Решавање услова у контексту симболичког извршавања

Семинарски рад у оквиру курса
Верификација софтвера
Математички факултет

Видо Младеновић, 1088/2018
vido995@gmail.com

10. децембар 2018

Сажетак

У овом тексту су обрађене неке од техника за проширење опсега програма који су подложни симболичном извршавању и оптимизацији перформанси решавања услова.

Садржај

1 Uvod

Приликом аутоматске анализе исправности програма обично се генеришу услови исправности чију је ваљаност потребно проверити адекватним системом. Систем за проверавање ваљаности може бити саставни део верификацијског алата, тј. специјализован за услове исправности које верификацијски алат генерише или независан од верификацијског алата. Као екстерни системи за проверавање ваљаности често се користе SAT и SMT решавачи. Испитивање ваљаности тада се своди на дуалан проблем испитивања задовољивости негиране формуле.

Проблеми са задовољавањем услова настају у многим доменима, укључујући анализу, тестирање и верификацију софтвера. Решавачи услова доносе одлуке за проблеме који се јављају у логичким изразима, на пример *Проблем задовољивости* (енг. Satisfiability problem, скраћено SAT) проблем је одлучивања да ли за исказну формулу у конјунктивној нормалној форми постоји валуација у којој су све њене клаузе тачне. Иако је SAT добро познати NP-комплетан проблем, недавни напредак је показао границе онога што је тешко када су у питању практичне апликације [1]. Постоји велики број SAT решавача и велики број проблема који се могу решити свођењем на SAT.

Неки проблеми се природније могу описати језицима који су изражајнији од исказних фромула са логичким везама. Из тог разлога *задовољивост у односу на теорију* (енг. Satisfiability Modulo Theory, скраћено SMT) генерализује проблем SAT помоћу одлучивих теорија формирајући формуле које укључују на пример линеарна аритметика и операције над низовима. SMT решавачи мапирају атоме у SMT формули са истинитосним променљивим: процедура одлучивања SAT проверава ваљаност преписане формуле, а решавач теорије проверава модел генерисан SAT процедуром.

SMT решавачи показују неколико предности. Њихови кључни алгоритми су генерички и могу се носити са сложеним комбинацијама неколико појединачних услова. Могу радити инкрементално и уназад како се услови додају или уклањају, и пружају објашњења за недоследности. Теорије могу бити додате и комбиноване на произвољне начине. Процедуре одлучивања се не морају изводити изоловано: често се комбинују како би се смањило време проведено у тежим процедурама нпр решавањем прво линеарних делова у нелинеарној аритметичкој формули. Непотпуни поступци су такође вредни: потпуне али скупе процедуре се позивају само када се не могу произвести задовољавајући одговори. Сви ови фактори омогућавају SMT решавачима да се суоче са великим проблемима које ни једна процедура не може решити самостално.

У симболичком извршавању, решавање услова игра кључну улогу у провери изводљивости неког пута, стварању значења симболичких променљивих и верификовању тврдњи. Током година, симболички извршиоци су користили различите решаваче. На пример, решавач SMT [2] је коришћен у нпр. EXE [3], KLEE [4] и AEG [5], који сви имају своју подршку за теорије бит-вектора и низова.

Постоје различите технике за проширење опсега програма који су подложни симболичном извршењу и оптимизацији перформанси решавања услова. Истакнути приступи се састоје од:

1. смањења величине и сложености услова за проверу,

2. отпуштања решавача путем, на пример, примене кеширања решења услова, одлагања питања решавача или конкретизације и
3. повећања симболичког извршавања приликом решавања проблема у поступцима одлучивања.

Технике које су обрађене су: лењи услови, конкретизација и решавање проблематичних услова.

2 Лењи услови

Лењи услови узимају временским приступ за питања у вези са решавањем услова. У својим иницијалним експериментима, аутори [6] су пратили већину временских интервала у симболичким дељењима и операцијама остатка, уз најгоре случајеве када је операција остатка имала симболичку вредност у имениоцу. Према томе су применили решење које функционише на следећи начин: када извршилац дође до израза гранања који укључује скупу симболичку операцију, узеће обе гране и додати лењи услов на резултат скупе операције у условној путањи. Када истраживање достигне стање које задовољава одређени циљ (нпр. пронађена је грешка), алгоритам ће проверити изводљивост пута и потиснути га ако се сматра неуспешним у стварном извршењу.

У поређењу са жељним приступом провере изводљивости гране која се среће [7], лења стратегија може довести до већег броја активних стања, а самим тим и до више упита решавача. Међутим, аутори [6] сматрају да су одложени упити у многим случајевима ефикаснији од њихових жељених приступа провере изводљивости гране: путање услова која су додата након лењог ограничења могу у суштини умањити простор солуција за решаваче.

3 Конкретизација

Конкретизација [8] говори о ограничењима класичног симболичког извршавања у присуству формула које решавачи услова не могу (бар не ефикасније) решити. Конколички извршилац ствара неки произвољни улаз за програм и извршава га конкретно и симболички: могућа вредност из конкретног извршавања може се користити као симболички операнд који је укључен у формулу која је тешка за решавача, по цену потенцијалног жртвовања исправности у истраживању.

```
1 void test(int x, int y) {
2     if (non_linear(y)) == x)
3         If (x > y + 10) ERROR;
4 }
5 int non_linear(int v) {
6     return  (v*v) % 50;
7 }
```

Пример. У делу приказаног кода, на линији 2 постоји услов облика $\alpha_x = (\alpha_y * \alpha_y) \ \% \ 50$ и грана која се извршава када је тај услов испуњен. Решавач који не подржава нелинеарну аритметику не може да генерише никакав улаз за програм. Међутим, конктеран симболички извршилац може искористити конкретне вредности како би помогао решавачу. На пример, ако су x=3 и y=5 случајно изабрани као почетни улазни параметри онда конкретно извршавање не узима

ни једну од две гране. Без обзира на то, извршилац може поново користити конкретну вредност за y, поједностављујући претходни упит као α_x=5 за α_y=5. Директно решење овог упита сада може користити извршилац за истраживање сваке од двеју грана. Треба имати на уму да ако је вредност y фиксирана на 5, онда нема начина генерисања новог уноса који улази у прву, али не и другу грану. У том случају, тривијално решење би могло бити поновно покретање програма са другачијом вредности за y. (нпр. Ако је y=2 онда је x =4, што задовољава први али не и други услов).

Да би се делимично превазишла некомплетност конкретизације, [8] предлаже се помешано конкретно-симболичко решавање, које разматра све путање услова пре везивања једног или више симбола за конкретну вредност. Заиста, DART[51] конкретизује симболе на основу путање услова како би се доспело до жељене гране. На овај начин, услов садржан у следећој грани на тој путањи се не узима у обзир и можда није задовољавајућ за већ конкретизоване симболе. Ако се ово деси, DART поново покреће извршавање са различитим случајним конкретним вредностима, у нади да ће задовољити услов за следећу грану. Приступ који је представљен у [78] захтева да се детектује решење које задовољава услове дуж целог пута и да одложи конкретизацију колико год је могуће.

4 Решавање проблематичних ограничења

Јаки СМТ решавачи омогућавају извршиоцима директан рад са већим бројем путева ограничења, смањујући потребу да се користи конкретизација. Ово такође резултира мањим ризиком да добијемо лоше конкретне вредности (енг. blind commitment) [8], што се дешава када су испод очекиване путање услова за произвољан избор конкретних вредности за неке променљиве резултира произвољним ограничењем простора за претраживање. Међутим, проблем одлучивања за одређене типове услова је добро познат као неодређен нпр као за нелинеарну целобројну аритметику, или теорију реалних бројева са тригонометријским функцијама које се често користе за моделовање реалних система.

[9] предлаже конколички алгоритам *ходања* који се може суочити са зависностима контролног тока који укључују нелинеарну аритметику и библиотечке позиве. Алгоритам третира додељивање вредности променљивама као простор за процену: решења линеарних услова дефинишу политоп који се може обићи хеуристички, док се преосталим условима додељује погодна функција која мери колико је тачка оцењивања у складу са условом. Прилагођено претраживање се врши над изабраним тачкама одговарајућег политопа и нелинеарни услови се процењују на основу њих. У поређењу са мешовитим конкретно-симболичким решавањем [10], обе технике покушавају да избегну лоше конкретне вредности (енг. blind commitment). Међутим, алгоритам се не ослања на решавача за добијање свих конкретних улаза потребних за процену сложених услова и имплементира хеуристике претраживања које пролазе кроз одговарајуће регионе политопа.

[11] описује симтерично извршавање, нову комбинацију симболичког извршавања уназад (SBE) и стандардног симболичког извршавања. Главна идеја је поделити истраживање у две фазе. У првој фази, SBE се изводи из циљне тачке и сакупља се траг за сваку сле-

дећу путању. Ако се испитују било какави проблематични услови у току повратног извршавања, извршилац их означава као потенцијално задовољавајуће додавањем посебног догађаја у трагу и наставља свој обрнути пролаз. Када год се постави улазна тачка програма по било којој од следећих путања, почиње друга фаза. Извршилац конкретно оцењује прикупљени траг, покушавајући да задовољи сваки услов који је током прве фазе означен као проблематичан. Ово се ради помоћу хеуристичког претраживања, као што је нпр горе описан конколички алгоритам ходања. Предност симетричког у односу на класично конколичко извршавање је то што може спречити истраживање неких неизводљивих путева. На пример, повратна фаза може утврдити да ли у наредби постоји неизводљива грана без обзира на то како је наредба достигнута, док би традиционални конколички извршилац открио неизводљивост само када се достигне наредба, што је неповољно за наредбе које су дубоко у путањи.

5 Zaključak

У оквиру овог рада представљене су неке од техника за оптимизацију перформанси решавања услова као што су лењи услови, конкретизација и решавање проблематичних ограничења. Битно је нагласити да ово нису једине технике, већ их има још, а неке од њих су редукција услова и поновна употреба решења услова.

Наведене технике имају битну улогу у решавању услова, а у симболичком извршавању, решавање услова игра кључну улогу у провери изводљивости неког пута, додељивању значења симболичким променљивим и верификовању тврдњи.

Технике симболичког извршења су се значајно развиле у последњој деценији, и показале су се добро у решавању проблема на неколико домена попут тестирања софтвера, сигурности и анализе кода. Овај тренд није побољшао само постојећа решења, већ је такође довео до стварања нових идеја.

Литература

[1] Leonardo De Moura and Nikolaj Bjørner. 2011. Satisfiability Modulo Theories: Introduction and Applications. Commun. ACM 54, 9 (2011), 69–77. https://doi.org/10.1145/1995376.1995394

[2] Dawson R. Engler and Daniel Dunbar. 2007. Under-constrained Execution: Making Automatic Code Destruction Easy and Scalable. In Proc. of 2007 Int. Symp. on Soft. Test. and Analysis (ISSTA'07). 1–4. https://doi.org/10.1145/1273463.1273464

[3] Cristian Cadar, Vijay Ganesh, Peter M. Pawlowski, David L. Dill, and Dawson R. Engler. 2006. EXE: Automatically Generating Inputs of Death. In Proc. 13th ACM Conf. on Computer and Communications Security (CCS'06). ACM, 322–335. https://doi.org/10.1145/1180405.1180445

[4] Cristian Cadar, Daniel Dunbar, and Dawson R. Engler. 2008. KLEE: Unassisted and Automatic Generation of High-coverage Tests for Complex Systems Programs. In Proc. 8th USENIX Conf. on Operating Systems Design and Implementation (OSDI'08). USENIX Association, 209–224

[5] Thanassis Avgerinos, Sang Kil Cha, Brent Lim Tze Hao, and David Brumley. 2011. AEG: Automatic Exploit Generation. In Proc. Network and Distributed System Security Symp. (NDSS'11).

[6] Roberto Baldoni, Emilio Coppa, Daniele Cono D'elia, Camil Demetrescu and Irene Finocchi,Sapienza University of Rome, A Survey of Symbolic Execution Techniques

[7] Roberto Baldoni, Emilio Coppa, Daniele Cono D'elia, Camil Demetrescu and Irene Finocchi,Sapienza University of Rome, A Survey of Symbolic Execution Techniques, Section 5.1

[8] Cristian Cadar and Koushik Sen. 2013. Symbolic Execution for Software Testing: Three Decades Later. Commun. ACM 56, 2 (2013), 82–90. https://doi.org/10.1145/2408776.2408795

[9] Peter Dinges and Gul Agha. 2014. Solving Complex Path Conditions Through Heuristic Search on Induced Polytopes. In Proc. 22nd ACM SIGSOFT Int. Symp. on Foundations of Software Engineering. 425–436. https://doi.org/10.1145/2635868.2635889

[10] Corina S. Pasareanu, Neha Rungta, and Willem Visser. 2011. Symbolic Execution with Mixed Concrete-symbolic Solving. In Proc. 2011 Int. Symp. on Software Testing and Analysis (ISSTA'11). ACM, 34–44. https://doi.org/10.1145/2001420.2001425

[11] Peter Dinges and Gul Agha. 2014. Targeted Test Input Generation Using Symbolic-concrete Backward Execution. In Proc. 29th ACM/IEEE Int. Conf. on Automated Software Engineering (ASE'14). 31–36. https://doi.org/10.1145/2642937.2642951

[12] Milena Vujošević Janičić, Automatsko generisanje i proveravanje uslova ispravnosti programa