Методы построения и анализа алгоритмов

Малышкин Виктор Эммануилович

Кафедра Параллельных Вычислительных Технологий
Новосибирский государственный технический университет

E_mail: **malysh@ssd.sscc.ru**
Телефон: **3308 994**

Новосибирск

Общая характеристика курса

1. Рассматривается неформальное понятие алгоритма и его представления
2. Тема курса очень обширна, доступно много учебных материалов. В 8 лекций материал невозможно уложить так, чтобы изучить проблемы и научиться пользоваться знанием. Потому для большей широты покрытия материала обсуждаются идеи, а остальное - самостоятельная работа студента.
3. Обсуждаются свойства алгоритмов, разработка алгоритмов
4. Вводится понятие сложности алгоритма
5. Рассматриваются и анализируются алгоритмы из различных предметных областей
6. Лекции и семинары/лабораторные рассматривают детально приложения.

Рекомендуемые учебники

- Ахо, Альфред, В., Хопкрофт, Джон, Ульман, Джеффри, Д. *Структуры данных и алгоритмы.* : Пер. с англ. : Уч. пос. — М. : Издательский дом "Вильяме", 2000. — 384 с.
- Кормен Т., Лейзерсон Ч., Риверс Р., Штайн К. *Алгоритмы. Построение и анализ* – М.: «Вильямс», 2012
- В.Э.Малышкин, В.Д.Корнеев. *Параллельное программирование мультикомпьютеров*. – В серии «Учебники НГТУ», Новосибирск, изд-во НГТУ, 2011, 296 стр. (есть в библиотеке

ДОПОЛНИТЕЛЬНЫЕ

- М.Гэри, Д.Джонсон. *Вычислительные машины и труднорешаемые задачи* // М. Мир, 1982, 416 стр.
- Седжвик Роберт. *Фундаментальные алгоритмы на С++. Анализ/Структуры данных/ Сортировка/Поиск*: Пер. с англ./Роберт Седжвик. - К.: Издательство «ДиаСофт», 2001.- 688 с.

Лекции 1 и 2
АЛГОРИТМ И ЕГО СВОЙСТВА

Основная задача лекций – обсудить неформально понятие алгоритма, его полезные свойства, его представления для тех или иных целей

1. Дается и обсуждается формальное определение алгоритма
2. Рассматривается связь алгоритма и программы:

$$Задача \rightarrow модель\_задачи \rightarrow F \rightarrow A \rightarrow P$$

3. Обсуждаются проблемы конструирования программы, реализующей алгоритм

Краткая классификация алгоритмов

ЗАДАЧИ	СВОЙСТВА
Численные Системные Комбинаторные Оптимизационные Прикладные (машиностроение, космос, наука, экономика, …) Теоретические Технологические И прочие и прочие	Параллельные Высокоточные Интенсивные вычисления Интенсивные данные Быстрые Динамические Настраивающиеся Самоорганизация Минимальное потребление ресурсов Надежность И прочие и прочие

ПОНЯТИЕ ВЫЧИСЛИМОЙ ФУНКЦИИ

- Формализация Клини понятия вычислимой функции и алгоритма нам понадобится для формирования правильной исходной точки зрения при анализе, конструировании и размышлениях об алгоритмах и программах.
- Понятие вычислимой функции - ключевое понятие из тех, которыми должен владеть программист и вообще всякий, работающий в области вычислений. Задача лекции - дать начальные понятия теории алгоритмов в ее приложении к программированию

- Говоря о программе, всегда будем иметь ввиду всевозможные способы ее исполнения, включая частный случай - последовательное исполнение. Термин *последовательная программа* используется, чтобы подчеркнуть единственно допустимый способ исполнения программы.
- Каждый легко может определить, является ли некоторый предъявленный процесс алгоритмом или нет. Например, вычисление корней квадратного полинома по формуле Виета есть алгоритм. Однако без формализации понятия алгоритма невозможно показать, что **не существует** алгоритмического решения некоторой проблемы

Неформальные свойства алгоритмов

- а).Алгоритм - это процесс последовательного построения величин, идущий в дискретном времени таким образом, что в начальный момент задается исходная конечная система величин, а в каждый следующий момент новая конечная система величин получается по определенному закону (программе) из системы величин, имеющихся в предыдущий момент времени (*дискретность алгоритма*).

- б).Система величин, полученная в какой-то (не начальный) момент времени, однозначно определяется системой величин, полученных в предшествующие моменты времени (*детерминированность алгоритма*).
- в).Закон получения последующей системы величин из предшествующей должен быть простым и локальным (*элементарность шагов алгоритма*).

- г).Если способ получения последующей величины из какой-нибудь заданной величины не дает результата, то должно быть указано, что надо считать результатом алгоритма (*направленность алгоритма*).

- д).Начальная система величин может выбираться из некоторого потенциально бесконечного множества (*массовость алгоритма*).

- Каждый алгоритм A вычисляет функцию F_A при некоторых заданных значениях входных величин. Функции, вычисляемые алгоритмом, называются *алгоритмически вычислимыми* функциями. Понятие вычислимой функции, так же как и понятие алгоритма, тоже здесь интуитивно.
- Существует много разных формализаций понятия алгоритм, удовлетворяющих неформальным требованиям. Однако класс алгоритмически вычислимых функций, определенный во всех таких формализациях оказался одним и тем же при самых разнообразных попытках расширить понятие алгоритма.

Аксиоматическое построение теории вычислимых функций и алгоритмов

- Строим теорию как аксиоматическую теорию, а именно, сначала выбирается множество простейших функций. Они объявляются вычислимыми “по определению”. Простейшие функции выбираются таким образом, чтобы в их вычислимости ни у кого не было никаких сомнений.

- Затем определяются *операторы* - схемы конструирования новых вычислимых функций из имеющихся. Каждый оператор будет определен таким образом, что в его вычислимости тоже не возникнет сомнений. Применение этих операторов к вычислимым функциям вновь должно дать вычислимую функцию. Необходимо так выбрать операторы, чтобы с их помощью (конечной комбинацией их применений) можно было построить любую вычислимую функцию.

Аксиоматическое построение теории вычислимых функций и алгоритмов

- **АКСИОМЫ** (простейшие вычислимые функции)

На первом шаге выбирается счетный *базис* простейших *рекурсивных* функций, вычислимых по определению. Это следующие функции:

1. Функция следования $s(x)=x+1$
2. Нулевая функция $o(x)=0$
3. Функции выбора $I_m^n(x_1,x_2,\ldots,x_n)=x_m,\ 1 \le m \le n$

Функции устроены крайне просто. Ни у кого, видимо, нет сомнений в интуитивной вычислимости этих функций, в возможности построить некоторое “механическое” устройство, вычисляющее эти функции.

ОПЕРАТОРЫ

1.СУПЕРПОЗИЦИЯ ВЫЧИСЛИМЫХ ФУНКЦИЙ

Пусть заданы n частичных функций m переменных $f_i^m{:}D_1\times D_2\times\ldots\times D_m\to D_o$, $i=1, 2,\ldots, n$, и пусть определена функция $f^n{:}D_0\times D_0\times\ldots\times D_0\to D$. Определим функцию m переменных g^m, определенную на множестве $D_1\times D_2\times\ldots\times D_m$ со значениями в D, $g_m(x_1,x_2,\ldots,x_m)=f_n(f_1^m(x_1,x_2,\ldots,x_m),\ldots,f_n^m(x_1,x_2,\ldots,x_m))$. Функция g^m конструируется *суперпозицией* (*подстановкой*) из функций f^n, f_1^m, ... , f_n^m. Оператор суперпозиции будем обозначать $\mathbf{S}^{n+1}$.

Понятно, что функция $g_m(x_1,x_2,\ldots,x_m)$ и есть та функция, которую определяет функциональный терм $f_n(f_1^m(x_1,x_2,\ldots,x_m),\ldots,f_n^m(x_1,x_2,\ldots,x_m))$ при соответствующей интерпретации.

- Программисты легко могут представить себе наличие процедур, вычисляющих простейшие вычислимые функции. Оператор суперпозиции определяет, с позиций программирования, простое “сцепление” процедур, при котором одни процедуры вырабатывают значения своих выходных переменных, а другие их использует в качестве входных величин. Понятен и “механический” характер определения оператора суперпозиции, его очевидная вычислимость, а именно: если известно, как вычислить значения функций $f, f_1, \ldots, f_n$, то понятен и алгоритм вычисления функции
- $g^m(x_1,x_2,\ldots,x_m)=f^n(f_1(x_1,x_2,\ldots,x_m), \ldots, f^n(x_1,x_2,\ldots,x_m))$.

Каждый программист легко узнает в нижеследующей программе

“реализацию” оператора суперпозиции.

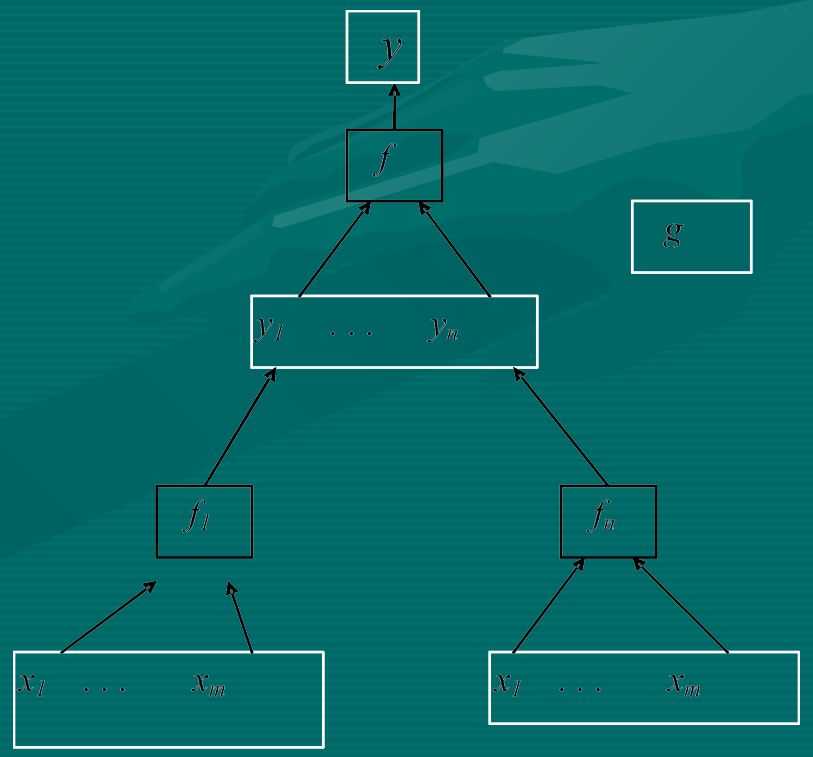
Пусть процедуры $f, f_1, \ldots, f_n$ вычисляют одноименные функции. Тогда программа P вычисляет функцию g.

P: $y_1 := f_1(x_1, x_2, \ldots, x_m)$;

. . .

$y_n := f_n(x_1, x_2, \ldots, x_m))$

$y := f(y_1, y_2, \ldots, y_n)$;



2.Оператор примитивной рекурсии

Оператор примитивной рекурсии позволяет определить циклические вычисления специального вида.

Пусть заданы:

- n-местная вычислимая функция g и
- $(n+2)$-местная вычислимая функция h.

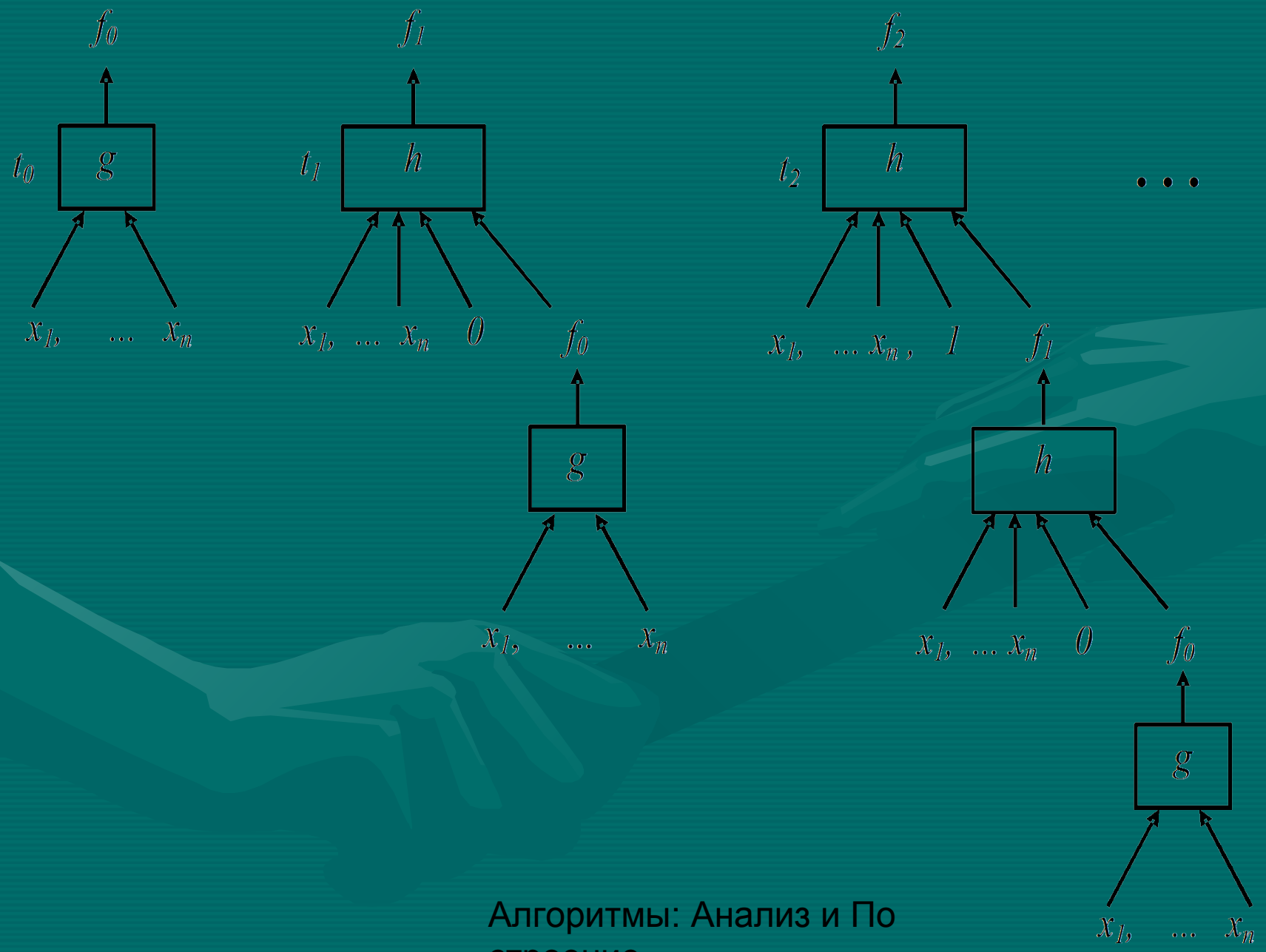
Тогда $(n+1)$-местная функция f строится *примитивной рекурсией* из функций g и h, если для всех натуральных значений $x_1, x_2, \ldots, x_n, y$ имеем:

$$f(x_1, x_2, \ldots, x_n, 0) = g(x_1, x_2, \ldots, x_n)$$

$$f(x_1, x_2, \ldots, x_n, y+1) = h(x_1, x_2, \ldots, x_n, y, f(x_1, x_2, \ldots, x_n, y))$$

- Соотношения называются *схемой примитивной рекурсии*. Они определяют *оператор примитивной рекурсии* **R**. Эта схема, собственно, и есть алгоритм вычисления функции *f*. Оператор **R** определен на множестве ϕ частичных вычислимых функций, $f=\mathbf{R}(g,h)$.
- Понятно, что функция *f* существует и единственна. Это следует из того, что определение функции, построенной примитивной рекурсией, фактически содержит схему (алгоритм) ее вычисления. Распишем детально эту схему, используя термальные представления.

- Пусть необходимо вычислить значение функции f при $y=k$. Тогда из определения (1) схемы примитивной рекурсии имеем последовательность функциональных термов $t_0, t_1, \ldots, t_k$, вычисляющих значение функции $f(x_1, x_2, \ldots, x_n, k)$:
- t_0: $f_0=f(x_1,x_2,\ldots,x_n,0)=g(x_1,x_2,\ldots,x_n)$
- t_1: $f_1=f(x_1,x_2,\ldots,x_n,1)=h(x_1,x_2,\ldots,x_n,0,g(x_1,x_2,\ldots,x_n))$
- t_2: $f_2=f(x_1,x_2,\ldots,x_n,2)= h(x_1,x_2,\ldots,x_n,1, f(x_1,x_2,\ldots,x_n,1))$
- .....
- t_k: $f_k=f(x_1,x_2,\ldots,x_n, k)=h(x_1,x_2,\ldots,x_n,k-1,f(x_1,x_2,\ldots,x_n,k-1))$
- Здесь представлен способ вычисления f, следовательно, функция f определена однозначно. Если одна из функций $f(x_1,x_2,\ldots,x_n,m)$, $m<k$, не определена, то и значение $f(x_1,x_2,\ldots,x_n,y)$ для всех $y\geq m$ тоже не определено. Эту последовательность $k+1$ функциональных термов $t_0, t_1, \ldots t_k$ удобно для наглядности изобразить графически.


f_0
t_0
g
$x_1, \ldots x_n$
f_1
t_1
h
$x_1, \ldots x_n$ 0
f_0
g
$x_1, \ldots x_n$
f_2
t_2
h
$x_1, \ldots x_n, 1$
f_1
h
$x_1, \ldots x_n$ 0
f_0
g
$x_1, \ldots x_n$
. . .

- Это множество функциональных термов и определяет алгоритм вычисления функции *f*. Опытные программисты легко узнают на рисунке «раскрутку» вызовов рекурсивной процедуры. Множество термов (представление алгоритмов в виде множества термов) позволяет легко построить программу *P*, реализующую применение оператора примитивной рекурсии к функциям *g* и *h* и вычисляющую функцию *f*.
- *P*: $s:=k$;
- $f[0]:=g(x_1,x_2,\ldots,x_n)$;
- **for** $i=1$ **to** s **do**
- $f[i]=h(x_1,x_2,\ldots,x_n, i-1, f[i-1])$;
- $f:=f[s]$;

- Программисты записывают, обычно, *P* более экономно, но не “функционально”.
- *P1* : $s:=k$;
- $f:=g(x_1,x_2,\ldots,x_n)$;
- **for** $i=1$ **to** s **do**
- $f=h(x_1,x_2,\ldots,x_n,i-1,f)$;
- Функция *f* называется *примитивно рекурсивной* относительно множества простейших функций, если она строится из простейших вычислимых функций конечным числом применения операторов суперпозиции и примитивной рекурсии.

- ПРИМЕР.
- Рассмотрим примитивно рекурсивную схему
- $x+0=x$
- $x+(y+1)=(x+y)+1=s(x+y)$
- Следовательно, функция $(x+y)$ строится применением оператора примитивной рекурсии **R** к примитивно рекурсивным функциям $g(x)=x$ и $h(x,y,z)=z+1$. Следовательно, функция $x+y$ примитивно рекурсивна.

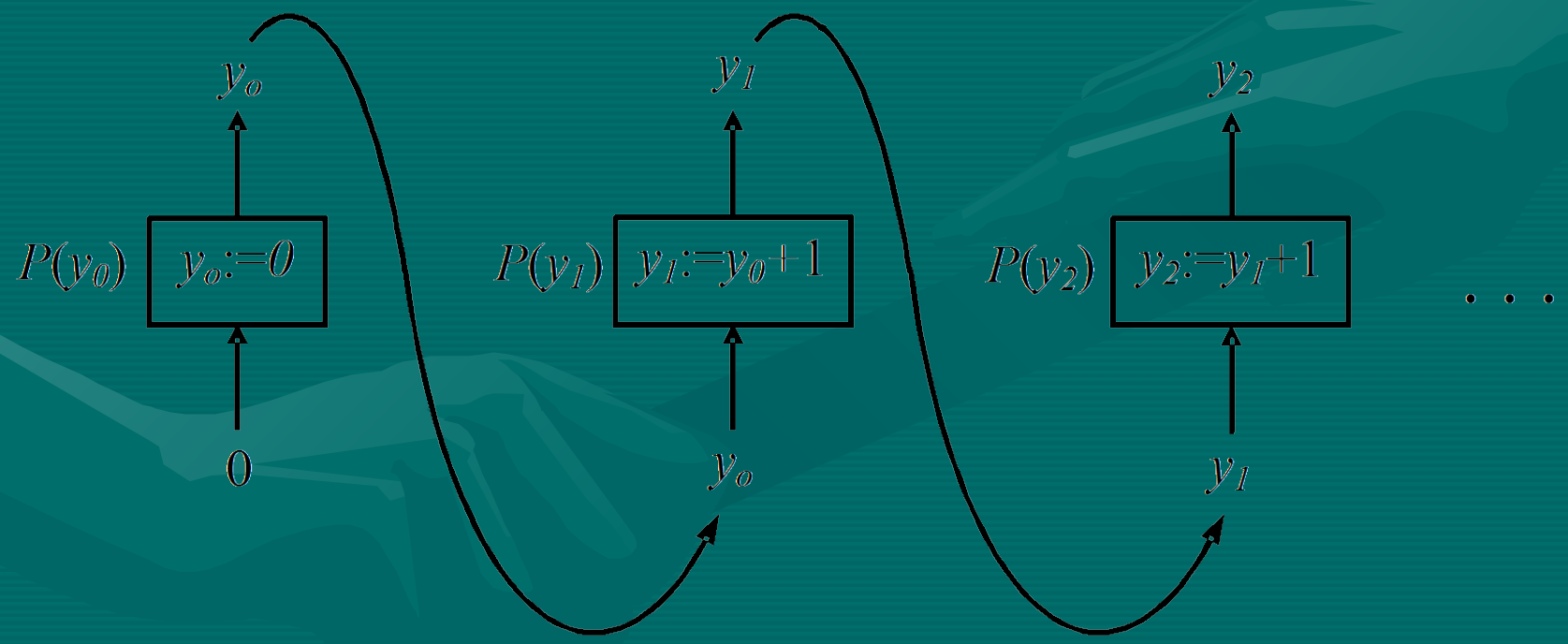
- Следует различать понятия алгоритмически вычислимой функции и алгоритма, а именно: пусть существует некоторая нерешенная математическая проблема x. Определим функцию
- $$h(x)=\begin{cases}1, \text{если проблема решается положительно}\\ 0, \text{если проблема решается отрицательно}\end{cases}$$
- Ясно, что функция $h(x)$-константа (либо 0, либо 1) и, следовательно, примитивно рекурсивна. Однако алгоритм (примитивно рекурсивная схема) ее вычисления неизвестен, поскольку функции $h(x)=0$ и $h(x)=1$ вычисляются разными алгоритмами, а сделать правильный выбор невозможно (нет решения проблемы).

•3.Оператор минимизации

- Пусть f - некоторая n-местная вычислимая функция, $n \geq 1$, алгоритм вычисления f известен. Вычисление значения функции f для некоторого значения аргумента невозможно лишь тогда, когда алгоритм не может завершиться.
- Зафиксируем некоторые значения первых n-1 переменных $x_1, x_2, \ldots, x_{n-1}$. Рассмотрим уравнение
- $$f(x_1, x_2, \ldots, x_{n-1}, y) = x_n \qquad (3)$$
- Для его решения начнем вычислять последовательно значения функции $f(x_1, x_2, \ldots, x_{n-1}, y)$ для y=0, 1, 2, ... Наименьшее число k такое, что $f(x_1, x_2, \ldots, x_{n-1}, k) = x_n$ есть решение этого уравнения. Это решение обозначается $\mu_y(f(x_1, x_2, \ldots, x_{n-1}, y) = x_n)$. Решения может и не оказаться, например если:
- - значения $f(x_1, x_2, \ldots, x_{n-1}, k)$, $k=0, 1, 2, ,,,$ определены и отличны от x_n, а значение $f(x_1, x_2, \ldots, x_{n-1}, k+1)$ не определено,
- - значения $f(x_1, x_2, \ldots, x_{n-1}, k)$, $k=0, 1, 2, \ldots$, все определены и отличны от x_n.

- В таких случаях значение $\mu_y(f(x_1, x_2, \ldots, x_{n-1}, y)=x_n)$ считается неопределенным. Величина $\mu_y(f(x_1, x_2, \ldots, x_{n-1}, y)=x_n)$ зависит от значений переменных $x_1, x_2, \ldots, x_{n-1}, x_n$ и потому она может рассматриваться как значение функции от аргументов $x_1, x_2, \ldots, x_{n-1}, x_n$. Эта функция обозначается $\boldsymbol{Mf}$, $\boldsymbol{M}$ называется оператором *минимизации*.
- Оператор минимизации очевидно частично вычислим, а значит и функция $\boldsymbol{Mf}$ частично вычислима. Функция f называется *частично рекурсивной* относительно простейших функций, если она строится конечным числом применений операторов суперпозиции, примитивной рекурсии и минимизации.

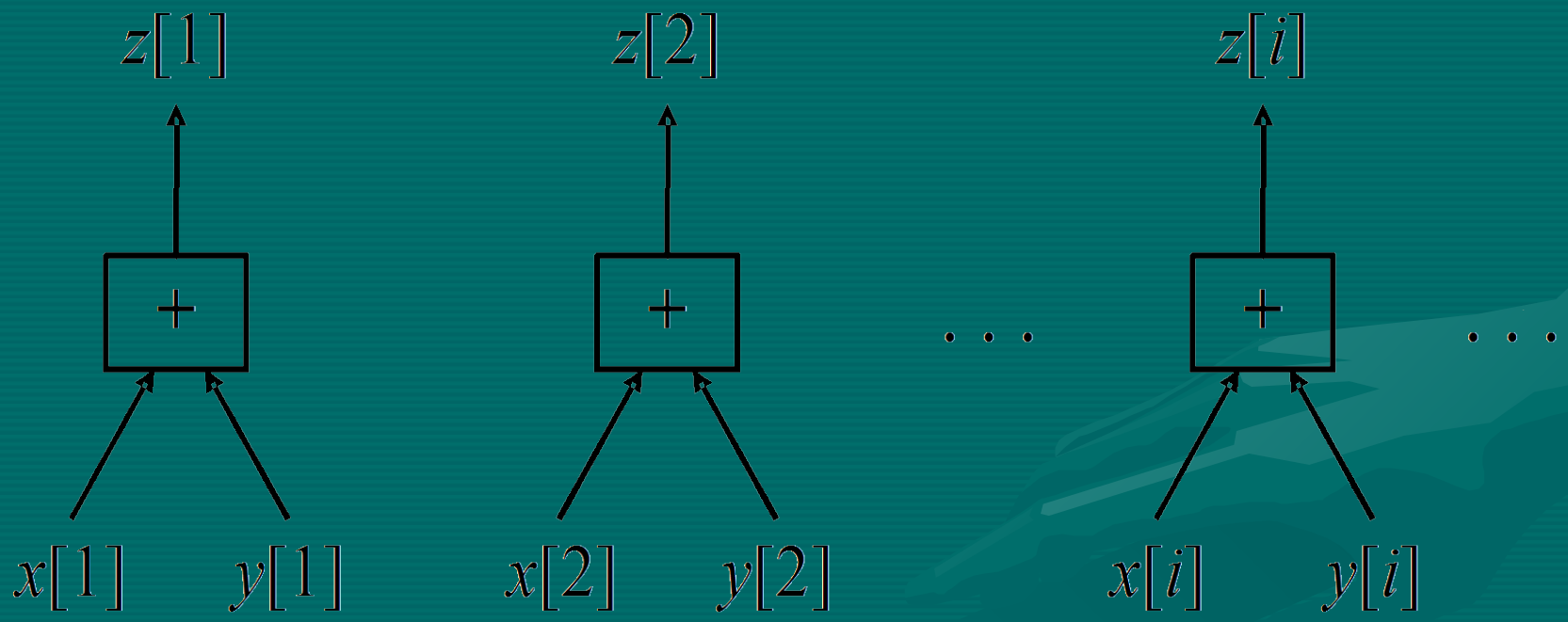
Как и для оператора примитивной рекурсии для оператора минимизации можно построить представление в виде счетного множества функциональных термов. Введем предикат $P(y_i)$:$f(x_1, x_2, \ldots, x_{n-1}, y_i)=x_n$. Тогда оператор минимизации представляется счетным множеством функциональных термов .



- Предикат $P(y_i)$ выделяет функциональный терм, который вырабатывает значение функции $\mu_y(f(x_1, x_2, \ldots, x_{n-1}, y)=x_n)$.
- Из всех этих термов лишь один может выработать значение функции $\boldsymbol{Mf}$, а именно тот, который выработает значение $z_k=x_n$ для минимального k. Таким образом и здесь, как и в случае оператора примитивной рекурсии, операторный терм $\boldsymbol{Mf}$ представляется счетным множеством функциональных термов. Одно из существенных отличий состоит в том, что конечное множество функциональных термов, представляющих оператор примитивной рекурсии, фиксируется для конкретных значений до начала вычислений. В случае оператора минимизации терм, вычисляющий значение функции, определяется динамически в ходе вычисления.

- Для конструирования программы, вычисляющей частично-рекурсивную функцию, оператора типа **for** уже недостаточно, нужен оператор типа **while**. Именно поэтому, любой процедурный язык программирования содержит операторы типа **while**. Следующая программа реализует оператор минимизации:
- P : $z:=f(x_1, x_2, \ldots ,x_{n-1},0)$;
- $i:=0$;
- **while** $z \neq x_n$ **do**
- {
- $i:=i+1$;
- $z:=f(x_1, x_2, \ldots ,x_{n-1},i)$;
- };
- $Mf:=i$;
- Понятно, что эта программа не всегда останавливается и, следовательно, вычисляемая ею функция может оказаться не всюду определенной.

- Было показано, что каждая частично рекурсивная функция представляется потенциально бесконечным множеством функциональных термов.
- И наоборот, каждой программе может быть сопоставлено множество функциональных термов, определяющее тот алгоритм, что реализован программой. Например, программа покомпонентного сложения двух векторов
- n:=.... ;
- **for** $i = 1$ **to** n **do**
- $z[i]:=x[i]+y[i]$;
- реализует алгоритм, представленный множеством термов


z[1]
z[2]
z[i]
+
+
+
. . .
. . .
x[1]
y[1]
x[2]
y[2]
x[i]
y[i]

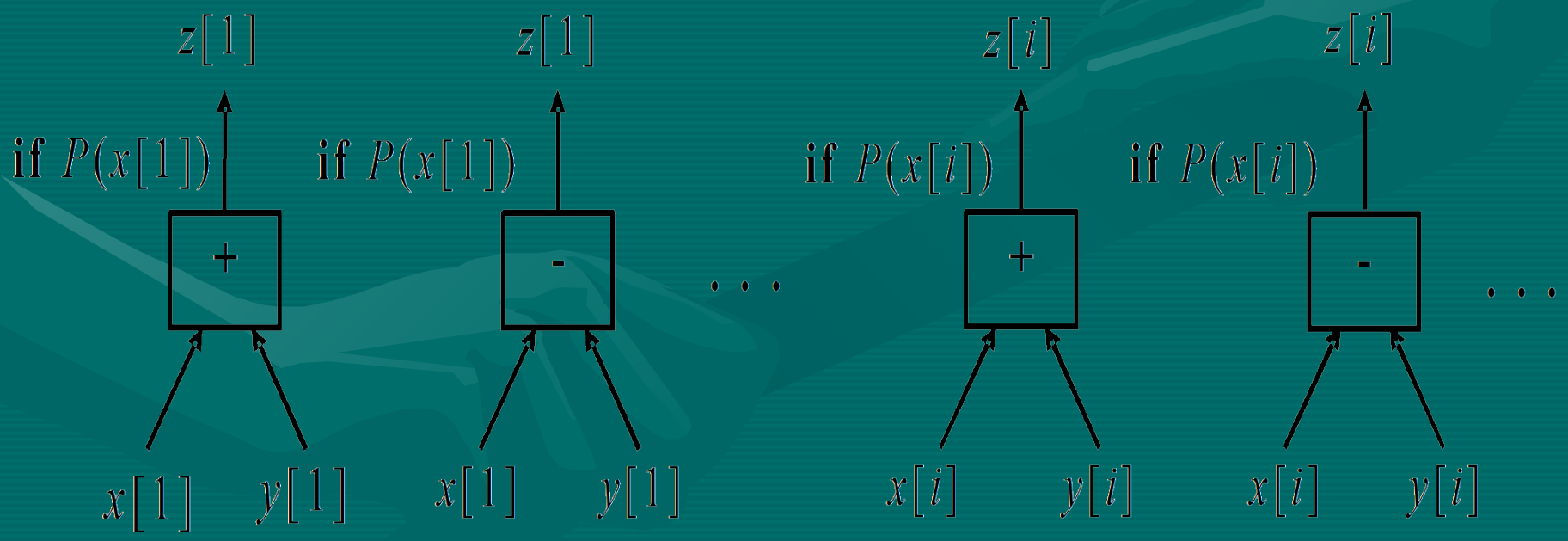
Рассмотрим другой пример.

n:=.... ;

for *i = 1* **to** *n* **do**

if *P(x[i])>0* **then** *z[i]:=x[i]+y[i]* **else** *z[i]:=x[i]+y[i]* ;

Реализуемый программой алгоритм может быть представлен множеством функциональных термов:

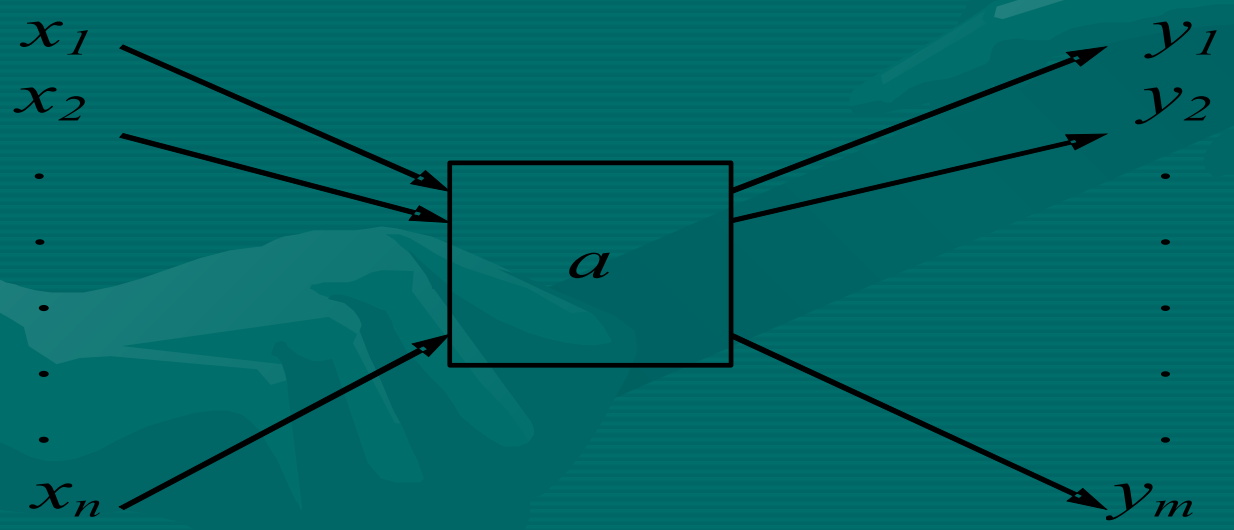


- Это множество функциональных термов строится фиксацией значения предиката *P*. Если фиксировать *P*=**true**, то программа реализует множество функциональных термов с использованием только операции “+”. А если зафиксировать *P*=**false**, то программа реализует множество функциональных термов с операцией “-“ . Объединение этих множеств и даст множество, показанное на рисунке.

Представление алгоритма

Рассматриваться будут лишь такие представления алгоритма, в которых в явном виде используются преобразователи значений входных величин (переменных) в значения выходных. Такой преобразователь изображает элементарный шаг в интуитивном понятии алгоритма. Представление алгоритма, которое может быть исполнено компьютером, называется *программой*

Каждый преобразователь a (будем впредь называть его операцией) вычисляет функцию f_α, значение которой есть результат выполнения преобразования (выполнение операции) a.
Для вычисления функции f_α используется модуль mod_a



Представление алгоритма A есть набор $S=(X,F,C,M)$, где $X=\{x,y,...,z\}$ - конечное множество переменных, $F=\{a,b,...,c\}$ - конечное множество операций. C - *управление*, т. е. множество ограничений на порядок выполнения операций. Каждое ограничение определяет порядок исполнения пары операций вида $(a<b)$, что означает, что операция a должна стартовать и завершиться раньше b. Таким образом, управление C – это бинарное отношение частичного порядка на множестве F операций алгоритма A

M - функция, задающая отображение множеств переменных и операций в физические устройства вычислительной системы (здесь обычно мультикомпьютер), т. е. *M* задает *распределение ресурсов* компьютера.

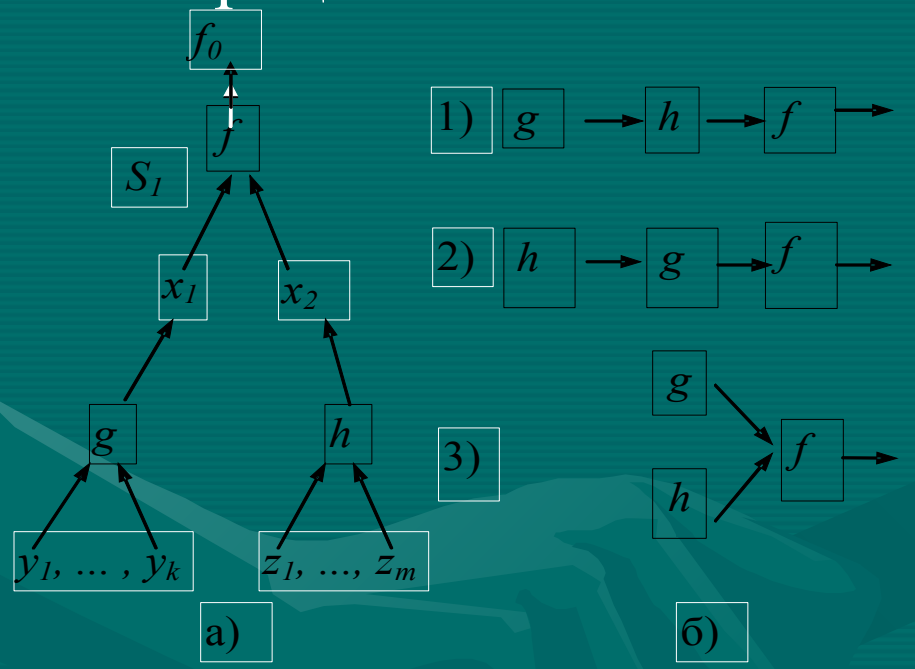
Часть ограничений управления *C*, определенных *информационной зависимостью* между операциями, называется *потоковым* управлением, остальные ограничения *C* задают *прямое* управление, связанное обычно с распределением ресурсов.

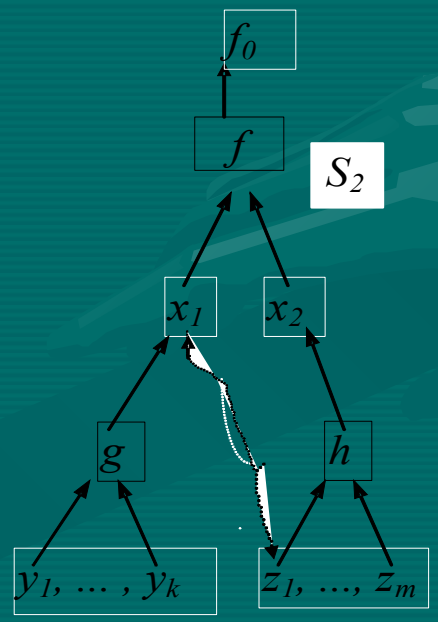
Чтобы перейти от алгоритма к программе в общем случае необходимо построить управление *C* и распределение ресурсов *M*

Реализацией алгоритма *A*, представленного в форме *S*, называется выполнение операций в некотором произвольном допустимом порядке, который не противоречит управлению *C*. Предполагается, что при любой реализации вычисляется функция F_A, т.е. *C* всегда содержит потоковое управление, которое и гарантирует вычисление F_A

Множество всех реализаций алгоритма *A*, представленного в форме *S*, обозначим *P(A,S)*. Если *P(A,S)* есть одноэлементное множество, то *S* - *последовательное* представление. *S* - *параллельное* представление алгоритма *A*, если множество *P(A,S)* содержит, более одной реализации

Пример 1. Пусть $F=f(x_1,x_2)$, $x_1=g(y_1,\ldots,y_k)$, $x_2=h(z_1,\ldots,z_m)$ - операции, $out(h)=\{x_2\}$, $out(g)=x_1$, $out(f)=\{f_0\}$. Тогда алгоритм A вычисления функции $F= f(g_1(y_1,\ldots, y_k),$ $h=(z_1,\ldots,z_m)$ представлен в параллельной форме S_1, f, g и h – операции





Управление C в представлении S_1 - это отношение частичного порядка $\{(g<f), (h<f)\}$

Допустим всякий порядок выполнения операций, при котором вычисление аргументов операции предшествует выполнению операции, управление *C* - потоковое, сохраняющее необходимые информационные зависимости между операциями.

Уменьшить потоковое управление (например, разрешить операции *f* выполняться раньше операции *g*) нельзя, так как при этом будет вычисляться, вообще говоря, не функция *F*, а какая-то другая. В этом смысле потоковое управление есть минимальное управление, гарантирующее вычислений функции *F*. Множество реализаций алгоритма *A*, представленного в форме *S1*, состоит из трёх элементов

Если для хранения значений переменных используются ячейки памяти и заданы представления $S_1=(X,F,C_1,R_1)$ и $S_2=(X,F,C_2,R_2)$ алгоритма вычисления функции $f=f(g(y_1,\ldots,y_k),h(z_1,\ldots,z_m))$. В S_2 потребуем, чтобы значения переменных алгоритма x_1 и z_1 хранились в одной и той же ячейке памяти.

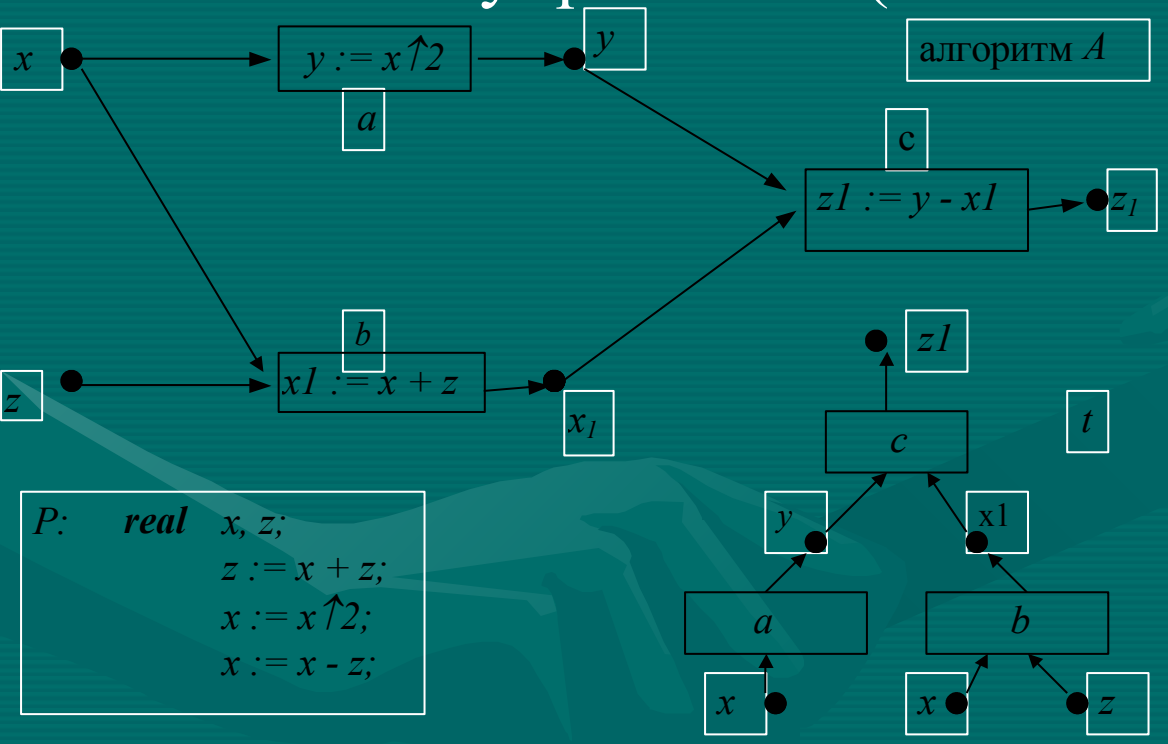
Тогда $P(A,S_1)\supset P(A,S_2)$, так как в силу сделанного распределения памяти для вычисления f прямое управление в C_2 должно теперь содержать требование, чтобы операция h выполнялась раньше g, то есть управление C_2 в представлении S_2 - это множество $\{(g<f), (h<f), (h<g)\}$. При этом операция h, начав исполняться, использует значение переменной z_1 до того, как в ту же ячейку памяти будет записано значение переменной x_1, вычисленное операцией g.

Пример 2. Алгоритм *A* задан в форме программы *P*, записанной на некотором последовательном языке программирования. Операторы языка, вычисляющие значения переменных, есть операции. Прямое управление задается последовательностью операторов в программе либо специальными операторами управления типа **go to.**

Порядок выполнения операций полностью фиксирован. Распределение памяти задаётся идентификаторами переменных программы, которые фактически есть имена ячеек памяти, имена переменных алгоритма в программе не сохраняются. Предполагается, что для выполнения алгоритма может быть использован только один процессор.

Пример 3 и разработка программы

- Представление $S=(X,F,C,M)$ алгоритма A в форме **блок-схемы** и одного из его возможных представлений в форме программы P. Здесь $X=\{x,y,z,x1,z1\}$, $F=\{a,b,c\}$. C - потоковое управление (показано стрелками)



•Сначала строится дерево (функциональный терм) t (алгоритм А) Затем распределяются ресурсы и строится управление в процедурном языке программирования

Пример 4. Задание алгоритма формулами

- Исходный алгоритм умножения квадратных матриц сформулирован в терминах элементов матриц

- $$C = A \times B, \quad A = (a_{ij})_{i,j=\overline{1,N}}, \quad B = (b_{ij})_{i,j=\overline{1,N}}, \quad C = (c_{ij})_{i,j=\overline{1,N}}$$

Исходный алгоритм

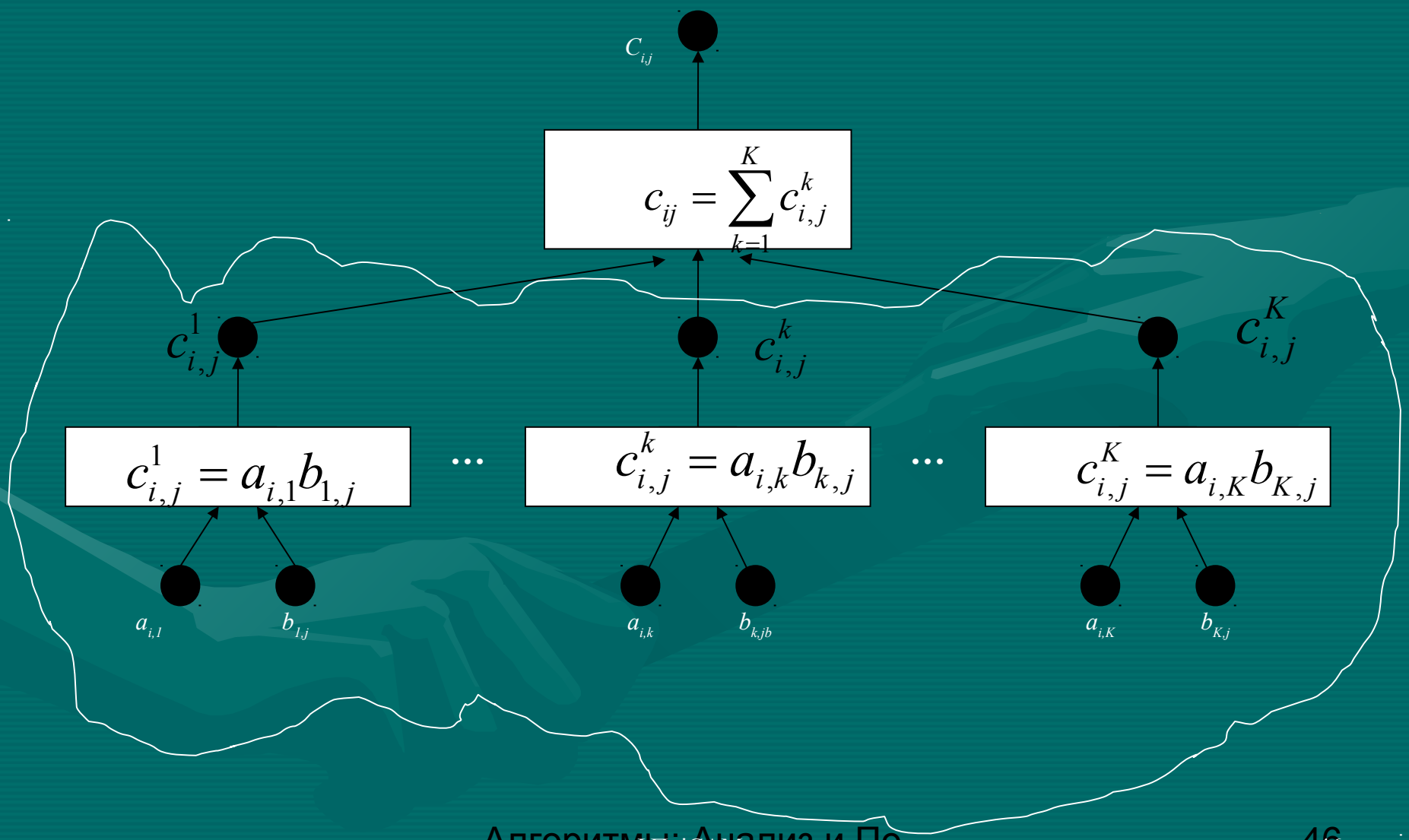
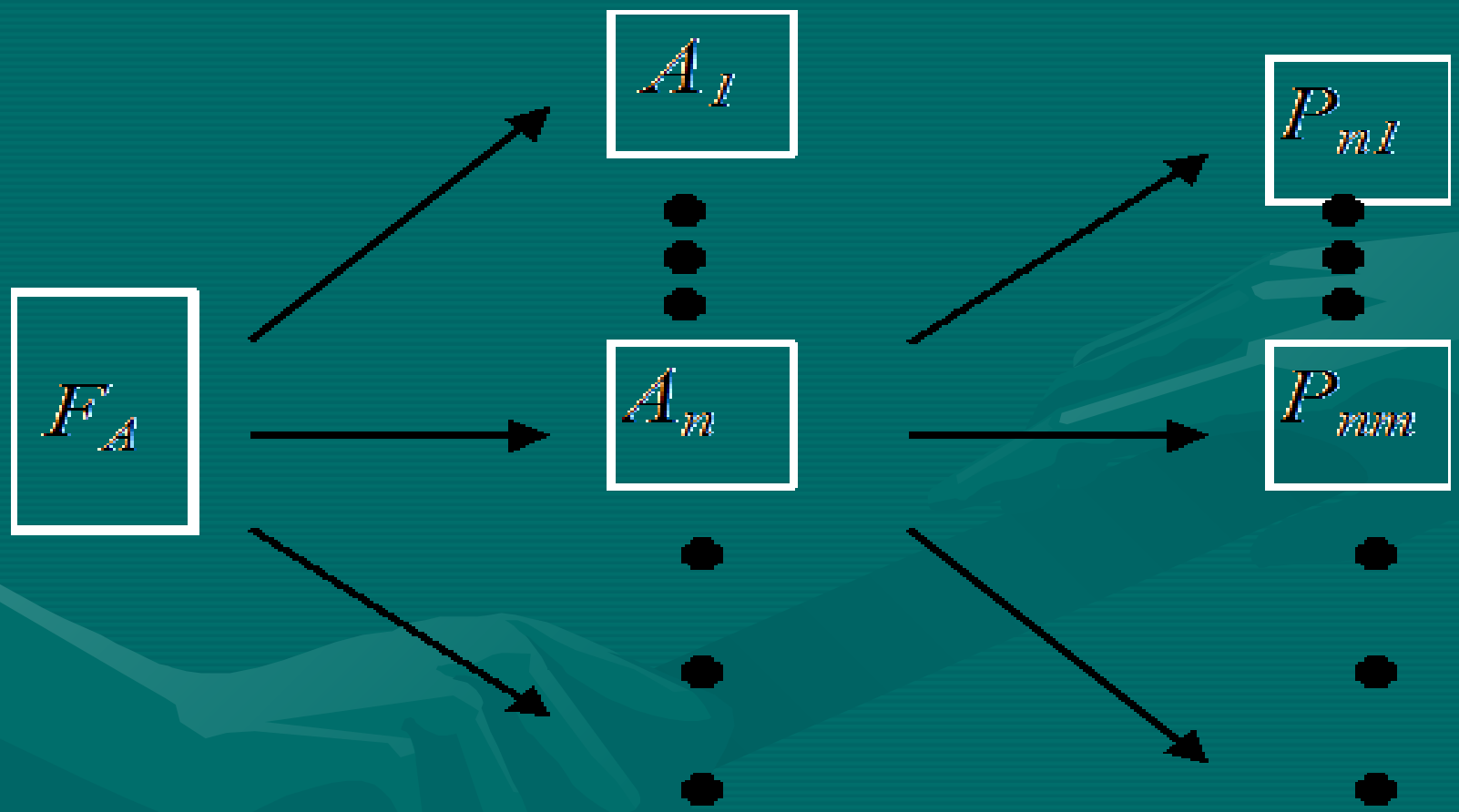
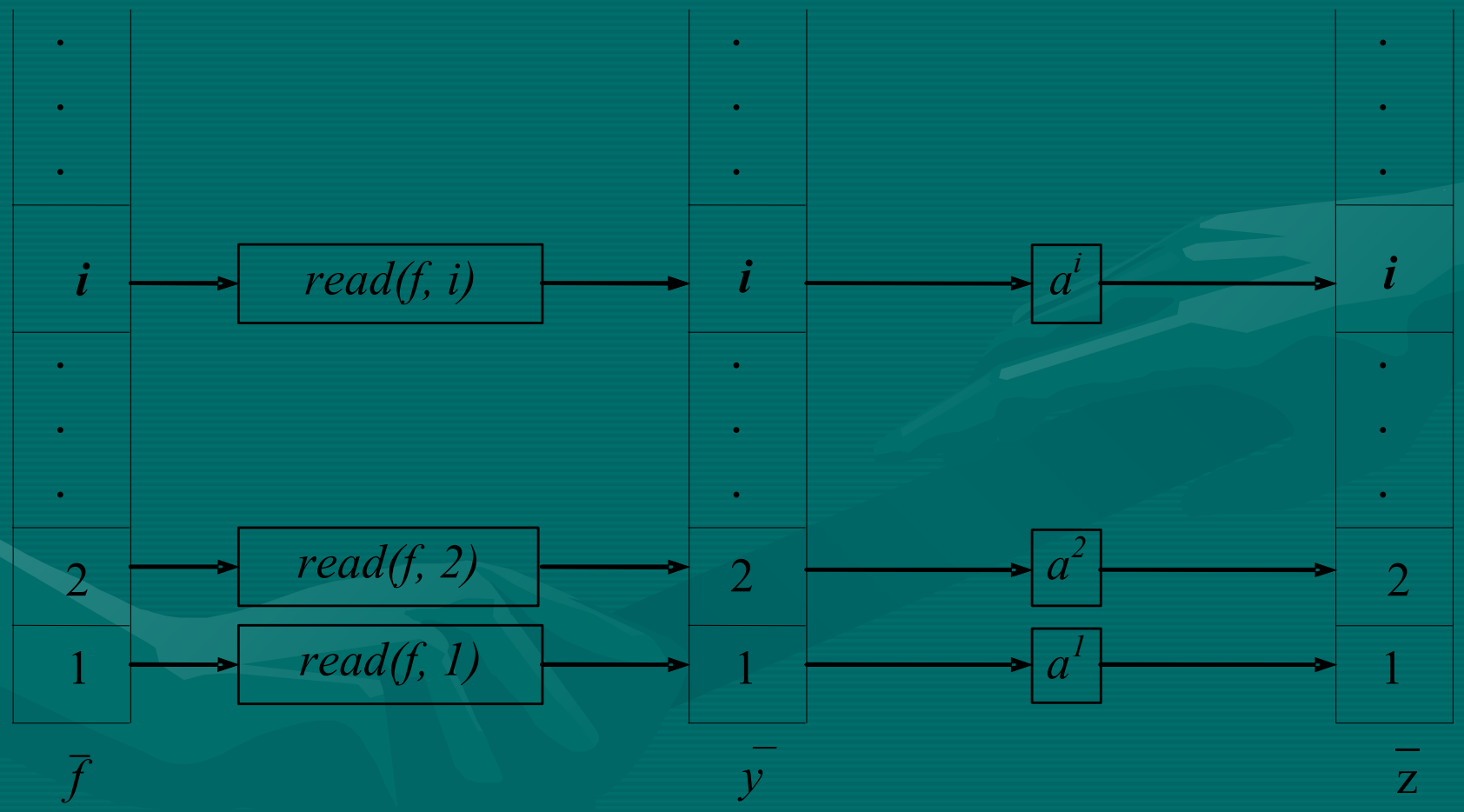


Диаграмма: алгоритм и программа

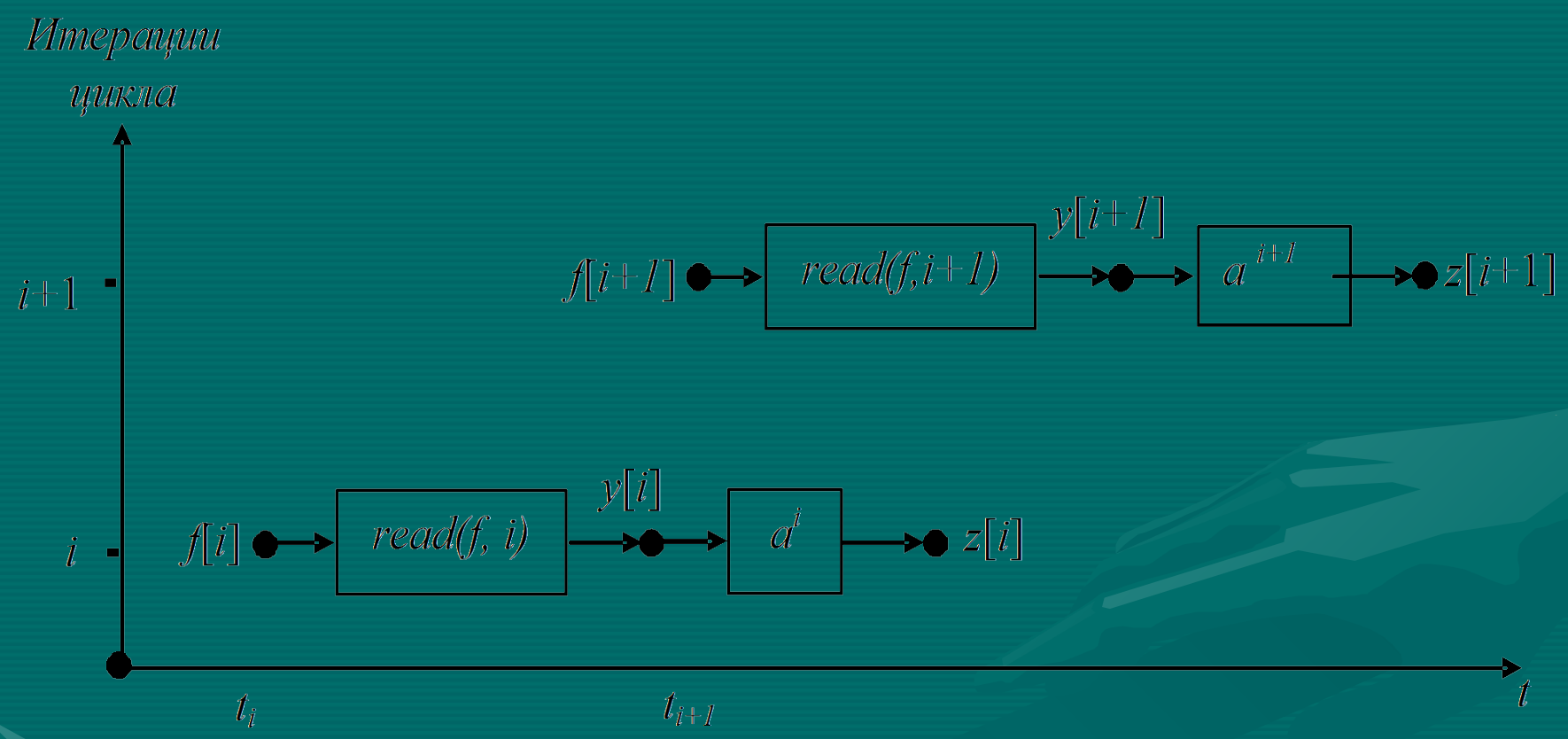


Ввод записей файла и их обработка



- Если отображение M сопоставляет массиву y участок памяти, в котором может быть размещена лишь одна запись, и доступен только один процессор, тогда прямым управлением C должен быть задан следующий жестко определенный порядок выполнения операций: $read(f,1)$, a^1, $read(f,2)$, a^2, ... (обычно в программе задается с использованием оператора цикла).
- При таком порядке исполнения считанная запись файла будет потреблена операцией a, память освобождена, и следующая запись может быть считана в тот же участок памяти, что и предшествующая запись. Алгоритм A_{file} будет правильно реализован, т.е. будет вычислена функция F_{Afile}.

- Если при тех же условиях на размещение массива y для исполнения алгоритма доступны два процессора, тогда, кроме последовательного, может быть ещё организовано конвейерное исполнение алгоритма (другая его реализация), при котором обработка i-ой записи файла делается одновременно со считыванием $(i+1)$-ой записи файла



Если разрешается разместить в памяти две записи, тогда может быть использован режим ввода с двумя буферами и две операции - a^i и a^{i+1} - смогут выполняться одновременно, когда есть доступные процессоры и т.д.