10. Доказательство свойств рекурсивных функций

Какие свойства? – **правильность**:

- *Завершаемость* работы функциональной программы.
- *Корректность* (действительно: выполняет свою работу)

Доказательство свойств рекурсивных функций осуществляется при помощи индукции, точнее, ***методом математической индукции***.

Замечание: существует глубокая внутренняя связь рекурсии и индукции.

Существует несколько версий (методов/принципов) математической индукции (различающихся по мощности).

1. **Простая (восходящая) индукция:**

Пусть P(n) – утверждение (высказывание) о целом числе, гипотеза индукции.

$P(0),$

$\underline{\forall n(P(n) \supset P(n+1))} \quad \forall n \geq 0,\ n \in \mathbb{N}$

$\forall m\ P(m)$

Тогда в общепринятой форме правил логического вывода (из справедливости для n следует справедливость для n+1).

Замечания:

а) Это правило интуитивно очевидно, но при **аксиоматической трактовке целых чисел** сам принцип математической индукции в такой формулировке следует рассматривать как аксиому.

б) Возможные обобщения/модификации:

$P(0) \rightarrow P(1) \rightarrow P(n_0)\ \ n_0 \in \mathbb{N}$

Тогда во второй посылке $\forall n \geq n_0$

Нисходящая индукция: переход во второй посылке правила $n \rightarrow n-1$ (а не $n \rightarrow n+1$).

2. **Строгая индукция (для натуральных чисел)**

Для неё характерна более строгая (сильная) гипотеза:

$P(0),$

$\underline{\forall n(P(n) \& P(1) \& \ldots P(n) \supset P(n+1))} \quad \forall n \geq 0,\ n \in \mathbb{N}$

$\forall m\ P(m)$

или

$P(0),$

$\underline{\forall n\{\forall k \geq 0(k \leq n \supset P(k)) \supset P(n+1)\}} \quad \forall n \geq 0,\ n \in \mathbb{N}$

$\forall m\ P(m)$

Такая версия необходима, т.к. есть утверждения, которые нельзя доказать методом простой индукции, но можно доказать методом строгой индукции, например:

Для чисел Фиббоначи ($f_0=f_1=1$, $\forall i \geq 2\ \ f_{i+1}=f_i+f_{i-1}$) P: $f_i \leq [(1+\sqrt{5})/2]^{i-1}\ \ i \geq 1$

Замечания: возможны аналогичные модификации строгой индукции:

а) первая посылка: $P(0) \rightarrow P(1) \rightarrow P(n_0)$ $n_0 \in \mathbb{N}$

б) нисходящая индукция (вторая посылка правила):

$\forall n\{\forall k(k>n \supset P(k)) \supset P(n)\}$

Для работы не только с числами, но и с другими типами данных, необходимо обобщение строгой индукции (обобщение порядка min/max элемента).

3. **Обобщённая индукция (полная)**

(для произвольного вполне упорядоченного или обоснованно-упорядоченного множества)

Предполагается, что в рассматриваемом множестве S введено бинарное отношение порядка (S,<) с определёнными свойствами.

Определение. Говорят, что (S,<) ***вполне упорядоченно***, если:

1. $\forall x,y,z \in S$ $x<y$, $y<z \supset$ $x<z$ (транзитивность отношения);
2. $\forall x,y \in S$ либо $x<y$, либо $x>y$, либо $x=y$ (полнота порядка);
3. $\forall$ $A \neq \varnothing \subset S$ (для непустого подмножества S)

   $\exists x_a \in A$: $\forall y \in A$ $x_a \leq y$ (существование минимального элемента у любого непустого подмножества S, в том числе – и у самого S)

Замечание: условие 3 по сути – отсутствие в S бесконечно убывающих последовательностей элементов.

Примеры вполне упорядоченных множеств:

1. $(\mathbb{N}, <)$
2. $(\mathbb{N} \times \mathbb{N}, <)$: множество пар неотрицательных целых чисел вполне-упорядочено с помощью отношения **лексико-графического порядка**: $(n_1, n_2)<(n_3, n_4) \Leftrightarrow n_1<n_3$ либо $n_1=n_3$, $n_2<n_4$

Но! $(\mathbb{R}_+, <)$ не является вполне упорядоченным (неверно условие 3)

Сформулируем правило вывода по индукции для **вполне упорядоченного** множества (S, <)

x_0 – наименьший элемент

нужно доказать для $\forall x$: $x>x_0$, что если справедливо P(y) для $\forall y<x$, то справедливо и P(x)

$P(x_0)$,

$\underline{\forall x\{\forall y(y<x \supset P(y)) \supset P(x)\}}$

$\forall z\ P(z)$

Замечания:

a. в справедливости метода/правила обобщённой индукции можно убедиться, рассуждая от противного;
b. если S – это $\mathbb{N}$, а < имеет обычный смысл, то принцип обобщённой индукции идентичен принципу строгой индукции для чисел;
c. ослабляя гипотезу/конкретизируя S и < можно получать разные версии;

d. это правило – восходящая или нисходящая индукция? Строго формально – восходящая по отношению порядка, но можно отношение порядка переопределить так, чтобы применить эту версию (или сформулировать аналогично нисходящую обобщённую индукцию, заменив < на >).

Будем работать с рекурсивными программами, обрабатывающими списки. Можно ли ввести полный порядок во множестве всех лисповских списков S? (т.е. сделать его вполне упорядоченным?) – Нет, т.к. не будет выполняться свойство 2 определения.

Например, $l_1<l_2$ если длина l_1 < длины l_2 (количество элементов на верхнем уровне), то несравнимы списки одинаковой длины, т.е. мы ввели частичный, а не общий порядок.

Как быть? На самом деле, ничего плохого нет, так как оказывается, что доказательство методом обобщённой индукции можно проводить и для множеств с частичным порядком, для которых справедливы свойства 1 и 3.

Определение. Говорят, что множество (S,<) с бинарным отношением порядка является ***обоснованно-упорядоченным***, если:

1. $\forall x,y,z \in S$ $x<y$, $y<z \supset$ $x<z$ (транзитивность отношения);
3. $\forall$ $A \neq \varnothing \subset S$ (для непустого подмножества S)

   $\exists x_a \in A$: $\forall y \in A$ (сравнимого с x_a) $x_a \leq y$ (существование минимального элемента)

Замечание: обоснованный порядок – более общий, чем вполне упорядочивающий.

Утверждение. Для доказательства свойств/утверждений об элементах множества, в котором установлен **обоснованный порядок**, можно использовать метод/принцип обобщённой математической индукции.

Наша цель: доказательство свойств рекурсивных программ (Лисп: рекурсия появляется вследствие того, что рекурсивно по своей природе определение списка и S-выражения).

Можно ли применять введённое правило обобщённой математической индукции? Да – можно и нужно, если на множестве данных, с которыми работает рекурсивная программа, установлен обоснованный порядок. Применение обобщённой индукции тогда по сути – *индукция по структуре данных* программы, поэтому такой метод индукции для доказательства свойств рекурсивных программ называется ***структурной индукцией***.

Схема доказательства правильности рекурсивной программы/функции методом структурной индукции:

1. Доказать, что программа работает правильно для простейших данных (аргументов функции).

2. Доказать, что программа работает правильно для более сложных данных (аргументов функции) в предположении, что она работает правильно для более простых данных (если отношение < интерпретируется как возрастающая сложность).

Замечания:

a. Интуитивная очевидность схемы: такая схема доказательства естественно соответствует основной схеме вычислений в рекурсивных программах.
b. Логическое обоснование метода – формализация интуитивного понятия о более простых/сложных данных, т.е. введение на множестве значений данных нужного отношения порядка; множество S данных должно быть либо вполне-упорядоченным, либо обоснованно-упорядоченным.

Вспомним правило АПР:

Правило самого левого и самого внутреннего вызова/обращения:

В любой момент вычислений первым начинает вычисляться самое левое из самых внутренних обращений к функции (из тех функциональных обращений, аргументы которых уже не содержат других функциональных вызовов).

АПР самый распространённый при реализации различных языков программирования. Если определямые вычисления заканчиваются при АПР, то результат вычислений (функциональное значение) будет тот же, что и при других правилах вычислений, приводящих к окончанию. В то же время АПР не всегда наилучший: иногда может приводить к неоканчивающейся последовательности вычислений, в то время как НПР может дать конечную последовательность вычислений.

Итак, АПР не лучший, но и не худший, но т.к. большинство рассматриваемых рекурсивных программ заканчивается при любых значениях аргументов независимо от того, какие правила вычислений используются, то результат во всех случаях будет одинаковым, и будем для определённости считать, что работает АПР.

Примеры доказательства правильности рекурсивных программ/функций

Пример 1. Доказательство правильности функции вычисления факториала:

```
(defun fact(X)
   (cond  ((= X 1)1)
          (T (* X (fact (- X 1))))))
```

Т.е. докажем, что а) функция заканчивает свои вычисления

б) `fact(n)=n!=1*2*3*…n` для $\forall n \geq 1$

Структурная обобщённая индукция для $\mathbb{N}_+$ (вполне упорядоченное множество) => простая индукция по положительным целым числам.

(1) Для `n=1` функция заканчивает работу и `fact(n)=1=1!`

(2) Пусть (предполагаем) функция заканчивает работу для n≥1 и fact(n)=n! (гипотеза индукции). Надо доказать, что это верно для n+1: т.к. n+1≠1, то работает вторая ветвь функции и

fact(n+1)=(n+1)*fact(<u>(n+1)-1</u>)=(n+1)*n!=(n+1)!

=n

=n! по гипотезе индукции

Очевидно при этом, что функция заканчивает свою работу.

Таким образом, fact(n)=n! ∀n≥1 + завершаемость вычислений.

Пример 2. Доказательство правильности функции append

```
(defun append(L1 L2)
    (cond  ((null L1) L2)
           (T (cons (car L1)(append (cdr L1) L2)))))
```

Завершаемость опускаем; осталось доказать, что append

- применима к любым двум спискам L1 и L2;
- в качестве результата даёт конкатенацию (слияние) двух списков – т.е. список, состоящий из элементов верхнего уровня списка L1, за которыми следуют элементы верхнего уровня списка L2.

Множество данных программы? Вообще: S×S (два аргумента), но т.к. в рекурсии и упрощении участвует только первый аргумент, то будем рассматривать только L1: S. Порядок, формализующий понятие простоты/сложности данных?

L1 $<_L$ L2, если длина (L1) < длина(L2)

Получим т.о. обоснованно-упорядоченное множество (S,$<_L$), минимальный элемент – пустой список NIL.

Теперь проводим обобщённую индукцию по (S,$<_L$):

(1) Для L1=() работает первая ветвь функции (append () L2)=L2 очевидно, доказываемый факт (свойство функции выполняется).

(2) Предположим, что append правильно работает для ∀ L1' $<_L$ L1 (длина которого меньше длины списка L1) – т.е. (apend L1' L2) есть список, составленный из элементов L1', за которыми следуют элементы из L2. (гипотеза индукции)

Докажем, что append правильно работает и для списка L1 длины n

Из гипотезы индукции предполагается, что n>0, следовательно работает вторая ветвь функции:

(append L1 L2)=(cons (car L1)<u>(append (cdr L1) l2)</u>)

Т.к. (cdr L1) $<_L$ L1 – по гипотезе это список, составленный из всех элементов L1 кроме первого, за которыми следуют элементы из L2. После применения cons получаем список, у которого первый элемент – первый элемент L1, затем идут последовательно оставшиеся элементы L1 (в нужном порядке), а затем все элементы из L2, т.е. все элементы L1 и L2 в нужном порядке! Что и требовалось доказать.

Замечания:

a. На самом деле нам для доказательства не нужна была столь строгая гипотеза ($\forall$ L1' $<_L$ L1), достаточно было доказать в пункте 2, что если append правильно работает для cdr некоторого списка, то она правильно работает для самого этого списка. Такой гипотезе соответствует более простая версия обобщённой индукции, называемая ***рекурсивной восходящей индукцией для списков*** с правилом вывода

$$\frac{P(NIL),\quad \forall L\{\neg null(L)\&P(cdr(L)) \supset P(L)\}}{\forall L\ P(L)}$$

Она аналогична простой индукции для натуральных чисел.

Утверждение. Рекурсивной восходящей индукции для списков достаточно для доказательства любых свойств рекурсивных функциональных программ, построенных по следующей рекурсивной схеме:

f(y) <= if null(y) then a else g(y, f(cdr(y)))

где a – константа, не зависящая от y, g – композиция функций (за исключением самой f).

По сути: это схема последовательной обработки всех элементов верхнего уровня заданного списка. append удовлетворяет этой схеме.

b. Доказательство правильности append можно было бы провести и на основе простой индукции для натуральных чисел, используя тот факт, что вполне упорядоченным является множество значений некоторых числовых свойств данных программы (т.е. списков), а именно: множество длин списков. Длина списка – количество элементов на верхнем уровне – неотрицательное целое число, а обычное отношение < вполне упорядочивает множество $\mathbb{N}$. Таким образом, множество S тображаем в множество $\mathbb{N}$ и структурная индукция для списков сводится к обычной, простой индукции для натуральных чисел.

Пример 3. Доказательство правильности функции flatten1, раскрывающей в произвольном S-выражении все внутренние скобки:

```
(defun flatten1(X)
   (cond  ((null X)NIL)
          ((atom X)(cons X NIL))
          (T (append (flatten1 (car X))
                     (flatten1 (cdr X))))))
```

Необходимо доказать, что $\forall X \in S$ выполняются 2 свойства:

(1) Результат (значение) функции – одноуровневый список (без внутренних, вложенных скобок).

(2) Результат-список содержит только атомы из исходного X, причём в том же порядке.

Как ввести отношений порядка, чтобы можно было сравнивать не только (cdr L) и L, но и (car L) и L?

Например, так: $S1 <_S S2 \Leftrightarrow n(S1) < n(S2)$

где n(S) – общее число вхождений скобок и атомов в выражение S.

Например, n=7 для S=>`(a (b) d)`

Так введённое $(S,<_S)$ обоснованно упорядочено (опять: проблема сравнимости S1 и S2 с n1=n2).

Доказательство методом обобщённой индукции для $(S,<_S)$

(1) Для минимального элемента в $(S,<_S)$ – атома легко проверить оба свойства, т.к. если

`X=NIL` – список без атомов (работает первая ветвь)

`X≠NIL` – один атом (работает вторая ветвь)

(2) Предположим, что эти два свойства верны и для `∀X'<X≠NIL` (для любого более простого списка `X'`) – гипотеза индукции

Покажем, что эти свойства верны и для `X`.

Т.к. работает третья ветвь функции `flatten1`, результат:

`(append (flatten1 (car X))(flatten1 (cdr X)))`

Аргументы обоих вызовов `flatten1` удовлетворяют свойствам 1 и 2, поскольку `(car X)` $<_S$ `X` и `(cdr X)` $<_S$ `X` `∀X`

а `append` "сохраняет" – по своему определению – эти свойства, то результат обладает указанными свойствами. Что и требовалось доказать.

Замечания:

a. Это доказательство может быть проведено также строгой индукцией для натуральных чисел, поскольку введённое n(S) – общее число вхождений скобок и атомов в список S, отображает множество S в множество $\mathbb{N}$ (опять: вполне упорядоченным является множество значений свойств списков).

b. Логическая структура S-выражения – дерево, поэтому для доказательства свойств рекурсивных программ, обрабатывающих элементы списка на всех уровнях, достаточна следующая версия математической индукции – ***индукция для деревьев***:

`∀y(atom(y) ⊃ G(y)),`

`∀ y{¬atom(y)&G(left(y))&G(right(y)) ⊃ G(y)}`

`∀x G(x)`

где `G` – некоторое свойство/предикат, `left` и `right` – левое и правое поддеревья.

Соответствующая схема функции (для которой производится доказательство):

```
f(y)<=if atom(y) then h(y)
                 else g(y, f(left(y)), f(right(y)))
```

где `h` и `g` – суперпозиции других (элементарных) функций.

Пример 4. Правильность программы слияния двух списков, содержащих упорядоченные по неубыванию числа, в результирующий упорядоченный по неубыванию список чисел.

```
(defun merge(L1 L2)
   (cond  ((null L1) L2)
          ((null L2) L1)
          ((< (car L)(car L2))
              (cons (car L1)(merge (cdr L1) L2)))
          (T (cons (car L2)(merge L1 (cdr L2))))))
```

Доказываемые свойства (правильность):

(1) Все элементы-числа результирующего списка принадлежат `L1` и `L2`.

(2) Результирующий список упорядочен по неубыванию.

Множество данных функции `merge` – пара списков, причём при рекурсивных вызовах либо первый аргумент, либо второй становится проще, но обработка обоих аргументов производится только на верхнем уровне.

Отношение порядка, вполне упорядочивающее (или обоснованно упорядочивающее) множество данных S×S – пар списков `<L1, L2>`? Обоснованный порядок:

`<L1',L2'>` $<_L$ `<L1,L2>` ⇔ `<n1',n2'>` < `<n1,n2>`

Где n – длины списков (количество элементов на верхнем уровне), а < – отношение лексикографического порядка на $\mathbb{N} \times \mathbb{N}$.

Итак, по существу: метод обобщённой индукции по парам возможных длин `<n1,n2>` списков `<L1,L2>` (лексикографически упорядоченных на основе числового порядка для $\mathbb{N}$), т.е. ({`<n1,n2>`},<) – вполне упорядоченное множество значений числовых свойств пар списков `<L1,L2>`.

(1) Наименьший элемент множества $\mathbb{N} \times \mathbb{N}$ – `(0,0)`, соответствующие списки `L1` и `L2` – пустые, и функция `merge` работает правильно (результирующий список пуст).

(2) Необходимо доказать для произвольной пары списков `L1` и `L2` с длинами `n1` и `n2`: `<n1,n2> > <0,0>`, что если для любых `L1'` и `L2'` таких, что `<n1',n2'> < <n1,n2>` функция `merge` работает правильно (гипотеза индукции), то она правильно работает и для `L1` и `L2` длины `n1` и `n2` соответственно.

Предположим, что гипотеза индукции верна. Рассмотрим случаи:

a. `L1` или `L2` равен `NIL` – легко проверить, что `merge` работает правильно.

b. `(car L1)<(car L2)`, тогда результат функции –
   `(cons (car L1)(merge (cdr L1) L2))`
   длина `(cdr L1)` равна `n1'` и меньше длины `n1` списка `L1`, т.е. для аргументов `merge` пара длин `<n1',n2> < <n1,n2>` и по гипотезе индукции `(merge (cdr L1)L2)` удовлетворяет свойствам 1 и 2. А после применения `cons` получаем список, удовлетворяющий этим свойствам (те же самые элементы + упорядоченность, т.к. `(car L1)` не больше любого элемента из `(cdr L1)` и из `L2`, т.е. самый меньший из всех элементов `L1` и `L2`). Что и требовалось доказать.

c. (car L1)≥(car L2) – этот случай доказывается аналогично, т.к. результат – (cons (car L2)(merge L1 (cdr L2))
длина (cdr L2) равна n2' и меньше длины n2 списка L2.

Замечание: можно было ввести отношение порядка другим образом:
<L1',L2'> $<_L$ <L1,L2> ⇔ n1'+n2' < n1+n2 (сумма длин списков)

Пример 5. Доказательство правильности функции f91 (Маккарти)

```
(defun f91(n)(cond ((> n 100)(- n 10))
                   (T (f91(f91(+n 11))))))   n ∈ℕ
```

$$f91(n)=\begin{cases} n-10, & n>100 \\ 91, & n\le 100 \end{cases}$$

Будем использовать нисходящую индукцию.

Соображение: при рекурсивном обращении аргумент увеличивается, а простой случай (ветвь) соответствует ∀n>100. Поэтому возьмём:
M≡{n ∈ℕ : n≤100}, отношение – обычное числовое отношение порядка <.
Получим т.о. вполне-упорядоченное множество M с максимальным элементом n_{max}=100. И доказываем лишь вторую строку: f91(n)=91 для n≤100.

Доказательство:

I. f91(100)=f91(f91(100+11))=f91(101)=91
работает первая ветвь функции при вычислении f91(111) и f91(101)

II. Нужно доказать для n<100, что если выполняется ∀n'>n f91(n')=91 (гипотеза индукции), то и для n f91(n)=91.

Рассмотрим два случая (когда можно пользоваться гипотезой и когда нет), поскольку во второй ветви два вложенных обращения к функции f91:

a. n+11 ∈ M, тогда n<n+11≤100, т.е. n≤89<91
f91(n)=f91(f91(n+11))=f91(91)=91
дважды применили гипотезу индукции при вычислении внутреннего, а затем внешнего обращения к функции f91.

b. n+11 ∉ M => n+11>100, т.е. n>89.
f91(n)=f91(f91(n+11))=f91(n+1)
n+11-10 (первая ветвь функции)
но f91(n+1)=91 по гипотезе индукции, т.к. n+1 ∈ M (n<100 => n+1≤100) и n+1>n.

Замечание: здесь используется АПР.

Методика доказательства:

- Определение множества значений изменяемых во время рекурсии данных функции.
- Введение порядка < таким образом, чтобы это множество было вполне упорядоченным.

- Доказательство методом обобщённой индукции (или её упрощённой версией).
  При этом учитывается:
  a. Количество аргументов функции (упрощаемых рекурсивно).
  b. Характер рекурсии (в глубину / в ширину)
  c. min/max элемент => восходящая/нисходящая рекурсия.

Задачи:

1. Доказать правильность функции `reverse1` (без накапливающего параметра), переворачивающей свой аргумент-список на верхнем уровне.
2. Функция-предикат `linp` проверяет, можно ли записать произвольное S-выражение в виде бесточечного списка. Запрограммировать эту функцию и доказать её правильность.
3. Составить функцию `interord` от двух аргументов – упорядоченных по неубыванию списков чисел, которая строит упорядоченное пересечение этих двух списков (т.е. упорядоченный список чисел, входящих как в первый, так и во второй список-аргумент), причём за один проход по спискам-аргументам. Доказать её правильность.
4. Доказать, что для любых списков `X`, `Y` и `Z`
   `(rev (append X Y))≡(append (rev X)(rev Y))`
   где rev – функция, переворачивающая элементы заданного списка на верхнем уровне
   `(append X (append Y Z))≡(append (append X Y) Z)`