Robert Čordaš

PrologAPI

PrologAPI se sastoji od Prolog interpretera i interfejsa koji omogućava korišćenje Prolog konstrukcija iz programskog jezika C++. Moguće je pozivanje Prolog upita kao C++ funkcija i obratno, C++ funkcije se mogu pozvati iz Prolog programa. Prolog izvorni kod se prosleđuje kao običan tekst. Kao argumente Prolog upiti mogu da prihvate tip podataka string, cele brojeve, logičke vrednosti, realne brojeve, nizove i strukture. Ceo interpreter je smešten u jedan DLL u Windows-u ili SO (shared object) na Linuxu i može se koristiti iz bilo kog C++ okruženja. Interpreter je i napisan u Object Pascal-u (Delphi, Kylix, FPC), a interfejs u C++.

Programski jezik Prolog i Prolog API

Prolog je deklarativni programski jezik - funkcioniše na potpuno drugačiji način nego proceduralni jezici. Programiranje u Prologu se svodi na opisivanje problema logičkim rečenicama. Prolog interpreter ima ugrađene mehanizme kojima ih može izračunavati.

Logičke rečenice mogu biti činjenice i pravila. Činjenice su izjave koje opisuju osobine nekog objekta. Pravila su postupci kojima se opisuju osobine nekog objekta. Promenljive se pišu sa velikim početnim slovom, a vrednost im se može dodeliti samo jednom. Prologu je moguće postavljati pitanja. Pitanja su tačna ukoliko odgovaraju nekoj od činjenica, ili ukoliko se mehanizmom izvođenja, koji je zasnovan na principu unifikacije i rezolucije, mogu dobiti iz pravila, primenom na postojeće činjenice.

Cilj ovog projekta je da olakša pisanje programa koji ne mogu celokupno da se implementiraju u Prologu ili je neke od delova programa jednostavnije implementirati u proceduralnim programskim jezicima. Tipični primeri su programi koji koriste veštačku inteligenciju. To su, na primer ekspertski sistemi koji iz nekih razloga treba da imaju grafički korisnički interfejs, ili ako je ekspertski sistem samo deo nekog većeg projekta koji se teže programira u Prologu nego u nekom od proceduralnih programskih jezika. Drugi primeri su igrice. Prolog ne podržava 3D okruženje, pa igra mora da se piše iz nekog proceduralnog jezika, što otežava pisanje veštačke inteligencije za botove. PrologAPI omogućava povezivanje bilo kog proceduralnog jezika sa Prologom.

PrologAPI radi na Windows i Linux operativnom sistemu. Interpreter uspešno konvertuje sve važnije C++ tipove podataka, osim klasa, koje je teško implementirati, jer u Prologu ne postoji koncept klasa kao u proceduralnim jezicima. Prolog objekti ne mogu sadržati metode i neki od metoda se ne mogu kao takvi napisati u Prologu (npr. brojač, koji kod svakog poziva povećava vrednost promenljive, pošto su promenljive već vezane). PrologAPI može interpretirati svaki Prolog program koji je kompatibilan sa Arity/Prologom, ali podržava manje ugrađenih predikata.

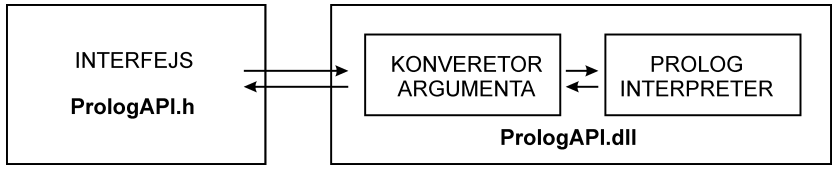
Izrada

Projekat je implementiran korišćenjem više alata. Za pisanje Prolog interpretera je korišćen Delphi za Windows i FPC (Free Pascal Compiler) na Linuxu (Fedora 4), a za izradu i testiranje interfejsa Microsoft Visual Studio na Windowsu i g++ (deo GCC-a) na Linuxu.

Najveći problem u ovom projektu predstavljala je konverzija između tipova podataka proceduralnog jezika (C++) i Prologa. Interpreter je tako dizajniran da lako može konvertovati tipove podataka iz proceduralnog jezika u argumente Prolog upita.

Robert Čordaš (1989), Mužlja, Doža Đerđa 25, učenik 1. razreda Zrenjaninske gimnazije

MENTOR:
Miloš Savić, student PMF u Novom Sadu, Departman za matematiku i informatiku



PrologAPI se sastoji od tri jasno odvojene celine:
1. Prolog interpreter koji izvršava programe
2. Deo za konverziju argumenata
3. Interfejs

Struktura PrologAPIa

Glavni deo interfejsa – prenos podataka

Glavni deo interfejsa omogućava lako korišćenje PrologAPIja. Konverzija podataka se vrši u dva koraka:

1. Konverzija u međutip – svaki programski jezik drugačije čuva promenljive, a potrebno je omogućiti univerzalnost prenosa parametara

2. Interna konverzija međutipova u Prolog tokene koje interpreter može da obrađuje

Međutip je realizovan kao jedna stuktura sa sledećim podacima:

1. pokazivač na vrednost promenljive
2. tip promenljive
3. da li je promenljiva prazna, tj. da li Prolog interpreter traži vrednost te promenljive
4. ako je u pitanju niz, veličina niza
5. oznaka da li da SE briše vrednost posle završetka funkcije.

Ako je promenljiva prazna treće polje treba da bude *true*. U ovom slučaju Prolog može da upiše (vrati) neku vrednost u tu promenljivu. Nizovi su predstavljeni pokazivačem na početak niza i brojem argumenata.

Prenos strukture je sličan prenosu promenljivih koji su prostog tipa (u daljem tekstu proste promenljive). Prvo se izračunavaju adrese svakog polja u stukturi. Ove adrese se prenose kao da su adrese prostih promenljivih. Osim toga prenosi se i ime strukture. Pointer za polje izračunava se pomoću većine Align-a i veličine svakog polja. Svako polje strukture pokazuje na drugu strukturu koja sadrži ime i vrednost promenljivih.

Algoritam za pronalažEnje adrese polja, u pseudo jeziku sličnom C-u dat je na dnu ove stranice. Ova funkcija za argument traži adresu strukture, broj polja i niz tipa polja. Vraća niz pokazivača na svako polje.

Konvertor argumenata

Ovaj deo je najvažniji za povezivanje Prologa sa proceduralnim jezicima. On prekonvertovane argumente pretvara u Prolog tokene, koje interpreter može koristiti. Promenljive se konvertuju direktno u tokene, a prazne promenljive se prvo dodaju kontroleru promenljivih, a ime promenljive se upisuje u to-

```
void** GetAddrs(void* thestruct, int argc, TVarType* types){
          char * bitmap=(char*) thestruct;

          void** result=new void*[argc];

         int currpos=0;
         for (int a=0; aargc; a++){
                    datasize=GetSizeInBytes(types[a]);
                    int currpage=currpos / ALIGN;
                    int nextpage=(currpos+datasize) / ALIGN;
                    if (currpage!=nextpage && currpage % ALIGN!=0){
                              currpos= (currpage+1)*ALIGN;
                    }

                    *result[a]=&bitmap[a];
          }
          return result;
```

ken. Prolog promenljive i prekonvertovane promenljive se zatim smeštaju u jednu stukturu koja se dodaje u listu. Posle interpretiranja Prolog upita, ova lista se koristi da bi se Prolog promenljive kopirale nazad u pre-konvertovane promenljive.

Prolog interpreter

Prolog interpreter je najvažniji deo PrologAPIja. Dizajniran je tako da radi sa takvim tokenima u koje je lako konvertovati argumente proceduralnih jezika. Činjenice su interno realizovane kao funkcije, tj. mogu se jednostavno pozivati i imaju svoju bazu promenljivih.

Interpreter se sastoji od više delova:

1. kontroler tokena
2. kontroler poziva funkcija (činjenica)
3. kontroler operatora
4. baza promenljivih (za svaku činjenicu po jedna)
5. baza za backtracking
6. glavni deo koji kontroliše celokupan rad

Kontroler tokena od izvornog koda generiše tokene po utvrđenim pravilima. Svi operatori su integrisani u izvorni kod i po tome kontroler tokena zna da generiše stablo zavisnosti od koda. Svaka činjenica, operator ili upit je posebna grana ovog stabla. Svaki token ima svoj tip, vrednost i pod-tokene. Vrednost tokena je običan string.

Kontroler poziva funkcija čuva u jednoj bazi sve činjenice, pravila i ugrađene predikate. Svako pitanje prvo stigne do kontrolera, zatim ga on izvrši i vodi računa da svako pitanje koji se može i na drugačiji način zadovoljiti, bude upisano u bazu za backtracking. Druga važna uloga mu je, da u slučaju padanja cilja, nastavi backtracking, tj. šalje komandu kontroleru za backtracking da oslobodi varijable i da pozove glavni deo za interpretiranje koda. Baza je realizovana kao binarno-pretraživajuća lista.

Zadatak kontrolera operatora je da izvrši sve operacije nad promenljivima. Druga uloga mu je da ih zapamti u bazi za backtracking da bi mogle biti oslobođene u slučaju padanja nekog cilja (da bi se backtracking mogao nastaviti). Kod izračunavanja operatora, ako operandi nisu istog tipa, prvo pokušava da ih konvertuje na tip koji je strože određen, a ako to ne uspe odradi obratno. Funkcije koje izračunavaju rezultate operacija su smešteni u jedan konstantni niz operatora i funkcija. U sledećoj fazi kontroler zadataka iz ovog niza potraži odgovarajuću funkciju, izvrši je i tako dobije rezultat.

Kontroler promeljivih radi sve što je vezano za čuvanje promenljivih. Glavni zadatak mu je da pronađe promenljivu po imenu i da automatski konvertuje različite tipove. Osim toga vodi računa i o dodavanju nove promenljive u bazu za backtracking. Tabela simbola je takođe realizovana kao binarno-pretraživajuća lista.

Baza za backtracking je lista čija polja sadrže podatke o pozvanoj funkciji, prazne promenljive na tom mestu i poziciju sledeće naredbe nakon što je obrada činjenica završena. Pre backtracking-a on oslobađa sve nove promenljive, da bi im se mogla ponovo dodeliti neka vrednost.

Glavni deo ima najviše zadataka. Sekvencijalno prelazi kroz listu tokena, proverava tip tokena i po tome poziva odgovarajuće funkcije iz kontrolera operatora ili funkcija. U slučaju da interpreter ne može nešto da izračuna pomoću datih argumenata (cilj padne), on nastavlja backtracking preko kontrolera funkcija. Ako nađe neki upit ova funkcija se poziva rekurzivno.

Korišćenje PrologAPIja

Interfejs se sastoji od klase pisane u C++-u i još nekoliko funkcija koje omogućavaju konverziju argumenata. Sama klasa ima sledeće metode:

- Konstruktor, koji automatski kreira novu instancu Prolog interpretera
- Procedura AddLine, koji dodaje liniju Prolog source koda interpreteru
- Procedura AddString, koji dodaje tekst interpreteru
- Procedura Run, koji se koristi za pokretanje Prolog programa
- Procedura CallFunction koja se koristi za pozivanje Prolog upita

PrologAPI takođe omogućava pozivanje funkcija napisanih u C++ iz Prologa. Funkcije moraju imati *cdecl* konverziju poziva. Interpreteru treba eksplicitno naznačiti koje C++ funkcije može pozivati.

Rezultati i diskusija

PrologAPI uspešno radi sa prostim tipovima podataka, nizovima i strukturama. Prepoznaje strukture sa bilo kakvim kombinacijama jednostavnih tipova podataka. Do sada ima malo ugrađenih funkcija, ali

je sasvim dovoljvo za implementaciju standardnih algoritama.

Prenos objekata bi se mogao napraviti pomoću pozivanja C++ funkcija iz Prologa. To je neophodno jer se ne može svaka C++ funkcija prevesti u ekvivalentnu konstrukciju u Prologu. Objekti bi se prenosili kao obične strukture, atributi bi se prenosili kao proste promenljive, a metode bi imale osim argumenata jedan pokazivač na samu strukturu. Promenljive korišćene u objektu bi se mogle konvertovati kao obični tipovi, i adresa bi im se mogla odrediti pomoću RTTI (Run Time Type Information).

Slični i najpoznatiji projekti koji odgovaraju problematici izloženoj u ovom radu su Amzi! C++ API (www.amzi.com; http://www.amzi.com/articles/lsapi_design.html)
i C++ interfejs za SWI-Prolog (http://www.swi-prolog.org/packages/pl2cpp.htm), ali nemaju podršku za prenos i konverziju strukture, samo za promenljive prostih tipova. Amzi! je realizovan, kao logički server. Prilično je težak za korišćenje. C++ interfejs za SWI-Prolog radi u obrnutom smeru, tj. omogućava pozivanja C++ koda iz Prologa.

Zaključak

PrologAPI olakšava rešavanje zadataka čiji su delovi jednostavni za implementaciju u Prologu, dok se ostatak zadatka implementira u nekom od proceduralnih programskih jezika. Jednostavan je za korišćenje i radi dovoljno stabilno, ali još nema savršenu kontrolu za prepoznavanje grešaka.

Mogućnost daljeg razvoja se kreće u smeru izrade mehanizama za prenos objekata i doimplementacija funkcija da bi interpreter bio potpuno kompatibilan sa Arity/Prologom, Turbo i Strawbery Prologom.

Literatura

Vavić B. 1988. *Prolog – osnove programiranja*. Požarevac

Robert Čordaš

Prolog API

PrologAPI consists of the Prolog interpreter and the application interface which enables usage of Prolog constructs from C++. It is possible to call Prolog queries from C++ as functions, and call C++ functions from Prolog. Prolog source code is transferred as plain text. Arguments which are passed could be of type String, Integer, Boolean, Real, Array and Record. The whole Prolog interpreter is compiled into one DLL (for Microsoft Windows) or one .so (shared object – for Linux) file and it is possible to use it from any C++ compiler. The interpreter is completely implemented in Object Pascal (Delphi, Kylix, FPC) while the interface is implemented in C++.