	0,006	0,054	0,057	0,883	3,9	0,5
10	0	20	0,005	0,89	0,006	0,054	0,057	0,883	3,9	0,5
10	0	50	0,004	0,85	0,006	0,094	0,057	0,843	3,3	0,1
10	0	100	0,006	0,85	0,006	0,094	0,057	0,843	3,3	0,1
10	0,2	0	0,006	0,92	0,001	0,016	0,063	0,920	3,9	0,5
10	0,2	20	0,005	0,91	0,004	0,033	0,059	0,903	3,9	0,9
10	0,2	50	0,004	0,84	0,009	0,111	0,054	0,826	3,1	0,5
10	0,2	100	0,004	0,84	0,009	0,111	0,054	0,826	3,1	0,5
10	0,5	0	0,007	0,84	0,013	0,107	0,051	0,829	5,0	1,0
10	0,5	20	0,007	0,80	0,013	0,147	0,051	0,789	4,2	0,6
10	0,5	50	0,005	0,72	0,019	0,241	0,044	0,696	3,7	0,7
10	0,5	100	0,006	0,72	0,019	0,241	0,044	0,696	3,7	0,7
50	0	0	0,026	0,90	0,043	0,080	0,018	0,859	17,3	5,8
50	0	20	0,027	0,84	0,044	0,139	0,018	0,800	16,0	5,7
50	0	50	0,028	0,84	0,044	0,139	0,018	0,800	16,0	5,7
50	0	100	0,029	0,84	0,044	0,139	0,018	0,800	16,0	5,7
50	0,2	0	0,044	0,95	0,017	0,008	0,044	0,931	18,4	2,9
50	0,2	20	0,043	0,83	0,034	0,146	0,027	0,793	17,7	4,0
50	0,2	50	0,041	0,80	0,034	0,175	0,027	0,764	16,7	3,6
50	0,2	100	0,044	0,80	0,034	0,175	0,027	0,764	16,7	3,6
50	0,5	0	0,056	0,94	0,008	0,004	0,053	0,935	21,3	1,3
50	0,5	20	0,044	0,80	0,033	0,167	0,028	0,772	17,9	3,5
50	0,5	50	0,048	0,76	0,033	0,207	0,028	0,732	17,1	3,1
50	0,5	100	0,050	0,76	0,033	0,207	0,028	0,732	17,1	3,1
100	0	0	0,071	0,99	0,057	0,009	0,002	0,932	29,1	16,0
100	0	20	0,058	0,97	0,059	0,033	0,000	0,908	28,5	17,2
100	0	50	0,060	0,97	0,059	0,033	0,000	0,908	28,5	17,2
100	0	100	0,063	0,97	0,059	0,033	0,000	0,908	28,5	17,2
100	0,2	0	0,160	0,96	0,031	0,015	0,028	0,926	35,5	8,7
100	0,2	20	0,160	0,85	0,054	0,148	0,005	0,793	32,6	11,4
100	0,2	50	0,166	0,81	0,054	0,191	0,005	0,750	29,9	9,9
100	0,2	100	0,172	0,81	0,054	0,191	0,005	0,750	29,9	9,9
100	0,5	0	0,200	0,93	0,014	0,029	0,045	0,912	42,3	4,8
100	0,5	20	0,223	0,71	0,042	0,272	0,017	0,668	37,4	6,3
100	0,5	50	0,240	0,71	0,042	0,272	0,017	0,668	37,4	6,3
100	0,5	100	0,251	0,71	0,042	0,272	0,017	0,668	37,4	6,3
200	0	0	0,161	1,00	0,057	0,000	0,000	0,943	37,9	22,4
200	0	20	0,161	0,99	0,057	0,011	0,000	0,933	37,2	21,9
200	0	50	0,165	0,99	0,057	0,011	0,000	0,933	37,2	21,9
200	0	100	0,169	0,99	0,057	0,011	0,000	0,933	37,2	21,9
200	0,2	0	0,693	0,96	0,040	0,022	0,016	0,922	57,2	12,2
200	0,2	20	0,492	0,88	0,056	0,122	0,000	0,822	47,6	14,2
200	0,2	50	0,532	0,88	0,056	0,122	0,000	0,822	47,6	14,2
200	0,2	100	0,545	0,88	0,056	0,122	0,000	0,822	47,6	14,2
200	0,5	0	1,141	0,89	0,031	0,085	0,024	0,858	81,5	11,8
200	0,5	20	0,956	0,81	0,054	0,188	0,002	0,756	65,6	11,0
200	0,5	50	0,993	0,77	0,054	0,230	0,002	0,714	61,7	10,0
200	0,5	100	1,016	0,77	0,054	0,230	0,002	0,714	61,7	10,0

Tabela C.2: Rezultati Aleph na neuravnoteženi domeni `issorted`.

N	data noise	noise	runtime	CA	TP	FP	FN	TN	#literals	#vars
10	0	0	0,004	0,86	0,013	0,088	0,051	0,849	3,1	0,4
10	0	20	0,002	0,79	0,025	0,176	0,038	0,761	2,6	0,8
10	0	50	0,002	0,79	0,025	0,176	0,038	0,761	2,6	0,8
10	0	100	0,002	0,79	0,025	0,176	0,038	0,761	2,6	0,8
10	0,2	0	0,006	0,94	0,000	0,000	0,064	0,936	3,7	0,0
10	0,2	20	0,006	0,90	0,006	0,044	0,057	0,892	3,4	0,2
10	0,2	50	0,003	0,81	0,013	0,138	0,051	0,799	2,8	0,3
10	0,2	100	0,003	0,81	0,013	0,138	0,051	0,799	2,8	0,3
10	0,5	0	0,007	0,90	0,006	0,044	0,057	0,893	4,4	0,2
10	0,5	20	0,006	0,81	0,013	0,138	0,051	0,799	3,8	0,3
10	0,5	50	0,005	0,73	0,019	0,231	0,044	0,705	3,2	0,4
10	0,5	100	0,005	0,73	0,019	0,231	0,044	0,705	3,2	0,4
50	0	0	0,194	0,89	0,052	0,096	0,009	0,843	9,4	2,0
50	0	20	0,174	0,85	0,055	0,146	0,006	0,792	9,7	2,6
50	0	50	0,133	0,72	0,058	0,275	0,004	0,664	7,5	2,7
50	0	100	0,134	0,72	0,058	0,275	0,004	0,664	7,5	2,7
50	0,2	0	0,493	0,95	0,007	0,001	0,054	0,938	17,2	0,4
50	0,2	20	0,269	0,91	0,053	0,085	0,008	0,854	11,6	1,8
50	0,2	50	0,161	0,58	0,053	0,415	0,008	0,524	8,0	1,9
50	0,2	100	0,161	0,58	0,053	0,415	0,008	0,524	8,0	1,9
50	0,5	0	0,594	0,94	0,003	0,007	0,058	0,932	20,6	0,2
50	0,5	20	0,387	0,60	0,031	0,370	0,030	0,569	13,6	1,3
50	0,5	50	0,251	0,52	0,045	0,470	0,016	0,469	9,1	1,6
50	0,5	100	0,252	0,52	0,045	0,470	0,016	0,469	9,1	1,6
100	0	0	0,355	1,00	0,059	0,002	0,000	0,939	8,7	2,3
100	0	20	0,342	0,98	0,059	0,015	0,000	0,926	8,1	2,1
100	0	50	0,281	0,91	0,059	0,089	0,000	0,852	7,7	2,3
100	0	100	0,279	0,91	0,059	0,089	0,000	0,852	7,7	2,3
100	0,2	0	2,144	0,96	0,025	0,003	0,034	0,939	25,9	1,7
100	0,2	20	1,030	0,99	0,059	0,009	0,000	0,933	15,6	2,0
100	0,2	50	0,762	0,70	0,059	0,296	0,000	0,645	12,9	2,4
100	0,2	100	0,761	0,70	0,059	0,296	0,000	0,645	12,9	2,4
100	0,5	0	3,886	0,95	0,011	0,003	0,049	0,937	38,5	0,7
100	0,5	20	2,930	0,96	0,042	0,016	0,017	0,924	32,1	2,2
100	0,5	50	1,082	0,44	0,059	0,564	0,000	0,377	13,2	2,5
100	0,5	100	1,080	0,44	0,059	0,564	0,000	0,377	13,2	2,5
200	0	0	0,717	1,00	0,057	0,000	0,000	0,943	8,0	2,0
200	0	20	0,718	1,00	0,057	0,000	0,000	0,943	8,0	2,0
200	0	50	0,719	1,00	0,057	0,000	0,000	0,943	8,0	2,0
200	0	100	0,719	1,00	0,057	0,000	0,000	0,943	8,0	2,0
200	0,2	0	15,342	0,97	0,028	0,004	0,028	0,940	43,1	2,9
200	0,2	20	7,248	0,99	0,056	0,011	0,000	0,933	25,6	2,3
200	0,2	50	6,107	0,86	0,056	0,137	0,000	0,807	22,5	2,5
200	0,2	100	6,131	0,86	0,056	0,137	0,000	0,807	22,5	2,5
200	0,5	0	31,257	0,95	0,009	0,004	0,047	0,940	76,5	1,5
200	0,5	20	24,622	0,96	0,039	0,022	0,017	0,922	61,3	2,6
200	0,5	50	7,998	0,49	0,056	0,514	0,000	0,430	25,6	3,1
200	0,5	100	8,018	0,49	0,056	0,514	0,000	0,430	25,6	3,1

Tabela C.3: Rezultati Progol na neuravnoteženi domeni `issorted`.

C.2 Rezultati v domeni `issorted` (uravnoteženi učni nabor)

N	data noise	cc	runtime	CA	TP	FP	FN	TN	#literals	#vars
10	0	1	0,005	0,06	0,063	0,937	0,000	0,000	1,0	1,0
10	0	0,1	0,019	0,06	0,063	0,937	0,000	0,000	1,0	1,0
10	0	0,01	0,190	0,15	0,056	0,841	0,007	0,096	1,0	1,5
10	0	0,001	0,669	0,56	0,047	0,425	0,016	0,512	1,9	3,3
10	0	0,0001	1,404	0,54	0,050	0,449	0,014	0,488	2,3	3,6
10	0,2	1	0,011	0,06	0,063	0,937	0,000	0,000	1,0	1,0
10	0,2	0,1	0,019	0,21	0,057	0,783	0,006	0,154	1,0	1,2
10	0,2	0,01	0,167	0,40	0,039	0,577	0,024	0,360	1,0	2,8
10	0,2	0,001	0,668	0,54	0,043	0,436	0,020	0,501	2,0	3,6
10	0,2	0,0001	1,432	0,54	0,043	0,436	0,020	0,501	2,2	3,8
10	0,5	1	0,008	0,06	0,063	0,937	0,000	0,000	1,0	1,0
10	0,5	0,1	0,023	0,21	0,057	0,783	0,006	0,154	1,0	1,2
10	0,5	0,01	0,178	0,38	0,045	0,598	0,018	0,339	1,0	2,1
10	0,5	0,001	0,638	0,69	0,030	0,274	0,034	0,663	1,7	3,4
10	0,5	0,0001	1,505	0,70	0,032	0,271	0,031	0,666	2,1	4,1
50	0	1	0,069	0,06	0,059	0,941	0,000	0,000	1,0	1,0
50	0	0,1	0,465	0,06	0,059	0,941	0,000	0,000	1,0	1,0
50	0	0,01	2,519	0,63	0,042	0,350	0,017	0,592	2,2	3,0
50	0	0,001	6,569	0,53	0,051	0,464	0,007	0,478	2,1	3,0
50	0	0,0001	9,517	1,00	0,059	0,000	0,000	0,941	4,0	4,0
50	0,2	1	0,052	0,06	0,059	0,941	0,000	0,000	1,0	1,0
50	0,2	0,1	0,498	0,06	0,059	0,941	0,000	0,000	1,0	1,0
50	0,2	0,01	2,697	0,41	0,050	0,582	0,008	0,360	1,9	2,4
50	0,2	0,001	7,125	0,46	0,054	0,541	0,005	0,400	2,0	2,8
50	0,2	0,0001	12,687	0,78	0,055	0,221	0,004	0,721	3,4	3,7
50	0,5	1	0,061	0,06	0,059	0,941	0,000	0,000	1,0	1,0
50	0,5	0,1	0,492	0,06	0,059	0,941	0,000	0,000	1,0	1,3
50	0,5	0,01	2,774	0,10	0,057	0,898	0,001	0,043	1,2	2,1
50	0,5	0,001	7,809	0,42	0,058	0,578	0,001	0,364	2,0	2,9
50	0,5	0,0001	15,965	0,58	0,053	0,420	0,005	0,522	2,7	3,4
100	0	1	0,144	0,05	0,053	0,947	0,000	0,000	1,0	1,0
100	0	0,1	2,020	0,05	0,053	0,947	0,000	0,000	1,0	1,0
100	0	0,01	7,508	0,49	0,049	0,504	0,004	0,443	2,1	3,0
100	0	0,001	17,878	0,46	0,053	0,544	0,000	0,403	2,0	3,0
100	0	0,0001	19,324	1,00	0,053	0,000	0,000	0,947	4,0	4,0
100	0,2	1	0,140	0,05	0,053	0,947	0,000	0,000	1,0	1,0
100	0,2	0,1	1,921	0,05	0,053	0,947	0,000	0,000	1,0	1,0
100	0,2	0,01	7,622	0,46	0,052	0,541	0,001	0,406	2,1	3,1
100	0,2	0,001	18,146	0,46	0,053	0,544	0,000	0,403	2,0	3,0
100	0,2	0,0001	31,704	0,73	0,053	0,272	0,000	0,675	3,1	3,7
100	0,5	1	0,144	0,05	0,053	0,947	0,000	0,000	1,0	1,0
100	0,5	0,1	2,002	0,05	0,053	0,947	0,000	0,000	1,0	1,0
100	0,5	0,01	8,028	0,47	0,052	0,533	0,002	0,415	2,4	3,4
100	0,5	0,001	19,995	0,48	0,052	0,518	0,002	0,429	2,1	3,3
100	0,5	0,0001	40,654	0,50	0,049	0,493	0,004	0,454	2,6	3,8
200	0	1	0,830	0,04	0,041	0,959	0,000	0,000	1,0	1,0
200	0	0,1	6,582	0,04	0,041	0,959	0,000	0,000	1,0	1,0
200	0	0,01	20,669	0,45	0,041	0,549	0,000	0,410	2,1	3,1
200	0	0,001	47,754	0,45	0,041	0,552	0,000	0,407	2,0	3,0
200	0	0,0001	41,942	1,00	0,041	0,000	0,000	0,959	4,0	4,0
200	0,2	1	0,827	0,04	0,041	0,959	0,000	0,000	1,0	1,0
200	0,2	0,1	6,764	0,08	0,041	0,915	0,000	0,044	1,2	1,3
200	0,2	0,01	21,901	0,46	0,041	0,546	0,000	0,413	2,2	3,2
200	0,2	0,001	50,472	0,46	0,041	0,546	0,000	0,413	2,2	3,2
200	0,2	0,0001	98,433	0,53	0,040	0,474	0,001	0,485	3,0	3,9
200	0,5	1	0,847	0,04	0,041	0,959	0,000	0,000	1,0	1,0
200	0,5	0,1	6,749	0,04	0,041	0,959	0,000	0,000	1,0	1,3
200	0,5	0,01	24,140	0,46	0,041	0,541	0,000	0,418	2,4	3,4
200	0,5	0,001	53,923	0,45	0,041	0,550	0,000	0,409	2,1	3,1
200	0,5	0,0001	105,356	0,51	0,041	0,495	0,000	0,464	3,0	3,8

Tabela C.4: Rezultati HYPER/CA na uravnoteženi domeni `issorted`.

N	data noise	noise	runtime	CA	TP	FP	FN	TN	#literals	#vars
10	0	0	0,007	0,79	0,020	0,168	0,043	0,769	5,5	2,3
10	0	20	0,006	0,65	0,039	0,329	0,024	0,608	5,4	3,8
10	0	50	0,006	0,65	0,039	0,329	0,024	0,608	5,4	3,8
10	0	100	0,006	0,65	0,039	0,329	0,024	0,608	5,4	3,8
10	0,2	0	0,008	0,83	0,017	0,121	0,046	0,816	5,5	2,2
10	0,2	20	0,007	0,69	0,034	0,285	0,029	0,652	4,9	2,9
10	0,2	50	0,004	0,48	0,041	0,499	0,022	0,438	3,1	1,8
10	0,2	100	0,004	0,48	0,041	0,499	0,022	0,438	3,1	1,8
10	0,5	0	0,007	0,90	0,009	0,050	0,054	0,887	5,0	1,6
10	0,5	20	0,007	0,82	0,012	0,127	0,051	0,810	4,8	1,6
10	0,5	50	0,005	0,66	0,022	0,298	0,041	0,639	3,3	0,9
10	0,5	100	0,007	0,66	0,022	0,298	0,041	0,639	3,3	0,9
50	0	0	0,046	0,91	0,047	0,075	0,012	0,866	25,2	11,6
50	0	20	0,020	0,64	0,057	0,361	0,001	0,580	14,8	9,0
50	0	50	0,018	0,58	0,058	0,421	0,001	0,521	11,9	7,1
50	0	100	0,018	0,58	0,058	0,421	0,001	0,521	11,9	7,1
50	0,2	0	0,067	0,88	0,023	0,087	0,036	0,854	26,9	6,9
50	0,2	20	0,031	0,65	0,054	0,343	0,005	0,598	18,0	10,0
50	0,2	50	0,007	0,31	0,057	0,687	0,002	0,254	5,3	3,0
50	0,2	100	0,007	0,31	0,057	0,687	0,002	0,254	5,3	3,0
50	0,5	0	0,079	0,93	0,008	0,021	0,051	0,920	26,8	1,8
50	0,5	20	0,057	0,70	0,035	0,276	0,024	0,665	20,5	6,7
50	0,5	50	0,013	0,26	0,055	0,736	0,003	0,205	5,8	3,1
50	0,5	100	0,012	0,26	0,055	0,736	0,003	0,205	5,8	3,1
100	0	0	0,174	0,94	0,051	0,054	0,002	0,893	48,4	24,0
100	0	20	0,076	0,80	0,053	0,202	0,000	0,745	29,2	18,4
100	0	50	0,039	0,58	0,053	0,423	0,000	0,524	15,0	8,1
100	0	100	0,038	0,58	0,053	0,423	0,000	0,524	15,0	8,1
100	0,2	0	0,290	0,90	0,029	0,075	0,024	0,873	52,5	9,6
100	0,2	20	0,182	0,51	0,051	0,481	0,002	0,466	36,7	15,2
100	0,2	50	0,025	0,29	0,053	0,711	0,000	0,236	9,4	4,8
100	0,2	100	0,026	0,29	0,053	0,711	0,000	0,236	9,4	4,8
100	0,5	0	0,216	0,94	0,015	0,022	0,038	0,925	52,6	6,2
100	0,5	20	0,171	0,65	0,048	0,344	0,005	0,603	34,0	9,5
100	0,5	50	0,046	0,34	0,052	0,664	0,001	0,284	11,8	5,0
100	0,5	100	0,046	0,34	0,052	0,664	0,001	0,284	11,8	5,0
200	0	0	2,701	0,99	0,040	0,007	0,001	0,952	84,4	53,6
200	0	20	0,280	0,78	0,041	0,220	0,000	0,740	35,7	17,7
200	0	50	0,194	0,66	0,041	0,339	0,000	0,621	25,1	11,3
200	0	100	0,195	0,66	0,041	0,339	0,000	0,621	25,1	11,3
200	0,2	0	4,966	0,67	0,032	0,318	0,009	0,641	103,7	17,5
200	0,2	20	0,456	0,65	0,040	0,350	0,001	0,609	43,4	13,6
200	0,2	50	0,113	0,36	0,041	0,641	0,000	0,318	14,5	5,6
200	0,2	100	0,114	0,36	0,041	0,641	0,000	0,318	14,5	5,6
200	0,5	0	1,430	0,81	0,017	0,163	0,024	0,796	106,1	8,7
200	0,5	20	0,743	0,50	0,040	0,501	0,001	0,458	54,4	12,0
200	0,5	50	0,157	0,24	0,041	0,760	0,000	0,199	11,8	3,0
200	0,5	100	0,155	0,24	0,041	0,760	0,000	0,199	11,8	3,0

Tabela C.5: Rezultati Aleph na uravnoteženi domeni `issorted`.

N	data noise	noise	runtime	CA	TP	FP	FN	TN	#literals	#vars
10	0	0	0,007	0,79	0,025	0,176	0,038	0,761	3,8	0,8
10	0	20	0,004	0,71	0,038	0,264	0,025	0,673	3,2	1,2
10	0	50	0,003	0,67	0,044	0,308	0,019	0,629	2,9	1,4
10	0	100	0,003	0,67	0,044	0,308	0,019	0,629	2,9	1,4
10	0,2	0	0,009	0,89	0,009	0,061	0,054	0,876	4,6	0,4
10	0,2	20	0,009	0,83	0,019	0,132	0,044	0,805	4,1	0,6
10	0,2	50	0,004	0,52	0,044	0,457	0,019	0,480	2,3	1,1
10	0,2	100	0,003	0,52	0,044	0,457	0,019	0,480	2,3	1,1
10	0,5	0	0,006	0,93	0,003	0,017	0,060	0,920	4,4	0,2
10	0,5	20	0,008	0,93	0,003	0,017	0,060	0,920	4,4	0,2
10	0,5	50	0,008	0,68	0,019	0,281	0,044	0,656	3,1	0,3
10	0,5	100	0,006	0,68	0,019	0,281	0,044	0,656	3,1	0,3
50	0	0	0,271	0,97	0,051	0,018	0,008	0,923	11,2	2,6
50	0	20	0,130	0,81	0,057	0,187	0,001	0,754	8,9	3,1
50	0	50	0,040	0,62	0,059	0,384	0,000	0,558	3,1	2,2
50	0	100	0,038	0,62	0,059	0,384	0,000	0,558	3,1	2,2
50	0,2	0	0,568	0,92	0,024	0,042	0,035	0,900	20,8	1,7
50	0,2	20	0,262	0,75	0,048	0,242	0,011	0,699	11,4	2,7
50	0,2	50	0,106	0,58	0,058	0,424	0,001	0,517	6,7	2,5
50	0,2	100	0,109	0,58	0,058	0,424	0,001	0,517	6,7	2,5
50	0,5	0	0,697	0,90	0,009	0,046	0,050	0,895	25,3	1,0
50	0,5	20	0,376	0,68	0,043	0,305	0,015	0,636	15,5	2,6
50	0,5	50	0,153	0,59	0,058	0,414	0,001	0,528	7,0	2,5
50	0,5	100	0,151	0,59	0,058	0,414	0,001	0,528	7,0	2,5
100	0	0	0,366	1,00	0,053	0,000	0,000	0,947	8,2	2,0
100	0	20	0,278	0,92	0,053	0,076	0,000	0,871	9,2	2,8
100	0	50	0,226	0,85	0,053	0,150	0,000	0,797	7,9	2,7
100	0	100	0,230	0,85	0,053	0,150	0,000	0,797	7,9	2,7
100	0,2	0	4,151	0,96	0,018	0,003	0,035	0,944	42,6	2,6
100	0,2	20	0,974	0,88	0,049	0,112	0,004	0,835	15,6	2,8
100	0,2	50	0,437	0,67	0,053	0,333	0,001	0,614	9,7	2,7
100	0,2	100	0,438	0,67	0,053	0,333	0,001	0,614	9,7	2,7
100	0,5	0	4,585	0,95	0,010	0,007	0,043	0,940	47,6	1,6
100	0,5	20	2,186	0,90	0,040	0,090	0,013	0,857	27,1	2,3
100	0,5	50	1,021	0,67	0,050	0,327	0,003	0,620	15,9	2,5
100	0,5	100	1,023	0,67	0,050	0,327	0,003	0,620	15,9	2,5
200	0	0	0,705	1,00	0,041	0,000	0,000	0,959	8,0	2,0
200	0	20	0,701	1,00	0,041	0,000	0,000	0,959	8,0	2,0
200	0	50	0,396	0,82	0,041	0,181	0,000	0,778	5,6	2,0
200	0	100	0,401	0,82	0,041	0,181	0,000	0,778	5,6	2,0
200	0,2	0	35,475	0,97	0,016	0,005	0,024	0,954	81,9	2,9
200	0,2	20	7,621	0,95	0,037	0,049	0,003	0,910	29,8	3,3
200	0,2	50	3,411	0,79	0,040	0,210	0,001	0,749	16,4	3,1
200	0,2	100	3,400	0,79	0,040	0,210	0,001	0,749	16,4	3,1
200	0,5	0	44,276	0,96	0,005	0,001	0,036	0,959	99,7	1,2
200	0,5	20	13,369	0,97	0,037	0,024	0,004	0,935	43,8	2,6
200	0,5	50	5,961	0,68	0,041	0,316	0,000	0,643	24,1	2,5
200	0,5	100	6,048	0,68	0,041	0,316	0,000	0,643	24,1	2,5

Tabela C.6: Rezultati Progol na uravnoteženi domeni `issorted`.

C.3 Rezultati v domeni `member`

N	data noise	cc	runtime	CA	TP	FP	FN	TN	#literals	#vars
10	0	1	0,011	0,62	0,516	0,244	0,135	0,105	1,0	2,0
10	0	0,1	0,047	0,60	0,425	0,174	0,226	0,175	1,0	2,5
10	0	0,01	0,175	0,62	0,407	0,141	0,244	0,208	1,8	3,7
10	0	0,001	0,256	0,75	0,529	0,131	0,122	0,218	2,6	4,9
10	0	0,0001	0,270	0,75	0,529	0,131	0,122	0,218	2,6	4,9
10	0,2	1	0,014	0,65	0,651	0,349	0,000	0,000	1,0	2,0
10	0,2	0,1	0,063	0,61	0,470	0,209	0,181	0,140	1,0	2,6
10	0,2	0,01	0,259	0,62	0,380	0,114	0,271	0,235	1,9	3,9
10	0,2	0,001	0,526	0,66	0,470	0,166	0,180	0,183	2,3	4,6
10	0,2	0,0001	0,768	0,70	0,486	0,140	0,165	0,209	2,4	5,1
10	0,5	1	0,011	0,65	0,651	0,349	0,000	0,000	1,0	2,0
10	0,5	0,1	0,079	0,62	0,586	0,314	0,065	0,035	1,0	2,8
10	0,5	0,01	0,279	0,57	0,429	0,207	0,222	0,142	1,3	3,7
10	0,5	0,001	0,629	0,58	0,461	0,227	0,189	0,122	2,0	4,7
10	0,5	0,0001	1,042	0,59	0,439	0,200	0,212	0,149	2,3	5,5
50	0	1	0,172	0,63	0,561	0,279	0,090	0,070	1,0	2,0
50	0	0,1	0,835	0,63	0,425	0,148	0,226	0,201	1,6	3,8
50	0	0,01	1,312	0,73	0,593	0,210	0,058	0,139	2,8	5,0
50	0	0,001	1,507	0,97	0,651	0,026	0,000	0,323	3,0	5,9
50	0	0,0001	1,496	0,97	0,651	0,026	0,000	0,323	3,0	5,9
50	0,2	1	0,178	0,63	0,561	0,279	0,090	0,070	1,0	2,0
50	0,2	0,1	0,983	0,64	0,559	0,268	0,091	0,081	1,2	4,3
50	0,2	0,01	1,911	0,73	0,564	0,184	0,087	0,165	2,7	5,0
50	0,2	0,001	2,927	0,88	0,584	0,053	0,067	0,296	2,8	5,7
50	0,2	0,0001	3,859	0,93	0,604	0,026	0,046	0,323	2,9	6,0
50	0,5	1	0,201	0,64	0,606	0,314	0,045	0,035	1,0	2,0
50	0,5	0,1	1,153	0,64	0,603	0,309	0,048	0,040	1,0	4,5
50	0,5	0,01	2,856	0,71	0,576	0,218	0,074	0,131	2,5	4,7
50	0,5	0,001	5,190	0,85	0,534	0,034	0,117	0,315	2,5	5,8
50	0,5	0,0001	7,755	0,90	0,554	0,000	0,097	0,349	2,8	6,2
100	0	1	0,697	0,64	0,606	0,314	0,045	0,035	1,0	2,3
100	0	0,1	2,104	0,72	0,606	0,237	0,045	0,112	2,8	4,7
100	0	0,01	3,292	0,74	0,651	0,263	0,000	0,085	3,0	5,0
100	0	0,001	3,203	1,00	0,651	0,000	0,000	0,349	3,0	6,0
100	0	0,0001	3,136	1,00	0,651	0,000	0,000	0,349	3,0	6,0
100	0,2	1	0,700	0,65	0,651	0,348	0,000	0,000	1,0	2,2
100	0,2	0,1	2,559	0,74	0,651	0,263	0,000	0,085	3,0	5,0
100	0,2	0,01	4,672	0,74	0,651	0,263	0,000	0,085	3,0	5,0
100	0,2	0,001	6,997	1,00	0,651	0,000	0,000	0,349	3,0	6,0
100	0,2	0,0001	9,729	1,00	0,651	0,000	0,000	0,349	3,0	6,0
100	0,5	1	0,775	0,65	0,651	0,349	0,000	0,000	1,0	2,0
100	0,5	0,1	3,633	0,68	0,650	0,321	0,001	0,027	1,6	4,5
100	0,5	0,01	7,656	0,70	0,483	0,136	0,169	0,212	2,5	5,0
100	0,5	0,001	12,931	0,96	0,629	0,023	0,023	0,326	3,0	6,0
100	0,5	0,0001	19,736	0,96	0,629	0,023	0,023	0,326	3,0	6,0
200	0	1	2,619	0,65	0,652	0,347	0,000	0,000	1,0	3,7
200	0	0,1	4,922	0,74	0,652	0,264	0,000	0,084	3,0	5,0
200	0	0,01	8,424	0,87	0,652	0,132	0,000	0,216	3,0	5,5
200	0	0,001	6,719	1,00	0,652	0,000	0,000	0,348	3,0	6,0
200	0	0,0001	6,739	1,00	0,652	0,000	0,000	0,348	3,0	6,0
200	0,2	1	2,644	0,65	0,652	0,347	0,000	0,000	1,0	3,5
200	0,2	0,1	6,608	0,74	0,652	0,264	0,000	0,084	3,0	5,0
200	0,2	0,01	13,163	0,82	0,652	0,185	0,000	0,163	3,0	5,3
200	0,2	0,001	17,781	1,00	0,652	0,000	0,000	0,348	3,0	6,0
200	0,2	0,0001	27,471	1,00	0,652	0,000	0,000	0,348	3,0	6,0
200	0,5	1	2,806	0,65	0,652	0,348	0,000	0,000	1,0	2,3
200	0,5	0,1	9,429	0,73	0,652	0,272	0,000	0,076	2,8	5,0
200	0,5	0,01	19,516	0,73	0,593	0,211	0,058	0,137	2,8	5,0
200	0,5	0,001	32,753	1,00	0,652	0,000	0,000	0,348	3,0	6,0
200	0,5	0,0001	48,118	1,00	0,652	0,000	0,000	0,348	3,0	6,0

Tabela C.7: Rezultati HYPER/CA na domeni `member`.

N	data noise	noise	runtime	CA	TP	FP	FN	TN	#literals	#vars
10	0	0	0,003	0,64	0,321	0,033	0,331	0,315	4,7	4,7
10	0	20	0,003	0,64	0,431	0,137	0,221	0,211	4,3	5,5
10	0	50	0,002	0,65	0,476	0,173	0,176	0,175	2,9	4,3
10	0	100	0,001	0,65	0,476	0,173	0,176	0,175	2,9	4,3
10	0,2	0	0,003	0,61	0,127	0,001	0,388	0,484	5,0	4,1
10	0,2	20	0,003	0,59	0,136	0,030	0,380	0,455	4,6	3,7
10	0,2	50	0,003	0,57	0,181	0,096	0,334	0,389	3,8	3,0
10	0,2	100	0,004	0,57	0,181	0,096	0,334	0,389	3,8	3,0
10	0,5	0	0,005	0,47	0,154	0,037	0,497	0,311	5,4	2,5
10	0,5	20	0,005	0,47	0,166	0,045	0,486	0,303	4,9	2,1
10	0,5	50	0,002	0,52	0,346	0,174	0,306	0,174	2,7	1,3
10	0,5	100	0,002	0,52	0,346	0,174	0,306	0,174	2,7	1,3
50	0	0	0,007	1,00	0,515	0,000	0,000	0,485	9,4	13,5
50	0	20	0,009	0,99	0,515	0,007	0,000	0,478	9,7	14,2
50	0	50	0,008	0,86	0,515	0,142	0,000	0,343	6,8	10,2
50	0	100	0,008	0,86	0,515	0,142	0,000	0,343	6,8	10,2
50	0,2	0	0,030	0,84	0,513	0,025	0,139	0,324	20,3	17,2
50	0,2	20	0,024	0,82	0,593	0,118	0,059	0,230	11,6	13,5
50	0,2	50	0,008	0,74	0,652	0,256	0,000	0,092	6,2	8,5
50	0,2	100	0,008	0,74	0,652	0,256	0,000	0,092	6,2	8,5
50	0,5	0	0,032	0,59	0,121	0,019	0,394	0,466	26,2	11,0
50	0,5	20	0,038	0,89	0,449	0,044	0,066	0,442	11,5	8,7
50	0,5	50	0,025	0,75	0,453	0,188	0,063	0,297	8,2	6,2
50	0,5	100	0,026	0,75	0,453	0,188	0,063	0,297	8,2	6,2
100	0	0	0,011	1,00	0,652	0,000	0,000	0,348	7,9	11,5
100	0	20	0,016	1,00	0,652	0,000	0,000	0,348	7,9	11,5
100	0	50	0,014	0,90	0,652	0,104	0,000	0,244	5,4	8,1
100	0	100	0,016	0,90	0,652	0,104	0,000	0,244	5,4	8,1
100	0,2	0	0,132	0,72	0,248	0,009	0,267	0,476	44,1	19,2
100	0,2	20	0,231	0,91	0,509	0,084	0,006	0,401	14,6	13,8
100	0,2	50	0,242	0,82	0,509	0,177	0,006	0,308	12,6	12,3
100	0,2	100	0,244	0,82	0,509	0,177	0,006	0,308	12,6	12,3
100	0,5	0	0,093	0,41	0,071	0,007	0,581	0,341	48,4	8,8
100	0,5	20	0,366	0,81	0,522	0,057	0,130	0,291	24,1	15,5
100	0,5	50	0,297	0,74	0,536	0,139	0,116	0,209	17,2	11,4
100	0,5	100	0,304	0,74	0,536	0,139	0,116	0,209	17,2	11,4
200	0	0	0,011	1,00	0,515	0,000	0,000	0,485	8,4	12,0
200	0	20	0,015	0,99	0,515	0,006	0,000	0,479	9,2	13,5
200	0	50	0,010	0,71	0,515	0,293	0,000	0,192	4,6	7,2
200	0	100	0,011	0,71	0,515	0,293	0,000	0,192	4,6	7,2
200	0,2	0	16,816	0,64	0,332	0,037	0,320	0,311	99,9	41,1
200	0,2	20	0,575	0,98	0,652	0,022	0,000	0,326	18,8	14,2
200	0,2	50	0,325	0,85	0,652	0,153	0,000	0,195	11,8	10,1
200	0,2	100	0,328	0,85	0,652	0,153	0,000	0,195	11,8	10,1
200	0,5	0	80,288	0,76	0,423	0,149	0,092	0,336	108,6	29,5
200	0,5	20	2,129	0,81	0,508	0,182	0,007	0,303	35,0	10,8
200	0,5	50	0,676	0,61	0,515	0,394	0,000	0,090	9,2	3,5
200	0,5	100	0,677	0,61	0,515	0,394	0,000	0,090	9,2	3,5

Tabela C.8: Rezultati Aleph na domeni `member`.

N	data noise	noise	runtime	CA	TP	FP	FN	TN	#literals	#vars
10	0	0	0,000	0,47	0,120	0,000	0,531	0,349	3,5	1,2
10	0	20	0,000	0,47	0,120	0,000	0,531	0,349	3,5	1,2
10	0	50	0,000	0,49	0,210	0,070	0,441	0,279	2,6	1,2
10	0	100	0,000	0,49	0,210	0,070	0,441	0,279	2,6	1,2
10	0,2	0	0,000	0,41	0,060	0,000	0,591	0,349	3,9	0,6
10	0,2	20	0,000	0,41	0,060	0,000	0,591	0,349	3,9	0,6
10	0,2	50	0,000	0,45	0,170	0,070	0,481	0,279	3,0	0,8
10	0,2	100	0,000	0,45	0,170	0,070	0,481	0,279	3,0	0,8
10	0,5	0	0,002	0,44	0,125	0,035	0,526	0,314	4,3	1,5
10	0,5	20	0,001	0,44	0,125	0,035	0,526	0,314	4,3	1,3
10	0,5	50	0,002	0,52	0,345	0,175	0,305	0,174	2,7	1,5
10	0,5	100	0,002	0,52	0,345	0,175	0,305	0,174	2,7	1,5
50	0	0	0,001	1,00	0,651	0,000	0,000	0,349	3,0	5,0
50	0	20	0,001	1,00	0,651	0,000	0,000	0,349	3,0	5,0
50	0	50	0,001	1,00	0,651	0,000	0,000	0,349	3,0	5,0
50	0	100	0,000	1,00	0,651	0,000	0,000	0,349	3,0	5,0
50	0,2	0	0,022	0,58	0,234	0,002	0,417	0,347	16,9	3,0
50	0,2	20	0,005	0,93	0,589	0,010	0,062	0,339	7,3	4,9
50	0,2	50	0,000	0,92	0,644	0,076	0,007	0,273	5,2	5,4
50	0,2	100	0,002	0,92	0,644	0,076	0,007	0,273	5,2	5,4
50	0,5	0	0,088	0,39	0,052	0,011	0,600	0,338	22,6	2,7
50	0,5	20	0,436	0,58	0,239	0,004	0,412	0,345	16,2	3,0
50	0,5	50	0,006	0,86	0,597	0,085	0,054	0,264	7,0	4,6
50	0,5	100	0,005	0,86	0,597	0,085	0,054	0,264	7,0	4,6
100	0	0	0,005	1,00	0,651	0,000	0,000	0,349	3,0	5,0
100	0	20	0,004	1,00	0,651	0,000	0,000	0,349	3,0	5,0
100	0	50	0,005	1,00	0,651	0,000	0,000	0,349	3,0	5,0
100	0	100	0,004	1,00	0,651	0,000	0,000	0,349	3,0	5,0
100	0,2	0	4,647	0,44	0,095	0,004	0,556	0,345	41,4	3,3
100	0,2	20	0,026	0,94	0,606	0,014	0,046	0,335	10,1	5,2
100	0,2	50	0,016	0,84	0,623	0,132	0,029	0,217	8,6	6,2
100	0,2	100	0,016	0,84	0,623	0,132	0,029	0,217	8,6	6,2
100	0,5	0	3,813	0,41	0,062	0,004	0,589	0,345	43,2	2,8
100	0,5	20	12,990	0,45	0,138	0,043	0,513	0,306	37,8	3,4
100	0,5	50	4,377	0,83	0,587	0,101	0,065	0,247	18,3	4,9
100	0,5	100	4,373	0,83	0,587	0,101	0,065	0,247	18,3	4,9
200	0	0	0,016	1,00	0,652	0,000	0,000	0,348	3,0	5,0
200	0	20	0,014	1,00	0,652	0,000	0,000	0,348	3,0	5,0
200	0	50	0,020	0,89	0,637	0,095	0,015	0,252	5,1	5,7
200	0	100	0,022	0,89	0,637	0,095	0,015	0,252	5,1	5,7
200	0,2	0	39,766	0,42	0,071	0,003	0,581	0,345	86,2	4,0
200	0,2	20	0,224	0,95	0,623	0,019	0,029	0,329	14,0	5,0
200	0,2	50	0,210	0,94	0,645	0,050	0,007	0,298	12,0	5,3
200	0,2	100	0,209	0,94	0,645	0,050	0,007	0,298	12,0	5,3
200	0,5	0	5,949	0,36	0,006	0,000	0,646	0,348	101,6	1,1
200	0,5	20	51,877	0,58	0,277	0,049	0,376	0,299	67,5	3,7
200	0,5	50	109,717	0,66	0,395	0,079	0,257	0,269	51,0	4,4
200	0,5	100	108,203	0,66	0,395	0,079	0,257	0,269	51,0	4,4

Tabela C.9: Rezultati Progol na domeni `member`.

C.4 Rezultati v domeni oddeven

N	data noise	cc	runtime	CA	TP	FP	FN	TN	#literals	#vars
10	0	1	0,023	0,50	0,500	0,500	0,000	0,000	1,0	2,0
10	0	0,1	0,209	0,60	0,450	0,350	0,050	0,150	1,0	1,7
10	0	0,01	2,084	0,60	0,417	0,317	0,083	0,183	1,0	2,1
10	0	0,001	12,922	0,63	0,408	0,279	0,092	0,221	1,4	2,8
10	0	0,0001	70,552	0,69	0,342	0,154	0,158	0,346	2,6	4,8
10	0,2	1	0,028	0,50	0,500	0,500	0,000	0,000	1,0	2,0
10	0,2	0,1	0,328	0,57	0,412	0,338	0,087	0,163	1,0	1,5
10	0,2	0,01	2,494	0,57	0,379	0,304	0,121	0,196	1,1	2,1
10	0,2	0,001	12,247	0,57	0,317	0,242	0,183	0,258	1,6	3,7
10	0,2	0,0001	58,740	0,58	0,303	0,220	0,197	0,280	2,4	4,9
10	0,5	1	0,025	0,50	0,500	0,500	0,000	0,000	1,0	2,0
10	0,5	0,1	0,203	0,57	0,379	0,304	0,121	0,196	1,0	1,6
10	0,5	0,01	2,036	0,57	0,379	0,304	0,121	0,196	1,1	1,8
10	0,5	0,001	11,816	0,57	0,317	0,242	0,183	0,258	1,4	2,5
10	0,5	0,0001	72,509	0,64	0,318	0,178	0,182	0,322	2,5	4,2
50	0	1	0,473	0,53	0,487	0,462	0,013	0,037	1,0	1,9
50	0	0,1	5,613	0,65	0,325	0,175	0,175	0,325	1,0	1,0
50	0	0,01	48,672	0,63	0,338	0,212	0,163	0,287	1,0	2,1
50	0	0,001	164,598	0,82	0,350	0,037	0,150	0,462	2,8	3,8
50	0	0,0001	664,232	0,85	0,350	0,000	0,150	0,500	3,0	4,1
50	0,2	1	0,445	0,50	0,500	0,500	0,000	0,000	1,0	2,0
50	0,2	0,1	5,501	0,57	0,463	0,387	0,037	0,113	1,0	1,8
50	0,2	0,01	48,303	0,53	0,388	0,362	0,113	0,137	1,3	2,5
50	0,2	0,001	188,810	0,72	0,377	0,161	0,124	0,339	2,8	4,9
50	0,2	0,0001	851,389	0,72	0,339	0,124	0,161	0,376	3,0	5,1
50	0,5	1	0,493	0,50	0,450	0,450	0,050	0,050	1,0	1,8
50	0,5	0,1	5,089	0,55	0,425	0,375	0,075	0,125	1,0	1,6
50	0,5	0,01	43,577	0,55	0,425	0,375	0,075	0,125	1,4	3,0
50	0,5	0,001	180,425	0,67	0,379	0,212	0,121	0,287	2,3	4,0
50	0,5	0,0001	948,917	0,71	0,379	0,175	0,121	0,325	2,7	4,7
100	0	1	2,204	0,60	0,450	0,350	0,050	0,150	1,0	1,6
100	0	0,1	26,544	0,55	0,375	0,325	0,125	0,175	1,0	1,4
100	0	0,01	140,895	0,56	0,375	0,313	0,125	0,187	1,3	2,9
100	0	0,001	560,488	0,80	0,300	0,000	0,200	0,500	3,0	4,3
100	0	0,0001	1819,307	0,80	0,300	0,000	0,200	0,500	3,0	4,3
100	0,2	1	1,930	0,53	0,487	0,463	0,013	0,037	1,0	1,9
100	0,2	0,1	25,911	0,57	0,362	0,287	0,137	0,213	1,0	1,3
100	0,2	0,01	140,421	0,48	0,325	0,342	0,175	0,158	1,3	3,2
100	0,2	0,001	698,926	0,64	0,408	0,275	0,091	0,225	2,7	5,1
100	0,2	0,0001	2818,468	0,69	0,436	0,245	0,063	0,255	3,3	5,7
100	0,5	1	2,061	0,55	0,475	0,425	0,025	0,075	1,0	1,8
100	0,5	0,1	23,308	0,60	0,350	0,250	0,150	0,250	1,0	1,2
100	0,5	0,01	125,706	0,60	0,350	0,250	0,150	0,250	1,2	2,6
100	0,5	0,001	690,940	0,58	0,325	0,241	0,175	0,259	1,6	3,0
100	0,5	0,0001	2833,683	0,60	0,317	0,216	0,183	0,284	2,2	5,2
200	0	1	11,310	0,63	0,438	0,312	0,062	0,188	1,0	1,5
200	0	0,1	80,835	0,70	0,401	0,199	0,099	0,301	1,0	1,2
200	0	0,01	465,606	0,87	0,376	0,012	0,124	0,488	2,9	3,9
200	0	0,001	1421,582	0,85	0,351	0,000	0,149	0,500	3,0	4,1
200	0	0,0001	5095,150	0,96	0,463	0,000	0,037	0,500	3,7	4,7
200	0,2	1	10,937	0,60	0,451	0,350	0,050	0,150	1,0	1,6
200	0,2	0,1	83,206	0,60	0,400	0,300	0,100	0,200	1,0	1,4
200	0,2	0,01	496,959	0,73	0,380	0,154	0,120	0,346	2,2	3,5
200	0,2	0,001	1840,996	0,73	0,379	0,151	0,121	0,348	2,6	4,6
200	0,2	0,0001	7497,005	0,93	0,438	0,012	0,062	0,487	3,4	4,6
200	0,5	1	11,684	0,50	0,500	0,500	0,000	0,000	1,0	2,0
200	0,5	0,1	88,767	0,55	0,425	0,375	0,075	0,125	1,0	1,6
200	0,5	0,01	575,864	0,55	0,413	0,362	0,087	0,138	1,5	3,7
200	0,5	0,001	2318,925	0,59	0,405	0,312	0,096	0,188	2,4	5,3
200	0,5	0,0001	8962,735	0,67	0,417	0,250	0,083	0,251	3,1	6,0

Tabela C.10: Rezultati HYPER/CA na domeni oddeven.

N	data noise	noise	runtime	CA	TP	FP	FN	TN	#literals	#vars
10	0	0	0,006	0,51	0,079	0,071	0,421	0,429	5,9	2,2
10	0	20	0,004	0,56	0,188	0,129	0,313	0,371	4,3	1,9
10	0	50	0,003	0,53	0,300	0,267	0,200	0,233	2,9	1,9
10	0	100	0,003	0,53	0,300	0,267	0,200	0,233	2,9	1,9
10	0,2	0	0,005	0,56	0,076	0,013	0,424	0,487	4,8	0,8
10	0,2	20	0,004	0,59	0,212	0,125	0,287	0,375	4,5	2,4
10	0,2	50	0,004	0,53	0,237	0,212	0,263	0,287	2,5	1,0
10	0,2	100	0,003	0,53	0,237	0,212	0,263	0,287	2,5	1,0
10	0,5	0	0,005	0,44	0,029	0,087	0,471	0,412	5,0	1,1
10	0,5	20	0,005	0,45	0,075	0,125	0,425	0,375	3,7	0,6
10	0,5	50	0,003	0,50	0,150	0,150	0,350	0,350	2,8	0,6
10	0,5	100	0,004	0,50	0,150	0,150	0,350	0,350	2,8	0,6
50	0	0	0,025	0,86	0,377	0,013	0,123	0,487	24,4	14,3
50	0	20	0,020	0,69	0,358	0,169	0,142	0,331	18,2	12,5
50	0	50	0,014	0,61	0,396	0,281	0,105	0,219	12,0	8,6
50	0	100	0,014	0,61	0,396	0,281	0,105	0,219	12,0	8,6
50	0,2	0	0,041	0,50	0,059	0,060	0,441	0,440	31,4	12,7
50	0,2	20	0,021	0,51	0,271	0,261	0,229	0,239	20,7	15,7
50	0,2	50	0,010	0,49	0,373	0,388	0,126	0,113	8,5	6,0
50	0,2	100	0,010	0,49	0,373	0,388	0,126	0,113	8,5	6,0
50	0,5	0	0,043	0,49	0,013	0,019	0,487	0,481	29,6	8,0
50	0,5	20	0,022	0,64	0,353	0,218	0,147	0,282	16,6	9,2
50	0,5	50	0,006	0,54	0,451	0,413	0,050	0,087	4,6	3,3
50	0,5	100	0,006	0,54	0,451	0,413	0,050	0,087	4,6	3,3
100	0	0	0,036	0,98	0,491	0,007	0,009	0,493	34,8	26,0
100	0	20	0,037	0,69	0,447	0,258	0,053	0,242	27,7	24,2
100	0	50	0,015	0,69	0,500	0,313	0,000	0,187	14,1	12,5
100	0	100	0,015	0,69	0,500	0,313	0,000	0,187	14,1	12,5
100	0,2	0	0,100	0,62	0,157	0,033	0,343	0,467	57,9	23,4
100	0,2	20	0,046	0,54	0,353	0,318	0,147	0,182	29,2	21,0
100	0,2	50	0,028	0,52	0,419	0,402	0,081	0,098	14,5	10,2
100	0,2	100	0,028	0,52	0,419	0,402	0,081	0,098	14,5	10,2
100	0,5	0	0,106	0,53	0,041	0,014	0,459	0,486	53,7	10,9
100	0,5	20	0,064	0,58	0,292	0,206	0,208	0,294	32,5	13,5
100	0,5	50	0,033	0,58	0,377	0,292	0,123	0,208	16,6	6,5
100	0,5	100	0,033	0,58	0,377	0,292	0,123	0,208	16,6	6,5
200	0	0	0,092	1,00	0,500	0,000	0,000	0,500	55,0	41,6
200	0	20	0,133	0,81	0,488	0,178	0,012	0,322	50,1	36,7
200	0	50	0,074	0,71	0,492	0,284	0,008	0,216	28,2	21,5
200	0	100	0,076	0,71	0,492	0,284	0,008	0,216	28,2	21,5
200	0,2	0	1,963	0,65	0,199	0,046	0,301	0,455	106,8	29,5
200	0,2	20	0,264	0,71	0,423	0,215	0,077	0,285	51,3	25,5
200	0,2	50	0,097	0,63	0,452	0,318	0,048	0,182	27,6	13,1
200	0,2	100	0,100	0,63	0,452	0,318	0,048	0,182	27,6	13,1
200	0,5	0	4,680	0,49	0,070	0,078	0,430	0,422	113,8	22,7
200	0,5	20	0,809	0,64	0,366	0,230	0,134	0,270	62,6	26,3
200	0,5	50	0,091	0,53	0,492	0,463	0,008	0,037	7,6	4,5
200	0,5	100	0,091	0,53	0,492	0,463	0,008	0,037	7,6	4,5

Tabela C.11: Rezultati Aleph na domeni `oddeven`.

N	data noise	noise	runtime	CA	TP	FP	FN	TN	#literals	#vars
10	0	0	0,000	0,54	0,054	0,017	0,446	0,483	4,9	0,9
10	0	20	0,000	0,54	0,138	0,100	0,362	0,400	4,3	1,0
10	0	50	0,000	0,50	0,250	0,250	0,250	0,250	2,8	1,0
10	0	100	0,000	0,50	0,250	0,250	0,250	0,250	2,8	1,0
10	0,2	0	0,004	0,50	0,000	0,000	0,500	0,500	4,9	0,3
10	0,2	20	0,004	0,50	0,000	0,000	0,500	0,500	4,9	0,3
10	0,2	50	0,000	0,50	0,200	0,200	0,300	0,300	2,7	0,8
10	0,2	100	0,000	0,50	0,200	0,200	0,300	0,300	2,7	0,8
10	0,5	0	0,000	0,50	0,050	0,050	0,450	0,450	4,5	0,4
10	0,5	20	0,001	0,50	0,150	0,150	0,350	0,350	3,4	1,0
10	0,5	50	0,000	0,50	0,200	0,200	0,300	0,300	2,6	1,0
10	0,5	100	0,000	0,50	0,200	0,200	0,300	0,300	2,6	1,0
50	0	0	0,016	0,93	0,432	0,000	0,069	0,500	8,8	4,0
50	0	20	0,017	0,88	0,432	0,050	0,069	0,450	8,5	4,0
50	0	50	0,013	0,76	0,439	0,184	0,061	0,316	7,3	4,2
50	0	100	0,014	0,76	0,439	0,184	0,061	0,316	7,3	4,2
50	0,2	0	0,100	0,72	0,243	0,023	0,257	0,477	17,5	4,5
50	0,2	20	0,059	0,80	0,418	0,121	0,082	0,379	11,0	5,1
50	0,2	50	0,500	0,70	0,405	0,207	0,095	0,293	9,3	4,1
50	0,2	100	0,504	0,70	0,405	0,207	0,095	0,293	9,3	4,1
50	0,5	0	0,712	0,63	0,190	0,059	0,310	0,441	22,1	4,9
50	0,5	20	3,097	0,66	0,372	0,216	0,128	0,284	12,9	6,2
50	0,5	50	0,033	0,58	0,451	0,375	0,049	0,125	5,6	3,2
50	0,5	100	0,032	0,58	0,451	0,375	0,049	0,125	5,6	3,2
100	0	0	0,019	0,98	0,485	0,010	0,015	0,491	8,7	4,7
100	0	20	0,015	0,98	0,490	0,010	0,010	0,490	7,5	4,5
100	0	50	0,018	0,96	0,490	0,026	0,010	0,474	7,5	4,6
100	0	100	0,017	0,96	0,490	0,026	0,010	0,474	7,5	4,6
100	0,2	0	12,625	0,70	0,242	0,041	0,258	0,460	31,7	9,7
100	0,2	20	0,254	0,93	0,490	0,058	0,010	0,442	13,3	6,1
100	0,2	50	19,801	0,77	0,466	0,190	0,034	0,310	15,2	6,3
100	0,2	100	19,791	0,77	0,466	0,190	0,034	0,310	15,2	6,3
100	0,5	0	25,300	0,52	0,032	0,011	0,468	0,488	50,9	10,1
100	0,5	20	237,938	0,60	0,294	0,189	0,206	0,311	28,5	5,5
100	0,5	50	25,519	0,59	0,469	0,380	0,031	0,120	13,1	5,5
100	0,5	100	25,588	0,59	0,469	0,380	0,031	0,120	13,1	5,5
200	0	0	0,030	1,00	0,500	0,000	0,000	0,500	6,0	4,0
200	0	20	0,031	1,00	0,500	0,000	0,000	0,500	6,0	4,0
200	0	50	0,029	0,92	0,500	0,076	0,000	0,423	5,9	5,1
200	0	100	0,029	0,92	0,500	0,076	0,000	0,423	5,9	5,1
200	0,2	0	308,983	0,55	0,056	0,004	0,444	0,495	90,3	11,8
200	0,2	20	30,018	0,90	0,475	0,071	0,025	0,429	27,7	8,9
200	0,2	50	0,885	0,87	0,491	0,123	0,009	0,377	16,6	7,9
200	0,2	100	0,888	0,87	0,491	0,123	0,009	0,377	16,6	7,9
200	0,5	0	399,448	0,51	0,014	0,007	0,486	0,493	107,8	16,4
200	0,5	20	31,591	0,82	0,461	0,141	0,038	0,359	37,0	8,5
200	0,5	50	19,553	0,60	0,468	0,367	0,032	0,134	24,6	7,6
200	0,5	100	19,586	0,60	0,468	0,367	0,032	0,134	24,6	7,6

Tabela C.12: Rezultati Progol na domeni `oddeven`.

C.5 Rezultati v domeni `concat`

N	data noise	cc	runtime	CA	TP	FP	FN	TN	#literals	#vars
10	0	1	0,076	0,33	0,333	0,667	0,000	0,000	1,0	3,0
10	0	0,1	0,357	0,46	0,232	0,437	0,101	0,230	1,0	3,0
10	0	0,01	0,961	0,62	0,121	0,168	0,212	0,499	1,4	3,7
10	0	0,001	2,997	0,61	0,134	0,189	0,199	0,478	1,8	4,6
10	0	0,0001	9,825	0,61	0,163	0,218	0,169	0,449	2,0	5,7
10	0,2	1	0,087	0,33	0,333	0,667	0,000	0,000	1,0	3,0
10	0,2	0,1	0,401	0,40	0,296	0,563	0,037	0,104	1,0	3,4
10	0,2	0,01	1,232	0,61	0,222	0,283	0,111	0,384	1,3	4,3
10	0,2	0,001	3,803	0,62	0,176	0,222	0,157	0,445	1,8	4,9
10	0,2	0,0001	13,197	0,63	0,176	0,214	0,157	0,453	1,9	6,1
10	0,5	1	0,072	0,33	0,333	0,667	0,000	0,000	1,0	3,0
10	0,5	0,1	0,396	0,45	0,232	0,452	0,101	0,215	1,0	3,0
10	0,5	0,01	1,117	0,58	0,093	0,183	0,240	0,484	1,3	3,5
10	0,5	0,001	3,349	0,57	0,132	0,231	0,201	0,436	1,7	5,1
10	0,5	0,0001	12,701	0,56	0,124	0,231	0,209	0,436	1,7	5,6
50	0	1	1,909	0,33	0,331	0,669	0,000	0,000	1,0	3,0
50	0	0,1	6,329	0,48	0,233	0,419	0,098	0,250	1,3	3,7
50	0	0,01	20,289	0,60	0,191	0,257	0,140	0,412	1,8	4,5
50	0	0,001	87,376	0,64	0,191	0,222	0,139	0,447	2,0	4,8
50	0	0,0001	326,447	0,64	0,168	0,194	0,163	0,475	2,0	5,1
50	0,2	1	2,100	0,33	0,331	0,669	0,000	0,000	1,0	3,0
50	0,2	0,1	6,888	0,41	0,265	0,525	0,066	0,144	1,2	3,7
50	0,2	0,01	22,554	0,56	0,233	0,346	0,098	0,323	1,7	4,8
50	0,2	0,001	94,505	0,61	0,235	0,292	0,096	0,378	2,0	5,1
50	0,2	0,0001	351,417	0,58	0,193	0,280	0,138	0,390	2,0	5,8
50	0,5	1	2,115	0,33	0,331	0,669	0,000	0,000	1,0	3,0
50	0,5	0,1	6,867	0,48	0,233	0,426	0,097	0,243	1,4	3,8
50	0,5	0,01	19,630	0,60	0,191	0,265	0,140	0,405	1,8	4,6
50	0,5	0,001	80,411	0,65	0,161	0,182	0,170	0,487	2,0	4,5
50	0,5	0,0001	299,402	0,62	0,147	0,201	0,184	0,468	2,0	5,2
100	0	1	4,137	0,33	0,328	0,672	0,000	0,000	1,0	3,0
100	0	0,1	16,588	0,59	0,299	0,383	0,029	0,289	1,2	4,5
100	0	0,01	81,153	0,64	0,275	0,306	0,053	0,366	1,9	5,3
100	0	0,001	332,543	0,65	0,267	0,288	0,061	0,384	2,0	5,4
100	0	0,0001	1105,857	0,65	0,256	0,282	0,072	0,390	2,0	5,5
100	0,2	1	4,025	0,33	0,328	0,672	0,000	0,000	1,0	3,0
100	0,2	0,1	15,195	0,50	0,304	0,474	0,025	0,198	1,5	5,0
100	0,2	0,01	81,250	0,59	0,285	0,369	0,043	0,303	1,7	5,3
100	0,2	0,001	324,885	0,65	0,246	0,272	0,082	0,400	2,0	5,5
100	0,2	0,0001	1005,136	0,65	0,174	0,198	0,154	0,474	2,0	6,0
100	0,5	1	3,873	0,34	0,327	0,664	0,001	0,008	1,0	3,1
100	0,5	0,1	14,974	0,49	0,322	0,509	0,005	0,163	1,4	4,9
100	0,5	0,01	77,506	0,58	0,306	0,402	0,022	0,270	1,9	5,7
100	0,5	0,001	322,166	0,61	0,301	0,362	0,026	0,310	2,0	5,6
100	0,5	0,0001	1090,470	0,63	0,249	0,293	0,079	0,380	2,0	5,8
200	0	1	11,995	0,33	0,321	0,664	0,001	0,014	1,0	3,2
200	0	0,1	75,155	0,59	0,316	0,398	0,006	0,280	1,2	4,5
200	0	0,01	275,903	0,67	0,322	0,329	0,000	0,349	2,0	5,0
200	0	0,001	930,728	0,67	0,322	0,329	0,000	0,349	2,0	5,0
200	0	0,0001	3494,211	0,68	0,320	0,322	0,002	0,356	2,0	5,3
200	0,2	1	11,137	0,32	0,322	0,678	0,000	0,000	1,0	3,0
200	0,2	0,1	73,267	0,61	0,318	0,385	0,004	0,293	1,2	4,4
200	0,2	0,01	266,196	0,67	0,322	0,332	0,000	0,346	2,0	5,2
200	0,2	0,001	911,147	0,67	0,322	0,332	0,000	0,346	2,0	5,2
200	0,2	0,0001	3405,819	0,67	0,312	0,318	0,010	0,360	2,0	5,8
200	0,5	1	11,503	0,33	0,321	0,662	0,001	0,015	1,0	3,2
200	0,5	0,1	79,719	0,60	0,316	0,396	0,006	0,282	1,2	4,5
200	0,5	0,01	286,982	0,66	0,320	0,345	0,002	0,333	2,0	5,3
200	0,5	0,001	1032,767	0,66	0,320	0,343	0,002	0,334	2,0	5,2
200	0,5	0,0001	3890,409	0,66	0,315	0,336	0,008	0,341	1,9	5,6

Tabela C.13: Rezultati HYPER/CA na domeni `concat`.

N	data noise	noise	runtime	CA	TP	FP	FN	TN	#literals	#vars
10	0	0	0,102	0,66	0,073	0,083	0,260	0,584	5,1	4,9
10	0	20	0,168	0,64	0,079	0,103	0,254	0,564	4,2	4,1
10	0	50	0,177	0,64	0,079	0,103	0,254	0,564	4,2	4,1
10	0	100	0,181	0,64	0,079	0,103	0,254	0,564	4,2	4,1
10	0,2	0	0,095	0,64	0,067	0,098	0,265	0,570	5,3	5,4
10	0,2	20	0,137	0,64	0,148	0,177	0,185	0,490	5,2	6,7
10	0,2	50	0,022	0,53	0,214	0,347	0,118	0,320	4,2	6,4
10	0,2	100	0,024	0,53	0,214	0,347	0,118	0,320	4,2	6,4
10	0,5	0	0,223	0,65	0,043	0,061	0,290	0,606	5,1	4,7
10	0,5	20	0,376	0,65	0,043	0,063	0,290	0,605	4,9	4,9
10	0,5	50	0,514	0,61	0,068	0,122	0,265	0,545	4,4	4,3
10	0,5	100	0,513	0,61	0,068	0,122	0,265	0,545	4,4	4,3
50	0	0	3,018	0,74	0,159	0,085	0,172	0,584	27,5	28,9
50	0	20	2,374	0,72	0,252	0,206	0,079	0,464	13,9	19,9
50	0	50	1,815	0,62	0,295	0,346	0,036	0,324	8,5	13,8
50	0	100	1,820	0,62	0,295	0,346	0,036	0,324	8,5	13,8
50	0,2	0	2,475	0,68	0,104	0,095	0,226	0,575	31,1	35,3
50	0,2	20	2,715	0,74	0,261	0,186	0,070	0,483	16,1	21,0
50	0,2	50	1,312	0,59	0,318	0,398	0,013	0,272	7,4	11,4
50	0,2	100	1,319	0,59	0,318	0,398	0,013	0,272	7,4	11,4
50	0,5	0	1,159	0,71	0,116	0,078	0,214	0,592	31,0	32,5
50	0,5	20	1,092	0,74	0,229	0,159	0,102	0,510	14,3	17,6
50	0,5	50	0,545	0,68	0,259	0,247	0,071	0,423	9,8	11,7
50	0,5	100	0,559	0,68	0,259	0,247	0,071	0,423	9,8	11,7
100	0	0	5,778	0,78	0,193	0,079	0,135	0,593	56,4	62,8
100	0	20	4,596	0,80	0,267	0,144	0,061	0,528	18,4	24,7
100	0	50	3,877	0,74	0,266	0,202	0,062	0,470	13,5	17,8
100	0	100	3,866	0,74	0,266	0,202	0,062	0,470	13,5	17,8
100	0,2	0	12,918	0,67	0,113	0,116	0,215	0,557	55,0	55,7
100	0,2	20	7,120	0,68	0,221	0,207	0,107	0,465	21,0	26,2
100	0,2	50	2,491	0,61	0,252	0,312	0,076	0,360	10,6	14,9
100	0,2	100	2,490	0,61	0,252	0,312	0,076	0,360	10,6	14,9
100	0,5	0	25,367	0,65	0,166	0,189	0,162	0,483	62,0	65,1
100	0,5	20	2,643	0,72	0,267	0,222	0,061	0,450	22,2	27,1
100	0,5	50	2,876	0,58	0,319	0,417	0,008	0,255	11,6	14,8
100	0,5	100	2,906	0,58	0,319	0,417	0,008	0,255	11,6	14,8
200	0	0	18,572	0,84	0,195	0,030	0,127	0,648	62,2	66,9
200	0	20	4,831	0,84	0,321	0,162	0,001	0,515	14,0	21,3
200	0	50	5,126	0,81	0,321	0,186	0,001	0,492	12,5	19,0
200	0	100	5,117	0,81	0,321	0,186	0,001	0,492	12,5	19,0
200	0,2	0	85,906	0,71	0,136	0,101	0,186	0,577	111,5	110,2
200	0,2	20	12,052	0,72	0,277	0,229	0,045	0,449	27,0	35,0
200	0,2	50	4,206	0,58	0,306	0,406	0,016	0,272	10,9	14,2
200	0,2	100	4,161	0,58	0,306	0,406	0,016	0,272	10,9	14,2
200	0,5	0	166,513	0,61	0,140	0,203	0,182	0,474	135,5	125,4
200	0,5	20	32,059	0,68	0,276	0,279	0,046	0,399	36,8	38,8
200	0,5	50	29,671	0,59	0,283	0,373	0,040	0,305	29,0	29,2
200	0,5	100	29,895	0,59	0,283	0,373	0,040	0,305	29,0	29,2

Tabela C.14: Rezultati Aleph na domeni concat.

N	data noise	noise	runtime	CA	TP	FP	FN	TN	#literals	#vars
10	0	0	0,011	0,64	0,050	0,075	0,283	0,593	4,3	3,3
10	0	20	0,013	0,64	0,050	0,076	0,283	0,591	4,1	3,3
10	0	50	0,010	0,61	0,082	0,145	0,251	0,523	3,6	3,4
10	0	100	0,010	0,61	0,082	0,145	0,251	0,523	3,6	3,4
10	0,2	0	0,018	0,59	0,135	0,214	0,198	0,453	3,8	3,5
10	0,2	20	0,020	0,57	0,166	0,261	0,168	0,407	3,7	4,0
10	0,2	50	0,006	0,47	0,262	0,464	0,071	0,204	2,5	3,8
10	0,2	100	0,005	0,47	0,262	0,464	0,071	0,204	2,5	3,8
10	0,5	0	0,011	0,58	0,094	0,179	0,239	0,488	3,8	2,9
10	0,5	20	0,011	0,58	0,094	0,179	0,239	0,488	3,8	2,9
10	0,5	50	0,019	0,52	0,149	0,296	0,184	0,371	3,2	3,5
10	0,5	100	0,020	0,52	0,149	0,296	0,184	0,371	3,2	3,5
50	0	0	1,706	0,77	0,206	0,104	0,124	0,566	10,6	9,9
50	0	20	2,246	0,60	0,251	0,325	0,079	0,345	10,0	9,8
50	0	50	1,962	0,56	0,264	0,378	0,066	0,292	7,0	9,0
50	0	100	1,975	0,56	0,264	0,378	0,066	0,292	7,0	9,0
50	0,2	0	3,473	0,65	0,148	0,165	0,183	0,504	17,9	9,1
50	0,2	20	5,077	0,53	0,240	0,380	0,090	0,290	9,9	8,5
50	0,2	50	2,156	0,56	0,299	0,413	0,032	0,256	7,2	8,2
50	0,2	100	2,144	0,56	0,299	0,413	0,032	0,256	7,2	8,2
50	0,5	0	12,661	0,57	0,186	0,284	0,145	0,386	17,0	12,6
50	0,5	20	3,460	0,56	0,220	0,332	0,111	0,337	11,5	11,4
50	0,5	50	5,712	0,55	0,200	0,317	0,131	0,352	11,0	9,2
50	0,5	100	5,704	0,55	0,200	0,317	0,131	0,352	11,0	9,2
100	0	0	19,496	0,86	0,239	0,046	0,089	0,626	17,5	9,1
100	0	20	22,608	0,70	0,237	0,205	0,091	0,467	18,2	9,4
100	0	50	21,628	0,63	0,294	0,336	0,034	0,336	9,9	8,1
100	0	100	21,619	0,63	0,294	0,336	0,034	0,336	9,9	8,1
100	0,2	0	79,085	0,69	0,111	0,093	0,217	0,579	31,9	18,6
100	0,2	20	45,638	0,64	0,198	0,225	0,130	0,447	25,3	15,4
100	0,2	50	38,307	0,52	0,264	0,420	0,064	0,252	16,1	12,6
100	0,2	100	38,433	0,52	0,264	0,420	0,064	0,252	16,1	12,6
100	0,5	0	75,498	0,57	0,117	0,216	0,210	0,456	33,0	22,7
100	0,5	20	31,928	0,51	0,287	0,453	0,041	0,219	13,4	9,1
100	0,5	50	12,454	0,52	0,303	0,455	0,024	0,217	11,2	10,6
100	0,5	100	12,519	0,52	0,303	0,455	0,024	0,217	11,2	10,6
200	0	0	354,798	0,89	0,227	0,011	0,095	0,667	27,3	11,3
200	0	20	150,769	0,83	0,264	0,113	0,058	0,565	16,9	14,7
200	0	50	5,523	0,76	0,322	0,241	0,000	0,437	7,4	11,7
200	0	100	5,557	0,76	0,322	0,241	0,000	0,437	7,4	11,7
200	0,2	0	3410,718	0,69	0,118	0,101	0,204	0,577	75,6	47,2
200	0,2	20	882,300	0,53	0,234	0,387	0,088	0,291	30,5	14,4
200	0,2	50	691,821	0,53	0,282	0,429	0,040	0,249	24,0	14,6
200	0,2	100	674,184	0,53	0,282	0,429	0,040	0,249	24,0	14,6
200	0,5	0	3767,413	0,62	0,042	0,101	0,280	0,577	91,9	39,5
200	0,5	20	1988,555	0,51	0,243	0,413	0,079	0,264	36,5	22,4
200	0,5	50	715,246	0,49	0,264	0,455	0,059	0,223	35,0	29,6
200	0,5	100	713,876	0,49	0,264	0,455	0,059	0,223	35,0	29,6

Tabela C.15: Rezultati Progol na domeni `concat`.

C.6 Primerjava rezultatov

		HYPER/CA				Progol				Aleph			
N	data noise	runtime	CA	#literals	#vars	runtime	CA	#literals	#vars	runtime	CA	#literals	#vars
10	0	0,092	0,93	1,1	1,4	0,004	0,86	3,1	0,4	0,004	0,89	3,9	0,5
10	0,2	0,796	0,84	2,1	2,8	0,006	0,94	3,7	0,0	0,006	0,92	3,9	0,5
10	0,5	0,540	0,75	1,6	2,8	0,007	0,90	4,4	0,2	0,007	0,84	5,0	1,0
50	0	7,359	0,92	2,6	3,1	0,194	0,89	9,4	2,0	0,026	0,90	17,3	5,8
50	0,2	9,017	0,95	2,5	3,1	0,493	0,95	17,2	0,4	0,044	0,95	18,4	2,9
50	0,5	6,359	0,84	2,2	3,5	0,594	0,94	20,6	0,2	0,056	0,94	21,3	1,3
100	0	15,879	1,00	4,0	4,0	0,355	1,00	8,7	2,3	0,071	0,99	29,1	16,0
100	0,2	23,846	1,00	4,0	4,0	1,030	0,99	15,6	2,0	0,160	0,96	35,5	8,7
100	0,5	35,939	0,75	3,4	4,2	2,930	0,96	32,1	2,2	0,200	0,93	42,3	4,8
200	0	34,941	1,00	4,0	4,0	0,717	1,00	8,0	2,0	0,161	1,00	37,9	22,4
200	0,2	63,413	1,00	4,0	4,0	7,248	0,99	25,6	2,3	0,693	0,96	57,2	12,2
200	0,5	104,487	0,58	3,2	4,5	24,622	0,96	61,3	2,6	1,141	0,89	81,5	11,8

Tabela C.16: Primerjava rezultatov na neuravnoteženi domeni `issorted`.

		HYPER/CA				Progol				Aleph			
N	data noise	runtime	CA	#literals	#vars	runtime	CA	#literals	#vars	runtime	CA	#literals	#vars
10	0	0,669	0,56	1,9	3,3	0,007	0,79	3,8	0,8	0,007	0,79	5,5	2,3
10	0,2	0,668	0,54	2,0	3,6	0,009	0,89	4,6	0,4	0,008	0,83	5,5	2,2
10	0,5	1,505	0,70	2,1	4,1	0,006	0,93	4,4	0,2	0,007	0,90	5,0	1,6
50	0	9,517	1,00	4,0	4,0	0,271	0,97	11,2	2,6	0,046	0,91	25,2	11,6
50	0,2	12,687	0,78	3,4	3,7	0,568	0,92	20,8	1,7	0,067	0,88	26,9	6,9
50	0,5	15,965	0,58	2,7	3,4	0,697	0,90	25,3	1,0	0,079	0,93	26,8	1,8
100	0	19,324	1,00	4,0	4,0	0,366	1,00	8,2	2,0	0,174	0,94	48,4	24,0
100	0,2	31,704	0,73	3,1	3,7	4,151	0,96	42,6	2,6	0,290	0,90	52,5	9,6
100	0,5	40,654	0,50	2,6	3,8	4,585	0,95	47,6	1,6	0,216	0,94	52,6	6,2
200	0	41,942	1,00	4,0	4,0	0,701	1,00	8,0	2,0	2,701	0,99	84,4	53,6
200	0,2	98,433	0,53	3,0	3,9	35,475	0,97	81,9	2,9	4,966	0,67	103,7	17,5
200	0,5	105,356	0,51	3,0	3,8	13,369	0,97	43,8	2,6	1,430	0,81	106,1	8,7

Tabela C.17: Primerjava rezultatov na uravnoteženi domeni `issorted`.

		HYPER/CA				Progol				Aleph			
N	data noise	runtime	CA	#literals	#vars	runtime	CA	#literals	#vars	runtime	CA	#literals	#vars
10	0	0,256	0,75	2,6	4,9	0,000	0,49	2,6	1,2	0,001	0,65	2,9	4,3
10	0,2	0,768	0,70	2,4	5,1	0,000	0,45	3,0	0,8	0,003	0,61	5,0	4,1
10	0,5	0,011	0,65	1,0	2,0	0,002	0,52	2,7	1,5	0,002	0,52	2,7	1,3
50	0	1,496	0,97	3,0	5,9	0,000	1,00	3,0	5,0	0,007	1,00	9,4	13,5
50	0,2	3,859	0,93	2,9	6,0	0,005	0,93	7,3	4,9	0,030	0,84	20,3	17,2
50	0,5	7,755	0,90	2,8	6,2	0,005	0,86	7,0	4,6	0,038	0,89	11,5	8,7
100	0	3,136	1,00	3,0	6,0	0,004	1,00	3,0	5,0	0,011	1,00	7,9	11,5
100	0,2	6,997	1,00	3,0	6,0	0,026	0,94	10,1	5,2	0,231	0,91	14,6	13,8
100	0,5	12,931	0,96	3,0	6,0	4,373	0,83	18,3	4,9	0,366	0,81	24,1	15,5
200	0	6,719	1,00	3,0	6,0	0,014	1,00	3,0	5,0	0,011	1,00	8,4	12,0
200	0,2	17,781	1,00	3,0	6,0	0,224	0,95	14,0	5,0	0,575	0,98	18,8	14,2
200	0,5	32,753	1,00	3,0	6,0	108,203	0,66	51,0	4,4	2,129	0,81	35,0	10,8

Tabela C.18: Primerjava rezultatov na domeni `member`.

		HYPER/CA				Progol				Aleph			
N	data noise	runtime	CA	#literals	#vars	runtime	CA	#literals	#vars	runtime	CA	#literals	#vars
10	0	70,552	0,69	2,6	4,8	0,000	0,54	4,9	0,9	0,004	0,56	4,3	1,9
10	0,2	58,740	0,58	2,4	4,9	0,000	0,50	2,7	0,8	0,004	0,59	4,5	2,4
10	0,5	72,509	0,64	2,5	4,2	0,000	0,50	4,5	0,4	0,003	0,50	2,8	0,6
50	0	664,232	0,85	3,0	4,1	0,016	0,93	8,8	4,0	0,025	0,86	24,4	14,3
50	0,2	188,810	0,72	2,8	4,9	0,059	0,80	11,0	5,1	0,021	0,51	20,7	15,7
50	0,5	948,917	0,71	2,7	4,7	3,097	0,66	12,9	6,2	0,022	0,64	16,6	9,2
100	0	560,488	0,80	3,0	4,3	0,015	0,98	7,5	4,5	0,036	0,98	34,8	26,0
100	0,2	2818,468	0,69	3,3	5,7	0,254	0,93	13,3	6,1	0,100	0,62	57,9	23,4
100	0,5	23,308	0,60	1,0	1,2	237,938	0,60	28,5	5,5	0,033	0,58	16,6	6,5
200	0	5095,150	0,96	3,7	4,7	0,030	1,00	6,0	4,0	0,092	1,00	55,0	41,6
200	0,2	7497,005	0,93	3,4	4,6	30,018	0,90	27,7	8,9	0,264	0,71	51,3	25,5
200	0,5	8962,735	0,67	3,1	6,0	31,591	0,82	37,0	8,5	0,809	0,64	62,6	26,3

Tabela C.19: Primerjava rezultatov na domeni `oddeven`.

		HYPER/CA				Progol				Aleph			
N	data noise	runtime	CA	#literals	#vars	runtime	CA	#literals	#vars	runtime	CA	#literals	#vars
10	0	0,961	0,62	1,4	3,7	0,011	0,64	4,3	3,3	0,102	0,66	5,1	4,9
10	0,2	13,197	0,63	1,9	6,1	0,018	0,59	3,8	3,5	0,095	0,64	5,3	5,4
10	0,5	1,117	0,58	1,3	3,5	0,011	0,58	3,8	2,9	0,223	0,65	5,1	4,7
50	0	87,376	0,64	2,0	4,8	1,706	0,77	10,6	9,9	3,018	0,74	27,5	28,9
50	0,2	94,505	0,61	2,0	5,1	3,473	0,65	17,9	9,1	2,715	0,74	16,1	21,0
50	0,5	80,411	0,65	2,0	4,5	12,661	0,57	17,0	12,6	1,092	0,74	14,3	17,6
100	0	332,543	0,65	2,0	5,4	19,496	0,86	17,5	9,1	4,596	0,80	18,4	24,7
100	0,2	324,885	0,65	2,0	5,5	79,085	0,69	31,9	18,6	7,120	0,68	21,0	26,2
100	0,5	1090,470	0,63	2,0	5,8	75,498	0,57	33,0	22,7	2,643	0,72	22,2	27,1
200	0	3494,211	0,68	2,0	5,3	354,798	0,89	27,3	11,3	4,831	0,84	14,0	21,3
200	0,2	266,196	0,67	2,0	5,2	3410,718	0,69	75,6	47,2	12,052	0,72	27,0	35,0
200	0,5	286,982	0,66	2,0	5,3	3767,413	0,62	91,9	39,5	32,059	0,68	36,8	38,8

Tabela C.20: Primerjava rezultatov na domeni `concat`.

Literatura

[1] E. Bengoetxea, P. Larrañaga, I. Bloch, A. Perchant, and C. Boeres. Inexact graph matching by means of estimation of distribution algorithms. *Pattern Recognition*, 35(12):2867–2880, 2002.

[2] S. Benson and N. Nilsson. Inductive learning of reactive action models. In *Proceedings of the International Conference on Machine Learning*, volume 95, pages 47–54, 1995.

[3] S. S. Benson. *Learning Action Models for Reactive Autonomous Agents*. PhD thesis, Stanford University, 1996.

[4] P. Berkhin. A survey of clustering data mining techniques. In *Grouping Multidimensional Data*, pages 25–71. Springer Berlin Heidelberg, 2006.

[5] G. Bisson. Conceptual clustering in a first order logic representation. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 458–462, 1992.

[6] H. Blockeel and L. D. Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1–2):285–297, 1998.

[7] I. Bratko. Refining complete hypotheses in ilp. In *Inductive Logic Programming*, volume 1634 of *Lecture Notes in Computer Science*, pages 44–55. Springer Berlin Heidelberg, 1999.

[8] I. Bratko. Comparison of machine learning for autonomous robot discovery. In *Advances in Machine Learning I*, volume 262 of *Studies in Computational Intelligence*, pages 441–456. Springer Berlin Heidelberg, 2010.

[9] I. Bratko. Discovery of abstract concepts by a robot. In *Discovery Science*, volume 6332 of *Lecture Notes in Computer Science*, pages 372–379. Springer Berlin Heidelberg, 2010.

[10] I. Bratko. Autonomous discovery of abstract concepts by a robot. In *Adaptive and Natural Computing Algorithms*, volume 6593 of *Lecture Notes in Computer Science*, pages 1–11. Springer Berlin Heidelberg, 2011.

[11] I. Bratko. *Prolog Programming for Artificial Intelligence*. Addison-Wesley, 4th edition, 2011.

[12] I. Bratko, D. Šuc, I. Awaad, J. Demšar, P. Gemeiner, M. Guid, B. Leon, M. Mestnik, J. Prankl, E. Prassler, et al. Initial experiments in robot discovery in xpero. In *Proceedings of IEEE International Conference on Robotics and Automation*, 2007.

[13] S. Brown. *A Relational Approach to Tool-use Learning in Robots*. PhD thesis, University of New South Wales, 2009.

[14] S. Brown and C. Sammut. A relational approach to tool-use learning in robots. In *Inductive Logic Programming*, volume 7842 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg, 2013.

[15] E. S. Burnside, J. Davis, V. S. Costa, I. de Castro Dutra, C. E. Kahn Jr, J. Fine, and D. Page. Knowledge discovery from structured mammography reports using inductive logic programming. In *AMIA Annual Symposium Proceedings*, volume 2005, page 96. American Medical Informatics Association, 2005.

[16] M. Carlsson et al. *SICStus Prolog User's Manual*. Swedish Institute of Computer Science, 4.2.0 edition, March 2011.

[17] E. Comission. Future emergent technologies conference 2009. `http://ec.europa.eu/information_society/events/fet/2009/`, July 2013.

[18] D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255, 1994.

[19] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158. ACM, 1971.

[20] J. J. Craig. *Introduction to Robotics: Mechanics and Control*. Prentice Hall, 3rd edition, 2004.

[21] J. Cussens, S. Džeroski, and T. Erjavec. Morphosyntactic tagging of slovene using progol. In *Inductive Logic Programming*, volume 1634 of *Lecture Notes in Computer Science*, pages 68–79. 1999.

[22] DARPA. Darpa robotics challenge. `http://www.theroboticschallenge.org`, July 2013.

[23] L. De Raedt. *Interactive Theory Revision: An Inductive Logic Programming Approach*. Academic Press, 1992.

[24] L. De Raedt. Probabilistic logic learning. In *Logical and Relational Learning*, Cognitive Technologies, pages 223–288. Springer Berlin Heidelberg, 2008.

[25] L. De Raedt and H. Blockeel. Using logical decision trees for clustering. In *Inductive Logic Programming*, volume 1297 of *Lecture Notes in Computer Science*, pages 133–140. Springer Berlin Heidelberg, 1997.

[26] L. De Raedt and K. Kersting. *Probabilistic Inductive Logic Programming*, volume 4911 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2008.

[27] M. R. Doğar, M. Çakmak, E. Uğur, and E. Şahin. From primitive behaviors to goal-directed behavior using affordances. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 729–734. IEEE, 2007.

[28] P. Domingos, S. Kok, H. Poon, M. Richardson, and P. Singla. Unifying logical and statistical ai. In *Proceedings of the 21st National Conference on Artificial Intelligence*, pages 2–7, 2006.

[29] B. Dynamics. Atlas – the agile anthropomorphic robot. `http://www.bostondynamics.com/robot_Atlas.html`, July 2013.

[30] S. Džeroski. Relational data mining. In *Data Mining and Knowledge Discovery Handbook*, pages 887–911. Springer US, 2010.

[31] S. Džeroski and L. Todorovski. Discovering dynamics. In *Proceedings of the 10th International Conference on Machine Learning*, 1993.

[32] R. Fikes and N. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3):189–208, 1972.

[33] P. Finn, S. Muggleton, D. Page, and A. Srinivasan. Pharmacophore discovery using the inductive logic programming system progol. *Machine Learning*, 30(2-3):241–270, 1998.

[34] P. Fitzpatrick, G. Metta, L. Natale, S. Rao, and G. Sandini. Learning about objects through action-initial steps towards artificial cognition. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 3, pages 3140–3145. IEEE, 2003.

[35] X. Gao, B. Xiao, D. Tao, and X. Li. A survey of graph edit distance. *Pattern Analysis and Applications*, 13(1):113–129, 2010.

[36] R. García-Martínez and D. Borrajo. An integrated approach of learning, planning, and execution. *Journal of Intelligent and Robotic Systems*, 29(1):47–78, 2000.

[37] J. J. Gibson. The theory of affordances. *Perceiving, acting, and knowing: toward an ecological psychology*, 1977.

[38] Y. Gil. Learning by experimentation: Incremental refinement of incomplete planning domains. In *Proceedings of the 11th International Conferece on Machine Learning*, 1994.

[39] C. Giraud-Carrier and C. Kennedy. Adfs: An evolutionary approach to predicate invention. In *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, page 224. Springer Verlag Wien, 2001.

[40] M. Gori, M. Maggini, and L. Sarti. Exact and approximate graph matching using random walks. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(7):1100–1111, 2005.

[41] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier. Mechatronic design of nao humanoid. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 769–774. IEEE, 2009.

[42] R. Hirose and T. Takenaka. Development of the humanoid robot asimo. *Honda R&D Technical Review*, 13(1):1–6, 2001.

[43] T. Horváth, S. Wrobel, and U. Bohnebeck. Relational instance-based learning with lists and terms. *Machine Learning*, 43(1):53–80, 2001.

[44] D. Hume and C. Sammut. Applying inductive logic programming in reactive environments. *Inductive Logic Programming*, pages 539–549, 1991.

[45] M. Jaeger. Relational bayesian networks. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence*, pages 266–273. Morgan Kaufmann Publishers Inc., 1997.

[46] I. Jonyer, D. J. Cook, and L. B. Holder. Graph-based hierarchical conceptual clustering. *The Journal of Machine Learning Research*, 2:19–43, 2002.

[47] R. D. King, K. E. Whelan, F. M. Jones, P. G. Reiser, C. H. Bryant, S. H. Muggleton, D. B. Kell, and S. G. Oliver. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427(6971):247–252, 2004.

[48] M. Kirsten and S. Wrobel. Relational distance-based clustering. In *Inductive Logic Programming*, volume 1446 of *Lecture Notes in Computer Science*, pages 261–270. Springer Berlin Heidelberg, 1998.

[49] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara. Robocup: A challenge problem for ai. *AI magazine*, 18(1):73, 1997.

[50] H. Kitano, S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjou, and S. Shimada. Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, volume 6, pages 739–743. IEEE, 1999.

[51] V. Klingspor, K. J. Morik, and A. D. Rieger. Learning concepts from sensor data of a mobile robot. *Machine Learning*, 23(2-3):305–332, 1996.

[52] S. Kok, M. Sumner, M. Richardson, P. Singla, H. Poon, D. Lowd, J. Wang, and P. Domingos. The alchemy system for statistical relational {AI}. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle, WA., 2009.

[53] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge, MA, USA, 1994.

[54] A. Košmerlj, I. Bratko, and J. Žabkar. Embodied concept discovery through qualitative action models. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 19(03):453–475, 2011.

[55] A. Košmerlj, G. Leban, J. Žabkar, and I. Bratko. Gaining insights about objects functions, properties and interactions. *XPERO Report D4.3*, 2009.

[56] V. Krunic, G. Salvi, A. Bernardino, L. Montesano, and J. Santos-Victor. Affordance based word-to-meaning association. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 4138–4143. IEEE, 2009.

[57] J. Laird. Toward cognitive robotics. In *Proceedings of SPIE*, volume 7332, 2009.

[58] P. Langley. *Scientific discovery: Computational Explorations of the Creative Processes*. MIT press, 1987.

[59] P. Langley. The computational support of scientific discovery. *International Journal of Human-Computer Studies*, 53(3):393–410, 2000.

[60] G. Leban, J. Žabkar, and I. Bratko. An experiment in robot discovery with ilp. In *Inductive Logic Programming*, volume 5194 of *Lecture Notes in Computer Science*, pages 77–90. Springer Berlin Heidelberg, 2008.

[61] H. Levesque and G. Lakemeyer. Cognitive robotics. In *Foundations of Artificial Intelligence*, volume 3, pages 869–886. Elsevier, 2008.

[62] D. Lin. *Logic Programs as Declarative and Procedural Bias in Inductive Logic Programming*. PhD thesis, Imperial College London, 2013.

[63] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag New York, Inc., 2nd edition, 1984.

[64] B. Long, Z. M. Zhang, and P. S. Yu. A probabilistic framework for relational clustering. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and data mining*, pages 470–479. ACM, 2007.

[65] C. Matuszek, J. Cabral, M. J. Witbrock, and J. DeOliveira. An introduction to the syntax and content of cyc. In *Proceedings of AAAI Spring Symposium: Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, pages 44–49, 2006.

[66] J. McCarthy. Situations, actions, and causal laws. Technical report, DTIC Document, 1963.

[67] J. McCarthy. Making robots conscious of their mental states. In *Proceedings of AAAI Spring Symposium on Representing Mental States and Mechanisms*, 1995.

[68] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4, 1969.

[69] J. McCarthy, M. Minsky, N. Rochester, and C. Shannon. A proposal for the dartmouth summer research project on artificial intelligence. dartmouth college, 1955.

[70] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl – the planning domain definition language. Technical report, Yale Center for Computational Vision and Control, 1998.

[71] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor. Modeling affordances using bayesian networks. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4102–4107. IEEE, 2007.

[72] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor. Learning object affordances: from sensory–motor coordination to imitation. *Robotics, IEEE Transactions on*, 24(1):15–26, 2008.

[73] S. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.

[74] S. Muggleton. Inverse entailment and progol. *New generation computing*, 13(3-4):245–286, 1995.

[75] S. Muggleton. Learning stochastic logic programs. *Electronic Transactions on Artificial Intelligence*, 4:141–153, 2000.

[76] S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the 5th International Conference on Machine Learning*, pages 339–352. Kaufmann, 1988.

[77] S. Muggleton and J. Chen. Guest editorial: special issue on inductive logic programming (ilp 2011). *Machine Learning*, 89:213–214, 2012.

[78] S. Muggleton, L. De Raedt, D. Poole, I. Bratko, P. Flach, K. Inoue, and A. Srinivasan. Ilp turns 20. *Machine Learning*, 86:3–23, 2012.

[79] M. E. Müller and M. D. Thosar. Learning to understand by evolving theories. In *Knowledge Representation and Reasoning in Robotics workshop at 29th International Conference on Logic Programming*, 2013.

[80] R. Murphy. *An Introduction to AI Robotics*. MIT Press, 1st edition, 2000.

[81] A. Oblak and I. Bratko. Learning from noisy data using a non-covering ilp algorithm. *Inductive Logic Programming*, 6489:190–197, 2011.

[82] H. M. Pasula, L. S. Zettlemoyer, and L. P. Kaelbling. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29:309–352, 2007.

[83] G. Plotkin. A further note on inductive generalization. *Machine intelligence*, 6(101-124), 1971.

[84] J. Quinlan. Learning logical definitions from relations. *Machine learning*, 5(3):239–266, 1990.

[85] M. Raibert, K. Blankespoor, G. Nelson, R. Playter, et al. Bigdog, the rough–terrain quadruped robot. In *Proceedings of the 17th IFAC World Congress*, pages 10823–10825, 2008.

[86] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. *Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy*, 27:359–380, 1991.

[87] S. J. Russell and P. Norvig. *Artificial intelligence: A Modern Approach*. Prentice hall Englewood Cliffs, 3 edition, 2010.

[88] S. J. Russell and P. Norvig. *Artificial intelligence: A Modern Approach*, chapter 25: Robotics. In [87], 3 edition, 2010.

[89] S. J. Russell and P. Norvig. *Artificial intelligence: A Modern Approach*, chapter 10: Classical Planning. In [87], 3 edition, 2010.

[90] E. Şahin, M. Çakmak, M. R. Doğar, E. Uğur, and G. Üçoluk. To afford or not to afford: A new formalization of affordances toward affordance-based robot control. *Adaptive Behavior*, 15(4):447–472, 2007.

[91] G. Salvi, L. Montesano, A. Bernardino, and J. Santos-Victor. Language bootstrapping: Learning word meanings from perception–action association. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 42(3):660–671, 2012.

[92] C. Sammut and R. Banerji. Learning concepts by asking questions. *Machine learning: An artificial intelligence approach*, 2:167–192, 1986.

[93] C. Sammut and D. Hume. Observation and generalisation in a simulated robot world. In *Proceedings of 4th International Workshop on Machine Learning*, pages 267–273, 1987.

[94] M. D. Schmill, T. Oates, and P. R. Cohen. Learning planning operators in real-world, partially observable environments. In *Proceedings of The 5th International Conference on Artificial Intelligence Planning and Scheduling Systems*, pages 246–253, 2000.

[95] P. D. Scott and S. Markovitch. Learning novel domains through curiosity and conjecture. In *Proceedings of the International Joint Conference for Artificial Intelligence*, pages 669–674. Morgan Kaufmann Publishers Inc., 1989.

[96] M. Shanahan and M. Witkowski. High-level robot control through logic. In *Intelligent Agents VII Agent Theories Architectures and Languages*, volume 1986 of *Lecture Notes in Computer Science*, pages 104–121. Springer Berlin Heidelberg, 2001.

[97] E. Shapiro. Algorithmic program debugging. *Dissertation Abstracts International Part B: Science and Engineering*, 43(5):1982, 1982.

[98] W.-M. Shen. Discovery as autonomous learning from the environment. *Machine Learning*, 12(1-3):143–165, 1993.

[99] J. Shrager and P. Langley. *Computational Models of Scientific Discovery and Theory Formation*. Morgan Kaufmann Publishers Inc., 1990.

[100] L. Specia, M. Stevenson, and M. d. G. V. Nunes. Learning expressive models for word sense disambiguation. In *Proceedings of the Annual Meeting – Association for Computational Linguistics*, volume 45, page 41, 2007.

[101] A. Srinivasan. *The Aleph Manual*. Oxford University, 2001.

[102] A. Srinivasan, R. D. King, and S. Muggleton. The role of background knowledge: using a problem from chemistry to examine the performance of an ilp program. *Transactions on Knowledge and Data Engineering*, 1999.

[103] A. Srinivasan, R. D. King, S. Muggleton, and M. J. Sternberg. Carcinogenesis predictions using ilp. In *Inductive Logic Programming*, volume 1297 of *Lecture Notes in Computer Science*, pages 273–287. Springer Berlin Heidelberg, 1997.

[104] A. Srinivasan, S. Muggleton, R. D. King, and M. J. Sternberg. Mutagenesis: Ilp experiments in a non-determinate biological domain. In *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237, pages 217–232, 1994.

[105] I. Stahl. Predicate invention in ilp – an overview. In *Machine Learning: ECML-93*, volume 667 of *Lecture Notes in Computer Science*, pages 311–322. 1993.

[106] M. Thosar. A naive approach for learning the semantics of effects of an action. Master's thesis, University of Applied Sciences Bonn-Rhein-Sieg, 2012.

[107] B. Tribelhorn and Z. Dodds. Evaluating the roomba: A low-cost, ubiquitous platform for robotics research and education. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1393–1399. IEEE, 2007.

[108] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[109] E. Uğur and E. Şahin. Traversability: A case study for learning and perceiving affordances in robots. *Adaptive Behavior*, 18(3-4):258–284, 2010.

[110] E. Uğur, M. R. Doğar, M. Çakmak, and E. Şahin. The learning and use of traversability affordance using range images on a mobile robot. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1721–1726. IEEE, 2007.

[111] T. van der Zant and T. Wisspeintner. Robocup@ home: Creating and benchmarking tomorrows service robot applications. *Robotic Soccer*, pages 521–528, 2007.

[112] M. Veloso, J. Carbonell, A. Perez, D. Borrajo, E. Fink, and J. Blythe. Integrating planning and learning: The prodigy architecture. *Journal of Experimental & Theoretical Artificial Intelligence*, 7(1):81–120, 1995.

[113] X. Wang. Learning planning operators by observation and practice. In *Proceedings of the 2nd International Conference on AI Planning Systems*, pages 335–340, 1994.

[114] X. Wang. Learning by observation and practice: An incremental approach for planning operator acquisition. In *Proceedings of the 12th International Conferece on Machine Learning*, pages 549–557, 1995.

[115] R. Xu, D. Wunsch, et al. Survey of clustering algorithms. *Neural Networks, IEEE Transactions on*, 16(3):645–678, 2005.

[116] T. Zimmerman and S. Kambhampati. Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine*, 24(2):73, 2003.

[117] J. M. Żytkow. Introduction: Cognitive autonomy in machine discovery. *Machine Learning*, 12(1):7–16, 1993.

[118] J. M. Żytkow. Creating a discoverer: autonomous knowledge seeking agent. In *Machine Discovery*, pages 253–283. Springer Netherlands, 1997.

[119] J. M. Żytkow. Automated discovery: A fusion of multidisciplinary principles. In *Advances in Artificial Intelligence*, volume 1822 of *Lecture Notes in Computer Science*, pages 443–448. Springer Berlin Heidelberg, 2000.