

УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ



Марија Марковић

ПРОШИРЕЊЕ АЛАТА URSA ПОДРШКОМ
ЗА SMT РЕШАВАЧЕ

мастер рад

Београд, 2026.

Ментор:

проф. др Предраг ЈАНИЧИЋ, редован професор
Универзитет у Београду, Математички факултет

Чланови комисије:

проф. др Филип МАРИЋ, редован професор
Универзитет у Београду, Математички факултет

др Милан БАНКОВИЋ, доцент
Универзитет у Београду, Математички факултет

Датум одбране: _____

Мојим родитељима

Наслов мастер рада: Проширење алата URSA подршком за SMT решаваче

Резиме: Алат URSA сваки задати проблем решава свођењем на SAT, проблем задовољивости формуле исказне логике, чиме постиже широку применљивост на комбинаторне проблеме, али остаје везан за исказну логику као једини циљни формализам. Та везаност ограничава класу проблема који се могу природно изразити, будући да исказна логика не досеже изражајност коју поједини проблеми захтевају. У овом раду описано је проширење алата URSA којим се исти улазни програм, поред досадашњег свођења на SAT, може свести и на SMT, задовољивост у односу на теорије, и то аритметике бит-вектора, линеарне аритметике целих бројева и теорије низова. Тако добијене формуле решавају се решавачима Z3 и cvc5. Проширење је оцењено над постојећим примерима алата и над новим примерима прилагођеним новоуведеним теоријама. Резултати показују да ново свођење даје иста решења као досадашње, чиме је потврђена његова исправност, и да уз то омогућава изражавање и решавање проблема који раније нису били у домаћају алата, попут провере програма са симболичким приступом низовима и задатака над целим бројевима без ограничења ширине и са означеним вредностима. Мерења показују да ниједан приступ није универзално бржи, па два начина остају комплементарна. Тежиште доприноса није у убрзању, већ у изражајности: пошто ново свођење резонује на нивоу теорија, а не нужно на нивоу појединачних битова, обухвата и проблеме на којима свођење на SAT нагло расте с порастом ширине бит-вектора и оне које оно уопште не може да изрази.

Кључне речи: URSA, SAT, SMT, теорија аритметике бит-вектора, теорија линеарне аритметике целих бројева, теорија низова, аутоматско резонување, симболичко извршавање

Садржај

1	Увод	1
2	Основни појмови и преглед области	5
2.1	Проблем исказне задовољивости – SAT	5
2.2	Проблем задовољивости у односу на теорију – SMT	6
2.2.1	Теорија аритметике бит-вектора	8
2.2.2	Линеарна аритметика целих бројева	9
2.2.3	Теорија низова	10
2.3	Решавачи и симболичко извршавање	12
2.3.1	Симболичко извршавање	12
2.3.2	Доменски специфични језици потпомогнути решавачима	13
2.3.3	Систем Rosette	13
2.3.4	Алат URSA	15
2.3.5	Алат URBIVA	16
3	Проширење система URSA	19
3.1	Архитектура имплементације	19
3.2	Организација кода	21
3.3	Подршка за QF_BV	24
3.4	Подршка за QF_LIA	27
3.5	Подршка за низове са симболичким индексима	29
3.5.1	Подршка за QF_ABV	30
3.5.2	Подршка за QF_ALIA	33
4	Експериментална евалуација	36
4.1	Методологија	36
4.2	Инфраструктура за тестирање	37

4.3	Резултати	38
4.3.1	Проблеми одлучивања	39
4.3.2	Оптимизациони проблеми	43
4.3.3	Зависност од ширине бит-вектора	45
4.3.4	Верификација програма са симболичким индексима	46
4.4	Анализа резултата	50
5	Закључак и правци даљег рада	54
A	Језик алата URSA	56
A.1	Лексичке конвенције	56
A.2	Изрази и оператори	56
A.3	Наредбе и контрола тока	57
A.4	Низови	57
A.5	Процедуре	58
A.6	Ограничења и оптимизација	58
A.7	Позивање алата	58
	Библиографија	59

Глава 1

Увод

Велики број комбинаторних и логичких проблема може се решити свођењем на проблем задовољивости исказне логике (SAT), за који постоје ефикасни решавачи. На тој идеји заснован је алат URSA, који програме написане у наменском језику преводи у исказну формулу и решава је позивом SAT решавача [20]. Све вредности у том приступу представљају се бит-векторима фиксне ширине, а сложене конструкције језика разлажу се на мрежу логичких формула. Такав приступ показао се успешним на широком спектру проблема, од комбинаторних загонетки до задатака претраге и верификације.

Свођење проблема и коришћење општег решавача за циљну теорију није својствено само алату URSA. Оно лежи у основи бројних алата који уз помоћ решавача обављају верификацију, синтезу и отклањање грешака. Најзахтевнији део у развоју таквих алата јесте превод програма у формулу коју решавач испитује. Управо из те тешкоће настала је класа доменски специфичних језика потпомогнутих решавачима (енгл. *solver-aided domain-specific languages*), чији је најпознатији представник систем Rosette [37]. Сама идеја свођења на општи решавач шире је распрострањена: срећемо је у алатима попут MiniZinc, Sugar и Alloy [29, 36, 38], а алату URSA сродан је алат URBIVA, који проблеме своди на теорију бит-вектора и решава их SMT решавачем. О наведеним алатима детаљније је реч у одељку 2.3.

Алат URSA везан је за исказну логику као једини циљни формализам, што ограничава класу проблема који се могу природно изразити. Семантика аритметике бит-вектора рачуна по модулу, па не обухвата неограничене нити

означене целе бројеве, а индекси низова морају бити познати пре генерисања формуле, што искључује симболички приступ низовима. Ова ограничења не постоје генерално у контексту разматрања задовољивости у односу на теорију (SMT). Реч је о проширењу исказне задовољивости у ком се атомичке формуле тумаче у односу на теорије, попут теорије аритметике бит-вектора, линеарне аритметике целих бројева и теорије низова [5, 22]. Савремени SMT решавачи, међу којима су Z3 и cvc5, прихватају улаз у стандардном формату SMT-LIB и решавају га комбинацијом претраге на исказном нивоу и процедура одлучивања за поједине теорије.

Поменуто ограничење алата URSA није апстрактно, што показује неколико једноставних примера које изворно свођење на SAT не може да формулише, у неким случајевима због ограничења самог алата, а у неким због недовољне изражајности исказне логице.

Први пример је Фробенијусов проблем новчића (енгл. *Chicken McNugget*), питање колико се укупно комада може купити од паковања величине шест, девет и двадесет:

```
assert(na>=0 && nb>=0 && nc>=0 && 6*na + 9*nb + 20*nc == 43);
```

Над неограниченим целим бројевима број 43 се не може саставити, док бит-векторска аритметика по модулу 2^l даје „лажно” решење услед прекорачења.

Други пример је древни кинески проблем о остацима, из списка Сунзи:

```
assert_all(nx>0 && nx<105 && nx%3==2 && nx%5==3 && nx%7==2);
```

Задатак користи оператор остатка при дељењу, који изворно свођење на SAT не обухвата.

Трећи пример је Дијкстрино питање о упису и читању низа на индексу који је и сам вредност из тог низа:

```
nA[nA[1]] = 2;  
assert(nA[nA[1]] != 2);
```

Читање не мора вратити управо уписану вредност, јер упис може да промени сам индекс, а тај индекс није познат пре генерисања формуле.

Четврти пример је проблем збира два броја (енгл. *two-sum*), проналажење два индекса чији збир вредности даје задату суму:

```
nA[0]=3; nA[1]=6; nA[2]=2; nA[3]=7; nA[4]=8;  
assert_all(ni<5 && nj<5 && ni<nj && nA[ni]+nA[nj]==10);
```

И овде су индекси `ni` и `nj` симболички.

Ниједно од ових ограничења, осим Фробенијусовог проблема, не произлази из ограничења саме исказне логике. Докле год су вредности ограничене, и остатак при дељењу и приступ низу преко симболичког индекса могу се представити исказном формулом, што показују и постојећи алати који проблеме ограничења свode на SAT [36]. Такво кодирање, међутим, изворно свођење алата URSA на SAT не подржава. Фробенијусов проблем је другачији, јер над неограниченим доменом не може да се представи исказном формулом.

У овом раду описано је проширење алата URSA којим се улазни програм, поред досадашњег свођења на SAT, може свести и на SMT теорије. Симболичким извршавањем улазног програма граде се изрази одговарајуће теорије, који се записују као формула у формату SMT-LIB, у теорији аритметике бит-вектора (QF_BV) или линеарне аритметике целих бројева (QF_LIA), а у присуству симболичких индекса низова циљна теорија се аутоматски проширује одговарајућом варијантом са низовима, QF_ABV односно QF_ALIA¹. Тако добијене формуле решавају решавачи Z3 и `svcs5`, уз подршку за набрајање свих решења и, уколико је потребно, за оптимизацију. Досадашњи начин рада остаје непромењен и подразумеван, тако да раније написани програми раде без измена.

Проширење је евалуирано над постојећим примерима алата URSA и над новим примерима прилагођеним новоуведеним теоријама. Свођење на SMT даје иста решења као свођење на SAT на свим примерима одлучивања, чиме је

¹Ознаке QF_BV, QF_LIA, QF_ABV и QF_ALIA су називи фрагмената теорија у стандарду SMT-LIB, детаљније описаном у одељку 2.2.

потврђена његова исправност, и то за оба коришћена SMT решавача. Најзначајнији резултат јесте проширење класе проблема које алат може да изрази и реши, будући да свођење на SMT омогућава верификацију програма са симболичким приступом низовима, резонување над неограниченим и означеним целим бројевима, као и коришћење дељења и остатка. Мерења, с друге стране, показују да ново свођење не замењује досадашње него га допуњује, будући да ниједан приступ није ефикаснији на свим врстама проблема.

Наредно поглавље излаже потребне појмове, затим је описана имплементација проширења, а потом експериментална евалуација, након чега следе закључци и правци даљег рада².

²Током израде овог рада коришћени су велики језички модели (енгл. *large language models*, LLM). Улога им је била помоћна и техничке природе: преглед кода, аутоматско генерисање и варирање илустративних примера, помоћне скрипте за мерење времена, као и предлози при обликовању LaTeX табела и појединих формулација. Замисао проширења, имплементација, евалуација и закључци дело су аутора, који преузима пуну одговорност за садржај рада.

Глава 2

Основни појмови и преглед области

У овом поглављу излажемо основне појмове неопходне за праћење рада и разумевање предложеног проширења. Најпре разматрамо проблем исказне задовољивости (SAT), а затим SMT, задовољивост у односу на теорије, која уз исказну логику обухвата и теорије попут линеарне аритметике целих бројева, аритметике бит-вектора и низова. Потом прелазимо на симболичко извршавање и алате који на тим идејама почивају.

2.1 Проблем исказне задовољивости – SAT

Проблем испитивања задовољивости исказне формуле назива се SAT (скраћено од енглеског *Boolean satisfiability problem*). За дату исказну формулу, SAT решавач одлучује да ли је задовољива и, у случају да јесте, враћа валуацију при којој формула постаје тачна [22]. SAT решавачи по правилу раде са формулама у конјунктивној нормалној форми (КНФ), која представља конјункцију клауза, где је свака клауза дисјункција литерала, а литерал исказна променљива или њена негација [5].

Значај SAT проблема у рачунарству историјски произилази из Куковог рада [9], у којем је он показао да је SAT НП-комплетан, чиме је уведен и сам појам НП-комплетности (резултат познат и као Кук-Левинова теорема) [9, 23]. Упркос експоненцијалној временској сложености у најгорем случају, развој у овој области довео је до изузетно ефикасних SAT решавача. Савремени алати

данас обрађују КНФ формуле са стотинама хиљада или чак милионима променљивих [22].

Значајно унапређење практичне ефикасности SAT решавача постигнуто је прелазом са алгоритма DPLL (Davis-Putnam-Logemann-Loveland) [11], заснованог на систематичној претрази кроз простор валуација и пропагацији јединичних литерала, на напреднији приступ CDCL (скраћено од енглеског Conflict-Driven Clause Learning) [25]. За разлику од хронолошког враћања уназад које DPLL примењује у случају конфликта, CDCL алгоритам конструише граф импликација над тренутном валуацијом и на основу њега изводи узрок конфликта у виду нове клаузе [25]. Тако научене клаузе додају се у базу клауза и спречавају понављање истог конфликта у каснијим корацима претраге, чиме се у пракси значајно смањује простор претраге [22].

Ефикасност савремених CDCL решавача заснива се и на два додатна механизма: нехронолошким враћању уназад (енгл. *backjumping*) и хеуристикама за избор променљиве одлучивања, међу којима је најпознатија VSIDS (енгл. *Variable State Independent Decaying Sum*), уведена у решавачу Chaff [27]. Нехронолошко враћање омогућава алгоритму да поништи више нивоа одлучивања одједном, враћајући се на ниво одређен структуром новонаучене клаузе [25].

Дефинисање стандардизованог DIMACS формата за приказ формула у КНФ [14] омогућио је да произвољан комбинаторни проблем, након свођења на SAT, буде третиран било којим савременим решавачем, у начелу независно од домена из којег тај проблем потиче [5].

2.2 Проблем задовољивости у односу на теорију – SMT

SMT проблем (енгл. *Satisfiability Modulo Theories*) представља проширење проблема исказне задовољивости у коме је улазна формула изражена над једном или више теорија [4]. Формално, за задату формулу φ и теорију T , SMT решавач одлучује да ли постоји валуација променљивих која формулу φ чини тачном у неком моделу теорије T . За разлику од исказне задовољивости, мо-

дел овде интерпретира и симболе теорије, попут аритметичких операција и релација, па чини целу структуру у односу на коју се формула тумачи, а не само доделу вредности променљивима. Многе SMT теорије, попут линеарне аритметике целих бројева, задате су семантички, а не аксиоматски, као скупови затворених формула тачних у некој фиксираној структури, па имају само тај, природни модел.

Увођење SMT приступа мотивисано је ограниченом изражајношћу исказне логике. Бројни задаци у верификацији софтвера и хардвера, планирању и симболичкој анализи природно се формулишу над богатијим доменима као што су цели бројеви, бит-вектори или низови. Директан превод оваквих проблема у исказну задовољивост резултирао би формулама екстремних димензија и недовољном практичном ефикасношћу [5, 22].

Савремени SMT решавачи најчешће комбинују SAT алгоритам са специјализованим процедурама одлучивања за конкретне теорије у оквиру архитектуре CDCL(T) [30]. У овом приступу, свака атомичка формула репрезентује се јединственом исказном променљивом како би се добила исказна апстракција оригиналне формуле. Док SAT језгро одлучивања анализира задовољивост ове апстрактне форме, процедура одлучивања за теорију T инкрементално проверава да ли је текућа парцијална валуација исказних променљивих конзистентна са законима саме теорије [5, 30]. Ако се открије неконзистентност, теоријска процедура генерише нову клаузу која се додаје у SAT језгро, што резултира сужавањем простора претраге. Детаљан преглед конкретних алгоритамских техника за процедуре за појединачне теорије излази из опсега овог рада [5, 30].

У циљу стандардизације улазног формата и лакшег поређења различитих имплементација, развијен је формални језик SMT-LIB [3]. Овај стандард дефинише јединствену синтаксу за опис инстанци проблема, обухвата одређен скуп стандардних теорија и дефинише класе теорија на основу комбинација ових теорија уз евентуална ограничења у присуству квантификатора. У стандарду SMT-LIB такве комбинације теорија називају се логикама, али у овом раду за њих доследно користимо термин теорија, подразумевајући логику првог реда у фрагменту без квантификатора. Као примери непо-

средно релевантни за овај рад издвајају се QF_BV (аритметика бит-вектора) и QF_LIA (линеарна целобројна аритметика), као и њихове варијанте са теоријом низова: QF_ABV (аритметика бит-вектора са низовима) и QF_ALIA (линеарна целобројна аритметика са низовима), сви у фрагментима без квантификатора [3]. Стандард SMT-LIB чини основу и за годишње такмичење SMT-COMP, чија методологија служи као уобичајени емпиријски оквир за упоредну евалуацију актуелних решавача [34].

Међу најзаступљенијим савременим SMT решавачима у научној и индустријској пракси данас се истичу Z3 [13], cvc5 [2] и MathSAT [8]. Наведени алати подржавају широк скуп теорија описаних у оквиру SMT-LIB стандарда, а избор конкретног решавача у практичним апликацијама примарно зависи од комбинације теорија присутних у проблему који се решава.

2.2.1 Теорија аритметике бит-вектора

Теорија аритметике бит-вектора представља формални оквир за резоновање над позитивним целобројним вредностима фиксне битне ширине, а њен фрагмент без квантификатора у SMT-LIB стандарду означен је као QF_BV [3]. Термови ове теорије представљају бит-векторе, односно низове битова фиксне ширине w , над којима су дефинисане стандардне операције рачунарске аритметике. У њих спадају целобројне рачунске операције попут сабирања, одузимања, множења, дељења и дељења са остатком, затим битовске операције конјункције, дисјункције, ексклузивне дисјункције и негације, као и операције битовског померања улево, логичког померања удесно и аритметичког померања удесно. Поред аритметичких операција, теорија обухвата означене и неозначене релације поређења, као и специфичне структурне операције као што су конкатенација (спајање вектора) и екстракција (издвајање подниза битова) [3, 22].

Семантика ових операција у потпуности одговара стандардној аритметици по модулу 2^w , где се прекорачење опсега природно регулише свођењем по модулу, док се означене вредности интерпретирају у репрезентацији потпуног комплемента. Оваквим избором репрезентације постиже се да иста хардверска реализација операција важи и за означену и за неозначену семантику код операција као што су сабирање, одузимање и множење [22]. Захваљујући

верном моделовању понашања хардвера на најнижем нивоу, теорија QF_BV представља примарни формални оквир за резонovanje у областима формалне верификације хардвера и статичке анализе програма на ниском нивоу апстракције [5, 22].

Стандардни приступ решавању формула у оквиру ове теорије заснива се на техници познатој као бит-бластовање (енгл. *bit-blasting*). Овај поступак сваку бит-векторску променљиву ширине w мења уређеним низом од w исказних променљивих, док се операције над бит-векторима преводе у одговарајућу мрежу логичких кола попут сабирача, множача и компаратора. Настала мрежа се потом своди на конјунктивну нормалну форму применом Цејтинове трансформације [22], чиме се одлучивост QF_BV проблема директно повезује са одлучивошћу SAT проблема. Иако оваква конверзија у општем случају генерише КНФ формуле чија димензија нелинеарно расте са повећањем ширине бит-вектора w , савремени решавачи примењују напредне методе препроцесирања на нивоу термова. Ове технике укључују презаписивање израза и локална поједностављивања, чиме се значајно унапређује практична ефикасност процеса решавања [5, 22].

2.2.2 Линеарна аритметика целих бројева

Линеарна аритметика целих бројева представља формални оквир за резонovanje над вредностима из скупа целих бројева, при чему су допуштене само линеарне комбинације променљивих, а њен фрагмент без квантификатора у SMT-LIB стандарду означен је као QF_LIA [3]. Термови ове теорије изражавају се као линеарне комбинације константи и целобројних променљивих, односно изрази облика $c_1x_1 + c_2x_2 + \dots + c_nx_n + d$, где су $c_i, d \in \mathbb{Z}$, а x_i означавају целобројне променљиве. Атомичке формуле састоје се од релација једнакости и поређења између оваквих израза: $=$, $\leq$, $<$, $\geq$ и $>$. Множење две променљиве не припада овом фрагменту, јер га чини нелинеарним, тако да је једини дозвољени облик множења онај у којем је један од чинилаца целобројна константа [3, 22].

Семантика ове теорије суштински се разликује од семантике теорије аритметике бит-вектора у томе што вредности променљивих припадају бесконачном скупу целих бројева $\mathbb{Z}$, без ограничења на конкретну битовску ширину,

док аритметичке операције имају уобичајено математичко значење без икаквих ограничења која намеће аритметика по модулу. Ова карактеристика омогућава прецизно моделовање проблема у којима су вредности природно неограничене, попут анализе неких алгоритама, планирања ресурса и одређених класа оптимизационих проблема [5, 22]. Са друге стране, одсуство ограничене ширине домена узрокује да теорија QF_LIA не одражава верно понашање операција у реалним рачунарским системима којима су вредности ограничене ширином машинске речи. Због тога је ова теорија мање погодна за верификацију на нивоу машинске аритметике у поређењу са претходно описаном теоријом QF_BV.

Одлучивост QF_LIA проблема представља директну последицу класичног резултата Презбургера, који је показао да је целобројна аритметика са сабирањем као једином аритметичком операцијом одлучива у општем случају [31]. У пракси, савремени SMT решавачи не примењују непосредно оригиналне алгоритме Презбургеровог типа, већ комбинују процедуре засноване на симплекс методи над рационалним бројевима са методама разграновања и ограничавања (енгл. *branch and bound*) ради проналажења целобројних решења [15, 12]. Класични алтернативни приступи, попут Куперове процедуре и Омега теста, такође су широко заступљени у литератури, а детаљан приказ ових метода излази из опсега овог рада [10, 32].

2.2.3 Теорија низова

Теорија низова представља формални оквир за резонување над функционалним пресликавањима из скупа индекса у скуп елемената, при чему су такви објекти у рачунарству познати као асоцијативни низови. У SMT-LIB стандарду, ова теорија је параметризована помоћу два домена: доменом индекса I и доменом елемената E , при чему се одговарајући тип низа означава као $\text{Array}(I, E)$ [3]. Основне операције ове теорије су читање вредности низа на задатом индексу и упис нове вредности на задати индекс. Ове операције се у оквиру стандарда означавају као *select* и *store* и имају следеће потписе:

$$\text{select} : \text{Array}(I, E) \times I \rightarrow E,$$

$$\text{store} : \text{Array}(I, E) \times I \times E \rightarrow \text{Array}(I, E).$$

Семантика ове теорије формално је задата аксиоматизацијом коју је Џон МекКарти увео 1962. године, а која се данас користи као стандардна формулација у савременој SMT литератури [26, 22]. Прва од две основне аксиоме гласи да читање са индекса на који је вредност непосредно уписана враћа ту исту вредност:

$$\forall a, i, v. \text{select}(\text{store}(a, i, v), i) = v,$$

док друга аксиома тврди да упис на један индекс не мења вредности низа на свим осталим индексима:

$$\forall a, i, j, v. i \neq j \implies \text{select}(\text{store}(a, i, v), j) = \text{select}(a, j).$$

Савремене имплементације у оквиру SMT-LIB стандарда типично проширују овај скуп аксиомом екстензионалности, која формализује да су два низа идентична ако и само ако су им једнаке вредности на свим индексима [4, 22]. Оваква семантика успешно обухвата суштинско понашање низова у реалним програмским језицима и чини математичку основу за моделовање меморије, показивача и комплексних структура података у процесу формалне верификације софтвера [22].

Фрагмент без квантификатора ове теорије у пракси се готово увек комбинује са другим позадинским теоријама. За овај рад релевантне су комбинације са аритметиком бит-вектора и са линеарном аритметиком целих бројева, које дају теорије QF_ABV односно QF_ALIA.

Са алгоритамске стране, одлучивање задовољивости QF_ABV и QF_ALIA формула у савременим SMT решавачима заснива се на комбиновању наменске процедуре за низове са процедуром одговарајуће подтеорије индекса и елемената. Уобичајени приступ у пракси полази од почетног апстраховања операција *select* и *store* (увођењем неинтерпретираних функцијских симбола), уз накнадно лењо инстанцирање МекКартијевих аксиома кроз генерисање лема у зависности од конкретних индекса који се појављују у формули [35, 4]. Формална повезаност и дељење једнакости између теорије низова и њених подтеорија остварује се у складу са Нелсон-Опеновом методом комбиновања одлучивих теорија [28, 22].

2.3 Решавачи и симболичко извршавање

Свођење проблема ограничења на проблеме задовољивости представља приступ у коме се, уместо развоја наменског алгоритма, проблем изражава као формула чију задовољивост испитује општи решавач. Идеја је заступљена у многим системима. Језик MiniZinc омогућава опис инстанци проблема на вишем нивоу апстракције и његово превођење у међуформат FlatZinc, за који постоје позадински решавачи засновани и на SAT и на SMT [29]. Алат Sugar своди проблеме задовољавања ограничења на исказну задовољивост одговарајућим кодирањем променљивих и ограничења [36]. Систем Alloy проблеме над релационом логиком своди на исказну задовољивост посредством алата Kodkod [38]. Заједничко овим приступима јесте да се решавање препушта општим решавачима, чиме се избегава развој наменског алгоритма за сваку класу проблема. У наставку се пажња усмерава на системе који формулу за решавач граде симболичким извршавањем улазног програма, приступ на коме почивају и алати URSA и URBIVA.

2.3.1 Симболичко извршавање

Симболичко извршавање (енгл. *symbolic execution*) је техника анализе програма у којој се, уместо конкретних улазних вредности, користе симболичке променљиве које уместо конкретних вредности репрезентују читаве скупове могућих улаза [21]. Извршавајући програм над таквим симболима, поступак прати услове које улази морају да задовоље да би се стигло до одређене тачке у програму. За сваку путању кроз програм тако се добија услов над симболичким улазима, познат као услов пута (енгл. *path condition*), а скуп услова пута описује понашање програма на симболичком нивоу.

Ова техника највише се користи у анализи и верификацији софтвера. Алати засновани на њој читају стварни програмски код, извршавају га симболички и генеришу ограничења која карактеришу исправност или неисправност програма, попут услова под којима се достиже нека грешка. Тако добијена ограничења прослеђују се решавачима, најчешће SMT решавачима, који испитују њихову задовољивост. Међу познатим алатима овог типа су DART [16], CUTE [33], EXE [7], KLEE [6] и SAGE [17]. У свима њима симболичко извршавање служи као начин да се из постојећег кода изведу

ограничења која се затим предају решавачу. Алати за решавање проблема ограничења URSA и URBIVA, међутим, симболичким извршавањем задају сама ограничења, а не изводе их из постојећег кода, о чему је реч у одељку 2.3.

2.3.2 Доменски специфични језици потпомогнути решавачима

Посебан вид спајања решавача и симболичког извршавања јесу доменски специфични језици потпомогнути решавачима (енгл. *solver-aided domain-specific languages*). Реч је о наменским језицима који аутоматске SAT и SMT решаваче користе за задатке попут верификације, синтезе и отклањања грешака, тако што програм симболички извршавају и добијена ограничења прослеђују решавачу [37]. Сам термин уводе и претежно користе аутори система Rosette, па још није општеприхваћен, али сажето описује класу система којој припадају и алати обрађени у наставку.

2.3.3 Систем Rosette

Rosette је језик потпомогнут решавачима, изграђен као проширење језика опште намене Racket [37]. Уместо да буде самосталан алат, он додаје Racket-у симболичке вредности и конструкције за верификацију и синтезу, па се програми пишу у самом Racket-у. У основи система је симболичка виртуелна машина која симболички извршава Racket програм и из њега изводи ограничења, која потом решава SMT решавач. Тиме Rosette служи као оквир (енгл. *framework*) заснован на приступу уградње у језик-домаћин.

Начин рада система Rosette илуструјемо примером из његове документације. Корисник дефинише функције у стандардној синтакси језика Racket, а затим уводи симболичке променљиве оператором `define-symbolic`. Позивом упита `verify` систем извршава програм, прикупља услове путање и преводи их у SMT формулу.

```
; midpoint index, as in binary search
(define (mid lo hi)
  (quotient (+ lo hi) 2))
```

```
; symbolic integer variables
(define-symbolic l h integer?)

; verification: is the midpoint always between l and h (assuming 0 <= l <= h)?
(define cex
  (verify
    (begin
      (assume (<= 0 l h))
      (assert (<= l (mid l h) h))))))
```

Упит `verify` у овом примеру проверава да ли је средишњи индекс увек између `l` и `h`, уз претпоставку (`assume`) да је $0 \leq l \leq h$. Над неограниченим целим бројевима својство важи, па решавач не проналази контрапример. Ако би се, међутим, `l` и `h` представили 32-битним бит-векторима, збир `(+ l h)` могао би да прекорачи опсег, услед чега би решавач пронашао контрапример.

Поред верификације, Rosette подржава и синтезу кода упитом `synthesize`. У наредном примеру корисник оставља „рупу” (енгл. *hole*) конструкцијом `(?? integer?)`, а Rosette аутоматски проналази целобројну константу која чини програм исправним за све улазе.

```
(require rosette/lib/synthax)

; symbolic input x
(define-symbolic x integer?)

; synthesis: fill the hole (?? integer?) so that x + hole equals x + 5, for all
; x
(define synth-result
  (synthesize
    #:forall (list x)
    #:guarantee (assert (= (+ x (?? integer?)) (+ x 5)))))
```

Оба примера показују како Rosette истим механизмом симболичког извршавања решава различите класе упита. Корисник пише код у језику-домаћину, а систем аутоматски гради формуле и прослеђује их позадинском SMT решавачу. За разлику од Rosette, алати URSA и URBIVA, изложени у наставку, самостални су алати са сопственим језиком спецификације.

2.3.4 Алат URSA

URSA (енгл. *Uniform Reduction to SAT*) је алат за униформно свођење комбинаторних проблема на проблем исказне задовољивости, детаљно описан у раду [20]. Основна замисао система јесте да се омогући решавање широког спектра комбинаторних проблема и проблема задовољивости ограничења (енгл. *Constraint Satisfaction Problems - CSP*) кроз њихово свођење на проблем исказне задовољивости. Овај приступ уклања потребу за развојем специјализованих алгоритама за сваку класу проблема појединачно. Улаз у систем чини спецификација проблема записана у наменском језику, док је крајњи излаз изражен у облику исказне формуле у конјунктивној нормалној форми у стандардном DIMACS формату, која се даље прослеђује SAT решавачу [20].

Језик

Језик алата URSA обликован је као наменски програмски језик који комбинује карактеристике императивне и декларативне програмске парадигме. Императивни карактер језика огледа се у подршци за стандардне контролне структуре, као што су условни изрази и петље. Са друге стране, језик је суштински декларативан јер корисник не дефинише алгоритам за проналажење решења, већ само описује услове и ограничења које то решење мора да испуни [20]. Свака нумеричка променљива у програму обавезно започиње префиксом **n**, док исказне, Булове променљиве садрже префикс **b**, чиме је типизирање експлицитно спроведено на нивоу самих идентификатора. Језик подржава рад са једнодимензионалним и вишедимензионалним низовима, уз услов да се за индексирање користе изрази чија је конкретна вредност детерминисана пре самог генерисања формула [20]. Задавање логичких ограничења врши се помоћу наредбе **assert**, док се наредба **assert_all** користи када је потребно пронаћи скуп свих задовољавајућих валуација. За решавање проблема оптимизације предвиђене су конструкције **minimize** и **maximize**, које идентификују циљну променљиву чију вредност треба оптимизовати унутар задатог интервала [20]. Пример кратког URSA програма који описује систем двеју линеарних једначина са две непознате приказан је у наставку:

```
b1 = (3*nx+4*ny==25);
b2 = (4*nx+3*ny==24);
```

```
assert_all(b1 && b2);
```

Потпунији преглед синтаксе и конструкција језика дати су у додатку А.

Свођење на SAT

Основна карактеристика алата URSA јесте да све нумеричке променљиве интерно представља као бит-векторе фиксне ширине w . Ову ширину корисник задаје преко параметра командне линије `-l` [20]. Аритметичке и битовске операције изражавају се преко одговарајућих логичких формула на нивоу појединачних битова, што директно одговара поступку бит-бластовања описаном у одељку 2.2.1.

Симболично извршавање улазног програма гради исказну формулу која представља конјункцију свих условних ограничења изведених из корисничких `assert` конструката. Тако добијена формула се затим Цејтиновом трансформацијом преводи у конјунктивну нормалну форму ради даље обраде од стране SAT решавача [20].

Интеграција са SAT решавачима

Систем URSA не имплементира сопствени SAT решавач, већ након генерисања релевантних клауза позива неки од уграђених решавача. Тренутно подржани решавачи су `clasp`, `MiniSAT` и `ArgoSAT`, а корисник избор врши опцијом `-s` на командној линији, док је подразумевани решавач `clasp` [20]. У случају да је формула задовољива, добијена валуација пресликава се назад у именоване променљиве URSA програма, чиме корисник добија модел у облику вредности изворних променљивих, а не у виду сирових исказних променљивих. У случају када се тражи потпуни скуп решења применом конструкта `assert_all`, URSA понавља позив решавачу уз додавање блокирајућих клауза које искључују већ пронађене моделе, све док решавач не пријави незадовољивост [20].

2.3.5 Алат URBIVA

URBIVA (енгл. *Uniform Reduction to Bit-Vector Arithmetic*) је алат аналоган алату URSA, али се у оквиру њега проблеми своде на аритметику

бит-вектора, односно на SMT теорију QF_BV , уместо на исказну задовољност [24]. Оба алата деле исту примарну мотивацију: униформно моделовање и решавање широког спектра рачунарски захтевних проблема у областима верификације хардвера и софтвера, као и комбинаторне оптимизације.

Језик алата URBIVA је наменски програмски језик синтаксички налик језику C, у коме се разликују нумеричке променљиве, чија имена почињу префиксом `n`, и Булове променљиве, чија имена почињу префиксом `b`. Ограничења проблема постављају се наредбама `assert` и `assert_all`, при чему прва тражи неку задовољавајућу валуацију, а друга скуп свих таквих валуација. Језик подржава стандардне структуре контроле тока и кориснички дефинисане функције, уз ограничење да услови гранања и индекси низова морају бити познати пре генерисања формуле [24]. Симболично извршавање улазног програма гради формулу над теоријом аритметике бит-вектора, а резултат се прослеђује једном од подржаних решавача, међу којима су Boolector, MathSAT и Yices. Поред њих, систем подржава и сопствену имплементацију засновану на бит-бластовању уз позивање SAT решавача ArgoSAT [24]. Комуникација са решавачима врши се непосредно преко њихових програмских интерфејса (API), чиме се избегава потреба за записивањем формула у међуформату попут SMT-LIB датотека.

Однос ова два алата непосредно је релевантан за предмет ове тезе. URBIVA показује да је концепт униформног свођења примењив и на теорије богатије од исказне логике, али он остаје ограничен само на теорију аритметике бит-вектора. У закључку рада о алату URBIVA аутори указују на могући правац даљег развоја у виду ширег система, под радним називом URSA MAJOR, који би подржао свођење на различите SMT теорије [24]. Рад у оквиру ове тезе реализује такво проширење, али кроз надградњу постојећег алата URSA и задржавање његовог језика, при чему се кроз генерисање SMT формула обухвата већи број теорија, прецизније теорије QF_BV , QF_LIA , QF_ABV и QF_ALIA .

За разлику од алата за симболично извршавање из одељка 2.3.1, који ту технику примењују да из постојећег кода изведу ограничења, алати URSA и URBIVA користе је обрнуто, као начин да се ограничења задају. Њихов

улазни програм није софтвер који се проверава, већ спецификација проблема, а његовим симболичким извршавањем гради се формула чију задовољивост решавач испитује.

Глава 3

Проширење система URSA подршком за SMT решаваче

Алат URSA, представљен у поглављу 2.3.4, сваки проблем своди на исказну задовољивост, при чему се све вредности представљају бит-векторима фиксне ширине [20]. Овакав приступ показао се успешним за широк спектар комбинаторних проблема, али везаност за исказну логику као једину циљну теорију ограничава класу проблема који се могу природно изразити. Проширење описано у овом поглављу омогућава да се улазни програм сведе и на богатије SMT теорије. Међу њима се издвајају резоновање над неограниченим целим бројевима и индексирање низова изразима чија вредност није унапред позната, што претходно није било подржано. Решавање ограничења добијених симболичким извршавањем препушта се постојећим SMT решавачима. Проширење је реализовано као надградња постојећег алата уз задржавање улазног језика¹, чија се архитектура и подршка за поједине теорије излажу у наставку поглавља.

3.1 Архитектура имплементације

Проширење је осмишљено тако да постојећа функционалност алата URSA остане непромењена, а да се подршка за SMT реализује као засебан начин рада који са изворним начином дели улазни део система. Читање програма и његово превођење у интерну репрезентацију остају непромењени, будући да

¹Изворни код проширења и сви примери коришћени у раду доступни су на <https://github.com/marijaa010/URSA>.

су језик и његова обрада заједнички за оба начина рада. Избор између изворног свођења на SAT и новог свођења на SMT врши се опцијом у командној линији, при чему подразумевано понашање остаје свођење на SAT, тако да раније написани програми раде без измена.

При свођењу на SMT, изведено ограничење записује се као формула у стандардном формату SMT-LIB. Формула се може исписати на стандардни излаз или записати у датотеку, ради прегледа, чувања и поновне употребе, чиме остаје применљива на сваки решавач који подржава одговарајућу теорију. У подразумеваном начину рада, међутим, алат сам решава добијену формулу, без ослањања на спољашњи решавач. У ту сврху су решавачи Z3 и `smt5` укључени као библиотеке и позивају се преко свог програмског интерфејса. Формула се притом не записује у датотеку, него се као ниска у формату SMT-LIB предаје непосредно програмском интерфејсу решавача, који је обрађује у меморији. Алат затим покреће проверу задовољивости и чита добијени модел. Када се тражи скуп свих решења, она се набрајају додавањем блокирајућих клауза, слично поступку који URSA примењује при свођењу на SAT, и приказују се у истом облику. Тиме алат остаје независан од конкретног решавача, будући да оба решавача прихватају исту формулу у формату SMT-LIB.

Свођење на SMT заснива се, као и у основној верзији, на симболичком извршавању улазног програма. Уместо да рачуна конкретне вредности, алат за сваки израз гради његов симболички облик, при чему се променљиве које се читају пре него што им је вредност додељена препознају као слободне. Тако изграђени изрази чине заједничку интерну репрезентацију из које се производе декларације слободних променљивих, тврдње изведене из ограничења и завршне команде за проверу задовољивости и добијање модела.

Иста процедура симболичког извршавања користи се за све подржане теорије. Разлика између циљних теорија локализована је на избор врсте израза који се гради и типа који се придружује слободним променљивама, док остатак поступка остаје заједнички. Циљна теорија бира се опцијом командне линије, а у случају када програм користи низове са симболичким индексима, циљна теорија се аутоматски проширује одговарајућом варијантом са низо-

вима. Појединости за сваку теорију излажу се у наставку поглавља.

3.2 Организација кода

Проширење уводи мали скуп нових класа, организованих око јединствене репрезентације израза. Класе `SMTExpr` и `SMTFactory` чине ту репрезентацију, усмерени ациклични граф чворова са дељењем заједничких подизраза. Класе `SMTNumber` и `SMTBoolean` су типизирани омотачи над чворовима графа, који нуде исти јавни интерфејс као класе `Number` и `Boolean` са стране свођења на SAT, тако да процедура симболичког извршавања задржава исти облик у оба начина рада. Класа `SMTInterpreter` управља симболичким извршавањем и исписом формуле, док класа `SMTSolverDriver` изграђену формулу предаје решавачу.

Сваки чвор графа је инстанца класе `SMTExpr`, а његову врсту одређује енумерација `SMTNodeType`. Чворови су груписани по теоријама: породица `BV_*` одговара операцијама теорије `QF_BV`, породица `INT_*` операцијама теорије `QF_LIA`, док су `BOOL_*`, `ARRAY_*` и `SMT_ITE` независни од базне теорије:

```
enum SMTNodeType {
    BV_CONST, BV_VAR,                // bit-vectors (QF_BV)
    BV_ADD, BV_SUB, BV_MUL, BV_NEG,
    BV_AND, BV_OR, BV_XOR, BV_NOT,
    BV_SHL, BV_LSHR, BV_UDIV, BV_UREM,
    BV_EQ, BV_ULT, BV_ULE, BV_UGT, BV_UGE,

    BOOL_CONST, BOOL_VAR, BOOL_AND,    // Boolean operators
    BOOL_OR, BOOL_XOR, BOOL_NOT, BOOL_EQ,

    INT_CONST, INT_VAR,                // integers (QF_LIA)
    INT_ADD, INT_SUB, INT_MUL, INT_NEG,
    INT_DIV, INT_MOD,
    INT_LT, INT_LE, INT_GT, INT_GE, INT_EQ,

    ARRAY_VAR, ARRAY_SELECT, ARRAY_STORE, // arrays (QF_ABV / QF_ALIA)

    SMT_ITE                            // if-then-else
};
```

Чворови се никада не стварају непосредно, већ искључиво преко класе `SMTFactory`, која спроводи дељење заједничких подизраза, тако да структурно једнаки подизрази деле један чвор. За сваку породицу постоји низ одговарајућих конструктора, а увођење нове теорије своди се на додавање врста у енумерацију, конструктора у фабрику и одговарајуће гране у испис.

```
class SMTFactory {
public:
    // bit-vector constructors (QF_BV)
    static SMTEExpr* makeBvConst(uint64_t value, int width);
    static SMTEExpr* makeBvVar(const std::string& name, int width);
    static SMTEExpr* makeBvAdd(SMTEExpr* a, SMTEExpr* b);
    // ... remaining bit-vector operations

    // integer constructors (QF_LIA)
    static SMTEExpr* makeIntConst(uint64_t value);
    static SMTEExpr* makeIntAdd(SMTEExpr* a, SMTEExpr* b);
    static SMTEExpr* makeIntMod(SMTEExpr* a, SMTEExpr* b);
    // ...

    // array constructors (extend the logic to QF_ABV / QF_ALIA)
    static SMTEExpr* makeArrayVar(const std::string& name,
                                   int indexWidth, int elementWidth);
    static SMTEExpr* makeArraySelect(SMTEExpr* array, SMTEExpr* index);
    static SMTEExpr* makeArrayStore(SMTEExpr* array, SMTEExpr* index,
                                     SMTEExpr* value);

    static void clear();           // empty the cache at shutdown
    static size_t cacheSize();    // number of unique nodes
};
```

Класа `SMTNumber` представља нумеричку вредност и постоји у два облика: као позната вредност, конкретна вредност (енгл. *ground*), израчуната у тренутку евалуације, или као симболички израз, показивач на чвор графа. Тај двоструки облик омогућава израчунавање константних подизраза током фазе превођења. Када су оба операнда позната, резултат се одмах израчунава, а тек ако је бар један симболички, гради се нови чвор. Избор породице чвора, `BV_*` или `INT_*`, зависи од циљне теорије, што илуструје тело оператора сабирања.

```
SMTNumber SMTNumber::operator+(const SMTNumber& other) const {
    if (m_isGround && other.m_isGround) {
        if (isLIAMode()) return liaGround(m_groundBig + other.m_groundBig);
        return SMTNumber(maskTo(m_groundValue + other.m_groundValue, m_width),
m_width);
    }
    if (m_isGround && (isLIAMode() ? m_groundBig == 0 : m_groundValue == 0))
return other;
    if (other.m_isGround && (isLIAMode() ? other.m_groundBig == 0 : other.
m_groundValue == 0)) return *this;
    SMTExpr* node = isLIAMode() ? SMTFactory::makeIntAdd(m_expr, other.m_expr)
                                : SMTFactory::makeBvAdd(m_expr, other.m_expr);
    return SMTNumber(node);
}
```

Константне вредности у теорији QF_LIA су цели бројеви произвољне величине, па се и константе записане у програму и резултати израчунавања константних подизраза добијају без губитка тачности. У теорији бит-вектора вредности задржавају фиксну ширину, коју та теорија захтева. Неограниченост целих бројева у теорији QF_LIA тако важи и за слободне променљиве, чије вредности решавач бира без ограничења ширине, и за константе познате у фази превођења. За рад са целим бројевима произвољне величине алат користи библиотеку GMP [18]. Свођење на SAT и на теорију бит-вектора од ње не зависи, па се она може изоставити, док је теорија QF_LIA без ње недоступна, о чему алат обавештава одговарајућом поруком.

Иста класа контролише и ограничења линеарности у режиму QF_LIA. Операције које нарушавају линеарност, попут множења две симболичке вредности или остатка са симболичким делиоцем, као и битовске операције и померања (енгл. *shift*), пријављују грешку још у току превођења. Класа `SMTBoolean` игра аналогну улогу за истинитосне вредности, при чему поређења над `SMTNumber` враћају управо `SMTBoolean`.

Класа `SMTInterpreter` управља превођењем. Пролази кроз наредбе програма и методама `ReadNumber` и `ReadBoolean` гради изразе типа `SMTNumber` и `SMTBoolean`. Метода `SolveConstraint` прикупља ограничења у једну тврдњу и исписује завршну формулу, декларације слободних променљивих, тврдње и

команде (`check-sat`) и (`get-model`), уз евиденцију о томе да ли је тражен скуп свих решења и да ли програм садржи захтев за оптимизацијом.

Најзад, класа `SMTSolverDriver` изграђену формулу у формату SMT-LIB предаје решавачу Z3 или `svcs5` преко његовог програмског интерфејса, покреће проверу задовољивости и исписује моделе у истом облику као у случају свођења на SAT. Набрајање свих решења изводи додавањем блокирајућих клауза, а оптимум пријављује само уз решавач Z3, будући да `svcs5` не подржава оптимизацију. Кôд специфичан за поједини решавач преводи се условно, у зависности од тога који су решавачи укључени при изградњи алата.

3.3 Подршка за QF_BV

Теорија аритметике бит-вектора је подразумевана циљна теорија при свођењу на SMT и природно полазиште овог проширења, будући да непосредно одговара начину на који URSA већ представља нумеричке вредности, као бит-векторе фиксне ширине. Уместо да изведена ограничења бит-бластовањем сведе на конјунктивну нормалну форму и проследи их SAT решавачу, алат у овом режиму симболичким извршавањем улазног програма гради изразе над бит-векторима фиксне ширине w и записује их као формулу у логици QF_BV. Ширина w задаје се истим параметром `-1` командне линије који URSA користи при свођењу на SAT, тако да остаје под контролом корисника.

Семантика операција над бит-векторима у потпуности прати семантику коју URSA примењује при свођењу на SAT. Све нумеричке вредности тумаче се као неозначени бит-вектори, а аритметичке операције дефинисане су по модулу 2^w , тако да се прекорачење опсега регулише свођењем по том модулу. Тиме исти улазни програм у оба режима описује исти скуп решења, што омогућава њихово непосредно поређење у поглављу 4. У складу са неозначеном семантиком, све релације поређења су неозначене.

Свођење на SMT спроводи специјализована процедура симболичког извршавања, која са свођењем на SAT дели заједнички улазни део система, читање и анализу улазног програма и креирање његове интерне репрезентације. Свођење на циљну формулу је различито и засебно за SAT и SMT, чиме постојећа

функционалност алата остаје неизмењена. Ова процедура пролази кроз наредбе програма и, уместо исказних формула, конструише изразе теорије бит-вектора. Променљиве које се читају пре него што им је вредност додељена препознају се као слободне и за њих се генеришу декларације бит-вектора одговарајуће ширине. Ограничења наведена наредбом **assert** прикупљају се и записују као тврдње у излазној формули.

Скуп подржаних операција приказан је у табели 3.1. Обухваћене су аритметичке операције, укључујући целобројно дељење и остатак при дељењу, битовске операције, померања (енгл. *shift*), неозначене релације поређења, једнакост и условни израз, као и логички везници над формулама.

Оператор (SMT-LIB)	Значење
bvadd	сабирање
bvsub	одузимање
bvmul	множење
bvneg	унарна негација
bvudiv	неозначено дељење
bvurem	неозначени остатак при дељењу
bvand	битовска конјункција
bvor	битовска дисјункција
bvxor	битовска ексклузивна дисјункција
bvnot	битовска негација
bvshl	померање улево
bvlshr	логичко померање удесно
bvult	неозначено поређење „мање“
bvule	неозначено поређење „мање или једнако“
bvugt	неозначено поређење „веће“
bvuge	неозначено поређење „веће или једнако“
=	једнакост
ite	условни израз
and	логичка конјункција
or	логичка дисјункција
xor	логичка ексклузивна дисјункција
not	логичка негација

Табела 3.1: Оператори теорије QF_BV које алат генерише.

Приликом генерисања формуле примењује се неколико поступака чији је циљ смањење величине излаза и растерећење решавача од тривијалних из-

рачунавања. Први је израчунавање константних подизраза током самог превода, као што је већ споменуто раније. Када су обе вредности познате у тренутку евалуације, операција се извршава одмах, уз свођење по модулу 2^w , па се у формули појављује само израчуната константа, а не и сама операција. Други поступак чине алгебарска поједностављивања, применом идентитета попут $x + 0 = x$, $x \cdot 1 = x$ и $x \cdot 0 = 0$, као и одговарајућих идентитета за битовске операције, чиме се уклањају сувишни чворови. Трећи поступак је дељење заједничких подизраза (енгл. *hash-consing*), при чему се структурно једнаки подизрази представљају једним чвором усмереног ацикличног графа, па се вишеструко коришћење истог израза не умножава. Четврти поступак примењује се при испису формуле, где се подизрази који се појављују више пута именују `let` везивањем, док се асоцијативни везници исписују у n -арном облику, што додатно смањује текстуалну величину излаза.

Као илустрацију, размотримо систем двеју линеарних једначина са две непознате, наведен као пример URSA програма у поглављу 2:

```
b1 = (3*nx+4*ny==25);
b2 = (4*nx+3*ny==24);
assert(b1 && b2);
```

При подразумеваној ширини од осам битова, овај програм се у SMT режиму преводи у следећу формулу у логици QF_BV:

```
(set-logic QF_BV)
(declare-fun nx () (_ BitVec 8))
(declare-fun ny () (_ BitVec 8))
(assert (and
  (= (bvadd (bvmul #x03 nx) (bvmul #x04 ny)) #x19)
  (= (bvadd (bvmul #x04 nx) (bvmul #x03 ny)) #x18)))
(check-sat)
(get-model)
```

Слободне променљиве `nx` и `ny` декларисане су као бит-вектори ширине осам, константе се исписују у хексадекадном облику (на пример, `#x19` одговара вредности 25), а ограничења су записана непосредно операцијама `bvadd`, `bvmul` и `=`. Даље решавање, укључујући и бит-бластовање, у потпуности преузима SMT решавач.

3.4 Подршка за QF_LIA

Поред теорије бит-вектора, SMT режим подржава и линеарну аритметику целих бројева, теорију QF_LIA. Овај режим бира се опцијом `-smtlogic=QF_LIA` у командној линији, која истовремено укључује и свођење на SMT. За разлику од теорије бит-вектора, у којој вредности припадају коначном опсегу одређеном ширином w , овде се нумеричке вредности тумаче као елементи неограниченог скупа целих бројева, у складу са теоријским разматрањем изложеним у одељку посвећеном линеарној аритметици целих бројева.

Подршка за QF_LIA не уводи засебну процедуру симболичког извршавања, већ у потпуности користи инфраструктуру описану у одељку 3.3, укључујући усмерени ациклични граф израза, дељење заједничких подизраза и израчунавање константних подизраза током превода. Разлика је локализована на избор врсте чворова који се граде за нумеричке изразе. Уместо бит-векторских чворова, у овом режиму се граде чворови целобројне аритметике, а слободне променљиве декларишу се типом `Int`. Целобројне константе исписују се у декадном облику.

Семантика се разликује од теорије бит-вектора на неколико суштинских начина. Вредности су означене, па променљива може узети и негативну вредност, услед чега ограничење попут $nx < 0$ може бити задовољиво. Аритметичке операције немају свођење по модулу, тако да се вредности не одсецају на ширину w . На пример, целобројни литерал 300 задржава своју вредност, док би у теорији бит-вектора при ширини од осам битоа био сведен на 44. Релације поређења су означене, за разлику од неозначених релација у теорији бит-вектора. Због наведених разлика, исти улазни програм може имати различит скуп решења у два теоријама, што је особина коју треба имати у виду при избору циљне теорије.

У логици QF_LIA подржане су линеарне аритметичке операције, односно сабирање, одузимање, унарна негација и множење у коме је бар један чинилац константа, као и целобројно дељење и остатак при дељењу код којих делилац мора бити константа. Поред тога, подржане су означене релације поређења, једнакост, условни израз и логички везници над формулама. Операције које

немају одговарајући пандан у линеарној целобројној аритметици, битовске операције и померања, као и облици који нарушавају линеарност, попут множења двеју променљивих или дељења и остатка са симболичким делиоцем, не припадају овом фрагменту. Алат такве случајеве препознаје још у току превођења и пријављује грешку са јасним објашњењем, уместо да генерише формулу коју би потом решаваач одбио. Тиме се кориснику пружа прецизнија повратна информација о узроку грешке. Провере алгебарских идентитета примењују се пре провере линеарности, тако да се израз попут $0 \cdot nx \cdot ny$ прво поједностављује на нулу и не пријављује као недозвољено нелинеарно множење.

Као илустрацију, размотримо исти систем двеју линеарних једначина из одељка 3.3:

```
b1 = (3*nx+4*ny==25);
b2 = (4*nx+3*ny==24);
assert(b1 && b2);
```

Покренут у режиму QF_LIA, овај програм се преводи у следећу формулу:

```
(set-logic QF_LIA)
(declare-fun nx () Int)
(declare-fun ny () Int)
(assert (and
  (= (+ (* 3 nx) (* 4 ny)) 25)
  (= (+ (* 4 nx) (* 3 ny)) 24)))
(check-sat)
(get-model)
```

За разлику од превода у теорију QF_BV, слободне променљиве су сада целобројне, а константе су записане декадно. За овај конкретан програм обе теорије дају исто решење ($nx = 3$, $ny = 4$), будући да он припада заједничком подскупу двеју теорија, у коме нема ни прекорачења опсега ни операција специфичних за теорију аритметике бит-вектора. Уз `assert_all` у обе теорије добија се само то једно решење. У теорији QF_BV не настају додатна решења услед рачуна по модулу, јер је детерминанта матрице коефицијената (-7) узајамно проста са 2^8 , па је решење јединствено и по том модулу.

Као и у случају теорије бит-вектора, циљна теорија се аутоматски проширује када се уведе симболичко индексирање низова. У том случају се уместо QF_LIA бира теорија QF_ALIA, о чему је реч у одељку 3.5.2.

3.5 Подршка за низове са симболичким индексима

Језик алата URSA подржава рад са низовима, али при свођењу на SAT индекс мора бити израз чија је вредност позната пре генерисања формуле. Низ индексиран искључиво константним вредностима представља се тако што се сваки елемент третира као засебна скаларна променљива, што је приступ који URSA већ користи и који је задржан. Симболичко индексирање, у коме индекс зависи од слободне променљиве, при том свођењу није било подржано. Увођењем теорије низова ово ограничење се уклања.

Теорија низова, изложена у одељку посвећеном тој теорији, располаже операцијама читања `select` и уписа `store`, чије је понашање одређено Мек-Картијевим аксиомама. Операција `store` је функционална, односно враћа нови низ који се од полазног разликује само на једном индексу, не мењајући полазни низ.

Да би се очувала ефикасност у уобичајеном случају када се низ користи само са константним индексима, теорија низова уводи се тек по потреби. Све док је низ индексиран искључиво константама, он остаје представљен засебним елементима, без увођења теорије низова. У тренутку када се низ први пут прочита или упише симболичким индексом, он се материјализује, односно уводи се као јединствена променљива типа низа, а од тог тренутка сви приступи том низу изражавају се операцијама `select` и `store`. Вредности уписане константним индексима пре материјализације поново се исписују као почетне тврдње облика `(assert (= (select a k) v))`, чиме се повезују са материјализованим низом. Ако се иста ћелија упише више пута константним индексом, важи последњи упис, будући да је реч о обичном скалару који се преписује, па се у почетној тврдњи појављује само та последња вредност. Присуство макар једног материјализованог низа доводи до аутоматског проширења циљне теорије одговарајућом варијантом са низовима, QF_ABV

односно `QF_ALIA`, без интервенције корисника.

Читање низа симболичким индексом преводи се у операцију `select`, а упис у операцију `store`. Пошто алат статички одмотава све контролне структуре, будући да услови гранања и границе петљи морају бити познати пре генерисања формуле, сваки упис представља праволинијску доделу. Стање низа зато се прати у облику низа верзија, где сваки упис ствара нову верзију над претходном применом операције `store`, а наредна читања виде последњу верзију. Оваква репрезентација одговара статичкој једнострукој додели (енгл. *static single assignment*), при чему додатне конструкције за спајање грана нису потребне јер симболичких гранања нема.

Вишедимензионални низови представљају се низовима чији су елементи такође низови, тако да дводимензионални низ има тип `(Array I (Array I E))`. Читање елемената дводимензионог низа изражава се двоструком применом операције `select`, док се упис, будући да стандард не предвиђа непосредну операцију уписа над две димензије, изражава композицијом облика `store(a, i, store(select(a, i), j, v))`.

3.5.1 Подршка за `QF_ABV`

Када је базна теорија теорија аритметике бит-вектора, комбинација са теоријом низова даје теорију `QF_ABV`. И индекс и елемент представљени су бит-векторима ширине w . Као илустрацију, размотримо проблем проналажења индекса на коме се у датом низу налази задата вредност, који при свођењу на SAT није био подржан:

```
nA[0] = 10;
nA[1] = 20;
nA[2] = 30;
nA[3] = 40;

assert(nA[nj] == 30 && nj < 4);
```

Овај програм се преводи у следећу формулу у логици `QF_ABV`:

```
(set-logic QF_ABV)
(declare-fun nj () (_ BitVec 8))
```

```
(declare-fun nA () (Array (_ BitVec 8) (_ BitVec 8)))
(assert (= (select nA #x00) #x0a))
(assert (= (select nA #x01) #x14))
(assert (= (select nA #x02) #x1e))
(assert (= (select nA #x03) #x28))
(assert (and
  (= (select nA nj) #x1e)
  (bvult nj #x04)))
(check-sat)
(get-model)
```

Слободна променљива `nj` декларисана је као бит-вектор, а `nA` као низ над бит-векторима. Четири уписа константним индексима, изведена пре појаве симболичког индекса, исписана су као почетне тврдње, док је читање `nA[nj]` изражено операцијом `select`. Циљна теорија је аутоматски проширена на `QF_ABV`, а решавач за наведени низ проналази `nj = 2`. Наредба `assert_all` потврђује да је то једино решење.

Наредни пример илуструје Дијкстрино питање о упису и читању низа на индексу који је и сам вредност из тог низа:

```
nA[nA[1]] = 2;
assert(nA[nA[1]] != 2);
```

Ако је стара вредност `nA[1]` једнака 1, упис мења баш `nA[1]`, па се друго читање односи на `nA[2]`, које није дирано и може имати произвољну вредност. Одговарајућа формула у логици `QF_ABV` гласи:

```
(set-logic QF_ABV)
(declare-fun nA () (Array (_ BitVec 8) (_ BitVec 8)))
(declare-fun nA_1_ () (_ BitVec 8))
(assert (= (select nA #x01) nA_1_))
(assert (let (($s0 (store nA nA_1_ #x02)))
  (not (= (select $s0 (select $s0 #x01)) #x02))))
(check-sat)
(get-model)
```

Стара вредност на индексу 1 уведена је као слободна променљива `nA_1_`, упис постаје операција `store`, а угнежђено читање изражава се двоструким `select`. Решавач проналази управо контрапример у ком је `nA_1_` једнако 1.

Слично, проблем збира два броја (енгл. *two-sum*) тражи два различита индекса чији збир вредности даје задату суму:

```
nA[0]=3; nA[1]=6; nA[2]=2; nA[3]=7; nA[4]=8;
assert_all(ni<5 && nj<5 && ni<nj && nA[ni]+nA[nj]==10);
```

Вредности уписане константним индексима исписују се као почетне тврдње, док читања на симболичким индексима `ni` и `nj` постају операције `select`:

```
(set-logic QF_ABV)
(declare-fun ni () (_ BitVec 8))
(declare-fun nj () (_ BitVec 8))
(declare-fun nA () (Array (_ BitVec 8) (_ BitVec 8)))
(assert (= (select nA #x00) #x03))
(assert (= (select nA #x01) #x06))
(assert (= (select nA #x02) #x02))
(assert (= (select nA #x03) #x07))
(assert (= (select nA #x04) #x08))
(assert (and
  (bvult ni #x05)
  (bvult nj #x05)
  (bvult ni nj)
  (= (bvadd (select nA ni) (select nA nj)) #x0a)))
(check-sat)
(get-model)
```

Решавач проналази решења $(0, 3)$ и $(2, 4)$, чије вредности редом дају $3 + 7$ и $2 + 8$.

Дводимензионални низови обрађују се на аналоган начин. Следећи програм попуњава матрицу и потом тражи позицију задате вредности при симболичким индексима реда и колоне:

```
for (ni = 0; ni < 3; ni++)
  for (nj = 0; nj < 3; nj++)
    nM[ni][nj] = ni * 3 + nj;

assert(nM[nr][nc] == 5 && nr < 3 && nc < 3);
```

Одговарајућа формула у логици QF_ABV гласи:

```
(set-logic QF_ABV)
(declare-fun nc () (_ BitVec 8))
(declare-fun nr () (_ BitVec 8))
(declare-fun nM () (Array (_ BitVec 8) (Array (_ BitVec 8) (_ BitVec 8))))
(assert (= (select (select nM #x00) #x00) #x00))
(assert (= (select (select nM #x00) #x01) #x01))
(assert (= (select (select nM #x00) #x02) #x02))
(assert (= (select (select nM #x01) #x00) #x03))
(assert (= (select (select nM #x01) #x01) #x04))
(assert (= (select (select nM #x01) #x02) #x05))
(assert (= (select (select nM #x02) #x00) #x06))
(assert (= (select (select nM #x02) #x01) #x07))
(assert (= (select (select nM #x02) #x02) #x08))
(assert (and
  (= (select (select nM nr) nc) #x05)
  (bvult nr #x03)
  (bvult nc #x03)))
(check-sat)
(get-model)
```

Низ `nM` декларисан је као низ чији су елементи такође низови. Девет уписа константним индексима, изведених у петљама, исписано је као почетне тврдње, док је читање `nM[nr][nc]` изражено двоструком применом операције `select`. За наведену матрицу решавач проналази `nr = 1` и `nc = 2`, будући да је $1 \cdot 3 + 2 = 5$. Наредба `assert_all` потврђује да је то једино решење.

3.5.2 Подршка за QF_ALIA

Када се теорија низова комбинује са линеарном аритметиком целих бројева добија се теорија QF_ALIA. Механизам је истоветан оном описаном за теорију QF_ABV, а разлика је искључиво у типовима индекса и елемената. У режиму линеарне аритметике и индекс и елемент имају тип `Int`, па једнодимензионални низ има тип `(Array Int Int)`, а дводимензиони `(Array Int (Array Int Int))`. Константе у почетним тврдњама исписују се декадно.

Исти пример проналажења индекса, покренут у режиму линеарне аритме-

тике, преводи се у формулу у логици QF_ALIA:

```
(set-logic QF_ALIA)
(declare-fun nj () Int)
(declare-fun nA () (Array Int Int))
(assert (= (select nA 0) 10))
(assert (= (select nA 1) 20))
(assert (= (select nA 2) 30))
(assert (= (select nA 3) 40))
(assert (and
  (= (select nA nj) 30)
  (< nj 4)))
(check-sat)
(get-model)
```

У поређењу са теоријом QF_ABV, тип низа је (Array Int Int), индекси и вредности исписани су декадно, а поређење је означено. Семантичке разлике између двеју базних теорија, у погледу опсега, знака и свођења по модулу, преносе се и на низове, како је описано у одељку 3.4. Једна последица тога тиче се набрајања свих решења: пошто целобројни индекс није ограничен, наредба `assert_all` овде даје бесконачно много решења, будући да се за сваки $nj < 4$ на позицији коју низ не ограничава вредност може поставити на 30.

Наредни пример показује потребу за неограниченом целобројном аритметиком, Фробенијусов проблем новчића: могу ли се 43 комада саставити од паковања величине шест, девет и двадесет:

```
assert(na>=0 && nb>=0 && nc>=0 && 6*na + 9*nb + 20*nc == 43);
```

У логици QF_LIA формула гласи:

```
(set-logic QF_LIA)
(declare-fun na () Int)
(declare-fun nb () Int)
(declare-fun nc () Int)
(assert (and
  (>= na 0)
  (>= nb 0)
  (>= nc 0)
  (= (+ (+ (* 6 na) (* 9 nb)) (* 20 nc)) 43)))
```

```
(check-sat)
(get-model)
```

Слободне променљиве су целобројне, а множење константом и сабирање изражени су операцијама $*$ и $+$. Над неограниченим целим бројевима решења нема, што решавач потврђује, док изворно свођење на SAT, које бројеве представља бит-векторима по модулу 2^l , при $l = 8$ даје „лажно” решење услед прекорачења опсега.

Сличну потребу за целобројном семантиком показује и древни кинески проблем о остацима из списка Сунзи, наведен у уводу. Тражи се број који при дељењу са три, пет и седам даје редом остатке два, три и два:

```
assert_all(nx > 0 && nx < 105 && nx % 3 == 2 && nx % 5 == 3 && nx % 7 == 2);
```

Оператор остатка при дељењу изворно свођење на SAT не обухвата, док се у логици QF_LIA изражава непосредно операцијом `mod`:

```
(set-logic QF_LIA)
(declare-fun nx () Int)
(assert (and
  (> nx 0)
  (< nx 105)
  (= (mod nx 3) 2)
  (= (mod nx 5) 3)
  (= (mod nx 7) 2)))
(check-sat)
(get-model)
```

У траженом опсегу решавач проналази јединствено решење $nx = 23$.

Глава 4

Експериментална евалуација

Експериментална евалуација има за циљ да утврди исправност и практичну употребљивост изведеног проширења. Испитивање се одвија у три правца. Најпре се проверава да ли свођење на SMT даје резултате сагласне са изворним свођењем на SAT. Потом се разматрају перформансе, поређењем времена потребног за добијање решења при свођењу на SAT и при свођењу на SMT. Најзад, посебна пажња посвећена је примерима који користе новоуведене теорије, а који претходно нису били подржани у оквиру алата.

Поређење са алатом URBIVA није укључено у евалуацију. Алат URBIVA није јавно доступан, а његов улазни језик разликује се од језика алата URSA, због чега непосредно и методолошки исправно поређење над истим скупом примера није лако изводљиво.

4.1 Методологија

Провера исправности заснива се на поређењу скупова решења. За сваки улазни програм упоређују се решења добијена изворним свођењем на SAT и свођењем на SMT, при чему се као критеријум исправности узима једнакост скупова добијених решења, независно од редоследа. Поред броја решења, бележи се и време решавања.

Мерења су спроведена решавачима Z3, верзије 4.16, и `svcs5`, верзије 1.3.4, оба укључена као библиотеке и позвана у оквиру истог процеса. Подразумевана ширина бит-вектора износила је осам битова, осим када је другачије

наведено. Постављено је временско ограничење од шездесет секунди по мерењу. Наведена времена одражавају протекло стварно време и служе као оквирни показатељ, а свако мерење укључује и покретање самог алата URSA.

4.2 Инфраструктура за тестирање

Пошто свођење на SMT представља нов начин извођења резултата у односу на постојеће свођење на SAT, било је потребно проверити да ли се решења добијена преко SMT решавача поклапају са решењима која даје SAT решавач. Свођење на SAT при томе служи као референца, будући да сматрамо да је његова исправност, искуствено, кроз велики број примера и примена, већ успостављена. У ту сврху развијена је наменска скрипта за поређење, која за исти улазни програм извршава оба начина решавања и упоређује њихове резултате.

Скрипта покреће алат URSA у два режима. Први је изворни SAT режим. Други је SMT режим, у коме алат добијену формулу решава једним од решавача Z3 или cvc5. Пошто оба режима исписују решења у истом облику, поређење се своди на упоређивање два скупа решења. Очекујемо да оба режима дају исти скуп решења.

Поређење се врши на нивоу скупа решења, независно од редоследа којим су решења пронађена. Ако су скупови решења једнаки, поређење се проглашава успешним, а у супротном се исписује разлика између два скупа, што омогућава откривање семантичких неслагања између ова два приступа.

Набрајање свих решења при свођењу на SMT изводи сам алат, будући да стандард SMT-LIB не предвиђа операцију за проналажење свих модела. Након што решавач пронађе модел, додаје се ограничење које искључује управо тај модел (тзв. блокирајућа клауза), а решавач се поново позива, све док не пријави незадовољивост. За модел у коме је nx једнако 3 и ny једнако 22, блокирајућа клауза има облик:

```
assert (not (and (= nx #x03) (= ny #x16)))
```

Наредба `assert_all`, која враћа сва решења, при свођењу на SMT ради управо додавањем блокирајућих клауза докле год постоје нови модели.

Алат изграђену формулу по правилу прослеђује решавачу и из добијеног модела ишчитава решења. У случају да се цео услов израчуна још у фази превођења и сведе на константну истинитосну вредност, решавач се не позива, него се резултат непосредно проглашава испуњеним или неиспуњеним. Када програм садржи захтев за оптимизацијом, решавачу се уз формулу задаје и циљна функција коју треба минимизовати или максимизовати, па он налази њен оптимум и вредности променљивих за које се он достиже.

Током развоја, скрипта је коришћена као провера сваке измене при свођењу на SMT, поређењем са референтним свођењем на SAT над скупом постојећих URSA примера. У случају симболичког индексирања низова поређење са свођењем на SAT није применљиво, будући да оно ту могућност не подржава. За те примере исправност се процењује поређењем добијеног резултата са унапред познатим, аналитички изведеним исходом сваког примера, попут очекиваног броја решења или очекиване незадовољивости. Поред тога, на примерима чија величина то допушта, непосредним увидом у генерисану SMT-LIB формулу проверено је да она одговара семантици задатог проблема. Сагласност решавача Z3 и `svcs5` при томе служи као додатна провера да резултат није последица понашања конкретног решавача, али сама по себи не потврђује исправност кодирања.

4.3 Резултати

Резултати су организовани по врсти проблема: најпре проблеми одлучивања, затим оптимизациони проблеми, потом поређење по ширини бит-вектора, и на крају задаци које изворно свођење на SAT не може да изрази. За сваки пример из прве три групе упоређују се изворно свођење на SAT и свођење на SMT уз оба решавача.

4.3.1 Проблеми одлучивања

Табела 4.1 приказује резултате над примерима одлучивања, укључујући решавање система линеарних једначина, полиномску једначину, два верификациона задатка (код којих одсуство решења значи да проверавано својство важи) и неколико комбинаторних проблема из дистрибуције алата URSA, попут проблема N дама, магичног квадрата и загонетке SEND+MORE=MONEY. За сваки пример наведен је број решења и измерено време.

Пример	Број решења	SAT	Z3 BV	Z3 LIA	cvc5 BV	cvc5 LIA
Систем двеју једначина	1	0,01	0,01	0,01	0,02	0,01
Полиномска једначина	1	0,01	0,01	-	0,02	-
XOR замена вредности (верификација)	0	0,01	0,01	-	0,01	-
Тачно један постављен бит (верификација)	0	0,01	0,01	-	0,01	-
Осам дама (<i>queens1</i>)	92	0,02	0,08	0,22	0,15	0,09
Осам дама (<i>queens2</i>)	92	0,03	0,06	-	0,11	-
Осам дама (<i>queens3</i>)	92	0,03	0,06	0,07	0,06	0,03
Осам дама (<i>queens4</i>)	92	0,02	0,05	0,07	0,06	0,03
SEND + MORE = MONEY (17 битова)	1	0,04	0,07	0,02	0,04	0,01
Магични квадрат	880	1,97	7,62	30,93	14,95	5,68

Табела 4.1: Резултати над примерима одлучивања (времена у секундама, обухватају покретање алата). Свођење на SMT приказано је за теорије QF_BV и QF_LIA (у заглављу скраћено BV и LIA), за оба решавача. Скупови решења поклапају се са изворним свођењем на SAT, као и између двеју теорија тамо где су обе применљиве. Цртица у пољу значи да се пример не може формулисати у QF_LIA.

На свим примерима из табеле 4.1 свођење на SMT даје иста решења као свођење на SAT, и то за оба коришћена решавача. Табела приказује и поређење теорија QF_BV и QF_LIA. Тамо где се пример може формулисати у обе, скупови решења поклапају се и између QF_BV и QF_LIA, што додатно потврђује исправност. Времена показују да ниједна теорија није доследно бржа: на неким примерима брже је свођење на QF_BV (на пример, решавач Z3 на проблему N дама), а на другима на QF_LIA (систем једначина и SEND + MORE = MONEY, а решавач cvc5 и на магичном квадрату). У QF_LIA

су, за примере који то захтевају, дође границе домена наведене експлицитно, будући да их у QF_BV намеће ширина бит-вектора.

Примери који се не могу формулисати у QF_LIA су они са нелинеарном аритметиком (полиномска једначина) или битовским операцијама (XOR замена вредности, провера тачно једног постављеног бита и једна од формулација проблема N дама), што линеарна аритметика целих бројева не подржава.

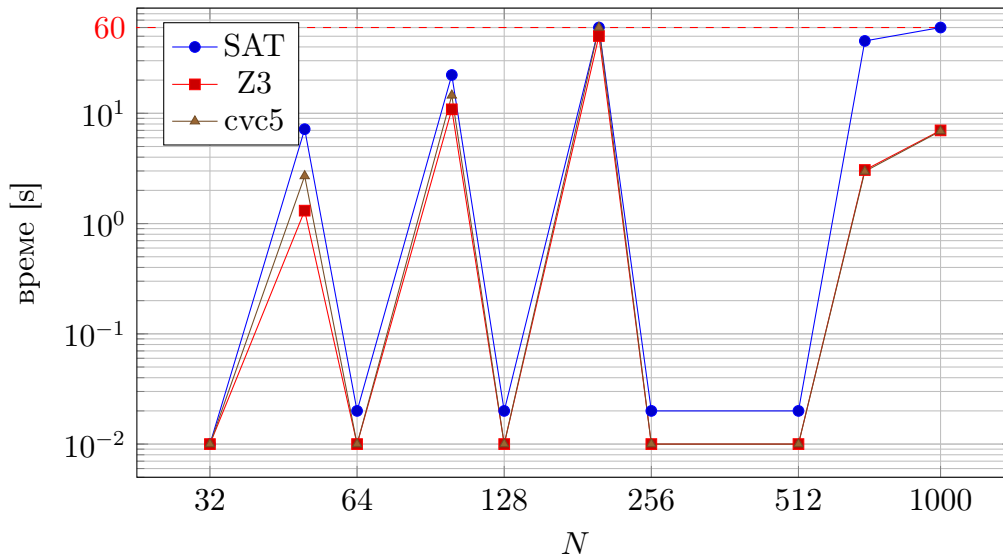
Време проналажења првог решења проблема N дама не расте глатко са N . Пресудан је однос N и ширине бит-вектора којим се вредности представљају: када је N степен двојке, инстанца је једноставна и решава се за неколико стотинки секунде чак и за велике табле (на пример, $N = 512$), док у супротном претрага постаје знатно тежа и неправилна, са прекорачењима времена већ на умерено великим таблама. На тако тешким инстанцама свођење на SMT уз решаваче Z3 и cvc5 по правилу боље скалира од изворног свођења на SAT, што се најјасније види за $N = 700$ и $N = 1000$, где свођење на SMT проналази решење за неколико секунди, а свођење на SAT прекорачује ограничење. Табела 4.2 и слика 4.1 приказују измерена времена. Пошто је реч о својству ширине бит-вектора, ова табела приказује свођење на SMT у теорији QF_BV.

N	SAT [s]	Z3 [s]	cvc5 [s]
32	0,01	0,01	0,01
50	7,18	1,31	2,70
64	0,02	0,01	0,01
100	22,28	10,87	14,54
128	0,02	0,01	0,01
200	>60	50,25	>60
256	0,02	0,01	0,01
512	0,02	0,01	0,01
700	45,33	3,06	2,97
1000	>60	7,00	6,90

Табела 4.2: Време проналажења првог решења проблема N дама при расту димензије табле. Осенчени су редови у којима је N степен двојке. Ознака >60 значи прекорачење ограничења од шездесет секунди. Свођење на SMT је у теорији QF_BV.

Разлог оваквог понашања лежи у начину на који се позиција даме пред-

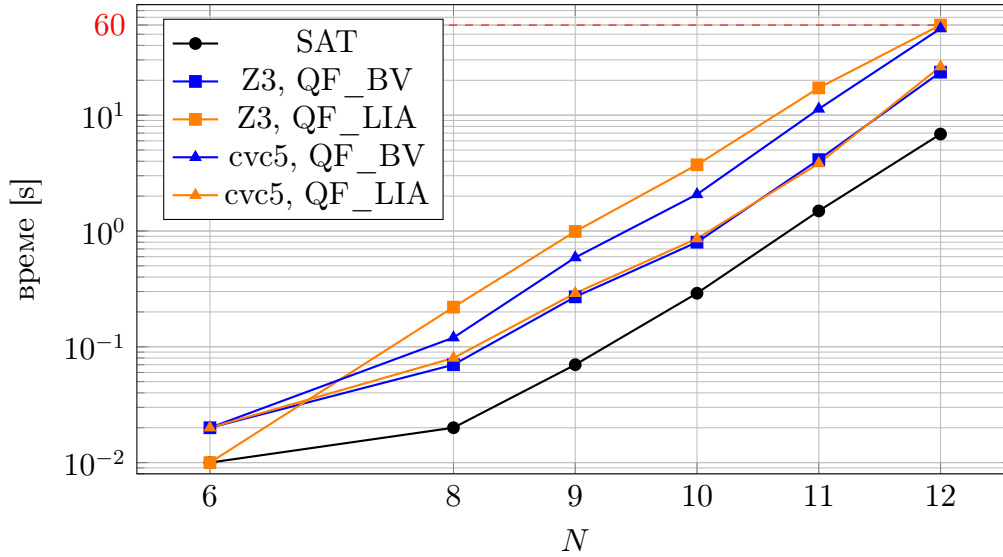
ставља. Свакој дами придружује се њена колона, представљена бит-вектором чија се ширина бира као најмањи број битова довољан да се представе вредности до N . Када је N степен двојке, тај бит-вектор представља тачно вредности из опсега $[0, N)$, па је ограничење да колона буде мања од N увек испуњено и практично нестаје. Када N није степен двојке, представљиви опсег је шири од $[0, N)$, па ограничење мора да искључује вишак вредности које не одговарају ниједној колони, што претрагу знатно отежава.



Слика 4.1: Време проналажења првог решења проблема N дама у зависности од N (логаритамске осе). Тачке на вредности 60 означавају прекорачење временског ограничења. Тачке за које је N степен двојке налазе се при дну, док остале нагло скачу.

Занимљиво је размотрити и резултате за набрајање свих решења истог проблема. Слика 4.2 приказује време потребно да се пронађу сва решења за мале табле. За разлику од проналажења првог решења, овде нема ефекта степена двојке, будући да број решења расте експоненцијално са N (за N од 6 до 12 редом 4, 40, 92, 352, 724, 2680 и 14200), па тај раст доминира временском сложености. Притом је изворно свођење на SAT доследно најбрже, у складу са тим што SAT решавач набраја наменском уграђеном процедуром, док свођење на SMT сваки модел искључује блокирајућом клаузом. Набрајање се може спровести и у теорији QF_LIA, при чему се скупови решења поклапају. У овом примеру је притом потребно експлицитно навести да су вредности

ненегативне, будући да то у QF_BV произлази из неозначене бит-векторске репрезентације. Занимљиво је да решавач cvc5 набраја брже у QF_LIA него у QF_BV, док је код решавача Z3 обрнуто, што показује да избор најповољније теорије зависи и од решавача.



Слика 4.2: Време набрајања свих решења проблема N дама у зависности од N (логаритамска временска оса), уз свођење на SAT и на SMT у теоријама QF_BV и QF_LIA. Скупови решења, и њихов број, поклапају се у свим приступима. Тачке на вредности 60 означавају прекорачење ограничења од шездесет секунди.

Разматра се и класичан проблем распоређивања послова у радионици (енгл. *job-shop scheduling*). У инстанци са три посла и три машине свака операција захтева одређену машину и има задато трајање, операције унутар посла извршавају се редом, а свака машина у сваком тренутку опслужује највише једну операцију. Последње ограничење је дисјунктивно, јер за сваки пар операција на истој машини једна мора да се заврши пре него што друга почне. Тражи се распоред чија дужина (енгл. *makespan*) не прелази K . Свођење на SMT у логици QF_LIA проналази распоред дужине 9, док за $K = 8$ враћа незадовољивост, чиме потврђује да је 9 најмања могућа дужина за ову инстанцу.

Да бисмо испитали како ово свођење скалира, генерисане су инстанце растуће величине, са N послова и N машина. Табела 4.3 приказује време

потребно да се пронађе оптимална дужина распореда и потврди њена оптималност, за N од 4 до 16. Решавач Z3 скалира знатно повољније од решавача cvc5: на највећој инстанци, димензије 16×16 , задатак решава за око петину секунде, док решавачу cvc5 треба неколико секунди. Изворно свођење на SAT овде није приказано, будући да ограничена бит-векторска аритметика користи свођење по модулу, па за неке дужине распореда враћа „лажно” задовољив распоред.

Инстанца	Оптимум	Z3	cvc5
4×4	28	0,008	0,026
6×6	41	0,013	0,017
8×8	52	0,029	0,053
10×10	65	0,031	0,163
12×12	78	0,058	0,832
14×14	89	0,108	1,213
16×16	102	0,198	4,745

Табела 4.3: Скалирање проблема распоређивања послова на генерисаним инстанцама $N \times N$ (N послова и N машина). Приказано је време (у секундама) да се пронађе оптимална дужина распореда и потврди њена оптималност, за решаваче Z3 и cvc5.

4.3.2 Оптимизациони проблеми

Задаци оптимизације приказани су у табели 4.4. При изворном свођењу на SAT, оптимизација се своди на низ провера задовољивости, вођен претрагом по вредности циљне променљиве. Циљна променљива се фиксира на једну вредност из интервала претраге, па се проверава да ли проблем тада има решење. Према исходу те провере интервал се сужава, све док се не пронађе оптимум. При свођењу на SMT уз решавач Z3 користи се уграђена подршка тог решавача за оптимизацију, док решавач cvc5 такву уграђену подршку нема. Иако оптимизација уз cvc5 у овом раду није изведена, она би се могла подржати описаним поступком претраге по вредности циљне променљиве, будући да он не зависи од појединачног решавача.

На сва три примера изворно свођење на SAT и свођење на SMT уз решавач Z3 проналазе исти, тачан оптимум. Времена показују да ниједан приступ није доследно најбржи. На Голомбовом леђиру реда осам свођење на SAT пред-

њачи, а свођење на QF_LIA прекорачује временско ограничење, будући да је проблем чисто комбинаторан. Мањи редови, попут реда четири, решавају се брзо и у QF_LIA. На задатку са олимпијаде, међутим, свођење на QF_LIA даје оптимум нешто брже од QF_BV. Подударане оптимума на том задатку постигнуто је тек по отклањању једне грешке у поступку оптимизације алата URSA, детаљније размотреној у одељку 4.4.

Пример	Оптимум	SAT	Z3 QF_BV	Z3 QF_LIA
Голомбов лењир	34	0,98	2,96	>60
Рамзијев проблем	3	0,02	0,02	0,05
ИМО 1992, задатак 3	32	0,10	0,20	0,08

Табела 4.4: Табела приказује пронађени оптимум и време решавања (у секундама) за задатке оптимизације, при свођењу на SAT и на SMT решавачем Z3 у теоријама QF_BV и QF_LIA. Оптимум се поклапа код сва три приступа, а решавач свс5 оптимизацију не подржава. Ознака >60 значи прекорачење ограничења од шездесет секунди. За Рамзијев проблем и задатак са олимпијаде вредности су у QF_LIA ограничене на ненегативне.

Као илустрацију начина на који се оптимизација задаје решавачу, размотримо Голомбов лењир реда четири. Наредба `minimize(nL, 1, 100)` при свођењу на SMT преводи се у следећу формулу у којој су ограничења лењира сажета ради прегледности:

```
(set-logic QF_LIA)
(declare-fun nA_1_ () Int)
(declare-fun nA_2_ () Int)
(declare-fun nL () Int)

(assert ...) ; uslovi Golombovog lenjira
(assert (>= nL 1))
(assert (<= nL 100))
(minimize nL)

(check-sat)
(get-objectives)
(get-model)
```

Границе интервала претраге, $1 \leq nL \leq 100$, записане су као обичне тврдње, док је сам циљ изражен наменском командом `(minimize nL)` (за

максимизацију би то била (`maximize nL`)). Такву формулу, заједно са командом `minimize`, алат учитава у свој оптимизациони објекат `z3::optimize`, а не у обичан решавач, па он једним позивом проналази и доказује оптимум. Команда (`get-objectives`) потом враћа оптималну вредност циљне променљиве, а (`get-model`) придружено решење. Решавач `svs5` не подржава команде `minimize` и `maximize`, па се задаци оптимизације решавају само уз `Z3`.

Голомбов лењир завређује детаљнији осврт, будући да илуструје скалирање оптимизације са величином проблема. Голомбов лењир је скуп ознака на целобројним позицијама код кога су сва растојања између парова ознака међусобно различита, а ред лењира је број ознака. За дати ред тражи се лењир најмање могуће дужине, где је дужина позиција последње ознаке. На пример, ознаке $\{0, 1, 4, 6\}$ чине оптималан лењир реда четири, дужине шест: од четири ознаке добија се шест парова, а сва растојања међу њима (1, 2, 3, 4, 5 и 6) међусобно су различита.

Табела 4.5 приказује пронађени оптимум и време за редове од четири до девет. За сваки ред и изворно свођење на SAT и свођење на SMT уз решавач `Z3` проналазе исти, тачан оптимум. Време је занемарљиво за мале редове, а од реда седам нагло расте за оба приступа, отприлике десет пута по реду. Изворно свођење на SAT при томе остаје ефикасније на већим редовима. Разлог је вероватно у природи проблема: Голомбов лењир је чисто комбинаторан, без структуре коју би решавач могао да искористи на нивоу термова, па предност преузима брза комбинаторна претрага изворног свођења, док решавач `Z3` добија предност тек када проблем има погодну структуру за такво резонување. Приказано свођење на SMT је у теорији QF_BV. У QF_LIA раст је још стрмији и већ на реду осам прекорачује временско ограничење, у складу са запажањем код табеле 4.4.

4.3.3 Зависност од ширине бит-вектора

Наредни примери проверавају два алгебарска идентитета над бит-векторима по модулу 2^l (где l означава ширину бит-вектора), дистрибутивност множења:

```
assert(na * (nb + nc) != na * nb + na * nc);
```

ред m	оптимум	SAT [s]	Z3 [s]
4	6	0,13	0,05
5	11	0,04	0,03
6	17	0,07	0,04
7	25	0,14	0,33
8	34	0,89	3,37
9	44	8,36	27,68

Табела 4.5: Голомбов лењир за редове од 4 до 9: пронађени оптимум (дужина лењира) и време при свођењу на SAT и на SMT уз решавач Z3. Решавач `svcs5` не подржава оптимизацију. Свођење на SMT је у теорији QF_BV.

и квадрат бинома:

```
assert((nx + ny)*(nx + ny) != nx*nx + 2*nx*ny + ny*ny);
```

Оба идентитета посматрају се искључиво у теорији QF_BV, из два разлога: питање се управо тиче понашања бит-векторске аритметике по модулу 2^l у зависности од ширине бит вектора, а сами изрази садрже производе променљивих, па као нелинеарни не припадају линеарној аритметици целих бројева.

У оба случаја формула је незадовољива. Изворно свођење на SAT мора да бит-бластује више множака ширине l , па величина исказне формуле нагло расте са l . Савремени SMT решавачи, међутим, примењују поједностављивања на нивоу термова пре бит-бластовања [19]. Захваљујући томе решавач Z3 препознаје оба идентитета пре бит-бластовања, па му време остаје практично независно од l . Решавач `svcs5` понаша се различито од идентитета до идентитета: на дистрибутивности остаје практично константан, док на квадрату бинома, попут свођења на SAT, нагло расте. Табеле 4.6 и 4.7 приказују измерена времена, при чему ознака >60 значи да је прекорачено ограничење од шездесет секунди.

4.3.4 Верификација програма са симболичким индексима

Наредни примери илуструју верификацију својстава кратких програма који користе симболичко индексирање низова, што је образац карактеристи-

l	SAT [s]	Z3 [s]	cvc5 [s]
4	0,02	0,01	0,02
6	0,28	0,01	0,02
8	25,48	0,01	0,02
10	>60	0,01	0,02
12	>60	0,01	0,02
16	>60	0,01	0,02

Табела 4.6: Провера дистрибутивности $a \cdot (b + c) = a \cdot b + a \cdot c$ при различитим ширинама l . Свођење на SAT нагло расте, док свођење на SMT остаје практично константно за оба решавача.

l	SAT [s]	Z3 [s]	cvc5 [s]
4	0,02	0,01	0,02
6	0,03	0,01	0,04
8	0,40	0,01	0,27
10	10,85	0,01	5,61
12	>60	0,01	>60
16	>60	0,01	>60

Табела 4.7: Провера квадрата бинома $(x + y)^2 = x^2 + 2xy + y^2$ при различитим ширинама l . Решавач Z3 остаје практично константан, док свођење на SAT и решавач cvc5 нагло расту.

чан за ограничену проверу модела програма [1]. У сваком од њих ограничење тражи контрапример, па незадовољивост значи да проверавано својство важи. Изворно свођење на SAT ниједан од ових задатака не подржава, будући да садрже читање или упис на симболичком индексу.

Први пример проверава исправност замене вредности на две симболичке позиције:

```
nOldNj = nA[nj];

nTmp   = nA[ni];
nA[ni] = nA[nj];
nA[nj] = nTmp;

assert(nA[ni] != nOldNj);
```

Након замене, вредност на позицији `ni` треба да буде једнака ранијој вред-

ности на позицији n_j , без обзира на то да ли се индекси поклапају. Свођење на SMT враћа незадовољивост и у теорији QF_ABV и у теорији QF_ALIA, за мање од стотинке секунде уз оба решавача, чиме потврђује да својство важи за све низове и све парове индекса. Свођење на SAT враћа само одговор да решење не постоји, будући да проблем не може ни да формулише.

Други пример је ограничена провера једног корака алгоритма *selection sort* над низом од пет елемената, при чему се проверава да након тог корака позиција нула садржи минимум разматраног опсега:

```
nmin_idx = 0;
for(nj=1; nj<5; nj++)
    nmin_idx = ite(nA[nj] < nA[nmin_idx], nj, nmin_idx);

nTmp = nA[0];
nA[0] = nA[nmin_idx];
nA[nmin_idx] = nTmp;

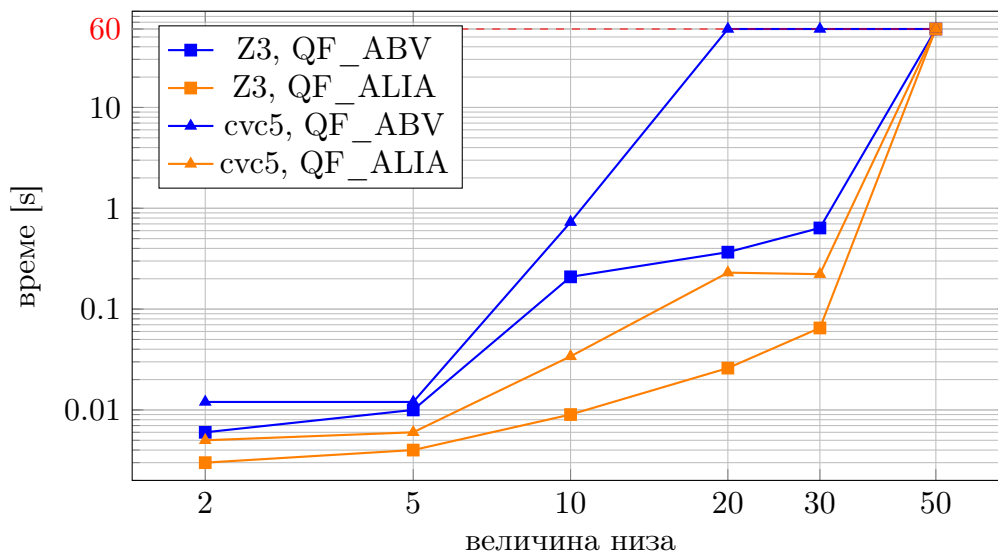
assert(nA[0] > nA[1] || nA[0] > nA[2] || nA[0] > nA[3] || nA[0] > nA[4]);
```

Негација својства је незадовољива, што свођење на SMT потврђује и у теорији QF_ABV и у теорији QF_ALIA, за око 0,01 секунду уз оба решавача. Изворно свођење на SAT овакав задатак не може да реши без подршке за низове са симболичким индексима, па би за решавање тим путем програм морао да буде знатно преформулисан.

Наредни пример показује како трошак резоновања о низовима са симболичким индексом расте са величином низа. Над низом од k симболичких елемената рачуна се максимум, а затим се проверава да ниједан елемент није већи од тог максимума, при чему се низ чита на симболичком индексу. Својство важи и у теорији аритметике бит-вектора са низовима (QF_ABV) и у теорији линеарне аритметике целих бројева са низовима (QF_ALIA), са истим исходом незадовољивости. У теорији QF_ALIA симболичком индексу се притом задаје експлицитна доња граница, како би домен био ограничен. Слика 4.3 приказује измерена времена за величине од 2 до 50 елемената.

Слика показује да је однос обрнут у поређењу са комбинаторним приме-

рима: теорија QF_ALIA скалира знатно боље од теорије QF_ABV. Провера максимума састоји се само од поређења целобројних вредности, што решавач над теоријом линеарне аритметике целих бројева обрађује непосредно, методом гранања над линеарним ограничењима. Са друге стране, решавач над теоријом аритметике бит-вектора свако поређење мора да бит-бластује у исказну формулу чија величина расте са ширином бит-вектора. Тако решавач cvc5 у теорији QF_ABV прекорачује ограничење већ на двадесет елемената, док у теорији QF_ALIA даје резултат за делић секунде. На педесет елемената обе теорије прекорачују ограничење. Изворно свођење на SAT овакав задатак не подржава, будући да се користи симболички индекс.



Слика 4.3: Провера да израчунати максимум симболичког низа није премашен ни на једном индексу, при расту величине низа, у теоријама QF_ABV и QF_ALIA (логаритамске осе). Приказано је време за решаваоче Z3 и cvc5. Тачке на вредности 60 означавају прекорачење ограничења од шездесет секунди. Изворно свођење на SAT не подржава симболичко индексирање, па није приказано.

Наредни пример илуструје набрајање свих решења када је на улазу низ. Над низом од пет симболичких елемената примењује се алгоритам *selection sort*, а затим се захтева да сортирани низ буде управо $[0, 1, 2, 3, 4]$:

```
for(ni=0; ni<5; ni++) {
```

```
nmin_idx = ni;
for(nj=ni+1; nj<5; nj++)
    nmin_idx = ite(nA[nj] < nA[nmin_idx], nj, nmin_idx);
nTmp = nA[ni];
nA[ni] = nA[nmin_idx];
nA[nmin_idx] = nTmp;
}
assert_all(nA[0]==0 && nA[1]==1 && nA[2]==2 && nA[3]==3 && nA[4]==4);
```

Наредба `assert_all` набраја све улазне низове који тај услов задовољавају. Пошто сортирање даје $[0, 1, 2, 3, 4]$ ако и само ако је улаз пермутација скупа $\{0, 1, 2, 3, 4\}$, очекивани број решења је $5! = 120$. Свођење на SMT проналази тачно 120 решења, чиме потврђује исправност и самог сортирања и набрајања свих решења. Исти број решења добија се и у теорији аритметике бит-вектора са низовима (QF_ABV) и у теорији линеарне аритметике целих бројева са низовима (QF_ALIA). Набрајање свих 120 решења траје око 0,2 секунде уз решавач Z3 и око 0,3 секунде уз `svcs5` у теорији QF_ABV, односно око 0,08 и 0,14 секунди у теорији QF_ALIA, мерено од покретања алата.

4.4 Анализа резултата

Поклапање броја решења на свим примерима одлучивања потврђује да свођење на SMT даје исте резултате као изворно свођење на SAT, чиме је потврђена његова исправност. Ово поклапање важи и за примере са већим бројем решења, попут магичног квадрата са осамсто осамдесет решења. Поред тога, на задацима задовољивости исти скупови решења добијени су са оба решавача, Z3 и `svcs5`, што је очекивано с обзиром на то да се формула размењује у стандардном формату SMT-LIB, и потврђује да приступ не зависи од избора решавача.

Задаци оптимизације заслужују посебан осврт. На Голомбовом лењиру, Рамзијевом проблему и задатку са Међународне математичке олимпијаде из 1992. године изворно свођење на SAT и свођење на SMT уз решавач Z3 дају исти, тачан оптимум, редом 34, 3 и 32. Ово подударане постигнуто је, међутим, тек по отклањању једне грешке у алату URSA. Ова грешка остала је дуго непримећена и налазила се у делу модула за решавање оптимизационих проблема који је ретко коришћен у постојећим примерима и досадашњим

применама система URSA. При почетном поређењу изворна оптимизација враћала је на задатку са олимпијаде вредност 18, док је решавач Z3 налазио 32. Испитивање је показало да фаза претраге у поступку оптимизације не грана по смеру оптимизације, него у оба случаја примењује логику за минимизацију, па се при максимизацији претрага спушта уместо да се пење и не враћа оптималну вредност. По отклањању те грешке изворно свођење на SAT такође налази вредност 32. Тиме је поређење са свођењем на SMT послужило и као провера која је открила дотад непримећену грешку у поступку оптимизације.

У погледу перформанси, за проналажење појединачног решења и за проблеме са малим бројем решења сва три случаја су упоредива и мере се у деловима секунде. При набрајању великог броја решења изворно свођење на SAT често је најефикасније. Оба приступа искључују већ пронађене моделе да их не би понављали, али различитим механизмом. SAT решавач набраја наменском уграђеном процедуром, у оквиру једне непрекидне претраге, задржавајући научене клаузе и прелазећи са модела на модел враћањем уназад, користећи специфичности уграђеног SAT решавача (clasp). Свођење на SMT нема такву наменску уграђену операцију, будући да је стандард SMT-LIB не предвиђа, па алат сваки нови модел искључује блокирајућом клаузом. То се изводи инкрементално, додавањем клаузе истој инстанци решавача преко његовог програмског интерфејса, тако да се претрага не покреће испочетка. Ипак, број блокирајућих клауза расте са бројем пронађених решења, па свака наредна провера постаје скупља, за разлику од наменске процедуре SAT решавача, која је за то посебно прилагођена.

При свођењу на SMT бира се циљна теорија, теорија аритметике бит-вектора (QF_BV) или теорија линеарне аритметике целих бројева (QF_LIA). Тај избор одређује и који се проблеми могу свести на дату теорију и колико је то свођење ефикасно. Аритметика бит-вектора је подразумевана и обухвата све конструкције које алат уме да преведе, будући да прати семантику изворног свођења на SAT. Линеарна аритметика целих бројева носи одређена ограничења: у њој се не могу записати битовске операције ни производ две променљиве. Обрнуто, линеарна аритметика решава и задатке које аритметика бит-вектора не може: пошто су у теорији QF_BV вредности неозначене

и ограничене ширином, задаци са негативним вредностима у њој остају без решења, а они са вредностима изван те ширине дају погрешан резултат услед свођења по модулу, док их теорија QF_LIA решава тачно над целим бројевима. Тамо где су обе теорије применљиве, ниједна није универзално боља. На претежно комбинаторним проблемима и при исцрпном набрајању решења теорија аритметике бит-вектора је по правилу бржа, будући да се формуле у њој решавају бит-бластовањем, што одговара комбинаторној природи таквих проблема, док се линеарна аритметика ослања на симплекс-метод и метод гранања, који су за такве проблеме мање погодни. На проблемима у којима преовлађује линеарно резонување предност прелази на линеарну аритметику целих бројева. Она је бржа на систему линеарних једначина и на загонетки SEND+MORE=MONEY (табела 4.1), а на провери израчунатог максимума низа њена предност знатно расте са величином низа (слика 4.3). Најзад, најповољнија теорија може зависити и од решавача: при набрајању свих решења проблема N дама решавач `svcs5` је бржи у теорији QF_LIA, док је код решавача `Z3` обрнуто.

Резултати показују да свођење на SMT није замена за свођење на SAT, већ његова допуна. На чисто комбинаторним проблемима са великим бројем решења изворно свођење на SAT често је ефикасније. Свођење на SMT добија на три фронта. Прво, код алгебарских идентитета чија величина расте са ширином бит-вектора, решавач `Z3` препознаје идентитет на нивоу термова, док изворно свођење на SAT нагло расте, што потврђују табеле 4.6 и 4.7. Друго, код програма са симболичким индексирањем низова изворно свођење на SAT не може ни да формулише проблем, док га свођење на SMT решава непосредно применом аксиома теорије низова. Треће, код линеарне целобројне аритметике над неограниченим доменом свођење на SMT даје праву семантику, без свођења по модулу које намеће бит-векторска репрезентација, а подржава и целобројно дељење и остатак, које изворно свођење на SAT уопште не обухвата.

Најзначајнији резултат евалуације огледа се у примерима које изворно свођење на SAT не решава, а које свођење на SMT решава, приказаним у уводу и у претходним одељцима. Ови примери показују да проширење не мења само начин на који се постојећи проблеми решавају, већ проширује и

класу проблема које алат URSA може да изрази и реши. Ту се пре свега издвајају задаци са симболичким приступом низовима, карактеристични за верификацију програма и претрагу над структурама података, задаци над целим бројевима без ограничења ширине и са означеним вредностима, као и задаци који користе операције дељења и остатка.

Глава 5

Закључак и правци даљег рада

У овом раду представљено је проширење алата URSA подршком за свођење на SMT. Поред постојећег свођења на SAT, исти улазни програм може се сада превести у формулу теорије аритметике бит-вектора или линеарне аритметике целих бројева, а у присуству симболичких индекса низова и у одговарајућу теорију са низовима. Формуле се записују у стандардном формату SMT-LIB и решавају решавачима Z3 и `svcs5`, уз подршку за набрајање свих решења и за оптимизацију. Досадашњи начин рада остао је непромењен и подразумеван.

Експериментална евалуација потврдила је исправност новоуведеног приступа, будући да свођење на SMT даје исте скупове решења као свођење на SAT на свим примерима одлучивања, и то за оба решавача. Показано је и да се два приступа допуњују, а не замењују. Изворно свођење на SAT остаје најефикасније при исцрпном набрајању решења, где уграђени механизам набрајања SAT решавача надмашује понављано решавање уз блокирајуће клаузе које свођење на SMT захтева, а показало се бржим и на појединим комбинаторним проблемима оптимизације, попут Голомбовог лењира. Свођење на SMT добија тамо где свођење на SAT постаје неповољно, на пример, код алгебарских идентитета чија величина расте са ширином бит-вектора, где решавач препознаје идентитет на нивоу термова. Најзначајнији допринос овог рада је проширење класе проблема које алат може да изрази, будући да свођење на SMT омогућава верификацију програма са симболичким приступом низовима, резонување над неограниченим и означеним целим бројевима и коришћење дељења и остатка, што раније није било могуће. Поред главног

доприноса, упоређивање новог и постојећег свођења показало се корисним и за сам алат URSA, будући да је указало на једну, до сада неоткривену, грешку у поступку оптимизације, која је потом исправљена.

Постоји више праваца за даљи рад. Најнепосреднији је проширење на додатне теорије које SMT решавачи подржавају, попут теорије реалних бројева, неинтерпретираних функција или ниски, чиме би се класа обухваћених проблема даље увећала. Могуће је и унапређење начина на који се формула предаје решавачу. И сада се решавач користи преко његовог програмског интерфејса, али тако што му се формула прослеђује у облику ниске у формату SMT-LIB, док би се уместо тога она могла градити непосредно, конструкцијом термова кроз тај интерфејс, чиме би се могло остварити извесно убрзање. Оптимизација би се, даље, могла подржати и уз решавач `svcs5`, који је изворно не подржава, применом истог поступка претраге који алат већ користи при свођењу на SAT. Најзад, комплементарност свођења на SAT и свођења на SMT природно упућује на портфолио приступ. Он се може остварити тако што се задати проблем истовремено своди и решава на оба начина, уз прихватање исхода оног решавача који први да резултат, или тако што се најпогоднији приступ бира унапред, на основу особина задатог проблема.

Додатак А

Језик алата URSA

Овај додатак сажето описује језик алата URSA, у обиму довољном за читање примера наведених у раду. Потпун опис језика дат је у изворном раду о алату [20]. Опис одговара понашању које алат остварује и при свођењу на SAT и при свођењу на SMT, будући да оба начина рада деле исти улазни језик и његову обраду.

А.1 Лексичке конвенције

Програм чини низ наредби. Идентификатори су типизирани самим именом: свако име нумеричке променљиве почиње словом **n**, а свако име Булове променљиве словом **b**. Нумеричке константе записују се декадно, а Булове константе чине **true** и **false**. Коментари се пишу између ознака **/*** и ***/**. Свака наредба завршава се тачком и зарезом.

А.2 Изрази и оператори

Нумерички изрази граде се од нумеричких променљивих, константи и оператора, а Булови изрази од Булових променљивих, литерала, релација поређења и логичких везника. Подржани оператори приказани су у табели А.1.

Поред наведених оператора, језик обухвата и неколико додатних језичких конструкција. Израз **ite(b, n1, n2)** враћа **n1** ако је услов **b** тачан, а иначе **n2**. Израз **sgn(n)** даје знак вредности. Конверзије између типова обављају

Група	Оператори
аритметички	+ - * / %
инкремент и декремент	++ --
битовски	& ^ ~
померања	<< >>
логички везници	&& ^^ !
поређење	== != < <= > >=
додела и сложена додела	= += -= *= /= %= &&= =

Табела А.1: Оператори језика алата URSA, груписани по врсти.

`num2bool`, који нумеричку вредност тумачи као тачну ако је различита од нуле, и `bool2num`, који Булову вредност пресликава у нулу или јединицу.

А.3 Наредбе и контрола тока

Основна наредба је додела вредности променљивој. Наредбе се могу груписати у блок витичастим заградама. Условно извршавање задаје се конструкцијом `if`, уз необавезну `else` грану, а понављање конструкцијама `for` и `while`, синтаксички налик одговарајућим конструкцијама језика C. Услови гранања и границе петљи морају бити конкретни, тј. познати пре генерисања формуле, па алат све контролне структуре статички одмотава.

```
bDomain = true;
for (ni = 0; ni + 1 < nM; ni++)
    bDomain &&= (nA[ni] < nA[ni+1]);
```

А.4 Низови

Низови се уводе имплицитно, доделом вредности елементу и могу бити вишедимензионални. Индекс се наводи угластим заградама, а вишедимензионални приступ узастопним заградама, на пример `b[ni][nj]`. При свођењу на SAT индекс мора бити конкретан, тј. познат пре генерисања формуле, док свођење на SMT, уз теорију низова, допушта и симболичке индексе, како је описано у поглављу 3.5.

А.5 Процедуре

Понављајући део програма може се издвојити у процедуру, уведену кључном речи `procedure`, а позвати наредбом `call`. Аргументи се преносе по имену, тако да измене над параметром унутар процедуре остају видљиве и након повратка.

```
procedure power(npowernk, nk) {  
    np = npowernk;  
    for (ni = 1; ni < nk; ni = ni + 1)  
        npowernk = npowernk * np;  
}  
  
call power(nxpowernk, nk);
```

А.6 Ограничења и оптимизација

Ограничења проблема задају се наредбама `assert` и `assert_all`. Прва тражи неку задовољавајућу валуацију променљивих, а друга скуп свих задовољавајућих валуација. За проблеме оптимизације предвиђене су наредбе `minimize` и `maximize`, које примају циљну променљиву и доњу и горњу границу интервала претраге, на пример `minimize(nL, 1, 100)`.

А.7 Позивање алата

Ширина бит-вектора којим се представљају нумеричке вредности задаје се опцијом `-l`, а подразумевана вредност износи осам битова. Избор SAT решавача при изворном свођењу врши се опцијом `-s`. Свођење на SMT укључује се опцијом `-smt`, циљна теорија бира се опцијом `-smtlogic` (вредности `QF_BV` и `QF_LIA`), решавач опцијом `-smtsolve` (вредности `z3` и `cvc5`), а исписивање генерисане формуле у формату SMT-LIB опцијом `-smtout`.

Библиографија

- [1] Alessandro Armando, Jacopo Mantovani, and Lorenzo Platania. Bounded model checking of software using SMT solvers instead of SAT solvers. In *Model Checking Software (SPIN 2006)*, volume 3925 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2006.
- [2] Haniel Barbosa, Clark Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Meng, Giles Reynolds, Elvira Stone, Cesare Tinelli, Andrew Reynolds, et al. cvc5: A versatile and industrial-strength SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2022)*, volume 13243 of *Lecture Notes in Computer Science*, pages 415–442. Springer, 2022.
- [3] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa, 2017.
- [4] Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185, pages 1267–1329. IOS Press, 2nd edition, 2021.
- [5] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, second edition, 2021.
- [6] Cristian Cadar, Daniel Dunbar, and Dawson R. Engler. KLEE: Unassisted and automatic generation of high-coverage booktitle = OSDI, year = 2008.
- [7] Cristian Cadar, Vijay Ganesh, Peter M. Pawlowski, David L. Dill, and Dawson R. Engler. EXE: Automatically generating inputs of death. In *CCS*, 2006.

- [8] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The MathSAT5 SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2013)*, volume 7795 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2013.
- [9] 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.
- [10] D. C. Cooper. Theorem proving in arithmetic without multiplication. In *Machine Intelligence 7*, pages 91–99. Edinburgh University Press, 1972.
- [11] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.
- [12] Leonardo de Moura and Nikolaj Bjørner. Integrating Simplex with DPLL(T). In *Verification, Model Checking, and Abstract Interpretation*, pages 214–229. Springer, 2008.
- [13] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008)*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.
- [14] DIMACS Challenge Organization. Satisfiability: Suggested format. Technical report, Center for Discrete Mathematics and Theoretical Computer Science (DIMACS), 1993. Available at <ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/doc/format.ps>.
- [15] Bruno Dutertre and Leonardo de Moura. A fast linear arithmetic solver for DPLL(T). In *International Conference on Computer Aided Verification*, pages 81–94. Springer, 2006.
- [16] Patrice Godefroid, Nils Klarlund, and Koushik Sen. DART: Directed automated random testing. In *PLDI*, 2005.
- [17] Patrice Godefroid, Michael Y. Levin, and David Molnar. SAGE: Whitebox fuzzing for security testing. *Communications of the ACM*, 55(3):40–44, 2012.
- [18] Torbjörn Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6.3.0 edition, 2023.

- [19] Liana Hadarean, Kshitij Bansal, Dejan Jovanović, Clark Barrett, and Cesare Tinelli. A tale of two solvers: Eager and lazy approaches to bit-vectors. In *Computer Aided Verification (CAV 2014)*, volume 8559 of *Lecture Notes in Computer Science*, pages 680–695. Springer, 2014.
- [20] Predrag Janičić. Ursa: A system for uniform reduction to SAT. *Logical Methods in Computer Science*, 8(3), 2012.
- [21] James C. King. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, 1976.
- [22] Daniel Kroening and Ofer Strichman. *Decision Procedures: An Algorithmic Point of View*. Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin, Heidelberg, 2nd edition, 2016.
- [23] Leonid A. Levin. Universal search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.
- [24] Filip Marić and Predrag Janičić. URBiVA: Uniform reduction to bit-vector arithmetic. In Jürgen Giesl and Reiner Hähnle, editors, *Automated Reasoning, 5th International Joint Conference (IJCAR 2010)*, volume 6173 of *Lecture Notes in Computer Science*, pages 388–395. Springer, 2010.
- [25] Joao P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.
- [26] John McCarthy. Towards a mathematical science of computation. In Cicely M. Popplewell, editor, *Information Processing 1962: Proceedings of IFIP Congress 62*, pages 21–28. North-Holland, 1963.
- [27] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Annual Design Automation Conference (DAC)*, pages 530–535, 2001.
- [28] Greg Nelson and Derek C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979.

- [29] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a standard CP modelling language. In *Principles and Practice of Constraint Programming (CP 2007)*, pages 529–543, 2007.
- [30] 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*, 53(6):937–977, 2006.
- [31] Mojżesz Presburger. über die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchem die addition als einzige operation hervortritt. In *Comptes Rendus du I Congrès de Mathématiciens des Pays Slaves*, pages 92–101, 1929.
- [32] William Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 35(8):102–114, 1991.
- [33] Koushik Sen, Darko Marinov, and Gul Agha. CUTE: A concolic unit testing engine for C. In *ESEC/FSE*, 2005.
- [34] SMT-COMP Organizers. The international Satisfiability Modulo Theories Competition (SMT-COMP). <https://smt-comp.github.io/>, 2026. Accessed: July 2026.
- [35] Aaron Stump, Clark Barrett, David L. Dill, and Jeremy Levitt. A decision procedure for an extensional theory of arrays. In *International Logic Programming Symposium*, pages 23–37, 2001.
- [36] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear CSP into SAT. *Constraints*, 14(2):254–272, 2009.
- [37] Emina Torlak and Rastislav Bodik. A lightweight symbolic virtual machine for solver-aided host languages. In *PLDI*, 2014.
- [38] Emina Torlak and Daniel Jackson. Kodkod: A relational model finder. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4424 of *Lecture Notes in Computer Science*, pages 632–647. Springer, 2007.

Биографија аутора

Марија Марковић рођена је 04.08.2000. године у Крушевцу. Основну школу и гимназију завршила је у Александровцу. Математички факултет, Универзитета у Београду, студијски програм Информатика, уписује 2019. године. Основне академске студије завршила је 11.09.2023. године са просечном оценом 9.54. Мастер академске студије на истом студијском програму уписује у октобру 2023. године.

Од октобра 2023. године до јуна 2024. године била је запослена на Математичком факултету у звању Сарадник у настави на Катедри за рачунарство и информатику, где је учествовала у извођењу наставе на курсевима Истраживање података 2, Релационе базе података, Програмирање база података и Програмске парадигме.

Тренутно је запослена у компанији TomTom на позицији Софтверски инжењер 2, где ради на развоју софтверских решења из области мапа и навигације.