Programavimo kalbų plėtros mechanizmų tyrimas ir tobulinimas

2015 - 2019

Doktorantas: Audrius Šaikūnas
Darbo vadovas: Albertas Čaplinskas

Tyrimo objektas ir tikslai

Objektas: refleksyviai plečiamos programavimo (REP) kalbos.

Tikslas: sukurti refkelsyviai plečiamos programavimo kalbos kompiliatorių.

Planuojami rezultatai

- ~~Išanalizuoti egzistuojantys sintaksinės analizės algoritmai.~~
- ~~**Naujas sintaksinės analizės algoritmas, kuris yra tinkamas REP kalbų analizei.**~~
- Pavyzdinė REP kalba.
- Pavyzdinė REP kalbos realizacija.
- Tokios kalbos demonstraciniai atvejai (pvz. SQL užklausų, XML konstruktorių integravimas į bazinę kalbą).

Kitų metų darbo planas (senas)

- Sukurti analizės algoritmą, tenkinantį visus REP kalbos analizės reikalavimus
- Aprašyti sukurtą algoritmą disertacijoje
- Sudalyvauti tarptautinėje konferencijoje
- Parašyti straipsnį apie Earley virtualias mašinas (planujamas straipsnis „*Earley Virtual Machine*“)
- Atsiskaityti kursą „Programavimo kalbų teorija“
- Atsiskaityti kursą „Generatyviojo ir aspektinio programavimo metodai“

Apibendriminas & Išvados (senas)

- Nėra REP programavimo kalbos, kuri pilnai tenkintų visus poreikius
- Nėra sintaksinio analizės algoritmo, kuris tiktų REP kalbai analizuoti. Tačiau yra keli geri kandidatai:
  - **RNGLR** – tenkina visus reikalavimus, bet nepalaiko mutuojamų gramatikų (R1).
  - **Yakker** – tenkina visus reikalavimus, bet nepalaiko mutuojamų gramatikų (R1).
  - **APEG** – tenkina visus reikalavimus, bet yra apribota gramatinių plėtinių klasė (R3).

Ataskaita

- Disertacijos teorinė dalis parašyta:
  - Sukurtas ir aprašytas sintaksinės analizės algoritmas tinkantis analizuoti REP kalbas.
- Praeitų metų straipsnis „*Critical Analysis of Extensible Parsing Tools and Techniques*“ išspausdintas į BJMC žurnalą.
- Sudalyvauta tarptautinėje konferencijoje FedCSIS 2017:
  - Parašytas ir pristatytas straipsnis „*Parsing with Earley Virtual Machines*“.
- Atsiskaitytas kursas „*Generatyvinio ir aspektinio programavimo metodai*“ (įvertinimas: **10**)
- Atsiskaitytas kursas „*Programavimo kalbų teorija*“ (įvertinimas: **10**)

Plečiamumas (priminimas)

- **Plečiamos kalbos** – nėra aišku ką reiškia žodis „plečiamos“.

- **Sintaksiškai plečiamos prog. kalbos** – nėra tiksliai įvardintas plėtimo mechanizmas. Plėtimas galimas per:
  - Išorinius įrankius
  - Kompiliatoriaus įskiepius (*plugins*)
  - Specialus meta-kalbų aprašus, kurie pateikiami kompiliatoriui prieš kompiliuojant
  - Makrokomandas

- **Refkelsyviai plečiamos prog. kalbos** (REP) – naujas terminas:
  - Kalba, kurios bekonteksčiai plėtiniai yra aprašomi metakalba.
  - Kalba, kur metakalba ir kalba sutampa.
    - Plėtimas vyksta kompiliavimo metu.
    - Metakalbą ir kalbą galima maišyti viename faile.

Reikalavimai analizatoriui

Pagrindiniai reikalavimai:
- Turi veikti be dedikuoto leksinio analizatoriaus.
- Turi atpažinti visas bekontekstes kalbas.
- Turi palaikyti adaptyvias/refleksyvias gramatikas.
- Turi veikti pakankamai greitai.

Papildomi reikalavimai:
- Turi palaikyti nuo duomenų priklausomus apribojimus.
- Turi palaikyti reguliarius operatorius gramatikose.

Sintaksinė analizė su EVM

- Vartotojas aprašo *pradines gramatikas*.
- EVM transliuoja *pradines gramatikas* į *kompiliuotas gramatikas*, sudarytas iš instrukcijų.
- Instrukcijas vykdo EVM interpretatorius.

- Gramatikos sudarytos iš gramatinių taisyklių.
- Vieną gramatinę taisyklę ties konkrečia įvesties vieta įvykdo viena *gija* (angl. *fiber*).

EVM: Term. simbolių atpažinimas

```
# S → a b c
rule S
  parse "abc"
end
```

```
10: i_match_char 'a' → 11
11: i_match_char 'b' → 12
12: i_match_char 'c' → 13
13: i_reduce "S"
14: i_stop
```

- Atpažinus vieną terminalinį simbolį, dabartinės gijos pozicija pastumiama per 1 simbolį į priekį.

EVM: Alternatyvos

```
# S → A | b
rule S
   parse A | "b"
end
```

```
10: i_fork 12
11: i_call_dyn A
11: i_match_sym A → 13
12: i_match_char 'b' → 13
13: i_reduce "S"
14: i_stop
```

Alternatyvos yra analizuojamos sukuriant dabartinės gijos kopiją su instrukcija `i_fork`.

```
# Q → c+
rule Q
   parse "c"+
end
```

```
10: i_match_char 'c' → 11
11: i_fork 10
12: i_reduce "Q"
13: i_stop
```

EVM: Neterminaliniai simboliai

```
# S → X Y Z
rule S
  parse X Y Z
end
```

```
10: i_call_dyn X
11: i_match_sym X → 12
12: i_call_dyn Y
13: i_match_sym Y → 14
14: i_call_dyn Z
15: i_match_sym Z → 16
16: i_reduce "S"
17: i_stop
```

Vieno neterminalinio simbolio atpažinimo procesas:
1) Gija(-os), kurios atpažįsta simbolio kūną yra sukuriamos.
2) Kviečiančioji gija yra sustabdoma.
3) Naujos gijos (-a) sėkmingai baigia analizę.
4) Kviečiančiojos gijos kopija yra pratęsiama.

EVM: Bendros paskirties skaičiavimai

```
var a = 0
var b = 1
var c = 0
while a < 10
  c = a + b
  a = b
  b = c
end
```

```
10: i_int_push 0
11: i_int_push 1
12: i_int_push 0
13: i_peek 0
14: i_int_push 10
15: i_int_less
16: i_bz 26
17: i_peek 0
18: i_peek 1
19: i_add
20: i_poke 2
21: i_peek 1
22: i_poke 0
23: i_peek 2
24: i_poke 1
25: i_br 13
```

EVM: Nuo duom. priklausomi aprib.

```
rule field(n)
  while n > 0
    parse 'a'
    n = n – 1
  end
end
```

```
10: i_peek 0
11: i_push_int 0
12: i_int_more
13: i_bz 20
14: i_match_char 'a'→15
15: i_peek 0
16: i_push_int 1
17: i_int_sub
18: i_poke 0
19: i_br 10
20: i_reduce "field"
21: i_stop
```

EVM: AST konstravimas

```
rule exp
  parse l:exp '+' r:exp
  return node("plus", l, r)
end
```

```
10: i_call_dyn exp
11: i_match_sym_r exp → 12
12: i_match_char + → 13
13: i_call_dyn exp
14: i_match_sym_r exp → 15
15: i_peek 0
16: i_peek 1
17: i_new_node 2, "plus"
18: i_reduce_r "exp"
19: i_stop
```

AST konstravimo būdai:

- **Rankinis**
- **Automatinis**
- Su atidėtais semantiniais veiksmais.

Trūkumai:

- Spekuliatyvus EVM pobūdis:
  - `2 + 3 * 4`
- Per norus AST šakų konstravimas.

EVM: Atidėti semantiniai veiksmai

Pagrindinė idėja: vietoj AST konstruoti žymių medį, pagal kūrį būtų galima atsekti analizės vykdymo tvarką ir įvykdyti semantinius veiksmus.

```
rule exp
  parse l:exp '+' r:exp
  return node("plus", l, r)
end
```

```
10: i_call_dyn exp
11: i_match_sym exp → 12
12: i_trace 100
13: i_match_char + → 13
14: i_call_dyn exp
15: i_match_sym exp → 15
16: i_trace 101
17: i_reduce "exp"
18: i_stop
```

EVM: Refleksyvios gramatikos

```
domain loop

@domains loop
rule statement
  parse "break"
end

rule while_loop
  parse "while" expr
  with_domains loop
    parse statement+
  end
  parse "end"
end
```

Nauji mechanizmai adaptyvioms gramatikoms realizuoti:

- Domenai.
- Gijos aktyvių domenų aibė.
- Dinaminis kompiliuotų gramatikų užkrovimas.

EVM: Poaibių konstrukcija (1/3)

```
rule exp
  parse exp '+' exp
end

rule exp
  parse exp '-' exp
end
```

```
10: i_call_dyn exp
11: i_match_sym exp -> 12
12: i_match_char '+' -> 13
13: i_call_dyn exp
14: i_match_sym exp -> 15
15: i_reduce "exp"
16: i_stop

20: i_call_dyn exp
21: i_match_sym exp -> 22
22: i_match_char '-' -> 23
23: i_call_dyn exp
24: i_match_sym exp -> 25
25: i_reduce "exp"
26: i_stop
```

EVM: Poaibių konstrukcija (2/3)

```
rule exp
  parse exp '+' exp
end

rule exp
  parse exp '-' exp
end
```

```
10: i_call 40
11: i_match_sym exp -> 12
12: i_match_char
  '+' -> 13, '-' -> 17
13: i_call_dyn exp
14: i_match_sym exp -> 15
15: i_reduce "exp0"
16: i_stop
17: i_call 40
18: i_match_sym exp -> 25
19: i_reduce "exp1"
20: i_stop
```

EVM: Poaibių konstrukcija (3/3)

Instrukcijų poaibių konstrukcijos procesas:
1) Instrukcijų ε-uždarinio skaičiavimas.
2) ε-uždarinio instrukcijų liejimo raktų skaičiavimas.
3) Panašių instrukcijų liejimas.
4) Sekančių instrukcijų poaibių konstrukcija (rekursyvus žingsnis).
5) Rezultato instrukcijų sekos generavimas.

Poaibių konstrukcijos nauda:
- Didesnis kompiliuotų gramatikų determinizmas.
- Dinaminių elementų eliminavimas.

Kitų metų darbo planas

- Aprašyti empirinę disertacijos dalį.
  - Įvertinti sukurtą sintaksinės analizės algoritmą.
  - Sukurti pavyzdinę REP kalbą.
  - Realizuoti pavyzdinę REP kalbą.
- Praplėsti straipsnį "*Parsing with Earley Virtual Machines*", kad jis tiktų žurnalinei publikacijai.
- Sudalyvauti tarptautinėje konferencijoje.

Ačiū už dėmesį!