Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Kristijan Pongerajter

Seminarski rad iz predmeta Prevođenje programskih jezika

Zadatak broj 3022

Zagreb, Siječanj 2010

Seminarski rad iz predmeta Prevođenje programskih jezika

Student: Kristijan Pongerajter
Matični broj: 0036389448
Zadatak broj 3022: Programski ostvariti generator postupka izgradnje jezičnog procesora prema opisu u udženiku. Ulaz u program je skup raspoloživih jezičnih procesora i ciljni jezični procesor, a izlaz je postupak dobivanja ciljnog jezičnog procesora.

Uvod

Software-i za rana računala godinama su pisani u asemblerskim jezicima. Viši programski jezici pojavili su se tek kada je potreba za mogućnošću korištenja software-a na različitim vrstama procesora postala znatno veća od cijene izrade samih prevoditelja pri čijoj su implementaciji tehničke probleme stvarali i vrlo mali memorijski kapaciteti ranih računala. Od sedamdesetih godina dvadesetog stoljeća, implementacija prevodioca u jezik koji prevodi postala je česta praksa čime je otvorena nova etapa razvoja jezičnih procesora.

Pojam jezičnog procesora podrazumijeva računalni program koji čita program napisan u izvornom jeziku, te ga prevodi u istovjetni program u ciljnom (najčešće strojnom) jeziku. Formalna se definicija jezičnog prevoditelja zasniva na tri jezika: izvorni jezik, ciljni jezik i jezik izgradnje, te se jezični prevoditelj prikazuje na sljedeći način:

$$JP_{L_g}^{L_i \rightarrow L_c}$$

gdje je JP jezični procesor, L_i izvorni jezik, L_c ciljni jezik, te L_g jezik izgradnje.

Ostvarenje

Riješenje problema generatora izgradnje jezičnog procesora ostvareno je u programskom jeziku C#. Kako je sam proces izgradnje jezičnog procesora korištenjem postojećih (dostupnih) jezičnih procesora jednoznačno određen pravilima prevođenja i ulančavanja, korišteni algoritam nametnuo se kao logično rješenje. Algoritam razlikuje više programske jezike (A-Z) od strojnih jezika (a-z) i na temelju raspoloživih jezičnih procesora započinje analizu.

Prvi korak rada algoritma jest stvaranje liste raspoloživih jezičnih procesora pri čemu se svi članovi liste uspoređuju sa ciljnim jezičnim procesorom kako bi se odredilo je li uopće potrebno izgraditi ciljni jezični procesor ili isti već postoji u skupu unesenih raspoloživih procesora. Ukoliko takav jezični procesor već postoji, program ga ispisuje kao izlaz i završava sa radom. Ukoliko pak ciljni jezični procesor nije u skupu raspoloživih jezičnih procesora, algoritam kreće u proces analiziranja svih parova dostupnih jezičnih procesora u svrhu pokušaja izgradnje novih jezičnih procesora bazirajući se na dvije osnove metode ulančavanja i prevođenja.

Ukoliko postoje jezični procesori JP1 i JP2 jednakog jezika izgradnje, pri čemu je ciljni jezik jezičnog procesora JP1 jednak izvornom jeziku jezičnog procesora JP2, logičnom akcijom ulančavanja dolazi se do novog jezičnog procesora JP3 koji prevodi izvorni jezik procesora JP1 u ciljni jezik jezičnog procesora JP2. Novi jezični procesor JP3 dodaje se u listu kandidata novih raspoloživih jezika, pri čemu se pamte jezični procesori JP1 i JP2 kao jezični procesori izgradnje jezičnog procesora JP3.

Primjer:

$$JP_X^{X \to Y} + JP_X^{Y \to Z} = JP_X^{X \to Z}$$

Ukoliko se uspoređuju jezični procesori različitih jezika izgradnje pri čemu je JP1 izgrađen na strojnom jeziku, a JP2 na višem programskom jeziku, algoritam uspoređuje jezik izgradnje jezičnog procesora JP2 (viši programski jezik) sa izvornim jezikom jezičnog procesora JP1. Ukoliko je riječ o istim jezicima, izgradit će se novi jezični procesor JP3 jezika izgradnje jednakog ciljnom jeziku jezičnog procesora JP1, a izvornog i ciljnog jezika jednakih izvornom i ciljnom jeziku jezika JP2. Novi jezični procesor dodaje se u listu kandidata novih raspoloživih jezika, pri čemu se pamte jezični procesori JP1 i JP2 kao jezični procesori izgradnje jezičnog procesora JP3.

Primjer:

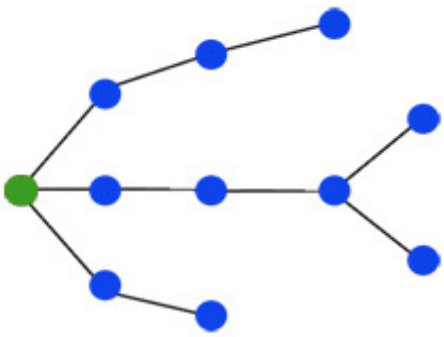
$$JP_X^{Y \to Z} + JP_a^{X \to Z} = JP_Z^{Y \to Z}$$

Kada se prethodno opisanim dvjema procedurama usporede svi trenutno raspoloživi jezični procesori, svi novoizgrađeni jezični procesori (koji se trenutno nalaze na listi kandidata raspoloživih jezičnih procesora) dodaju se na listu raspoloživih jezika. Sada se svi jezični procesori sa liste raspoloživih uspoređuju sa ciljnim jezičnim procesorom. Ovime je završen jedan ciklus rada algoritma koji se ponavlja do trenutka uspješnog pronalaska ciljnog jezičnog procesora u listi trenutno raspoloživih jezičnih procesora pri čemu program ispisuje korake izgradnje ciljnog procesora i završava s radom. Program se također zaustavlja ukoliko u nekom ciklusu rada algoritma niti jedan jezični procesor ne dospije na listu kandidata što znači da su iscrpljene sve mogućnosti izgradnje novih jezičnih procesora, a kako u prijašnjim koracima nije pronađen jezični procesor istovjetan ciljnom jezičnom procesoru, logično je zaključiti kako iz raspoloživih jezičnih procesora nije moguće dobiti traženi, ciljni jezični procesor.

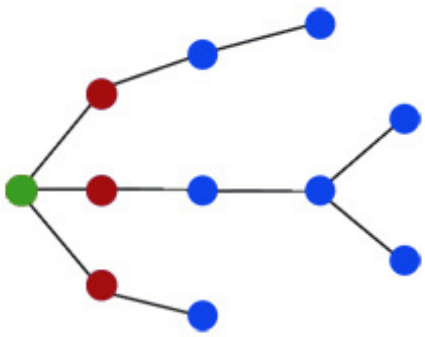
Opisani algoritam napreduje metodom pretraživanja po širini programskog stabla. Ovakav pristup razvoju programa odabran je zbog jednostavnosti praćenja napretka samog algoritma što je iznimno važno budući se u izlazu programa ispisuje cjelokupni postupak izgradnje ciljnog jezičnog procesora. Nadalje, ovakav pristup jamči prolazak kroz programsko stablo u najkraćem broju ciklusa.

***Skica obrade programskog stabla po širini**:

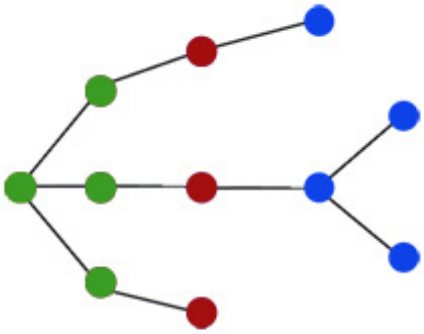
1.



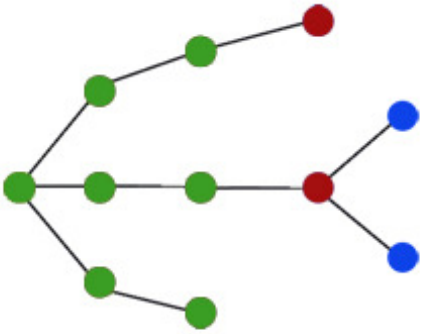
2.



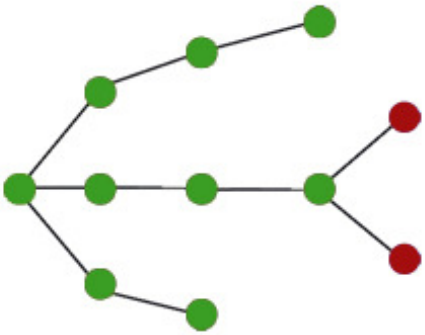
3.



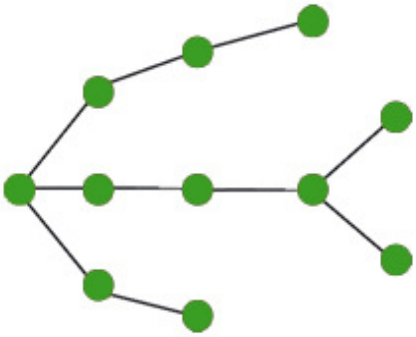
4.



5.



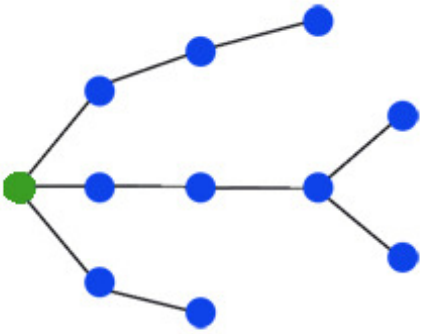
6.



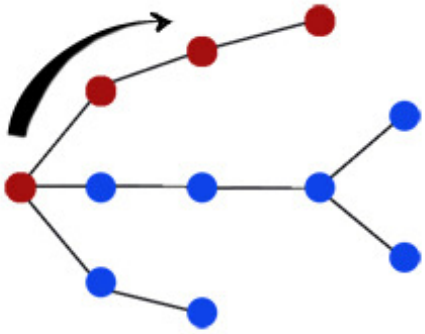
Alternativu opisanom algoritmu predstavlja prolazak programskog stabla po dubini. Takav pristup ima prednosti u vidu korištenja memorijskih resursa kao i brzine pronalaska ciljnog jezičnog procesora, budući je moguće da algoritam pronađe rješenje već u prvom prolasku kroz stablo pri čemu nije utrošeno vrijeme na obradu eventualno neiskoristivih jezičnih procesora.

***Skica obrade programskog stabla po dubini**:

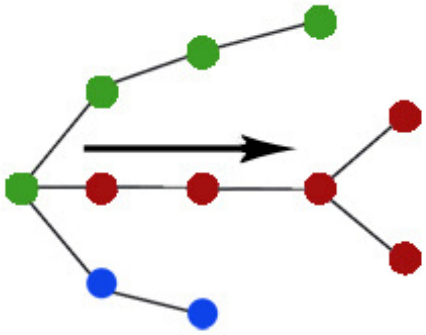
1.



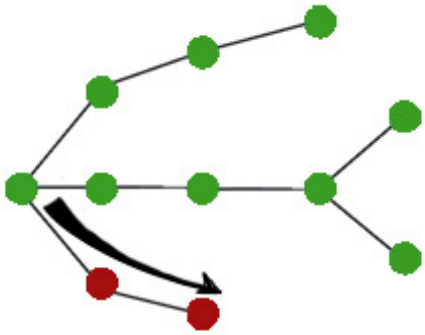
2.



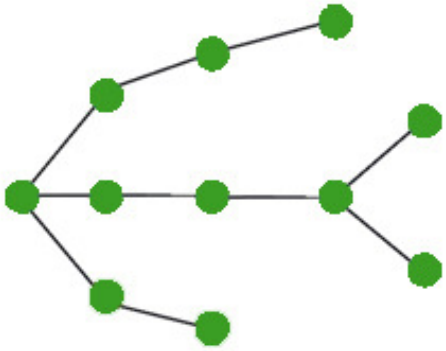
3.



4.



5.



***Primjer ispisa**:

```
X -> C in a
  X -> P in a
    X -> K in a
      X -> Y in a
      Y -> K in a
        Y -> Z in a
        Z -> K in a
    K -> P in a
      K -> P in C
      C -> a in a [exec]
        C -> X in a
        X -> a in a
  P -> C in a
```

Primjer pokazuje postupak dobivanja jezičnog procesora $JP_a^{X \to C}$ (X -> C in a). Ciljni jezični procesor izgrađen je ulančavanjem jezičnih procesora (X -> P in a) i (P -> C in a), jezični procesor (X -> P in a) izgrađen je ulančavanjem jezičnih procesora (X –> K in a) i (K -> P in a) itd... Jezični procesor (C -> a in a) ima oznaku [exec] koja ga označava kao prevodioca jezika izgradnje jezičnog procesora (K -> P in C) čime je izgrađen jezični procesor (K -> P in a).

Pri pokretanju programa korisnik može odabrati između unosa sa tipkovnice, čitanja iz datoteke ili pokretanja predefiniranog unosa (primjer) čiji izlaz je prikazan na gornjoj slici.

Zaključak

Programsko ostvarenje odlično funkcionira kao demonstracijski program generičkog pristupa rješavanja problema izgradnje ciljnog jezičnog procesora iz skupa raspoloživih jezičnih procesora. Glavni nedostatak programa jest činjenica da je ostvaren kao konzolna aplikacija, što je prilično ograničavajući faktor područja same estetike. Ukoliko bi se sličan program radio u obrazovne svrhe, svakako bi trebalo napraviti korisnički intuitivan GUI koji bi pojednostavnio sam proces unosa ulaznih parametara (raspoloživi jezični procesori i ciljni jezični procesor) i slikovitije prikazao cjelokupni proces izgradnje ciljnog jezičnog procesora.