Globális korlátok megvalósítása elemi korlátok strukturált hálójaként

Hanák Dávid

Számítástudományi és Információelméleti Tanszék
dhanak@cs.bme.hu

Neumann János Doktoranduszi Konferencia

2003. október 2.

1. Bevezetés

CLP

- a *Constraint Logic Programming* (korlát logikai programozás) rövidítése;
- olyan programozási nyelvek csoportját jelöli, melyeket adott összefüggéseket (*korlátokat*) kielégítő, meghatározott tartományú értékek keresésére használunk;
- az alaphalmaztól függően több ága van:

tartomány	módszer	név	tartomány	módszer	név
boole	SAT	CLP($\mathcal{B}$)	rac./valós	LP	CLP($\mathcal{Q}/\mathcal{R}$)
egész/term.	CSP	CLP($\mathcal{FD}$)	tetszőleges		CHR

- többnyire egy általános programozási nyelvbe, pl. Prologba ágyazzák.

CLP($\mathcal{FD}$)

- az értékek lehetséges tartománya *természetes számok véges halmaza*;
- az értékeket *tartományaik változását propagáló korlátok* kötik össze;
- a megoldásokat *címkézéssel* soroltathatjuk fel;
- a korlátok lehetnek egyszerű és *globális korlátok*.

```
| ?- A in 4..7, B in 0..10, A*2 #= B, labeling([], [A,B]).
A = 4, B = 8 ; A = 5, B = 10 ; {no}
```

2. Globális korlátok ábrázolása gráfként

Háttér, célok

- Nicolas Beldiceanu (SICS) elmélete;
- absztrakt leíró nyelv globális korlátok definiálására;
- megkönnyíti a korlátok osztályozását;
- lehetővé teszi a nyelvet értő propagátorok készítését.

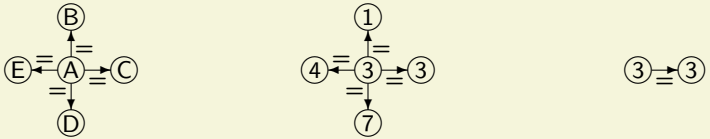
Kezdeti gráf

- a korlát leírása alapján generáljuk;
- minden argumentumhoz (ismeretlenhez) rendelünk egy csúcsot;
- a csúcsok között szabályos mintázatot követő *irányított éleket* veszünk fel;
- az élekhez *elemi* (egyszerű, kis argumentumszámú) korlátokat rendelünk.

Végső gráf

- a korlát argumentumainak adott behelyettesítéséhez tartozik;
- a kezdeti gráf teljesített elemi korlátokhoz rendelt éleit, és
- a legalább egy megmaradt éllel kapcsolódó csúcsait tartalmazza;
- a korlát csak akkor teljesül, ha a végső gráfra teljesül a lerírásban megadott gráftulajdonság (megkötés a csúcsok, élek, források, összefüggő komponensek, stb. számára vonatkozóan).

Példa: az `element` korlát



(a) Kezdeti gráf **(b)** Egy behelyettesítés **(c)** Végső gráf

Az ábra az `element` korlát egy egyszerűsített változatát mutatja. A korlát teljesül, ha az első argumentum (A), egyenlő a második argumentumként kapott lista (itt B, C, D és E) egy elemével. A megkövetelt gráftulajdonság az, hogy az élek száma pontosan egy legyen.

3. Az absztrakt leíró nyelv

Nyelvi elemek

- a kezdeti gráf és az elvárt gráftulajdonság leírására;
- *típus- és értékmegkötések* az argumentumok elfogadhatóságának feltételeire;
- a leírásbeli sorrend követi a feldolgozásuk sorrendjét.

Típusmegkötés

- a korlátok minden argumentuma típusos;
- egyszerű adattípusok: egész, ismeretlen, atom;
- *kollekció:* címkézett, típusos attribútumok halmazainak – rendezett listája, pl.

```
{ index-1 value-6 , index-2 value-9 ,
  index-3 value-2 , index-4 value-9 }
```

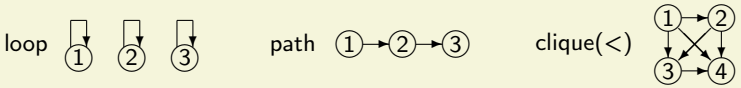
Értékmegkötés

- előfeltétel az argumentumok értékére vonatkozóan;
- *név relop kifejezés*;
- `distinct`(*Koll*/*Attr*);
- `required`(*Koll*.*Attr*);
- és sok más...

3.1. Gráfgenerálás

1. fázis: csúcsgenerálás – egy megadott kollekció minden eleme egy-egy csúcs;
2. fázis: élgenerálás – a leírásban megnevezett minta alapján;
3. fázis: a megadott elemi korlát élekhez rendelése.

Példa élgenerátorok



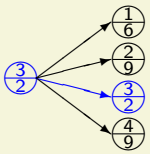
Elvárt gráftulajdonság

Egyenlet (egyenlőtlenség) alakú megkötés, bal oldalán a gráftulajdonság nevével, jobb oldalán konstansokkal és a korlát argumentumaival. Pl. `narc = 1`.

Példa: az `element` korlát leírása

```
Constraint:      element(ITEM,TABLE)
Arguments:       ITEM:  collection(index-dvar, value-dvar)
                 TABLE: collection(index-int, value-int)
Restrictions:    required([ITEM.index,ITEM.value]), |ITEM| = 1,
                 ITEM.index ≥ 1, ITEM.index ≤ |TABLE|,
                 required([TABLE.index,TABLE.value]),
                 TABLE.index ≥ 1, TABLE.index ≤ |TABLE|,
                 distinct(TABLE/index)
Arc generator:   product
Arc input:       ITEM, TABLE
Arc constraint:  ITEM.index[1] = TABLE.index[2] ∧
                 ITEM.value[1] = TABLE.value[2]
Graph property:  narc = 1
```

```
element({index-3 value-2},
        {index-1 value-6,
         index-2 value-9,
         index-3 value-2,
         index-4 value-9})
```



4. A leíró nyelv a gyakorlatban

Konkrét szintaxis

- az absztrakt szintaxis Prologhoz igazított változata;
- rávilágított a leíró nyelv egy-két gyenge pontjára.

4.1. Egyes részletek tisztázása

A `distinct` operátor. Tekintsük a következő kollekciót:

```
COLL: collection(c-collection(val-int))
```

Két lehetséges értelmezés:

1. A belső kollekciók halmazokat jelölnek, külön-külön mind egyszeresen előforduló elemekkel, de ugyanaz az elem több halmazban is előfordulhat.
2. A belső kollekciók egyetlen halmaz partíciói, azaz páronként diszjunkt részhalmazai.

A `distinct` operátorral az absztrakt nyelvben nem lehet mindkettőt leírni!

Kiválasztók

- kiválasztó ::= *Koll* | kiválasztó . *Attr*
- értékek egyenkénti kijelölésére szolgál, ha róluk *külön-külön* akarunk állítani valamit;
- tipikus előfordulása: `required` argumentuma, `TABLE.index` $\geq$ `1`.

Kigyűjtők

- kigyűjtő ::= kiválasztó | kigyűjtő / *Attr*
- értékek listáját jelöli ki, ha róluk *együttesen* akarunk állítani valamit;
- tipikus előfordulása: `distinct` argumentuma, `|KOLL/attr|`.

A `distinct` operátor problémájának megoldása

- Halmazok listája: `distinct(COLL.c/val)`, azaz minden `c` kollekcióra *külön-külön* a `val` értékek listájának ismétlődésmentesnek kell lennie;
- Egyetlen halmaz partícionálása: `distinct(COLL/c/val)`, azaz az összes `c` kollekcióban található összes `val` értéknek *együtt* kell különbözőnek lennie.

Elemi korlátok jelölése

- `ITEM.value[1]` jelentése kb.: *vedd a* `value` *attribútumát az első argumentumnak, ami* `ITEM` *típusú* – nem szerencsés;
- jobb lenne `Args[1].value` vagy `Arg1.value` jellegű jelölés.

4.2. Prolog-szerű szintaxis

Korlát definíció. Egy korlátot egy hétargumentumú Prolog klózzal definiálunk. A hét argumentum:

- a korlát neve és argumentumai;
- a típusmegkötések listája;
- az értékmegkötések listája;
- az élgenerátor bemenete (egy kollekció)
- az élgenerátor neve;
- az elemi korlát `Argk => Törzs` alakban;
- a teljesítendő gráftulajdonság.

Kollekciók

- a kollekciók alakja {*Elem1* ; *Elem2* ; ...}, ahol
- *Elem*i alakja *Attr1*-*Ért1* , *Attr2*-*Ért2* , ... ahol *Attr*i egy attribútum címkéje, *Ért*i pedig az értéke.

```
{ index-1,value-6 ; index-2,value-9 ;
  index-3,value-2 ; index-4,value-9 }
```

Példa: az `element` korlát végső formája

```
graphfd:global(element(Item, Table),
       [
        Item-collection(index-dvar, value-dvar),
        Table-collection(index-int, value-int)
       ],
       [
        required(Item.index), required(Item.value),
        size(Item) =:= 1,
        Item.index #>= 1, Item.index #=< size(Table),
        Item.value in Table/value,
        required(Table.index), required(Table.value),
        Table.index >= 1, Table.index =< size(Table),
        distinct(Table/index)
       ],
       [Item,Table],
       product,
       {A;B} => {A}.index #= {B}.index #/\ {A}.value #= {B}.value,
       narc = 1).
```

5. A megvalósításról

Tesztelés és propagálás

- Beldiceanu elmélete lehetővé teszi a korlát által leírt reláció *ellenőrzését konkrét értékekre*, de
- nem foglalkozik azzal az esettel, amikor ismeretleneket kezelünk;
- a tartományokból azok az értékek hagyhatók el, amelyek esetén a végső gráfra bizonyosan nem teljesülne az elvárt gráftulajdonság.

Fejlesztési célok

- reláció tesztelő és ezen alapuló egyszerű propagátor megvalósítása;
- a gráf pillanatnyi állapotának vizsgálatán alapuló direkt propagátor kifejlesztése;
- SICStus Prolog környezetben, annak CLP($\mathcal{FD}$) könyvtárába beágyazva.

5.1. A tesztelő és az egyszerű propagátor

Szolgáltatások

- teljes típusellenőrzés, (a `dvar` típust is `int`-ként kezelve);
- kiválasztók és kigyűjtők teljes támogatása;
- gyakoribb értékmegkötések:
  - `distinct(...)` és `required(...)`, valamint
  - tetszőleges Prolog hívás,
  - `size(...)`: az argumentumaként átadott lista vagy kollekció elemeinek száma;
- az elmélet által definiált összes élgenerátor;
- gráftulajdonságok széles választéka.

Felhasználás

- elsősorban korlátok leírásának helyességének ellenőrzésére;
- néhány hibát sikerült vele felderíteni.

Egyszerű propagálás

- „Generál-és-tesztel" jellegű;
- a tartományok megváltozásakor előállítjuk az összes lehetséges értékkombinációt, és csak azokat hagyjuk meg, amelyek átmennek a teszten;
- nagyon rossz hatékonyságú, de teljes és kimerítő propagálást ad.

A tesztelő példa futtatása

Pozitív teszt

```
Testing element({index-2,value-3}, {index-1,value-1;index-2,value-3}).
Type checking passed.
Type restrictions held.
Graph properties held.
Relation is sustained.
```

Negatív teszt

```
Testing element({index-2,value-1}, {index-1,value-1;index-2,value-3}).
Type checking passed.
Type restrictions held.
Graph properties failed.
Relation is not sustained.
```

5.2. Direkt propagálás

Élek osztályozása. A korlát egy adott állapotában a gráf élei a következőképpen osztályozhatók:

- azon élek, amelyek elemi korlátja *biztosan sikerül*, azaz bekerülnek a végső gráfba;
- azon élek, amelyek elemi korlátja *biztosan meghiúsul*, azaz kiesnek a végső gráfból;
- a többi, még bizonytalan sorsú él.

A propagálás alapelve

- a gráf *megszorítása*, azaz
- a bizonytalan sorsú élek besorolása az első két kategóriába,
- biztosítandó, hogy az elvárt gráftulajdonság teljesüljön;
- az élek besorolása a megfelelő elemi korlátok kikényszerítésével jár együtt;
- ez a tartományok szűkítését vonja maga után.

Példa: az narc $= N$ tulajdonság propagálása

- két élhalmaz: M : a biztosan megmaradók, B : a bizonytalanok.
- if $|M| > N$, meghiúsulás – túl sok az él;
- if $|M| = N$, az összes B-beli él kihagyása a végső gráfból;
- if $|M| + |B| = N$, az összes B-beli él bevétele a végső gráfba;
- if $|M| + |B| < N$, meghiúsulás – túl kevés az él.

Támogatott gráftulajdonságok

- csúcsok, élek, források és nyelők száma;
- a meglévő korlátleírások nagy része ezeket használja;
- jelenleg a „szigorúan összefüggő komponensek száma" tulajdonság vizsgálatán dolgozom.

Példa futtatás

```
| ?- graph_global(element({index-A,value-B},
       {index-1,value-6 ; index-2,value-2 ; index-3,value-2})).
A in 1..3, B in{2}\/{6} ? ;
no

| ?- graph_global(element({index-A,value-B},
       {index-1,value-6 ; index-2,value-2 ; index-3,value-2})),
     labeling([], [A]).
A = 1, B = 6 ? ;
A = 2, B = 2 ? ;
A = 3, B = 2 ? ;
no
```

6. További feladatok, köszönetnyilvánítás

További feladatok

- további gráftulajdonságok vizsgálata, a meglévő propagáló következtetések javítása;
- elemi korlátok hatékonyabb, erősebb propagációt adó megvalósítása;

```
| ?- graph_global(element({index-A,value-B},
       {index-1,value-6 ; index-2,value-2 ; index-3,value-2})),
     B #= 2.
B = 2,
A in 1..3 ?
```

- propagáló-algoritmus generátor megvalósíthatóságának vizsgálata.

Köszönet

- Nicolas Beldiceanu-nak az elmélet megalkotásáért és gyakorlati ötleteiért;
- Szeredi Péternek a kutatómunka irányításáért.