Rešavanje uslova u kontekstu simboličkog izvršavanja

Seminarski rad u okviru kursa
Verifikacija softvera
Matematički fakultet

Lazar Ranković, 1099/2016
lazar.rankovic@outlook.com

10. decembar 2018

Sažetak

Prilikom simboličkog izvršavanja rešavanje uslova je ključan deo. U ovom radu je pomenut način rešavanja uslova, glavni akcenat je na metodima ubrzavanja rešavanja uslova. Predstavljena su dva skupa metoda za ubrzavanje rešavanja uslova koji mogu značajno da ubrzaju rešavanje uslova, a samim tim i poboljšaju performanse simboličkog izvršavanja.

Sadržaj

1 Uvod u simboličko izvršavanje

Simboličko izvršavanje predstavlja tehniku verifikacije softvera u kojoj se konkretne vrednosti ulaza u program apstrahuju simbolima. Potreba za apstrakcijom konkretnih vrednosti simboličkim promenljivama proizilazi iz nemogućnosti potpunog testiranja softvera tradicionalnim pristupima. Tradicionalni pristupi testiraju softver tako što za ulaz koriste konkretne vrednosti i analiziraju ponašanje programa. Prostor ulaza može biti veliki (nekada i neograničen) te tradicionalne tehnike ne omogućavaju celovito testiranje.

Prilikom simboličkog izvršavanja promenljive u kodu su zamenjene simboličkim promenljivama te se program izvršava nad simbolima i prolaskom kroz kod se obrazuju sve moguće putanje u programu. Svaka putanja sadrži uslov putanje koji se sastoji iz svih odluka koje su dovele do te putanje. Uslov na početku simboličkog izvršavanja svakog programa je tačno. Uslov putanje predstavlja formulu (bez kvantifikatora). Sve putanje programa obrazuju stablo izvršavanja. Jedna od ključnih stvari u simboličkom izvršavanju je to što simboličko izvršavanje omogućava određivanje kritičnih ulaza za svaku od putanja. Puštanjem uslova putanja kroz SMT rešavač [3] mogu se dobiti kritične vrednosti ulaza za koje program neće dobro raditi.

2 Rešavanje uslova

Rešavači uslova predstavljaju procedure odlučivanja za probleme opisane logičkim formulama. [2] Jedan od primera je rešavanje *problema zadovoljivosti* (eng. boolean satisfiability problem) SAT problema. U okviru ovog problema formule koje se rešavaju su iskazne formule. U kontekstu simboličkog izvršavanja potreban je ekspresivniji jezik od logičkih formula. Problemi zadovoljivosti u teoriji (eng. satisfiability modulo theories) (SMT) [4] generaliziju SAT problem tako što omogućavaju dodatne teorije kao što su: bit-vektoska teorija, linearna aritmetika, operacije nad nizovima i mnoge druge. U duhu simboličkog izračunavanja rešavanje uslova igra bitnu ulogu u proveravanju dostižnosti puteva i verifikovanja pretpostavki. Neki od rešavača su: STP [10] koji se koristi u EXE [6], KLEE [5] и AEG [1] alatu. Ostali rešavači kao što je Java PathFinder [7] sadrže dodatne procedure odlučivanja.

Trenutno jedna od najboljih rešavača je Z3 rešavač [9] koji je proizvela kompanija Microsoft.

U nastavku će biti predstavljena dva skupa metoda za ubrzavanje rešavanja uslova. Prvi skup metoda je *redukovanje uslova* čija je suština smanjenje dimenzionalnosti uslova. Drugi skup metoda, *ponovno korišćenje rešenja uslova*, se bavi upotrebom već rešenih uslova zarad izuzimanja ponovnih izračunavanja istih uslova, te je ubrzanje koje se ovde postiže očigledno.

2.1 Redukovanje uslova

Ovaj skup metoda ima za cilj da smanji dimenziju i kompleksnost formule koja se proverava. Jedan od primera je tehnika prezapisivanja koja se koristi prilikom rešavanja logičkih formula i cilj joj je smanjenje broj promenljivih bez promene semantike.

Neki od načina redukovanja uslova:

- *optimizacija usled nezavisnosti uslova* (eng. constraint independence optimization) - oslanja se na činjenicu da skup uslova može biti podeljen na više nezavisnih skupova uslova. Ova činjenica omogućava brže rešavanje uslova jer se nebitna ograničenja uklanjaju. Ova tehnika uvedena je u alatu EXE [6].
- *konkretizacija podrazumevane vrednosti* (eng. implied value concretization) - u slučaju da postoji promenljiva x koja ima konkretnu vrednost (npr. x = 21) svi uslovi koji sadrže pojavljivanje ove promenljive i zadovoljeni su konkretnom vrednošću (npr. x > 11) se mogu redukovati u tačno. Takođe, promenljiva se na svim mestima može konkretizovati, tj. zameniti inicijalizovanom vrednošću. Ovaj vid redukovanja implementiran je u alatu KLEE [5].
- *pojednostavljivanje izraza bitovskih polja* (eng. bitfield-theory expression simplifier) - omogućava zamenu simboličkih vrednosti konkretnim vrednostima koje bitovske operacije maskiraju. Ovaj vid redukovanja implementiran je u alatu S^2E [8].

2.2 Ponovno korišćenje rešenja uslova

Ponovno korišćenje rešenja uslova daje dobre rezulate posebno kada se kombinuje sa optimizacijom usled nezavisnosti uslova. Najviše pristupa ponovnog korišćenja rešenja uslova se zasniva na sintakstičnoj i semantičkoj jednakosti uslova.

Neki od načina ponovnog korišćenja rešenja uslova su:

- *keširanje* - EXE [6] kešira rezultate izračunavanja uslova u cilju što manjeg broja poziva rešavača. Kad god je potrebno rešavanje uslova sistem pre poziva proveri da li je uslov već rešen i smešten u keš.
- *keširanje kontraprimera* (eng. counterexample caching) - spada u grupu inkrementalnih strategija. Kada se uslov S propusti kroz rešavač rešenje se kešira u memoriju na dva načina. Ukoliko je uslov zadovoljiv promenljive se zamenjuju konkretnim vrednostima za koje je taj uslov zadovoljiv, u suprotnom, kada uslov nije zadovoljiv, promenljive se zamenjuju specijalnom null vrednošću. Ako je keširani uslov nezadovoljiv, a nakon toga se izračunava uslov T i važi da je T $\subset$ S, onda je i T nezadovoljiv. Suprotno ako je keširan uslov zadovoljiv važi i da je S $\subset$ T, tada je i T zadovoljiv.

  Ako se rešava uslov S i već postoji nekoliko rešenja uslova u kešu, tada algoritam pokušava da kombinovanjem rešenja iz keša reši uslov S.
- *zapamćeno simboličko izračunavanje* (eng. memoized symbolic execution) - odabrani izbori tokom istraživanja puteva se enkodiraju u prefiksnom stablu otvarajući priliku za ponovno korišćenje izračunatih rezultata tokom uspešnih pokretanja. Ovo je posledica zapažanja da se tokom simboličkog izračunavanja potproblemi izračunavaju veliki broj puta jer se tokom ispravke manjih bagova izračunavanje ponovo pušta.
- *kanoničko ponovno korišćenje* - ovaj vid ponovnog korišćenja predstavlja ponovnu upotrebu rešenja čak i na različitim programima. Uslovi se raznim transformacijama dovode do kanonične forme koja omogućava ponovnu upotrebu. Ovaj vid ponovnog korišćenja je predstavljen u [11].

3 Zaključak

U prethodnom tekstu prikazane su neke od tehnika ubrzavanja rešavanja uslova u konktekstu simboličkog izvršavanja. Predstavljeni pristupi mogu značajno da ubrzaju rešavanje uslova u situacijama kada ih je moguće primeniti. Pristup redukovanja uslova je standardan pristup u rešavanju logičkih iskaza. Ponovno korišćenje rešenja uslova očigledno je da može značajno ubrzati rešavanja logičkih izraza jer se ponovna izračunavanja izbacuju. Ovo se može neformalno shvatiti kao vid dinamičkog programiranja u kom se rezultati pamte tokom rešavanja nekog problema. Bilo bi vrlo zanimljivo istražiti da li postoje, a ako ne postoje onda i probati sa istrazivanjem, algoritama koji kombinuju keširan skup tako da ako je moguće iz keširanog skupa izgenerisati uslov koji se rešava. Na ovaj način bi još veći skup uslova bio unapred rešen, što bi dodatno optimizovalo rad simboličkog izvršavanja.

Literatura

[1] Thanassis Avgerinos, Sang Kil Cha, Brent Lim Tze Hao, and David Brumley. AEG: automatic exploit generation. In *NDSS*. The Internet Society, 2011.

[2] Roberto Baldoni, Emilio Coppa, Daniele Cono D'Elia, Camil Demetrescu, and Irene Finocchi. A survey of symbolic execution techniques. *ACM Comput. Surv.*, 51(3), 2018.

[3] Clark Barrett, Daniel Kroening, and Thomas Melham. *Problem solving for the 21st century: Efficient solver for satisfiability modulo theories*. Knowledge Transfer Report, Technical Report 3. London Mathematical Society and Smith Institute for Industrial Mathematics and System Engineering, 6 2014.

[4] Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. *Satisfiability modulo theories*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. 1 edition, 2009.

[5] Cristian Cadar, Daniel Dunbar, and Dawson Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 209–224, Berkeley, CA, USA, 2008. USENIX Association.

[6] Cristian Cadar, Vijay Ganesh, Peter M. Pawlowski, David L. Dill, and Dawson R. Engler. Exe: Automatically generating inputs of death. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 322–335, New York, NY, USA, 2006. ACM.

[7] Vitaly Chipounov, Volodymyr Kuznetsov, and George Candea. The s2e platform: Design, implementation, and applications. *ACM Trans. Comput. Syst.*, 30(1):2:1–2:49, February 2012.

[8] Vitaly Chipounov, Volodymyr Kuznetsov, and George Candea. The s2e platform: Design, implementation, and applications. *ACM Trans. Comput. Syst.*, 30(1):2:1–2:49, February 2012.

[9] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and*

Analysis of Systems, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[10] Vijay Ganesh and David L. Dill. A decision procedure for bit-vectors and arrays. In Werner Damm and Holger Hermanns, editors, *Computer Aided Verification*, pages 519–531, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[11] Willem Visser, Jaco Geldenhuys, and Matthew B. Dwyer. Green: Reducing, reusing and recycling constraints in program analysis. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 58:1–58:11, New York, NY, USA, 2012. ACM.