LATVIJAS UNIVERSITĀTES
EKSAKTO ZINĀTŅU UN TEHNOLOĢIJU FAKULTĀTES
DATORIKAS NODAĻA

INTERPRETATORS PROGRAMMĒŠANAS VALODAI IMMUTANT

KVALIFIKĀCIJAS DARBS DATORZINĀTNĒS

Autors: Rolands Frīdemanis
Studenta apliecības Nr.: rf23009
Darba vadītājs: asociētais profesors Dr. sc. comp. Aleksandrs Belovs

RĪGA, 2025

Anotācija

Mūsdienās liela daļa no vispārīga pielietojuma programmēšanas valodām, satur sematiku, kas paredz, ka datu mainība ir ierasta lieta, tomēr šim pastāv būtisks trūkums, kas ir datu neparedzamība. Lai spriešanu par programmas stāvokli padarītu paredzamu, šī darba ietvaros tiek izstrādāta programmēšanas valoda, kuras gramatika un sintakse ierosina noteikta datu nemainību. Šis darbs satur valodas specifikāciju un interpretatora arhitektūras dokumentāciju lekserim, parsētājam un abstraktā sintakses koka pārstaigāšanas algoritmam. Rezultātā, izmantojot valodu C, tiek izveidota augsta līmeņa, intepretējama, dinamiski tipizēta valoda ar atomāriem datu tipiem.

Atslēgvārdi: interpretators, AST, funkcionālā programmēšana, C valoda, datu nemainība

Abstract

Most modern general-purpose programming languages use grammar and syntax that suggests mutable data being an ordinary matter, which in reality complicates reasoning about program state. To make such reasoning predictable, this work explores the design and delelopment of a programming language with explicit syntax and semantics of immutable data. This work contains specification for such language and documentation of the architecture for the lexer, parser and AST-walker of the underlying interpreter. As a result, a high-level, interpreted, dynamically typed programming language with only atomic data types is created. The language is implemented in C.

Keywords: interpreter, AST, functional programming, C language, data immutability

SATURS

APZĪMĒJUMU SARAKSTS

AST	Abstraktās sintakses koks
immutability	Pilnīga datu nemainība (koncepts)
immutable	Nemainīgs (datu īpašība)
read-only	Tikai lasāmi (dati)

IEVADS

Datu mainība un to nepārtraukta plūsma no viena mainīga uz otru ir neapstrīdama daļa no lielas daļas programmatūras. Droši var apgalvot, ka datorsistēmu arhitektūru atmiņas koncepcija ļauj izmantot atmiņu vairākkārtēji. Atmiņu var relocēt, izdzēst, pieprasīt lielākus atmiņas gabalus u.t.t. No šī izriet datu mainības būtība, un līdz šai dienai tā ir iekalta lielā daļā, ja ne visu, programmēšanas un skriptēšanas valodu.

No otras puses, mainīgiem datiem pastāv īpašība būt neparedzamiem. Kamēr mazas sistēmas spēj tikt galā ar nelielu daudzumu mainīgo, tad lielajās sistēma tas var kļūt par acīmredzamu problēmu. Atmiņa datoram ir viena, tomēr atsaukties uz to var no dažādām vietām dažādos laika brīžos, kas padara spriešanu par datu stāvokli daudz sarežģītāk.

Programmētāji izmanto iespēju mainīt datus brīvā veidā, jo programmēšanas valodas un to abstrakcijas ir padarījušas to tik vienkāršu. Lai gūtu vairāk kontroles pār iepriekš minēto uzvedību, valodas no C saimes, Java, JavaScript un citas valodas satur gramatikas, kas ierobežo mainīgu datu inicializāciju un to turpmāko vērtību maiņu. Kā piemēru var minēt `const` atslēgvārdu no valodas C, kas fiksē doto mainīgo un liedz pārrakstīt tā vērtību, tomēr tas tikai daļēji attiecas uz atsauces tipa vērtībām. Kaut arī šāda tipa mainīgais vienmēr atsauksies tikai uz vienu konkrētu atmiņas apgabalu, nav noteikts, ka vērtība, kas tajā atrodas, nemainīsies. Līdz ar to, `const` atslēgvārds negarantē patiesu datu nemainību, bet tikai noslēdz to uz `readonly` pieeju. Neskatoties uz dažāiem programmēšanas valodu centieniem ierobežot datu mainību, tādas problēmas kā neparedzamas mainīgo vērtības pavedienu izpildes laikā, mainīgo aizstājējvārdu nepārdomāta ieviešana un izmantošana, mainīgo nejauša re-inicializācija u.c. joprojām sastāda lielu daļu programmu. [1]

Šis darbs izskata jaunas interpretējamas programmēšanas valodas izveidi, kurai pielietota datu nemainības semantika, cenšoties mazināt programmatūras izstrādes problēmas saistītas ar stāvoklu maiņu. Tas arī nozīmē, ka galvenā ideja kalpo far faktoru valodas nosaukuma izvēlei. Savukārt interpretatora sākotnējā versijā, kas šeit ir izskatīta, ietilpst apstrādes loģika sekojošām daļām: primitīvi datu tipi un to glabāšana mainīgos, ar to saistītā datu nemainības semantika, izteikumi, tīras un netīras funckijas, kontroles plūsma un cikli.

Šī darba pirmajā nodaļā tiks izskatīta programmēšanas valodas specifikācija, kurai pakārtosies interpretators. Savukārt otrā nodaļa būs veltīta interpretatora arhitektūras pārskatam un tā tehniskai specifikācijai, kā arī tehnoloģijām izmantotām porgrammatūras izstrādei. Trešā un ceturtā nodaļa virzīs uzmanību uz leksera un parsētāja izstrādes detaļām, bet piektā nodaļa būs par rezultējošā AST apstrādi. Tālākajā sestajā nodaļā būs apokopoti rezultāti par jauniegūto valodu un tās interpretatoru, un praktiski demonstrēts valodas pielietojums. Septītajā nodaļā tiks izvirzīts turpmākais valodas attīstības plāns un idejas. Visbeidzot, astotajā nodaļā seko nobeigums un kopsavilkums par paveikto darbu.

BIBLIOGRĀFIJA

[1] Michael Coblenz, Joshua Sunshine, Jonathan Aldrich, Brad Myers, Sam Weber, and Forrest Shull. Exploring language support for immutability. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, pages 736–747. IEEE/ACM, 2016.