Prolog rendszerek külső nyelvi interfészei

Biener Péter
bp131@hszk.bme.hu

Van, amire a Prolog nem a legalkalmasabb

- Hatékonyság
- Más programnyelvek kiegészítő szolgáltatásai
- Kevés Prologhoz értő szakember (drágább fejlesztés és karbantartás)
- Nem „zöldmezős” fejlesztések
  - szeretnénk a meglévő kód minél nagyobb részét megtartani
  - ha ezek nem Prologban íródtak, az új programrészek hozzáillesztéséhez eszközök kellenek

Különböző nyelvű programok együttműködése

- Nincs olyan nyelv, ami – akár egy rendszeren belül is – minden feladatra a legalkalmasabb
- Igények szerteágazóak, rengeteg különböző típusú funkció
- Nem megfelelő nyelv választása egy adott feladathoz növeli a fejlesztési- és karbantartási költségeket, és mindemellett a végeredmény sem fog a kívánt hatásfokkal működni
- Érdemes minden részfeladathoz vagy modulhoz külön kiválasztani a legalkalmasabb nyelvet
- Ezeknek a programrészeknek legtöbbször valamilyen szinten együtt kell működniük
- Ehhez szükség van olyan interfészekre, ami a különböző programnyelven íródott modulok között biztosítanak hívási felületet
- Elterjedt Prolog rendszerek tartalmaznak eszközöket az összekapcsoláshoz (elengedhetetlen a sikerhez)

A Prolog külső nyelvi kapcsolatai

- Elterjedt imperatív nyelvek és a Prolog közti intefészekről lesz szó
- Az egyik nyelvből meg tudjuk hívni a másik oldal egy eljárását/függvényét/predikátumát
- Adatokat kell átadni a két oldal között (pl. a C felé):
  - Term típust kap a függvény: meg lehet kérdezni, hogy egész-e, el lehet kérni az értékét, stb.
  - Előre specifikált típust kap (pl. int): a konverzióról, ellenőrzésről a „glue code” gondoskodik

Egy interfész további jellemzői

- Adatok leképezése: „glue code”-dal kell-e foglalkozni
- Szemlélet:
  - procedurális: Az adott (pl. C) függvény az argumentumként kapott változóban szolgáltatja az eredményt. A visszatérési értékben lehet a sikerességet/hibát jelezni.
  - funkcionális: A függvény a visszatérési értékében szolgáltatja az eredményt. Hibát (amennyiben nincsenek kivételek az adott nyelvben) lehet valamilyen érvénytelen értékkel jelezni.
- Termek létrehozása alulról felfelé, vagy felülről lefelé?
- Hívási irányok
- Kivételkezelés
- Fordítás, szerkesztés, futtatás

Kényelmi szolgáltatások

- Nemdeterminizmus az imperatív oldalon
- Megoldható egy burkoló-predikátum segítségével

```
% pred(Param, St0, St1): C-ben van megvalósítva

nd_pred(Param):-
   nd_pred(Param, initial).

nd_pred(Param, St0):-
   pred(Param, St0, St1),
   ( S1 == completed -> true      % nincs több megoldás
   ; true                         % még újrahívható
   ; nd_pred(Param, St1)          % visszalépéskor az új
   ).                             % állapottal hívjuk
```

C interfészek

- Szinte minden Prolog rendszerhez létezik
- GNU, SWI, SICStus interfészeiről lesz szó
- Közös képességek/lehetőségek:
  - két irányú hívások lehetősége, korlátlan mélységben egymásba ágyazva
  - Prolog felé irányuló (nemdeterminisztikus) hívások esetén az eredmények egyesével való visszaadása
  - C-ből Prolog kivétel dobása, Prologban keletkezett kivétel fogadása C oldalon
  - Termek kezelése: azonosítóval (egész szám), pl. egy belső - Prolog verembe mutató mutatókat tartalmazó – tömb indexe (SICStus)
  - Atomok kezelése: atomtáblába mutató tömbindexszel (hatékonyság). Karaktertömbként elkérve az atomot át lehet írni az atomtáblát (!), mivel nem készít másolatot a rendszer. Erre vigyázni kell.

A GNU Prolog C interfésze

- Funkcionális megközelítés: a függvények visszatérési értéke többnyire maga az eredmény
- „Glue code” automatikusan generálható
- Term létrehozásakor a típust meg kell adni. Minden típushoz külön létrehozó függvény tartozik.
- Lehet nemdeterminisztikus predikátumot C-ben írni
  - Fordítási időben meg kell adni az állapot tárolásához szükséges hely nagyságát (gépi szó)
  - `Get_Choice_Buffer()`: az állapotot tároló címet adja vissza
  - `Get_Choice_Counter()`: a visszalépések eddigi száma
  - `No_More_Choice()`: választási pont törlése

A GNU Prolog C interfésze II.

- Megadható a paraméterek módja:
  - bemenő: érték szerinti paraméterátadás (pl. `int`). A behelyettesítettséget, a típust és a tartományt (pl. pozitív egész) a rendszer ellenőrzi, és adott esetben kivételt dob.
  - kimenő: cím szerinti paraméterátadás (pl. `int*`)
  - mindkettő egyszerre (kényelmi szolgáltatás): egy előre definiált struktúra címét kapja meg a függvény.

```
typedef struct
{
  Bool is_var; /* híváskor változó volt-e */
  Bool unify;  /* visszatéréskor egyesítsen-e */
  union        /* a típus a specifikációból derül ki */
  {
    long l;    /* egész v. term (azonosító) */
    char *s;
    double d;
  } value;
} FIOArg;
```


Példa: one_dif/2

- `one_dif(X, Y)`: X és Y két egész szám, különbségük abszolútértéke 1. Két bemenő argumentummal meghívva ellenőrző jellegű predikátum, egy bemenő argumentummal meghívva visszaadja a szomszédos egészeket, két változóval meghívva kivételt dob.
- Prolog kód:

```
:- foreign(one_dif(?integer, ?integer), [choice_size(1)]).
```

- C kód (eleje):

```
#include "gprolog.h"
Bool one_dif(FIOArg *x, FIOArg *y)
{
    long *state;
    /* az állapot-mutató elkérése */
    state = Get_Choice_Buffer(long*);
    if (Get_Choice_Counter() == 0)      /* első végrehajtás */
    {
        *state = -1;
    }
```



```
    if (!x->is_var)             /* x nem változó */
        if (!y->is_var)         /* y nem változó */
        {
            No_More_Choice();   /* nem kell választási pont */
            /* siker, ha a különbség 1, egyébként meghiúsulás */
            return ((x->value.l - y->value.l) == -1 ||
                    (x->value.l - y->value.l) == 1);
        }
        else                    /* y változó */
        {
            y->unify = TRUE;    /* y kimenő argumentum lesz */
            y->value.l = x->value.l + *state;
            if (*state == 1)
                No_More_Choice();   /* nincs több megoldás */
            else
                (*state = 1);
            return TRUE;        /* siker */
        }
    else                        /* x változó */
        if (!y->is_var)         /* y nem változó */
        {
            ...
        }
        else                    /* mindkét argumentum változó */
            Pl_Err_Instantiation();     /* kivétel dobás */

    return FALSE;   /* ide nem kerülhet a vezérlés */
}
```


A GNU Prolog C interfésze III.

- Prolog felé irányuló hívás:
  - `Pl_Query_Begin()`: előkészítés
  - `Pl_Query_Call()`: név, aritás, argumentumok tömbje paraméterek. Végrehajtása az első megoldást is szolgáltatja.
  - `Pl_Query_Next_Solution()`: a többi megoldáshoz. Visszatérési értéke: siker, meghiúsulás vagy kivétel.
  - `Pl_Get_Exception()`: a kivétel term elkéréséhez
  - `Pl_Query_End()`: befejezés a megadott módon. Lehet teljes lezárás, vágó vagy nyitva hagyás. Utóbbi nemdeterminisztikus függvény írásakor lehet hasznos.
- Nincs lekérdezés-azonosító, minden parancs a legbelső megnyitott lekérdezésre vonatkozik.

A GNU Prolog C interfésze IV.

- Beépített Prolog kivételek kiváltására vannak függvények: pl. `Pl_Err_Instantiation()`.
- Egyedi kivételek kiváltása a `Pl_Exec_Continuation()` függvénnyel. Ez lecseréli az aktuális célsorozatot a megadott predikátumra (pl. `throw/1`). Ekkor a program a C-ben futó predikátumból kilép, és a végrehajtás a megadott predikátummal folytatódik.
- Csak statikus szerkesztés: a Prolog kód és a C kód egyetlen futtatható állománnyá áll össze.

13

Az SWI Prolog C interfésze

- (többnyire) Procedurális szemléletű: az eredmények egy már létrehozott, paraméterként átadott tárolóhelyre kerülnek
- Nincs „glue code”, mivel nincs automatikus típuskonverzió: a Prologból hívott C függvények minden paramétere term típusú.
- Minden term a létrehozásakor változó lesz. Az adattal való feltöltés történhet egyesítéssel, illetve a tárolóhely átírásával is (pl. `PL_put_integer()`).
- Van külön `string` típus, ami tartalmazhat 0 kódértékű karaktereket is (SICStus-ban ez karakterkódok listájával oldható meg).

14

Az SWI Prolog C interfésze II.

- Van nemdeterminisztikus C függvény (némileg hasonlóan a GNU interfészéhez).
- Prolog felé irányuló hívás:
  - `PL_open_query()`: lekérdezés megnyitása modul-azonosító, predikátum-azonosító és argumentumok alapján. Eredmény egy lekérdezés azonosító. Az argumentumoknak összefüggő memóriaterületen kell lenni. Létrehozás a `PL_new_term_refs` függvény segítségével.
  - `PL_next_solution()`: első, vagy következő megoldás kérése (siker esetén TRUE, meghiúsulás és kivétel esetén FALSE visszatérési érték)
  - `PL_cut_query()`: vágó
  - `PL_close_query()`: lezárás
  - `PL_Exception()`: visszatérési értéke NULL, ha nem volt kivétel, egyébként a kivétel-term.
  - Nincs egymásba ágyazott lekérdezés! (komoly hiányosság)

15

Az SWI Prolog C interfésze III.

- C oldalon tetszőleges Prolog kivétel kiváltható: `PL_raise_exception()` függvény.
- Statikus és dinamikus szerkesztésre is van lehetőség

16

A SICStus Prolog C interfésze

- Glue code automatikusan generálható
- Procedurális szemlélet: a függvények a visszatérési értékükben a sikerességet jelzik, az eredmény a paraméterben átadott változóba kerül
- Term létrehozáskor üres lista kezdőértékű (valószínűleg ez a legolcsóbb). „Feltöltésre” külön függvények, pl. `SP_put_integer`.
- Összetett termek létrehozásakor az argumentumokat változó argumentumszámú függvénynek kell átadni. Lehet csupa változó argumentummal is (mint a `functor/3`).
- Saját memóriakezelő függvények a rendszer által kezelt erőforrások hatékonyabb kihasználásához.

17

A SICStus Prolog C interfésze II.

- Nincs nemdeterminisztikus C függvény (nem olyan nagy probléma)
- Az argumentumok lehetnek bemenő (érték szerinti átadás) illetve kimenő (cím szerinti) típusúak. Utóbbi esetben visszatérés után egyesítés lesz.
- Prolog stream-ek C-ből történő írása/olvasása
- Karaktertömbből termet lehet csinálni, a term változóit (mélységi előfordulási sorrendben) is megkaphatjuk egy tömbben
- Nagy egészek bájtok tömbjeként kezelhetők C oldalon.

18

A SICStus Prolog C interfésze III.

- Prolog felé irányuló hívás:
  - `SP_open_query()`: lekérdezés megnyitása predikátum-azonosító és argumentumok alapján. Eredmény egy lekérdezés azonosító. Az argumentumok ezen változó argumentumszámú függvény paraméterei.
  - `SP_next_solution()`: következő megoldás kérése (visszatérési érték: siker, meghiúsulás, kivétel)
  - `SP_cut_query()`: vágó
  - `SP_close_query()`: lezárás
  - `SP_get_exception_term()`: a kivétel-term elkérésére
  - Tetszőleges mélységben egymásba egymásba ágyazhatóak a lekérdezések
- C oldalon tetszőleges Prolog kivétel kiváltható (`SP_raise_exception`).
- Statikus és a dinamikus szerkesztés is van.
- Inicializációs és deinicializációs kódok: a C kód dinamikus betöltésekor és eltávolításakor futnak le (pl. file-ok megnyitását/lezárását végezhetik).

19

Példa: m_sum/2

- `m_sum(M,S)`: Az `M` egész számokból álló mátrix elemeinek az összege `S`.
- Prolog kód:

```
foreign(msum, m_sum(+term, [-integer])).
foreign_resource(ex, [msum]).
:- load_foreign_resource(ex).
:- use_module(library(lists)).
```

20

```
/* ex.c */
#include <sicstus/sicstus.h>

long msum(SP_term_ref l)
{
    SP_term_ref t1 = SP_new_term_ref();
    SP_term_ref t2 = SP_new_term_ref();
    SP_pred_ref mp = SP_predicate("member", 2, "lists");
    SP_qid q1, q2;
    long i, result = 0;
    SP_put_variable(t1);
    SP_put_variable(t2);
    q1 = SP_open_query(mp, t1, l);
    while (SP_next_solution(q1) == SP_SUCCESS)
    {
        q2 = SP_open_query(mp, t2, t1);
        while (SP_next_solution(q2) == SP_SUCCESS)
        {
            SP_get_integer(t2, &i);
            result += i;
        }
        SP_close_query(q2);
    }
    SP_close_query(q1);
    return result;
}
```

21

Példa2: list_filter/2

- `list_filter(L,I)`: Az `L` lista egész típusú elemeinek listája `I`.
- Prolog kód:

```
foreign(list_filter, list_filter(+term, [-term])).
foreign_resource(ex2, [list_filter]).
:- load_foreign_resource(ex2).
```

22

```
/* ex2.c */

/* A list egész elemeiből álló listát adja vissza. */
SP_term_ref list_filter(SP_term_ref list)
{
    SP_term_ref result = SP_new_term_ref();
    SP_term_ref actual = SP_new_term_ref();
    SP_term_ref head = SP_new_term_ref();
    SP_term_ref tail = SP_new_term_ref();
    SP_term_ref temp = SP_new_term_ref();
    SP_put_variable(actual);
    SP_put_term(result, actual);
    while (SP_is_list(list))
    {
        SP_get_list(list, head, list);
        if (SP_is_integer(head))
        {
            SP_put_variable(tail);
            SP_cons_list(temp, head, tail);
            SP_unify(actual, temp);
            SP_put_term(actual, tail);
        }
    }
    SP_unify(actual, SP_new_term_ref());
    return result;
}
```

23

A SICStus Prolog Visual Basic interfésze

- Csak Prolog felé irányuló hívások
- A megoldások elkérése hasonló a C interfészhez: megnyitás, megoldások egyenként, lezárás
- Nincs term típus a Basic oldalon
- Sztringként kell megadni a lekérdezést
- Az egyes változók értékét a nevük alapján lehet elkérni egész, illetve sztring (az összes többi típus esetén) alakban
- A kivételeket szintén sztring alakban lehet elkérni

24

Az SWI Prolog C++ interfésze

- A C interfészre épül, burkolóosztályok a típusokhoz és függvényekhez
- A termekhez és atom-azonosítókhoz külön osztályok
- Kivételkezelés feljettebb: Prolog kivételek C++ kivételekként jelentkeznek a másik oldalon
- Öröklött megszorítások:
  - itt sincsenek egymásba ágyazott lekérdezések
  - csak term típusú paraméter megengedett
- Nem lehet átírni a létrehozott termet (OO szemlélet)
- `PlTerm` (term) és `PlTermv` (termek tömbje) osztályok

25

Az SWI Prolog C++ interfésze II.

- A `PlTerm` osztályon definiálva vannak a különböző típusokat (pl. `int`, `double`, `char*`) fogadó konstruktorok, valamint az objektumot ugyanezekre a típusokra alakító konverziós operátorok.
- Összehasonlító operátorok a `PlTerm` és int típusokkal
- Értékadó operátor a `PlTerm` mellett több C típusra is (`long`, `double`, `char*`). Egyesítésnek felel meg az adott (esetleges konverziók utáni) termmel.
- Indexelő operátor (`[]`): `arg/3`
- Áttekinthető kód

26

Az SWI Prolog C++ interfésze III.

- `PREDICATE` makró a C++-ban definiált predikátumokhoz. Paraméterei a predikátumnév és az aritás. A függvény törzsében az argumentumokra az A1, A2 stb. makrókkal hivatkozhatunk:

```
PREDICATE(add,3)
{
   return A3 = (long)A1 + (long)A2;
}
```

- A lekérdezés osztály (`PlQuery`), használata szinte teljesen megegyezik a C interfészével
- Nincsenek nemdeterminisztikus C++ predikátumok, bár nem lenne nehéz támogatni (a fejlesztők szerint).

27

Az SWI Prolog C++ interfésze IV.

- Objektumok átadása mutatókkal: `void*` konverziós operátor kell az adott osztályon:

```
PREDICATE(make_my_object,1)
{
   MyClass* myobj = new MyClass();
      return A1 = (void*)myobj;
}

PREDICATE(free_my_object,1)
{
      MyClass* myobj = (void*)A1;
      delete myobj;
      return TRUE;
}
```

- Csak statikus szerkesztés, kell a C++ linker (az SWI betöltő helyett)

28

Az SWI Prolog Java interfésze

- Csak Java-ból Prolog felé irányuló hívást támogat
- Nincsenek egymásba ágyazott lekérdezések
- Absztrakt `Term` osztály, a leszármazottak az egyes Prolog típusoknak megfelelő osztályok
- Lekérdezéshez a `Query` osztályt kell példányosítani
- Ez az osztály megvalósítja a Java java.util.Enumeration interfészét, megoldások elkérése ennek segítségével
- `hasMoreElements()`: van-e még megoldás
- `nextElement()`: elkérjük a megoldást. Az eredmény egy java.util.Hashtable objektum, aminek kulcsai a lekérdezés változói (változó objektumok), értékei pedig az egyes változók értékei (`Term` objektumok) a végrehajtás után.

29

Az SWI Prolog Java interfésze II.

- Példa: kiírja az `[1,2,3]` lista elemeit

```
Variable X = new Variable();
List L = new List(1,2,3);
Term args[] = {X, L};
Query q = new Query(new Atom("member"), args);
while (q.hasMoreElements())
{
  Hashtable binding = (Hashtable) q.nextElement();
  Term t = (Term) binding.get(X);
  System.out.println(t);
}
```

- Kényelmes konstruktorok 10 argumentumig
- A Prolog oldalon keletkezett kivételek Java kivételekké alakulnak (mint a C++ interfésznél)

30

Az SICStus Prolog Jasper interfésze

- Prolog és Java kód egy processzben
- Kétirányú interfész
- Egymásba ágyazott lekérdezések
- (szinte) Tetszőleges Java metódus meghívható
- A metódus paramétereinek típusa a hívást végző predikátumban (`jasper_call/4`) adható meg:

```
...
jasper_call(JVM,
   method('java/lang/Integer', 'toString', [static]),
   int_to_chars(+integer, [-chars]),
   int_to_chars(2004, N)),
...
```

31

A SICStus Prolog Jasper interfésze II.

- Az `SPTerm` osztály a C interfész `SP_term_ref` burkolóosztálya
- A Jasper át tudja írni a term-objektum belső azonosítóját
- Nincsenek a Prolog típusoknak megfelelő osztályok. Csak az `SPTerm` osztály van (nem OO szemléletű).
- Átmenet a procedurális és a funkcionális megközelítés között (pl. procedurális: `getArg`, `getList`)
- Karaktertömbbel adott term termmé történő átalakításakor a változónév-változó párokat is megkaphatjuk egy `java.util.Map` objektumban.
- Nem támogatja a nagy egészek átadását (a Java `BigInteger` osztálya kínálja magát)

32

A SICStus Prolog Jasper interfésze III.

- Prolog felé irányuló hívás
  - Tetszőleges mélységben egymásba ágyazhatók
  - Megnyitás a `SICStus` objektum `openQuery` metódusával:
    - sztringként: `java.util.Map`-be kapjuk a megoldásokat, változónév kulcs, term-objektum érték párok formájában
    - strukturáltan: modulnév, predikátumnév, argumentumok tömbje
  - A egy SPQuery objektumot ad vissza. Ennek metódusai:
    - `next()`: első-, vagy következő megoldás
    - `cut()`: vágó (az eredmények megmaradnak)
    - `close()`: visszaáll a megnyitás előtti verem-állapot

33

A SICStus Prolog Jasper interfésze IV.

- Java objektumok kezelése
  - létrehozása Prologból
  - két oldal közötti átadás
  - Java oldalon tárolódnak
  - azonosítóval lehet hivatkozni rájuk
  - példány-metódus meghívása
- Több szálból indított lekérdezések
  - A Prolognak egy verme van
  - Az előbb indított lekérdezés lezárhatja a később indítottat
- Kivételek átalakulnak a másik oldal kivételévé
  - Elvileg tökéletes megoldás
  - Java kivétel Prolog oldalon a kivétel objektum azonosítója lesz
  - A kivételről információt (pl. szöveg) csak visszahívással szerezhetünk
  - Sok esetben ez már nem sikerül, mert a Java környezet összeomlott

34

A SICStus PrologBeans interfésze

- A két oldal külön processzben fut, socket-en keresztül kommunikálva
- Futhat a két oldal külön gépen (pl. Java oldal egy böngészőben)
- Megadható, hogy milyen kliens számítógépek (IP cím vagy név alapján) használhatják a Prolog kiszolgáló szolgáltatásait.
- Jelenleg csak a Prolog felé irányuló hívásokat támogat
- Nincs visszalépés, csak az első megoldást lehet elkérni
- A Java oldali termek Java oldalon tárolódnak. Az absztrakt `Term` osztály leszármazottai az egyes Prolog típusoknak megfelelő osztályok (OO szemlélet).

35

A SICStus PrologBeans interfésze II.

- A hívást sztringként lehet megadni, az argumentumokat egy változónév-érték párokat tartalmazó leképezés objektum (`Bindings`) segítségével lehet közölni.
- Az egyes leképezések megadásához különböző típusok is használhatók: egész, lebegőpontos, sztring (karakterkódok listája illetve atom) és általános `Term` objektum is.
- A megoldás a leképezés osztály egy leszármazott osztályának (`QueryAnswer`) segítségével nyerhető ki.

36

Példa: evaluate/2

- Prolog kód:

```
...
main:-
     register_query(evaluate(C,P), my_predicate(C,P)),
     start.

my_predicate(Chars, P):-
     read_from_chars(Chars, X),
     P is X.
...
```

- Java kód:

```
...
Bindings bindings = new Bindings().bind("E", "2+3.");
QueryAnswer answer =
     session.executeQuery("evaluate(E,R)", bindings);
Term result = answer.getValue("R");
if (result != null)
     System.out.println(result);
...
```

37

Java alapú Prolog rendszerek

- Az egész rendszer Java nyelven íródott
  - GNU Prolog for Java
  - Moksa
  - PrologCafé
  - stb.
- A Prolog motor Java osztályok segítségével ágyazható be a Java alkalmazásba
- Jól integrálható
- Nagymértékben OO szemléletű, mivel a fejlesztéskor nem kellett semmihez alkalmazkodni
- A „nagy” Prolog rendszerekhez képest kevés kiegészítő modul illetve könyvtár (nagy hátrány)

38

A Sofia rendszer

- Saját készítésű Java interfész
- A két oldal külön processzben fut, kapcsolat socket-en keresztül
- A Jasper-rel próbál kompatibilis lenni
- Oda-vissza irányú hívások, tetszőleges mélységben egymásba ágyazva
- Tetszőleges mélységben egymásba ágyazható lekérdezések
- Nagy egészek átvitele mindkét irányban (Java `BigInteger` segítségével)

39

A Sofia rendszer II.

- Kivételek oda-vissza átalalkulnak a másik oldal kivételévé
- Java kivételek esetén a kivétel-objektum azonosítója mellett a kivétel összes jellemzője is átmegy:
  - a kivétel osztályának neve
  - szöveges reprezentációja
  - a teljes stacktrace Prolog lista alakban (osztálynév, metódusnév stb.)
- Egy gépen a hívások ideje egy nagyságrenddel lassabb a Jasper-nél (a taszkváltások miatt)
- A hívások adminisztrációs költsége többnyire elhanyagolható a futási idő mellett, emiatt a sebességbeli hátrány sem lesz számottevő

40

Köszönöm a figyelmet!

41