«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Факультет: Информатика и системы управления

Кафедра: ИУ7

Функциональное и логическое программирование

Студент группы ИУ7-63,

Степанов Александр Олегович

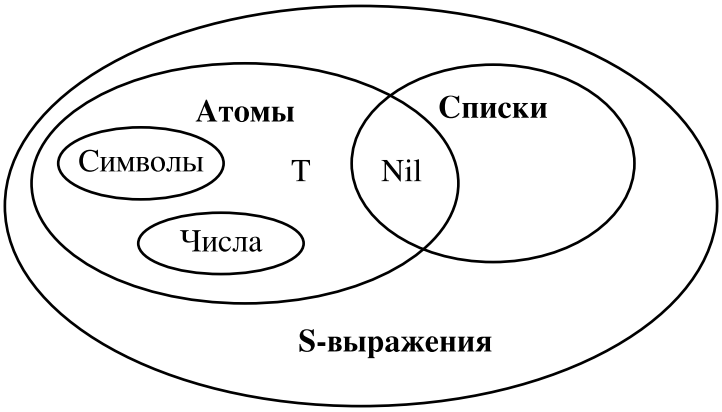
Преподаватель:

Толпинская Наталья Борисовна

2020 г.

Оглавление

1 | Введение



1.1 Классификация функций

1. **Чистые математические функцие** (фиксированное количество аргументов, всегда возращает один результат)
2. **Форма** (функции, которые имеют переменное количество аргументов либо они по-разному обрабатывают все аргументы)
3. **Функцианал** (вместо одного из аргументов принимает функцию)

1.2 Базис языка

Базис языка – базовые структуры и атомы и базовые функции.

1.2.1 Классификация базисных функций

- **Функции селекторы**
  - `car`
  - `cdr`
- **Работа со списками**
  - Создание списка – `cons` (два указателя)

    `(cons 'A 'B)` – точечная пара

    `(cons 'A '(B))` – список
  - Создание списковых ячеек по количеству агрументов – `list` (произволтное количество агрументов)

    `(list 'A 'B)` – список
- **Предикаты**

  Все что не `Nil`, то `T (True)`
  - Атом или нет – `atom`
  - Пустой список или нет – `Null`
  - Список или нет (или списковые ячейки) – `consp`
- **Функции сравнения объектов**
  - `eq` – сравнивает по указателю
  - `eql` – сравнивает числа одного типа (синтаксической формы представления)

    `(eql 3 3.0) - Nil`

    `(eql 3 3) - T`
  - `=` – сравнивает числа по значению

    `(= 3 3.0) - T`
  - `equal` – корректно сравнивает списки
  - `equalp` – наиболее качественно и долго

1.2.2 Способы определения функций

С именем

```
( defun fn(args)
    (S-expression)
)
```

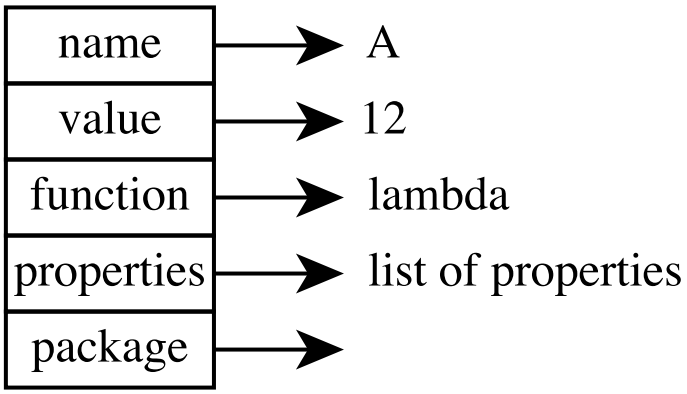
Без имени

```
(lambda (args)
    (S-expression)
)
```

```
(apply #'($\lambda$ expression) arg1...argN)
```

```
(fncall #'($\lambda$ expression) arg1...argN)
```

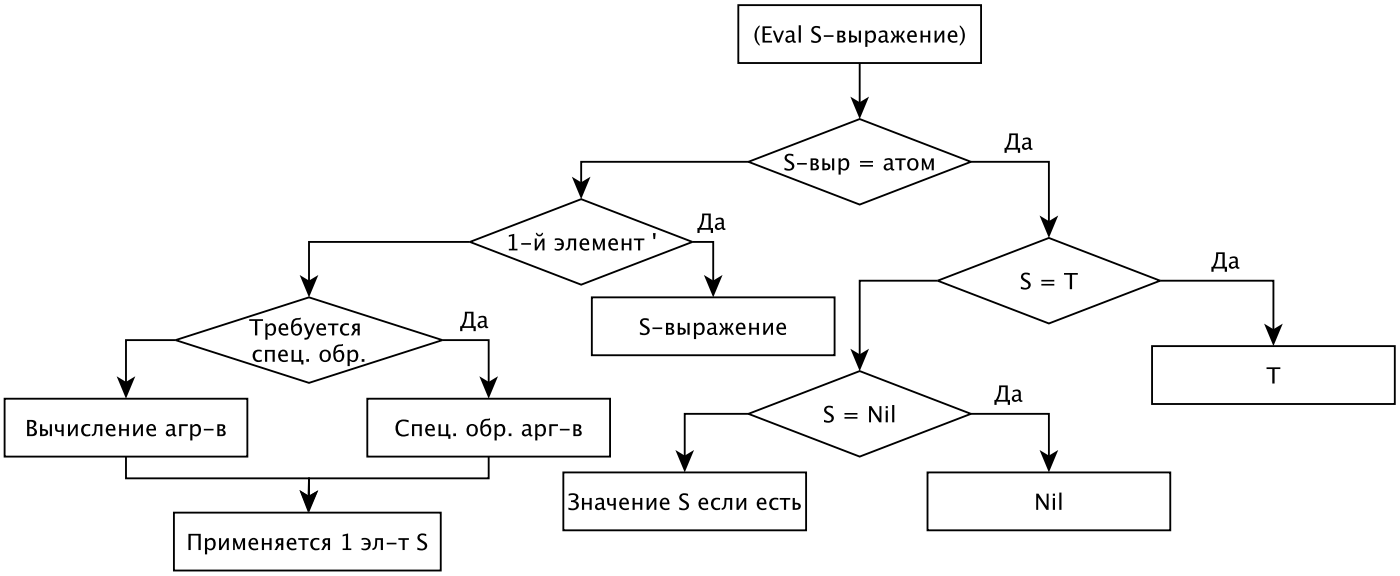
1.2.3 Атом в памяти



Установка значения в атом

```
(setq 'A 12)
```

1.2.4 Диаграмма вычисления S-выражения



1.3 Специальные формы

1.3.1 Ветвление

```
(cond ((conditional1) (body1))
      ((conditional2) (body2))
)
```

1.3.2 Условный оператор

```
(if (conditional) (t-body) (f-body))
```

1.3.3 Локальные переменные

```
(let
    (var1 value1)
    (var2 value2)
    ...
    (varN valueN)
    body
)
```

Сначала вычисляются все значения `value`, и только потом они связываются с переменными, поэтому сослаться на переменную до тела нельзя.

```
(let*
    (var1 value1)
    (var2 value2)
    ...
    (varN valueN)
    body
)
```

Здесь уже можно обращаться до тела.

1.4 Функции модификации списков

- Структуроразрушающие
- Не разрушающие структуру

1. `(append list1 list2)` Работает с копиями, создаются копии всех элементов
2. `(reverse list)` –
3. `(nreverse list)`
4. `nconc` – конкатенация
5. `last` – последний элемент
6. `(nth n list)` – n-я списковая ячейка (нумерация с 0)
7. `(nthcdr n list)` – хвост n-списковой ячейки
8. `(length list)` – количество списковых ячеек на верхнем уровне
9. `(remove el list)` – удаление элемента (сравнение с помощью `eql`)
10. `(rplaca list el)`
11. `(rplacd list el)`

12. `(member el list)` – проверяет есть ли элемент среди списковых ячеек верхнего уровня (возвращает список, начинающийся с перврого вхождения элемента) используется `eql`, которая не сравнивает списки
13. `(union list1 list2)` – множество из двух список (без дубликатов)
14. `(intersection list1 list2)` – пересечение множеств
15. `(set_difference list1 list2)` – разность множеств
16. `(assoc key list)` – работает с ассоциативной таблицей (возвращается списковая ячейка, где ключ совпадает с искомым)
17. `(rassoc value list)` – работает с ассоциативной таблицей (возвращается списковая ячейка, где значение совпадает с искомым)
18. `(acons)` – добавление нового элемента в ассоциативный список

1.5 Ключевые параметры

```
(member '(a b) (f (a b) c) :test #'equal)
```

Вместо стандартного `eql` будет использоваться `equal`.

1.6 Функционалы

1. `(mapcar #'fun list)`
2. `(mapcar #'fun list1 list2 ... listN)`
3. `(maplist #'fun list)`
4. `(mapcan)` – разрушают структуру (работает эффективней)
5. `(reduce '+ '(1 2 3))` – применяет функцию каскадным образом

В качестве функций могут выступать предикаты