УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

Милан Банковић

УНАПРЕЂИВАЊЕ SMT РЕШАВАЧА КОРИШЋЕЊЕМ CSP ТЕХНИКА И ТЕХНИКА ПАРАЛЕЛИЗАЦИЈЕ

докторска дисертација

Београд, 2016.

UNIVERSITY OF BELGRADE
FACULTY OF MATHEMATICS

Milan Banković

IMPROVING SMT SOLVERS USING CSP TECHNIQUES AND PARALLELIZATION TECHNIQUES

Doctoral Dissertation

Belgrade, 2016.

Ментор:

др Филип Марић, ванредни професор
Универзитет у Београду, Математички факултет

Чланови комисије:

др Миодраг Живковић, редовни професор
Универзитет у Београду, Математички факултет

др Предраг Јаничић, редовни професор
Универзитет у Београду, Математички факултет

др Зоран Огњановић, научни саветник
Математички институт САНУ

Датум одбране: _____

Мојим родитељима

Наслов дисертације: Унапређивање SMT решавача коришћењем CSP техника и техника паралелизације

Резиме: SMT решавачи имплементирају процедуре одлучивања за испитивање задовољивости логичких формула првог реда у односу на неку унапред задату одлучиву теорију. Ослањају се на моћне технике за решавање проблема исказне задовољивости (SAT) у комбинацији са специфичним процедурама одлучивања које омогућавају резоновање у конкретној теорији. Типична примена SMT решавача је у верификацији софтвера, па су зато теорије које се стандардно користе у SMT решавачима прилагођене тој врсти примене. Проширивање примењивости на друге области може захтевати дефинисање нових SMT теорија, као и развој одговарајућих процедура одлучивања. У том смислу, у овој тези разматрамо унапређивање SMT решавача у правцу проширивања њихове примењивости на решавање CSP проблема. Ови проблеми подразумевају коначне домене за задате промењљиве, а циљ је придружити вредности промењљивама тако да буду задовољена сва задата ограничења. Када је у питању решавање CSP проблема, главни недостатак постојећих SMT теорија је то што нису прилагођене проблемима са коначним доменима, а немају ни могућност резоновања о глобалним ограничењима која играју значајну улогу у већини CSP проблема. Због тога је у овој тези дефинисана нова SMT теорија која омогућава природно изражавање CSP проблема, јер укључује подршку за нека често коришћена глобална ограничења. За ову теорију развијена је процедура одлучивања заснована на постојећим техникама позајмљеним из традиционалних CSP решавача, али адаптираним тако да се могу користити у оквиру SMT решавача. Процедура одлучивања у овом тренутку подржава два типа глобалних ограничења: `alldifferent` ограничење и линеарно ограничење. У случају `alldifferent` ограничења, најзначајнији допринос је нови алгоритам за генерисање објашњења пропагација и конфликата, што је неопходно у процесу анализе конфликата. У случају линеарног ограничења, развијен је нови алгоритам филтрирања који узима у обзир постојање `alldifferent` ограничења у датом проблему, што може појачати моћ пропагације. Други правац унапређивања SMT решавача који се разматра у тези је коришћење техника паралелизације. Разматра се паралелизација симплекс алгоритма који се у SMT решавачима користи као процедура одлучивања за линеарну аритметику. Такође се разматра и

паралелни портфолио, као и хибридизација ова два приступа. Даје се опсежна експериментална евалуација на великом броју, како индустријских, тако и случајно генерисаних инстанци. За потребе свих наведених истраживања, у току рада на тези развијен је и нови SMT решавач `argosmt` отвореног кода, заснован на DPLL($\mathcal{T}$) архитектури.

Кључне речи: рачунарство, аутоматско резоновање, SAT решавачи, SMT решавачи, CSP решавачи, паралелизација

Научна област: рачунарство

Ужа научна област: аутоматско резоновање

УДК број: 004.832.38:510.66(043.3)

Dissertation title: Improving SMT solvers using CSP techniques and parallelization techniques

Abstract: SMT solvers implement decision procedures that check for satisfiability of first-order logical formulae with respect to some decidable theory. They rely on the powerful techniques that are used for solving boolean satisfiability problem (SAT), combined with specific procedures that permit reasoning in a particular theory. Typical applications of SMT solvers are mostly found in software verification, and for this reason the theories frequently considered in SMT solvers are best suited for such kind of applications. Extending the applicability of SMT solvers to other areas may require defining new SMT theories, as well as developing the corresponding decision procedures for such theories. In that sense, in this thesis we consider improving of SMT solvers by extending their applicability to solving CSP problems. Such problems consider variables with finite domains, and the goal is to find the assignment of values to the variables of the problem satisfying all the imposed constraints. When CSP solving is considered, the main drawback of the existing SMT theories is that they are not adapted to the problems with finite domains, and are also incapable of reasoning about global constraints, which play the significant role in most CSP problems. For this reason, in this thesis a novel SMT theory is defined that enables natural representation of CSP problems, since it includes support for some frequently used global constraints. For such theory, we developed a decision procedure based on the existing techniques borrowed from the traditional CSP solvers, but these techniques are adapted in the way that permits their usage within SMT solvers. The decision procedure currently supports two types of global constraints: the `alldifferent` constraint and the linear constraint. In case of the `alldifferent` constraint, the main contribution is a novel algorithm for generating explanations of conflicts and propagations, which is required for the conflict analysis. In case of the linear constraint, a novel filtering algorithm is developed that takes into consideration the existence of the `alldifferent` constraints in the given problem, which can lead to a stronger propagation. Another direction of improving SMT solvers that is considered in the thesis is the usage of the parallelization techniques. We consider the parallelization of the simplex algorithm, which is used in SMT solvers as the decision procedure for the linear arithmetic. We also consider the parallel portfolio approach, as well as the hybridization of the two approaches. A comprehensive experimental evaluation is provided on a large

number of instances, both industrial and randomly generated. For the purpose of all the mentioned research, during the work on the thesis a new open-source SMT solver called `argosmt` is developed, based on DPLL($\mathcal{T}$) architecture.

Keywords: computer science, automated reasoning, SAT solvers, SMT solvers, CSP solvers, parallelization

Research area: computer science

Research sub-area: automated reasoning

UDC number: 004.832.38:510.66(043.3)

Предговор

SMT решавачи представљају веома моћан алат који се доминантно користи у верификацији софтвера, а теорије које се у оквиру SMT решавача разматрају су обично прилагођене управо тој области примене. И поред великог успеха који SMT решавачи постижу, и даље постоји много простора за њихово даље унапређивање. У току свог вишегодишњег изучавања области SMT решавача, препознао сам два основна правца даљег усавршавања ове технологије. Први правац јесте проширивање примењивости SMT решавача на друге врсте проблема који нису стриктно верификацијски, дефинисањем нових теорија и развојем одговарајућих процедура одлучивања (тзв. теоријских решавача). Конкретно, у оквиру ове тезе, бавио сам се пре свега могућношћу примене SMT решавача у решавању CSP проблема. Други правац унапређивања SMT решавача који сам изучавао је њихова паралелизација. Притом, циљ је био истражити могућности паралелизације скупих алгоритама који се у SMT решавачима користе, упоредити овај приступ са доминантним приступом који је заснован на паралелном портфолију, као и испитати могућности хибридизације ова два приступа. Закључци до којих сам дошао у оквиру својих истраживања изнети су у овој тези.

Велику захвалност дугујем свом ментору, професору Филипу Марићу. Заиста није фраза када кажем да ове докторске дисертације не би било без његове несебичне и истрајне подршке. Његови савети и сугестије као и обиље корисних знања и информација које сам од њега добио у многобројним дискусијама током претходних неколико година немерљиво су ми помогли да на прави начин усмерим своја истраживања. Изнад свега, оптимизам и позитивна енергија којом ме је обасипао и подизао у тренутцима када сам посустајао и губио наду је нешто без чега свакако никада не бих успео да овај тежак и мукотрпан посао завршим до краја. Нарочиту захвалност такође дугујем члановима комисије, професору Предрагу Јаничићу, професору Миодрагу Живковићу и професору

Зорану Огњановићу на детаљном читању текста и низу корисних сугестија које су свакако допринеле да ова теза на крају буде много боља и квалитетнија. Најзад, захвалио бих се и осталим професорима са Катедре за рачунарство и информатику са којима сам успешно сарађивао у протеклом периоду, као и мојим драгим колегама асистентима који су, будући да су пролазили кроз исту животну фазу, били пуни разумевања и подршке у тренутцима када ми је то заиста било неопходно.

Наравно, за овако озбиљан подухват какав је израда докторске дисертације није довољна само стручна и научна подршка ментора, професора и колега. Потребно је имати ослонац и подршку од стране породице и пријатеља, а ја сам ту неограничену подршку имао. Зато овде желим да се захвалим пре свега својим родитељима којима посвећујем ову тезу. Они су заслужни за мој одгој, васпитање и образовање, али пре свега за емотивну зрелост и моралне вредности којима су ме научили и учинили ме добрим и вредним човеком. Захваљујући њима сам и дошао у прилику да се ухватим у коштац са оваквим изазовом, а њихова подршка током израде ове тезе била је истрајна и безрезервна. Захваљујем се својој дугогодишњој девојци Данијели на огромном стрпљењу и разумевању, јер само она зна колико је било тешко заједно са мном преживљавати све успоне и падове, трпети мој хронични недостатак слободног времена, умор и расејаност. Имајући у виду сва њена одрицања у претходним годинама, може се слободно рећи да је ово наш заједнички успех. Захваљујем се својој сестри Милени, зету Душану, својим пријатељима Марку, Немањи, Марку, Срђану и свима осталима који су ме подржавали и заједно са мном радовали се појединачним успесима који су на крају довели до ове тезе.

Садржај

Глава 1

Увод

Решавање проблема *задовољивости у теорији* (енгл. *satisfiability modulo theory* (SMT)) [11] је област истраживања која је веома интензивно проучавана у претходним годинама. Задовољивост у теорији представља уопштење проблема *исказне задовољивости* (SAT) [17], где се уместо исказних формула разматрају формуле првог реда чија се задовољивост испитује у односу на неку унапред задату одлучиву теорију. Алати за испитивање задовољивости у теорији, популарно названи *SMT решавачи*, се у великој мери ослањају на постојеће SAT технологије, које укључују моћне технике попут учења клауза, нехронолошког враћања уназад и ефикасних стратегија гранања, а које омогућавају ефикасан обилазак простора претраге. Ове технике се у SMT решавачима комбинују са специфичним процедурама одлучивања које омогућавају паметно резоновање у задатој теорији. Оваква комбинација чини SMT решаваче веома моћним алатима за решавање различитих врста проблема. Убрзани развој SAT и SMT технологија је пре свега узрокован њиховом изузетном примењивошћу у индустрији, пре свега у области верификације софтвера, али и у другим областима.

Имајући у виду велику примењивост и изузетан значај SMT решавача, нарочита пажња се посвећује њиховом развоју и унапређивању коришћењем различитих техника. У овој тези разматрамо два правца унапређивања SMT решавача. Први правац се односи на проширивање примењивости SMT решавача изван своје уобичајене области примене у верификацији софтвера. Конкретно, у тези се разматра унапређивање SMT решавача када је у питању решавање *проблема задовољавања ограничења* (енгл. *constraint satisfaction problem* (CSP)) [85]. Актуелни SMT решавачи нису прилагођени решавању проблема ове врсте, зато што теорије које се уобичајено разматрају у оквиру SMT решавача

нису адекватне за изражавање CSP проблема. Ова неадекватност се огледа у томе што CSP проблеми обично подразумевају промењљиве са коначним доменима, док постојеће SMT теорије обично разматрају бесконачне или веома велике коначне домене. Такође, CSP проблеми најчешће укључују тзв. *глобална ограничења* са специфичном семантиком о којој се обично не може на прави начин резоновати у оквиру уобичајених теорија које се јављају у данашњим SMT решавачима. Због тога је неопходно дефинисати нову теорију која ће представљати природан логички оквир за изражавање CSP проблема, што подразумева подршку за директно представљање неких честих глобалних ограничења. За тако дефинисану теорију треба развити процедуру одлучивања коју би било могуће уградити у SMT решавач. У том смислу, најприроднији приступ који је разматран и у овој тези је да се приликом развоја процедуре одлучивања за овакву теорију искористе технике и алгоритми који су развијени у оквиру традиционалних CSP решавача. Ту се пре свега мисли на *алгоритме филтрирања* који омогућавају резоновање о глобалним ограничењима, узимајући у обзир њихову специфичну семантику. Велики изазов у том послу је свакако адаптација поменутих алгоритама тако да се могу користити у контексту SMT решавача. У овој тези се пре свега разматра адаптација одговарајућих алгоритама филтрирања за `alldifferent` ограничење [103], које захтева да све промењљиве наведене у ограничењу узму међусобно различите вредности, као и за *линеарно ограничење* над коначним доменима (попут $2x + 4y - z \leq 7$).

Други правац унапређивања SMT решавача који је разматран у тези јесте коришћење техника паралелизације, у циљу што бољег искоришћења савремених вишејезгарних и вишепроцесорских рачунара који су данас постали широко доступни. Најчешћи приступ паралелизацији SAT и SMT решавача је тзв. *паралелни портфолио* у коме се више различито подешених инстанци решавача истовремено покрећу над истом инстанцом проблема, а решавање се завршава оног тренутка када један од решавача пронађе решење [16, 50, 104, 57]. Овај приступ је релативно једноставан за имплементацију, а даје добре резултате у пракси. Други приступ је да се паралелизују поједини алгоритми велике временске сложености који се користе у оквиру SAT и SMT решавача. На пример, у оквиру савремених SAT решавача преко 80% времена извршавања одлази на алгоритам који обилази скуп клауза у потрази за конфликтима и јединичним пропагацијама. Паралелизација овог алгоритма би, самим тим, могла значајно убрзати рад SAT решавача [63]. Слична ситуација је и у случају SMT реша-

вача, с обзиром на то да се у оквиру процедура одлучивања за поједине теорије користе изузетно скупи алгоритми. Карактеристичан пример је *симплекс алгоритам* [29], који се у SMT решавачима користи као процедура одлучивања за линеарну аритметику [37]. У овој тези се управо разматра паралелизација симплекс алгоритма у оквиру SMT решавача. Најзад, разматра се и упоређивање ефективности симплекс паралелизације са паралелним портфолијом, а испитује се и могућност хибридизације ова два приступа.

Доприноси тезе. У наставку наводимо главне доприносе тезе:

- у оквиру рада на тези развијен је SMT решавач `argosmt` отвореног кода и флексибилног дизајна, што омогућава једноставно прилагођавање потребама, како текућих, тако и будућих истраживања у области развоја и примене SMT технологија. Отвореност кода решавача може бити од значаја и за ширу заједницу. Колико је аутору тезе познато, ово је први домаћи SMT решавач заснован на DPLL($\mathcal{T}$) архитектури [75], а његово постојање и јавна доступност свакако може охрабрити нове потенцијалне домаће истраживаче у области SMT решавача.

- формулисана је нова SMT теорија која је погодна за изражавање CSP проблема у оквиру SMT решавача. За тако дефинисану теорију развијена је процедура одлучивања заснована на техникама позајмљеним из традиционалних CSP решавача. Ове технике су адаптиране тако да ефикасно функционишу у оквиру SMT решавача. Ту се пре свега мисли на ефикасну комуникацију између алгоритама филтрирања који резонују о појединачним глобалним ограничењима.

- развијен је нови алгоритам за *генерисање објашњења* пропагација и конфликата код `alldifferent` ограничења. Алгоритми за генерисање објашњења су неопходни за потребе *анализе конфликата*, а њихов развој и прилагођавање SMT окружењу је управо највећи изазов при уградњи CSP алгоритама филтрирања за глобална ограничења у SMT решаваче. Развијени алгоритам је детаљно упоређен са другим релевантним приступима, а дата је и експериментална евалуација [4].

- развијен је нови алгоритам филтрирања за линеарна ограничења који узима у обзир постојање `alldifferent` ограничења у датом проблему и користи те информације за израчунавање јачих граница променљивих, чиме

се додатно сужава простор претраге. Алгоритам је детаљно упоређен са другим релевантним приступима, уз адекватну експерименталну евалуацију [6].

- имплементирана је паралелизација симплекс алгоритма, уграђена је подршка за паралелни портфолио, као и за хибридизацију ова два приступа. Урађена је и опсежна експериментална евалуација поменутих техника паралелизације на великом скупу како индустријских, тако и случајно генерисаних проблема [5].

Теза садржи велики број тврђења која су такође оригинални допринос тезе. Ипак, нису сва тврђења дата у тези оригинална. У оквиру појединих тврђења наведени су резултати који су од раније познати и који су преузети из литературе. Како бисмо јасно нагласили која тврђења нису оригинални допринос тезе, испред сваког тврђења преузетог из литературе дата је одговарајућа референца.

Структура тезе. Остатак тезе организован је на следећи начин. Глава 2 представља преглед основних појмова и термина који се у тези користе, као и приказ значајних резултата у области на које се ова теза ослања. У глави 3 описујемо структуру решавача `argosmt`, принцип његовог рада, као и основне карактеристике његовог дизајна. Глава 4 приказује резултате тезе везане за унапређивање SMT решавача CSP техникама. У оквиру ове главе описана је CSP теорија (поглавље 4.2), структура теоријског решавача за CSP теорију (поглавље 4.3), алгоритми за `alldifferent` ограничење, укључујући и поменути нови алгоритам за генерисање објашњења (поглавље 4.4), као и побољшани алгоритам филтрирања за линеарно ограничење у присуству `alldifferent` ограничења (поглавље 4.5). Глава 5 приказује резултате тезе везане за унапређивање SMT решавача техникама паралелизације. Најзад, у глави 6 дајемо закључке и наводимо неке правце будућег рада.

Глава 2

Основе

2.1 SAT и SMT проблем

2.1.1 Вишесортна логика првог реда

Вишесортна логика првог реда представља логички оквир за сва разматрања у овој тези. Дефиниција синтаксе и семантике вишесортне логике првог реда која следи у наставку је у највећој мери преузета из SMT-LIB стандарда [12]. Иако постоје и нешто другачије дефиниције у литератури, дефиниција дата у наставку је најједноставнија. Такође, и друга разматрања која се излажу у оквиру ове тезе су усклађена са терминологијом датом у SMT-LIB стандарду [12], тако да је то био додатни разлог за избор дефиниције у наставку.

Синтакса. Под *сигнатуром* подразумевамо уређену тројку $\Sigma = (\mathcal{S}, \mathcal{F}, r)$, где је $\mathcal{S}$ највише пребројив скуп *сорти*, $\mathcal{F}$ је највише пребројив скуп *функцијских симбола*, а $r : \mathcal{F} \to \mathcal{S}^* \times \mathcal{S}$ је пресликавање, при чему је са $\mathcal{S}^*$ означен скуп свих коначних n-торки сорти из $\mathcal{S}$ (укључујући и празну n-торку). За свако $f \in \mathcal{F}$, вредност $r(f) = ((s_1, \ldots, s_n), s)$ зовемо *рангом* функцијског симбола f. Уместо $((s_1, \ldots, s_n), s)$, користићемо запис $[s_1, \ldots, s_n] \to s$, при чему сорте $s_1, \ldots, s_n$ зовемо *сортама аргумената*, док сорту s зовемо *повратном сортом* функцијског симбола f. Симболе чији је ранг облика $[\,] \to s$ зовемо *константама* сорте s. Подразумевамо да постоји сорта $\texttt{Bool} \in \mathcal{S}$. Функцијске симболе чија је повратна сорта `Bool` зовемо и *предикатским симболима*. Даље, подразумевамо да је $\{\top, \bot, \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\} \subseteq \mathcal{F}$. Ове симболе зовемо *логичким симболима*. При том је $r(\top) = r(\bot) = [\,] \to \texttt{Bool}$, $r(\neg) = [\texttt{Bool}] \to \texttt{Bool}$,

$r(\wedge) = r(\vee) = r(\Rightarrow) = r(\Leftrightarrow) = [\texttt{Bool}, \texttt{Bool}] \to \texttt{Bool}$. Такође подразумевамо да за сваку сорту $s \in \mathcal{S} \setminus \{\texttt{Bool}\}$, постоји симбол $=_s \in \mathcal{F}$ чији је ранг $[s, s] \to \texttt{Bool}$ и који називамо *симболом једнакости*. Када је јасно из контекста о којој се сорти ради, уместо $=_s$ писаћемо само $=$. Поред сигнатуре, нека је за сваку сорту $s \in \mathcal{S}$ дат скуп $\mathcal{V}^s$ *променљивих* сорте s и нека је $\mathcal{V} = \bigcup_{s \in \mathcal{S}} \mathcal{V}^s$. *Израз* над сигнатуром Σ и скупом променљивих $\mathcal{V}$ дефинишемо на следећи начин:

- свака променљива $x \in \mathcal{V}^s$ је израз сорте s
- ако је $f \in \mathcal{F}$ и $r(f) = [s_1, \ldots, s_n] \to s$, а $t_1, \ldots, t_n$ су изрази чије су сорте редом $s_1, \ldots, s_n$, тада је $f(t_1, \ldots, t_n)$ израз сорте s. Специјално, константа сорте s је и израз сорте s.
- за сваку променљиву $x \in \mathcal{V}$ и израз F сорте $\texttt{Bool}$, $(\forall x)F$ и $(\exists x)F$ су такође изрази чија је сорта $\texttt{Bool}$

Сви изрази се могу добити искључиво коначном применом датих правила. Сваки израз има једнозначно одређену сорту. Изразе чија је сорта $\texttt{Bool}$ зовемо *формулама*, док изразе других сорти зовемо *термовима*. Симболе $\forall$ и $\exists$ зовемо, редом, *универзалним* и *егзистенцијалним* квантификатором. Приметимо да су на основу горње дефиниције $\top$ и $\bot$ формуле. Такође, ако су F и G формуле, тада су и $\neg F$, $F \wedge G$, $F \vee G$, $F \Rightarrow G$ и $F \Leftrightarrow G$ такође формуле. Слично, ако су u и v термови који имају исту сорту s, тада је $u =_s v$ формула. Симболе $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$ и $=_s$ записујемо инфиксно, као што је и уобичајено.[1] Формуле које не садрже логичке симболе ни квантификаторе зовемо *атомима*. Атоме и негације атома зовемо *литералима*. Специјално, атоме облика $u = v$ зовемо *једнакостима*, а њихове негације $\neg(u = v)$ *различитостима* (које обично записујемо као $u \neq v$). Дисјункција литерала назива се *клауза*. За формулу кажемо да је у *конјунктивној нормалној форми* (КНФ) ако је представљена као конјункција клауза.

Свако појављивање променљиве x у формули F може бити *слободно* или *везано* неким квантификатором. Ако формула не садржи квантификаторе, тада су у њој све променљиве слободне. Сва слободна појављивања променљиве x у формули F постају везана у формулама $(\forall x)F$ и $(\exists x)F$ тим квантификатором.

[1] Приликом записивања формула, подразумеваћемо да негација и квантификатори имају највиши приоритет, након чега следи конјункција, затим дисјункција, импликација и на крају еквиваленција која има најнижи приоритет. Приоритет се може променити заградама.

Сва слободна појављивања осталих променљивих (различитих од x) у F остају слободна и у формулама $(\forall x)F$ и $(\exists x)F$. Ако формула F не садржи слободне променљиве, тада кажемо да је F *затворена формула* (или *реченица*). Израз (терм или формула) који не садржи променљиве је *базни* израз.

Пример 2.1.1. Нека је дата сигнатура $\Sigma = (\mathcal{S}, \mathcal{F}, r)$, где је $\mathcal{S} = \{\texttt{Real}\}$, $\mathcal{F} = \{+, \cdot, \leq, 0, 1\}$ при чему је $r(+) = r(\cdot) = [\texttt{Real}, \texttt{Real}] \to \texttt{Real}$, $r(\leq) = [\texttt{Real}, \texttt{Real}] \to \texttt{Bool}$, и $r(0) = r(1) = [\,] \to \texttt{Real}$ (сорту `Bool`, логичке симболе, као ни симбол једнакости не наводимо експлицитно јер се њихово постојање у сигнатури подразумева). Нека је дат и скуп променљивих $\mathcal{V}^{\texttt{Real}} = \{x, y, z\}$ сорте `Real`. Термови $0+1$, $1 \cdot y$, $x+0 \cdot y$ и формуле $x+1 \leq z$, $(\forall x)(\forall y)(\forall z)((x+y)+z = x+(y+z))$ су примери исправних израза над овом сигнатуром, при чему су симболи $+, \cdot, \leq$ записани инфиксно ($\cdot$ има највиши приоритет , након чега следи $+$, док $\leq$ има најнижи приоритет).

Семантика. Под *структуром* над датом сигнатуром Σ (или Σ-*структуром*) подразумевамо пар $\mathcal{M} = (\mathcal{D}, \mathcal{I})$, где је $\mathcal{D} = \{D^s \mid s \in \mathcal{S}\}$ фамилија непразних скупова који су придружени сортама из $\mathcal{S}$ (скуп D^s зовемо *доменом* или *универзумом* сорте s), а $\mathcal{I}$ је *интерпретација* која сваком симболу $f \in \mathcal{F}$ ранга $[s_1, \ldots, s_n] \to s$ придружује функцију $f^{\mathcal{I}} : D^{s_1} \times \ldots \times D^{s_n} \to D^s$. Специјално, симбол a чији је ранг $[\,] \to s$ се интерпретира као фиксирани елемент $a^{\mathcal{I}} \in D^s$.

Сорта `Bool`, логички симболи, као и симбол једнакости се у свим структурама интерпретирају на исти начин. Домен сорте `Bool` је скуп $D^{\texttt{Bool}} = \{\texttt{true}, \texttt{false}\}$. Даље, важи да је $\top^{\mathcal{I}} = \texttt{true}$, $\bot^{\mathcal{I}} = \texttt{false}$. Симбол $\neg$ се интерпретира функцијом која има вредност `true` ако и само ако јој аргумент има вредност `false`. Симбол $\wedge$ се интерпретира функцијом која има вредност `true` ако и само ако су јој оба аргумента `true`. Симбол $\vee$ се интерпретира функцијом која има вредност `true` ако и само ако јој бар један од два аргумента има вредност `true`. Симбол $\Rightarrow$ се интерпретира функцијом која има вредност `false` ако и само ако јој је први аргумент `true` а други `false`. Симбол $\Leftrightarrow$ се интерпретира функцијом која има вредност `true` ако и само ако су јој или оба аргумента `true` или оба аргумента `false`. Најзад, за сваку сорту $s \in \mathcal{S} \setminus \{\texttt{Bool}\}$, симбол $=_s$ се интерпретира функцијом која има вредност `true` ако и само ако су јој оба аргумента исти елемент из D^s. Нека је, даље, дата $\mathcal{V}$-*валуација* $v : \mathcal{V} \to \bigcup_{s \in \mathcal{S}} D^s$ која свакој променљивој $x \in \mathcal{V}^s$ придружује вредност $v(x) \in D^s$ (тј. променљивама сорте s придружује вредности из D^s). Ако за две $\mathcal{V}$-валуације v и v' важи да је

$v(y) = v'(y)$ за сваку променљиву $y \neq x$, тада пишемо $v \sim_x v'$. За Σ-структуру $\mathcal{M}$ и $\mathcal{V}$-валуацију v, једнозначно је одређена *интерпретација*[2] *израза* e у $(\mathcal{M}, v)$ (у ознаци $I_v^{\mathcal{M}}(e)$) на следећи начин:

- за променљиву $x \in \mathcal{V}$, важи $I_v^{\mathcal{M}}(x) = v(x)$
- за константу a, важи $I_v^{\mathcal{M}}(a) = a^{\mathcal{I}}$
- за израз $f(t_1, \ldots, t_n)$, важи $I_v^{\mathcal{M}}(f(t_1, \ldots, t_n)) = f^{\mathcal{I}}(I_v^{\mathcal{M}}(t_1), \ldots, I_v^{\mathcal{M}}(t_n))$
- за формулу $(\forall x)F$ ће бити $I_v^{\mathcal{M}}((\forall x)F) = \texttt{true}$, ако и само ако за *сваку* валуацију v' за коју је $v' \sim_x v$, важи да је $I_{v'}^{\mathcal{M}}(F) = \texttt{true}$.
- за формулу $(\exists x)F$ ће бити $I_v^{\mathcal{M}}((\exists x)F) = \texttt{true}$, ако и само ако *постоји* валуација v' таква да је $v' \sim_x v$, за коју је $I_{v'}^{\mathcal{M}}(F) = \texttt{true}$.

Интерпретација израза e који не садржи слободне променљиве не зависи од $\mathcal{V}$-валуације v, већ само од изабране Σ-структуре $\mathcal{M}$. У том случају говоримо о *интерпретацији израза* у $\mathcal{M}$, а означавамо је са $I^{\mathcal{M}}(e)$. Специјално, то ће бити случај за све затворене формуле, као и за све базне изразе.

За Σ-структуру $\mathcal{M}$ и $\mathcal{V}$-валуацију v, формула F је *тачна* у $(\mathcal{M}, v)$ (у ознаци $(\mathcal{M}, v) \vDash F$) ако је $I_v^{\mathcal{M}}(F) = \texttt{true}$, у супротном је *нетачна* у $(\mathcal{M}, v)$. Затворена формула F је *тачна* у структури $\mathcal{M}$ (у ознаци $\mathcal{M} \vDash F$) ако је $I^{\mathcal{M}}(F) = \texttt{true}$, у супротном је *нетачна* у $\mathcal{M}$.

За формулу F кажемо да је *задовољива* ако постоји неко $(\mathcal{M}, v)$ за које је F тачна, у супротном је *незадовољива*, *контрадикторна* или *неконзистентна* (у ознаци $F \vDash \bot$). За формулу F кажемо да је *ваљана* (у ознаци $\vDash F$) ако је тачна за свако $(\mathcal{M}, v)$. Кажемо да је формула F *логичка последица* скупа формула Δ (у ознаци $\Delta \vDash F$), ако је F тачна за свако $(\mathcal{M}, v)$ за које су тачне и све формуле из Δ. Две формуле F и G су *логички еквивалентне* (у ознаци $F \equiv G$) ако имају исте интерпретације за свако $(\mathcal{M}, v)$. Проблем испитивања задовољивости (и ваљаности) произвољне формуле првог реда је неодлучив.

Пример 2.1.2. Под претпоставком да имамо исту сигнатуру као у примеру 2.1.1, формула $(\forall x)(\forall y)(x \leq y \vee \neg(x \leq y))$ је ваљана, јер је тачна у свим структурама. Са друге стране, формула $(\forall x)(\forall y)(\forall z)((x + y) + z = x + (y + z))$ је тачна у

[2]Термин *интерпретација* се користи и за пресликавање $\mathcal{I}$ структуре $\mathcal{M}$ које одређује значење *симбола* сигнатуре Σ, али и за пресликавање $I_v^{\mathcal{M}}$ које одређује значење *израза* у конкретној структури $\mathcal{M}$ и валуацији v. Ово не би требало да доведе до забуне, с обзиром на то да ће увек бити јасно да ли се говори о симболу или о изразу.

структури $\mathcal{R}$ у којој се сорта `Real` интерпретира скупом реалних бројева $\mathbb{R}$, а симболи $0, 1, +, \cdot, \leq$ се интерпретирају на уобичајен начин. Отуда је ова формула задовољива. Ипак, она није ваљана: ако бисмо симбол $+$ интерпретирали као операцију одузимања у скупу реалних бројева, тада ова формула у таквој структури не би била тачна.

2.1.2 Исказна логика и SAT проблем

Исказна логика се може сматрати фрагментом вишесортне логике првог реда, где сигнатура Σ садржи једино сорту `Bool`, а од функцијских симбола, поред логичких симбола, садржи још једино константе сорте `Bool` (које зовемо и *исказним словима*, или *исказним атомима*). Такође, подразумевамо да су све формуле које разматрамо базне, тј. скуп променљивих $\mathcal{V}$ је празан. Структуре над оваквим сигнатурама зовемо и *исказним валуацијама*. За фиксирану формулу, валуација има коначно много (2^n, где је n број различитих исказних слова у формули). Проблем испитивања задовољивости исказне формуле зове се *SAT проблем* [17]. Овај проблем је одлучив и NP комплетан [28].

Пример 2.1.3. Нека је сигнатура задата скупом исказних слова $\{p, q, r\}$. Једна исказна формула над овом сигнатуром је нпр. $(\neg p \wedge q) \Rightarrow \neg r$. Ова формула је задовољива (нпр. тачна је у било којој валуацији у којој се p интерпретира као `true`), али није ваљана (нпр. нетачна је у валуацији у којој се p интерпретира као `false`, а q и r као `true`).

2.1.3 Теорије првог реда и SMT проблем

Теорија првог реда $\mathcal{T}$ над датом сигнатуром $\Sigma = (\mathcal{S}, \mathcal{F}, r)$ је скуп Σ-структура.[3] Структуре које припадају теорији зовемо и *моделима* теорије $\mathcal{T}$. Ако се сорта $s \in \mathcal{S}$ у оквиру теорије $\mathcal{T}$ може интерпретирати произвољним непразним скупом, тада кажемо да је сорта s *неинтерпретирана* или *слободна* у теорији $\mathcal{T}$. Слично, ако се функцијски симбол $f \in \mathcal{F}$ ранга $[s_1, \ldots, s_n] \to s$ (или константа $a \in \mathcal{F}$ сорте s) у оквиру теорије $\mathcal{T}$ може интерпретирати произвољном функцијом $f^{\mathcal{I}} : D^{s_1} \times \ldots \times D^{s_n} \to D^s$ (односно произвољним елементом скупа D^s), где су $D^{s_1}, \ldots, D^{s_n}, D^s$ домени одговарајућих сорти, тада кажемо да

[3] Поред овакве *семантичке* дефиниције теорије првог реда, често се користи и *синтаксно-дедуктивна* дефиниција теорије $\mathcal{T}$ као скупа свих реченица које се могу доказати у оквиру неког дедуктивног система полазећи од задатог скупа *аксиома*. Семантичка дефиниција је нешто општија, јер обухвата и теорије које се не могу аксиоматски задати.

је f (односно a) *неинтерпретиран* или *слободан* у теорији $\mathcal{T}$. За формулу F над сигнатуром Σ кажемо да је *ваљана у теорији* $\mathcal{T}$ (у ознаци $\models_{\mathcal{T}} F$) ако је тачна за свако $(\mathcal{M}, v)$, где је $\mathcal{M}$ модел теорије $\mathcal{T}$, а v произвољна валуација. Формула F је *задовољива у теорији* $\mathcal{T}$ ако постоји модел $\mathcal{M}$ теорије $\mathcal{T}$ и валуација v таква да је F тачна у $(\mathcal{M}, v)$. Формула F је *логичка последица скупа* Δ *у теорији* $\mathcal{T}$ (у ознаци $\Delta \models_{\mathcal{T}} F$) ако је тачна за свако $(\mathcal{M}, v)$ за које су тачне и све формуле из Δ, при чему је $\mathcal{M}$ модел теорије $\mathcal{T}$, а v је произвољна валуација. Формуле F и G су *логички еквивалентне у теорији* $\mathcal{T}$ (у ознаци $F \equiv_{\mathcal{T}} G$) ако за свако $(\mathcal{M}, v)$ имају једнаке интерпретације, при чему је $\mathcal{M}$ модел теорије $\mathcal{T}$, а v је произвољна валуација. Проблем испитивања задовољивости у теорији $\mathcal{T}$ се зове *SMT проблем* за теорију $\mathcal{T}$ (енгл. *satisfiability modulo theory*) [11]. Одлучивост овог проблема (као и његова сложеност) зависи од саме теорије $\mathcal{T}$. Најчешће се ограничавамо на SMT проблеме за *базне фрагменте* теорија, тј. разматрамо одговарајући проблем само за базне формуле. У том случају се SMT проблем своди на испитивање да ли се слободне сорте и симболи могу у теорији $\mathcal{T}$ интерпретирати тако да формула буде тачна.

Пример 2.1.4. Претпоставимо поново да имамо сигнатуру као у примеру 2.1.1, тј. нека је $\mathcal{S} = \{\texttt{Real}\}$ и $\mathcal{F} = \{0, 1, +, \cdot, \leq\}$. Нека је, даље, $\mathcal{V}^{\texttt{Real}} = \{x, y, z\}$. Претпоставимо да теорија $\mathcal{T}$ има само један модел — структуру $\mathcal{R}$ у којој се сорта `Real` интерпретира као скуп реалних бројава $\mathbb{R}$, а симболи $0, 1, +, \cdot, \leq$ се интерпретирају на уобичајен начин. Формула $(\forall x)(\forall y)(\forall z)((x + y) + z = x+(y+z))$ је ваљана у теорији, јер је тачна у једином моделу теорије $\mathcal{R}$. Са друге стране, формуле $x \leq x$ и $1 + x \leq y$ нису затворене и њихова интерпретација зависи и од избора $\mathcal{V}$-валуације v. Формула $x \leq x$ је у моделу $\mathcal{R}$ тачна за сваку $\mathcal{V}$-валуацију v, па је ваљана у теорији $\mathcal{T}$. Формула $1 + x \leq y$ није ваљана у $\mathcal{T}$, али јесте задовољива у $\mathcal{T}$, јер постоји $\mathcal{V}$-валуација v за коју је $(\mathcal{R}, v) \models 1+x \leq y$ (нпр. таква је $\mathcal{V}$-валуација v у којој је $v(x) = 5$ и $v(y) = 8$).

Пример 2.1.5. У претходном примеру је теорија имала само један модел, али су формуле које нису затворене (попут $1 + x \leq y$) могле да буду интерпретиране на више начина, у зависности од избора $\mathcal{V}$-валуације. Други приступ који се често користи је да се x и y тумаче као неинтерпретиране константе. Формално, разматрајмо исту сигнатуру као у претходном примеру која је проширена додатним симболима константи x, y, z сорте `Real` (тј. сада су x, y и z константе, а скуп променљивих $\mathcal{V}^{\texttt{Real}}$ је празан). Нека скуп модела теорије $\mathcal{T}$ садржи све структуре у којима се сорта `Real` интерпретира као скуп реалних бројева $\mathbb{R}$,

а симболи $0, 1, +, \cdot, \leq$ се интерпретирају на уобичајен начин. Приметимо да у овом случају скуп модела теорије $\mathcal{T}$ није једночлан, јер је могуће варирати интерпретације слободних константи x, y и z. Сада је формула $1 + x \leq y$ базна формула која је задовољива у теорији $\mathcal{T}$, јер постоји модел теорије у коме је формула тачна (то је, нпр. модел у коме је константа x интерпретирана бројем 5, а y бројем 8). Слично, формула $x \leq x$ је ваљана у теорији $\mathcal{T}$, јер је тачна у свим моделима теорије.

Из претходна два примера видимо да, када је у питању задовољивост или ваљаност у теорији, не постоји суштинска разлика између слободних променљивих и неинтерпретираних константи одговарајуће сорте. Због тога су оба наведена приступа еквивалентна и равноправно се користе у литератури. У овој тези, ми ћемо се држати другог приступа, што значи да ћемо уместо слободних променљивих увек подразумевати неинтерпретиране константе.[4] Ово значи да ће сигнатуре које ћемо разматрати, поред експлицитно наведених интерпретираних симбола, увек подразумевано садржати и довољан број неинтерпретираних константи одговарајућих сорти, што најчешће нећемо експлицитно наводити. Такође, када кажемо да нека теорија за модел има неку структуру $\mathcal{M}$, подразумеваћемо да је заправо у питању фамилија структура које се добијају из $\mathcal{M}$ варирањем интерпретација неинтерпретираних константи.

2.1.4 Комбинација теорија првог реда

За сигнатуре $\Sigma_1 = (\mathcal{S}_1, \mathcal{F}_1, r_1)$ и $\Sigma_2 = (\mathcal{S}_2, \mathcal{F}_2, r_2)$ кажемо да су *компатибилне*, ако је $\mathcal{S}_1 = \mathcal{S}_2$ и за свако $f \in \mathcal{F}_1 \cap \mathcal{F}_2$ важи $r_1(f) = r_2(f)$. Интуитивно, ово значи да сигнатуре имају исте скупове сорти, а заједнички функцијски симболи имају исте рангове у обе сигнатуре. За сигнатуру $\Sigma = (\mathcal{S}, \mathcal{F}, r)$ кажемо да је *садржана* у сигнатури $\Sigma' = (\mathcal{S}', \mathcal{F}', r')$ (у ознаци $\Sigma \subseteq \Sigma'$), ако важи $\mathcal{S} = \mathcal{S}'$, $\mathcal{F} \subseteq \mathcal{F}'$ и $r(f) = r'(f)$ за свако $f \in \mathcal{F}$ (приметимо да су тада Σ и Σ' компатибилне сигнатуре). За сигнатуре $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ такве да су сваке две међусобно компатибилне, њихова *комбинована сигнатура* (у ознаци $\Sigma_1 + \Sigma_2 + \ldots + \Sigma_n$) је минимална сигнатура Σ (у смислу релације $\subseteq$) таква да је $\Sigma_i \subseteq \Sigma$ за свако $i \in \{1, \ldots, n\}$.

Ако је $\Sigma \subseteq \Sigma'$, тада за Σ-структуру $\mathcal{M}$ кажемо да је Σ-*сужење* Σ'-структуре $\mathcal{M}'$ ако се све сорте и функцијски симболи из Σ интерпретирају на исти на-

[4] У овом приступу, променљиве се ипак могу појављивати у формулама, али искључиво као везане променљиве. С обзиром на то да у овој тези не разматрамо квантификаторе, све формуле којима ћемо се бавити ће, самим тим, бити базне формуле.

чин у $\mathcal{M}$ и $\mathcal{M}'$. Нека су $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$ теорије над компатибилним сигнатурама $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$, редом. Под њиховом комбинацијом (у ознаци $\mathcal{T}_1 + \mathcal{T}_2 + \ldots + \mathcal{T}_n$) подразумевамо теорију $\mathcal{T}$ над сигнатуром $\Sigma = \Sigma_1 + \Sigma_2 + \ldots + \Sigma_n$, такву да је произвољна Σ-структура $\mathcal{M}$ модел теорије $\mathcal{T}$ ако и само ако је њено Σ_i-сужење модел теорија $\mathcal{T}_i$ за свако $i \in \{1, \ldots, n\}$.

Пример 2.1.6. Нека је дата сигнатура $\Sigma_1 = (\mathcal{S}_1, \mathcal{F}_1, r_1)$, где је $\mathcal{S}_1 = \{\texttt{Real}\}$, $\mathcal{F}_1 = \{+, x, y, z\}$, $r_1(+) = [\texttt{Real}, \texttt{Real}] \to \texttt{Real}$ и $r_1(x) = r_1(y) = r_1(z) = [\,] \to \texttt{Real}$. Нека је, даље, дата и друга сигнатура $\Sigma_2 = (\mathcal{S}_2, \mathcal{F}_2, r_2)$, где је $\mathcal{S}_2 = \{\texttt{Real}\}$, $\mathcal{F}_2 = \{x, y, z, f\}$, $r_2(x) = r_2(y) = r_2(z) = [\,] \to \texttt{Real}$, $r_2(f) = [\texttt{Real}] \to \texttt{Real}$. Ове две сигнатуре су компатибилне, јер им се скупови сорти поклапају, као и рангови заједничких симбола. Комбинована сигнатура $\Sigma = \Sigma_1 + \Sigma_2$ садржи сорту `Real` и функцијске симболе $x, y, z, +, f$ са ранговима наслеђеним из сигнатура Σ_1 и Σ_2. Формула $f(x + y) = z$ је пример формуле над комбинованом сигнатуром.

Нека су модели теорије $\mathcal{T}_1$ над сигнатуром Σ_1 све структуре које сорту `Real` интерпретирају скупом реалних бројева $\mathbb{R}$, а симбол $+$ операцијом сабирања у скупу реалних бројева (константе x, y и z су неинтерпретиране). Нека је, даље, $\mathcal{T}_2$ теорија над сигнатуром Σ_2 чији су модели све структуре над овом сигнатуром, тј. сорта `Real` и сви функцијски симболи су неинтерпретирани. Комбинована теорија $\mathcal{T}_1 + \mathcal{T}_2$ има за моделе све $(\Sigma_1 + \Sigma_2)$-структуре које сорту `Real` интерпретирају скупом реалних бројева, симбол $+$ операцијом сабирања у скупу реалних бројева, константе x, y, z произвољним реалним бројевима, а симбол f произвољном функцијом која пресликава $\mathbb{R}$ у $\mathbb{R}$. Лако се види да су одговарајућа сужења ових структура на сигнатуре Σ_1 и Σ_2 редом модели теорија $\mathcal{T}_1$ и $\mathcal{T}_2$.

2.1.5 SAT и SMT решавачи

Већина модерних SAT решавача је заснована на DPLL процедури (*Davis-Putnam-Logemann-Loveland*) [30]. Ова процедура се примењује на формуле у КНФ-у, а заснива се на претрази са враћањем (енгл. *backtracking*) уз додатна правила исказног резоновања, попут *једничних пропагација* (енгл. *unit propagation*) и елиминације *чистих литерала* (енгл. *pure literal*). Оригинални алгоритам развијен 1962. године био је дефинисан рекурзивно — најпре се неки од корака примени на формулу чиме се изврши њено поједностављивање, а затим се алгоритам примени рекурзивно на поједностављену формулу (или на

две формуле, у случају корака гранања). Модерне имплементације су по правилу итеративне, а поред тога имају и бројна алгоритамска и имплементациона побољшања. Тако побољшани алгоритми се често називају CDCL алгоритми (скраћеница од *conflict-driven-clause-learning*) [65].

Са друге стране, модерни SMT решавачи се најчешће заснивају на тзв. *лењом приступу* [11]. Идеја је да се најпре у датој формули сви атоми првог реда посматрају као исказни атоми (занемарујући њихову унутрашњу структуру), тј. разматра се само исказна структура дате формуле првог реда (тзв. *исказна апстракција*). Задовољивост овакве исказне формуле испитује се помоћу неког SAT решавача, а затим се за добијену задовољавајућу исказну валуацију (ако постоји) проверава задовољивост одговарајуће конјункције литерала првог реда у датој теорији помоћу неке специфичне процедуре одлучивања. Овакви решавачи се типично састоје из SAT решавача и скупа процедура одлучивања за различите подржане теорије. SAT решавач се бави исказном структуром формуле, што подразумева гранање, претрагу и исказно резоновање (у чему су модерни SAT решавачи веома ефикасни). Процедуре одлучивања (које се још зову и *теоријски решавачи*) се баве резоновањем у конкретним теоријама, тј. испитују да ли постоји модел теорије који истовремено задовољава све литерале у датој конјункцији. Најпознатија софтверска архитектура заснована на лењом приступу је позната под називом DPLL($\mathcal{T}$) [75], што имплицира да је SAT решавач заснован на DPLL процедури (заправо, најчешће на CDCL процедури).

У оквиру SMT решавача могуће је разматрати и комбинације теорија првог реда. Најчешћи приступ је да се адекватним комбиновањем постојећих процедура одлучивања за неке подржане теорије $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$ добије процедура одлучивања за комбиновану теорију $\mathcal{T}_1 + \mathcal{T}_2 + \ldots + \mathcal{T}_n$. Две најчешће коришћене методе за комбиновање процедура одлучивања су *Нелсон-Опенова* процедура (енгл. *Nelson-Oppen* (NO)) [69], и такозвана *одложена комбинација теорија* (енгл. *delayed theory combination* (DTC)) [22]. У оба случаја, претпоставља се да су теорије које се комбинују над међусобно *дисјунктним* сигнатурама. За две сигнатуре кажемо да су *дисјунктне* ако осим логичких симбола и симбола једнакости немају других заједничких функцијских симбола.[5] Друга битна претпоставка је да су теорије које се комбинују *стабилно бесконачне*. За

[5]Приметимо да, у складу са овом дефиницијом, дисјунктне сигнатуре могу имати заједничке сорте. Ово значи да можемо имати дисјунктне сигнатуре које су међусобно компатибилне, тј. скупови сорти им се поклапају, што је неопходно за комбиновање сигнатура.

теорију кажемо да је *стабилно бесконачна* ако је свака формула која је задовољива у тој теорији задовољива и у неком бесконачном моделу те теорије. Може се показати [101, 24] да су наведене две претпоставке довољне за коректност обе наведене методе комбиновања.

2.1.6 DPLL($\mathcal{T}$) архитектура

У овом одељку описујемо начин рада типичног SMT решавача заснованог на DPLL($\mathcal{T}$) архитектури [75] који допушта комбиновање процедура одлучивања DTC методом [22]. Претпоставка је да је улаз базна КНФ формула Ψ првог реда над сигнатуром Σ неке теорије $\mathcal{T}$, при чему теорија $\mathcal{T}$ може бити комбинација стабилно бесконачних теорија $\mathcal{T}_1, \ldots, \mathcal{T}_n$ над компатибилним, међусобно дисјунктним сигнатурама $\Sigma_1, \ldots, \Sigma_n$, редом. Излаз решавача је или *sat* (ако је формула задовољива у $\mathcal{T}$) или *unsat* (ако је формула незадовољива у $\mathcal{T}$). У случају задовољиве формуле, решавач може на излазу приказати и интерпретације произвољних израза у неком моделу теорије који задовољава улазну формулу Ψ.

Као што је већ речено, основу DPLL($\mathcal{T}$)-заснованог SMT решавача чини CDCL SAT решавач. Поред тога, за сваку од теорија $\mathcal{T}_i$ постоји и један *теоријски решавач* (или $\mathcal{T}_i$-*решавач*) који имплементира процедуру одлучивања за испитивање задовољивости произвољне конјункције литерала у теорији $\mathcal{T}_i$. Ако је дата конјункција незадовољива у $\mathcal{T}_i$, тада $\mathcal{T}_i$-решавач мора да врати подскуп литерала из дате конјункције који је узрок незадовољивости (тзв. *објашњење*). У случају задовољиве конјункције, $\mathcal{T}_i$-решавач *не мора* да пронађе конкретан модел теорије који задовољава дату конјункцију (поједине процедуре одлучивања су синтаксно-дедуктивне природе и само утврђују постојање таквог модела без његовог ефективног одређивања). Ипак, могућност проналажења неког задовољавајућег модела је једна од пожељних особина теоријског решавача. Осим тога, пожељно је да решавач има и неке друге особине, попут *инкременталности* (могућност ефикасног испитивања задовољивости конјункције након што се додају нови литерали, без поновног покретања процедуре из почетка), подршке за *теоријске пропагације* (могућност детекције логичких последица у теорији), подршке за *теоријске леме* (могућност генерисања клауза ваљаних у теорији) и слично [75]. Комуникација између SAT решавача и теоријских решавача се остварује преко унапред дефинисаног интерфејса, што омогућава једноставно додавање нових теоријских решавача. Овај интерфејс

омогућава SAT решавачу да информише теоријске решаваче о променама исказне валуације, као и да од њих захтева информације о задовољивости текуће конјункције литерала у теорији. Један пример интерфејса теоријског решавача дат је у раду Ниевенхуиса и других [75].

У наредном тексту, са $atoms(\Sigma)$ и $lits(\Sigma)$ означавамо, респективно, све атоме, односно, све литерале над сигнатуром Σ. Слично, за формулу (или скуп формула) Ψ, са $atoms(\Psi)$ означавамо све атоме који се појављују у Ψ, а са $lits(\Psi)$ све литерале над атомима који се појављују у Ψ, тј. $lits(\Psi) = atoms(\Psi) \cup \{\neg\alpha \mid \alpha \in atoms(\Psi)\}$.

2.1.6.1 Пречишћавање

Уколико је теорија $\mathcal{T}$ комбинација теорија, тада је пре него што отпочне процес решавања потребно да се формула трансформише у *чист* облик. За базни израз e кажемо да је $\mathcal{T}_i$-*чист* (енгл. $\mathcal{T}_i$-*pure*) ако сви симболи који су садржани у e припадају сигнатури Σ_i теорије $\mathcal{T}_i$. Специјално, за једнакост $u = v$ кажемо да је $\mathcal{T}_i$-чиста ако су изрази u и v $\mathcal{T}_i$-чисти. Нека је даље e базни израз који не садржи симбол једнакости нити неки од логичких симбола (другим речима, сваки симбол који се појављује у e припада тачно једној од сигнатура). За подизраз e' у таквом изразу e кажемо да је *страни* (енгл. *alien*) подизраз у e ако водећи функцијски симбол f' подизраза e' припада сигнатури Σ_j, а сви функцијски симболи који су претци тог симбола f' у стаблу израза e припадају некој другој сигнатури Σ_i. За једнакост $u = v$ кажемо да је *мешовита* ако водећи симболи израза u и v припадају различитим сигнатурама. Најзад, за КНФ формулу Ψ кажемо да је *чиста* ако за сваки њен атом $\alpha \in atoms(\Psi)$ постоји теорија $\mathcal{T}_i$ за коју је α $\mathcal{T}_i$-чист.

Циљ поступка *пречишћавања* (енгл. *purification*) [60] је да се улазна формула Ψ трансформише у еквизадовољиву чисту формулу Ψ'. Ово се постиже тако што се сви страни изрази у атомима формуле Ψ замењују новоуведеним константама одговарајућих сорти. Такође, за мешовите једнакости, обе стране једнакости се замењују новоуведеним константама. Свака нова константа се додаје у све сигнатуре $\Sigma_1, \ldots, \Sigma_n$, чиме ове сигнатуре престају да буду дисјунктне. Ове константе зовемо *дељене* константе. Једнакости облика $a = b$, где су a и b дељене константе зовемо *дељене* једнакости. Са друге стране, једнакости облика $c = \overline{r}$, где је $\overline{r}$ пречишћени облик неког подизраза r израза e, а c је новоуведена константа којом је r замењен у e зовемо *дефиниционе* једнако-

сти. Ове једнакости се додају у формулу као јединичне клаузе. Овако добијена чиста формула Ψ' ће бити еквизадовољива са полазном формулом Ψ.

Пример 2.1.7. Нека су дате две теорије $\mathcal{T}_1$ и $\mathcal{T}_2$ над дисјунктним сигнатурама Σ_1 и Σ_2, и нека је над комбинованом сигнатуром $\Sigma_1+\Sigma_2$ дата формула $f(g(a+b)) \leq f(a) + f(b) + b \wedge a + a = f(b)$, при чему симболи $\leq, +, a, b$ припадају сигнатури Σ_1, а симболи f, g припадају сигнатури Σ_2. У атому $f(g(a+b)) \leq f(a)+f(b)+b$ страни подизрази су $f(g(a+b))$, $f(a)$ и $f(b)$, јер симболи $\leq$ и $+$ припадају Σ_1, док f припада Σ_2. У изразу $f(g(a+b))$ страни подизраз је $a+b$ (јер f и g припадају Σ_2, а $+$ је из Σ_1), док су у изразима $f(a)$ и $f(b)$ страни подизрази, респективно, a и b. Са друге стране, једнакост $a+a = f(b)$ је мешовита, јер су водећи симболи израза $a+a$ и $f(b)$ у различитим сигнатурама. Пречишћавањем се наведени страни изрази (као и обе стране мешовите једнакости) замењују новоуведеним дељеним константама, након чега се одговарајуће дефиниционе једнакости додају у формулу. Резултат пречишћавања је формула $c_1 \leq c_2+c_3+b \wedge c_4 = c_3 \wedge c_1 = f(g(c_5)) \wedge c_2 = f(c_6) \wedge c_3 = f(c_7) \wedge c_4 = a+a \wedge c_5 = a+b \wedge c_6 = a \wedge c_7 = b$. Ова формула је чиста, јер је сваки од атома у формули $\mathcal{T}_i$-чист за неку од теорија $\mathcal{T}_1$ или $\mathcal{T}_2$. Притом, дељена једнакост $c_4 = c_3$ је једини атом који је истовремено и $\mathcal{T}_1$-чист и $\mathcal{T}_2$-чист.

2.1.6.2 Правила за промену стања

Нека је Ψ чиста КНФ формула за коју испитујемо задовољивост у комбинованој теорији $\mathcal{T} = \mathcal{T}_1 + \ldots + \mathcal{T}_n$ над сигнатуром $\Sigma = \Sigma_1 + \ldots + \Sigma_n$, при чему су сигнатуре Σ_i међусобно дисјунктне, до на дељене константе које су уведене у поступку пречишћавања. Рад DPLL($\mathcal{T}$)-заснованог SMT решавача се може описати помоћу система *правила за промену стања* [42, 60]. Овде описујемо један такав систем правила који представља варијанту система датог у раду Крстића и Гоела [60], поједностављену и прилагођену за потребе ове тезе. *Стање* решавача се може представити четворком (F, A, M, C), где је F текући скуп клауза које морају бити задовољене, A је текући скуп атома ($atoms(F) \subseteq A$), M је *траг тврђења*, а C је *конфликтни скуп*. *Трагом тврђења* M представљена је парцијална исказна валуација која се формира у оквиру SAT решавача. Траг је организован као стек на коме се налазе литерали из $lits(A)$ за које сматрамо да су тачни у текућој парцијалној валуацији.[6] Ово

[6]Дакле, када кажемо *парцијална валуација*, тада мислимо на скуп (или конјункцију) свих литерала на трагу M. Уз злоупотребу нотације, парцијалну валуацију такође обележавамо

значи да ни за једно $\alpha \in A$ не важи да је и $\alpha \in M$ и $\neg\alpha \in M$. За литерал l кажемо да је *тачан* у M ако је $l \in M$. За литерал l кажемо да је *нетачан* у M ако за њему супротан литерал $\bar{l}$ важи $\bar{l} \in M$. Уколико ни l ни $\bar{l}$ нису у M, тада кажемо да је l *недефинисан* у M. Клауза $c \in F$ је нетачна у M ако је сваки од њених литерала нетачан у M. Са $l_1 \prec l_2$ означавамо да литерал l_1 претходи литералу l_2 на трагу M. Да бисмо омогућили враћање уназад, траг M је партиционисан на *нивое одлучивања* $M^{(i)}$ нумерисане почев од 0, што означавамо као $M = M^{(0)}|M^{(1)}|\ldots|M^{(k)}$, при чему последњи k-ти ниво зовемо и *текући ниво одлучивања*. Са $level(l)$ означавамо ниво одлучивања коме припада литерал l на трагу M. Префикс трага M који се састоји из свих нивоа одлучивања од нултог до m-тог (тј. $M^{(0)}|\ldots|M^{(m)}$) означавамо са $M^{[m]}$. Литерали којима почиње сваки нови ниво одлучивања (изузев нултог) су *литерали одлучивања* (тј. литерали који су резултат *претпоставки*), док су преостали литерали *изведени литерали* (тј. литерали који су резултат *пропагација*). Литерали одлучивања представљају тачке гранања у току претраге, тј. тачке у које се алгоритам може вратити и бирати алтернативне путање приликом враћања уназад. *Конфликтни скуп* C је или неконзистентни подскуп литерала из M (тј. подскуп такав да је $F \wedge C \vDash_{\mathcal{T}} \bot$) ако је такав подскуп пронађен, или специјална вредност no_cflct у супротном. Почетно стање решавача је (F_0, A_0, M_0, no_cflt), где је F_0 скуп клауза који одговара улазној формули Ψ, $A_0 = atoms(F_0)$, а M_0 је празан стек.

Правила за промену стања су дата на слици 2.1. Свако правило има скуп *премиса* (написаних изнад хоризонталне линије) и *акцију* по којој се мења стање решавача (написану испод хоризонталне линије). Правило `Decide` је правило гранања које успоставља нови ниво одлучивања у M, тј. литерал који се овим правилом поставља на M је литерал одлучивања. Правила `UnitPropagate` и $\texttt{TheoryPropagate}_i$ су правила закључивања, тј. литерали који се овим правилима постављају на M су изведени литерали. Притом, правило `UnitPropagate` изводи закључак на основу клауза из F (исказно резоновање), док правило $\texttt{TheoryPropagate}_i$ резонује на основу логичких последица у теорији $\mathcal{T}_i$. Под *конфликтом* подразумевамо ситуацију у којој је или нека клауза $c \in F$ нетачна у M (исказни конфликт), или је скуп литерала из M незадовољив у некој теорији $\mathcal{T}_i$ (конфликт у теорији). Када се препозна конфликт, правило `Conflict` (у случају исказног конфликта), односно $\texttt{TheoryConflict}_i$ (у случају конфликта

са M.

`Decide`: $$\frac{C = no\_cflct \quad l \in lits(A) \quad l, \bar{l} \notin M}{M := M \mid l}$$

`UnitPropagate`: $$\frac{C = no\_cflct \quad l \vee \bar{l}_1 \vee \ldots \vee \bar{l}_k \in F \quad l_1, \ldots, l_k \in M \quad l, \bar{l} \notin M}{M := Ml}$$

$\texttt{TheoryPropagate}_i$: $$\frac{C = no\_cflct \quad M \vDash_{\mathcal{T}_i} l \quad l, \bar{l} \notin M}{M := Ml}$$

`Conflict`: $$\frac{C = no\_cflct \quad \bar{l}_1 \vee \ldots \vee \bar{l}_k \in F \quad l_1, \ldots, l_k \in M}{C := \{l_1, \ldots, l_k\}}$$

$\texttt{TheoryConflict}_i$: $$\frac{C = no\_cflct \quad l_1, \ldots, l_k \vDash_{\mathcal{T}_i} \bot \quad l_1, \ldots, l_k \in M}{C := \{l_1, \ldots, l_k\}}$$

`Explain`: $$\frac{l \in C \quad l \vee \bar{l}_1 \vee \ldots \vee \bar{l}_k \in F \quad l_1, \ldots, l_k \prec l}{C := C \cup \{l_1, \ldots, l_k\} \setminus \{l\}}$$

$\texttt{TheoryExplain}_i$: $$\frac{l \in C \quad l_1, \ldots, l_k \vDash_{\mathcal{T}_i} l \quad l_1, \ldots, l_k \prec l}{C := C \cup \{l_1, \ldots, l_k\} \setminus \{l\}}$$

`Backjump`: $$\frac{C = \{l, l_1, \ldots, l_k\} \quad level(l) > m \geq level(l_i)}{C := no\_cflct \quad F := F \cup \{\bar{l} \vee \bar{l}_1 \vee \ldots \vee \bar{l}_k\} \quad M := M^{[m]}\bar{l}}$$

`IntroduceSharedEq`: $$\frac{a, b \in \bigcap_i \Sigma_i \quad a = b \notin A}{A := A \cup \{a = b\}}$$

$\texttt{IntroduceAtom}_i$: $$\frac{\alpha \in atoms(\Sigma_i) \quad \alpha \notin A}{A := A \cup \{\alpha\}}$$

$\texttt{TheoryLemma}_i$: $$\frac{c = l_1 \vee \ldots \vee l_k \quad l_1, \ldots, l_k \in lits(A) \cap lits(\Sigma_i) \quad c \notin F \quad \vDash_{\mathcal{T}_i} c}{F := F \cup \{c\}}$$

`Restart`: $$\frac{C = no\_cflct}{M := M^{[0]}}$$

`Forget`: $$\frac{C = no\_cflct \quad c \in F \quad F \setminus \{c\} \vDash_{\mathcal{T}} c}{F := F \setminus \{c\}}$$

Слика 2.1: DPLL($\mathcal{T}$) правила

у теорији) се примењује и тиме започиње *анализа конфликта*. На почетку анализе конфликта, конфликтни скуп C се иницијализује скупом литерала из M који је одговоран за конфликт. Овај скуп зовемо *објашњење конфликта*. У случају исказног конфликта, то је просто скуп негираних литерала клаузе за коју је утврђено да је нетачна у M. У случају конфликта у теорији $\mathcal{T}_i$, објашњење је било који подскуп $R \subseteq M$ такав да је $R \vDash_{\mathcal{T}_i} \bot$. Затим се изведени литерали из конфликтног скупа C *објашњавају* један по један помоћу правила `Explain` и $\texttt{TheoryExplain}_i$. Ово значи да се изведени литерал l из C замењује својим

објашњењем, тј. подскупом литерала из M који претходе литералу l на трагу M, а који за логичку последицу имају литерал l.[7] У случају литерала који је изведен правилом `UnitPropagate` објашњење се једноставно очитава из клаузе на основу које је правило примењено. У случају литерала l који је изведен правилом $\texttt{TheoryPropagate}_i$, објашњење је произвољан скуп R литерала из M који претходе литералу l такав да важи $R \vDash_{\mathcal{T}_i} l$. Поступак анализе конфликта се наставља све док се или не дође до празног конфликтног скупа (у ком случају је формула незадовољива), или се не достигне *тачка једнозначне импликације* (енгл. *unique implication point* (UIP)) [42]. У питању је стање у коме постоји неко m такво да су сви литерали из конфликтног скупа C на нивоима у M који су мањи или једнаки m, осим једног литерала l који је на нивоу који је строго већи од m (најчешће на текућем нивоу одлучивања). У том тренутку се примењује `Backjump` правило, чиме се решавач враћа на ниво одлучивања m, литерал $\bar{l}$ се додаје у M (као изведени литерал), а затим се наставља са претрагом. Притом се негацијом конфликтног скупа добија клауза која се додаје у скуп F. За ову клаузу кажемо да је *научена*, и она је увек последица скупа клауза из F у теорији $\mathcal{T}$.[8]

Правило `IntroduceSharedEq` је важно за имплементацију поступка комбинације теорија. Наиме, може се показати [101] да је за коректност комбиноване процедуре одлучивања неопходно и довољно да се процедуре одлучивања за појединачне теорије усагласе по питању *разврставања* дељених константи. Под *разврставањем* (енгл. *arrangement*) [101] скупа S дељених константи подразумевамо било коју релацију еквиваленције над скупом S. Запишимо литерале из M у облику конјункције $M_{\mathcal{T}_1} \wedge \ldots \wedge M_{\mathcal{T}_n}$, где је $M_{\mathcal{T}_i}$ конјункција свих литерала из M над сигнатуром Σ_i теорије $\mathcal{T}_i$. Парцијална валуација M ће бити задовољива у теорији $\mathcal{T} = \mathcal{T}_1 + \ldots + \mathcal{T}_n$ ако и само ако постоји разврставање $\sim$ скупа S такво

[7] Уколико конфликтни скуп $C = \{l_1, \ldots, l_n, l\}$ посматрамо као клаузу $\neg l_1 \vee \ldots \vee \neg l_n \vee \neg l$, а логичку последицу $l'_1, \ldots, l'_k \vDash_{\mathcal{T}} l$ као клаузу $\neg l'_1 \vee \ldots \vee \neg l'_k \vee l$, тада се објашњавање литерала l може разумети као примена *правила резолуције* над претходне две клаузе, чиме се добија клауза $\neg l_1 \vee \ldots \vee \neg l_n \vee \neg l'_1 \vee \ldots \vee \neg l'_k$, тј. конфликтни скуп $\{l_1, \ldots, l_n, l'_1, \ldots, l'_k\}$. Самим тим, цео поступак анализе конфликта се може разумети као извођење методом резолуције, чији је резултат нова клауза која је логичка последица постојећих клауза у теорији $\mathcal{T}$.

[8] У неким системима (попут система описаног у раду Крстића и Гоела [60]), правило `Backjump` не укључује учење клаузе. Уместо тога, уводи се посебно правило `Learn` које у сваком тренутку у току анализе конфликта може негацијом конфликтног скупа добити нову научену клаузу која се додаје у F. Иако таква варијанта правила даје већу флексибилност приликом учења, већина решавача ипак и даље учи клаузе искључиво непосредно пре примене `Backjump` правила. Због тога је у нашем систему правила учење обједињено са враћањем уназад, у виду јединственог `Backjump` правила.

да за сваку од теорија $\mathcal{T}_i$ постоји њен модел $\mathcal{M}_i$ који задовољава конјункцију $M_{\mathcal{T}_i}$ такав да за произвољне константе $a, b \in S$ важи $I^{\mathcal{M}_i}(a = b) = \texttt{true}$ ако и само ако је $a \sim b$ [101]. Овакво разврставање звaћемо *задовољиво разврставање*. У случају DTC методе комбиновања [22], задовољиво разврставање се формира инкрементално, а одређује се постављањем дељених једнакости $a = b$ или њихових негација $a \neq b$ на траг M. Ови литерали утичу на теоријске решаваче да се при испитивању задовољивости ограниче искључиво на моделе теорије који су сагласни са изабраним разврставањем. У оригиналној формулацији DTC методе [22] све овакве дељене једнакости се одмах на почетку додају у скуп атома A. Међутим, ефикасније је да се дељене једнакости у скуп атома A уводе *лењо*, тек када се за тим укаже потреба. У нашем систему правила, дељене једнакости се уводе у скуп атома A управо правилом `IntroduceSharedEq`.

Правилом $\texttt{IntroduceAtom}_i$ се могу уводити нови $\mathcal{T}_i$-чисти атоми за произвољну теорију $\mathcal{T}_i$, док се правилом $\texttt{TheoryLemma}_i$ уводе нове клаузе које су ваљане у $\mathcal{T}_i$. Ова два правила омогућавају имплементацију технике *гранања на захтев* (енгл. *splitting on demand*) [10]. Наиме, често је у оквиру теоријских решавача потребно разматрати различите случајеве како би се испитала задовољивост датог скупа литерала у некој теорији. Формално, ово резоновање по случајевима у теорији $\mathcal{T}_i$ се може изразити логичком последицом $l_1, \ldots, l_r \vDash_{\mathcal{T}_i} l'_1 \vee \ldots \vee l'_s$, где су $l'_1, \ldots, l'_s$ литерали помоћу којих су изражени одговарајући случајеви по којима треба гранати, а $\{l_1, \ldots, l_r\}$ је скуп претпоставки под којима дисјункција $l'_1 \vee \ldots \vee l'_s$ важи у теорији $\mathcal{T}_i$. На пример, уколико је $\mathcal{T}_i$ теорија реалне аритметике и ако дати скуп претпоставки садржи литерал $x \neq y$, тада мора да важи или $x < y$ или $x > y$, па се испитивање задовољивости датог скупа претпоставки може свести на разматрање ова два случаја. Ово се формално изражава логичком последицом $x \neq y \vDash_{\mathcal{T}_i} x < y \vee x > y$. Традиционални приступ је да се ово гранање по случајевима имплементира у оквиру самог теоријског решавача за теорију $\mathcal{T}_i$. Ефикаснији приступ који је превладао у новије време [10] је да се гранање по случајевима препусти SAT решавачу, који је знатно ефикаснији када је у питању имплементација претраге са враћањем. Ово се постиже тако што се одговарајућа логичка последица изрази у облику клаузе $\bar{l}_1 \vee \ldots \bar{l}_r \vee l'_1 \vee \ldots \vee l'_s$ која се научи од стране SAT решавача који је затим користи у наставку претраге. С обзиром на то да литерали $l'_1, \ldots, l'_s$ не морају бити у скупу литерала $lits(A)$ које познаје SAT решавач, може бити потребно да се претходно одговарајући атоми додају у скуп A. Притом, да би се гаранто-

вало заустављање алгоритма, неопходно је да се ограничи скуп потенцијалних атома који се правилом $\texttt{IntroduceAtom}_i$ могу додати у скуп A (тј. тај скуп мора бити коначан и унапред фиксиран [10]). Ипак, ово ограничење не представља суштински проблем за већину стандардних SMT теорија.

Правило $\texttt{TheoryLemma}_i$ се може користити и заједно са правилом IntroduceSharedEq као подршка комбинацији теорија у случају *неконвексних теорија* [22, 60]. За теорију $\mathcal{T}_i$ кажемо да је *конвексна* ако за сваку логичку последицу $M \models_{\mathcal{T}_i} \bigvee_{j=1}^{s} a_j = b_j$, где су $a_j = b_j$ дељене једнакости, постоји неко $j \in \{1, \ldots, s\}$ такво да важи $M \models_{\mathcal{T}_i} a_j = b_j$. У супротном, за теорију кажемо да је *неконвексна*. Интуитивно, код конвексних теорија логичке последице су увек појединачне дељене једнакости, тако да се одговарајући закључак увек може проследити другим теоријама правилом $\texttt{TheoryPropagate}_i$. Са друге стране, ако је теорија неконвексна, тада је у поступку комбиновања теорија понекад потребно разматрати дисјункције дељених једнакости. На пример, ако је $\mathcal{T}_i$ теорија целобројне аритметике, тада важи логичка последица $a \geq 1, a \leq 2, b \geq 1, b \leq 2, c \geq 1, c \leq 2 \models_{\mathcal{T}_i} a = b \vee a = c \vee b = c$, при чему из датог скупа претпоставки не следи ни једна од појединачних једнакости дате дисјункције. Разматрање различитих случајева се и овде може постићи тако што се идентификује подскуп литерала $\{l_1, \ldots, l_r\} \subseteq M$ такав да је $l_1, \ldots, l_r \models_{\mathcal{T}_i} a_1 = b_1 \vee \ldots \vee a_s = b_s$, а затим се правилом $\texttt{TheoryLemma}_i$ научи клауза $\neg l_1 \vee \ldots \vee \neg l_r \vee a_1 = b_1 \vee \ldots \vee a_s = b_s$ (претходно се правилом IntroduceSharedEq уведу оне дељене једнакости $a_j = b_j$ које већ нису у скупу A). Учењем ове клаузе омогућено је одговарајуће гранање по случајевима у наставку рада решавача.

Правило Restart се примењује периодично како би се решавач извукао из неперспективних грана простора претраге. Ово правило се такође мора примењивати пажљиво, како би се гарантовало заустављање [75]. Најзад, правило Forget се користи за заборављање научених клауза које се више не користе у резоновању, ради ослобађања меморијског простора и убрзавања процеса претраге.

Постоје две врсте *завршних стања*. Прва врста завршних стања су стања у којима је $C = \varnothing$. Ово значи да је $F \models_{\mathcal{T}} \bot$, па је формула незадовољива (решавач пријављује *unsat*).[9] Друга врста завршних стања су стања у којима

[9]Може се показати да се незадовољива завршна стања могу извести из било ког стања у коме постоји конфликт на нултом нивоу трага M (узастопном применом правила Explain и $\texttt{TheoryExplain}_i$ докле год је то могуће). Због тога многи решавачи пријављују *unsat* чим

је $C = no\_cflt$, није могуће применити правила `Conflict` и $\texttt{TheoryConflict}_i$, и ни један атом из скупа A није недефинисан у M. Ово значи да ни једна од клауза из F није нетачна у M, а такође постоје и модели теорија $\mathcal{T}_i$ који задовољавају конјункције одговарајућих литерала из M (што укључује и одабрано разврставање скупа дељених константи S које је одређено дељеним једнакостима и различитостима из M). Ово значи да постоји модел теорије $\mathcal{T}$ у коме је конјункција литерала из M тачна, па је самим тим и полазна формула Ψ задовољива у $\mathcal{T}$ (решавач пријављује *sat*).

2.1.7 SMT-LIB иницијатива

У претходним годинама развијен је велики број SMT решавача, а неки од најпознатијих су *Z3* [33], *Yices* [38], *BarcelogicTools* [19], *MathSAT* [27], *CVC* [8], *OpenSMT* [25] и други. У циљу боље координације развоја и лакшег поређења различитих SMT решавача, заједница истраживача је 2003. године покренула иницијативу за стандардизацију SMT решавача под називом *SMT-LIB*[10]. Стандард SMT-LIB [9] укључује прецизну дефиницију теорија које се користе у SMT решавачима, дефиницију стандардне синтаксе улазног и излазног језика, као и прецизну семантику логичког оквира у коме се цео проблем разматра. У тренутку писања овог текста, актуелни стандард је SMT-LIB 2.5 [9]. Такође, део SMT-LIB иницијативе је и велика колекција инстанци проблема који се могу користити за тестирање и упоређивање SMT решавача.

2.1.8 Неке стандардне SMT теорије

Иако теорија која се разматра може бити било која одлучива теорија првог реда, обично је акценат на оним теоријама које су примењиве у пракси. С обзиром на то да се SMT решавачи доминантно користе у верификацији софтвера, типичне SMT теорије су прилагођене тој области примене. Наводимо неке од најзначајнијих стандардних SMT теорија:

- *теорија једнакости са неинтерпретираним функцијским симболима* (енгл. *equality with uninterpreted functions* (EUF)) — модели ове теорије су

се догоди конфликт на нултом нивоу. Примена поступка анализе конфликта све до извођења празног конфликтног скупа је корисна у случају да желимо да добијемо комплетан резолуцијски доказ незадовољивости.

[10] `http://smtlib.cs.uiowa.edu/`

све структуре над датом сигнатуром, без ограничења, тј. интерпретације свих симбола су потпуно произвољне, изузев логичких симбола и симбола једнакости, који се интерпретирају онако како је раније описано (отуда и назив, јер су сви симболи у сигнатури неинтерпретирани). Самим тим, једино резоновање које је у овој теорији могуће је на основу особина релације једнакости. Ова теорија је неодлучива. Базни фрагмент ове теорије (означен са `QF_UF` у SMT-LIB стандарду) је одлучив у полиномијалном времену, а процедуре одлучивања које се обично користе су засноване на *конгруентним затворењима* [70, 73].

- *теорија реалне аритметике* — ова теорија као модел има стандардну структуру реалних бројева $\mathbb{R}$. Теорија је одлучива [100] у општем случају. У SMT-у је обично од интереса базни *линеарни*[11] фрагмент ове теорије (означен са `QF_LRA` у SMT-LIB стандарду). Испитивање задовољивости конјункција базних линеарних литерала над овом теоријом је одлучиво у полиномијалном времену. Ипак, чешће се користе методе засноване на симплекс алгоритму [29] које су експоненцијалне сложености у најгорем случају, али дају добре резултате у пракси [37].

- *теорија целобројне аритметике* — ова теорија за модел има стандардну структуру целих бројева $\mathbb{Z}$. Теорија је неодлучива у општем случају. Њен линеарни фрагмент (тзв. *Презбургерова аритметика*) је одлучив. У SMT-у се обично ограничавамо на базни линеарни фрагмент ове теорије (означен са `QF_LIA` у SMT-LIB стандарду). Испитивање задовољивости конјункција базних линеарних литерала над овом теоријом је одлучив и NP комплетан проблем. Процедуре одлучивања су обично засноване на поменутом симплекс алгоритму, уз разне додатне технике за проналажење целобројног решења [37, 48, 34].

- *теорија низова* — сигнатура ове теорије садржи три сорте `Index`, `Value` и `Array`, као и два функцијска симбола $read : [\texttt{Array}, \texttt{Index}] \to \texttt{Value}$ и $write : [\texttt{Array}, \texttt{Index}, \texttt{Value}] \to \texttt{Array}$, а модели ове теорије су све структуре које задовољавају следеће аксиоме:

  - $(\forall x)(\forall y)(\forall z)(read(write(x, y, z), y) = z)$

[11] Под *линеарним* аритметичким изразима подразумевамо изразе који не укључују операцију множења, осим ако је у питању множење целобројним коефициентом које у ствари представља скраћени запис за узастопно сабирање (нпр. $3x$ је скраћени запис за $x + x + x$).

– $(\forall x)(\forall y_1)(\forall y_2)(\forall z)(y_1 \neq y_2 \Rightarrow read(write(x, y_1, z), y_2) = read(x, y_2))$

– $(\forall x_1)(\forall x_2)((\forall y)(read(x_1, y) = read(x_2, y)) \Rightarrow x_1 = x_2)$

Ова теорија се типично користи у верификацији, у циљу апстракције меморијских операција. Базни фрагмент ове теорије (означен са `QF_AX` у SMT-LIB стандарду) је одлучив и NP комплетан [97]. Неке од процедура одлучивања дате су у литератури [97, 31, 18].

- *теорија коначних битвектора* — сигнатура ове теорије садржи фамилију сорти $\texttt{BitVec}_n$ (за свако $n \in \mathbb{N}$) које се интерпретирају скуповима свих вектора битова дате фиксиране дужине n. Сигнатура садржи и одређени број функцијских симбола који се интерпретирају као одговарајуће операције над битвекторима (нпр. $bvadd_n$ представља операцију сабирања два битвектора дужине n по модулу 2^n, док симбол $bvshl_n$ представља операцију померања у лево садржаја битвектора дужине n за дати број позиција). Сваки битвектор се по потреби може тумачити и као неозначен бинарни број и као означен број у потпуном комплементу, у зависности од тога која се операција над њим примењује (нпр. $bvult_n$ означава упоређивање битвектора као неозначених бројева, док $bvslt_n$ означава упоређивање битвектора као бројева у потпуном комплементу). Ова теорија омогућава апстракцију стварне хардверске аритметике. Базни фрагмент ове теорије (означен са `QF_BV` у SMT-LIB стандарду) је одлучив и NP комплетан [26, 23].

2.2 CSP проблем

Проблем задовољавања ограничења или *CSP проблем* (енгл. *constraint satisfaction problem*) је проблем одлучивања задат тројком $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, где је $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ коначан скуп *променљивих*, $\mathcal{D} = \{D_{x_1}, D_{x_2}, \ldots, D_{x_n}\}$ је скуп *коначних*[12] *домена*[13] (при чему је D_{x_i} домен променљиве x_i), а $\mathcal{C} =$

[12]Напоменимо да се у литератури може наћи и општија дефиниција CSP проблема у којој се не захтева да домени променљивих буду коначни [2]. С обзиром на то да се у овој тези разматрају искључиво CSP проблеми над коначним доменима, ми ћемо се ипак држати наведене дефиниције која подразумева коначне домене.

[13]Термин *домен* овде означава скуп вредности које може узети нека CSP променљива у датом CSP проблему и не треба га мешати са раније уведеним појмом домена као скупом којим се интерпретира нека сорта у логици првог реда.

$\{C_1, C_2, \ldots, C_m\}$ је коначан скуп *ограничења*. *Ограничење* $C \in \mathcal{C}$ над променљивама $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ (што означавамо са $C(x_{i_1}, x_{i_2}, \ldots, x_{i_k})$) је произвољни подскуп скупа $D_{x_{i_1}} \times D_{x_{i_2}} \times \ldots \times D_{x_{i_k}}$. Уколико је тај подскуп непразан, за ограничење кажемо да је *конзистентно*, а у супротном кажемо да је *неконзистентно*. Број k се зове *арност* ограничења C. *Решење* CSP проблема је било која n-торка $(d_1, d_2, \ldots, d_n)$ из $D_{x_1} \times D_{x_2} \times \ldots \times D_{x_n}$ таква да за свако ограничење $C \in \mathcal{C}$ над променљивама $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ k-торка $(d_{i_1}, d_{i_2}, \ldots, d_{i_k})$ припада C. CSP проблем је *конзистентан* ако има бар једно решење, а у супротном је *неконзистентан*. Два CSP проблема P_1 и P_2 су *еквивалентна* ако је свако решење проблема P_1 истовремено и решење проблема P_2 и обратно. У општем случају, CSP проблем је NP комплетан.[14] Детаљније информације о CSP проблему се могу наћи у литератури [85, 2].

Пример 2.2.1. Нека је дат CSP проблем $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, где је $\mathcal{X} = \{x, y, z, u, v\}$, $\mathcal{D} = \{D_x, D_y, D_z, D_u, D_v\}$, при чему је $D_x = \{1, 2, 3\}$, $D_y = \{1, 2\}$, $D_z = \{3, 5\}$, $D_u = \{2, 4, 6\}$, $D_v = \{1, 3, 5\}$, а скуп ограничења је $\mathcal{C} = \{C_1, C_2, C_3, C_4\}$, где је $C_1(x, y) = \{(1, 1), (3, 2), (3, 1), (2, 2)\}$, $C_2(x) = \{1, 2\}$, $C_3(x, y, z) = \{(1, 2, 5), (1, 2, 3), (2, 2, 5), (3, 2, 3)\}$, и $C_4(z, u, v) = \{(3, 2, 1), (3, 4, 5), (5, 4, 3), (5, 6, 1)\}$. Овај проблем је конзистентан, а два могућа решења су $(2, 2, 5, 4, 3)$ и $(2, 2, 5, 6, 1)$.

2.2.1 CSP решавачи

Процедуре за решавање CSP проблема зову се *CSP решавачи*. Решавање CSP проблема се обично састоји из *претраге*, и *пропагације ограничења*. Под *претрагом* се, формално, подразумева подела проблема P на два или више подпроблема $P_1, \ldots, P_n$ таквих да је скуп решења проблема P једнак унији скупова решења подпроблема $P_1, \ldots, P_n$. Проблем P ће у том случају бити конзистентан ако и само ако је бар један од подпроблема конзистентан. Подпроблеми се решавају један по један докле год се међу њима не пронађе конзистентан подпроблем, или се не утврди да су сви подпроблеми неконзистентни. Имплементација претраге се по правилу своди на претрагу са враћањем (енгл. *backtracking*).

Један од уобичајених начина да се проблем P подели на подпроблеме је да се разматрају различите вредности за неку изабрану променљиву x. Формално, под *придруживањем* вредности $d \in D_x$ променљивој x (у ознаци $x = d$)

[14] NP комплетност CSP проблема једноставно следи из његове припадности класи NP, као и из чињенице да је његов специјални случај – SAT проблем NP комплетан.

подразумевамо трансформацију проблема P којом се домен D_x сужава тако да садржи само вредност d. Проблем који се добија од проблема P придруживањем $x = d$ означавамо са $P[x = d]$. Уколико је оригинални домен променљиве x једнак $D_x = \{d_1, \ldots, d_n\}$, тада се проблем P може поделити на подпроблеме $P[x = d_1], \ldots, P[x = d_n]$, тј. покушавамо са придруживањем различитих вредности променљивој x, уз враћање (тј. поништавање придруживања) у случају неконзистентности.

Уколико су домени уређени, тада је чест начин поделе проблема P дељењем домена неке променљиве x на два дисјунктна подскупа, на основу релације поретка у домену. Ако је релација поретка означена са $\prec$, тада се проблем P може поделити на два подпроблема $P[x \prec d]$ и $P[x \not\prec d]$ за неку вредност $d \in D_x$, тј. најпре разматрамо само вредности за x које су мање од d, а затим, уколико не пронађемо решење, покушавамо са вредностима за x које нису мање од d.

Са друге стране, *пропагација ограничења* подразумева резоновање са циљем трансформације проблема P у еквивалентан проблем P' који је једноставнији. Ово се обично постиже поступком *филтрирања домена*, којим се проналазе и уклањају *неконзистентне вредности*, тј. вредности из домена променљивих које нису део ни једног од решења датог проблема. Формално, трансформација проблема P којом се из домена променљиве x уклања нека вредност d зове се *одсецање* (енгл. *pruning*), а означава се са $x \neq d$. Проблем настао из P одсецањем вредности d из домена променљиве x означавамо са $P[x \neq d]$. Филтрирање домена се обично обавља на нивоу појединачних ограничења (ређе на нивоу конјункција више ограничења), тј. за свако ограничење се одсецају оне вредности које су неконзистентне са тим ограничењем (нису део ни једног решења које задовољава то ограничење). Тако добијена одсецања се затим *пропагирају*, тј. намећу се другим ограничењима, што доводи до даљег филтрирања и нових пропагација. Овај поступак се наставља док се не постигне жељени *ниво конзистентности*.

2.2.2 Нивои конзистентности

Ограничење C над променљивама $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ има својство *конзистентности домена* (енгл. *domain consistency*) или *конзистентности лукова* (енгл. *arc consistency*) ако за сваку вредност $d_{i_r} \in D_{x_{i_r}}$ $(r \in \{1, \ldots, k\})$ постоје вредности $d_{i_s} \in D_{x_{i_s}}$ за свако $s \in \{1, \ldots, k\} \setminus \{r\}$, такве да је $(d_{i_1}, \ldots, d_{i_k}) \in C$.

Конзистентност домена је најјачи облик локалне конзистентности, јер подразумева уклањање свих вредности које су неконзистентне са датим ограничењем.

Под претпоставком да су домени променљивих уређени, ограничење C над променљивама $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ има својство *конзистентности граница* (енгл. *bound consistency*) ако за сваку вредност $d_{i_r} \in \{min(D_{x_{i_r}}), max(D_{x_{i_r}})\}$ ($r \in \{1, \ldots, k\}$) постоје вредности $d_{i_s} \in [min(D_{x_{i_s}}), max(D_{x_{i_s}})]$ за свако $s \in \{1, \ldots, k\} \setminus \{r\}$, такве да је $(d_{i_1}, \ldots, d_{i_k}) \in C$. Конзистентност граница је нешто слабија, јер не подразумева уклањање свих вредности неконзистентних са датим ограничењем, али се често може постићи ефикаснијим алгоритмом.

Кажемо да CSP проблем има својство *конзистентости домена* (*конзистентности граница*) ако то својство имају сва његова ограничења. Приметимо да конзистентност домена (а ни граница) CSP проблема не имплицира његову конзистентност (у смислу постојања решења) у општем случају.

За проблем кажемо да је у *решеној форми* ако има својство конзистентности домена, а домени свих променљивих су једночлани. Може се показати да је у том случају проблем конзистентан, а његово једино решење је управо одређено јединственим вредностима у доменима променљивих. Поступак решавања CSP проблема се обично завршава тако што претрагом и пропагацијом ограничења дођемо до подпроблема у решеној форми, или тако што утврдимо да такав подпроблем не постоји.

2.2.3 Глобална ограничења

Ограничења се у CSP проблемима често изражавају помоћу релација над променљивама проблема. Поред уобичајених математичких релација (нпр. $x \neq y$, $x \leq y$, $x|y$), често се користе и релације са специфичном семантиком које се могу применити на произвољан подскуп променљивих датог проблема, тј. могу имати арност већу од два. Ограничења дефинисана оваквим релацијама зовемо *глобална ограничења*. Иако се ова ограничења често могу представити као конјункције других, једноставнијих ограничења, то обично доводи до губитка информација о структури датог ограничења, што смањује могућност резоновања о том ограничењу. Зато глобална ограничења захтевају посебан третман приликом решавања CSP проблема. Ово подразумева развој специфичних *алгоритама филтрирања* који одсецају неконзистентне вредности узимајући у обзир специфичности семантике датог глобалног ограничења.

У овој тези су за нас посебно значајна два типа глобалних ограничења. Једно од њих је `alldifferent` ограничење које је дефинисано на следећи начин:

$$\texttt{alldifferent}(x_{i_1}, x_{i_2}, \ldots, x_{i_k}) = \{(d_{i_1}, d_{i_2}, \ldots, d_{i_k}) \mid d_{i_j} \in D_{x_{i_j}}, r \neq s \Rightarrow d_{i_r} \neq d_{i_s}\}$$

Испитивање конзистентности `alldifferent` ограничења се обично своди на проблем проналажења максималних упаривања у *бипартитним графовима* [103]. Конзистентност домена над `alldifferent` ограничењем се може успоставити Региновим алгоритмом филтрирања [80] који се детаљно разматра касније у овој тези.

Други тип ограничења која су за нас интересантна су *линеарна ограничења* која су облика:

$$a_1 \cdot x_1 + \ldots + a_k \cdot x_k \bowtie c$$

где је $\bowtie \in \{=, \neq, \leq, <, \geq, >\}$, при чему су a_i и c цели бројеви, а x_i су целобројне промењиве са коначним доменима. Приметимо да без губитка општости можемо претпоставити да су једине релације које се појављују у проблему $\leq$ и $\geq$. Заиста, строга неједнакост $e < c$ ($e > c$) се увек може заменити са $e \leq c - 1$ ($e \geq c + 1$), једнакост $e = c$ се може заменити конјункцијом $e \leq c \wedge e \geq c$, а различитост $e \neq c$ се може заменити дисјункцијом $e \leq c-1 \vee e \geq c+1$. Ове замене се увек могу обавити у фази претпроцесирања. Конзистентност граница над линеарним ограничењем се може успоставити алгоритмом филтрирања који је дат, рецимо, у раду Шултеа и Стакија [87], а који ће такође бити разматран детаљније касније у овој тези.

Пример 2.2.2. Као што је већ речено, ограничења се у CSP проблемима могу изражавати помоћу релација, уместо да се задају скуповима допустивих комбинација вредности. На пример, један CSP проблем би могао бити задат на следећи начин:

$$\begin{aligned}
&x_1 \in \{1, \ldots, 10\},\ x_2 \in \{2, \ldots, 10\} \\
&x_3 \in \{1, \ldots, 10\},\ x_4 \in \{3, \ldots, 10\} \\
&x_5 \in \{3, \ldots, 15\},\ x_6 \in \{9, \ldots, 40\} \\
&\texttt{alldifferent}(x_1, x_2, x_3, x_4) \\
&\texttt{alldifferent}(x_3, x_4, x_5, x_6) \\
&4x_1 + 2x_3 + 4x_5 \leq 85 \\
&x_6 < x_4
\end{aligned}$$

У овом примеру имамо два `alldifferent` ограничења, једно линеарно ограничење и једно ограничење које је задато релацијом $<$ у скупу целих бројева. Овакав начин задавања CSP проблема је знатно чешћи у пракси.

Више детаља о разним типовима глобалних ограничења се може наћи у литератури [83]. У циљу стандардизације дефиниција глобалних ограничења, заједница истраживача одржава *каталог глобалних ограничења*[15]. Каталог садржи велики број глобалних ограничења, а за свако ограничење дата је формална дефиниција, типичне примене, алгоритми филтрирања, референце на релевантне ресурсе и друге потребне информације.

2.3 Сажетак

У овој глави наведени су основни појмови који ће се користити у наставку тезе, а описани су и неки постојећи приступи и приказани релевантни резултати на које се материја изложена у оквиру ове тезе непосредно ослања. Најпре се разматра синтакса и семантика вишесортне логике првог реда која представља основни логички оквир за сва даља разматрања у наставку ове тезе. Затим се дефинише исказна логика и SAT проблем. Појам теорије првог реда и њему придружени SMT проблем разматрају се у наставку. Затим се разматра DPLL($\mathcal{T}$) архитектура на којој је заснована већина модерних SMT решавача. Такође, наведене су и неке стандардне SMT теорије које се често јављају у типичним применама SMT решавача. Најзад, описан је и CSP проблем, а разматрају се и неке основне технике решавања CSP проблема (претрага, пропагација ограничења). На крају су дефинисана два за ову тезу значајна типа глобалних ограничења — `alldifferent` ограничење и линеарно ограничење.

[15] `http://sofdem.github.io/gccat/`

Глава 3

Структура и дизајн SMT решавача `argosmt`

3.1 Увод

У овој глави описујемо структуру и детаље имплементације `argosmt` решавача. У питању је SMT решавач заснован на DPLL($\mathcal{T}$) архитектури [75] (одељак 2.1.6), отвореног кода развијен од стране аутора ове тезе за потребе истраживања која су у тези спроведена. Рад решавача је у потпуности усклађен са системом правила који је описан у тачки 2.1.6.2. Решавач такође подржава комбинацију теорија DTC методом [22]. Од осталих специфичности `argosmt` решавача издвајамо:

- решавач има *флексибилан дизајн* и организован је тако да се лако може мењати и проширивати. Највећи број функционалности решавача су издвојене у посебне компоненте које се лако могу заменити другим компонентама, под условом да имплементирају исти интерфејс. Ово важи и за теоријске решаваче, али и за различите стратегије које решавач користи у току претраге. Неизмењиво језгро решавача садржи само имплементацију система правила, као и основни, прилично општи алгоритам решавања. Ово нам даје велику слободу када је у питању експериментисање са DPLL($\mathcal{T}$) архитектуром, како у оквиру постојећих, тако и у оквиру будућих истраживања.

- решавач има ефикасну имплементацију *библиотеке израза* првог реда, усклађену са стандардом SMT-LIB 2.0 [12]. Ова библиотека је осмишљена

тако да се може користити независно од `argosmt` решавача, у оквиру других алата и решавача.

- не постоји класично пречишћавање, већ се страни подизрази означавају као *дељени*, а затим се свуда по потреби третирају као дељене константе. Овај приступ значајно поједностављује имплементацију, јер нема увођења дељених константи као ни дефиниционих једнакости.

- решавач подржава *вишеслојни приступ* [21], што омогућава да се најпре у резоновању користе једноставније и јефтиније процедуре, а да се касније, по потреби, позивају скупље процедуре које имају већу моћ резоновања.

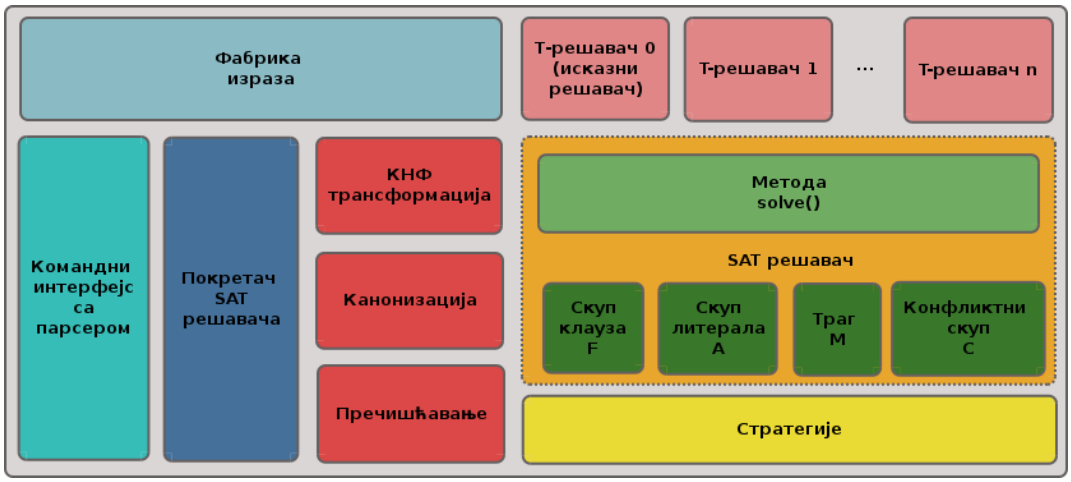
- исказно резоновање (јединичне пропагације и исказни конфликти) је измештено из SAT решавача и *издвојено у посебан теоријски решавач*. Сада је SAT решавач задужен само за претрагу (гранање и анализа конфликата), док се целокупно резоновање обавља у теоријским решавачима. Ово омогућава SAT решавачу да на исти начин третира исказне конфликте (пропагације, објашњења) и теоријске конфликте (пропагације, објашњења), што значајно поједностављује имплементацију. У наставку ове главе подразумевамо да поред теоријских решавача $\mathcal{T}_1, \ldots, \mathcal{T}_n$ постоји и додатни теоријски решавач $\mathcal{T}_0$ који врши исказно резоновање и који се, уколико није посебно наглашено, третира на исти начин као и остали теоријски решавачи.

- решавач омогућава *паралелно покретање* више инстанци SAT решавача које ће независно једна од друге тражити решење, уз могућност размене научених клауза између инстанци решавача (*паралелни портфолио*). Такође, решавач допушта и *паралелизацију на ниском нивоу*, у оквиру појединачних процедура одлучивања.

Структура решавача `argosmt` приказана је на слици 3.1. Појединачне компоненте решавача приказане на слици описујемо детаљније у наставку ове главе.

3.2 Библиотека израза

Један од основних аспеката имплементације SMT решавача је ефикасно представљање израза првог реда, с обзиром на то да се изрази користе у свим деловима решавача. Такође, с обзиром на то да је овај део решавача најближи



Слика 3.1: Структура `argosmt` решавача

улазу који задаје корисник, он и највише зависи од улазног језика, спецификације подржаних теорија и сигнатура, и других захтеваних својстава. У овом поглављу укратко описујемо главне аспекте имплементације библиотеке израза у `argosmt` решавачу [3]. Ова библиотека је усклађена са споменутим SMT-LIB 2.0 стандардом [12]. Нека њена основна својства су: ефикасна имплементација сорти и израза, потпуна провера синтаксне исправности у складу са изабраном сигнатуром (енгл. *well-sortedness*), аутоматско одређивање сорте израза (енгл. *sort inference*), једноставно комбиновање и проширивање сигнатура, додавање подршке за нове теорије, као и апликативни програмски интерфејс (енгл. *application programming interface* (API)) који је усклађен са стандардом. Део библиотеке је и парсер улаза. Библиотека је написана тако да омогућава једноставно проширивање функционалности, како се стандард буде мењао. Сама библиотека је осмишљена као засебна компонента, која се лако може уграђивати и у друге решаваче и SMT алате (иако се за сада користи само у `argosmt` решавачу). У наставку наводимо неке најзначајније аспекте имплементације.

3.2.1 Сигнатуре, изрази и сорте

Основни типови. У основне типове спадају промењиве, функцијски симболи и симболи сорти, као и специјалне константе (попут нумерала и децималних бројева). Сви ови типови се интерно представљају стринговима. Због ефикасности, користи се *дељена имплементација*, тј. сваки стринг се чува само

на једном месту, а свуда где је потребно референсати на њега користе се показивачи. Овим се избегава непотребно копирање стрингова које може бити веома скупо (на пример, једна те иста променљива или константа се може користити на много места у различитим изразима).

Изрази. Најважнији аспект имплементације израза је коришћење технике *дељења заједничких подизраза*. Сви изрази се чувају у оквиру *фабрике израза*, која је одговорна за креирање и уништавање израза у меморији. Сваки израз је представљен јединственим *чвором*. Сви креирани чворови се чувају у хеш табели, а фабрика израза на захтев враћа кориснику *дељени показивач* на жељени чвор. Заједнички подизрази су дељени између различитих чворова — сваки од чворова у себи чува само дељене показиваче на чворове који одговарају подизразима. Овим се драстично смањује количина меморије која је потребна за чување свих израза у библиотеци. При креирању новог израза, најпре се проверава да ли такав израз већ постоји у бази. Ако постоји, враћа се дељени показивач на постојећи чвор. Уколико не постоји, тада се креира нови чвор који ће садржати дељене показиваче на подизразе који су претходно креирани на исти начин. У наставку се изразом барата искључиво путем дељених показивача које фабрика враћа кориснику приликом креирања. Ови показивачи се могу ефикасно копирати, креирати и уништавати. Такође, с обзиром на то да је сваки израз представљен јединственим објектом у меморији, упоређивање два израза на једнакост се своди на упоређивање адреса на које показују два дељена показивача, што је веома ефикасно. Чвор се из фабрике уклања онда када последњи дељени показивач престане да постоји у програму. Изрази су опремљени богатим интерфејсом који омогућава удобан рад (нпр. издвајање подизраза, обилазак стабла израза, издвајање слободних променљивих, замена слободних променљивих произвољним изразима, и сл.).

Сорте. У стандарду SMT-LIB 2.0 [12], сорте имају сличну структуру као и изрази, тј. постоје симболи сорти дате *арности* који се могу примењивати на постојеће сорте како би се добијале сложеније сорте (нпр. листа целих бројева, или листа листи целих бројева). Због тога их у нашој библиотеци израза представљамо на аналоган начин као и изразе. Постоји *фабрика сорти* у оквиру које се чувају јединствени чворови који представљају сорте, а којима се барата путем дељених показивача које фабрика враћа при креирању сорти. Сваки

чвор садржи дељене показиваче на чворове који одговарају подсортама.

Сигнатуре. Сигнатура је представљена структуром података која укључује хеш табеле у којима се чувају функцијски симболи као и симболи сорти. Уз сваки симбол сорте чува се његова арност, док се уз функцијске симболе чувају њихови рангови. Интерфејс сигнатуре омогућава једноставно додавање нових симбола, што је нарочито корисно за слободне симболе који се декларишу за конкретну формулу на улазу.

Исправност сорти. При креирању израза (сорти), фабрика консултује сигнатуру како би проверила да ли функцијски симбол (симбол сорте) постоји и да ли има одговарајући ранг (арност). Том приликом се врши и аутоматско одређивање сорте новокреираних израза, на основу повратних сорти функцијских симбола који се чувају у сигнатури. Ова провера исправности сорти се може искључити, због ефикасности. У том случају, сорта се може задати експлицитно, а може се и касније аутоматски одредити на захтев корисника.

3.2.2 Скриптови и команде

Стандард SMT-LIB 2.0 [12] дефинише појам *скрипта* који представља секвенцу *команди*. Могуће врсте команди су прецизно дефинисане у стандарду. Путем ових команди се одвија комплетна комуникација између корисника решавача и самог решавача. У нашој библиотеци постоји посебна компонента која се зове *командни интерфејс*, а која омогућава како софтверско позивање команди (помоћу одговарајућих процедура апликативног програмског интерфејса), тако и читање команди са улаза на стандардној SMT-LIB 2.0 синтакси, за коју библиотека садржи одговарајући парсер. Неке команде нису везане за сам процес решавања, па се могу у потпуности имплементирати у оквиру библиотеке израза (нпр. декларисање нових симбола, постављање разних опција, и сл.). С друге стране, постоје команде које захтевају покретање процеса решавања (нпр. захтев за проверу задовољивости текућег скупа претпоставки). Због тога библиотека путем посебног интерфејса[1] ове захтеве прослеђује другим де-

[1]Уколико желимо да библиотеку израза користимо у неком другом решавачу или алату, треба само да имплементирамо овај интефејс, тј. да дефинишемо на који начин решавач или алат реагује на корисничке захтеве који му се прослеђују од стране библиотеке.

ловима решавача у којима је имплементиран алгоритам решавања (типично SAT решавачу који управља свим осталим компонентама у процесу решавања).

3.2.3 Подаци израза

У току рада решавача, често је потребно неком изразу придружити одређене податке који се могу брзо и ефикасно очитати када затреба. Библиотека подржава два начина придруживања података изразима. Први начин је да се користе хеш табеле. Сваком изразу је придружен хеш кôд који се израчунава одговарајућом хеш функцијом. Како се ово израчунавање не би изводило при свакој операцији са хеш табелом, вредност хеш кôда сваког израза се израчунава само једном, приликом креирања израза, и чува се у одговарајућем чвору израза. С обзиром на то да је вредност хеш кôда сачувана у самом изразу, она се може ефикасно очитати по потреби. Сложеност приступа придруженом податку је једнака сложености приступа у хеш табели. Овај начин придруживања података изразима је погодан за привремене податке који се креирају и користе локално. Други начин је да се искористи генерички показивач који постоји у самом чвору израза, а који се може иницијализовати да показује на произвољан податак (или колекцију података). Овај начин је нешто ефикаснији, јер је приступ податку директан, а користи се за трајне податке којима се приступа глобално.

3.3 Покретање решавача

Када путем командног интерфејса корисник захтева решавање формуле (тј. испитивање њене задовољивости), овај захтев библиотека израза прослеђује компоненти која се зове *покретач SAT решавача*. Ова компонента преузима унету формулу из библиотеке израза, а затим је припрема за решавање, тако што на њу примењује *КНФ трансформацију* (описану у следећем поглављу). Ово је неопходно, зато што, у складу са SMT-LIB стандардом [12], унета формула не мора бити у КНФ-у. Резултат ове трансформације је скуп клауза који је еквизадовољив са полазном формулом. Затим се креира инстанца SAT решавача као и инстанце теоријских решавача који су потребни, у складу са изабраном теоријом (у случају *паралелног портфолија*, креира се више инстанци SAT решавача који ће се извршавати паралелно). Након креирања и

конфигурисања SAT решавача, прослеђују му се клаузе добијене КНФ трансформацијом. Над литералима ових клауза се примењује *нормализација* (поглавље 3.5) и *пречишћавање* (поглавље 3.6). Након тога се иницијализује стање SAT решавача и позива се метода `solve`() (одељак 3.8.2), чиме отпочиње процес решавања. Након што се заврши са решавањем, резултат се прослеђује командном интерфејсу, како би корисник могао да га очита, а инстанца SAT решавача се уклања из меморије, заједно са припадајућим теоријским решавачима.

3.4 КНФ трансформација

Улазна базна формула се у еквизадовољиви КНФ преводи Цајтиновом трансформацијом [102]. Ова трансформација се заснива на увођењу нових константи сорте `Bool` (тј. нових исказних слова) којима се у стаблу оригиналне формуле идући од листова ка корену замењују делови формуле повезани логичким везницима, уз увођење еквиваленције којом се дефинише новоуведена константа (која се затим трансформише у КНФ и додаје у резултујућу формулу). На пример, конјункција[2] $\alpha_1 \wedge \ldots \wedge \alpha_n$ (где су α_i атоми првог реда) се у оригиналној формули замењује неким новим исказним словом s, а у резултујући КНФ се додају клаузе $(\neg\alpha_1 \vee \neg\alpha_2 \vee \ldots \vee \neg\alpha_n \vee s) \wedge (\neg s \vee \alpha_1) \wedge (\neg s \vee \alpha_2) \wedge \ldots \wedge (\neg s \vee \alpha_n)$. Сличан поступак је и за остале логичке везнике. Поступак се наставља док се полазна формула не сведе на један литерал, који се у том случају додаје у резултујући КНФ као јединична клауза. Резултујућа КНФ формула је еквизадовољива са полазном.

Поред Цајтинове трансформације, у склопу КНФ трансформације обављају се и неке друге трансформације формуле, како би се формула свела на жељени облик. На пример, пре саме Цајтинове трансформације, врши се елиминација логичких константи. Такође, бинаризују се n-арне једнакости и различитости.

3.5 Нормализација

Нормализација је процес којим се литерали добијене КНФ формуле своде на неки пожељни *нормални облик*. С обзиром на то да SAT решавач не разуме структуру литерала првог реда, посао нормализације се мора обављати уз аси-

[2]SMT-LIB улазни језик подразумева да су конјункције, дисјункције, импликације, еквиваленције, једнакости и различитости n-арни везници.

стенцију теоријских решавача.[3] У ту сврху сваки теоријски решавач $\mathcal{T}_i$ имплементира процедуру $\mathcal{T}_i \rightarrow$ `normalize`()[4] која враћа нормализовану форму израза e који јој се предаје као аргумент. Притом претпостављамо да је теорија $\mathcal{T}_i$ *власник* израза e, тј. да водећи функцијски симбол израза e припада сигнатури теорије $\mathcal{T}_i$. Такође, подразумевамо да су подизрази израза e већ нормализовани. Другим речима, процедура $\mathcal{T}_i \rightarrow$ `normalize`() не улази рекурзивно у подизразе, већ се нормализује само структура израза на горњем нивоу. Рекурзивни пролаз кроз стабло израза обавља се процедуром `normalizeExpression`() (алгоритам 3.1). Ова процедура примењује нормализацију свих подизраза у стаблу израза од листова ка корену. За сваки подизраз се у зависности од водећег функцијског симбола бира теоријски решавач који ће обавити нормализацију — то је управо теоријски решавач за теорију која је власник датог подизраза. Овај теоријски решавач познаје значење водећег функцијског симбола и самим тим зна које се трансформације могу применити на том нивоу. Теорија власник се одређује помоћном процедуром `findOwner`() (алгоритам 3.2). Ова процедура за сваки од теоријских решавача позива процедуру $\mathcal{T}_i \rightarrow$ `isOwned`() која би требало да врати `true` ако и само ако је теорија $\mathcal{T}_i$ власник израза који јој се предаје као аргумент. С обзиром на то да је власник сваког израза јединствен (јер су сигнатуре дисјунктне), само један од позива ће вратити `true`, па процедура `findOwner`() управо враћа индекс те теорије.

3.6 Пречишћавање

У решавачу `argosmt` не врши се класично пречишћавање, какво је описано у тачки 2.1.6.1. Наиме, уместо да се уводе нове дељене константе којима се замењују страни подизрази, приликом проласка кроз стабло израза идентификују се *дељени подизрази*. Израз e ће бити дељен ако је страни подизраз у неком другом изразу, или ако је лева или десна страна неке мешовите једнакости. Другим речима, дељени изрази управо одговарају изразима који се у класичном пречишћавању замењују новоуведеним дељеним константама. У нашем приступу, те константе се не уводе, већ се просто сваки дељени израз e сматра

[3]Приметимо да су изрази које нормализујемо у општем случају мешовити, тј. садрже симболе из сигнатура разних теорија.

[4]Ознака $\mathcal{T}_i \rightarrow$ `normalize`() означава да је процедура `normalize` део интерфејса теоријског решавача $\mathcal{T}_i$. Сличан запис ћемо користити и за друге интерфејс процедуре теоријског решавача, како бисмо их разликовали од осталих процедура које нису део интерфејса теоријског решавача.

```
procedure normalizeExpression(e)
begin
  {Претпоставка је да је e базни израз који не садржи логичке симболе}
  if e ≡ a, где је a константа then
    ē := a
  else if e ≡ f(e_1, ..., e_n) then
    for all k := 1 to n do
      ē_k := normalizeExpression(e_k) {рекурзивни позив за подизраз e_k}
    end for
    i := findOwner(e) {одређујемо индекс теорије која је власник израза e}
    ē := T_i → normalize(f(ē_1, ..., ē_n))
  end if
  return ē
end
```

Алгоритам 3.1: `normalizeExpression`(e)

```
procedure findOwner(e)
begin
  {Претпоставка је да је e базни израз који не садржи логичке симболе}
  for all i := 1 to n do
    if T_i → isOwned(e) then
      return i
    end if
  end for
end
```

Алгоритам 3.2: `findOwner`(e)

константом у теоријским решавачима свих теорија, изузев оне теорије која је власник израза e. Све једнакости између дељених израза биће третиране као *дељене једнакости*.

Процедура `checkSharedInFormula`() (алгоритам 3.3) одређује скуп свих дељених израза S у формули Ψ. Она за сваки атом α формуле Ψ позива процедуру `checkSharedInExp`() (алгоритам 3.4). Ова рекурзивна процедура обилази стабло атома α и за све подизразе одређује власнике (поље *owner*). Затим, уколико неки непосредни подизраз e' неког израза e има различитог власника од e, тада се e' додаје у скуп дељених израза S и обележава се као *shared*. Слично, за мешовите једнакости, обе стране једнакости се додају у скуп S. Поља *owner* и *shared* се чувају у самим чворовима којима су изрази представљени у фабрици израза, тако да им се може веома брзо приступити.

Пример 3.6.1. Нека су $\mathcal{T}_1$ и $\mathcal{T}_2$, редом, теорија реалне аритметике и EUF теорија. Претпоставимо да је у комбинованој теорији $\mathcal{T}_1 + \mathcal{T}_2$ потребно испитати задовољивост следеће конјункције литерала: $x - t = f(x + y) \wedge y + t = f(z + t) \wedge x - t \neq y + t \wedge x + 2y + z + 2t \leq 0 \wedge 2x + y \geq z + 2t \wedge f(x) - 3x = 5y + 2t \wedge x + y - z = t$, при чему

```
procedure checkSharedInFormula(Ψ, var S)
begin
  S := ∅
  {Поље owner је иницијално undef за све изразе}
  {Поље shared је иницијално false за све изразе}
  for all α ∈ atoms(Ψ) do
    checkSharedInExp(α, S)
  end for
end
```

Алгоритам 3.3: `checkSharedInFormula`$(\Psi, \texttt{var}\ S)$

```
procedure checkSharedInExp(e, var S)
begin
  {Претпоставка је да је e базни израз који не садржи логичке симболе}
  if e.owner ≠ undef then
    return {израз је већ обрађен}
  end if
  if e ≡ a, где је a константа then
    e.owner := findOwner(e)
    return {излаз из рекурзије}
  else if e ≡ (u = v) then
    checkSharedInExp(u, S)
    checkSharedInExp(v, S)
    if u.owner ≠ v.owner then {ако је (u = v) мешовита једнакост}
      S := S ∪ {u, v}
      u.shared := true
      v.shared := true
      e.owner := −1 {дељена једнакост нема власника}
    else
      e.owner := u.owner {није дељена једнакост}
    end if
  else if e ≡ f(e_1, ..., e_n) then
    e.owner := findOwner(e)
    for all e_k do
      checkSharedInExp(e_k, S)
      if e_k.owner ≠ e.owner then {ако је e_k страни подизраз у e}
        S := S ∪ {e_k}
        e_k.shared := true
      end if
    end for
  end if
end
```

Алгоритам 3.4: `checkSharedInExp`$(e, \texttt{var}\ S)$

је симбол f из сигнатуре EUF теорије, док су сви остали симболи из сигнатуре реалне аритметике. Процедура `checkSharedInFormula`() проналази све стране подизразе и проглашава их за дељене. На пример, израз $x+y$ је страни подизраз у изразу $f(x+y)$, па га зато проглашавамо за дељени израз. Такође, за сваку

мешовиту једнакост, њена лева и десна страна биће дељени изрази. На пример, израз $f(x+y)$ биће дељен, зато што је једнакост $x-t = f(x+y)$ мешовита. Добијени скуп дељених израза је $S = \{x+y, z+t, f(x+y), f(z+t), x-t, y+t, x, f(x)\}$.

Приметимо да је ефикасна имплементација описаног приступа могућа захваљујући начину на који је имплементирана библиотека израза. Сетимо се да је сваки израз представљен јединственим чвором у фабрици израза, а да се свугде где је то потребно на њега реферише путем дељених показивача. Дељени показивачи нам дају могућност да ефикасно баратамо изразом (што подразумева копирање, преношење израза као аргумената и повратне вредности из функција, упоређивање на једнакост, чување у различитим структурама података, приступање подацима који су придружени изразу), а да притом не улазимо у унутрашњу структуру самог израза. Из овог разлога, за неки дељени израз увек можемо по потреби сматрати да је константа, тј. потпуно игнорисати његову структуру, а да притом не изгубимо потребну функционалност. Са друге стране, чињеница да не уводимо нове дељене константе, као и да нема дефиниционих једнакости значајно поједностављује имплементацију.

3.7 Интерфејс теоријског решавача

Да би теоријски решавач могао да се угради у `argosmt` решавач, треба да имплементира интерфејс дат на слици 3.2. Процедура $\mathcal{T}_i \rightarrow$ `addAtom`() омогућава увођење атома из скупа A у контекст теоријског решавача, како би се иницијализовале његове интерне структуре података. Ово је неопходно, како на почетку рада решавача за атоме из иницијалне формуле над сигнатуром Σ_i, тако и касније приликом додавања нових атома и дељених једнакости правилима $\texttt{IntroduceAtom}_i$ и `IntroduceSharedEq`. Да би подршка за нивое одлучивања и враћање уназад (правила `Decide` и `Backjump`) била обезбеђена, сваки теоријски решавач мора да имплементира процедуре $\mathcal{T}_i \rightarrow$ `newLevel`() и $\mathcal{T}_i \rightarrow$ `backjump`() које омогућавају да се запамти стање теоријског решавача на крају неког нивоа одлучивања и касније реконструише по потреби приликом враћања уназад на тај ниво. Интерфејс процедура $\mathcal{T}_i \rightarrow$ `assertLiteral`() се позива од стране SAT решавача кад год се нови литерал појави на трагу M, како би се теоријском решавачу проследио тај литерал. Сваком теоријском решавачу се прослеђују само они литерали за које је заинтересован, тј. литерали који су у власништву одговарајуће теорије, као и дељене једнакости и различитости. Најсложенија

Процедура	Опис
$\mathcal{T}_i \rightarrow$ `addAtom`(α)	позива се од стране SAT решавача кад год се неки атом α дода у скуп атома A
$\mathcal{T}_i \rightarrow$ `newLevel`()	позива се од стране SAT решавача кад год се успостави нови ниво одлучивања у M
$\mathcal{T}_i \rightarrow$ `backjump`(m)	позива се од стране SAT решавача кад год се врати на ниво m
$\mathcal{T}_i \rightarrow$ `assertLiteral`(l)	позива се од стране SAT решавача кад год се литерал l дода у M
$\mathcal{T}_i \rightarrow$ `checkAndPropagate`($layer$)	проверава да ли је текући траг M задовољив у $\mathcal{T}_i$ и пропагира литерале (или додаје нове клаузе) који су $\mathcal{T}_i$-последице од M (по потреби примењује правила $\texttt{TheoryConflict}_i$, `IntroduceSharedEq`, $\texttt{TheoryPropagate}_i$, $\texttt{IntroduceAtom}_i$, $\texttt{TheoryLemma}_i$)
$\mathcal{T}_i \rightarrow$ `explainLiteral`(l)	генерише објашњење пропагираног литерала l и примењује правило $\texttt{TheoryExplain}_i$
$\mathcal{T}_i \rightarrow$ `getModel`($exps$)	враћа интерпретације израза из скупа $exps$ у неком моделу теорије $\mathcal{T}_i$ који задовољава све литерале на трагу M
$\mathcal{T}_i \rightarrow$ `normalize`(e)	враћа израз настао нормализацијом израза e
$\mathcal{T}_i \rightarrow$ `isOwned`(e)	враћа `true` ако и само ако је водећи симбол израза e у сигнатури теорије $\mathcal{T}_i$

Слика 3.2: Интерфејс $\mathcal{T}_i$-решавача

процедура коју треба имплементирати је $\mathcal{T}_i \rightarrow$ `checkAndPropagate`() процедура. Она би требало да може да детектује $\mathcal{T}_i$-незадовољивост трага M. Ако је конфликт у теорији уочен, процедура генерише објашњење конфликта и примењује правило[5] $\texttt{TheoryConflict}_i$, чиме се започиње анализа конфликта. Ако нема конфликта у $\mathcal{T}_i$, процедура испитује постојање литерала који су последица литерала из M у теорији $\mathcal{T}_i$ и пропагира их примењујући правило $\texttt{TheoryPropagate}_i$. У случају пропагације дељених једнакости и различитости, неопходно је претходно (применом правила `IntroduceSharedEq`) увести одговарајућу једнакост у контекст SAT решавача (уколико већ није у скупу A). Такође, ова процедура може применити и правило $\texttt{TheoryLemma}_i$ како би захтевала додатно гранање од стране SAT решавача. Ако научена клауза садржи нове литерале који нису у A, тада процедура може применити правило $\texttt{IntroduceAtom}_i$. Попут многих

[5]Уколико нека од процедура теоријског решавача треба да примени неко правило, она то може учинити искључиво позивом одговарајуће интерфејс процедуре SAT решавача, описане у одељку 3.8.1.

савремених SMT решавача, наш решавач подржава тзв. *вишеслојни приступ* (енгл. *layered approach*) [21]. Ово значи да се приликом провере задовољивости и детекције теоријских пропагација најпре позивају јефтинији и једноставнији алгоритми (*нижи дедуктивни слојеви*), док се дедуктивно јачи, али много сложенији алгоритми позивају касније, или се покрећу само периодично (*виши дедуктивни слојеви*). Због тога процедура $\mathcal{T}_i \rightarrow$ `checkAndPropagate`() прихвата параметар *layer* који представља број дедуктивног слоја и који одређује који ће се алгоритми позивати. Процедура $\mathcal{T}_i \rightarrow$ `explainLiteral`() се позива од стране SAT решавача у току анализе конфликта за објашњавање оних литерала који су раније пропагирани од стране $\mathcal{T}_i$-решавача. Процедура генерише објашњење пропагације литерала и примењује правило $\texttt{TheoryExplain}_i$. Процедура $\mathcal{T}_i \rightarrow$ `getModel`() се може позивати само када је скуп литерала из M задовољив у теорији $\mathcal{T}_i$, а користи се за очитавање вредности које су придружене произвољним изразима у неком моделу теорије $\mathcal{T}_i$ који задовољава M.[6] Процедура $\mathcal{T}_i \rightarrow$ `normalize`() се користи за нормализацију израза који припадају теорији $\mathcal{T}_i$, као што је објашњено у поглављу 3.5, док се процедуром $\mathcal{T}_i \rightarrow$ `isOwned`() испитује да ли је дати израз у власништву теорије $\mathcal{T}_i$, што је корисно приликом нормализације и пречишћавања (поглавље 3.6).

3.8 SAT решавач

SAT решавач `argosmt` решавача имплементира стање решавача (F, A, M, C), као и систем правила за промену стања који је у потпуности усаглашен са системом датим на слици 2.1 у одељку 2.1.6. Поред тога, SAT решавач имплементира централни алгоритам (процедура `solve`()) који је задужен за покретање теоријских решавача, гранање, анализу конфликата, рестартовање и заборављање. Са друге стране, као што је раније речено, исказно резоновање (јединичне пропагације и откривање конфликтних клауза) није део SAT решавача, већ је издвојено у посебан теоријски решавач $\mathcal{T}_0$ који имплементира исти интерфејс као и сви остали теоријски решавачи (слика 3.2). Решавач $\mathcal{T}_0$ се у методи `solve`() третира потпуно исто као и сви остали теоријски решавачи.

[6] У зависности од природе процедуре одлучивања која је имплементирана, поједини теоријски решавачи могу да не подржавају ову функционалност.

3.8.1 Интерфејс SAT решавача

Структуре података које одговарају компонентама стања решавача су интерне за SAT решавач и није им могуће директно приступати из других делова решавача. Самим тим, било каква промена стања је могућа искључиво коришћењем интерфејса SAT решавача који је дат на слици 3.3. Стање решавача се иницијализује процедуром `initSolver`() којој се предаје иницијални скуп клауза F_0. Ова процедура покреће нормализацију и пречишћавање (тј. идентификацију дељених израза), а затим се сви атоми који се појављују у клаузама додају у скуп A, чиме се успоставља иницијално стање решавача. Стање се у наставку може мењати искључиво применом правила. За свако од правила постоји одговарајућа процедура у оквиру интерфејса SAT решавача која примењује то правило. Свака од процедура након што промени стање решавача, о томе обавештава друге компоненте решавача, пре свега теоријске решаваче, путем одговарајућих интерфејс процедура (нпр. процедура `applyBackjump`(), након враћања решавача на одговарајући ниво, позива $\mathcal{T}_i \rightarrow$ `backjump`() за све теоријске решаваче $\mathcal{T}_i$). С обзиром на то да се исказно резоновање обавља у посебном теоријском решавачу који имплементира исти интерфејс као и остали теоријски решавачи, са становишта SAT решавача нема никакве разлике између нпр. пропагација које врше теорије (применом правила $\texttt{TheoryPropagate}_i$) и јединичних пропагација које врши тај издвојени теоријски решавач (применом правила `UnitPropagate`). Због тога постоји само једна процедура `applyPropagate`() која врши пропагацију, без обзира да ли је резоновање потекло из теорије или из клауза формуле. Индекс теорије i који се предаје функцији као аргумент ће у случају јединичне пропагације имати вредност 0. Слично је и са процедурама `applyConflict`() и `applyExplain`(). Процедуре за примену правила (`apply`$*$ процедуре) се могу позвати само ако су испуњени услови за примену одговарајућег правила. Најзад, уколико процедура `solve`() пријави *sat* као резултат, процедуром `getModel`() се могу очитати интерпретације израза у неком моделу теорије који задовољава улазну формулу. Процедура `getModel`() по потреби позива истоимене методе теоријских решавача за добијање информација о моделима конкретних теорија.

Процедура	Опис
`initSolver`(F_0)	Иницијализује стање решавача са скупом клауза F_0
`solve`()	Покреће процес решавања
`applyIntroduceAtom`(α, i)	Примењује правило $\texttt{IntroduceAtom}_i$ за атом α
`applyIntroduceSharedEq`(u, v)	Примењује правило `IntroduceSharedEq` за једнакост $u = v$
`applyDecide`(l)	Примењује правило `Decide` за литерал l
`applyPropagate`(l, i)	Примењује правило `UnitPropagate` или $\texttt{TheoryPropagate}_i$ за литерал l (i је индекс теорије, или 0 за `UnitPropagate`)
`applyConflict`($cflt, i$)	Примењује правило `Conflict` или $\texttt{TheoryConflict}_i$ са конфликтним скупом $cflt$ (i је индекс теорије, или 0 за `Conflict`)
`applyExplain`($l, expl, i$)	Примењује правило `Explain` или $\texttt{TheoryExplain}_i$ за литерал l са објашњењем $expl$ (i је индекс теорије, или 0 за `Explain`)
`applyBackjump`()	Примењује правило `Backjump`
`applyTheoryLemma`(c, i)	Примењује правило $\texttt{TheoryLemma}_i$ за клаузу c
`applyRestart`()	Примењује правило `Restart`
`applyForget`(cl_set)	Примењује правило `Forget` за сваку од клауза из скупа cl_set
`getModel`($exps$)	Враћа интерпретације израза из скупа $exps$ у неком моделу теорије који задовољава улазну формулу

Слика 3.3: Интерфејс SAT решавача

3.8.2 Процедура `solve`()

Процедура `solve`() (алгоритам 3.5) представља главни алгоритам целог решавача. У питању је итеративни алгоритам у коме се најпре позивају процедуре $\mathcal{T}_i \rightarrow$ `checkAndPropagate`() за све теоријске решаваче, најпре у најнижем слоју (за $layer = 1$), а затим и за више слојеве (када се нижи слојеви дедуктивно засите, тј. више нема промене стања). Овај поступак се прекида или када се засите сви дедуктивни слојеви, или када нека од теорија пријави конфликт (позивом процедуре `applyConflict`()). У првом случају не постоји конфликт, па се најпре проверава да ли је потребно рестартовати (у складу са неком одабраном политиком рестартовања) и по потреби се позива процедура `applyRestart`(). Затим се, на сличан начин, проверава да ли је потребно применити заборављање, и по потреби се бира скуп клауза које треба заборавити,

а затим се позива процедура `applyForget`(). На крају се, уколико постоји бар један литерал недефинисан у M, позива процедура `applyDecide`(), након што се претходно неком хеуристичком методом изабере погодни литерал одлучивања. Након примене правила `Decide` прелази се на следећу итерацију главне петље, чиме почиње нови дедукцијски циклус. Уколико не постоји недефинисан литерал, тада је формула задовољива, па излазимо из петље.

У случају да је нека од теорија пријавила конфликт, започињемо анализу конфликта тако што у петљи објашњавамо литерале конфликтног скупа C један по један, увек узимајући онај литерал из C који је последњи на трагу M. За генерисање објашњења упошљавамо одговарајући теоријски решавач који је пропагирао тај литерал, тако што позивамо његову интерфејс процедуру `explainLiteral`(). Ова процедура генерише објашњење и позива `applyExplain`() процедуру SAT решавача. Поступак се наставља докле год не достигнемо UIP тачку, или не испразнимо скуп C. У првом случају примењујемо правило `Backjump`, док је у другом случају формула незадовољива, па излазимо из петље.

3.8.3 Стратегије и хеуристике

Процедура `solve`() у свом раду користи извесне стратегије и хеуристике од којих веома зависи ефикасност претраге. Решавач је осмишљен тако да се стратегије могу лако мењати. јер су имплементиране као засебне компоненте решавача. Тренутна верзија решавача подразумевано користи стратегије инспирисане решавачем `argosat` [64]. У наставку у кратким цртама описујемо ове стратегије.

Стратегија гранања. Ова стратегија је веома значајна, јер одређује редослед обиласка простора претраге, а односи се на избор литерала у правилу `Decide`. Овај литерал се бира у две фазе, тако што се најпре одабере један од атома $\alpha \in A$ недефинисаних у M, а затим се одабере један од два могућа поларитета тог атома (α или $\neg\alpha$). За избор атома, користи се варијанта VSIDS стратегије [68], која приоритет даје оним атомима који су у блиској прошлости учествовали у конфликтима. Ова стратегија се користи у 95% случајева, док се у 5% случајева атом бира на случајан начин.[7] За избор поларитета, користи

[7]За потребе свих случајних избора у решавачу користи се јединствени генератор случајних бројева чија се почетна вредност (тзв. *семе*) задаје као опција при покретању решавача.

се стратегија *сачуваног поларитета*, где се увек бира онај поларитет у ком је одговарајући литерал био последњи пут на трагу.

Стратегија рестартовања. Решавач `argosmt` користи *геометријску стратегију рестартовања*, која подразумева да се рестартовање примењује након одговарајућег броја конфликата, при чему се тај број увећава геометријском прогресијом након сваког рестарта. Подразумевано подешавање је да се први рестарт примењује након 100 конфликата, а онда се сваки следећи рестарт примењује након 1.5 пута више конфликата (тј. следећи након 150 конфликата, па затим након 225 конфликата, итд.).

Стратегија заборављања. Стратегија заборављања мора да одговори на два питања. Прво је да ли у неком тренутку треба применити заборављање. Друго питање је које клаузе треба заборавити. У нашем решавачу, за избор тренутка у коме се примењује заборављање користи се *геометријска стратегија*, код које се заборављање примењује сваки пут када број научених клауза постане већи од неког задатог броја, при чему се тај број након сваког рестарта увећава геометријском прогресијом. На овај начин се омогућава постепено увећавање дозвољеног броја научених клауза. Подразумевано се прво заборављање врши након што број научених клауза премаши трећину иницијалног броја клауза, а затим се потребни број научених клауза након сваког рестарта увећава за 10%. При избору клауза које треба заборавити, увек се бирају само научене клаузе, јер за њих увек важи да су последица осталих клауза у бази. Притом се заборавља извесни проценат (подразумевано 50%) научених клауза, а бирају се оне клаузе које су биле *мање активне* у скорашњим конфликтима.

3.9 Подржани теоријски решавачи

Исказни решавач. Исказни решавач игра посебну улогу у `argosmt` решавачу, јер врши исказно резоновање. Иако овај решавач не разматра теорију првог реда у правом смислу, његов интерфејс је потпуно исти као и код свих осталих теоријских решавача. Интерно, овај решавач имплементира алгоритам заснован на *шеми два посматрана литерала* (енгл. *two watched literals scheme*) [68]. Овај алгоритам омогућава ефикасну претрагу базе клауза у потрази за клаузама из којих се може извести јединична пропагација, као и за конфликт-

ним клаузама. У свакој од клауза се најпре изаберу два *посматрана* литерала — докле год ови литерали нису нетачни у M, нема ни конфликта ни пропагације из те клаузе. За сваки од литерала из $lits(A)$ одржава се листа клауза у којима је тај литерал један од посматраних (дакле, свака клауза се у сваком тренутку налази на две овакве листе). Након сваког додавања новог литерала на траг M, обилази се листа која одговара литералу који је управо постао нетачан. За сваку од клауза се тада покушава наћи алтернативни посматрани литерал, при чему се онда клауза пребацује на листу тог литерала. Ако то није могуће, то значи да клауза има или тачно један литерал који није нетачан (у ком случају имамо јединичну пропагацију), или су сви литерали клаузе нетачни (у ком случају пријављујемо конфликт). Овај теоријски решавач има само један дедуктивни слој. Објашњења се генеришу једноставно, на основу клауза које су изазвале пропагације или конфликте.

EUF решавач. Овај решавач разматра базни фрагмент EUF теорије. Процедура одлучивања је заснована на *конгруентним затворењима*, а инспирисана је процедуром датом у раду Ниевенхуиса и Оливерас [73]. Решавач укључује ефикасне структуре података за представљање класа еквиваленције које се могу динамички мењати (енгл. *union-find*). Класе еквиваленције се спајају приликом додавања једнакости на траг M, док је резоновање засновано на конгруентности релације једнакости (нпр. из $a = b \wedge c = d$ следи да је $f(a, c) = f(b, d)$). Објашњења се генеришу на исти начин као у раду Ниевенхуиса и Оливерас [73]. Решавач има само један дедуктивни слој.

Решавач за линеарну аритметику. Овај решавач је заснован на симплекс процедури, прилагођеној SMT контексту [37]. Детаљи овог решавача дати су у глави 5. Решавач омогућава паралелно извршавање симплекс алгоритма, што је оригинални допринос ове тезе. Има два дедуктивна слоја, при чему се у нижем слоју препознају само тривијалне пропагације и конфликти, док се скупа симплекс процедура позива у вишем дедуктивном слоју.

Решавач за CSP теорију. Овај решавач разматра теорију глобалних ограничења над коначним доменима. Представља оригинални допринос ове тезе и детаљно се разматра у глави 4. Има два дедуктивна слоја.

3.10 Комбинација теорија

Комбинација теорија у `argosmt` решавачу је заснована на DTC методи [22]. Дељеним једнакостима се сматрају све једнакости између израза који су означени као дељени (тј. налазе се у скупу дељених израза S). Ове једнакости се прослеђују свим теоријским решавачима кад год се појаве на трагу. Поред постојећих дељених једнакости које су део иницијалне формуле, додатне дељене једнакости се у скуп атома A могу уводити *лењо*, по потреби, применом правила `IntroduceSharedEq`. Лењо увођење дељених једнакости смањује просторну сложеност алгоритма и истовремено одлаже непотребна гранања по дељеним једнакостима докле год то није заиста неопходно. Притом је од изузетног значаја када и за које дељене једнакости се правило `IntroduceSharedEq` примењује, како бисмо са једне стране гарантовали коректност поступка комбиновања теорија, а са друге стране обезбедили што ефикаснији рад решавача.

Уведимо најпре неке основне дефиниције. За разврставање (тј. релацију еквиваленције) $\sim$ скупа дељених израза S кажемо да је *сагласно* са трагом M ако за свака два дељена израза $u, v \in S$ за која је $u = v \in M$ важи да је $u \sim v$, а за свака два дељена израза $u, v \in S$ за која је $u \neq v \in M$ важи да је $u \nsim v$. Другим речима, у питању су разврставања која су конзистентна са дељеним једнакостима и различитостима из M. За дељену једнакост $u = v$ кажемо да је *одређена* у M ако важи или да је $u \sim v$ у сваком разврставању $\sim$ сагласном са M, или да је $u \nsim v$ у сваком разврставању $\sim$ сагласном са M. У супротном је *неодређена* у M. На пример, ако на трагу имамо $u = w, w = v, v \neq t, t \neq s$, тада у сваком сагласном разврставању мора да важи $u \sim v$, па је једнакост $u = v$ одређена у M. Слично, у сваком сагласном разврставању је $u \nsim t$, па је и једнакост $u = t$ одређена у M. Са друге стране, једнакост $u = s$ је неодређена у M. Кажемо да траг M *одређује* разврставање скупа S ако постоји јединствено разврставање сагласно са M. Ово јединствено разврставање ћемо обележавати са $\sim_M$. Једноставно се може показати да траг M одређује разврставање скупа S ако и само ако су све дељене једнакости одређене у M. Одређеност разврставања трагом M је довољан услов да поступак комбиновања теорија буде коректан, о чему говори следећа теорема.

Теорема 3.10.1. Претпоставимо да је у текућем стању решавача $C = no\_cflt$. Претпоставимо, даље, да не постоје дељени изрази $u, v \in S$ такви да је једнакост $u = v$ неодређена у M. Тада постоји модел $\mathcal{M}$ теорије $\mathcal{T} = \mathcal{T}_1 + \ldots + \mathcal{T}_n$ такав

да задовољава све литерале са трага M.

Доказ. Како не постоје дељене једнакости неодређене у M, следи да постоји јединствено разврставање $\sim_M$ сагласно са M, а како је $C = no\_cflt$, следи да за сваку теорију T_i постоји модел $\mathcal{M}_i$ који задовољава литерале из M, па самим тим и разврставање $\sim_M$ које је одређено дељеним једнакостима и различитостима из M. Ово значи да за свако $u, v \in S$ важи да је $I^{\mathcal{M}_i}(u) = I^{\mathcal{M}_i}(v)$ ако и само ако је $u \sim_M v$. Сада на основу разматрања датог у раду Тинелија и Харандија [101] следи да постоји модел $\mathcal{M}$ комбиноване теорије $\mathcal{T}$ који задовољава све литерале из M. □

Из претходне теореме следи да за коректност поступка комбиновања теорија није неопходно да све дељене једнакости буду *дефинисане* у M. Поједине дељене једнакости не морају чак бити ни у скупу A. Довољно је да све дељене једнакости буду *одређене* у M. Ово значи да је у случају лењог увођења дељених једнакости у скуп A довољно уводити само оне једнакости које су у датом тренутку неодређене у M. Једнакости одређене у M се не морају уводити, јер је њихова интерпретација у моделима комбиноване теорије који задовољавају литерале из M једнозначно одређена. На пример, ако на трагу имамо $u = w$ и $w = v$, тада у сваком моделу $\mathcal{M}$ теорије $\mathcal{T}$ који задовољава све литерале из M мора да важи да је $I^{\mathcal{M}}(u = v) = \texttt{true}$. Зато нема потребе уводити и пропагирати једнакост $u = v$, јер би литерал $u = v$ на трагу био редундантан.[8] На овај начин се спречава увођење великог броја непотребних дељених једнакости у скуп A.

Наравно, остаје питање конкретне стратегије којом се неодређене дељене једнакости уводе у скуп атома A. У наставку описујемо случајеве у којима се правило `IntroduceSharedEq` примењује у нашем решавачу.

Пропагација дељених једнакости. Уколико теоријски решавач $\mathcal{T}_i$ за неку дељену једнакост $u = v \notin A$ *неодређену* у M закључи да важи $M \models_{\mathcal{T}_i} u = v$, или $M \models_{\mathcal{T}_i} u \neq v$, тада се једнакост $u = v$ уводи у скуп A применом правила `IntroduceSharedEq`, након чега се за одговарајући литерал примењује правило $\texttt{TheoryPropagate}_i$. Слично, ако теоријски решавач закључи да важи $l_1, \ldots, l_r \models_{\mathcal{T}_i} u_1 = v_1 \vee \ldots \vee u_s = v_s$, где су $u_k = v_k$ дељене једнакости неодређене

[8] Наравно, ако је једнакост $u = v$ већ од раније у скупу A, она ће свакако бити пропагирана, јер сви атоми који су већ у A морају пре или касније постати дефинисани у M. Претходна дискусија се односи само на увођење дељених једнакости које у датом тренутку нису у A.

у M, а $l_1, \ldots, l_r$ литерали из M, тада се све ове дељене једнакости уводе у скуп A (ако већ нису у њему), а затим се примењује правило $\texttt{TheoryLemma}_i$ за клаузу $\neg l_1 \vee \ldots \vee \neg l_r \vee u_1 = v_1 \vee \ldots \vee u_s = v_s$. На овај начин је омогућено резоновање о дељеним једнакостима, како за конвексне, тако и за неконвексне теорије. Може се показати да је овакав вид увођења дељених једнакости довољан за коректност поступка комбиновања теорија, у случају да су сви теоријски решавачи *дедуктивно комплетни за дељене једнакости* [69]. За теоријски решавач кажемо да је *дедуктивно комплетан за дељене једнакости* ако је у стању да препозна сваку логичку последицу облика $M \vDash_{\mathcal{T}_i} u = v$ или $M \vDash_{\mathcal{T}_i} \bigvee_{j=1}^{s} u_s = v_s$, где су $u = v, u_1 = v_1, \ldots, u_s = v_s$ дељене једнакости. Другим речима, у случају да су сви теоријски решавачи дедуктивно комплетни, коректност поступка комбиновања теорија важи и под слабијим условима од оних наведених у теореми 3.10.1 (конкретно, траг не мора једнозначно одређивати разврставање скупа дељених израза S).

Пример 3.10.1. Претпоставимо да, као у примеру 3.6.1, испитујемо задовољивост конјункције $x-t = f(x+y) \wedge y+t = f(z+t) \wedge x-t \neq y+t \wedge x+2y+z+2t \leq 0 \wedge 2x + y \geq z + 2t \wedge f(x) - 3x = 5y + 2t \wedge x + y - z = t$, над комбинованом теоријом $\mathcal{T}_1 + \mathcal{T}_2$, где је $\mathcal{T}_1$ теорија реалне аритметике, а $\mathcal{T}_2$ је EUF теорија (симбол f је из сигнатуре EUF теорије, док су сви остали симболи из сигнатуре реалне аритметике). Изрази $x + y$ и $z + t$ су дељени, али једнакост $x + y = z + t$ иницијално није у скупу A, јер се не појављује у датој конјункцији. Након што се јединичним пропагацијама сви литерали у конјункцији поставе на траг M, оба теоријска решавача утврђују да је M задовољив у појединачним теоријама, али то још увек не значи да је траг задовољив и у комбинованој теорији. Ако претпоставимо да је теоријски решавач за реалну аритметику дедуктивно комплетан, тада он може да из $x+y-z = t$ закључи да мора бити $x+y = z+t$, због чега он примењује правило `IntroduceSharedEq`, а затим и $\texttt{TheoryPropagate}_i$ за дељену једнакост $x + y = z + t$. Ова једнакост доводи до незадовољивости у EUF решавачу, јер из $x + y = z + t$ следи да је $f(x + y) = f(z + t)$ одакле из $x - t = f(x + y)$ и $y + t = f(z + t)$ следи да је $x - t = y + t$, што је контрадикција, јер је $x - t \neq y + t \in M$. Отуда је формула незадовољива у комбинованој теорији.

Гранање по дељеним једнакостима. Једнакости уведене пропагацијама нису довољне да би се гарантовала коректност поступка комбиновања тео-

рија у случају када теоријски решавачи нису дедуктивно комплетни за дељене једнакости. На пример, претпоставимо да за неке две теорије $\mathcal{T}_i$ и $\mathcal{T}_j$ важи $M \vDash_{\mathcal{T}_i} u = v$ и $M \vDash_{\mathcal{T}_j} u \neq v$ за неку дељену једнакост $u = v$ неодређену у M. Претпоставимо да одговарајући теоријски решавачи нису у стању да открију наведене логичке последице, па самим тим једнакост $u = v$ неће бити уведена у скуп A услед пропагације, као што је раније описано. У оригиналној формулацији DTC методе [22] овакве ситуације не би представљале проблем, зато што би све дељене једнакости од самог почетка рада решавача биле у скупу A. Ово значи да би SAT решавач свакако у неком тренутку применио правило `Decide`, чиме би на траг била постављена једнакост $u = v$ или различитост $u \neq v$, у зависности од изабраног поларитета. У првом случају би теоријски решавач $\mathcal{T}_j$ пријавио конфликт, а у другом случају би то учинио $\mathcal{T}_i$, из чега следи незадовољивост трага M у комбинованој теорији. Међутим, с обзиром на то да се у нашем решавачу дељене једнакости уводе лењо, једнакост $u = v$ не мора бити у скупу A, па до овог гранања по једнакости $u = v$ не мора доћи, што за последицу може имати погрешан закључак решавача да је траг M задовољив. Да би се овакав сценарио предупредио, мора се у неком тренутку увести дељена једнакост $u = v$ у скуп A, како би се гранањем у SAT решавачу одредио њен статус који није било могуће одредити резоновањем. У `argosmt` решавачу, поступак увођења дељених једнакости које су кандидати за гранање је у надлежности теоријских решавача, а стратегија избора дељених једнакости које ће у датом тренутку бити уведене зависи од конкретног теоријског решавача. Оно што је једино битно за коректност поступка комбиновања је да се рад решавача не може завршити докле год постоји бар једна дељена једнакост која је неодређена у M, као што је и наведено у теореми 3.10.1.

Једна могућа стратегија је да се увођење дељених једнакости по којима ће се вршити гранање одлаже колико год је могуће, тј. обавља се само када је решавач у стању у коме су сви атоми из A већ дефинисани у M, $C = no\_cflt$ и није могуће увести ни једну дељену једнакост путем пропагације. Тада теоријски решавач бира једну од једнакости неодређених у M и примењује правило `IntroduceSharedEq` (приметимо да неодређене једнакости свакако нису у A, јер су сви атоми из A већ дефинисани у M). Сада SAT решавач може применити правило `Decide` за уведену дељену једнакост. Овакву стратегију називамо *лењо гранање по дељеним једнакостима.* Сличан приступ се користи и у оригиналној формулацији DTC методе [22], с тим што се тамо дељене једнакости не уводе

лењо, већ се гранање по дељеним једнакостима одлаже тако што се на одговарајући начин адаптира стратегија гранања. Иако би описани поступак лењог гранања могао да примењује било који од теоријских решавача, једноставније је да постоји један теоријски решавач који је за то задужен. У `argosmt` решавачу, ту улогу има теоријски решавач за EUF теорију.

Пример 3.10.2. Претпоставимо да имамо исту ситуацију као у примеру 3.10.1, осим што теоријски решавач за реалну аритметику није дедуктивно комплетан за дељене једнакости, па није у стању да препозна логичку последицу $x + y - z = t \vDash_{\mathcal{T}_1} x + y = z + t$. С обзиром на то да је, након иницијалних јединичних пропагација, траг M задовољив у обе теорије појединачно, а дељена једнакост $x + y = z + t$ није у скупу A, уколико би се решавање овде завршило, дошли бисмо до погрешног закључка да је формула задовољива у комбинованој теорији. Међутим, у описаном приступу лењог гранања, у оваквој ситуацији ће неки од теоријских решавача (EUF решавач у `argosmt` решавачу) увести неку једнакост неодређену у M чиме ће бити омогућено гранање по тој једнакости. Претпоставимо да је то баш дељена једнакост $x + y = z + t$. Након што се ова једнакост дода у скуп A, SAT решавач ће над њом применити правило `Decide`. Претпоставимо да је изабран негативан поларитет, тј. различитост $x + y \neq z + t$ је додата на траг. Сада теоријски решавач $\mathcal{T}_1$ пријављује конфликт, јер је конјункција литерала $x + y - z = t$ и $x + y \neq z + t$ незадовољива у аритметици. Након враћања уназад покушавамо са позитивним поларитетом, али овог пута незадовољивост пријављује EUF решавач, као и у примеру 3.10.1.

Друга могућа стратегија је примењива само у случају да постоји неки теоријски решавач који у току свог рада одржава *текући модел* одговарајуће теорије. Нека су u и v дељени изрази такви да је једнакост $u = v$ неодређена у M и није у A. Ако у текућем моделу $\mathcal{M}_i$ неке теорије $\mathcal{T}_i$ који је пронађен у одговарајућем теоријском решавачу и који задовољава све литерале из M важи $I^{\mathcal{M}_i}(u) = I^{\mathcal{M}_i}(v)$, тада се за једнакост $u = v$ примењује правило `IntroduceSharedEq`. Такође се у стратегији гранања SAT решавача једнакости $u = v$ даје висок приоритет, како би се стимулисао SAT решавач да што пре примени правило `Decide` за ову једнакост (приметимо да се овде не може применити $\texttt{TheoryPropagate}_i$, зато што једнакост $u = v$ важи само у текућем моделу, а не обавезно и у свим моделима теорије $\mathcal{T}_i$ који задовољавају литерале из M). Применом правила `Decide` једнакост $u = v$ се на траг M поставља као претпоставка која је формулисана на основу текућег модела једне од теорија, а о којој

треба да се изјасне остали теоријски решавачи. На овај начин се текући модели теорија користе за усмеравање процеса формирања задовољивог разврставања скупа дељених израза S, чиме теоријски решавачи преузимају активну улогу у том процесу чак и када не постоје дељене једнакости које је могуће пропагирати. Овај приступ је познат као *комбинација теорија заснована на моделу* (енгл. *model-based theory combination*) [32]. Недостатак овог приступа је што је примењив само у случају да теоријски решавач интерно одржава текући модел, што, у зависности од процедуре одлучивања која се користи, није увек случај. Пример процедуре одлучивања која има ову особину је процедура одлучивања за линеарну аритметику заснована на симплекс алгоритму [37]. У нашем решавачу овај приступ се користи управо у теоријском решавачу за линеарну аритметику.

Пример 3.10.3. Претпоставимо поново ситуацију из примера 3.10.2, при чему овог пута претпостављамо да теоријски решавач за реалну аритметику интерно одржава текући модел. У том случају се дељене једнакости могу уводити у скуп A у произвољном тренутку решавања (чак и ако нису сви атоми из A дефинисани у M), ако се интерпретације дељених израза у текућем моделу поклопе. На пример, један модел $\mathcal{M}_1$ теорије $\mathcal{T}_1$ који задовољава траг M може бити модел у коме је $I^{\mathcal{M}_1}(x) = 1$, $I^{\mathcal{M}_1}(y) = -1$, $I^{\mathcal{M}_1}(z) = 0$, $I^{\mathcal{M}_1}(t) = 0$, $I^{\mathcal{M}_1}(f(x+y)) = 1$, $I^{\mathcal{M}_1}(f(z+t)) = -1$ и $I^{\mathcal{M}_1}(f(x)) = -2$. У овом моделу је $I^{\mathcal{M}_1}(x+y) = I^{\mathcal{M}_1}(z+t) = 0$, па зато теоријски решавач уводи дељену једнакост $x + y = z + t$ у скуп A. Овој једнакости се придружује висок приоритет, па SAT решавач убрзо примењује `Decide` за ову једнакост, узимајући позитиван поларитет. Ово доводи до конфликта у EUF теорији, што узрокује враћање уназад и постављање супротног литерала на траг. Међутим, конјункција литерала $x + y - z = t$ и $x + y \neq z + t$ је незадовољива у аритметици, па је траг M незадовољив у комбинованој теорији.

3.11 Подршка паралелизацији

Један од циљева приликом развоја решавача `argosmt` био је и обезбеђивање подршке за имплементацију различитих техника паралелизације. Паралелизација се у оквиру `argosmt` решавача разматра искључиво у моделу паралелног извршавања са *дељеном меморијом*.

Паралелни портфолио. На високом нивоу, решавач подржава *паралелни портфолио*. Главна идеја паралелног портфолија је да се више различито подешених инстанци решавача покрену паралелно над истом улазном формулом, док нека од њих не пронађе решење. Ако нема никакве сарадње између решавача, такав портфолио ћемо називати *прост портфолио*. Са друге стране, решавачи могу размењивати научене клаузе како би делили знање које су стекли у току процеса претраге. Овакав портфолио зваћемо *кооперативни портфолио*.

У оквиру `argosmt` решавача, за креирање паралелног портфолија задужен је *покретач SAT решавача*, који, уколико то корисник захтева, може уместо једне креирати и покренути више инстанци SAT решавача над истом формулом. Свакој инстанци SAT решавача придружују се посебне инстанце одговарајућих теоријских решавача, као и осталих компоненти које се користе у току решавања (попут стратегија гранања, рестартовања и заборављања). Најосетљивије питање је питање коришћења израза првог реда. С обзиром на то да библиотека израза није *безбедна за паралелно извршавање* (енгл. *thread-safe*), креирање и уништавање израза би захтевало синхронизацију између *нити* (енгл. *threads*) у оквиру којих се извршавају инстанце решавача, што би могло да доведе до значајног успоравања услед *сударања на мутексима* (енгл. *mutex contention*). Да бисмо ово избегли, за сваку инстанцу решавача креирамо *засебну фабрику израза*. Библиотека израза подржава *клонирање израза*, тј. креирање израза у оквиру једне фабрике који је идентичан неком изразу у некој другој фабрици израза. Овим механизмом сваки решавач направи у својој фабрици израза сопствену копију улазних клауза на почетку решавања. Такође, у наставку решавања сваки од решавача по потреби креира нове изразе искључиво у оквиру своје фабрике. На овај начин се структуре података са којима раде различите инстанце решавача у потпуности раздвајају, што омогућава њихово истовремено извршавање без потребе за синхронизацијом.

У случају кооперативног портфолија, размењују се научене клаузе чија је дужина мања од неке унапред задате вредности. Све такве клаузе решавачи *извозе*, тј. стављају на располагање другим решавачима. Када неки од решавача жели да увезе клаузе које је други решавач извезао, он клонирањем прави копије извезених клауза у оквиру сопствене фабрике израза. У наставку решавања, сваки решавач барата искључиво својим копијама дељених клауза. Увоз клауза које су извезене од стране других решавача се врши сваки пут када се достигне нулти ниво одлучивања (било након примене правила `Restart`, или

након враћања уназад правилом `Backjump` на ниво 0).

Свака инстанца решавача се може засебно подесити, избором различитих параметара. Један од најједноставних начина је да се сваком од решавача зада различито *семе* (енгл. *seed*) за генерисање случајних бројева. С обзиром на то да подразумевана стратегија избора литерала одлучивања у нашем решавачу користи 5% случајно одабраних литерала, избором различитих семена за генерисање случајних бројева постижемо да се различите инстанце решавача усмеравају у потпуно различите делове простора претраге, чиме се постиже њихова комплементарност.

Паралелизација ниског нивоа. На ниском нивоу, могу се паралелизовати неки скупи алгоритми у оквиру појединих теоријских решавача, чиме се може убрзати њихов рад. Овај вид паралелизације на примеру *симплекс алгоритма* један је од доприноса ове тезе и детаљно се разматра у глави 5, где се упоређује са паралелним портфолијом, а разматра се и могућност хибридизације ова два приступа паралелизацији.

Имплементација паралелизације. За потребе паралелизације, решавач користи Интелову библиотеку TBB (енгл. *Threading Building Blocks* (TBB))[9]. У питању је библиотека која омогућава различите начине паралелизације у оквиру модела паралелног извршавања са дељеном меморијом. Основна парадигма на којој се библиотека заснива је *паралелизација заснована на задацима* (енгл. *task-based parallelization*). Ова техника подразумева да се посао који треба обавити подели на већи број *задатака* који се могу одвијати паралелно, при чему библиотека добијене задатке распоређује доступним нитима водећи рачуна о равномерном распоређивању оптерећења (енгл. *load balancing*). Ово је искоришћено за потребе паралелног портфолија, где се свака од инстанци решавача покреће у оквиру посебног задатка који се предаје TBB библиотеци. Са друге стране, библиотека подржава и одређени број уграђених метода који омогућавају једноставну паралелизацију петљи различитог типа (`tbb::parallel_for`, `tbb::parallel_reduce`, `tbb::parallel_do` и сл.). Ово је искоришћено за паралелизацију ниског нивоа у оквиру симплекс процедуре (глава 5).

[9] `http://www.threadingbuildingblocks.org`

3.12 Сажетак

У овој глави тезе детаљно је описана структура `argosmt` решавача, као и његов принцип рада. Решавач је у потпуности усклађен са DPLL($\mathcal{T}$) архитектуром и системом правила описаним у одељку 2.1.6. Анализиране су његове главне карактеристике и описане појединачне компоненте које га сачињавају. Детаљно је описан интерфејс теоријског решавача, што је од посебног значаја уколико желимо проширење функционалности решавача, уградњом теоријских решавача за нове теорије. Једна од битних карактеристика решавача је и ефикасна имплементација библиотеке израза, као и чињеница да решавач не примењује класично пречишћавање, чиме се избегава увођење дељених константи и дефиниционих једнакости. Нарочито занимљива одлика архитектуре `argosmt` решавача је то што је, за разлику од традиционалног приступа, исказно резоновање измештено из SAT решавача и организовано као посебан теоријски решавач (који смо назвали *исказни решавач*). Овакав приступ значајно поједностављује имплементацију, зато што омогућава да се исказно резоновање третира на исти начин као и резоновање у уобичајеним SMT теоријама. Укратко су описани и остали постојећи теоријски решавачи: решавач за EUF теорију, аритметички решавач и CSP решавач. Најзад, разматрана је и могућност паралелизације `argosmt` решавача.

```
procedure solve()
begin
  layer := 1
  loop
    do {покрећемо резоновање за задати слој (layer)}
      state_changed := false
      for all T_i do
        T_i → checkAndPropagate(layer)
      end for
    while state_changed ∧ C = no_cflt {стање је промењено и нема конфликта}
    if C = no_cflt then {ако нема конфликта}
      layer := layer + 1
      if layer ≤ max_layers then {ако постоји следећи слој, прелазимо на њега}
        continue
      else {у супротном, идемо на проверу рестарта}
        layer := 1
      end if
    else {анализа конфликта}
      while ¬isUIP(C) ∧ C ≠ ∅ do {тражимо UIP или празан конфликтни скуп}
        {Нека је l литерал из C који је последњи на трагу M}
        T_i → explainLiteral(l) {T_i је теорија која је пропагирала l}
      end while
      if C = ∅ then {формула је незадовољива}
        break
      else {пронађен UIP}
        applyBackjump()
        layer := 1
        continue
      end if
    end if
    if треба применити рестартовање then
      applyRestart()
      continue
    end if
    if треба применити заборављање then
      Бирамо скуп клауза cl_set за заборављање
      applyForget(cl_set)
    end if
    if постоји литерал недефинисан у M then
      Бирамо литерал l међу недефинисаним литералима
      applyDecide(l)
    else {формула је задовољива}
      break
    end if
  end loop
  if C = ∅ then
    return unsat
  else
    return sat
  end if
end
```

Алгоритам 3.5: `solve()`

Глава 4

Унапређивање SMT решавача CSP техникама

4.1 Увод

Као што је раније речено, теорије које се обично користе у SMT-у се најчешће бирају на основу њихове примењивости у индустрији — пре свега у *верификацији софтвера* (ово укључује теорије попут аритметике, теорије битвектора, теорије низова, теорије неинтерпретираних функција, и сл.). Дефинисањем нових теорија и укључивањем одговарајућих процедура одлучивања за те нове теорије у SMT решаваче може се проширити примењивост SMT решавача на друге области примене. Једна таква могућност произилази из комбиновања SMT технологија са техникама и алгоритмима који су развијени у оквиру CSP решавача [85]. Опремљени овим алгоритмима, SMT решавачи би могли решавати CSP проблеме знатно ефикасније него што то могу сада. Могућност такве интеграције се помиње и у радовима других аутора [71, 74].

Имајући ове чињенице у виду, у овом делу тезе разматрамо унапређивање SMT решавача у погледу њихове примењивости у решавању CSP проблема. У том циљу, највећи изазов са којим се суочавамо је дефинисање адекватног логичког оквира за изражавање CSP проблема у оквиру SMT-а. Анализом структуре типичних CSP проблема може се уочити да доминантно место заузимају аритметичка ограничења која укључују промењљиве са коначним доменима. Поред њих, већина CSP проблема садржи и нека специјализована глобална ограничења која омогућавају да се поједини услови једноставније и компактније изразе. Ова глобална ограничења се најчешће и сама могу изразити по-

моћу аритметичких ограничења, тако да је често аритметика сасвим довољна за изражавање многих CSP проблема. Управо због тога су се досадашњи покушаји примене SMT решавача у решавању CSP проблема [20] од постојећих SMT теорија углавном ослањали на целобројну линеарну аритметику. Два су основна недостатка овог приступа. Први недостатак је у томе што се приликом кодирања CSP проблема на језику линеарне аритметике, услед разбијања глобалних ограничења и њиховог изражавања у облику аритметичких услова, губе значајне информације о структури разматраног проблема. Иако су добијени аритметички услови еквивалентни са полазним глобалним ограничењем, аритметички теоријски решавач није у стању да препозна семантику глобалног ограничења и да резонује у складу са њом. Други значајан недостатак је у томе што стандардна процедура одлучивања за линеарну аритметику у SMT-у која је заснована на симплекс алгоритму подразумева бесконачне домене за променљиве које се у проблему јављају. Чињеница да у CSP проблемима разматрамо аритметичка ограничења над коначним доменима даје нам могућност да упослимо једноставније процедуре одлучивања које ће радити ефикасније.

Како ове недостатке није могуће отклонити у оквиру линеарне аритметике, решење које предлажемо у оквиру ове тезе је увођење нове SMT теорије, коју ћемо звати *CSP теорија* (поглавље 4.2). Ова теорија ће за своју основу имати целобројну линеарну аритметику, али ће додатно укључивати дефиницију синтаксе и семантике неких типичних глобалних ограничења. Такође, ограничавамо се на фрагмент ове теорије који подразумева коначне границе за неинтерпретиране целобројне константе којима се у формулама представљају CSP променљиве. За овако дефинисани фрагмент имплементираћемо одговарајући теоријски решавач који ћемо звати *CSP теоријски решавач* (поглавље 4.3), а који ће укључивати алгоритме филтрирања за одговарајућа глобална ограничења. Ови алгоритми биће адаптирани и по потреби проширени додатним алгоритмима, како би се уклопили у постојеће SMT окружење. У том смислу постоје два основна изазова која треба превазићи.

Први проблем је интеграција алгоритама филтрирања у оквиру CSP теоријског решавача и њихова *ефикасна комуникација*. Овај проблем у суштини потиче од саме природе CSP техника које покушавамо да искористимо у оквиру нашег теоријског решавача. Наиме, као што је раније речено, од сваког теоријског решавача се очекује да испита да ли постоји модел одговарајуће теорије који истовремено задовољава *све литерале* (тј. сва ограничења) на трагу M. У

случају CSP теорије, такви модели одговарају решењима CSP проблема. Међутим, сваки CSP алгоритам филтрирања типично разматра *само једно* ограничење (тј. један литерал на трагу) и испитује постојање модела (тј. решења) који задовољава то једно ограничење. С обзиром на то да се свако глобално ограничење у проблему разматра понаособ, постојање *локалних* модела који задовољавају појединачна ограничења не значи да мора постојати и *заједнички* модел који истовремено задовољава сва глобална ограничења на трагу решавача. Управо оваква структура CSP теоријског решавача индукује поменути проблем комуникације између ограничења, јер је потребно обезбедити њихово ефикасно усаглашавање у циљу проналажења заједничког модела. Комуникација се остварује на начин који је аналоган раду типичног CSP решавача: *одсецања* и *придруживања* која врше алгоритми филтрирања се механизмом пропагације ограничења размењују између глобалних ограничења, чиме се синхронизују домени CSP променљивих у оквиру различитих ограничења. На жалост, на овај начин се не може доћи до заједничког модела, све док се домени не сведу на једночлане скупове, тј. док се CSP проблем не сведе на *решени облик*. Ово значи да је неопходно у процес решавања укључити и механизам претраге, који укључује гранање и враћање уназад. Приступ који је примењен у нашем решавачу је да се и комуникација између ограничења и процес претраге пренесу на SAT решавач. Све рестрикције домена променљивих (одсецања и придруживања) се изражавају као литерали првог реда, а њиховим постављањем на траг SAT решавача симулира се и претрага (применом правила `Decide`) и пропагација ограничења (применом правила $\texttt{TheoryPropagate}_i$). Литерали се након додавања на траг M од стране SAT решавача прослеђују назад теоријском решавачу, који их даље прослеђује заинтересованим глобалним ограничењима, чиме се затвара круг комуникације. Овим питањима се детаљније бавимо у поглављу 4.3.

Још један могући приступ решавању горе описаног проблема је унапређивање постојећих алгоритама филтрирања тако да, уместо да разматрају свако ограничење понаособ, узимају у обзир и постојање других глобалних ограничења у датом проблему, чиме се може значајно побољшати поступак филтрирања и, самим тим, брже конвергирати ка заједничком моделу. Овакав приступ је у општем случају веома тешко применити, али за неке специфичне комбинације глобалних ограничења се ипак могу постићи значајни резултати. У овој тези се бавимо проблемом комбинације линеарног ограничења и `alldifferent`

ограничења (поглавље 4.5).

Други проблем је *генерисање објашњења* пропагација и конфликата који потичу од глобалних ограничења. Иако се анализа конфликата и нехронолошко враћање уназад у одређеној форми користе и у неким традиционалним CSP решавачима [94], постојећи алгоритми филтрирања често немају могућност генерисања објашњења, или су објашњења која генеришу неадекватна за примену у оквиру SMT решавача. Проблем генерисања објашњења се, због различите природе глобалних ограничења, мора решавати за сваки тип ограничења понаособ. У овој тези бавимо се генерисањем објашњења за `alldifferent` ограничење (поглавље 4.4).

Главни резултат који се постиже уградњом CSP техника у SMT решаваче је да се добија хибридни решавач који се ослања на најзначајнија достигнућа обе истраживачке области. Са једне стране, имамо SMT технологију која је заснована на моћним CDCL SAT решавачима који су, када је у питању обилазак простора претраге, много ефикаснији од традиционалних CSP решавача. Са друге стране, паметни CSP алгоритми филтрирања који су развијени имајући у виду специфичну семантику глобалних ограничења омогућавају резоновање прилагођено тој специфичној семантици. Ово и јесте суштина SMT приступа — резонујемо на знатно вишем нивоу у оквиру одговарајуће теорије са специфичном семантиком, а претрагу вршимо помоћу ефикасних SAT решавача који су у томе без премца у протеклих пар деценија.

4.1.1 Преглед релевантних резултата

Коришћење SAT и SMT технологија за решавање CSP проблема је веома актуелна истраживачка област. Много је радова који се тичу кодирања CSP проблема на језику исказне логике [98, 78, 95, 53]. Ови приступи су *вредни* (енгл. *eager*), тј. проблем се комплетно искодира у облику исказне формуле и предаје SAT решавачу. SMT приступ се такође користи за решавање CSP проблема [20, 54], при чему се CSP проблеми кодирају на SMT-LIB [12] језику, коришћењем стандардних SMT теорија (целобројна линеарна аритметика и теорија битвектора). Колико је аутор ове тезе упознат, не постоје објављени радови који се тичу развоја специфичних SMT теорија које укључују глобална ограничења, као ни укључивања алгоритама филтрирања за таква ограничења у SMT (иако је аутор свестан да је у току известан рад на ту тему [74]).

Још један занимљив приступ коришћења SAT технологије у решавању CSP проблема је тзв. *лењо генерисање клауза* (енгл. *lazy clause generation* (LCG)) [77]. У овом приступу, SAT решавач је интегрисан са алгоритмима филтрирања за глобална ограничења. Када алгоритам филтрирања жели да изврши пропагацију, генерише се клауза која репрезентује ту уочену импликацију. Клауза се онда научи од стране SAT решавача, што изазива јединичну пропагацију одговарајућег литерала. Приступ је доста сличан нашем приступу, с обзиром на то да комбинује SAT технологије са алгоритмима филтрирања за глобална ограничења. Главна разлика је у механизму пропагације: у нашем приступу се не генерише клауза којом се изазива јединична пропагација, већ су алгоритми филтрирања организовани у оквиру SMT теоријског решавача који сâм пропагира имплициране литерале, а касније обезбеђује објашњења по потреби.[1]

4.2 CSP теорија

4.2.1 Синтакса и семантика

CSP теорија се заснива на теорији целобројне аритметике која је проширена глобалним ограничењима. Сигнатура ове теорије садржи само једну сорту `Int` која се интерпретира скупом целих бројева $\mathbb{Z}$. Такође, сигнатура садржи симболе $+, -, \cdot, \leq, \geq, <, >$ који имају уобичајене интерпретације у структури целих бројева. Поред ових симбола, сигнатура CSP теорије укључује и предикатске симболе којима су представљена глобална ограничења. На пример, када је у питању `alldifferent` ограничење, сигнатура садржи пребројиву фамилију симбола:

$$\{\texttt{alldiff}_n : [\underbrace{\texttt{Int}, \ldots, \texttt{Int}}_{n}] \to \texttt{Bool} \mid n \geq 2\}$$

који представљају `alldifferent` ограничења различитих арности. За свако од глобалних ограничења дефинишемо семантику, тако што уводимо услове којима фиксирамо интерпретације одговарајућих предикатских симбола у моделима CSP теорије. На пример, за `alldifferent` ограничење, интерпретација

[1] У последње време, LCG решавачи су такође увели лењо генерисање објашњења, као што је назначено у раду Питера Стакија [96]. Такође, и SAT литерали се у контекст SAT решавача уводе лењо, баш као и у нашем решавачу. Ипак, колико је познато аутору ове тезе, детаљи имплементације и даље нису објављени.

одговарајућег симбола мора задовољавати следећи услов:

$$(\forall X_1)\dots(\forall X_n)(\texttt{alldiff}_n(X_1,\dots,X_n) \Leftrightarrow \bigwedge_{i\neq j}(X_i \neq X_j))$$

Додатно, сигнатура CSP теорије садржи и пребројиво много симбола константи $x_1, x_2, \dots$ сорте `Int` који су неинтерпретирани и које ћемо користити за представљање CSP променљивих[2] у проблемима. Модели CSP теорије се међусобно разликују једино у интерпретацијама ових константи. У наставку наводимо две значајне теореме које се тичу одлучивости и стабилне бесконачности CSP теорије.

Теорема 4.2.1. SMT проблем за CSP теорију је неодлучив.

Доказ. Неодлучивост SMT проблема за CSP теорију следи из неодлучивости проблема задовољивости за целобројну аритметику која је у CSP теорији садржана. □

Теорема 4.2.2. CSP теорија је стабилно бесконачна.

Доказ. Стабилна бесконачност CSP теорије следи из чињенице да сви њени модели за домен имају скуп целих бројева који је бесконачан. □

4.2.2 Фрагмент `QF_CSP`

У даљем разматрању ограничавамо се на базни линеарни фрагмент ове теорије, уз додатну рестрикцију да све константе којима су представљене CSP променљиве у разматраним формулама имају *коначне границе*. Ово значи да за сваку константу x_i у формули постоји бар једна јединична клауза облика $x_i \leq u_i$ и бар једна јединична клауза облика $x_i \geq l_i$, где су u_i и l_i целобројни коефицијенти. На овај начин обезбеђујемо да све CSP променљиве које се користе у проблему имају коначне домене.[3] Описани фрагмент CSP теорије означавамо са `QF_CSP`.

[2]Термин *CSP променљива* означава променљиву у CSP проблему и не треба га мешати са појмом *променљиве у логици првог реда*. Све формуле првог реда које разматрамо у овој глави тезе су базне, тј. не укључују променљиве у смислу логике првог реда, тако да термин *променљива* у наставку ове главе увек означава CSP променљиву.

[3]Термин *домен CSP променљиве* означава скуп вредности које може узети нека CSP променљива у датом CSP проблему и не треба га мешати са појмом *домена модела CSP теорије* који је увек скуп целих бројева $\mathbb{Z}$.

Теорема 4.2.3. SMT проблем за фрагмент `QF_CSP` је одлучив.

Доказ. С обзиром на то да све неинтерпретиране константе у формулама над овим фрагментом имају коначне границе, број потенцијалних модела CSP теорије које треба размотрити је коначан, одакле следи да је испитивање задовољивости таквих формула одлучив проблем. □

Теорема 4.2.4. SMT проблем за фрагмент `QF_CSP` је NP-комплетан.

Доказ. Нека су $x_1, \ldots, x_n$ неинтерпретиране константе које се појављују у датој формули F `QF_CSP` фрагмента, и нека су l_i односно u_i редом доње и горње границе које су у формули F задате за константе x_i. Интерпретација формуле F зависи само од избора вредности за константе x_i. Нека је $(d_1, \ldots, d_n)$ произвољна n-торка из $\mathbb{N}^n$, таква да је $d_i \in [l_i, u_i]$. За сваку такву n-торку можемо у полиномијалном времену у односу на величину формуле проверити да ли задовољава формулу F, при чему се d_i узима за вредност константе x_i. Због тога проблем задовољивости припада класи NP.

За доказ NP-комплетности довољно је свести SAT проблем на овај проблем у полиномијалном времену. Нека су $p_1, \ldots, p_n$ исказна слова која се појављују у датој КНФ исказној формули Ψ. Овој формули придружујемо `QF_CSP` формулу F на следећи начин. За свако исказно слово p_i уведимо две константе x_i и y_i сорте `Int` (интуитивно, x_i одговара литералу p_i, а y_i одговара литералу $\neg p_i$). Сада за свако p_i у формулу F додајемо јединичне клаузе $0 \leq x_i$, $x_i \leq 1$, $0 \leq y_i$ и $y_i \leq 1$. Затим додајемо јединичне клаузе $x_i + y_i = 1$ (овим гарантујемо да је тачно један од литерала p_i и $\neg p_i$ тачан). Најзад, за сваку клаузу $l_1 \vee \ldots \vee l_k$ формуле Ψ у формулу F додајемо јединичну клаузу $L_1 + \ldots + L_k \geq 1$, где је

$$L_i = \begin{cases} x_j, & \text{ако је } l_i = p_j \\ y_j, & \text{ако је } l_i = \neg p_j \end{cases}$$

Овим условом гарантујемо да је бар један од литерала дате клаузе тачан. Добијена формула F припада фрагменту `QF_CSP` и еквизадовољива је са исказном формулом Ψ. Трансформација је, притом, извршена у полиномијалном времену у односу на величину полазне исказне формуле. Одавде следи NP-комплетност полазног SMT проблема, с обзиром на то да је SAT проблем NP-комплетан. □

4.2.3 Представљање CSP проблема

CSP проблеми се на језику `QF_CSP` фрагмента ове теорије представљају базним формулама које укључују неинтерпретиране константе које одговарају CSP променљивама датог проблема. Домени променљивих се задају горе описаним јединичним клаузама које дефинишу границе за константе којима су CSP променљиве представљене у формули. Уколико је потребно да постоје „рупе" у доменима, оне се могу задати јединичним клаузама које садрже одговарајуће различитости (нпр. $x \geq 1 \wedge x \leq 5 \wedge x \neq 3$ дефинише домен $\{1, 2, 4, 5\}$, тј. интервал $[1, 5]$ са „рупом" 3). Најзад, ограничења се изражавају аритметичким условима, као и уз помоћ предикатских симбола који представљају глобална ограничења.

Пример 4.2.1. Размотримо следећи CSP проблем:

$$
\begin{aligned}
&x_1 \in \{1, 2, \ldots, 9\}, x_2 \in \{1, 2, \ldots, 9\}, x_3 \in \{1, 2, \ldots, 9\} \\
&x_4 \in \{1, 2, \ldots, 9\}, x_5 \in \{1, 2, \ldots, 9\}, x_6 \in \{1, 2, \ldots, 9\} \\
&x_1 + 2x_2 \leq 13 \\
&2x_3 + x_4 + x_5 = 25 \\
&x_3 + x_6 \geq x_1 \\
&\texttt{alldifferent}(x_1, x_2) \\
&\texttt{alldifferent}(x_1, x_3, x_4, x_5) \\
&\texttt{alldifferent}(x_2, x_4, x_6)
\end{aligned}
$$

Овом проблему одговара следећа `QF_CSP` SMT формула:

$$
\begin{aligned}
&x_1 \geq 1 \wedge x_1 \leq 9 \wedge x_2 \geq 1 \wedge x_2 \leq 9 \wedge x_3 \geq 1 \wedge x_3 \leq 9 \wedge \\
&x_4 \geq 1 \wedge x_4 \leq 9 \wedge x_5 \geq 1 \wedge x_5 \leq 9 \wedge x_6 \geq 1 \wedge x_6 \leq 9 \wedge \\
&x_1 + 2x_2 \leq 13 \wedge \\
&2x_3 + x_4 + x_5 = 25 \wedge \\
&x_3 + x_6 \geq x_1 \wedge \\
&\texttt{alldiff}_2(x_1, x_2) \wedge \\
&\texttt{alldiff}_4(x_1, x_3, x_4, x_5) \wedge \\
&\texttt{alldiff}_3(x_2, x_4, x_6)
\end{aligned}
$$

CSP променљиве су представљене константама $x_1, \ldots, x_6$ сорте `Int`. С обзиром на то да решавамо CSP проблеме над коначним доменима, све променљиве морају имати коначне домене и то мора бити задато у формули (у овом примеру, све променљиве узимају вредности из домена $\{1, 2, \ldots, 9\}$). Након што су

домени задати, наводе се ограничења. У овом примеру имамо линеарна ограничења која се записују на уобичајен начин, као и `alldifferent` ограничења која су представљена $\texttt{alldiff}_n$ предикатским симболима одговарајућих арности.

4.3 CSP теоријски решавач

4.3.1 CSP литерали

CSP теоријски решавач разуме два типа литерала: *литерале ограничења* (попут $\texttt{alldiff}_3(x_1, x_2, x_3)$, или $2x_1 - 3x_2 + x_3 \leq 17$) и *литерале рестрикција домена* (попут $x \leq 7$, $x = 3$, или $x \neq 2$).[4] Литерали ограничења одговарају глобалним ограничењима која морају бити задовољена, док литерали рестрикција домена садрже само једну променљиву и користе се за изражавање промена домена променљивих. Оба типа литерала се од стране SAT решавача шаљу теоријском решавачу чим се појаве на трагу M, позивом `assertLiteral` методе теоријског решавача. Такође, пропагације и објашњења која генерише теоријски решавач се изражавају оваквим литералима. Другим речима, комплетна комуникација између теоријског решавача и SAT решавача се остварује помоћу ових литерала. Исти важи и за међусобну комуникацију глобалних ограничења, с обзиром на то да се пропагација ограничења остварује разменом литерала рестрикција домена између глобалних ограничења (посредством трага SAT решавача), у циљу синхронизације домена променљивих. Притом се нови литерали рестрикција домена, по потреби, могу креирати и динамички, у току рада решавача. То се догађа онда када се одговарајућа промена домена деси први пут у току рада решавача. Овакав лењи приступ за увођење литерала је веома значајан у смислу ефикасности, јер број потенцијалних литерала рестрикција домена може постати веома велики, у случају великих CSP проблема. Решавач подржава литерале рестрикција домена који су облика $x \bowtie d$, где је $\bowtie \in \{=, \neq, \leq, \geq, <, >\}$.

4.3.2 Структура теоријског решавача

CSP теоријски решавач се састоји из *руковалаца доменима* и *руковалаца ограничењима*. Свакој променљивој придружујемо један руковалац доменом,

[4] Притом, литерале облика $U \leq V$, $U \geq V$ и $U = V$ сматрамо *атомима*, док су литерали $U > V$, $U < V$ и $U \neq V$, респективно, њихове негације.

док сваком глобалном ограничењу које се појављује у формули придружујемо један руковалац ограничењем. CSP теоријски решавач подржава и вишеслојни приступ, а у циљу ефикасног позивања алгоритама филтрирања за сваки од подржаних слојева постоји и један *ред за обраду руковалаца* који садржи руковаоце које треба обрадити у том слоју.

Руковаоци доменима. Улога руковаоца доменом је да иницијализује и одржава (коначни) домен одговарајуће промењљиве. Он препознаје тривијалне логичке последице између литерала рестрикција домена (попут $x < 5 \vDash x < 7$ или $x = 3 \vDash x \neq 4$) и пропагира такве изведене литерале. Такође детектује конфликте који су последица *пражњења домена* (енгл. *domain wipeouts*), када су све вредности одсечене из домена промењљиве. Генерисање објашњења за овакве пропагације и конфликте је једноставно, а такође је у надлежности одговарајућег руковаоца доменом.

Руковаоци ограничењима. Руковаоци ограничењима могу бити различитих типова, у зависности од тога које глобално ограничење представљају. Тренутно су само два типа руковалаца ограничењима имплементирана — за `alldifferent` ограничење и за линеарна ограничења. Сваки руковалац ограничењем имплементира одговарајући алгоритам филтрирања и у могућности је да открије неконзистентности, као и да пропагира литерале који одговарају променама домена које је алгоритам филтрирања извршио. Руковаоци ограничењима морају да омогућавају реконструкцију стања из претходних нивоа одлучивања, за потребе враћања уназад. Најзад, сваки руковалац ограничењем мора бити у могућности да објасни конфликте и пропагације које је открио.

Руковалац ограничењем може бити *активан* и *неактиван*, у зависности од тога да ли је одговарајући литерал ограничења дефинисан или не у текућој парцијалној валуацији M. Притом, приметимо да за сваки литерал ограничења постоји и одговарајући супротни литерал који представља комплементарно ограничење које може захтевати сасвим другачији третман (нпр. за литерал $\texttt{alldiff}_3(x_1, x_2, x_3)$ супротни литерал је $\neg\texttt{alldiff}_3(x_1, x_2, x_3)$, чије је значење да бар две промењљиве морају имати једнаке вредности). Због тога сваки руковалац ограничењем мора имплементирати алгоритме филтрирања и за основно ограничење и за његов комплемент. У зависности од тога који је од литерала на трагу, руковалац покреће одговарајуће алгоритме.

Сваки руковалац ограничењем може да подржава вишеслојни приступ. Ова техника је и раније коришћена у CSP решавачима [86]. Типично се на нижим дедуктивним слојевима имплементирају једноставнији алгоритми филтрирања који успостављају нижи ниво конзистентности, док се скупљи алгоритми који успостављају више нивое конзистентности покрећу у оквиру виших дедуктивних слојева. Укупан број подржаних дедуктивних слојева CSP теоријског решавача ће бити једнак максималном броју дедуктивних слојева које подржава неки од руковалаца ограничењима. Притом, руковаоци који подржавају мањи број слојева могу у оквиру виших слојева покретати исти алгоритам филтрирања као и на нижим слојевима.

Редови за обраду руковалаца. Алгоритми филтрирања се позивају када се домени CSP промењљивих промене, при чему је потребно покренути само алгоритме за она ограничења која укључују неке од промењљивих чији су домени промењени. У том циљу, теоријски решавач садржи *редове за обраду руковалаца*. За сваки подржани слој постоји по један овакав ред. Руковаоци се у редове додају онда када се некој од промењљивих за које су заинтересовани (тј. некој промењљивој која се јавља у ограничењу за које је тај руковалац задужен) промени домен, а са одговарајућег реда се скидају онда када се врши обрада руковалаца у одговарајућем слоју.

4.3.3 Принцип рада теоријског решавача

Као и остали теоријски решавачи `argosmt` решавача, CSP теоријски решавач имплементира интерфејс дат на слици 3.2 (поглавље 3.7). У наставку детаљно описујемо начин рада теоријског решавача.

Нивои одлучивања и враћање уназад. Позивом процедуре `newLevel` се у оквиру свих руковалаца домена и ограничења успоставља нови ниво одлучивања. Ово подразумева памћење стања тих руковалаца на одговарајући начин. Приликом позива `backjump` процедуре, сви руковаоци се враћају у стање које је запамћено на крају одговарајућег нивоа одлучивања. Такође, том приликом се празне сви редови за обраду руковалаца, а руковаоци ограничења за које су одговарајући литерали поништени на трагу M се деактивирају.

Прослеђивање литерала са трага. Позивом `assertLiteral` процедуре SAT решавач обавештава теоријски решавач о литералу који је додат на траг M. Ако је у питању литерал ограничења, тада се одговарајући руковалац ограничењем активира и додаје у све редове руковалаца. Са друге стране, ако је у питању неки литерал рестрикције домена, тада се руковаоцу одговарајућим доменом као и свим руковаоцима ограничења који су заинтересовани за променљиву ограничену тим литералом прослеђује одговарајућа информација о промени домена. Сви наведени руковаоци који су *активни* се затим додају у све редове руковалаца, ако већ нису у њима.

Покретање алгоритама филтрирања. Када се позове процедура `checkAndPropagate`() за неки дедуктивни слој, руковаоци се скидају са одговарајућег реда један по један и одговарајући алгоритам филтрирања се позива. Сваки руковалац примењује правило $\texttt{TheoryConflict}_i$ уколико открије неконзистентност у одговарајућем ограничењу. Такође, руковалац примењује правило $\texttt{TheoryPropagate}_i$ кад год се домен неке од променљивих промени због одсецања учињених од стране алгоритма филтрирања. Притом руковалац може креирати одговарајуће литерале рестрикција домена и увести их у контекст SAT решавача (тако што се одговарајући атом дода у скуп атома A правилом $\texttt{IntroduceAtom}_i$). Након што се литерали пропагирају и поставе на траг M, SAT решавач ће поново позвати процедуру `assertLiteral` нашег теоријског решавача. Овим се пропагирани литерали прослеђују осталим заинтересованим руковаоцима, који се затим додају у редове за обраду руковалаца. Овај поступак се наставља све док одговарајући ред руковалаца не постане празан.

Комуникација између ограничења. Из претходног описа следи да се комуникација између руковалаца одвија искључиво разменом литерала рестрикција домена посредством трага решавача M. На први поглед, ово делује као неефикасан приступ, зато што литерали прелазе дужи пут — најпре се прослеђују SAT решавачу (правилом $\texttt{TheoryPropagate}_i$), а затим их он враћа теоријском решавачу (позивом `assertLiteral` процедуре) који их прослеђује осталим руковаоцима. Алтернатива овом приступу је било остваривање целокупне комуникације између руковалаца унутар теоријског решавача. Ово би захтевало компликованију имплементацију како пропагације ограничења, тако и анализе

конфликта, јер би многе функционалности које већ постоје у SAT решавачу морале бити имплементиране и на нивоу теоријског решавача. Још једна од позитивних страна нашег приступа је и то што се све информације о пропагацијама, конфликтима и објашњењима провлаче кроз SAT решавач који те информације затим користи у својим стратегијама гранања, што омогућава ефикаснију претрагу. Наш приступ је заправо мотивисан раније описаном DTC методом [22] за комбиновање теоријских решавача у SMT-у код које се информације од заједничког интереса за све теоријске решаваче размењују посредством трага решавача, насупрот Нелсон-Опеновој методи код које се комуникација остварује директно између теоријских решавача. Аргументација дата у раду Брутомеса и других [24] која говори у прилог DTC приступу наспрам Нелсон-Опенове методе се може у највећој мери пренети и на случај комуникације између руковалаца у нашем теоријском решавачу.

Формирање заједничког модела. Пропагација ограничења доводи до сужавања домена променљивих, али у општем случају не своди проблем на решени облик, у коме су сви домени променљивих једночлани. У контексту нашег CSP теоријског решавача, свођење на решени облик одговара проналажењу модела теорије који задовољава сва задата ограничења на трагу решавача. Да би се ово постигло, често је потребно додатно гранање. Невоља је у томе што се, услед лењог увођења литерала рестрикција домена, може догодити да су сви постојећи литерали дефинисани на трагу M, а домени CSP променљивих и даље нису једночлани.

Пример 4.3.1. Размотримо следећу формулу:

$$x_1 \geq 1 \wedge x_1 \leq 10 \wedge x_2 \geq 1 \wedge x_2 \leq 10 \wedge x_3 \geq 1 \wedge x_3 \leq 10 \wedge$$
$$\texttt{alldiff}_3(x_1, x_2, x_3) \wedge x1 + 2x_2 + x_3 \leq 9$$

Из ове формуле следи да су иницијални домени све три CSP променљиве $\{1, \ldots, 10\}$, а дата су и два глобална ограничења. Када решавач започне са радом, најпре се сви литерали из горње формуле поставе на траг решавача јединичним пропагацијама. Приметимо да су у том тренутку ово једини литерали који су познати SAT решавачу. Тада руковалац за линеарно ограничење закључује да је $x_1 \leq 5$, $x_2 \leq 3$ и $x_3 \leq 5$ (алгоритам филтрирања ће бити детаљно разматран касније, у поглављу 4.5). Руковалац уводи одговарајуће литерале рестрикција домена у контекст SAT решавача, а затим их пропагира. Руковаоцу `alldifferent` ограничењем се прослеђују ови литерали, али одговарајући

алгоритам филтрирања није у могућности да изврши додатна одсецања, те се процес пропагације ограничења овде зауставља. Сада у скупу свих литерала који постоје у SAT решавачу поред иницијалних литерала из формуле имамо и ова додатна три литерала. Међутим, сви постојећи литерали су већ дефинисани у M, а домени променљивих и даље нису једночлани: $x_1 \in \{1, \ldots, 5\}$, $x_2 \in \{1, \ldots, 3\}$ и $x_3 \in \{1, \ldots, 5\}$.

Наравно, да су сви могући литерали рестрикција домена били унапред уведени у контекст SAT решавача, тада би у наставку SAT решавач применио правило `Decide` за неки од недефинисаних литерала, чиме би се процес тражења решења наставио. Овако, решавач ће се зауставити, иако решење CSP проблема није пронађено. Да би се процес претраге поново покренуо, CSP теоријски решавач бира једну од CSP променљивих x чији домен није једночлан и за њу уводи нови литерал $x \leq d$, где је $d = (l + d)/2$, при чему су l и d доња и горња граница текућег домена променљиве x. Након што се овај литерал уведе, SAT решавач може да грана по том литералу, чиме се претрага наставља. За променљиву гранања узимамо ону променљиву чији је домен најмањи (а није једночлан).[5] У горњем примеру, биће уведен литерал $x_2 \leq 2$ над којим ће бити примењено правило `Decide`.

Генерисање објашњења. У току анализе конфликта, када се позове процедура `explainLiteral`, CSP теоријски решавач делегира тај позив руковаоцу који је изазвао пропагацију (информације о пореклу сваког пропагираног литерала се чувају у оквиру CSP теоријског решавача). Руковалац генерише објашњење и примењује правило $\texttt{TheoryExplain}_i$.

4.4 Руковалац `alldifferent` ограничењем

У овом поглављу описујемо руковалац `alldifferent` ограничењем који имплементира алгоритме за резоновање о `alldifferent` ограничењу у оквиру CSP теоријског решавача. Ово ограничење захтева да све променљиве које су обухваћене тим ограничењем узимају међусобно различите вредности (из својих коначних домена). Ово је једно од најпознатијих глобалних ограничења чије

[5] Наравно, избор променљиве гранања и вредности по којој се врши гранање је питање конкретне стратегије која је изабрана у нашем теоријском решавачу и може се по потреби променити.

примене обухватају решавање логичких загонетки, проблеме распоређивања, комбинаторног дизајна и сл. Због своје примењивости, ово ограничење је интензивно проучавано претходних деценија, при чему је развијено више различитих алгоритама филтрирања. Најпознатији међу њима је *Регинов алгоритам филтрирања* [80] који се заснива на проблему упаривања у бипартитним графовима, и који успоставља *конзистентност домена* над овим ограничењем. Као што је раније речено, интеграција CSP алгоритама филтрирања са SMT решавачем је могућа под условом да алгоритам претходно буде проширен тако да може да генерише објашњења пропагација и конфликата. С обзиром на то да и поједини CSP решавачи такође подржавају анализу конфликата и нехронолошко враћање уназад [94], алгоритми за генерисање објашњења су у претходном периоду такође развијани и у оквиру CSP решавача. Најпознатији међу њима (када је у питању `alldifferent` ограничење) је *Катсирелосов алгоритам* [58], као и алгоритми засновани на проточним мрежама [84, 36]. Ови алгоритми имају значајне недостатке, пре свега када је у питању прецизност и структура добијених објашњења (објашњења или нису минимална, или садрже литерале који су такође пропагирани од стране истог ограничења, што може довести до извесних аномалија у току анализе конфликта). Због тога у овом поглављу разматрамо алтернативни алгоритам за генерисање објашњења за `alldifferent` ограничење који је заснован на *минималном скупу препрека* (енгл. *minimal obstacle set* (MOS)) [7, 4], а који нема наведене недостатке. Овај алгоритам је и најзначајнији допринос овог дела тезе.

У наставку овог поглавља најпре дајемо преглед значајних радова других истраживача на тему `alldifferent` ограничења. Затим презентујемо све алгоритме везане за `alldifferent` ограничење који се користи у оквиру руковаоца `alldifferent` ограничењем нашег CSP теоријског решавача. У циљу потпуности излагања, најпре илуструјемо алгоритме који нису допринос ове тезе, али се користе у оквиру решавача и важни су за разумевање нашег новог алгоритма за генерисање објашњења. Описујемо Форд-Фулкерсонов алгоритам [39], Регинов алгоритам [80] и Катсирелосов алгоритам [58]. Након тога, дајемо детаљан опис MOS проблема и предлажемо алгоритам за његово решавање, уз доказ коректности алгоритма. Затим разматрамо свођење проблема генерисања објашњења за `alldifferent` ограничење на MOS проблем. Такође, упоређујемо објашњења добијена помоћу MOS алгоритма са објашњењима добијеним Катсирелосовим алгоритмом, као и са објашњењима која су добијена алгоритмима

који су засновани на минималним пресецима у проточним мрежама. Поглавље завршавамо експерименталном евалуацијом нашег приступа.

4.4.1 Преглед релевантних резултата

Преглед резултата везаних за `alldifferent` ограничење може се наћи у раду Вилем-Јан ван Ховеа [103], где су приказани алгоритми филтрирања који успостављају различите нивое конзистентности над овим ограничењем. Ово укључује и поменути Регинов алгоритам [80] за успостављање конзистентности домена, као и алгоритме за успостављање конзистентности граница [79, 66, 62] . Када су у питању алгоритми за генерисање објашњења, најрелевантнији резултат је поменути Катсирелосов алгоритам [58]. Овај резултат је даље проширен у раду Нила Мура [67], где је приказан исти алгоритам, али се овог пута објашњења могу генерисати *лењо*, тек када буду потребна. Приступ заснован на проточним мрежама [39] је такође разматран у литератури [84, 36]. Објашњења која се добијају помоћу алгоритма датог у раду Рохарта и других [84] су слична нашим објашњењима, али могу садржати и неке редунданте литерале које наш алгоритам ефикасно одстрањује. Објашњења која се добијају алгоритмом описаним у раду Даунинга и других [36] су иста као и Катсирелосова објашњења, осим што предложени алгоритам може да се користи и за објашњавање придруживања, док Катсирелосов алгоритам може да објашњава само одсецања (наш алгоритам такође може да објашњава и пропагирана одсецања и придруживања на униформан начин). У свом другом раду [35], исти аутори дају детаљнију дискусију о генерисању објашњења за `alldifferent`. У овом раду разматрају се и конзистентност домена и конзистентност граница за `alldifferent` ограничење, а у оба случаја се предлажу алгоритми за генерисање објашњења. У случају конзистентности домена, предложени алгоритам је исти као и алгоритам изложен у раду Даунинга и других [36], што значи да је еквивалентан Катсирелосовом алгоритму, са додатном могућношћу објашњавања пропагираних придруживања. Као и у случају нашег алгоритма, алгоритми дати у поменутим радовима [36, 35] могу да генеришу објашњења лењо.

4.4.2 Бипартитни граф и `alldifferent` ограничење

Ограничење `alldifferent` се обично разматра тако што се своди на проблем упаривања у бипартитном графу [103, 80]. *Бипартитни граф* $B = (U, V, E)$ је неусмерени граф чији су чворови подељени у два дисјунктна подскупа U и V, таква да за сваку грану $(u, v) \in E$, u је из U и v је из V. *Упаривање* у бипартитном графу B је скуп грана $\mathbb{M} \subseteq E$ такав да никоје две гране из $\mathbb{M}$ немају заједнички чвор. Ако грана (u, v) припада $\mathbb{M}$, кажемо да су чворови u и v *упарени*. Чвор је *слободан* ако није упарен ни са једним чвором. Упаривање $\mathbb{M}$ је *оптимално* ако не постоји упаривање веће кардиналности (тј. које садржи већи број грана). Оптимално упаривање је *савршено* ако су сви чворови из U упарени.

Ограничењу `alldifferent`$(x_{i_1}, \ldots, x_{i_k})$ можемо придружити бипартитни граф $B = (U, V, E)$ на следећи начин: U садржи по један чвор u^x за сваку променљиву $x \in \{x_{i_1}, \ldots, x_{i_k}\}$, а V садржи по један чвор v^d за сваку вредност $d \in \bigcup_{j=1}^{k} D_{x_{i_j}}$. Грана (u^x, v^d) постоји у E ако и само ако $d \in D_x$ (свака променљива је повезана са вредностима из свог домена). Сада се проблеми испитивања конзистентности, успостављања конзистентности домена, као и проблем генерисања објашњења своде на одговарајуће проблеме у придруженом бипартитном графу, о чему говоримо у наредним одељцима.

4.4.3 Испитивање конзистентности `alldifferent` ограничења

Проблем испитивања конзистентности `alldifferent` ограничења се своди на проблем проналажења оптималног упаривања у бипартитном графу B који је придружен `alldifferent` ограничењу. Наиме, лако је видети да важи следећа теорема [103].

Теорема 4.4.1. Придруживање $x_{i_1} = d_{i_1}, \ldots, x_{i_k} = d_{i_k}$ задовољава ограничење `alldifferent`$(x_{i_1}, \ldots, x_{i_k})$ ако и само ако је $\mathbb{M} = \{(u^{x_{i_j}}, v^{d_{i_j}}) \mid 1 \leq j \leq k\}$ савршено упаривање у B. Последично, ограничење `alldifferent`$(x_{i_1}, \ldots, x_{i_k})$ је конзистентно ако и само ако постоји савршено упаривање у B. □

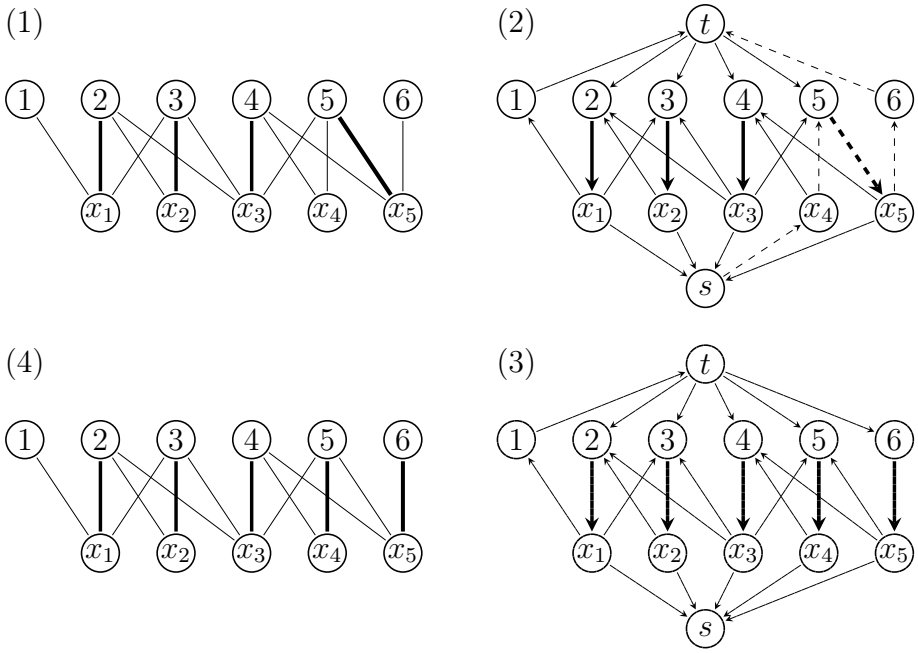
Из теореме 4.4.1 следи да је за испитивање конзистентности довољно пронаћи неко оптимално упаривање у придруженом бипартитном графу. Ако пронађено оптимално упаривање није савршено, тада је ограничење неконзи-

стентно. Процедура која се често користи за проналажење оптималног упаривања у бипартитном графу је *Форд-Фулкерсонов алгоритам* [39]. Алгоритам полази од неког постојећег (неоптималног, могуће и празног) упаривања $\mathbb{M}$ и инкрементално га проширује док се не добије оптимално упаривање. *Резидуални граф* $G_{B,\mathbb{M}}$ за бипартитни граф B са упаривањем $\mathbb{M}$ је усмерени граф са скупом чворова $U \cup V \cup \{s,t\}$ $(s,t \notin U \cup V)$. Његове гране одговарају гранама графа B, где су гране из $\mathbb{M}$ усмерене од V према U, а гране које нису у $\mathbb{M}$ су усмерене од U према V. Чвор s је повезан са свим чворовима из U — свака од грана је оријентисана од s ка $u \in U$ ако u није упарен, а од u ка s у супротном. Слично, чвор t је повезан са свим чворовима из V — свака од грана је усмерена од t ка $v \in V$ ако је v упарен, а од v ка t у супротном. Може се доказати да упаривање $\mathbb{M}$ може бити *увећано* у B (замењено упаривањем веће кардиналности) ако и само ако постоји усмерена путања од s до t у графу $G_{B,\mathbb{M}}$ [103]. Овакве путање зовемо *увећавајуће путање*. Након што је пронађена увећавајућа путања (нпр. претрагом резидуалног графа у ширину), усмерења свих грана у путањи се обрћу, а упаривање се мења у складу са тим (тј. гране усмерене од V ка U се додају у упаривање, а гране усмерене од U ка V се уклањају из упаривања). На овај начин, свака увећавајућа путања повећава кардиналност упаривања за један. Поступак се понавља докле год постоји увећавајућа путања. Временска сложеност алгоритма је $O(k \cdot (|U| + |V| + |E|))$, где је k разлика између броја грана у оптималном упаривању и броја грана у иницијалном упаривању. Због тога ће се алгоритам извршавати брже ако је иницијално упаривање ближе оптималном. Ово чини процедуру погодном за примене где је битна инкременталност (као што је SMT).

Пример 4.4.1. Размотримо пример илустрован на слици 4.1. Први граф је бипартитни граф придружен `alldifferent` ограничењу (подебљане гране су у $\mathbb{M}$), а други граф представља одговарајући резидуални граф (испрекидане гране сачињавају увећавајућу путању). Трећи граф је резидуални граф након промене усмерења грана у откривеној увећавајућој путањи, док у четвртом графу можемо видети ново (увећано) упаривање у бипартитном графу.

4.4.4 Успостављање конзистентности домена

Да бисмо успоставили конзистентност домена над неким ограничењем, морамо да уклонимо све вредности из домена променљивих које су неконзистентне



Слика 4.1: Форд-Фулкерсонов алгоритам

са датим ограничењем (тј. нису део ни једног решења које задовољава то ограничење). У случају `alldifferent` ограничења, неконзистентне вредности одговарају гранама у бипартитном графу које нису део ни једног савршеног упаривања. Такве гране зовемо *неконзистентне гране*. Све такве гране треба пронаћи и уклонити (одсећи) из графа, а одговарајућа одсецања треба пропагирати.

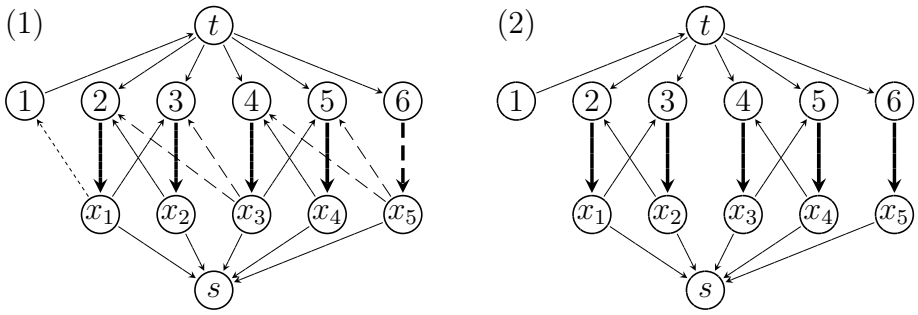
Други битан тип грана чине *витални гране* — гране које припадају свим савршеним упаривањима. Ако је грана витална, то значи да је дата вредност придружена одговарајућој променљивој у свим решењима која задовољавају ограничење, тако да се одговарајуће придруживање може пропагирати.

Гране које нису ни витални ни неконзистентне, тј. које припадају неким, али не свим савршеним упаривањима зову се *алтернирајуће гране*. Следећа теорема једноставно следи [80].

Теорема 4.4.2. Свака грана e у бипартитном графу B са савршеним упаривањем $\mathbb{M}$ је или неконзистентна, или витална или алтернирајућа. Даље, ако грана e није алтернирајућа, тада је она витална ако припада савршеном упаривању $\mathbb{M}$, а у супротном је неконзистентна. □

Теорема 4.4.2 нам говори да ако успемо да пронађемо све алтернирајуће

гране у графу, тада ће бити лако да разврстамо остале гране на виталне и неконзистентне, на основу њиховог присуства у текућем савршеном упаривању $\mathbb{M}$. Ово је главна идеја Региновог алгоритма [80].



Слика 4.2: Регинов алгоритам

Наредна теорема [14] обезбеђује ефективан начин за проналажење свих алтернирајућих грана у бипартитном графу.

Теорема 4.4.3. Нека је B бипартитни граф са савршеним упаривањем $\mathbb{M}$. Грана e је алтернирајућа у B ако и само ако припада неком усмереном циклусу у резидуалном графу $G_{B,\mathbb{M}}$. □

Доказ ове теореме се може наћи у литератури [14]. Гране које припадају усмереним циклусима графа $G_{B,\mathbb{M}}$ могу се открити тако што се пронађу све компоненте јаке повезаности у $G_{B,\mathbb{M}}$. *Компонента јаке повезаности* у усмереном графу је максимални подскуп чворова такав да су свака два чвора међусобно достижна (тј. постоји усмерена путања од једног до другог чвора и обратно). Грана (u, v) припада неком усмереном циклусу ако и само ако су чворови u и v у истој компоненти повезаности. Компоненте јаке повезаности се могу пронаћи *Тарџановим алгоритмом* [99] који је заснован на једном обиласку графа у дубину.

Пример 4.4.2. Размотримо пример илустрован на слици 4.2. Претпоставимо да је одсецање $x_1 \neq 1$ задато (тј. наметнуто ограничењу). Ово доводи до уклањања гране $(x_1, 1)$ из графа (тачкаста грана у графу 1). Када се покрене Регинов алгоритам, он прво пронађе и означи све алтернирајуће гране (позивом Тарџановог алгоритма). Гране које нису означене том приликом (испрекидане гране у графу 1) су или виталне (грана $(x_5, 6)$) или неконзистентне (остале четири

испрекидане гране). Након што се одговарајућа придруживања и одсецања пропагирају, неконзистентне гране се уклањају из графа (као што је приказано у графу 2).

Регинов алгоритам се може алтернативно описати у терминима Холових скупова. *Холов скуп* S (енгл. *Hall set*) је подскуп скупа променљивих датог `alldifferent` ограничења такав да важи $|S| = |T|$, где је T унија домена свих променљивих из S (тзв. *комбиновани домен*). С обзиром на то да променљиве из S морају узети различите вредности, следи да ће све вредности из T бити искоришћене од стране променљивих из S. Због тога можемо одсећи све вредности које припадају T из свих домена осталих променљивих које нису у S. Регинов алгоритам у суштини ради управо то — позива Тарџанов алгоритам да пронађе компоненте повезаности, а затим одсеца све гране чији су чворови у различитим компонентама повезаности. С обзиром на то да компоненте повезаности одговарају Холовим скуповима, грана која повезује чворове из различитих компоненти повезаности заправо повезују променљиву x из једног Холовог скупа са вредношћу d која припада комбинованом домену другог Холовог скупа. Због тога придруживање $x = d$ не може бити део решења које задовољава дато ограничење.

4.4.5 Катсирелосов алгоритам за генерисање објашњења

У контексту `alldifferent` ограничења, објашњења ће бити формулисана као скупови одсецања вредности из домена променљивих (тј. грана из одговарајућег бипартитног графа). Формално, под објашњењем пропагације (одсецања или придруживања) l ћемо овде подразумевати било који подскуп E скупа свих одсецања R која су се догодила пре l такав да E имплицира l. Слично, објашњење конфликта (тј. неконзистентности ограничења) је било који подскуп E скупа свих одсецања R у тренутку конфликта такав да је E довољан да изазове неконзистентност ограничења.

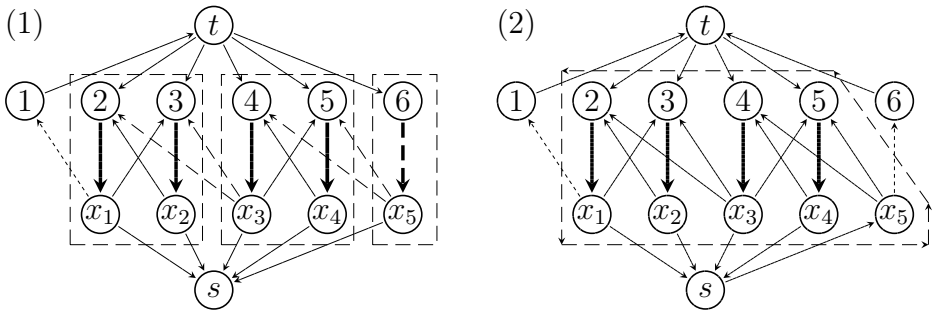
Катсирелосов алгоритам за генерисање објашњења [58] је непосредно заснован на претходној дискусији о односу између Региновог алгоритма и Холових скупова. О томе детаљно говори следећа теорема [58].

Теорема 4.4.4. Нека је $x \neq d$ одсецање које треба објаснити, и нека су S и T, редом, Холов скуп и одговарајући комбиновани домен такви да $x \notin S$ и $d \in T$. Нека је $E = \{x' \neq d' \mid x' \in S,\ d' \notin T\}$, тј. скуп свих одсецања вредности које не

припадају комбинованом домену T из домена променљивих које припадају S. Тада је E објашњење одсецања $x \neq d$.

Доказ. Приметимо да је скуп E управо скуп свих одсецања која су скуп S учинила Холовим скупом. Због тога се, према претходној дискусији, вредност $d \in T$ не може придружити променљивој $x \notin S$, што значи да E имплицира $x \neq d$. □

С обзиром на то да оригинални Катсирелосов алгоритам генерише објашњења *вредно*, тј. одмах након позива Региновог алгоритма, компоненте јаке повезаности су познате у том тренутку, тако да је једноставно пронаћи ону компоненту повезаности која садржи одсечену вредност (тј. компоненту $S \cup T$, где је S Холов скуп, а T је одговарајући комбиновани домен од S такав да $d \in T$). Да бисмо могли да објашњење генеришемо *лењо*, тј. касније када (и ако) за њим заиста буде постојала потреба, морамо бити у могућности да реконструишемо претходно стање графа (које је важило у тренутку када је одсецање пропагирано). Ова реконструкција претходног стања је могућа уз помоћ запамћене историје одсецања грана из графа која се одржава у току рада решавача. Након што се претходно стање графа реконструише, мора се поново позвати Тарцанов алгоритам, како би се поново открили компоненте јаке повезаности у том стању графа [67, 45].



Слика 4.3: Катсирелосова објашњења

Конфликт може бити објашњен на сличан начин [35]. Ограничење `alldifferent` је неконзистентно ако и само ако постоји скуп променљивих S такав да је $|S| > |T|$, тј. има више променљивих у S него што има вредности у његовом комбинованом домену T [103]. Следећа теорема описује објашњења конфликата заснована на Катсирелосовом алгоритму [35].

Теорема 4.4.5. Нека је B бипартитни граф придружен неконзистентном `alldifferent` ограничењу, и нека је $\mathbb{M}$ оптимално (али не и савршено) упаривање у B. Нека је $u \in U$ неупарен чвор (који одговара променљивој којој није придружена вредност), нека је S скуп променљивих које одговарају чворовима из U који су достижни из u у графу $G_{B,\mathbb{M}}$ и нека је T комбиновани домен променљивих из S. Скуп одсецања $E = \{x' = d' \mid x' \in S,\ d' \notin T\}$ је једно објашњење конфликта.

Доказ. Приметимо да скуп чворова који су достижни из u у графу $G_{B,\mathbb{M}}$ не садржи ни један неупарени чвор из V (зато што је упаривање $\mathbb{M}$ оптимално). Такође, овај скуп мора садржати више чворова из U (који одговарају променљивама из S) него чворова из V (који одговарају вредностима из T), с обзиром на то да је чвор u неупарен. Скуп E је управо разлог зашто је $|S| > |T|$, па је самим тим, на основу претходне дискусије, он управо разлог зашто је ограничење неконзистентно. □

Пример 4.4.3. Настављајући пример 4.4.2, након што се обаве одсецања, идентификујемо 3 Холова скупа (уоквирени делови графа 1 на слици 4.3). Два од њих одговарају компонентама повезаности ($\{x_1, x_2\}$ и $\{x_3, x_4\}$) док је трећи скуп једночлан ($\{x_5\}$). Објашњење пропагираног придруживања $x_5 = 6$ се састоји из одсечених грана које напуштају оквир одговарајућег Холовог скупа ($x_5 \neq 4$ and $x_5 \neq 5$). Објашњење одсецања $x_5 \neq 4$ се састоји из одсечених грана које напуштају оквир Холовог скупа који користи вредност 4 ($x_3 \neq 2$ and $x_3 \neq 3$). На сличан начин, објашњење одсецања $x_3 \neq 3$ је одсецање $x_1 \neq 1$.

Граф 2 на слици 4.3 је пример објашњења конфликта. Претпоставимо да су $x_1 \neq 1$ и $x_5 \neq 6$ наметнута одсецања. Када одсечемо одговарајуће гране (тачкасте гране на графу 2), чвор x_5 више није упарен. Ако започнемо обилазак графа од тог чвора, достижемо све чворове у уоквиреној зони графа. Уоквирена зона садржи 5 чвора из U (скуп променљивих $S = \{x_1, x_2, x_3, x_4, x_5\}$) и 4 чвора из V (комбиновани домен $T = \{2, 3, 4, 5\}$). Приметимо да је $|S| > |T|$. Објашњење неконзистентности ограничења се састоји из одсечених грана које напуштају уоквирену област ($x_1 \neq 1$ и $x_5 \neq 6$).

4.4.6 Проблем минималног скупа препрека (MOS)

Нека је $G = (V, E)$ усмерени граф, са скупом *почетних чворова* $S \subseteq V$ и скупом *завршних чворова* $F \subseteq V$. Кажемо да скуп *препрека* $O \subseteq E$ *раздваја*

S од F ако произвољна путања од било ког чвора $v \in S$ ка било ком чвору $f \in F$ садржи бар једну грану из O. Кажемо да је чвор $v \in V$ *блокиран* скупом препрека O (или O-*блокиран*) ако O раздваја скуп $\{v\}$ од скупа F, а у супротном је O-*неблокиран*. Притом, ако O раздваја S од F, тада су сви чворови $v \in S$ O-блокирани, а сви чворови $f \in F$ су O-неблокирани. За путању која не садржи ни једну путању из O кажемо да је *проходна* или O-*проходна* путања. Скуп препрека O који раздваја S од F је *минимални скуп препрека* ако не постоји прави подскуп O' скупа O који такође раздваја S од F. Следећа теорема даје неопходне и довољне услова да скуп O буде минимални скуп препрека.

Теорема 4.4.6. Ако је дат граф G и скупови чворова S и F, тада је скуп препрека O који раздваја S од F минималан ако и само ако је за сваку препреку $e = (v, w) \in O$ чвор v достижан из неког чвора $u \in S$ помоћу неке O-проходне путање, а чвор w је O-неблокиран.

Доказ. Претпоставимо да O задовољава наведене услове и докажимо да је O тада минимални скуп препрека. Претпоставимо супротно, да неки скуп препрека O' који је прави подскуп од O такође раздваја S од F. Тада постоји препрека $e = (v, w) \in O \setminus O'$. С обзиром на то да је v достижан из неког чвора $u \in S$ помоћу неке O-проходне путање, а чвор w је O-неблокиран, што значи да је неки чвор $f \in F$ достижан из w помоћу неке O-проходне путање, а с обзиром на то да $e \notin O'$, тада постоји O'-проходна путања од u до f, што је у контрадикцији са претпоставком да O' раздваја S од F.

Сада претпоставимо да је O минимални скуп препрека и докажимо да за сваку грану $e = (v, w) \in O$ важи да је чвор v достижан из неког чвора $u \in S$ помоћу неке O-проходне путање, као и да је чвор w O-неблокиран. Претпоставимо супротно, да за неку препреку $e = (v, w)$ или не постоји O-проходна путања од чворова из S ка v, или не постоји O-проходна путања од w ка чворовима из F. У том случају, скуп $O' = O \setminus \{e\}$ такође раздваја S од F, зато што не постоји O'-проходна путања од S ка F. Ово је у контрадикцији са претпоставком да је O минимални скуп препрека. □

Проблем минималног скупа препрека (енгл. *minimal obstacle set problem*, скраћено *MOS*) за дати скуп препрека O који раздваја S од F дефинишемо на следећи начин: пронаћи скуп препрека $O_{min} \subseteq O$ такав да је O_{min} минимални скуп препрека који раздваја S од F. Такав скуп не мора бити јединствен.

```
procedure findMinimalObstacleSet(G, S, F, O)
begin
  {G = (V, E) је усмерени граф, S ⊆ V, F ⊆ V}
  {O ⊆ E је скуп грана који раздваја S од F}
  O_r := findReachableObstacles(G, S, O)
  V_ub := findUnblockedVertices(G, F, O_r)
  O_min := ∅
  for all (v, w) ∈ O_r do
    if w ∈ V_ub then
      O_min := O_min ∪ {(v, w)}
    end if
  end for
  return O_min {минимални скуп препрека}
end
```

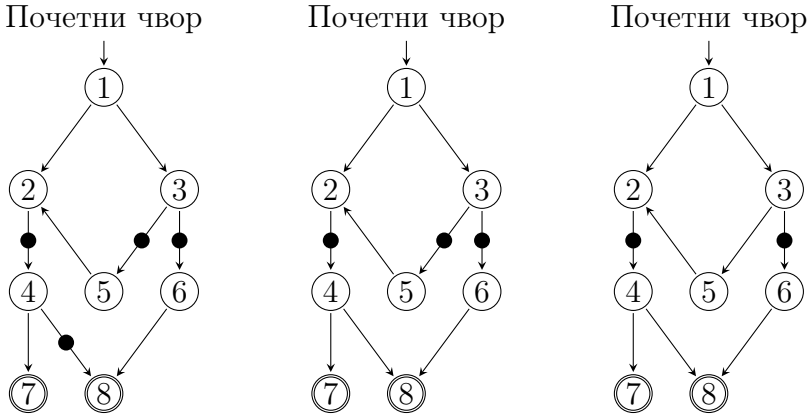
Алгоритам 4.1: `findMinimalObstacleSet`(G, S, F, O)

Процедура `findMinimalObstacleSet`() (алгоритам 4.1) најпре проналази скуп препрека $e = (v, w) \in O$ таквих да је v достижан (помоћу O-проходне путање) из неког чвора из скупа S. Она позива процедуру `findReachableObstacles`() (алгоритам 4.2) која покреће претрагу графа G у ширину полазећи од чворова из S, користећи само O-проходне путање. Скуп чворова достигнут на тај начин означимо са O_r. Лако се може видети да скуп препрека O_r такође раздваја S од F.

У другој фази алгоритма, за сваку препреку $e = (v, w) \in O_r$ проверавамо да ли је чвор w O_r-неблокиран. У ову сврху се позива процедура `findUnblockedVertices`() (алгоритам 4.3). Ова процедура враћа скуп свих O_r-неблокираних чворова из G (означимо овај скуп са V_{ub}). Препрека $e = (v, w) \in O_r$ ће бити у скупу O_{min} ако и само ако је $w \in V_{ub}$. На основу теореме 4.4.6, овако добијен скуп препрека O_{min} је минималан.

Пример 4.4.4. Размотримо граф на слици 4.4. Скуп почетних чворова је $S = \{1\}$, а скуп завршних чворова је $F = \{7, 8\}$. У првом графу је дат иницијални скуп препрека $O = \{(2, 4), (3, 6), (3, 5), (4, 8)\}$ (гране означене црним тачкама). Скуп препрека O_r након уклањања недостижне препреке $(4, 8)$ је дат у другом графу. Најзад, минимални скуп препрека $O_{min} = \{(2, 4), (3, 6)\}$ је дат у трећем графу. Препрека $(3, 5)$ је уклоњена зато што је чвор 5 блокиран од стране препреке $(2, 4)$.

Процедура `findUnblockedVertices`() (алгоритам 4.3) захтева детаљније објашњење. Заснива се на Тарџановом алгоритму за проналажење компоненти јаких повезаности у усмереном графу [99]. Мотивација за коришћење Тарџа-



Слика 4.4: Скуп минималних препрека

```
procedure findReachableObstacles(G, S, O)
begin
  {G = (V, E) је усмерени граф}
  {S ⊆ V је скуп почетних чворова}
  {O ⊆ E је скуп препрека}
  q.init() { q је ред чворова }
  O_r := ∅
  for all u ∈ S do {полазимо од почетних чворова}
    q.enqueue(u)
    u.marked := true
  end for
  while not q.empty() do {претрага у ширину}
    v := q.dequeue()
    for all (v, w) ∈ E do
      if (v, w) ∈ O then {пронађена препрека}
        O_r := O_r ∪ {(v, w)}
      else if not w.marked then
        w.marked := true
        q.enqueue(w)
      end if
    end for
  end while
  return O_r {O_r је скуп препрека достижних из S кроз O-проходне путање}
end
```

Алгоритам 4.2: `findReachableObstacles`(G, S, O)

новог алгоритма за проналажење скупа неблокираних чворова лежи у следећој чињеници: за сваку компоненту повезаности W графа G, ако је један њен чвор неблокиран, тада су и сви остали њени чворови неблокирани (с обзиром на узајамну достижност чворова из исте компоненте). Дакле, неблокираност је својство компоненте повезаности, а не појединачних чворова. Кажемо да

```
procedure findUnblockedVertices(G, F, O)
begin
  {G = (V, E) је усмерени граф, F ⊆ V}
  {O ⊆ E је скуп препрека}
  index := 1
  S.init() {S је стек чворова}
  V_ub := ∅
  for all v ∈ V do {покрећемо претрагу из свих недостигнутих чворова}
    if v.index = undef then
      find_ub_rec(v)
    end if
  end for
  return V_ub {V_ub је скуп свих O-неблокираних чворова}
end

procedure find_ub_rec(v)
begin
  v.index := index
  v.lowlink := index
  index := index + 1
  S.push(v)
  if v ∈ F then
    V_ub := V_ub ∪ {v} {завршни чворови су неблокирани}
  end if
  for all (v, w) ∈ E \ O do {претрага кроз „проходне" гране}
    if w.index = undef then
      find_ub_rec(w)
      v.lowlink := min(v.lowlink, w.lowlink)
    else if w ∈ S then
      v.lowlink := min(v.lowlink, w.index)
    end if
    if w ∈ V_ub then
      V_ub := V_ub ∪ {v} {ако је w неблокиран, тада је и v неблокиран}
    end if
  end for
  if v.lowlink = v.index then {v је корен компоненте повезаности}
    repeat {скидамо елементе компоненте повезаности са стека}
      w := S.pop()
      if v ∈ V_ub then
        V_ub := V_ub ∪ {w} {ако је корен v неблокиран, тада је и w неблокиран}
      end if
    until w = v
  end if
end
```

Алгоритам 4.3: `findUnblockedVertices`(G, F, O)

је компонента W O-неблокирана ако су њени чворови O-неблокирани, а у супротном је O-блокирана. Због бољег разумевања, најпре ћемо објаснити оригинални Тарџанов алгоритам, а затим ћемо указати на модификације у нашем алгоритму и детаљно их размотрити. Тарџанов алгоритам у основи врши стан-

дардни обилазак графа G у дубину. Током обиласка, чворови у V се нумеришу у складу са поретком у ком су први пут посећени, тј. врши се долазна нумерација (енгл. *preorder*). Број који је придружен чвору v зваћемо *индекс* и означаваћемо га са $v.index$ (иницијално је $v.index = undef$, што значи да чвор још увек није посећен). За сваку компоненту повезаности W, дефинишемо *корен* од W као чвор из W који је први посећен у току обиласка, тј. чвор из W са најмањим индексом. Сваком чвору v је такође придружен број $v.lowlink$ који се израчунава у току обиласка и који представља најмањи индекс међу индексима свих чворова који су достижни из v. Последично, чвор u је корен неке компоненте повезаности ако и само ако важи $u.index = u.lowlink$. Чворови се најпре постављају на стек док се не пронађе корен њихове компоненте повезаности, а онда се скидају са стека и додају у одговарајући скуп чворова који представља текућу компоненту повезаности.

Процедура `findUnblockedVertices`() се разликује од описаног Тарџановог алгоритма у две ствари. Прво, разматра се јака повезаност само у односу на O-проходне путање (тј. користи се само скуп грана $E \setminus O$). Друго, алгоритам не враћа пронађене компоненте повезаности као резултат. Уместо тога, користи те компоненте повезаности да пронађе скуп V_{ub} O-неблокираних чворова. Процедура додаје следеће чворове у скуп V_{ub} (и само њих):

- ако је $v \in F$, тада се v додаје у V_{ub} (завршни чворови су тривијално неблокирани)
- ако је грана $(v, w) \notin O$ и чвор w је већ у V_{ub}, тада се v додаје у V_{ub}.
- ако је v корен компоненте повезаности W и v је у V_{ub}, тада се сви чворови из W додају у V_{ub}.

Коректност алгоритма следи из следеће две теореме.

Теорема 4.4.7. Процедура `findUnblockedVertices`() је *сагласна*: ако је v у V_{ub} на крају извршавања процедуре, тада је v O-неблокиран.

Доказ. Ако је чвор v додат у скуп V_{ub} током извршавања процедуре, тада је то или зато што је $v \in F$, или зато што постоји грана $(v, w) \notin O$ и w је већ у V_{ub}, или је корен компоненте повезаности којој припада v већ у скупу V_{ub}. У сваком од ова три случаја, нека је k_v кардиналност скупа V_{ub} у тренутку непосредно пре додавања чвора v у скуп V_{ub}. Доказујемо да је v O-неблокиран индукцијом

по k_v. За $k_v = 0$ (тј. скуп V_{ub} је празан), једини могући случај је да је $v \in F$, тако да је v тривијално O-неблокиран. Претпоставимо сада да је $k_v > 0$ и да тврђење важи за свако $k' < k_v$. У првом случају, ако је $v \in F$, тада је v поново тривијално неблокиран. У другом случају, чвор v се додаје у скуп V_{ub} зато што је неки чвор w већ у V_{ub}, а постоји грана $(v, w) \notin O$. С обзиром на то да је w додат у скуп V_{ub} пре чвора v, тада је $k_w < k_v$, па је по индуктивној хипотези w O-неблокиран. Због тога је v такође неблокиран, с обзиром на то да постоји грана $(v, w) \notin O$. У трећем случају, чвор v се додаје у скуп V_{ub} зато што је корен w његове компоненте повезаности већ у V_{ub}. Поново, w је додат у скуп V_{ub} пре чвора v, тако да је $k_w < k_v$, што значи да је w O-неблокиран, на основу индуктивне хипотезе. Зато је v такође O-неблокиран, с обзиром на то да постоји усмерена O-проходна путања од v до w. Овим смо доказали да сваки чвор v који је додат у скуп V_{ub} у било ком кораку алгоритма заиста O-неблокиран. □

Теорема 4.4.8. Процедура `findUnblockedVertices`() је *комплетна*: ако је чвор u O-неблокиран, тада ће u бити у скупу V_{ub} по завршетку рада процедуре.

Доказ. Да бисмо доказали комплетност, морамо да докажемо да ће за сваку неблокирану компоненту повезаности W сви њени чворови бити додати у скуп V_{ub}. У доказу користимо следеће ознаке:

- $W(v)$ је компонента јаке повезаности која садржи чвор v.
- $root(W)$ је корен компоненте W.
- $root(v)$ је корен компоненте повезаности која садржи v (тј. $root(v) = root(W(v))$).
- *удаљеност* неблокиране компоненте повезаности W од F (означена са $d(W)$) је дужина најкраће O-проходне путање $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, такве да је $v_0 \in W$ и $v_k \in F$. Приметимо да ако је $W \cap F \neq \varnothing$, тада је $d(W) = 0$.
- Рекурзивни позив `find_ub_rec`(v) (краће означен као v-*позив*) је *активан* ако процедура или извршава управо v-позив или неки рекурзивни позив који је директно или индиректно позван из v-позива.
- $v \to w$ означава да је w-позив покренут (директно) из v-позива. Рефлексивно и транзитивно затворење релације $\to$ означавамо са $v \to^* w$ (то значи да је или $v = w$, или $v \to v_1 \to v_2 \to \ldots \to w$). Другим речима,

$v \rightarrow^* w$ значи да је v-позив још увек активан у тренутку када се w-позив извршава. Приметимо и да важи $root(v) \rightarrow^* v$ за сваки чвор v.

- кажемо да је чвор v *једноставно неблокиран* (или *s-неблокиран*) ако је v додат у V_{ub} *пре* завршетка v-позива. У супротном, чвор је *одложено неблокиран* (или *d-неблокиран*). Одложено неблокирани чвор v је додат у V_{ub} *након* завршетка v-позива, али пре завршетка $root(v)$-позива (у завршној фази $root(v)$-позива, када се чворови из компоненте јаке повезаности чвора $root(v)$ скидају са стека).

Приметимо да ако је $v \in F$, тада је v s-неблокиран, с обзиром на то да на почетку v-позива процедура проверава да ли је v у F и додаје v у V_{ub} по потреби. Отуда ће v сигурно бити у V_{ub} пре краја v-позива.

Најпре доказујемо да ако је $v \rightarrow^* w$ и чвор w је s-неблокиран, тада је и v такође s-неблокиран. Да бисмо то доказали, најпре размотримо случај када је $v \rightarrow w$. У том случају, w-позив се покреће директно из v-позива. Када се w-позив заврши и ток извршења алгоритма се врати у v-позив, биће примећено да је w у скупу V_{ub} (јер је w s-неблокиран) и чвор v ће бити додат у V_{ub}. Исто важи и за релацију $\rightarrow^*$, с обзиром на то да је у питању транзитивно затворење релације $\rightarrow$. Приметимо да, с обзиром на то да важи $root(v) \rightarrow^* v$, ако је v s-неблокиран, тада је и $root(v)$ такође s-неблокиран.

У наставку доказа, показујемо да је за сваку неблокирану компоненту повезаности W чвор $root(W)$ s-неблокиран. Ово доказујемо индукцијом по $d(W)$. За $d(W) = 0$ тврђење тривијално важи, с обзиром на то да постоји чвор $v \in W \cap F$ и v је s-неблокиран, па је $root(v) = root(W)$ такође s-неблокиран, на основу претходне дискусије. Претпоставимо да тврђење важи за $d(W) < k$ и докажимо да тада важи и за $d(W) = k$. Заиста, ако важи $d(W) = k$, то значи да постоји O-проходна путања $P = (v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, таква да је $v_0 \in W$, $v_k \in F$, при чему је то најкраћа таква путања (последично, $v_1, \ldots, v_k \notin W$). Докажимо да док се извршава v_0-позив, v_1-позив није активан. Заиста, у супротном би важило да је $v_1 \rightarrow^* v_0$, што значи да је v_0 достижан из v_1. То даље значи да је $v_1 \in W$, супротно претпоставци. Исто тврђење важи и за $root(v_1)$-позив. Ово значи да су или и v_1-позив и $root(v_1)$-позив оба већ завршена или нису још ни започети у тренутку када је грана (v_0, v_1) посећена из v_0-позива. Може наступити једна од следећих ситуација:

- v_1 је већ раније био посећен. У том случају, v_1-позив и $root(v_1)$-позив су

већ завршени. Компонента $W(v_1)$ је такође неблокирана (с обзиром на то да постоји путања $P \setminus (v_0, v_1)$ дужине $k-1$) и $d(W(v_1)) = k-1 < d(W)$. По индуктивној претпоставци $root(v_1)$ је s-неблокиран. Због тога је чвор v_1 у скупу V_{ub}, с обзиром на то да су чворови из компоненте $W(v_1)$ додати у скуп V_{ub} најкасније на крају $root(v_1)$-позива.

- v_1 није до сада био посећен. У том случају, v_1-позив и $root(v_1)$-позив још увек нису покренути. Ово значи да је v_1 први чвор из $W(v_1)$ који ће бити посећен, тј. $root(v_1) = v_1$. Тада покрећемо v_1-позив из v_0-позива. С обзиром на то да је $d(W(v_1)) = k-1 < d(W)$, на основу индуктивне претпоставке следи да је чвор v_1 s-неблокиран и да ће бити у скупу V_{ub} по повратку из v_1-позива.

У оба случаја, процедура открива да је $v_1 \in V_{ub}$ и додаје чвор v_0 у скуп V_{ub}. С обзиром на то да се ово дешава у току извршења v_0-позива, чвор v_0 ће бити s-неблокиран. Због тога је $root(v_0) = root(W)$ такође s-неблокиран.

Ако је за компоненту W чвор $root(W)$ s-неблокиран, тада ће у завршној фази извршавања $root(W)$-позива сви чворови из W бити додати у скуп V_{ub}. С обзиром на то да смо доказали да је за сваку неблокирану компоненту W њен корен s-неблокиран, тада следи да ће сви њени чворови бити у скупу V_{ub} по завршетку извршавања процедуре. Овим смо доказали комплетност. □

Сложеност. Иако је процедура `findMinimalObstacleSet`() заснована на два обиласка графа, може се оптимизовати тако да се извршава у само једном обиласку графа. Наиме, чворови који су посећени у току извршења процедуре `findReachableObstacles`() су достижни из S и свакако су O_r-блокирани (иначе би неки чвор из $u \in S$ био неблокиран, а то је у контрадикцији са претпоставком да скуп препрека O_r раздваја S од F). Ово значи да нема потребе да посећујемо ове чворове поново у процедури `findUnblockedVertices`() како бисмо проверили да ли су они неблокирани. Самим тим, сложеност алгоритма је $O(|V| + |E|)$.

4.4.7 Коришћење MOS алгоритма за генерисање објашњења

Нека је B бипартитни граф који је иницијално придружен ограничењу `alldifferent`$(x_1, \ldots, x_k)$. Након што је задат скуп наметнутих одсецања R,

неке гране ће бити уклоњене из графа. Означимо овај скуп уклоњених грана са E_R, а ново стање графа са B_R. Претпоставимо да је стање графа B_R конзистентно, тј. постоји савршено упаривање $\mathbb{M}$ у графу B_R, и претпоставимо да је Регинов алгоритам открио неку виталну или неконзистентну грану у B_R. Одговарајућа пропагација (придруживање или одсецање, респективно) може бити објашњена касније у било ком тренутку, о чему говори следећа теорема.

Теорема 4.4.9. Нека је дат MOS проблем за резидуални граф $G_{B,\mathbb{M}}$, са скупом препрека $O = E_R$, скупом почетних чворова $S = \{v\}$ и скупом завршних чворова $F = \{u\}$, где је (u, v) усмерена грана која одговара пропагираном одсецању или придруживању. Нека је O_{min} пронађени минимални скуп препрека. Скуп одсецања који одговара скупу грана из O_{min} је једно минимално објашњење дате пропагације.

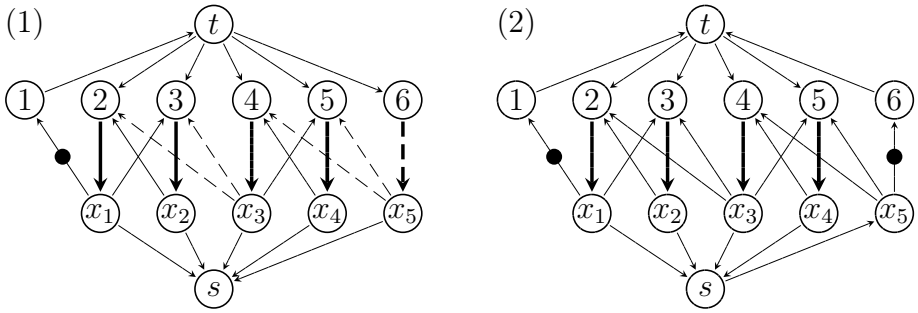
Доказ. Најпре приметимо да су све гране из $\mathbb{M}$ такође присутне и у B. Грана (u, v) је алтернирајућа у графу B, али је витална или неконзистентна у B_R. То значи да су сви усмерени циклуси који садрже (u, v) у графу $G_{B,\mathbb{M}}$ прекинути уклањањем грана из E_R. Дакле, све усмерене путање у графу $G_{B,\mathbb{M}}$ од чвора v ка чвору u морају да садрже бар једну грану из $O = E_R$ — то управо значи да је O скуп препрека који раздваја чвор v од чвора u. Пронађени минимални скуп препрека одговара минималном (у смислу инклузије) објашњењу пропагације — управо та одсецања су довела до тога да грана (u, v) више не буде алтернирајућа у B_R. □

Претпоставимо сада да је B_R у неконзистентном стању, што значи да оптимално упаривање $\mathbb{M}$ није савршено. Објашњавање конфликта се може свести на MOS на сличан начин, о чему говори следећа теорема.

Теорема 4.4.10. Нека је дат MOS проблем у резидуалном графу $G_{B,\mathbb{M}}$ са скупом почетних чворова $S = \{u\}$, где је u било који неупарен чвор из U (који одговара некој променљивој којој није придружена вредност), скупом завршних чворова $F = \{t\}$ и скупом препрека $O = E_R$. Нека је O_{min} пронађени минимални скуп препрека. Скуп одсецања који одговара скупу грана из O_{min} је једно минимално објашњење конфликта.

Доказ. С обзиром на то да је $\mathbb{M}$ оптимално у B_R, не постоји увећавајућа путања у резидуалном графу $G_{B_R,\mathbb{M}}$, тј. све путање од чвора u до чвора t у графу $G_{B,\mathbb{M}}$ садрже бар једну грану из $O = E_R$ — ово управо значи да скуп препрека O

раздваја u од t. Пронађени минимални скуп препрека одговара минималном објашњењу конфликта — управо због тих одсецања чвор u не може бити упарен у графу B_R. □



Слика 4.5: MOS објашњења (црне тачке представљају препреке)

Пример 4.4.5. Размотримо поново исти граф као у примеру 4.4.3. Први граф на слици 4.5 је резидуални граф са препреком постављеном на грани $(x_1, 1)$ (грана уклоњена услед наметнутог одсецања $x_1 \neq 1$). Испрекидане линије одговарају придруживањима и одсецањима која су узрокована уклањањем гране $(x_1, 1)$. Како бисмо пронашли објашњење придруживања $x_5 = 6$ (витална грана $(6, x_5)$) тражимо минимални скуп препрека који раздваја x_5 од чвора 6. У случају објашњавања одсецања $x_5 \neq 4$ (неконзистентна грана $(x_5, 4)$), тражимо скуп препрека који раздваја чвор 4 од чвора x_5. У оба случаја, у питању је препрека $(x_1, 1)$, па је објашњење $x_1 \neq 1$.

Можемо се послужити другим графом на слици 4.5 да илуструјемо објашњавање конфликта. Постављамо препреке на гране $(x_1, 1)$ и $(x_5, 6)$ које одговарају претпоставкама $x_1 \neq 1$ и $x_5 \neq 6$. Да бисмо пронашли објашњење, тражимо минимални скуп препрека који раздваја x_5 од t. У овом случају, минимални скуп садржи обе препреке, па је објашњење $x_1 \neq 1, x_5 \neq 6$.

4.4.8 Поређење MOS и Катсирелосовог алгоритма

Ако упоредимо примере 4.4.3 и 4.4.5, видимо да су Катсирелосова објашњења за придруживање $x_5 = 6$ и одсецање $x_5 \neq 4$ редом $\{x_5 \neq 4, x_5 \neq 5\}$ и $\{x_3 \neq 2, x_3 \neq 3\}$. У случају MOS алгоритма, обе пропагације су објашњене одсецањем $x_1 \neq 1$. Уопште, MOS алгоритам обично проналази објашњења која

су дубље у конфликтном графу у поређењу са објашњењима која се добијају Катсирелосовим алгоритмом. Размотримо поново пропагације пронађене од стране Региновог алгоритма у примеру 4.4.2. Описано у терминима Холових скупова, најпре је грана $(x_1, 1)$ уклоњена, што је довело до тога да скуп $\{x_1, x_2\}$ постане Холов скуп. Због тога су гране $(x_3, 2)$ и $(x_3, 3)$ морале бити одсечене, што је довело до тога да и скуп $\{x_3, x_4\}$ постане Холов. Даље су због тога одсечене гране $(x_5, 4)$ и $(x_5, 5)$, што је довело до тога да грана $(x_5, 6)$ постане витална. Дакле, имамо следећи ланац импликација:

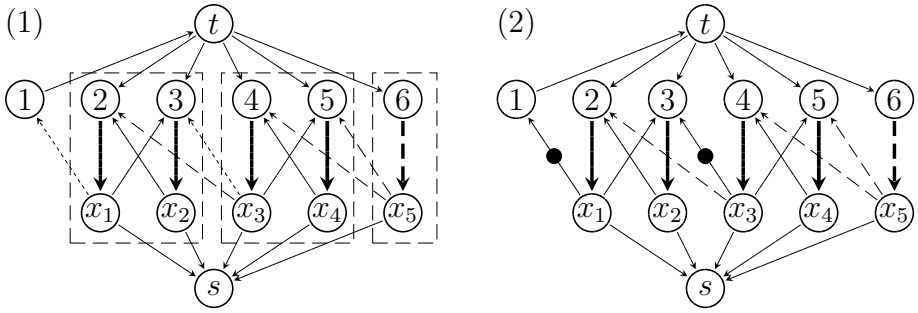
$$x_1 \neq 1 \Rightarrow \{x_3 \neq 2, x_3 \neq 3\} \Rightarrow \{x_5 \neq 4, x_5 \neq 5\} \Rightarrow x_5 = 6$$

У току анализе конфликта, Катсирелосов алгоритам се враћа само један корак уназад у овом ланцу импликација, док MOS алгоритам одлази све до краја ланца. Приметимо да су све три импликације заправо откривене у оквиру тог истог `alldifferent` ограничења у току извршавања Региновог алгоритма које је изазвано наметнутим одсецањем $x_1 \neq 1$. Прави разлог свих ових пропагација је заправо одсецање $x_1 \neq 1$, али та чињеница се не може открити Катсирелосовим алгоритмом. Као што можемо видети из овог примера, једно одсецање може разбити компоненту повезаности на три или више мањих компоненти. Катсирелосов алгоритам се заснива на компонентама повезаности, тако да може пронаћи само *минималне* Холове скупове — оне који директно одговарају компонентама повезаности. Лако се види да је унија дисјунктних Холових скупова такође Холов скуп, али се такви Холови скупови не могу открити Катсирелосовим алгоритмом, јер исти не одговарају директно појединачним компонентама повезаности које Катсирелосов алгоритам открива и користи за објашњења. На пример, ако бисмо пронашли Холов скуп $\{x_1, x_2, x_3, x_4\}$, тада бисмо могли одмах да дамо објашњење $x_1 \neq 1$, али овај скуп не може бити откривен Катсирелосовим алгоритмом. Са друге стране, MOS алгоритам се извршава над иницијалним графом у коме су све гране још увек присутне, а препреке се постављају искључиво на гране које су у каснијим стањима графа уклоњене услед наметнутих одсецања. Гране које ће касније бити уклоњене од стране Региновог алгоритма (као резултат пропагација) не садрже препреке, па због тога MOS алгоритам може да достигне препреке на гранама које су стварни разлог пропагација. Дакле, важно је нагласити да MOS алгоритам даје објашњења искључиво у терминима наметнутих одсецања, а не у терминима одсецања која су пропагирана од стране самог `alldifferent` ограничења, што је случај са Катсирелосовим алгоритмом.

Наравно, ово ограничење Катсирелосовог алгоритма се на први поглед може превазићи у току анализе конфликта тако што ће алгоритам објашњавања бити позван више пута. Ипак, има случајева када овај проблем не може бити превазиђен. Размотримо пример илустрован на слици 4.6. Пример је исти као и раније, с тим што сада имамо два наметнута одсецања $x_1 \neq 1$ и $x_3 \neq 3$ (приметимо да би друго одсецање било последица првог у Региновом алгоритму). У првом графу на слици 4.6, тачкасте гране одговарају наметнутим одсецањима, испрекидане гране одговарају имплицираним одсецањима (пропагацијама), а уоквирени делови графа одговарају Холовим скуповима. Катсирелосов алгоритам објашњава одсецања $x_5 \neq 4$ помоћу скупа $\{x_3 \neq 2, x_3 \neq 3\}$ као и раније. У следећој итерацији анализе конфликта, Катсирелосов алгоритам ће одсецање $x_3 \neq 2$ даље објаснити одсецањем $x_1 \neq 1$. Са друге стране, одсецање $x_3 \neq 3$ неће даље бити објашњавано у оквиру датог `alldifferent` ограничења, с обзиром на то да је оно наметнуто, тј. није пропагирано од стране тог `alldifferent` ограничења. Објашњавање одсецања $x_3 \neq 3$ преузеће неко друго ограничење које га је и пропагирало, што може увести нова одсецања у анализу конфликта. Ова одсецања биће редундантна, зато што се њима објашњава одсецање које није било неопходно за пропагацију одсецања $x_5 \neq 4$. У другом графу на слици 4.6 видимо две препреке које одговарају наметнутим одсецањима. Када објашњавамо одсецање $x_5 \neq 4$, у првој фази MOS алгоритма (полазећи од чвора 4) достижемо обе препреке, али у другој фази ће препрека $(x_3, 3)$ бити елиминисана, с обзиром на то да је чвор 3 блокиран од стране препреке $(x_1, 1)$. Због тога је добијено објашњење $x_1 \neq 1$. Приметимо да се овим избегава објашњавање одсецања $x_3 \neq 3$ од стране другог ограничења које га је пропагирало, с обзиром на то да ово одсецање не доприноси одсецању $x_5 \neq 4$. Овај пример показује да MOS алгоритам може потенцијално дати мања објашњења, тако што елиминише неке редундантне гране у поступку анализе конфликта.

4.4.9 Поређење MOS објашњења са објашњењима заснованим на проточним мрежама

Други занимљив приступ објашњавању пропагација и конфликата код `alldifferent` ограничења је заснован на *проточним мрежама* (енгл. *flow networks*) [39]. Под проточном мрежом подразумевамо усмерени граф $G = (V, E)$ коме је придружена функција $f : E \to \mathbb{R}$ коју зовемо *проток*. Вредност



Слика 4.6: Поређење Катсирелосових и MOS објашњења

$f(u,v)$ зовемо *проток кроз грану* (u,v). Свакој грани (u,v) такође придружујемо *доњу границу* L_{uv} и *горњу границу* U_{uv} такве да је $L_{uv} \leq U_{uv}$. За проток f кажемо да је *допустив* ако важи $L_{uv} \leq f(u,v) \leq U_{uv}$ за сваку грану $(u,v) \in E$, а за сваки чвор $v \in V$ важи следеће својство *одржања протока*:

$$\sum_{(u,v)\in E} f(u,v) = \sum_{(v,w)\in E} f(v,w)$$

Уколико је f допустив проток, тада за одговарајући проток кроз грану $f(u,v)$ кажемо да је *максималан* (*минималан*) ако не постоји други допустиви проток f' такав да је $f'(u,v) > f(u,v)$ $(f'(u,v) < f(u,v))$. За проточну мрежу кажемо да је *редукована* ако за сваку грану $(u,v) \in E$ важи да је $min\{f(u,v) \mid f \text{ је допустив}\} = L_{uv}$ и $max\{f(u,v) \mid f \text{ је допустив}\} = U_{uv}$. Свака проточна мрежа се може редуковати тако што се одреде максимални и минимални протоци кроз сваку од грана (u,v), а затим се границе L_{uv} и U_{uv} поставе на те вредности. За проток f кажемо да је *целобројан* уколико је вредност $f(u,v)$ целобројна за сваку грану $(u,v) \in E$. Може се показати да уколико су све границе L_{uv} и U_{uv} целобројне, тада за сваку грану (u,v) постоји целобројни проток такав да је максималан (минималан) за ту грану. Због тога ћемо у наставку подразумевати искључиво целобројне протоке.

Резидуални граф $G(f)$ за проточну мрежу $G = (V,E)$ и проток f је усмерени граф који има исти скуп чворова V, као и следећи скуп грана: за сваку грану $(u,v) \in E$, ако је $f(u,v) < U_{uv}$, тада постоји грана (u,v) у резидуалном графу са *капацитетом* $c_{uv} = U_{uv} - f(u,v)$; такође, за сваку грану $(u,v) \in E$, ако је $f(u,v) > L_{uv}$, тада постоји грана (v,u) у резидуалном графу са капацитетом $c_{vu} = f(u,v) - L_{uv}$. Усмерени циклус p у резидуалном графу који садржи грану (u,v), али не садржи грану (v,u) се зове *увећавајући циклус* за $f(u,v)$ — његов

капацитет $c(p)$ је једнак минималном капацитету његових грана. Слично, усмерени циклус p у резидуалном графу који садржи грану (v, u), а не садржи грану (u, v) се зове *редукујући циклус* за $f(u, v)$. Интуитивно, увећавајући циклус се може искористити да се увећа проток кроз грану (u, v), док се редукујући циклус може искористити да се смањи проток кроз грану (u, v) у одговарајућој проточној мрежи за износ $c(p)$. Може се показати [81] да је проток кроз грану (u, v) максималан (минималан) ако и само ако не постоји увећавајући (редукујући) циклус за $f(u, v)$ у резидуалном графу $G(f)$.

Приметимо да се индукцијом по броју чворова може показати да својство одржања протока важи за произвољан подскуп чворова S у проточној мрежи, тј. за сваки допустиви проток f сума свих протока кроз гране које улазе у S мора бити једнака суми протока кроз гране које излазе из S. Под *пресеком* за грану (u, v) у проточној мрежи $G = (V, E)$ подразумевамо пар $(S, \overline{S})$, где су S и $\overline{S}$ подскупови од V такви да је $S \cup \overline{S} = V$ и $S \cap \overline{S} = \varnothing$, при чему је $v \in S$ и $u \in \overline{S}$. Под *капацитетом* пресека подразумевамо вредност:

$$\sum_{(w_1,w_2)\in E, w_1\in S, w_2\in\overline{S}} U_{w_1,w_2} - \sum_{(w_1,w_2)\in E\setminus\{(u,v)\}, w_1\in\overline{S}, w_2\in S} L_{w_1,w_2} \tag{4.1}$$

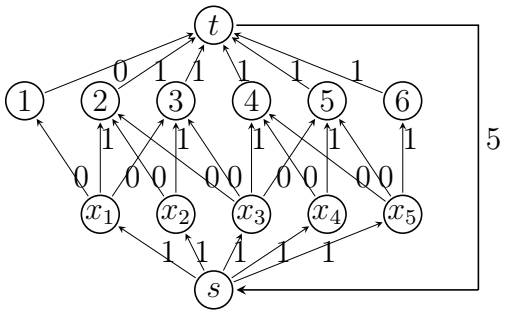
Интуитивно, то је управо максимални проток који *може* ући у S (изаћи из $\overline{S}$) кроз грану (u, v). Од посебног интереса су управо пресеци *минималног капацитета*. Позната *теорема о минималном пресеку* [39] нам говори да је максимални допустиви проток кроз грану (u, v) једнак минималном капацитету који може имати неки пресек $(S, \overline{S})$ за грану (u, v). Аналогно можемо дефинисати *праг* пресека:

$$\sum_{(w_1,w_2)\in E, w_1\in S, w_2\in\overline{S}} L_{w_1,w_2} - \sum_{(w_1,w_2)\in E\setminus\{(u,v)\}, w_1\in\overline{S}, w_2\in S} U_{w_1,w_2} \tag{4.2}$$

Ово је управо минимални проток који *мора* ући у S (изаћи из $\overline{S}$) кроз грану (u, v). Сада је минимални допустиви проток кроз грану (u, v) једнак максималном прагу који може имати неки пресек $(S, \overline{S})$ за грану (u, v).

Ограничењу `alldifferent` из претходних примера можемо придружити проточну мрежу дату на слици 4.7. Све гране изузев (t, s) имају доњу границу 0 и горњу границу 1. Проток 1 за грану (x, d) значи да је ова грана део текућег упаривања, а проток 0 значи да није. Грана (t, s) има и горњу и доњу границу једнаку 5, чиме се гарантује да допустиви протоци одговарају савршеним упаривањима. Сада се највећи део претходне приче о `alldifferent` ограничењу

може реформулисати у терминима проточних мрежа. Провера конзистентности ограничења је еквивалентна проналажењу допустивог протока у одговарајућој проточној мрежи. Проналажење виталних и неконзистентних грана је еквивалентно проналажењу максималних и минималних протока у одговарајућим гранама у проточној мрежи. Грана (x, d) ће бити неконзистентна ако и само ако је максимални проток кроз ту грану једнак 0. Слично, грана (x, d) ће бити витална ако и само ако је минимални проток кроз ту грану једнак 1. Одавде следи да се проблем објашњавања неконзистентности и пропагација у `alldifferent` ограничењу може свести на објашњавање максималних и минималних протока, а које се заснива на проналажењу пресека минималног капацитета (максималног прага) у проточној мрежи за одговарајућу грану. Када се пронађе такав пресек, ограничење протока за грану (x, d) се може објаснити границама протока које су наметнуте попречним гранама тог пресека (тј. границама L_{w_1,w_2} и U_{w_1,w_2} у изразима (4.1) и (4.2)) [84, 36]. Оно што је овде значајно нагласити је да такви пресеци *нису јединствени* у општем случају, при чему неки од њих дају боља објашњења, а неки лошија. У овом одељку укратко дискутујемо различите приступе засноване на проточним мрежама и поредимо их са нашим приступом заснованим на MOS проблему.



Слика 4.7: Проточна мрежа за `alldifferent` ограничење из претходних примера. Горња и доња граница протока у свим гранама су 0 и 1, респективно, осим за грану (t, s), код које су и горња и доња граница протока једнаке 5 (овим гарантујемо да ће све променљиве бити упарене). Допустиви протоци у овој мрежи одговарају савршеним упаривањима у бипартитном графу. Пример допустивог протока је дат бројевима поред грана на слици.

Први приступ је заснован на откривању компоненти јаких повезаности у резидуалном графу[6]. Овај приступ је примењен у раду Даунинга и других [36], и

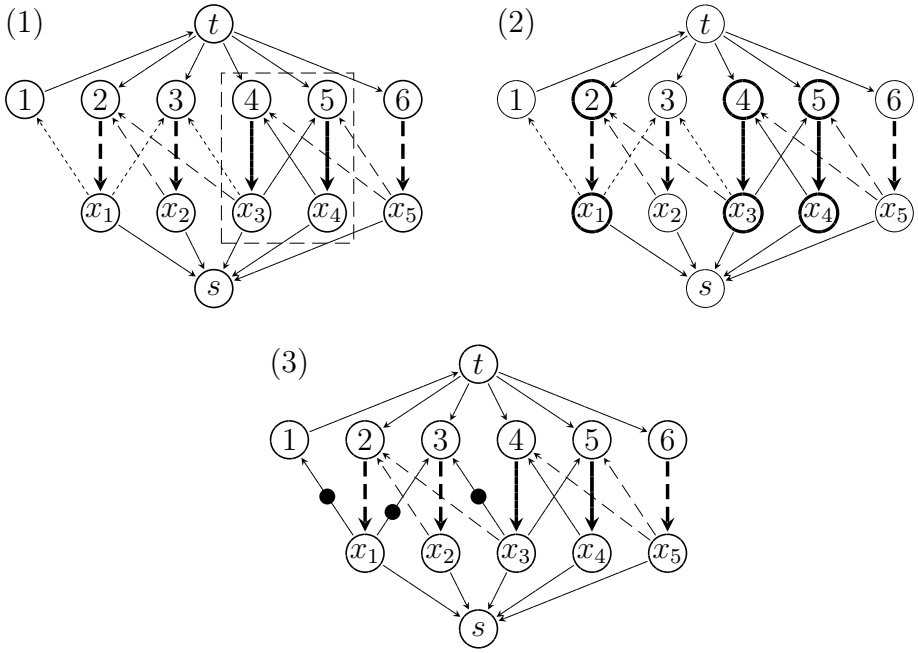
[6]Приметимо да је наша дефиниција резидуалног графа који је придружен `alldifferent`

заснива се на чињеници да су гране које спајају различите компоненте повезаности у резидуалном графу управо оне гране које имају особину да је вредност протока кроз такву грану у сваком допустивом протоку кроз мрежу увек једнак горњој или доњој граници протока за ту грану (доказ овог тврђења се може наћи у раду Регина [81]). У случају `alldifferent` ограничења, то су управо виталне и неконзистентне гране. Након што се проточна мрежа редукује, тј. границе протока се ажурирају на основу израчунатих максималних и минималних вредности (што одговара уклањању неконзистентних грана и означавању виталних грана у бипартитном графу), компонента повезаности S која садржи чвор d ће одређивати минимални пресек $(S, \overline{S})$ за грану (x, d) и може се искористити за објашњавање граница протока за ову грану. У случају `alldifferent` ограничења, Катсирелосов алгоритам ради управо то. У раду Даунинга и других [36] овај приступ је генерализован на произвољне проточне мреже, али аутори признају да њихов приступ, примењен на `alldifferent` ограничење даје иста објашњења као и Катсирелосов алгоритам [36, 35]. Већ смо раније дискутовали у каквом су односу таква објашњења према објашњењима која даје наш MOS алгоритам.

Други приступ је заснован на откривању достижних чворова у резидуалном графу [84]. Ако је дата грана (x, d) чији се максимални/минимални проток објашњава, обиласком резидуалног графа полазећи од чвора d означавамо све чворове достижне из d. Овако добијени скуп достижних чворова S такође одређује минимални пресек $(S, \overline{S})$ [84] који се може искористити за објашњавање граница протока кроз грану (x, d). Да бисмо упоредили тако добијена објашњења са нашим MOS објашњењима, сетимо се да MOS алгоритам објашњен у одељку 4.4.6 има две фазе. У првој фази се откривају достижне препреке (тј. означавају се чворови достижни из d и проналазе се одсечене гране које су раније повезивале ове достижне чворове са осталим чворовима графа). У другој фази, алгоритам елиминише редунданте препреке чији су завршни чворови блокирани осталим достижним препрекама (тј. које одговарају одсеченим гранама које и тако не могу бити део неке увећавајуће путање, с обзиром на то да се из њихових завршних чворова не може достићи чвор x). Очигледно, приступ описан у раду Рохарта и других [84] је еквивалентан само првој фази нашег MOS алгоритма.

ограничењу наведена у одељку 4.4.3 прилично упрошћена, у поређењу са уобичајеном дефиницијом резидуалног графа у проточној мрежи коју смо навели у овом одељку, а која је преузета из литературе [81]. Ипак, две дефиниције су потпуно еквивалентне у случају графа који је придружен `alldifferent` ограничењу.

Ово значи да добијена објашњења могу садржати сувишне гране, као што и аутори сами напомињу [84]. У ствари, објашњења добијена MOS алгоритмом управо одговарају објашњењима која су у раду Рохарта и других [84] описана као пожељна објашњења — она садрже искључиво одсечене гране које повезују чворове достижне из d са чворовима из којих се може достићи чвор x (напоменимо да у раду Рохарта и других [84] не постоји опис алгоритма који би генерисао описана пожељна објашњења, већ само горе описани алгоритам који одговара првој фази MOS алгоритма).



Слика 4.8: Поређење објашњења заснована на проточним мрежама и MOS објашњења

Пример 4.4.6. Размотримо поново исти резидуални граф из претходних примера (слика 4.8). Претпоставимо да су задата три одсецања: $x_1 \neq 1$, $x_1 \neq 3$ и $x_3 \neq 3$. Преведено на језик протока кроз мрежу, ова три одсецања значе да су задате горње границе протока за одговарајуће гране једнаке 0 (тачкасте гране у графовима на слици). Алгоритам филтрирања израчунава минималне и максималне протоке кроз гране и пропагира одговарајућа одсецања за оне гране код којих је израчунати максимални проток једнак 0, као и одговарајућа придруживања за оне гране код којих је израчунати минимални проток једнак 1 (испрекидане гране у графовима на слици). Претпоставимо да желимо да објаснимо одсецање $x_5 \neq 5$. У случају алгоритма који је заснован

на откривању компоненти повезаности, најпре ће бити откривена компонента повезаности која садржи чвор 5 (уоквирени део првог графа на слици 4.8) а затим ће (након редукције проточне мреже) та компонента бити искоришћена за одређивање пресека. Долазни ток у ту компоненту је ограничен одозго са 0, зато што је њен одлазни ток ограничен одозго са 0, услед граница за протоке кроз гране $(x_3, 2)$ и $(x_3, 3)$. Отуда је објашњење $x_3 \neq 2$, $x_3 \neq 3$ (исто као и Катсирелосово објашњење). Други начин да објаснимо ово одсецање је да пронађемо скуп достижних чворова из чвора 5, не користећи гране означене тачкасто на слици (то су чворови који су подебљани у другом графу на слици 4.8). Овај скуп чворова одређује пресек. Његов долазни ток је такође ограничен одозго са 0, а разлог су поново задате границе протока у његовим излазним гранама — објашњење је, дакле, управо $x_3 \neq 3$, $x_1 \neq 1$, $x_1 \neq 3$. Најзад, трећи начин да објаснимо ово одсецање је да користимо MOS алгоритам (трећи граф на слици 4.8). Скуп достижних препрека се састоји од све три препреке (као и у претходном приступу), али наш алгоритам препознаје да је само грана $(x_1, 1)$ заиста неопходна да прекине све путање које воде ка чвору x_5, па је објашњење $x_1 \neq 1$.

4.4.10 Имплементациони детаљи

Руковалац за ограничење `alldifferent`$(x_1, \ldots, x_k)$ имплементира бипартитни граф.[7] Приликом успостављања новог нивоа одлучивања, стање графа се чува, тако да се може ефикасно реконструисати када се примени правило `Backjump`. Када се литерали рестрикција домена наметну од стране SAT решавача, одговарајуће гране графа се бришу. За сваку уклоњену грану e, памти се литерал који је одговоран за њено уклањање (означимо га са $cause(e)$). Ова информација се касније користи за објашњавање.

Руковалац подржава два дедуктивна слоја. У првом слоју, откривају се само једноставни конфликти попут $x_i = d, x_j = d \vDash \bot$, као и тривијалне пропагације попут $x_i = d \vDash x_j \neq d$ (за $i \neq j$). Ово значи да први дедуктивни слој остварује исти ниво конзистентности ограничења као и у случају декомпози-

[7] У овој фази развоја `argosmt` решавача, руковалац `alldifferent` ограничењем још увек не укључује процедуре које се тичу супротног ограничења ¬`alldifferent`$(x_1, \ldots, x_k)$. Због тога се ограничење `alldifferent` не сме јављати на трагу M са негативним поларитетом. Ово не представља значајно ограничење, с обзиром на то да се у формулама које одговарају типичним CSP проблемима `alldifferent` ограничења јављају искључиво као јединичне клаузе.

ције `alldifferent` ограничења на скуп различитости $x_i \neq x_j$, где је $1 \leq i, j \leq k$ и $i \neq j$.

У другом дедуктивном слоју се најпре позива Форд-Фулкерсонов алгоритам да се провери да ли постоји конфликт. Ако постоји, позива се процедура за објашњавање конфликта и примењује се правило $\texttt{TheoryConflict}_i$. У супротном, позива се Регинов алгоритам који открива и пропагира једнакости (које одговарају новопронађеним виталним гранама) и различитости (које одговарају одсеченим неконзистентним гранама графа).

Имплементирани су и Катсирелосов и MOS-заснован алгоритам за генерисање објашњења, како бисмо могли упоредити понашање два алгоритма у оквиру истог решавача. С обзиром на то да оба алгоритма изражавају објашњења као скупове одсецања (као што је описано у одељку 4.4.5), добијено објашњење се мора трансформисати у скуп литерала и то тако што се свака одсечена грана e у објашњењу замени литералом $cause(e)$. Приметимо да тако добијено објашњење може садржати и неједнакости, с обзиром на то да неједнакост такође може бити узрок уклањања гране.

4.4.11 Експериментална евалуација

У овом одељку приказујемо експерименталну евалуацију нашег `argosmt` решавача који је описан у претходним одељцима. Главни циљ евалуације је да се упореде перформансе Катсирелосовог и MOS-заснованог алгоритма за генерисање објашњења у оквиру нашег решавача. Други циљ је да се упореди наш решавач са неким од најпознатијих савремених SMT и CSP решавача. Како бисмо постигли ове циљеве, експериментисано је са следећим решавачима:

- `argosmt-mos` — наш решавач[8] који користи MOS објашњења
- `argosmt-kat` — наш решавач који користи Катсирелосова објашњења
- `sugar-argosmt` — *Sugar* SAT-заснован CSP решавач [98] који користи `argosmt` као позадински SAT решавач
- `sugar-minisat` — *Sugar* SAT-заснован CSP решавач који користи `minisat`[9] као позадински SAT решавач
- `minion` — Minion CSP решавач [43]

[8] `http://www.matf.bg.ac.rs/~milan/argosmt/`
[9] `http://minisat.se/`

- `g12lazy` — G12[10] CSP решавач заснован на лењом генерисању клауза [77]
- `opturion` — Opturion[11] CSP решавач заснован на лењом генерисању клауза
- `yices-lia` — Yices SMT решавач [38], при чему се проблеми кодирају у `QF_LIA` теорији, користећи `distinct` предикат за представљање `alldifferent` ограничења
- `yices-bv` — Yices SMT решавач, при чему се користи `QF_BV` теорија (битвектори фиксиране дужине)

Наведене решаваче тестирали смо на следећих шест скупова инстанци:[12]

судоку25 — скуп се састоји од 200 случајно генерисаних инстанци *судоку* [61] проблема. *Судоку* проблем је задат таблицом димензије $n^2 \times n^2$ ($n \in \mathbb{N}$) чија поља треба попунити бројевима од 1 до n^2 тако да су вредности у свакој врсти, свакој колони и сваком од n^2 квадрата димензије $n \times n$ међусобно различите. Притом, неке вредности у матрици су обично унапред задате. Овај проблем се једноставно кодира као CSP проблем, тако што се уведе по једна променљива x_{ij} са доменом $\{1, \ldots, n^2\}$ за свако поље (i, j) матрице, а затим се задају `alldifferent` ограничења — по једно за сваку врсту, сваку колону и сваки квадрат. Унапред задате вредности се кодирају једнакостима. Све инстанце овог скупа су величине 25×25. Користили смо временско ограничење од 200 секунди по инстанци. Таблице су генерисане са око 45% унапред задатих вредности — судоку проблем има фазну транзицију и управо овако попуњене таблице би требало да су најтеже за решавање [61]. Све инстанце су задовољиве.

судоку36 — слично као и у претходном случају, овај скуп се састоји од 100 случајно генерисаних судоку инстанци, али веће димензије — 36×36 (овог пута, гранично време по инстанци је 600 секунди).

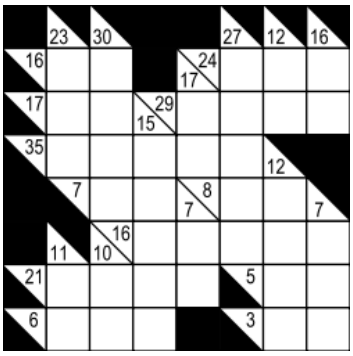
какуро — скуп се састоји од 100 случајно генерисаних инстанци *какуро* проблема [90]. Проблем *какуро* (слика 4.9) се састоји из таблице која садржи празна (бела) поља, као и поља која нису празна, а која могу бити или обојена у црно или попуњена бројевима (ова поља ћемо звати *преградна поља*). Свако празно поље треба попунити бројем од 1 до 9, тако да су у свакој од *линија* (вертикалној

[10] `http://www.minizinc.org/g12distrib.html`

[11] `http://www.opturion.com/`

[12] Све инстанце се могу наћи на локацији: `http://www.math.rs/~milan/argosmt/instances.zip`

или хоризонталној секвенци суседних празних поља) сви бројеви међусобно различити, а њихова сума је једнака броју који је дат у преградном пољу које је суседно тој линији. Проблем се може представити као CSP проблем, где је сваком празном пољу придружена променљива са доменом $\{1, \ldots, 9\}$, при чему су променљиве из сваке од линија ограничене `alldifferent` ограничењем, као и једним линеарним ограничењем које захтева да је сума променљивих једнака датом броју. Генерисане инстанце су димензије 20×20 и све су задовољиве. Користили смо гранично време од 600 секунди по инстанци.



Слика 4.9: Пример какуро проблема

голфери — овај скуп се састоји од 65 инстанци проблема *друштвених голфера* (енгл. *social golfers*) [44] различитих величина. Проблем се може описати на следећи начин: $m \cdot n$ играча играју голф и у свакој рунди се деле на m екипа од по n играча. Циљ је направити распоред по екипама за p рунди тако да никоја два играча не играју више од једном заједно у истој екипи. Користили смо гранично време од 1200 секунди за сваку од инстанци у скупу, иако тежина ових инстанци варира од веома лаких до екстремно тешких, у зависности од величине. Кодирање је засновано на сличном моделу као и у раду Гента и Линса [44]. Играче ћемо означити бројевима $0, 1, \ldots, mn - 1$. Променљива x_{ijk} има домен $\{0, 1, \ldots, mn - 1\}$, а њена вредност одређује j-тог играча у i-тој групи у k-тој рунди. За свако фиксирано $k \in \{1, \ldots, p\}$ имамо једно `alldifferent` ограничење над променљивама x_{ijk}, где је $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$ — ово ограничење захтева да у свакој од рунди ни један од играча не може бити на две различите позиције у исто време. Да бисмо изразили захтев да никоја два играча не играју више од једном у истој екипи, успостављамо „1-1“ кореспонденцију између парова играча и бројева из скупа $\{0, \ldots, m^2n^2 - 1\}$:

$(u, v) \mapsto nm \cdot u + v$. Сада уводимо по једну променљиву $y_{ij_1j_2k}$ за сваки пар променљивих (x_{ij_1k}, x_{ij_2k}), где је $j_1 < j_2$. Домен свих ових променљивих је $\{0, \ldots, m^2n^2 - 1\}$. Уводимо линеарно ограничење $y_{ij_1j_2k} = nm \cdot x_{ij_1k} + x_{ij_2k}$ за сваку од „y" променљивих. Најзад, имамо једно додатно `alldifferent` ограничење над свим „y" променљивама — оно управо каже да никоја два пара играча не могу играти више од једном у истој екипи. Овакво кодирање је углавном засновано на `alldifferent` ограничењу, али исто тако укључује и доста линеарних ограничења над променљивама са великим доменима.

распоред часова — овај скуп садржи инстанце проблема који има практичнију примену: скуп се састоји од 100 случајно генерисаних инстанци проблема распореда часова: разматрамо средњу школу са 40 одељења, где свако одељење има 35 часова недељно, распоређених у пет радних дана (седам часова дневно). Сваки час заузима један од доступних временских термина једнаке дужине (сваког дана имамо тачно седам оваквих термина). Такође се захтева да неки од часова морају бити распоређени у блоковима (величине 2 или 3, тј. двочаси или трочаси) — часови који чине блок морају бити распоређени истог дана и морају да заузимају узастопне временске термине. У свакој од наших инстанци имамо по пет трочаса и по девет двочаса, док се преостала два часа могу распоредити произвољно. Часове предаје известан број наставника, при чему сваки од наставника има од 14 до 20 часова недељно. Случајност приликом генерисања инстанци је обезбеђена тако што су часови додељивани наставницима на случајан начин (при чему часови који чине блок морају бити додељени истом наставнику). Недељни број часова за сваког од наставника је такође случајно изабран из интервала $[14, 20]$. Да бисмо кодирали проблем, уводимо променљиву x_{ijk} за сваког наставника i који предаје k-ти час групи j (на основу поделе часова наставницима која је унапред позната за сваку од инстанци). Овако уведене променљиве имају домен $\{0, \ldots, 34\}$ — вредност променљиве x_{ijk} одређује термин у коме се одговарајући час држи (вредности $0, \ldots, 6$ су термини првог дана, $7, \ldots, 13$ су термини другог дана и тд.). Да бисмо изразили захтев да i-ти наставник не може да држи два часа истовремено, задајемо `alldifferent` ограничење над свим променљивама x_{ijk} за фиксирано i. Слично, да бисмо изразили захтев да j-то одељење не може похађати два различита часа истовремено, задајемо `alldifferent` ограничење над променљивама x_{ijk} за фиксирано j. Најзад, да бисмо изразили услове о блоковима, захтевамо да за сваке две променљиве x_{ijk_1} и x_{ijk_2} које представљају два суседна часа у неком блоку

мора да важи да је $x_{ijk_1} + 1 = x_{ijk_2}$. Такође, за сваки од блокова сви часови осим последњег у блоку не смеју бити распоређени у последњем термину неког дана (односно, одговарајућа променљива не може узети ни једну од следећих вредности: 6, 13, 20, 27, 34).

KTK — скуп се састоји од 100 случајно генерисаних инстанци проблема *комплетирања тежинских квазигрупа*[13] (енгл. *weighted quasigroup completion problem*) [88]. Проблем је сличан стандардном проблему комплетирања квазигрупа, али додатно се сваком пољу (i, j) датог *латинског квадрата* придружује *тежина* p_{ij} — позитиван цео број из неког предефинисаног интервала. Циљ је комплетирати квазигрупу полазећи од задатих вредности у таблици, при чему треба минимизовати вредност $M = min_i(\sum_j p_{ij}x_{ij})$, где су x_{ij} променљиве које одговарају вредностима које се уписују у поља латинског квадрата. Наравно, ово је проблем оптимизације, али ми ћемо разматрати одговарајући проблем одлучивања — да ли постоји исправна квазигрупа таква да је $M \leq K$, где је K позитиван цео број? Проблем се кодира помоћу `alldifferent` ограничења над променљивама x_{ij} (по једно `alldifferent` ограничење за сваку врсту и сваку колону). Такође, за сваку врсту уводимо променљиву $y_i = \sum_j p_{ij}x_{ij}$ која представља одговарајућу тежину те врсте, а затим задајемо клаузу $\bigvee_i(y_i \leq K)$ — ова клауза обезбеђује да је најмања од y_i променљивих (тј. минимум тежина по врстама) највише K. Све инстанце у скупу су величине 30×30, тежине су између 1 и 100, а таблице су генерисане са око 42% унапред попуњених поља (ово је управо тачка фазне транзиције за овај проблем [46]). Број K смо бирали на основу неког претходно пронађеног комплетирања квазигрупе, тако да знамо да су све инстанце задовољиве.

Резултати у табели 4.1 показују да се у просеку наш решавач боље понаша када користи MOS алгоритам за објашњења него када у исту сврху користи Катсирелосов алгоритам. Највеће убрзање се постиже за *судоку* инстанце, али је такође приметно и за *KTK* инстанце и инстанце *распореда*. На другој страни, нема значајне разлике у перформансама на *какуро* инстанцама и *голферима*. С обзиром на то да просечно време само за себе не даје довољно информација о дистрибуцији времена извршавања, на слици 4.10 приказујемо како се времена извршавања односе на појединачним инстанцама. Тачке изнад дијагонале

[13]Судећи по ономе што је речено у раду Селмана и Кадиоглуа [88], овај проблем, као типичан представник *проблема комбинаторног дизајна*, има структурна својства која се веома често виђају у многим практичним применама.

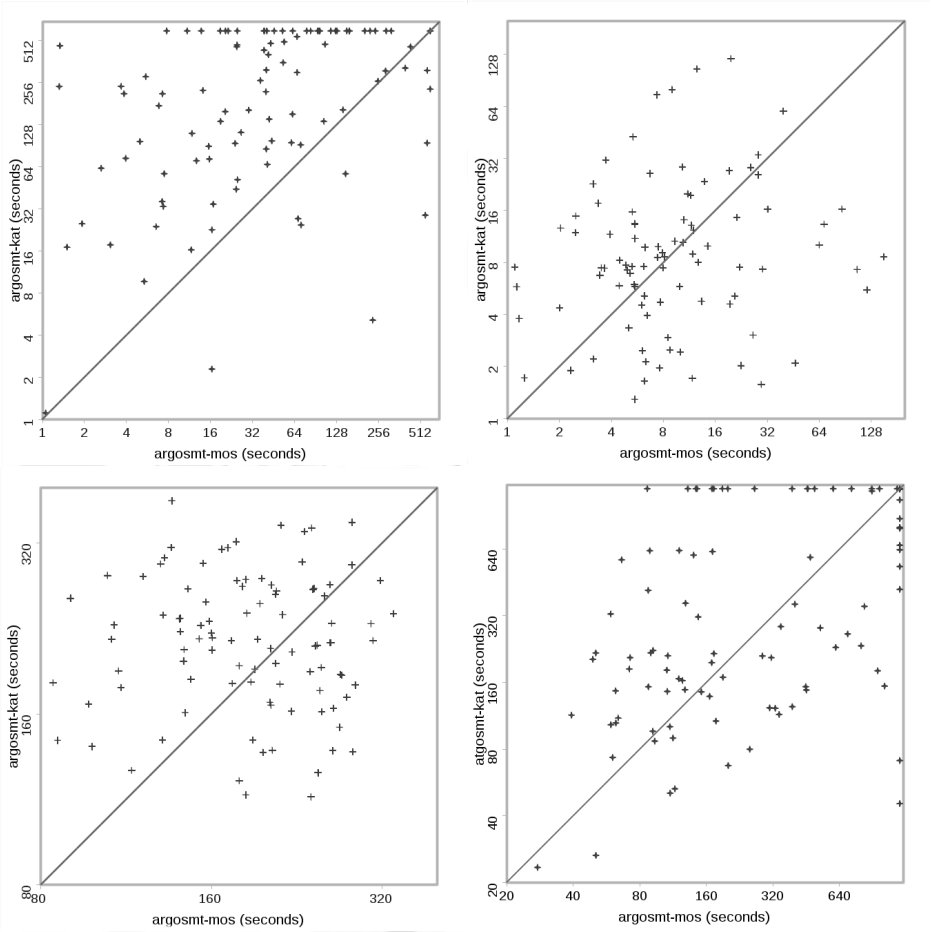
	судоку25		судоку36		какуро	
	# инстанци	гранично	# инстанци	гранично	# инстанци	гранично
	200	300s	100	600s	100	600s
	# решених	просек	# решених	просек	# решених	просек
argosmt-mos	200	1.03s	97	99.7s	100	15.7s
argosmt-kat	200	2.46s	69	307s	99	24.7s
sugar-argosmt	188	72.5s	0	600s	100	10.8s
sugar-minisat	200	13.3s	7	456s	100	2.03s
minion	186	35.9s	23	476s	16	520s
g12-lazy	198	21.8s	3	587s	98	20.7s
opturion	198	39.2s	0	600s	100	4.9s
yices-lia	49	277s	0	600s	12	566s
yices-bv	0	300s	0	600s	100	32.4s

	голфери		распоред		KTK	
	# инстанци	гранично	# инстанци	гранично	# инстанци	гранично
	65	1200s	100	1200s	100	1200s
	# решених	просек	# решених	просек	# решених	просек
argosmt-mos	44	463s	99	187s	82	444s
argosmt-kat	45	458s	99	209s	75	516s
sugar-argosmt	24	780s	12	1160s	0	1200s
sugar-minisat	30	660s	100	103s	0	1200s
minion	22	817s	0	1200s	14	1033s
g12-lazy	51	305s	19	1084s	13	1073s
opturion	43	439s	0	1200s	100	43s
yices-lia	1	1193s	0	1200s	0	1200s
yices-bv	46	492s	0	1200s	0	1200s

Табела 4.1: У табели су за сваки решавач и сваки скуп инстанци дати број инстанци решених у датом граничном времену, као и просечно време решавања (за нерешене инстанце користи се гранично време при рачунању просека)

представљају инстанце за које се MOS алгоритам показао боље од Катсирелосовог алгоритма. График делује веома убедљиво за *судоку* инстанце (график горе лево), где видимо да је већина тачака значајно изнад дијагонале. Доња два графика показују односе времена извршавања за инстанце *распореда* (лево) и *KTK* инстанце (десно) — иако не тако драстично овог пута, графици и даље пресуђују у корист MOS-заснованог алгоритма. Најзад, за *какуро* инстанце (график горе десно), тачке су готово равномерно разбацане са обе стране дијагонале, тако да не можемо закључити који је алгоритам бољи. Информације о редукцији простора претраге (табела 4.2) такође потврђују да се управо за *судоку* инстанце добија највеће побољшање при коришћењу MOS алгоритма.

Различита убрзања добијена коришћењем MOS алгоритма уместо Катсирелосовог алгоритма на различитим скуповима инстанци могу се угрубо објаснити бројем и дужинама `alldifferent` ограничења која се појављују у инстанцама. Као што је дискутовано у одељку 4.4.8, главна корист од коришћења MOS алгоритма је у скраћивању ланаца импликација, а то ће се вероватније дешавати



Слика 4.10: `argosmt-mos` наспрам `argosmt-kat` времена решавања: *судоку36* (горе лево), *какуро* (горе десно), *распоред* (доле лево), *KTK* (доле десно)

	судоку25		**судоку36**		**какуро**	
	# одлуч.	**# конфл.**	**# одлуч.**	**# конфл.**	**# одлуч.**	**# конфл.**
argosmt-mos	388	162	13059	6953	44761	31690
argosmt-kat	705	298	22440	9503	44491	31915
opturion	240008	149023	—	—	130489	74390

	голфери		**распоред**		**KTK**	
	# одлуч.	**# конфл.**	**# одлуч.**	**# конфл.**	**# одлуч.**	**# конфл.**
argosmt-mos	66192	1877	188905	5833	27272	20364
argosmt-kat	58742	1740	220613	5051	28538	21200
opturion	187985	99113	—	—	112373	52255

Табела 4.2: Просечан број одлучивања и конфликата (за решене инстанце)

у дужим `alldifferent` ограничењима (тј. ограничењима са већим арностима). Као што се види у табели 4.3, *судоку*, *распоред* и *KTK* инстанце имају и велики број `alldifferent` ограничења, али и велику просечну дужину ових ограни-

чења (што даје просечну кумулативну дужину по инстанци већу од 1500). У случају *какуро* инстанци, број `alldifferent` ограничења је 124 у просеку (што је упоредиво са *судоку* и инстанцама *распореда*), али је просечна дужина ограничења само 4.3. Са друге стране, дужина `alldifferent` ограничења за инстанце *голфера* је 52 у просеку, али тако велики просек је услед једног великог `alldifferent` ограничења у свакој инстанци (оно над „*y*" променљивама), док су сва остала ограничења значајно краћа. Просечан број `alldifferent` ограничења по инстанци је само 10, па је кумулативна дужина по инстанци само 520 у просеку, што је значајно мање него код *судоку* инстанци, а знатно упоредивије са *какуро* инстанцама. Друга занимљива информација која се види у табели 4.3, а која може бити корисна у анализи ефективности MOS алгоритма на различитим скуповима инстанци је удео пропагација које долазе из Региновог алгоритма: у случају *судоку25* инстанци, око 34% свих пропагација долазе из Региновог алгоритма, док су око 48% свих објашњења управо објашњења Регинових пропагација. За *судоку36*, имамо око 39% Регинових пропагација у просеку и 47% Регинових објашњења. Због тога избор алгоритма за генерисање објашњења Регинових пропагација може имати велики утицај на перформансе решавача на овим инстанцама. Са друге стране, за *какуро* инстанце, мање од 1% свих пропагација потичу из Региновог алгоритма, а мање од 2% свих објашњења се тичу ових пропагација. С обзиром на то да је проценат објашњења Регинових пропагација веома мали, избор алгоритма за генерисање објашњења не може значајно утицати на ефикасност решавача.

	судоку25	**судоку36**	**какуро**	**голфери**	**распоред**	**КТК**
# alldiff-ова	75	108	124	10	126	60
прос. дужина	25	36	4.3	52	22.2	30
кум. дужина	1875	3888	546	520	2800	1800
% Регинових проп.	34%	39%	0.8%	2%	6.4%	0.6%
% Регинових обј.	48%	47%	1.7%	5.7%	9.5%	7.7%

Табела 4.3: Табела показује просечан број `alldifferent` ограничења по инстанци, просечну дужину `alldifferent` ограничења, као и просечну кумулативну дужину свих `alldifferent` ограничења по инстанци у сваком од скупова. Такође показује удео пропагација које долазе из Региновог алгоритма, као и удео објашњавања таквих пропагација у укупном броју свих објашњавања у решавачу

У поређењу са осталим решавачима, може се видети (табела 4.1) да је наш решавач најбољи избор за *судоку* инстанце, док је на осталим проблемима упо-

редив са осталим решавачима (обично други најбољи избор). Штавише, ако сумирамо резултате на свим скуповима инстанци, `argosmt-mos` је укупно решио 622 инстанце, док је `argosmt-kat` укупно решио 587 инстанце. У тим терминима, први следећи најбољи решавач је `opturion` који је решио само 441 инстанцу, док је `sugar-minisat` решио 437 инстанци. Сви остали решавачи су решили значајно мање инстанци. Такође је веома важно нагласити да док су перформансе осталих решавача веома добре на неким скуповима инстанци, а веома лоше на другим, наш решавач је униформно добар на свим скуповима инстанци. Ово значи да је наш решавач прилично поуздан алат за решавање CSP проблема.

Још један занимљив закључак се може извући из поређења понашања `sugar` решавача када се користе `argosmt`, односно `minisat` као позадински SAT решавачи. У табели 4.1 можемо видети да је `minisat` неколико пута бржи на неким скуповима инстанци од нашег SAT решавача. Ово показује да имплементација SAT алгоритма који покреће наш решавач `argosmt` још увек није потпуно упоредива са савременим SAT решавачима (на којима су засновани неки од решавача који су коришћени у експериментима). Повећавање ефикасности SAT решавача може додатно убрзати наш решавач и учинити га још више упоредивим са другим алатима. Додатни податак који може да потврди ову хипотезу је информација о величини истраженог простора претраге која је дата у табели 4.2: решавач `opturion` има значајно већи број одлучивања и конфликата на свим скуповима инстанци, иако на неким скуповима инстанци даје боље резултате од нашег решавача. То може да значи да је имплементација значајно бржа, док је величина простора претраге знатно већа него код нашег решавача (ипак, нисмо сигурни да ли су дати бројеви упоредиви, с обзиром на то да структура простора претраге и природа литерала који се користе у `opturion` решавачу може бити значајно другачија).

Такође видимо да `yices` решавач, као један од најбољих савремених стандардних SMT решавача није добар избор за решавање CSP проблема. При коришћењу `QF_LIA` теорије, решавач показује веома лоше понашање на свим скуповима инстанци. Ово је донекле и очекивано, с обзиром на то да се ова теорија углавном користи за сасвим другачије проблеме у SMT-у, где обично имамо бесконачне или веома велике домене. Када се користи теорија `QF_BF`, решавач се понаша значајно боље на *какуро* инстанцама, као и на инстанцама *голфера*, али је и даље неупотребљив на осталим скуповима инстанци. Ово по-

тврђује наша очекивања да се уградњом алгоритама филтрирања за глобална ограничења може драматично побољшати ефикасност SMT технологије, када је решавање CSP проблема у питању.

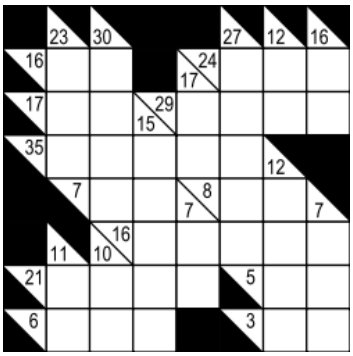
4.5 Руковалац линеарним ограничењем

У овом поглављу разматрамо руковалац *линеарним ограничењем*, тј. ограничењем облика $a_1 \cdot x_1 + \ldots + a_n \cdot x_n \bowtie c$, где су a_i и c целобројни коефицијенти, x_i су CSP промењљиве са коначним доменима, а $\bowtie \in \{\leq, \geq\}$[14]. Алгоритам филтрирања који се уобичајено користи у CSP решавачима за линеарна ограничења је дат у раду Шултеа и Стакија [87]. Овај алгоритам успоставља конзистентност граница над линеарним ограничењем. Његова основна идеја је да се нове границе неке промењљиве у изразу израчунају на основу текућих граница осталих промењљивих, а да се затим одсеку вредности из домена промењљиве које су изван тих новоизрачунатих граница. Наш руковалац линеарним ограничењем такође имплементира овај једноставни алгоритам. Поред тога, овај руковалац имплементира и побољшани алгоритам филтрирања који је оригинални допринос овог дела тезе [6], а који узима у обзир постојање `alldifferent` ограничења чији се скупови промењљивих делимично или потпуно преклапају са скупом промењљивих линеарног ограничења. Овакав приступ често омогућава израчунавање јачих граница за промењљиве, што доводи до додатног сужавања простора претраге и брже конвергенције ка решењу. Размотримо следећи пример.

Пример 4.5.1. На слици 4.11 дат је пример раније описаног *какуро* проблема (одељак 4.4.11). Све хоризонталне и вертикалне линије (секвенце узастопних празних поља) ограничене су `alldifferent` ограничењима, као и линеарним ограничењима која захтевају да сума вредности у одговарајућој линији буде једнака датом суседном броју у таблици.

Размотримо, на пример, прву хоризонталну линију у врсти на дну таблице — она се састоји из три поља чије вредности треба да у збиру буду 6. Претпоставимо да су промењљиве придружене тим пољима редом x, y и z. Над овим промењљивама имамо ограничења `alldifferent`(x, y, z) и $x + y + z = 6$. Ако би алгоритам филтрирања за линеарно ограничење био потпуно несвестан посто-

[14] Релацијски симболи $=$, $<$, $>$, $\neq$ су такође допуштени, али се они могу свести на $\leq$ и $\geq$, као што је описано у одељку 2.2.3.



Слика 4.11: Пример какуро проблема

јања `alldifferent` ограничења, он би закључио да вредности променљивих x, y and z морају припадати интервалу $[1, 4]$, с обзиром на то да би вредност већа од 4 за променљиву x заједно са најмањим могућим вредностима за променљиве y и z (што је 1) у збиру дала вредност која је бар 7 (симетрична ситуација је са променљивама y и z). Са друге стране, ако би алгоритам филтрирања био свестан постојања `alldifferent` ограничења, тада би закључио да је допустиви интервал $[1, 3]$, с обзиром на то да вредност 4 придружена променљивој x захтева да променљиве y и z морају обе имати вредност 1, што није могуће, јер y и z морају узети различите вредности.

Горњи пример демонстрира ситуацију у којој се узимањем у обзир релевантних `alldifferent` ограничења могу израчунати јаче границе за променљиве у линеарном ограничењу. Мотивисани овим примером, у овом делу тезе представљамо побољшани алгоритам филтрирања који омогућава овакву врсту резоновања. Због потпуности излагања, најпре описујемо поменути стандардни алгоритам филтрирања [15] за линеарна ограничења [87], а затим разматрамо побољшани алгоритам и доказујемо његову коректност. На крају овог поглавља дајемо и експерименталну евалуацију која показује веома добро понашање побољшаног алгоритма на специфичним проблемима који у значајној мери укључују комбинације линеарних и `alldifferent` ограничења. Морамо нагласити да, за разлику од неких других приступа, наш алгоритам не успоставља конзистентност граница над конјункцијом линеарног и `alldifferent` ограничења,

[15] Термин *стандардни алгоритам филтрирања* можда није термин који се типично користи за овај метод у литератури, али то *јесте* стандардно коришћени метод за рачунање граница променљивих у линеарним ограничењима. Ми ћемо користити термин *стандардни алгоритам филтрирања* у овој тези како бисмо га јасно разликовали од нашег *побољшаног алгоритма филтрирања*.

али упркос слабијем нивоу конзистентности, његово понашање је веома добро у пракси. Са друге стране, главна предност нашег алгоритма у односу на друге приступе је његова општост, с обзиром на то да допушта произвољне комбинације линеарних и `alldifferent` ограничења (на пример, алгоритам може да се примени на конбинацију једног линеарног ограничења са више `alldifferent` ограничења чији се скупови променљивих само делимично преклапају са променљивама датог линеарног ограничења).

4.5.1 Стандардни алгоритам филтрирања за линеарно ограничење

Претпоставимо да имамо линеарно ограничење $e \leq c$, где је $e \equiv a_1 \cdot x_1 + \ldots + a_n \cdot x_n$. Процедура `calculateBoundsStandard`() (алгоритам 4.4) имплементира стандардни алгоритам филтрирања. Процедура прихвата e и c као улазне параметре и враћа `true` ако је ограничење $e \leq c$ конзистентно, а `false` у супротном. Такође, процедура има излазни параметар *bounds* у који се уписују израчунате границе (само ако је ограничење конзистентно). Идеја је да се најпре израчуна минимум израза e (који означавамо са $min(e)$) на следећи начин:

$$min(e) = min(a_1 \cdot x_1) + \ldots + min(a_n \cdot x_n)$$

где је

$$min(a_i \cdot x_i) \quad = \quad \begin{cases} a_i \cdot min(x_i), & a_i > 0 \\ a_i \cdot max(x_i), & a_i < 0 \end{cases}$$

Ако је $min(e) > c$, тада је ограничење неконзистентно. У супротном, за сваку од променљивих x_i рачунамо минимум израза e_{x_i} који се добија од израза e уклањањем монома $a_i \cdot x_i$, тј. $e_{x_i} \equiv a_1 \cdot x_1 + \ldots + a_{i-1} \cdot x_{i-1} + a_{i+1} \cdot x_{i+1} + \ldots + a_n \cdot x_n$. Овај минимум $min(e_{x_i})$ се тривијално може израчунати као $min(e) - min(a_i \cdot x_i)$. Сада, за сваку од променљивих x_i имамо следеће:

$$x_i \leq \left\lfloor \frac{c - min(e_{x_i})}{a_i} \right\rfloor \text{ ако је } a_i > 0 \quad \text{и} \quad x_i \geq \left\lceil \frac{c - min(e_{x_i})}{a_i} \right\rceil \text{ ако је } a_i < 0 \quad (4.3)$$

Вредности које не задовољавају израчунате границе треба одсећи из домена одговарајуће променљиве. Овим одсецањима се успоставља конзистентност граница над датим линеарним ограничењем.

Слична анализа се може извести и за ограничење $e \geq c$, с тим што се тада разматрају максимуми уместо минимума:

$$max(e) = max(a_1 \cdot x_1) + \ldots + max(a_n \cdot x_n)$$

где је

$$max(a_i \cdot x_i) \quad = \quad \begin{cases} a_i \cdot max(x_i), & a_i > 0 \\ a_i \cdot min(x_i), & a_i < 0 \end{cases}$$

Ако је $max(e) < c$, тада је ограничење неконзистентно. У супротном, границе за промењљиве се могу израчунати на следећи начин:

$$x_i \geq \left\lceil \frac{c - max(e_{x_i})}{a_i} \right\rceil \text{ ако је } a_i > 0 \quad \text{и} \quad x_i \leq \left\lfloor \frac{c - max(e_{x_i})}{a_i} \right\rfloor \text{ ако је } a_i < 0 \quad (4.4)$$

procedure `calculateBoundsStandard`$(e, c,$ `var` $bounds)$
begin
  {Претпоставка је да разматрамо ограничење $e \leq c$, где је $e \equiv a_1 \cdot x_1 + \ldots + a_n \cdot x_n$}
  {Нека је $V(e)$ скуп свих промењљивих које се појављују у e}
  {Најпре израчунавамо $min(e)$}
  $min(e) := 0$
  for all $x_i \in V(e)$ **do**
    if $a_i > 0$ **then**
      $min(e) := min(e) + a_i \cdot min(x_i)$
    else
      $min(e) := min(e) + a_i \cdot max(x_i)$
    end if
  end for
  if $min(e) > c$ **then** {испитујемо неконзистентност}
    return `false` {неконзистентност је откривена}
  end if
  for all $x_i \in V(e)$ **do** {израчунавамо границе промењљивих из $V(e)$}
    if $a_i > 0$ **then**
      $min(e_{x_i}) := min(e) - a_i \cdot min(x_i)$
      $bounds[x_i] := \left\lfloor \frac{c - min(e_{x_i})}{a_i} \right\rfloor$ {горња граница}
    else
      $min(e_{x_i}) := min(e) - a_i \cdot max(x_i)$
      $bounds[x_i] := \left\lceil \frac{c - min(e_{x_i})}{a_i} \right\rceil$ {доња граница}
    end if
  end for
  return `true` {неконзистентност *није* откривена}
end

Алгоритам 4.4: `calculateBoundsStandard`$(e, c,$ `var` $bounds)$

4.5.2 Побољшани алгоритам филтрирања за линеарно ограничење у присуству `alldifferent` ограничења

Претпоставимо поново да имамо ограничење $e \leq c$, где је $e \equiv a_1 \cdot x_1 + \ldots + a_n \cdot x_n$. Циљ је израчунати *побољшани минимум* израза e (који означавамо са $min^*(e)$), узимајући у обзир `alldifferent` ограничења над промењљивама из e.

Ако је тај побољшани минимум већи од c, тада је ограничење неконзистентно. У супротном, рачунамо на сличан начин побољшане минимуме $min^*(e_{x_i})$ који се затим користе за рачунање граница променљивих (као и код стандардног алгоритма):

$$x_i \leq \left\lfloor \frac{c - min^*(e_{x_i})}{a_i} \right\rfloor \text{ ако је } a_i > 0 \text{ и } x_i \geq \left\lceil \frac{c - min^*(e_{x_i})}{a_i} \right\rceil \text{ ако је } a_i < 0 \quad (4.5)$$

Због ефикасности, желели бисмо да избегнемо израчунавање сваког од минимума $min^*(e_{x_i})$ понављањем истог поступка од почетка. Дакле, имамо два основна проблема која овде разматрамо: први је како израчунати побољшани минимум $min^*(e)$, а други је како ефикасно израчунати *поправку* за $min^*(e_{x_i})$, тј. вредност c_i такву да је $min^*(e_{x_i}) = min^*(e) - c_i$. Слична ситуација је са ограничењем $e \geq c$, осим што се у том случају рачунају побољшани максимуми $max^*(e)$ и $max^*(e_{x_i})$, а онда се границе добијају из следећих израза:

$$x_i \geq \left\lceil \frac{c - max^*(e_{x_i})}{a_i} \right\rceil \text{ ако је } a_i > 0 \text{ и } x_i \leq \left\lfloor \frac{c - max^*(e_{x_i})}{a_i} \right\rfloor \text{ ако је } a_i < 0 \quad (4.6)$$

Побољшани алгоритам развијамо у две фазе. Најпре разматрамо једноставни случај где су сви коефицијенти a_i истог знака, и где у разматраном CSP проблему постоји ограничење `alldifferent`$(x_1, \ldots, x_n)$ (другим речима, све променљиве из e морају бити међусобно различите). Након тога разматрамо општи случај, где можемо имати више од једног `alldifferent` ограничења које се делимично преклапа са променљивама израза e, а коефицијенти a_i могу бити и позитивни и негативни.

4.5.2.1 Једноставни случај

Претпоставимо да је $a_i > 0$. Побољшани минимум $min^*(e)$ се израчунава процедуром `calculateImprovedMinimum`() (алгоритам 4.5). Процедура узима израз e као свој улазни параметар и рачуна побољшани минимум од e (који се враћа преко излазног параметра min), Додатно, процедура као излазне параметре има и *минимизујуће упаривање* $\mathbb{M}$ и *вектор индекса следећих најбољих кандидата* p (који се користи касније за израчунавање поправки). Под *упаривањем* $\mathbb{M}$ овде подразумевамо секвенцу придруживања $[x_{i_1} = d_{i_1}, \ldots, x_{i_n} = d_{i_n}]$, где је $x_{i_1}, \ldots, x_{i_n}$ пермутација скупа променљивих $x_1, \ldots, x_n$, таква да је $d_{i_j} \geq min(x_{i_j})$ (тј. свака променљива узима вредност која је већа или једнака од њеног мини-

мума), а $d_{i_k} < d_{i_l}$ за $k < l$ (тј. `alldifferent` ограничење је задовољено).[16] Упаривање $\mathbb{M}$ је *минимизујуће* ако сума $\sum_{j=1}^{n} a_{i_j} \cdot d_{i_j}$ има најмању могућу вредност. Процедура конструише такво упаривање и придружује вредност $\sum_{j=1}^{n} a_{i_j} \cdot d_{i_j}$ излазном параметру min.

procedure `calculateImprovedMinimum`(e, `var` min, `var` $\mathbb{M}$, `var` p)
begin
{Претпостављамо да је $e \equiv a_1 \cdot x_1 + \ldots + a_n \cdot x_n$, где је $a_i > 0$}
{Нека је $V(e)$ скуп променљивих из e}
$vars := sort(V(e))$ {сортирамо $V(e)$ по растућим минимумима}
$heap.init()$ {$heap$ је хип који ће садржати променљиве, а променљива којој одговара највећи коефициент у e ће увек бити на врху хипа (у случају више таквих променљивих, она променљива која има најмањи индекс у изразу e ће бити на врху). Иницијално, хип је празан}
$min := 0$
$\mathbb{M} := [\,]$ {иницијално, $\mathbb{M}$ је празна секвенца}
$i := 1$
$d := min(vars[1]) - 1$
for $j := 1$ **to** n **do**
$d := max(d+1, min(vars[j]))$ {израчунавамо следећу вредност која ће бити коришћена}
while $i \leq n \wedge min(vars[i]) \leq d$ **do**
$heap.add(vars[i])$ {додајемо кандидате за вредност d у хип}
$i := i + 1$
end while
$x := heap.get\_top()$ {скидамо променљиву са врха хипа (означену са x)}
$\mathbb{M}.push\_back(x = d)$
$min := min + a \cdot d$ {где је a коефициент уз x у изразу e}
if $heap$ није празан **then**
$next[j] := heap.view\_top()$ {очитавамо и памтимо следећу променљиву са врха хипа без њеног уклањања из хипа}
else
$next[j] := undef$
end if
$index[x] := j$ {памтимо индекс од x у секвенци $\mathbb{M}$}
end for
for $j := 1$ **to** n **do**
$p[j] := index[next[j]]$ {уз злоупотребу нотације, претпоставимо да је индекс од $undef$ такође $undef$}
end for
{На крају је $\mathbb{M} = [x_{i_1} = d_{i_1}, \ldots, x_{i_n} = d_{i_n}]$ минимизујуће упаривање, а $min = \sum_{j=1}^{n} a_{i_j} \cdot d_{i_j}$}
end

Алгоритам 4.5: `calculateImprovedMinimum`(e, `var` min, `var` $\mathbb{M}$, `var` p)

Процедура `calculateImprovedMinimum`() ради на следећи начин. Нека је $V(e)$ скуп променљивих које се појављују у изразу e. На почетку процедуре, најпре сортирамо променљиве из $V(e)$ у растућем поретку њихових минимума

[16] Приметимо да упаривање $\mathbb{M}$ не мора бити део неког решења датог CSP проблема, с обзиром на то да максимуми променљивих могу бити нарушени у $\mathbb{M}$.

(вектор означен са $vars$ у алгоритму 4.5). Основна идеја је да се пролази кроз вектор $vars$ и да се том приликом свакој промењљивој x придружи најмања вредност $d \geq min(x)$ која још увек није придружена некој другој промењљивој, при чему фаворизујемо промењљиве са већим коефициентима кад год је то могуће. Из тог разлога, у податку d у сваком тренутку чувамо највећу вредност која је до сада додељена некој од промењљивих (иницијално, вредност податка d је мања од најмањег минимума, тј. $d = min(vars[1]) - 1$). У свакој итерацији рачунамо следећу вредност коју треба доделити некој од преосталих промењљивих. То ће бити управо најмања вредност већа од d за коју постоји бар један *кандидат* међу преосталим промењљивама. *Промењљиве кандидати* за неку вредност су оне промењљиве чији су минимуми мањи или једнаки од те вредности. У случају више кандидата, бирамо ону промењљиву која има највећи коефициент у e, јер желимо да минимизујемо суму. Ако кандидат са највећим коефициентом није јединствен (с обзиром на то да више промењљивих могу имати једнаке коефициенте у e), узима се промењљива x_i са најмањим индексом i у e.[17] Како би се ефикасно одредио такав кандидат, такође одржавамо хип кандидата (податак $heap$ у алгоритму 4.5). Сваки пут када израчунамо следећу вредност d, додајемо нове кандидате у хип (кандидати који су претекли из претходних итерација такође остају на хипу, с обзиром на то да су те промењљиве такође кандидати и за нову вредност d). Промењљива на врху хипа је управо она са највећим коефициентом (и са најмањим индексом). Након што придружимо текућу вредност d промењљивој x која је скинута са врха хипа, дописујемо придруживање $x = d$ на секвенцу $\mathbb{M}$, а $a \cdot d$ додајемо на min (где је a коефицијент уз x у e) и прелазимо на следећу итерацију. На крају процедуре, излазни параметар min има вредност $\sum_{j=1}^{n} a_{i_j} \cdot d_{i_j}$, где је $\mathbb{M} = [x_{i_1} = d_{i_1}, \ldots, x_{i_n} = d_{i_n}]$, и та вредност се узима за побољшани минимум $min^*(e)$.

Након што је израчунат $min^*(e)$ (и уз претпоставку да је $min^*(e) \leq c$), желимо да даље израчунамо $min^*(e_{x_i})$ за свако $i \in \{1, \ldots, n\}$, али бисмо желели да избегнемо покретање алгоритма из почетка n пута. Идеја је да искористимо израчунати побољшани минимум $min^*(e)$ и минимизујуће упаривање $\mathbb{M}$ да ефикасно реконструишемо резултат који би био добијен када би се алгоритам покренуо за израз e_{x_i}. Претпоставимо да је $\mathbb{M} = [x_{i_1} = d_{i_1}, \ldots, x_{i_n} = d_{i_n}]$,

[17] У ствари, ако кандидат са највећим коефициентом није јединствен, било који од тих кандидата се може изабрати — израчунати побољшани минимум ће бити исти. Правило којим бирамо баш кандидата са најмањим индексом у e се користи у алгоритму само због одређености.

као и да за неко x_{i_j} желимо да израчунамо $min^*(e_{x_{i_j}})$. Када бисмо покренули алгоритам за $e_{x_{i_j}}$, добијено минимизујуће упаривање $\mathbb{M}_j$ би се поклапало са $\mathbb{M}$ све до j-те позиције у секвенци. Прва разлика би се појавила тек на j-тој позицији, с обзиром на то да се променљива x_{i_j} не појављује у $e_{x_{i_j}}$, па у том случају не би ни била на хипу. У таквој ситуацији, постојала би два могућа случаја:

- ако је променљива x_{i_j} била једини кандидат за вредност d_{i_j} у току конструкције $\mathbb{M}$, у одсуству променљиве x_{i_j} алгоритам не би могао да користи ову вредност, тако да би наставио са следећом вредношћу $d_{i_{j+1}}$ која би била додељена најбољем кандидату за ту вредност — то би опет била променљива $x_{i_{j+1}}$, као и раније. У наставку алгоритма, променљиве би биле распоређене на исти начин као у $\mathbb{M}$. Самим тим, упаривање $\mathbb{M}_j$ би било исто као и $\mathbb{M}$, осим што би придруживање $x_{i_j} = d_{i_j}$ било уклоњено.

- ако променљива x_{i_j} није била једини кандидат за вредност d_{i_j} у току конструкције упаривања $\mathbb{M}$, у одсуству променљиве x_{i_j} алгоритам би изабрао следећег најбољег кандидата са хипа и придружио би тој променљивој вредност d_{i_j}. Нека је x_{i_l} изабрани следећи најбољи кандидат (приметимо да мора бити $l > j$, јер је x_{i_l} распоређена након x_{i_j} у $\mathbb{M}$). Након што би се променљива x_{i_l} уклонила са хипа и придруживање $x_{i_l} = d_{i_j}$ додало на $\mathbb{M}_j$, преостале променљиве би биле распоређене на исти начин као и у минимизујућем упаривању $\mathbb{M}_l$, тј. оном које би било добијено да је алгоритам био позван за израз $e_{x_{i_l}}$. Другим речима, једина разлика између $\mathbb{M}_j$ и $\mathbb{M}_l$ је у придруживању $x_{i_l} = d_{i_j}$ уместо придруживања $x_{i_j} = d_{i_j}$, тако да можемо свести проблем проналажења $\mathbb{M}_j$ (и $min^*(e_{x_{i_j}})$) на проблем проналажења $\mathbb{M}_l$ (и $min^*(e_{x_{i_l}})$).

Како бисмо могли да реконструишемо описано понашање алгоритма за $e_{x_{i_j}}$ без да га заиста позовемо, током извршавања алгоритма за израз e памтимо следећег најбољег кандидата за сваку вредност у добијеном минимизујућем упаривању $\mathbb{M}$. Након што је најбољи кандидат уклоњен са хипа, а одговарајуће придруживање додато на $\mathbb{M}$, очитавамо следећу променљиву са врха хипа (али је не уклањамо са хипа) и памтимо ту променљиву као следећег најбољег кандидата за текућу вредност (променљива означена са $next[j]$ у алгоритму 4.5). Ако је хип празан, користимо специјалну вредност $undef$ да назначимо да не постоје алтернативни кандидати. На крају алгоритма, за сваку вредност d_{i_j} у $\mathbb{M}$ рачунамо $p[j]$, што представља вредност индекса у $\mathbb{M}$ на ком је позициони-

рана променљива $next[j]$ ($p[j] > j$). Ако нема других кандидата за вредност d_{i_j}, тада је $p[j] = undef$.

Последично, када рачунамо поправке $c[j] = min^*(e) - min^*(e_{x_{i_j}})$, имамо два могућа случаја. Ако је $p[j] = undef$, тада би минимизујуће упаривање $\mathbb{M}_j$ било идентично упаривању $\mathbb{M}$, осим што би придруживање $x_{i_j} = d_{i_j}$ било уклоњено, па је зато $c[j] = a_{i_j} \cdot d_{i_j}$. Са друге стране, ако је $p[j] = l > j$, тада можемо претпоставити да је поправка $c[l]$ већ израчуната (тј. претпостављамо да се корекције израчунавају у обрнутом поретку). С обзиром на то да се минимизујуће упаривање $\mathbb{M}_j$ може реконструисати на основу минимизујућег упаривања $\mathbb{M}_l$ тако што се замени придруживање $x_{i_j} = d_{i_j}$ придруживањем $x_{i_l} = d_{i_j}$, следи да је $min^*(e_{x_{i_j}}) = min^*(e_{x_{i_l}}) - a_{i_j} \cdot d_{i_j} + a_{i_l} \cdot d_{i_j} = min^*(e) - c[l] - (a_{i_j} - a_{i_l}) \cdot d_{i_j}$, па је зато $c[j] = (a_{i_j} - a_{i_l}) \cdot d_{i_j} + c[l]$.

Процедура `calculateBoundsImproved`() (алгоритам 4.6) има исте параметре и повратну вредност као и процедура `calculateBoundsStandard`(), али имплементира описани побољшани алгоритам филтрирања. Она најпре покреће процедуру `calculateImprovedMinimum`(), затим проверава неконзистентност (тј. да ли је $min^*(e) > c$), а затим израчунава поправке као што је претходно описано. На крају израчунава горње границе за све променљиве, као у изразу (4.5).

procedure `calculateBoundsImproved`($e, c,$ `var` $bounds$)
begin
  {Претпоставка је да разматрамо ограничење $e \leq c$, где је $e \equiv a_1 \cdot x_1 + \ldots + a_n \cdot x_n$ и $a_i > 0$}
  {Најпре израчунавамо побољшани минимум $min^*(e)$, минимизујуће упаривање $\mathbb{M}$ као и вектор индекса следећих најбољих кандидата p}
  `calculateImprovedMinimum`($e, min^*(e), \mathbb{M}, p$)
  {Сада је $\mathbb{M}$ секвенца $[x_{i_1} = d_{i_1}, \ldots, x_{i_n} = d_{i_n}]$}
  if $min^*(e) > c$ **then** {испитујемо неконзистентност}
    return `false` {неконзистентност је откривена}
  end if
  for $j := n$ **downto** 1 **do** {рачунамо границе за променљиве из $\mathbb{M}$}
    {Рачунамо поправку $c[j]$}
    if $p[j] = undef$ **then**
      $c[j] := a_{i_j} \cdot d_{i_j}$
    else
      $c[j] := d_{i_j} \cdot (a_{i_j} - a_{i_{p[j]}}) + c[p[j]]$
    end if
    $min^*(e_{x_{i_j}}) := min^*(e) - c[j]$
    $bounds[x_{i_j}] := \left\lfloor \frac{c - min^*(e_{x_{i_j}})}{a_{i_j}} \right\rfloor$
  end for
  return `true` {неконзистентност *није* откривена}
end

Алгоритам 4.6: `calculateBoundsImproved`($e, c,$ `var` $bounds$)

Коришћењем побољшаних минимума, често можемо да одсечемо више вредности из домена, као што се може видети у следећем примеру.

Пример 4.5.2. Размотримо следећи CSP проблем:

$$x_1 \in \{1, \ldots, 10\},\ x_2 \in \{2, \ldots, 10\}$$
$$x_3 \in \{1, \ldots, 10\},\ x_4 \in \{3, \ldots, 10\}$$
$$x_5 \in \{3, \ldots, 15\},\ x_6 \in \{9, \ldots, 40\}$$
$$\texttt{alldifferent}(x_1, x_2, x_3, x_4, x_5, x_6)$$
$$6x_1 + 8x_2 + 7x_3 + 4x_4 + 2x_5 + x_6 \leq 85$$

Означимо са e израз на левој страни линеарног ограничења у овом проблему. Ако применимо стандардни алгоритам да израчунамо границе (тј. игноришемо `alldifferent` ограничење), тада је $min(e) = 56$, што значи да није откривена неконзистентност, а вредности $min(e_{x_i})$ као и израчунате границе су дате у следећој табели:

Променљива	x_1	x_2	x_3	x_4	x_5	x_6
$min(e_{x_i})$	50	40	49	44	50	47
$x_i \leq$	**5**	**5**	**5**	10	17	**38**

Вредности написане масним фонтом означавају израчунате границе које узрокују одсецања. Размотримо сада побољшани алгоритам примењен на исти проблем. Најпре рачунамо $min^*(e)$ тако што позовемо процедуру `calculateImprovedMinimum`() за e:

- прва вредност коју алгоритам узима је $d = 1$ за коју постоје два кандидата: x_1 и x_3. Процедура бира x_3, јер је њен коефициент у e већи (x_1 остаје на хипу као кандидат за следећу вредност). Следећи најбољи кандидат за ову вредност је управо x_1.

- следећа вредност која се разматра је $d = 2$ за коју такође постоје два кандидата: x_1 и x_2. Овог пута процедура бира x_2, а x_1 ће морати да сачека на хипу и следећу вредност (ова променљива је поново следећи најбољи кандидат и за вредност $d = 2$).

- следећа вредност је $d = 3$, а овог пута имамо три кандидата: x_1, x_4 и x_5. Овог пута узимамо променљиву x_1, с обзиром на то да има највећи коефициент. Следећи најбољи кандидат је x_4.

- за следећу вредност $d = 4$ постоје два кандидата на хипу: x_4 и x_5. Процедура бира промењљиву x_4, а x_5 памти као следећег најбољег кандидата.
- следећа вредност је $d = 5$ која се придружује промењљивој x_5, с обзиром на то да је она једина преостала на хипу (следећи најбољи кандидат за ову вредност је зато $undef$).
- следећа вредност за коју постоји бар један кандидат је $d = 9$. Једини кандидат за ову вредност је промењљива x_6 која се зато узима са хипа (следећи најбољи кандидат је поново $undef$).

Добијено упаривање је $\mathbb{M} = [x_3 = 1, x_2 = 2, x_1 = 3, x_4 = 4, x_5 = 5, x_6 = 9]$, побољшани минимум је $min^*(e) = 76$, а вектор индекса следећих најбољих кандидата је $p = [3, 3, 4, 5, undef, undef]$ (ово значи да је за вредност 1 следећи најбољи кандидат промењљива x_1 која је на позицији 3 у $\mathbb{M}$, за вредност 2 следећи најбољи кандидат је x_1 који је на позицији 3, за вредност 3 следећи најбољи кандидат је x_4 који је на позицији 4, итд.). С обзиром на то да је $76 \leq 85$, ни овог пута није откривена неконзистентност. Након што се израчунају поправке, можемо израчунати побољшане минимуме $min^*(e_{x_i})$ и искористити ове вредности за рачунање граница промењљивих, приказаних у следећој табели:

Промењљиве	x_1	x_2	x_3	x_4	x_5	x_6
j (индекс у $\mathbb{M}$)	3	2	1	4	5	6
$p[j]$	4	3	3	5	$undef$	$undef$
$c[j]$	24	28	25	18	10	9
$min^*(e_{x_i})$	52	48	51	58	66	67
$x_i \leq$	5	**4**	**4**	**6**	**9**	**18**

Вредности исписане масним фонтом представљају границе које су јаче у поређењу са оним границама које су добијене стандардним алгоритмом. Овај пример потврђује да наш алгоритам може произвести више одсецања него стандардни алгоритам.

Сложеност алгоритма. Сложеност процедуре `calculateImprovedMinimum`() је $O(n \log(n))$, с обзиром на то да се низ промењљивих најпре једном сортира, а затим се свака од промењљивих по једном додаје и једном скида са хипа. Процедура `calculateBoundsImproved`() једном позива процедуру

`calculateImprovedMinimum`(), а затим рачуна поправке и границе у линеарном времену. Стога, сложеност ове процедуре је такође $O(n\log(n))$.

Ниво конзистентности. Као што смо видели из примера 4.5.2, наш побољшани алгоритам заиста има већу пропагацијску моћ у односу на стандардни алгоритам. Ипак, наш алгоритам не успоставља конзистентност граница над конјункцијом `alldifferent` ограничења и линеарног ограничења. На овом месту је можда занимљиво прокоментарисати однос нашег алгоритма и алгоритма који су описали Белдицеану и други [13], а који третира сличан проблем — конјункцију $x_1 + \ldots + x_n \leq c \ \wedge$ `alldifferent`$(x_1, \ldots, x_n)$ (приметимо да се овде захтева да сви коефициенти у суми буду једнаки 1). Овај алгоритам успоставља конзистентност граница над оваквом конјункцијом, а има сасвим сличну структуру као и наш алгоритам. Главна разлика је у поретку који се користи за кандидате на хипу. Алгоритам који су описали Белдицеану и други [13] са хипа увек узима ону променљиву која има најмањи максимум. Аутори показују да, у случају да су сви коефициенти једнаки 1, ова стратегија доводи до упаривања $\mathbb{M}$ које минимизује суму и задовољава оба ограничења, поштујући притом *и минимуме и максимуме* променљивих. Ако такво упаривање не постоји, алгоритам пријављује неконзистентност. У нашем алгоритму, допуштамо произвољне позитивне коефициенте, а најбољи кандидат на хипу је увек онај са највећим коефициентом. Упаривање $\mathbb{M}$ ће и у нашем случају задовољавати оба ограничења, али само минимуми променљивих ће бити испоштовани (максимуми могу бити нарушени, с обзиром на то да их алгоритам потпуно игнорише). Услед ове релаксације, израчунати побољшани минимум може бити мањи од стварног минимума, па неке неконзистентности можда неће бити откривене, а конзистентност граница неће бити успостављена. Другим речима, жртвовали смо конзистентност граница да бисмо покрили знатно општији случај, где су допуштени произвољни коефициенти.

Ограничење $e \geq c$. Као што је раније речено, случај ограничења $e \geq c$ је сличан, осим што се разматрају побољшани максимуми $max^*(e)$ и $max^*(e_{x_i})$. Да бисмо израчунали побољшани максимум $max^*(e)$, користимо процедуру аналогну процедури `calculateImprovedMinimum`() (алгоритам 4.5), уз две битне разлике. Прво, променљиве се на почетку сортирају по опадајућим максимумима. Друго, вредност d се иницијализује на $max(M_1, \ldots, M_n) + 1$, где је

$M_i = max(x_i)$, а затим се у свакој следећој итерацији поставља на највећу могућу вредност мању од своје претходне вредности за коју постоји бар један кандидат међу преосталим променљивама (овога пута, кандидат је свака променљива y за коју је $max(y) \geq d$). Поправке за $max^*(e_{x_i})$ се рачунају на идентичан начин као у алгоритму 4.6. Границе се рачунају као што је наведено у изразу (4.6).

Негативни коефициенти. Размотримо сада случај ограничења $e \leq c$ и $e \geq c$, где је $e \equiv a_1 \cdot x_1 + \ldots + a_n \cdot x_n$, а $a_i < 0$. Приметимо да је у том случају $e \equiv -|e|$, где је $|e| \equiv |a_1| \cdot x_1 + \ldots + |a_n| \cdot x_n$. Из овог разлога, важи да је $min^*(e) = -max^*(|e|)$, и слично, $max^*(e) = -min^*(|e|)$, тако да се проблем проналажења побољшаних минимума/максимума лако своди на претходни случај са позитивним коефициентима. Поправке за $min^*(e_{x_i})$ и $max^*(e_{x_i})$ рачунамо тако што израчунамо поправке за $min^*(|e_{x_i}|)$ и $max^*(|e_{x_i}|)$ као у алгоритму 4.6, уз промену знака.

4.5.2.2 Општи случај

Претпоставимо сада општи случај, где променљиве могу имати и позитивне и негативне коефициенте у e и може бити више `alldifferent` ограничења која се само делимично преклапају са променљивама из $V(e)$. Овај општи случај сводимо на претходни једноставни случај тако што партиционишемо скуп $V(e)$ на дисјунктне подскупове на које се могу применити претходни алгоритми. Партиционисање се обавља процедуром `findPartitions`() (алгоритам 4.7). Ова процедура као улазе узима израз e и скуп свих `alldifferent` ограничења у разматраном проблему (означен са *csts*). Излазни параметар *pts* ће садржати резултат. Први корак је да се скуп $V(e)$ партиционише као $V^+ \cup V^-$, где је V^+ скуп свих променљивих из $V(e)$ са позитивним коефициентима, а V^- садржи све променљиве из $V(e)$ са негативним коефициентима. Сваки скуп $V \in \{V^+, V^-\}$ се даље партиционише на дисјунктне *ad-партиције*. Кажемо да је скуп променљивих $V' \subseteq V$ `alldifferent` *партиција* (или *ad-партиција*) у V, ако постоји `alldifferent` ограничење у разматраном CSP проблему које укључује све променљиве из V'. Веће *ad*-партиције су пожељније, па најпре проналазимо највећу *ad*-партицију у V тако што разматрамо све пресеке скупа V са релевантним `alldifferent` ограничењима и бирамо онај пресек који има највећу кардиналност. Када је таква *ad*-партиција идентификована, промен-

љиве које чине ту партицију се уклањају из скупа V, а преостале ad-партиције се даље траже на исти начин међу преосталим промењивама из V. Веома је битно нагласити да овако добијене партиције зависе једино од `alldifferent` ограничења у проблему, као и од знакова коефицијената у e. С обзиром на то да се ово не мења у току решавања, сваки позив процедуре `findPartitions`() би дао исти резултат. Ово значи да се ова процедура може позвати *само једном*, на почетку решавања, а резултат се може сачувати и користити касније по потреби.

procedure `findPartitions`($e, csts,$ `var` pts)
begin
  {$csts$ је скуп свих `alldifferent` ограничења у разматраном проблему}
  {Претпостављамо да је $e \equiv a_1 \cdot x_1 + \ldots + a_n \cdot x_n$, где a_i могу бити произвољног знака}
  {Нека је $V(e) = V^+ \cup V^-$, где је V^+ скуп промењивих са позитивним коефициентима у e, а V^- је скуп промењивих са негативним коефициентима у e}
  $pts := \varnothing$ {pts ће на крају садржати скуп партиција}
  for all $V \in \{V^+, V^-\}$ **do** {партиционисање се обавља посебно за V^+ и V^-}
    $V_{curr} := V$
    while $V_{curr} \neq \varnothing$ **do**
      $S_{max} := \varnothing$ {највећа ad-партиција}
      for all $ad \in csts$ **do**
        $S_{curr} := V(ad) \cap V_{curr}$ {$V(ad)$ означава скуп промењивих у ограничењу ad}
        if $|S_{curr}| > |S_{max}|$ **then**
          $S_{max} := S_{curr}$
        end if
      end for
      if $|S_{max}| \geq 2$ **then** {ако највећа ad-партиција има бар 2 промењиве}
        $pts := pts \cup \{S_{max}\}$
        $V_{curr} := V_{curr} \setminus S_{max}$
      else {ако све преостале промењиве чине јединичне партиције}
        for all $x \in V_{curr}$ **do**
          $S := \{x\}$
          $pts := pts \cup \{S\}$
        end for
        $V_{curr} := \varnothing$
      end if
    end while
  end for
end

Алгоритам 4.7: `findPartitions`($e, csts,$ `var` pts)

Процедура `calculateBoundsImprovedGen`() (алгоритам 4.8) има исте параметре и повратну вредност као и процедура `calculateBoundsImproved`() (алгоритам 4.6), али се примењује на општи случај. Ослања се на партиционисање израза $e \equiv e^1 + \ldots + e^s$ добијено процедуром `findPartitions`() (алгоритам 4.7). С обзиром на то да за сваку од партиција e^k све њене промењиве имају

коефициенте истог знака и покривене су неким `alldifferent` ограничењем, процедура `calculateImprovedMinimum`() (алгоритам 4.5) се може искористити да се израчунају побољшани минимуми $min^*(e^k)$. Побољшани минимум целог израза $min^*(e)$ се даље израчунава као сума побољшаних минимума за изразе e^k, односно $min^*(e) = \sum_{k=1}^{s} min^*(e^k)$. Ако је $min^*(e) > c$, процедура `calculateBoundsImprovedGen`() пријављује неконзистентност. У супротном, процедура рачуна поправке и границе на сличан начин као у алгоритму 4.6. Постоје две битне разлике. Прво, користимо вредност $b^k = c - (min^*(e) - min^*(e^k))$ уместо c као горњу границу за израз e^k када рачунамо границе за промењљиве из e^k. Друга битна разлика је та што морамо разликовати два случаја, у зависности од знака коефицијената: за позитивне коефициенте израчуната граница је горња граница, док је за негативне коефициенте израчуната граница доња граница (зато што се знак неједнакости окреће приликом дељења негативним бројем).

Сложеност алгоритма. Нека је $n_k = |V(e^k)|$, и нека је $n = |V(e)| = n_1 + \ldots + n_s$. Процедура `calculateBoundsImprovedGen`() (алгоритам 4.8) позива процедуру `calculateImprovedMinimum`() (Algorithm 4.5) по једном за сваку партицију e^k, што узима $\sum_{k=1}^{s} O(n_k \log(n_k)) = O(n \log(n))$ времена. С обзиром на то да се поправке и границе рачунају у линеарном времену (у односу на n), укупна сложеност је $O(n \log(n))$.

4.5.2.3 Доказ коректности алгоритма

У наредном тексту доказујемо коректност наведених алгоритама. Користићемо следећу нотацију. За линеарни израз $e \equiv a_1 \cdot x_1 + \ldots + a_n \cdot x_n$ и n-торку $\mathbf{d} = (d_1, \ldots, d_n)$, уводимо ознаку $e[\mathbf{d}] = \sum_{i=1}^{n} a_i \cdot d_i$.[18] За упаривање $\mathbb{M} = [x_{i_1} = d_{i_1}, \ldots, x_{i_n} = d_{i_n}]$ над скупом промењљивих $V(e)$, нека је са $\mathbf{d}(\mathbb{M})$ означена n-торка која одговара $\mathbb{M}$, тј. n-торка $(d_1, \ldots, d_n)$ таква да је d_i управо вредност која је у $\mathbb{M}$ придружена промењљивој x_i. Такође уводимо ознаку $e(\mathbb{M}) = e[\mathbf{d}(\mathbb{M})] = \sum_{j=1}^{n} a_{i_j} \cdot d_{i_j}$.

Коректност процедуре `calculateImprovedMinimum`() (алгоритам 4.5) следи из теореме 4.5.1 која тврди да је побољшани минимум добијен овом процедуром

[18] Другим речима, $e[\mathbf{d}]$ је вредност коју израз e узима када се промењљивама x_i придруже вредности d_i. Уопште, у наставку текста, кад год говоримо о n-торци $(d_1, \ldots, d_n)$, подразумевамо да се вредност d_i придружује промењљивој x_i.

procedure `calculateBoundsImprovedGen`$(e, c,$ `var` $bounds)$
begin
  {Претпоставка је да разматрамо ограничење $e \leq c$, где је $e = a_1 \cdot x_1 + \ldots + a_n \cdot x_n$}
  {Нека је $e^1 + \ldots + e^s$, партиционисање израза e добијено алгоритмом 4.7}
  {Најпре рачунамо побољшани минимум за e}
  $min^*(e) := 0$
  for $k := 1$ **to** s **do**
    `calculateImprovedMinimum`$(e^k, min^*(e^k), \mathbb{M}^k, p^k)$
    $min^*(e) := min^*(e) + min^*(e^k)$
  end for
  if $min^*(e) > c$ **then** {проверавамо неконзистентност}
    return `false` {неконзистентност је откривена}
  end if
  for $k := 1$ **to** s **do** {За сваку партицију e^k рачунамо поправке и границе}
    $b^k := c - (min^*(e) - min^*(e^k))$ {b^k је израчуната горња граница за e^k}
    {Нека је $\mathbb{M}^k = [x_{i_{k,1}} = d_{i_{k,1}}, \ldots, x_{i_{k,n_k}} = d_{i_{k,n_k}}]$}
    for $j := n_k$ **downto** 1 **do**
      {Рачунамо поправку $c^k[j]$}
      if $p^k[j] = undef$ **then**
        $c^k[j] := a_{i_{k,j}} \cdot d_{i_{k,j}}$
      else
        $c^k[j] := d_{i_{k,j}} \cdot (a_{i_{k,j}} - a_{i_{k,p^k[j]}}) + c^k[p^k[j]]$
      end if
      $min^*(e^k_{x_{i_{k,j}}}) := min^*(e^k) - c^k[j]$
      if $a_{i_{k,j}} > 0$ **then**
        $bounds[x_{i_{k,j}}] := \left\lfloor \frac{b^k - min^*(e^k_{x_{i_{k,j}}})}{a_{i_{k,j}}} \right\rfloor$ {горња граница}
      else
        $bounds[x_{i_{k,j}}] := \left\lceil \frac{b^k - min^*(e^k_{x_{i_{k,j}}})}{a_{i_{k,j}}} \right\rceil$ {доња граница}
      end if
    end for
  end for
  return `true` {неконзистентност *није* откривена}
end

Алгоритам 4.8: `calculateBoundsImprovedGen`$(e, c,$ `var` $bounds)$

заиста најмања могућа вредност коју дати израз може узети, под условом да све променљиве узимају међусобно различите вредности које су веће или једнаке од њихових минимума.

Теорема 4.5.1. Нека је $e \equiv a_1 \cdot x_1 + \ldots + a_n \cdot x_n$, где је $a_i > 0$, и нека је $m_i = min(x_i)$ минимум променљиве x_i. Нека је $\mathbb{M} = [x_{i_1} = d_{i_1}, \ldots, x_{i_n} = d_{i_n}]$ упаривање добијено покретањем процедуре `calculateImprovedMinimum`() за израз e, и нека је $min = e(\mathbb{M}) = \sum_{j=1}^{n} a_{i_j} \cdot d_{i_j}$ добијени побољшани минимум. Тада за сваку n-торку $\mathbf{d} = (d_1, \ldots, d_n)$ такву да је $d_i \geq m_i$ и $d_i \neq d_j$ за $i \neq j$, важи да

је $e[\mathbf{d}] \geq min$.

Доказ. Нека је $T = \{(d_1, \ldots, d_n) \mid d_i \geq m_i,\ i \neq j \Rightarrow d_i \neq d_j\}$ скуп n-торки које задовољавају услове теореме, и нека је $E(T) = \{e[\mathbf{d}] \mid \mathbf{d} \in T\}$. Скуп T је непразан (на пример, $(h, h+1, h+2, \ldots, h+n-1) \in T$, где је $h = max(m_1, \ldots, m_n)$), а скуп $E(T)$ је ограничен одоздо (на пример, са $\sum_{i=1}^{n} a_i \cdot m_i$). Зато је минимум $min(E(T))$ коначан, и постоји бар једна n-торка $\mathbf{d} \in T$ за коју је $e[\mathbf{d}] = min(E(T))$. Желимо да докажемо да је n-торка $\mathbf{d}(\mathbb{M})$ која одговара упаривању $\mathbb{M}$ које је добијено нашим алгоритмом једна таква n-торка, тј. важи да је $e(\mathbb{M}) = min(E(T))$. Прво, приметимо да је $\mathbf{d}(\mathbb{M}) \in T$ за било које упаривање $\mathbb{M}$ (ово следи из дефиниције упаривања, као и из дефиниције скупа T). Стање алгоритма пред сваку итерацију главне петље се може описати тројком $(V^c, \mathbb{M}^c, d^c)$, где је V^c скуп променљивих којима још увек није придружена вредност (иницијално $\{x_1, \ldots, x_n\}$), $\mathbb{M}^c$ је текуће парцијално упаривање (иницијално празна секвенца $[\,]$), а d^c је највећа вредност која је до сада придружена некој променљивој (иницијално $min(m_1, \ldots, m_n) - 1$). Доказаћемо следеће својство: за свако стање $(V^c, \mathbb{M}^c, d^c)$ које је добијено полазећи од иницијалног стања извршавањем алгоритма, парцијално упаривање $\mathbb{M}^c$ може бити проширено до потпуног упаривања $\mathbb{M}'$ за које је $e(\mathbb{M}') = min(E(T))$, и то придруживањем вредности које су веће од d^c променљивама из V^c. Ово својство ће бити доказано индукцијом по $k = |\mathbb{M}^c|$. За $k = 0$ својство тривијално важи, с обзиром на то да се празна секвенца свакако може проширити до неког минимизујућег упаривања коришћењем вредности које су веће од $d^c = min(m_1, \ldots, m_n) - 1$ (с обзиром на то да такво упаривање постоји). Претпоставимо да ово својство важи за неко стање $(V^c, \mathbb{M}^c, d^c)$, где је $|\mathbb{M}^c| = k$, и докажимо да онда то својство мора да важи и за стање $(V^c \setminus \{\overline{x}\}, [\mathbb{M}^c, \overline{x} = \overline{d}], \overline{d})$, где је $\overline{x} = \overline{d}$ управо оно придруживање које наш алгоритам бира у тој итерацији. Подсетимо се да је вредност $\overline{d}$ коју наш алгоритам бира управо најмања могућа вредност већа од d^c за коју постоји бар један кандидат међу променљивама из V^c (тј. неко $y \in V^c$ такво да је $min(y) \leq \overline{d}$), и да је променљива $\overline{x}$ коју наш алгоритам узима са хипа управо онај кандидат за вредност $\overline{d}$ који има највећи коефициент (ако такав кандидат није јединствен, тада се између њих бира променљива са најмањим индексом у e). Ако је $\mathbb{M}'$ било које потпуно минимизујуће упаривање које проширује парцијално упаривање $\mathbb{M}^c$ користећи вредности веће од d^c, тада важи следеће:

- вредност $\overline{d}$ мора бити искоришћена у $\mathbb{M}'$. Заиста, с обзиром на то да $\overline{d}$ има бар једног кандидата $y \in V^c$, уколико не бисмо користили вредност $\overline{d}$ у упаривању $\mathbb{M}'$ имали бисмо да је $y = d' \in \mathbb{M}'$, где је $d' > \overline{d}$. Ово је због тога што су све вредности придружене променљивама из V^c веће од d^c, а $\overline{d}$ је најмања могућа вредност већа од d^c са бар једним кандидатом. Због тога би се придруживање $y = d' \in \mathbb{M}'$ могло заменити са $y = \overline{d}$, чиме би се добило друго потпуно упаривање $\mathbb{M}''$ такво да је $e(\mathbb{M}'') < e(\mathbb{M}')$. Ово је у контрадикцији са претпоставком да је $\mathbb{M}'$ минимизујуће упаривање.

- ако је $y = \overline{d} \in \mathbb{M}'$, тада променљива y мора бити кандидат за $\overline{d}$ са највећим могућим коефициентом. Заиста, ако је коефициент уз y у линеарном изразу једнак a, и ако би постојао други кандидат $y' \in V^c$ са већим коефициентом a', тада би избор кандидата y уместо y' за вредност $\overline{d}$ поново резултирао упаривањем $\mathbb{M}'$ које није минимизујуће: с обзиром на то да је $y' = d' \in \mathbb{M}'$, при чему је $d' > \overline{d}$, и $y = \overline{d} \in \mathbb{M}'$, заменом вредности променљивих y и y' добили бисмо друго упаривање $\mathbb{M}''$ такво да је $e(\mathbb{M}'') < e(\mathbb{M}')$.

- ако је $y = \overline{d} \in \mathbb{M}'$, где је y један од кандидата за вредност $\overline{d}$ са највећим коефициентом, тада за било ког другог кандидата z са највећим коефициентом (ако постоји) одговарајуће придруживање $z = \overline{d}$ припада неком минимизујућем упаривању $\mathbb{M}''$ које проширује $\mathbb{M}^c$. Заиста, ако је $y = \overline{d} \in \mathbb{M}'$ и z је неки други кандидат за $\overline{d}$ који има коефициент у изразу e као и y (и коме је придружена вредност $d' > \overline{d}$ у $\mathbb{M}'$), заменом вредности за кандидате y и z добили бисмо друго упаривање $\mathbb{M}''$ за које би вредност израза e остала непромењена, тј. $e(\mathbb{M}'') = e(\mathbb{M}') = min(E(T))$.

Из претходно доказаних чињеница следи да придруживање $\overline{x} = \overline{d}$ изабрано од стране нашег алгоритма такође припада неком минимизујућем упаривању $\mathbb{M}'$ које проширује $\mathbb{M}^c$ придруживањем вредности већих од d^c променљивама из V^c. С обзиром на то да је најмања вредност већа од d^c са бар једним кандидатом управо $\overline{d}$, и с обзиром на то да је $\overline{x} = \overline{d} \in \mathbb{M}'$, следи да вредности придружене променљивама из $V^c \setminus \{\overline{x}\}$ морају бити веће од $\overline{d}$ у таквом упаривању $\mathbb{M}'$. Овим смо доказали да и парцијално упаривање $[\mathbb{M}^c, \overline{x} = \overline{d}]$ такође може бити проширено до истог тог минимизујућег упаривања $\mathbb{M}'$ коришћењем вредности већих од $\overline{d}$. На овај начин смо доказали да, под претпоставком да наведено својство (које покушавамо да докажемо индукцијом) важи за неко стање $(V^c, \mathbb{M}^c, d^c)$, исто својство важи и за стање $(V^c \setminus \{\overline{x}\}, [\mathbb{M}^c, \overline{x} = \overline{d}], \overline{d})$ што је управо наредно

стање добијено нашим алгоритмом, при чему је $|[\mathbb{M}^c, \overline{x} = \overline{d}]| = k + 1$. Сада индукцијом следи да наведено својство важи за било које стање $(V^c, \mathbb{M}^c, d^c)$ добијено извршавањем алгоритма, укључујући и завршно стање $(\{\ \}, \mathbb{M}, d_{i_n})$, где је $\mathbb{M} = [x_{i_1} = d_{i_1}, \ldots, x_{i_n} = d_{i_n}]$. Самим тим, $min = e(\mathbb{M}) = min(E(T))$, чиме је теорема доказана. □

Нека $subst(\mathbb{M}, A, B)$ означава резултат који се добија када се подсеквенца A замени (потенцијално празном) подсеквенцом B у упаривању $\mathbb{M}$. Следећа теорема 4.5.2 тврди да се поправке у процедури `calculateBoundsImproved`() (алгоритам 4.6) израчунавају на исправан начин.

Теорема 4.5.2. Нека је $e \equiv a_1 \cdot x_1 + \ldots + a_n \cdot x_n$, где је $a_i > 0$. Нека је $\mathbb{M} = [x_{i_1} = d_{i_1}, \ldots, x_{i_n} = d_{i_n}]$ минимизујуће упаривање добијено процедуром `calculateImprovedMinimum`() за израз e, и нека је $min^*(e) = e(\mathbb{M})$ израчунати побољшани минимум. Нека је $\mathbb{M}_j$ минимизујуће упаривање које би било добијено да је процедура позвана за $e_{x_{i_j}}$ и нека је $min^*(e_{x_{i_j}}) = e(\mathbb{M}_j)$ одговарајући побољшани минимум. Најзад, нека је $c_j = min^*(e) - min^*(e_{x_{i_j}})$. Ако је x_{i_j} био једини кандидат за вредност d_{i_j} приликом извршавања процедуре за израз e, тада је $\mathbb{M}_j = subst(\mathbb{M}, [x_{i_j} = d_{i_j}] \rightarrow [\ ])$. У супротном, ако је следећи најбољи кандидат био x_{i_l} $(l > j)$, тада је $\mathbb{M}_j = subst(\mathbb{M}_l, [x_{i_j} = d_{i_j}] \rightarrow [x_{i_l} = d_{i_j}])$. Последично, одговарајућа поправка је $c_j = a_{i_j} \cdot d_{i_j}$, или $c_j = (a_{i_j} - a_{i_l}) \cdot d_{i_j} + c_l$, респективно.

Доказ. Најпре, нека је $\mathbf{S} = (V, \mathbb{M}, d)$ било које стање извршавања алгоритма, са истим значењем као у доказу теореме 4.5.1. *Релација преласка* између стања ће бити означена са $\rightarrow$: ако је $\mathbf{S}'$ стање у које се непосредно прелази из стања $\mathbf{S}$, то ћемо записивати као $\mathbf{S} \rightarrow \mathbf{S}'$. Транзитивно затворење ове релације ће бити означено са $\rightarrow^*$. Приметимо да је релација $\rightarrow$ детерминистичка, што значи да за било које стање $\mathbf{S}$ постоји јединствено стање $\mathbf{S}'$ такво да је $\mathbf{S} \rightarrow \mathbf{S}'$. Самим тим, ово такође важи и за релацију $\rightarrow^*$. Ово значи да тренутно стање у потпуности одређује даље извршавање алгоритма — за два идентична стања остатак извршавања алгоритма ће бити идентичан. Међутим, чак и за два различита стања даље извршавање алгоритма може делимично или потпуно да се поклапа. Кажемо да су два стања $\mathbf{S}' = (V', \mathbb{M}', d')$ и $\mathbf{S}'' = (V'', \mathbb{M}'', d'')$ *слабо слична* ако је следеће придруживање које алгоритам бира исто за оба стања, тј. постоји неко придруживање $\overline{x} = \overline{d}$ такво да је $(V', \mathbb{M}', d') \rightarrow (V' \setminus \{\overline{x}\}, [\mathbb{M}', \overline{x} = \overline{d}], \overline{d})$, и $(V'', \mathbb{M}'', d'') \rightarrow (V'' \setminus \{\overline{x}\}, [\mathbb{M}'', \overline{x} = \overline{d}], \overline{d})$. Са друге стране, кажемо да

су два стања $\mathbf{S}'$ и $\mathbf{S}''$ *слична* ако ће *сва* наредна придруживања бити изабрана на идентичан начин за оба стања, тј. постоји нека секвенца придруживања S таква да је $(V', \mathbb{M}', d') \rightarrow^* (\{\ \}, [\mathbb{M}', S], d_f)$, и $(V'', \mathbb{M}'', d'') \rightarrow^* (\{\ \}, [\mathbb{M}'', S], d_f)$, где је d_f последња придружена вредност у оба упаривања. Најпре приметимо да важе следећа два својства:

својство 1 ако за два стања $(V', \mathbb{M}', d')$ и $(V'', \mathbb{M}'', d'')$ важи да је $d' = d''$ и $V'' = V' \setminus \{y\}$, где $y \in V'$ није променљива која би била изабрана за следеће придруживање у стању $(V', \mathbb{M}', d')$, тада су та два стања слабо слична. Заиста, с обзиром на то да у стању $(V', \mathbb{M}', d')$ алгоритам бира неко придруживање $\overline{x} = \overline{d}$, где је $\overline{x} \neq y$ и $\overline{d} > d'$, исто придруживање ће бити изабрано и у стању $(V'', \mathbb{M}'', d'')$, јер је $\overline{x} \in V''$, и $d'' = d'$.

својство 2 ако за два стања $(V', \mathbb{M}', d')$ и $(V'', \mathbb{M}'', d'')$ важи да је $V' = V''$ и $d' = d''$, тада су та два стања слична. Заиста, претходна придруживања из $\mathbb{M}'$ и $\mathbb{M}''$ не утичу на будућа придруживања која ће алгоритам одабирати, већ на њих утичу искључиво преостале променљиве (а знамо да је $V' = V''$), као и последња додељена вредност (а важи $d' = d''$).

Нека је даље $\mathbb{M} = [x_{i_1} = d_{i_1}, \ldots, x_{i_n} = d_{i_n}]$ минимизујуће упаривање добијено покретањем алгоритма за израз e, и нека је $\mathbb{M}_j$ минимизујуће упаривање добијено покретањем алгоритма за израз $e_{x_{i_j}}$. Подсетимо се да је израз $e_{x_{i_j}}$ добијен уклањањем монома $a_{i_j} \cdot x_{i_j}$ из e, тј. $V(e_{x_{i_j}}) = V(e) \setminus \{x_{i_j}\}$. Нека је $\mathbf{S}^k$ стање након k итерација алгоритма покренутог за израз e, и нека је $\mathbf{S}^k_j$ стање након k итерација алгоритма покренутог за израз $e_{x_{i_j}}$. С обзиром на то да променљива x_{i_j} није најбољи кандидат све до j-те позиције у $\mathbb{M}$, према првом својству, стања $\mathbf{S}^k$ и $\mathbf{S}^k_j$ су слабо слична за $k < j - 1$. Другим речима, придруживања изабрана од стране алгоритма пре j-те позиције ће бити идентична у упаривањима $\mathbb{M}$ и $\mathbb{M}_j$. Ово значи да је секвенца $P = [x_{i_1} = d_{i_1}, \ldots, x_{i_{j-1}} = d_{i_{j-1}}]$ заједнички префикс за упаривања $\mathbb{M}$ и $\mathbb{M}_j$.

Размотримо сада понашање алгоритма на j-тој позицији. Нека је $V^k = V(e) \setminus \{x_{i_1}, \ldots, x_{i_k}\}$ за свако $k \leq n$. Када се алгоритам позове за израз e, имамо следеће преласке стања: $\mathbf{S}^{j-1} \rightarrow \mathbf{S}^j \rightarrow \mathbf{S}^{j+1}$, где је $\mathbf{S}^{j-1} = (V^{j-1}, P, d_{i_{j-1}})$, $\mathbf{S}^j = (V^j, [P, x_{i_j} = d_{i_j}], d_{i_j})$, и $\mathbf{S}^{j+1} = (V^{j+1}, [P, x_{i_j} = d_{i_j}, x_{i_{j+1}} = d_{i_{j+1}}], d_{i_{j+1}})$. Са друге стране, ако би алгоритам био позван за израз $e_{x_{i_j}}$, када се достигне стање $\mathbf{S}^{j-1}_j = (V^{j-1} \setminus \{x_{i_j}\}, P, d_{i_{j-1}})$, имали бисмо два могућа случаја. Ако је x_{i_j} био једини кандидат за d_{i_j} у стању $\mathbf{S}^{j-1}$, тада не би било кандидата за d_{i_j} у стању

$\mathbf{S}_j^{j-1}$. Због тога алгоритам проналази прву већу вредност за коју постоји бар један кандидат у $V^{j-1} \setminus \{x_{i_j}\}$, а то је управо вредност $d_{i_{j+1}}$. Најбољи кандидат за такву вредност ће бити исти као у стању $\mathbf{S}^j$, зато што су скупови преосталих променљивих исти у оба стања ($V^{j-1} \setminus \{x_{i_j}\} = V^j$). Зато ће следеће стање бити $\mathbf{S}_j^j = (V^{j-1} \setminus \{x_{i_j}\} \setminus \{x_{i_{j+1}}\}, [P, x_{i_{j+1}} = d_{i_{j+1}}], d_{i_{j+1}})$. У складу са другим доказаним својством, стања $\mathbf{S}^{j+1}$ и $\mathbf{S}_j^j$ су слична (јер је $V^{j-1} \setminus \{x_{i_j}\} \setminus \{x_{i_{j+1}}\} = V^{j+1}$), па ће се добијена потпуна упаривања $\mathbb{M}$ и $\mathbb{M}_j$ поклапати на преосталим позицијама, односно $\mathbf{S}^{j+1} \rightarrow^* (\{\ \}, [P, x_{i_j} = d_{i_j}, x_{i_{j+1}} = d_{i_{j+1}}, S], d_{i_n})$ и $\mathbf{S}_j^j \rightarrow^* (\{\ \}, [P, x_{i_{j+1}} = d_{i_{j+1}}, S], d_{i_n})$, где је $S = [x_{i_{j+2}} = d_{i_{j+2}}, \ldots, x_{i_n} = d_{i_n}]$ заједнички суфикс упаривања $\mathbb{M}$ и $\mathbb{M}_j$. Због тога је $\mathbb{M}_j = subst(\mathbb{M}, [x_{i_j} = d_{i_j}] \rightarrow [\ ])$. Одговарајућа поправка је тада $c_j = min^*(e) - min^*(e_{x_{i_j}}) = e(\mathbb{M}) - e_{x_{i_j}}(\mathbb{M}_j) = a_{i_j} \cdot d_{i_j}$. Са друге стране, ако постоји следећи најбољи кандидат x_{i_l} за вредност d_{i_j} ($l > j$) у стању $\mathbf{S}^{j-1}$, тада ће та променљива бити најбољи кандидат за d_{i_j} у стању $\mathbf{S}_j^{j-1}$. Због тога ће алгоритам изабрати придруживање $x_{i_l} = d_{i_j}$, тј. $\mathbf{S}_j^j = (V^{j-1} \setminus \{x_{i_j}\} \setminus \{x_{i_l}\}, [P, x_{i_l} = d_{i_j}], d_{i_j})$. Како бисмо одредили наредне преласке стања, размотримо најпре понашање нашег алгоритма када је позван за израз $e_{x_{i_l}}$. Као што је претходно речено, упаривања $\mathbb{M}_l$ и $\mathbb{M}$ ће се поклапати све до l-те позиције, а с обзиром на то да је $l > j$, следи да ће $\mathbb{M}_l$ такође имати P за префикс. Због тога је $\mathbf{S}_l^{j-1} = (V^{j-1} \setminus \{x_{i_l}\}, P, d_{i_{j-1}})$. Ово стање је слабо слично са $\mathbf{S}^{j-1}$ (на основу првог својства), па ће наредно стање бити $\mathbf{S}_l^j = (V^{j-1} \setminus \{x_{i_l}\} \setminus \{x_{i_j}\}, [P, x_{i_j} = d_{i_j}], d_{i_j})$. Сада су стања $\mathbf{S}_l^j$ и $\mathbf{S}_j^j$ очигледно слична (на основу другог својства), па важи $\mathbf{S}_j^j \rightarrow^* (\{\ \}, [P, x_{i_l} = d_{i_j}, S'], d')$ и $\mathbf{S}_l^j \rightarrow^* (\{\ \}, [P, x_{i_j} = d_{i_j}, S'], d')$, где је d' највећа придружена вредност, а S' је заједнички суфикс. Из овога следи да је $\mathbb{M}_j = subst(\mathbb{M}_l, [x_{i_j} = d_{i_j}] \rightarrow [x_{i_l} = d_{i_j}])$. Одговарајућа поправка је тада $c_j = min^*(e) - min^*(e_{x_{i_j}}) = e(\mathbb{M}) - e_{x_{i_j}}(\mathbb{M}_j) = e(\mathbb{M}) - (e_{x_{i_l}}(\mathbb{M}_l) - a_{i_j} \cdot d_{i_j} + a_{i_l} \cdot d_{i_j}) = c_l + (a_{i_j} - a_{i_l}) \cdot d_{i_j}$, као што је и наведено у тврђењу. □

Из следеће теореме следи коректност алгоритма 4.6 (једноставни случај).

Теорема 4.5.3. Нека је P CSP проблем који садржи ограничења `alldifferent`$(x_1, \ldots, x_n)$, и $e \leq c$, где је $e \equiv a_1 \cdot x_1 + \ldots + a_n \cdot x_n$ и $a_i > 0$. Ако процедура `calculateBoundsImproved`() примењена на израз e враћа *false*, тада проблем P нема решења. У супротном, нека је P' CSP проблем добијен из P одсецањем вредности изван израчунатих граница за променљиве из $V(e)$. Тада су проблеми P и P' еквивалентни.

Доказ. Нека је $\mathbf{d} = (d_1, \ldots, d_n)$ било која n-торка из $\mathbb{D} = D_{x_1} \times \ldots \times D_{x_n}$, и нека је $min^*(e)$ побољшани минимум добијен процедуром `calculateImprovedMinimum`() за израз e. Најпре приметимо да ако $\mathbf{d}$ задовољава ограничење `alldifferent`$(x_1, \ldots, x_n)$, тада је $min^*(e) \leq e[\mathbf{d}]$. Заиста, с обзиром на то да је $\mathbf{d} \in \mathbb{D}$, следи да је $d_i \geq m_i$, где је $m_i = min(x_i)$. Даље, с обзиром на то да $\mathbf{d}$ задовољава ограничење `alldifferent`$(x_1, \ldots, x_n)$, следи да је $i \neq j \Rightarrow d_i \neq d_j$. Ово значи да су услови теореме 4.5.1 испуњени, тако да мора бити $e[\mathbf{d}] \geq min^*(e)$. Слично се може доказати да је $e_{x_i}[\mathbf{d}] \geq min^*(e_{x_i})$.

Ако процедура `calculateBoundsImproved`() враћа *false* када се примени на израз e, то значи да је $min^*(e) > c$. Ово значи да за свако $\mathbf{d} \in \mathbb{D}$ које задовољава ограничење `alldifferent`$(x_1, \ldots, x_n)$ важи да је $e[\mathbf{d}] \geq min^*(e) > c$, што значи да не постоји n-торка $\mathbf{d} \in \mathbb{D}$ која задовољава и `alldifferent` ограничење и ограничење $e \leq c$. Другим речима, проблем P нема решења.

Да бисмо доказали други део теореме, морамо доказати да ни једна вредност која је одсечена нашим побољшаним алгоритмом није била део неког решења проблема P. Претпоставимо да је за неку променљиву x_i нека вредност d_i одсечена из њеног домена нашим побољшаним алгоритмом. На основу израза (4.5), ово значи да је $d_i > \left\lfloor \frac{c - min^*(e_{x_i})}{a_i} \right\rfloor$, односно, $min^*(e_{x_i}) + a_i \cdot d_i > c$. Зато за сваку n-торку $\mathbf{d} \in \mathbb{D}$ која укључује d_i као вредност за променљиву x_i и која задовољава ограничење `alldifferent`$(x_1, \ldots, x_n)$ мора важити да је $e[\mathbf{d}] = e_{x_i}[\mathbf{d}] + a_i \cdot d_i \geq min^*(e_{x_i}) + a_i \cdot d_i > c$ (с обзиром на то да је $e_{x_i}[\mathbf{d}] \geq min^*(e_{x_i})$), што значи да ограничење $e \leq c$ није задовољено, па $\mathbf{d}$ није решење проблема P. Ово значи да не постоји ни једно решење проблема P које садржи d_i као вредност за x_i и које задовољава и ограничење `alldifferent`$(x_1, \ldots, x_n)$ и ограничење $e \leq c$. Ово доказује теорему, с обзиром на то да су сва решења проблема P уједно и решења проблема P'. □

Најзад, доказујемо коректност алгоритма 4.8 (општи случај).

Теорема 4.5.4. Нека је P CSP проблем који садржи ограничење $e \leq c$, где је $e \equiv a_1 \cdot x_1 + \ldots + a_n \cdot x_n$ (a_i могу бити и позитивни и негативни), као и једно или више `alldifferent` ограничења која се могу делимично преклапати са $V(e)$. Ако процедура `calculateBoundsImprovedGen`() примењена на израз e враћа *false*, тада проблем P нема решења. У супротно, нека је P' CSP проблем добијен из P одсецањем вредности изван израчунатих граница за променљиве из $V(e)$. Тада су проблеми P и P' еквивалентни.

Доказ. Нека је $e \equiv e^1 + \ldots + e^s$ партиционисање израза e добијено процедуром `findPartitions`() (алгоритам 4.7). С обзиром на то да за сваку партицију e^k постоји покривајуће `alldifferent` ограничење у P, и с обзиром на то да све променљиве у партицији имају коефициенте истог знака, из теореме 4.5.1 следи да за сваку n-торку $\mathbf{d} \in \mathbb{D} = D_{x_1} \times \ldots \times D_{x_n}$ која задовољава сва `alldifferent` ограничења проблема P мора важити да је $e[\mathbf{d}] = e^1[\mathbf{d}] + \ldots + e^s[\mathbf{d}] \geq min^*(e^1) + \ldots + min^*(e^s) = min^*(e)$. Ако је $min^*(e) > c$, тада не постоје n-торке које задовољавају и `alldifferent` ограничења проблема P и ограничење $e \leq c$. Овим је доказан први део теореме.

Да бисмо доказали други део теореме, најпре приметимо да за сваку n-торку $\mathbf{d}$ која задовољава и `alldifferent` ограничења из P и ограничење $e \leq c$ важи да је $min^*(e^1) + \ldots + min^*(e^{k-1}) + e^k[\mathbf{d}] + min^*(e^{k+1}) + \ldots + min^*(e^s) \leq e^1[\mathbf{d}] + \ldots + e^{k-1}[\mathbf{d}] + e^k[\mathbf{d}] + e^{k+1}[\mathbf{d}] + \ldots + e^s[\mathbf{d}] \leq c$. Зато важи $e^k[\mathbf{d}] \leq c - (min^*(e) - min^*(e^k))$ за произвољно решење проблема P. Ово оправдава употребу вредности $b^k = c - (min^*(e) - min^*(e^k))$ као горње границе за израз e^k. Остатак доказа је аналоган доказу теореме 4.5.3, осим што се вредности b^k користе уместо c. □

4.5.3 Имплементациони детаљи

Руковалац линеарним ограничењем има само један дедуктивни слој у оквиру кога се покрећу алгоритми описани у претходним одељцима. Имплементирани су стандардни алгоритам филтрирања, побољшани алгоритам филтрирања и алгоритам који су предложили Белдицеану и други [13]. Алгоритам који ће бити коришћен се може изабрати задавањем одговарајуће опције при покретању решавача.

Генерисање објашњења неконзистентности и одсецања за линеарно ограничење се своди на објашњавање максимума или минимума променљивих у ограничењу. Претпоставимо поново да имамо ограничење $e \leq c$, где је $e \equiv a_1 \cdot x_1 + \ldots + a_n \cdot x_n$. Ако је откривена неконзистентност, то значи да је $min(e) > c$ (или $min^*(e) > c$, у случају побољшаног алгоритма). С обзиром на то да је вредност $min(e)$ одређена вредностима $min(x_i)$ (или $max(x_i)$ у случају негативног коефициента a_i), морамо пронаћи литерале са трага M који су одговорни за успостављање таквих граница за променљиве. На пример, ако је $min(x) = 2$, а литерал $x \geq 2$ је на трагу M, тада ће литерал $x \geq 2$ бити искоришћен као објашњење за мининум. Са друге стране, ако се у неком каснијем кораку литерал $x \neq 2$ дода на траг M, нови минимум ће бити $min(x) = 3$, а

његово објашњење ће бити $x \geq 2, x \neq 2$ (први литерал је одсекао вредности мање од 2, док је други одсекао вредност 2 из домена, услед чега је минимум променљиве постао једнак 3). У случају коришћења побољшаног алгоритма, релевантна `alldifferent` ограничења се такође морају додати у објашњење, с обзиром на то да ова ограничења такође утичу на израчунату вредност $min^*(e)$. Са друге стране, да бисмо објаснили пропагације, тј. одсецања вредности из домена неке променљиве x_i, довољно је објаснити вредност израчунате границе за променљиву x_i, што се своди на објашњавање минимума $min(e_{x_i})$ (односно $min^*(e_{x_i})$) на аналоган начин као у претходном случају.

4.5.4 Експериментална евалуација

У овом одељку дата је експериментална евалуација чији је главни циљ да се упореде перформансе стандардног алгоритма филтрирања (алгоритам 4.4) и нашег побољшаног алгоритма филтрирања (алгоритам 4.8) у оквиру нашег `argosmt` решавача. Други циљ је да се упореди наш алгоритам са алгоритмом који су развили Белдицеану и други [13]. Овај алгоритам ћемо у наставку називати *bel* алгоритам. Поређење са *bel* алгоритмом ће бити могуће само на проблемима на које је могуће применити *bel* алгоритам (с обзиром на то да он захтева да сви коефициенти у суми буду једнаки 1). Најзад, трећи циљ је да упоредимо наш решавач са неким актуелним CSP решавачима.

Решавачи и проблеми. Како бисмо постигли наведене циљеве, експерименти су вршени над следећим решавачима:

- `argosmt-sbc` — наш решавач[19], користећи стандардни алгоритам филтрирања
- `argosmt-ibc` — наш решавач, користећи побољшани алгоритам филтрирања
- `argosmt-bel` — наш решавач, користећи *bel* алгоритам
- `sugar` — Sugar[20] SAT-заснован CSP решавач, користећи `minisat`[21] као позадински SAT решавач

[19] `http://www.matf.bg.ac.rs/~milan/argosmt-5.5/`
[20] `http://bach.istc.kobe-u.ac.jp/sugar/`
[21] `http://http://minisat.se/`

- `mistral` — Mistral[22] CSP решавач
- `opturion` — Opturion[23] решавач заснован на *лењом генерисању клауза* [77]

Решавачи су тестирани на следећих пет скупова инстанци:[24]

какуро — скуп се састоји од 100 случајно генерисаних инстанци раније описаног *какуро* проблема величине 30×30. Коришћено је исто кодирање као и у претходној експерименталној евалуацији (одељак 4.4.11), једино што су у овом случају инстанце веће димензије. Све инстанце су и овог пута задовољиве.

KA — овај скуп инстанци је инспирисан следећим познатим *криптоаритметичким* проблемом: сваком од 26 слова енглеске абецеде треба доделити *вредност* — број из скупа $\{1, \ldots, 26\}$ такав да сва слова имају различите вредности, као и да суме вредности слова у датим речима буду једнаке датим бројевима:

ballet	=	45,	cello	=	43,	concert	=	74,	flute	=	30,
fugue	=	50,	glee	=	66,	jazz	=	58,	lyre	=	47,
oboe	=	53,	opera	=	65,	polka	=	59,	quartet	=	50,
saxophone	=	134,	scale	=	51,	solo	=	37,	song	=	61,
soprano	=	82,	theme	=	72,	violin	=	100	waltz	=	34

На случајан начин смо генерисали 100 сличних задовољивих инстанци — свака инстанца се састоји од 20 случајно генерисаних „речи" чије су дужине од 4 до 9. И овог пута кодирање је сасвим праволинијско: сваком слову се придружује променљива чији је домен управо скуп $\{1, \ldots, 26\}$, постоји једно `alldifferent` ограничење над свих 26 променљивих (јер сва слова морају узети различите вредности), а за сваку реч постоји и по једно линеарно ограничење (сума одговарајућих променљивих мора бити једнака датом броју). Приметимо да коефицијенти у линеарним ограничењима не морају бити једнаки 1, јер се неко слово може појављивати више пута у речи.

KTK — скуп се састоји од 100 случајно генерисаних инстанци проблема *комплетирања тежинских квазигрупа* (енгл. *weighted quasigroup completion problem*) [88]. У питању је потпуно исти скуп инстанци који је коришћен и у претходној експерименталној евалуацији (одељак 4.4.11). Инстанце овог скупа су од

[22] `http://homepages.laas.fr/ehebrard/mistral.html`

[23] `http://www.opturion.com/`

[24] Инстанце се могу наћи на локацији: `http://www.math.rs/~milan/argosmt-5.5/instances.zip`

значаја и за ову експерименталну евалуацију, зато што садрже велики број `alldifferent` ограничења која се преклапају са линеарним ограничењима са великим коефициентима.

MK — четврти скуп се састоји из 40 инстанци проблема *магичних квадрата* које су преузете из *CSP-LIB*[25] колекције. Уопштено, проблем магичних квадрата се може описати на следећи начин: бројеви $1, 2, \ldots, n^2$ треба да се упишу у поља матрице димензија $n \times n$, тако да све вредности у матрици буду међусобно различите, а суме вредности у свакој врсти, свакој колони и свакој од две дијагонале морају бити једнаке. Неке вредности су унапред дате, а циљ је комплетирати матрицу на описан начин. Да бисмо кодирали овај проблем, уводимо променљиву x_{ij} за поље у i-тој врсти и j-тој колони матрице. Све овако уведене променљиве узимају вредности из домена $\{1, \ldots, n^2\}$. Имамо једно `alldifferent` ограничење које покрива све x_{ij} променљиве. Најзад, имамо по једно линеарно ограничење за сваку врсту, сваку колону и сваку дијагоналу, чиме ограничавамо одговарајуће променљиве да у суми дају вредност $n \cdot (n^2 + 1)/2$. Предефинисане вредности се једноставно кодирају као једнакости. Наш конкретни скуп инстанци се састоји из 40 инстанци величине 9×9, при чему 20 инстанци има по 10 предефинисаних вредности у матрици, док осталих 20 инстанци има по 50 предефинисаних вредности у матрици.

у-какуро — пети проблем који разматрамо је уопштење какуро проблема такво да свако од поља у таблици такође има и предефинисану *тежину* — позитиван број од 1 до 9. Сада свако поље треба попунити вредношћу од 1 до 9 тако да су у свакој линији све вредности међусобно различите, и да вредности у свакој од линија помножене тежинама одговарајућих поља у суми дају број који је дат у таблици. У поређењу са стандардним какуром, овог пута коефициенти у линеарним ограничењима нису сви једнаки 1, што ће нам помоћи у евалуацији нашег алгоритма у овом уопштеном случају. Скуп се састоји из 100 случајно генерисаних задовољивих инстанци величине 100×100.

Поређење са стандардним алгоритмом. Табела 4.4 приказује број решених инстанци и просечно време извршавања за све решаваче. За све скупове инстанци, временско ограничење по инстанци је било 3600 секунди (за нерешене инстанце, ово временско ограничење је коришћено при рачунању просеч-

[25] `http://www.csplib.org/Problems/prob019/data/MagicSquareCompletionInstances.zip`

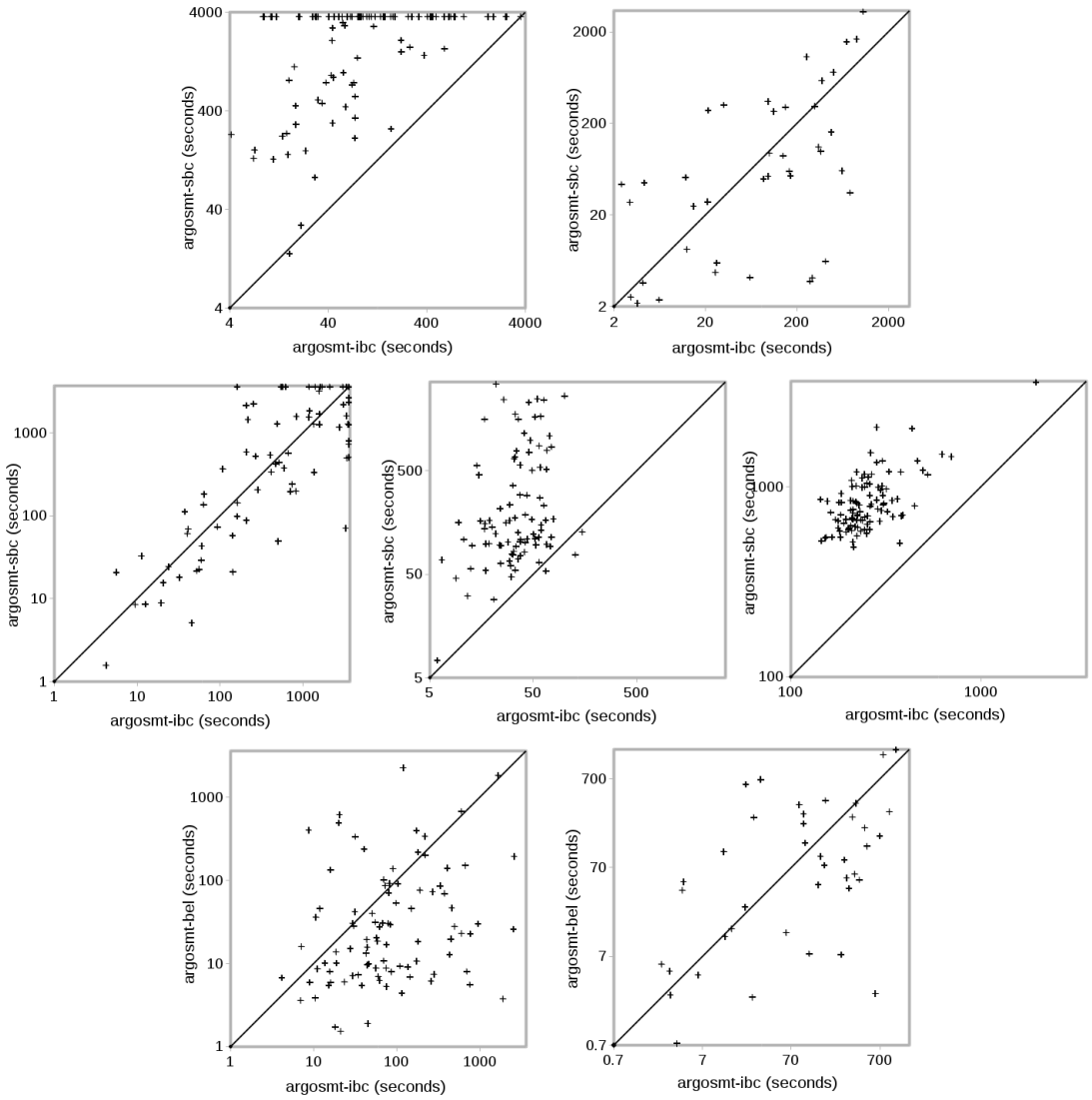
ног времена извршавања). Резултати показују да су укупне перформансе нашег решавача са побољшаним алгоритмом значајно боље у поређењу са стандарднм алгоритмом филтрирања. С обзиром на то да просечно време само за себе не даје довољно информација о дистрибуцији времена извршавања, на слици 4.12 (горњих пет графика) приказујемо како се времена извршавања односе на појединачним инстанцама. Тачке изнад дијагонале представљају инстанце за које је побољшани алгоритам био бољи од стандардног алгоритма. Графици делују прилично убедљиво за *какуро* инстанце (горњи леви график), *KTK* инстанце (централни график) и *у-какуро* инстанце (средњи десни график), где је већина тачака изнад дијагонале. Побољшање није тако очигледно за *MK* инстанце (горњи десни график) и *KA* инстанце (средњи леви график).

	какуро		**MK**	
	# инстанци	**гранично**	**# инстанци**	**гранично**
	100	3600s	40	3600s
	# решених	**просек**	**# решених**	**просек**
argosmt-sbc	40	2521s	40	296s
argosmt-ibc	94	464s	40	226s
argosmt-bel	99	196s	40	198s
sugar	100	91s	40	57s
mistral	10	3305s	39	111s
opturion	55	2126s	40	10s

	KA		**KTK**		**у-какуро**	
	# инстанци	**гранично**	**# инстанци**	**гранично**	**# инстанци**	**гранично**
	100	3600s	100	3600s	100	3600s
	# решено	**просек**	**# решено**	**просек**	**#решено**	**просек**
argosmt-sbc	63	1745s	92	722s	99	904s
argosmt-ibc	70	1691s	100	44s	100	283s
sugar	95	557s	0	3600s	100	42s
mistral	96	215s	33	2412s	0	3600s
opturion	84	985s	100	43s	100	62s

Табела 4.4: За сваки решавач и сваки скуп инстанци приказан је број решених инстанци у оквиру задатог временског ограничења, као и просечно време извршавања (за нерешене инстанце, временско ограничење је коришћено као време извршавања)

Табела 4.5 приказује информације о редукцији простора претраге. С обзиром на то да је наш решавач у основи SMT решавач заснован на CDCL SAT решавачу, најбољи начин за поређење простора претраге је да се погледа просечан број гранања и конфликата. За *какуро* инстанце, простор претраге је редукован скоро четири пута. Слична је ситуација и са *у-какуро* инстанцама, а још драстичнија је за *KTK* инстанце. Са друге стране, редукција простора претраге је безначајна за *KA* и *MK* инстанце. Ово се у великој мери поклапа са



Слика 4.12: Поређење времена извршавања: *какуро* – *ibc* наспрам *sbc* (горе лево), *MK* – *ibc* наспрам *sbc* (горе десно), *KA* – *ibc* наспрам *sbc* (средње лево), *KTK* – *ibc* наспрам *sbc* (центар), *у-какуро* – *ibc* наспрам *sbc* (средње десно), *какуро* – *ibc* наспрам *bel* (доле лево), *MK* – *ibc* наспрам *bel* (доле десно)

редукцијом просечног времена извршавања коју смо видели у табели 4.4. Табела 4.5 такође приказује просечни удео времена извршавања које је проведено у алгоритму филтрирања. Као што је очекивано, побољшани алгоритам конзумира значајно више времена (с обзиром на то да му је сложеност $O(n \log(n))$, док стандардни алгоритам има линеарну сложеност), али је редукција простора претраге коју он проузрокује свакако вредна тог додатног трошка.

	какуро			MK		
	sbc	**ibc**	**bel**	**sbc**	**ibc**	**bel**
# конфликата	274370	72785	45635	13206	11021	8516
# гранања	375863	98192	59064	25983	23126	17627
% времена у алг. филт.	9.1%	25.9%	24.8%	1.7%	5.6%	4.5%

	KA		KTK		у-какуро	
	sbc	**ibc**	**sbc**	**ibc**	**sbc**	**ibc**
# конфликата	187910	184409	31598	2345	12738	4464
# гранања	231503	225675	39852	6052	76492	23876
% времена у алг. филт.	14.6%	23.6%	5.0%	20.6%	1.7%	4.4%

Табела 4.5: Табела приказује просечан број конфликата и гранања, као и просечан проценат времена проведеног у алгоритму филтрирања

Табела 4.6 приказује просечни проценат граница које су заиста побољшане у поређењу са границама које би биле добијене стандардним алгоритмом, као и просечан износ побољшања граница. Можемо видети да су у случају *какуро* инстанци 41% свих израчунатих граница јаче када се користи побољшани алгоритам. Иако просечно побољшање граница изгледа мало — само 2.48, ово је заправо значајно побољшање за *какуро* инстанце, с обзиром на јако мале домене променљивих (величине 9). У случају *KA* инстанци, просечно побољшање граница је 1.44, али су домени знатно већи овог пута (величине 26), што може бити објашњење за скромно побољшање перформанси на овим инстанцама. Слична ситуација је и за *MK* инстанце, где су домени величине 81, а просечно побољшање је само 1.56. Са друге стране, за *KTK* инстанце скоро све границе (95.5%) су побољшане, а просечно побољшање је чак 1331, услед великих коефицијената у линеарним ограничењима.

	какуро		MK	
	ibc/sbc	**bel/sbc**	**ibc/sbc**	**bel/sbc**
% побољшано	41.0%	52.2%	24.0%	30.8%
пр. побољшање	2.48	2.72	1.56	1.59

	KA	KTK	у-какуро
	ibc/sbc	**ibc/sbc**	**ibc/sbc**
% побољшано	27.2%	95.5%	33.2%
пр. побољшање	1.44	1331.58	2.43

Табела 4.6: Табела приказује проценат побољшаних граница, као и просечан износ побољшања граница

Поређење са *bel* алгоритмом. Размотримо сада поређење нашег побољшаног алгоритма филтрирања (који користи решавач `argosmt-ibc`) и *bel* алгоритма [13], који се користи у `argosmt-bel` решавачу. Поређење је засновано на два проблема (*какуро* и *MK*) на која се оба алгоритма могу применити. Пре свега, у табели 4.4 можемо видети да је `argosmt-bel` значајно бољи на *какуро* инстанцама, док не постоји значајно убрзање на *MK* инстанцама. Ово такође потврђује и слика 4.12, где доња два графика представљају поређење `argosmt-ibc` и `argosmt-bel` решавача (доњи леви график је за *какуро*, а доњи десни график је за *MK*). И овог пута тачке изнад дијагоналне линије представљају инстанце за које је `argosmt-ibc` бољи. За *какуро* инстанце, већина тачака су испод дијагонале, што јасно потврђује да је `argosmt-bel` бољи. Ово није случај са *MK* инстанцама, где су тачке готово равномерно разбацане са обе стране дијагонале.

Редукција простора претраге приказана у тебели 4.5 је такође конзистентна са претходним опажањима. Још можемо видети (табела 4.5) да алгоритам филтрирања у решавачу `argosmt-bel` конзумира сличан удео времена као и алгоритам у решавачу `argosmt-ibc`. Ово је очекивано, с обзиром на то да и наш алгоритам и *bel* алгоритам имају исту временску сложеност [13].

Додатне информације корисне за упоређивање ова два алгоритма су дате у табели 4.6. За решавач `argosmt-bel`, проценат побољшаних граница (у поређењу са стандардним алгоритмом, тј. са решавачем `argosmt-sbc`) за *какуро* инстанце је 52.2% у просеку, што је нешто боље него у случају нашег побољшаног алгоритма (41%). Слична ситуација је и за *MK* инстанце (30.8% наспрам 24%), али је ефекат овог побољшања мање значајан на овим инстанцама због већих домена. Просечно побољшање је слично за оба алгоритма. Даљим испитивањем открили смо да за *какуро* инстанце око 34% свих детектованих конфликата од стране *bel* алгоритма не би било детектовано од стране нашег побољшаног алгоритма (а мање од 20% у случају *MK* инстанци). Можемо закључити да ће се *bel* алгоритам сасвим извесно понашати боље од нашег алгоритма, али ова разлика не мора бити драстична, иако *bel* алгоритам успоставља јачи ниво конзистентности. Са друге стране, значајна предност нашег алгоритма је у његовој примењивости на знатно широј класи проблема.

Поређење са актуелним решавачима. Табела 4.4 такође показује да је наш решавач упоредив са актуелним решавачима који су коришћени у експе-

риментима. Иако наш решавач није најбољи избор ни за један од пет пробема (други најбољи је за *какуро* и скоро једнако добар као `opturion` за *KTK* инстанце), ако саберемо резултате на свих пет проблема, видимо да је наш решавач (са побољшаним алгоритмом филтрирања) решио 404 инстанци укупно, што је најбољи резултат од свих решавача (`sugar` је решио 335 инстанци, `mistral` је решио 178 инстанци, а `opturion` је решио 379 инстанци укупно). Ово значи да је наш решавач укупно гледано најбољи, када су у питању ових пет проблема које смо разматрали.

4.5.5 Преглед и поређење са другим релевантним приступима

Постоји значајан број радова у којима се разматрају комбинације `alldifferent` и линеарног ограничења. Један такав приступ предложио је Регин [82], који је развио алгоритам за успостављање конзистентности домена над *ограничењем глобалне кардиналности са тежинама* (енгл. *global cardinality constraint with costs*). Ово крајње уопштено ограничење се може искористити за моделовање конјункције $a_1 \cdot x_1 + \ldots + a_n \cdot x_n \leq c \wedge \texttt{alldifferent}(x_1, \ldots, x_n)$. Два су основна недостатка овог приступа. Прво, на овај начин се може представити само комбинација једног `alldifferent` ограничења и једног линеарног ограничења чији скупови променљивих морају да се поклапају. Други недостатак је у сложености алгоритма, с обзиром на то да се разматра знатно општији тип ограничења. Други занимљив приступ дат је у већ поменутом раду Белдицеана и других [13]. Аутори презентују алгоритам који успоставља конзистентност граница над конјункцијом $x_1 + \ldots + x_n \leq c \ \wedge \ \texttt{alldifferent}(x_1, \ldots, x_n)$ и који има временску сложеност $O(n \log(n))$. Алгоритам је, заправо, нешто општији, јер се ограничење $x_1 + \ldots + x_n \leq c$ може заменити, на пример, ограничењем $x_1 \times \ldots \times x_n \leq c$, или $x_1^2 + \ldots + x_n^2 \leq c$. Са друге стране, нису дозвољени произвољни коефицијенти у суми, попут ограничења $a_1 \cdot x_1 + \ldots + a_n \cdot x_n \leq c$. Као и у претходном приступу, овде се такође захтева да се скуп променљивих које се појављују у линеарном ограничењу поклапа са скупом променљивих `alldifferent` ограничења (или, макар, да буде његов подскуп). У том специјалном случају, алгоритам који су развили Белдицеану и други [13] се понаша значајно боље од Региновог алгоритма [82], упркос чињеници да је успостављена само конзистентност граница [13]. У поређењу са ова два приступа, наш алгори-

там је општији, када је у питању комбиновање `alldifferent` и линеарног ограничења. Прво, наш алгоритам допушта произвољне коефициенте (позитивне и негативне). Друго, нисмо ограничени на једно `alldifferent` ограничење, нити морамо имати скуп променљивих у `alldifferent` ограничењу који се поклапа са променљивама у линеарној суми. Другим речима, наш алгоритам допушта вишеструка `alldifferent` ограничења која се могу делимично преклапати са променљивама линеарног ограничења. Сложеност нашег алгоритма је такође $O(n\log(n))$, што га чини прилично ефикасним. Са друге стране, наш алгоритам не успоставља конзистентност граница (ни домена) над конјункцијом `alldifferent` и линеарног ограничења. Наш алгоритам само чини пропагацију ограничења нешто јачом, у поређењу са стандардним приступом, када се оба ограничења разматрају засебно.

Други могући правац истраживања је да се разматрају технике моделовања које омогућавају постојећим CSP решавачима да боље ухвате интеракцију између `alldifferent` и линеарних ограничења. Главна идеја је да се приликом кодирања проблема додају нова имплицирана линеарна ограничења којима се дефинишу границе за суме променљивих које су последица `alldifferent` ограничења. Након тога, друге технике трансформације, попут *елиминације заједничких подизраза* могу бити примењене. На пример, претпоставимо да имамо следећи скуп ограничења: $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 \leq 15$, `alldifferent`$(x_1, x_2, x_3)$, `alldifferent`$(x_5, x_6, x_7)$ (све променљиве имају домен $\{1, \ldots, 10\}$). Из првог `alldifferent` ограничења можемо закључити да је $x_1 + x_2 + x_3 \geq 6$ и $x_1 + x_2 + x_3 \leq 27$ (слично је и са другим `alldifferent` ограничењем). Даље уводимо нове променљиве, $y = x_1 + x_2 + x_3$ и $z = x_5 + x_6 + x_7$ (чији су домени $\{6, \ldots, 27\}$), а затим презаписујемо оригинално ограничење као $y + x_4 + z \leq 15$. Коришћењем овако модификованог кодирања, пропагација ограничења је нешто јача. Заиста, у оригиналном кодирању, стандардни алоритам филтрирања за линеарно ограничење може једино да закључи да је $x_i \leq 9$ (за $i \in \{1, \ldots, 7\}$). У модификованом кодирању алгоритам филтрирања за презаписано линеарно ограничење најпре закључује да је $x_4 \leq 3$, $y \leq 8$, $z \leq 8$. Након тога, из $y = x_1 + x_2 + x_3$ следи да је $x_1 \leq 6$, $x_2 \leq 6$ и $x_3 \leq 6$. Један начин да се искористи ова идеја је да се ове технике ручно инкорпорирају у кодирање неког конкретног проблема. На пример, у раду Хелмута Симониса [90] се разматра специјални случај поменутог какуро проблема. Слично, поједини аутори разматрају примену описаних техника на познати проблем *Го-*

ломбовог лењира [93, 41]. У општем случају, можемо развити алгоритам који трансформише кодирање проблема аутоматски, што чини метод примењивим на знатно широј класи проблема. На пример, у раду Најтингела и других [76] се описује алгоритам за елиминацију заједничких подизраза. Аутори показују да се коришћењем ове трансформације у конјункцији са додавањем линеарних ограничења којима се представљају границе изведене из `alldifferent` ограничења могу значајно поправити резултати за инстанце *Killer Sudoku* проблема. Слична идеја се користи у раду Фришча и других [40], где је представљен систем заснован на правилима за трансформацију CSP проблема. У поређењу са описаним техникама моделовања, наш алгоритам може успоставити јачи ниво конзистентности. У претходном примеру, рецимо, наш алгоритам би (примењен на оригинално кодирање) закључио да је $x_4 \leq 3$, $x_1 \leq 5$, $x_2 \leq 5$ и $x_3 \leq 5$.

Најзад, идеја комбиновања различитих типова глобалних ограничења у циљу побољшања пропагације је такође примењена и на нека друга ограничења. На пример, у раду Хнича и других [51] се разматра комбинација *ограничења лексикографског уређења* (енгл. *lexicographic ordering constraint*) са два линеарна ограничења. Други занимљив случај је разматран у раду Бесиера и других [15], где аутори разматрају комбинацију `alldifferent` ограничења и *ограничења приоритета* (енгл. *precedence constraint*).

4.6 Закључак

У овом делу тезе уведена је нова SMT теорија у оквиру које је дефинисана семантика неких значајних глобалних ограничења (у текућој верзији решавача, подржана су линеарна и `alldifferent` ограничења). Ова теорија омогућава изражавање CSP проблема користећи подржана глобална ограничења, тј. без свођења ових ограничења на језик линеарне аритметике, чиме би се изгубила природна структура проблема. Теоријски решавач за ову теорију се заснива на постојећим алгоритмима филтрирања за одговарајућа глобална ограничења. Ови алгоритми су на одговарајући начин адаптирани тако да се могу уклопити у постојеће SMT окружење. Ово подразумева надоградњу алгоритама тако да имају могућност генерисања објашњења. Такође, посебна пажња у имплементацији теоријског решавача посвећена је проблему комуникације између глобалних ограничења која се остварује разменом литерала помоћу којих се изражавају промене у доменима CSP променљивих. Ова размена се обавља преко

трага SAT решавача.

У оквиру руковаоца `alldifferent` ограничењем имплементирани су Форд-Фулкерсонов алгоритам за проверу конзистентности, Регинов алгоритам за успостављање конзистентности домена, као и Катсирелосов алгоритам за генерисање објашњења. Поред тога, осмишљен је и имплементиран нови алгоритам за генерисање објашњења за `alldifferent` ограничење. Алгоритам решава проблем генерисања објашњења свођењем на MOS проблем који је такође дефинисан у оквиру ове главе тезе, а дат је и алгоритам за његово решавање. Предложени алгоритам је упоређен са стандардним алгоритмима који се у CSP решавачима користе у ову сврху (Катсирелосов алгоритам и алгоритми засновани на проточним мрежама). Дата је и експериментална евалуација која показује да је наш алгоритам бољи од Катсирелосовог алгоритма на проблемима у којима доминира `alldifferent` ограничење. Такође је доказана коректност предложеног алгоритма за генерисање објашњења.

У оквиру руковаоца линеарним ограничењем имплементиран је стандардни алгоритам филтрирања који успоставља конзистентност граница над линеарним ограничењем. Поред тога, осмишљен је и имплементиран нови побољшани алгоритам филтрирања који узима у обзир присуство `alldifferent` ограничења у датом проблему. Ово често омогућава да се израчунају јаче границе за променљиве линеарног ограничења, што доводи до додатног сужавања домена променљивих. Овако нешто може бити корисно код CSP проблема који укључују велики број линеарних сума које су додатно ограничене `alldifferent` ограничењима. Тестирали смо наш приступ на пет таквих проблема и упоредили га са другим релевантним приступима, као и са неким актуелним решавачима. Експерименти су потврдили ефективност предложеног унапређења. Главна предност нашег приступа, у поређењу са другим релевантним приступима, је у широј примењивости, с обзиром на то да допушта произвољне коефициенте у линеарним изразима и даје могућност комбиновања линеарног ограничења са више `alldifferent` ограничења која се делимично преклапају са скупом променљивих тог линеарног ограничења.

У оквиру споменутих експерименталних евалуација, такође смо упоредили наш решавач са другим актуелним решавачима. Резултати показују да се коришћењем SMT решавача који је опремљен алгоритмима филтрирања за глобална ограничења добијају резултати који су упоредиви са другим приступима, када је у питању решавање CSP проблема. Ово је веома значајно, с обзиром на то да

је тиме показан потенцијал SMT решавача за решавање проблема изван оквира уобичајене примене у области верификације.

Даљи рад у овој области подразумева истраживање других могућности интеграције SMT и CSP технологија. Први циљ је свакако укључивање алгоритама филтрирања за друга често коришћена глобална ограничења у наш SMT решавач, што може проширити његову примењивост у решавању CSP проблема. Када је у питању `alldifferent` ограничење, имплементација алгоритама филтрирања који успостављају конзистентност граница и њихово укључивање у наш SMT решавач може такође побољшати перформансе нашег решавача када је у питању решавање неких CSP проблема. Ови алгоритми су по правилу мање сложености, а пропагацијска моћ им је слабија од Региновог алгоритма само у случају да постоје „рупе" у доменима променљивих, што често није случај. Ово може имати и двоструки значај, зато што се алгоритми за успостављање конзистентности граница лакше могу адаптирати тако да подржавају бесконачне или веома велике коначне домене, што можда може бити корисно и у решавању стандардних SMT проблема. У том правцу, потенцијални допринос SMT решавању може бити комбиновање предложене CSP теорије са другим типичним SMT теоријама, попут теорије неинтерпретираних функцијских симбола, теорије низова и сл.

Глава 5

Унапређивање SMT решавача техникама паралелизације

5.1 Увод

Последњих година, фокус многих истраживача премешта се ка развоју паралелних решавача, како би се искористиле могућности вишепроцесорских и вишејезгарних рачунара који су постали широко доступни. Већина модерних SAT и SMT решавача је заснована на DPLL алгоритму [30], што значи да обилазе простор претраге помоћу гранања и враћања уназад. Због тога су први покушаји паралелизације углавном били засновани на подели простора претраге, након чега се добијене гране простора претраге обилазе паралелно [52]. Овај приступ се често зове *завади-па-владај*. Други приступ је да се више инстанци SAT или SMT решавача покрену паралелно независно једна од друге, са различитим подешавањима (нпр. са различитим параметрима, хеуристикама, семенима за генерисање случајних бројева, и сл.) све док једна од њих не пронађе решење. Комуникација између инстанци решавача је обично минимална и укључује размену неких од научених клауза (обично оних краћих). Овај приступ је познат као *паралелни портфолио*. Још једна, сасвим другачија идеја је да се паралелизују алгоритми који проверавају задовољивост и детектују пропагације у току обиласка простора претраге, а који могу бити веома скупи. Један пример таквог алгоритма је алгоритам који се користи за детекцију конфликтних клауза и јединичних пропагација заснован на *шеми два посматрана литерала* (енгл. *two-watched-literals algorithm*) [68] који узима више од 80% времена извршавања у модерним SAT решавачима. Други пример је *симплекс*

алгоритам [37] који је један од најскупљих алгоритама у модерним SMT решавачима. Паралелизација алгоритма два посматрана литерала се разматра у раду Норберта Мантија [63]. У овој глави разматра се паралелизација симплекс алгоритма у оквиру SMT решавача.

Симплекс алгоритам је веома важан део модерних SMT решавача, с обзиром на то да се користи за испитивање задовољивости конјункција линеарних аритметичких ограничења (над целим или реалним бројевима). Оригинални симплекс алгоритам који је развио Данциг [29] се користи у *линеарном програмирању* (енгл. *linear programming* (LP)) за проналажење решења које оптимизује дату линеарну функцију задовољавајући притом дати скуп линеарних ограничења. Процедура која се користи у SMT решавачима [37] је упрошћена варијанта оригиналног алгоритма која разматра једино задовољивост датог скупа линеарних ограничења (тј. не врши се оптимизација, већ се тражи било које решење). Међутим, специфична природа проблема који се разматрају у SMT-у намеће додатне захтеве по питању имплементације симплекс процедуре у оквиру SMT решавача. Док су проблеми који се разматрају у линеарном програмирању обично задовољиви (при чему обично постоји велики број могућих решења), а циљ је пронаћи оптимално решење, у SMT-у су проблеми често незадовољиви, а циљ је да се испита задовољивост проблема са апсолутном поузданошћу. Ове разлике у циљевима намећу и различите захтеве у имплементацији: док LP решавачи могу користити уграђену хардверску аритметику у покретном зарезу (енгл. *floating point arithmetic* (FPU)) за сва потребна израчунавања, код SMT решавача је често неопходно[1] користити *аритметику произвољне прецизности* у оквиру симплекс алгоритма како би се гарантовао добијени резултат (који је *sat* или *unsat*). Аритметика произвољне прецизности се емулира у оквиру софтвера и зато је обично веома скупа у погледу временске сложености, што чини симплекс једним од најскупљих алгоритама у оквиру SMT-а.

Стандардни симплекс алгоритам допушта паралелизацију на сасвим природан начин: с обзиром на то да се операције овог алгоритма обично извршавају независно над свим врстама *симплекс таблоа*, оне се могу извршавати паралелно. Ипак, треба имати у виду да LP проблеми обично подразумевају веома

[1]Због ефикасности, већина модерних SMT решавача користи аритметику произвољне прецизности само када је то заиста неопходно, тј. уграђена хардверска аритметика се користи докле год се не догоди прекорачење, након чега се прелази на аритметику произвољне прецизности.

велике и веома ретке таблое, па овај наивни приступ паралелизацији обично не може да надмаши специјализоване секвенцијалне алгоритме који су прилагођени ретким матрицама [49]. Таблои који се разматрају у SMT-у су такође ретки у већини случајева (мада не увек), али када се користи аритметика произвољне прецизности, чак и овакав праволинијски приступ паралелизацији може да буде ефикасан. Ово је свакако случај са SMT инстанцама које имају густе таблое, јер током решавања оваквих инстанци обично добијамо веома велике коефициенте у таблоу, што повећава сложеност израчунавања. У овој тези разматрамо само ову врсту паралелизације симплекс алгоритма и то само у оквиру модела паралелног израчунавања са *дељеном меморијом*.

Циљ овог дела тезе је да се истраже могућности и ефективност паралелизације симплекс алгоритма на различитим скуповима SMT инстанци које укључују линеарну аритметику. Такође желимо да упоредимо овај приступ са другим приступима паралелизацији, пре свега са паралелним портфолиом. Коначно, желимо да размотримо и хибридни приступ — паралелизацију симплекс алгоритма у комбинацији са паралелним портфолиом. Иако симплекс паралелизација можда неће бити веома ефикасна у општем случају, ипак очекујемо да ће имати позитиван ефекат на инстанце које укључују велики број израчунавања у оквиру симплекс алгоритма. Конкретно, ово може бити случај са класама инстанци које имају велике и густе таблое, где се највећи део времена извршавања троши у симплекс алгоритму. Такође можемо очекивати да хибридни приступ буде користан на рачунарима са великим бројем процесорских језгара. На пример, ако рачунар има 16 процесорских језгара, можемо покренути четири решавача у паралелном портфолију, при чему сваком од решавача дајемо по четири језгра за интерну паралелизацију симплекс алгоритма. На овај начин сваки од решавача у портфолију се може убрзати, што ће убрзати и укупно извршавање паралелног портфолија. Хибридни приступ може искористити процесорске ресурсе на бољи начин, с обзиром на то да упошљава већи број језгара. У овој тези покушавамо да пронађемо одговоре на сва ова питања.

Два су основна доприноса овог дела тезе. На првом месту, детаљно објашњавамо како се симплекс алгоритам који се користи у оквиру SMT решавача [37] може адаптирати тако да се извршава паралелно. Други допринос је детаљна експериментална евалуација и поређење различитих приступа паралелизацији (симплекс паралелизација, паралелни портфолио и хибридни приступ) на различитим скуповима SMT инстанци (укључујући индустријске инстанце као и

вештачки генерисане инстанце). Сви тестови су извршавани на вишепроцесорском рачунару са дељеном меморијом. Такође је битно нагласити да су тестови покретани са различитим бројем *нити* (енгл. *threads*) у оквиру процеса (до 32) како бисмо видели како се различити приступи паралелизацији понашају када се број доступних процесора повећава. Резултати приказани у овој глави тезе су објављени у једном од радова аутора ове тезе [5].

5.1.1 Преглед релевантних резултата

Најпознатија процедура одлучивања за линеарну аритметику заснована на симплекс алгоритму је описана у раду Дутертреа и де-Муре [37]. Ова процедура је даље унапређивана од стране више аутора, а напори су уложени да се побољша подршка за целобројну аритметику [48, 34], као и да се побољша ефикасност имплементације процедуре [47, 59]. Са друге стране, фрагменти линеарне аритметике попут *диференцне логике*[2] (енгл. *difference logic*) се могу решавати коришћењем ефикаснијих процедура одлучивања које нису засноване на симплексу, а једна од најпознатијих таквих процедура у оквиру DPLL($\mathcal{T}$) архитектуре је описана у раду Ниевенхуиса и Оливераса [72]. Такође постоје примери успешних аритметичких решавача који нису засновани на DPLL($\mathcal{T}$) архитектури. Један такав приступ је описан у раду Шеинија и Сакалаха [89]. Овај приступ је такође заснован на интеграцији са CDCL SAT решавачем и то на начин сличан DPLL($\mathcal{T}$) приступу. Решавач садржи у себи две процедуре за решавање аритметичких ограничења — једна је тесно интегрисана са SAT решавачем и бави се искључиво такозваним UTVPI[3] ограничењима, док је друга општа процедура заснована на симплексу која се покреће само када се формира потпуна исказна валуација. Други занимљив приступ за решавање проблема линеарне аритметике је дат у раду Јовановића и де-Муре [55]. Иако ова процедура није изграђена над CDCL SAT решавачем, она има доста сличности са CDCL приступом — укључује гранање, пропагације, анализу конфликата, учење и нехронолошко враћање уназад, али притом користи *равни одсецања* (енгл. *cutting planes*) уместо клауза.

[2]Диференцна логика је фрагмент линеарне аритметике у коме разматрамо искључиво литерале облика $x - y \bowtie c$, где је $\bowtie \in \{\leq, \geq, <, >, =, \neq\}$.

[3]UTVPI фрагмент линеарне аритметике (скраћено од *unit-two-variable-per-inequality*) је уопштење диференцне логике где су сви литерали облика $ax + by \bowtie c$, при чему је $a, b \in \{+1, -1\}$ и $\bowtie \in \{\leq, \geq, <, >, =, \neq\}$.

Када су у питању паралелни решавачи, у протеклих пар деценија је развијено много паралелних SAT решавача, као и мали број паралелних SMT решавача. Познати паралелни SAT решавачи који су засновани на приступу завади-па-владај су *PSATO* [105], *PaSAT* [92] и *PSatz* [56]. Са друге стране, познати решавачи који су засновани на паралелном портфолију су *ManySAT* [50] и *Plingeling* [16]. Детаљнији преглед паралелних SAT решавача се може наћи у литератури [91, 52]. Када је у питању паралелизација SMT решавача, паралелни портфолио је најпре разматран у раду Винтерштајгера и других [104], где се описује паралелизована верзија Z3 решавача [33]. Нешто касније, у оквиру решавача *Picoso* [57], разматрају се и завади-па-владај приступ и паралелни портфолио за паралелизацију SMT решавача.

Приступ заснован на паралелизацији временски сложених алгоритама за проверу задовољивости и детекцију пропагација у SAT и SMT решавачима је такође проучаван, али само у извесној мери. У раду Норберта Мантија [63], разматра се паралелизација детекције конфликата и јединичних пропагација која је заснована на поменутој шеми два посматрана литерала [68] у SAT решавачима. Скуп клауза се партиционише на дисјунктне подскупове, а затим се добијени подскупови клауза обрађују паралелно у оквиру посебних нити. Када је SMT у питању, идеја да се паралелизују комплексни и временски сложени алгоритми у оквиру SMT решавача се спомиње у оквиру *OpenSMT* [25] пројекта[4], али, бар колико је аутору ове тезе познато, ништа на ту тему до сада није објављено.

Најзад, паралелизација симплекс алгоритма у општем случају је веома добро обрађена тема у претходним деценијама. Добар преглед ове теме је дат у раду Хола [49]. Највећи напор у овој области је уложен у проучавање могућности ефикасне паралелизације у случају ретких таблоа за које постоје оптимизовани секвенцијални алгоритми који су веома ефикасни и самим тим их је веома тешко побољшати паралелизацијом. У овој тези не покушавамо да се поредимо са овим приступима, с обзиром на то да се у оквиру SMT-а разматрају проблеми сасвим другачије природе у поређењу са типичним LP проблемима, као што је већ раније споменуто.

[4] `http://www.inf.usi.ch/09-urop-11-sharygina-151196.pdf`

5.2 Симплекс алгоритам у SMT решавачима

Већина модерних SMT решавача користи теоријски решавач за линеарну аритметику који је заснован на симплекс алгоритму. Варијанта алгоритма која се најчешће користи у SMT-у (као и у решавачу `argosmt`) је описана у раду Дутертреа и де-Муре [37]. У овом поглављу укратко описујемо основну (секвенцијалну) верзију овог алгоритма, док у наредном поглављу описујемо начин на који је постигнута његова паралелизација. Основна варијанта овог алгоритма ради са линеарним ограничењима над рационалним бројевима. Претпостављамо да су сви литерали који се појављују у улазној формули у једној од следећих форми:

$$\sum_j a_j x_j \bowtie b \tag{5.1}$$

или

$$x \bowtie b \tag{5.2}$$

где су a_j и b елементи скупа рационалних бројева $\mathbb{Q}$, $\bowtie \in \{\leq, \geq\}$[5], а x и x_j су *непознате*[6] чије вредности (из $\mathbb{Q}$) које задовољавају формулу треба одредити. Литерале облика (5.1) зваћемо *полиномијалним литералима*, док ћемо литерале облика (5.2) звати *елементарним литералима*. За сваки полиномијални литерал уводи се нова непозната s, а сва појављивања тог литерала у формули се замењују елементарним литералом $s \bowtie b$ (са истом релацијом $\bowtie$ као и у оригиналном литералу). Најзад, литерал:

$$s = \sum_j a_j x_j$$

се додаје у формулу као јединична клауза. Након ове трансформације, формула више не садржи полиномијалне литерале, већ само елементарне литерале,

[5] У општем случају, скуп могућих релација је $\bowtie \in \{\leq, \geq, <, >, =, \neq\}$. Међутим, лако је видети да се симболи $\neq$ и $=$ могу једноставно елиминисати у фази претпроцесирања ($x = y$ се замењује са $x \leq y \wedge x \geq y$, а $x \neq y$ се замењује са $x < y \vee x > y$). Једини нетривијални део је елиминација строгих неједнакости. Ово се обично постиже разматрањем векторског простора $\mathbb{Q}_\delta$ парова рационалних бројева уместо поља $\mathbb{Q}$. За детаље, погледати рад Дутертреа и де-Муре [37].

[6] У улазној базној SMT формули, непознате су представљене неинтерпретираним константама сорте `Real` која се интерпретира скупом реалних бројева $\mathbb{R}$. Приметимо да у случају линеарне аритметике не постоји суштинска разлика између реалног и рационалног случаја, јер ће свака процедура која се заснива на четири основне рачунске операције увек проналазити рационална решења, с обзиром на то да су коефицијенти у изразима рационални бројеви.

као и једнакости у којима су неке непознате изражене као линеарне комбинације других непознатих (које зовемо *једнакосним литералима*). Једнакосни литерали се, притом, појављују само у јединичним клаузама.

Једнакосни литерали се одмах на почетку решавања јединичним пропагацијама постављају на траг решавача, након чега их теоријски решавач прихвата и интерно чува у облику матрице — *симплекс таблоа*. Непознате које су изражене помоћу ових линеарних комбинација зваћемо *базне* непознате, док ћемо непознате које се користе за изражавање базних непознатих звати *не-базне* непознате. Скупове базних и не-базних непознатих у таблоу ћемо означавати редом са B и N (приметимо да је $B \cap N = \varnothing$). Са друге стране, елементарни литерали се могу инкрементално наметати теоријском решавачу, као и поништавати, у случају враћања уназад. Елементарни литерали успостављају *границе* непознатих. За сваку непознату x њену доњу и горњу границу означавамо респективно са $l(x)$ и $u(x)$. Иницијално, $l(x) = -\infty$ и $u(x) = +\infty$, за сваку непознату x. Кад год се неки нови литерал наметне теоријском решавачу, одговарајућа граница се мења у складу са тим литералом. Када се примењује враћање уназад, границе се морају вратити на претходне вредности (из тог разлога морамо запамтити вредности граница приликом сваке примене правила `Decide`).

За сваку непознату x, такође одржавамо *вредност* (тј. интерпретацију) од x у текућем моделу теорије, означену са $\beta(x)$. Како би обезбедио да текући модел задовољава наметнуте литерале, решавач мора да обезбеди да за сваку непознату важи $l(x) \leq \beta(x) \leq u(x)$. За не-базне непознате, ова инваријанта се константно одржава. Кад год се граница неке не-базне непознате x промени (зато што је неки елементарни литерал наметнут теоријском решавачу), вредност $\beta(x)$ се мења уколико је потребно како би упала у интервал између граница (типично, поставља се управо на вредност границе која је претходно нарушена). Вредности базних непознатих које зависе од x се затим ажурирају на одговарајући начин. Са друге стране, ако је допустиви интервал постао празан након наметања нове границе (тј. $u(x) < l(x)$), теоријски решавач пријављује конфликт.

За базне непознате ова инваријанта се не одржава непрекидно. Уместо тога, одговарајућа процедура за проверу сагласности модела са наметнутим границама се покреће периодично. Ако се том приликом детектује базна непозната x_i чија вредност излази ван граница, процедура мора пронаћи не-базну непознату

x_j у једнакости којом је изражена непозната x_i у таблоу, такву да вредност од x_j у текућем моделу може бити промењена на одговарајући начин (тј. померена ка својој доњој или горњој граници), тако да вредност непознате x_i упадне у интервал одређен њеним границама. Ако таква непозната x_j постоји, операција коју зовемо *пивотирање* се примењује на табло: x_i постаје не-базна, а x_j постаје базна. Ово се постиже трансформисањем таблоа: узимамо једначину којом је изражена непозната x_i и користимо је да помоћу ње изразимо непознату x_j. Након тога, једноставно користимо добијену трансформисану једнакост да помоћу ње елиминишемо све појаве непознате x_j у другим једнакостима у таблоу (приметимо да скупови B и N остају дисјунктни). Након што су непознате x_i и x_j испивотиране, мењамо вредност непознате x_i тако што је поставимо на вредност претходно нарушене границе. Вредности базних непознатих се након тога мењају у складу са промењеном вредношћу сада не-базне непознате x_i. Описани корак пивотирања се понавља докле год постоји бар једна базна непозната x_i чија вредност је неконзистентна са својим границама, а за коју постоји не-базна непозната x_j таква да се x_i може пивотирати са x_j. Самим тим, на крају ове процедуре ћемо или пронаћи модел конзистентан са свим наметнутим границама, или ћемо идентификовати базну непознату чија се вредност не може променити тако да упадне у интервал између задатих граница (у ком случају пријављујемо конфликт у теорији).

Може се доказати [37] да се описана процедура увек зауставља, под условом да постоји унапред фиксиран потпуни поредак скупа непознатих који се поштује приликом избора непознатих x_i и x_j које се пивотирају. Ово значи да увек бирамо најмању базну непознату (у односу на тај фиксирани поредак) чија вредност нарушава задате границе, након чега увек бирамо најмању не-базну непознату међу оним које се могу пивотирати са изабраном базном непознатом. Ова страгетија је позната под називом *Бландово правило* (енгл. *Bland's rule*). Иако ова стратегија гарантује заустављање, она обично не води до брзе конвергенције, тако да се често разматрају други хеуристички приступи. На пример, приступ који се користи у решавачу MathSAT [47] ради на следећи начин: бирамо базну непознату такву да је апсолутна разлика између њене вредности и нарушене границе минимална. Након тога, бирамо не-базну непознату (међу оним које се могу пивотирати са изабраном базном непознатом) такву да се појављује најмањи број пута у осталим једнакостима (врстама) у таблоу. Ова страгетија не гарантује заустављање, па се зато описано правило

примењује само унапред задати број пута, након чега решавач прелази на Блендово правило. На основу резултата приказаних у литератури [47], коришћење овакве стратегије значајно побољшава перформансе симплекс алгоритма. Наш решавач `argosmt` такође користи ову стратегију.

Када је у питању целобројна аритметика (или у општијем случају, комбинација целобројне и реалне аритметике), све (или неке) непознате морају узимати целобројне вредности. Такве непознате називамо *целобројне непознате*. У том случају се обично покреће иста симплекс процедура која је претходно описана, како би се пронашао рационални модел. Ако не постоји рационални модел, тада се пријављује конфликт. У супротном, ако постоји бар једна целобројна непозната која у текућем рационалном моделу има не-целобројну вредност, тада модел мора бити промењен тако да се задовољи услов целобројности за ту непознату. Једна варијанта методе *гранања са одсецањем* (енгл. *branch-and-bound*) се најчешће користи у ову сврху. Илустрације ради, ако целобројна непозната x има вредност $\frac{5}{2}$ у текућем моделу, тада покушавамо да коригујемо модел тако што разматрамо два подслучаја: или је $x \leq 2$, или је $x \geq 3$. Ако постоји модел који задовољава било који од два подпроблема, тада ће то бити и модел за оригинални проблем. Овај метод често захтева велики број гранања по случајевима. Сам за себе, овај метод не гарантује заустављање процедуре, а такође обично не води до брзе конвергенције, тако да се најчешће комбинује са другим техникама (попут *Гоморијевих резова* (енгл. *Gomory cuts*) [37], решавања *Диофантових једначина* [48] и технике *пресека изведених из доказа* (енгл. *cuts-from-proofs*) [34]).

5.3 Паралелизација симплекс алгоритма

У овом поглављу разматрамо паралелизацију претходно описаног симплекс алгоритма, детаљно описујући начин на који су паралелизовани различити делови алгоритма. У тексту који следи подразумевамо да је базна непозната $x_i \in B$ изражена у таблоу помоћу линеарног полинома L_i, односно:

$$x_i = L_i(\mathbf{x}_N) = \sum_{x_j \in N} a_{ij} x_j$$

где је $\mathbf{x}_N = (x_j)_{x_j \in N}$ и a_{ij} су рационални коефицијенти.

Ажурирање вредности базних непознатих. Претпоставимо да је граница $u(x_j)$ не-базне непознате x_j промењена тако да је $u(x_j) < \beta(x_j)$. Како бисмо задовољили ову границу, вредност $\beta(x_j)$ треба поставити на $u(x_j)$. Последично, вредности базних непознатих које зависе од x_j морају бити ажуриране извршавањем следећег алгоритма:

$$\texttt{for\_each}\ x_i \in B\ :\ \beta(x_i) := \beta(x_i) + a_{ij}(u(x_j) - \beta(x_j));$$
$$\beta(x_j) := u(x_j);$$

Приметимо да се у свакој итерацији мења једино вредност $\beta(x_i)$. С обзиром на то да се ова вредност не користи ни у једној од наредних итерација, следи да се итерације горње петље могу извршавати паралелно:

$$\texttt{parallel\_for\_each}\ x_i \in B\ :\ \beta(x_i) := \beta(x_i) + a_{ij}(u(x_j) - \beta(x_j));$$
$$\beta(x_j) := u(x_j);$$

Притом се петља `parallel_for_each` извршава на следећи начин: скуп итерација се најпре партиционише на k дисјунктних подскупова (где је k број нити које су дате решавачу), а затим се сваки од подскупова итерација извршава у посебној нити. Овај модел израчунавања је имплементиран у многим популарним окружењима и библиотекама, па је овај вид паралелизације петље веома једноставно остварити. У горњој петљи, укупан број операција је $3m$, где је m број базних непознатих (имамо по једно сабирање, одузимање и множење за сваку од базних непознатих). Са k нити, свака од нити ће извршавати приближно $3m/k$ операција.

Ажурирање вредности пре пивотирања. Претпоставимо да су x_i и x_j, респективно, базна и не-базна непозната које су одабране за пивотирање. Као што је објашњено у поглављу 5.2, пивотирање се врши зато што је вредност базне непознате x_i нарушила једну од својих граница. Из тог разлога, x_i мора да постане не-базна (како не би била везана за вредност линеарног полинома), што нам омогућава да јој променимо вредност на произвољан начин. Изабрана небазна непозната x_j мења своје место са x_i и постаје базна. Вредност непознате x_i се поставља на претходно нарушену границу v, а вредност непознате x_j као и вредности осталих базних непознатих се ажурирају на основу нове вредности за x_i. Ово ажурирање вредности се обично обавља пре самог пивотирања [37],

следећим алгоритмом:

$$\begin{aligned}
&\theta := \tfrac{v-\beta(x_i)}{a_{ij}};\\
&\beta(x_i) := v;\\
&\beta(x_j) := \beta(x_j) + \theta;\\
&\texttt{for_each } x_k \in B \setminus \{x_i\} \ : \ \beta(x_k) := \beta(x_k) + a_{kj}\theta;
\end{aligned}$$

Поново, можемо приметити да се петља у последњој линији може паралелизовати:

$$\texttt{parallel\_for\_each } x_k \in B \setminus \{x_i\} \ : \ \beta(x_k) := \beta(x_k) + a_{kj}\theta;$$

У овој петљи, свака од k нити извршава $2m/k$ операција.

Пивотирање. Након што се вредности ажурирају, извршава се операција пивотирања. Као што је објашњено у поглављу 5.2, ово се ради тако што се искористи једнакост којом је x_i изражена у таблоу да се помоћу ње изрази x_j, а затим се тако трансформисана једнакост искористи за елиминацију непознате x_j у свим осталим једнакостима у таблоу. Претпоставимо да је базна непозната x_i представљена у следећем облику:

$$x_i = L_i(\mathbf{x}_N) = \left(\sum_{x_l \in N \setminus \{x_j\}} a_{il} x_l \right) + a_{ij} x_j = L_i^*(\mathbf{x}_N) + a_{ij} x_j$$

где $L_i^*(\mathbf{x}_N)$ означава суму $L_i(\mathbf{x}_N)$ из које је избачен израз $a_{ij}x_j$. Сада се не-базна непозната x_j може изразити на следећи начин:

$$x_j = \frac{1}{a_{ij}} \left(x_i - L_i^*(\mathbf{x}_N) \right)$$

Користећи горњу релацију, можемо трансформисати табло на следећим алгоритмом:

$$\begin{aligned}
&L_j(\mathbf{x}_{N^*}) := \tfrac{1}{a_{ij}} \left(x_i - L_i^*(\mathbf{x}_N) \right);\\
&L_i(\mathbf{x}_{N^*}) := x_i;\\
&\texttt{for_each } x_k \in B \setminus \{x_i\} \ : \ L_k(\mathbf{x}_{N^*}) := L_k^*(\mathbf{x}_N) + a_{kj} L_j(\mathbf{x}_{N^*});
\end{aligned}$$

Овде је $N^* = (N \cup \{x_i\}) \setminus \{x_j\}$ (тј. N^* је N након операције пивотирања). На крају се x_j премешта из N у B, а x_i из B у N. Поново можемо приметити да се трансформисање сваког од полинома L_k (за $x_k \in B \setminus \{x_i\}$) може изводити паралелно, тако да можемо заменити петљу у последњој линији горњег алгоритма са:

$$\texttt{parallel\_for\_each } x_k \in B \setminus \{x_i\} \ : \ L_k(\mathbf{x}_{N^*}) := L_k^*(\mathbf{x}_N) + a_{kj} L_j(\mathbf{x}_{N^*})$$

Размотримо сложеност трансформација полинома које се примењују у горњој петљи. Најпре се сваки од коефициената полинома $L_j(\mathbf{x}_{N^*})$ мора помножити са a_{kj}, а затим се добијени полином мора додати полиному $L_k^*(\mathbf{x}_N)$ (ово укључује сабирање одговарајућих коефициената ова два полинома). Ово захтева $2(m-1)n$ операција у најгорем случају, где су m и n, редом, бројеви базних и не-базних непознатих у таблоу. Сетимо се да се у многим случајевима ове операције морају извршавати у оквиру аритметике произвољне прецизности, тако да операција пивотирања може бити веома скупа. У пракси, сложеност зависи од броја коефициената различитих од нуле у полиномима L_k, тако да укупни број операција може бити значајно мањи од $2(m-1)n$ у случају ретких таблоа. Ово значи да, заједно са величином таблоа, густина таблоа такође утиче на сложеност поступка пивотирања и, самим тим, на ефективност паралализације симплекс алгоритма. Други веома битан чинилац је величина коефициената у таблоу. Ако су коефициенти мали, коришћење скупе аритметике произвољне прецизности може бити избегнуто, тако да ефективност паралелизације може бити мања у таквим случајевима.

Теоријске пропагације. Један тип пропагација које се користе у типичним аритметичким теоријским решавачима је заснован на детекцији *слабијих* ограничења. На пример, ако је наметнут литерал $x \leq c$, тада за свако $c_1 > c$, мора важити да је $x \leq c_1$. Сви овакви елементарни литерали се пропагирају. Ова врста пропагације је јефтина и нема је смисла паралелизовати. Други тип теоријских пропагација је заснован на *израчунавању граница* непознатих на основу једнакости таблоа. Наиме, нове границе неке непознате се могу израчунати коришћењем текућих наметнутих граница осталих непознатих које се појављују у некој конкретној једнакости у таблоу. На пример, ако имамо једнакост:

$$x_i = \sum_{x_j \in N} a_{ij} x_j$$

тада можемо израчунати горњу границу базне непознате x_i помоћу следећег израза:

$$\sum_{a_{ij}>0} a_{ij} u(x_j) + \sum_{a_{ij}<0} a_{ij} l(x_j)$$

Израз за доњу границу је сличан. Овако израчуната граница ће бити коначна ако и само ако су све релевантне наметнуте границе не-базних непознатих које се користе у тој једнакости коначне. На сличан начин можемо израчунати границе било које од не-базних непознатих које се појављују у горњој једнакости,

под условом да су релевантне наметнуте границе осталих не-базних непознатих, као и базне непознате x_i коначне. Оваква израчунавања граница се могу применити на сваку од једнакости у таблоу.

С обзиром на то да различите једнакости у таблоу могу имати заједничких не-базних непознатих, границе израчунате на основу једне једнакости могу утицати на израчунавања граница у другим једнакостима. Ово значи да бисмо можда морали да поновимо алгоритам више пута ако желимо да достигнемо неку фиксну тачку. Због једноставности, у нашој имплементацији се не покушава достизање такве фиксне тачке за границе непознатих. Уместо тога, свака од једнакости се разматра само једном, а у свакој од једнакости израчунавање нових граница се врши искључиво на основу текућих *наметнутих граница* за непознате. Другим речима, користе се оне границе које су успостављене наметнутим елементарним литералима пре него што је покренут алгоритам за детекцију пропагација. Ово значи да се новоизрачунате границе из једне једнакости не користе при израчунавању граница у другим једнакостима. Нове границе ће бити наметнуте онда када се сва израчунавања (коришћењем старих граница) заврше, јер тек тада пропагирамо одговарајуће елементарне литерале.

С обзиром на то да се свака базна непозната појављује у тачно једној једанкости у таблоу, њене границе се једноставно израчунавају користећи ту једнакост. Са друге стране, свака од не-базних непознатих се може појављивати у више једнакости, тако да морамо израчунати њене границе из сваке од тих једнакости и искористити оне које су најјаче. Самим тим, цео алгоритам је дефинисан на следећи начин:

$$\texttt{for\_each}\ x_i \in B\ :\ \texttt{calculate\_bounds}(e(x_i), bounds);$$
$$\texttt{propagate\_entailed\_literals}(bounds);$$

где је $e(x_i)$ једнакост која изражава непознату x_i у таблоу, а *bounds* је аргумент који се преноси по референци[7] и који представља структуру података која чува израчунате горње и доње границе за сваку од непознатих (иницијално, свим непознатима су придружене бесконачне границе). Процедура `calculate_bounds`() најпре израчунава границе за све непознате које се појављују у једнакости $e(x_i)$, као што је претходно описано. За сваку израчунату границу која је коначна и јача од текуће границе која је сачувана у *bounds*

[7]Структура података *bounds* се најефикасније може имплементирати као хеш мапа која придружује пар граница (доњу и горњу) свакој од непознатих.

структури врши се одговарајуће ажурирање те сачуване границе. Када се петља заврши, литерали који су имплицирани из израчунатих граница које су сачуване у *bounds* структури се пропагирају.

Испоставља се да је израчунавање граница за све непознате прилично скупа процедура. Израчунавање граница може захтевати множење и сумирање вредности у оквиру аритметике произвољне прецизности. Из тог разлога, ова процедура је добар кандидат за паралелизацију. На жалост, једноставни `parallel_for_each` приступ се овде не може применити, зато што би сви позиви процедуре `calculate_bounds`() користили исту инстанцу *bounds* структуре, па би синхронизација између нити била неопходна. Решење које се може применити је *паралелна редукција*:

$$\texttt{parallel\_reduce}\ x_i \in B\ :\ \texttt{calculate\_bounds}(e(x_i), bounds);$$

Редукција функционише на следећи начин. Скуп једнакости у таблоу се подели на две или више партиције (у зависности од броја доступних нити). Свака нит процесира једну од тих партиција позивањем процедуре `calculate_bounds`() за сваку од једнакости у партицији. Најбитнији детаљ на који треба обратити пажњу је да свака од нити има своју приватну инстанцу *bounds* структуре која се предаје свим позивима процедуре `calculate_bounds`() у оквиру те нити. На овај начин, нити се могу извршавати независно, без потребе за синхронизацијом. Када две нити заврше свој посао, њихови *bounds* објекти се морају *спојити* у један објекат тако што се комбинују одговарајуће израчунате границе из оба објекта, при чему се увек бира она граница која је јача. Ову операцију спајања обавља једна од те две нити. Процес се наставља докле год све нити не заврше свој посао, а њихови *bounds* објекти не буду спојени у један.

Приметимо да ће паралелно извршавање претходно описаног алгоритма пропагације увек бити детерминистичко, тј. неће зависити од случајног поретка извршавања операција од стране различитих нити (енгл. *race conditions*). Наиме, процедура `calculate_bounds`() користи само претходно наметнуте границе, тако да не зависи од новоизрачунатих граница које су добијене у другим нитима. Одговарајући литерали се пропагирају секвенцијално, након што су све границе већ израчунате. Самим тим, резултат паралелног извршавања алгоритма је увек идентичан резултату секвенцијалног израчунавања.

5.4 Имплементација теоријског решавача

У овом поглављу су дати детаљи имплементације теоријског решавача за линеарну аритметику у оквиру `argosmt` решавача. Као и сви остали теоријски решавачи `argosmt` решавача, овај теоријски решавач имплементира интерфејс дат на слици 3.2 у поглављу 3.7. Решавач је заснован на алгоритмима описаним у претходним поглављима. Оптимизован је за реалну (рационалну) линеарну аритметику, уз ограничену подршку за целобројну линеарну аритметику. Решавач подржава симплекс паралелизацију, на начин описан у овој глави тезе.

Аритметички теоријски решавач користи GMP библиотеку за израчунавања произвољне прецизности (енгл. *GNU multiprecision library* (GMP))[8] у оквиру аритметичког теоријског решавача. Због ефикасности, GMP се користи само када је то заиста неопходно. Другим речима, решавач користи хардверске типове података докле год је то могуће, а прелази на GMP када се детектује потенцијално прекорачење (слична оптимизација је имплементирана у оквиру MathSAT решавача [47]). Стратегије за избор базне и не-базне непознате приликом пивотирања у аритметичком решавачу су такође позајмљене од MathSAT решавача (као што је описано у поглављу 5.2). Када су у питању целобројне непознате, наш решавач имплементира само наивну стратегију засновану на гранању са одсецањем (описану у поглављу 5.2). Свако гранање по случајевима се изражава као клауза и предаје се SAT решавачу (применом правила $\texttt{IntroduceAtom}_i$ и $\texttt{TheoryLemma}_i$) који даље грана по литералима у клаузи.

Теоријски решавач подржава два дедуктивна слоја. У нижем слоју се врши провера граница не-базних непознатих, као и ажурирање одговарајућих вредности по потреби. Такође се врши пропагација слабијих ограничења која је једноставна и јефтина. Са друге стране, скупе процедуре попут провере граница базних непознатих и алгоритма пивотирања, као и пропагације на основу израчунатих граница се обављају у оквиру вишег дедуктивног слоја, који се покреће само периодично (фреквенција покретања овог слоја аритметичког теоријског решавача се може одабрати приликом покретања решавача).

Структуре података аритметичког теоријског решавача у оквиру `argosmt` решавача су осмишљене тако да допуштају паралелизацију на начин описан у поглављу 5.3. Ово значи да је имплементација таблоа *независна по врстама* (енгл. *row-independent*). Ово значи да је решавач у могућности да обавља опе-

[8] `http://www.gmplib.org`

рације над више врста паралелно. Базне непознате се чувају у вектору, а све петље описане у поглављу 5.3 заправо итерирају кроз овај вектор. Свака једнакост (врста) у таблоу којом се изражава нека базна непозната x_i је имплементирана као хеш мапа која је придружена непознатој x_i и која повезује не-базне непознате које се појављују у тој једнакости са одговарајућим коефициентима (само коефициенти различити од нуле се чувају у мапи). Ово значи да можемо ефикасно итерирати кроз сваку врсту, проверити да ли се нека не-базна непозната појављује у датој врсти, као и очитати коефициент придружен тој не-базној непознатој. Са друге стране, имплементација таблоа не зна ништа о колонама, тј. различита појављивања једне исте не-базне непознате x_j у различитим врстама нису међусобно повезана ни на који начин. Због тога не можемо ефикасно итерирати кроз колоне. Итерирање кроз колону која одговара некој не-базној непознатој x_j се може симулирати итерацијом кроз вектор свих базних непознатих (тј. кроз све врсте таблоа), при чему се прескачу оне врсте које не садрже непознату x_j (што се може ефикасно проверити). Оваква имплементација која је независна по врстама омогућава ефикасну паралелизацију, тако што се вектор базних непознатих подели на дисјунктне партиције и свака од њих се процесира од стране засебне нити. Приметимо да су описане структуре података веома сличне структурама које се користе у MathSAT решавачу [47]. Ово значи да постоје и актуелни SMT решавачи у оквиру којих би се могле применити технике паралелизације које су описане у овој тези. Са друге стране, решавачи чије имплементације таблоа нису независне по врстама не могу директно користити технике паралелизације описане у поглављу 5.3, без значајних промена у структурама података.

Као што је речено у глави 3, за потребе паралелизације користи се Интелова TBB[9] библиотека. Највећи број петљи се записује помоћу уграђеног `tbb::parallel_for` алгоритма који постоји у библиотеци и ради управо онако како је описано извршавање апстрактне `parallel_for_each` конструкције у поглављу 5.3. За паралелну редукцију (која се користи за паралелизацију израчунавања граница), библиотека обезбеђује алгоритам `tbb::parallel_reduce` који ради управо онако како је описано извршавање апстрактне `parallel_reduce` конструкције у поглављу 5.3.

Приликом враћања уназад, аритметички решавач мора бити у стању да за сваку од непознатих x_i врати границе $l(x_i)$ и $u(x_i)$ на претходне вредности које

[9] `http://www.threadingbuildingblocks.org`

су биле валидне на крају одговарајућег нивоа одлучивања. Из тог разлога се морају сачувати вредности граница на крају сваког нивоа одлучивања. Ипак, неке оптимизације су могуће. На пример, на сваком од нивоа одлучивања многе од граница се уопште не мењају. Било би изузетно неефикасно чувати све границе свих непознатих сваки пут када се нови ниво одлучивања успостави на трагу решавача и касније се враћати на те сачуване границе, иако су многе од њих заправо све време биле непромењене. Из тог разлога у нашем решавачу се користи техника *копирања при модификацији* (енгл. *copy-on-write*) за чување граница. Ово значи да се граница наслеђена из претходног нивоа одлучивања чува за потребе враћања уназад само онда када се заиста захтева промена њене вредности на текућем нивоу. Са друге стране, вредности непознатих $\beta(x_i)$ у текућем моделу се не морају ресетовати при враћању уназад, с обзиром на то да се том приликом границе $l(x_i)$ и $u(x_i)$ могу учинити само слабијим, а вредност $\beta(x_i)$ која је задовољавала јаче границе ће свакако задовољавати и слабије.

5.5 Паралелни портфолио и хибридни приступ

У овом поглављу разматрамо варијанте паралелног портфолија које ћемо користити за упоређивање (и за хибридизацију) са симплекс паралелизацијом која је описана у поглављу 5.3. Сви разматрани портфолији се састоје из n инстанци `argosmt` решавача који користе различита семена за генерисање случајних бројева који утичу на стратегију гранања. За сваки овакав портфолио разматрамо његову просту и кооперативну варијанту (тј. без и са разменом дељених клауза).

Када је доступан велики број процесора, може се разматрати *хибридни приступ*. У том случају се покреће паралелни портфолио са n решавача, при чему је у сваком од решавача у портфолију такође омогућена и симплекс паралелизација (на начин како је то описано у поглављу 5.3). Цео портфолио на располагању има m процесора, при чему је $m > n$, тако да се додатни процесори могу користити за паралелизацију симплекса (типично, $m = n \cdot k$, чиме је сваком решавачу у просеку дато k процесора за паралелизацију симплекса).

У општем случају, свака од конфигурација решавача ће бити означена паром (n, k), где је n број решавача у портфолију, а $k = m/n$, где је m укупан број процесора (нити) који су доступни решавачу. Конфигурације $(1, k)$ одго-

варају обичном решавачу са омогућеном паралелизацијом симплекса (без портфолија). Конфигурације $(n, 1)$ одговарају портфолију без симплекс паралелизације. Све остале конфигурације су хибридне конфигурације. Приметимо да хибридне конфигурације такође могу бити просте и кооперативне. У поглављу 5.6 презентујемо експерименталну евалуацију са различитим конфигурацијама код којих је укупан број доступних процесора $m \leq 32$.

5.6 Експериментална евалуација

У овом поглављу презентујемо експерименталне резултате везане за паралелизацију. Сви тестови су обављени на рачунару са четири AMD Opteron 6168 1.6GHz процесора са по 12 језгара (тј. 48 језгара укупно), и 94GB RAM-а. Главни циљ експерименталне евалуације је да се испита ефекат паралелизације симплекса на различитим скуповима инстанци. Други циљ је да се приступ заснован на симплекс паралелизацији упореди са паралелним портфолијом, као и да се истражи потенцијал хибридног приступа. Најзад, желимо и да упоредимо наш решавач са неким актуелним секвенцијалним SMT решавачима. Како бисмо постигли наведене циљеве, спровели смо експерименте са следећим решавачима:

- `argosmt` решавач[10] — у питању је наш решавач са различитим конфигурацијама (n, k), укључујући секвенцијални решавач (конфигурација $(1, 1)$), конфигурације са паралелним симплексом ($(1, k)$ конфигурације), прост и кооперативни паралелни портфолио ($(n, 1)$ конфигурације), као и хибридне конфигурације (просте и кооперативне).
- `mathsat5` — секвенцијални SMT решавач MathSAT5[11] [27].
- `cvc4` — секвенцијални SMT решавач CVC4[12] [8].

Тестови су спроведени над неколико скупова SMT инстанци из SMT-LIB[13] колекције, као и над неким случајно генерисаним густим инстанцама. Начин избора инстанци као и добијени резултати ће бити детаљније описани у тексту

[10] `http://www.matf.bg.ac.rs/~milan/argosmt-5.21/`
[11] `http://http://mathsat.fbk.eu/`
[12] `http://cvc4.cs.nyu.edu/web/`
[13] `http://www.smt-lib.org/`

који следи. Приликом тестирања користили смо одговарајуће временско ограничење по инстанци. Ако инстанца није решена у задатом временском ограничењу, тада смо узимали управо то временско ограничење као време решавања.

5.6.1 Случајно генерисане густе инстанце

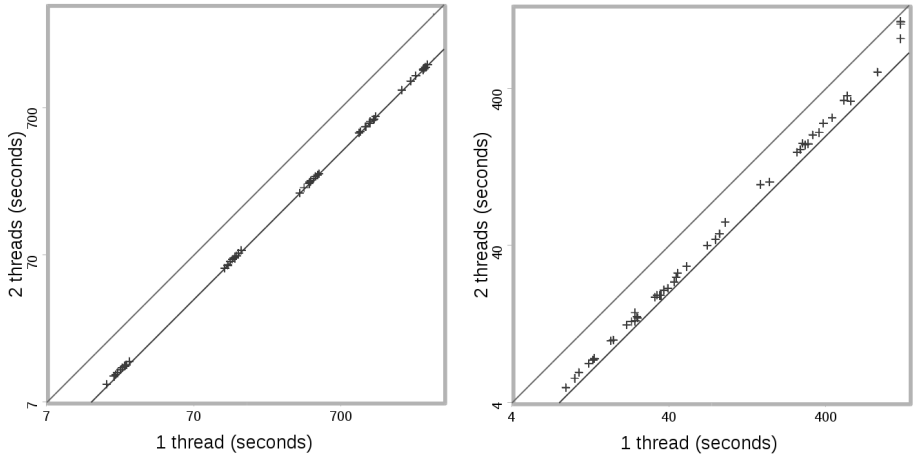
У овом одељку презентујемо резултате које смо добили применом симплекс паралелизације на два скупа случајно генерисаних *густих* инстанци. Први скуп (означен као *густи* `QF_LRA`) се састоји из 50 случајно генерисаних густих инстанци у теорији реалне линеарне аритметике. Ове инстанце су заправо генерисане као задовољиве конјункције линеарних једнакости, различитости и неједнакости, при чему свако од ових ограничења укључује све непознате дате инстанце. С обзиром на то да нема других исказних везника, ове инстанце имају прилично тривијалну исказну структуру, а с обзиром на то да су инстанце генерисане као густе, таблои су комплетно попуњени (тј. нема нултих коефицијената). Други скуп (означен као *густи* `QF_LIA`) је сасвим сличан, али овог пута је у питању *целобројна* линеарна аритметика. И у овом случају имамо 50 инстанци у скупу, при чему су све инстанце задовољиве. Иако су овакве густе инстанце прилично вештачке и немају практични значај, експериментална евалуација симплекс паралелизације на овако једноставним инстанцама може бити од користи при мерењу издвојеног утицаја симплекс паралелизације, изоловане од било ког другог утицаја који потиче од исказне структуре или структуре таблоа конкретне инстанце. Самим тим, убрзање[14] које се добије на овим инстанцама се може сматрати горњом границом убрзања које се може добити симплекс паралелизацијом.

Временско ограничење по инстанци је 1200 секунди за оба скупа. Резултати у табели 5.1 показују скоро линеарно убрзање на густим `QF_LRA` инстанцама, као и веома добро убрзање на густим `QF_LIA` инстанцама. Тако добри резултати се могу објаснити уделом времена извршавања које је проведено у паралелизованим деловима симплекс алгоритма: скоро 100% за `QF_LRA` и око 95% за `QF_LIA` инстанце. Ово је природна последица густине таблоа (с обзиром на то да су сви коефицијенти различити од нуле) и тривијалне исказне структуре улазне формуле (у случају `QF_LIA`, један мали део времена извршавања се троши на гранање по случајевима у SAT решавачу).

[14] *Убрзање* се дефинише као однос времена проведеног у секвенцијалном и паралелном извршавању.

		argosmt конфигурација					
		(1,1)	**(1,2)**	**(1,4)**	**(1,8)**	**(1,16)**	**(1,32)**
густи QF_LRA	**Решено (of 50)**	*50*	*50*	*50*	*50*	*50*	*50*
	Прос. време	820.0	410.9	210.8	115.6	63.7	36.2
густи QF_LIA	**Решено (of 50)**	*43*	*47*	*47*	*47*	*47*	*47*
	Прос. време	303.1	232.0	197.0	170.8	156.9	153.8

Табела 5.1: Резултати симплекс паралелизације на густим инстанцама добијени `argosmt` решавачем користећи различит број нити, тј. процесорских језгара (једна нит значи да нема паралелизације). Времена решавања су дата у секундама. Код нерешених инстанци за време решавања узима се задато временско ограничење по инстанци (1200 секунди).



Слика 5.1: Паралелизација симплекса на густим инстанцама (једна нит наспрам две нити): густи `QF_LRA` (лево), густи `QF_LIA` (десно)

Графици на слици 5.1 показују однос времена извршавања (једна нит наспрам две нити) на појединачним инстанцама за густе `QF_LRA` инстанце (леви график) и густе `QF_LIA` инстанце (десни график). Дијагонала представља однос 1 (тј. без убрзања), док поддијагонална линија представља убрзање 2 (тј. идеално убрзање са две нити). Као што је и очекивано, већина тачака је близу поддијагоналне линије (нарочито у случају густих `QF_LRA` инстанци, где скоро све инстанце имају скоро идеално убрзање). Најбоље убрзање за `QF_LRA` је 2.0, а за `QF_LIA` is 1.86.

5.6.2 SMT-LIB `QF_LRA` инстанце

Главни део ове експерименталне евалуације односи се на `QF_LRA` инстанце из различитих скупова који су преузети из SMT-LIB колекције инстанци. Иницијално, узето је 11 различитих скупова са укупно 785 инстанци. С обзиром на ограничене ресурсе са којима смо располагали, желели смо да одмах на почетку елиминишемо веома тешке инстанце које наш решавач вероватно не би могао да реши у реалном времену. Из тог разлога, прелиминарни тестови су спроведени помоћу `mathsat5` и `cvc4` решавача на свим инстанцама из ових скупова. На основу добијених резултата, изабрали смо оне и само оне инстанце које су оба решавача успешно решила за највише 10 минута (претпоставка је да су такве инстанце довољно лаке, тако да ће значајан део њих решити и наш решавач). Након тога, сви тестови су вршени искључиво над овако одређеним подскупом инстанци, а сви презентовани резултати у овом одељку се односе на тај подскуп. Приметимо да је у претходно описаном поступку одабира инстанци коришћен унапред фиксиран критеријум који не зависи од резултата који би били добијени нашим решавачем. Детаљи о одабраним инстанцама су сумирани у табели 5.2. Првих шест скупова представљају `QF_LRA` инстанце, док су преосталих пет скупова заправо `QF_RDL` инстанце (тј. инстанце *реалне диференцне логике*, која је, синтаксно гледано, фрагмент од `QF_LRA`). Разумно је очекивати да ће специфична структура `QF_RDL` инстанци индуковати другачију структуру таблоа, у поређењу са обичним `QF_LRA` инстанцама. Ово може довести до другачијег понашања симплекс паралелизације на таквим инстанцама.[15] Ово је главни разлог зашто су и ове инстанце узете у разматрање у овој експерименталној евалуацији.

Први тестови су спроведени коришћењем секвенцијалне верзије нашег `argosmt` решавача. Резултати су упоређени са актуелним секвенцијалним решавачима `mathsat5` и `cvc4` (табела 5.3). Коришћено је временско ограничење од 1800 секунди по инстанци. Може се видети да наш решавач, иако свакако није ефикасан као друга два решавача, ипак успева да реши значајан проценат изабраних инстанци (око 69%), што га чини упоредивим са актуелним решавачима. Решавач је око шест пута спорији од `cvc4` решавача, али ово је делом и због тога што је временско ограничење од 1800 секунди коришћено као време

[15] Занимљиво је да, на основу резултата које смо добили из експеримената, таблои који одговарају `QF_RDL` инстанцама заправо нису ништа ређи у просеку од таблоа који су придружени `QF_LRA` инстанцама.

Скуп инстанци	Укупно инстанци	Изабрано инстанци	SMT-LIB категорија
clock_synchro	36	36	QF_LRA
cooperatingt2	219	113	QF_LRA
latendresse	18	10	QF_LRA
miplib	42	16	QF_LRA
svcomp	94	43	QF_LRA
ultimate	123	60	QF_LRA
sal	60	54	QF_RDL
scheduling	106	58	QF_RDL
skdmxa2	32	27	QF_RDL
skdmxa	4	3	QF_RDL
temporal	51	45	QF_RDL
УКУПНО	**785**	**465**	

Табела 5.2: Сумарни приказ изабраних SMT-LIB QF_LRA инстанци

решавања за инстанце које наш решавач није могао да реши. Просечно време извршавања на решеним инстанцама је 275.9 секунди, што је само око два пута спорије од `cvc4` решавача, а само око три пута спорије од `mathsat5` решавача.

Тестирање симплекс паралелизације. Тестирање симплекс паралелизације је спроведено са различитим бројем нити које су биле доступне решавачу (са 2, 4, 8, 16 и 32 нити). Вишенитни решавач је покретан са истим семеном за генерисање случајних бројева у стратегијама гранања, тако да се при решавању понаша идентично секвенцијалном решавачу, тј. прати потпуно исту путању претраге у процесу решавања. Временско ограничење по инстанци је 1800 секунди. Добијени резултати су дати у табели 5.4.

Резултати у табели 5.4 показују да симплекс паралелизација даје значајно убрзање у просеку на `QF_LRA` инстанцама, иако убрзање варира на различитим скуповима инстанци. Табела 5.5 приказује просечно убрзање за сваки од скупова инстанци (једна нит наспрам две нити). Просечно убрзање на свим инстанцама је 1.26, док је максимално убрзање 1.74. Табела такође приказује просечан проценат времена које је проведено у паралелизованим деловима алгоритма. Из тих података се јасно види да већи проценат времена проведеног у паралелизованим деловима алгоритма доводи и до већег убрзања. На пример, `clock_synchro` и `latendresse` скупови имају добра просечна убрзања, с обзиром на то да је просечно време проведено у паралелизованим петљама на овим скуповима 96% и 87%, респективно. На другој страни, инстанце скупа `sal` немају готово никакво убрзање, с обзиром на то да је удео времена проведеног

		mathsat5	cvc4	argosmt
clock_synchro	**Решено (од 36)**	*36*	*36*	*28*
	Прос. време	44.4	50.1	479.0
cooperatingt2	**Решено (од 113)**	*113*	*113*	*59*
	Прос. време	180.4	233.0	1130.8
latendresse	**Решено (од 10)**	*10*	*10*	*8*
	Прос. време	1.2	11.5	362.7
miplib	**Решено (од 16)**	*16*	*16*	*10*
	Прос. време	97.7	69.6	725.0
svcomp	**Решено (од 43)**	*43*	*43*	*21*
	Прос. време	80.2	237.6	1159.0
ultimate	**Решено (од 60)**	*60*	*60*	*37*
	Прос. време	108.7	171.2	979.0
sal	**Решено (од 54)**	*54*	*54*	*40*
	Прос. време	50.2	45.1	503.4
scheduling	**Решено (од 58)**	*58*	*58*	*54*
	Прос. време	7.4	56.5	331.1
skdmxa2	**Решено (од 27)**	*27*	*27*	*19*
	Прос. време	149.0	127.4	974.4
skdmxa	**Решено (од 3)**	*3*	*3*	*2*
	Прос. време	93.2	30.1	718.9
temporal	**Решено (од 45)**	*45*	*45*	*45*
	Прос. време	14.8	2.3	22.9
УКУПНО	**Решено (од 465)**	***465***	***465***	***323***
	Прос. време	**89.5**	**126.9**	**741.3**
	На решеним	**89.5**	**126.9**	**275.9**

Табела 5.3: Резултати на изабраним SMT-LIB `QF_LRA` инстанцама, коришћењем секвенцијалних решавача `mathsat5`, `cvc4` и нашег решавача `argosmt`. Времена решавања су дата у секундама. Код нерешених инстанци за време решавања узима се задато временско ограничење по инстанци (1800 секунди). Просечно време на решеним инстанцама је такође наведено.

у паралелизованим петљама свега 17% у просеку.

Претходна опсервација је природна последица *Амдахловог закона* [1]. Наиме, ако је удео времена извршавања које је проведено у паралелизованим деловима алгоритма једнак p, тада је теоријска горња граница убрзања (са две нити) једнака $2/(2-p)$. Убрзање је обично и мање од тога, због додатних трошкова које уноси вишенитно извршавање. На слици 5.2 се могу видети детаљи о убрзању на појединачним инстанцама (једна нит наспрам две нити). Леви график приказује однос између времена извршавања. Главна дијагонала представља однос 1 (тј. нема убрзања), док поддијагонална линија представља однос 2 (тј. идеално убрзање). Крстићи представљају појединачне инстанце. Приказане су све инстанце из свих скупова за чије је решавање било потребно

		argosmt конфигурација					
		(1,1)	**(1,2)**	**(1,4)**	**(1,8)**	**(1,16)**	**(1,32)**
clock_synchro	**Решено (од 36)**	*28*	*28*	*29*	*30*	*31*	*31*
	Прос. време	479.0	448.2	433.6	405.0	383.0	401.9
cooperatingt2	**Решено (од 113)**	*59*	*67*	*71*	*73*	*75*	*75*
	Прос. време	1130.8	1065.1	1016.7	980.7	928.5	924.5
latendresse	**Решено (од 10)**	*8*	*8*	*8*	*8*	*8*	*8*
	Прос. време	362.7	361.7	361.3	361.4	361.5	361.5
miplib	**Решено (од 16)**	*10*	*10*	*10*	*10*	*10*	*10*
	Прос. време	725.0	727.0	728.0	727.2	729.0	733.6
svcomp	**Решено (од 43)**	*21*	*22*	*22*	*23*	*25*	*25*
	Прос. време	1159.0	1099.6	1067.9	1034.5	1014.2	1001.9
ultimate	**Решено (од 60)**	*37*	*40*	*40*	*41*	*41*	*41*
	Прос. време	979.0	914.6	881.9	857.0	821.9	819.3
sal	**Решено (од 54)**	*40*	*40*	*40*	*40*	*40*	*40*
	Прос. време	503.4	503.5	504.0	504.3	505.2	506.7
scheduling	**Решено (од 58)**	*54*	*54*	*54*	*54*	*54*	*54*
	Прос. време	331.1	272.5	256.4	263.3	307.7	326.9
skdmxa2	**Решено (од 27)**	*19*	*21*	*21*	*21*	*21*	*21*
	Прос. време	974.4	901.0	833.3	822.7	770.0	772.5
skdmxa	**Решено (од 3)**	*2*	*2*	*2*	*2*	*2*	*2*
	Прос. време	718.9	709.3	705.2	710.0	700.0	704.4
temporal	**Решено (од 45)**	*45*	*45*	*45*	*45*	*45*	*45*
	Прос. време	22.9	19.3	19.3	14.7	13.9	13.0
УКУПНО	**Решено (од 465)**	***323***	***337***	***342***	***347***	***352***	***352***
	Прос. време	**741.3**	**697.2**	**671.3**	**653.9**	**635.6**	**637.4**
	На решеним	**275.9**	**278.4**	**265.4**	**264.2**	**261.8**	**264.2**

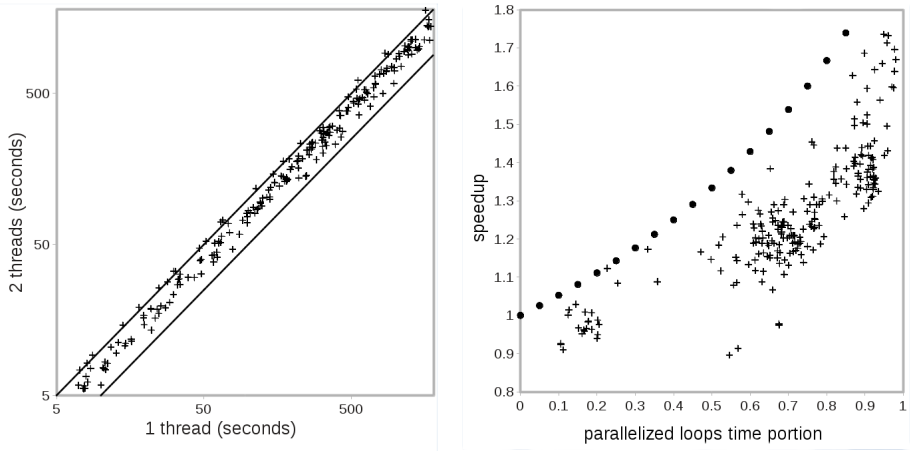
Табела 5.4: Симплекс паралелизација на SMT-LIB `QF_LRA` инстанцама, коришћењем `argosmt` решавача са различитим бројем нити (процесорских језгара). Резултати добијени секвенцијалном верзијом решавача су такође поновљени овде ради лакшег упоређивања (конфигурација $(1, 1)$). Времена решавања су дата у секундама. Код нерешених инстанци за време решавања узима се задато временско ограничење по инстанци (1800 секунди). Просечно време на решеним инстанцама је такође наведено.

бар 5 секунди у секвенцијалном извршавању (228 инстанци укупно). Десни график приказује однос између убрзања и удела времена проведеног у паралелизованим деловима алгоритма. Црни кружићи представљају теоријску горњу границу убрзања, по Амдахловом закону. График потврђује да су добијена убрзања у складу са Амдахловим законом, уз извесне губитке.

Друго важно питање је да ли постоји неко структурно својство инстанце (или њеног таблоа) које одређује понашање симплекс паралелизације на тој конкретној инстанци. У наредном тексту истражујемо како ефективност симплекс паралелизације зависи од следећег својства инстанце, које називамо *тежином*

	Прос. убрзање	**% времена у паралел.**
clock_synchro	1.54	96
cooperatingt2	1.23	69
latendresse	1.58	87
miplib	1.18	44
svcomp	1.27	74
ultimate	1.29	73
sal	0.98	17
scheduling	1.39	90
skdmxa2	1.18	63
skdmxa	1.08	30
temporal	1.20	70
УКУПНО	**1.26**	**70**

Табела 5.5: Просечна убрзања и проценти времена извршавања проведеног у паралелизованим петљама на различитим скуповима инстанци



Слика 5.2: Односи времена извршавања (лево) и убрзање изражено као функција од удела времена проведеног у паралелизованим деловима алгоритма (десно) за `QF_LRA` инстанце (једна нит наспрам две нити)

инстанце I: $weight(I) = nc(I) \cdot mp(I)$, где је $nc(I)$ укупан број коефицијената различитих од нуле у таблоу инстанце I, док је $mp(I)$ једнако 10 ако инстанца I захтева коришћење аритметике произвољне прецизности приликом израчунавања, а 1 у супротном. Мотивација за увођење таквог параметра се заснива на неколико једноставних претпоставки. Прво, ефективност паралализације би требало да доста зависи од величине таблоа, тј. броја врста и колона у таблоу. С обзиром на то да већина паралелизованих делова алгоритма заправо представљају петље које итерирају кроз врсте таблоа, велики број врста значиће и више посла који треба паралелизовати. Са друге стране, с обзиром на то да

користимо имплементацију матрице која је прилагођена ретким матрицама, од велике је важности колико коефициената различитих од нуле постоји просечно у врстама, тј. колика је просечна густина врсте таблоа. С обзиром на то да оба ова својства (величина и густина) директно одређују укупни број коефициената различитих од нуле, делује природно да очекујемо значајну корелацију између овог броја и ефективности паралализације симплекса. Најзад, важна чињеница је и да ли инстанца користи аритметику произвољне прецизности при израчунавањима. С обзиром на то да аритметика произвољне прецизности може бити веома скупа, паралелизација алгоритама који укључују таква израчунавања може бити много ефективнија него паралелизација алгоритама који користе само уграђену хардверску аритметику. Из тог разлога, множимо број коефициената различитих од нуле са 10 за такве инстанце, тј. претпостављамо да коришћење аритметике произвољне прецизности увећава цену паралелизованих делова алгоритма за ред величине.[16] У табели 5.6 можемо видети колико инстанци у сваком од скупова заиста захтева коришћење аритметике произвољне прецизности током извршавања. Може се закључити да је то релативно мали проценат изабраних инстанци.

	Користи прецизну аритм.	**Користи хардверску аритм.**
clock_synchro	31	0
cooperatingt2	13	62
latendresse	8	0
miplib	0	10
svcomp	5	20
ultimate	10	31
sal	0	40
scheduling	0	54
skdmxa2	0	21
skdmxa	0	2
temporal	0	45
УКУПНО	**67**	**285**

Табела 5.6: Број инстанци које користе аритметику произвољне прецизности и оних које је не користе

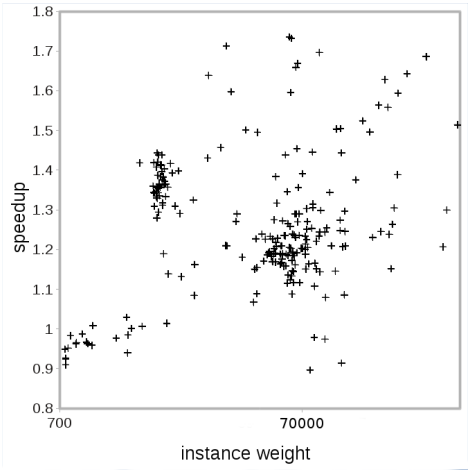
На слици 5.3 је приказан однос између убрзања и тежине инстанце (која је дефинисана у претходном пасусу). График углавном потврђује да тежина инстанце значајно утиче на понашање паралелизације симплекса. На графику су

[16]Фактор 10 је узет произвољно, тј. нисмо прецизно мерили колико пута је аритметика произвољне прецизности заиста спорија од хардверске аритметике. Међутим, чак и када користимо неку другу упоредиву вредност, добијени закључци ће бити веома слични.

приказане све QF_LRA инстанце из свих скупова за чије је решавање било потребно бар 5 секунди помоћу секвенцијалног решавача (228 инстанци укупно). Од тих инстанци, само њих 18 имају екстремно велику тежину (од 150000 до 1300000), док су тежине осталих инстанци готово равномерно дистрибуиране између 700 и 150000. Просечно убрзање на тих 18 инстанци велике тежине је 1.42, док је на осталим инстанцама 1.25. Ипак, постоје и инстанце на којима понашање симплекс паралелизације није у складу са овим уоченим правилом. На пример, може се приметити група инстанци која је најближа горњем левом углу графика на слици 5.3. Већина тих инстанци припада `scheduling` скупу инстанци. Иако ове инстанце немају тако велике тежине (већина тежина варира између 3000 и 7000), паралелизација симплекса се понаша сасвим добро на овим инстанцама (просечно убрзање је 1.39). Да би се објаснило овакво понашање, потребна је детаљнија анализа која у овој тези није разматрана. Ипак, чак и за инстанце из `scheduling` скупа, тежина инстанце као параметар игра прилично значајну (иако можда не одлучујућу) улогу: само 3 од 54 решених `scheduling` инстанци имају тежине преко 50000, а убрзање на тим инстанцама је 1.7, док је просечно убрзање на осталим инстанцама из овог скупа (чије тежине су испод 7000) једнака 1.34. Ово значи да тежина инстанце, иако вероватно није једино својство инстанце које утиче на понашање симплекс паралелизације, свакако јесте један од најбитнијих параметара који утичу на ефективност паралелизације.

Приметимо да ни једно од два својства које смо разматрали у претходним пасусима (удео времена проведеног у паралелизованим петљама са једне стране и тежина инстанце са друге) нам не може помоћи да заиста *предвидимо* понашање симплекс паралализације унапред. Са једне стране, с обзиром на то да не знамо унапред колико ће процената времена бити потрошено на паралелизоване делове алгоритма за неку конкретну инстанцу, не можемо искористити ово својство да унапред одлучимо да ли би било корисно применити паралелизацију симплекса на тој инстанци. Са друге стране, упркос чињеници да се тежина инстанце *може* израчунати унапред (на почетку решавања), ово обично није добра идеја, с обзиром на то да тежина инстанце има тенденцију да буде променљива у времену (зато што се густина таблоа обично мења у току решавања, а аритметика произвољне прецизности може бити уведена у употребу касније, иако на почетку није била неопходна). Из овог разлога, иницијална тежина инстанце може бити значајно различита од тежине инстанце мерене касније у току

процеса решавања, па самим тим иницијална тежина не мора бити добар параметар за предвиђање понашања симплекс паралелизације. У оба случаја, једно практично решење може бити да се решавач покрене у секвенцијалном режиму у почетном стадијуму решавања, како би се сакупиле имформације које нам могу помоћи да предвидимо понашање симплекс паралелизације. На пример, можемо мерити удео времена које је проведено у паралелизованим петљама у току тог временског интервала и на основу тога одлучити да ли у наставку решавања применити симплекс паралелизацију. Слична идеја се може искористити и за тежину инстанце. Наиме, на основу резултата добијених у нашим експериментима, густина таблоа се обично стабилизује након неког релативно малог броја пивотирања. Такође, ако је за ту инстанцу потребно користити аритметику произвољне прецизности, обично се потреба за тим јави веома рано у току решавања, опет након релативно малог броја пивотирања. Ово значи да се и тежина инстанце обично стабилизује након релативно кратког временског периода, а њена вредност се у даљем решавању не мења много. Отуда можемо израчунати тежину инстанце након довољног броја пивотирања (тј. када се стабилизује), и на основу тога одлучити да ли у наставку решавања применити паралелизацију симплекса. Дакле, описани поступак нам даје ефективан начин на који можемо искористити оба наведена својства да одлучимо да ли да користимо симплекс паралелизацију или не на конкретној инстанци.



Слика 5.3: Убрзање на конкретним `QF_LRA` инстанцама изражено као функција тежине инстанце (једна нит наспрам две нити)

Тестирање паралелног портфолија. Тестирање простог паралелног портфолија је извођено коришћењем конфигурација решавача које се састоје од, редом 4, 8, 16 и 32 инстанце `argosmt` решавача који користе различита семена за генерисање случајних бројева у стратегији гранања. Да би поређење било фер, први решавач у портфолију увек користи исто семе као и секвенцијални решавач, док остали користе различита семена. Резултати су приказани у табели 5.7.

		argosmt конфигурација				
		(1,1)	**(4,1)**	**(8,1)**	**(16,1)**	**(32,1)**
clock_synchro	**Решено (од 36)**	*28*	*29*	*30*	*30*	*29*
	Прос. време	479.0	453.5	442.7	433.5	448.9
cooperatingt2	**Решено (од 113)**	*59*	*69*	*71*	*73*	*71*
	Прос. време	1130.8	1026.6	996.8	996.0	1005.3
latendresse	**Решено (од 10)**	*8*	*9*	*9*	*9*	*9*
	Прос. време	362.7	184.0	184.2	184.5	185.4
miplib	**Решено (од 16)**	*10*	*10*	*10*	*10*	*10*
	Прос. време	725.0	699.0	697.8	698.0	698.6
svcomp	**Решено (од 43)**	*21*	*28*	*28*	*28*	*29*
	Прос. време	1159.0	903.4	839.2	814.2	814.8
ultimate	**Решено (од 60)**	*37*	*40*	*41*	*41*	*41*
	Прос. време	979.0	879.8	839.6	847.3	860.2
sal	**Решено (од 54)**	*40*	*40*	*40*	*40*	*40*
	Прос. време	503.4	495.4	495.3	499.0	503.2
scheduling	**Решено (од 58)**	*54*	*56*	*56*	*56*	*56*
	Прос. време	331.1	175.0	134.8	118.9	114.3
skdmxa2	**Решено (од 27)**	*19*	*20*	*20*	*20*	*19*
	Прос. време	974.4	865.8	844.1	870.7	903.5
skdmxa	**Решено (од 3)**	*2*	*2*	*2*	*2*	*2*
	Прос. време	718.9	717.6	716.9	728.4	747.5
temporal	**Решено (од 45)**	*45*	*45*	*45*	*45*	*45*
	Прос. време	22.9	21.2	20.0	19.9	21.8
УКУПНО	**Решено (од 465)**	***323***	***348***	***352***	***354***	***351***
	Прос. време	**741.3**	**645.9**	**620.3**	**618.2**	**625.5**
	На решеним	**275.9**	**258.0**	**241.6**	**247.7**	**244.0**

Табела 5.7: Резултати простог портфолија (резултати секвенцијалног решавача, тј. конфигурације $(1, 1)$ су поновљени због лакшег упоређивања). Времена решавања су дата у секундама. Код нерешених инстанци за време решавања узима се задато временско ограничење по инстанци (1800 секунди). Просечно време на решеним инстанцама је такође наведено.

Кооперативни портфолио је тестиран са потпуно истим конфигурацијама као и прост портфолио. Решавачи у кооперативном портфолију међусобно размењују научене клаузе до дужине 25 (ова дужина је одређена на основу прели-

минарних тестова који су спроведени на неком малом подскупу инстанци које су коришћене у експериментима). Резултати су приказани у табели 5.8.

		argosmt конфигурација				
		(1,1)	**(4,1)**	**(8,1)**	**(16,1)**	**(32,1)**
clock_synchro	**Решено (од 36)**	*28*	*33*	*34*	*34*	*34*
	Прос. време	479.0	292.6	214.0	184.7	164.9
cooperatingt2	**Решено (од 113)**	*59*	*82*	*87*	*93*	*99*
	Прос. време	1130.8	957.6	801.9	665.0	570.6
latendresse	**Решено (од 10)**	*8*	*9*	*9*	*9*	*9*
	Прос. време	362.7	184.5	183.8	184.2	184.4
miplib	**Решено (од 16)**	*10*	*7*	*7*	*7*	*7*
	Прос. време	725.0	1014.4	1013.7	1013.7	1013.3
svcomp	**Решено (од 43)**	*21*	*32*	*33*	*37*	*38*
	Прос. време	1159.0	754.9	659.6	551.9	470.3
ultimate	**Решено (од 60)**	*37*	*42*	*48*	*50*	*51*
	Прос. време	979.0	796.8	693.6	612.0	565.4
sal	**Решено (од 54)**	*40*	*40*	*40*	*40*	*39*
	Прос. време	503.4	527.2	518.7	509.7	531.5
scheduling	**Решено (од 58)**	*54*	*55*	*56*	*56*	*56*
	Прос. време	331.1	137.7	144.8	118.3	112.8
skdmxa2	**Решено (од 27)**	*19*	*22*	*23*	*23*	*23*
	Прос. време	974.4	752.8	689.5	685.3	693.3
skdmxa	**Решено (од 3)**	*2*	*2*	*3*	*3*	*3*
	Прос. време	718.9	651.6	497.1	443.6	384.1
temporal	**Решено (од 45)**	*45*	*45*	*45*	*45*	*45*
	Прос. време	22.9	19.1	16.9	17.7	18.7
УКУПНО	**Решено (од 465)**	***323***	***369***	***385***	***397***	***404***
	Прос. време	**741.3**	**595.0**	**523.9**	**463.0**	**427.0**
	На решеним	**275.9**	**281.5**	**258.8**	**234.1**	**219.7**

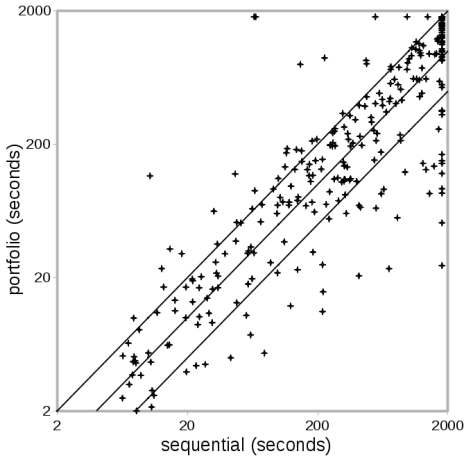
Табела 5.8: Резултати кооперативног портфолија (резултати секвенцијалног решавача, тј. конфигурације $(1, 1)$ су поновљени због лакшег упоређивања). Времена решавања су дата у секундама. Код нерешених инстанци за време решавања узима се задато временско ограничење по инстанци (1800 секунди). Просечно време на решеним инстанцама је такође наведено.

Одмах се може видети да паралелни портфолио даје боље резултате од симплекс паралелизације, чак и без кооперације између решавача. На пример, 4-нитни прост портфолио успева да реши 348 од 465 инстанци за 645 секунди у просеку, док 4-нитни решавач са паралелним симплексом реши 342 инстанце за 671 секунду у просеку. Резултати су (као што је и очекивано) још бољи код кооперативног портфолија (4-нитни решавач реши 369 инстанци за 595 секунди у просеку).

Наравно, овде треба поново нагласити да је убрзање које се може добити

симплекс паралелизацијом ограничено Амдахловим законом [1]. Горња граница зависи од удела времена извршавања које је проведено у паралелизованим деловима симплекс алгоритма. На пример, ако решавач проведе 60% времена у паралелизованим деловима алгоритма, са две нити овај део времена извршавања се може у најбољем случају преполовити, што би дало укупно убрзање 1.43. Са четири нити, најбоље што можемо је да овај део времена извршавања смањимо четири пута, што даје укупно убрзање од 1.82 итд. Како се број нити повећава, убрзање конвергира ка 2.5, али само у теорији. У пракси, можемо очекивати нешто мање убрзање, с обзиром на губитке који се уводе од стране оперативног система приликом распоређивања нити. Са друге стране, паралелни портфолио нема таква ограничења. Ово је посебно случај када се решавачи у портфолију ослањају на неку врсту случајности (нпр. решавачи који користе различита семена за генерисање случајних бројева при гранању). У таквим ситуацијама, најбољи решавач у портфолију може дати и суперлинеарно убрзање, чак и без кооперације између решавача. Из тог разлога, не можемо очекивати да ће симплекс паралелизација бити боља од паралелног портфолија у општем случају. Ипак, на неким конкретним инстанцама, кооперативни паралелни портфолио може бити значајно спорији у решавању чак и од секвенцијалног решавача, с обзиром на то да је његово извршавање недетерминистичко и веома непредвидиво, услед случајног распоређивања нити од стране оперативног система. Овај феномен је илустрован на слици 5.4 (горња, средња и доња линија представљају, респективно, убрзања 1, 2 и 4). Може се видети да постоје инстанце са суперлинеарним убрзањем (тј. већим од 4 са 4-нитним портфолијом), али такође постоје и инстанце са убрзањима значајно мањим од 1. У случају паралелизације симплекса, са друге стране, решавач ради детерминистички и предвидиво, пратећи потпуно исту путању приликом претраге као и секвенцијални решавач, а добијено убрзање се може угрубо проценити унапред (нпр. коришћењем једног од два параметра који су описани у претходном одељку).

Тестирање хибридног приступа. Хибридни приступ је тестиран са конфигурацијама $(2, 16)$, $(4, 8)$, $(8, 4)$ и $(16, 2)$ (тј. свим хибридним конфигурацијама које користе укупно 32 нити). Разматране су како просте, тако и кооперативне варијанте. За кооперативне варијанте, решавачи су размењивали клаузе дужине до 25. У случају простих хибридних конфигурација (табела 5.9), ре-



Слика 5.4: Кооперативни паралелни портфолио (конфигурација $(4, 1)$) наспрам секвенцијалног решавача на `QF_LRA` инстанцама

зултати показују да је у просеку најбоља конфигурација $(8, 4)$. Приметимо да ова конфигурација даје боље резултате и од конфигурације у којој је само симплекс паралелизација укључена (тј. конфигурације $(1, 32)$) и од конфигурације са 32 решавача у портфолију, без симплекс паралелизације (тј. конфигурације $(32, 1)$). Ово значи да ако кооперација између решавача није могућа, тада хибридизација са симплекс паралелизацијом може бити најбољи избор. Са друге стране, резултати кооперативног хибридног решавача (табела 5.10) показују да је најбоља конфигурација $(32, 1)$. Ово значи да ако имплементација допушта кооперацију између решавача у портфолију, тада је обично боље просто узети више решавача у портфолију него укључивати симплекс паралелизацију у оквиру појединачних решавача како би се искористио већи број доступних процесорских језгара. Ипак, ако погледамо резултате на појединим скуповима, можемо приметити да постоје скупови инстанци (попут `temporal` и `skdmxa2`) код којих хибридне конфигурације ($(4, 8)$ и $(8, 4)$, респективно) дају најбоља просечна времена решавања. Ово значи да хибридни приступ може бити користан за неке инстанце, али је потребно даљније истраживање како би се боље разумело у којим случајевима можемо очекивати добар резултат од стране хибридних конфигурација.

		argosmt конфигурација					
		(1,32)	**(2,16)**	**(4,8)**	**(8,4)**	**(16,2)**	**(32,1)**
clock_synchro	**Решено (од 36)**	*31*	*32*	*30*	*30*	*29*	*29*
	Прос. време	401.9	342.8	367.6	397.6	449.5	448.9
cooperatingt2	**Решено (од 113)**	*75*	*79*	*84*	*84*	*79*	*71*
	Прос. време	924.5	888.4	862.4	872.3	940.6	1005.3
latendresse	**Решено (од 10)**	*8*	*9*	*9*	*9*	*9*	*9*
	Прос. време	361.5	182.7	182.5	183.9	185.8	185.4
miplib	**Решено (од 16)**	*10*	*10*	*10*	*10*	*10*	*10*
	Прос. време	733.6	728.8	703.3	701.4	700.9	698.6
svcomp	**Решено (од 43)**	*25*	*31*	*31*	*32*	*29*	*29*
	Прос. време	1001.9	808.4	800.8	767.1	816.5	814.8
ultimate	**Решено (од 60)**	*41*	*42*	*40*	*41*	*40*	*41*
	Прос. време	819.3	784.5	795.4	791.4	847.4	860.2
sal	**Решено (од 54)**	*40*	*40*	*40*	*40*	*40*	*40*
	Прос. време	506.7	504.6	502.8	499.6	501.6	503.2
scheduling	**Решено (од 58)**	*54*	*56*	*56*	*56*	*56*	*56*
	Прос. време	326.9	208.6	163.3	150.7	177.3	114.3
skdmxa2	**Решено (од 27)**	*21*	*20*	*21*	*22*	*20*	*19*
	Прос. време	772.5	777.2	742.2	757.0	830.0	903.5
skdmxa	**Решено (од 3)**	*2*	*2*	*2*	*2*	*2*	*2*
	Прос. време	704.4	697.5	706.6	710.1	738.1	747.5
temporal	**Решено (од 45)**	*45*	*45*	*45*	*45*	*45*	*45*
	Прос. време	13.0	14.3	14.6	16.0	19.4	21.8
УКУПНО	**Решено (од 465)**	***352***	***366***	***368***	***371***	***359***	***351***
	Прос. време	**637.4**	**583.0**	**570.7**	**570.8**	**611.5**	**625.5**
	На решеним	**264.2**	**253.8**	**246.7**	**259.3**	**260.6**	**244.0**

Табела 5.9: Резултати хибридног приступа (проста варијанта). Резултати за конфигурације $(1, 32)$ и $(32, 1)$ су поновљене због лакшег упоређивања. Времена решавања су дата у секундама. Код нерешених инстанци за време решавања узима се задато временско ограничење по инстанци (1800 секунди). Просечно време на решеним инстанцама је такође наведено.

5.6.3 SMT-LIB `QF_LIA` инстанце

Подсетимо се да наш `argosmt` решавач имплементира само једноставну технику гранања са одсецањем када је у питању целобројна аритметика, тако да није оптимизован за решавање аритметичких проблема који укључују целобројне непознате. Из тог разлога, ова теза не садржи детаљну евалуацију описаних техника паралелизације на `QF_LIA` инстанцама. У овом одељку приказујемо само прелиминарне резултате добијене симплекс паралелизацијом на малом скупу `QF_LIA` инстанци преузетих из SMT-LIB колекције. Тестови су спровођени над инстанцама из два `QF_LIA` скупа инстанци из SMT-LIB колекције, а критеријум селекције је исти као и код `QF_LRA` инстанци — изабрали смо

		argosmt конфигурација					
		(1,32)	**(2,16)**	**(4,8)**	**(8,4)**	**(16,2)**	**(32,1)**
clock_synchro	**Решено (од 36)**	*31*	*34*	*34*	*34*	*34*	*34*
	Прос. време	401.9	204.9	174.5	173.6	181.0	164.9
cooperatingt2	**Решено (од 113)**	*75*	*82*	*87*	*91*	*94*	*99*
	Прос. време	924.5	843.6	776.7	703.7	636.3	570.6
latendresse	**Решено (од 10)**	*8*	*9*	*9*	*9*	*9*	*9*
	Прос. време	361.5	182.9	183.1	184.0	184.8	184.4
miplib	**Решено (од 16)**	*10*	*9*	*7*	*7*	*7*	*7*
	Прос. време	733.6	950.0	1015.1	1014.2	1013.8	1013.3
svcomp	**Решено (од 43)**	*25*	*31*	*35*	*35*	*36*	*38*
	Прос. време	1001.9	790.5	710.3	558.9	556.4	470.3
ultimate	**Решено (од 60)**	*41*	*44*	*44*	*50*	*50*	*51*
	Прос. време	819.3	743.8	691.5	631.7	582.5	565.4
sal	**Решено (од 54)**	*40*	*39*	*40*	*40*	*40*	*39*
	Прос. време	506.7	539.3	529.4	534.2	534.6	531.5
scheduling	**Решено (од 58)**	*54*	*53*	*53*	*56*	*56*	*56*
	Прос. време	326.9	227.6	205.4	140.6	153.1	112.8
skdmxa2	**Решено (од 27)**	*21*	*23*	*22*	*23*	*23*	*23*
	Прос. време	772.5	704.6	652.8	613.3	622.3	693.3
skdmxa	**Решено (од 3)**	*2*	*2*	*2*	*2*	*2*	*3*
	Прос. време	704.4	670.5	656.3	654.7	644.7	384.1
temporal	**Решено (од 45)**	*45*	*45*	*45*	*45*	*45*	*45*
	Прос. време	13.0	14.6	13.0	13.9	16.2	18.7
УКУПНО	**Решено (од 465)**	***352***	***371***	***378***	***392***	***396***	***404***
	Прос. време	**637.4**	**564.2**	**526.5**	**477.2**	**457.1**	**427.0**
	На решеним	**264.2**	**251.1**	**233.4**	**230.9**	**223.2**	**219.7**

Табела 5.10: Резултати хибридног приступа (кооперативна варијанта). Резултати за конфигурације $(1, 32)$ и $(32, 1)$ су поновљене због лакшег упоређивања. Времена решавања су дата у секундама. Код нерешених инстанци за време решавања узима се задато временско ограничење по инстанци (1800 секунди). Просечно време на решеним инстанцама је такође наведено.

оне и само оне инстанце које су и `mathsat5` и `cvc4` решавач решили за највише 10 минута. Детаљи о изабраним инстанцама се могу видети у табели 5.11.

Скуп инстанци	Укупно инстанци	Изабрано инстанци	SMT-LIB категорија
slacks	233	143	`QF_LIA`
tropical-matrix	108	19	`QF_LIA`
УКУПНО	**341**	**162**	

Табела 5.11: Сумарни приказ изабраних SMT-LIB QF_LIA инстанци

Резултати секвенцијалног извршавања су приказани у табели 5.12. Наш решавач делује много спорије од друга два решавача, али је ово опет узроковано

тиме што је временско ограничење од 1800 секунди коришћено као време решавања за инстанце које наш решавач није успео да реши. Просечно време решавања на решеним инстанцама је 7 секунди за `slacks` инстанце, а 538 секунди за `tropical-matrix` инстанце, што делује знатно упоредивије са друга два решавача.

		mathsat5	**cvc4**	**argosmt**
slacks	**Решено (од 143)**	*143*	*143*	*131*
	Прос. време	15.1	20.2	157.5
tropical-matrix	**Решено (од 19)**	*19*	*19*	*13*
	Прос. време	78.5	159.4	937.1
УКУПНО	**Решено (од 162)**	***162***	***162***	***144***
	Прос. време	**22.5**	**36.5**	**248.9**
	На решеним	**22.5**	**36.5**	**55.0**

Табела 5.12: Резултати на изабраним SMT-LIB `QF_LIA` инстанцама коришћењем секвенцијалних решавача `mathsat5`, `cvc4` и нашег решавача `argosmt`. Времена решавања су дата у секундама. Код нерешених инстанци за време решавања узима се задато временско ограничење по инстанци (1800 секунди). Просечно време на решеним инстанцама је такође наведено.

Резултати симплекс паралелизације су приказани у табели 5.13. Резултати показују значајно убрзање на `tropical-matrix` инстанцама, док готово да нема убрзања на `slacks` инстанцама. Тешко је без детаљније анализе утврдити разлог за овакво понашање. Са једне стране, скуп инстанци `tropical-matrix` се састоји из само 19 инстанци (од којих су само 16 решене помоћу паралелног `argosmt` решавача). Тешко је дати неке веродостојне закључке засноване на тако малом скупу инстанци, али делује да су ове инстанце довољно тешке, тако да се значајан проценат времена проводи у паралелизованим петљама у симплекс алгоритму, што позитивно утиче на ефекат паралелизације. Са друге стране, с обзиром на то да је просечно време решавања на `slacks` инстанцама само 7 секунди, не можемо очекивати да ће симплекс паралелизација на тако лаким инстанцама дати неко значајно побољшање. На жалост, били смо прилично ограничени приликом избора инстанци, с обзиром на недостатак ефикаснијих техника за целобројну аритметику у оквиру нашег решавача. У крајњој линији, можемо закључити да симплекс паралелизација на `QF_LIA` инстанцама може бити једнако ефективна као и код `QF_LRA` инстанци, али је потребна детаљнија евалуација на већем броју скупова инстанци и уз помоћ решавача који имплементира ефикасније технике за решавање целобројне линеарне аритме-

тике, како би био у стању да решава знатно теже QF_LIA инстанце.

		argosmt конфигурација					
		(1,1)	**(1,2)**	**(1,4)**	**(1,8)**	**(1,16)**	**(1,32)**
slacks	**Решено (од 143)**	*131*	*132*	*132*	*132*	*132*	*131*
	Прос. време	157.5	154.1	153.7	153.7	153.7	154.4
tropical-matrix	**Решено (од 19)**	*13*	*15*	*16*	*16*	*16*	*16*
	Прос. време	937.1	858.0	784.3	718.9	677.9	665.6
УКУПНО	**Решено (од 162)**	***144***	***147***	***148***	***148***	***148***	***147***
	Прос. време	**248.9**	**236.6**	**227.6**	**220.0**	**215.2**	**214.3**
	На решеним	**55.0**	**77.1**	**78.9**	**70.5**	**65.3**	**52.5**

Табела 5.13: Симплекс паралелизација на SMT-LIB QF_LIA инстанцама, коришћењем argosmt решавача са различитим бројем нити (процесорских језгара). Резултати добијени секвенцијалном верзијом решавача су такође поновљени овде ради лакшег упоређивања (конфигурација $(1, 1)$). Времена решавања су дата у секундама. Код нерешених инстанци за време решавања узима се задато временско ограничење по инстанци (1800 секунди). Просечно време на решеним инстанцама је такође наведено.

5.7 Закључак

У овој глави тезе анализиран је ефекат различитих приступа паралелизацији SMT решавача, када је у питању решавање проблема у теорији линеарне аритметике. Први приступ је паралелизација симплекс алгоритма на коме су засноване процедуре одлучивања за испитивање задовољивости конјункције линеарних аритметичких ограничења у оквиру SMT решавача. Ово је једна од најсложенијих процедура у модерним SMT решавачима, па њена паралелизација може значајно да утиче на перформансе решавача. Овај приступ паралелизацији је заснован на паралелном извршавању истих операција над више различитих врста таблоа истовремено. У овој глави тезе је детаљно описан начин на који је ова паралелизација постигнута. Други приступ је покретање паралелног портфолија решавача који користе различита семена за генерисање случајних бројева у хеуристикама гранања. Трећи приступ је хибридизација претходна два: сваком од решавача у портфолију се дају додатни процесори које он може искористити за паралелизацију симплекса.

Ова глава тезе такође садржи детаљну евалуацију и упоређивање предложених техника паралелизације. Тестови су спроведени над великим бројем инстанци (како индустријских, тако и вештачки генерисаних), коришћењем ра-

чунара са великим бројем процесора (сваки од приступа је тестиран са различитим бројем нити, а највише до 32). Такође је дато упоређивање са неким од актуелних SMT решавача (конкретно са MathSAT5 и CVC4 решавачима). Цело истраживање је захтевало много месеци развоја и тестирања имплементације, као и скоро три месеца (око 80 дана) непрекидног извршавања експеримената како би се добили резултати који су овде презентовани. Аутор се нада да добијени резултати могу бити извор значајних информација за истраживаче који раде на развоју паралелних SMT решавача.

Експериментални резултати су потврдили потенцијал приступа заснованог на паралелизацији симплекса за неке класе инстанци. Општи закључак је да ефективност симплекс паралелизације непосредно зависи од удела времена извршавања које је проведено у паралелизованим деловима симплекс алгоритма. Када су структурне особине инстанце у питању, понашање симплекс паралелизације зависи од броја коефицијената различитих од нуле у таблоу инстанце, а који зависи од величине и густине таблоа. Још један битан фактор је и коришћење аритметике произвољне прецизности у израчунавањима. У поглављу 5.6, ова својства су обједињена у једно својство које смо назвали *тежином* инстанце, а које се показало као корисно у процени потенцијала симплекс паралелизације на конкретној инстанци.

Резултати такође показују да паралелни портфолио у просеку даје боље резултате од симплекс паралелизације (услед теоријских ограничења убрзања симплекс паралелизације, у складу са Амдахловим законом), али детерминистичко извршавање и предвидиво понашање симплекс паралелизације чини овај приступ знатно поузданијим од паралелног портфолија. Најзад, експерименти са хибридним приступом нам говоре да у неким случајевима постоји могућност побољшања паралелног портфолија његовим комбиновањем са симплекс паралелизацијом која је описана у овој тези.

У оквиру даљег рада, планирано је детаљније испитивање неких феномена који су примећени у овом истраживању. На пример, делује да постоје и друга структурна својства инстанце (којих ми нисмо свесни у овом тренутку) који утичу на паралелизацију симплекса, осим тежине инстанце. Даље, слична анализа се може урадити и за портфолио приступ, тј. можемо покушати да откријемо својства инстанце која утичу на понашање паралелног портфолија. Може бити занимљиво да се резултати добијени у таквом истраживању упореде са резултатима добијеним у овој тези који се тичу симплекс паралелизације, као и да

се анализирају сличности и разлике. На пример, ако погледамо резултате који су добијени за инстанце из `sal` скупа, видимо да код ових инстанци сви приступи паралелизацији дају лоше резултате, док код `temporal` инстанци имамо боље резултате помоћу симплекс паралелизације него помоћу паралелног портфолија. Питање који приступ паралелизацији применити на којој инстанци (и да ли уопште има смисла примењивати паралелизацију на некој инстанци) као и остала занимљива питања која се тичу паралелизације SMT решавача могу се разматрати у будућем раду.

Глава 6

Закључци и даљи рад

6.1 Закључци

У оквиру ове тезе је разматрано унапређивање SMT решавача CSP техникама и техникама паралелизације. За потребе ових истраживања најпре је развијен нови SMT решавач `argosmt` отвореног кода и флексибилног дизајна, заснован на DPLL($\mathcal{T}$) архитектури. Неке од значајних карактеристика `argosmt` решавача су ефикасна имплементација израза, лењо увођење дељених једнакости код комбинације теорија, подршка за паралелизацију, итд. Нарочито занимљива одлика архитектуре `argosmt` решавача је то што је, за разлику од традиционалног приступа, исказно резоновање измештено из SAT решавача и организовано као посебан теоријски решавач (који смо назвали *исказни решавач*). Овакав приступ значајно поједностављује имплементацију, зато што омогућава да се исказно резоновање третира на исти начин као и резоновање у уобичајеним SMT теоријама.

Када су у питању CSP технике, најпре је дефинисана нова SMT теорија коју смо назвали *CSP теорија*, а која представља логички оквир који је погодан за изражавање CSP проблема на SMT језику. Ово пре свега подразумева могућност непосредног задавања неких често коришћених глобалних ограничења. За тако дефинисану теорију развијен је теоријски решавач који је уграђен у `argosmt` решавач. Овај теоријски решавач је заснован на постојећим техникама позајмљеним из традиционалних CSP решавача које су адаптиране тако да могу да функционишу у оквиру SMT решавача. Приликом имплементације, нарочита пажња је посвећена ефикасности комуникације између алгоритама филтрирања за појединачна глобална ограничења. У оквиру тезе разматрана

је подршка за два типа глобалних ограничења: `alldifferent` ограничење и линеарно ограничење. У случају `alldifferent` ограничења, најважнији резултат тезе је нови алгоритам за генерисање објашњења конфликата и пропагација који је било неопходно развити за потребе анализе конфликата. Када је у питању линеарно ограничење, поред стандардног алгоритма филтрирања развијен је и потпуно нови алгоритам филтрирања који узима у обзир постојање `alldifferent` ограничења у датом проблему, што му омогућава да израчуна јаче границе променљивих, чиме се додатно сужава простор претраге. За све предложене технике дата је и одговарајућа експериментална евалуација, а решавач је такође упоређен и са другим актуелним решавачима.

Када су у питању технике паралелизације, најзначајнији допринос је имплементација паралелизације симплекс алгоритма у теоријском решавачу за линеарну аритметику. Такође, уграђена је и подршка за паралелни портфолио, а омогућена је и хибридизација ова два приступа. Веома значајан допринос овог дела тезе је и опсежна експериментална евалуација наведених техника паралелизације, као и анализа добијених резултата која може пружити значајне информације истраживачима који се баве развојем и имплементацијом, како паралелних SMT решавача, тако и SMT решавача уопште.

6.2 Даљи рад

Правци даљег рада укључују уградњу алгоритама филтрирања за друге типове глобалних ограничења, као и адаптацију CSP техника тако да се могу користити у комбинацији са стандардним SMT теоријама (на пример, комбинација CSP теорије са EUF теоријом). Ово може укључивати и подршку за бесконачне домене. Када је у питању паралелизација, даљи рад може подразумевати паралелизацију других процедура одлучивања које се користе у SMT решавачима. Такође, детаљнија анализа неких феномена који су остали неразјашњени у овој тези може бити део будућих истраживања. Најзад, један могући правац даљег рада је и примена паралелног портфолија у решавању CSP проблема. С обзиром на то да `argosmt` решавач у овом тренутку подржава и паралелни портфолио и алгоритме за ефикасно решавање CSP проблема, идеја о комбиновању ове две технике у решавању CSP проблема се природно намеће, а верујемо да би се на тај начин постигли још бољи резултати у овој области.

Библиографија

[1] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM, 1967.

[2] Krzysztof Apt. *Principles of Constraint Programming*. Cambridge University Press, New York, NY, USA, 2003.

[3] Milan Bankovic. ArgoSMTExpression: an SMT-LIB 2.0 compliant expression library. In *Workshop of the SAT (June 2012)*, 2012.

[4] Milan Banković. Extending SMT solvers with support for finite domain alldifferent constraint. *Constraints*, 21(4):463–494, 2015.

[5] Milan Banković. Parallelizing simplex within SMT solvers. *Artificial Intelligence Review*, pages 1–30, 2016.

[6] Milan Banković. Solving finite-domain linear constraints in presence of the `alldifferent`. *Logical Methods in Computer Science*, 12(3):1–28, 2016.

[7] Milan Bankovic and Filip Maric. An Alldifferent constraint solver in SMT. In *8th International Workshop on Satisfiability Modulo Theories*, 2010.

[8] Clark Barrett, Christopher L Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. Cvc4. In *Computer aided verification*, pages 171–177. Springer, 2011.

[9] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.5. Technical report, Department of Computer Science, The University of Iowa, 2015. Available at `http://www.SMT-LIB.org`.

[10] Clark Barrett, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Splitting on demand in SAT modulo theories. In *International Conference*

on Logic for Programming Artificial Intelligence and Reasoning, pages 512–526. Springer, 2006.

[11] Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability Modulo Theories. In *Handbook of Satisfiability*, chapter 26, pages 825–885. IOS Press, 2009.

[12] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB Standard: Version 2.0. Technical report, Department of Computer Science, The University of Iowa, 2010. Available at `http://www.SMT-LIB.org`.

[13] Nicolas Beldiceanu, Mats Carlsson, Thierry Petit, and Jean-Charles Régin. An O (nlog n) Bound Consistency Algorithm for the Conjunction of an alldifferent and an Inequality between a Sum of Variables and a Constant, and its Generalization. In *ECAI*, volume 12, pages 145–150, 2012.

[14] Claude Berge. Graphes et hypergraphes. 1970.

[15] Christian Bessiere, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. The alldifferent constraint with precedences. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 36–52. Springer, 2011.

[16] Armin Biere. Lingeling, Plingeling and Treengeling Entering the SAT Competition 2013. In *Proceedings of SAT Competition 2013*, pages 51–52. University of Helsinki, 2013.

[17] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[18] Miquel Bofill, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. A write-based solver for SAT modulo the theory of arrays. In *Proceedings of the 2008 International Conference on Formal Methods in Computer-Aided Design*, pages 1–8. IEEE Press, 2008.

[19] Miquel Bofill, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. The barcelogic SMT solver. In *International Conference on Computer Aided Verification*, pages 294–298. Springer, 2008.

[20] Miquel Bofill, Miquel Palahí, Josep Suy, and Mateu Villaret. Solving constraint satisfaction problems with SAT modulo theories. *Constraints*, 17(3):273–303, 2012.

[21] Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi Junttila, Peter Van Rossum, Stephan Schulz, and Roberto Sebastiani. An incremental and layered procedure for the satisfiability of linear arithmetic logic. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 317–333. Springer, 2005.

[22] Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi A. Junttila, Silvio Ranise, Peter van Rossum, and Roberto Sebastiani. Efficient Satisfiability Modulo Theories via Delayed Theory Combination. In *CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 335–349. Springer, 2005.

[23] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, Ziyad Hanna, Alexander Nadel, Amit Palti, and Roberto Sebastiani. A Lazy and Layered SMT ($\mathcal{BV}$) Solver for Hard Industrial Verification Problems. In *International Conference on Computer Aided Verification*, pages 547–560. Springer, 2007.

[24] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani. Delayed theory combination vs. Nelson-Oppen for satisfiability modulo theories: A comparative analysis. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 527–541. Springer, 2006.

[25] Roberto Bruttomesso, Edgar Pek, Natasha Sharygina, and Aliaksei Tsitovich. The OpenSMT Solver. In *TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 150–153. Springer, 2010.

[26] Randal E Bryant, Daniel Kroening, Joël Ouaknine, Sanjit A Seshia, Ofer Strichman, and Bryan Brady. Deciding bit-vector arithmetic with abstraction. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 358–372. Springer, 2007.

[27] 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.

[28] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[29] George B Dantzig, Alex Orden, Philip Wolfe, et al. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, 5(2):183–195, 1955.

[30] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, July 1962.

[31] Leonardo De Moura, Nikolaj Bj, et al. Generalized, efficient array decision procedures. In *Formal Methods in Computer-Aided Design, 2009. FMCAD 2009*, pages 45–52. IEEE, 2009.

[32] Leonardo de Moura and Nikolaj Bjørner. Model-based theory combination. *Electronic Notes in Theoretical Computer Science*, 198(2):37–49, 2008.

[33] Leonardo de Moura and Nikolaj Bjorner. Z3: An Efficient SMT Solver. In *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[34] Isil Dillig, Thomas Dillig, and Alex Aiken. Cuts from proofs: A complete and practical technique for solving linear inequalities over integers. In *Computer Aided Verification*, pages 233–247. Springer, 2009.

[35] Nicholas Downing, Thibaut Feydy, and Peter J Stuckey. Explaining alldifferent. In *Proceedings of the Thirty-fifth Australasian Computer Science Conference-Volume 122*, pages 115–124. Australian Computer Society, Inc., 2012.

[36] Nicholas Downing, Thibaut Feydy, and Peter J Stuckey. Explaining flow-based propagation. In *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, pages 146–162. Springer, 2012.

[37] Bruno Dutertre and Leonardo de Moura. Integrating simplex with DPLL(T). Technical report, CSL, SRI International, 2006.

[38] Bruno Dutertre and Leonardo De Moura. The Yices SMT solver. *Tool paper at http://yices. csl. sri. com/tool-paper. pdf*, 2:2, 2006.

[39] Lester R Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8(3):399–404, 1956.

[40] Alan M Frisch, Ian Miguel, and Toby Walsh. CGRASS: A system for transforming constraint satisfaction problems. In *Recent Advances in Constraints*, pages 15–30. Springer, 2003.

[41] Philippe Galinier and Québec) Centre for Research on Transportation (Montréal. *A constraint-based approach to the Golomb ruler problem*. Montréal: Centre for Research on Transportation= Centre de recherche sur les transports (CRT), 2003.

[42] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T): Fast Decision Procedures. In *CAV*, volume 3114 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2004.

[43] Ian P Gent, Christopher Jefferson, and Ian Miguel. Minion: A fast scalable constraint solver. In *ECAI*, volume 141, pages 98–102, 2006.

[44] Ian P Gent and Inês Lynce. A SAT encoding for the social golfer problem. *Modelling and Solving Problems with Constraints*, page 2, 2005.

[45] Ian P Gent, Ian Miguel, and Neil CA Moore. Lazy explanations for constraint propagators. In *Practical Aspects of Declarative Languages*, pages 217–233. Springer, 2010.

[46] Carla Gomes and David Shmoys. Completing quasigroups or latin squares: A structured graph coloring problem. In *proceedings of the Computational Symposium on Graph Coloring and Generalizations*, pages 22–39, 2002.

[47] Alberto Griggio. *An effective SMT engine for Formal Verification*. PhD thesis, University of Trento, 2009.

[48] Alberto Griggio. A practical approach to satisfiability modulo linear integer arithmetic. *Journal on Satisfiability, Boolean Modeling and Computation*, 8:1–27, 2012.

[49] JAJ Hall. Towards a practical parallelisation of the simplex method. *Computational Management Science*, 7(2):139–170, 2010.

[50] Youssef Hamadi, Said Jabbour, and Lakhdar Sais. ManySAT: a parallel SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:245–262, 2009.

[51] Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. Combining Symmetry Breaking with Other Constraints: Lexicographic Ordering with Sums. In *AMAI*, 2004.

[52] Steffen Hölldobler, Norbert Manthey, Van Hau Nguyen, Peter Steinke, and Julian Stecklina. Modern Parallel SAT-Solvers. Technical report, TR 2011-6, Knowledge Representation and Reasoning Group, TU Dresden, Germany, 2011.

[53] Predrag Janicic. Uniform Reduction to SAT. *Logical Methods in Computer Science*, 8(3), 2010.

[54] Predrag Janicic and Filip Maric. Uniform reduction to SMT. 2010.

[55] Dejan Jovanović and Leonardo De Moura. Cutting to the chase solving linear integer arithmetic. In *Automated Deduction–CADE-23*, pages 338–353. Springer, 2011.

[56] Bernard Jurkowiak, Chu Min Li, and Gil Utard. A Parallelization Scheme Based on Work Stealing for a Class of SAT Solvers. *Journal of Automated Reasoning*, 34(1):73–101, 2005.

[57] Natalia Kalinnik, Erika Abraham, Tobias Schubert, Ralf Wimmer, and Bernd Becker. Exploiting Different Strategies for the Parallelization of an SMT Solver. In *MBMV*, pages 97–106. Fraunhofer Verlag, 2010.

[58] George Katsirelos. *Nogood processing in CSPs*. PhD thesis, University of Toronto, 2008.

[59] Tim King. *Effective Algorithms for the Satisfiability of Quantifier-Free Formulas Over Linear Real and Integer Arithmetic*. PhD thesis, NEW YORK UNIVERSITY, 2014.

[60] Sava Krstic and Amit Goel. Architecting Solvers for SAT Modulo Theories: Nelson-Oppen with DPLL. In *FroCoS*, volume 4720 of *Lecture Notes in Computer Science*, pages 1–27. Springer, 2007.

[61] Rhyd Lewis. Metaheuristics can solve sudoku puzzles. *Journal of heuristics*, 13(4):387–401, 2007.

[62] Alejandro López-Ortiz, Claude-Guy Quimper, John Tromp, and Peter Van Beek. A fast and simple algorithm for bounds consistency of the alldifferent constraint. In *IJCAI*, volume 3, pages 245–250, 2003.

[63] Norbert Manthey. Parallel SAT Solving-Using More Cores. In *Pragmatics of SAT Workshop*, 2011.

[64] Filip Marić. *Formalizacija, implementacija i primene SAT rešavača*. PhD thesis, University of Belgrade, Serbia, 2009.

[65] Joao Marques-Silva, Ines Lynce, and Sharad Malik. Conflict-Driven Clause Learning SAT Solvers. In *Handbook of Satisfiability*, chapter 4, pages 131–155. IOS Press, 2009.

[66] Kurt Mehlhorn and Sven Thiel. Faster algorithms for bound-consistency of the sortedness and the alldifferent constraint. In *Principles and Practice of Constraint Programming–CP 2000*, pages 306–319. Springer, 2000.

[67] Neil Moore. *Improving the efficiency of learning CSP solvers*. PhD thesis, University of St Andrews, 2011.

[68] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Annual ACM IEEE Design Automation Conference*, pages 530–535. ACM, 2001.

[69] Greg Nelson and Derek C. Oppen. Simplification by Cooperating Decision Procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979.

[70] Greg Nelson and Derek C Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM (JACM)*, 27(2):356–364, 1980.

[71] Robert Nieuwenhuis. Sat modulo theories: Enhancing SAT with special-purpose algorithms. In *Theory and Applications of Satisfiability Testing-SAT 2009*, pages 1–1. Springer, 2009.

[72] Robert Nieuwenhuis and Albert Oliveras. DPLL (T) with exhaustive theory propagation and its application to difference logic. In *Computer Aided Verification*, pages 321–334. Springer, 2005.

[73] Robert Nieuwenhuis and Albert Oliveras. Fast congruence closure and extensions. *Information and Computation*, 205(4):557–580, 2007.

[74] Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. Challenges in satisfiability modulo theories. In *Term Rewriting and Applications*, pages 2–18. Springer, 2007.

[75] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL (T). *Journal of the ACM (JACM)*, 53(6):937–977, 2006.

[76] Peter Nightingale, Özgür Akgün, Ian P Gent, Christopher Jefferson, and Ian Miguel. Automatically improving constraint models in Savile Row through associative-commutative common subexpression elimination. In *Principles and Practice of Constraint Programming*, pages 590–605. Springer, 2014.

[77] Olga Ohrimenko, Peter J Stuckey, and Michael Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009.

[78] Justyna Petke and Peter Jeavons. The order encoding: from tractable CSP to tractable SAT. In *Theory and Applications of Satisfiability Testing-SAT 2011*, pages 371–372. Springer, 2011.

[79] Jean-François Puget. A fast algorithm for the bound consistency of alldiff constraints. In *AAAI/IAAI*, pages 359–366, 1998.

[80] Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *AAAI*, volume 94, pages 362–367, 1994.

[81] Jean-Charles Régin. Generalized arc consistency for global cardinality constraint. In *Proceedings of the thirteenth national conference on Artificial intelligence-Volume 1*, pages 209–215. AAAI Press, 1996.

[82] Jean-Charles Régin. Cost-based arc consistency for global cardinality constraints. *Constraints*, 7(3-4):387–405, 2002.

[83] Jean-Charles Régin. Global constraints and filtering algorithms. In *Constraint and Integer Programming*, pages 89–135. Springer, 2004.

[84] Guillaume Rochart, Narendra Jussien, and François Laburthe. Challenging explanations for global constraints. In *CP03 Workshop on User-Interaction in Constraint Satisfaction (UICS'03)*, pages 31–43, 2003.

[85] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[86] Christian Schulte and Peter J Stuckey. Speeding up constraint propagation. In *Principles and Practice of Constraint Programming–CP 2004*, pages 619–633. Springer, 2004.

[87] Christian Schulte and Peter J Stuckey. When do bounds and domain propagation lead to the same search space? *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 27(3):388–425, 2005.

[88] Meinolf Sellmann and Serdar Kadioglu. Dichotomic search protocols for constrained optimization. In *Principles and Practice of Constraint Programming*, pages 251–265. Springer, 2008.

[89] Hossein M Sheini and Karem A Sakallah. A scalable method for solving satisfiability of integer linear arithmetic logic. In *Theory and Applications of Satisfiability Testing*, pages 241–256. Springer, 2005.

[90] Helmut Simonis. Kakuro as a constraint problem. *Proc. seventh Int. Works. on Constraint Modelling and Reformulation*, 2008.

[91] Daniel Singer. *Parallel Resolution of the Satisfiability Problem: A Survey*, pages 123–147. Wiley, 2006.

[92] Carsten Sinz, Wolfgang Blochinger, and Wolfgang Küchlin. PaSAT - Parallel SAT-Checking with Lemma Exchange: Implementation and Applications. *Electronic Notes in Discrete Mathematics*, 9:205–216, 2001.

[93] Barabara M Smith, Kostas Stergiou, and Toby Walsh. Modelling the golomb ruler problem. *RESEARCH REPORT SERIES-UNIVERSITY OF LEEDS SCHOOL OF COMPUTER STUDIES LU SCS RR*, 1999.

[94] Richard M Stallman and Gerald J Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial intelligence*, 9(2):135–196, 1977.

[95] Mirko Stojadinović and Filip Marić. meSAT: multiple encodings of CSP to SAT. *Constraints*, pages 1–24, 2014.

[96] Peter J Stuckey. Lazy clause generation: Combining the power of SAT and CP (and MIP?) solving. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 5–9. Springer, 2010.

[97] Aaron Stump, Clark W Barrett, David L Dill, and Jeremy R Levitt. A Decision Procedure for an Extensional Theory of Arrays. In *lics*, volume 1, pages 29–37. DTIC Document, 2001.

[98] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear CSP into SAT. *Constraints*, 14(2):254–272, 2009.

[99] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[100] Alfred Tarski. A decision method for elementary algebra and geometry. 1951.

[101] Cesare Tinelli and Mehdi Harandi. A new correctness proof of the Nelson-Oppen combination procedure. In *Frontiers of Combining Systems*, pages 103–119. Springer, 1996.

[102] G. S. Tseitin. On the complexity of derivations in the propositional calculus. *Studies in Mathematics and Mathematical Logic*, Part II:115–125, 1968.

[103] Willem-Jan van Hoeve. The alldifferent constraint: A survey. *arXiv preprint cs/0105015*, 2001.

[104] Christoph M. Wintersteiger, Youssef Hamadi, and Leonardo de Moura. A Concurrent Portfolio Approach to SMT Solving. In *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 715–720. Springer, 2009.

[105] Hantao Zhang, Maria Paola Bonacina, and Jieh Hsiang. PSATO: a Distributed Propositional Prover and Its Application to Quasigroup Problems. *Journal of Symbolic Computation*, 21:543–560, 1996.

Биографија аутора

Милан Банковић је рођен 3. 5. 1982. године у Петровцу на Млави. Математички факултет, смер *Рачунарство и информатика*, уписао је октобра 2001. године. Дипломирао је децембра 2006. године са просечном оценом 9,93.

Докторске студије на смеру *Информатика* на Математичком факултету уписао је 2007. године. Све испите предвиђене планом студија положио је са оценом 10. У току рада на тези објавио је три самостална научна рада у часописима са SCI листе. Поред тога, похађао је и једну летњу школу из области SAT/SMT решавача, а учествовао је и у раду две водеће међународне конференције, где је у оквиру придружених радионица имао запажена излагања. Такође је учествовао у организацији више научних скупова у земљи које је организовала *Арго група* Математичког факултета, чији је члан од 2009. године.

Миланова научна интересовања су пре свега у области SAT и SMT решавача, са акцентом на њиховом развоју, унапређивању и применама. Такође је заинтересован и за програмирање ограничења. До сада је у два циклуса био учесник научних пројеката које је финансирало Министарство просвете, науке и технолошког развоја Владе Републике Србије.

Од 2007. године ангажован је на Математичком факултету, најпре као сарадник у настави, а од 2010. године као асистент за научну област *рачунарство и информатика*.

Прилог 1.

Изјава о ауторству

Потписани-а _____

број индекса _____

Изјављујем

да је докторска дисертација под насловом

_____

_____

- резултат сопственог истраживачког рада,
- да предложена дисертација у целини ни у деловима није била предложена за добијање било које дипломе према студијским програмима других високошколских установа,
- да су резултати коректно наведени и
- да нисам кршио/ла ауторска права и користио интелектуалну својину других лица.

Потпис докторанда

У Београду, _____

_____

Прилог 2.

Изјава о истоветности штампане и електронске верзије докторског рада

Име и презиме аутора _____

Број индекса _____

Студијски програм _____

Наслов рада _____

Ментор _____

Потписани/а _____

Изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао/ла за објављивање на порталу **Дигиталног репозиторијума Универзитета у Београду.**

Дозвољавам да се објаве моји лични подаци везани за добијање академског звања доктора наука, као што су име и презиме, година и место рођења и датум одбране рада.

Ови лични подаци могу се објавити на мрежним страницама дигиталне библиотеке, у електронском каталогу и у публикацијама Универзитета у Београду.

Потпис докторанда

У Београду, _____

_____

Прилог 3.

Изјава о коришћењу

Овлашћујем Универзитетску библиотеку „Светозар Марковић“ да у Дигитални репозиторијум Универзитета у Београду унесе моју докторску дисертацију под насловом:

_____

_____

која је моје ауторско дело.

Дисертацију са свим прилозима предао/ла сам у електронском формату погодном за трајно архивирање.

Моју докторску дисертацију похрањену у Дигитални репозиторијум Универзитета у Београду могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons) за коју сам се одлучио/ла.

1. Ауторство

2. Ауторство - некомерцијално

3. Ауторство – некомерцијално – без прераде

4. Ауторство – некомерцијално – делити под истим условима

5. Ауторство – без прераде

6. Ауторство – делити под истим условима

(Молимо да заокружите само једну од шест понуђених лиценци, кратак опис лиценци дат је на полеђини листа).

Потпис докторанда

У Београду, _____

_____