Компьютерная алгебра

(курс лекций)

Игорь Алексеевич Малышев

Computer.Algebra@yandex.ru

КСПТ

КОМПЬЮТЕРНЫЕ СИСТЕМЫ И ПРОГРАММНЫЕ ТЕХНОЛОГИИ

Лекция 3

Основы символьных вычислений

Содержание лекции

- Структуры данных в компьютерной алгебре
- Техника символьных вычислений

План лекции: тема подраздела

- **Структуры данных в компьютерной алгебре**
- Техника символьных вычислений

Структуры данных: основные определения

Структурой данных называется совокупность множеств $\{M_1, M_2, \ldots M_N\}$ и совокупность отношений $\{P_1, P_2, \ldots P_R\}$, определённых над элементами этих множеств:
$S = \{M_1, M_2, \ldots M_N ; P_1, P_2, \ldots P_R\}$

Пример. Структура массива определяется следующим образом:

$M = \{a_1, a_2, \ldots a_N\}$,

$P(a_i, a_j)$ = true, если $j=i+1$,

= false - в противном случае.

(P () – функция следования)

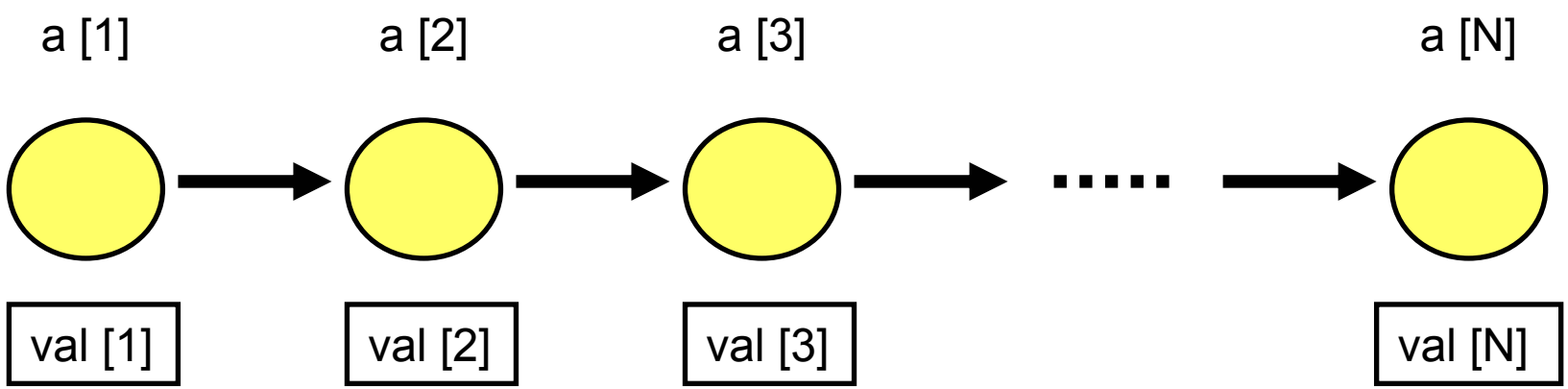
Бинарное отношение, задающее массив – орграф.

Структура данных линейна, если орграф не содержит циклов и может быть изображен в виде одной линии.

Структуры данных: основные определения

Диаграммы отношений в структурах данных

- Отношение следования (элементы множеств – вершины, отношения следования - стрелки)
- Отношение «иметь имя» (обеспечивает доступ к элементам множеств в терминах алгоритма – a [i])
- Отношение «иметь значение» (обеспечивает функциональные преобразования данных – val [i])



Структуры данных: основные определения

Структура машинной памяти

Память вычислительной (алгоритмической) машины имеет линейную структуру.

Обработка любого типа информации (имеющего структуру произвольной сложности) должна моделироваться на схеме массива – линейной структуре.

Линейная структура памяти – вектор памяти.

Отношение «иметь имя» переопределяется с помощью отношения «иметь адрес». Адрес произвольного элемента массива вычисляется по формуле: $a_i = a_0 + i * b$ (a_0 – база, адрес 1-го элемента массива; i – номер адресуемого элемента; b – число ячеек, занимаемых одним элементом массива).

Структуры данных: основные определения

Экземпляром структуры данных называется совокупность

IS = { M_{ai}, V, P, val },

где M_{ai} – множество элементов a_i;

V – множество значений;

P – множество отношений следования;

val – отношение «иметь значение».

Схемой структуры данных называется совокупность

SS = { M_{ai}, P },

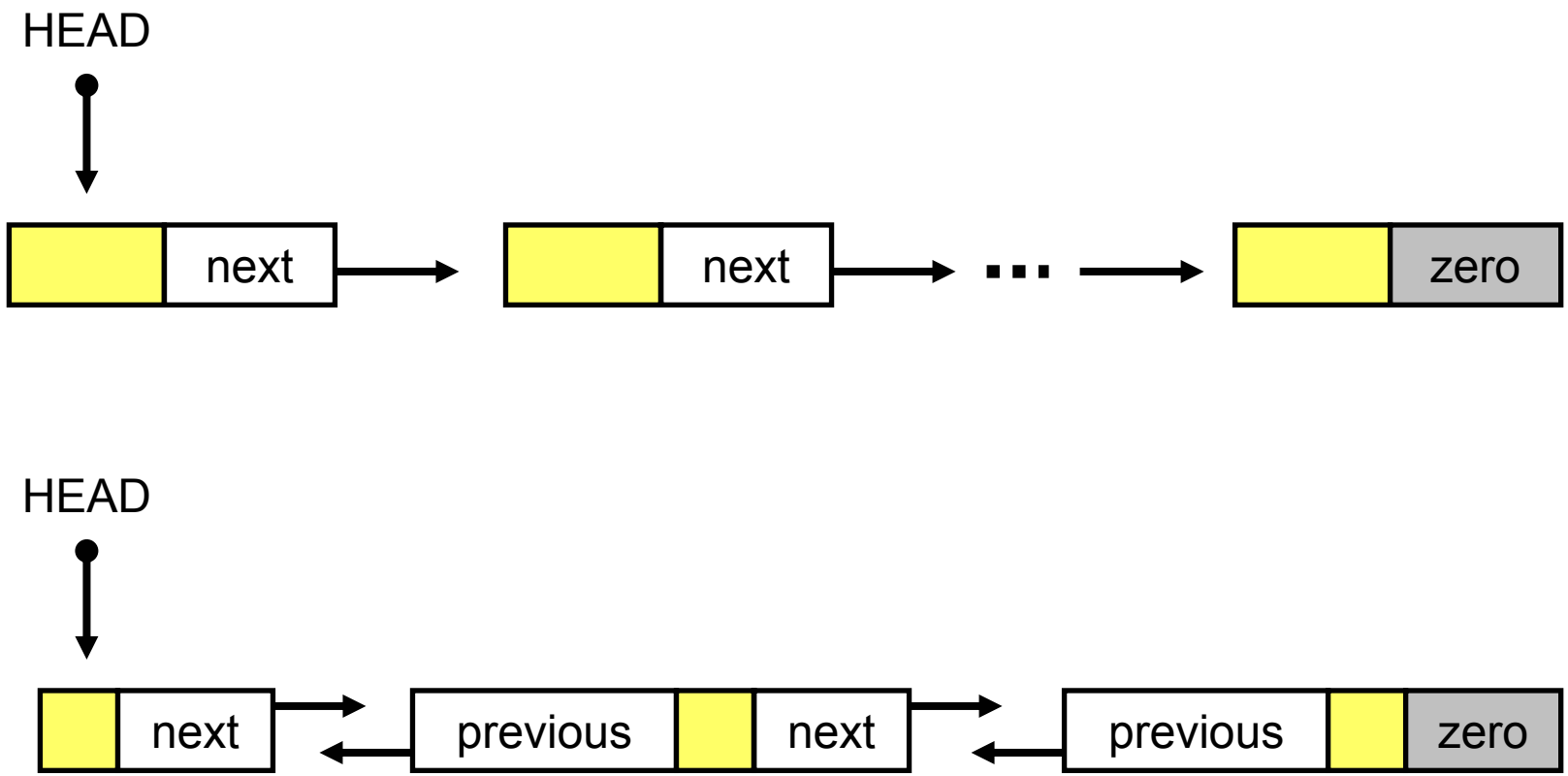
где M_{ai} – множество элементов a_i;

P – множество отношений следования.

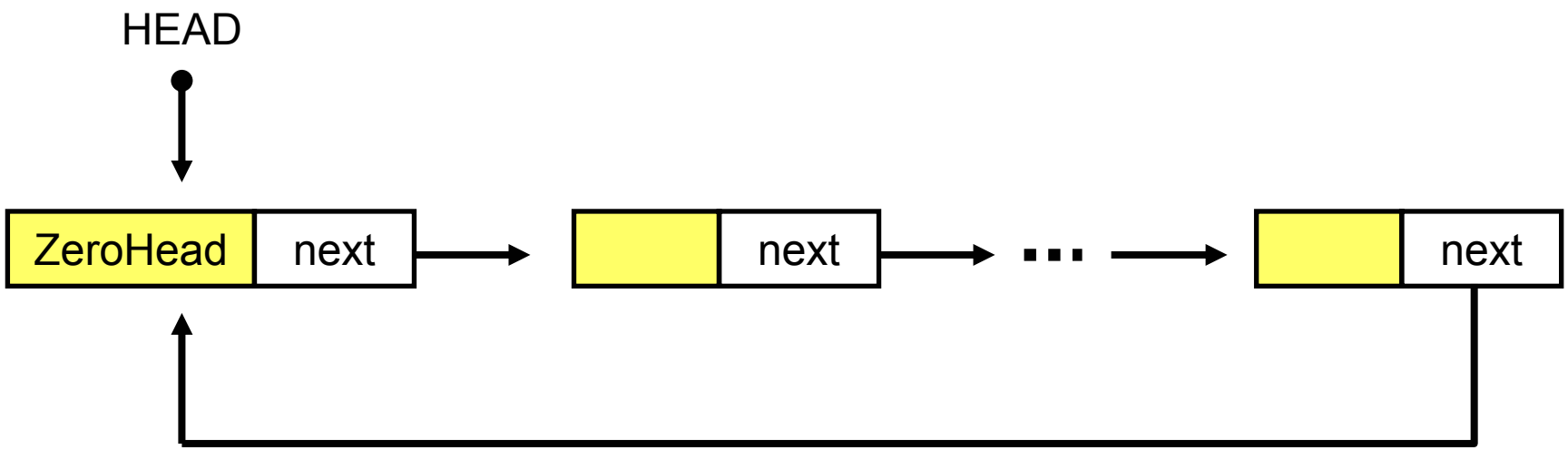
Одной SS может соответствовать множество ES. Алгоритм реализуется над схемой, а конкретные вычисления (преобразования) по алгоритму производятся над экземплярами.

Структуры данных: списки

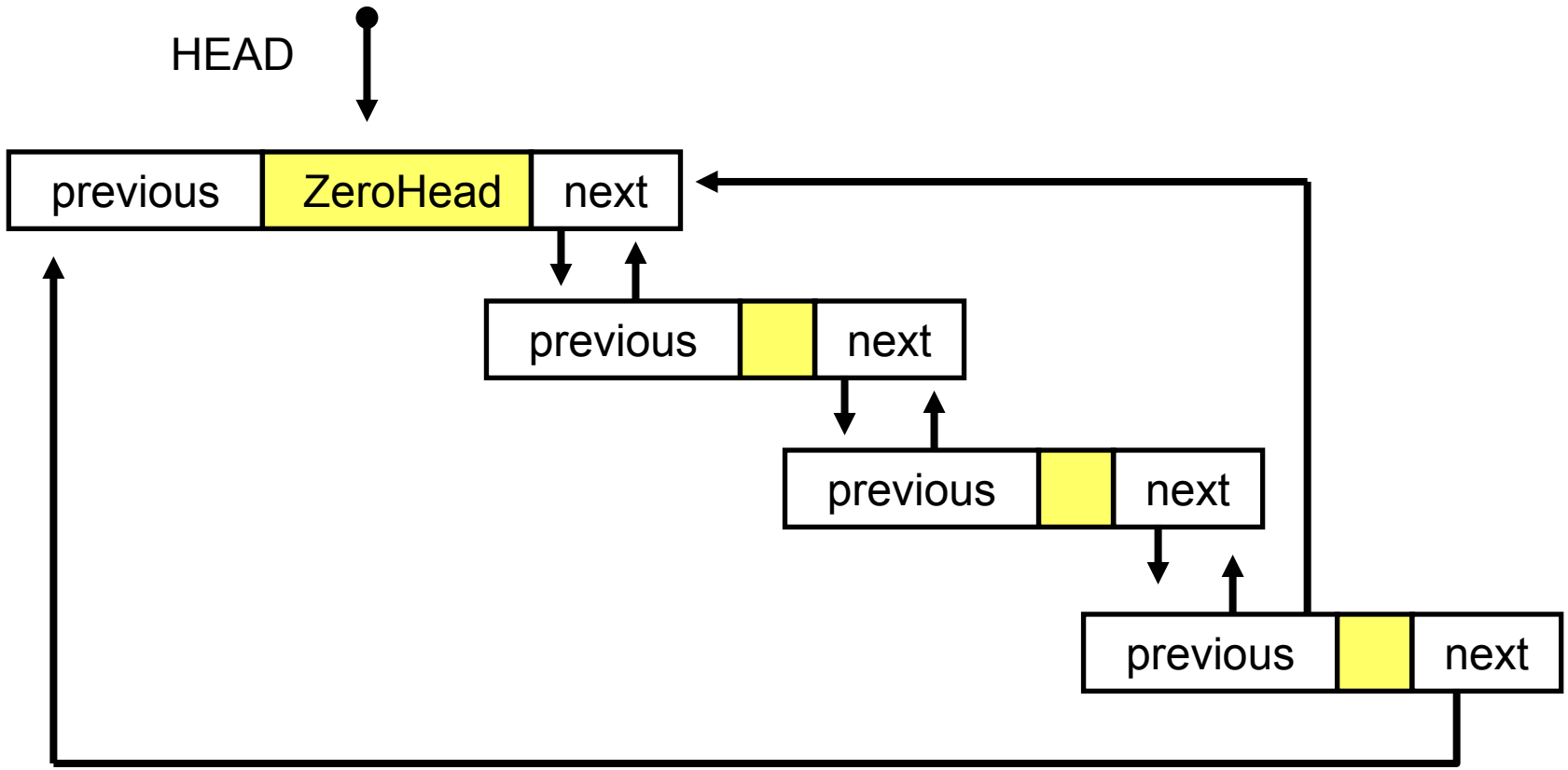
Линейные списки: одно- и двух- связные



Циклический список: односвязный



Циклический список: двухсвязный



Структуры данных: основные операции

Операции над структурами данных :

- Создание и уничтожение структуры данных;
- Поиск элемента данных в структуре;
- Обновление структуры данных: вставка нового и удаление старого элемента;
- Обход структуры данных с выполнением определённых, наперёд заданных действий.

Структуры данных: списки

Рекурсивные списки

Рекурсивным называется список, элементами которого могут являться списками.

Рекурсивные списки способны представлять данные любого уровня структурной сложности.

Пример. Представление полинома.
Полином – список мономов.
Моном – список термов.
Терм – список атомов.

Элементами рекурсивного списка могут быть не списки.

Структуры данных: основные операции

Операции над списками

Пусть $L = (A_1, A_2, A_3, \ldots A_i, \ldots A_n)$ – список.
Тогда определены следующие операции:

Базовые:

- Создание нулевого списка: $L = ( )$
- Получение 1-го элемента (головы) списка: A_1 для списка L
- Получение остатка списка (переход по ссылке к следующему элементу): $(A_2, A_3, \ldots A_i, \ldots A_n)$ для списка L
- Конкатенация (слияние) двух списков L_1 и L_2: $L = (L_1 , L_2)$

Дополнительные:

- Получение следующего элемента A_{i+1} , если известен предыдущий A_i
- Вставка нового элемента B после элемента A_i, т.е. получение из списка $L = (A_1, A_2, A_3, \ldots A_i, A_{i+1}, \ldots A_n)$ нового списка $M = (A_1, A_2, A_3, \ldots A_i, B, A_{i+1}, \ldots A_n)$
- Удаление элемента, следующего за элементом A_i, т.е. получение из списка $L = (A_1, A_2, A_3, \ldots A_i, A_{i+1}, \ldots A_n)$ нового списка $M = (A_1, A_2, A_3, \ldots A_i, A_{i+2}, \ldots A_n)$

Структуры данных: базовые типы

Базовые типы данных

- Числа (целые, рациональные, алгебраические, комплексные).

- Математические выражения (арифметика, функции, производные, интегралы, матрицы, уравнения).

Структуры данных: базовые типы

Типы целых чисел

- Короткие целые числа
  (целые числа одинарной точности).

- Длинные целые числа
  (целые числа кратной точности).

Структуры данных: типы представлений

Представление чисел произвольной точности

- **МАССИВЫ**

  (разрядность представления чисел – постоянная),
  (тип представления – не масштабируемое)
  (способ доступа к элементу – прямой (по индексу))

- **ПОСЛЕДОВАТЕЛЬНОСТИ**

  (разрядность представления чисел – переменная)
  (тип представления – масштабируемое)
  (способ доступа к элементу – последовательный (по указателям))

- **СПИСКИ**

  (разрядность представления чисел – переменная)
  (тип представления – масштабируемое)
  (способ доступа к элементу – последовательный (по указателям))
  (способ изменения разрядности – встроенный)

Структуры данных: типы представлений

Длинные целые числа

(примитивное представление – массив):

$$D = \sum_{0 \le i \le n} d_i B^i$$

$$D = (d_0, d_1, \ldots d_n)$$

$$n \ge 1, \quad |d_i| \le (B - 1)$$

$$d_i \ge 0, \quad \text{если} \quad D < 0$$

$$d_i \le 0, \quad \text{если} \quad D > 0$$

$$B - 1 = 2^{\mu} - 1$$

Структуры данных: типы представлений

Длинные целые числа (масштабируемое представление – последовательность):

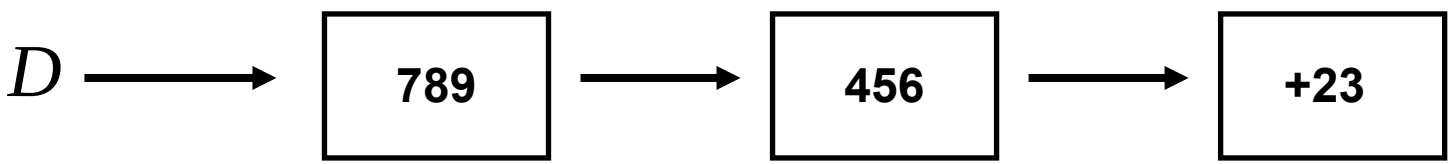
Пример.

D = + 23456789

B = 10^3 (в машинном слове – 3 десятичных цифры)

Наиболее (наименее) значимая цифра:

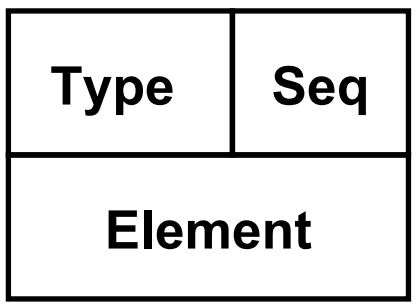
- хранит знак числа;
- находится в конце последовательности (для всех операций, кроме операции деления). Деление выполняется, начиная со старших разрядов.



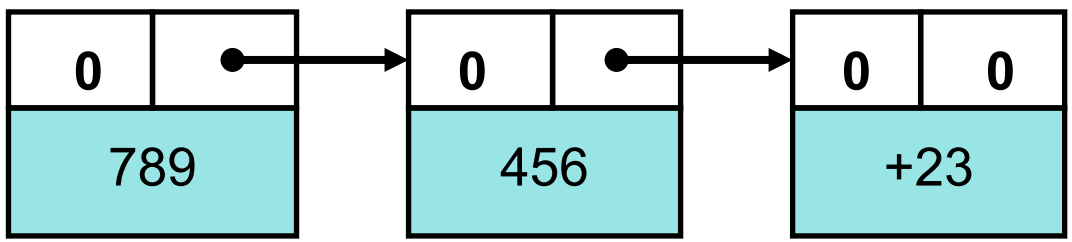
Структуры данных: типы представлений

Длинные целые числа (эффективное представление - список):

Одна ячейка



Список ячеек



Поле типа: Type = {0, если ячейка – атом; 1, если ячейка - список}

Поле ссылки: Seq = <адрес_след_ячейки_списка>

Поле элемента: Element = {d_i , если ячейка - атом; <адрес_списка>, если ячейка - список}

Структуры данных: типы представлений

Представление рациональных чисел и полиномов ?

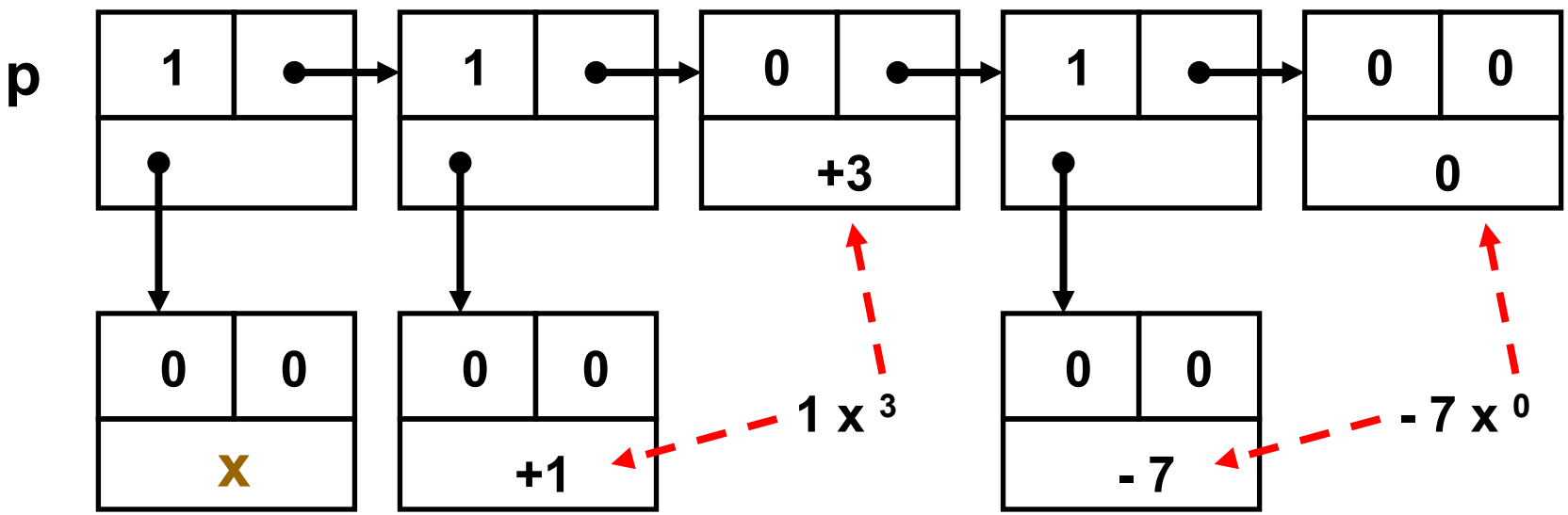
- Рациональное число r = (n, d),
  где n, d – длинные целые числа
  представляется в виде списка 2-х списков.

- Полином от одной переменной с целыми коэффициентами представляется 2-х связным списком.
  (Коэффициенты – длинные целые числа).
  (Показатели степеней – короткие целые числа).

Структуры данных: типы представлений

Представление полиномов

(эффективное представление p (x) = x^3 - 7)



Структуры данных: объекты символьных вычислений

Выводы :

1) Вся память машины символьных вычислений состоит из ячеек.

2) Каждая ячейка входит в состав определённого списка.

3) Начальная конфигурация – один список свободного места, объединяющий все ячейки.

4) Каждая следующая конфигурация – это результат операции над данными (при этом требуется перераспределение ячеек – изменение указателей):

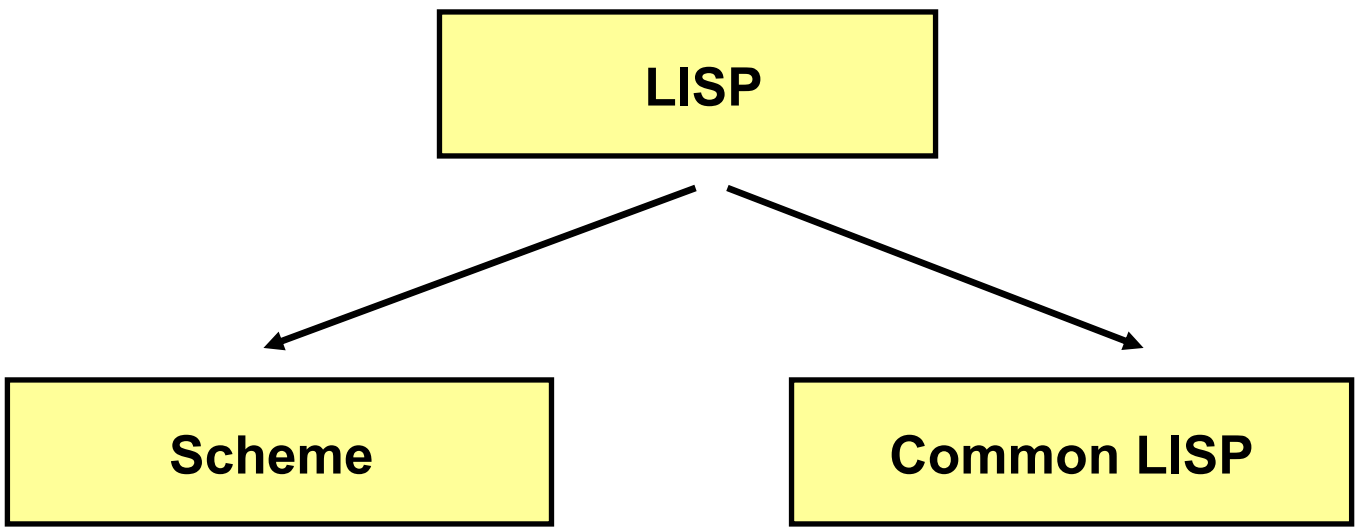
- создание нового списка для вновь поступивших данных;
- увеличение длины списка (за счёт первой ячейки списка свободного места);
- уменьшение длины списка (освобождение некоторых, ранее занятых ячеек; освобождаемые ячейки присоединяются к голове списка свободного места);
- уничтожение списка (освобождение всех ячеек списка, которые таким же образом пополняют список свободного места).

План лекции: тема подраздела

- Структуры данных в компьютерной алгебре
- **Техника символьных вычислений**

Язык LISP

Массачусетский технологический институт,
Лаборатория искусственного интеллекта,
Джон Маккарти (John McCarthy), 1959 г.



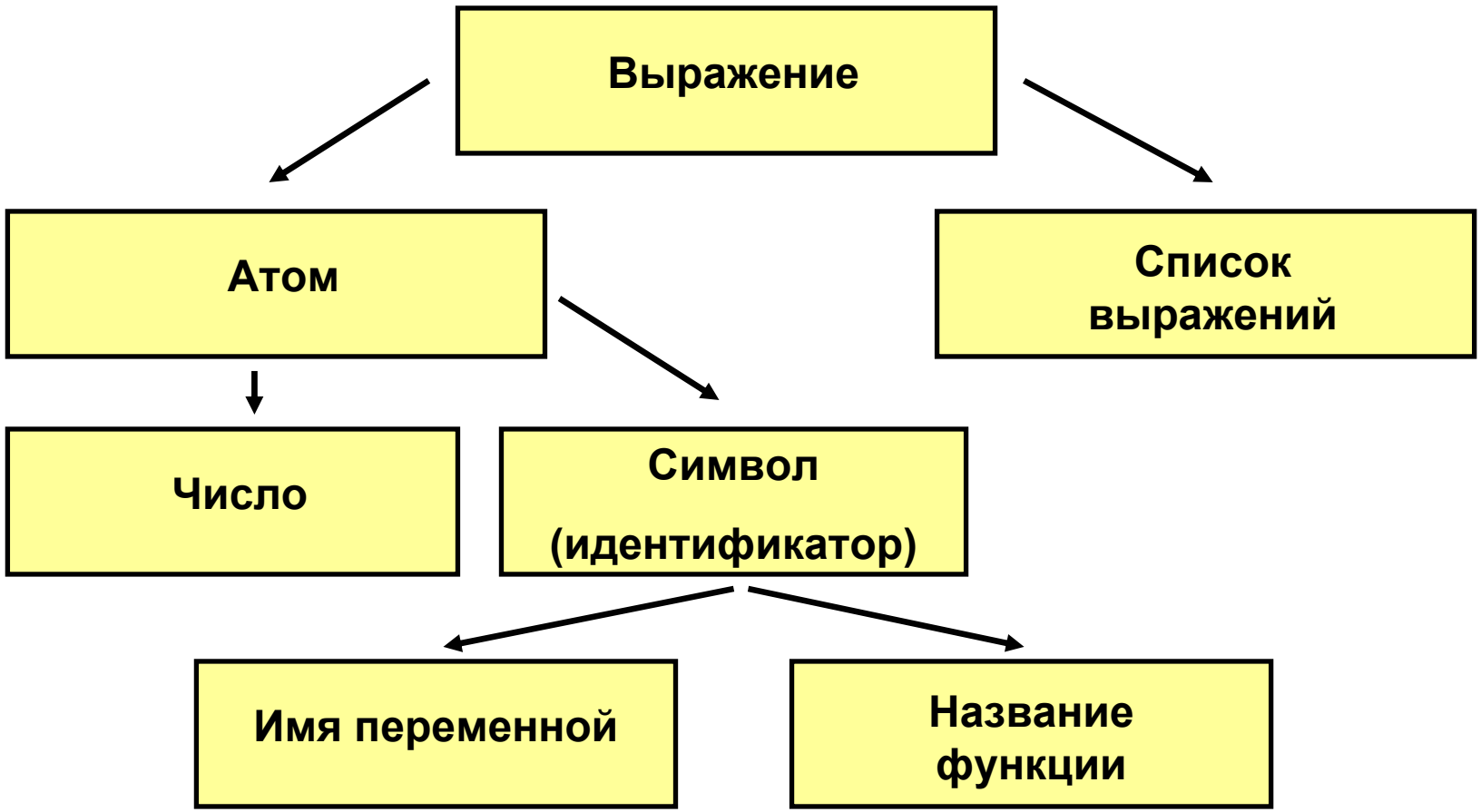
LISP-машина

- Принцип работы – интерпретатор выражений.
- REPL-цикл:

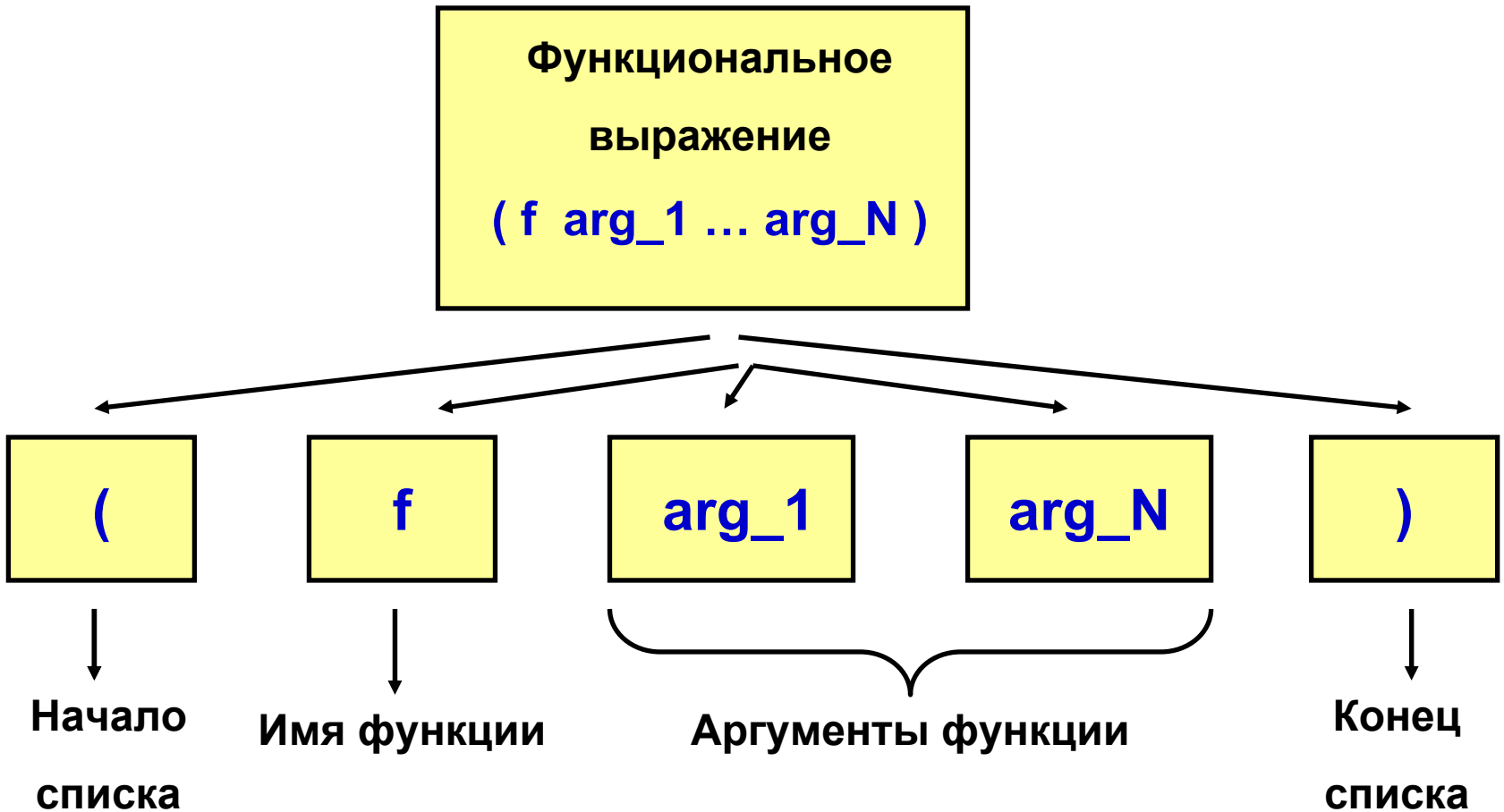
- R – read (чтение)
- E – eval (оценивание)
- P – print (печать)
- L – loop (повтор)

Техника символьных вычислений

LISP-выражения



Функциональные LISP-выражения



LISP-функции для выражений

(quote <выражение>) или **‘ <список>**
Возврат аргумента без оценивания (отложенное вычисление).

(eval <выражение>)
Оценивание аргумента и возвращение результата оценивания.

(setq <имя-переменной> <выражение>)
Оценивание выражения и присвоение результата переменной.

LISP-константы истинности:

t – истина
nil – ложь (пустой список)

LISP-функции для списков

(car <список>)
Возвращает первый элемент списка.

(cdr <список>)
Возвращает копию исходного списка без первого элемента.

(cons <выражение> <список>)
Возвращает копию исходного списка, в начало которого добавлено выражение.

(list <список>)
Возвращает копию исходного списка, в которой все элементы списка прошли оценивание.

(<выражение1> . <выражени2>)
Конструктор списка - пары из двух выражений.

(assoc <ключ> <список пар>)
Возвращает результат выбора из списка пар вида (ключ . значение) первой пары,
в которой ключ совпадает с заданным.
Если такой пары нет, то возвращается пустой список – () или nil.

LISP-предикаты

(atom <выражение>)

Возвращает t, если выражение является атомом, и nil в противном случае.

(listp <выражение>)

Возвращает t, если выражение является списком, и nil в противном случае.

(equal <выражение1> <выражение2>)

Возвращает t, если выражения равны между собой, и nil в противном случае.

Техника символьных вычислений

(if <предикат> <выражение1> <выражение2>)
Если предикат истинен, то функция возвращает выражение1,
в противном случае – выражение2.

(while <предикат> <выражение>)
Функция оценивает выражение, пока предикат истинен
(вызов предиката производится перед каждой итерацией).
Возвращает результат последнего оценивания.

(cond (<предикат1> <выражение1>) … (<предикатN> <выражениеN>))
Функция просматривает свои аргументы слева направо. Если i-й предикат истинен,
то производится оценивание i-го выражения. Возвращает результат последнего оценивания.

(progn <выражение1> … <выражениеN>)
Функция просматривает свои аргументы слева направо и оценивает их.
Возвращает результат последнего оценивания.

(do ((<переменная1> <начальное-значение1> <шаг1>)
...
(<переменнаяN> <начальное-значениеN> <шагN>)
(<предикат> <результат>)
<тело цикла>)
Функция программирования циклов. Сначала все переменные получают указанные начальные значения. Затем на каждом шаге цикла проверяется предикат и, если он истинен, то оценивается и возвращается результат. В противном случае выполняется тело цикла, каждой переменной присваивается значение соответствующего шага и итерация повторяется.

Техника символьных вычислений

LISP-конструкторы функций

(defun <имя-функции> <список-аргументов> <выражение>)
Определяет новую функцию с заданным именем.
При вызове функции сначала оцениваются ее фактические аргументы и полученные значения присваиваются локальным переменным, перечисленным в списке аргументов.

(defmacro <имя-макроса> <список-аргументов> <выражение>)
Определяет новый макрос с заданным именем.
При вызове макроса фактические аргументы присваиваются локальным переменным без оценивания.

(lambda <аргументы> <выражение>)
Определяет безымянную функцию.

(macro <аргументы> <выражение>)
Определяет безымянный макрос.

LISP-функции вывода результатов

(terpri)
Выводит на стандартное устройство вывода перевод строки.

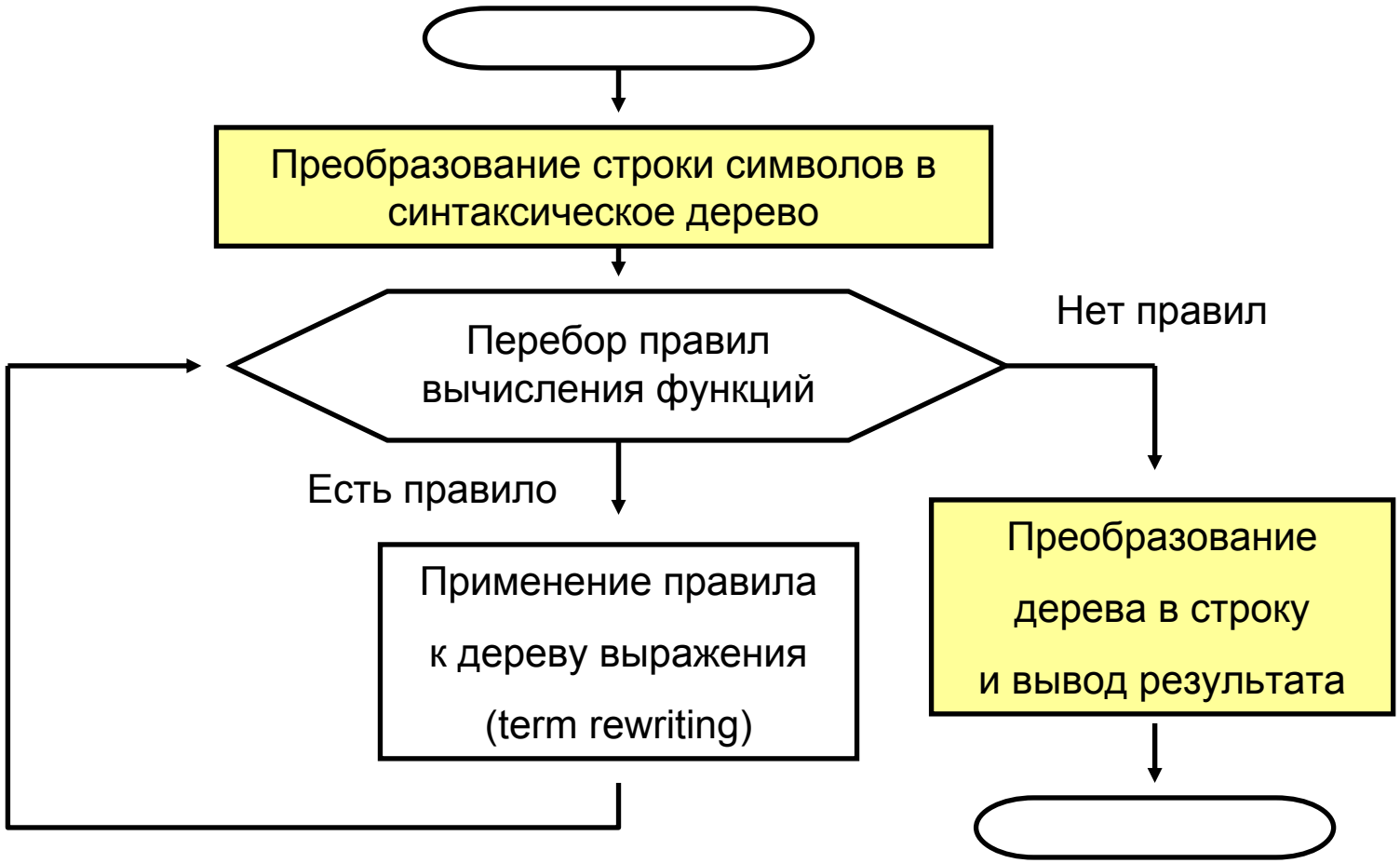
(prin1 <выражение>)
Оценивает выражение, преобразует его в строку и выводит на стандартное устройство вывода.

(print <выражение>)
То же самое, но перед выводом выражения выводится перевод строки, а после него пробел.

(write-string <текст>)
Выводит на стандартное устройство вывода заданную текстовую строку.

Техника символьных вычислений

Общая схема вычисления выражения



Техника символьных вычислений

Пример. Программа сложения двух полиномов a и b.

```
(defun add-poly (a  b)
    (cond ( (nul  a)  b)
          ( ( nul  b)  a)
          ( ( >  (caar  a)  (caar  b) )
              (cons  (car  a) (add-poly  (cdr  a)  b)  )  )
          ( ( >  (caar  b)  (caar  a) )
              (cons  (car b)  (add-poly  a  (cdr  b)  )  )  )
          ( (zerop  ( +  (cdar  a)  (cdar  b)  )  )
              (add-poly  (cdr  a)  (cdr  b)  )  )
          (t  (cons  (cons  (caar  a)  ( + (cdar  a)  (cdar  b)  )  )
                     (add-poly  (cdr  a)  (cdr  b)  )  )  )  )  )
```

Техника символьных вычислений

Пример.

Разработать систему правил упрощения полиномов:

(+ (+ (* 3 (* (* 2 (^ x 1)) 1)) (* (^ x 2) 0)) 0)

(Это результат дифференцирования полинома $3x^2 + 4$)

Набор правил упрощения полиномиальных выражений:

(p – любой терм)

№	1	2	3	4	5	6	7	8	9	10	11
Вх	*p0	*0p	*p1	*1p	+p0	+0p	-p0	^p0	^p1	^0p	^1p
Вых	0	0	p	p	p	p	p	1	p	0	1

Техника символьных вычислений

Пример (продолжение-1) - LISP-реализация списка правил:

```
(defun simplify-step (poly)
   (cond
     ((null poly) nil)
     ((atom poly) poly)
     ((equal '+ (operation poly))
        (cond
           ((equal 0 (term1 poly)) (simplify-step (term2 poly)))
           ((equal 0 (term2 poly)) (simplify-step (term1 poly)))
           (t (list '+ (simplify-step (term1 poly)) (simplify-step (term2 poly)))) ))
     ((equal '- (operation poly))
        (cond
           ((equal 0 (term2 poly)) (simplify-step (term1 poly)))
           (t (list '- (simplify1 (term1 poly)) (simplify-step (term2 poly)))) ))
     ((equal '* (operation poly))
        (cond
           ((equal 0 (term1 poly)) 0)
           ((equal 0 (term2 poly)) 0)
           ((equal 1 (term1 poly)) (simplify-step (term2 poly)))
           ((equal 1 (term2 poly)) (simplify-step (term1 poly)))
           (t (list '* (simplify-step (term1 poly)) (simplify-step (term2 poly)))) ))
     ((equal '^ (operation poly))
        (cond
           ((equal 0 (term2 poly)) 1)
           ((equal 1 (term2 poly)) (simplify-step (term1 poly)))
           ((equal 0 (term1 poly)) 0)
           ((equal 1 (term1 poly)) 1)
           (t (list '^ (simplify-step (term1 poly)) (simplify-step (term2 poly)))) ))
     (t poly) ))
```

Техника символьных вычислений

Пример (продолжение-2) - LISP-реализация списка правил:

Функция simplify-step рекурсивно применяет приведенные в таблице правила к дереву полинома, начиная с корня. Однако она не гарантирует окончательного упрощения полинома.

В чем проблема ?
По мере прохождения дерева мы не можем вернуться на его верхние уровни.

Простейшее решение проблемы - написание функции-оболочки (она будет применять функцию simplify-step к полиному, пока выходное дерево не совпадет с входным. Иными словами вычисление завершится, когда дальнейшие упрощения станут невозможными.

Реализация функции-оболочки (с помощью цикла do):

```
(defun simplify (poly)
   (do ((poly1 poly poly2)
        (poly2 (simplify-step poly) (simplify-step poly2)) )
        ((equal poly1 poly2) poly1) ))
```

Замечание.
Разработанный набор правил упрощения не полон. В частности, в нём отсутствуют правила упрощения в целочисленной арифметике (вместо (* 3 (* 2 x)) следует получить (* 6 x)).

Техника символьных вычислений

Пример (продолжение-3) – Замечания по реализации

Замечание 1.

Некоторые используемые функции не являются встроенными в язык LISP. К ним относятся функции доступа к корню дерева (знаку операции) и его непосредственным потомкам (операндам):

```
(defun operation (poly) (car poly) )
(defun term1 (poly) (cadr poly) )
(defun term2 (poly) (caddr poly) )
```

Замечание 2.

Разработанный набор правил упрощения не полон.

В частности, в нём отсутствуют правила упрощения выражений в целочисленной арифметике:

вместо (* 3 (* 2 x)) следует получить (* 6 x).

Техника символьных вычислений

Выводы:

1) Внутреннее представление математического выражения в системе символьных вычислений – синтаксическое дерево (список списков).

2) Суть аналитических преобразований (символьных вычислений) – переписывание терма с помощью последовательного применения правил из определённого (пользователем или системой) списка.

3) Преобразование из внешнего представления во внутреннее и обратно обеспечивается дополнительными (вне LISP-машины) инструментальными средствами.

Спасибо за внимание !

Вопросы ?