

Jezik za dokazivanje funkcionalne korektnosti - Dafni

Seminarski rad u okviru kursa
Verifikacija softvera
Matematički fakultet

Strahinja Stanojević

21. maj 2018.

Sažetak

U ovom radu je opisan programski jezik Dafni (eng. Dafny) koji služi za verifikaciju algoritama, i programa. Dafni u suštini radi tako što pravi specifikaciju programa, zadaje preduslove i postuslove i zavisno od njihove ispunjenosti i zadovoljenj cele specifikacije dokazuje da je program ispravan, odnosno neispravan.

Sadržaj

1	Uvod	2
2	Odlike jezika Dafni	2
2.1	Tipovi	2
2.2	Pred i post uslovi	2
2.3	Duhovi promenljive	2
2.4	Izmene	3
2.5	Specifikacija invarijanti struktura podataka	3
2.6	Skupovi	4
2.7	Parametarski tipovi	4
2.8	Algebarski tipovi	4
2.9	Funkcije	5
3	Slični jezici	5
4	Zaključak	5

1 Uvod

Dokazivači koji se koriste za verifikaciju programa idu u dve krajnosti. U jednoj krajnosti oni koriste interaktivne asistente za dokazivanje što zahteva visok nivo stručnosti osobe koja ih koristi kako bi "vodili" rešavač kroz niz simboličkih manipulacija. S druge strane postoje dokazivači koji imaju već određene procedure za odlučivanje [1][2] (SMT) ili oni koji koriste iste.

U ovom radu će biti opisan jezik Dafni. Dafni radi tako što program prevodi u jezik za verifikaciju Bugi 2 [5][6] (eng. Boogie 2) tako da ako se dokaže korektnost Bugi programa iz nje sledi da je i Dafni program korektan. Kao i kod velikog broja jezika za dokazivanje semantika Dafni je definisana u terminima Bugija. Bugi se dalje koristi kako bi generisao uslove korektnosti u logici prvog reda što se dalje prosleđuje SMT rešavaču Z3.

2 Odlike jezika Dafni

Jezik je imperativan, sekvencijalan, klasni, podržava alokaciju memorije, i ima ugrađene konstrukte za pravljenje specifikacije. Jezik takođe podržava i korisnički definisane matematičke i algebarske tipove kao duhove promenljive (eng. ghost variables). Pored navedenog postoje i skupovi kao i sekvence.

2.1 Tipovi

Tipovi koji postoje u Dafni su logičke promenljive (eng. boolean), celobrojne vrednosti, (moguće i NULL) reference na instance korisnički definisanih klasa, skupovi, sekvence i korisnički definisani algebarski tipovi. Ne postoje podtipovi, s izuzetkom da su svi klasni tipovi zapravo podtipovi ugrađenog tipa objekat (eng. object).

2.2 Pred i post uslovi

Metode imaju standardnu deklaraciju preduslova (ključna reč **requires**) i postuslova (ključna reč **ensures**). Na primer, naredni metod garantuje da će postaviti izlazni parametar r na celobrojnu vrednost korena ulaznog parametra n , koji mora biti pozitivan ili 0.

```
method ISqrt(n: int) returns (r: int)
    requires 0 ≤ n;
    ensures r * r ≤ n ∧ n < (r + 1) * (r + 1);
{ /*method body goes here...*/ }
```

Zadatak pozivaoca je da obezbedi korektnost poreduslova, a zadatak implementacije da obezbedi korektnost postuslova. Ukoliko dođe do greške u bilo kom delu, dokazivač će prijaviti grešku.

2.3 Duhovi promenljive

Jednostavna i veoma dobra karakteristika jezika je što promenljive mogu biti deklarisanе kao duhovi (eng. ghost). Takva promenljiva se koristi u verifikaciji ali nije potrebna u izvršavanju (eng. run time). Zbog toga kompajler (eng. compiler) može da izbegne alokaciju memorije za duh promenljivu. Jedino ograničenje je da vrednost duh promenljive ne

sme biti dodeljena stvarnoj (fizičkoj promenljivoj). To se može proveriti sintaksnom analizom. Kao i sve druge promenljive i duhovi promenljive mogu da dobijaju vrednost naredbom dodele. Na primer:

```
class C {
    var x: int; var y: int; ghost var g: int;
    method Update() modifies{this}
    {x := x + 1; g := g + 2;}
}
```

2.4 Izmene

Jedna od važnih stavki specifikacije metoda je definisanje koje delove programa je dozvoljeno menjati. Ovo se zove *framing* i u Dafni se radi pomoću **modifies** kao u prethodnom primeru. Zbog jednostavnosti, u Dafni se **modifies** odnosi na ceo objekat a ne na neko polje istog. Kada kažemo da neki objekat može da se menja to znači i da svako polje tog objekta može da se menja. Ukoliko želimo da obezbedimo da neka promenljiva ne može da se menja, to se može postići pomoću **ensures**. Na primer, ukoliko bismo želeli da u prethodnom primeru promenljiva y ne može da bude izmenjena to bi se postiglo pomoću **ensures** y = **old**(y);

Klauza **modifies** metoda mora da uračuna sve moguće izmene metoda. Zbog velikog broja objekata i njihovih polja koja se mogu naći u skupu objekata koje metod može da izmeni kao i zbog činjenice da ovaj skup može dinamički da se menja ovo deluje kao jako komplikovan posao. Dafni to rešava *dynamic framing-a*, koristeći "sirovu" **modifies** klauzu, što omogućava da frame bude određen pomoću vrednosti duh promenljive. Standardni idiom je da se deklarise duh promenljiva tipa skup, koja dinamički održava skup objekata koji se mogu menjati.

```
class MyClass {
    ghost var Repr: set<object>;
    method SomeMethod() modifies Repr { /*...*/ }
}
```

Klauza **modifies** dozvoljava metodu da promeni bilo koje polje bilo kog objekta koji se nalazi u Repr.

2.5 Specifikacija invarijanti struktura podataka

Kada imamo program koji predstavlja nekoliko slojeva modula, važno je da imamo specifikaciju koja pruža jedan apstrakt nad detaljima svakog modula. Postoji više različitih pristupa, a Dafni koristi jedan koji je inspirisan dinamičkim frejmovima (eng. dynamic frames), gde je ideja da se frejmovi definišu kao skup struktura podataka koji se dinamički menja i gde konzistentnost struktura podataka zavisi od ispravnosti funkcija.

Duh promenljiva Repr predstavlja dinamički frejm reprezentacije objekata. Dafni program treba eksplicitno da da ažurira promenljivu Repr kad dođe do promene objekta. Funkcija Valid() služi da definiše šta za objekat znači da je u konzistentnom stanju.

Metod Init() služi za konstrukciju objekta. On takođe kaže da može da izmeni polja objekta koji se inicijalizuje, što uključuje i Repr. S obzirom da može da menja Repr, u postuslovim je navedeno i kako se to može desiti. Naredba **fresh** (Repr - **this**); znači da su svi objekti osim **this**

inicijalizovani u pozivu `Init()`; Dakle, može se zaključiti da je `c.Repr` različit od svih prethodnih skupova objekata.

Metod `Update()` zahteva i održava invarijantu objekta, `Valid()`. Postuslov **fresh**(...) kaže da će `Update` dodati samo novo alocirane objekte u `Repr`, što znači da neće doći do promena u objekata iz početnog skupa `Repr`.

```
class C {
  ghost var Repr: set <object>; function Valid(): bool
    reads {this} ∪ Repr;
    {this ∈ Repr ∧ ... };
  method Init()
    modifies {this};
    ensures Valid() ∧ fresh(Repr - {this});
    { ... Repr := {this} ∪ ...; }
  method Update()
    requires Valid();
    modifies Repr();
    ensures Valid() ∧ fresh(Repr - old (Repr));
    { ... }
  ...
}
```

2.6 Skupovi

Dafni podržava konačne skupove. Skup elemenata tipa `t` se deklarise kao `set<T>`. Podržane operacije su element (da li element pripada skupu), unija, razlika, podskup, ali ne i komplement i kardinalnost.

2.7 Parametarski tipovi

Klase, metodi, funkcije, algebarski tipovi su generički, što znači da mogu da primaju parametarske tipove.

```
class Pair <T> {
  var a: T; var b: T;
  method Flip() modifies {this}; ensures a = old(b) ∧ b = old (a);
    { var tmp := a; a := b; b := tmp; }
}
```

Korišćenje generičkih tipova zahteva instancijaciju. Izraz koji ima parametarski tip u sebi može da se posmatra samo parametarski, odnosno Dafni ne vrši nikakvo kastovanje. Na primer, telo metoda `Flip()` ne može da koristi promenljive `a` i `b` kao celobrojne, ali klijentski kod u narednom primeru može.

```
var p := new Pair<int>; p.b := 7; call p.flip(); assert p.a = 7;
```

2.8 Algebarski tipovi

Algebarski tipovi se definišu kao skup strukturnih vrednosti. Na primer, generičko drvo koje podatke čuva u listovima se može definisati na sledeći način:

```
datatype Tree <T> { Leaf(T); Branch(Tree<T>, Tree<T>); }
```

U ovoj deklaraciji imamo zapravo 2 konstruktora, jedan za grane, jedan za listove. Konstruktor se može pozvati pomoću `#Tree.Leaf(5)` za stablo `Tree<int>`.

2.9 Funkcije

U Dafni mogu biti definisane matematičke funkcije. Imaju potpis, koji može da uključuje parametarske tipove, specifikaciju funkcije, i telo. Uzmimo za primer narednu funkciju:

```
function F(x: T) requires P; reads R; decreases D; { Body }
```

T predstavlja neki tip, P je logički izraz, R je skupovni izraz, D je lista izraza, i Body je izraz, ne naredba. Klauza **requires** u kojim stanjima funkcija može da se koristi. Može se desiti da funkcija može biti pozvana samo pod određenim uslovima, što se rešava upravo na ovaj način. Pomoću klauze **reads** se definišu objekti od kojih funkcija može da zavisi. Klauza **decreases** pruža metriku prekida koja služi za verifikaciju (zaustavljanja) rekurzivnih funkcija. Podrazumevano su funkcije "duhovi" i mogu se koristiti samo u specifikaciji. Ali ako se funkcija deklarise sa **function method** onda funkcija može biti korišćena i u iskompajliranom kodu.

3 Slični jezici

U ovom poglavlju biće predstavljeni neki jezici koji počivaju na sličnim konceptima kao i Dafni.

Javin jezik za modeliranje [3] (eng. Java Modeling Language - JML) ima dosta sličnosti sa Dafni. Ima slične načine specifikacije kao i Dafni. Ključna razlika između ova 2 jezika je u tome što Dafni koristi automatski a JML interaktivni dokazivač.

Spec# [4] je objektno-orijentisani programski jezik sa specifikacijom. Bazira se na automatskim SMT-rešavačima. Za razliku od Dafni nema duhove promenljive, skupove, algebarske tipove, sekvence i metriku zaustavljanja.

I JML I Spec# imaju čiste metode, za razliku od Dafni koji koristi matematičke funkcije.

4 Zaključak

U ovom radu u predstavljeni osnovni koncepti na kojima se zasniva jezik Dafni koji služi za verifikaciju. Objašnjeni su tipovi koje koristi, funkcije i strukture podataka. Takođe su predstavljene i neke od osnovnih ideja na kojima pored njega počivaju i drugi jezici za verifikaciju, kao što su preduslovi, postuslovi i slično. Napravljeno je kratko poređenje Dafni sa drugim sličnim jezicima. Autor takođe navodi dokaz algoritam Schorr-Waite koji je uspešno dokazan pomoću ovog jezika. Za dokaz je potrebno manje od 5 sekundi. Autor navodi i da je uspešno rešio veliki broj problema vezanih za verifikaciju koristeći upravo Dafni. Rad je napisan 2010. godine. Danas je Dafni još popularniji, i ima veliki broj pokušaja dokazivanja njime na raznim takmičenjima. Često je korišćen među pobedničkim timovima na istim.

Literatura

- [1] David Detlefs, Greg Nelson, and James B. Saxe. Simplify: a theorem prover for program checking. *Journal of the ACM*, 52(3):365–473, May 2005.
- [2] Clark Barrett, Silvio Ranise, Aaron Stump, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org, 2008.
- [3] Gary T. Leavens, Albert L. Baker, and Clyde Ruby. Preliminary design of JML: A behavioral interface specification language for Java. *ACM SIGSOFT Software Engineering Notes*, 31(3):1–38, March 2006.
- [4] Mike Barnett, K. Rustan M. Leino, and Wolfram Schulte. The Spec# programming system: An overview. In Gilles Barthe, Lilian Burdy, Marieke Huisman, Jean-Louis Lanet, and Traian Muntean, editors, *CASSIS 2004, Construction and Analysis of Safe, Secure and Interoperable Smart Devices*, volume 3362 of *Lecture Notes in Computer Science*, pages 49–69. Springer, 2005.
- [5] Mike Barnett, Bor-Yuh Evan Chang, Robert DeLine, Bart Jacobs, and K. Rustan M. Leino. Boogie: A modular reusable verifier for object-oriented programs. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul de Roever, editors, *Formal Methods for Components and Objects: 4th International Symposium, FMCO 2005*, volume 4111 of *Lecture Notes in Computer Science*, pages 364–387. Springer, September 2006.
- [6] K. Rustan M. Leino. This is Boogie 2. Manuscript KRML 178, 2008. Available at <http://research.microsoft.com/~leino/papers.html>