Синтез операторов предикатной программы

Шелехов В.И.

Институт Систем Информатики, Новосибирск

05.04.17, Ростов-на-Дону,

Языки программирования и компиляторы – 2017

Пусть x – набор аргументов, y – набор результатов

Предикатная программа H(x: y) –

предикат (логическая формула)

в форме вычислимого оператора

Язык предикатного программирования **P**

Вычислимые предикаты языка $\mathbf{P_0}$ и их программная форма

Вычислимый предикат	Программа на языке $\mathbf{P_0}$
$H(x{:}\ y) \cong \exists z.\ B(x{:}\ z)\ \&\ C(z{:}\ y)$	H(x: y) { B(x: z); C(z: y) }
$H(x{:}\ y, z) \cong B(x{:}\ y)\ \&\ C(x{:}\ z)$	H(x: y, z) { B(x: y) || C(x: z) }
$H(x{:}\ y) \cong (e \Rightarrow B(x{:}\ y))\ \&\ (\neg e \Rightarrow C(x{:}\ y))$	H(x: y) { **if** (e) B(x: y) **else** C(x: y) }
$H(x{:}\ y) \cong B(x^{\sim}{:}\ y)$	H(x: y) { B(x˜: y) }
$H(A, x{:}\ y) \cong A(x{:}\ y)$	H(A, x: y) { A(x: y) }
$H(x{:}\ D) \cong \forall y,z.\ D(y{:}\ z) \equiv B(x, y{:}\ z)$	H(x: D) { D(y: z) {B(x, y: z)} }
$H(A, x{:}\ D) \cong \forall y,z.\ D(y{:}\ z) \equiv A(x, y{:}\ z)$	H(A, x: D) { D(y: z) {A(x, y: z)} }

Для языка предикатного программирования **P** построена **формальная операционная семантика**

(T) Исполнение программы $H(x: y)$ завершается вычислением y $\Leftrightarrow$ предикат $H(x: y)$ является истинным

Спецификация предикатной программы $H(x: y)$:

$P(x)$ — *предусловие*, $Q(x, y)$ — *постусловие*

Тотальная корректность программы:

$H(x: y)$ **corr** $[P(x), Q(x, y)] \cong$

- $P(x)\ \&\ H(x: y) \Rightarrow Q(x, y)$ – частичная корректность
- $P(x) \Rightarrow \exists y.\ H(x: y)$ – тотальность

Задача программного синтеза:

$$P(x)\ \&\ H(x: y) \Rightarrow Q(x, y)$$

Система правил доказательства корректности операторов.

$$\mathbf{QC:}\ \frac{B(x\!:\,y)\ \mathbf{corr}\ [P(x)\ \&\ E(x),\ Q(x,y)];\quad C(x\!:\,z)\ \mathbf{corr}\ [P(x)\ \&\ \neg E(x),\ Q(x,y)]}{\{\mathbf{if}\ (E(x))\ B(x\!:\,y)\ \mathbf{else}\ C(x\!:\,y)\}\ \mathbf{corr}\ [P(x),\ Q(x,y)]}$$

Корректность системы правил доказана в PVS:

http://www.iis.nsk.su/persons/vshel/files/rules.zip

$H(x\!:\,y)\ \{\mathbf{if}\ (E(x))\ [*]\ \mathbf{else}\ C(x\!:\,y)\}$

$H(x\!:\,y)\ \{\mathbf{if}\ (E(x))\ Z(x\!:\,y)\ \mathbf{else}\ C(x\!:\,y)\}$

$Z(x\!:\,y)\ \mathbf{corr}\ [P(x)\ \&\ E(x),\ Q(x,y)]$

$P(x)\ \&\ E(x)\ \&\ Z(x\!:\,y) \Rightarrow Q(x,y)$

Задача синтеза сводится к задаче разрешимости логических формул относительно неизвестных термов и предикатов

Программный синтез фрагментов предикатной программы в интеграции с дедуктивной верификацией в редакторе Eclipse

Цель – анализ методов синтеза операторов на примере эффективной программы сортировки простыми вставками

Конструирование программы в стиле доказательного программирования: программа строится исходя из формальных спецификаций вместе с набором теорий, поддерживающих доказательство генерируемых формул корректности

Пример. Сортировка простыми вставками

```
program SORT (type T, nat n) {  // T – тип элементов с "≤"
  import Total_order(T, "≤")
  type natn = 0 .. n;
  type Arn = array (T, natn);
.......................................................................................
}
```

Массивы вместо списков

Спецификация программы сортировки:

sort(Arn a: a’) **post** perm(a, a’) & sorted(a’)

```
theory Sort {
  formula sorted(Arn a) = ∀ natn i, j. i < j ⇒ a[i] ≤ a[j];
};
```

Библиотечная программа

with(A, i, z**:** D)

{ D(x**:** y) { **if** (x = i) y = z **else** A(x**:** y)} } .

Вызов with(B, i, z**:** C) записывается C = B **with** (i**:** z).

theory Perm {

formula swap(Arn a, b) = ∃ i,j=0..m. b = a **with** (i**:** a[j], j**:** a[i]);

formula perm(Arn a, b) =

a = b **or** ∃ Arn c. swap(a, c) & perm(c, b);

Arn a, b, c;

pe1: lemma perm(a, a);

pe2: lemma perm(a, b) & perm(b, c) ⇒ perm(a, c);

};

Сведение к более общей задаче

theory Sort { ……..

formula sorted(Arn a, natn m) = $\forall$ i, j = 0..m. i < j $\Rightarrow$ a[i] $\leq$ a[j]

Arn a;

so1: lemma sorted(a, n) $\Rightarrow$ sorted(a);

};

sort1(Arn a, natn m**:** Arn a')

pre sorted(a, m) **post** perm(a, a') & sorted(a');

Синтез программы sort

sort(Arn a**:** Arn a') **post** perm(a, a') & sorted(a') { sort1([*]) };

sort(Arn a**:** Arn a') **post** perm(a, a') & sorted(a')
{ sort1(a, X**:** a') };

Синтез программы sort

sort(Arn a: Arn a’) **post** perm(a, a’) & sorted(a’)

{ sort1(a, X: a’) };

$$\textbf{RBE:}\ \frac{\forall z\ C(x, z\!: y)\ \textbf{corr*}\ [P_C(x, z), Q(x, y)];\quad P(x) \to \exists z.\ B(x\!: z)\ \&\ P_C{}^*(x, B(x));}{C(x, B(x)\!: y)\ \textbf{corr}\ [P(x), Q(x, y)]}$$

z = B(x) является эквивалентом B(x: z)

C(x, B(x): y) эквивалентно B(x: z);C(x, z: y)

$$\textbf{RBE:}\ \frac{\forall\ \text{natn } m.\ \text{sort1}(a, m\!: a')\ \textbf{corr}\ [\text{sorted}(a, m), Q(a, a')];\quad \exists\ \text{natn } m.\ X(a\!: m)\ \&\ \text{sorted}(a, X(a));}{\text{sort1}(a, X(a)\!: y)\ \textbf{corr}\ [\textbf{true}, Q(a, a')]}$$

formula Q(a, a’) = perm(a, a’) & sorted(a’);

Задача синтеза: найти такой терм X(a), чтобы формула корректности sorted(a, X(a)) стала истинной.

Задача синтеза: найти такой терм X(a), чтобы формула корректности sorted(a, X(a)) стала истинной.

Перебор термов: X(a) = 0, 1, n, n-1, len(a), …

formula sorted(Arn a, natn m) = $\forall$ i, j = 0..m. i < j $\Rightarrow$ a[i] $\leq$ a[j]

sorted(a, 0) истинна !!!

Результат синтеза:

sort(Arn a: Arn a') **post** perm(a, a') & sorted(a')

{ sort1(a, 0**:** a') };

Спецификация программы pop_into вставки очередного элемента a[m+1] внутрь отсортированной части a[0..m] :

pop_into(Arn a, natn m**:** Arn a’)

pre m < n & sorted(a, m)

post perm(a, a’) & sorted(a’, m+1);

Синтез фрагментов в программе сортировки:

sort1(Arn a, natn m: Arn a’)

pre sorted(a, m) **post** perm(a, a’) & sorted(a’)

{ **if** (m = n) [*] **else** { pop_into(a, m**:** Arn c); sort1([*]) } };

if (m = n) X(a, m**:** a’)

else { pop_into(a, m**:** Arn c); sort1(U(c, m), V(c, m) **:** a’) } sort1(Arn a, natn m: Arn a’)

pre sorted(a, m) **post** perm(a, a’) & sorted(a’)

if (m = n) X(a, m**:** a’)

else { pop_into(a, m**:** Arn c); sort1(U(c, m), V(c, m) **:** a’) }

Формулы корректности:

sorted(a, m) & m = n & X(a, m**:** a’) $\Rightarrow$ perm(a, a’) & sorted(a’)

sorted(a, m) & m<n & perm(a, c) & sorted(c, m+1) & perm(c, a’) & sorted(a’) $\Rightarrow$ perm(a, a’) & sorted(a’);

sorted(a, m) & m<n & perm(a, c) & sorted(c, m+1) $\Rightarrow$ sorted(U, V) & h(U, V) < h(a, m);

sorted(a, m) & m<n $\Rightarrow$ m < n & sorted(a, m)

pe1: lemma perm(a, a); $\rightarrow$ решение: X есть оператор a’ = a

so1: lemma sorted(a, n) $\Rightarrow$ sorted(a); pe2 - транзитивность

Унификация sorted(U, V) и sorted(c, m+1)

Результат синтеза:

sort1(Arn a, natn m**:** Arn a’)

pre sorted(a, m) **post** perm(a, a’) & sorted(a’) **measure** n - m

{ **if** (m = n) a’ = a

else { pop_into(a, m**:** Arn c); sort1(c, m+1**:** a’) }

};

$a_0 \ldots\ldots\ldots a_{k-1}$		$a_{k+1} \ldots\ldots\ldots a_{m+1}$	e

Схема работы алгоритма pop_into

Спецификация обобщенной программы pop_into :

pop_into(Arn a, natn k, m, T e**:** Arn a’)// k – пустая позиция

pre m < n & k > 0 & sorted(a, k-1) & sorted(a, k+1, m+1) &

(k > m **or** e < a[k+1] & a[k-1] <= a[k+1])

post perm(a **with** (k**:** e), a’) & sorted(a’, m+1);

formula sorted(Arn a, natn k, m) = ∀ i, j = k..m. i < j ⇒ a[i] ≤ a[j];

В итоге получим следующую программу pop_into:

```
pop_into(Arn a, natn k, m, T e: Arn a') measure k
{ if (a[k-1] <= e) a' = a with (k: e)
  else { Arn b = a with (k: a[k-1]);
           if (k = 1) a' = b with (0: e) else pop_into(b, k-1, m, e: a')
     }
}
```

Применение оптимизирующих трансформаций дает следующую программу сортировки на императивном расширении языка **P**:

```
sort(Arn a)
{  for (natn m = 0; ! m = n; m = m+1) {
         T e = a[m+1];
         for (natn k = m+1; ; k = k-1)
             if (a[k-1] <= e) { a[k]= e; break; }
             else { a[k] = a[k-1]; if (k = 1) { a[0] = e; break; } }
   }
}
```

pop_into(Arn a, natn k, m, T e**:** Arn a’) ***//*** k – позиция дырки

pre m < n & k > 0 & sorted(a, k-1) & sorted(a, k+1, m+1) &

(k > m **or** e < a[k+1] & a[k-1] <= a[k+1])

post perm(a **with** (k**:** e), a’) & sorted(a’, m+1);

Упростить спецификацию pop_into !!!!

Поскольку в новом релизе a[m] > e, можно будет начинать с a **with** (m+1**:** a[m])

pop_into1(Arn a, natn k, m, T e**:** Arn a’) ***//*** k – позиция дырки

pre m < n & k > 0 & sorted(a, k-1) & sorted(a, k+1, m+1) &

a[k+1] & a[k-1] <= a[k+1]

post perm(a **with** (k**:** e), a’) & sorted(a’, m+1);

```
sort1(Arn a, natn m: Arn a')
pre sorted(a, m) post perm(a, a') & sorted(a')
{ if (m = n) a' = a
  else { T e = a[m+1];
             if (a[m] <= e) sort1(a, m+1: a')
             else {  Arn b = a with (m+1: a[m]);
                        if (m=0) a' = b with (0: e)
                        else { pop_into1(b, m, m, e: Arn c);
                                  sort1(c, m+1: a')
                                }
                     }
           }
};
```

formula Ppop(a, k, m, e) = m < n & k > 0 & sorted(a, k-1)
& sorted(a, k+1, m+1) & a[k-1] <= a[k+1] & e < a[k+1];

formula Qpop(a, k, m, e, a’) = perm(a **with** (k**:** e), a’) &
sorted(a’, m+1)

pop_into1(Arn a, natn k, m, T e**:** Arn a’) ***//*** k – позиция дырки

pre Ppop(a, k, m, e) **post** Qpop(a, k, m, e, a’)

{ **if** (a[k-1] <= e) [*] **else if** (k = 1) [*] **else** pop_into1([*]) }

Синтез фрагментов:

if (a[k-1] <= e) X(a, k, m, e**:** a’)

else if (k = 1) Y(a, k, m, e**:** a’)

else pop_into1(A, K, M, E**:** a’)

Первая формула корректности:

$m < n$ & $k > 0$ & sorted(a, k-1) & sorted(a, k+1, m+1) &
a[k-1] <= a[k+1] & e < a[k+1] & a[k-1] <= e &
X(a, k, m, e**:** a’) $\Rightarrow$ perm(a **with** (k**:** e), a’) & sorted(a’, m+1)

Лемма pe1**:** perm(a, a) $\rightarrow$ решение: X есть оператор
a’ = a **with** (k**:** e) .

Используется старая лемма so3.

so3: lemma m < n & k > 0 & sorted(a, k-1) &
sorted(a, k+1, m+1) & e < a[k+1] & a[k-1] <= e $\Rightarrow$
sorted(a **with** (k**:** e), m+1);

Вторая формула корректности:

m < n & k > 0 & sorted(a, k-1) & sorted(a, k+1, m+1) &
a[k-1] <= a[k+1] & e < a[k+1] & a[k-1] > e & k = 1 &
Y(a, k, m, e**:** a’) ⇒ perm(a **with** (k**:** e), a’) & sorted(a’, m+1)

В качестве Y используется a’ = a **with** (0**:** e, 1**:** a[0])

Синтезатору автоматически этого не сделать !!!

pe3: lemma a[0] > e ⇒
perm(a **with** (0**:** e, 1**:** a[0]), a **with** (1**:** e)));

Второй конъюнкт доказывается с использованием леммы:

so5: lemma m < n & sorted(a, 2, m+1) & e < a[2] & a[0] > e ⇒
sorted(a **with** (1**:** a[0], 0**:** e), m+1)

formula Pp3(a,k,m,e) = Ppop(a, k, m, e) & a[k-1] > e & k $\neq$ 1

Третья формула корректности:

Pp3(a,k,m,e) & perm(A **with** (K**:** E), a') & sorted(a', M+1)
$\Rightarrow$ perm(a **with** (k**:** e), a') & sorted(a', m+1);

Решение:

A = a **with** (k**:** a[k-1]), K = k-1, M = m, E= e

Необходима лемма (аналог **sb1**):

pe4: lemma k > 0 & k <=n $\Rightarrow$
perm(a **with** (k**:** a[k-1], k-1**:** e), a **with** (k**:** a[k-1]));

Перебором решение получить можно.

Доказать без леммы невозможно.

Построить лемму **pe4** без формулы корректности нереально.

Четвертая формула корректности:

$m < n \ \& \ k > 0 \ \& \ sorted(a, k-1) \ \& \ sorted(a, k+1, m+1) \ \&$

$a[k-1] <= a[k+1] \ \& \ e < a[k+1] \ \& \ a[k-1] > e \ \& \ k \neq 1 \Rightarrow$

$m < n \ \& \ K > 0 \ \& \ sorted(A, K-1) \ \& \ sorted(A, K+1, m+1) \ \&$

$A[K-1] <= A[K+1] \ \& \ E < A[K+1]$

$$A = a \textbf{ with } (k\textbf{:}\ a[k-1])$$

so6: lemma $m < n \ \& \ k > 1 \ \& \ sorted(a, k-1) \Rightarrow$

$sorted(a \textbf{ with } (k\textbf{:}\ a[k-1]), k-2)$

so7: lemma $m < n \ \& \ k > 0 \ \& \ k < m \ \& \ sorted(a, k+1, m+1) \ \&$

$a[k-1] <= a[k+1] \Rightarrow sorted(a \textbf{ with } (k\textbf{:}\ a[k-1]), k, m+1)$

Заключение

Автоматический синтез нереализуем для программ по сложности выше некоторого уровня.

Как улучшить интегрированную систему дедуктивной верификации и синтеза ?

Специализированный интерактивный решатель:

- Преобразования: унификация термов, перебор термов, подстановки, обеспечивающие истинность формул с использованием лемм, и др.
- Удобная визуализация генерируемых формул корректности.

Спасибо

Вопросы

Видеолекции по Формальным Методам:

http://wasp.iis.nsk.su