29.10.21 ФМ-3 Лекция 3

Доказательство в системе Why3

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

Пример2: Задача 12

Исправление ошибок трансляции

Ошибки по недоказуемости

Методы декомпозиции доказательства

Использование лемм

Пример3: memweight – число элементов множества в виде битовой шкалы

Техника доказательства.

Методы локализации ошибок

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)
};
```

Дополнительные леммы

```
lemma ConsApp: forall u : stri. c: char.
                                  Cons c u = (Cons c Nil) ++ u
lemma AppDe: forall u v t r: stri. u++v = t++r ->
    u=t /\ v=r   \/
    exists x: stri. t = u++x /\  v = x++r
    exists y: stri. r = y++v /\  u = t++y
```

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.