22.10.21 ФМ-3 Лекция 2

Доказательство в системе Why3

http://why3.lri.fr/manual.pdf
http://why3.lri.fr/stdlib/

Стандартная библиотека

Пример1: инверсия списка.

Рекурсивный вариант в why3.

Версия в предикатном программировании.

Версия в why3 с инвариантом цикла

Пример2: Задача 12

Пример3: memweight – число элементов множества в виде битовой шкалы

Техника доказательства.

Методы локализации ошибок

Стандартная библиотека Why3

algebra : Basic Algebra Theories array : Arrays bag : Multisets (aka bags)
bintree : Polymorphic binary trees with elements at nodes
bool : Booleans bv : Bit Vectors exn : General-purpose exceptions
floating_point : Formalization of Floating-Point Arithmetic
fmap : Finite Maps function : Injections, surjections and bijections
graph : Graph theory hashtbl : Hash tables
int : Theory of integers list : Polymorphic Lists map : Theory of maps
matrix : Matrices null : A possibly null, yet safe, value number : Number theory
option : Option type ocaml : General functions related to OCaml extraction
pigeon : Pigeon hole principle pqueue : Priority queues
queue : Polymorphic mutable queues random : Pseudo-random generators
real : Theory of reals ref : References regexp : Theory of regular expressions
relations : Relations seq : Sequences set : Set theories stack : Stacks
string : Theory of strings tree : Polymorphic n-ary trees
mach.array : Arrays with bounded-size integers
mach.bv : Program functions on bitvectors with preconditionsenforcing absence of overflow .....

Пример. Обращение списка

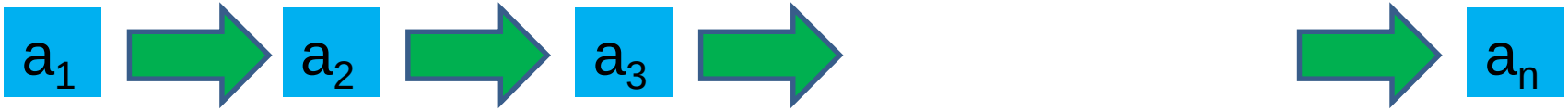
Результат обращения списка:

Односвязный список**:**

struct llist_node { **struct** llist_node *next; };

Программа реверсирования списка:

struct llist_node *llist_reverse_order(**struct** llist_node *head);



Обобщение задачи:

$a_k, a_{k-1}, \ldots a_1$


```
struct llist_node *llist_reverse_order( struct llist_node *head) {
   struct llist_node *new_head = NULL;
   while (head) {
        struct llist_node *tmp = head;
        head = head->next;
        tmp->next = new_head;
        new_head = tmp;
    }
   return new_head;
}
```

Программа после трансформаций

```
type  T;
type slist = union ( Nul, Node(T data, slist next) );


slist  llist_reverse_order(slist head){
        slist new_head = Nul;
        while (head != Nul) {
          slist tmp = head;
          head = head.next;
          tmp.next = new_head;
          new_head = tmp;
        }
        return new_head;
}
```

Предикатная программа

```
type  T;
type slist = union ( Nul, Node(T data, slist next) );

formula tolist(slist x) = if x = Nul then nil else cons(x.data, tolist(x.next);
llist_reverse_order(slist head : slist new_head)
  post tolist(new_head) = reverse(tolist(head))
{          reverseC(Nul, head :  new_head); }

reverseC(slist new_head, head :  slist new_head1)
  post tolist(new_head1) = reverse(tolist(head)) + tolist(new_head)
  measure len(new_head)
{   if head = Nul then new_head1 = new_head
    else reverseC(Node(head.data, new_head), head.next: new_head)
}
```

Рекурсивная программа на языке WhyML

```
theory Slist
  use int.Int, list.List, list.Reverse,  list.Append

  type data
  type listD = list data
  type tT
  type slist = Nul | Node data slist


  let rec function tolist(x: slist) : listD =
          match x with
            | Nul -> Nil
            | Node d sl -> Cons d (tolist sl)
          end
```

Программа на языке WhyML прод

```
let rec function reverseC(new_head head : slist): slist
        ensures{ tolist result = reverse (tolist head) ++ tolist new_head }
      (* variant { length new_head} *)
=
       match head with
         | Nul -> new_head
         | Node d sn -> reverseC (Cons d new_head) sn
       end

let function llist_reverse_order(head : slist): slist
        ensures{ tolist result = reverse (tolist head) }
 =
        reverseC Nul head
end (* of Slist theory *)
```

Формулы корректности рекурсивной программы

```
goal reverseC'vc :
  forall new_head:slist, head:slist.
   forall result:slist.
    match head with
    | Nul -> result = new_head
    | Node d sn ->
       tolist result = (reverse (tolist sn) ++ tolist (Node d new_head))
    end -> tolist result = (reverse (tolist head) ++ tolist new_head)

goal llist_reverse_order'vc :
  forall head:slist.
   let o = Nul in
   let result = reverseC o head in
   tolist result = (reverse (tolist head) ++ tolist o) ->
   tolist result = reverse (tolist head)
```

Формулы корректности предикатной программы

Теория на языке P

```
type T;
type listD = list(T);
type slist = union ( Nul, Node(T data, slist next) );

function tolist(slist x): listD = if x = Nul then nil else cons(x.data, tolist(x.next));
formula qRev(slist head, new_head) =  tolist(new_head) = reverse(tolist(head))

formula reverseG(slist new_head, head, slist new_head1) =
        tolist(new_head1) = reverse(tolist(head)) ++ tolist(new_head)
lemma RB1:  reverseG(Nul, head, new_head) ⇒  qRev(head, new_head);
lemma COR:     head = Nul & new_head1 = new_head  ⇒
                        reverseG(new_head, head, new_head1);

RB2: head != Nul ⇒ length(tolist(head.next)) < length(tolist(head))
RB3: head != Nul & reverseG(Node(head.data, new_head), head.next, new_head1)
                ⇒      reverseG(new_head, head, new_head1)
```

Императивная программа на языке WhyML

```
let function llist_reverse_order(head0 : slist) : slist
    ensures{  reverse(tolist result) = tolist head0 }
=     let ref new_head = Nul   in
      let ref head = head0   in
      while match head with | Nul -> false | Node _ _ -> true end do
         invariant { reverse (tolist head0) =
                              reverse (tolist head) ++ tolist new_head }
         variant { length (tolist head)}
         match head with
           | Nul -> absurd
           | Node d ne ->
             let ref tmp = head in
             head <- ne;
             tmp <- Node d new_head;
             new_head <- tmp
          end
      done ;
      new_head
```

12. Выделить ближайшую слева максимальную подстроку из десятичных цифр.

Десятичная строка d (строка из десятичных цифр) длины n:

```
formula dec(string d, nat n) =
    (d=nil & n=0) or  (digit(d.car) & dec(d.cdr, n-1));
All(string s, H(char)) ≅ if (s=nil) true else H(s.car) & All(s.cdr)
formula dec(string d, nat n) = All(d, lambda char c. digit(c)) &
      len(d) = n
```

Десятичная подстрока d длины n:

```
formula sub(string s, d, nat n) =
              exists string u, v. s = u + d + v & dec(d, n);
```

Максимальная десятичная подстрока d длины n:

```
formula mSub(string s, d, nat n) =
sub(s, d, n) & forall string u, nat m. sub(s, u, m) ⇒ m <= n;
```

Максимальная слева десятичная подстрока d длины n:

formula mLeft(**string** s, d, **nat** n) =

mSub(s, d, n) & **exists** **string** u, v, d1, **nat** m.

s = u+d+v & mSub(u, d1, m) ⇒ m < n;

Максимальная слева десятичная подстрока d :

formula mLeft(**string** s, d) = **exists** **nat** n. mLeft(s, d, n);

Спецификация задачи 12:

ext(**string** s: **string** d) **post** mLeft(s, d);

Обобщение исходной задачи:

d – *максимальная слева из* d0 и максимальной слева десятичной подстроки в s :

formula mExt(**string** s, d, d0, **nat** m) =

exists **string** d1, **nat** n. mLeft(s, d1, n) & d = (n > m)? d1: d0;

extG(string d0, nat m, string s: string d)
pre dec(d0, m) post mExt(s, d, d0, m);

Сведение к задаче extG :

ext(string s: string d) post mLeft(s, d)

{ extG(nil, 0, s: d)}

d1 – десятичная подстрока длины k в начале s;
s1 – остаток строки s.

Dec(s: nat k, string d1, s1) pre s ≠ nil & digit(s.car))
post dec(d1, k) & s = d1 + s1 &
(s1 = nil or not digit(s1.car));

Выделение десятичной подстроки d1 длины k в начале s.

```
Dec(s: nat k, string d1, s1) pre s ≠ nil & digit(s.car) )
      post dec(d1, k) & s = d1 + s1 &
                                  (s1 = nil or not digit(s1.car));

extG(string d0, nat m, string s: string d)
      pre dec(d0, m) post mExt(s, d, d0, m);
      measure len(s)
{  if (s = nil) d = d0
   else if (digit(s.car)) {
        Dec(s: nat k, string d1, s1);
        if (k <= m) extG(d0, m, s1: d)
        else extG(d1, k, s1: d)
   } else extG(d0, m, s.cdr: d)}
}
```

Дополнение d0 длины k0 до десятичной строки d1 длины k
выделением из начала строки s; s1 – остаток строки s.

```
DecO(string d0, s0, nat k0: nat k, string d1, s1) pre dec(d0, k0)
    post dec(d1, k) & d0 + s0 = d1 + s1 &
            (s1 = nil or not digit(s1.car))
Dec(s: nat k, string d1, s1) pre s ≠ nil & digit(s.car) )
post dec(d1, k) & s = d1 + s1 & (s1 = nil or not digit(s1.car))
{  DecO(s.car, s.cdr, 1: k, d1, s1)  };

DecO(string d0, s0, nat k0: nat k, string d1, s1) pre dec(d0, k0)
    post dec(d1, k) & d0 + s0 = d1 + s1 &
            (s1 = nil or not digit(s1.car))
    measure len(s0)
{  if (s0 = nil or not digit(s0.car)) d1 = d0 || k = k0 || s1 = s0
   else  DecO(d0+s0.car, s.cdr, k0+1: k, d1, s1)
};
```

Программа memweight – число элементов множества

Вычисляет число элементов множества, представленного в виде битовой шкалы. *Вес* Хэмминга.

size_t memweight(**const void** *ptr, **size_t** bytes)

Указатель ptr фиксирует начало области памяти. Параметр bytes – размер области в байтах. Программа вычисляет число единиц в двоичном представлении области памяти.

int bitmap_weight(**const unsigned long** *src, **unsigned int** nbits)

Область памяти – массив слов src, nbits – размер области в битах.