**Язык описания атрибутных структур ASL**

# Теория атрибутных систем

## Атрибутная система

Атрибутная система – это вид систем переходов.

Атрибутная система – это кортеж (TG, E, S, →) такой, что

* TG = AN ∪ AM ∪ U – объединение трех непересекающихся множеств. Элементы TG называются вещами (things).
  + AN – множество атрибутонов (attributons),
  + AM – множество атомов (atoms),
  + UN – множество неопределенных значений (undefined values).
* E – множество выражений (expressions) такое, что stop ∈ E. Выражение stop называется выражением останова.
* S – множество состояний (states).
  + Состояние s ∈ S – это кортеж (e, lcs, gc, av) такой, что
    - e ∈ E – вычисляемое выражение.
    - lcs = (lc1, …, lcn), где n ≥ 0, и eci ∈ AN для 1 ≤ i ≤ n - последовательность локальных контекстов (local contexts), в которых вычисляется выражение e.
    - gc ∈ AN – глобальный контекст (global context), в котором вычисляется выражение e.
    - av – тотальная функция (attribute value) из TG × TG в TG такая, что
      * Если tg1 ∈ AM, и tg2 ∈ TG, то av(tg1, tg2) ∈ UN.
      * Если tg1 ∈ TG, и tg2 ∈ Undef, то av(tg1, tg2) ∈ UN.
* → ∈ S × S – бинарное отношение на состояниях (отношение перехода, transition relation) такое, что
  + Если n > 1, и (e, (lci), gci, avi) →\* (stop, lcsi, gci+1, avi+1) для 1 ≤ i ≤ n, то (e, (lc1, …, lcn), gc1, av1) → (stop, ecs1 + … + ecsn, gcn+1, avn+1). Здесь + обозначает конкатенацию последовательностей. Здесь →\* - транзитивное замыкание отношения → (определено ниже в п. 1.3).
  + Если e ≠ stop, то (e, (), gc, av) → (stop, (), gc, av).
  + Если e ≠ stop, то для любых lc, gc и av существует s такое, что (e, (lc), gc, av) → s.

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

## Атрибуты

Вещь tg1 имеет атрибут tg2 в состоянии s, если av(tg1, tg2) ∉ UN. Вещь tg1 не имеет атрибута tg2 в состоянии s, если av(tg1, tg2) ∈ UN.

Пусть вещь tg1 имеет атрибут tg2 в состоянии s. Атрибут tg2 вещи tg1 имеет значение tg3 в состоянии s, если av(tg1, tg2) = tg3.

Из определения состояния следуют два свойства:

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

## Состояния

Будем обозначать (s1, s2) ∈ → как s1 → s2. Состояние s2 называется финальным, если не существует состояния s1 такого, что s1 → s2.

Согласно определению атрибутной системы, если s – финальное состояние, то s = (stop, lcs, gc, av) для некоторых lcs, gc и av.

Отношение →\* ∈ S × S называется транзитивным замыканием отношения →, если (s0, sn) ∈ →\* тогда и только тогда, когда существуют s1, …, sn-1 такие, что s0 → s1, s1 → s2, …, sn-1 → sn, и sn – финальное состояние. Будем обозначать (s1, s2) ∈ →\* как s1 →\* s2.

Выражение e возвращает состояние s1 в состоянии s (обозначается (e, s) ↦s s1), если s →\* s1.

## Локальные контексты

Вычисление выражений порождает новые локальные контексты вычислений.

Выражение e возвращает локальный контекст lc1 в состоянии s (обозначается (e, s) ↦lc lc1), если s = (e, lcs, av) →\* (stop, lcs1, av1), и lc1 ∈ lcs1.

Выражение e возвращает состояние s1 с локальным контекстом lc1 в состоянии s (обозначается (e, s) ↦lc,st (lc1, s1), если s = (e, lcs, gc, av) →\* s1 = (stop, lcs1, gc1, av1), и lc1 ∈ lcs1.

## Джампы

Вычисление выражений может возвращать джампы – специальные значения, описывающие ситуации, требующие специальной обработки. Джампы определяются специальным атрибутом jump.

Джамп tg имеет тип tg1 в состоянии s, если av(tg, tg1) = true.

Пусть tg ∉ UN.

Выражение e возвращает джамп tg в состоянии s (обозначается (e, s) ↦jump tg), если s = (e, lcs, gc, av) →\* (stop, lcs1, gc1, av1), и существует lc1 ∈ lcs1 такое, что av1(lc1, jump) = tg.

Выражение e возвращает джамп в состоянии s (обозначается (e, s) ↦jump\* ()), если s = (e, lcs, gc, av) →\* (stop, lcs1, gc1, av1), и существует lc1 ∈ lcs1 такое, что av1(lc1, jump) ∉ UN.

Выражение e возвращает состояние s1 с джампом tg в состоянии s (обозначается (e, s) ↦jump,st (tg, s1)), если s = (e, lcs, gc, av) →\* s1 = (stop, lcs1, gc1, av1), и существует lc1 ∈ lcs1 такое, что av1(lc1, jump) = tg.

Выражение e возвращает состояние s1 с джампом в состоянии s (обозначается (e, s) ↦jump\*,st s1), если s = (e, lcs, gc, av) →\* s1 = (stop, lcs1, gc1, av1), и существует lc1 ∈ lcs1 такое, что av1(lc1, jump) ∉ UN.

Выражение e возвращает состояние s1 с локальным контекстом lc1 с джампом tg в состоянии s (обозначается (e, s) ↦jump,lc,st (tg, lc1, s1)), если s = (e, lcs, gc, av) →\* s1 = (stop, lcs1, gc1, av1), lc1 ∈ lcs1, и av1(lc1, jump) = tg.

Выражение e возвращает состояние s1 с локальным контекстом lc1 с джампом в состоянии s (обозначается (e, s) ↦jump\*,lc,st (lc1, s1)), если s = (e, lcs, gc, av) →\* s1 = (stop, lcs1, gc1, av1), lc1 ∈ lcs1, и av1(lc1, jump) ∉ UN.

Выражение e возвращает локальный контекст lc1 с джампом tg в состоянии s (обозначается (e, s) ↦jump,lc (tg, lc1)), если s = (e, lcs, gc, av) →\* (stop, lcs1, gc1, av1), lc1 ∈ lcs1, и av1(lc1, jump) = tg.

Выражение e возвращает состояние s1 с локальным контекстом lc1 с джампом в состоянии s (обозначается (e, s) ↦jump\*,lc lc1), если s = (e, lcs, gc, av) →\* (stop, lcs1, gc1, av1), lc1 ∈ lcs1, и av1(lc1, jump) ∉ UN.

## Специализация отношения ↦ для локального контекста

Пусть (e, s, lc1) ↦… … означает (e, s1) ↦… …, где s1 получается из s заменой последовательности локальных контекстов на последовательность из одного локального контекста lc1.

Например, (e, s, lc1) ↦jump\*,lc lc2 означает (e, s1) ↦jump\*,lc lc2.

## Значения выражений

Вычисление выражений может возвращать значения. Значения выражений определяются специальным атрибутом value.

Пусть tg ∈ Thing.

Выражение e возвращает значение tg в состоянии s (обозначается (e, s) ↦value tg), если s = (e, lcs, gc, av) →\* (stop, lcs1, gc1, av1), и существует lc1 ∈ lcs1 такое, что av1(lc1, jump) ∈ UN, и av1(lc1, value) = tg.

Выражение e возвращает значение в состоянии s (обозначается (e, s) ↦value\* ()), если s = (e, lcs, gc, av) →\* (stop, lcs1, gc­1, av1), и существует lc1 ∈ lcs1 такое, что av1(lc1, jump) ∈ UN.

Выражение e возвращает состояние s1 со значением tg в состоянии s (обозначается (e, s) ↦value,st (tg, s1)), если s = (e, lcs, gc, av) →\* s1 = (stop, lcs1, gc1, av1), и существует lc1 ∈ lcs1 такое, что av1(lc1, jump) ∈ UN, и av1(lc1, value) = tg.

Выражение e возвращает состояние s1 со значением в состоянии s (обозначается (e, s) ↦value\*,st s1), если s = (e, lcs, gc, av) →\* s1 = (stop, lcs1, gc1, av1), и существует lc1 ∈ lcs1 такое, что av1(lc1, jump) ∈ UN.

Выражение e возвращает состояние s1 с локальным контекстом lc1 со значением tg в состоянии s (обозначается (e, s) ↦value,lc,st (tg, lc1, s1)), если s = (e, lcs, gc, av) →\* s1 = (stop, lcs1, gc1, av1), lc1 ∈ lcs1, и av1(lc1, jump) ∈ UN, и av1(lc1, value) = tg.

Выражение e возвращает состояние s1 с локальным контекстом lc1 со значением в состоянии s (обозначается (e, s) ↦value\*,lc,st (lc1, s1)), если s = (e, lcs, gc, av) →\* s1 = (stop, lcs1, gc1, av1), lc1 ∈ lcs1, и av1(lc1, jump) ∈ UN.

Выражение e возвращает локальный контекст lc1 со значением tg в состоянии s (обозначается (e, s) ↦value,lc (tg, lc1)), если s = (e, lcs, gc, av) →\* (stop, lcs1, gc1, av1), lc1 ∈ lcs1, и av1(lc1, jump) ∈ UN, и av1(lc1, value) = tg.

Выражение e возвращает локальный контекст lc1 со значением в состоянии s (обозначается (e, s) ↦value\*,lc,st lc1), если s = (e, lcs, gc, av) →\* (stop, lcs1, gc1, av1), lc1 ∈ lcs1, и av1(lc1, jump) ∈ UN.

## Атрибутные типы

Любую вещь tg можно рассматриваться как атрибутный тип. Множество элементов атрибутного типа tg определяется через атрибут tg следующим образом: tg1 ∈ tg тогда и только тогда, когда av(tg1, tg) = true. Таким образом, тип tg включает все атрибутоны, у которых атрибут tg определен и равен true.

# Язык ASL

ASL – язык описания атрибутных систем (Attribute Systems Language). 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

Множество UN атрибутной системы определяется в языке ASL подтипом Undef типа Thing. Этот тип включает выделенный элемент undef.

## Тип Attributon

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

### Подтипы типа Attributon

Язык ASL включает следующие подтипы типа Attributon.

#### Тип Sequence

Элементы подтипа Sequence типа Attributon называются последовательностями. Они определяют конечные последовательности вещей и характеризуются атрибутами first и rest. Для элемента tg ∈ Sequence выполнены следующие свойства в любом состоянии s:

* av(tg, rest) ∉ Atom для любого состояния.
* Если av(tg, rest) – последовательность длины n > 0, то tg – последовательность длины n+1.
* Если av(tg, rest) = undef, то tg – последовательность длины 1.
* Если tg – последовательность длины n, то av(tg, first) – первый элемент tg, av(tg, rest, first) – второй элемент tg, av(g, rest, rest, first) – третий элемент tg и т. д.
* Если tg – последовательность длины n, то вещи tg, av(tg, rest), av(tg, rest, rest) и т. д. называются местами элементов av(tg, first), av(tg, rest, first), av(tg, rest, rest, first) и т. д.
* Все места последовательности tg попарно различны.

## 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, если av(s, variables, u) ≠ undef. В этом случае, говорят, что переменная $u имеет значение av(s, variables, 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 = av(lc2, parent).

## Глобальный контекст

Глобальный контекст gc имеет следующие специальные атрибуты:

* Атрибут attributons содержит информацию об атрибутонах, определенных в текущем состоянии. Если av(gc, attributons, tg) ∉ Undef в s, то атрибутон tg является определенным в s.
* Атрибут functions содержит информацию о пользовательских функциях. Значения av(gc, functions, tg, n, parameters) и av(gc, functions, tg, n, body) в текущем состоянии s соответствуют параметрам и телу функции с именем tg ∈ QName местности n, соответственно.

## 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 tg ∈ QName, e1, …, en – синтаксические выражения, то tg(e1, …, en) – синтаксическое выражение (вызов функции с именем tg).

### 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 определен в в состоянии s и имеет в этом состоянии атрибуты a1, …, an со значениями tg1, … tgn. Пусть (tg1, …, tgn) обозначает последовательность с элементами tg1, …, tgn в состоянии s. 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 = true, name = u};
* Если e – атрибутонная переменная с именем u, то e →ont {attributonVariable = true, name = u};
* Если e →ont tg, то (e) →ont tg;
* Если e1 →ont tg1,…, en →ont tgn, то tg(e1, …, en) →ont {functionCall = true, function = tg, arguments = (tg1, …, 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).

Пусть {% a1 = tg1, …, an = tgn %} обозначает порождение нового атрибутона и добавление к нему атрибутов a1, …, an со значениями tg1, …, tgn. Атрибутон является новым в состоянии s, если он не определен в состоянии s, т. е. av(gc, attributons, tg) ∈ Undef. Порождение означает присваивание значения true атрибуту tg атрибутона av(gc, attributons).

Пусть s = (e, (lc), av) – текущее состояние.

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

1. Случай переменных
   * Если e ∈ Attributon, и av(e, variable) = true, то
     + Если av(e, name) = tg ∈ QName, то s → (stop, (lc), gc, upd(av, value, av(lc, variables, tg)));
     + В противном случае, s → (throw({% variableJump = true %}), (lc), av);
2. Случай атрибутонных переменных
   * Если e ∈ Attributon, и av(e, attributonVariable) = true, то
     + Если av(e, name) = tg ∈ QName, то
       - Если av(lc, attributonVariables, tg) ∉ Undef, то s → (stop, (lc), gc, upd(av, value, av(lc, attributonVariables, tg)));
       - В противном случае, если tg – новый атрибутон, то s → (stop, (lc), gc, upd(upd(av, gc, attributons, tg, true), value, av(lc, attributonVariables, tg)));
     + В противном случае, s → (throw({% attributonVariableJump = true %}), (lc), gc, av).
3. Случай вызова функции
   * Если e ∈ Attributon, и av(e, functionCall) = true, то
     + Если av(e, function) = tg1 ∈ QName, av(e, arguments) = tg ∈ Sequence ∪ Undef, и выполнено одно из двух свойств: (tg1, (lc), av) →pf(g) s1, или (tg1, (lc), gc, av) →df(tg) s1, то (e, (lc), gc, av) → s1.
     + В противном случае, (e, (lc), gc, av) → (throw({% functionCallJump = true %}), (lc), gc, av);
4. В противном случае, s → (stop, (lc), gc, upd(av, lc, value, e)).

Отношения перехода →pf(tg) и →df(tg) определяют вызовы предопределенных и определяемых пользователем функций, соответственно, на последовательности аргументов tg и именем, хранящемся на месте выражения в текущем состоянии.

### Отношение перехода для вызова функции для нескольких контекстов

Отношения →pf(tg) и →df(tg) для нескольких контекстов определяются также как отношение →:

Если n > 1, и (e, (lci), gci, avi) → pf(tg) (stop, lcsi, gci+1, avi+1) для 1 ≤ i ≤ n, то (e, (lc1, …, lcn), gc1, av1) → fv(tg) (stop, ecs1 + … + ecsn, gcn+1, avn+1).

Если n > 1, и (e, (lci), gci, avi) → df(tg) (stop, lcsi, gci+1, avi+1) для 1 ≤ i ≤ n, то (e, (lc1, …, lcn), gc1, av1) → fv(tg) (stop, ecs1 + … + ecsn, gcn+1, avn+1).

### Отношение перехода для вызова предопределенной функции для одного контекста

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

### Отношение перехода для вызова пользовательской функции для одного контекста

Пусть s = (f, (lc), gc, av), и последовательность аргументов tg состоит из элементов e1, …, en. Отношение →df(tg) определяется следующим образом:

* Если w = av(gc, functions, f, n) ∈ Attributon, последовательность av(w, parameters) состоит из элементов p1, …, pn, и av(w, body) ≠ undef, то

# Породить копию lc1 контекста lc.

# Присвоить атрибуту variables атрибутона lc1 новый атрибутон вида {p1 = e1, …, pn = en} в качестве значения.

# Присвоить атрибуту parent контекста lc1 значение lc.

# Сделать lc1 текущим локальным контекстом.

# Если progn(av(f, body)) возвращает джамп u типа, отличного от returnJump, то атрибуту jump атрибутона lc присвоить значение u, и сделать lc текущим локальным контекстом.

# Если progn(av(f, body)) возвращает джамп типа returnJump со значением u, то атрибуту value атрибутона lc присвоить значение u и сделать lc текущим локальным контекстом.

# Если progn(av(f, body)) не возвращает джамп, то атрибуту jump атрибутона lc присвоить джамп типа functionCallJump, и сделать lc текущим локальным контекстом.

* В противном случае, s →df(tg) (throw({% functionCallJump = true %}), (lc), gc, av).

# 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 (x1) then y1 else if x2 then y2 … if xn then yn else z | cond(x1, y1, x2, y2, …, xn, yn, z) |
| if (x1) then y1 else if x2 then y2 … if xn then yn | cond(x1, y1, x2, y2, …, xn, yn) |
| {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 | 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) |
| x1 && x2 && … && xn, где n>0 | and(x1, x2, …, xn) |
| x1 || x2 || … || xn, где n>0 | or(x1, x2, …, xn) |
| x = y | setq(x, y) |
| x1. … .xn, где n>0 (конструкция .xi может заменяться на [xi]) | aref(x1, …, xn) |
| x1. … .xn = y, где n>1 (конструкция .xi может заменяться на [xi]) | aset(x1, …, xn, y) |
| .x.x1. … .xn, где n ≥ 0 (конструкция .xi может заменяться на [xi]) | globaref(x, x1, …, xn) |
| .x.x1. … .xn = y, где n ≥ 0 (конструкция .xi может заменяться на [xi]) | globaset(x, x1, …, xn, y) |
| ..x.x1. … .xn, где n ≥ 0 (конструкция .xi может заменяться на [xi]) | stateref(x, x1, …, xn) |
| ..x.x1. … .xn = y, где n ≥ 0 (конструкция .xi может заменяться на [xi]) | stateset(x, x1, …, xn, 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.

# 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 = true, 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 = true, 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 типа Undef, то удалить атрибут uxi атрибутона u.
* Если yi не возвращает значение типа 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 типа Undef, то удалить атрибут uxi атрибутона z.
  + Если yi не возвращает значение типа 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 определяется следующим образом:

* Пусть x возвращает значение u.
* Если u ∉ Attributon, то возвратить значение u.
* Если u – последовательность длины n > 0 с местами u1, …, un, то породить новую последовательность длины n с новыми местами v1, …, vn, которые имеют те же самые атрибуты с теми же самыми значениями.

## 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.

## 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 u ∈ Undef, то возвратить значение true.
* Если x возвращает значение u ∈ 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 u ∈ Undef, то возвратить значение true.
* Если x не возвращает значение u ∉ 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.

## 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 = true, 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 = true, 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:

* Вычислить conz(first, x, rest, y).

## seqCopy

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

* Пусть x возвращает значение u.
* Если u ∈ Atom, то возвратить значение u.
* Если u ∈ Attributon, то породить новую последовательность w элементами которой являются соответствующие элементы последовательности u и возвратить значение w.

## 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:

* Вычислить aref(x, first).

## seqLength

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

* Пусть x возвращает значение u.
* Возвратить значение, которое является длиной последовательности u.

## seqPrepend

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

* Пусть x1, …, xn, x возвращают значения u1, …, un, u.
* Породить новую последовательность w, которая является результатом добавления в начало последовательности u элементов u1, …, un, и возвратить значение w.

## seqRest

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

* Пусть x возвращает значение u.
* Если последовательность u имеет длину 0 или 1, то возвратить джамп типа seqRestJump.
* Если последовательность u имеет длину, большую чем 1, то возвратить значение av(u, rest).

## seqReverse

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

* Пусть x возвращает значение u.
* Породить последовательность w, которая состоит из элементов последовательности u, расположенных в обратном порядке, и возвратить значение w.

## setq

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

* Пусть y возвращает значение u.
* Если av(x, Variable) = true, то
  + Присвоить атрибуту x атрибутона av(lc, variables) значение u.
  + Возвратить значение u.
* Если av(x, AttributonVariable) = true, 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 = true.
  + 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 = true, 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 атрибутоны | globaref, globaset, sysFunctions |
| Квалифицированные имена (тип QName) | qname, isQName |
| Консоль | print, println |
| 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, 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 |