

Codegen #2

Горбань Игорь

27 марта 2018

g.i.b@list.ru

Variables on stack

На прошлой лекции обсуждали возможность сохранения контекста на стеке. В текущей модели - на стеке сохраняются временные переменные. Попробуем объединить обе модели.

Проблема - при вычислении значений выражений стек может расширяться, поэтому может возникнуть проблема с доступом к аргументам функции(\$sp - будет указывать на вершину стека, которая не соответствует последнему аргументу).

Решение - для доступа к аргументам можно использовать \$fp.

Пусть x_i - i -й аргумент, тогда $cgen(x_i) = "lw \$a0 z(\$fp)"$, $z = 4 * i$

Manage temporary registers

Проблема - compile-time можно заранее узнать, какое количество временных переменных необходимо использовать для вычисления выражения, почему бы нам не выделить для них память заранее.

Т.е. генератор кода должен зафиксировать местоположение временных переменных в AR (activation record).

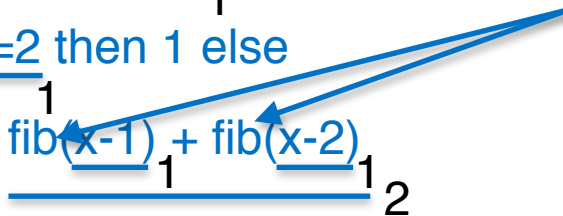
Ex: Сколько временных переменных нужно для вычисления функции fib.

```
def fib(x) = if x=1 then 0 else
```

```
  if x=2 then 1 else
```

```
    fib(x-1) + fib(x-2)
```

Вычисленные значения fib -
нужны одновременно



Number of temporaries

Давайте введем функцию $NT(e)$ - количество переменных, необходимых для выполнения e .

Тогда: $NT(e_1 + e_2) = \max \begin{cases} NT(e_1) \\ NT(e_2) + 1 \end{cases}$ - т.к. переменные, используемые в e_1 могут быть пере использованы в e_2

$$NT(e_1 + e_2) = \max(NT(e_1), 1 + NT(e_2))$$

$$NT(e_1 - e_2) = \max(NT(e_1), 1 + NT(e_2))$$

$$NT(\text{if } e_1 = e_2 \text{ then } e_3 \text{ else } e_4) = \max(NT(e_1), 1 + NT(e_2), NT(e_3), NT(e_4))$$

$$NT(id(e_1, \dots, e_n)) = \max(NT(e_1), \dots, NT(e_n))$$

$$NT(\text{int}) = 0$$

$$NT(id) = 0$$

New AR

Для определения функции $f(x_1, \dots, x_n) = e$

В AR сохраняется $2+n+NT(e)$
элементов:

- Адрес возврата
- Frame pointer
- n аргументов
- $NT(e)$ - размещение промежуточных результатов

Temp1
...
Temp NT(e)
Return addr
x1
...
xn
Old FP

Example

Генератор должен знать, как много переменных используется в каждой точке программы.

При добавлении нового аргумента при генерации кода - доступно место следующей позиции для временной переменной (в стеке).

Место для переменных используется как маленький стек фиксированного размера.

```
          cgen(e1 + e2) =
cgen(node->e1);
std::cout <<    " sw          $a0 0($sp) \n"
               " addiu        $sp $sp -4 \n";

cgen(node->e2);
std::cout <<    " lw          $t1 4($sp) \n"
               " addiu        $sp $sp 4 \n"
               " add          $a0 $t1 $a0 \n";
```

```
          cgen(e1 + e2, nt) =
cgen(node->e1, nt);
std::cout <<    " sw          $a0 nt($fp) \n";

cgen(node->e2, nt);
std::cout <<    " lw          $t1 nt($fp) \n"
               " add          $a0 $t1 $a0 \n";
```

Classes

Рассмотрим модель конструирования объекта.

Ожидания - если В унаследован от А, значит объект класса В может быть использован везде, где ожидается объект класса А.

Это значит, что код класса А должен работать без изменений для класса В.

Для генерации кода объекта класса необходимо ответить на вопросы:

- Как объекты располагаются в памяти (layout)
- Как происходят динамические вызовы

Classes example (write on a board).

Class A {

a : Int <- 0;

d : Int <- 1;

f () : Int { a <- a + d }; }

Class C inherits A {

c : Int <- 3;

h () : Int { a <- a * c }; }

Class B inherits A {

b : Int <- 2;

f () : Int { a };

g () : Int { a <- a - c }; }

Доступны в B, C

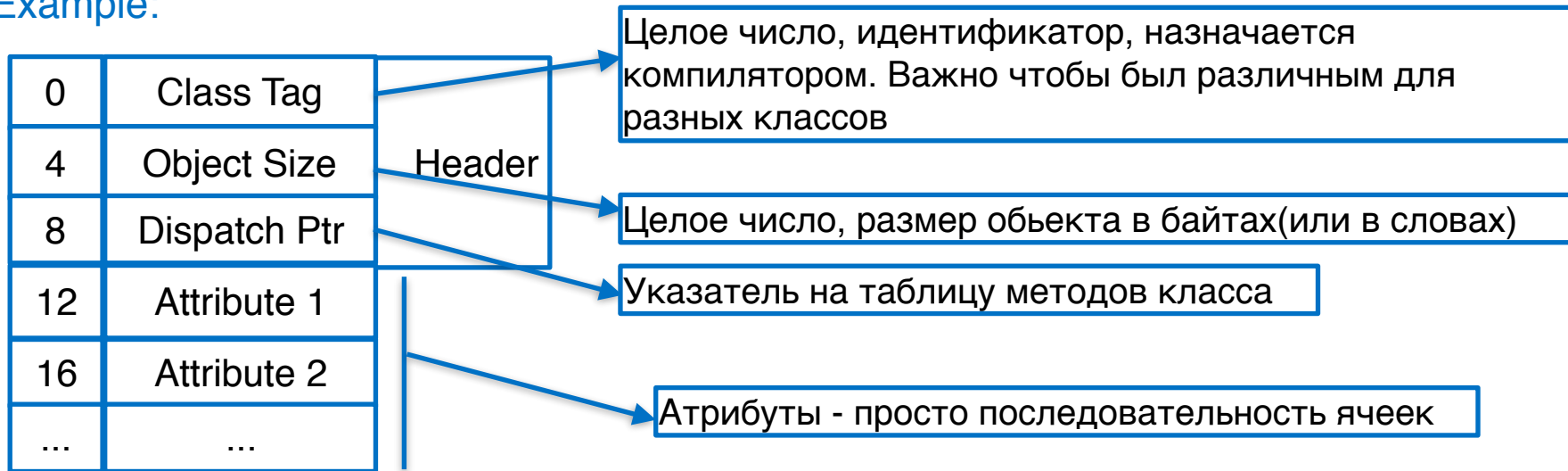
Может вызываться во всех наследниках (с точностью до переопределений). То есть атрибуты a и d должны лежать в одном и том же месте, для каждого унаследованного объекта

Class object in memory.

То есть объекты должны размещаться в смежной памяти (объект - блок памяти). Каждый атрибут расположен по фиксированному отступу в объекте.

Когда вызывается метод, объект для него - `self` и поля - это атрибуты объекта.

Example:



Class object in memory 2. Example

При реализации наследования нужно учитывать, что расположение объекта A в памяти должно полностью включаться в расположение объекта для B. Т.е. получается, что расположение объекта A - неизменно, а B - только его расширение.

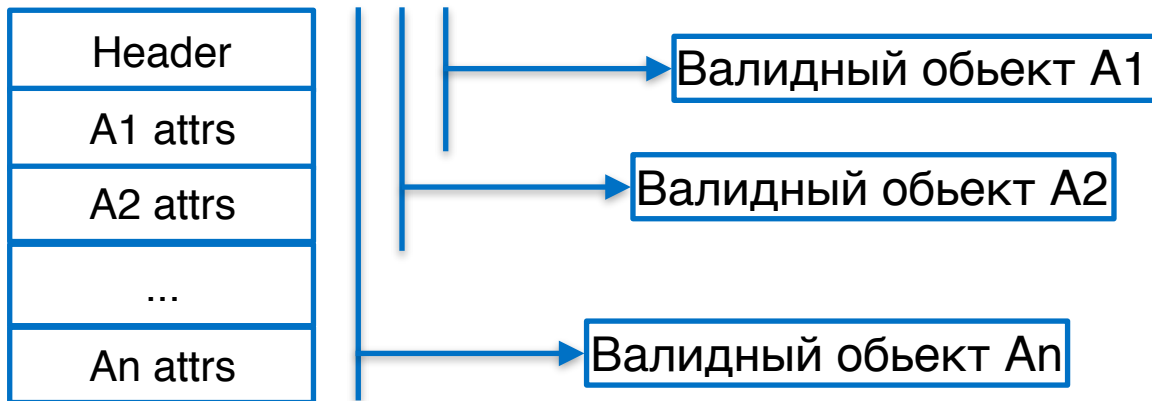
Example:

Offset Class	0	4	8	12	16	20
A	Atag	5	*	a	d	
B	Btag	6	*	a	d	b
C	Ctag	6	*	a	d	c

Class object in memory 3. Example

И еще раз: offset для атрибутов - такой же для всех наследников (каждый метод A может быть использован в подклассах).

Легко видеть, что если есть последовательность $A_n < \dots < A_3 < A_2 < A_1$, то их атрибуты будут лежать последовательно от A1 до An.



Вопрос:
Что делать, если в
языке разрешено
множественное
наследование???

Class methods in memory.

Для того же примера, пусть мы вызываем `e.f()` - в зависимости от объекта вызывается переопределенный или базовый метод.

Каждый класс имеет фиксированный набор методов (включая унаследованные).

Dispatch table (таблица методов):

- Массив точек входа в методы
- Метод `f` - находится по фиксированному оффсету для класса и всех его потомков
- Методы не могут изменяться runtime (определяются compile-time)

Class methods in memory 2: example.

Offset		
Class	0	4
A	fA	
B	fB	g
C	fA	h

Не одинаков для всех потомков, но находится на одном оффсете

Расширяют Dispatch table

Каждый метод f класса X имеет определенный оффсет (отступ) Of в Dispatch Table (который так же распространяется на наследников) во время компиляции.

Очевидно, что таблица - не создается для каждого объекта, а одна на Класс

Вопрос:
Что произойдет, если поменять таблицу для наследника на таблицу базового класса?

Class methods in memory 3.

Для реализации динамического вызова $e.f()$ необходимо:

- Вычислить выражение e , получить объект x
- Вызвать $D[Of]$
 - D - таблица методов для X (лежит в хидере x)
 - При исполнении нужно учитывать, что $self = x$

Language semantic.

Что такое семантика языка и зачем она нужна:

- Необходимо определить для каждого выражения (expression) cool-языка, что происходит, когда он выполняется (это "значение" выражения)
- Определение программного языка
 - токены - лексический анализ
 - грамматика - синтаксический анализ
 - правила вывода типов - семантический анализ
 - правила исполнения - кодген + оптимизации

Language semantic 2.

Описание правил исполнения косвено:

- компиляция `cool` в `stack-machine`
- исполнение правил `stack-machine`

Достаточно ли этого?

Это лишь один из вариантов реализации компилятора:

то, каким образом реализован кодген, куда растёт стек, как реализован AR и динамические вызовы - может косвенно влиять на язык (а так же как представлены `integer` переменные, какие инструкции используются при генерации)

То есть необходимо закончить описание, но не ограничивать его спецификацию.

Operational semantic.

Предлагаемый вариант решения (он не единственный) - это операционная семантика - описывает исполнение программы как правила исполнения на абстрактной машине (достаточно высокоуровневой).

Так же существуют альтернативы ([вики](#)):

- Денотационная семантика - программа как математическая функция
- Аксиоматическая семантика - доказываемая как теорема (что вывод соответствует функции). Так же используется в верификации программ

Operational semantics 2.

Формальные нотации и логические правила вывода, как в проверке типов семантического анализатора.

Еще раз о суждении: $Context \vdash e : C$

В текущем контексте выражение e имеет тип C .

Будем использовать подобные для исполнения: $Context \vdash e : v$

Означает, что в текущем контексте выражение e имеет значение v .

Example:

$$\frac{Context \vdash e_1 : 5 \quad Context \vdash e_2 : 7}{Context \vdash e_1 + e_2 : 12}$$

Tables

Рассмотрим $y \leftarrow x + 1$

Необходимо узнать:

- где в памяти хранится переменная (table vars \rightarrow mem)
- куда сохранить результат (table mem \rightarrow vars)

Введем отображение (map), которая:

- Отслеживает, какие переменные в scope
- Говорит, где переменные лежат в памяти

$$E = [a : l_1, b : l_2]$$

Переменная

Положение в памяти

$$S = [l_1 \rightarrow 5, l_2 \rightarrow 7]$$

Location

Значение

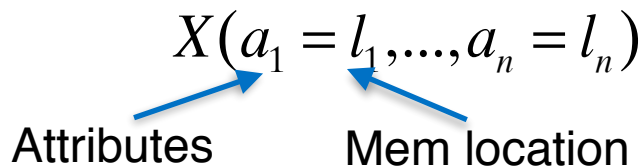
Class defines.

$S' = S[12 / l_1]$ Определяет store S' такой, что:

$S'(l_1) = 12$ и $S'(l) = S(l)$, если $l \neq l_1$

В Cool переменные - это объекты (все объекты наследуются от какого-то класса):

- X - class объекта
- a_i - атрибуты (включая унаследованные)
- l_i - расположения в памяти



Специальные классы:

- $\text{Int}(5)$
- $\text{Bool}(\text{true})$
- $\text{String}(4, \text{"Cool"})$
- void - типа Object

не может использоваться в операциях, кроме теста `isVoid`, может использоваться как `NULL`.

Context.

Рассмотрим выражение $so, E, S \vdash e : v, S'$

- so - текущее значение `self`
- E - текущие переменные окружения
- S - текущее окружение памяти (+ пустое место для записи)

Если исполнение e завершается, то значение e - это v и новое окружение памяти - S' .

Результат исполнения - это значение и адрес в S' .

Examples #1

$$\frac{}{so, E, S \vdash true : Bool(true), S} [true]$$
$$\frac{}{so, E, S \vdash false : Bool(false), S} [false]$$
$$\frac{i \text{ is an int literal}}{so, E, S \vdash i : Int(i), S} [Int]$$
$$\frac{}{so, E, S \vdash self : so, S} [self]$$

[Identifier]

$$E(id) = l_{id}$$

Поиск в окружении
(location)

$$S(l_{id}) = v$$

Значение в
памяти по адресу
lid

$$\frac{}{so, E, S \vdash id : v, S}$$

Чтение памяти

Stores не меняется

s is an string literal

$$\frac{h \text{ is the length of } s}{so, E, S \vdash s : String(n, s), S} [String]$$

Examples #2

Обновляет Store

$so, E, S \vdash e, v, S_1$

Вычисление
выражения (expression)

$E(id) = l_{id}$

Поиск в памяти id

$$\frac{S_2 = S_1[v / l_{id}]}{so, E, S \vdash id \leftarrow e : v, S_2} [Assignment]$$

$so, E, S \vdash e_1, v_1, S_1$

$so, E, S_1 \vdash e_2, v_2, S_2$

...

$$\frac{so, E, S_{n-1} \vdash e_n, v_n, S_n}{so, E, S \vdash \{e_1; \dots e_n\} : v_n, S_n} [Block]$$

Stores

ОБНОВЛЯЮТСЯ

$so, E, S \vdash e_1, v_1, S_1$

$so, E, S_1 \vdash e_2, v_2, S_2$

$$\frac{so, E, S_1 \vdash e_2, v_2, S_2}{so, E, S \vdash e_1 + e_2 : v_1 + v_2, S_2} [Add]$$

$so, E, S \vdash e_1 : Bool(true), S_1$

$so, E, S_1 \vdash e_2 : v, S_2$

$$\frac{so, E, S_1 \vdash e_2 : v, S_2}{so, E, S \vdash if\ e_1\ then\ e_2\ else\ e_3\ fi : v, S_2} [if - true]$$

Examples #3

$Ex : \{x \leftarrow 7 + 5; 4\}$

$$so, [x : l], [l \leftarrow 0] \vdash 7 : Int(7), [l \leftarrow 0]$$
$$so, [x : l], [l \leftarrow 0] \vdash 5 : Int(5), [l \leftarrow 0]$$
$$\frac{so, [x : l], [l \leftarrow 0] \vdash 5 : Int(5), [l \leftarrow 0]}{so, [x : l], [l \leftarrow 0] \vdash 7 + 5 : Int(12), [l \leftarrow 0]}$$
$$[l \leftarrow 0](12 / l) = [l \leftarrow 12]$$
$$\frac{[l \leftarrow 0](12 / l) = [l \leftarrow 12]}{so, [x : l], [l \leftarrow 0] \vdash x \leftarrow 7 + 5 : 12, [l \leftarrow 12]}$$
$$so, [x : l], [l \leftarrow 12] \vdash 4 : Int(4), [l \leftarrow 12]$$
$$\frac{so, [x : l], [l \leftarrow 0] \vdash x \leftarrow 7 + 5 : 12, [l \leftarrow 12] \quad so, [x : l], [l \leftarrow 12] \vdash 4 : Int(4), [l \leftarrow 12]}{so, [x : l], [l \leftarrow 0] \vdash \{x \leftarrow 7 + 5; 4\} : Int(4), [l \leftarrow 12]}$$

Examples #4

$$\frac{so, E, S \vdash e_1 : Bool(false), S_1}{so, E, S \vdash while\ e_1\ loop\ e_2\ pool : void, S_1} [while - false]$$

$$so, E, S \vdash e_1 : Bool(true), S_1$$

$$so, E, S_1 \vdash e_2 : v, S_2$$

$$so, E, S_2 \vdash while\ e_1\ loop\ e_2\ pool : void, S_3$$

$$\frac{so, E, S_2 \vdash while\ e_1\ loop\ e_2\ pool : void, S_3}{so, E, S \vdash while\ e_1\ loop\ e_2\ pool : void, S_3} [while - true]$$

Рекуррентное
исполнение с
новым store

Let

$$\frac{\begin{array}{c} so, E, S \vdash e_1, v_1, S_1 \\ so, ??? \vdash e_2, v_2, S_2 \end{array}}{so, E, S \vdash \text{let } id \leftarrow e_1 \text{ in } e_2 : v_2, S_2}$$

Для заполнения новой переменной id необходимо добавить новую память и заполнить ее.

$l_{new} = newloc(S)$ - новая область в памяти, не использованная в S (malloc)

$$\frac{\begin{array}{c} so, E, S \vdash e_1, v_1, S_1 \\ l_{new} = newloc(S_1) \\ so, E[l_{new} / id], S_1[v_1 / l_{new}] \vdash e_2, v_2, S_2 \end{array}}{so, E, S \vdash \text{let } id \leftarrow e_1 \text{ in } e_2 : v_2, S_2}$$

Allocation an objects and dynamic dispatch

Аллокация объектов и динамический вызов :

для выполнения вызова "new T" необходимо - **выделение памяти** для хранения всех атрибутов для объекта класса T, **выставление дефолтных значений** атрибутов, **выполнение инициализации**(вычисление expression) и установление посчитанных атрибутов и возврат созданного объекта (указателя на его начало).

Дефолтные значения:

$$D_{\text{int}} = \text{Int}(0)$$

$$D_{\text{bool}} = \text{Bool}(\text{false})$$

$$D_{\text{string}} = \text{String}(0, "")$$

$$D_A = \text{void} \ (\forall \text{ other classes})$$

Class init.

$$class(A) = (a_1 : T_1 \leftarrow e_1, \dots, a_n : T_n \leftarrow e_n)$$

a_i – атрибуты (включая унаследованные)

T_i – типы

e_i – инициализации значений

$$T_0 = \text{if } (T \equiv SELF_TYPE \text{ and } so = X(\dots)) \text{ then } X \text{ else } T$$
$$class(T_0) = (a_1 : T_1 \leftarrow e_1, \dots, a_n : T_n \leftarrow e_n)$$
$$l_{new} = \text{newloc}(S) \text{ for } i = 1, \dots, n$$
$$v = T_0(a_1 = l_1, \dots, a_n = l_n)$$

Возвращаемое
значение

$$S_1 = S[D_{T_1} / l_1, \dots, D_{T_n} / l_n]$$

Дефолтные
значения

$$E' = [a_1 : l_1, \dots, a_n : l_n]$$
$$v, E', S_1 \vdash \{a_1 \leftarrow l_1; \dots; a_n \leftarrow l_n\} : v_n, S_2$$
$$\frac{}{so, E, S \vdash \text{new } T : v, S_2} [new]$$

Первые 3 шага - это алокация объекта, остальные выполнение последовательности для определения значений.

В score попадают только атрибуты.

Инициализация значений по-умолчанию, а заполнение происходит run-time.

Dinamic dispatch $e_0.f(e_1, \dots, e_n)$

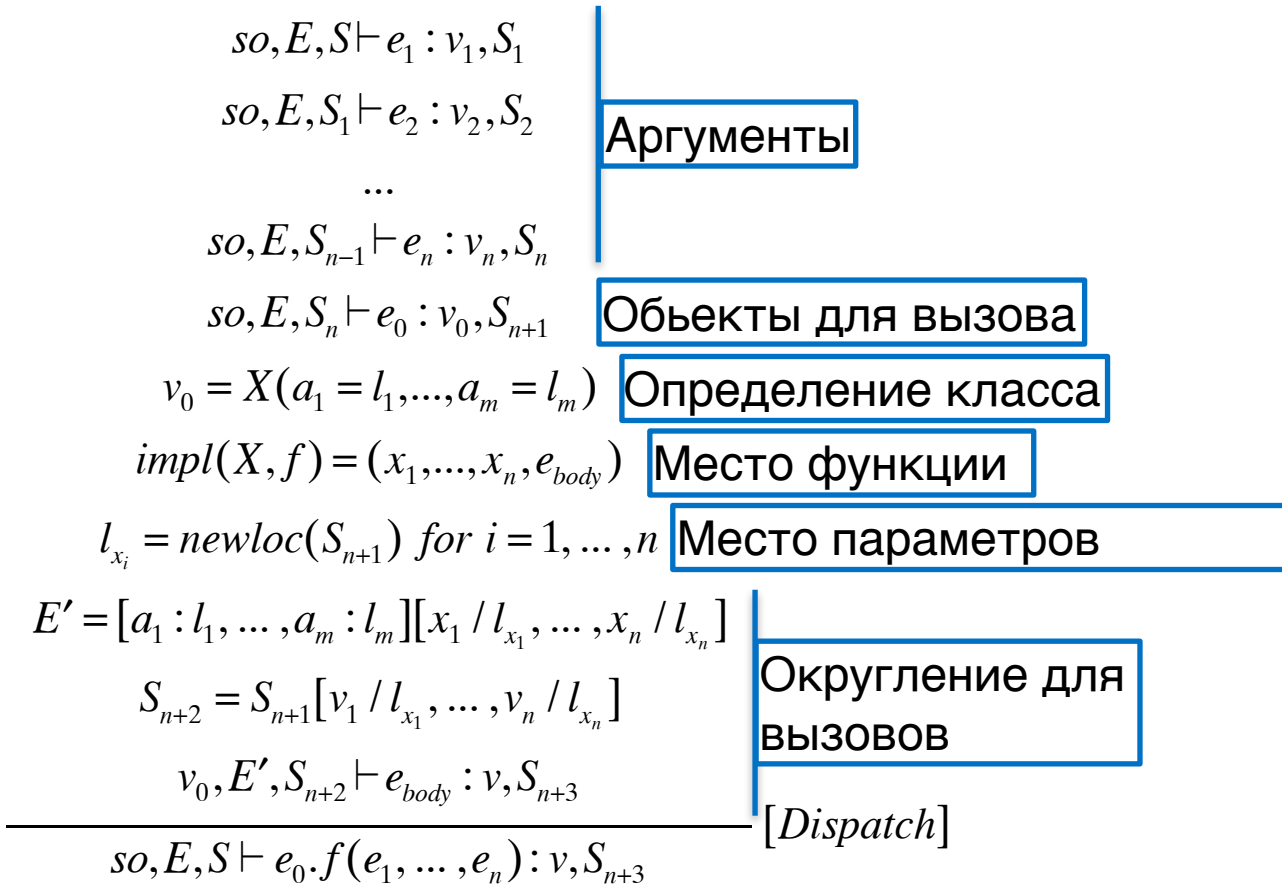
1. Выполнить e_1, \dots, e_n
2. Выполнить e_0
3. Заполнить X - динамическим типом объекта
4. Получить из X определение f (с аргументами)
5. Создать n новых локаций и окружений для f аргументов (параметров)
6. Инициализировать их
7. Поменять $self$ на определенный выше объект типа X

Для A и метода f из A (возможно унаследованного)

$$impl(A, f) = (x_1, \dots, x_n, e_{body})$$

x_i — аргументы

e_{body} — тело метода



[Dispatch]

Тело вызова метода
использует

- E - определение аргументов и атрибутов класса
- S - место аргументов

Определение AR -
неявно, дает
свободу.

Summary

Правила семантики полагаются на семантический анализ, т.е. гарантии того, что методы и аргументы существуют.

Однако возможны ошибки времени выполнения:

- вызов void
- деление на ноль
- подстрока выходит за границу
- переполнение кучи

Хотелось бы иметь на такие случаи осмысленный вывод, а не segfault.