УНИВЕРЗИТЕТ У БЕОГРАДУ

Математички факултет

Катедра за рачунарство и информатику

Интегрисано планирање кретања и редоследа извршавања задатака коришћењем SMT решавача

-МАСТЕР РАД-

Студент:
Александра Ђурић

Ментор:
др Предраг Јаничић

Комисија:
др Мирјана Ђорић
др Филип Марић

Београд,

2020.

Садржај

1. Увод

Примена роботике и аутономних система у свим аспектима живота је у порасту, а проблеми који се могу решити њиховом употребом су све бројнији. Роботи временом постају физички способнији и свеснији окружења у којем раде, па су и њихова навигација и планирање начина на који врше акције над објектима све захтевнији. Једна од препознатих класа проблема је комбиновање процедура за планирање редоследа извршавања задатака и алгоритама за планирање кретања. Ефикасни алгоритми решавају ове проблеме изоловано, док њихова интеграција доводи до изазова који се односе како на скалабилност, тако и на ефикасност. Неопходно је интегрисати планер кретања на ниском нивоу, који ради над континуалним простором, и планер редоследа извршавања задатака на високом нивоу, који ради над дискретним доменом. Овом проблему се може приступити на различите начине, а за овај рад је изабран итеративни приступ који се бави прво планирањем редоследа извршавања задатака, а затим планирањем кретања. Ова мастер теза инспирисана је радом *Синтеза интегрисаног планера редоследа извршавања задатака и кретања уз контуру плана, заснована на SMT решавачима* (енг. *SMT-based synthesis of integrated task and motion plans from plan outlines*) [73].

У разматраном проблему централну улогу има робот, који је помоћник у извршењу спољне акције, чије пројектовање и извршење не спада у домен овог система. Главни задатак му је да покупи објекте из свог окружења и смести их у унапред дефинисану циљну област, како би главна спољна акција у којој он помаже, могла да се изврши. Након извршења те акције, задатак робота је да објекте, које је претходно спаковао, сада премести у другу област, где се они складиште, или одакле почиње нека нова спољна акција над њима.

Окосницу представљеног решења проблема чини употреба решавача за испитивање задовољивости у односу на теорију неинтерпретираних функција и теорију линеарне аритметике. Као решавач за испитивање задовољивости у односу на теорију (енг. *satisfiability modulo theory*), односно SMT решавач, коришћен је Z3. Имплементирано решење комбинује информације добијене посредством планера кретања, као и информације неопходне за планирање

редоследа извршавања задатака кроз позивање SMT решавача.

Улази у систем су *опис сцене*, који описује робота и физичко окружење у којем он манипулише, *контура плана*, која описује знање високог нивоа, а коју задаје програмер на основу знања које поседује о изводљивим плановима и *логички захтеви* које план мора да задовољи. Из описа сцене, алгоритам планирања кретања користи се за конструкцију графа распореда, чије путање представљају могуће планове кретања. SMT решавач се користи за симболичко испитивање простора свих могућих планова који одговарају и путањама задатим у графу распореда, али задовољавају и ограничења препозната на основу контуре плана и захтева. Кораци који су коришћени у имплементацији су формално описани помоћу *семантике најслабијег предуслова* (енг. *weakest precondition semantics*) [27].

2. Коришћени појмови и технике

У првом делу ове главе биће описана семантика најслабијег предуслова, која је коришћена као средство за формално описивање препознатих акција које робот треба да изврши. Коришћењем семантике најслабијег предуслова, на основу формалног описа акција, могуће је доказати коректност имплементираног алгоритма за планирање. У другом делу пажња ће бити усмерена на опис решавача задовољивости у теорији, односно SMT решавача, чија је улога да симболички испита простор свих могућих планова.

2.1 Семантика најслабијег предуслова

Да би проблем, за чијим решењем се трага, могао успешно да се моделује, неопходно је кораке за његово извршавање дефинисати као низ акција које је потребно извршити. Да би се могло тврдити да је овакво решење прихватљиво и коректно, неопходно је да се његове акције дефинишу у терминима неког формализма. Пре него што се приступи опису формализма којим је могуће испитати да ли је решење проблема коректно, неопходно је дефинисати појам коректности имплементираног решења.

Дефиниција 2.1.1 (Парцијална коректност)**.** Парцијална коректност подразумева да неки програм, уколико се заустави, даје коректан резултат (тј. резултат који задовољава спецификацију) [50].

Дефиниција 2.1.2 (Тотална коректност)**.** Тотална коректност подразумева да се програм за све улазе (спецификацијом допуштене) зауставља, као и да су добијени резултати парцијално коректни [50].

Једна од техника за доказивање коректности императивних програма је **семантика најслабијег предуслова**. Дефиниције и описи су преузети из књиге Едсгера Дејкстре (енг. *Edsger W. Dijkstra*) *Дисциплина програмирања* [27].

Дефиниција 2.1.3 (Стање система)**.** Свака променљива у систему има скуп могућих вредности који представља њен простор стања. Променљива у сваком

тренутку има једну од могућих вредности, односно она се увек налази у једној тачки простора својих могућих стања. Стање система описује тренутна стања свих промењљивих у систему. Простор стања система дефинише степен слободе система.

Извршавање алгоритма може бити посматрано као систем који путује кроз свој простор могућих стања, прелазећи из једне у другу тачку. Семантика најслабијег предуслова се примарно фокусира на групу система који имају иницијално стање (енг. *initial state*) и завршавају у финалном стању (енг. *final state*) које зависи од иницијалног стања. Претпоставка је да је дизајн ових система заснован на активностима усмереним ка циљу (енг. *goal*).

Пример 2.1.1. (Систем која налази највећи заједнички делилац). *Задатак је направити систем који је одговоран за проналажење највећег заједничког делиоца x за бројеве X и Y. Функција која га проналази је означена са GCD. Главни захтев система, који мора да буде испуњен, може бити представљен на следећи начин:*

$$x = GCD(X, Y) \tag{2.1}$$

Услов (2.1) се назива постуслов, јер означава стање у ком систем треба да се нађе након извршења своје активности. Важно је приметити да он може бити задовољен великим бројем стања. У оваквом случају сматра се да су сва једнако задовољавајућа и, сходно томе, нема разлога да се захтева да финално стање буде нека конкретна функција над иницијалним стањем.
Када је потребно користити овај систем за достизање финалног стања задовољењем постуслова (2.1) за задати скуп вредности X и Y, потребно је пронаћи, тј. израчунати, одговарајући скуп иницијалних стања које ће довести до исправног финалног стања које задовољава постуслов. Ако је систем доведен у једно од тих стања, зна се да ће он произвести одговарајући резултат. Услов којим је описан тај иницијални скуп стања се назива предуслов. Међутим, ако се, на пример, погледа предуслов који осигурава да свако стање које га задовољава, задовољава и постуслов (2.1):

$$GCD(x, y) = GCD(X, Y) \text{ and } 0 < x \leq 500 \text{ and } 0 < y \leq 500 \tag{2.2}$$

и ако се овај предуслов инстанцира за пар конкретних вредности (X, Y) такав да је $GCD(X, Y) = 713$, онда не постоји пар (x, y) који га задовољава. Другим

речима, за те вредности X и Y предуслов (2.2) је нетачан, што значи да задати систем не може да се користи да израчуна највећи заједнички делилац GCD за X и Y.

За много (X, Y) комбинација постоји и много стања које задовољавају услов (2.1). Услов који описује цео скуп иницијалних стања који доводе до финалног стања које задовољава постуслов се назива **најслабији предуслов који одговара задатом постуслову**. Назван је најслабији, јер што је услов слабији то више стања може да опише, а циљ је да опише сва могућа стања која могу довести до жељеног финалног стања. Ако је са S означен систем, а са R жељени постуслов, тада је одговарајући најслабији предуслов могуће записати као

$$wp(S, R) \tag{2.3}$$

Ако иницијално стање задовољава $wp(S, R)$, тада се зна да ће систем успети да успостави истинитост за R. Пошто је $wp(S, R)$ најслабији предуслов, такође се зна да се за иницијална стања која не задовољавају овај предуслов не може гарантовати да ће задовољавати R у свом финалном стању, док се такође може десити да не успеју да достигну финално стање уопште.

Дефиниција 2.1.4 (Најслабији предуслов)**.** За дати систем S и постуслов R, најслабији предуслов $wp(S, R)$ за неко задато иницијално стање, представља предуслов чијим се задовољењем, у случају регуларног извршења, долази до жељеног резултата, а систем S се налази у финалном стању које задовољава постуслов R.

Пример 2.1.2. *За наредбу доделе* $y := 2 * y$; *потребно је пронаћи најслабији предуслов тако да постуслов* $y < 5$ *буде испуњен.*
У овом примеру систем S је $y := 2 * y$; , а постуслов R је $y < 5$. Најслабији предуслов $wp(S, R)$ је $2 * y < 5$, односно

$$wp(y := 2 * y; y < 5) = (2 * y < 5) \tag{2.4}$$

2.1.1 Трансформатор предиката

Ако се детаљније посвети пажња семантици најслабијег предуслова јасно се уочавају две ствари. Прва је да је скуп могућих постуслова огроман и да је њиме због тога немогуће управљати, па је због тога и бескористан. Због

тога је дефиниција семантике дата у другачијем облику који подразумева правила која описују како за било који постуслов R одговарајући најслабији предуслов $wp(S, R)$ може бити изведен. За дати систем S, за који је познат његов простор стања, такво правило, које прослеђено заједно са постусловом задатим у облику предиката R, и које враћа предикат $wp(S, R)$ који означава одговарајући најслабији предуслов, назива се **трансформатор предиката**. Одатле се закључује да се, када се захтева дефиниција семантике за систем S, заправо тражи његов одговарајући трансформатор предиката. Друга ствар која се уочава је да се никада не захтева прецизан опис потпуне семантике неког система због чињенице да се систем S користи само у специфичне сврхе. Другим речима, захтева се успостављање истинитости само за онај постуслов R за који је систем дизајниран. Чак и тада, за специфичан постуслов R, није битна тачна форма која је обухваћена са $wp(S, R)$ и најчешће се мења јачим условом P за који је могуће показати да је

$$P \Rightarrow wp(S, R), \text{за сва стања} \tag{2.5}$$

испуњено.

Предуслови се описују помоћу предиката. Предикат означава формални израз којим се описује услов. Два предиката P и Q су једнака, ако означавају исти услов. Стање за које је предикат *тачан* је оно стање у ком су задовољени услови. Ако је извесно да ће систем доћи до стања које задовољава услов P, тада се за њега каже да је извесно да ће установити *истинитост* предиката P.

Подразумева се да је сваки предикат дефинисан у свакој тачки простора стања, односно, у свакој тачки простора стања предикат је или *тачан* или *нетачан*. Сврха предиката је да опише скуп тачки простора стања где је он тачан. У складу са тим, могуће је тврдити и да су два предиката једнака ако описују исти скуп стања.

Два предиката имају специјалну улогу и они су означени са T и F. T је предикат који је тачан у свакој тачки простора могућих стања. Другим речима, предикат T представља *универзум*. F је предикат који је нетачан у свакој тачки простора могућих стања и он одговара *празном скупу*.

Трансформатор предиката има много особина ако се посматра као функција над постусловом R. Биће објашњене неке од њих.

Тврђење 2.1.1 (Правило искључења чуда (енг. *Law of the excluded miracle*))**.** За сваки систем S важи

$$wp(S, F) = F \tag{2.6}$$

Доказ. Претпоставимо да ово није тачно. Под том претпоставком, постојало би бар једно стање такво да задовољава $wp(S, F)$. Узмимо то стање као иницијално стање за систем S. На основу наше дефиниције, његово активирање би у случају регуларног завршавања, оставило систем S у финалном стању које задовољава F, што је контрадикција. □

Тврђење 2.1.2 (Својство монотоности (енг. *Property of monotonicity*))**.** За сваки систем S и било која два постуслова Q и R за које важи

$$Q \implies R, \text{ за сва стања} \tag{2.7}$$

такође важи и

$$wp(S, Q) \implies wp(S, R), \text{ за сва стања} \tag{2.8}$$

Доказ. Свако иницијално стање које задовољава $wp(S, Q)$ ће, по дефиницији 2.1.4, активацијом изазвати да Q постане тачно. На основу тврђења 2.7 такође ће изазвати да R постане тачно, па ће иницијално стање такође задовољити и $wp(S, R)$, као што је у тврђењу 2.8 и наведено. □

Тврђење 2.1.3. За сваки систем S и било која два постуслова Q и R важи

$$(wp(S, Q) \textbf{ and } wp(S, R)) = wp(S, Q \textbf{ and } R) \tag{2.9}$$

Доказ. У свакој тачки простора стања, лева страна једнакости тврђења 2.9 имплицира десну страну једнакости, зато што за свако иницијално стање које задовољава истовремено $wp(S, Q)$ и $wp(S, R)$ имамо комбиновано знање да ће финално стање задовољавати и Q и R. По дефиницији следи да важи

$$(Q \textbf{ and } R) \implies Q, \text{ за сва стања} \tag{2.10}$$

а на основу правила монотоности можемо да закључимо да важи

$$wp(S, Q \textbf{ and } R) \implies wp(S, Q), \text{ за сва стања} \tag{2.11}$$

Аналогно важи:

$$wp(S, Q \textbf{ and } R) => wp(S, R), \text{ за сва стања} \tag{2.12}$$

Ако важи $A \implies B$ и важи $A \implies C$, онда важи и $A \implies (B \textbf{ and } C)$. Одавде, десна страна једнакости тврђења 2.9 имплицира леву страну у свакој тачки простора стања. Пошто обе стране имплицирају једна другу, оне морају бити једнаке и тиме је тврђење доказано. □

Тврђење 2.1.4. За сваки систем S и било која два постуслова Q и R важи

$$(wp(S, Q) \textbf{ or } wp(S, R)) \implies wp(S, Q \textbf{ or } R), \text{ за сва стања} \tag{2.13}$$

Доказ. На основу дефиниције следи да важи

$$Q \implies (Q \textbf{ or } R), \text{ за сва стања} \tag{2.14}$$

а на основу правила монотоности можемо да закључимо да важи

$$wp(S, Q) \implies wp(S, Q \textbf{ or } R), , \text{ за сва стања} \tag{2.15}$$

Аналогно важи:

$$wp(S, R) \implies wp(S, Q \textbf{ or } R), \text{ за сва стања} \tag{2.16}$$

Ако важи $A \implies C$ и важи $B \implies C$, важи и $(A \textbf{ or } B) \implies C$ и тиме је тврђење доказано. □

У општем случају, правило не важи у другом смеру, док код детерминистичких система важи и може бити на сличан начин доказано.

2.1.2 Семантика најслабијег предуслова као техника доказивања коректности императивних програма

Програм написан у неком добро дефинисаном програмском језику може бити разматран као систем који је добро познат, под условом да знамо одговарајући трансформатор предиката. Да би то било могуће неопходно је увести одговарајуће трансформаторе предиката који ће дефинисати наредбе.

2.1.2.1 Идентитет

За почетак ту је трансформација која представља **идентитет**. Описује се као систем S, такав да за сваки постуслов R, важи да је $wp(S, R) = R$. Оваква

наредба је програмерима позната као празан израз и користи се на местима где се у коду захтева да буде израз, али је у конкретном случају непотребан. Ова наредба је названа **skip** и њена семантика је задата као:

$$wp(\textbf{skip}, R) = R, \text{ за сваки постуслов } R \tag{2.17}$$

2.1.2.2 Прекид

Други једноставан трансформатор предиката је **прекид**, који води до константног најслабијег предуслова који не утиче на постуслов R. Како је већ речено, дате су две константе као предикати, T и F. Систем S такав да је $wp(S, R) = T$ за свако R не постоји јер би то нарушило правило искључења чуда, описано тврђењем 2.6. Систем S такав да је $wp(S, R) = F$ за свако R има трансформатор предиката који задовољава сва неопходна својства. Ова наредба је названа **abort** и њена семантика је задатака као:

$$wp(\textbf{abort}, R) = F, \text{ за сваки постуслов } R \tag{2.18}$$

Улога наредбе је да остави ствари какве су тренутно. Ако је

$$R = T$$

што означава да не постоје више никакви захтеви до финалног стања, чак и тада не постоји одговарајуће иницијално стање. Када се позове, систем неће успети да дође до финалног стања и покушај његове активације се интерпретира као знак неуспеха.

2.1.2.3 Наредба доделе

До сада су описане две наредбе, једна која не ради ништа и друга која увек доводи до неуспеха. Она која следи је свакако занимљивија и базира се на **супституцији**, односно замењивању сваког појављивања промењљиве у формалном запису постуслова R нечим другим, што је јединствено за свако појављивање те промењљиве. Ако је у предикату R свако појављивање промењљиве x замењено неким изразом (E), тада резултат те трансформације записујемо као $R_{x:=E}$. За дато x и E неопходно је размотрити такав систем, да за сваки постуслов R, постоји $wp(S, R) = R_{x:=E}$, где је x координатна

променљива у простору стања, а E израз одговарајућег типа. Ово описује целу класу трансформатора предиката и та класа се назива **наредба доделе**. За њену спецификацију су неопходне три ствари:

1. Идентификација промењиве која треба да буде замењена
2. Чињеница да је супституција одговарајуће правило за трансформатор предиката
3. Дефиниција израза који треба да замени свако појављивање промењиве коју мењамо у постуслову

Ако се променљива x мења изразом (E), најчешће се такав израз записује као:

$$x := E \tag{2.19}$$

Одатле се наредба може дефинисати као:

$$wp(\text{“}x := E\text{”}, R) = R_{x:=E}, \text{ за сваки постуслов } R \tag{2.20}$$

код ког свака координатна променљива x и сваки израз (E) одговарајућег типа могу, ако је то потребно, да се посматрају као семантичка дефиниција оператора доделе.

2.1.2.4 Функционална композиција

До сад су биле описане примитивне наредбе. Неопходно је дефинисати и начине за њихово компоновање. За њихово дефинисање је за почетак неопходно спецификовати два система $S1$ и $S2$ чији су трансформатори предиката познати. Када су они познати, следећи корак је дефинисање правила која се користе за извођење нових трансформатора предиката на основу њих. Резултујући трансформатори предиката су описани својствима композитних објеката, направљених на специфичне начине од $S1$ и $S2$.

Један од најједоставнијих начина за извођење нових функција, на основу постојећих је **функционална композиција**, која подразумева прослеђивање вредности једне функције као аргумент за другу функцију. Ако систем који се разматра представља функционални композитни објекат обележен са “$S1; S2$”, тај систем се дефинише као:

$$wp(\text{“}S1; S2\text{”}, R) = wp(S1, wp(S2, R)) \tag{2.21}$$

Важно је напоменути да с обзиром на то да оба система, и $S1$ и $S2$ имају сва четири наведена својства трансформатора предиката, и њихова композиција ће имати сва та својства. Пример који следи показује да је очувано правило искључења чуда.

Пример 2.1.3. За $S1$ и $S2$ правило искључења чуда говори да је

$$wp(S1, F) = F \text{ и } wp(S2, F) = F \tag{2.22}$$

Одатле се може закључити супституцијом F за R у дефиницији 2.21:

$$\begin{aligned} wp(\text{“}S1; S2\text{”}, F) &= wp(S1, wp(S2, F)) \\ &= wp(S1, F) \\ &= F \end{aligned} \tag{2.23}$$

На сличан начин се може доказати да су и остала три својства очувана.

Композитни систем “$S1; S2$” може бити имплементиран коришћењем правила *прво активирај* $S1$*, а када се он његова активност заврши активирај* $S2$. У дефиницији за $wp(\text{“}S1; S2\text{”}, R)$ задат је постуслов R целог композитног система, који је такође и постуслов трансформатора предиката $S2$, што нам сугерише на то да се активност “$S1; S2$” завршава активношћу $S2$. Одговарајући предуслов за $S2$ $wp(S2, R)$ је дат постусловом трансформатора предиката $S1$, одакле је очигледно да је иницијално стање за $S2$ оно стање које је финално стање за $S1$. Ово осликава временску зависност између њих, односно да је завршетак активности $S1$ временски праћен активирањем $S2$. Увођењем композиције писање програма је могуће конкатенацијом наредби.

2.1.2.5 Заштићена наредба (наредба гранања)

Коришћењем композиције, извршавање система је условљено једино исправним завршетком претходног. Да би се омогућило активирање подсистема које зависи од стања система, неопходно је увести **заштићену наредбу** (енг. *guarded command*), односно **наредбу гранања**. Буловски израз, који је услов да би се системи који припадају наредби која се штити извршили, назива се **чувар** (енг. *guard*). Претпоставка је да се захтева конструисање система код ког ће, ако иницијално стање задовољава Q, финално стање задовољавати R. Претпоставка је и да не постоји јединствена листа наредби

чије би секвенцијално извршавање довело до жељеног исхода у сваком случају. Када би постојала, тада не би било потребе за заштићеном наредбом. Битно је да је могуће наћи листе израза, од којих ће свака радити за неки подскуп иницијалних стања. Свакој од њих се додељује чувар који чини буловски израз који описује за који од могућих подскупа стања је листа адекватна. Ако су чувари дефинисани тако да је, ако је Q задовољено, бар један од чувара задовољен, тада ће систем бити доведен у стање R за свако иницијално стање које задовољава Q. Пре формалне дефиниције, важно је напоменути следеће:

1. Претпоставка је да су сви чувари дефинисани. Ако то није испуњено и ако је могуће да приликом евалуације чувара дође до грешке, тада је и за систем могуће да се не заврши исправно, већ грешком.
2. Ако било које иницијално стање које задовољава услове више од једног чувара тада цео систем постаје недетерминистички, јер није јасно који од њих ће бити одабран за активацију. Ако за свака два чувара важи да искључују један другог, тада је систем детерминистички.
3. Ако за неко иницијално стање важи да не задовољава услове ниједног чувара, такав систем ничему не служи, јер активацијом таквог иницијалног стања долази до неуспеха.

Дефиниција 2.1.5 (Наредба гранања)**.** Ако је IF команда описана као

$$\textbf{if } B_1 \to (SL)_1 \mid B_2 \to (SL)_2 \mid ...B_n \to (SL)_n \textbf{ fi} \tag{2.24}$$

тада за сваки постуслов R важи

$$\begin{aligned} wp(IF, R) = (\exists j : 1 < j < n : B_i) \textbf{ and} \\ (\forall j : 1 < j < n : B_j \Rightarrow wp((SL)_j, R)) \end{aligned} \tag{2.25}$$

Ова формула се чита као: $wp(IF, R)$ је тачно у свакој тачки простора за коју постоји бар једно j за које важи $1 < j < n$ и за које је њен одговарајући чувар, означен са B_j, тачан. Такође, за свако j за које важи $I < j < n$ тако да је B_j тачно, $wp((SL)_j, R)$ је такође тачно, при чему је са SL_j означена варијанта система која ће бити извршена у случају истинитости чувара B_j. Може се

користити и алтернативна форма записа

$$\begin{aligned} wp(IF, R) = (B_1 \textbf{ or } B_2 \textbf{ or } ... \textbf{ or } B_n) \textbf{ and} \\ (B_1 \Rightarrow wp((SL)_1, R)) \textbf{ and} \\ (B_2 \Rightarrow wp((SL)_2, R)) \textbf{ and } ... \textbf{ and} \\ (B_n \Rightarrow wp((SL)_n, R)) \end{aligned} \tag{2.26}$$

У наставку ће се за запис чувара користити скраћеница BB дефинисана као

$$BB = (\exists j : 1 < j < n : B_j) \tag{2.27}$$

Теорема 2.1.1 (Основна теорема наредбе гранања)**.** Нека је са IF означена наредба гранања, а са Q и R предикатски пар, такав да за сва стања важе

$$Q \implies BB \tag{2.28}$$

и

$$(\forall j : 1 \leq j \leq n : (Q \textbf{ and } B_j) \implies wp((SL)_j, R) \tag{2.29}$$

тада за сва стања такође важи и

$$Q \implies wp(IF, R) \tag{2.30}$$

2.1.2.6 Понављање

Да би у систему било могуће понављање неког низа наредби, неопходно је увести **наредбу понављања**, односно **наредбу петље**. Уколико је са BB означен чувар петље, а са S тело петље, наредба петље се интуитивно може посматрати као

$$\textbf{while } BB \textbf{ do } S$$

Уместо разматрања стања кроз која се пролази у свакој итерацији петље посебно, углавном се посматрају услови који описују тај скуп стања.

Дефиниција 2.1.6 (Инваријанта петље)**.** Инваријанта петље је логичка формула која укључује вредности променљивих које се јављају у некој петљи и која важи при сваком испитивању услова петље (тј. непосредно пре, за време и непосредно након извршавања петље) [50].

Основна теорема наредбе гранања 2.1.1 постаје добра основа за дефинисање наредбе понављања, у случају када је предикатски пар Q и R дефинисан као

$$\begin{aligned} R &= P \\ Q &= P \textbf{ and } BB \end{aligned} \tag{2.31}$$

У случају да је формула 2.28 тачна, а формула 2.29 се, на основу тога што важи

$$(BB \textbf{ and } B_j) = B_j \tag{2.32}$$

своди на

$$(\forall j : 1 \leq j \leq n : (P \textbf{ and } B_j) \implies wp((SL)_j, P) \tag{2.33}$$

па на основу формуле 2.30 можемо закључити да важи

$$(P \textbf{ and } BB) \implies wp(IF, P) \text{ , за сва стања} \tag{2.34}$$

Теорема 2.1.2 (Основна теорема понављања)**.** Ако је за предикат P и заштићену наредбу IF формула 2.34 тачна за сва стања, тада за одговарајућу наредбу понављања DO можемо да закључимо да важи

$$(P \textbf{ and } wp(DO, T)) \Rightarrow wp(DO, P \textbf{ and not } BB) \text{ , за сва стања} \tag{2.35}$$

Доказ. Са DO је означен систем који представља тело петље. Са P је означен предикат који представља иваријанту петље. Са BB, чија је дефиниција дата једначином 2.27, у овом случају означен услов петље, а са Bj чувар сваке итерације петље. Терм $wp(DO, T)$ представља најслабији предуслов за који ће се конструкција понављања завршити. Формула 2.35 говори да уколико знамо да важе инваријанта и најслабији предуслов који нам осигурава да ће се конструкција понављања завршити, тада знамо да ће важити и најслабији предуслов који ће осигурати да након извршења тела петље (односно система DO) услов петље више неће бити испуњен, док ће инваријанта и даље бити задовољена. За неко конкретно DO је тешко, ако не и немогуће израчунати $wp(DO, T)$, тако да је предлог да се услови понављања конструишу са заустављањем на уму, тј. тако што се изабере одговарајући доказ заустављања. Програм ће затим бити направљен тако да задовољава услове из претходно замишљеног доказа. Дакле, нека је P инваријанта и нека важи

$$(P \textbf{ and } BB) \Rightarrow wp(IF, P) \text{ , за сва стања} \tag{2.36}$$

и нека је t коначна целобројна функција над тренутним стањем, таква да важи

$$(P \textbf{ and } BB) \Rightarrow (t > 0) \text{ , за сва стања} \tag{2.37}$$

и нека за сваку вредност t_0 и сва стања важи

$$(P \textbf{ and } BB \textbf{ and } t \leq t_0 + 1) \Rightarrow wp(IF, t \leq t_0) \tag{2.38}$$

Тада важи

$$P \Rightarrow wp(DO, T) \text{ , за сва стања} \tag{2.39}$$

одакле, заједно са почетном теоремом за понављање 2.1.2 може да се закључи да важи и

$$P \Rightarrow wp(DO, P \textbf{ and not } BB) \text{ , за сва стања} \tag{2.40}$$

Ово се доказује, математичком индукцијом код које

$$(P \textbf{ and } t \leq k) \Rightarrow H_k(T) \text{ , за сва стања} \tag{2.41}$$

важи за свако $k \geq 0$. Прво је неопходно доказати базу индукције, односно да формула 2.41 важи за $k = 0$. Пошто је $H_0(T) = \textbf{not } BB$, потребно је показати да важи

$$(P \textbf{ and } t \leq 0) \Rightarrow \textbf{ not } BB \text{ , за сва стања} \tag{2.42}$$

Лако је приметити да је формула 2.42 исто што и формула 2.37, јер су обе једнаке

$$\textbf{not } P \textbf{ or not } BB \textbf{ or } (t > 0) \tag{2.43}$$

и одатле важи да је тврђење 2.41 исправно за $k = 0$. Следеће је потребно доказати корак индукције. Ако је претпоставка да је тврђење 2.41 исправно за $k = K$, потребно је доказати да је

$$\begin{aligned}(P \textbf{ and } BB \textbf{ and } t \leq K + 1) &\Rightarrow wp(IF, P \textbf{ and } t \leq K) \\ &\Rightarrow wp(IF, H_k(T))\end{aligned} \tag{2.44}$$

и

$$\begin{aligned}(P \textbf{ and not } BB \textbf{ and } t \leq K + 1) &\Rightarrow \textbf{ not } BB \\ &= H_0(T)\end{aligned} \tag{2.45}$$

Ове две импликације могу бити комбиноване, јер ако важи $A \Rightarrow C$ и $B \Rightarrow D$, тада важи и $(A \textbf{ or } B) \Rightarrow (C \textbf{ or } D)$, па је

$$(P \textbf{ and } t \leq K+1) \Rightarrow wp(IF, H_k(T)) \textbf{ or } H_0(T) = H_{K+1}(T) \tag{2.46}$$

и тиме је доказ корака индукције завршен, што значи да је и тврђење 2.41 доказано за свако $k \geq 0$. Пошто је t коначна фукнција, следи да је

$$(\exists k : k \geq 0 : t \leq k) \tag{2.47}$$

као и

$$\begin{aligned} P &\Rightarrow (\exists k : k \geq 0 : P \textbf{ and } t \leq k) \\ &\Rightarrow (\exists k : k \geq 0 : H_k(T)) \\ &= wp(DO, T) \end{aligned} \tag{2.48}$$

и одатле је доказано тврђење 2.39.

□

Пример 2.1.4. *Доказивање коректности алгоритма за проналажење највећег заједничког делиоца коришћењем семантике најслабијег предуслова*
Петља која се користи за рачунање најмањег заједничког делиоца описана је као

```
while (x <> y) do
    if (x > y)
        x := x - y;
    else if (x < y)
        y := y - x;
    fi
endwhile
```

и она представља систем за који се рачуна најслабији предуслов. Улази који се прослеђују систему су различити ненегативни цели бројеви X и Y за које се тражи најмањи заједнички делилац. За рачунање најмањег заједничког делиоца користе се помоћне променљиве x и y. Иницијално стање система задовољава услов

$$Q : (x = X) \textbf{ and } (y = Y) \textbf{ and } (X <> Y) \textbf{ and } (X > 0) \textbf{ and } (Y > 0) \tag{2.49}$$

Инваријанта петље је

$$P : (GCD(x, y) = GCD(X, Y)) \textbf{ and } (0 < x \leq X) \textbf{ and } (0 < y \leq Y) \tag{2.50}$$

Финално стање задовољава постуслов

$$R : x = y = GCD(X, Y) \tag{2.51}$$

На основу дефиниције инваријанте 2.1.6 и на основу *основне теореме понављања* 2.1.2, закључује се да на почетку петље, пре извршења тела петље важи P **and** BB, где је BB чувар петље, и описан је као $BB : x <> y$, након изласка из петље важи P **and not** BB, а непосредно након извршења тела петље важи P. Систем који представља тело петље чини заштићена наредба. Да би се доказала *парцијална коректност* 2.1.1 неопходно је доказати следећа тврђења:

1. Пошто нема наредби пре уласка у петљу, а пре извршења тела петље важи P **and** BB, то значи да би требало да важи

$$Q \implies (P \textbf{ and } BB) \tag{2.52}$$

Када се P инстанцира формула постаје

$$\begin{aligned} Q \implies ((GCD(x, y) = GCD(X, Y)) \textbf{ and } (0 < x \leq X) \\ \textbf{and } (0 < y \leq Y) \textbf{ and } (x <> y)) \end{aligned} \tag{2.53}$$

Пошто из услова Q важи да је $(x = X)$ и $(y = Y)$, супституцијом се добија формула

$$\begin{aligned} Q \implies ((GCD(X, Y) = GCD(X, Y)) \textbf{ and } (0 < X \leq X) \\ \textbf{and } (0 < Y \leq Y) \textbf{ and } (X <> Y)) \end{aligned} \tag{2.54}$$

Из које директно следи

$$Q \implies ((0 < X) \textbf{ and } (0 < Y) \textbf{ and } (X <> Y)) \tag{2.55}$$

Када се Q инстанцира, формула која се доказује постаје

$$\begin{aligned}((x = X)\ \textbf{and}\ (y = Y)\ \textbf{and}\ (X > 0)\ \textbf{and}\ (Y > 0)\ \textbf{and}\ (X <> Y))\\ \implies ((0 < X)\ \textbf{and}\ (0 < Y))\ \textbf{and}\ (X <> Y)\end{aligned} \tag{2.56}$$

Ова формула се може заменити са T, па је тиме и доказ тврђења 2.52 завршен.

2. Због чињенице да након изласка из петље важи P **and not** BB, а петља је такође и последња наредба у алгоритму, требало би да следећа формула буде истинита

$$(P\ \textbf{and not}\ BB) \implies R \tag{2.57}$$

Уколико се инстанцира P, **not** BB и примени се супституција, формула постаје

$$\begin{aligned}((GCD(x, x) = GCD(X, Y))\ \textbf{and}\ (0 < x \leq X)\ \textbf{and}\ (0 < x \leq Y)\\ \textbf{and}\ (x = y)) \implies R\end{aligned} \tag{2.58}$$

Важи да је $GCD(x, x) = x$, па се супституцијом добија да је формула која се доказује

$$\begin{aligned}((x = GCD(X, Y))\ \textbf{and}\ (0 < x \leq X)\ \textbf{and}\ (0 < x \leq Y)\\ \textbf{and}\ (x = y)) \implies R\end{aligned} \tag{2.59}$$

Конјукцију једнакости $(x = GCD(X, Y))$ **and** $(x = y)$ је могуће записати као $(x = y = GCD(X, Y))$, а то одговара постуслову R, па је формула тачна, чиме је и доказ тврђења 2.57 завршен.

3. Унутар петље се налази if наредба, односно систем S_1. Пошто на почетку извршења петље, пре извршења система S_1 важи $(P$ **and** $BB)$, а након извршења важи P, то значи да би требало да важи

$$(P\ \textbf{and}\ BB) \implies wp(S_1, P) \tag{2.60}$$

Са S_1 је означена заштићена наредба. На основу дефиниције заштићене наредбе, као и формуле 2.26, закључује се да је неопходно да следеће формуле буду истините

$$(P\ \textbf{and}\ BB) \implies (B_1\ \textbf{or}\ B_2) \tag{2.61}$$

$$(P \textbf{ and } BB \textbf{ and } B_1) \implies wp(S_2, P) \tag{2.62}$$

$$(P \textbf{ and } BB \textbf{ and } B_2) \implies wp(S_3, P) \tag{2.63}$$

при чему је са B_1 означен чувар $x > y$, са S_2 је означена наредба доделе $x := x-y;$, са B_2 је означен чувар $x < y$, а са S_3 наредба доделе $y := y-x;$. Могуће је на сличан начин, као и за претходна два тврђења 2.52 и 2.57, утврдити коректност формула 2.61, 2.62 и 2.63, уз поштовање правила којим се описује нареба доделе. С обзиром на то да се доказује на сличан начин, у овој тези доказу њихове коректности неће бити посвећена пажња.

За доказивање *тоталне коректности* 2.1.2 неопходно је и доказати да се алгоритам, односно у овом случају петља, зауставља. Неопходно је пронаћи граничну фукнцију t, такву да

$$(P \textbf{ and } BB) \implies t \geq 0 \tag{2.64}$$

Потребно је да важи и да се t из итерације у итерацију смањује и то може бити описано условом

$$(P \textbf{ and } BB) \implies wp(\text{“}t_1 := t; S_1\text{”}, t_1 > t) \tag{2.65}$$

Пошто је неопходно да гранична функција зависи од променљивих чија се вредност мења у свакој итерацији. Пошто се у итерацији може мењати или x или y, гранична функција мора зависити и од x и од y. У складу са тим, она треба да буде дефинисана као:

$$t = x + y \tag{2.66}$$

Услови да је $t \geq 0$ и да се вредност функције t смањује из итерације у итерацију могу бити формално доказани, али пошто су докази интуитивни овде им неће бити посвећена пажња.

2.2 SMT проблем и SMT решавачи

Дуго времена су се SMT технике користиле углавном као подршка верификацији софтвера. Већ више од 10 година њихова примена се проширује и на друге области рачунарства, као што су планирање, провера модела и аутоматско генерисање тестова. Користе се као главна технологија у многим

сферама које захтевају велике и дискретне просторе могућих стања [20, 51]. Овај приступ се све чешће користи јер велики, чак и бесконачни скупови стања система, могу компактно да се представе у облику формула првог реда.

2.2.1 Теорије првог реда и задовољивост у теорији

Да би било могуће формално дефинисати проблем задовољивости у теорији, неопходно је прво се упознати са основним појмовима теорије првог реда. Користи се логички основ уведен у књизи *Математичка логика у рачунарству* [49], па су и формуле које су истакнуте у наставку текста преузете из ње. На почетку је неопходно дефинисати задовољивост у $\mathcal{L}$-структури, шта значи да је формула тачна у њој, као и модел у $\mathcal{L}$-структури.

Дефиниција 2.2.1. Ако је интерпретација $\mathcal{I}_v$ одређена $\mathcal{L}$-структуром $\mathcal{D}$ и валуацијом v и ако за $\mathcal{L}$-формулу $\mathcal{A}$ важи да је $\mathcal{I}_v(\mathcal{A}) = 1$, онда кажемо да интерпретација $\mathcal{I}_v$ **задовољава** формулу $\mathcal{A}$, да је формула $\mathcal{A}$ **тачна** у интерпретацији $\mathcal{I}_v$ и да је $\mathcal{L}$-структура $\mathcal{D}$ са валуацијом v **модел** формуле $\mathcal{A}$ и пишемо $(\mathcal{D}, v) \models \mathcal{A}$. Формула $\mathcal{A}$ је **задовољива** у $\mathcal{L}$-структури $\mathcal{D}$ ако постоји валуација v таква да је $(\mathcal{D}, v) \models \mathcal{A}$. $\mathcal{L}$-формула $\mathcal{A}$ је **задовољива** ако постоје $\mathcal{L}$-структура $\mathcal{D}$ и валуација v такве да је $(\mathcal{D}, v) \models \mathcal{A}$.

Након што су дефинисани модел и задовољивост, могуће је дефинисати појам ваљаности у $\mathcal{L}$-структури.

Дефиниција 2.2.2. Ако је за неку $\mathcal{L}$-структуру $\mathcal{D}$ формула $\mathcal{A}$ тачна за сваку валуацију v, тј. у свакој интерпретацији $\mathcal{I}_v$, онда кажемо да је $\mathcal{L}$-структура $\mathcal{D}$ **модел** формуле $\mathcal{A}$, кажемо да је формула $\mathcal{A}$ **ваљана** у $\mathcal{L}$-структури $\mathcal{D}$ и пишемо $\mathcal{D} \models \mathcal{A}$. Ако је формула над сигнатуром $\mathcal{L}$ ваљана у свакој $\mathcal{L}$-структури, онда за ту формулу кажемо да је **ваљана** и то записујемо као $\models \mathcal{A}$.

Теорија првог реда може бити задата синтаксички и семантички, па ће прво бити усмерена пажња на њену синтаксичку дефиницију, а одмах затим и на семантичку.

Дефиниција 2.2.3. Теорија првог реда $\mathcal{T}$ задата је сигнатуром $\mathcal{L}$, аксиомама $\mathcal{A}_1, \mathcal{A}_2 ...$ над том сигнатуром и системом $\mathcal{D}$ за дедукцију у логици првог реда. Формула $\mathcal{F}$ је **теорема** теорије $\mathcal{T}$ ако важи $\mathcal{A}_1, \mathcal{A}_2 ... \vdash_D \mathcal{F}$.

Дефиниција 2.2.4. Теорија првог реда $\mathcal{T}$ над језиком $\mathcal{L}$ задата је $\mathcal{L}$ структуром $\mathcal{D}$. Формула $\mathcal{F}$ је **теорема** тако засноване теорије $\mathcal{T}$ ако важи $\mathcal{D} \models \mathcal{F}$.

На пример, линеарна аритметика може се задати структуром целих бројева или структуром реалних бројева са операцијом сабирања и релацијом поређења.

Да би било могуће формално дефинисати задовољивост у теорији, неопходно је претходно дефинисати и модел теорије $\mathcal{T}$.

Дефиниција 2.2.5. Нека је $\mathcal{T}$ теорија првог реда над сигнатуром $\mathcal{L}$. За $\mathcal{L}$-структуру $\mathcal{D}$ кажемо да је **модел теорије** $\mathcal{T}$ ако је свака аксиома теорије $\mathcal{T}$ ваљана у $\mathcal{D}$.

Дефиниција 2.2.6. Формула $\mathcal{F}$ је **задовољива у теорији** $\mathcal{T}$ ако постоји модел $\mathcal{M}$ теорије $\mathcal{T}$ и валуација v таква да је $\mathcal{F}$ тачна у $(\mathcal{M}, v)$.

У наставку ће бити дато пар примера који илуструју испитивање задовољивости у теорији.

Пример 2.2.1. *Ако се посматра теорија линеарне целобројне аритметике и ако су a и b целобројне константе, потребно је пронаћи решење система једначина:* $3x - 4y = 3 \wedge x + 8y = 29$
Задатак се своди на испитивање задовољивости формуле $3x-4y = 3 \wedge x+8y = 29$ у теорији линеарне целобројне аритметике. За валуацију v за коју важи да је $\mathcal{I}_v(x) = 5$ и $\mathcal{I}_v(y) = 3$, систем једначина је задовољив.

Пример 2.2.2. *Коришћењем теорије неинтерпретираних функција могуће је испитати задовољивост базне формуле:* $f(x) = y \wedge g(y) = x \wedge x \neq g(f(x))$

Пример 2.2.3. *Проблем постављања 4 краљице и 4 скакача тако да ниједна фигура не напада друге фигуре се може решити у теорији линеарне аритметике испитивањем задовољивости [2].*

2.2.2 SMT проблем

Проблем испитивања задовољивости у теорији $\mathcal{T}$ се назива **SMT проблем** за теорију $\mathcal{T}$. Одлучивост и сложеност проблема зависи од саме теорије, а испитивање задовољивости најчешће је ограничено на одлучиве фрагменте теорија које имају примену у рачунарству.

Алати који се баве испитивањем задовољивости у теорији се називају **SMT решавачи**. Комплексне проблеме из наведених области је могуће решити малим бројем позива SMT решавача. Због своје све шире примене у последње време развијен је велики број SMT решавача. Неки од најпознатијих, који се и

даље активно развијају, су Z3 [19], Yices [32] и MathSAT 5 [12], од којих ће Z3 бити детаљније приказан у наставку.

Рад модерних SMT решавача се заснива на лењом приступу [3]. Овај приступ прво испитује задовољивост исказне структуре задате формуле логике првог реда употребом решавача буловске, тј. исказне, задовољивости, познатијег као SAT решавач (енг. *Boolean satisfiability solvers*). Уколико је формула задовољива, за добијену исказну валуацију се проверава задовољивост одговарајуће конјункције литерала првог реда у изабраној теорији. За сваку теорију се користи специфична процедура одлучивања. Тако је на пример, за испитивање задовољивости формуле, обједињен рад SAT решавача базираног на DPLL процедури (енг. *Davis–Putnam–Logemann-Loveland procedure*), решавача у теорији функција са једнакошћу и неинтерпретираних функција, решавача за остале теорије (теорија аритметике, низова итд.) и апстрактне машине која омогућава рад са квантификаторима.

2.2.3 SMT-LIB иницијатива

За запис проблема у виду SMT формула користи се SMT-LIB стандард 2.6 [5] дефинисан 2017. године. SMT-LIB је иницијатива покренута 2003. године од стране групе истраживача са идејом да се дефинише заједнички стандард и оформи база референтних улазних проблема, који би у великој мери помогли у будућем развоју, процени и поређењу SMT система. Стандард SMT-LIB 2.6 укључује језик за писање термова и формула у вишесортној логици првог реда, језик за дефинисање позадинских теорија коришћених у решавачима, језик за дефинисање класа формула којима се проверава задовољивост, као и команди које се користе за интеракцију са SMT решавачима преко текстуалног интерфејса.

SMT-LIB дефинише SMT решаваче као софтвер који имплементира процедуру за испитивање задовољивости формуле неке задате теорије. Уобичајене теорије укључују варијанте аритметике, низове, бит-векторе, алгебарске типове података, неитерпретиране функције и њихове међусобне комбинације. SMT решавач састоји се од:

- Основне логике (логика првог реда, модална логика...)
- Позадинске теорије, у односу на коју се проверавају формуле
- Улазних формула, које представљају класе формула које решавач прихвата као улаз

- Интерфејса, односно скупа функционалности које решавач пружа

Основна логика која је коришћена у SMT-LIB формату је верзија вишесторне логике са једнакошћу. Она, по угледу на традиционалне вишесорне логике, подржава основне типове у облику *сорти*, као и *сортне термове*. За разлику од традиционалних логика, овде формуле не представљају синтаксички различиту категорију у односу на термове, већ су то само термови над сортама.

SMT-LIB иницијатива јасно дефинише каталог **позадинских теорија**. Оне почињу од малог скупа популарних теорија, којима се додају нове теорије кад се решавач за њих развије. Теорије које су спецификоване у оквиру стандарда су независне од било ког конкретног решавача. Са друге стране, свака скрипта која је писана у складу са SMT-LIB стандардом индиректно реферише на једну или више теорија. Стандард разликује *основне теорије*, које су дефинисане у каталогу, а у које спадају теорије реалних бројева, низова и слично, и *комбиноване теорије*, које се дефинишу имплицитно у терминима основних теорија.

SMT-LIB прописује јединствени језик првог реда, који користи сорте, а који се користи за задавање **улазних логичких формула**. Често се приликом примене SMT проблема формуле изражавају у неком посебном фрагменту тог језика, и тада се углавном упарују теорија и језик којим се задају улази тако да чине целину. Због тога, стандард прописује начин на који се могу упарити улазни језик и позадинска теорија.

Почевши од верзије 2.0, SMT-LIB стандард дефинише текстуални **интерфејс** за SMT решавач као скриптни језик. SMT решавач имплементира интерфејс као интерпретер скриптног језика. Језик се базира на коришћењу команди, а улазне и излазне функционалности су дефинисане тако да омогуће више од једноставне провере задовољивости. У складу са тим, у интерфејс су укључене команде за подешавање решавача, декларацију нових симбола, додавање и уклањање формула, проверу тренутног скупа додатих формула, испитивање добијеног модела у случају задовољивости, добијање дијагностичких информација и слично.

2.2.4 Z3

Z3 [19] је SMT решавач који је развила компанија Мајкрософт (енг. *Microsoft*). Овај решавач може бити коришћен за испитивање задовољивости у једној или више теорија. То је алат ниског нивоа јер је могуће користити

га само као компоненту у контексту употребе других алата који захтевају решавање логичких формула. У складу са тим, Z3 омогућава своје коришћење прослеђивањем SMT-LIB2 скрипти или путем API-ја (енг. *application programming interface*). API позиви из програмских језика високог нивоа представљају прокси (енг. *proxy*), односно посредника, за позиве Z3 преко API-ја заснованог на програмском језику C. Не постоји наменски алат који би омогућио кориснику директну интеракцију са Z3 без посредства неког другог алата. Имплементиран је у програмском језику `C++`.

Улазни формат за Z3 представља проширење формата описаног у SMT-LIB 2.0 стандарду [4]. Скрипта која се прослеђује решавачу је секвенца наредби. Z3 чува *стек* (енг. *stack*) декларација и тврдњи, а задовољивост испитује за формулу која се налази на стеку. Захтев за проверу задовољивости може вратити информацију да је формула задовољива (и у том случају је могуће употребом Z3 добити њен модел), да није задовољива или да је задовољивост непозната, у случају када решавач не може да утврди њену задовољивост. Приликом неких примена неопходно је испитати задовољивост неколико ограничења који деле неке заједничке дефиниције или тврдње. Због тога су подржане `push` и `pop` наредбе. Команда `push` креира нови домет, чувањем тренутне величине стека. Команда `pop` уклања све тврдње или декларације додате између ње и њене упарене `push` наредбе. Када се позове наредба за испитивање задовољивости она увек ради над садржајем глобалног стека.

На улазне формуле се прво примењује непотпуна *симплификација*, која подразумева коришћење стандардних алгебарских правила за поједностављивање, као што је $p \wedge true \mapsto p$, али и једноставно поједностављивање које без контекста идентификује једнакосне формуле и врши редукције користећи дефиницију, на пример $x = 4 \wedge q(x) \mapsto x = 4 \wedge q(4)$. *Компајлирањем* се прави апстрактно стабло које укључује клаузе, као и конгруетно затворене чворове. Након тога, *језгро конгруетног затворења* (енг. *Congruence closure core*) додељује истинитосне вредности атомима добијене посредством SMT решавача [26].

Z3 подржава *комбиновање теорија*, која користи способност решавача да изведе све импликације, као и да за време претпроцесирања уведе додатне литерале у простор претраге [18]. Сакупљач отпада уклања клаузе које нису коришћене у затвореним гранама, заједно са њиховим атомима и термовима, осим у случају конфликтних клауза које оставља. DPLL решавач који Z3 користи интегрише напредне технике за одсецање. Неке од њих су учење на

основу конфликтних клауза, примена бектрекинга и слично. DPLL решавач потенцијално додељује вредност свим атомима који се јављају у циљу. У пракси, неки од њих нису битни. У случају компликованих теорија, Z3 не додељује вредност тим атомима [21] и то представља главну одлику *релевантне пропагације*. Такође је подржано и *инстанцирање квантификатора* [17]. Z3 омогућава рад са *исказном логиком* подржавањем буловских оператора, као и буловске сорте (енг. *Boolean sort*).

Главни градивни блокови за SMT формуле *теорије неинтерпретираних функција* су константе, функцијски симболи и предикатски симболи. Константе се могу посматрати као функције без аргумената, док се предикатски симболи могу посматрати као функције чија је повратна сорта буловског типа. То значи да се у SMT-у и предикатски симболи третирају слично као функцијски. За разлику од програмских језика, код којих функције могу имати бочне ефекте, могу подићи изузетке или се никад не вратити из функције, у логици првог реда без квантификатора нема бочних ефеката и функције су тоталне. То значи да су дефинисане за сваку улазну вредност. Функцијски симболи и симболи константи у чистој логици првог реда су неинтерпретирани, тј. њихово тумачење није унапред одређено. За разлику од њих, функцијски симболи који припадају сигнатурама теорија имају дефинисано и фиксирано значење. Пример би била функција сабирања означена са + у аритметици. Неинтерпретиране функције и константе су максимално флексибилне јер дозвољавају било какву интерпретацију која је у складу са ограничењима над функцијама или константама.

Z3 подржава *теорију линеарне аритметике над целим бројевима* и *теорију линеарне аритметике над реалним бројевима*. Интерпретација представља додељивање бројева свакој константи. Ако интерпретација за задата тврђења постоји, ту интерпретацију називамо модел за задате формуле. Осим подршке за основне аритметичке операције (сабирање, одузимање, поређење, множење константом итд.), подржано је и дељење константом, целобројно дељење константом, рачунање модула и остатка при дељењу, при чему су те операције интерно мапиране на одговарајуће операције са множењем. Подршка за нелинеарне формуле, односно оне које у себи садрже израз у форми $(t * s)$, при чему ни t ни s нису константни бројеви, у Z3 није потпуна и веома је скупа. Z3 подржава и рад са бит-векторима и низовима као и квантификаторима, али о томе овде неће бити речи јер није од интереса за ову тезу.

3. Интегрисано планирање кретања и редоследа извршавања задатака

Потреба за интегрисаним системима који комбинују планирање кретања и редоследа извршења задатака се у роботици најчешће јавља код система чији је циљ да робот, кретањем кроз простор, успешно врши акције над предметима у сврху постизања циља. Осим физичких ограничења робота, простора у коме се налази и самих предмета, код оваквих система је неопходно испланирати и редослед корака које робот мора да изврши.

На почетку ове главе биће описано шта се подразумева под планирањем редоследа извршавања задатака, а шта под планирањем кретања, као и њихов историјски развој. У наставку ће бити објашњени начини за њихову интеграцију, биће класификовани на основу тога како су компоненте које припадају планерима међусобно побезане и биће дат осврт на радове који су се бавили њиховим комбиновањем. Ова мастер теза је инспирисана чланком који имплементира *Планирање редоследа извршавања задатака прво, затим планирање кретања – итеративни приступ*, па ће опис овог приступа представити добар увод за наредну главу где ће бити описан чланак који јој је послужио као главна инспирација, као и рад који који је том чланку претходио, а први је увео проблем којим се теза бави.

3.1 Планирање редоследа извршавања задатака и геометријско планирање кретања

Планирање редоследа извршавања задатака налази низ акција које воде од задатог почетног стања до жељеног циљног стања [36]. Геометријско планирање кретања обухвата налажење путање без колизије, тј. сударања, од задате почетне позиције до жељене циљне позиције [11].

3.1.1 Планирање редоследа извршавања задатака

Аутоматски планери редоследа извршавања задатака (енг. *task planner*) имају дугу историју у вештачкој интелигенцији и роботици [39]. Међу њима су истакнути хеуристички планери (енг. *heuristic planners*). Такви планери се користе у случајевима када је до испуњења циља могуће доћи на много начина и због тога је потребно користити комбинаторну претрагу (енг. *combinatorial search*) [47, 48]. Такође се истичу и планери који користе методе базиране на ограничењима (енг. *constraint-based methods*) [15, 56, 76, 78], као и они који се заснивају на логичком програмирању [65, 83]. Аутоматски планери ове врсте се могу ослањати и на SAT решаваче [55, 77], као и на SMT решаваче [73, 85]. У том приступу је решавачу неопходно проследити конструисану буловску формулу која представља постојање плана. Задовољивост формуле прослеђене решавачу значи да план постоји, као и да кораци плана могу бити добијени из модела који решавач врати у том случају. Неопходно је да планер редоследа извршавања задатака подржи спецификацију у формату који одговара жељеном домену. Постоје разноврсни логички оквири који се користе за ову сврху, у које спадају темпорална логика (енг. *temporal logic*) [7, 46, 59], структуиран и природан језик [58, 70], логичка база знања [34, 83], PDDL (енг. *Planning Domain Definition Language*) [38, 40, 82], контексно-слободна граматика [14, 69, 84] итд. Генерално се тежи томе да планери буду независни од синтаксе и да дозволе употребу већине примењивих нотација. Први аутономни планер редоследа извршавања корака, коришћен у роботици, звао се СТРИПС (енг. *STRIPS*). Развијен је крајем 60их и почетком 70их на универзитету Станфорд за потребе мобилног робота који се звао Шејки (енг. *Shakey*) [35, 36]. Употреба планирања у вештачкој интелигенцији је значајно напредовала од тада.

3.1.2 Геометријско планирање кретања

Рани радови који су се бавили планирањем кретања (енг. *motion planning*) су се фокусирали само на налажење путање, па је и њихов задатак био дефинисан једино почетном и крајњом позицијом робота. Померање објеката на сцени утиче на конфигурациони простор робота. Неки објекти су везани и кинетички, па у складу са тим планер кретања мора да користи репрезентацију која ће на ефикасан начин омогућити промене, као и подржати интеракцију међу објектима сцене [16]. Најбитније приступе чине хеуристичка претрага [13]

и методе базиране на узорковању (енг. *sampling-based method*) [57, 63]. Планере базиране на хеуристичком планирању је могуће користити за потребе вршења акција над објектима, али је јако тешко и изазовно наћи опште хеуристике које раде за различите врсте акција. С друге стране, планери базирани на узорковању се ефикасно носе са изазовима великих степени слободе робота, без модификација којима би се прилагодили специфичном систему. Мана им је што могу да обезбеде само пробабилистичку потпуност (енг. *probabilistic completeness*), што значи да ако планер не пронађе план то не значи да он не постоји.

Већ одавно је примећено да се за потребе вршења акција над објектима, осим кретања кроз простор, захтева и планирање редоследа секвенцијалног извршавања корака, које укључује подизање, спуштање, гурање или неке друге акције које су неопходне за вршење акција над објектима [6, 28, 44, 66, 67, 81, 86]. Димензионалност, а самим тим и време извршавања, ових проблема се повећава у односу на планирање кретања у којем постоји само акција кретања кроз простор. Највећи део радова који се баве проблемима интеграције се фокусирају на перформансе пре него на потпуност или на налажење генерално применљивог решења.

3.2 Комбиновање планера редоследа извршавања задатака и геометријског планера

Расте потреба за интегрисаним системима који комбинују дискретну апстракцију неопходну за планирање редоследа извршавања задатака, и континуално планирање кретања. Да би се такви системи могли успешно користити, неопходно је да буде могућа њихова анализа и резоновање о њима. Ефикасни алгоритми решавају проблеме планирања редоследа извршавања задатака и планирање кретања изоловано, док њихова интеграција доводи до изазова који се односе како на скалабилност, тако и на ефикасност [15]. Интегрисано планирање кретања и редоследа извршавања задатака је изазовна класа проблема због тога што представља комплексну комбинацију планирања редоследа извршења задатака на високом нивоу и планирање кретања на ниском нивоу. Интеграција овакве врсте лако може довести до система који су превише комплексни и високо-димензионални, због тога што се истовремено

мора вршити претрага у циљу налажења плана у дискретном и континуалном простору, уз бригу о физичком свету, динамици кретања, избегавања колизије и привременим циљевима.

Када се роботу да динамика, иницијално стање, крајње стање и циљ, неопходно је наћи путању којом ће се кретати од почетног до крајњег стања, уз истовремено извршавање задатака које ће допринети да дефинисани циљ на крају буде испуњен. Проблеми са планирањем у овом контексту долазе због узајамног дејства између путања кретања и ограничења неопходних за извршење задатака. Неопходно је да планер редоследа извршавања задатака генерише план кретања који је могућ у физичком свету и који је без колизија са предметима око њега. У комплексним системима, ефикасне технике које се баве дискретним планирањем могу довести до решења које није изводљиво због динамичких ограничења. С друге стране, ефикасне методе за планирање кретања и генерисање путања без колизија могу довести до нарушавања ограничења које захтева планер редоследа извршавања задатака.

3.3 Класификација комбинованих планера

Комбиновање процедура за планирање редоследа извршавања задатака и алгоритама за планирање кретања обједињује алгоритме за обе врсте планирања. Задатак комбинованог планера је да установи и омогући везу између операција које је неопходно извршити у фази планирања задатака и проблема у вези са планирањем кретања. На основу геометријске сцене неопходно је да уме да опише задатке које је неопходно извршити, али и - када је дат план акција, потребно је да нађе који план кретања му одговара. С једне стране потребно је донети дискретне одлуке о задацима и акцијама, и у ту сврху се уводе *симболичка стања*. С друге стране, доносе се одлуке о континуалној путањи без колизија, и оне су описане *геометријским стањима*. Неопходно је одлуке обе врсте доносити истовремено, па одатле и потреба да се приликом репрезентације система, стања представе у облику *јединствених хибридних стања*, која зависе и од симболичких и од геометријских компоненти. Сваку од процедура карактерише предуслов којим је њена акција дефинисана, при чему у дефиницији предуслова за овакве, хибридне потребе, могу учествовати обе врсте компоненти истовремено и равноправно. Приликом валуирања хибридних стања, односно додељивања вредности свим променљивама које утичу на хибридно стање, добијена валуација може носити инфомацију о обе групе

компоненти.

Приступи комбиновању планера редоследа извршавања задатака и планера кретања могу се поделити на основу тога како су њихове компоненте међусобно повезане. У складу са тим, разликују се три категорије [8]:

1. Планирање кретања вођено планирањем редоследа извршавања задатака (енг. *Motion planning guided by task planning*)
2. Планирање редоследа извршавања задатака постављањем упита планеру кретања (енг. *Task planning querying motion planning*)
3. Прво планирање редоследа извршавања задатака, затим планирање кретања - итеративни приступ (енг. *Task planning first, then motion planning - iterated*)

3.3.1 Планирање кретања вођено планирањем редоследа извршавања задатака

Код приступа *планирање кретања вођено планирањем редоследа извршавања задатака* планирање кретања има примарни, а планирање редоследа извршавања задатака секундарни приоритет. Главна пажња је усмерена ка планирању кретања, али је такође подржана и симболичка репрезентација домена (енг. *symbolic representation of domain*). Улога симболичке репрезентације је да прикаже стања планера редоследа извршавања задатака. Стања се репрезентују у сврху структурирања планера кретања и усмеравања претраге ка задовољавајућем решењу. Овај принцип комбиновања користе системи aSyMov [9] и SamplSGD [74]. Такође, у овај приступ може се сврстати и I-TMP [43], иако не укључује сам алгоритам за планирање редоследа извршавања задатака, већ су му могући планови представљени у облику графичке репрезентације графа задатака. Овакви планери имају високо димензионалан конфигурациони простор, који се јавља као последица тога што у планирање кретања може ући велики број препрека, више робота или роботи који имају велики број зглобова и степени слободе. На основу тога може се приметити да робот ради са сложеним акцијама.

Пример 3.3.1. (Померање једног предмета уз услов да сви остали предмети не буду померени са свог места). *Задатак планера редоследа извршавања задатака је одређивање акције и потпроблема који треба да буду истражени да би се успешно дошло до решења. На пример, поменути систем aSyMov позива планер редоследа извршавања задатака као хеуристику за изабране*

акције. Свака дефинисана акција постаје задатак планера кретања, од регије унутар конфигурационог простора у ком је предуслов испуњен, до регије где је ефекат те акције постигнут. aSyMov алтернира између проширивања постојећих мапа акција које су већ укључене и додавања нових акција или почињања нове мапе [8].

3.3.2 Планирање редоследа извршавања задатака постављањем упита планеру кретања

Приступ *планирање редоследа извршавања задатака постављањем упита планеру кретања* подразумева да је направљен план за редослед извршавања задатака, укључујући и неке од акција које се односе на планирање кретања, а које су решене посредством планера кретања или других геометријских решавача. Сваки проблем који се односи на планирање кретања је решен одмах након што је планер редоследа извршавања задатака применио акцију [1, 22, 23, 29–31, 37, 41, 87]. Позив планера кретања иницирају углавном специфичне логичке формуле из предуслова или ефекти примењених акција. То би, на пример, значило да ако се у предуслову налази услов да путања постоји, то би захтевао да се позове планер кретања који проверава да ли путања заиста постоји и изгенерише је. С друге стране, уколико је акција, на пример, позиционирање предмета, неопходно је као ефекат њене примене заиста позиционирати предмет. Да би се акција комплетно испланирала, потребно је позивом планера кретања, а у складу са матрицом трансформације која описује позиционирање, добити информације о начину на који ефекат те акције може да буде остварен.

Један од радова који се базира на овом приступу, а коме ће бити у наставку посвећено више пажње, јер се бави истим проблемом као и мастер теза је *Хијерархијско планирање редоследа извршавања задатака и планирања кретања у садашњости* (енг. *Hierarchical Task and Motion planning in the now*) [52].

3.3.3 Планирање редоследа извршавања задатака прво, затим планирање кретања – итеративни приступ

Интеграција планера редоследа извршавања задатака и планера кретања је слабије спрегнута када се користи *планирање редоследа извршавања задатака*

прво, затим планирање кретања – итеративни приступ. Планер редоследа извршавања задатака прво генерише свој план, а затим планер кретања конструише континуалну путању за акције добијене посредством планера редоследа извршавања задатака. Ако планер кретања не успе да нађе решење, планер редоследа извршавања задатака помоћу бектрекинга (енг. *backtracking*) покушава да дође до другог задовољавајућег решења.

Једно од решења која се базира на овом приступу издваја прављење плана редоследа извршавања задатака као прву фазу [33, 45]. У ту фазу је могуће укључити и геометријске провере задовољивости коришћењем екстерних предикција или функција. Када се направи план редоследа извршавања задатака, позива се планер кретања да конструише континуално кретање робота. Ако се испостави да је то немогуће, он може да дода ограничења планеру редоследа извршавања задатака који покушава, избегавајући додата ограничења, да генерише нови план. На овај начин се итерира све док није нађено задовољавајуће решење на оба нивоа. За редослед извршавања задатака се у предложеном решењу користи неформални калкулатор (енг. *Causal calculator*) [71], који нуди богатију и слободнију репрезентацију од осталих традиционалних формализама који се користе за планирање.

Коришћен је и другачији принцип решавања, код којег пробабилистички модели могу да уче на основу примера [24, 25]. Ови модели се могу користити за геометријско узорковање добијено за време геометријског планирања. Овај принцип подразумева да је мапирање планова научено из скупа лабелираних конфигурација коришћених за тренинг и симболичких предиката који су истинити у тим конфигурацијама. Значајна предност оваквог принципа је да тренинг подаци долазе из природних сцена – део генерише човек, а део представљају ранији планови тог робота. Мапирање представља имплицитна ограничења над подацима, на пример, испуњеност услова да ниједан објекат није близу ивице површине. Из тренинг скупа се учи оцена густине језгра (енг. *kernel density estimate*) за сваки предикат [72, 80]. Ово омогућава да се, осим лабелирања непознатих геометријских стања, пронађу и геометријска стања која се поклапају са симболичким предикатима са великом вероватноћом. То се даље може проширити на налажење геометријских стања за конјукцију симболичких предиката, као и за налажење значајно различитих геометријских стања који одговарају предикату за потребе бектрекинга. Примењују се, на пример, на роботској платформи Џастин (енг. *Justin robotic platform*), у контексту GeRT пројекта. Замишљено је да то буде комплементарно са

оригиналном идејом [54], која је примењива на проблеме који су мање геометријски захтевни. На сличан начин, контролисање платформе Џастин се може унапредити искоришћавањем особине покретљивости. Тада је главни фокус на објектно оријентисаној репрезентацији која енкодира информације о томе како је могуће вршити акције над објектима који припадају различитим класама [64].

Трећа категорија која припада истом принципу се заснива на томе да се геометријске одлуке представљају Сколемовим функцијама (енг. *Skolem functions*). Тада се предлаже коришћење је чисто симболичких планера, уз употребу стандардног PDDL-а (што, подсетимо се, означава *Planning Domain Definition Language*) у кораку планирања редоследа извршавања. Једном када је план редоследа извршавања задатака нађен, покушава се инстанцирање геометријског плана придруживањем вредности Сколемовим функцијама. Када је то немогуће, информација о неуспешној акцији се прослеђује као додатак предуслову и покушава се поново [75].

Радови који се баве поређењим различитих приступа [42, 61, 79] показују да њихов избор зависи од тога колико одсецање се постиже провером геометријских предуслова – висок проценат одсецања, у које спада одсецање изнад 70%, би предност дало планеру који користи приступ *планирање редоследа извршавања задатака постављањем упита планеру кретања*, док би низак проценат одсецања, који подразумева одсецање испод 30%, би боље перформансе постигло коришћењем приступа *планирање кретања вођено планирањем редоследа извршавања задатака* [61]. Некада, најбољи избор би био употреба једног приступа на почетку планирања, а употреба другог приступа за каснију фазу планирања. Идеални планер би био способан да мења приступ, или да их комбинује, у зависности од домена.

4. Варијанта система за интегрисано планирање засновано на употреби SMT решавача

Ова глава је посвећена идејама на којима се заснива у оквиру овог рада пројектовани систем, који интегрише планирање кретања и редоследа извршавања задатака. Први део главе се бави начином спецификовања проблема. С обзиром на то да је овај рад инспирисан чланком *Синтеза интегрисаног планера редоследа извршавања задатака и кретања уз контуру плана, заснована на SMT решавачима* [73], у овој глави ће бити дат и преглед система Robosint који тај рад описује. Основни задатак система Robosint је да испланира акције и начин кретања за робота, који представља помоћника у кући. Он ради у кухињи, а задатак му је да узме сваки прљави суд који се налази на некој од кухињских површина, смести га у машину за прање судова, укључи машину и када она заврши прање сваки опрани суд смести на своје место у остави. Овај проблем је први пут описан у раду *Хијерархијско планирање редоследа извршавања задатака и планирања кретања у садашњости* [52]. С обзиром на то да су се аутори оба рада бавили истим проблемом, неке од идеја које предлажу у првом раду су настале као последица уочених недостатака решења које је предложено када је први пут проблем описан. Због тога ће и њему у овој глави бити посвећена пажња. Уколико се проблем кухињског робота уопшти тако да му се, уместо контекста употребе у кухињи, додели нови општи контекст, у коме је битно да се некој спољној акцији омогући да изврши обраду над предметима, а затим се ти предмети проследе следећој акцији, могуће примене постају бројне. Овај рад се бави тим уопштеним контекстом, при чему идеје за налажење плана и начин спецификовања проблема дели са системом Robosint.

4.1 Начини спецификовања проблема

Улази (енг. *inputs*) у систем су категоризовани на основу тога које информације у себи носе. Први део улазних информација чине информације које описују физичко окружење робота, са свим информацијама које су релевантне за посматрани проблем. Један од предлога је да се део посвећен кретању формулише као проблем над дискретним, предпроцесираним подскупом конфигурационог простора [68]. Такође, информације неопходне за кретање се могу чувати у облику статичких планова (енг. *roadmap*), односно *конфигурационих графова* [73, 85].

Други део улазних информација се односи на захтеве или ограничења које систем мора да испуни да би успешно дошао до циља. Радови који се баве овим проблемима понекад проширују улазе додатним информацијама које помажу у имплементацији специфичних приступа решавању овог проблема. У неким случајевима, сврха додатних услова је да ограниче простор претраге и тиме повећају ефикасност, на пример, прављењем базе знања за робота који помаже у кући коришћењем логичке парадигме [34]. Слично се постиже и коришћењем геометријских ограничења у циљу ограничавања простора у коме се планирање одвија и одбацивања немогућих геометријских решења [60]. Као додатак проблему планирања, систему се може проследити и знање о домену над којим се планирање врши. Тако, на пример, планирање хијерархијске мреже задатака (енг. *hierarchical task network*) прима као улаз схему у којој је описано како да се рекурзивно декомпонују комплексни задаци. Те схеме могу да се интерпретирају као граматике над језиком исправних планова [39]. Редукција простора могућих решења се може решити и прослеђивањем контуре плана, а у том случају парцијално знање о домену корисник прослеђује у облику лако читљивог програмског кода. [73].

4.1.1 Употреба графа распореда за потребе планирања кретања

Најчешћи начин приказа проблема управљања, тј. манипулационих проблема, је коришћењем манипулационог графа (енг. *manipulation graph*) [62]. Чвор манипулационог графа представља појединачну конфигурацију робота и објеката који се могу померати унутар његовог радног простора. Гране графа представљају преласке из једне конфигурације у другу, који

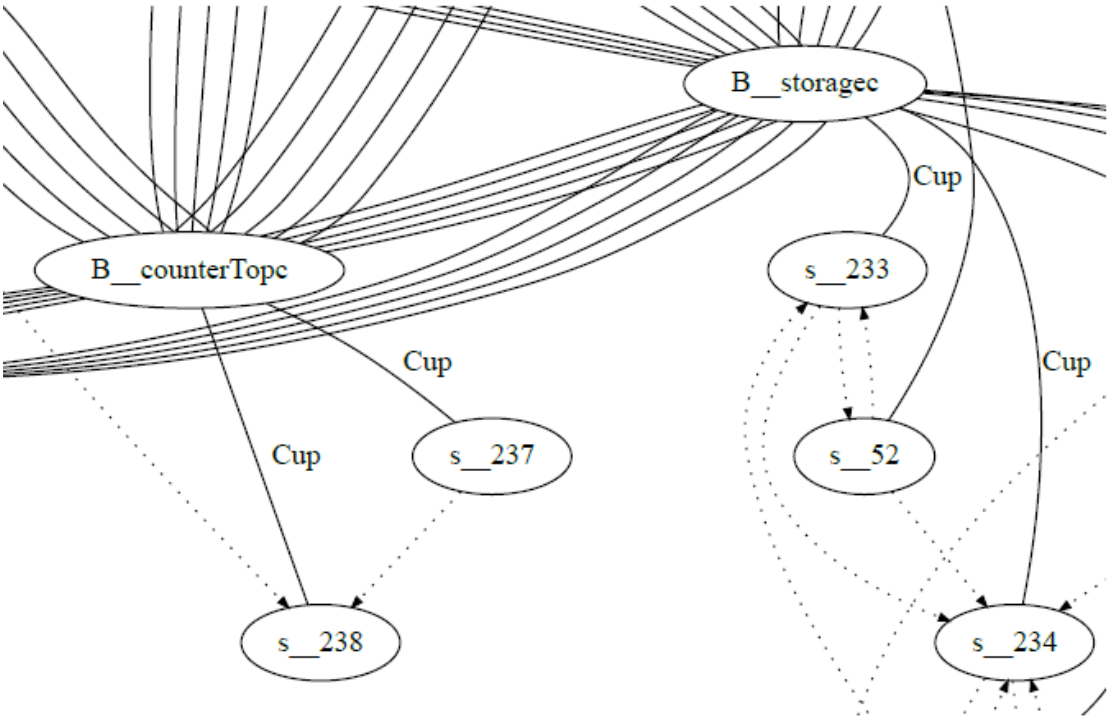
су у физичком свету могући. За потребе интегрисаног планирања мобилног робота, овде је коришћена варијанта манипулационог графа која се зове **граф распореда** (енг. *placement graph*).

Конструисање графа распореда припада фази описа физичког окружења у ком се робот креће и врши акције над предметима, као и особина самог робота, али и предмета над којима врши акције. На основу тога да ли су особине физичког окружења сталне или се мењају у зависности од иницијалног стања сцене, тај опис се дели на *елементе домена* и *елементе сцене*. Елементи домена представљају непроменљиве информације о физичком окружењу и у њих спадају распоред сталних препрека, површина на којима могу бити смештени предмети, места где се налазе машина за обраду и простор за складиштење, као и путање које везују те елементе. Елементима сцене припадају информације које су променљиве за сваки специфични проблем, а који описују почетни распоред предмета, њихов број и позицију робота.

Чворови графа распореда су, на основу тога шта је њима репрезентовано, подељени на **базне тачке** (енг. *base points*) тј. б-тачке, и **стабилне тачке** (енг. *stable points*) тј. с-тачке. **Б-тачке** носе информацију о томе које су могуће конфигурације робота такве да он, када се налази у њима, може без сопственог померања да дохвати локације на којима могу бити смештени објекти. Приликом избора б-тачке пожељно је водити рачуна да она буде изабрана на начин који омогућава да се из ње приступи што је могуће већем броју с-тачака. **С-тачке** представљају места на површинама где могу бити смештени објекти којима робот треба да манипулише у циљу извршења задатка, као и конфигурације у којима робот стабилно држи тај објекат. За сваку групу објеката је дефинисан скуп стабилних тачака. У овој тези с-тачке представљају само места на површинама и није направљена подела између с-тачке сваке појединачне групе предмета, већ се посматра као јединствен скуп.

Пример на слици 4.1, која је преузета из рада [73], приказује део једног графа распореда. Б-тачке су обележене префиксом $b\_$, док с-тачке имају префикс $s\_$. На слици су на различит начин означене различите врсте грана, при чему испрекидане гране репрезентују везе између б и с-тачака, док су пуном линијом обележене гране између б-тачака. Да би се разумела трећа врста грана која означава које с-тачке блокирају друге с-тачке, неопходно је дефинисати појам блокирања између с-тачака.

Дефиниција 4.1.1. (Међусобно блокирање с-тачака). *Кажемо да с-тачка i*



Слика 4.1: Граф распореда за проблем робота који помаже у кухињи

блокира c-тачку j ако и само ако објекат који је смештен у c-тачки i доводи до колизије са роботом или другим објектом дуж путање од c-тачке j до њене родитељске b-тачке.

4.2 Хијерархијско планирање редоследа извршавања задатака и планирања кретања у садашњости

Хијерархијско планирање редоследа извршавања задатака и планирања кретања у садашњости [52] је похлепан хијерархијски приступ који подразумева да планер ради са различитим нивоима апстракције, при чему на сваком потпроблему ради регресивни планер. Под термином регресивни планер се подразумева да планер ради уназад, односно да од главног циља прави потциљеве, на чијем задовољењу рекурзивно ради. Стања су представљена погодним структурама података које могу чувати и геометријске информације и буловске атрибуте (енг. *Boolean attributes*). Стање компоненти није експлицитно репрезентовано. С друге стране, скуп исказа, као и њихова интерпретација у контексту ових структура могу бити задати. Акције су представљене

различитим нивоима апстракције, што се огледа у одлагању неких од предуслова.

Употреба генератора који праве везу између симболичког и геометријског резоновања је од изузетне важности за овај приступ. Генератори подржавају издвајање специфичних региона, као и путања које доводе робота у нову конфигурацију, односно конфигурацију из које је могуће да дохвати или смести задати објекат. Генератори путања могу и додатно проширити простор информација које су иницијално познате за време посећивања пронађених путања. Те додатне информације могу бити коришћене у предусловима. Пример њихове употребе је чишћење путања које имају препреке. Главни принцип планирања и извршавања подразумева да се примитивна верзија акције изврши чим буде пронађена. То значи да је овај принцип потпун једино за проблеме који су серијабилни. Иако је веома моћан приступ, он се темељи на претпоставци да акције могу бити враћене уназад. То није увек случај, јер неке акције понекад трајно мењају систем, па овај приступ није универзално примењив. Некада, иако је то теоретски могуће, враћање великог броја акција је практично немогуће. Овај приступ је проширивани до варијанти када сам исход постаје неизвестан [53].

Док се итерира између две врсте планирања, неопходно је прослеђивати и обрађивати повратне информације у циљу побољшања перформанси наредне итерације. У случајевима када није пронађено решење, често се јавља потреба за коришћењем *геометријског бектрекинга*. Она настаје због чињенице да једно симболичко стање или акција могу бити геометријски инстанцирани на бесконачно много начина. Када симболичка акција није геометријски исправна јавља се потреба за применом бектрекинга у простору геометријске конфигурације, што значајно повећава комплексност планирања. Једно од предложених решења је коришћење *интервала* за репрезентацију геометријске конфигурације, уз истовремено коришћење техника пропагације ограничења (енг. *constraint propagation techniques*) у циљу смањења ових интервала. Након пропагације или је интервал смањен, што такође смањује и простор претраге у коме се може јавити потреба за применом бектрекинга, или је ограничење контрадикторно, што индикује да је низ акција неизводљив, без потребе за даљом обрадом [60]. Главни захтев је налажење алтернативних планова. Код овог система итерира се између планирања и извршавања. Простор претраге се тиме смањује, али се и захтева примена реверзних акција када дође до бектрекинга.

4.3 Интегрисано планирање на примеру система Robosint

Једна од идеја којом су се аутори рада водили је да се приликом имплементације користи SMT решавач као додатак континуалним алгоритмима за планирање. Као што је већ поменуто у поглављу 2.2, SMT решавачи проверавају задовољивост логичке формуле у формулама логике првог реда без квантификатора.

Рад *Синтеза интегрисаног планера редоследа извршавања задатака и кретања уз контуру плана, базирана на SMT основама* (енг. *SMT-based synthesis of integrated task and motion plans from plan outlines*) [73] интеграцију планера редоследа извршавања задатака и кретања имплементира системом који је назван **Robosint**. Аутори предлажу да се имплементација система заснива на двема кључним идејама:

- **Од корисника се захтева да систему проследи информације о спецификацији.** На основу информација које програмер зна о планирању акција робота, а које се односе на то како прихватљива решења изгледају, решавач може да одбаци велики број неприхватљивих решења унапред и тиме смањи свој простор претраге. Те информације се задају на неком интуитивном језику високог нивоа.
- **Коришћење SMT решавача као додатка континуалним алгоритмима за планирање.**

Систем је пројектован у складу са следећим комплексним додацима:

- Скуп прљавог посуђа може бити смештен било где у кухињи. Иако за сваки специфичан случај њихов положај јесте познат, приликом задавања контуре плана сматра се да те информације нису познате, јер варирају од случаја до случаја.
- Редослед паковања судова у машину је важан.
- Постоје ограничења у вези са путањама које робот прелази. На пример, може да постоји простор кроз који не сме да прође или ограничење у максималној дужини путање коју прелази у циљу потпуног извршења задатка.

Улази (енг. *inputs*) у систем Robosint могу бити категоризовани на основу тога које информације у себи носе. Препознате су три категорије улаза које се задају систему:

- **Опис сцене** (енг. *scene description*), којим се задаје спецификација робота и физички простор у ком он ради.
- **Контура плана** (енг. *plan outline*), која на високом нивоу описује оно што програмер зна о томе како прихватљиви планови изгледају. Те информације се користе да би се дефинисао простор претраге интегрисаног планера.
- **Скуп захтева** (енг. *requirements*), који укључује логичке и семантичке захтеве које план мора да испуни.

4.3.1 Опис сцене

Први део улаза је **опис сцене**. Он се дели у два независна скупа информација која се зову **домен** (енг. *domain*) и **сцена** (енг. *scene*), а који се деле у односу на то да ли су информације које у себи носе променљиве или су сталне и не зависе од иницијалног стања.

Домен у себи садржи информације које су сталне и не зависе од инстанце на којој се тренутно ради. У овом случају може носити информацију о томе које су сталне препреке које постоје у кухињи и где су оне смештене, позицију површина на којима могу стајати прљави судови и стабилне позиције за робота и судове, о чему ће бити више речи касније.

Сцена садржи информације које се мењају од инстанце до инстанце проблема. У конкретном примеру, то се огледа у броју и почетном распореду прљавих судова.

4.3.2 Контура плана

Као што је већ поменуто, овај улаз задаје програмер. Он доноси закључке на високом нивоу о томе како успешан план може да изгледа, у циљу одбацивања неупотребљивих решења. То доприноси повећању ефикасности. Ове информације не варирају од случаја до случаја, него се односе на генералне закључке у вези решења.

План који се добија имплементацијом решења које је предложено у раду [52], а описано у поглављу 4.2, може захтевати да се тек смештен предмет поново премешта, да би се померио са пута или да би се следећем предмету обезбедило место да буде смештен. Овакав план је неефикасан јер има кораке које је могуће избећи, а захваљујући контури плана која омогућава да се

једноставном петљом обезбеди да план буде ефикасан, планови у које би ушле овакве непотребне акције су елиминисани.

Пример једног од ограничења које програмер може задати, а које је очигледно у стварном свету, али систем о томе не може да има знање, је да робот прво мора да покупи прљави суд, пре него што га смести у машину. Иако делује као нешто што се само по себи подразумева, планови који не испуњавају овај услов би такође ушли у простор могућих планова, а њима се не би дошло до жељеног циља.

Алгоритам 1: Контура плана за проблем прања судова

Резултат : clean($DIRTY$) & contains(Storage, $DIRTY$)
Инваријанта : (length(?$path$) <= 10) & !crosses(?$path$, FoodPrepArea)

```
1  # import KitchenDomain
2  # import KitchenScene
3
4  Path path, path1, path2, pathR
5  Region tempR, somewhere
6  Location loc1
7
8  void main()
9  begin
10     foreach dish o ∈ DIRTY do
11         findPlace(?loc1, Dishwasher)
12         pickup(o, ?somewhere, path1)
13         place(o, ?loc1, ?path2)
14     end
15     run(Dishwasher)
16     foreach dish o ∈ DIRTY do
           /* move dishes from Dishwasher to Storage                    */
17     end
18 end
```

Идеја је да се систему Robosint проследе информације у облику лако читљивог кода, који подсећа на код у програмском језику C. Пример који аутори предлажу је дат у алгоритму 1. Кроз код су идентификовани циљ главног проблема и неопходни кораци којим се проблем решава. Дефинисан је и редослед извршавања корака, као и улази који се у облику познатих променљивих, прослеђују сваком од препознатих корака. Препознати су и излази (енг. *outputs*), који представљају вредности променљивих, а који су добијени извршавањем конкретног корака. Да би се боље разумела идеја на којој цело решење почива, у наставку ће бити објашњена контура плана.

Линије 1 и 2 су задужене за импортовање домена и сцене, док линије 4, 5 и 6 декларишу променљиве. Непознатим променљивама Robosint додељује вредност. Типови променљивих су `Path`, `Location` и `Region` и оне одговарају типовима роботових путања, локацијама објеката и скупу могућих локација објеката, у овом случају, кухињској области или, још конкретније, површини. Линије 8-17 представљају тело главне функције. Оне се састоје од кода написаног на високом нивоу, а који представља акције које сваки план који доводи до коначног решења мора да испуни у наведеном редоследу.

Редослед извршавања главне функције представља окосницу рада. Подразумевано је да су акције `findPlace`, `pickup` и `place` унапред дефинисане у систему. Главна функција дефинише секвенцијално извршавање следећих акција:

1. Све док нису сви прљави судови смештени у машину за прање судова понављај следеће акције:
   (a) Пронађи место у машини `?loc` извршавањем акције `findPlace`. С обзиром на то да је ова локација непозната програмеру пре почетка извршавања програма, као и то да ће њена вредност за сваки појединачан прљави суд бити додељена посредством извршавања корака `findPlace`, у коду је препозната и означена као променљива и због тога има префикс ?.
   (b) Робот треба да прати непознату путању означену као `?path1`, а која води од робота до непознате кухињске области `?somewhere` у којој је изабрани објекат `?o` смештен. Потребно је да затим одатле покупи изабрани прљави суд, примењивањем акције `pickup`. За време овог корака, а на основу описа сцене, променљива `?somewhere` ће бити инстанцирана. Такође ће бити инстанцирана и путања `?path1` од тренутне позиције робота до површине на којој је изабрани суд смештен. То је могуће јер је путања између сваке две сигурне тачке робота задата дефиницијом домена. О сигурним тачкама робота ће бити више речи, јер се користе и у овој тези. Да би било јасније, важно је додати да је `?path1` означена као непозната јер зависи од тренутне позиције робота, суда који је тренутно изабран, као и домена.
   (c) Пратећи непознату путању `?path2`, а која зависи од позиције изабраног прљавог суда, а у чијем сигурном чвору је он тренутно смештен, сигурном чвору машине, као и домена, потребно је да

робот дође до машине и посредством акције place смести прљави суд у њу, на изабрано место `?loc`.

2. Када су сви прљави судови смештени у машину, покрени машину посредством акције `run`.
3. Када је машина завршила прање, потребно је сместити све судове у оставу, проласком кроз петљу која је скоро идентична као прва петља која служи за смештање прљавих судова у машину за прање судова.

4.3.3 Захтеви и догађаји

Захтеви су укључени у проблем у облику циља који се задаје планеру, и инваријанти које морају да испуњавају све изабране путање. У линији 17 је дефинисан захтев, и он говори да је за испуњење задатка неопходно да сви прљави судови који су дефинисани у оквиру сцене, а у коду чувани у скупу који се зове `DIRTY`, морају да буду чисти и смештени у оставу. Пример неке од инваријанти су да, због санитарних или сигурносних разлога, неке области у кухињи морају да се заобилазе приликом извршења главног задатка, као и да минимално растојање између области мора да постоји, како би робот могао несметано да се креће без ризика да на неки начин додирне или обори храну која је ту смештена. Такође, може се захтевати и да се због постојања протокола за чишћење просторије области обилазе неким дефинисаним редоследом. Захтев који настаје као последица капацитета батерије робота може бити да укупна дужина путање буде испуњена и слично. Аутори су препознали два захтева. Први се односи на дужину путање која не сме бити већа од 10 јединица мере. Други забрањује да робот пролази кроз област припремања хране и подразумева да такве путање одмах буду искључене из скупа могућих путања. Предност је дата SMT решавачу у односу на SAT решавач, због тога што он подржава линеарну аритметику као и функције које су неопходне за инваријанту.

Захтеве је могуће развијати даље. Аутори су понудили и синтаксу која се користи за сценарије који укључују изузетке (енг. *exceptional scenarios*). Они омогућавају да се прекине тачно дефинисан редослед извршавања акција и да се, у том изузетном случају, заобиђе контура плана. Један пример би био када се на путу до прљавог суда налазе неке препреке, које је потребно пре његовог узимања сместити на неке сигурне локације. Природан начин решавања оваквих изузетака је да се пре и после акције покрене управљач догађајима

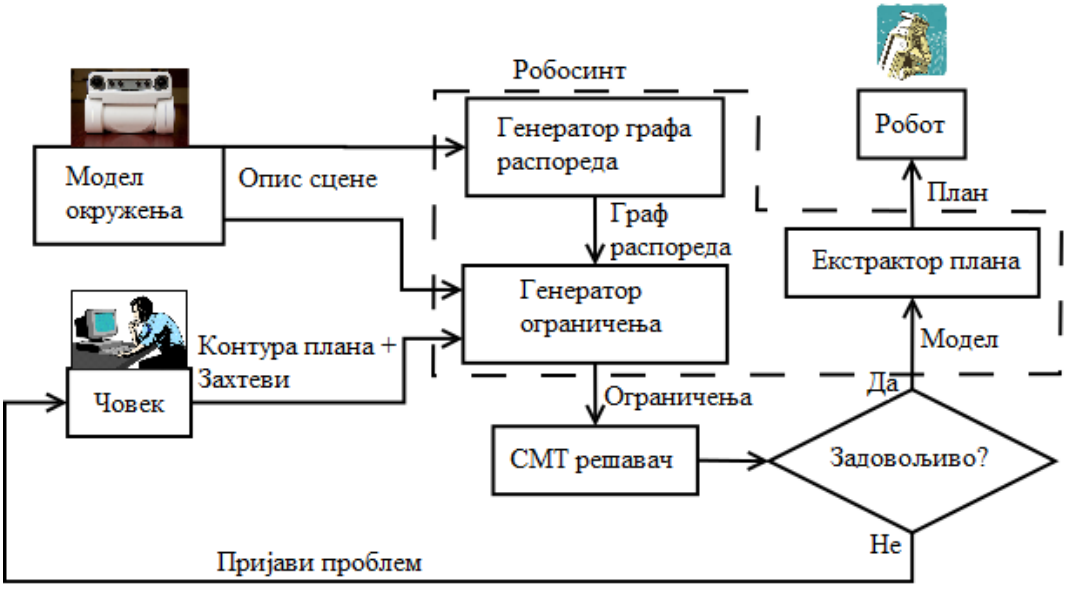
Алгоритам 2: Управљање акцијом pickup

Резултат : Не постоје препреке које блокирају изабрани прљави суд *obj*

```
1 prehandler for pickup(obj,rgn,_)
2 begin
3     while obstruct(?obst, obj) do
4         pickup(?obst, ?rgn, pathR)
5         place(?obst, ?tempR, ?pathR)
6     end
7 end
```

(енг. *event handler*). Програмер дефинише како ће се систем опходити према појави изузетака и какве акције за корекцију је неопходно извести. Пример који аутори предлажу је описан псеудокодом 2. Он подразумева да се на појаву изузетка реагује тако што се покупе све препреке и сместе се на сигурно место које је систем пронашао за њих.

4.3.4 Синтеза плана



Слика 4.2: Архитектура система Robosint

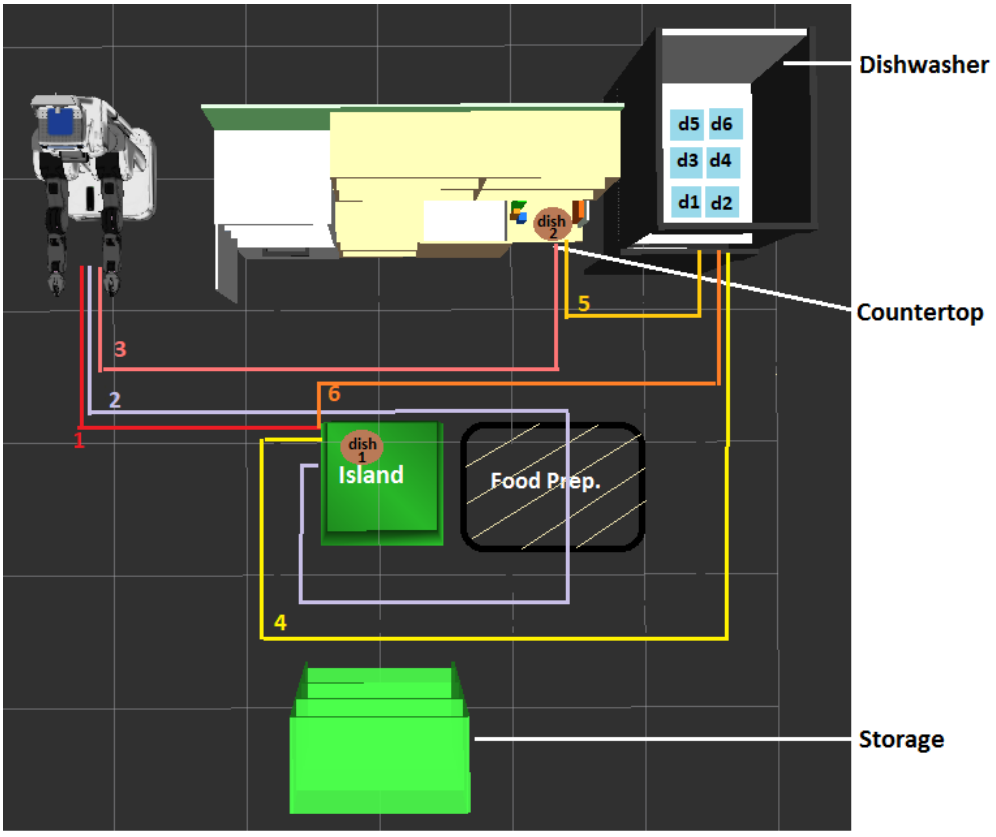
Опис сцене, контура плана и захтеви, који су претходно описани, представљају улазе који се прослеђују систему Robosint. Начин на који се они користе је приказан на слици 4.2. Robosint на основу описа сцене прави граф распореда (енг. *placement graph*) и касније га користи за инстанцирање непознатих променљивих из контуре плана тако да сви захтеви који су му

прослеђени буду испуњени. Пример би било инстанцирање променљивих `?loc` и `?path1` из контуре плана описаног псеудокодом 1. Ако Robosint не успе да нађе план који задовољава све услове, пријавиће да је проблем незадовољив.

Граф распореда представља везу између решавача који је дискретан и планера кретања. Он у себи носи информације о могућим преласцима из једног конфигурационог стања у друго. Важно је скренути пажњу на то да када је једном додељена вредност непознатим променљивама, контура плана може бити инстанцирана и могуће је на основу графа распореда добити тачан низ изабраних путања. Добијени резултат, који представља потпун план, је детерминистички низ корака за робота.

Пример 4.3.1. Модел окружења је приказан на слици 4.3, при чему `dish1` и `dish2` представљају прљаве судове које је потребно сместити у машину за прање судова, а `d1...d6` представљају места у машини за прање судова. На слици се виде позиције површина на које могу бити смештени прљави судови, и оне су означене са `Countertop` и `Island`, као и позиција оставе означена са `Storage` и машине за прање судова `Dishwasher`. Модел окружења садржи и информације о путањама које повезују те области, као и почетну позицију робота.

Генератор графа распореда са слике 4.2 би од датих путања направио граф распореда који би у себи носио информације о томе које путање постоје и које тачке оне повезују. Те информације би се проследиле *генератору ограничења*, заједно са *захтевима* који морају да буду испуњени, као и *контуром плана* описане псеудокодом 1. Захтеви су да робот не пролази кроз област припремања хране означену са `Food prep.`, као и да је укупна дужина пређеног пута највише 10. Одатле се јасно види да употреба путање означена бројем 2 никада неће бити део плана. Генератор ограничења од свих прослеђених информација прави формуле које прослеђује SMT решавачу. Уколико је он препознао да је формула задовољива, читањем из модела и посредством *екстрактора плана* је могуће добити вредности за непознате променљиве из контуре плана у циљу конструисања плана и прослеђивања плана роботу. За пример са слике 4.3 би први део плана, добијен првим проласком кроз петљу алгоритма описаног псеудокодом 1 тражио начин да се суд `dish1` смести у машину. Добијени план би, на пример, у том кораку инстанцирао `?somewhere` вредношћу `island`, `?path1` као путању која је на слици означена бројем 1, а путању `?path2` као путању која је на слици означена бројем 6, при чему би изабрано место у машини, `?loc`, било инстанцирано местом у машини `d1`.



Слика 4.3: Пример модела окружења

5. Архитектура пројектованог система

Систем описан у тези се темељи на принципима интегрисања планера редоследа извршавања задатака и планера кретања. Бави се налажењем редоследа акција, којим ће робот који представља помоћника у извршењу неке спољне акције, у складу са геометријским ограничењима добијеним посредством планера кретања, успешно извршити свој задатак. Централну улогу има робот који представља помоћника у извршењу неке спољне акције. Акција је названа спољна, јер је потребно нагласити да за њено извршавање није одговоран систем који се пројектује. У зависности од контекста, спољне акције омогућавају широк спектар примена система. Уколико се ради о индустријској примени, спољна акција може бити обрада предмета неком индустријском машином, фарбање дела, паковање производа у амбалажу и слично. Ту су и друге, свакодневне примене. На пример, у контексту помоћника у кући акција може бити прање веша или судова [52, 73]. Главни задатак робота је да узме сваки објекат који се налази на некој од задатих површина и смести га на унапред дефинисану циљну област. Након тога, спољна акција се извршава. По завршетку акције, робот предмете које је претходно сместио на циљну површину премешта у нову област, која може бити складиште, камион, остава или циљна област за примену неке друге спољне акције.

Проблем којим се теза бави представља уопштење проблема кухињског робота, који је први пут препознат у раду који је описан у поглављу 4.2, а чије су идеје потом развијане и проширене у раду који је описан у поглављу 4.3. Као што је већ речено, овај проблем комбинованог планирања се са контекста кухињског робота, који пакује судове у машину, може проширити на обављање било којих акција код којих је потребно спаковати предмете на место где се обрађују, а након тога пребацити на место одакле почиње примена нове спољне акције над њима и то представља проблем којим се теза бави. Планирање редоследа извршавања задатака је дискретно и захтева комбинаторно испитивање простора могућих планова. Планер врши претрагу кроз простор који је експоненцијалне величине у односу на број акција које је

неопходно извршити да би се дошло до унапред дефинисаног циља. С друге стране, планер кретања је одговоран за налажење плана у континуалном простору, при чему је он P комплетан у односу на број степени слободе робота [10]. Комбиновање оваква два проблема није тривијално. Код једноставног хијерархијског комбиновања, планер редоследа извршавања задатака ради са апстракцијом и прослеђује своје решење планеру кретања. Такав приступ доводи до жртвовања потпуности или захтева велико време извршавања због употребе бектрекинга. Због тога је идеја да се комбиновање имплементира кроз приступ *прво планирање редоследа извршавања задатака, затим планирање кретања - итеративно*. Иако је и у имплементираном решењу присутан бектрекинг, његов утицај није велики због тога што је једино могуће враћање малог броја једноставних акција уназад.

5.1 Планирање на ниском нивоу

Улази у систем, који описују физичко окружење робота и спецификују окружење за инстанцу на којој се тренутно ради, слично као и у другим радовима који се баве планирањем кретања, као и рада описаног у поглављу 4.3, подељени су на *елементе сцене* и *елементе домена*. У овом случају то значи да сталне препреке, површине на које могу бити смештени предмети и путање између површина припадају елементима домена, док елементима сцене припадају позиција робота и позиције предмета над којима он извршава акције.

За планирање на ниском нивоу се користи *граф распореда*, описан у поглављу 4.1.1, који информације о окружењу добија у облику прослеђених елемената домена и сцене. Гране графа које представљају везу између с-тачака и б-тачака, у практичном смислу говоре о томе да када се робот налази у наведеној б-тачки он из ње може да дохвати објекат који је смештен на месту наведене с-тачке или да смести објекат на наведену сигурну позицију с. Гране које представљају међусобне везе између базних тачака у физичком свету представљају могуће путање робота. Скуп свих грана које међусобно спајају б-тачке дефинише простор могућих путања. Ова теза уводи и додатну оптимизацију у виду употребе Флојд-Варшаловог алгоритма (енг. *Floyd-Warshall Algorithm*) у фази иницијализације домена, која налази најкраће путеве између сваког пара б-тачака, ако они постоје, што додатно доприноси ефикасности целог решења.

Важно је још поменути и да имплементирани систем интерно чува информације о томе где се сваки од објеката налази приликом сваке сцене, као и локацију робота, али и информацију о томе када је објекат потребно покупити или спустити.

5.2 Дискретни ниво

Дискретни ниво се ослања на идеје описане у поглављу 4.3. Акције робота *покупи* (енг. *pickup*) и *смести* (енг. *place*), које систем користи приликом планирања, дају предност спецификацији локација у облику региона (енг. *regions*), односно површина. Граф распореда служи за информисање система о б и с тачкама који припадају овим регионима. Информације о путањама добијене посредством графа распореда се користе за проналажење путања без колизија у континуалном простору, које испуњавају додатне услове о исправним плановима. У складу са тим, у *фази конструкције ограничења* заједно са ограничењима која носе информацију о путањама учествују и ограничења која описују неопходне услове да би сваки од корака био извршен. Ти кораци су формално описани *семантиком најслабијег предуслова*. Поред тога, у фазу конструкције ограничења могу ући и додатни захтеви. Слично као код система Robosint, и у овде конструисаном систему један захтев је ограничење максималне пређене путање робота, у циљу уштеде енергије коју робот троши. Задавање контуре плана на језику високог нивоа, описано у потпоглављу 4.3.2, није имплементирано. Међутим, идеја је искоришћена приликом избора корака, њиховог редоследа и одабира места за позивање SMT решавача, као и касније приликом дефинисања услова који треба да важе у сваком кораку. Конструкција корака се ослањала на пример контуре описан псеудокодом 1 и тиме је цео конструисан алгоритам усмераван ка исправним плановима, у циљу редукције простора могућих планова.

5.2.1 Семантика најслабијег предуслова као начин описивања корака

Коришћењем семантике најслабијег предуслова биће дефинисани појединачни кораци које је неопходно да систем секвенцијално извршава, као и детаљи имплементације задавањем услова SMT решавачу. Приликом описа саме имплементације главни проблем ће бити подељен на два потпроблема

- *паковање предмета у машину за обраду* и *смештање обрађених предмета у складиште*. Важно је напоменути да је приликом дизајнирања корака предвиђено да сваки предмет буде обрађен тачно једном. То значи да ће за потребе првог корака сваки предмет бити највише једном спакован у машину за обраду. Такође, у овом кораку ће то бити његова једина промена места, односно није дозвољена промена његове позиције на неку с-тачку која не припада машини за обраду. Приликом избора места за његово смештање је могуће да оно блокира друга празна места у машини, или да је због избора мањег предмета касније немогуће сместити неки већи предмет. Због тога је веома важно да се у кораку одређивања додатних услова који описују како изгледа исправан план, што више таквих случајева покрије. Пример додатних услова који се прослеђују систему је да не сме да постоји већи предмет од изабраног предмета ван машине, као и да изабрано место не сме да блокира остала празна места у машини за обраду. Још једна врло важна ствар је да је једини начин да се окружење робота промени нека акција коју је он предузео. Другим речима, он може да мења своје место као и место предмета, а ниједна спољна промена окружења робота за време извршавања ових задатака није предвиђена.

5.2.1.1 Употреба семантике најслабијег предуслова у пројектованом систему

Задатак система на дискретном нивоу је да непознатим променљивама додели вредност на основу датог плана P, који представља низ поткорака. Додељена вредност је таква да извршавањем интегрисаног плана, добијеног замењивањем непознатих које су задате у плану P овим вредностима, полазећи од иницијалне сцене, сви захтеви плана буду испуњени. Да би ово било могуће, неопходно је да постоји јасно значење или семантика за план, као и то да се на основу плана могу добити вредности непознатих променљивих из корака. За опис корака и задавање смерница за њихову имплементацију се користи семантика најслабијег предуслова, као и у радовима чијим идејама се води ова мастер теза.

Семантика најслабијег предуслова, као што је раније објашњено, даје правила помоћу којих најслабији услов може бити израчунат за низ наредби, `if-then-else` наредбу, као и `while` петљу. У овој тези је семантика најслабијих услова коришћена као средство за формално описивање корака, док је додавање

ограничења SMT решавачу додато у коду. У наставку ће бити више речи о томе како је семантика најслабијих услова коришћена за потребе главног проблема ове тезе. Најслабији предуслов је дефинисан као формула логике првог реда без квантификатора.

Да би се дефинисала исправност интегрисаног плана, неопходно је увести неколико појмова. За почетак, потребно је претпоставити да је дат план P и захтев g који мора да буде испуњен на крају извршавања интегрисаног плана. Уведено је и стање s које представља додељивање вредности одговарајућих типова познатих и непознатих променљивих из P. Сада је потребно скренути пажњу на то да је план ништа друго, него инстанцирање променљивих из P вредностима доделе s, што се може означити као $P[s]$. Свака акција из $P[s]$ ефективно мења стање из ког полази. То се јавља као последица тога што робот својим акцијама мења локације објеката. Резултат је финално стање до ког се долази када су сви кораци извршени.

Дефиниција 5.2.1. Интегрисани план је *исправан* за циљ G ако његово извршење води до финалног стања у коме важи G. Тада најслабији предуслов из P за који важи G, записан као $wp(P, G)$, одговара највећем скупу стања $\{s\}$ за који је план $P[s]$ коректан.

Дефиниција 5.2.2. Нека формула I садржи ограничења над променљивама из P који описују иницијалну сцену. Тада формула $I \wedge wp(P, g)$ дефинише највећи скуп валуација непознатих из P таквих да план који одговара овим валуацијама одговара иницијалним захтевима дефинисаним у опису сцене, а уједно и испуњава задати циљ.

На основу претходне дефиниције 5.2.2 може се закључити да је за налажење задовољавајућих додела вредности непознатим променљивама из P неопходно проверити задовољивост најслабијег предуслова. У ту сврху коришћен је SMT решавач. Важно је још и скренути пажњу на то да је најслабији услов изведен у потпуности независно од почетног стања физичког простора, односно иницијалне сцене.

5.2.2 Реконструисање плана добијеног провером задовољивости препознатих ограничења

Након што су ограничења препозната, груписана су тако да сваком кораку који осликава акцију над предметом буду јасно додељена ограничења која је

неопходно укључити у процес провере задовољивости те специфичне акције. Испитивање задовољивости се врши секвенцијално за сваки корак у складу са ограничењима која су за њега препозната. Ово представља унапређење у односу на идеје из сличних радова, јер се код њих испитује задовољивост за све кораке одједном. У односу на такав приступ, овде је мањи број ограничења која учествују у свакој провери, па је и провера ефикаснија. С друге стране, уколико је планирање неке од акција незадовољиво, постоји потреба да робот врати сцену на стање на ком се налазила на почетку провере низа акција. У том случају је довољно да робот врати изабрани предмет на старо место, уколико га је већ покупио, и нема потребе да се он врати корак уназад, у чвор из ког је дошао, захваљујући примени Флојд-Варшаловог алгоритма за налажење најкраћих путева у графу пре почетка планирања. Он обезбеђује да сваки чвор, који је достижан из чвора из ког је робот дошао, буде достижан и из чвора у ком се робот тренутно налази. Такође, путања између чвора у који робот треба да оде је краћа, или у најгорем случају иста, као када би се робот вратио на своју претходну позицију, а затим отишао у нови чвор. У складу са тим, бектрекинг не би требало да утиче превише на перформансе. Уколико је појединачан корак задовољив, тада је могуће из модела који је добијен посредством SMT решавача добити информације о томе како план за тај корак изгледа. План се прослеђује роботу који извршава акцију у складу са њим. Када је сваки корак тог низа задовољив, може се сматрати да су све неопходне акције над једним предметом успешно испланиране. Планирање низа акција над појединачним предметом се понавља све док акције које робот извршава над свим предметима нису испланиране и извршене, или док је проблем постао незадовољив. У случају када је задовољење ограничења за сваки корак сваког предмета немогуће, систем ће пријавити да је цео проблем незадовољив. У супротном је могуће, на основу информација добијених из модела, реконструисати цео план који је робот извршио.

6. Имплементација система

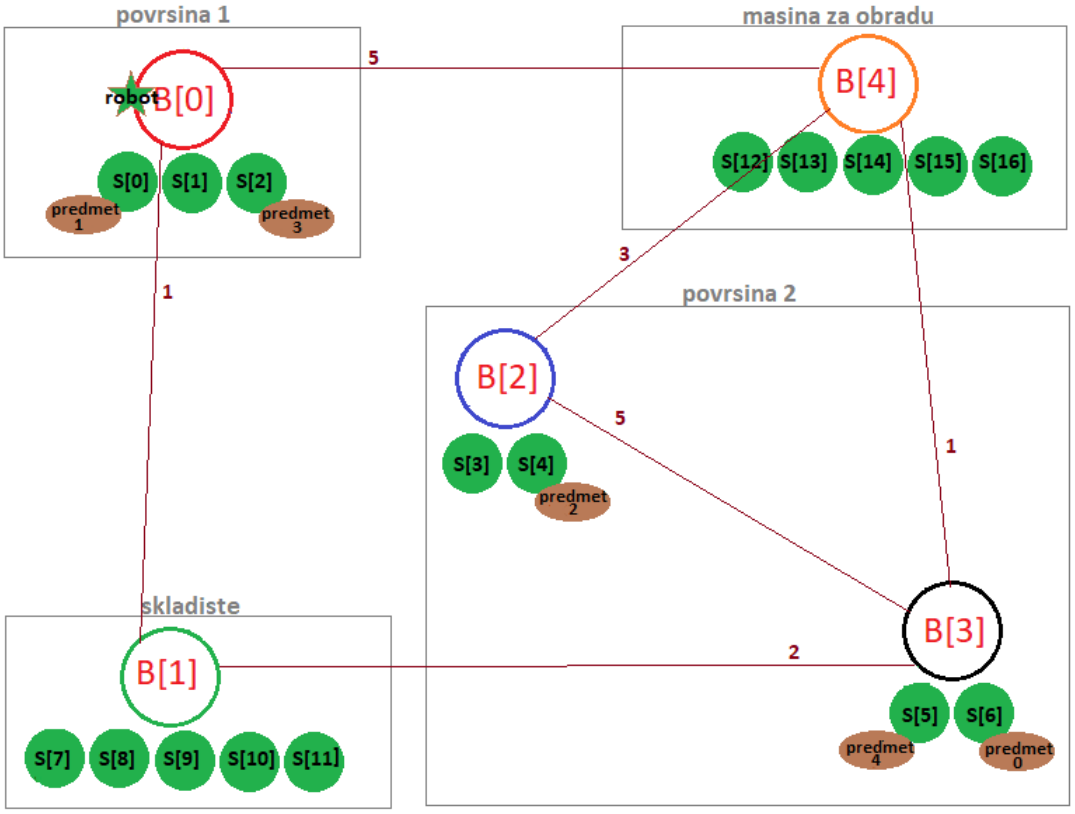
Први део главе се односи на начин на који је имплементирано планирање на ниском нивоу, које је одговорно за то да план буде у складу са геометријским особинама робота и окружења у коме он ради, као и особинама предмета над којима врши акције. Други део главе се односи на планирање високог нивоа, које је одговорно за налажење плана које је у складу са информацијама добијеним посредством планера кретања, препознатих логичких услова који описују систем и која дефинишу правила за редослед извршења корака у складу са тим како исправни планови треба да изгледају, као и додатних захтева. У наставку је описан начин на који је имплементирано исцртавање генерисаног плана. Мерене су и брзине планирања у зависности од почетних конфигурација робота, комплексности домена и броја предмета у сврху процењивања ефикасности и примењивости имплементираног алгоритма.

Приликом имплементације је коришћено развојно окружење `Qt Creator`. Систем је имплементиран у програмском језику `C++`, као SMT решавач је коришћен `Z3`, а за графичку репрезентацију генерисаног плана су коришћене `Qt` библиотеке. Код је доступан на линку https://github.com/aleksandradjuric/MasterRad.

6.1 Опис физичког окружења робота

Прослеђивање информација релевантних за опис физичког окружења робота је мануелно. То значи да се у систем једном, на почетку планирања, учитавају позиције робота и предмета над којима је потребно извршити манипулацију, локације препрека и површина на које могу бити смештени предмети, локације машине за обраду и складишта, као и путања које постоје, а којима се робот може кретати.

Опис физичког окружења се, у односу на то да ли се информације које у себи носи мењају од сцене до сцене или од почетка до краја извршења планирања остају непромењене, дели на две конфигурационе датотеке. На основу првог се систему прослеђују информације о елементима сцене, а на основу другог информације о елементима домена.



Слика 6.1: Граф распореда за проблем робота који помаже у извршењу спољне акције

6.1.1 Дефинисање домена

Домен у себи садржи информације о површинама, б-тачкама, с-тачкама, путањама између б-тачака и међусобним блокирањима с-тачака. Као што је раније поменуто, ове информације се учитавају једном из конфигурационе датотеке и њени елементи остају непромењени за време планирања.

6.1.1.1 Имплементација домена

Домен је у коду имплементиран класом `Domen` и у себи садржи елементе домена. Да би било могуће чување тих елемената, сваки од елемената има дефинисану класу која га описује. Због тога ће прво бити посвећена пажња сваком појединачном елементу и његовој класи, а тек затим ће бити дата декларација класе `Domen`.

Први елемент домена који ће бити описан је **површина** на коју се предмети могу смештати. За њу је једино карактеристичан назив површине,

па је њена класа декларисана као

```
class Povrsina
{
public:
    Povrsina(string _naziv);
    string uzmi_ime_povrsine();
private:
    string naziv;
};
```

Следећи елемент домена је **б-тачка**. Она представља место на ком робот може стајати тако да дохвати предмете смештене на неку површину. У складу са тим, за њу је битна информација о томе којој површини се из ње може приступити. Једној површини се може приступити из више б-тачака. Информација о томе се чува у облику индекса који одговара тој површини у низу свих површина. Њена класа је декларисана на следећи начин:

```
class B_tacka
{
public:
  B_tacka(int _indeks_povrsine);
  int uzmi_indeks_povrsine();
private:
  int indeks_povrsine;
};
```

Наредни елемент је **с-тачка**. Она означава тачно место на површини на коме се може налазити предмет. За њу је важно из које б-тачке јој је могуће приступити, па се информације о томе чувају као индекс који одговара тој б-тачки у низу свих б-тачака. Једној с-тачки се може приступити из тачно једне б-тачке, оне која јој је најближа. Декларација њене класе је:

```
class S_tacka
{
public:
  S_tacka(int _indeks_b_tacke);
  int uzmi_indeks_b_tacke();
private:
  int indeks_b_tacke;
};
```

Робот се креће по путањи састављеној од низа путања које повезују парове б-тачака. За **путању** је битно чувати индексе б-тачака које повезује и дужину те

путање. С обзиром на то да се на путање између сваког могућег пара б-тачака примењује Флојд-Варшалов алгоритам, којим се од оригинално учитаних путања добијају најкраће путање између сваког пара б-тачака, неопходно је чувати и низ б-тачака који је неопходно посетити да би се робот кретао том најкраћом путањом, идући од једне до друге б-тачке. У складу са тим, декларација класе која описује путању је:

```
class Putanja
{
public:
  Putanja(int _indeks_b_tacke_1, int _indeks_b_tacke_2, int _duzina);
  Putanja(int _indeks_b_tacke_1, int _indeks_b_tacke_2, int _duzina,
          vector<int>& _sekvenca_b_tacki);
  int uzmi_indeks_prve_b_tacke();
  int uzmi_indeks_druge_b_tacke();
  int uzmi_duzinu();
  vector<int> uzmi_sekvencu_b_tacki();
private:
  int indeks_b_tacke_1;
  int indeks_b_tacke_2;
  int duzina;
  vector<int> sekvenca_b_tacki; //putanja koju je Flojd-Varsal
  // nasao kao najkracu za ovaj par tacaka
};
```

Међусобна блокирања с-тачака немају своју класу, већ се информације о томе чувају у низу који чува парове индекса с-тачака. Овај низ чува информацију о сваком пару с-тачака код ког прва с-тачка пара блокира другу с-тачку.

Класа која представља домен чува информације о елементима домена и помоћу својих метода даје начин да се ти елементи користе у процесу планирања. Она је декларисана на следећи начин:

```
class Domen
{
public:
  Domen();
  ~Domen();
  int uzmi_duzinu_niza_povrsina();
  Povrsina uzmi_povrsinu(int indeks_povrsine);
  int uzmi_duzinu_niza_b_tacaka();
  B_tacka uzmi_b_tacku(int indeks_b_tacke);
  int uzmi_duzinu_niza_s_tacaka();
  S_tacka uzmi_s_tacku(int indeks_s_tacke);
```

```
  int uzmi_duzinu_niza_putanja();
  Putanja uzmi_putanju(int indeks_putanje);
  int uzmi_duzinu_niza_medjusobnih_blokiranja_s_tacaka();
  bool da_li_blokira(int indeks_s_tacke_1, int indeks_s_tacke_2);

  //neophodno za iscrtavanje domena
  void inicijalizuj_domen(string putanja_do_fajla);
  vector<int> uzmi_sekvencu_tacaka_za_par_b_tacaka(int indeks_b_tacke_1,
                                                   int indeks_b_tacke_2);
  void ocisti_domen();
  shared_ptr<Oblast_za_crtanje> oblast_za_crtanje;
private:
  vector<Povrsina> povrsine;
  vector<B_tacka> b_tacke;
  vector<S_tacka> s_tacke;
  vector<Putanja> putanje;
  vector<vector<bool>> medjusobna_blokiranja_s_tacaka;

  //neophodno za primenu Flojd-Varsalovog algoritma
  vector<vector<int>> matrica_prelaza;  //pomocna matrica
  void ispisi_matricu(vector<vector<int>>& matrica);
  void pronadji_sekvencu_tacaka(int i, int j,
                                vector<int>& sekvenca_tacaka);

  void nadji_najkrace_putanje_izmedju_b_tacaka();
  void inicijalizuj_medjusobna_blokiranja();

};
```

6.1.1.2 Учитавање домена

Учитавање домена је омогућено методом `inicijalizuj_domen()`. За пример са слике 6.1, пример конфигурационе датотеке на основу које се иницијализују елементи домена је:

```
Povrsine
povrsina_1 0 50
povrsina_2 330 330
masina_za_obradu 320 50
skladiste 0 340
B tacke
0 100 150
3 110 440
1 550 370
1 420 450
2 430 150
```

```
S tacke
0 60 210
0 100 210
0 140 210
2 530 430
2 570 430
3 400 510
3 440 510
1 30 500
1 70 500
1 110 500
1 150 500
1 190 500
4 350 210
4 390 210
4 430 210
4 470 210
4 510 210
Putanje
0 1 1
0 4 5
1 3 2
2 3 5
2 4 3
3 4 1
Medjusobna blokiranja S tacaka
12 15
13 16
```

Први део информација из датотеке, означен као `Povrsine`, у линијама које следе иза те ознаке, прослеђује називе површина којима је потребно иницијализовати сваку од површина која се претходно креира, као и координате неопходне за њено исцртавање приликом приказа креираног плана, а потом их додаје у низ `povrsine`. Други део информација, означен као `B tacke`, прослеђује систему информације о индексима површине којој свака појединачна б-тачка приступа и координате неопходне за њихово исцртавање. Након креирања и иницијализације сваке б-тачке прослеђеним информацијама, неопходно их је додати у низ `b_tacke`. Слично, редови који следе иза ознаке `S tacke`, за сваку с-тачку коју је неопходно креирати и иницијализовати, а затим и додати у низ `s_tacke`, носе информацију о индексу б-тачке којој та с-тачка припада и њене координате. Линије које следе иза ознаке `Putanje`, систему у облику тројки прослеђују, редом, информације о индексу прве б-тачке, индексу друге б-тачке коју путања спаја и дужини те путање. Ове информације се користе за креирање и иницијализацију сваке појединачне прослеђене путање, а затим се

додају у низ `putanje`. Линије после ознаке `Medjusobna blokiranja S tacaka`, у облику парова прослеђују индексе којим се иницијализује сваки елемент низа `medjusobna_blokiranja_s_tacaka`. У овом низу, први број сваког елемента представља индекс с-тачке која блокира, а други елемент пара представља индекс с-тачке која је блокирана.

Након учитавања информација из конфигурационе датотеке за домен и иницијализације елемената домена, над прослеђеним путањама се позива Флојд-Варшалов алгоритам. Његова улога је да пронађе најкраће путање између сваког пара тачака, које су му прослеђене заједно са директним путањама које их повезују. Уколико постоји више директних путања између истог пара тачака, у матрицу ће пре позива Флојд-Варшаловог алогоритма бити уписана најкраћа директна путања. Алгоритам је описан псеудокодом 3. Као резултат извршења алгоритма, резултујућа матрица путања ће садржати дужине најкраћег пута између сваке две тачке. Коришћењем информација из матрице прелаза могуће је добити секвенцу тачака које је неопходно посетити да би се робот кретао по том најкраћем путу.

Алгоритам 3: Флојд-Варшалов алгоритам

Улаз : Почетна матрица путања и дужина низа б-тачака
Излаз : Резултујућа матрица путања која садржи најкраће путање између сваке две тачке и матрица прелаза

```
1  rezultujuca_matrica_putanja := pocetna_matrica_putanja
2  for k ← 0 to duzina_niza_b_tacaka do
3      for i ← 0 to duzina_niza_b_tacaka do
4          for j ← 0 to duzina_niza_b_tacaka do
5              if rezultujuca_matrica_putanja [i][j] >
                (rezultujuca_matrica_putanja [i][k] +
                rezultujuca_matrica_putanja [k][j]) and
                rezultujuca_matrica_putanja [i][k] ≠ INF and
                rezultujuca_matrica_putanja [k][j] ≠ INF then
6                  rezultujuca_matrica_putanja [i][j] :=
                    rezultujuca_matrica_putanja [i][k] +
                    rezultujuca_matrica_putanja [i][k]
7                  matrica_prelaza[i][j] := matrica_prelaza[k][j]
8              end
9          end
10     end
11 end
```

6.1.2 Дефинисање сцене

Сцена садржи информације о позицијама предмета над којима је потребно применити акције, као и о тренутној позицији робота. Након сваке промене позиције робота или предмета, стање елемената сцене је промењено.

Информације о сцени се учитавају једном на почетку планирања и они представљају почетну конфигурацију елемената сцене.

6.1.2.1 Имплементација сцене

Сцена је у коду имплементирана класом `Scena` и чува стање елемената сцене. Први елемент представља **тренутну позицију робота** и чува индекс б-тачке у којој се робот тренутно налази. Други елемент сцене описује **предмет**. Да би стање сваког предмета било комплетно, неопходно је да садржи информације о позицији, величини, томе да ли је већ обрађен или чека обраду и томе да ли га робот тренутно држи. Класа `Predmet` је декларисана на следећи начин:

```
class Predmet
{
public:
    Predmet(int _pozicija, int _velicina, bool _ceka_obradu,
            bool _robot_drzi_predmet);
    int uzmi_poziciju();
    void promeni_poziciju(int indeks_s_cvora);
    int uzmi_velicinu();
    bool da_li_ceka_obradu();
    void promeni_status_obrade(bool status);
    bool da_li_robot_drzi_predmet();
    void uhvati_predmet();
    void spusti_predmet();
private:
    int pozicija; //indeks s tacke u kojoj je predmet smesten
    int velicina;
    bool ceka_obradu;
    bool robot_drzi_predmet;
};
```

Класа `Scena` која чува низ предмета и позицију робота је декларисана као:

```
class Scena
{
  Scena();
  ~Scena();
  bool da_li_je_obrada_u_toku();
  void zapocni_obradu();
  void zavrsi_obradu();
  int uzmi_trenutnu_poziciju_robota();
  void promeni_trenutnu_poziciju_robota(int indeks_b_cvora);
```

```
  Predmet& uzmi_predmet(int indeks_predmeta);
  int uzmi_duzinu_niza_predmeta();
  void inicijalizuj_scenu(string putanja_do_fajla);
  //koristi se za iscrtavanje scene
  void dodaj_stanje_scene(int pozicija);
  void nacrtaj_plan(int trenutni_frejm);
  void nacrtaj_stanje_scene(int trenutni_frejm);
  vector<std::pair<int, bool> > uzmi_pozicije_predmeta(vector<Predmet>&
                                                       stanje_predmeta);
  void ocisti_scenu();
  int uzmi_duzinu_stanja_scene();
  void promeni_stanje_animacije(bool stanje);
  shared_ptr<Oblast_za_crtanje> oblast_za_crtanje;
private:
  vector<Predmet> predmeti;
  //sluzi za iscrtavanje plana
  vector<std::pair<int, vector<Predmet>>> stanja_scene;
  bool obrada_u_toku;
  int trenutna_pozicija_robota; //indeks b tacke u kojoj se robot nalazi
  bool animacija_u_toku;
signals:
  void on_azuriraj_slajderSignal(int);
};
```

6.1.2.2 Учитавање сцене

Учитавање сцене је могуће коришћењем методе `inicijalizuj_scenu()`. За пример са слике 6.1, пример конфигурационе датотеке коришћене за иницијализацију сцене је:

```
Pocetna pozicija robota
0
Predmeti
6 6 true false
0 4 true false
4 3 true false
2 7 true false
5 5 true false
```

Први број у датотеци, означен линијом `Pocetna pozicija robota` носи информацију о индексу б-тачке у којој се робот иницијално налази. Наредни део датотеке, испод линије `Predmeti`, садржи четворке коришћене за иницијализацију сваког предмета. Први број означава индекс с-тачке у којој је предмет смештен на почетку планирања, други број говори о величини

предмета, затим следе редом информације о томе да ли предмет чека обраду и да ли га робот држи.

6.2 Дефинисање корака

Кораци представљају препознате акције које је неопходно да робот секвенцијално изврши да би успешно испунио свој циљ. Да би се план успешно описао, неопходно је препознати услове који морају да буду испуњени за сваки појединачан корак, да би његово извршење било могуће. У те услове спадају како логичка ограничења и захтеви, тако и ограничења над путањама, настала као последица физичког описа сцене. Имплементација корака се налази у методи `generisi_plan` класе `Planiranje`. У наставку ће бити посвећена пажња сваком препознатом кораку и биће објашњени неки од сегмената кода из те методе, који се користе за планирање.

За проверу задовољивости препознатих услова комбинује се испитивање задовољивости у односу на теорију неинтерпретираних фукнција и теорију линеарне аритметике. Задовољивост услова описаних у сваком од корака се врши у односу на теорију неинтерпретираних функција. С друге стране, за време планирања, на крају сваког корака се, уколико се робот померио, ажурира дужина до тада пређеног пута. У сваком кораку се додаје услов да пређени пут, заједно са новом путањом коју је потребно прећи у том кораку, не сме да буде дужа од унапред дефинисане максималне пређене дужине. Испитивање задовољивости овог услова се врши у односу на теорију линеарне аритметике. Овај услов је додат да би се илустровало да се на лак начин могу у процес планирања укључити и аритметичка ограничења.

6.2.1 Паковање предмета у машину за обраду

Имплементација овог корака у тези је описана у алгоритму 4. Приликом паковања предмета у машину за обраду препознат је низ корака који је неопходно извршавати све док постоје необрађени предмети који су ван машине:

1. *нађи место* за изабрани предмет
2. *покупи* изабрани предмет
3. *смести* изабрани предмет на пронађено место

Избор предмета и место за његово смештање су препуштени SMT решавачу, који, на основу прослеђених услова прави задовољавајућу валуацију, тј. модел.

Примери услова, као и начини њиховог прослеђивања решавачу су дати у опису корака. Уколико је немогуће сместити све предмете у машину за обраду решавач ће препознати да је проблем незадовољив и пријавити неуспех.

SMT решавач се позива за сваки корак посебно. С обзиром на то да је тиме смањен број услова приликом сваког позива, то утиче и на ефикасност сваког позива јер је смањен број формула које учествују у испитивању, па самим тим и простор претраге. То практично значи да ће први корак изабрати предмет и место за његово смештање. Информације о одабраном предмету и месту се добијају читањем из модела. Оне се прослеђују, као додатни услов, следећим корацима. Уколико се, међутим, деси да је неки од следећа два корака незадовољив, могуће је да је то последица избора који је направио први корак. Због тога се користи *бектрекинг*. То значи да се, уколико се деси да су кораци *покупи* или *смести* незадовољиви при тренутном избору места и предмета, додаје услов да комбинација којом је представљен избор не сме бити тако инстанцирана. Затим се покушава поново, од корака *нађи место*. С обзиром на то да кораци мењају стање система, конкретно позицију предмета и позицију робота, неопходно је да, уколико је потребан бектрекинг, систем буде враћен на стање које је било пре почетка првог корака. Тек када први корак постане незадовољив и цео проблем постаје незадовољив.

До завршетка дела проблема који се бави паковањем предмета у машину за обраду, односно до изласка из главне петље описане у алгоритму 4, може доћи на два начина:

1. Када више није испуњен $uslov\_petlje$, односно када нема више предмета ван машине. То представља **успешан исход** и у том случају има смисла наставити извршавање програма. Систем ће спољној акцији пријавити да је први корак успешан и да она може да почне своје извршење.
2. Када није могуће задовољити први корак и тада је проблем паковања предмета у машину **незадовољив**, нема смисла наставити са извршавањем и потребно је пријавити да је цео проблем, за задату иницијалну сцену, незадовољив.

Низ корака је задовољив уколико је решавач пронашао да је решење трећег корака задовољиво. То је последица чињенице да трећи корак и његова задовољивост зависе од резултата претходна два корака. У том случају је неопходно информацију о изабраном предмету и изабраном месту за тај корак сачувати, у циљу приказа свих акција робота, добијених планирањем, на крају извршења алгоритма.

Алгоритам 4: Паковање предмета у машину за обраду

Резултат: Сви предмети који су чекали обраду су спаковани у машину

```
1  додај информације о путањама између б тачака
2  додај информације о томе које с-тачке блокирају друге с-тачке
3  додај информације о односу величине предмета
4  uslov_petlje := провери да ли постоје предмети ван машине
5  dodatni_uslov := true; prvi_korak_zadovoljiv := true
6  while uslov_petlje and prvi_korak_zadovoljiv do
7      додај информације о томе који предмети чекају обраду
8      додај услове за корак нађи место
9      додај услов dodatni_uslov // омогућава бектрекинг
10     prvi_korak_zadovoljiv := провери задовољивост
11     if !(prvi_korak_zadovoljiv) then
12         break
13     else
14         izabrani_predmet, izabrano_mesto := узми вредности из добијеног
             модела
15     end
16     if prvi_korak_zadovoljiv then
17         додај услове за корак покупи
18         drugi_korak_zadovoljiv := провери задовољивост
19         if !(drugi_korak_zadovoljiv) then
20             dodatni_uslov := dodatni_uslov and (изабрано место различито
                 од izabrano_mesto or изабрани предмет различит од
                 izabrani_predmet)
21         else
22             промени позицију робота на б-тачку изабраног предмета
23             промени стање робота тако да држи izabrani_predmet
24         end
25     end
26     if drugi_korak_zadovoljiv then
27         додај услове за корак смести
28         treci_korak_zadovoljiv := провери задовољивост
29         if !(treci_korak_zadovoljiv) then
30             dodatni_uslov := dodatni_uslov and (изабрано место различито
                 од izabrano_mesto or изабрани предмет различит од
                 izabrani_predmet)
31             врати предмет у с-тачку у ком је био пре корака покупи
32             промени стање робота тако да више не држи изабрани предмет
33             врати робота у б-тачку у ком је био пре корака покупи
34         else
35             промени позицију робота на б-тачку машине
36             промени с-тачку изабраног предмета на с-тачку изабраног места
37             промени стање робота тако да више не држи изабрани предмет
38             dodatni_uslov := true
39         end
40     end
41     uslov_petlje := провери да ли постоје предмети ван машине
42 end
43 if prvi_korak_zadovoljiv then
44     пријави да је проблем паковања предмета у машину задовољив
45 else
46     пријави да је проблем паковања предмета у машину незадовољив
47 end
```

6.2.1.1 Корак *нађи место*

Задатак овог корака је да пронађе место за изабрани предмет. Потребни услови да би се он успешно извршио су:

- неопходно је да изабрани предмет чека обраду
- машина за обраду не сме да буде пуна
- изабрано место мора да припада машини за обраду
- изабрано место треба да буде празно
- изабрани предмет не сме да буде смештен у машину за обраду
- изабрано место у машини не сме да блокира остала празна места
- не сме да постоји већи предмет који чека обраду ван машине

Препознати услови помажу да се корак опише семантиком најслабијег предуслова. За њене потребе, овај корак је описан акцијом $findPlace(?loc1, Machine, ?o)$. Циљ g из дефиниције 5.2.1 је да је пронађена $?loc$, као и $?o$. У овом кораку се проналази адекватно место у машини за обраду, као и избор предмета. Предуслов $wp(P, g)$ представља скуп стања описаних следећим условима:

$$\begin{aligned}
&\exists loc1.loc1 \in Machine \wedge \\
&\nexists o'.isLocEqual(loc(o'), loc1) \wedge \\
&isDirty(o) \wedge \\
&\nexists o'.blocks(loc1, loc2) \wedge loc2 \in Machine \wedge isLocEqual(loc(o'), loc2) \wedge \\
&\nexists o'.isBigger(o', o) \wedge loc(o') \notin Machine \wedge \\
&loc(o) \notin Machine
\end{aligned} \tag{6.1}$$

Овај предуслов говори да је неопходно да постоји место у машини за обраду $loc1$, у коме се не налази ниједан објекат o'. Ако постоји такво место, очигледно ни машина није пуна, па је додавање тог додатног услова сувишно, иако се логично намеће као још једна ствар о којој треба водити рачуна. Такође, захтева се да изабрани предмет o чека обраду и да не постоји ниједна непопуњена локација $loc2$ која припада машини за обраду, а коју изабрано место $loc1$ блокира. Ту је и услов да не постоји већи предмет који већ није смештен у машину и он се јавља као последица физичког ограничења машине. Неопходно је и да се предмет o, који је одабран, не налази у машини.

Претходно дефинисани предуслов је имплементиран прослеђивањем услова, записаних као логичке формуле, SMT решавачу. У наставку је

приказана имплементација једног сегмента овог предуслова, који решавачу прослеђује услов да изабрано место у машини не сме да блокира остала празна места. Да би било могуће проследити решавачу овај услов, неопходно је прво проследити информације о стању система. У складу са тим, први део приказаног кода представља прослеђивање информације о томе које с-тачке блокирају друге с-тачке. Она се прослеђује само једном, на почетку планирања, јер зависи од елемената домена, који су непромењиви за време планирања. Затим следи део кода који је везан за елементе сцене, који су променљиви, па је неопходно да се понавља за сваки предмет. У том делу се из система извлаче информације о томе које с-тачке су слободне. На основу тога се у последњем делу приказаног кода, који припада имплементацији корака *наћи место*, прави формула којом се решавачу прослеђује услов да изабрано место у машини не сме да блокира остала празна места:

```
/* informacije dodate jednom na pocetku izvrsavanja */
// dodavanje informacija o tome koje s tacke blokiraju jedne druge
expr informacije_o_blokiranju = boolean_true;
for(i=0; i<duzina_niza_s_tacaka; i++)
  for(j=0; j<duzina_niza_s_tacaka; j++)
    if(domen.da_li_blokira(i, j))
      informacije_o_blokiranju = informacije_o_blokiranju &&
        (blokira(robot_kontekst.int_val(i),
                 robot_kontekst.int_val(j)) == boolean_true);
robot_resavac.add(informacije_o_blokiranju);

/* informacije koje zavise od elemenata scene */
//niz koji cuva indeks s tacaka slobodnih mesta u masini za obradu
vector<int> slobodna_mesta_u_masini;

for(i=0; i<duzina_niza_s_tacaka_masine; i++){
  cvor_je_zauzet = false;
  for(j=0; j<duzina_niza_predmeta; j++)
    if(scena.uzmi_predmet(j).uzmi_poziciju() == s_tacke_masine[i])
    {
      cvor_je_zauzet = true;
      break;
    }
    //ako nijedan predmet nije u toj s tacki, a ona pripada masini
    // za obradu, tada ta s tacka nije zauzeta
    if(cvor_je_zauzet == false)
      slobodna_mesta_u_masini.push_back(s_tacke_masine[i]);
}
int duzina_niza_slobodnih_mesta_u_masini =
    slobodna_mesta_u_masini.size();
```

```
/* uslovi za korak "nadji mesto" */
//izabrano mesto u masini za obradu ne treba da blokira ostala
//prazna mesta u masini
expr izabrano_mesto_ne_blokira_prazno_mesto = boolean_true;
for(i=0; i<duzina_niza_slobodnih_mesta_u_masini; i++){
  izabrano_mesto_ne_blokira_prazno_mesto =
    izabrano_mesto_ne_blokira_prazno_mesto &&
    (blokira(izabrano_mesto_u_masini,
    robot_kontekst.int_val(slobodna_mesta_u_masini[i]))
    == boolean_false);
}
```

Остали сегменти предуслова за овај корак су имплементирани на сличан начин.

На крају извршења испитивања задовољивости овог корака могућа су следећа два исхода:

1. **Корак *нађи место* је задовољив**. Тада је потребно из добијеног модела прочитати вредности за изабрани предмет o, који је у коду означен као `izabrani_predmet`, и одабрано место за његово смештање $loc1$, у коду означено као `izabrano_mesto_u_masini`.
2. **Корак *нађи место* није задовољив**. У том случају, као што је претходно речено, цео проблем паковања предмета у машину је незадовољив.

6.2.1.2 Корак *покупи*

Овај корак има задатак да покупи изабрани предмет са локације где се налази. То подразумева да робот дође до б-тачке из које има приступ с-тачки у којој је смештен изабрани предмет и одатле га покупи. Услови који морају да важе су:

- неопходно је да изабрани предмет и изабрано место имају вредност добијену читањем модела из корака *нађи место*
- робот не сме да држи ниједан објекат
- мора да постоји путања од тренутне позиције робота до б-тачке изабраног предмета

У складу са препознатим условима, корак *покупи* је, за потребе описа помоћу семантике најслабијег предуслова, описан акцијом $pickup(o, Surface, ?path1)$. Тада је g из дефиниције 5.2.1 $holding(o)$, што значи да је циљ да након извршења овог корака робот држи објекат o. Додатно, робот ће променити своју позицију на bpt која одговара изабраном предмету, што се може записати као

$isEqualLoc(CURR, bpt)$. Предуслов $wp(P, g)$ представља скуп стања описаних следећим условима:

$$\begin{aligned} &\exists bpt.loc(o) \in reachOf(bpt) \wedge \\ &\nexists o'.blocks(loc(o'), loc(o)) \wedge \\ &path(path1, CURR, bpt) \wedge \\ &\nexists o'.holding(o') \end{aligned} \tag{6.2}$$

Овај предуслов говори да је неопходно да постоји б-тачка bpt из које робот може дохватити место које представља локацију изабраног предмета. Такође говори и да не сме постојати ниједан предмет o' који блокира изабрани предмет, као и да мора постојати путања $path1$ дефинисана у графу распореда која повезује тренутну б-тачку робота $CURR$ и bpt. Последњи део услова је да робот не сме држати ниједан предмет у том тренутку.

У наставку ће бити приказан део кода који описује прослеђивање услова SMT решавачу, који формулом описује услов да мора да постоји путања од тренутне позиције робота до б-тачке изабраног предмета. Информације о томе између којих б-тачака постоје путање, као и томе колика је њихова дужина, су SMT решавачу прослеђени једном на почетку планирања, јер зависе од елемената домена. На овај услов утиче и б-тачка изабраног места, па је неопходно приликом имплементације корака *покупи* решавачу проследити информације о томе које место је претходни корак изабрао. Тек тада има смисла решавачу проследити услов да је неопходно да путања између поменутих б-тачака постоји. Код који описује овај сегмент предуслова корака *покупи* је:

```
/* informacije dodate jednom na pocetku izvrsavanja */
expr uslov_za_sve_putanje = boolean_false;
int duzina_niza_putanja = domen.uzmi_duzinu_niza_putanja();
for(i=0; i<duzina_niza_putanja; i++)
{
  expr uslov_za_putanju1 = putanja(trenutna_pozicija, b) == boolean_true;
  expr uslov_za_putanju2 = (trenutna_pozicija ==
    domen.uzmi_putanju(i).uzmi_indeks_prve_b_tacke());
  expr uslov_za_putanju3 = (b ==
    domen.uzmi_putanju(i).uzmi_indeks_druge_b_tacke());
  expr uslov_za_putanju4 = (duzina_putanje ==
    domen.uzmi_putanju(i).uzmi_duzinu());
  uslov_za_sve_putanje = uslov_za_sve_putanje || (uslov_za_putanju1 &&
    uslov_za_putanju2 && uslov_za_putanju3 && uslov_za_putanju4);
}
robot_resavac.add(uslov_za_sve_putanje);
```

```
/* dodavanje modela dobijenog u prethodnom koraku */
expr uslov_iz_prethodnih_koraka =
  (izabrani_predmet == izabrani_predmet_dobijen_iz_modela) &&
  (izabrano_mesto_u_masini == izabrano_mesto_u_masini_dobijeno_iz_modela);
robot_resavac.add(uslov_iz_prethodnih_koraka);
bool drugi_korak_je_zadovoljiv = true;

/* preduslovi za korak "pokupi" */
int pozicija_robota_pre_koraka_pokupi =
    scena.uzmi_trenutnu_poziciju_robota();

//postoji putanja od b-tacke robota do b-tacke izabranog predmeta
robot_resavac.add(trenutna_pozicija == pozicija_robota_pre_koraka_pokupi);
robot_resavac.add(putanja(trenutna_pozicija, b) == boolean_true);
```

Остали сегменти предуслова су имплементирани на сличан начин.

Важно је скренути пажњу на то да дефиниција предуслова за овај корак користи помоћне функције и предикате, *loc* и *holding* који се израчунавају на основу описа сцене, као и $reachOf$, $\in$, *path* и *blocks* који се рачунају на основу задатог графа распореда.

На крају извравања овог корака, у зависности од задовољивости добијене посредством SMT решавача, могућа су два исхода:

1. **Корак *покупи* је задовољив**. У том случају је неопходно ажурирати позицију робота $CURR$ на б-тачку изабраног предмета *bpt*. Такође је неопходно променити стање робота тако да он држи изабрани предмет.
2. **Корак *покупи* није задовољив**. У том случају, као што је претходно речено, потребно је додати услов да такав заједнички избор предмета o, у коду означеног са `izabrani_predmet` и локације за његово смештање *loc*1, у коду означене са `izabrano_mesto_u_masini` не сме да важи. Потребно је скренути пажњу на део кода за корак *покупи*, који дефинише `uslov_iz_prethodnih_koraka`, као и на део кода корака *пронађи место*, којим се SMT решавачу додаје тај услов. Бектрекинг и одсецање су имплементирани тако да, у случају оваквог исхода, додатни услов буде проширен као `dodatni_uslov = dodatni_uslov && uslov_iz_prethodnih_koraka`.

6.2.1.3 Корак *смести*

Задатак корака *смести* је да смести изабрани предмет на изабрано место. Препознати услови су:

- неопходно је да изабрани предмет и изабрано место имају вредност добијену читањем модела добијеног у кораку *нађи место*
- неопходно је да робот држи изабрани предмет
- мора да постоји путања од б-тачке изабраног предмета до б-тачке машине
- робот мора да се налази у б-тачки изабраног предмета

Корак је у семантици најслабијег предуслова описан акцијом $place(o, ?loc1, ?path2)$, а циљ g му је да робот више не држи објекат o, што се записује као $!holding(o)$. Осим тога циљ је и да буде промењена локација робота на б-тачку машине за обраду $isEqualLoc(CURR, Machine)$, као и да изабрани предмет o буде смештен на изабрано место машине за обраду $loc1$, односно да важи $isEqualLoc(loc(o), Machine)$. Предуслов $wp(P, g)$ представља скуп стања описаних следећим условима:

$$\begin{aligned} &path(path2, bpt, Machine) \wedge \\ &holding(o) \wedge \\ &isEqualLoc(CURR, bpt) \end{aligned} \tag{6.3}$$

Овај предуслов говори да је неопходно да постоји б-тачка bpt из које робот може дохватити место које представља локацију предмета. Такође говори и да не сме постојати ниједан предмет o' који блокира изабрани предмет, као и да мора постојати путања $path1$ дефинисана у графу распореда која повезује тренутну б-тачку робота $CURR$ и bpt. Последњи део услова је да робот не сме држати ниједан предмет у том тренутку.

Следећи код представља део имплементације корака *смести* који описује да је неопходно да робот држи изабрани предмет. Овде је такође неопходно решавачу проследити информацију о избору који су кораци који су му претходили направили. Први део приказаног кода описује прослеђивање избора претходних корака, а други део задавање услова да робот мора да држи изабрани предмет.

```
//dodavanje modela dobijenog u prethodnim koracima
uslov_iz_prethodnih_koraka = uslov_iz_prethodnih_koraka &&
  (b == b_dobijeno_iz_modela); // sadrzi indeks b tacke izabranog predmeta
robot_resavac.add(uslov_iz_prethodnih_koraka);

//robot drzi izabrani predmet (uzimanje trenutnog stanja)
expr uslov_robot_drzi_izabrani_predmet = robot_drzi_izabrani_predmet ==
  (scena.uzmi_predmet(izabrani_predmet_dobijen_iz_modela)
  .da_li_robot_drzi_predmet() ? boolean_true : boolean_false);
robot_resavac.add(uslov_robot_drzi_izabrani_predmet);
```

На основу дефиниције циља g, као и анализом псеудокода 4, препознато је да су на крају извршавања овог корака могућа два исхода:

1. **Корак *смести* је задовољив**. У том случају је неопходно ажурирати позицију робота $CURR$ на б-тачку машине за обраду, означену као $Machine$, ажурирати позицију предмета на изабрану с-тачку машине $loc1$, као и променити стање робота, јер је потребно да он више не држи изабрани предмет, који је у том тренутку већ смештен у машину. Да би имплементација бектрекинга била потпуна, неопходно је још и скинути све додатне услове, јер с обзиром на то да је стање система промењено јер су место одабраног објекта и позиција робота промењени, нема смисла задржавати претходне додатне услове.
2. **Корак *смести* није задовољив**. У том случају, као и у претходном кораку, потребно је додати услов да такав заједнички избор прљавог предмета o, у коду означеног са `izabrani_predmet` и локације за његово смештање $loc1$, у коду означене са `izabrano_mesto_u_masini`. Међутим, с обзиром на то да сваки корак подразумева успешно извршавање претходног корака и чињенице да је у кораку *покупи* промењена позиција робота, али и његово стање, па он сад држи одабрани предмет, неопходно је да прво врати предмет на место где се налазио на почетку петље, као и да се сам робот врати у б-тачку у којој се на почетку петље налазио.

6.2.2 Вађење предмета из машине за обраду и смештање у складиште

Проблем вађења предмета из машине за обраду и њихово паковање у складиште је симетричан проблему паковања предмета у машину, па ће овде бити дат опис само у виду псеудокода и корацима неће бити посвећена додатна пажња. Оно што је важно напоменути је да уколико је корак паковања у

машину успешно прошао, спољној акцији је послата информација да може да започне своје извршавање. Када је она извршена, стање предмета је ажурирано тако да сви предмети који се налазе у домету б-тачке машине за обраду добију стање да су обрађени. Након тога је потребно извршити кораке:

1. *нађи место* у складишту
2. *покупи* изабрани предмет
3. *смести* изабрани предмет у складиште

Акције које је потребно извршити у оквиру сваког од препознатих корака су наведене у алгоритму 5. Разлике у појединачним корацима у односу на кораке које је потребно извршити да би се предмети спаковали у машину за обраду су минималне, као што се види и у приложеном псеудокоду. Из тог разлога целом проблему паковања предмета у складиште неће бити посвећено много пажње у раду. Једино што ће још бити истакнуто су препознати услови који морају да важе за сваки од корака, као и циљ који ће бити испуњен на крају његовог извршавања, ако су услови задовољени.

Први корак, *нађи место*, захтева да буду задовољени следећи услови:

- изабрани предмет треба да буде обрађен
- складиште не сме да буде пуно
- изабрано место мора да припада складишту
- изабрано место треба да буде празно
- изабрани предмет мора да буде смештен у б-тачку машине
- изабрано место у складишту не сме да блокира остала празна места
- изабрани предмет нема други предмет у машини који га блокира

Уколико је корак задовољив, последица ће бити да је изабран обрађени предмет који ће бити смештен, као и само место у складишту на које се он смешта. У случају незадовољивости и цео проблем паковања предмета у складиште је незадовољив.

За други корак, *покупи*, препознати услови који морају да буду испуњени су:

- неопходно је да изабрани предмет и изабрано место имају вредност добијену читањем модела из корака *нађи место*
- робот не сме да држи ниједан објекат
- мора да постоји путања од тренутне позиције робота до б-тачке машине

Ако су услови задовољиви, робот ће променити своју позицију и доћи до б-тачке машине за обраду. Такође, његово стање ће се променити, па ће сада држати изабрани предмет. У случају незадовољивости, као и код проблема

паковања предмета у машину, приступа се бектрекингу.

За успешно извршавање трећег корака, *смести* неопходно је да важе следећи услови:

- неопходно је да изабрани предмет и изабрано место имају вредност добијену читањем модела из корака *нађи место*
- неопходно је да робот држи изабрани предмет
- мора да постоји путања од б-тачке машине до б-чвора складишта
- робот мора да се налази у б-тачки машине за обраду

У случају задовољивости корака, робот је променио своју позицију на б-тачку складишта, сместио изабрани предмет у складиште и више не држи ниједан предмет. Као и код корака *покупи*, уколико је захтеве немогуће задовољити, неопходна је примена бектрекинга.

6.3 Имплементација графичког приказа плана

За графички приказ плана су коришћене библиотеке `Qt`-а. Свака промена сцене за време планирања је битна за реконструкцију плана. Због тога се информације о сваком њеном стању чувају у оквиру класе `Scena`, у низу `stanja_scene`, који садржи информације о позицијама предмета, да ли робот држи предмет или не, као и о тренутној позицији робота.

Предмети, б-тачке и с-тачке, површине, робот и блокирање су за потребе исцртавања генерисаног плана описани класом `Objekat_za_iscrtavanje`, чија је декларација:

```
class Objekat_za_iscrtavanje : public QGraphicsItem
{
public:
  enum Oblik { krug, kvadrat, elipsa, ispunjeni_kvadrat };
  Objekat_za_iscrtavanje(int x, int y, QColor _boja, int _velicina,
                         Oblik _oblik, std::string _naziv);
  ~Objekat_za_iscrtavanje();
  QPoint Point () const;
  int uzmi_x_tacku_pozicije();
  int uzmi_y_tacku_pozicije();
  int uzmi_velicinu();
  QPoint uzmi_poziciju();
  void promeni_poziciju(int x, int y);
  void postavi_poziciju(QPoint nova_pozicija);
protected:
  QRectF boundingRect() const Q_DECL_OVERRIDE;
  void paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
             QWidget *widget) Q_DECL_OVERRIDE;
private:
  QPoint pozicija;
  int velicina;
  QColor boja;
  Oblik oblik;
  std::string naziv;
};
```

Иницијално учитане путање, као и путање добијене након примене Флојд-Варшаловог алгоритма за налажење најкраћих путања у графу су у сврху исцртавања описани класом `Putanja_za_iscrtavanje`, која је декларисана као:

```
class Putanja_za_iscrtavanje
{
public:
  Putanja_za_iscrtavanje(shared_ptr<Objekat_za_iscrtavanje> prva_b_tacka,
            shared_ptr<Objekat_za_iscrtavanje> druga_b_tacka, int _duzina,
            bool _direktna_putanja);
  ~Putanja_za_iscrtavanje();
  shared_ptr<QGraphicsLineItem> Linija();
  shared_ptr<QGraphicsTextItem> Duzina();
private:
  shared_ptr<QGraphicsLineItem> linija;
  shared_ptr<QGraphicsTextItem> duzina;
  bool direktna_putanja;
};
```

Логика којом се добијени план исцртава се налази у оквиру класе `Oblast_za_crtanje`. Класа `Oblast_za_crtanje` је декларисана као:

```
class Oblast_za_crtanje : public QGraphicsView
{
  Q_OBJECT
public:
  Oblast_za_crtanje(QWidget *parent);
  ~Oblast_za_crtanje();
  void dodaj_b_tacku(int x, int y, std::string naziv);
  void dodaj_s_tacku(int x, int y, std::string naziv);
  void dodaj_povrsinu(int x, int y, std::string naziv);
  void dodaj_blokiranje(shared_ptr<Objekat_za_iscrtavanje> druga_s_tacka,
                        std::string naziv);
  shared_ptr<Objekat_za_iscrtavanje> uzmi_b_tacku_za_iscrtavanje(int indeks);
  shared_ptr<Objekat_za_iscrtavanje> uzmi_s_tacku_za_iscrtavanje(int indeks);
  void dodaj_putanju(shared_ptr<Objekat_za_iscrtavanje> prva_b_tacka,
        shared_ptr<Objekat_za_iscrtavanje> druga_b_tacka, int duzina,
        bool direktna_putanja);
  void dodaj_putanju_flojd_varsal(shared_ptr<Objekat_za_iscrtavanje>
        prva_b_tacka, shared_ptr<Objekat_za_iscrtavanje> druga_b_tacka,
        int duzina, bool direktna_putanja);
  void nacrtaj_scenu(int indeks_b_tacke_robota,
        std::vector<std::pair<int, bool>> pozicije_predmeta);
  void azuriraj_stanje_scene(int indeks_b_tacke_robota,
        std::vector<std::pair<int, bool>> pozicije_predmeta,
        bool animacija_u_toku);
```

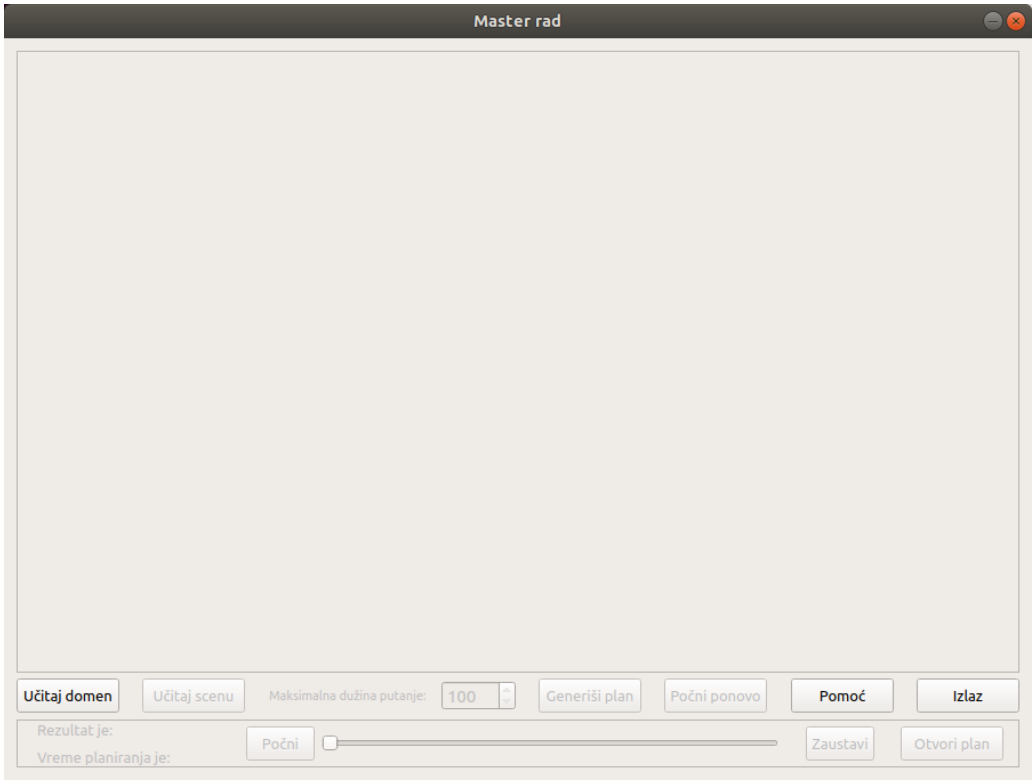
```
  void nacrtaj_domen();
  void ocisti_scenu();
  void delay(int sekunde);
private:
  std::vector<shared_ptr<Objekat_za_iscrtavanje>> b_tacke_za_iscrtavanje;
  std::vector<shared_ptr<Objekat_za_iscrtavanje>> s_tacke_za_iscrtavanje;
  std::vector<shared_ptr<Objekat_za_iscrtavanje>> povrsine_za_iscrtavanje;
  std::vector<shared_ptr<Objekat_za_iscrtavanje>> blokiranja_za_iscrtavanje;
  std::vector<shared_ptr<Putanja_za_iscrtavanje>> putanje_za_iscrtavanje;
  std::vector<shared_ptr<Putanja_za_iscrtavanje>> putanje_flojd_varsal;
  std::vector<shared_ptr<Objekat_za_iscrtavanje>> predmeti_za_iscrtavanje;
  shared_ptr<Objekat_za_iscrtavanje> robot_za_iscrtavanje;
};
```

6.4 Пример планирања

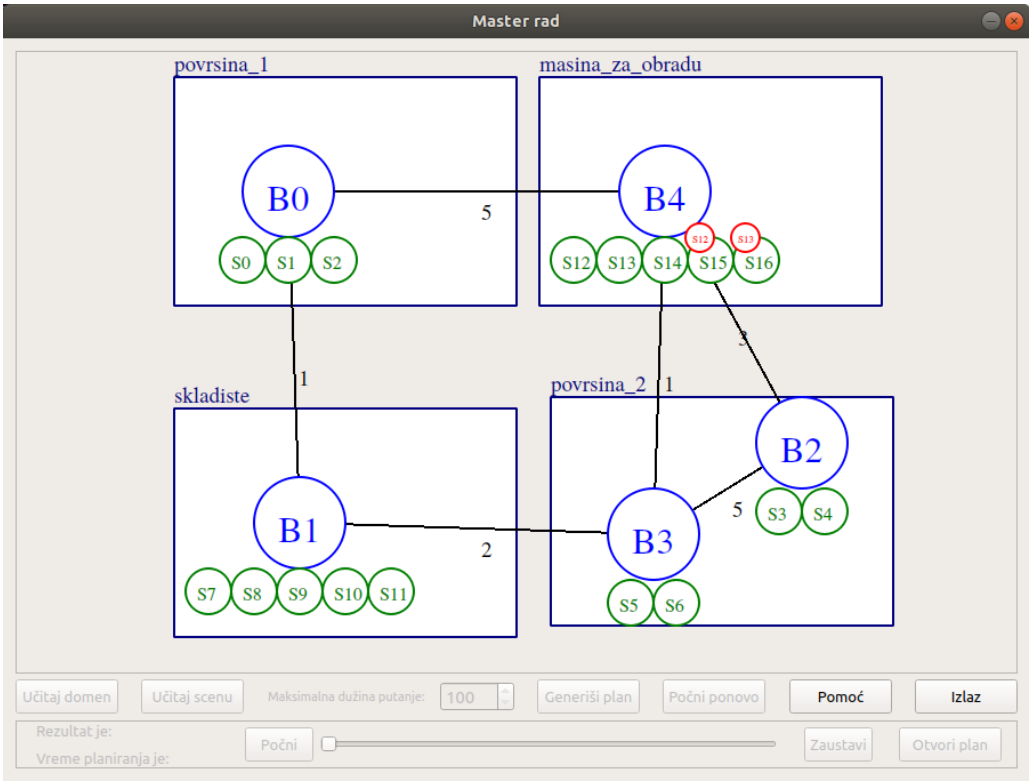
Иницијални прозор добијен покретањем апликације је приказан на слици 6.2. Први корак је учитавање датотеке којом је описан домен у `.txt` формату. За овај пример је коришћена датотека са садржајем који је наведен у примеру конфигурационе датотеке 6.1.1.2. Након учитавања домена, област за приказ

плана је промењена тако да приказује учитани домен и изгледа као на слици 6.3. Тамно плавим правоугаоницима су означене површине, светло плавим круговима б-тачке, зеленим круговима с-тачке и црвеним круговима међусобна блокирања с-тачака, при чему с-тачка унутар црвеног круга блокира с-тачку у чији зелени круг је смештена. Иницијално учитане путање су означене црном бојом и приказана је њихова дужина. На иницијално учитане путање домена се примењује Флојд-Варшалов алгоритам за налажење најкраћих путања у графу и приказане путање се ажурирају. Ажурирано стање путања је приказано на слици 6.4. Путање означене црном бојом представљају директне путање, док су сивом бојом означене путање до којих се стиже обиласком низа чворова. Затим је неопходно учитати сцену у `.txt` формату. Коришћена је кофигурација из примера 6.1.2.2. Област за приказ плана након тога садржи и елементе сцене и приказана је на слици 6.5. Предмети су означени црним елипсама, а робот бордо правоугаоником.

Као што је раније поменуто, постоји ограничење у максималној дужини пређеног пута и могуће га је мењати са екрана, уносом жељене дужине у поље `Maksimalna dužina putanje`. Након тога се помоћу дугмета `Generiši plan` започиње планирање. Уколико је немогуће задовољити све услове и изгенерисати план, биће пријављено да је проблем планирања незадовољив. У том случају је могуће коришћењем дугмета `Otvori plan`, на основу текстуалне репрезентације, видети како је процес планирања текао. Уколико је планирање било успешно и могуће је задовољити све услове, биће пријављено да је проблем планирања задовољив и биће приказано време које је потрошено за планирање. Помоћу дугмета `Počni` унутар области за приказ ће почети анимирано извршење планираних акција. Анимацију је могуће зауставити и померањем слајдера приказати сцену по сцену акције. На крају приказа добијеног плана судови су смештени у складиште и то је приказано на слици 6.6. Пример текстуалног приказа генерисаног плана, добијеног помоћу дугмета `Otvori plan`, за случај када је планирање било успешно, је приказан на слици 6.7. Након планирања на изабраној почетној конфигурацији, доступно је дугме `Počni ponovo`, којим се омогућава учитавање нове сцене и домена. У сваком тренутку су доступна дугмад `Pomoć` и `Izlaz`.



Слика 6.2: Почетни екран апликације

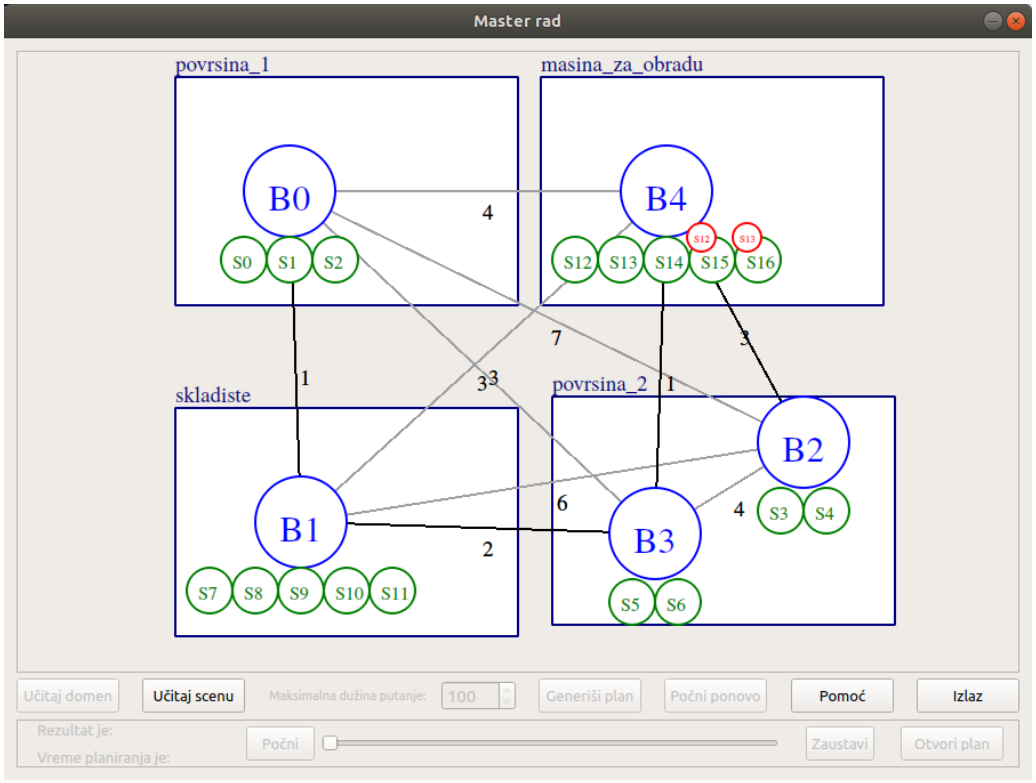


Слика 6.3: Екран добијен након учитавања домена

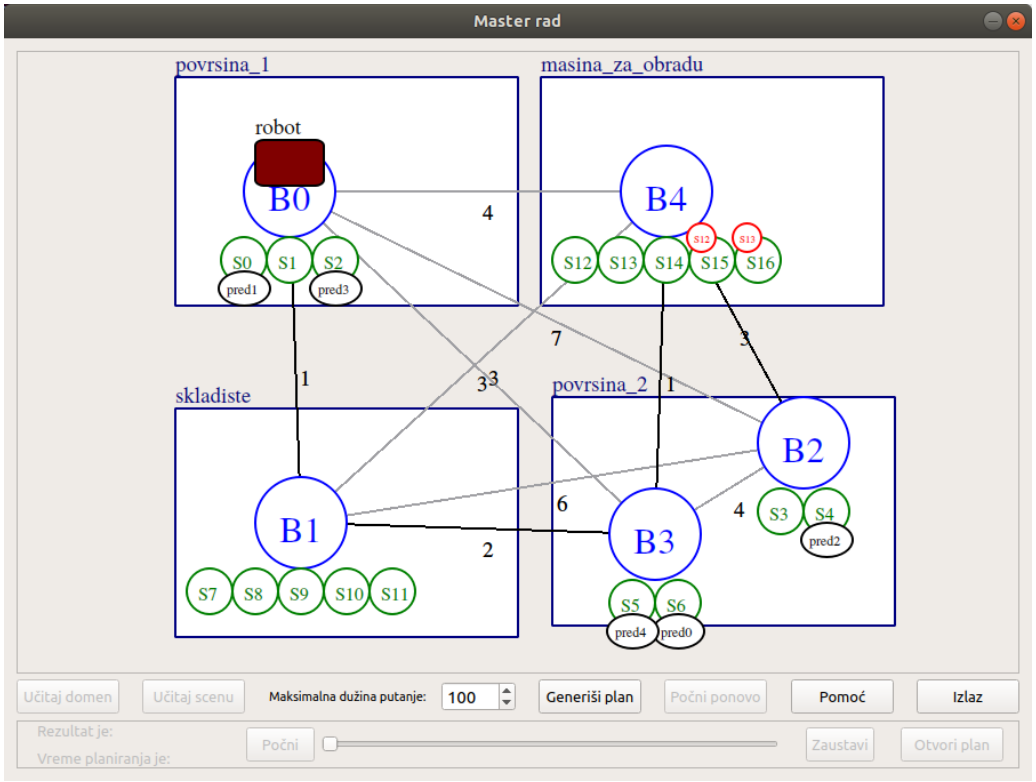
Алгоритам 5: Паковање обрађених предмета из машине за обраду у складиште

Резултат: Сви обрађени предмети су спаковани у складиште

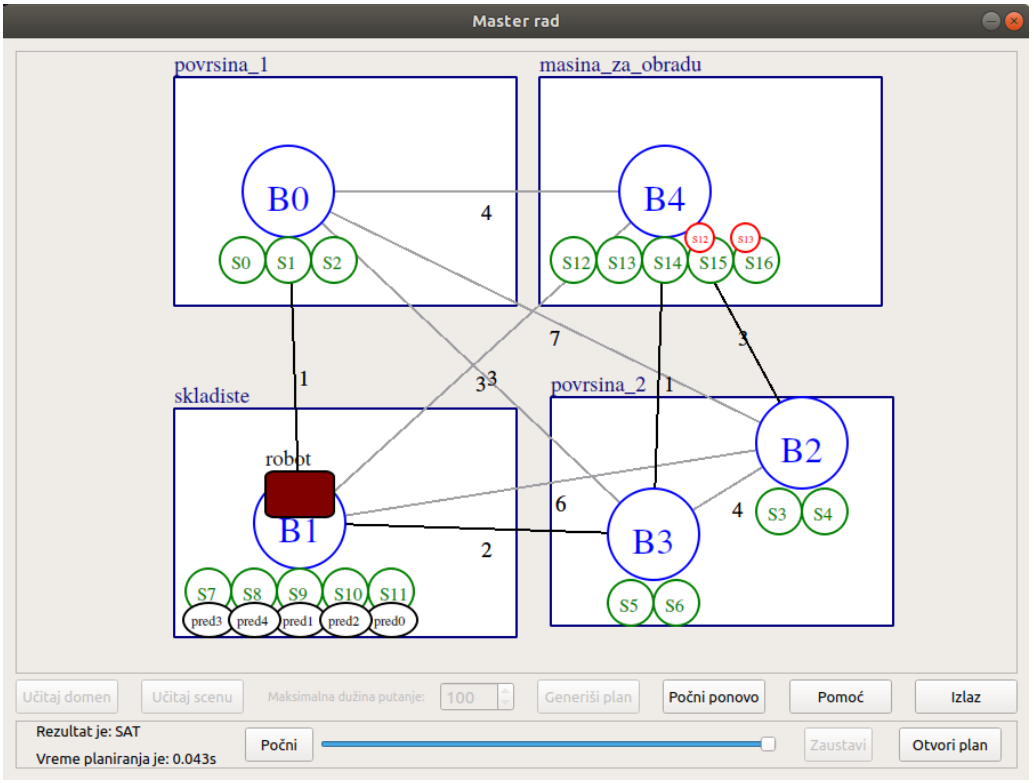
```
1  uslov_petlje := провери да ли постоје обрађени предмети у машини
2  dodatni_uslov := true; prvi_korak_zadovoljiv := true
3  while uslov_petlje and prvi_korak_zadovoljiv do
4      додај информације о томе који су предмети обрађени
5      додај услове за корак нађи место у складишту
6      додај услов dodatni_uslov // омогућава бектрекинг
7      prvi_korak_zadovoljiv := провери задовољивост
8      if !(prvi_korak_zadovoljiv) then
9          break
10     else
11         izabrani_predmet, izabrano_mesto := узми вредности из добијеног
             модела
12     end
13     if prvi_korak_zadovoljiv then
14         додај услове за корак покупи изабрани предмет
15         drugi_korak_zadovoljiv := провери задовољивост
16         if !(drugi_korak_zadovoljiv) then
17             dodatni_uslov := dodatni_uslov and (изабрано место различито
                 од izabrano_mesto or изабрани премет различит од
                 izabrani_predmet)
18         else
19             промени позицију робота на б-тачку машине
20             промени стање робота тако да држи izabrani_predmet
21         end
22     end
23     if drugi_korak_zadovoljiv then
24         додај услове за корак смести изабрани предмет у складиште
             treci_korak_zadovoljiv := провери задовољивост
25         if !(treci_korak_zadovoljiv) then
26             dodatni_uslov := dodatni_uslov and (изабрано место различито
                 од izabrano_mesto or изабрани предмет различит од
                 izabrani_predmet)
27             врати предмет у с-тачку у ком је био пре корака покупи
28             промени стање робота тако да не држи изабрани предмет
29             врати робота у б-тачку у ком је био пре корака покупи
30         else
31             промени позицију робота на б-тачку складишта
32             промени с-тачку изабраног предмета на с-тачку изабраног места
33             промени стање робота тако да више не држи изабрани предмет
34             dodatni_uslov := true
35         end
36     end
37     uslov_petlje := провери да ли постоје обрађени предмети ван
         складишта
38 end
39 if prvi_korak_zadovoljiv then
40     пријави да је проблем паковања предмета у складиште задовољив
41 else
42     пријави да је проблем паковања предмета у складиште незадовољив
43 end
```



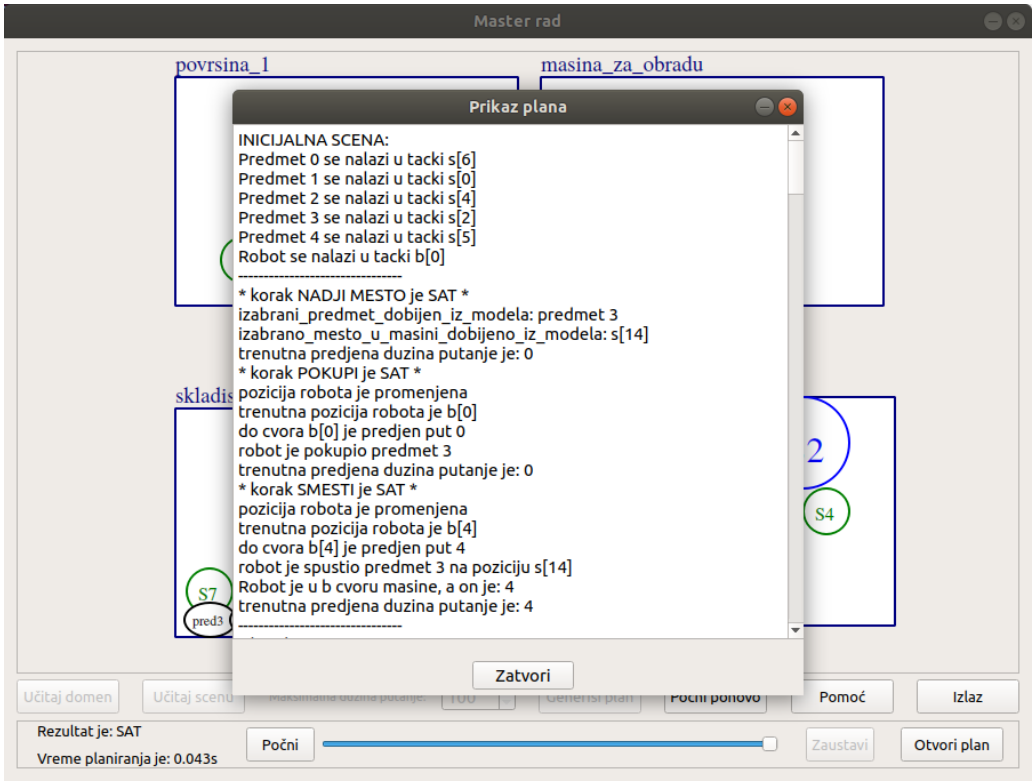
Слика 6.4: Екран добијен након примене Флојд-Варшаловог алгоритма



Слика 6.5: Екран добијен након учитавања сцене



Слика 6.6: Екран након извршења свих планираних акција



Слика 6.7: Текстуална репрезентација генерисаног плана

6.5 Поређење брзине извршавања планирања у зависности од почетних конфигурација

Ефикасност имплементираног решења ће бити поређена генерисањем плана за различите почетне конфигурације сцене и домене различите комплексности. У првом делу ће бити анализирана брзина извршавања за задовољиве конфигурације, а у другом делу ће пажња бити посвећена незадовољивим конфигурацијама. Времена су мерена на *Linux* виртуелној машини која ради са 64-битним *Ubuntu 18* оперативним системом. Виртуелна машина користи два језгра процесора Intel Core i5-6200U CPU @ 2.30GHz и 4GB RAM меморије.

6.5.1 Брзине планирања за задовољиве конфигурације

У табели 6.5.1 су приказани резултати добијени планирањем над задовољивим конфигурацијама. С обзиром на то да у свакој итерацији на избор предмета и места за његово смештање, као и на могућност потребе за применом бектрекинга, највише утичу број предмета који се распоређују и број међусобних блокирања с-тачака, време извршавања планирања ће бити мерено у односу на конфигурације код којих ова два броја варирају.

Табела 6.1: Брзина планирања у зависности од различитих конфигурација изражена у секундама

број предмета \ број блокирања	2	5	10	20	30
5	0.032	0.035	0.034	0.041	0.036
10	0.073	0.076	0.123	0.091	0.077
20	0.549	0.539	0.433	0.459	0.448

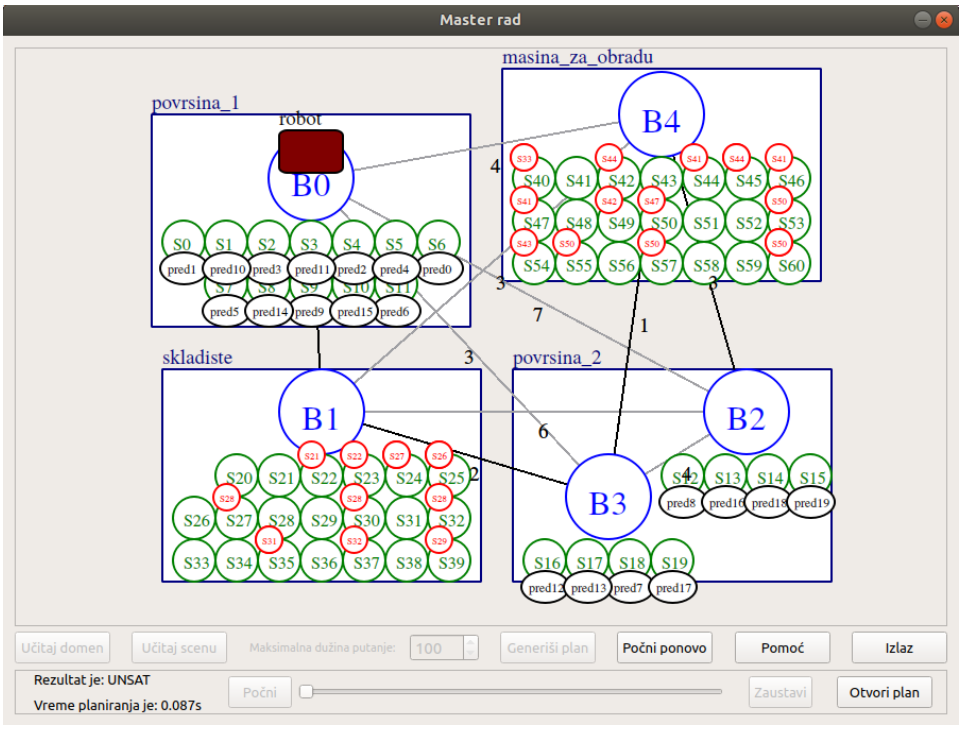
На основу добијених резултата се примећује да брзина извршавања зависи од броја предмета, као и то да је за исти број предмета који се распоређују брзина извршавања стална и не зависи од броја међусобних блокирања с-тачака и почетног распореда предмета. С обзиром на комлексност проблема за који је алгоритам планирања имплементиран, времена добијена експериментално показују да је предложено решење реално применљиво и на релативно велики број предмета.

6.5.2 Планирање над незадовољивим конфигурацијама

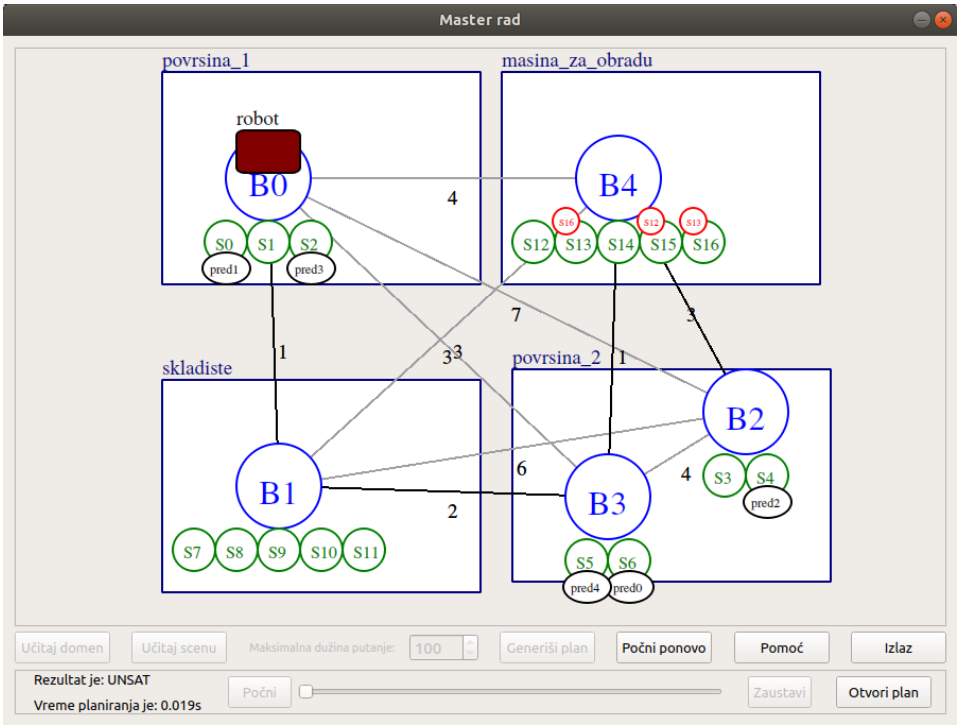
Различити аспекти домена и сцене могу допринети томе да решења буду незадовољива. Један од препознатих случајева укључује конфигурацију у којој је потребно 20 предмета који чекају обраду сместити у складиште која има само 19 места. Приказ учитаног домена и сцене је приказан на слици 6.8. Са слике се види да време планирања од 0.087 секунди не одступа од планирања над сличним доменом и сценом код којих је план задовољив, па се закључује да је овај пример незадовољивости ефикасно покривен имплементираним решењем.

Наредна конфигурација је незадовољива због тога што постоје две с-тачке које међусобно једна другу блокирају унутар машине за обраду. Учитани домен и сцена су приказани на слици 6.9. Резултат који говори да је план незадовољив је добијен за 0.019 секунди, па се одатле види да је имплементирано решење примељиво и на овакве конфигурације.

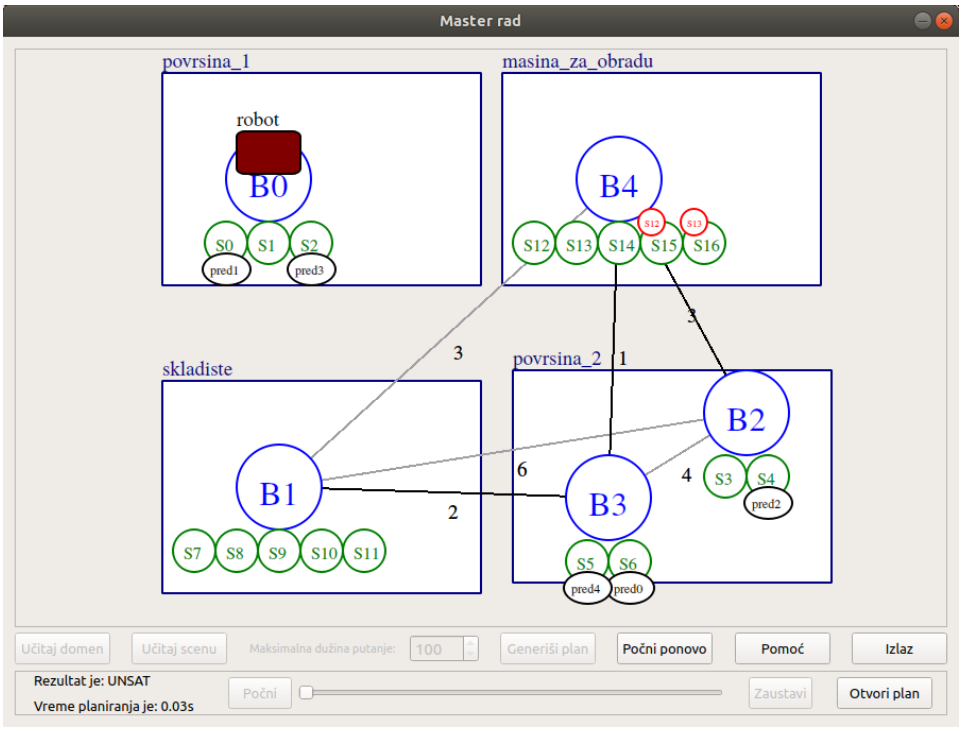
Уколико не постоји пут до неког од предмета који чека обраду, као што је приказано на слици 6.10, брзина планирања од 0.03 секунде не одступа много од сличне конфигурације за коју је проблем планирања задовољив. Међутим, уколико је велики број непостојећих путања до предмета које је потребно покупити, или уколико не постоји путања од машине за обраду до складишта, као у случају приказаном на слици 6.11, брзина планирања се значајно повећава и износи 1.111 секунду. У овом примеру се први део алгоритма, који подразумева да се сваки предмет покупи и смести у машину за обраду, извршава ефикасно и задовољив је. Други део алгоритма, када је потребно сместити предмете из машине за обраду у складиште је незадовољив. С обзиром на то да се провера да ли путања постоји врши тек у другом кораку, а уколико се открије да је други корак незадовољив претпоставља се да је први корак направио лош избор предмета или места у складишту, па се затим покушава са новим избором предмета и места у складишту. Овај поступак ће бити понављан за све парове предмета и слободних места и тек када су све комбинације испробане биће пријављено да је проблем незадовољив. Ово је најлошији анализирани незадовољавајући сценарио и уколико би се проблем даље развијао, био би једна од ствари на коју је потребно обратити пажњу и пронаћи начин да се трагање за планом у овом случају оптимизује.



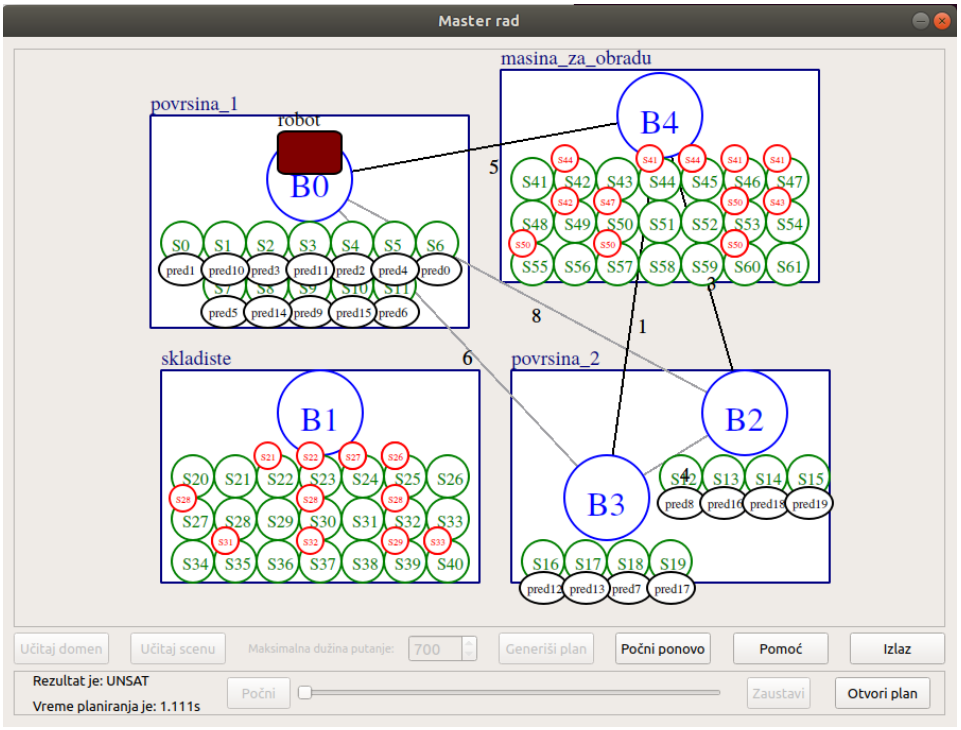
Слика 6.8: Учитана конфигурација за коју је планирање незадовољиво због недовољног броја места у складишту



Слика 6.9: Учитана конфигурација за коју је планирање незадовољиво због узајамног блокирања једног пара с-тачака



Слика 6.10: Учитана конфигурација за коју је планирање незадовољиво због непостојања путање до неких од предмета који чекају обраду



Слика 6.11: Учитана конфигурација за коју је планирање незадовољиво због непостојања путање од машине за обраду до складишта

7. Закључци и даљи рад

У раду је описан имплементирани систем који се користи за решавање проблема интегрисаног планирања кретања и редоследа извршавања задатака на проблему паковања предмета у машину за обраду, где се извршава нека спољна акција над њима и њихово смештање у складиште, из ког је могуће започети неку нову акцију над њима. Овај проблем припада области роботике и аутоматизације и примену налази у индустрији, а среће се и у контексту помоћника у обављању кућних послова. Посматрани проблем је примењив на мобилног робота који треба, у складу са геометријским ограничењима окружења и предмета над којима врши акције, да испланира редослед акција којим ће постићи свој циљ. Примена роботике и аутоматизације је у порасту претходних година, па је тема којом се бави овај рад актуелна.

Главни допринос рада је решавање проблема интегрисаног планирања трансформацијом ограничења геометријских услова околине и правила за извршење задатака у улаз који је могуће проследити SMT решавачу. Додатно, решавачу се прослеђују ограничења конструисана у складу са информацијама које програмер зна о томе како исправни планови изгледају, а која се користе у сврху оптимизације рада алгоритма планирања усмеравањем претраге ка исправном плану. Геометријско планирање се ослања на улаз који описује физичко окружење, а који је направљен на основу графа распореда. Оригинално учитане путање су оптимизоване применом Флојд-Варшаловог алгоритма за налажење најкраћих путева у графу. У раду је описана семантика најслабијег предуслова као средство за доказивање коректности програма, а формални описи акција и услова неопходних за извршење сваког корака алгоритма су описани помоћу ње. На основу тога би било могуће доказати коректност имплементираног алгоритма планирања.

Експериментално је показано да се брзина извршавања алгоритма планирања не мења значајно у зависности од почетне конфигурације сцене и домена. Брзина зависи доминантно од броја предмета над којима је потребно вршити акције. С обзиром на комплексност проблема, алгоритам показује задовољавајуће перформансе и примењив је на проблеме који укључују већи број предмета и комплексније домене. У складу са тим, рад представља добру

основу за будућа истраживања у класи проблема који се баве интегрисаним планирањем кретања и редоследа извршавања задатака употребом SMT решавача. Имплементирано решење налази било који план, уколико он постоји, па би наредна истраживања требала да буду усмерена ка проналажењу начина да се оптимизује кретање тако да пређени пут постане најкраћи могући. Унапређење ове врсте није тривијално, због начина на који је избор премета и места за смештање тренутно имплементиран. Велики је број услова које треба испунити да би неки предмет могао да се одабере, при чему тај избор зависи и од стања сцене које је променљиво за време планирања и у складу са тим није очигледно колика је преостала дужина пута, што би било потребно искористити у случају једноставне оптимизације путање. Могуће је само у сваком тренутку ослонити се на локалне минимуме, а то не води нужно до оптималног укупно пређеног пута. Решење описано у раду је могуће унапредити и тако да додатно усмери претрагу у свакој итерацији на основу информација добијених из претходно пронађених неисправних планова, код којих је било потребно применити бектрекинг. Описани алгоритам планирања је могуће применити на софтверским симулаторима робота, као и на физичким роботима. У том случају би било потребно аутоматизовати процес претварања информација добијених из физичког окружења робота у граф распореда, на основу ког би касније, такође аутоматски, била генерисана ограничења која описују физичко окружење у ком робот обавља акције.

С обзиром на то да су појединачни услови, који су кодирани у логици, једноставне егзистенцијалне формуле повезане конјукцијом, није сигурно да ли је било неопходно користити SMT решавач или би се исти, а можда и бољи резултати могли добити неким једноставно имплементираним, доменски специфичним процедурама. Уколико би се планирао даљи рад на овде приказаном решењу, ово би требало имати на уму као потенцијално унапређење. Међутим треба бити опрезан, с обзиром на то да је овај рад имао за задатак имплементацију решења описаног у раду *Синтеза интегрисаног планера редоследа извршавања задатака и кретања уз контуру плана, заснована на SMT решавачима* [73] и није предвиђен за планирање над конкретним физичким роботом. Планирање је због тога поједностављено, док би за правог робота било потребно направити компликованије услове који потичу од детаљнијих ограничења за сваки корак, више утицаја из спољног света, реалнијих ограничења о његовој динамици, али и динамици света око њега и већег броја акција које је он способан да примени. Ово би можда захтевало

додавање услова са сложенијом структуром, па би употреба SMT решавача тада можда и даље била исправан избор.

Литература

[1] Samir Alili, Amit Kumar Pandey, Emrah Sisbot, and Rachid Alami. Interleaving symbolic and geometric reasoning for a robotic assistant. 01 2010.

[2] Milan Banković. Materijali za vežbe iz kursa automatsko rezonovanje, matematički fakultet, 2015/2016.

[3] Clark Barrett, Roberto Sebastiani, S.A. Seshia, and C. Tinelli. Satisfiability modulo theories. *Handbook of Satisfiability*, pages 825–885, 01 2009.

[4] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.

[5] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa, 2017. Available at `www.SMT-LIB.org`.

[6] Jennifer Barry, Kaijen Hsiao, Leslie Kaelbling, and Tomás Lozano-Pérez. *Manipulation with Multiple Action Types*, volume 88, pages 531–545. 01 2013. doi: 10.1007/978-3-319-00065-7_36.

[7] Calin Belta, Volkan Isler, and George Pappas. Discrete abstractions for robot motion planning and control in polygonal environments. *Robotics, IEEE Transactions on*, 21:864 – 874, 11 2005. doi: 10.1109/TRO.2005.851359.

[8] Julien Bidot, Lars Karlsson, Fabien Lagriffoul, and Alessandro Saffiotti. Geometric backtracking for combined task and motion planning in robotic systems. *Artificial Intelligence*, 247, 05 2015. doi: 10.1016/j.artint.2015.03.005.

[9] Stéphane Cambon, Rachid Alami, and Fabien Gravot. A hybrid approach to intricate motion, manipulation and task planning. *I. J. Robotic Res.*, 28: 104–126, 01 2009. doi: 10.1177/0278364908097884.

[10] John Canny. The complexity of robot motion planning. 01 1988.

[11] Howie Choset, K. Lynch, S. Hutchinson, George Kantor, Wolfram Burgard, Lydia Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms and Implementation*. 01 2005.

[12] Alessandro Cimatti, Alberto Griggio, Bastiaan Schaafsma, and Roberto Sebastiani. The MathSAT5 SMT Solver. In Nir Piterman and Scott Smolka, editors, *Proceedings of TACAS*, volume 7795 of *LNCS*. Springer, 2013.

[13] Benjamin Cohen, Sachin Chitta, and Maxim Likhachev. Single- and dual-arm motion planning with heuristic search. *International Journal of Robotics Research*, 33:305–320, 02 2014. doi: 10.1177/0278364913507983.

[14] Neil Dantam and Mike Stilman. The motion grammar: Analysis of a linguistic method for robot control. *Robotics, IEEE Transactions on*, 29:704–718, 06 2013. doi: 10.1109/TRO.2013.2239553.

[15] Neil Dantam, Zachary Kingston, Swarat Chaudhuri, and Lydia Kavraki. Incremental task and motion planning: A constraint-based approach. 06 2016. doi: 10.15607/RSS.2016.XII.002.

[16] Neil Dantam, Zachary Kingston, Swarat Chaudhuri, and Lydia Kavraki. An incremental constraint-based framework for task and motion planning. *The International Journal of Robotics Research*, page 027836491876157, 03 2018. doi: 10.1177/0278364918761570.

[17] Leonardo de Moura and Nikolaj Bjørner. Efficient e-matching for smt solvers. volume 4603, pages 183–198, 07 2007. doi: 10.1007/978-3-540-73595-3_13.

[18] Leonardo de Moura and Nikolaj Bjørner. Model-based theory combination. *Electronic Notes in Theoretical Computer Science*, 198:37–49, 05 2008. doi: 10.1016/j.entcs.2008.04.079.

[19] Leonardo de Moura and Nikolaj Bjørner. Z3: an efficient smt solver. volume 4963, pages 337–340, 04 2008. doi: 10.1007/978-3-540-78800-3_24.

[20] Leonardo de Moura and Nikolaj Bjørner. Satisfiability modulo theories: Introduction and applications. *Commun. ACM*, 54:69–77, 09 2011. doi: 10.1145/1995376.1995394.

[21] Leonardo de Moura and Nikolaj Bjørner. Relevancy propagation. 05 2020. doi: 10.1007/978-0-387-39940-9_3460.

[22] Lavindra de Silva, Amit Kumar Pandey, and Rachid Alami. An interface for interleaved symbolic-geometric planning and backtracking. pages 232–239, 11 2013. doi: 10.1109/IROS.2013.6696358.

[23] Lavindra de Silva, Amit Kumar Pandey, Mamoun Gharbi, and Rachid Alami. Towards combining htn planning and geometric task planning. 07 2013.

[24] R. Dearden and C. Burbridge. An approach for efficient planning of robotic manipulation tasks. *ICAPS 2013 - Proceedings of the 23rd International Conference on Automated Planning and Scheduling*, pages 55–63, 01 2013.

[25] Richard Dearden and Chris Burbridge. Manipulation planning using learned symbolic state abstractions. *Robotics and Autonomous Systems*, 62, 01 2013. doi: 10.1016/j.robot.2013.09.015.

[26] David Detlefs, Greg Nelson, and James Saxe. Simplify: A theorem prover for program checking. *Journal of the ACM*, 52, 09 2003. doi: 10.1145/1066100.1066102.

[27] Edsger Dijkstra. *A Discipline of Programming*. 01 1976.

[28] Mehmet Dogar and Siddhartha Srinivasa. A framework for push-grasping in clutter. 06 2011. doi: 10.15607/RSS.2011.VII.009.

[29] Christian Dornhege, Marc Gissler, Matthias Teschner, and Bernhard Nebel. Integrating symbolic and geometric planning for mobile manipulation. pages 1 – 6, 12 2009. doi: 10.1109/SSRR.2009.5424160.

[30] Christian Dornhege, Patrick Eyerich, Thomas Keller, Michael Brenner, and Bernhard Nebel. Integrating task and motion planning using semantic attachments. 01 2010.

[31] Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner, and Bernhard Nebel. *Semantic Attachments for Domain-Independent Planning Systems*. 01 2012.

[32] Bruno Dutertre. Yices 2.2. In Armin Biere and Roderick Bloem, editors, *Computer-Aided Verification (CAV'2014)*, volume 8559 of *Lecture Notes in Computer Science*, pages 737–744. Springer, July 2014.

[33] Esra Erdem, Kadir Haspalamutgil, Can Palaz, Volkan Patoglu, and Tansel Uras. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. pages 4575–4581, 05 2011. doi: 10.1109/ICRA.2011.5980160.

[34] Esra Erdem, Erdi Aker, and Volkan Patoglu. Answer set programming for collaborative housekeeping robotics: Representation, reasoning, and execution. *Intelligent Service Robotics*, 5, 10 2012. doi: 10.1007/s11370-012-0119-x.

[35] Richard Fikes. Monitored execution of robot plans produced by strips. *Proceedings IFIP Congress 71*, 02 1972.

[36] Richard Fikes and Nils Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 12 1971. doi: 10.1016/0004-3702(71)90010-5.

[37] Andre Gaschler, Ronald Petrick, Manuel Giuliani, Markus Rickert, and Alois Knoll. Kvp: A knowledge of volumes approach to robot task planning. pages 202–208, 11 2013. doi: 10.1109/IROS.2013.6696354.

[38] Malik Ghallab, Craig Knoblock, David Wilkins, Anthony Barrett, Dave Christianson, Marc Friedman, Chung Kwok, Keith Golden, Scott Penberthy, David Smith, Ying Sun, and Daniel Weld. Pddl - the planning domain definition language. 08 1998.

[39] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning. Theory and practice*. 05 2004. ISBN 978-1-55860-856-6.

[40] M. Gharbi, R. Lallement, and R. Alami. Combining symbolic and geometric planning to synthesize human-aware plans: toward more efficient combined search. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6360–6365, 2015.

[41] Julien Guitton and Jean-Loup Farges. Taking into account geometric constraints for task-oriented motion planning. *ICAPS Workshop on Bridging the Gap between Task and Motion Planning*, 01 2009.

[42] Julien Guitton and Jean-Loup Farges. Towards a hybridization of task and motion planning for robotic architectures. 01 2009.

[43] Kris Hauser and Jean-Claude Latombe. Integrating task and prm motion planning: Dealing with many infeasible motion planning queries. *Proc. Bridging the Gap Between Task and Motion Planning, ICAPS Workshop, BTAMP'09*, 01 2009.

[44] Kris Hauser and Victor Ng-Thow-Hing. Randomized multi-modal motion planning for a humanoid robot manipulation task. *I. J. Robotic Res.*, 30: 678–698, 05 2011. doi: 10.1177/0278364910386985.

[45] Giray Havur, Kadir Haspalamutgil, Can Palaz, Esra Erdem, and Volkan Patoglu. A case study on the tower of hanoi challenge: Representation, reasoning and execution. pages 4552–4559, 05 2013. ISBN 978-1-4673-5641-1. doi: 10.1109/ICRA.2013.6631224.

[46] Keliang He, Morteza Lahijanian, Lydia Kavraki, and Moshe Vardi. Towards manipulation planning with temporal logic specifications. *Proceedings - IEEE International Conference on Robotics and Automation*, 2015:346–352, 06 2015. doi: 10.1109/ICRA.2015.7139022.

[47] M. Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, Jul 2006. ISSN 1076-9757. doi: 10.1613/jair.1705.

[48] Jörg Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research - JAIR*, 14, 06 2011. doi: 10.1613/jair.855.

[49] Predrag Janičić. *Matematička logika u računarstvu.* Matematički fakultet, Studentski trg 16, Beograd, 2008. Available at `http://poincare.matf.bg.ac.rs/ janicic/books/mlr.pdf`.

[50] Predrag Janičić and Filip Marić. *PROGRAMIRANJE 2, Osnove programiranja kroz programski jezik C.* Matematički fakultet, Studentski trg 16, Beograd, 2020. Available at `http://poincare.matf.bg.ac.rs/ janicic//courses/p2-a4.pdf`.

[51] Rushby JM. Harnessing disruptive innovation in formal verification. pages 21 – 30, 10 2006. doi: 10.1109/SEFM.2006.24.

[52] Leslie Kaelbling and Tomas Lozano-Perez. Hierarchical task and motion planning in the now. pages 1470 – 1477, 06 2011. doi: 10.1109/ICRA.2011.5980391.

[53] Leslie Kaelbling and Tomas Lozano-Perez. Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, 32:1194–1227, 08 2013. doi: 10.1177/0278364913484072.

[54] Lars Karlsson, Julien Bidot, Fabien Lagriffoul, Alessandro Saffiotti, Ulrich Hillenbrand, and Florian Schmidt. Combining task and path planning for a humanoid two-arm robotic system. 01 2012.

[55] Henry Kautz and Bart Selman. Planning as satisfiability. pages 359–363, 01 1992.

[56] Henry Kautz and Bart Selman. Unifying sat-based and graph-based planning. *IJCAI International Joint Conference on Artificial Intelligence*, 1, 05 1999.

[57] Lydia Kavraki, Petr Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12:566 – 580, 09 1996. doi: 10.1109/70.508439.

[58] Hadas Kress-Gazit, Georgios Fainekos, and George Pappas. Translating structured english to robot controllers. *Advanced Robotics*, 22:1343–1359, 10 2008. doi: 10.1163/156855308X344864.

[59] Hadas Kress-Gazit, Georgios Fainekos, and George Pappas. Temporal-logic-based reactive mission and motion planning. *Robotics, IEEE Transactions on*, 25:1370 – 1381, 01 2010. doi: 10.1109/TRO.2009.2030225.

[60] Fabien Lagriffoul, Dimitar Dimitrov, Alessandro Saffiotti, and Lars Karlsson. Constraint propagation on interval bounds for dealing with geometric backtracking. pages 957–964, 10 2012. doi: 10.1109/IROS.2012.6385972.

[61] Fabien Lagriffoul, Lars Karlsson, Julien Bidot, and Alessandro Saffiotti. Combining task and motion planning is not always a good idea. In *RSS 2013*, 2013.

[62] Steven Lavalle. Planning algorithms. 11 2004.

[63] Steven Lavalle and James Kuffner. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and computational robotics: New directions*, 01 2000.

[64] Daniel Leidner and Christoph Borst. Hybrid reasoning for mobile manipulation based on object knowledge. 11 2013.

[65] Vladimir Lifschitz. Answer set planning. pages 373–374, 01 1999. doi: 10.1007/3-540-46767-X_28.

[66] T. Lozano-Perez, J. Jones, Emmanuel Mazer, P. O'Donnell, W. Grimson, P. Tournassoud, and A. Lanusse. Handey: A robot system that recognizes, plans, and manipulates. pages 843 – 849, 04 1987. doi: 10.1109/ROBOT.1987.1087847.

[67] Tomas Lozano-Perez. Automatic planning of manipulator transfer movements. *Systems, Man and Cybernetics, IEEE Transactions on*, 11:681 – 698, 11 1981. doi: 10.1109/TSMC.1981.4308589.

[68] Tomas Lozano-Perez and Leslie Kaelbling. A constraint-based method for solving sequential manipulation planning problems. *IEEE International Conference on Intelligent Robots and Systems*, pages 3684–3691, 10 2014. doi: 10.1109/IROS.2014.6943079.

[69] Damian Lyons and Michael Arbib. A formal model of computation for sensory-based robotics. *Robotics and Automation, IEEE Transactions on*, 5:280 – 293, 07 1989. doi: 10.1109/70.34764.

[70] Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. Learning to parse natural language commands to a robot control system. *Experimental Robotics*, 88, 01 2013. doi: 10.1007/978-3-319-00065-7_28.

[71] Norman Mccain and Hudson Turner. Causal theories of action and change. *Proc. AAAI-97*, 04 1997.

[72] Vlad Morariu, Balaji Srinivasan, Vikas Raykar, Ramani Duraiswami, and Larry Davis. Automatic online tuning for fast gaussian summation. volume 2008, pages 1113–1120, 01 2008.

[73] Srinivas Nedunuri, Sailesh Prabhu, Mark Moll, Swarat Chaudhuri, and Lydia Kavraki. Smt-based synthesis of integrated task and motion plans from plan outlines. pages 655–662, 05 2014. doi: 10.1109/ICRA.2014.6906924.

[74] Erion Plaku and Gregory Hager. Sampling-based motion and symbolic action planning with geometric and differential constraints. pages 5002–5008, 05 2010. doi: 10.1109/ROBOT.2010.5509563.

[75] L. Riano, S. Russell, and P. Abbeel. Using classical planners for tasks with continuous operators in robotics. *AAAI Workshop - Technical Report*, pages 85–91, 01 2013.

[76] Jussi Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45 – 86, 2012. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2012.08.001.

[77] Jussi Rintanen. Engineering efficient planners with sat. *Frontiers in Artificial Intelligence and Applications*, 242:684–689, 01 2012. doi: 10.3233/978-1-61499-098-7-684.

[78] Jussi Rintanen. Madagascar : Scalable planning with sat. 2014.

[79] Peter Schüller, Volkan Patoglu, and Esra Erdem. A systematic analysis of levels of integration between low-level reasoning and task planning. 05 2013.

[80] David Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. 03 2015. ISBN 9781118575536.

[81] Thierry Siméon. Manipulation planning with probabilistic roadmaps. *The International Journal of Robotics Research*, 23:729–746, 08 2004. doi: 10.1177/0278364904045471.

[82] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. pages 639–646, 05 2014. doi: 10.1109/ICRA.2014.6906922.

[83] Moritz Tenorth and Michael Beetz. Representations for robot knowledge in the knowrob framework. *Artificial Intelligence*, 247, 06 2015. doi: 10.1016/j.artint.2015.05.010.

[84] R. Vijaykumar, S. Venkataraman, G. Dakin, and D. M. Lyons. A task grammar approach to the structure and analysis of robot programs. Technical report, USA, 1987.

[85] Yue Wang, Neil Dantam, Swarat Chaudhuri, and Lydia Kavraki. Task and motion policy synthesis as liveness games. 06 2016.

[86] Gordon Wilfong. Motion planning in the presence of movable obstacles. pages 279–288, 01 1988.

[87] Jason Wolfe, Bhaskara Marthi, and Stuart Russell. Combined task and motion planning for mobile manipulation. pages 254–258, 01 2010.