Logikai programok verifikálása

Patai Gergely

2004. december 6.

Bevezető gondolatok

- a deklaratív programozás kapcsán felmerül a verifikálhatóság fogalma
- rengeteg cikk hozzáférhető, ám ehhez képest kevés a működő implementáció
- LPTP tételbizonyító

A Logic Program Theorem Prover

- tiszta Prolog programok formális verifikálására képes, *működő* rendszer
- maga is Prologban íródott, és sok Prolog rendszeren fut (SICStus, SWI, GNU...)
- kényelmes emacs felület az interaktív tételbizonyításhoz
- TEX és HTML kimenetet is tud generálni
- `http://www.inf.ethz.ch/personal/staerk/lptp.html`

Az LPTP képességei

- vizsgálható tulajdonságok: terminálódás, predikátumok ekvivalenciája, determinizmus...
- megengedett konstrukciók: if-then-else, `is/2`, `integer/1`, `call/n+1`, `arg/3`...
- nem megengedett: !, `assert/1`, `retract/1`, `var/1`...

A vizsgált program (`list.pl`)

```
list([]).
list([X|L]) :- list(L).

member(X, [X|L]).
member(X, [Y|L]) :- member(X, L).

append([], L, L).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).
```

Tömör alakra hozás

- a rendszer a programmal közvetlenül nem tud adatként dolgozni a változók miatt, előbb át kell azt alakítani megfelelő alakra:

```
?- [lptp].
?- compile_gr(list).  % list.pr létrehozása
?- needs_gr(list).  % list.pr betöltése
```

- ellenőrzésképpen lekérdezhetjük az alapdefiníciók formáját a megadott predikátumokon: terminálódás, siker, meghiúsulás
- nem következtetés, csupán egyszerű transzformáció

Alapdefiníciók – folytatás

```
?- def(succeeds member(?x, [?y|?m])).
?x = ?y \/ succeeds member(?x,?m)

?- def(succeeds member(?x, ?m)).
(ex l: ?m = [?x|?l]) \/
(ex [y,l]: ?m = [?y|?l] & succeeds member(?x,?l))

?- def(fails member(?x, ?m)).
(all l: ~ ?m = [?x|?l]) &
(all [y,l]: ?m = [?y|?l] => fails member(?x,?l))

?- def(terminates member(?x, ?m)).
all [y,l]: ?m = [?y|?l] => terminates member(?x,?l)
```

Alapdefiníciók – siker és meghiúsulás

$$S(\texttt{true}) := \texttt{tt},$$
$$S(\texttt{fail}) := \texttt{ff},$$
$$S(s \texttt{ = } t) := s \texttt{ = } t,$$
$$S(A) := \texttt{succeeds } A,$$
$$S(\texttt{\~{}}G) := F(G),$$
$$S(G_1 \texttt{ \& } \ldots \texttt{ \& } G_n) := S(G_1) \texttt{ \& } \ldots \texttt{ \& } S(G_n),$$
$$S(G_1 \texttt{ \textbackslash/ } \ldots \texttt{ \textbackslash/ } G_n) := S(G_1) \texttt{ \textbackslash/ } \ldots \texttt{ \textbackslash/ } S(G_n).$$

$$F(\texttt{true}) := \texttt{ff},$$
$$F(\texttt{fail}) := \texttt{tt},$$
$$F(s \texttt{ = } t) := s \texttt{ <> } t,$$
$$F(A) := \texttt{fails } A,$$
$$F(\texttt{\~{}}G) := S(G),$$
$$F(G_1 \texttt{ \& } \ldots \texttt{ \& } G_n) := F(G_1) \texttt{ \textbackslash/ } \ldots \texttt{ \textbackslash/ } F(G_n),$$
$$F(G_1 \texttt{ \textbackslash/ } \ldots \texttt{ \textbackslash/ } G_n) := F(G_1) \texttt{ \& } \ldots \texttt{ \& } F(G_n).$$

Alapdefiníciók – terminálódás

$$
\begin{aligned}
&T(\texttt{true}) := \texttt{tt},\\
&T(\texttt{fail}) := \texttt{tt},\\
&T(s \texttt{ = } t) := \texttt{tt},\\
&T(A) := \texttt{terminates } A,\\
&T(\texttt{\~{}}G) := T(G) \texttt{ \& gr(}x_1\texttt{) \& ... \& gr(}x_n\texttt{)}, \text{ if } \mathrm{FV}(G) = \{x_1, \ldots, x_n\},\\
&T(G_1 \texttt{ \& ... \& } G_n) := \texttt{terminates (}G_1 \texttt{ \& ... \& } G_n\texttt{)},\\
&T(G_1 \texttt{ \textbackslash/ ... \textbackslash/ } G_n) := T(G_1) \texttt{ \& ... \& } T(G_n).
\end{aligned}
$$

Az első bizonyítás

A következő állítást próbáljuk bebizonyítani: ha L-re teljesül `list(L)`, akkor a `member(X, L)` cél biztosan lefut bármely X és L esetén. Ehhez létre kell hozni egy `.pr` kiterjesztésű file-t a következő tartalommal:

```
:- initialize.
:- needs_gr(list).

:- lemma(member:termination,
all [y,l]: succeeds list(?l) => terminates member(?y,?l),
all [y,l]: succeeds list(?l) => terminates member(?y,?l)
 by [ind]
).
```

Az első bizonyítás – levezetés

A programot betöltve a rendszer válasza:

```
induction([all l: succeeds list(?l) =>
 (all y: terminates member(?y,?l))],
 [step([],[],[],all y: terminates member(?y,[])),
  step([x,l],
   [all y: terminates member(?y,?l),
    succeeds list(?l)],
   [],
   all y: terminates member(?y,[?x|?l]))])
```

Az első bizonyítás – levezetés

File: `example.pr`

Lemma 1 $[member{:}termination]$ $\forall y, l\,(\mathbf{S}\,\mathtt{list}(l) \rightarrow \mathbf{T}\,\mathtt{member}(y, l))$.

Proof.

Induction_0: $\forall l\,(\mathbf{S}\,\mathtt{list}(l) \rightarrow \forall y\ \mathbf{T}\,\mathtt{member}(y, l))$.

Hypothesis_1: none.

Conclusion_1: $\forall y\ \mathbf{T}\,\mathtt{member}(y, [])$.

Hypothesis_1: $\forall y\ \mathbf{T}\,\mathtt{member}(y, l)$ and $\mathbf{S}\,\mathtt{list}(l)$.

Conclusion_1: $\forall y\ \mathbf{T}\,\mathtt{member}(y, [x|l])$. □

Az első bizonyítás – befejezés

A kapott választ vissza kell másolni a bizonyítás hiányos levezetési lépésének a helyére, azaz a `lemma/3` harmadik paraméterét kell kicserélni az eredményként előállt `induction/2` struktúrára.
A további bizonyításokhoz a program fel tudja használni ezt a lemmát.

A második bizonyítás

A beépített és a felhasználói predikátumok mellett definiálhatunk rövidítéseket is, amelyek végeredményben makrók. Például a részhalmaz reláció definíciója:

```
:- definition_pred(sub,2,
all [l1,l2]: sub(?l1,?l2) <=>
 (all x: succeeds member(?x,?l1) =>
 succeeds member(?x,?l2))
).
```

A második bizonyítás – folytatás

Próbáljuk megmutatni, hogy a reláció tranzitív! Az előző bizonyítással ellentétben most a rendszerre bízzuk a módszert:

```
:- lemma(sub:transitive,
all [l1,l2,l3]: sub(?l1,?l2) & sub(?l2,?l3) =>
 sub(?l1,?l3),
all [l1,l2,l3]: sub(?l1,?l2) & sub(?l2,?l3) =>
 sub(?l1,?l3) by [auto(9)]
).
```

A második bizonyítás – levezetés

```
assume(sub(?l1,?l2) & sub(?l2,?l3),
 [assume(succeeds member(?x,?l1),
   [all x: succeeds member(?x,?l1) =>
     succeeds member(?x,?l2) by
     elimination(sub,2),
    all x: succeeds member(?x,?l2) =>
     succeeds member(?x,?l3) by
     elimination(sub,2)],
   succeeds member(?x,?l3)),
  sub(?l1,?l3) by introduction(sub,2)],
 sub(?l1,?l3))
```

További bizonyítások

- sub:member lemma (részhalmaz bővítése meglevő taggal is részhalmaz)
- sub:member:2 következmény (ugyanez két taggal az előzőt felhasználva)
- append:member lemma (minden tag szerepel egy partícióban; több lépésben)
- sub:append lemma (részhalmazok konkatenáltja is részhalmaz)

Általános tapasztalatok

- nagyon jól használható kezelőfelület
- sok előre levezetett tétel van hozzá
- nem lassul le számottevően a tudásbázis gyarapodásával (az `auto` szabályt leszámítva)
- nem érdemes egyszerre nagyon bonyolult következtetéseket rábízni
- magára vállalja az elemi lépésekkel járó papírmunkát; gyakran csak a taktikát kell megválasztanunk a továbblépéshez

Bizonyítási stratégiák

- támogatja a top-down és a bottom-up jellegű bizonyításokat is
- jól struktúrálhatók a levezetések segédtételek bevezetésével, és ezeket nem előre kell kitalálni...
- problémaforrások:
  - körkörös érvelés
  - hamis állítás bizonyítása
  - eltérő alakú állítások és tételek