UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

David Nestorović

PRIMENA KOMPRESOVANOG GLOB-PREFIKSNOG STABLA U SISTEMU GRAALVM NATIVE IMAGE

master rad

Beograd, 2026.

Mentor:

prof. dr Milena Vujošević Janičić, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

prof. dr Vesna Marinković, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

Strahinja Stanojević, asistent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: 2026

Mami, tati, sestri Teodori, supruzi Ivani, i kolegama iz Oracle Labs-a

Naslov master rada: Primena kompresovanog glob-prefiksnog stabla u sistemu GraalVM Native Image

Rezime: Savremeni softverski sistemi zahtevaju mehanizme koji omogućavaju pouzdano funkcionisanje uz istovremeno smanjenje mogućnosti nastanka korisničkih grešaka. Jedan od načina za postizanje ovog cilja jeste pažljivo odmeravanje izražajne moći ulaznih specifikacija. Previše izražajni formati mogu povećati složenost obrade, potrošnju resursa i mogućnost grešaka, dok previše restriktivni formati ograničavaju praktičnu upotrebljivost. Stoga je pronalaženje odgovarajuće ravnoteže između fleksibilnosti i jednostavnosti važan aspekt projektovanja softverskih interfejsa.

U ovom radu analiziran je i rešen problem efikasnog definisanja i obrade šablona za registraciju Java resursa u okviru sistema *GraalVM Native Image*. Postojeći mehanizam zasnovan na Java regularnim izrazima pruža izražajnu moć koja prevazilazi stvarne potrebe sistema, što povećava mogućnost korisničkih grešaka, zahteva uključivanje celokupnog mehanizma za obradu regularnih izraza u izvršivu datoteku i onemogućava efikasno strukturiranje obrazaca. Kao alternativa, u radu su uvedeni jednostavniji glob obrasci i projektovana je specijalizovana struktura podataka, kompresovano glob-prefiksno stablo, namenjena njihovoj efikasnoj obradi.

Predloženo rešenje obuhvata celokupan proces obrade šablona za registraciju resursa, uključujući njihovu validaciju, tokenizaciju, determinističko sortiranje prema opštosti, prepoznavanje i eliminaciju redundantnih obrazaca, konstrukciju kompresovanog glob-prefiksnog stabla i efikasno pretraživanje putanja u formiranoj strukturi. Posebna pažnja posvećena je mogućnosti odsecanja grana tokom pretrage, čime se izbegava iscrpna provera svih obrazaca i značajno smanjuje broj potrebnih provera. Rešenje podržava i specifične zahteve sistema *GraalVM Native Image*, uključujući uslovnu registraciju resursa i registraciju resursa u okviru Java modula.

Rezultati pokazuju da predloženi pristup smanjuje prostor za korisničke greške pri definisanju obrazaca i omogućava smanjenje veličine izvršive datoteke eliminacijom potrebe za mehanizmom za obradu Java regularnih izraza. Istovremeno, strukturirano organizovanje obrazaca u kompresovanom glob-prefiksnom stablu omogućava efikasnije pronalaženje odgovarajućih obrazaca, kao i značajno poboljšanje efikasnosti registracije resursa i pronalaženja obrazaca koji nedostaju u konfiguracionim datotekama. Na taj način ostvaruje se povoljan kompromis između izražajnosti i jednostavnosti šablona, memorijskih zahteva i efikasnosti obrade u sistemu *GraalVM Native Image*.

Ključne reči: prefiksna stabla, strukture podataka, algoritmi, GraalVM, Native Image, glob, regex

Sadržaj

Glava 1

Uvod

Za stabilno i pouzdano funkcionisanje savremenih softverskih sistema od ključne je važnosti stvoriti okruženje koje minimizuje mogućnost pravljenja korisničkih grešaka [10, 13]. Efikasan način da se ovo postigne jeste pažljivo odmeravanje izražajne moći ulaznih specifikacija koje se nameću korisnicima. U praksi često postoji inverzan odnos između fleksibilnosti zapisa i efikasnosti njegove obrade: prevelika izražajnost višestruko uvećava broj opcija koje sistem mora da analizira, što se negativno odražava na brzinu rada i zauzeće resursa. S druge strane, preterano restriktivne specifikacije narušavaju praktičnu upotrebljivost. Stoga je pronalaženje pravog balansa između jednostavnosti i funkcionalnosti jedan od centralnih izazova pri projektovanju softverskih interfejsa.

Ovaj problem je naročito izražen u okviru okruženja *GraalVM Native Image* prilikom definisanja putanja do Java resursa u konfiguracionim datotekama. Dosadašnja praksa oslanjala se na Java regularne izraze, čija izražajna moć znatno prevazilazi stvarne potrebe sistema. Takav pristup je često dovodio do nenamernih grešaka u konfiguraciji, uključivanja nepotrebnih resursa u izršnu datoteku, povećanog memorijskog zauzeća i znatno složenije verifikacije ispravnosti.

U ovom radu analizira se i unapređuje mehanizam definisanja i obrade šablona za poklapanje putanja ka Java resursima unutar sistema *GraalVM Native Image*. Takođe, biće pokazano kako zamena kompleksnih regularnih izraza jednostavnijim glob obrascima, u kombinaciji sa modifikovanom strukturom prefiksnog stabla (kompresovanim glob-prefiksnim stablom), uspešno rešava navedene nedostatke. U radu će biti izloženo i kako nepromenljiva struktura kompresovanog glob-prefiksnog stabla omogućava izuzetno brzo pretraživanje obrazaca, uz mogućnost efikasnog odsecanja grana, dok korisnicima pruža jednostavniji format koji je manje podložan greška-

ma pri definisanju resursa. Time je postignuta bolja ravnoteža između fleksibilnosti šablona i performansi samog sistema *GraalVM Native Image*. Predloženo rešenje biće implementirano u programskom jeziku Java i obuhvataće celokupan ciklus obrade šablona za registraciju resursa: od validacije, tokenizacije i determinističkog sortiranja glob obrazaca po opštosti, preko same konstrukcije kompresovanog prefiksnog stabla, pa sve do efikasnog algoritmatskog pretraživanja putanja u nastaloj strukturi.

Rad je organizovan na sledeći način. Poglavlje 2 detaljno opisuje arhitekturu sistema *GraalVM*, komponentu *Native Image*, dinamička svojstva programskog jezika Java, kao i ograničenja i proces registracije resursa. Poglavlje 3 analizira šablone za poklapanje teksta, sa fokusom na poređenje Java regularnih izraza i glob obrazaca nad putanjama Java resursa. Poglavlje 4 opisuje prefiksna stabla, njihovu strukturu i osnovne operacije. Poglavlje 5 predstavlja centralni deo rada u kom je detaljno izloženo kompresovano glob-prefiksno stablo, zajedno sa procesima tokenizacije, validacije, sortiranja glob obrazaca, kao i samom konstrukcijom i pretragom u stablu. Na kraju, poglavlje 6 donosi zaključak rada.

Glava 2

Sistem GraalVM

Sistem GraalVM [14, 5] predstavlja kompajlersku infrastrukturu projektovanu sa ciljem ubrzanja izvršavanja aplikacija napisanih u programskom jeziku Java, kao i izvršavanja drugih programskih jezika koji se zasnivaju na Java virtuelnoj mašini (engl. *Java Virtual Machine - JVM*). Funkcionisanje sistem *GraalVM* oslanja se na tri osnovne komponente:

1. Java virtuelna mašina *Java HotSpot VM* [12],
2. Kompajler GraalVM [4] — JIT (engl. *Just-In-Time*) kompajler namenjen programskom jeziku Java,
3. Biblioteka *Truffle* [7] — biblioteka razvijena sa ciljem da omogući izvršavanje jezika kao što su *Python*, *Ruby*, *R*, *C*, *C++* i drugih na Java virtuelnoj mašini.

Primenom navedene arhitekture, primarni fokus projekta usmeren je ka unapređenju performansi jezika iz JVM familije, sa ciljem da se brzina njihovog izvršavanja približi brzini programa koji se unapred prevode u izvršive datoteke (engl. *native languages*). Pored toga, teži se smanjenju vremena potrebnog za pokretanje aplikacije, kao i smanjenju perioda neophodnog da ona dostigne potpuno optimizovani režim rada. Istovremeno, ova arhitektura omogućava razvoj poliglot aplikacija (engl. *polyglot applications*), odnosno definisanje funkcija u različitim programskim jezicima unutar istog izvornog koda. Naposletku, obezbeđuje se kontrolisano i predvidivo izvršavanje aplikacije putem ograničavanja mogućnosti spoljne manipulacije tokom rada samog koda.

Realizacijom navedenih ciljeva, *GraalVM* obezbeđuje superiornije performanse u pogledu brzine izvršavanja aplikacija i memorijskog zauzeća, kao i viši nivo bezbednosti u poređenju sa tradicionalnim pristupom izvršavanju Java aplikacija.

2.1 Komponenta *GraalVM Native Image*

Pored osnovnih funkcionalnosti, ključna karakteristika koja izdvaja *GraalVM* od ostalih Java razvojnih kompleta (engl. *Java Development Kit - JDK*) jeste mogućnost prevođenja Java aplikacija u samostalne izvršive datoteke (engl. *standalone executables*). U ovu svrhu, sistem *GraalVM* nudi komponentu *Native Image*, koja predstavlja infrastrukturu zasnovanu na kompajleru namenjenom prevođenju pre samog vremena izvršavanja — AOT (engl. *Ahead-Of-Time*) kompajleru. Izvršiva datoteka generisana upotrebom tehnologije *GraalVM Native Image* obuhvata u potpunosti optimizovan kôd, kao i sve prateće komponente koje su neophodne za neposredno i samostalno izvršavanje aplikacije uz postizanje maksimalnih performansi.

Kako bi ovakav pristup bio moguć, *GraalVM Native Image* svoj rad zasniva na pretpostavci zatvorenog sveta (engl. *closed-world assumption*). Osnovna polazna tačka ovog koncepta jeste da su sav kôd, kao i sve informacije neophodne za njegovo izvršavanje, u potpunosti poznati u trenutku kreiranja izvršive datoteke. Ovakva konzervativna orijentacija nosi sa sobom određene prednosti, ali i nedostatke za komponentu *GraalVM Native Image*. Sa jedne strane, prednost leži u činjenici da *GraalVM Native Image* unapred u potpunosti poznaje kôd namenjen optimizaciji, čime se omogućava postizanje boljih performansi, kao i sve moguće tokove izvršavanja aplikacije, što povećava bezbednost usled smanjenja broja nepredviđenih scenarija. Sa druge strane, osnovni nedostatak ogleda se u tome što je prikupljanje svih neophodnih informacija unapred često izuzetno otežano usled dinamičke prirode programskog jezika Java.

2.2 Dinamička svojstva programskog jezika Java

Jedno od ključnih dinamičkih svojstava programskog jezika Java jeste refleksija [6] (engl. *reflection*)— mehanizam koji programu omogućava da u toku izvršavanja ispituje i vrši manipulaciju nad strukturom klasa, interfejsa, metoda, polja i konstruktora, nezavisno od njihovog nivoa vidljivosti i bez potrebe za poznavanjem njihove konkretne implementacije. Pored refleksije, Java obezbeđuje i druga dinamič-

ka svojstva koja dodatno doprinose fleksibilnosti izvršavanja aplikacija. Dinamičko učitavanje i upravljanje resursima omogućava pristup spoljnim datotekama i konfiguracionim podacima tokom samog izvršavanja programa. Dodatno, mehanizam dinamičkih proksija [2] omogućava presretanje i prilagođavanje poziva metoda u toku izvršavanja, čime se ostvaruje implementacija naprednih funkcionalnosti kao što su bezbednosne provere, evidencija poziva i upravljanje transakcijama. Naposletku, serijalizacija predstavlja dinamičko svojstvo pomoću kojeg se objekti prevode u format pogodan za skladištenje ili prenos, sa mogućnošću njihove naknadne rekonstrukcije u izvorni oblik.

Pored navedenih, programski jezik Java obuhvata i niz drugih dinamičkih funkcionalnosti, međutim, detaljnije razmatranje njihovih karakteristika ne ulazi u opseg ovog rada.

2.3 Ograničenja komponente *GraalVM Native Image*

Navedena dinamička svojstva programskog jezika Java predstavljaju najveći izazov u ispunjavanju zahteva koncepta zatvorenog sveta. Naime, kako bi se izgradila samostalna izvršiva datoteka, neophodno je unapred obezbediti informacije o svim dinamičkim svojstvima koja aplikacija potencijalno koristi. Za deklarisanje ovih informacija, *GraalVM Native Image* koristi konfiguracione datoteke zapisane u formatu *JSON* (engl. *JavaScript Object Notation*)[1]. Izgled i struktura navedene konfiguracione datoteke ilustrovani su u listingu 2.3.

```
{
  "reflection" : [
    {
      "condition" : {
        "typeReached" : "org.codehaus.classworlds.EmbeddedLauncher"
      },
      "type" : "java.lang.String[]"
    },
    {
      "condition" : {
        "typeReached" : "org.codehaus.classworlds.Launcher"
```

[1]U novijim verzijama sistema GraalVM koristi se jedinstvena, objedinjena konfiguraciona datoteka.

```
      },
      "type" : "java.lang.String[]"
    }
  ]
}
```

Listing 2.1: Primer konfiguracione datoteke

2.4 Registracija Java resursa u komponenti *GraalVM Native Image*

Većina aplikacija napisanih u programskom jeziku Java u određenom trenutku zahteva pristup spoljnim resursima, kao što su slikovne ili tekstualne datoteke i drugi prateći sadržaji. Kod programa koji se prevode primenom standardnog Java JIT kompajlera, ovakvi resursi se učitavaju dinamički, u trenutku kada nastane potreba za njima tokom samog vremena izvršavanja (engl. *runtime*). Nasuprot tome, prevođenje programa korišćenjem komponente *GraalVM Native Image* zahteva unapred definisanje svih resursa koji bi potencijalno mogli biti upotrebljeni tokom rada aplikacije. Navođenje resursa korisnik najčešće vrši putem *JSON* datoteka, koje sadrže spisak elemenata predviđenih za registraciju[2] unutar izvršive datoteke. Pored toga, kako svi resursi nisu uvek neophodni, korisnik može definisati uslov pod kojim se određeni resurs registruje. Ovi uslovi se zasnivaju na dostupnosti određenog tipa podataka (engl. *type reachable*) prilikom statičke analize. Ukoliko statička analiza prepozna da se određeni tip podataka koristi u programu, taj tip postaje dostupan, čime se aktivira uslov za registraciju odgovarajućeg resursa.

U ranijim verzijama komponente *GraalVM Native Image*, korisnik je resurse unutar JSON datoteka definisao primenom Java regularnih izraza. Radi potpunijeg razumevanja potencijalnih problema koje ovakav pristup nosi sa sobom, u nastavku će najpre biti opisan sam proces registracije resursa u okviru komponente *GraalVM Native Image*.

Proces je započinjao parsiranjem *JSON* datoteka i prikupljanjem svih definisanih regularnih izraza. Nakon toga, vršila se iteracija kroz sve resurse koji čine deo putanje projektnih klasa (engl. *classpath*), pri čemu se za svaki resurs proveravalo da li odgovara nekom od regularnih izraza[3]. S obzirom na to da su se resursi zahtevali

[2]Resurs se smatra registrovanim kada je putanja do njega ugrađena u izvršivu datoteku.
[3]Tačnije, da li postoji regularni izraz koji može da poklopi putanju do datog resursa.

pod određenim uslovima[4], za svaki regularni izraz koji odgovara određenom resursu registrovala bi se funkcija povratnog poziva (engl. *callback function*). Aktivacija, odnosno izvršavanje ovih funkcija, bila je uslovljena pronalaskom zahtevanog tipa podataka tokom statičke analize. U trenutku kada statička analiza identifikuje tip podataka i proglasi ga dostupnim (engl. *reachable*), sve funkcije povratnog poziva čiji je uslov ispunjen pokretale bi preostali deo procesa registracije resursa. Taj završni korak registracije podrazumevao je zapisivanje putanje resursa u centralizovani registar resursa (engl. *resources singleton*).

Opisani pristup pokazao se nepovoljnim iz nekoliko razloga. Pre svega, velika ekspresivnost regularnih izraza dovodila je do čestih grešaka prilikom definisanja skupa resursa, usled čega je izvršiva datoteka neretko obuhvatala znatno veći obim resursa od onog koji je korisnik primarno planirao. Pored toga, Java mehanizam za obradu regularnih izraza (engl. *Java Regex Engine*) predstavlja kompleksnu strukturu čiji memorijski otisak (engl. *memory footprint*) nije zanemarljiv. Iz tog razloga, njegovo uključivanje u krajnju izvršivu datoteku nije bilo poželjno, s obzirom na to da bi znatno povećalo memorijsko zauzeće. Sa druge strane, neuključivanje ovog mehanizma u izvršivu datoteku onemogućavalo je (ili u velikoj meri otežavalo) utvrđivanje u toku izvršavanja programa da li resurs nije registrovan zato što ga korisnik nije zahtevao ili zato što ne postoji na sistemu. Naposletku, sama činjenica da se za svaki dostupni resurs moralo iterirati kroz sve prikupljene regularne izraze ukazuje na neefikasnost ovakvog pristupa, budući da nije bilo moguće primeniti nikakve optimizacije u vidu odsecanja pretrage.

[4]Uslov podrazumeva identifikaciju određenog tipa podataka tokom statičke analize.

Glava 3

Šabloni za poklapanje teksta

Tokom razvoja aplikacija često se javlja potreba za identifikovanjem dela ili celokupnog teksta koji odgovara određenim pravilima, odnosno šablonima, unutar obimnih tekstualnih podataka. Radi lociranja teksta usaglašenog sa definisanim šablonom, mnogi programski jezici, a neretko i sami operativni sistemi, razvili su specifične formate koji primenjuju specijalne karaktere za opisivanje takvih obrazaca. Pored samih formata, razvijeni su i složeni mehanizmi obrade koji za zadati šablon i prosleđeni tekst utvrđuju postojanje međusobnog podudaranja. U ovom radu biće razmotrena dva takva formata: Java regularni izrazi i glob obrasci.

3.1 Java regularni izrazi

Regularni izrazi u programskom jeziku Java predstavljaju niz karaktera namenjen filtriranju i pretraživanju teksta [8]. Pored karaktera sa doslovnim značenjem (engl. *literals*), regularni izrazi definišu i niz specijalnih karaktera (engl. *wildcards*), čijim se kombinovanjem mogu formirati složeni obrasci za podudaranje teksta. U Tabeli 3.1 dat je prikaz osnovnih specijalnih karaktera.

Pored osnovnih specijalnih karaktera, regularni izrazi u Javi podržavaju napredne konstrukcije kao što su grupe, kvantifikatori, klase karaktera i graničnici, čime se omogućava precizno modelovanje strukture ulaznih podataka

U programskom jeziku Java, regularni izrazi se najčešće definišu u obliku niski karaktera, koje potom prolaze kroz niz faza obrade pre primene nad ulaznim tekstom. Prvi korak obuhvata parsiranje i prevođenje niske u objekat klase šablona (engl. *Pattern*), pri čemu se specijalni karakteri, kvantifikatori i grupe interpretiraju i prevode u internu strukturu podataka optimizovanu za efikasno pretraživanje.

Tabela 3.1: Specijalni karakteri u Java regularnim izrazima

Karakter	Značenje
`.`	Predstavlja bilo koji pojedinačni karakter, osim znaka za novi red (podrazumevano ponašanje).
`+`	Označava da se prethodni element[1] pojavljuje jedan ili više puta.
`*`	Označava da se prethodni element pojavljuje nula ili više puta.
`?`	Označava da se prethodni element pojavljuje nula ili jedan put, ili da kvantifikator bude lenj (engl. *lazy*).
`^`	Označava početak niske ili linije (u zavisnosti od režima rada).
`$`	Označava kraj niske ili linije.
`( )`	Definišu grupu i omogućavaju grupisanje podizraza i hvatanje podudaranja.
`[ ]`	Definišu skup karaktera od kojih se jedan može pojaviti na datoj poziciji.
`{ }`	Definišu tačan ili ograničen broj ponavljanja prethodnog elementa.
`|`	Predstavlja logičko „ili“, omogućavajući izbor između alternativnih izraza.
`\`	Koristi se za „escape“ specijalnih karaktera ili za definisanje posebnih sekvenci.

Nakon toga sledi kreiranje objekta klase poklapanja (engl. *Matcher*), koji povezuje prevedeni obrazac sa konkretnim nizom karaktera koji se analizira. Tokom izvršavanja, Java mehanizam za obradu regularnih izraza koristi algoritam zasnovan na nedeterminističkom konačnom automatu (engl. *Non-deterministic Finite Automaton - NFA*) sa povratnim pretraživanjem (engl. *backtracking*), pri čemu iterativno proverava svaku poziciju u tekstu i primenjuje pravila definisana izrazom. Karakteri sa doslovnim značenjem proveravaju se direktno, dok se specijalni karakteri i kvantifikatori obrađuju u skladu sa definisanim pravilima, pri čemu se grupe memorišu radi izdvajanja poklopljenih celina. Rezultat ovog procesa jeste informacija o uspe-

šnosti podudaranja, uz mogućnost pristupa pojedinačnim grupama koje odgovaraju zadatom obrascu. Ovakav način izvršavanja obezbeđuje visok nivo fleksibilnosti prilikom obrade tekstualnih podataka, ali istovremeno predstavlja i čest izvor problema u pogledu performansi ili neočekivanog ponašanja sistema.

3.2 Glob obrasci

Glob obrasci [3] predstavljaju jednostavan, ali efikasan mehanizam za definisanje obrazaca pretrage i filtriranja datoteka u sistemu. Njihova osnovna svrha jeste da omoguće korisnicima i aplikacijama izdvajanje skupa datoteka čiji nazivi odgovaraju zadatom obrascu. Pored karaktera sa doslovnim značenjem, glob obrasci primenjuju i specijalne karaktere radi efikasnijeg zapisivanja složenijih izraza. U tabeli 3.2 dat je prikaz osnovnih specijalnih karaktera. U svojoj osnovnoj formi, glob obrasci obuhvataju specijalne karaktere `*`, `?` i `[...]`, dok se karakter `**` smatra delom proširene sintakse. Ovaj mehanizam ima naročitu primenu u komandnim linijama, skript programiranju i automatizaciji, gde omogućava brzo i precizno pronalaženje i obradu putanja do fajlova kao i samog sadržaja fajlova.

Tabela 3.2: Specijalni karakteri u glob obrascima

Karakter	**Značenje**
`*`	Predstavlja bilo koji broj, bilo kojih karaktera, u imenu fajla ili direktorijuma.
`**`	Reprezentuje nula ili više direktorijuma u putanji (koristi se za rekurzivno pretraživanje).
`?`	Predstavlja tačno jedan bilo koji karakter.
`[abc]`	Podudara tačno jedan karakter od karaktera navedenih unutar zagrada.
`[a-z]`	Podudara tačno jedan karakter iz opsega karaktera navedenog unutar zagrada.

Putanje Java resursa

Kada je reč o glob obrascima, oni se najčešće primenjuju nad putanjama u sistemu datoteka (engl. *file system*). Međutim, isti mehanizmi poklapanja mogu se pri-

meniti i nad putanjama Java resursa (engl. *Java resource path*). Zapisi ovih putanja nezavisni su od operativnog sistema na kojem se program izvršava, što omogućava jednostavniji i fleksibilniji rad sa Java resursima. Za razliku od standardnih putanja do datoteka na sistemu datoteka, putanje do Java resursa obično se ne navode kao apsolutne, već relativno u odnosu na korene direktorijume projektnih klasa. U Java projektima najčešće postoje specijalizovani direktorijumi koji se nazivaju resursi (engl. *resources*) i koji se postavljaju kao koreni direktorijumi projektnih klasa. Samim tim, sve putanje do Java resursa mogu se posmatrati kao relativne u odnosu na resursni direktorijum. Ove putanje, navode se kao sekvenca direktorijuma razdvojena znakom „/" nezavisno od operativnog sistema na kojem se program izvršava. Direktorijum između dva znaka „/" označava se kao nivo putanje[2].

U nastavku kada budemo govorili o glob obrascima, podrazumevaćemo da je domen nad kojim oni operišu zapravo skup putanja do Java resursa.

[2]Poslednji nivo ne mora se završavati oznakom „/"

Glava 4

Prefiksna stabla

Prefiksno stablo [9] predstavlja strukturu podataka sa asocijativnim pristupom[1], koja se najčešće primenjuje u obradi prirodnog jezika i proveri pravopisa. Njen primarni cilj jeste efikasno pronalaženje svih ključeva sa zadatim prefiksom[2], pri čemu ključevi predstavljaju niske ili nizove. U stranoj literaturi ova struktura se još naziva i *Trie* (od engleske reči reTRIEval).

4.1 Struktura prefiksnog stabla

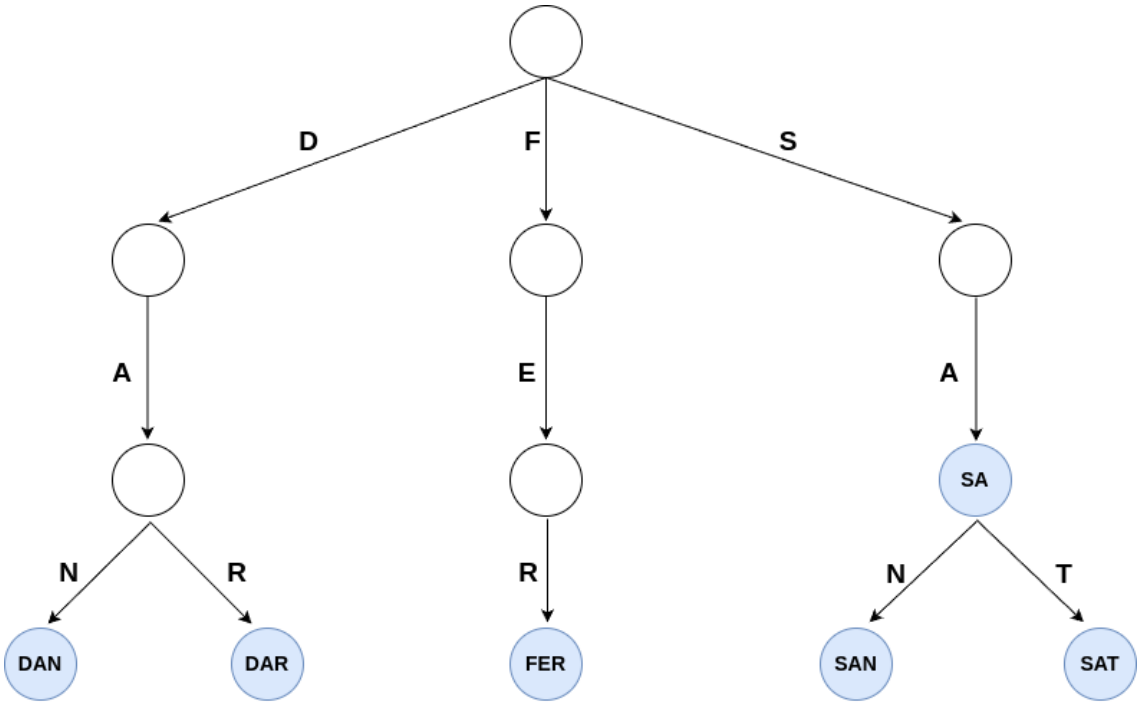
Struktura prefiksnog stabla zasniva se na deljenju zajedničkih prefiksa među ključevima. Za razliku od binarnih stabala pretrage, čvorovi u prefiksnim stablima ne skladište celokupan ključ koji predstavljaju, već se on formira nadovezivanjem karaktera sa grana duž putanje od korenog do posmatranog čvora. U situacijama kada na određenoj putanji nema račvanja, više uzastopnih grana može se spojiti u jednu radi uštede memorijskih resursa. Ovaj postupak naziva se kompresija grana prefiksnog stabla.

Prefiksno stablo predstavlja specijalizovanu vrstu n-arnog stabla pretrage. Radi jednostavnije implementacije, koreni čvor obično sadrži praznu reč. Svaki čvor u strukturi može imati proizvoljan broj izlaznih grana, pri čemu svaka grana mora biti jedinstvena. Ukoliko implementacija ne podrazumeva kompresiju grana, broj izlaznih grana po čvoru ne može biti veći od broja karaktera u upotrebljenoj azbuci. Čvorovi bez izlaznih grana nazivaju se listovi i oni označavaju kraj reči, odnosno

[1] Pristup elementima strukture vrši se pomoću ključa, a ne na osnovu rednog broja, odnosno indeksa.

[2] Otuda potiče i sam naziv prefiksno stablo.

ključa. U zavisnosti od implementacije, ovakvi čvorovi mogu (ali ne moraju) sadržati celokupnu reč radi lakše pretrage. Pored listova, u pojedinim implementacijama i unutrašnji čvorovi mogu označavati kraj reči. Čvorovi kojima se reč završava nazivaju se terminirajući čvorovi. Kako bi se terminirajući čvorovi razlikovali od ostalih, svaki čvor prefiksnog stabla sadrži informaciju o tome da li je terminirajući ili ne. Na Slici 4.1 prikazan je primer prefiksnog stabla čiji terminirajući čvorovi mogu biti kako listovi, tako i unutrašnji čvorovi.



Slika 4.1: Prefiksno stablo. Plavom bojom označeni su terminirajući čvorovi koji sadrže celu vrednost ključa.

4.2 Osnovne operacije nad stablom

Efikasnost ove strukture potiče iz činjenice da operacije zavise isključivo od dužine ključa, a ne od ukupnog broja elemenata koji su već uskladištenih u strukturi. Nad prefiksnim stablom jednostavano je definisati operacije umetanja i pretrage, dok je operacija brisanja nešto složenija.

Vremenska složenost osnovnih operacija nad prefiksnim stablom je $O(l)$ pri čemu l predstavlja dužinu reči, odnosno ključa. Navedena složenost odnosi se na prefiksna stabla u kojima struktura čvora čuva izlazne grane u obliku neuređene mape, kod

koje je pristup elementu složenosti $O(1)$ u prosečnom slučaju. U pojedinim implementacijama, izlazne grane se mogu čuvati i u obliku uređene mape, pri čemu se vremenska složenost povećava na $O(l \cdot \log(K))$, gde K predstavlja trenutnu veličinu mape. U svakom slučaju, prednost prefiksnog stabla leži u tome što vreme potrebno za izvršavanje operacija nad njim, zavisi isključivo od dužine reči koja se dodaje u stablo.

S druge strane, glavni nedostatak prefiksnog stabla ogleda se u njegovoj prostornoj složenosti. Naime, u najgorem slučaju ona iznosi $O(N \cdot M \cdot K)$ gde N predstavlja broj ključeva (reči) u stablu, M maksimalnu dužinu ključa, dok K predstavlja prostornu složenost čuvanja mape potomaka u svakom čvoru. Najgori slučaj javlja se kada nema nikakvog preklapanja među ključevima, pa je maksimalni mogući broj čvorova u stablu jednak $N \cdot M$, pri čemu svaki čvor ima maksimalan broj potomaka, što odgovara veličini azbuke koja se koristi (u ovom slučaju dužina azbuke označena je sa K). Međutim, jasno je da u realnim primenama, gotovo izvesno dolazi do preklapanja, te je prostorno zauzeće u praksi povoljnije.

Operacija pretraživanja

Algoritam pretraživanja reči u prefiksnom stablu iterira kroz nisku simbol po simbol i za svaki simbol pokušava da pronađe postojeću granu iz tekućeg čvora ka odgovarajućem čvoru detetu. Ukoliko čvor nema izlaznu granu koja odgovara simbolu koji se trenutno obrađuje, pretraga je neuspešna. Takođe, ukoliko je dostignut kraj niske, a trenutno posećeni čvor nije označen kao terminirajući, pretraga je takođe neuspešna. U suprotnom, ako se stiglo do kraja niske i tekući čvor ima oznaku terminirajućeg čvora, pretraga je uspešna. Operacija pretraživanja prikazana je u algoritmu 1.

Algorithm 1 Algoritam pretraživanja reči u prefiksnom stablu

Require: Koren prefiksnog stabla *koren*, niska *reč* koja se pretražuje
Ensure: Istinitosna vrednost (**true** ako reč postoji u stablu, **false** u suprotnom)

```
1: function PRETRAŽI(koren, reč)
2:     tekući ← koren
3:     for all simbol ∈ reč do
4:         if not POSTOJIGRANA(tekući, simbol) then
5:             return false          ▷ Pretraga je neuspešna - nema grane za simbol
6:         tekući ← DOHVATIDETE(tekući, simbol)
7:     if not tekući.jeTerminirajući then
8:         return false              ▷ Dostignut kraj niske, ali čvor nije terminirajući
9:     return true                   ▷ Dostignut kraj niske i čvor je terminirajući
```

Operacija umetanja

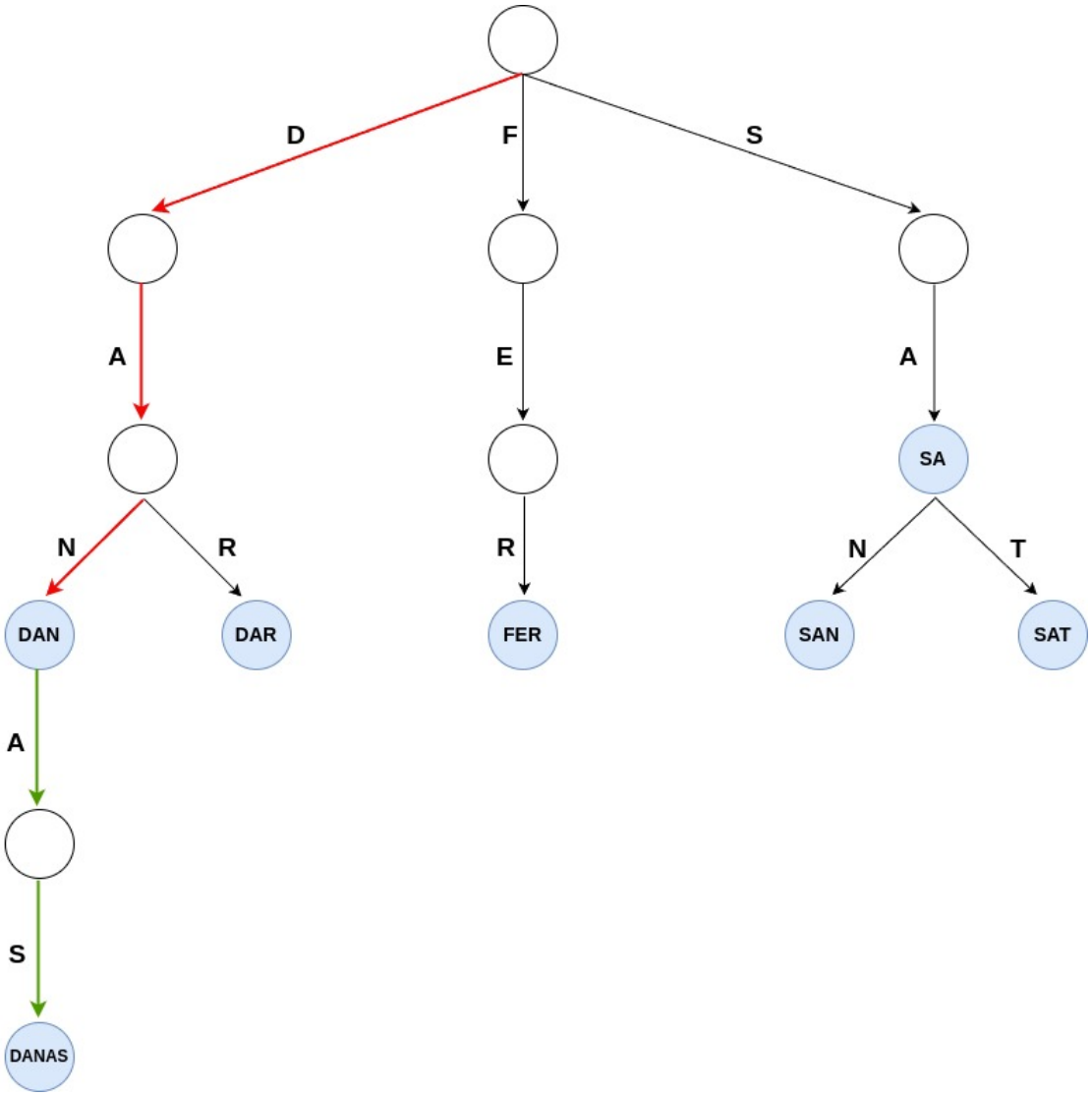
Operacija umetanja podrazumeva dodavanje nove reči (ključa) u postojeće stablo. Prilikom umetanja, algoritam iterira kroz simbole niske, krećući se od korena naniže duž grana koje već postoje u stablu. Ukoliko grana za tekući simbol ne postoji, kreira se novi čvor. Naposletku, kada se dođe do kraja niske, poslednji posećeni čvor postavlja se kao terminirajući, pri čemu se u njega opciono upisuje celokupna vrednost ključa. Operacija umetanja prikazana je u algoritmu 2.

Na Slici 4.2 prikazan je primer dodavanja ključa „DANAS" u prethodno prikazano stablo. Može se primetiti da čvor koji je bio terminirajući za ključ „DAN" nakon umetanja prestaje da bude list, ali zadržava svojstvo terminirajućeg čvora.

Algorithm 2 Algoritam umetanja reči u prefiksno stablo

Require: Koren prefiksnog stabla *koren*, niska *reč* koja se umeće
Ensure: Modifikovano prefiksno stablo koje sadrži novu reč

```
1: function DODAJREČ(koren, reč)
2:     tekući ← koren
3:     for all simbol ∈ reč do
4:         if not POSTOJIGRANA(tekući, simbol) then
5:             noviČvor ← KREIRAJNOVIČVOR
6:             DODAJGRANU(tekući, simbol, noviČvor)
7:         tekući ← DOHVATIDETE(tekući, simbol)
8:     tekući.terminirajući ← true
9:     tekući.vrednostKljuča ← reč        ▷ Opciono upisivanje celokupnog ključa
```



Slika 4.2: Primer dodavanja ključa „DANAS" u postojeće prefiksno stablo. Algoritam prolazi kroz simbole ključa pri čemu se najpre spušta postojećim granama (označenim crvenom bojom) a potom kreira nove grane (označene zelenom bojom) kada u datom stablu iz tekućeg čvora ne postoji grana koja vodi do narednog simbola u nisci.

Operacija brisanja

Operacija brisanja nešto je složenija jer se pri uklanjanju jedne reči mora voditi računa da čvorovi koji sačinjavaju tu reč, istovremeno ne učestvuju u izgradnji neke druge reči. Kako bi se održao integritet stabla, dozvoljeno je brisati samo čvorove koji zadovoljavaju sledeće uslove:

1. Čvor nema dece.

2. Čvor nije označen kao terminirajući.

3. Do čvora smo se spustili granama koje odgovaraju karakterima reči koju brišemo.

Na osnovu gore navedenog opisa, algoritam 3 implementira rekurzivnu funkciju za brisanje reči iz stabla.

Algorithm 3 Algoritam brisanja ključa iz prefiksnog stabla

Require: Tekući čvor *čvor*, niska koja se briše *reč*
Ensure: Ažurirana struktura prefiksnog stabla nakon uklanjanja reči *reč*

```
1:  function OBRIŠI(čvor, reč)
2:      if reč = "" then
3:          čvor.terminirajući ← false          ▷ Tekući čvor više nije terminirajući
4:      else
5:          karakter ← PRVIKARAKTER(reč)
6:          if SADRŽIGRANU(čvor, karakter) then
7:              sledeći ← DOHVATIDETE(čvor, karakter)
8:              ostatak ← UKLONIPRVIKARAKTER(reč)
9:              OBRIŠI(sledeći, ostatak)
10:     if not IMADECE(čvor) and not čvor.jeTerminirajući then
11:         UKLONIČVOR(čvor)
```

Glava 5

Kompresovano glob-prefiksno stablo

Da bi se prevazišli problemi prevelike ekspresivnosti kao i memorijska i vremenska neefikasnost regularnih izraza (detaljno opisani u sekciji 2.4), Java regularni izrazi mogu se zameniti glob obrascima, dok se za njihovo efikasno pretraživanje može koristiti struktura podataka u čijoj se osnovi nalazi prefiksno stablo. Ideja je da se svi prikupljeni glob obrasci dodaju u prefiksno stablo, čime bi se omogućilo odsecanje pri proveri da li je neka putanja do resursa obuhvaćena nekim glob obrascem. Međutim, s obzirom na to da glob obrasci mogu koristiti i specijalne karaktere, standardno prefiksno stablo nije u stanju da proveri da li postoji glob obrazac koji obuhvata datu putanju do Java resursa. Stoga je neophodno proširiti logiku prefiksnog stabla tako da ona obuhvati semantiku glob obrazaca. Tako dobijenu strukturu nazvaćemo *glob-prefiksno stablo* (eng. *glob trie*).

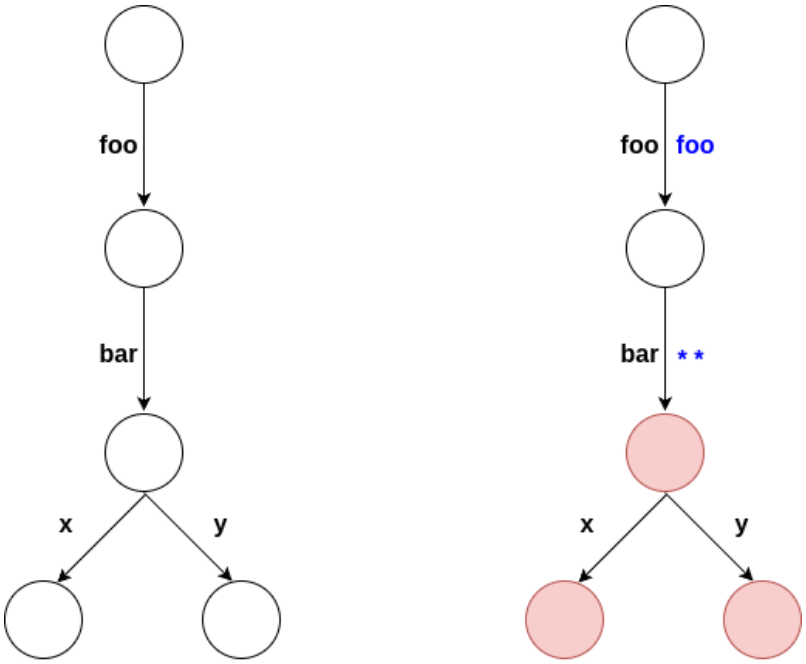
Pored toga, lako se može primetiti da dva glob obrasca mogu imati zajednički prostor poklapanja. Jedan glob obrazac može obuhvatati nadskup skupa putanja nekog drugog glob obrasca. U tom slučaju kažemo da je prvi glob obrazac opštiji od drugog. Kada je reč o smeštanju ovakvih obrazaca u glob-prefiksno stablo, jasno je da je u stablu neophodan samo opštiji glob obrazac, jer on obuhvata kako svoje putanje, tako i putanje koje bi pokrio specifičniji glob obrazac. Ova činjenica omogućava kompresiju glob-prefiksnog stabla bez gubitka informacija o putanjama koje je moguće poklopiti. Uvođenjem kompresije, glob-prefiksno stablo postaje *kompresovano glob-prefiksno stablo* (eng. *Compressed Glob Trie*, skraćeno CGT).

Preduslov da bi kompresija uopšte bila moguća jeste da svi glob obrasci budu poznati unapred kako bi se mogli sortirati[1]. Činjenica da svi obrasci moraju biti poznati unapred direktno ukazuje na to da će kompresovano stablo biti nepromenljivo

[1]Objašnjenje zašto je obrasce neophodno sortirati nalazi se u sekciji 5.3

(imutabilno).

Slika 5.1 ilustruje kako dodavanje obrazaca u nesortiranom redosledu može prouzrokvati brisanja podstabala kada opštiji obrazac treba da „istisne" specifičnije obrasce čiji je on nadskup. Obzirom da operacija brisanja može biti skupa, jer može zahvatati veliki deo stabla, treba težiti što ređoj njenoj upotrebi. Da je redosled bio obratan, i da je opštiji obrazac prvi upisan u stablo, ostali obrasci bi samo bili odbačeni, jer je trenutni obrazac u stablu već dovoljan da pokrije ono što bi ti specifičniji obrasci pokrili.



Slika 5.1: Dodavanje obrazaca u proizvoljnom redosledu. Najpre se dodaju obrasci $foo/bar/x/$ i $foo/bar/y/$ (njihov redosled dodavanja nije bitan), a zatim se dodaje obrazac $foo/**/$. Plavom bojom označen je novi obrazac koji treba dodati. Crvenom bojom označeni su čvorovi koje je potrebno obrisati pre dodavanja obrasca $foo/**/$.

Na posletku, treba napomenuti da iako u osnovnu specifikaciju glob obrazaca spadaju i specijalni karakteri poput „?" i „`[...]`", podrška za njih se ne nalazi u implementaciji koju prati ovaj rad. Ovakva odluka doneta je kako bi se dodatno smanjila izražajna moć i prostor za korisničke greške.

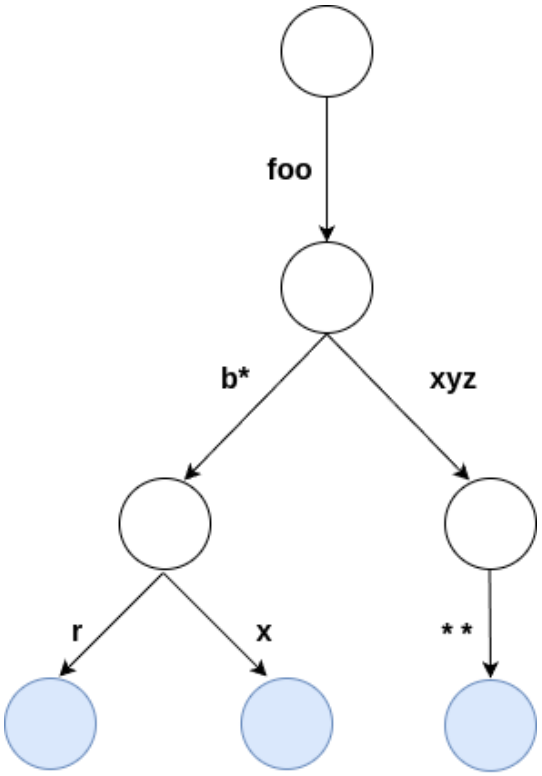
5.1 Tokenizacija glob obrazaca

Tokenizacija je proces raščlanjivanja kontinuiranog niza podataka (najčešće teksta ili koda) na manje, smislene jedinice koje se nazivaju tokeni. Za razliku od klasičnih prefiksnih stabala, kod kojih se tokenizacija ključa vrši na nivou pojedinačnog simbola (tj. literala), za glob obrasce potrebno je primeniti nešto složeniju tokenizaciju, pri čemu se kao krajnji rezultat dobijaju tri vrste tokena: token dvostruke zvezde, token jednostruke zvezde (opciono sa pridruženim prefiksom) i token literala.

Da bi se izvršila tokenizacija obrasca, on se najpre deli na nivoe, pri čemu nivo predstavlja deo putanje razdvojen karakterom kose crte (eng. *slash*). Potom se svaki nivo dodatno tokenizuje u jedan od sledećih oblika:

- dvostruka zvezda (primer: */**/* se tokenizuje u token dvostruke zvezde ****)
- samostalna jednostruka zvezda (primer: */*/* se tokenizuje u token jednostruke zvezde ***)
- proizvoljan broj literala praćen jednostrukom zvezdom (primer: */ab*cd*e/* se tokenizuje u tri tokena i to u tokene jednostruke zvezde *ab** i *cd**, i token literala *e*)
- literali (primer: */abc/* se tokenizuje u literalni token *abc*)

Tokeni dobijeni tokenizacijom nivoa kasnije predstavljaju čvorove u kompresovanom glob-prefiksnom stablu. Na slici 5.2 prikazan je primer formiranog glob-prefiksnog stabla. Procesu formiranja prethodi tokenizacija obrazaca, nakon koje svaki token predstavlja po jedan čvor. Svaki token biće predstavljen odgovarajućim čvorom u stablu.



Slika 5.2: Stablo formirano naivnim algoritmom koji koristi prosto prefiksno stablo. Stablo se formira od sledećih obrazaca nakon njihove tokenizacije: $foo/b*r/$ (tokenizuje se na: foo, $b*$, r), $foo/b*x/$ (tokenizuje se na: foo, $b*$, x), $foo/xyz/**/$ (tokenizuje se na: foo, xyz, $**$).

5.2 Validacija glob obrazaca

Da bi kompresovano glob-prefiksno stablo uvek davalo korektne rezultate, neophodno je najpre ukloniti sve nevalidne glob obrasce. Glob obrazac se smatra nevalidnim ako:

- je definisan kao prazna niska
- sadrži prazan nivo
- ima više od dva uzastopna znaka ” $*$ ”
- počinje dvostrukom zvezdom[2]

[2]Ovo zapravo nije greška, ali je prilično neefikasno jer u startu obuhvata veliki broj proizvoljnih direktorijuma.

5.3 Sortiranje glob obrazaca

Verovatno najkritičniji deo konstrukcije kompresovanog glob-prefiksnog stabla jeste pitanje kako na efikasan način postići kompresiju stabla. Dodavanje glob obrazaca u nasumičnom redosledu nije dobar izbor jer može zahtevati veliki broj operacija brisanja (videti sliku 5.1), koje su u realnim situacijama sporije od ostalih operacija nad stablom. Stoga je bolji pristup da najpre sve glob obrasce sortiramo po opštosti, polazeći od najopštijeg obrasca ka specifičnijim, i da ih u tom redosledu dodajemo u strukturu.

Samo definisanje toga kada je jedan glob obrazac opštiji od drugog (tj. poređenje dva glob obrasca) nije trivijalan problem. Međutim, ovaj problem se može raščlaniti na jednostavnije podprobleme:

- Krenimo najpre od dva glob obrasca koji nemaju nijedan specijalni karakter. Njihovo poređenje nije od interesa jer prefiksno stablo u svojoj osnovnoj implementaciji jednostavno rešava dodavanje proizvoljnog broja ovakvih glob obrazaca u proizvoljnom redosledu.

- Zatim, ako imamo jedan glob obrazac koji sadrži specijalni karakter ”$*$” (ili ”$**$”) na bilo kom svom nivou, a drugi glob obrazac ne sadrži specijalne karaktere, opštiji je obrazac koji sadrži specijalne karaktere. Naime, ukoliko ta dva obrasca nemaju zajednički prefiks do pojave karaktera ”$*$”, oni će završtiti u odvojenim delovima prefiksnog stabla, te pretpostavka o opštosti postaje irelevantna. S druge strane, ako oba glob obrasca pripadaju istom podstablu prefiksnog drveta, i na istom nivou se razlikuju samo po tome što jedan ima specijalni karakter ”$*$”, a drugi ne, prvo treba dodati obrazac sa specijalnim karakterom, jer je on potencijalno nadskup drugog (prostijeg) obrasca.

- Isto razmišljanje važi ukoliko poredimo jedan obrazac koji sadrži specijalni karakter ”$**$” i jedan obrazac koji sadrži specijalni karakter ”$*$”. Ako im se prefiksi razlikuju, nije bitno koji obrazac je opštiji od kojeg jer će završtiti u različitim podstablima. S druge strane, ako im je prefiks isti, i na istom nivou jedan obrazac sadrži ”$**$” a drugi ”$*$”[3], opštiji je obrazac sa specijalnim karakterom ”$**$”, jer ”$**$” može da poklopi sve što može da poklopi ”$*$” i potencijalno još neke nivoe.

[3]Ovaj karakter najčešće stoji uz neki literal

Iz prethodno navedenog može se uočiti da uvek najpre treba dodati obrasce koji sadrže barem jedan specijalni karakter dvostruke zvezde[4] (" ∗ ∗"), zatim obrasce koji ne sadrže dvostruke zvezde ali sadrže jednostruke zvezde (" ∗ "), a tek onda obrasce koji ne sadrže nikakve specijalne karaktere. Stoga se obrasci pre samog sortiranja najpre razvrstavaju u tri grupe: obrasci sa dvostrukom zvezdom, obrasci sa jednostrukom zvezdom i obrasci bez specijalnih karaktera.

Nakon što se obrasci razvrstaju u grupe, neophodno je sortirati elemente unutar svake od njih. Proces sortiranja zasniva se na poređenju dva obrasca. Dva obrasca iz iste grupe porede se na sledeći način:

1. Obrasce podelimo na nivoe (nivoe dobijamo deljenjem putanje po karakteru "/").

2. Iteriramo nivo po nivo paralelno u oba obrasca koja poredimo.

3. Proveravamo da li nivoi sadrže dvostruku zvezdu.

   a) Ukoliko jedan obrazac na tekućem nivou sadrži dvostruku zvezdu a drugi ne, taj obrazac proglašavamo za opštiji.

   b) Ukoliko oba obrasca na tekućem nivou imaju karakter dvostruke zvezde prelazi se na sledeći nivo.

   c) Ukoliko ni jedan ni drugi obrazac na tom nivou ne sadrže dvostruku zvezdu, nastavlja se dalje sa proverama istog nivoa.

4. Proveravamo da li nivoi sadrže jednostruku zvezdu.

   a) Ako jedan obrazac na tekućem nivou ima jedonstruku zvezdu a drugi je nema, taj obrazac proglašavamo za opštiji.

   b) Ako oba obrasca imaju na istom nivou jednostruku zvezdu, opštiji je onaj koji ima manje *„ne zvezda"* karaktera. Naime ako posmatrani nivoi ne predstavljaju istu reč, nije nam bitno da li smo napravili pogrešnu pretpostavku, jer će obrasci pripadati različitim podstablima u strukturi. Zatim, ako tekući nivoi posmatranih obrazaca, predstavljaju istu reč, obrazac koji sadrži više literala može da opiše manje različitih varijacija

[4]U grupi obrazaca koji sadrže karakter dvostruke zvezde, mogu se naći i obrasci koji, pored dvostruke zvezde, u delu svoje putanje sadrže i jednostruku zvezdu. Jedini bitan uslov da bi obrazac pripadao ovoj grupi je da sadrži karakter dvostruke zvezde.

reči jer literali sužavaju prostor pretrage dok ga specijalni karakteri proširuju. Na posletku, ako imamo i isti broj literala to znači da: ili imamo iste reči sa različitim brojem zvezda (dajemo prednost obrascu sa većim brojem zvezda jer je on opštiji), ili imamo iste reči samo sa zvezdama na različitim mestima, pa je nebitno koji je opštiji jer će opet pripadati različitim podstablima.

5. Prednost dajemo kraćim obrascima[5] jer se oni završavaju ranije u hijerarhiji direktorijuma.

Sam algoritam sortiranja sastoji se iz raspoređivanja obrazaca u grupe (opisano algoritmom 4) kao i sortiranja svake od grupa (opisano algoritmom 5).

5.4 Konstrukcija stabla

Konstrukcija kompresovanog glob-prefiksnog stabla, u osnovi podseća na konstrukciju klasičnog prefiksnog stabla[6]. Međutim, zbog specifičnosti glob obrazaca, neophodno je uvesti određene modifikacije.

Prva bitna razlika ogleda se u tokenizaciji ključeva. Kod klasičnih prefiksnih stabala reč se najčešće deli po karakterima, dok se za tokenizaciju glob obrazaca koristi tehnika opisana u sekciji 5.1. Stoga se novi obrazac dodaje u stablo token po token. Razlog za ovakvu odluku leži u činjenici da kod klasičnog prefiksnog stabla svako slovo nosi dovoljnu informaciju za nastavak pretrage, dok je u slučaju glob obrazaca nosilac informacije jedan nivo putanje. Takođe, radi lakše implementacije, specijalni karakter „zvezda" vezuje se za svoj prefiks (sa tog nivoa), dok je njegov sufiks predstavljen u vidu njegovih potomaka u stablu.

Sledeća bitna razlika ogleda se u tome što se nekada sve do samog kraja novog obrasca koji se dodaje ne zna da li ga može poklopiti neki postojeći ključ iz stabla. Kao primer može se uzeti situacija u kojoj se u stablu nalazi obrazac $foo/bar/a*c/x/y/z/fg$, i treba dodati novi obrazac $foo/bar/abc/x/y/z/f*$. Novi obrazac bi se poklapao sa postojećim obrascem sve do poslednjeg nivoa, ali bi potom poklapanje bilo neuspešno jer fg ne može poklopiti $f*$[7]. U tom slučaju potrebno je vratiti se unazad i pronaći prvo mesto na kome se ova dva obrasca razilaze.

[5] Onima sa manjim brojem nivoa

[6] U slučaju kada glob obrazac ne sadrži specijalne karaktere implementacija je identična implementaciji prefiksnog stabla do na tokeniuaciju koja se ne vrši po karakterima, već po nivoima.

[7] fg je specifičniji (uži) od $f*$

Algorithm 4 Algoritam raspoređivanja i sortiranja glob obrazaca po grupama

Require: Lista obrazaca O
Ensure: Uređena lista obrazaca sastavljena iz tri sortirane podgrupe

```
1: function Sortiraj(O)
2:     O_nevalidni ← [ ]
3:     O_** ← [ ]
4:     O_* ← [ ]
5:     O_literalni ← [ ]
6:     for all o ∈ O do
7:         Rasporedi(o, O_nevalidni, O_**, O_*, O_literalni)
8:     if O_nevalidni ≠ ∅ then
9:                                          ▷ Obrada u slučaju nevalidnih obrazaca
10:    SortirajGrupu(O_**)
11:    SortirajGrupu(O_*)
12:    SortirajGrupu(O_literalni)
13:    return O_** ∘ O_* ∘ O_literalni
14: function Rasporedi(o, O_nevalidni, O_**, O_*, O_literalni)
15:    if not ValidanObrazac(o) then
16:        O_nevalidni ← O_nevalidni ∘ [o]
17:        return
18:    if o sadrži "**" then
19:        O_** ← O_** ∘ [o]
20:        return
21:    if o sadrži "*" then
22:        O_* ← O_* ∘ [o]
23:        return
24:    O_literalni ← O_literalni ∘ [o]
```

Iz tog razloga, postupak dodavanja novog obrasca pogodnije je podeliti u dve faze. Prva faza podrazumeva proveru da li novi obrazac može biti poklopljen nekim ključem koji se trenutno nalazi u stablu. Ako je to slučaj, dodavanje novog obrasca se preskače. U suprotnom (druga faza), pošto je izvesno da novi obrazac mora biti dodat, primenjuje se identičko poklapanje, bez razmatranja značenja specijalnih karaktera. Ono podrazumeva da je spuštanje niz granu stabla moguće isključivo ako je ona u potpunosti identična trenutnom tokenu novog obrasca koji se dodaje. Kada se dostigne čvor iz kog nije moguće dalje spuštanje identičkim poklapanjem, rekurzivno se kreira čvor (kao i grana od tekućeg čvora ka njemu) za svaki token preostao do kraja novog obrasca. Nakon kreiranja poslednjeg čvora, indikator da je čvor terminirajući postavlja se na istinitosnu vrednost tačno, čime se proces dodavanja novog

Algorithm 5 Algoritam sortiranja grupe glob obrazaca

Require: Nesortirana lista obrazaca obrazaca O_{grupa}
Ensure: Sortirana lista obrazaca obrazaca O_{grupa}

1: **function** SortirajGrupu(O_{grupa})
2: Sortiraj elemente liste O_{grupa} koristeći PorediObrasce kao funkciju poređenja
3: **function** PorediObrasce(o_1, o_2)
4: $N_1 \leftarrow o_1$.split("/")
5: $N_2 \leftarrow o_2$.split("/")
6: $i \leftarrow 0$
7: $j \leftarrow 0$
8: **while true do**
9: $n_1 \leftarrow N_1[i]$
10: $n_2 \leftarrow N_2[j]$
11: **if** n_1 sadrži "**" **and** n_2 sadrži "**" **then**
12: $i \leftarrow i + 1$
13: $j \leftarrow j + 1$
14: **continue**
15: **if** n_1 sadrži "**" **and not** n_2 sadrži "**" **then**
16: **return** o_1 ima prednost
17: **else if** n_2 sadrži "**" **and not** n_1 sadrži "**" **then**
18: **return** o_2 ima prednost
19: **if** n_1 sadrži "*" **and not** n_2 sadrži "*" **then**
20: **return** o_1 ima prednost
21: **else if** n_2 sadrži "*" **and not** n_1 sadrži "*" **then**
22: **return** o_2 ima prednost
23: **if** n_1 sadrži "*" **and** n_2 sadrži "*" **then**
24: **return** onaj obrazac koji ima manje ne-zvezda karaktera na ovom nivou
25: **if** $n_1 = n_2$ **then**
26: $i \leftarrow i + 1$
27: $j \leftarrow j + 1$
28: **continue**
29: ▷ Ako su nivoi različiti, prioritet kraćem obrascu u pogledu nivoa
30: **if** length(N_1) < length(N_2) **then**
31: **return** o_1 ima prednost
32: **else**
33: **return** o_2 ima prednost

obrasca završava.

5.5 Pretraga u kompresovanom glob-prefiksnom stablu

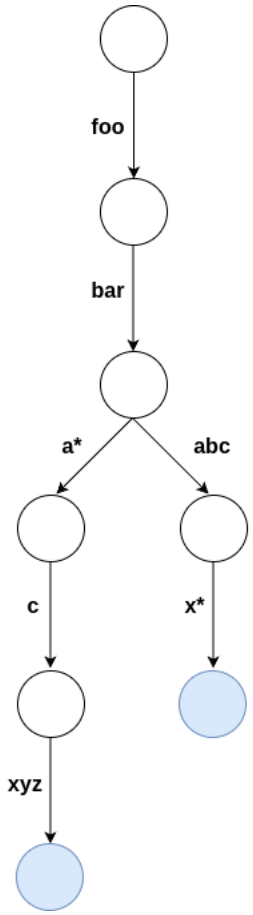
Kada u kompresovanom glob-prefiksnom stablu ne bi bilo specijalnih karaktera, pretraga bi se vršila kao i u klasičnom prefiksnom stablu, spuštanjem niz jedinstvene grane sve dok je to moguće. Međutim, zbog prirode specijalnih karaktera u glob obrascima, ovakav pristup nije uvek primenjiv. Štaviše, uvođenjem specijalnih karaktera gubi se jedinstvenost rešenja, jer istu putanju do resursa može poklopiti više obrazaca. Na slici 5.3 prikazan je primer stabla koje sadrži dva obrasca koja mogu poklopiti istu putanju. U zavisnosti od potreba, moguće je implementirati dve vrste pretrage:

- pretraga koja samo daje odgovor da li postoji obrazac koji poklapa zadatu putanju,
- pretraga koja vraća sve obrasce koji mogu poklopiti zadatu putanju.

Takođe, iteriranje po granama koje sadrže specijalne karaktere zahteva posebnu obradu, jer se u tom slučaju narušava odnos paralelnog prolaska kroz stablo i pretraživane putanje. Naime, moguće je preskočiti više nivoa putanje uz prelazak preko samo jedne grane stabla (prilikom prelaska preko grane koja sadrži dvostruku zvezdu). Pored toga, moguća je i obrnuta situacija, u kojoj se prelazi preko više grana stabla, a preskače se samo jedan nivo putanje (prilikom prelaska preko ulančanih čvorova sa jednostrukom zvezdom).

Radi bolje čitljivosti implementacije, postupak pretrage može se podeliti u tri koraka:

1. spuštanje niz granu sa dvostrukom zvezdom (pretraga uz preskakanje nivoa)
2. spuštanje niz granu koja sadrži jednostruku zvezdu (pretraga jednog nivoa)
3. spuštanje niz grane koje sadrže isključivo literale (identička pretraga).



Slika 5.3: Stablo u kojem dva obrasca $foo/bar/a*c/xyz$ i $foo/bar/abc/x*$ mogu da poklope putanju $foo/bar/abc/xyz$. Primetimo da oba obrasca moraju biti dodata u stablo jer nijedan nije opštiji od drugog.

Pretraga nivoa sa dvostrukom zvezdom

S obzirom na to da je najefikasniji način poklapanja putanje onaj u kojem se istovremeno poklopi veći broj nivoa, prilikom izbora grane za spuštanje niz stablo uvek se najpre bira grana koja sadrži dvostruku zvezdu. Zbog svog specifičnog značenja, ova grana zahteva drugačiju obradu u odnosu na standardni način pretrage u prefiksnom stablu. Poklapanje putanje sa ovom granom vrši se po principu opisanom u nastavku.

Ukoliko grana sa dvostrukom zvezdom vodi do terminirajućeg čvora, ona može u potpunosti poklopiti preostali deo putanje. U tom slučaju vraća se taj terminirajući čvor, uz potvrdu da je pretraga uspešno završena.

Ukoliko grana sa dvostrukom zvezdom vodi do čvora sa više potomaka, najpre se pokušava preskakanje trenutnog nivoa putanje, kako bi se naredni nivo poklopio

sa nekim od potomaka tekućeg čvora. Ukoliko nijedan potomak ne uspe da poklopi naredni nivo putanje, postupak se ponavlja preskakanjem i tog nivoa[8], a njegov sledbenik pokušava se poklopiti sa istim potomcima. Postupak se nastavlja sve dok se n nivoa putanje ne poklopi granom dvostruke zvezde, a $(n+1)$-i nivo ne poklopi nekim potomkom čvora do kog vodi grana dvostruke zvezde. Takođe, postupak se prekida ukoliko se dostigne kraj putanje bez uspešnog poklapanja.

Potrebno je napomenuti da potomci čvora do kog vodi grana sa dvostrukom zvezdom ne moraju biti literali. Implementacija omogućava da potomci čvora sa dvostrukom zvezdom budu i čvorovi sa jednostrukom zvezdom[9]. U zavisnosti od toga da li je potomak prost (literal) ili složen (sa jednostrukom zvezdom), primenjuju se različite procedure koje su opisane u nastavku.

Pretraga nivoa sa jednostrukom zvezdom

Spuštanje niz granu koja sadrži jednostruku zvezdu predstavlja verovatno najkompleksniji deo postupka poklapanja putanje sa čvorovima kompresovanog glob-prefiksnog stabla. Pre nego što bude izložen algoritam poklapanja putanje sa čvorom jednostruke zvezde, treba uzeti u obzir da tekući nivo putanje ne mora nužno biti poklopljen samo jednim obrascem, zbog čega algoritam vraća niz čvorova koji mogu nastaviti poklapanje. Ovaj slučaj prikazan je u primeru 1.

Primer 1. *Nivoi predstavljeni obrascima "$a*c$" i "$a*b*c$" mogu poklopiti nivo putanje "adbdc". U stablu ova dva nivoa imaju zajednički čvor $a*$, nakon čega sledi grananje na čvor c i čvor $b*$ (čiji je potomak takođe čvor c). Na ovom primeru može se uočiti da pri poklapanju nivoa putanje adbdc iz zajedničkog čvora $a*$ postoji više validnih putanja za spuštanje niz stablo.*

Algoritam poklapanja putanje sa čvorom sa jednostrukom zvezdom funkcioniše po principu opisanom u nastavku.

Najjednostavniji slučaj pri pretrazi nivoa sa zvezdom nastupa kada čvor koji se obrađuje sadrži samo karakter zvezde. Takav čvor nema prefiks i nosi informaciju da predstavlja kraj kompleksnog čvora koji sadrži jednostruku zvezdu. Ove dve činjenice ukazuju na to da čvor poklapa celokupan nivo, pa procedura vraća taj isti čvor i informaciju da je poklapanje bilo uspešno.

[8] Taj nivo biva poklopljen dvostrukom zvezdom.

[9] Radi sažetijeg izražavanja, izrazi oblika „čvor do kojeg vodi grana jednostruke zvezde" zamenjivaće se sa „čvor sa jednostrukom zvezdom".

Ukoliko čvor koji se obrađuje nije u mogućnosti da poklopi ceo nivo (sadrži prefiks ili sufiks), poklapanje se odvija u dve faze: poklapanje prefiksa i poklapanje ostatka. Prisetimo se da se sufiksi predstavljaju kao potomci tekućeg čvora. Takođe, s obzriom na to da na jednom nivou može postojati više karaktera jednostruke zvezde, ovakav nivo biće preslikan u nekoliko uzastopnih čvorova u stablu, gde svaki čvor sadrži neki prefiks i karakter jednostruke zvezde. Ovakvi kompleksni nivoi mogu se završavati literalom ili takođe čvorom do kojeg vodi grana koja sadrži jednostruku zvezdu. Radi lakše implementacije, svaki čvor nosi informaciju da li započinje naredni nivo, čime se lako može proveriti da li neki potomak pripada istom nivou ili započinje novi nivo obrasca.

Dakle, prva faza podrazumeva poklapanje prefiksa i obavlja se na sledeći način. Najpre se provera se da li tekući nivo putanje sadrži isti prefiks kao i tekuća grana koju obrađujemo.

1. Ukoliko to nije slučaj, tekući čvor koji sadrži zvezdu ne može poklopiti tekući nivo putanje koji se obrađuje.

2. U suprotnom, ukoliko tekući čvor i tekući nivo imaju isti prefiks, iz nivoa putanje uklanja se prefiks zajednički sa tekućim čvorom.

Nakon uklanjanja prefiksa, dobija se ostatak koji je potrebno poklopiti u drugoj fazi. Pre početka druge faze proverava se da li je tekući čvor sadržao potomke sa istog nivoa[10]. Ukoliko čvor ne sadrži potomke na istom nivou obrasca, ostatak nivoa se može poklopiti karakter jednostruke zvezde kojim se čvor završavao. S druge strane, ukoliko tekući čvor nije krajnji na tekućem nivou, odnosno sadrži potomke koji se odnose na isti nivo obrasca, pretraga se nastavlja. Najpre se pokušava poklapanje ostatka tekućeg nivoa putanje sa nekim literalnim potomkom tekućeg čvora. Ako je to moguće, taj potomak se dodaje u listu[11] potomaka koji mogu nastaviti pretragu. Nezavisno od rezultata poklapanja sa literalnim potomkom, algoritam nastavlja sa obradom ostalih čvorova koji pripadaju istom nivou obrasca. Preostali čvorovi na tekućem nivou obrasca moraju sadržati jednostruku zvezdu, pa se prolazi kroz listu takvih čvorova i spušta se isključivo kroz čvorove čiji se prefiks poklapa sa prefiksom nivoa putanje koji treba obraditi. Dodatno, treba napomenuti da postoji mogućnost da se prefiks čvora sa zvezdom ponavlja više puta na tekućem nivou, pa nije dovoljno

[10]Potomci koji ne započinju novi nivo.

[11]Jer može postojati više rešenja ako imamo i potomke sa jednostrukom zvezdom koji mogu nastaviti poklapanje.

uzeti samo prvo ili poslednje pojavljivanje prefiksa, već se mora uzeti u obzir svaki potencijalni slučaj. Takav slučaj objašnjen je u Primeru 2.

Primer 2. *Ukoliko treba poklopiti putanju "aaaa", pri čemu tekući čvor ima oblik a∗ i dete oblika a∗, moraju se ispitati slučajevi kada:*

- *tekući čvor poklapa prvo a kao prefiks, a njegov karakter zvezda ne poklapa ni jedno slovo "a", dok potomak poklapa ostatak*
- *tekući čvor poklapa prvo a kao prefiks, a njegov karakter zvezda poklapa drugo slovo "a", a potomak poklapa ostatak*
- *tekući čvor poklapa prvo a kao prefiks, a njegov karakter zvezda poklapa drugo i treće slovo "a", a potomak poklapa ostatak*

Identička pretraga

Identička pretraga predstavlja najjednostavniji vid pretrage u stablu. Nakon obrade specijalnih karaktera, čak i u slučaju prethodnog prolaska niz grane sa specijalnim karakterima, potrebno je obraditi i potencijalni slučaj spuštanja niz stablo granom tekućeg čvora koja sadrži isključivo literale. Za razliku od ostalih delova pretrage, identička pretraga se može odvijati isključivo niz jednu granu koja odgovara trenutnom nivou pretraživane putanje. Ova pretraga odgovara standardnom načinu pretrage ključa u prefiksnom stablu i zbog jedinstvenosti rešenja predstavlja najefikasniji vid pretrage.

Prilagođavanje pretrage slučaju dodavanja novog obrasca

Pored poklapanja putanja do resursa, kao što je već pomenuto u sekciji 5.4, pretraga nad kompresovanim glob-prefiksnim stablom može se koristiti i kao početna faza dodavanja novog obrasca u stablo. Naime, ukoliko novi obrazac može biti poklopljen nekim obrascem koji se već nalazi u stablu, taj novi obrazac se odbacuje. Za proveru da li se novi obrazac može poklopiti koristi se isti postupak pretrage kao i pri poklapanju putanja, uz nekoliko pretpostavki omogućenih pretprocesiranjem[12] i uz određena prilagođavanja.

[12]Sortiranjem i redosledom pretraživanja koji se kreće od čvorova sa dvostrukom zvezdom, preko čvorova sa jednostrukom zvezdom, pa do čvorova koji sadrže isključivo literale.

Najpre, kada je reč o pretrazi nivoa sa dvostrukom zvezdom, nisu potrebne nikakve izmene, jer dvostruka zvezda poklapa proizvoljan sadržaj, uključujući i specijalne karaktere novog obrasca.

U slučaju kada pretraga dođe do čvora sa jednostrukom zvezdom, potrebno je izvršiti određeno prilagođavanje nad nivoom obrasca koji se dodaje. Naime, do sada je razmatran slučaj poklapanja tekućeg čvora sa jednostrukom zvezdom sa jednim nivoom pretraživane putanje (kao što je objašnjeno u poglavlju 5.5). U slučaju putanja do resursa, takav nivo sadrži isključivo literale. Međutim, kada se pretraga vrši nad drugim obrascem (umesto nad putanjom), taj nivo takođe može sadržati specijalne karaktere. Tada se javlja problem kako poklopiti tekući čvor koji sadrži specijalni karakter sa nivoom koji takođe sadrži specijalni karakter. Ovaj problem moguće je raščlaniti na tri slučaja:

1. Novi nivo sadrži dvostruku zvezdu — do ovog slučaja nije moguće doći jer su obrasci prethodno sortirani, te je nemoguće u formirano podstablo dodati novi obrazac u kojem se na istom nivou pojavljuje karakter dvostruke zvezde. Takav obrazac našao bi se ranije u nizu obrazaca u odnosu na onaj koji se već nalazi u stablu, pa bi on ranije bio uvršten u stablo i do ove situacije ne može da dođe.

2. Novi nivo sadrži jednostruku zvezdu — u ovom slučaju ne primenjuje se tokenizacija nivoa[13] (kako je objašnjeno u sekciji 5.1), već se ceo nivo posmatra kao celina u kojoj specijalni karakter $*$ nema posebno značenje (ponaša se kao literal). Ovakav nivo, interpretiran u potpunosti kao literal, može se proslediti standardnoj funkciji za pretragu čvora sa jednostrukom zvezdom koja je ranije opisana.

3. Novi nivo sadrži samo literale — nije potrebno vršiti dodatna prilagođavanja, jer se takav nivo novog obrasca ponaša kao običan nivo putanje koja se pretražuje.

.

Naposletku, nakon obrade čvorova sa dvostrukom i jednostrukom zvezdom, pretraga dolazi do potencijalnog spuštanja ka čvoru koji sadrži literal. Za ovaj deo

[13] Ako je tokenizacija već izvršena pre početka rada algoritma, primenjuje se postupak sažimanja nivoa pri čemu se više tokena spaja u jedan, počevši od tokena jednostruke zvezde koji sadrži oznaku početka nivoa, pa sve do prvog narednog tokena sa takvom oznakom.

pretrage nisu potrebna nikakva prilagođavanja, budući da tekući nivo obrasca koji se dodaje može sadržati isključivo literale. Važno je uočiti da je nemoguća situacija u kojoj se u stablu došlo do čvora sa literalom, a da na tom istom nivou novi obrazac sadrži specijalni karakter, i to upravo zbog sortiranja izvršenog na početku rada algoritma. Kada bi novi obrazac sadržao specijalni karakter na tekućem nivou, on bi se u sortiranom redosledu našao pre obrasca koji se već nalazi u stablu (koji na tom nivou ne sadrži specijalni karakter). Samim tim, bio bi dodat u stablo pre obrasca bez specijalnih karaktera, pa je ovakav scenario nemoguć.

5.6 Obrada izbegnutih karaktera

Iako se to u praksi retko dešava, operativni sistem Linux dozvoljava da naziv datoteke sadrži karaktere kao što su +, ∗, ? i slično. S obzirom na to da u slučaju kompresovanog glob-prefiksnog stabla karakter ∗ ima specijalno značenje, uveden je koncept izbegnutih (engl. *escaped*) karaktera po uzoru na regularne izraze u programskom jeziku Java.

Kako bi se izbeglo specijalno značenje karaktera zvezde pri definisanju obrasca, korisnik ovaj karakter navodi sa prefiksom \ (na primer, datoteka `,,ime*prezime''` bi se u obrascu navela u obliku `,,ime\*prezime''`). S druge strane, pre poklapanja putanje, sam algoritam prefiksuje svaki karakter zvezde karakterom za izbegavanje specijalnog značenja. Dodatno, pri poklapanju putanje sa čvorom sa jednostrukom zvezdom, algoritam eksplicitno definiše prefiks takvog čvora kao „putanju do prve neizbegnute zvezde", pa će se svaka izbegnuta zvezda posmatrati kao literal. Takođe, prilikom tokenizacije, nivo koji sadrži izbegnutu zvezdu biće mapiran u literalni čvor, budući da postupak tokenizacije uzima u obzir prefiks za izbegavanje specijalnog značenja.

5.7 Adaptacija stabla potrebama komponente *GraalVM Native Image*

Radi poboljšanja performansi komponente *GraalVM Native Image*, izbegava se bezuslovna registracija svih resursa. Znatno bolji pristup predstavlja registracija resursa tek onda kada je on zaista neophodan aplikaciji. Međutim, s obzirom na to

da se komponenta *GraalVM Native Image* temelji na pretpostavci zatvorenog sveta, sve dinamičke komponente moraju biti poznate unapred.

Da bi se prevazišla ova kontradikcija, prilikom definisanja resursa u konfiguracionim datotekama se, pored glob putanje, navode i uslovi pod kojima resurs treba registrovati. Uslov pod kojim se resurs registruje definiše se navođenjem tipa podataka koji mora biti dostupan u aplikaciji tokom njene statičke analize (proces je detaljno opisan u sekciji 2.4).

Jedan od početnih koraka izgradnje izvršive datoteke posredstvom komponente *GraalVM Native Image* jeste upravo statička analiza same aplikacije. Bez ulaženja u detalje statičke analize, za ovu temu je važno napomenuti da, pored ostalog, ona zaključuje koji će se tipovi podataka koristiti tokom rada aplikacije. Kada statička analiza prepozna određeni tip podataka kao dostupan, registruju se svi resursi čiji je uslov sadržao taj tip. Primer 3 ilustruje uslovni način registracije resursa.

Primer 3. *Recimo da imamo konfiguracionu datoteku sledeće sadržine:*

```
{
  "resources" : [
    {
      "condition" : {
        "typeReached" : "java.lang.String"
      },
      "glob" : "foo/bar/datoteka.txt"
    }
  ]
}
```

Ukoliko statička analiza zaključi da će `java.lang.String` *biti dostupan tip podataka u vremenu izvršavanja aplikacije (engl.* runtime*), resurs* `foo/bar/datoteka.txt` *će biti registrovan. Ukoliko bi glob obrazac sadržao specijalne karaktere, bili bi registrovani svi resursi obuhvaćeni tim obrascem.*

Postoji veza između uslova registracije i obrasca koji treba registrovati. Zbog toga se, za potrebe komponente *GraalVM Native Image*, implementacija kompresovanog glob-prefiksnog stabla proširuje tako da terminirajući čvorovi pamte i listu uslova pod kojima se resurs za poklopljene putanje može registrovati. Posledica toga jeste da sama dodatna informacija koja se čuva u terminirajućem čvoru određuje da li će se prilikom izgradnje stabla novi obrazac dodati, ili će se smatrati redundantnim i

biti preskočen. Primerom 4 prikazano je kako dodatna informacija utiče na odluku da li se obrazac smatra redudantnim ili ne.

Primer 4. *Pretpostavimo da se u stablu nalazi samo obrazac $a/b*/x$ i da njegov terminirajući čvor pamti informaciju da se ovaj obrazac primenjuje isključivo kada je ispunjen uslov X. Nakon toga, u stablo se dodaje obrazac $a/b*c/x$ sa uslovom registracije Y. Ukoliko se ne bi pamtila dodatna informacija o uslovu registracije, moglo bi se zaključiti da je postojeći obrazac $a/b*/x$ opštiji od obrasca $a/b*c/x$, te da se novi obrazac može odbaciti. Međutim, postavlja se pitanje šta učiniti sa uslovom registracije. Ukoliko bi se i on odbacio zajedno sa samim obrascem, resuri kao što je $a/bxc/x$ ne bi se registrovali ako statička analiza utvrdi da je uslov Y ispunjen, a uslov X nije*[14]*. Takođe bi došlo do greške i ukoliko bismo odbacili novi obrazac $a/b*c/x$ zato što je poklopljen postojećim obrascem, a dodatni uslov registracije pridodali listi dodatnih uslova koje čuva terminirajući čvor obrasca $a/b*/x$. Naime, u tom slučaju bi obrazac $a/b*/x$ čuvao informaciju da se resursi, koji se poklopaju sa njim, registruju ukoliko je ispunjen uslov X ili uslov Y. Međutim, ako bi tokom registracije resursa bio razmatran resurs $a/bx/x$, a statička analiza utvrdi da je ispunjen samo uslov Y, a ne i uslov X, taj resurs bi bio registrovan. Time bi bila napravljena greška, jer je pod uslovom Y trebalo registrovati isključivo resurse koji odgovaraju obrascu $a/b*c/x$, što resurs iz datog primera ne ispunjava.*

Dakle, u ovom slučaju, novi obrazac $a/b*c/x$ zaista se može poklopiti već postojećim obrascem $a/b*/x$, ali se zbog dodatne informacije o uslovu registracije ovaj obrazac ne sme odbaciti. Jedina ispravna akcija koju treba preduzeti jeste da se novi obrazac doda u stablo, kao da ne može biti poklopljen nijednim drugim obrascem.

Ipak, postoje dve situacije kada se novi obrazac može odbaciti i time očuvati kompresovanost stabla:

1. Ukoliko se novi obrazac može poklopiti nekim postojećim obrascem iz stabla, pri čemu oba obrasca imaju isti uslov registracije, novi obrazac se može bezbedno odbaciti. Na primer, bezbedno je odbaciti obrazac $a/b*c/x$ koji se registruje pod uslovom X ukoliko se u stablu već nalazi obrazac $a/b*/x$ čiji je uslov registracije takođe X[15].

[14] Budući da uslov X nije ispunjen, obrazac $a/b*/x$ se ne aktivira. S druge strane, uslov Y je ispunjen, ali je odbačen tokom konstrukcije stabla.

[15] U prethodno opisanom primeru novi obrazac nije mogao biti odbačen upravo zbog toga što je imao drugačiji uslov registracije u odnosu na postojeći obrazac.

2. Ukoliko je novi obrazac identičan nekom postojećem obrascu iz stabla, a razlikuje se samo po dodatnoj informaciji koju nosi, postupa se na sličan način. Treba primetiti da trenutno opisana implementacija ne vodi računa o tome da li je novi obrazac identičan nekom postojećem obrascu iz stabla, već samo o tome da li se sa njim može poklopiti. Međutim, ova dodatna restrikcija za odbacivanje novog obrasca zapravo ne menja samu implementaciju. Razlog leži u tome što se, nakon uspešnog poklapanja i uvida da se novi obrazac ipak mora dodati zbog drugačije dodatne informacije, prilikom dodavanja novog obrasca kreće kroz već postojeće grane stabla sve do terminirajućeg čvora, gde se samo ažurira lista dodatnih informacija. Time se dodaju novi obrazac i njegov uslov registracije, pri čemu se u stablo ne dodaje nijedan novi čvor.

Pored toga, za potrebe projekta *GraalVM Native Image* bilo je neophodno izvršiti još jedno prilagođavanje stabla. S obzirom na to da se vrši registracija Java resursa, potrebno je uzeti u obzir da se od verzije Java 9 resursi mogu nalaziti unutar Java modula. Java modul predstavlja imenovani, samostalni skup povezanih paketa i resursa koji pomoću specijalizovane datoteke eksplicitno definiše koje svoje delove deli sa drugim modulima, a koje spoljne module sam koristi za rad. Preciznije, Java modul obezbeđuje grupisanje paketa programskog koda uz dodatni nivo enkapsulacije. U praktičnoj primeni to znači da mogu postojati dva resursa sa identičnim putanjama ukoliko se nalaze u različitim modulima[16].

U slučaju kompresovanog glob-prefiksnog stabla, ovo podrazumeva neophodnost čuvanja informacija o modulima, kako se ne bi dogodilo da korisnik zahteva resurs iz jednog modula, a da se registruju svi resursi sa tom putanjom iz svih dostupnih modula. Najjednostavniji način da se ovaj problem reši jeste dodavanje naziva modula kao prefiksa glob obrascu u okviru kog treba poklapati traženi resurs. Time se već u prvom koraku poklapanja putanje do resursa odbacuju svi resursi koji ne pripadaju modulu koji je korisnik naveo[17].

Implementacija ovako prilagođene strukture javno je dostupna kao integralni deo projekta *GraalVM Native Image* [1].

[16]Na primer, mogu postojati dva različita resursa na istoj putanji $a/b/c$, pri čemu se jedan nalazi u modulu $M1$, a drugi u modulu $M2$.

[17]U ranijim verzijama projekta *Native Image*, korisnik je u konfiguracionim datotekama putanju do resursa navodio sa modulom kao prefiksom. Od verzije projekta koja uvodi glob obrasce, moduli su izdvojeni u zasebno polje unutar entiteta koji nosi informacije o samom glob obrascu i uslovu registracije.

5.8 Integrisanje glob-prefiksnog stabla u komponentu *GraalVM Native Image*

Naposletku, treba navesti i proces uvrštavanja opisane strukture u samu komponentu *GraalVM Native Image*. Naime, kako bi se očuvala kompatibilnost sa ranije razvijenim projektima, zadržana je registracija resursa pomoću regularnih izraza, uz napomenu korisniku da se prethodno navedene prednosti ne mogu ostvariti dok se konfiguracione datoteke ne prevedu u novi format.

Samo prevođenje starih konfiguracionih datoteka koje koriste regularne izraze u nove konfiguracione datoteke zasnovane na glob obrascima nije moguće automatizovati. Problem leži u tome što regularni izrazi poseduju znatno veću izražajnu moć, te se ne mogu svi regularni izrazi preslikati u glob obrasce. Ovaj korak migracije zahteva intervenciju korisnika, što predstavlja jedan od nedostataka uvođenja novog formata.

S druge strane, suprotan smer je u potpunosti izvodljiv — svaki glob obrazac se može prevesti u ekvivalentni regularni izraz. Iz tog razloga, starije verzije projekta *GraalVM Native Image* unapređene su ovom automatizacijom, kako bi korisnici starijih verzija mogli da koriste konfiguracione datoteke pisane u novom formatu.

Naime, postoje projekti otvorenog koda (npr. *GraalVM Reachability Metadata* [11]) koji centralizovano čuvaju konfiguracione datoteke za veliki broj popularnih biblioteka i radnih okvira (engl. *frameworks*). Ovakvim repozitorijumima se često pristupa automatizovano radi preuzimanja konfiguracionih datoteka, te je uvođenje mehanizma za prevođenje glob obrazaca u regularne izraze od izuzetnog značaja. Zahvaljujući tome, repozitorijumi koji skladište konfiguraciju mogu preći na novi format, dok korisnici starijih verzija komponente *GraalVM Native Image* mogu neometano da koriste te datoteke, bez potrebe da vode računa o formatu u kom su one zapisane.

5.9 Benefiti korišćenja kompresovanog glob-prefiksnog stabla

Uvođenjem kompresovanog glob-prefiksnog stabla u projekat *GraalVM Native Image* ostvareno je nekoliko ključnih prednosti.

Osnovna motivacija za uvrštavanje ove strukture u projekat *GraalVM Native*

Image bila je omogućavanje efikasnog pretraživanja resursa prilikom analize korisničkih grešaka tokom izvršavanja aplikacije. Naime, ukoliko se tokom rada aplikacije pokuša pristup resursu koji nije registrovan, aplikacija ne može nastaviti sa radom. Razlog zbog kog određeni resurs nije registrovan može biti taj što sam resurs zapravo ne postoji na sistemu ili što ga korisnik nije naveo u konfiguracionoj datoteci. Kako bi se pružio odgovor na pitanje zašto određeni resurs nije registrovan, bilo je neophodno ponovo proći kroz sve obrasce koje je korisnik zadao. Pre uvođenja kompresovanog glob-prefiksnog stabla, uzrok odsustva resursa tokom izvršavanja nije bilo moguće efikasno utvrditi iz nekoliko razloga:

1. Obrasci se nisu čuvali u strukturiranom obliku, već se, kako bi se utvrdilo da li je korisnik zahtevao resurs, konfiguraciona datoteka morala ponovo čitati.

2. Obrasci su mogli poticati iz više različitih konfiguracionih datoteka, čije putanje tokom izvršavanja nisu bile poznate.

3. Ukoliko bi se obrasci ponovo pročitali iz konfiguracionih datoteka, za njihovo poklapanje bilo bi neophodno da u vreme izvršavanja bude uvršten čitav mehanizam za obradu regularnih izraza. S obzirom na robusnost regularnih izraza, ceo mehanizam bi značajno povećao memorijsko zauzeće aplikacije.

4. Provera da li se putanja do resursa može poklopiti nekim regularnim izrazom zahtevala bi njeno poređenje sa celim skupom regularnih izraza navedenih u konfiguracionim datotekama, što bi bilo i vremenski neefikasno.

Uvođenjem kompresovanog glob-prefiksnog stabla ovi problemi su prevaziđeni. Najpre, svi obrasci su objedinjeni na jednom mestu, strukturirani i mogu se efikasno pretraživati. Sama struktura podataka kreira se jednom, tokom faze prevođenja aplikacije, i ostaje dostupna i u vreme njenog izvršavanja. Naposletku, mehanizam za pretraživanje putanja do resursa unutar ovog stabla gotovo da nema uticaja na veličinu izvršive datoteke.

Zatim, čak i bez formalnog merenja, može se tvrditi da je korišćenje kompresovanog glob-prefiksnog stabla vremenski efikasnije od prethodnog pristupa. U ranijem pristupu se za svaki pronađeni resurs pokušavalo poklapanje sa svakim pojedinačnim regularnim izrazom koji je korisnik naveo u konfiguracionoj datoteci. Ovakav pristup predstavlja iscrpnu pretragu, pri čemu se u svakom njenom koraku koristi mehanizam za poklapanje regularnih izraza, što dodatno povećava vremensku složenost starog algoritma.

Nasuprot tome, važno je uočiti da se zahvaljujući kompresiji stabla potencijalno neće svi obrasci koristiti prilikom poklapanja resursa. Obzirom da je stari mehanizam koristio iscrpnu pretragu, on nije mogao da prepozna redundantnost niti da preskoči evaluaciju određenih regularnih izraza. To se najbolje može ilustrovati sledećim primerom. Pretpostavimo da jedna konfiguraciona datoteka sadrži N jednostavnih regularnih izraza[18] i jedan složeni regularni izraz koji obuhvata sve prethodno navedene izraze. Stari mehanizam bi u tom slučaju izvršio $N + 1$ provera za svaki pronađeni resurs. Za R resursa, stari algoritam bi izvršio ukupno $R \cdot (N + 1)$ provera. Nasuprot tome, ukoliko bi konfiguraciona datoteka sadržala N jednostavnih glob obrazaca[19] i jedan glob obrazac koji poklapa sve prethodno navedene obrasce, novi algoritam bi za svaki pronađeni resurs izvršio tačno jednu proveru, jer bi se u kompresovanom glob-prefiksnom stablu nalazio samo poslednji (najopštiji) obrazac. Odnosno, za R resursa izvršilo bi se svega R poklapanja. Iz navedenog jasno sledi da je novi pristup znatno efikasniji.

Pored toga, čak i bez kompresije stabla, algoritam zasnovan na glob-prefiksnom stablu radiće efikasnije jer će, zahvaljujući svojstvima prefiksnog stabla, vršiti odsecanja u pretrazi. Treba napomenuti da do odsecanja ne mora doći prilikom obrade svakog čvora stabla, jer se u pojedinim, ranije opisanim, situacijama može desiti da algoritam poklapanja mora da razmotri više od jednog potencijalnog potomka[20]. Međutim, u praksi se slučajevi sa više uzastopnih specijalnih karaktera ne javljaju često, te se odsecanja zasnovana na prefiksnom stablu veoma frekventno ostvaruju.

Takođe, prostor za greške koje korisnik može napraviti prilikom definisanja obrazaca za poklapanje resursa znatno je manji, jer glob obrasci poseduju manju izražajnu moć u poređenju sa regularnim izrazima.

[18] Regularni izraz koji ne sadrži specijalne karaktere, već samo literale.

[19] Glob obrazac koji ne sadrži specijalne karaktere, već samo literale.

[20] Zbog specijalnog karaktera $*$ može postojati više od jednog čvora potomka koji može da poklopi naredni nivo u putanji.

Glava 6

Zaključak

U okviru ovog rada analiziran je i rešen problem efikasnog definisanja i obrade šablona za registraciju Java resursa u komponenti *GraalVM Native Image*. Istraživanje je obuhvatilo tranziciju sa tradicionalnih Java regularnih izraza na jednostavnije i bezbednije glob obrasce, kao i projektovanje i implementaciju specijalizovane strukture podataka — kompresovanog glob-prefiksnog stabla. Rad demonstrira da uvođenje strukturiranog pristupa zasnovanog na glob obrascima smanjuje prostor za korisničke greške pri definisanju obrazaca za registraciju resursa kao i smanjenje memorijskog zauzeća aplikacija. Takođe, pokazano je i da se efikasnost registracije resursa, kao i vreme potrebno za pronalazak obrazaca koji nedostaju u konfiguracionoj datoteci, značajno poboljšava.

Prethodni mehanizam, zasnovan na Java regularnim izrazima služio je kao osnova za evaluaciju unapređenja kvaliteta performansi i smanjenja prostora grešaka novog mehanizma. Pokazano je da je korišćenje regularnih izraza nosilo visoku složenost i rizik od korisničkih grešaka, uz zahtev da se u izvršivu datoteku uvrsti čitav mehanizam za obradu regularnih izraza, što je nepotrebno opterećivalo memorijsko zauzeće. Pored toga, nemogućnost strukturiranog čuvanja obrazaca uslovljavala je iscrpnu pretragu bez mogućnosti odsecanja, gde se za svaki resurs proveravao celokupan skup navedenih izraza.

Projektovano kompresovano glob-prefiksko stablo pokazalo je da se kroz preciznu tokenizaciju obrazaca i determinističko sortiranje po opštosti, redundantni obrasci mogu efikasno prepoznati i odbaciti. Time je postignuto da stablo zadrži kompresovanu i nepromenljivu formu, omogućavajući da se u prisustvu opštijih obrazaca izvrši znatno manji broj provera u poređenju sa iscrpnom pretragom starog mehanizma, čime je potvrđena vrednost strukturiranog organizovanja konfiguracionih

podataka.

Najznačajnija poboljšanja postignuta su u domenu efikasnosti pretrage i smanjenja veličine izvršive datoteke. Eliminacijom potrebe za mehanizmom regularnih izraza u vremenu izvršavanja, izbegava se nepotrebno uvećanje memorijskog zauzeća konstruisane izvršive datotke. Istovremeno, omogućeno je efikasno evaluiranje korisničkih grešaka pri pristupu neregistrovanim resursima bez ponovnog čitanja konfiguracionih fajlova ili dodatnog memorijskog opterećenja. Mogućnost odsecanja grana tokom pretrage unutar stabla dodatno je ubrzala proces poklapanja putanja resursa u praksi.

Ovaj rad posebno izdvaja prilagođavanje stabla specifičnim zahtevima komponente *GraalVM Native Image*, gde je implementirana podrška za uslovnu registraciju resursa i enkapsulaciju kroz Java module. Analiza je pokazala da čuvanje uslova registracije u terminirajućim čvorovima i dodavanje prefiksa sa imenom modula omogućava očuvanje tačnosti registracije bez narušavanja pravila kompresije stabla, čime je garantovana precizna izolacija resursa i ispravan rad aplikacija.

Takođe, otvara se prostor za buduća istraživanja koja bi obuhvatala mogućnost primene kompresovanog glob-prefiksnog stabla i na druge delove komponente *GraalVM Native Image*, ali i van samog sistema, jer pomenuta struktura podataka u opštem slučaju rešava problem poklapanja glob obrazaca sa zadatim tekstom. Pored toga, unapređenja ove strukture podataka mogu ići u pravcu podržavanja još nekih specijalnih karaktera koje definišu određeni prošireni formati glob obrazaca.

Bibliografija

[1] Compressed Glob Trie. on-line at: https://github.com/oracle/graal/tree/master/substratevm/src/com.oracle.svm.core/src/com/oracle/svm/core/jdk/resources/CompressedGlobTrie.

[2] Dynamic Proxies in Java. on-line at: https://www.baeldung.com/java-dynamic-proxies.

[3] Glob patterns. on-line at: https://docs.oracle.com/en-us/iaas/Content/devops/using/glob-patterns.htm.

[4] Graal Compiler. on-line at: https://www.graalvm.org/latest/reference-manual/java/compiler/.

[5] GraalVM. on-line at: https://www.graalvm.org/.

[6] Guide to Java Reflection. on-line at: https://www.baeldung.com/java-reflection.

[7] Truffle Language Implementation Framework. on-line at: https://www.graalvm.org/latest/graalvm-as-a-platform/language-implementation-framework/.

[8] Using Regular Expressions. on-line at: https://docs.oracle.com/en/industries/communications/session-border-controller/9.2.0/hmr/using-regular-expressions.html#GUID-F4FC8ED2-4473-49E4-8AAD-A87EB443399F.

[9] Vesna Marinković and Filip Marić. *Konstrukcija i analiza algoritama*. Matematički fakultet, Univerzitet u Beogradu, 2024.

[10] Roy A. Maxion and Robert W. Reeder. Improving user-interface dependability through mitigation of human error. *International Journal of Human-Computer Studies*, 63(1):25–50, 2005. HCI research in privacy and security.

[11] Oracle. GraalVM reachability metadata repository. `https://github.com/oracle/graalvm-reachability-metadata`, 2026.

[12] Oracle Corporation. The Java HotSpot performance engine architecture. `https://www.oracle.com/java/technologies/whitepaper.html`, 2019.

[13] Fabio Paterno and Carmen Santoro. Preventing user errors by systematic analysis of deviations from the system task model. *Int. J. Hum.-Comput. Stud.*, 56(2):225–245, February 2002.

[14] Christian Wimmer, Codrut Stancu, Peter Hofer, Vojin Jovanovic, Paul Wögerer, Peter B. Kessler, Oleg Pliss, and Thomas Würthinger. Initialize once, start fast: Application initialization at build time. 3:1–29.

Biografija autora

David Nestorović rođen je u Prijepolju 11. Februara 1999. godine. Osnovnu i srednju školu završava u rodnom gradu uz zvanje đaka generacije Prijepoljske gimnaizije. Godine 2017. upisuje Matematički fakultet univerziteta u Beogradu, smer informatika, na kojem diplomira 2022. godine.

Tokom poslednje godine osnovnih akademskih studija, počinje da radi u kompaniji *Logeecom*. Nakon nepunih godinu dana prelazi u kompaniju Oracle, u kojoj provodi naredne tri i po godine radeći na projektu *GraalVM Native Image*. Svoj rad u kompaniji *Oracle* većinski fokusira na poboljšanje korisničkog iskustva kroz razvoj u oblasti obrade i prikupljanja Java resursa u okvirima projekta *GraalVM*. Takođe, tokom ovog perioda značajan deo radnog vremena provodi na razvoju projekata otvorenog koda(*Reachability Metadata* i *Native Build Tools*), koje *GraalVM Native Image* koristi radi poboljšanja korisničog iskustva. U sklopu ovih projekata, konstantno komunicira sa inženjerima iz vodećih kompanija u domenu programskog jezika Java, osiguravajući integraciju projekta *GraalVM Native Image* sa ostalim članovima Java ekosistema. Aprila 2026. godine prelazi u kompaniju *JetBrains*, na projekat *TeamCity*, gde je zadužen za *Maven* i *Gradle* integracije.