Проекты для группы 3822М1ПМкн

Для получения зачета нужно выполнить проект и знать содержание лекций, в частности, информацию про теорию зависимых типов. Сдача проектов будет происходить до или одновременно со сдачей других зачетов.

Почти в каждом проекте нужно написать рекурсивную функцию и ее спецификацию, а затем доказать, что написанная функция удовлетворяет спецификации. Пример: нахождение максимума массива в лекции 6.

Проекты выполняются индивидуально или парами. Нужно выбрать одну из тем, приведенных ниже, и записать свой выбор в этой форме. Над каждой темой может работать не более одной команды. Темы распределяются в порядке поступления заявок. Прежде заполнения формы следует посмотреть в таблицу, в которой записываются поданные заявки, и выбрать тему, которая еще не задана. Если будет подано более одной заявки с одной и той же темой, то приоритет будет за первой командой.

После того, как вы напишете функцию и спецификацию и до начала доказательства пришлите спецификацию преподавателю для проверки на портале или по электронной почте (в теме письма укажите `[СПКН2023]` с квадратными скобками). Также присылайте все возникающие вопросы.

Главное правило при выполнении проекта заключается в следующем: у вас должно быть очень подробное доказательства требуемых утверждений на бумаге, прежде чем вы начнете реализовывать их на компьютере. В каждый момент доказательства на компьютере вы должны понимать, в каком месте бумажного доказательства вы находитесь и почему Coq предлагает вам доказать именно эти цели. Вы, а не компьютер, должны определять направление, в котором идет доказательство. Если вы начнете случайным образом подбирать тактики, надеясь, что это приведет к нужному результату, вы вряд ли добьетесь успеха.

Вспомогательные факты следует оформить как леммы. Например, так можно поступить с шагом индукции, чтобы основное доказательство было компактным.

Не забывайте скопировать инструкции `import` и определение тактики `bdestruct`, которые находятся в начале лекции 7.

В начале файла напишите номер и краткое описание проекта (одно-два предложения). Следует писать подробные комментарии. В частности, перед каждой функцией нужно написать, что она принимает и возвращает. Простые спецификации комментировать необязательно, но если в реализации функций или в формулах есть что-то неочевидное, это нужно описать.

Описание проектов

1. Определить для данных `f : nat → nat` и `n : nat`, является ли ограничение `f` на `{0, ..., n}` инъекцией.

2. Определить для данных `f : nat → nat` и `m n : nat`, имеет ли место `{0, ..., n} ⊆ {f 0, ..., f m}`. Таким образом, если `f` отображает `{0, ..., m}` в `{0, ..., n}`, нужно определить, является ли `f` сюръекцией.

3. Определить, встречается ли число в двумерном массиве. Массив моделируется функцией типа `nat → nat → nat`, количеством строк и количеством столбцов.

Сначала задачу нужно решить для одномерного массива. Для этого нужно дать следующее определение:

```
Fixpoint search1 (n x : nat) : bool := ...
```

и доказать

```
Theorem search1Spec :
  ∀ n x, (∃ i, i < n ∧ array i = x) ↔ search1 n x = true.
```

Эти определения и доказательства нужно поместить в раздел. Как описано в лекции 7, после закрытия этого раздела функция `search1` будет принимать дополнительный аргумент `array : nat → nat`, а спецификация `search1Spec` будет начинаться с `∀ (array : nat → nat)`.

В следующем разделе нужно написать функцию

```
Fixpoint search2 (m n x : nat) : bool := ...
```

и доказать

```
Theorem search2Spec:
  ∀ m n x,
    (∃ j k, j < m ∧ k < n ∧ array2 j k = x) ↔
    search2 m n x = true.
```

В определении `search2` нужно использовать `search1`, потому что если `array : nat → nat → nat`, то для каждого `j : nat` выражение `array j` имеет тип `nat → nat`. Аналогично в доказательстве `search2Spec` следует использовать `search1Spec`.

4. Найти максимум двумерного массива. Массив моделируется, как в предыдущей задаче. Как и там, задачу сначала нужно решить для одномерного массива и затем использовать этот результат.

5. Проверить, является ли массив `a` палиндромом, то есть имеет ли место `∀ i j, i + j = n → a i = a j`.

Предлагается написать вспомогательную функцию `isPalindromeHelp : nat → nat → bool`, такую что, например,

```
isPalindromeHelp 0 3
= a 0 =? a 3 && isPalindromeHelp 1 2
= a 0 =? a 3 && a 1 =? a 2 && isPalindromeHelp 2 1
= a 0 =? a 3 && a 1 =? a 2 && a 2 =? a 1 && isPalindromeHelp 3 0
= a 0 =? a 3 && a 1 =? a 2 && a 2 =? a 1 && a 3 =? a 0
```

В этом проекте не следует использовать усеченную разность на натуральных числах, обозначенную обычным минусом в стандартной библиотеке.

6. Проверить, является ли список палиндромом.

7. Найти наибольший общий делитель чисел `m` и `n` с помощью алгоритма Евклида. Необходимо использовать сильную (возвратную) индукцию, как описано в лекции 8.

8. Найти наибольший общий делитель чисел `m` и `n`, перебирая числа от `min m n` до `1`.

В Coq есть операция `m mod n` (или `modulo m n`), которая возвращает остаток от деления `m` на `n`. Однако поскольку все функции в Coq являются тотальными, значение выражения `m mod 0` равно `0` по определению. Также есть предикат `(n | m)` (пишется в скобках; или `divide n m`), который равен `True`, если `n делит m`. Следует обратить внимание, что `modulo` возвращает `nat` и следовательно может быть использован в программах, в то время как `divide n m` возвращает `Prop` и может использоваться только в спецификациях.

9. Проверить, является ли число `n` простым с помощью перебора всех потенциальных делителей от `2` до `n-1`. При желании можно перебирать числа до квадратного корня из `n`.

10. Найти целочисленный корень уравнения `f x = y`. Рассмотрим следующие предположения (должны быть объявлены внутри `Section`).

```
Variables f g : nat → nat.
Variable y : nat.

Hypothesis f_unbounded : ∀ n, n < f (g n).
Hypothesis f0 : f 0 ≤ y.
```

Гипотеза `f_unbounded` говорит, что `f` неограниченна, а функция `g` при этом выступает в качестве свидетеля. Целочисленным корнем уравнения `f x = y` называется такой `x`, что `f x ≤ y < f (x+1)` (здесь двойное неравенство есть сокращение для конъюнкции неравенств; такая запись возможна и в Coq). Нужно написать функцию, которая в предположениях неограниченности `f` и `f 0 ≤ y` находит целочисленный корень. Функция может перебирать значения `x`, начиная с числа, предшествующего `f (g y)`, до 0 и проверять, являются ли они корнями уравнения. Проверьте работу функции при `f x = x×x` и `g n = n+1`.

11. Рассмотрим представление натуральных чисел по основанию `b > 1`. Цифры числа содержатся в массиве типа `nat → nat`, причем самый младший разряд имеет индекс 0. Нужно написать следующие функции.

```
Fixpoint carry (a1 a2 : nat → nat) (n : nat) : nat := ...
```

Возвращает перенос с n-го на (n+1)-й разряд при сложении чисел, представленных массивами `a1` и `a2`.

```
Definition add (a1 a2 : nat → nat) (n : nat) : nat := ...
```

Возвращает n-ую цифру суммы чисел, представленных массивами `a1` и `a2` (n ≥ 0).

```
Fixpoint toNat (a : nat → nat) (n : nat) : nat := ...
```

Возвращает число, представленное массивом `a`.

Также требуется доказать, что функции `carry` и `add` определены корректно, то есть число, представленное массивом `add a1 a2`, является суммой чисел, представленных `a1` и `a2`.

12. Данный проект аналогичен предыдущему, но цифры двух чисел хранятся в двух списках. Функция `add` также возвращает список.

13. Вычислить значение полинома в точке `x` методом Горнера и доказать, что оно равно значению, вычисленному обычным образом. Для данного `n` массив a представляет полином `(a 0)*x^n + (a 1)*x^(n-1) + ... + a n`. Про метод Горнера см. упражнение 3 в домашнем задании 6 из прошлого семестра. Под вычислением обычным образом имеется в виду функция, также определенная рекурсией по `n`, но которая вычисляет `(a i)*x^i` для каждого монома и возвращает сумму результатов.

14. Данный проект аналогичен предыдущему, но коэффициенты многочлена хранятся в списке. Младший коэффициент находится в голове списка.

15. Реализовать быстрое умножение двоичного числа на число Пеано. Двоичное число представлено массивом из `0` и `1` длины `n+1`. Старшие биты числа, как обычно, находятся слева. Второе число — это просто элемент типа `nat`. Требуется реализовать метод умножения, описанный в прошлом семестре. Также нужно написать функцию `binToNat : nat → nat`, такую что `binToNat n` возвращает число, представляемое массивом длины `n+1`. Спецификация алгоритма, которую требуется доказать, говорит

```
∀ n y, fastMult n y = (binToNat n) × y.
```

Можно написать гипотезу (`Hypothesis`, внутри `Section`), говорящую, что массив содержит только нули и единицы.

16. Данный проект аналогичен предыдущему, но цифры первого числа хранятся в списке (младший бит находится в голове списка).

17. Даны следующие функции и предположения.

```
Variables f, g : nat → nat.

Hypothesis g_not_surj : ∀ n, g n ≠ 0
```

Таким образом, `g` не является сюръекцией. Нужно доказать, что композиция `f ∘ g ∘ f` не является тождественной функцией, то есть найдется `n`, такой что `f (g (f n)) ≠ n`. Существует конструктивное и неконструктивное доказательства этого факта. Нужно написать конструктивное доказательство, то есть написать алгоритм, который находит `n`, и доказать, что `f (g (f n)) ≠ n`. Полученный алгоритм достаточно прост, но найти его может быть не совсем тривиальной задачей.

Если у вас есть другая идея для проекта, вы можете обсудить ее с преподавателем.

This page has been generated by coqdoc