**Язык описания атрибутных структур ASL**

# Теория атрибутных систем

## Атрибутная система

Атрибутная система – это вид помеченных систем переходов.

Атрибутная система as – это помеченная система переходов (A, L, S, →) из алфавита A, множества меток L, множества состояний S и отношения перехода →, где

* A = TG ∪ E. Из алфавита строятся метки, состояния и отношение перехода.
  + TG = AN ∪ AM ∪ {undef} – объединение трех непересекающихся множеств. Элементы TG называются вещами (things).
    - AN – множество атрибутонов (attributons),
    - AM – множество атомов (atoms),
    - undef –неопределенное значение (undefined value).
  + E – множество выражений (expressions). Выражения вычисляются в состояниях.
* S = TG × TG ↦ TG удовлетворяет ограничениям:
  + Если tg1 ∈ AM, и tg2 ∈ TG, то s(tg1, tg2) = undef.
  + Если tg ∈ TG, то s(undef, tg2) = undef.
  + Если tg ∈ TG, то s(tg, undef) = undef.
* L = E × GC × LCS × LCS, где
  + GC = TH – множество глобальных контекстов. Глобальный контекст – это контекст, в котором вычисляется выражение. Глобальный контекст не меняется при вычислении выражения.
  + LCS = LC\* – множество последовательностей локальных контекстов. Пусть lc1 + lc2 обозначает конкатенацию последовательностей lc1 и lc2. Пусть LC+ обозначает множество непустых последовательностей локальных контекстов.
  + LC = TH – множество локальных контекстов. Локальный контекст – это контекст, в котором вычисляется выражение. Локальный контекст может меняться при вычислении выражения.
* → ∈ L ↦ S × S удовлетворяет следующим ограничениям:
  + Пусть s1 →(l) s2 – сокращение для (s­1, s2) ∈ →(l).
  + Правило декомпозиции последовательности локальных контекстов: Если lcs1, lcs2 ∈ LC+, s1 →(e, gc, lcs1, lcs3) s2, s2 →(e, gc, lcs2, lcs4) s3, то s1 →(e, gc, lcs1 + lcs2, lcs3 + lcs4) s3.
  + Правило для пустой последовательности локальных контекстов: Если lcs1 – пустая последовательность, то для любого s1 не существует e, gc, lcs2 и s2 таких, что s1 →(e, gc, lcs1, lcs2) s2.

Если s1 →(e, gc, lcs1, lcs2) s2, то говорят, что при вычислении выражения e в состоянии s1 относительно глобального контекста gc и последовательности локальных контекстов lcs1 система as переходит из состояния s1 в состоянии s2 и возвращает (или порождает) последовательность локальных контекстов lcs2. Кортежи (s1, gc, lcs1) и (s2, lcs2) называются входом и выходом (вычисления) выражения e, соответственно.

Если в утверждении s1 →(e, gc, lcs1, lcs2) s2 некоторые компоненты заменены на звездочку (\*), то это означает, что существуют значения этих компонент, при которых утверждение истинно. Например, s1 →(e, \*, \*, lcs2) \*означает, что существуют gc, lcs1, и s2 такие, что s1 →(e, gc, lcs1, lcs2) s2.

Система as называется тотальной, если для любых e ∈ E, lcs1 ∈ LC+, gc ∈ GC и s1 ∈ S существуют s2 ∈ S и lcs2 ∈ LCS такие, что s1 →(e, gc, lcs1, lcs2) s2. Тотальная атрибутная система вычисляет любое выражение для непустой последовательности локальных контекстов.

Пусть s(tg1, tg2, tg3, …, tgn) – сокращение для s(s(tg1, tg2), tg3, …, tgn).

## Свойства атрибутонов

Состояния определяют содержимое атрибутонов (какие атрибуты имеют атрибутоны и значения этих атрибутов).

Атрибутон tg1 имеет атрибут tg2 в состоянии s, если s(tg1, tg2) ≠ undef. Атрибутон tg1 не имеет атрибута tg2 в состоянии s, если s(tg1, tg2) = undef.

Пусть атрибутон tg1 имеет атрибут tg2 в состоянии s. Атрибут tg2 атрибутона tg1 имеет значение tg3 в состоянии s, если s(tg1, tg2) = tg3.

Из определения состояния следуют два свойства:

* Атомы и неопределенные значения не имеют атрибутов.
* Неопределенные значения не могут являться атрибутами никаких вещей.

Специальный атом type определяет типы атрибутонов. Атрибутон tg1 имеет тип tg2 в состоянии s, если av(tg1, tg2) = type. Это пример утиной типизации, когда типы определяются по именам и значениям атрибутов.

## Атрибутные системы с последовательностями

Атрибутная система с последовательностями включает атрибутоны специального вида, называемые последовательностями, которые определяют конечные последовательности вещей и характеризуются атрибутами seqLen и start.

Атрибутная система as называется атрибутной системой с последовательностями, если {seqLen, start} ∪ Integer ⊂ AM, где Integer – множество целых чисел.

Атрибут seqLen определяет количество элементов в последовательности.

Элементы последовательности проиндексированы последовательными целыми числами. Необязательный атрибут start определяет индекс первого элемента последовательности. Если атрибут start не определен, то по умолчанию индекс первого элемента последовательности 1.

Пусть Integer ⊂ AM, где Integer – множество целых чисел

Атрибутон tg называется последовательностью в состоянии s, если s(tg, seqLen) ∈ Integer, и s(tg, seqLen) ≥ 0. Последовательность tg называется пустой последовательностью в состоянии s, если s(tg, seqLen) = 0.

Элемент последовательности tg с индексом i ∈ Integer определяется в состоянии s как s(tg, i). Он является (i-s(tg, start)+1)-м по порядку элементом последовательности.

Заметим, что undef может быть элементом последовательности.

Для атрибутных систем с последовательностями множество меток может быть переопределено как L = E × GC × LC × LC, считая, что локальные контексты могут быть простыми (не последовательность) и составными (последовательность).

## Атрибутные системы с актуальными атрибутонами

В атрибутной системе с актуальными атрибутонами в каждом состоянии множество атрибутонов разделено на два непересекающихся подмножества – множество актуальных атрибутонов (атрибутонов, которые доступны в этом состоянии) и множество потенциальных атрибутонов (атрибутонов, которые не доступны в этом состоянии, но возможно будут доступны в следующих состояниях). Такие системы обеспечивают механизмы добавления новых атрибутонов (перевода их из множества потенциальных атрибутонов в множество актуальных атрибутонов) и удаления старых атрибутонов (перевода их из множества актуальных атрибутонов в множество потенциальных атрибутонов).

Атрибутная система as называется атрибутной системой с актуальными атрибутонами, если actualAttributon ∈ AM.

Атрибутон tg является актуальным в состоянии s, если tg имеет тип actualAttributon в состоянии s. Атрибутон tg является потенциальным в состоянии s, если tg не имеет типа actualAttributon в состоянии s.

## Свойства выражений

Вычисление выражения может переводить систему as из одного состояния в другое.

Выражение e возвращает (или порождает) состояние s2 относительно входа (s1, gc, lcs1) и выхода lcs2, если s1 →(e, gc, lcs1, lcs2) s2.

Атрибутная система as называется атрибутной системой с джампами, если jump ∈ AM. Для атрибутных систем с джампами вычисление выражений может возвращать (или порождать) джампы – специальные значения, описывающие ситуации, требующие специальной обработки. Джампы определяются атрибутом jump.

Выражение e возвращает (или порождает) джамп tg относительно входа (s1, gc, lcs1) и выхода (s2, lcs2), если tg ≠ undef, s1 →(e, gc, lcs1, lcs2) s2, и существует lc ∈ lc2 такой, что s2(lc, jump) = tg.

Атрибутная система as называется атрибутной системой со значениями, если value ∈ AM. Для атрибутных систем со значениями вычисление выражений может возвращать значения. Значения выражений определяются атрибутом value.

Выражение e возвращает (или порождает) значение tg относительно входа (s1, gc, lcs1) и выхода (s­2, lcs2), если tg ≠ undef, s1 →(e, gc, lcs1, lcs2) s2, и существует lc ∈ lc2 такой, что s2(lc, value) = tg, и если as – атрибутная система с джампами, то s2(lc, jump) = undef.

# Язык ASL

ASL (Attribute Systems Language) – язык описания атрибутных систем с джампами, значениями, последовательностями и актуальными атрибутонами. Он является регистро-зависимым языком.

# Описание компонент атрибутных систем в языке ASL

## Тип Thing

Множество TG атрибутной системы определяется в языке ASL супертипом Thing.

## Тип Atom

Множество AN атомов атрибутной системы определяется в языке ASL подтипом Atom типа Thing.

### Подтипы типа Atom

Язык ASL включает следующие подтипы типа Atom.

#### Тип String

Тип String строк определяется следующим образом:

* если u - Unicode-строка, то ′′u′′ ∈ String.

Как обычно, в строке u могут экранироваться символы ′′, \ , r, n и t, например, \t.

#### Тип QName

Тип QName квалифицированных имен определяется следующим образом:

* если u­1, u2, …, un – строки, которые состоят из цифр 0-9, латинских букв (a-z, A-Z) и прочерка (\_) и начинаются не с цифры, то u1:u2: … :un ∈ QName. В частности, u1 ∈ QName.
* undef ∉ QName.
* если u ∈ String, то qname(u) ∈ QName.

Предопределенная функция qname строит по строке новый элемент типа QName. Для одной и той же строки элемент строится только один раз.

#### Тип Boolean

Тип Boolean является подтипом типа QName и имеет только два элемента true и false.

#### Тип Integer

Элементами типа Int являются целые числа в стандартной нотации, например -2, 6, 0 и т. п.

#### Тип Double

Элементами типа Double являются числа с фиксированной точкой в стандартной нотации, например, -2.6, 3.2e-5 и т. п.

Элементы типа Integer могут неявно приводиться к типу Double.

### Синтаксические атомы

К синтаксическим атомам относятся элементы типов String, QName, Boolean, Integer и Double. Заметим, что могут быть атомы, которые не являются синтаксическими.

## Тип Undef

Неопределенное значение undef определяется в языке ASL подтипом Undef типа Thing.

## Тип Attributon

Множество AN атрибутной системы определяется в языке ASL подтипом Attributon типа Thing.

## Локальные контексты

Локальные контексты атрибутных структур определяются в языке ASL элементами типа Attributon. В дополнении к атрибутам value и jump, к специальным атрибутам (атрибутов с фиксированной семантикой) локальных контекстов в языке ASL относится атрибуты:

* variables, который описывает имена и значения переменных: переменная $u определена в состоянии s относительно локального контекста lc, если s(lc, variables, u) ≠ undef. В этом случае, говорят, что переменная $u имеет значение s(lc, variables, u) в состоянии s;
* attributonVariables, который описывает имена и значения атрибутонных переменных: атрибутонная переменная #u определена в состоянии s относительно локального контекста lc, если s(lc, attributonVariables, u) ≠ undef. В этом случае, говорят, что переменная $u имеет значение s(lc, attributonVariables, u) в состоянии s;
* parent, который специфицирует локальный контекст, в котором была вызвана функция, которая вычисляется в текущем контексте. Локальный контекст lc1 называется родителем контекста lc2 в состоянии s, если lc1 = s(lc2, parent).

## Глобальный контекст

Глобальный контекст gc имеет следующие специальные атрибуты:

* Атрибут functions содержит информацию о пользовательских функциях. Значениe w = s(gc, functions, f, n) является последовательностью длины n, элементы которой являются параметрами функции с именем f ∈ QName местности n. Атрибут body последовательности w определяет тело этой функции.

## Выражения

Множество E атрибутной системы определяется в языке ASL на синтаксическом и онтологическом уровнях как множество синтаксических и онтологических выражений, соответственно.

### Переменные

Переменные используются в качестве ссылок на вещи. Они определяются следующим образом:

* если u ∈ QName, то $u – переменная с именем u.

### Атрибутонные переменные

Атрибутонные переменные используются в качестве ссылок на атрибутоны. Они определяются следующим образом:

* если u ∈ QName, то #u – переменная с именем u.

### Синтаксические выражения

Множество синтаксических выражений определяется следующим образом:

* Если e – синтаксический атом, то e – синтаксическое выражение;
* undef – синтаксическое выражение;
* Если e – переменная, то e – синтаксическое выражение;
* Если e – атрибутонная переменная, то e – синтаксическое выражение;
* Если e – синтаксическое выражение, то (e) – синтаксическое выражение (выражение в скобках);
* Если f ∈ QName, e1, …, en – синтаксические выражения, то tg(e1, …, en) – синтаксическое выражение (вызов функции с именем f местности n).

### Онтологические выражения

Множество онтологических выражений совпадает с множеством элементов типа Thing.

### Онтологические модели синтаксических выражений

Для каждого синтаксического выражения есть соответствующее ему онтологическое выражение, называемое онтологической моделью синтаксического выражения. Модель называется онтологической, так как она определяется на концептуальном уровне (как сеть вещей, связанных через значения атрибутов). Синтаксическое выражение может иметь более одной соответствующей ему онтологической модели. Отношение →ont, сопоставляющее синтаксические выражения с их онтологическими моделями называются онтологической семантикой синтаксических выражений.

### Онтологическая семантика синтаксических выражений

Пусть {a1 = tg1, …, an = tgn} обозначает атрибутон, который является актуальным в состоянии s и имеет в этом состоянии помимо атрибута actualAttributon атрибуты a1, …, an со значениями tg1, … tgn. Пусть (tg1, …, tgn) обозначает последовательность {seqLen = n, 1 = tg1, …, n = tgn}. В частности, () обозначает последовательность {seqLen = 0}. Отношение →ont определяется следующим образом относительно состояния s:

* Если e – синтаксический атом, то e →ont e;
* undef →ont undef;
* Если e – переменная с именем u, то e →ont {variable = type, name = u};
* Если e – атрибутонная переменная с именем u, то e →ont {attributonVariable = type, name = u};
* Если e →ont tg, то (e) →ont tg;
* Если e1 →ont tg1,…, en →ont tgn, то tg(e1, …, en) →ont {functionCall = type, function = tg, seqLen = n, start = 1, 1 = tg1, …, n = tgn}.

Отношение перехода в языке ASL определяется на онтологическом уровне (на онтологических выражениях).

## Отношение перехода для онтологических выражений

Пусть upd(f, x, y) обозначает функцию f2, которая совпадает с функцией f для всех аргументов кроме возможно аргумента x, и f2(x) = y. Пусть upd(f, x1, y1, x2, y2, …, xn, yn) – сокращение для upd(upd(f, x1, y1), x2, y2, …, xn, yn).

Фраза «породить джамп типа t» означает выполнить переход s →(e, gc, (lc), (lc)) upd(s, (tg, actualAttributon), type, (tg, t), type, (lc, jump), tg), где tg – потенциальный атрибутон в s.

Отношение перехода для онтологических выражений (в частности, онтологических моделей синтаксических выражений) определяется следующим образом:

1. Случай переменных
   * Если e ∈ Attributon, и e имеет тип variable в s, то
     + Если s(e, name) = tg ∈ QName, то s →(e, gc, (lc), (lc)) upd(s, (lc, value), s(lc, variables, tg));
     + В противном случае, породить джамп типа variableJump;
2. Случай атрибутонных переменных
   * Если e ∈ Attributon, и e имеет тип attributonVariable в s, то
     + Если av(e, name) = tg ∈ QName, то
       - Если s(lc, attributonVariables, tg) ≠ undef, то s →(e, gc, (lc), (lc)) upd(s, (lc, value), s(lc, attributonVariables, tg));
       - В противном случае, если tg1 – потенциальный атрибутон в s, то s →(e, gc, (lc), (lc)) upd(s, (tg1, actualAttributon), type, (lc, value), tg1);
     + В противном случае, породить джамп типа attributonVariableJump.
3. Случай вызова функции
   * Если e ∈ Attributon, и e имеет тип functionCall в s, то
     + Если s(e, function) ∈ QName, e - последовательность, и выполнено одно из двух свойств: s →pf(e, gc, (lc), lcs) s1, или s →df(e, gc, (lc), (lc)) s1, то s(e, gc, (lc), lcs) → s1.
     + В противном случае, породить джамп типа functionCallJump;
4. В противном случае, s →(e, gc, (lc), (lc)) upd(as, (lc, value), e).

Отношения перехода →pf и →df определяют вызовы предопределенных и определяемых пользователем функций.

### Отношение перехода для вызова предопределенной функции

Отношение →pf определяется реализацией в соответствие со спецификацией предопределенных функций (раздел 6).

### Отношение перехода для вызова пользовательской функции

Пусть e – вызов функции с именем f местности n. Тогда f = s(e, function), и e - последовательность длины n, где s(e, i) – i-й аргумент вызова e, 1 ≤ i ≤ n. Пусть переменная cur определена на состояниях и содержит текущее состояние. Отношение →df определяется следующим алгоритмом:

* Присвоить значение s переменной cur.
* Если w = s(gc, functions, f, n) ∈ Attributon, последовательность w состоит из элементов p1, …, pn, и s(w, body) ≠ undef, то
  + Породить копию lc1 контекста lc (новый атрибут с тем же самым содержимым, что и lc).
  + Присвоить атрибуту variables контекста lc1 значение undef.
  + Присвоить значение lc атрибуту parent контекста lc1.
  + Для каждого i от 1 до n
    - Вычислить выражение s(e, i).
    - Если s(e, i) возвращает джамп u в cur, то возвратить джамп u. Завершить алгоритм.
    - Если s(e, i) возвращает значение u в cur, то добавить атрибут pi со значением u к атрибутону s(lc1, variables).
  + Если тело s(w, body) функции f, вычисляемое как progn, возвращает джамп u типа, отличного от returnJump, в состоянии cur относительно локального контекста lc1, то атрибуту jump атрибутона lc присвоить значение u, а атрибутам, отличным от jump, value, variables, attributonVariable и parent – значения соответствующих атрибутов атрибутона lc1. Завершить алгоритм.
  + Если тело s(w, body) функции f, вычисляемое как progn, возвращает джамп типа returnJump со значением u в состоянии cur относительно локального контекста lc1, то атрибуту value атрибутона lc присвоить значение u, а атрибутам, отличным от jump, value, variables, attributonVariables и parent – значения соответствующих атрибутов атрибутона lc1. Завершить алгоритм.
  + Если тело s(w, body) функции f, вычисляемое как progn, возвращает значение v в состоянии cur относительно локального контекста lc1, то атрибуту value атрибутона lc присвоить значение v, а атрибутам, отличным от jump, value, variables, attributonVariable и parent – значения соответствующих атрибутов атрибутона lc1. Завершить алгоритм.
* В противном случае, породить джамп типа functionCallJump.

Заметим, что если выражение s(e, i) порождает несколько локальных контекстов, то нужно применить правило декомпозиции последовательности локальных контекстов.

# Заметки по интерпретатору

Нам не требуется заводить класс State, так как вся информация о состоянии s хранится в экземплярах класса Thing и производных от него классов.

Тогда интерпретатор можно определить как функцию

Thing\* interpret(Thing e, Thing gc, Thing\* lcs1).

Функция вычисляет выражение e в текущем состоянии s1 относительно глобального контекста gc и последовательности локальных контекстов lcs1, возвращает последовательность локальных контекстов lcs2 и переводит программу в состоянии s2 в соответствие с переходом s1 →(e, gc, lc1, lc2) s2.

Тип Thing\* можно определить двумя способами:

# как вектор (на языке Java или C++) элементов типа Thing;

# как Thing, используя тот факт, что последовательности – частный случай атрибутонов.

В объектно-ориентированном стиле интерпретатор можно определить как метод на соответствующих классах Variable, AttributonVariable, FunctionCall, Attributon, Atom, Undef.

Thing\* interpret(Thing gc, Thing\* lcs1).

Выражение e (в онтологическом представлении) является в этом случае экземпляром класса, на котором вызывается интерпретатор.

# Синтаксические расширения языка ASL

Для удобства ряд функций языка ASL представляются в виде операторов и операций.

## Операторы

|  |  |
| --- | --- |
| **Синтаксическое расширение** | **Выражение** |
| defun f(x1, …, xn) {z1;}. Символ ; в z1; может опускаться. | defun(f, x1, …, xn, z1) |
| defun f(x1, …, xn) {z1; … ; zm;}, где m ≠ 1. Символ ; в zm; может опускаться. | defun(f, x1, …, xn, progn(z1, …, zm)) |
| while (x) y | while(x, y) |
| if (x) then y else z | cond(x, y, z) |
| if (x) then y | cond(x, y) |
| if (x) then y else z | cond(x, y, z) |
| if (x) then y | cond(x, y) |
| {x1; …; xn;}, где n ≥ 0. Символ ; в xn; может опускаться. | progn(x1, …, xn) |

## Операции

|  |  |
| --- | --- |
| **Синтаксическое расширение** | **Выражение** |
| -x | uminus(x) |
| !x | not(x) |
| x + y | add(x, y) |
| x – y | sub(x, y) |
| x \* y | mul(x, y) |
| x / y | div(x, y) |
| x % y | mod(x, y) |
| x == y | eq(x, y) |
| x != y | neq(x, y) |
| x > y | gt(x, y) |
| x >= y | gte(x, y) |
| x < y | lt(x, y) |
| x <= y | lte(x, y) |
| x && y | and(x, y) |
| x || y | or(x, y) |
| x = y | setq(x, y) |
| x.y, (конструкция x.y может заменяться на x[y]) | aref(x, y) |
| x.y = z (конструкция x.y может заменяться на x[y]) | aset(x, y, z) |
| .x | globaref(x, y) |
| .x = y | globaset(x, y) |
| ..x | locaref(x) |
| ..x = y | locaset(x, y) |
| {% x1 = y1, …, xn = yn %} | conz(x1, y1, …, xn, yn) |
| #z = {% x1 = y1, …, xn = yn %} | conzExtra(#z, x1, y1, …, xn, yn) |
| (% x1, …, xn %), где n > 0 | seqCreate(x1, …, xn) |
| #z = (% x1, …, xn %), где n > 0 | seqCreateExtra(#z = seqCons(x1, undef), x2, …, xn) |

Встроенные функции в алфавитном порядке перечислены в Разделе 6 и по категориям в Разделе 7.

## Предвычисляемые выражения

Предвычисляемые выражения вычисляются ровно один раз при загрузке выражения (перевода его в онтологическую модель), в которое они входят.

Предвычисляемое выражение (или backslash-выражение) определяется следующим образом:

Если e – выражение, то `e` – предвычисляемое выражение.

Оно имеет следующую семантику:

* Вычислить выражение e.
* Заменить вхождение предвычисляемого выражения `e` на значение выражения e.

Предвычисляемые выражения могут вкладываться друг в другу. В этом случае, они вычисляются слева направо (самое левое выражение первым) и снизу вверх (самое внутреннее выражение первым).

Трансляция выражения в онтологическую модель выполняется только после того как вычислены все входящие в него предвычисляемые выражения.

## Таблица приоритетов операторов и операций

От высокого к низкому

|  |  |
| --- | --- |
| **Приоритет** | **Операторы и операции** |
| 1 | `…` |
| 2 | {…}, (…), {%…%}, (%…%) |
| 3 | .. |
| 4 | .(унарная) |
| 5 | .(бинарная) |
| 6 | !, -(унарный), |
| 7 | \*, /, % |
| 8 | +, -(бинарный) |
| 9 | >, >=, <, <= |
| 10 | =, != |
| 11 | && |
| 12 | || |
| 13 | = |
| 14 | while, if, defun |

## Таблица ассоциативности операций

|  |  |
| --- | --- |
| **Тип ассоциативности** | **Операции** |
| левая | .(бинарная), \*, /, %, +, -(бинарный), &&, || |
| правая | нет |
| отсутствует | остальные операции |

# Встроенные функции в алфавитном порядке

Пусть opt перед аргументом означает, что этот аргумент необязательный.

## add

Функция add имеет аргументы (x1, …, xn) и определяется следующим образом:

* Пусть x1, …, xn возвращают значения u1, …, un.
* Если n = 0, то возвратить значение 0.
* Если n > 0, то
  + Если u1, …, un ∈ Integer ∪ Double, то возвратить значение, которое является суммой w чисел u1, …, un.
    - Если u1, …, un ∈ Integer, то w ∈ Integer.
    - Если ui ∈ Double для некоторого 1 ≤ i ≤ n, то w ∈ Double.
  + Если u1, …, ui-1 ∈ Integer ∪ Double, и ui ∉ Integer ∪ Double, то не вычислять xi+1, …, xn и возвратить джамп типа addJump.

## and

Функция and имеет аргументы (x1, …, xn) и определяется следующим образом:

* Пусть x1, …, xn возвращают значения u1, …, un.
* Если n = 0, то возвратить значение true.
* Если n > 0, то
  + Если u1, …, un ∈ {true}, то возвратить значение true.
  + Если u1, …, ui-1 ∈ {true}, и ui = false, то не вычислять xi+1, …, xn и возвратить значение false
  + Если u1, …, ui-1 ∈ {true}, и ui ∉ Boolean, то не вычислять xi+1, …, xn и возвратить джамп типа andJump.

## aref

Функция aref имеет аргументы (x0, x1, …, xn) и определяется следующим образом:

* Пусть x0 возвращает значение u0.
* Если n = 0, то возвратить значение u0.
* Если n > 0, то
  + Если u0 ∉ Attributon, то возвратить джамп типов arefJump и Undef.
  + Если u0 ∈ Attributon, то.
    - Пусть выражения x1, …, xn возвращают значения u1, …, un.
    - Пусть vi = av(ui-1, ui).
    - Если v1, …, vi-1 ∈ Attributon, vi ∉ Attributon, и 1≤ i<n, то xi+1, …, xn не вычислять, и возвратить джамп типов arefJump и Undef.
    - Если v1, …, vn-1 ∈ Attributon, то возвратить значение vn.

## aset

Функция aset имеет аргументы (x0, x1, …, xn, y, z) и определяется следующим образом:

* Пусть aref(x0, x1, …, xn) возвращает значение u.
* Если u ∉ Attributon, то возвратить джамп типа asetJump.
* Если u ∈ Attributon, то
  + Пусть у возвращает значение uy.
  + Если z возвратить джамп типа Undef, то удалить атрибут uy у атрибутона u.
  + Если z возвращает джамп w, который не имеет тип Undef, то возвратить джамп w.
  + Если z не возвращает джамп и возвращает значение uz, то присвоить значение uz атрибуту uy атрибутона u и возвратить значение uz.

## assert

Функция assert имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Если u = type, то возвратить текущее состояние.
* Если u ∉ {true}, то возвратить джамп типа assertJump.

## assertEq

Функция assertEq имеет аргументы (x, y) и определяется следующим образом:

* Пусть x и y возвращают значения ux и uy.
* Если ux = uy, то возвратить значение true.
* Если ux ≠ uy, то возвратить джамп типа assertEqJump.

## catch

Функция catch имеет аргументы (x, y1, …, yn) и определяется следующим образом:

* Пусть av(lc, jump) = u.
* Если u = undef, то возвратить текущее состояние.
* Если u ≠ undef, то
  + Вычислить x.
  + Если x возвращает джамп w, то возвратить джамп w.
  + Если x не возвращает джамп, то
    - Пусть x возвращает значение ux.
    - Если u имеет тип ux, то присвоить атрибуту previousJump текущего состояния значение u и вычислить progn(y1, …, yn).
    - Если u не имеет тип ux, то возвратить джамп u.

## coalesce

Функция coalesce имеет аргументы (x1, …, xn) и определяется следующим образом:

* Если n = 0, то coalesce возвращает текущее состояние.
* Если n > 0, то
  + Вычислить x1, …, xn. Перед вычислением каждого xi, где i > 1, атрибут jump удаляется из текущего состояния.
  + Если 0 < i ≤ n, x1, …, xi-1 возвращают джампы типа Undef, xi возвращает значение u, то возвратить значение u.
  + Если 0 < i ≤ n, x1, …, xi-1 возвращают джампы типа Undef, и xi возвращает джамп w, который не имеет тип Undef, то возвратить джамп w.
  + Если x1, …, xn возвращают джампы типа Undef, и xn возвращает джамп w, то возвратить джамп w.

## cond

Функция cond имеет аргументы (x1, y1, …, xn, yn, opt z), где n>1. Аргументы x1, …. xn называются условиями. Эта функция определяется следующим образом:

* Пусть x1, …. xn возвращают значения u1, …, un.
* Если u1, …, ui-1 ∈ {false}, ui = type, 1≤ i≤ n, то вычислить yi.
* Если u1, …, ui-1 ∈ {false}, ui ∉ Boolean, и 1≤ i≤ n, то не вычислять xi+1, …, xn и возвратить джамп типа condJump.
* Если u1, …, un ∈ {false}, и z присутствует, то вычислить z.
* Если u1, …, un ∈ {false}, и z отсутствует, то возвратить текущее состояние.

## conz

Функция conz имеет аргументы (x1, y1, …, xn, yn) и определяется следующим образом:

* Породить новый атрибутон u.
* Вычислить последовательно x1, y1, …, xn, yn.
* Пусть x1, y1, …, xn, yn возвращают значения ux1, uy1, …, uxn, uyn.
* Если yi возвращает значение undef, то удалить атрибут uxi атрибутона u.
* Если yi не возвращает значение undef, то присвоить атрибуту uxi атрибутона u значение uyi.

## conzExtra

Функция conzExtra имеет аргументы (z, x1, y1, …, xn, yn) и определяется следующим образом:

* Если z ∈ Attributon, то
  + Вычислить последовательно x1, y1, …, xn, yn.
  + Пусть x1, y1, …, xn, yn возвращают значения ux1, uy1, …, uxn, uyn.
  + Если yi возвращает значение undef, то удалить атрибут uxi атрибутона z.
  + Если yi не возвращает значение undef, то присвоить атрибуту uxi атрибутона z значение uyi.
* В противном случае, возвратить джамп типа conzExtraJump.

## copyDeep

Функция copyDeep имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Если u ∉ Attributon, то copyDeep возвращает значение u.
* Если u ∈ Attributon, то породить новый атрибутон u1, определить те же самые атрибуты у u1 как у u, и рекурсивно применить copyDeep к значениям этих атрибутов.
* Возвратить значение u1.

## copySeq

Функция copySeq имеет аргументы (x) и эквивалентна copyShallow(x).

## copyShallow

Функция copyShallow имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Если u ∉ Attributon, то возвратить значение u.
* Если u ∈ Attributon, то породить новый атрибутон u1, определить те же самые атрибуты у u1 как у u, и присвоить им те же самые значения.
* Возвратить значение u1.

## copyState

Функция copyState имеет аргументы () и определяется следующим образом:

* Возвратить значение, которое является копией текущего состояния s.
  + Копия имеет те же самые атрибуты, что и s.
  + Значениями атрибутов variableValue, jump и previousJump копии являются копии значений соответствующих атрибутов состояния s.
  + Значения остальных атрибутов совпадают со значениями соответствующих атрибутов состояния s.

## defun

Функция defun имеет аргументы (x, y, z) и определяется следующим образом:

* Пусть n – длина последовательности y.
* Вычислить globaref(functions, x, n, conz(DefinedFunction, true, parameters, quote(y), body, quote(z)).

## div

Функция div имеет аргументы (x, y) и определяется следующим образом:

* Пусть x, y возвращают значения ux, uy.
* Если ux ∉ Integer ∪ Double, то не вычислять y и возвратить джамп типа Div.
* Если ux ∈ Integer ∪ Double, то
  + Если uy ∉ Integer ∪ Double, то возвратить джамп типа divJump.
  + Если uy ∈ Integer ∪ Double, то
    - Если uy ≠ 0, то
      * Если ux, uy ∈ Integer, то возвратить значение, которое является остатком от деления ux на uy.
      * Если ux ∉ Integer, или uy ∉ Integer, то возвратить значение, которое является частным от деления ux на uy.
    - Если uy = 0, то возвратить джамп типа divJump.

## eq

Функция eq имеет аргументы (x1, …, xn), где n > 0, и определяется следующим образом:

* Пусть x1, …, xn возвращают значения u1, …, un.
* Если 1<i≤ n, и u1, …, ui-1 попарно равны, и ui ≠ ui-1, то не вычислять xi+1, …, xn и возвратить значение false.
* Если u1, …, un попарно равны, то возвратить значение true.

## eqDeep

Функция eqDeep имеет аргументы (x1, …, xn), где n > 0. Пусть свойство P(u1, u2) означает, что либо u1 и u2 – не атрибутоны и u1 = u2, либо u1 и u2 –атрибутоны, которые имеют один и тот же набор атрибутов и для значений соответствующих атрибутов функция eqDeep возвращает значение true. Тогда eqDeep определяется следующим образом:

* Пусть x1, …, xn возвращают значения u1, …, un.
* Если 1<i≤ n, и P(u1, uj) для 1 < j < i, и P(u1, ui), то не вычислять xi+1, …, xn и возвратить значение false.
* Если P(u1, ui) для 1 < i ≤ n, то возвратить значение true.

## eqSeq

Функция eqSeq имеет аргументы (x1, …, xn), где n > 0. Пусть свойство P(u1, u2) означает, что либо u1 и u2 – последовательности длины 0, либо u1 и u2 – последовательности длины n > 0, которые имеют те же самые элементы и в том же самом порядке. Тогда eqSeq определяется следующим образом:

* Пусть x1, …, xn возвращают значения u1, …, un.
* Если 1<i≤ n, и P(u1, uj) для 1 < j < i, и P(u1, ui), то не вычислять xi+1, …, xn и возвратить значение false.
* Если P(u1, ui) для 1 < i ≤ n, то возвратить значение true.

## eqShallow

Функция eqShallow имеет аргументы (x1, …, xn), где n > 0. Пусть свойство P(u1, u2) означает, что либо u1 и u2 – не атрибутоны и u1 = u2, либо u1 и u2 –атрибутоны, которые имеют один и тот же набор атрибутов и значения соответствующих атрибутов равны. Тогда eqShallow определяется следующим образом:

* Пусть x1, …, xn возвращают значения u1, …, un.
* Если 1<i≤ n, и P(u1, uj) для 1 < j < i, и P(u1, ui), то не вычислять xi+1, …, xn и возвратить значение false.
* Если P(u1, ui) для 1 < i ≤ n, то возвратить значение true.

## eval

Функция eval имеет аргументы (x, opt y, opt z). По умолчанию аргументы y и z возвращают значения, которые являются текущим локальным контекстом и текущим глобальным контекстом. Эта функция определяется следующим образом:

* Пусть y, z возвращают значения uy, uz.
* Если y присутствует, то заменить текущий локальный контекст на uy.
* Если z присутствует, то заменить текущий глобальный контекст на uz.
* Вычислить x.
* Вернуться к текущим локальному и глобальному контекстам.
* Установить атрибуты value и jump текущего контекста равными значениям этих атрибутов контекста uy.

## evalArg

Функция evalArg имеет аргументы (x). Эта функция определяется следующим образом:

* Пусть lc – текущий локальный контекст, и lcp – родитель lc.
* Вычислить выражение eval(x, lcp).

## fileToString

Функция fileToString имеет аргументы (x), где x – строка, определяющая полное имя файла.

Пока не нужно реализовывать.

## functions

Функция functions имеет аргументы () и определяется следующим образом:

* Возвратить значение av(s, functions).

## getState

Функция getState имеет аргументы () и определяется следующим образом:

* Возвратить значение, которое является копией текущего состояния s.
  + Копия имеет те же самые атрибуты, что и s.
  + Значениями атрибутов variableValue, jump и previousJump копии являются копии значений соответствующих атрибутов состояния s.
  + Значения остальных атрибутов совпадают со значениями соответствующих атрибутов состояния s.

## globaref

Функция globaref имеет аргументы (x1, …, xn) и определяется следующим образом:

* Вычислить aref(quote(gc), x1, …, xn).

## globaset

Функция globaset имеет аргументы (x1, …, xn) и определяется следующим образом:

* Вычислить aset(quote(gc), x1, …, xn).

## gt

Функция gt имеет аргументы (x1, …, xn). Пусть > - бинарная операция «больше» на числах или строках. Функция gt определяется следующим образом:

* Пусть x1, …, xn возвращают значения u1, …, un.
* Если n ≤ 1, то возвратить значение true.
* Если n > 1, то
  + Если u1, …, ui ∈ Integer ∪ Double ∪ String, uj-1 > uj для 1 < j < i < n, и неверно, что ui-1 > ui, то не вычислять xi+1, …, xn и возвратить значение false.
  + Если u1, …, ui ∈ Integer ∪ Double ∪ String, uj-1 > uj для 1 < j < i < n, и ui ∉ Integer ∪ Double ∪ String, то не вычислять xi+1, …, xn и возвратить джамп типа gtJump.
  + Если u1, …, un ∈ Numeric ∪ String, и ui-1 > ui для 1 < i ≤ n, то возвратить значение true.

## gte

Функция gte имеет аргументы (x1, …, xn). Пусть ≥ - бинарная операция «больше или равно» на числах или строках. Функция gte определяется следующим образом:

* Пусть x1, …, xn возвращают значения u1, …, un.
* Если n ≤ 1, то возвратить значение true.
* Если n > 1, то
  + Если u1, …, ui ∈ Integer ∪ Double ∪ String, uj-1 ≥ uj для 1 < j < i < n, и неверно, что ui-1 ≥ ui, то не вычислять xi+1, …, xn и возвратить значение false.
  + Если u1, …, ui ∈ Integer ∪ Double ∪ String, uj-1 ≥ uj для 1 < j < i < n, и ui ∉ Integer ∪ Double ∪ String, то не вычислять xi+1, …, xn и возвратить джамп типа gteJump.
  + Если u1, …, un ∈ Integer ∪ Double ∪ String, и ui-1 ≥ ui для 1 < i ≤ n, то возвратить значение true.

## HLSRLLoad

Семантика будет описана позже.

## isAtom

Функция isAtom имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Если u ∈ Atom, то возвратить значение true.
* Если u ∉ Atom, то возвратить значение false.

## isBoolean

Функция isBoolean имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Если u ∈ Boolean, то возвратить значение true.
* Если u ∉ Boolean, то возвратить значение false.

## isDef

Функция isDef имеет аргументы (x) и определяется следующим образом:

* Если x не возвращает значение undef, то возвратить значение true.
* Если x возвращает значение undef, то возвратить значение false.

## isDouble

Функция isDouble имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Если u ∈ Double, то возвратить значение true.
* Если u ∉ Double, то возвратить значение false

## isInteger

Функция isInteger имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Если u ∈ Integer, то возвратить значение true.
* Если u ∉ Integer, то возвратить значение false.

## isNumeric

Функция isNumeric имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Если u ∈ Numeric, то возвратить значение true.
* Если u ∉ Numeric, то возвратить значение false.

## isQName

Функция isQname имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Если u ∈ QName, то возвратить значение true.
* Если u ∉ QName, то возвратить значение false.

## isString

Функция isString имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Если u ∈ String, то возвратить значение true.
* Если u ∉ String, то возвратить значение false.

## isAttributon

Функция isAttributon имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Если u ∈ Attributon, то возвратить значение true
* Если u ∉ Attributon, то возвратить значение false.

## isUndef

Функция isUndef имеет аргументы (x) и определяется следующим образом:

* Если x возвращает значение undef, то возвратить значение true.
* Если x не возвращает значение undef, то возвратить значение false.

## jump

Функция jump имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Возвратить джамп u.

## load

Функция load имеет аргументы (x), где x – имя файла, и определяется следующим образом:

* Пусть x возвращают значение u.
* Если u ∉ String, то возвратить джамп типа loadJump.
* Если файл с именем u не загружается, то возвратить джамп типа loadJump.
* Если содержимое файла с именем u не является последовательностью ASL-выражений, то возвратить джамп типа loadJump.
* Если e1, …, en – содержимое файла с именем u, то вычислить progn(e1, …, en).

## loadData

Функция loadData имеет аргументы (x), где x – имя файла, и определяется следующим образом:

* Пусть x возвращают значение u
* Если u ∉ String, то возвратить джамп типа loadDataJump.
* Если файл с именем u не загружается, то возвратить джамп типа loadDataJump.
* Если содержимое файла с именем u не является последовательностью ASL-выражений, то возвратить джамп типа loadDataJump.
* Если e1, …, en – содержимое файла с именем u, то преобразовать e1, …, en в их онтологические модели m1, …, mn и возвратить значение mn.

## locaref

Функция locaref имеет аргументы (x1, …, xn) и определяется следующим образом:

* Вычислить aref(quote(lc), x1, …, xn).

## locaset

Функция locaset имеет аргументы (x1, …, xn) и определяется следующим образом: Вычислить aset(quote(lc), x1, …, xn).

## lt

Функция lt имеет аргументы (x1, …, xn). Пусть < - бинарная операция «меньше» на числах или строках. Функция lt определяется следующим образом:

* Пусть x1, …, xn возвращают значения u1, …, un.
* Если n ≤ 1, то возвратить значение true.
* Если n > 1, то
  + Если u1, …, ui ∈ Integer ∪ Double ∪ String, uj-1 < uj для 1 < j < i < n, и неверно, что ui-1 < ui, то не вычислять xi+1, …, xn и возвратить значение false.
  + Если u1, …, ui ∈ Integer ∪ Double ∪ String, uj-1 < uj для 1 < j < i < n, и ui ∉ Integer ∪ Double ∪ String, то не вычислять xi+1, …, xn и возвратить джамп типа ltJump.
  + Если u1, …, un ∈ Integer ∪ Double ∪ String, и ui-1 < ui для 1 < i ≤ n, то возвратить значение true.

## lte

Функция lte имеет аргументы (x1, …, xn). Пусть ≤ - бинарная операция «меньше или равно» на числах или строках. Функция lte определяется следующим образом:

* Пусть x1, …, xn возвращают значения u1, …, un.
* Если n ≤ 1, то возвратить значение true.
* Если n > 1, то
  + Если u1, …, ui ∈ Integer ∪ Double ∪ String, uj-1 ≤ uj для 1 < j < i < n, и неверно, что ui-1 ≤ ui, то не вычислять xi+1, …, xn и возвратить значение false.
  + Если u1, …, ui ∈ Integer ∪ Double ∪ String, uj-1 ≤ uj для 1 < j < i < n, и ui ∉ Integer ∪ Double ∪ String, то не вычислять xi+1, …, xn и возвратить джамп типа gteJump.
  + Если u1, …, un ∈ Integer ∪ Double ∪ String, и ui-1 ≤ ui для 1 < i ≤ n, то возвратить значение true.

## mapFirst

Функция mapFirst имеет аргументы (x, y) и определяется следующим образом:

* Пусть x, y возвращают значения ux, uy.
* Если ux – не функция, или ux не допускает один аргумент, то возвратить джамп типа mapFirstJump.
* Если ux – функция, и ux допускает один аргумент, то
  + Если uy – пустая последовательность, то возвратить значение uy.
  + Если uy – непустая последовательность, то породить новую последовательность u, где элементами являются результаты применения функции ux слева направо к элементам последовательности uy, и возвратить значение u.

## mapKeyValues

Функция mapKeyValues имеет аргументы (x, y) и определяется следующим образом:

* Пусть x, y возвращают значения ux, uy.
* Если ux ∉ Function, или ux не допускает два аргумента, то возвратить джамп типа mapKeyValuesJump.
* Если ux ∈ Function, и ux допускает два аргумента, то
  + Если uy ∉ Attributon, то возвратить значение uy.
  + Если uy ∈ Attributon, то породить новый атрибутон u с теми же атрибутами, что и uy, значениями которых являются результаты применения функции ux в произвольном порядке к парам (имя атрибута, значение атрибута), и возвратить значение u.

## mapSortedKeyValues

Функция mapSortedKeyValues имеет аргументы (x, y) и определяется следующим образом:

* Пусть x, y возвращают значения ux, uy.
* Если ux ∉ Function, или ux не допускает два аргумента, то возвратить джамп типа mapSortedKeyValuesJump.
* Если ux ∈ Function, и ux допускает два аргумента, то
  + Если uy ∉ Attributon, то возвратить значение uy.
  + Если uy ∈ Attributon, то породить новый атрибутон u с теми же атрибутами, что и uy, значениями которых являются результаты применения функции ux слева направо (в возрастающем порядке имен атрибутов) к парам (имя атрибута, значение атрибута), и возвратить значение u.

## mod

Функция mod имеет аргументы (x, y) и определяется следующим образом:

* Пусть x, y возвращают значения ux, uy.
* Если ux ∉ Integer, то не вычислять y и возвратить джамп типа modJump.
* Если ux ∈ Integer, то
  + Если uy ∉ Integer, то возвратить джамп типа modJump.
  + Если uy ∈ Integer, то
    - Если uy ≠ 0, то возвратить значение, которое является остатком от деления ux на uy.
    - Если uy = 0, то возвратить джамп типа modJump.

## mul

Функция mul имеет аргументы (x1, …, xn) и определяется следующим образом:

* Пусть x1, …, xn возвращают значения u1, …, un.
* Если n = 0, то возвратить значение 1.
* Если n > 0, то
  + Если u1, …, un ∈ Integer ∪ Double, то возвратить значение, которое является произведением чисел u1, …, un.
  + Если u1, …, ui-1 ∈ Integer ∪ Double, и ui ∉ Integer ∪ Double, то не вычислять xi+1, …, xn и возвратить джамп типа mulJump.

## neq

Функция neq имеет аргументы (x, y) и определяется следующим образом:

* Пусть x, y возвращают значения ux, uy.
* Если ux ≠ uy, то возвратить значение true.
* Если ux = uy, то возвратить значение false.

## nop

Функция nop имеет аргументы (x1, …, xn).

Пока не нужно реализовывать.

## not

Функция not имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Если u = type, то возвратить значение false.
* Если u = false, то возвратить значение true.
* Если u ∉ Boolean, то возвратить джамп типа notJump.

## or

Функция or имеет аргументы (x1, …, xn) и определяется следующим образом:

* Пусть x1, …, xn возвращают значения u1, …, un.
* Если n = 0, то возвратить значение false.
* Если n > 0, то
  + Если u1, …, un ∈ {false}, то возвратить значение false.
  + Если u1, …, ui-1 ∈ {false}, и ui = type, то не вычислять xi+1, …, xn и возвратить значение true.
  + Если u1, …, ui-1 ∈ {false}, и ui ∉ Boolean, то не вычислять xi+1, …, xn и возвратить джамп типа orJump.

## print

Функция print имеет аргументы (x1, …, xn) и определяется следующим образом:

* Пусть x1, …, xn возвращают значения u1, …, un.
* Если v1, …, vn – строковые представления элементов u1, …, un, то выдать на печать строку v1, …, vn.

## println

Функция println имеет аргументы (x1, …, xn) и определяется следующим образом:

* Пусть x1, …, xn возвращают значения u1, …, un.
* Если v1, …, vn – строковые представления элементов u1, …, un, то выдать на печать строку v1, …, vn и выполнить перевод строки.

## progn

Функция progn имеет аргументы (x1, …, xn) и определяется следующим образом:

* Пусть x1, …, xn возвращают значения u1, …, un.
* Если i > 0, xi-1 возвращает джамп u, xi – не catch, то возвратить джамп u.
* Если xn не возвращает джамп, то возвратить значение un.
* Если xn возвращает джамп u, то возвратить джамп u.

## qname

Функция qname имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Если u ∉ QName ∪ String, то возвратить джамп типа qnameJump.
* Если u ∈ QName, то возвратить значение u.
* Если u ∈ String, и для u порождалось квалифицированное имя v, то возвратить значение v.
* Если u ∈ String, и для u не порождалось квалифицированное имя, то породить новое квалифицированное имя для u и возвратить значение, которое является этим квалифицированным именем.

## quote

Функция quote имеет аргументы (quote x) и определяется следующим образом:

* Возвратить значение x.

## read

Функция read имеет аргументы (x1, …, xn) и определяется следующим образом:

* Пусть x1, …, xn возвращают значения u1, …, un.
* Если i = 0, то возвратить текущее состояние.
* Если i > 0, то
  + u1, …, ui-1 ∈ String, и ui ∉ String, то возвратить джамп типа readJump.
  + Если u1, …, un ∈ String, то преобразовать u1, …, un в онтологические модели m1, …, mn и возвратить значение mn.

## return

Функция return имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Вычислить jump(conz(returnJump, true, value, quote(u))).

## seqApply

Функция seqApply имеет аргументы (x, y) и определяется следующим образом:

* Пусть x, y возвращают значения ux, uy.
* Если ux – не функция, то возвратить джамп типа seqApplyJump.
* Если ux – функция, и uy – последовательность длины n, то
  + Если ux допускает последовательность аргументов длины n, то применить ux к последовательности аргументов, соответствующих элементам последовательности uy.
  + Если ux не допускает последовательность аргументов длины n, то возвратить джамп типа seqApplyJump.

## seqCons

Функция seqCons имеет аргументы (x, y) и определяется следующим образом:

* Пусть x, y возвращают значения ux, uy.
* Если x возвращает джамп u, то не вычислять y и возвратить джамп u.
* Если y возвращает джамп u, то возвратить джамп u.
* Если uy – не последовательность, то возвратить джамп типа sequenceJump.
* Если uy = {seqLen = n, start = i, i+0 = u1, …, i+n-1 = un, w}, то заменить содержимое атрибутона uy на {seqLen = n+1, start = i, i+0 = ux, i+1 = u1, …, i+n = un, w} и возвратить значение uy.

## seqCopy

Функция seqCopy имеет аргументы (x) и эквивалентна copyShallow(x).

## seqCreate

Функция seqCreate имеет аргументы (x1, …, xn) и определяется следующим образом:

* Пусть x1, …, xn возвращают значения u1, …, un.
* Породить новую последовательность w из элементов u1, …, un и возвратить значение w.

## seqCreateExtra

Функция seqCreateExtra имеет аргументы (z, x1, …, xn) и определяется следующим образом:

* Пусть x1, …, xn возвращают значения u1, …, un.
* Добавить элементы u1, …, un в конец последовательности z.

## seqFirst

Функция seqFirst имеет аргументы (x) и определяется следующим образом:

* Если x – не последовательность, то возвратить джамп типа seqFirstJump.
* В противном случае,
  + Если aref(x, start) возвращает значение u ∉ Integer ∪ Undef, то возвратить джамп типа seqFirstJump.
  + Если aref(x, start) возвращает undef, то вычислить aref(x, 1).
  + В противном случае, вычислить aref(x, aref(x, start)).

## seqLength

Функция seqLength имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Если u – последовательность, то возвратить значение, которое является длиной последовательности u.
* В противном случае, возвратить джамп типа seqLengthJump.

## seqPrepend

Функция seqPrepend имеет аргументы (x1, …, xn, x) и определяется следующим образом:

* Пусть x1, …, xn, x возвращают значения u1, …, un, u.
* Если w1 – первый элемент последовательности (u1, …, un, u), и w1 возвращает джамп w2, то не вычислять остальные элементы этой последовательности и возвратить джамп w2.
* Если u – не последовательность, то возвратить джамп типа seqPrependJump.
* В противном случае, если u = {seqLen = m, start = i, i+0 = v1, …, i+m-1 = vm, w}, то заменить содержимое u на {seqLen = m+n, start = i, i+0 = u1, …, i+n-1 = un, i+n = v1, …, i+n+m-1 = vm, w} и возвратить значение u.

## seqRest

Функция seqRest имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Если u – не последовательность, то возвратить джамп типа seqRestJump.
* Если u – пустая последовательность, то возвратить джамп типа seqRestJump.
* В противном случае, если u = {seqLen = n, start = i, i+0 = u1, …, i+n-1 = un, w}, то заменить содержимое u на {seqLen = n-1, start = i, i+0 = u2, …, i+n-2 = un, w} и возвратить u.

## seqReverse

Функция seqReverse имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Если x возвращает джамп w, то возвратить джамп w.
* Если u – не последовательность, то возвратить джамп типа seqReverseJump.
* В противном случае, если u = {seqLen= n, start = i, i+0 = u1, …, i+n-1 = un, w}, то заменить содержимое u на {seqLen= n, start = i, i+0 = un, …, i+n-1 = u1, w} и возвратить значение u.

## setq

Функция setq имеет аргументы (x, y) и определяется следующим образом:

* Пусть y возвращает значение u.
* Если av(x, Variable) = type, то
  + Присвоить атрибуту x атрибутона av(lc, variables) значение u.
  + Возвратить значение u.
* Если av(x, AttributonVariable) = type, то
  + Если u ∉ Atom, то
    - Присвоить атрибуту x атрибутона av(lc, attributonVariables) значение u.
    - Возвратить значение u.
  + В противном случае, возвратить джамп типа setqJump.
* В противном случае, не вычислять y и возвратить джамп типа setqJump.

## stop

Функция stop имеет аргументы () и определяется следующим образом:

* Пусть av – функция значения атрибутов текущего состояния.
* Возвратить состояние (stop, undef, av).

## strConcat

Функция strConcat имеет аргументы (x1, …, xn) и определяется следующим образом:

* Пусть x1, …, xn возвращают значения u1, …, un.
* Пусть w1, …, wn – результаты конвертации u1, …, un в строки.
* Если n = 0, то возвратить значение undef.
* Если n > 0, то
  + Если 0 < i ≤ n, u1, …, ui-1 могут быть конвертированы в строки, элемент ui не может быть конвертирован в строку, то возвратить джамп типа strConcatJump.
  + Если u1, …, un могут быть конвертированы в строки, то породить новую строку w, которая является конкатенацией строк w1, …, wn и возвратить значение w.

## strToFile

Функция strToFile имеет аргументы (…).

Пока не нужно реализовывать.

## sub

Функция sub имеет аргументы (x, y) и определяется следующим образом:

* Пусть x, y возвращают значения ux, uy.
* Если ux, uy ∈ Integer ∪ Double, то возвратить значение, которое является разностью w чисел ux и uy.
  + Если ux, uy ∈ Integer, то w ∈ Integer.
  + Если ux ∈ Double, или uy ∈ Double, то w ∈ Double.
* Если ux ∉ Integer ∪ Double, то не вычислять y и возвратить джамп типа subJump.
* Если ux ∈ Integer ∪ Double, и uy ∉ Integer ∪ Double, то возвратить джамп типа subJump.

## toBoolean

Функция toBoolean имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Пусть w – результат приведения u к Boolean.
  + Если u ∈ Boolean, то w = u.
  + Если u ∈ Integer ∪ Double, то
    - Если u = 0, то w = false.
    - Если u ≠ 0, то w = type.
  + Если u ∉ Boolean ∪ Integer ∪ Double, то u не приводится к Boolean.
* Если u приводится к Boolean, то возвратить значение w.
* Если u не приводится к Boolean, то возвратить джамп типа toBooleanJump.

## toDouble

Функция toDouble имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Пусть w – результат приведения u к Double.
  + Если u ∈ Double, то w = u.
  + Если u ∈ Integer, то u приводится к Double без потери точности.
  + Если u ∉ Integer ∪ Double, то u не приводится к Boolean.
* Если u приводится к Double, то возвратить значение w.
* Если u не приводится к Double, то возвратить джамп типа toDoubleJump.

## toInteger

Функция toInteger имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Пусть w – результат приведения u к Integer.
  + Если u ∈ Integer, то w = u.
  + Если u ∈ Double, то
    - Если u приводится к Integer без округления, то w – результат этого приведения.
    - Если u не приводится к Integer без округления, то u не приводится к Integer.
  + Если u ∉ Integer ∪ Double, то u не приводится к Integer.
* Если u приводится к Integer, то возвратить значение w.
* Если u не приводится к Integer, то возвратить джамп типа toIntegerJump.

## toString

Функция toString имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Пусть w – результат приведения u к String.
* Если u приводится к String, то возвратить значение w.
* Если u не приводится к String, то возвратить джамп типа toStringJump.

## uminus

Функция uminus имеет аргументы (x) и определяется следующим образом:

* Пусть x возвращает значение u.
* Если u ∈ Integer ∪ Double, то возвратить значение, которое является отрицанием числа u.
* Если u ∉ Integer ∪ Double, то возвратить джамп типа uminusJump.

## variChain

Функция variChain имеет аргументы (x1, …, xn) и определяется следующим образом:

* Пусть lcc0 – копия текущего контекста.
* Пусть x1, …, xn возвращают локальные контексты lc1, …, lcn.
* Пусть lcc1, …, lcn-1 – копии локальных контекстов lc1, …, lcn-1, вычисляемые перед выполнением x2, …, xn.
* Возвратить состояние, которое получается из текущего состояния заменой последовательности локальных контекстов на последовательность с элементами lcc0, lcc1, …, lccn-1, lcn.

## variants

Функция variants имеет аргументы (x1, …, xn) и определяется следующим образом:

* Пусть lcc1, …, lccn – копии текущего локального контекста.
* Пусть x1, …, xn вычисляются в локальных контекстах lcc1, …, lccn в порядке слева направо и возвращают последовательности локальных контекстов lcs1, …, lcsn.
* Возвратить состояние, которое получается из текущего состояния заменой последовательности локальных контекстов на конкатенацию последовательностей lcs1, …, lcsn.

## variantsForKeyValues

Функция variantsForKeyValues имеет аргументы (x, y, z) и определяется следующим образом:

* Пусть z возвращает значение u.
* Пусть {a1, …, an} – множество атрибутов элемента u.
* Пусть lc1, …, lcn – копии текущего контекста lc, расширенные переменными x и y со значениями av(lc, a1), …, av(lc, an).
* Если x, y – переменные, то
  + Если n = 0, то возвратить lc.
  + Если n > 0, то возвратить состояние, полученное из текущего состояния заменой lc на последовательность из локальных контекстов lc1, …, lcn, входящих в нее в произвольном порядке.
* Если x или y – не переменная, то не вычислять z и возвратить джамп типа variantsForKeyValuesJump.

## while

Функция while имеет аргументы (x, y) и определяется следующим образом:

* 1: Вычислить x.
* Пусть x возвращает значение ux.
* Если ux = type, то вычислить u и перейти к шагу 1.
* Если ux = false, то закончить вычисление.
* Если ux ∉ Boolean, то возвратить джамп типа whileJump.

# Встроенные функции по категориям

|  |  |
| --- | --- |
| **Категория** | **Функции** |
| Атомы (тип Atom) | isAtom |
| Атрибутоны (тип Attributon) | aref, aset, conz, isAttributon, mapKeyValues, mapSortedKeyValues |
| Булевские значения (тип Boolean) | and, isBoolean, not, or, toBoolean |
| Выражения | assert, assertEq, catch, cond, eval, jump, nop, progn, quote, read, while |
| Глобальные контексты | globaref, globaset, sysFunctions |
| Квалифицированные имена (тип QName) | qname, isQName |
| Консоль | print, println |
| Локальные контексты | locaref, locaset |
| Мультисостояния | stop, variChain, variants, variantsForKeyValues |
| Неопределенные значения | coalesce, isDef, isUndef, undef |
| Переменные | setq |
| Последовательности | copySeq, eqSeq, mapFirst, seqApply, seqCons, seqCopy, seqCreate, seqFirst, seqLength, seqPrepend, seqRest, seqReverse |
| Состояния | copyState, getState, stop |
| Строки (тип String) | gt, gte, isString, lt, lte, read, strConcat, toString |
| Файлы | fileToString, HLSRLLoad, load, loadData, strToFile |
| Функции | defun, evalArg, functions, return |
| Числа | add, div, gt, gte, isNumeric, lt, lte, mul, sub, uminus |
| Числа (тип Double) | isDouble, toDouble |
| Числа (тип Integer) | isInteger, mod, toInteger |
| Элементы типа Thing | copyDeep, copyShallow, eq, eqDeep, eqShallow, neq, read |