LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

PIRMKODA STATISTISKĀS ANALĪZES AUTOMATIZĀCIJA

BAKALAURA DARBS

Autors: Jānis Knets
Studenta apliecības Nr.: jk07108
Darba vadītājs: profesors Dr. Leo Seļāvo

RĪGA 2013

ANOTĀCIJA

ABSTRACT

ATSLĒGVĀRDI

Saturs

APZĪMĒJUMU SARAKSTS

1 IEVADS

Dotais darbs apskatīs autora izveidotu programmu kopumu, kuru mērķis ir atpazīt C valodas pirmkoda piemērus un salīdzināt tos.

Pirmajā dokumenta daļā tiks izskaidrota kopējā arhitektūra un katrs modulis atsevišķi. Moduļu iekšējie procesi un dažās saistības ar citiem moduļiem.

Otrā dokumenta daļa stāsta par datu plūsmām caur programmu kopumu. Tiek izskaidrotas visas datu pārveidošanas. Īsumā – tiek iziets pilns datu cikls no padotā pirmkoda līdz izvadītiem rezultātiem.

Nobeigumā dotais risinājums tiks salīdzināts ar līdzīgiem, brīvi pieejamiem un tiks apspriesti dotā darba rezultāti un nākotne.

1.1 Mērķi

Šī dokumenta mērķis ir rast sapratni par iekšējo processu norisi, sniegt gan pamācoša rakstura informāciju, gan pamatu nākotnes attīstībai. Izstrādātās programmatūras mērķis ir spēt un novērtēt divus padot pirmkoda piemērus. Salīdzināšana notiek gan rezultātu ziņā, gan paša pirmkoda līmenī.

1.2 Darba struktūra

Sākumā radās ideja par ģenētiskas un modulāras programmatūras izveidošanu, kura varētu būt spējīga pati attīstīties. Tas tika aprakstīts autora iepriekšējā rakstā [ATSAUCE] . Balstoties uz paustām idejām minetā darbā tika sākts veidot sistēmu, kura būtu spējīga pati modificēt padoto pirmkodu.

Uz doto bridi tika izvēlēts mērķis iemācīt programmtu atpazīt primkodu un saprast to. Tas tiks panākts ar LEX un YACC, kuri jau tika pieminēti. Autora paša izstrādāti likumi pēc kuriem tiks glabāti dati par pirmkodu. Būtu arī jāpiemin ka datu glabāšanai un ātrumam tiek izmantoti AVL koki. [PIEMINĒT CITUS ALGORITMUS, KURI TIKS IZMANTOTI SALIDZINAŠANĀ].

Izstrādes gaitā tika izmantoti tādī rīki kā git, gdb, gcc, eclipse u.c.

2 BILDE NO AUGŠAS \ PAMATPROCESU APRASKTS

Dotā nodaļa stāsta par katru moduli atsevišķi. Ikkatram modulim ir savs nolūks. Bet programmatūra ir spējīga darboties tikai šo moduļu ciešas sadarbības rezultātā. Tomēr šī modularitāte atstāj iespēju pievienot jaunus moduļus, kuri veiks papildus darbības. Tas ir iespējams, jo pamatā tiek izmantota vienota komunikācija ar iepriekš aprakstītu protokolu caur ligzdām(unix socket) (Protokols tiks aprakstītis pie komunikāciju moduļa kopējā apraskta.), visa komunikācija ir veicama caur iepriekš izveidotu moduli, kurā būtu viegli pievienot papildus funkcionalitāti, kā rezultātā tiek iegūta pietiekoši veikla un viegli lokāma sistēma dažādām vajadžibām. Tomēr ir vajadzīgas zināšanas, lai šos rīkus varētu effektīvi izmantot. Daļu no šīm zināšānām arī cenšas sniegt dotais dokuments.

2.1 Eksistējošie moduļi.

Moduļi, kuri tiek izmantoti izstrādātās programmatūrā uz dokumenta uzrakstīšanas brīdi ir šie:

1. LEX\YACC tulkotājs,
2. Pārveidotājs (Transformer),
3. Analizātors (Analayzer),
4. Datu bāze (Primal),
5. Tiesnesis (Judge),
6. Organisms (Organism),

kā arī eksīstē, bet netiek gluži izmantoti:

1. Pārraugs (Caregiver),
2. Sargsuns (Watchdog),
3. Mutētājs (Mutator).

Moduļi ir sakārtoti pēc datu virzības secības caur tiem.

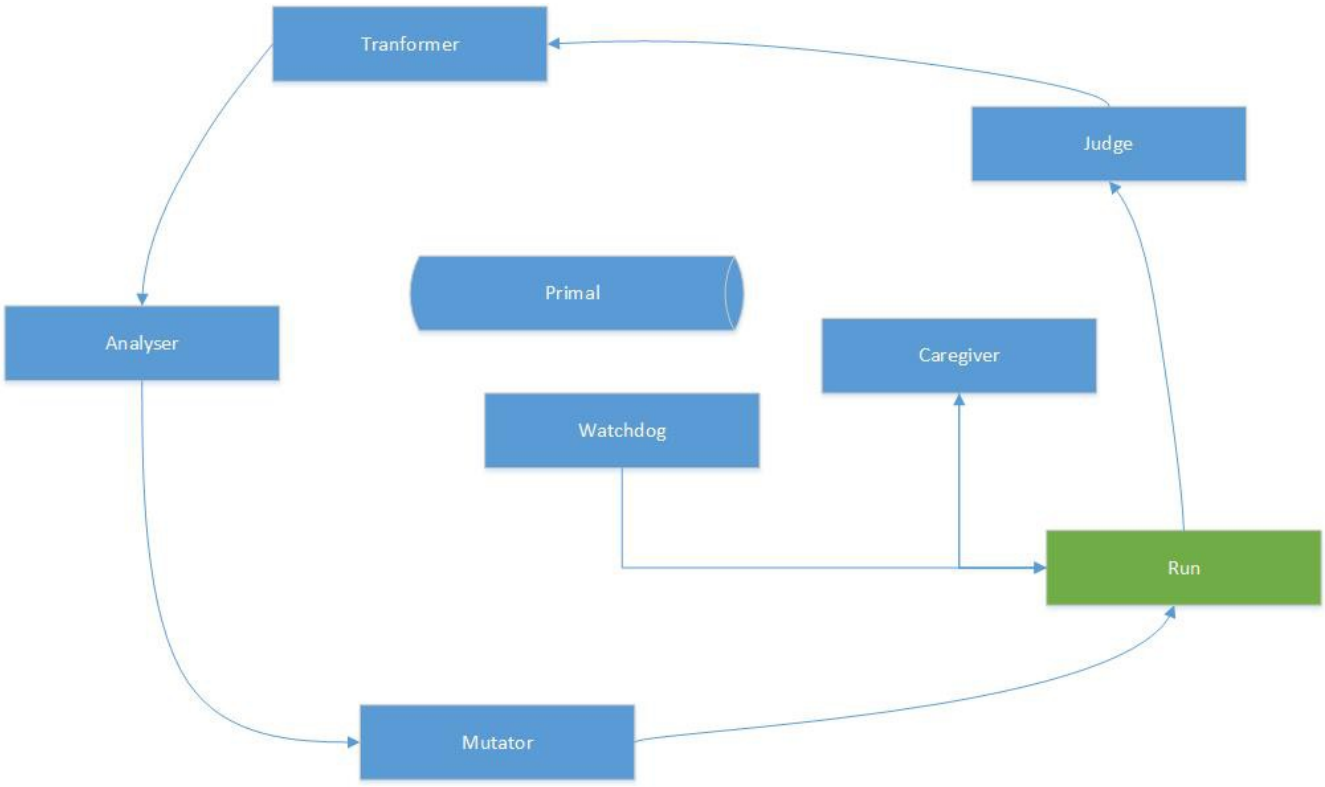
2.2 Moduļu sadalijums pa izpildāmām programmām

Pirmie seši minētie moduļi veido trīs programmatūras failus -

1. lexer.bin – sastāv no LEX YACC ģenerētiem failiem. Bāzes likumu kopumus, kurš ļauj atpazīt pirmkodu, un pārveidot to Pasaulei saprotamos datos.
2. world.bin – jeb Pasaule, sastāv no visiem pārējiem moduļiem izņemot Organisms. Galvenais process, kurš pārrauga visu notiekošo, lielākā algoritmu daļa ir iekļauta tieši šajā daļā.
3. organism<N>.bin, kur N ir nummurs pēc kārtas. Organisms ir čaulas modulis ap padoto pirmkodu. Tas veic pirmkoda kompilēšanu, startēšanu un pēc pabeigšanas paziņo Pasaulei par darba beigām. Kā var noprast, varbut vairāki šādi moduļi, kuri ir vienlaikus palaisti. Turpmākā izstrādē šie moduļi konkurēs par resursiem.

2.3 Vienkāršots skats uz datu plūsmu

Lai novērstu lasītāja nesaprašanu kādā veidā moduļu ir savstarpēji saistīti tiek dots īss apraksts par datu virzību caur visiem moduļiem datu plūsmas diagrammā.



BILDE NESKARĪT AR APRAKSTU!

1. Lietotājs startē world.bin un padod tai sākotnējo konfigurāciju.
2. Notiek inicializācija, startēts lexer.bin, kurš izveido savienojumu ar world.bin un gaida informāciju par apskatāmām pirmkoda datnēm.
3. Lietotne ievāc informāciju par padoto pirmkoda datņu nosaukumiem. Informācija var būt ievākt no iepriekš paredzētas,vai konfigurētas vietas, Ievāktie dati tiek apkopoti un ievietoti attiecīgajā datu glabāšanas struktūrā.
4. Informācija par pārbaudāmo pirmkodu tiek nogādāta lexer.bin.
5. Tiek sanemta informācija par doto pirmkodu.
6. Informācija tiek pārveidota uz datubāzē glabājamu tipu. Pēc LEX\YACC likumiem

pirmkods tiek pārveidots par programmai saprotamu datu kopumu.

7. Datu kokā ievietoti dati tiek pārveidoti ar specifisku algoritmu, kura rezultātā padotā koda mainīgo kārtība ir sakārtota noteiktā secībā, kā arī mainīgo vārdi ir standartizēti un tiek izveidotas bloku salīdzināšnas vajdzībām TLV datu virknes. Šis solis nākotnē palīdz salīdzināšanai starp diviem pirmkoda piemēriem.
8. Ievākto datu salīdzināšana.
9. Tiek izveidoti visi nepieciešamie organismi. Katrs organisms pārbauda vai pirmkods ir kompilējams un paziņo par izpildes rezultātiem Pasaulei.
10. Pasaule salīdzina pirmkoda rezultātus.
11. Balstoties uz visu ievākto informāciju tiek izlikts vērtējums par datu līdzību.

3 MODUĻI

Šī nodaļa vairāk fokusējās uz katru moduli atsevišķi un izstāsta tā mērķus, iespējas, moduļa nepieciešamību. Iekšējās datu plūsmas un metožu izejas un izvades dati.

3.1 Datu iegūšana no pirmkoda

Visi dati sākumā tiek ieguti no padota C valodas pirmkoda. Šis pirmkods tiek pārveidots par atslēgvārdiem, kuri vēlāk grupās veido loģisko vienumu, piemēram, *int a = 0;* jebkurš kaut cik zinošs programmētājs saprot ka tika definēts veselas vērtības tipa mainīgais 'a' ar sākotnējo vērtību , kura ir vienāda ar 0. Ar definēto atslēgvārdu palīdzibu programmas algoritms mēģina atpazīt tam padoto simbolu virkni, kā programmas kodu.

3.2 Kas ir LEX?

LEX ir datora programma, kura ģenerē leksikas analizatorus. Bieži tiek izmantots kopā ar YACC. Sākumā tiek uzrakstīti likumi kā veidot atslēgvārdus, ar lex (vai dotajā gadijumā FLEX, kas ir Ubuntu Linux programmas implementācija) tiek uzģenerēts C programmēšanas valodā yy.lex.h fails. [ATSAUCE UZ LEX INFORMĀCIJU]

3.2.1 LEX atslēgvārdi

Dotā tabula apraksta uz doto brīdi eksistējošos atslēgvārdus, kuri tiek izmantoti, lai izveidotu leksikas analizatoru. 1. Tabula apraksta iepriekš definētus

1. Tabula

O	[0-7]
D	[0-9]
NZ	[1-9]
L	[a-zA-Z_]
A	[a-zA-Z_0-9]
H	[a-fA-F0-9]
HP	(0[xX])
E	([Ee][+-]?{D}+)
P	([Pp][+-]?{D}+)
FS	(f|F|l|L)
IS	(((u|U)(l|L|ll|LL)?)|((l|L|ll|LL)(u|U)?))
CP	(u|U|L)
SP	(u8|u|U|L)
ES	(\\(['"\?\\abfnrtv]|[0-7]{1,3}|x[a-fA-F0-9]+))
WS	[\t\v\n\f]

2. Tabula

Maska	**Atslēgvārds**
"/*"	comment
"//".*	/* consume //-comment */
"#include <"{A}*"."{A}*">"	/* ignore */
"#include \""{A}*"."{A}*"\""	/* ignore */
"auto"	AUTO
"break"	BREAK
"case"	CASE
"char"	CHAR
"int"	INT
"short"	SHORT
"long"	LONG
"void"	VOID
"signed"	SIGNED

Maska	**Atslēgvārds**
"unsigned"	UNSIGNED
"const"	CONST
"continue"	CONTINUE
"default"	DEFAULT
"do"	DO
"double"	DOUBLE
"else"	ELSE
"enum"	ENUM
"extern"	EXTERN
"float"	FLOAT
"for"	FOR
"goto"	GOTO
"if"	IF
"inline"	INLINE
"register"	REGISTER
"restrict"	RESTRICT
"return"	RETURN
"sizeof"	SIZEOF
"static"	STATIC
"struct"	STRUCT
"switch"	SWITCH
"typedef"	TYPEDEF
"union"	UNION
"volatile"	VOLATILE
"while"	WHILE
"_Alignas"	ALIGNAS
"_Alignof"	ALIGNOF
"_Atomic"	ATOMIC
"_Bool"	BOOL
"_Complex"	COMPLEX
"_Generic"	GENERIC
"_Imaginary"	IMAGINARY
"_Noreturn"	NORETURN
"_Static_assert"	STATIC_ASSERT
"_Thread_local"	THREAD_LOCAL
"__func__"	FUNC_NAME
Maska	**Atslēgvārds**

{L}{A}*	IDENTIFIER
{HP}{H}+{IS}?	I_CONSTANT
{NZ}{D}*{IS}?	I_CONSTANT
"0"{O}*{IS}?	I_CONSTANT
{CP}?"'"([^'\\\n]|{ES})+"'"	I_CONSTANT
{D}+{E}{FS}?	F_CONSTANT
{D}*"."{D}+{E}?{FS}?	F_CONSTANT
{D}+"."{E}?{FS}?	F_CONSTANT
{HP}{H}+{P}{FS}?	F_CONSTANT
{HP}{H}*"."{H}+{P}{FS}?	F_CONSTANT
{HP}{H}+"."{P}{FS}?	F_CONSTANT
({SP}?\"([^"\\\n]|{ES})*\"{WS}*)+	STRING_LITERAL
"..."	ELLIPSIS
">>="	RIGHT_ASSIGN
"<<="	LEFT_ASSIGN
"+="	ADD_ASSIGN
"-="	SUB_ASSIGN
"*="	MUL_ASSIGN
"/="	DIV_ASSIGN
"%="	MOD_ASSIGN
"&="	AND_ASSIGN
"^="	XOR_ASSIGN
"|="	OR_ASSIGN
">>"	RIGHT_OP
"<<"	LEFT_OP
"++"	INC_OP
"--"	DEC_OP
"->"	PTR_OP
"&&"	AND_OP
"||"	OR_OP
"<="	LE_OP
">="	GE_OP
"=="	EQ_OP
"!="	NE_OP
";"	';'
("{"|"<%")	'{'
Maska	**Atslēgvārds**
("}"|"%>")	'}'

","	','
":"	':'
"="	'='
"("	'('
")"	')'
("["|"<:")	'['
("]"|":>")	']'
"."	'.'
"&"	'&'
"!"	'!'
"~"	'~'
"-"	'-'
"+"	'+'
"*"	'*'
"/"	'/'
"%"	'%'
"<"	'<'
">"	'>'
"^"	'^'
"|"	'|'
"?"	'?'
{WS}	/* whitespace separates tokens */
.	/* discard bad characters */

3.2.2 LEX\YACC mijiedarbība

Kā jau minēts LEX tiek bieži izmantots ar YACC. Ja LEX definēt leksiku un pārveido padoto tekstu par atslēgvārdu virkni, tad YACC pārveido atslēgvārdus sakārtotus BNF[ATSAUCE] par autora izveidotaj programmaturai sparotamu datu virkni.

Šīs abas lietotnes rada spēcīgu rīku jebkādu datu apstrādei. Tieši tāpēc tie tika izvēlēti ar nolūku "tulkot" C programmēšanas valodu.

Sākumā LEX pārveido visus ieejas datus par atslēgvārdu virkni. Vēlāk tiek izsaukta YACC procedūra, kura izmantojot BNF [ATSAUCE UZ BNF] tipa likumus mēģina atpazīt kādu vajadzigu informāciju. Tai brīdī, kad tiek sasniegts stāvoklis, kurā var viennozīmīgi pateikt, ka pēdējo atslēgvārdu kopums nes sev līdzi kādu jēgu, tas tiek ielikts ziņojumā priekš pasaules. Pēc ielikšanas iešana cauri atslēgvārdiem turpinās tādā pašā stilā līdz tiek sasniegtas pārveidotu ievaddatu beigas.

Dati tiek apkopoti datu struktūrā [PIELIKUMS]

Apraksts par to kādā veidā LEX atslēgvārdi tiek sagrupēti un sagatavoti ziņojumam Pasaules modulim

Otrā līmeņa virsraksti varētu būt katra moduļa skaidrojums un mērķis. Sekojošie pirmā līmeņa virsraksti apraksta procesa gaitu no ieejas pirmkoda līdz rezultāta izvadei

3.3 Pārveidotājs \ “Transformer”

3.4 Analizētājs \ “Analyser”

3.5 Datubāze \ “Primal”

3.6 Organisms \ “Organism”

3.7 Communicator \ “Communicator”

4 DATU APSTRĀDE\ANALĪZE

Kādā veidā tiek apstrādāti saņemtie dati par struktūrām no LEX\YACC. Datu glabāšanas un salīdzināšanas procesi. Dotās nodaļas mērķis ir rast atbildi kā tiks noteikts ka divi pirmkoda faili ir īstenībā viena un tā paša algoritma realizācija.

Uz doto brīdi ko konkrētu ielikt otrā līmeņa virsrakstos jo dotais solis vēl netika uzprogrammēts. Tas atbilst arī nākošiem pirmā līmeņa virsrakstiem.

5 PROGRAMMAS IZVADA APSTRĀDE

Apraksts par to kāda informācija tiek glabāta, tās izvades veidi. Darbs ar saskarni, kādu nu tā būs tas ir laika jautājums.

6 SALĪDZINĀJUMS AR LĪDZĪGU PORGRAMMATŪRU

Otrā līmeņa virsraksti varētu būt konkrētas programmas ar līdzīgu funkcionalitāti.

7 REZULTĀTI.

Kas ir sanācis un nākotnes plāni attiecībā uz doto darbu.

8 IZMANTOTĀ LITERATŪRA

Informācija par LEX\YACC - LEX & YACC TUTORIAL

Why Source Code Analysis and Manipulation Will Always Be Important

Savu kursa darbs(?)..

OTHER

9 PIELIKUMI

Izejas koda piemēri(?)

Dokumentārā lapa