Об обратной проекции Футамуры и новой реализации Рефала

П.Н. Советов, РТУ МИРЭА

День Проблем 2024

1. **Обратная проекция Футамуры.**
2. Новая реализация Рефала.

Первая проекция Футамуры

$$\texttt{spec}(int_b,\ p_a) \rightarrow p_b$$

Специализатор по **интерпретатору** и некоторой **программе** выдает **скомпилированную версию программы**.

Обратная “первая проекция Футамуры”

$$\texttt{magic}(P, C) \rightarrow int$$

“Магическая” функция по некоторому **набору программ** и набору **архитектурных ограничений** выдает **интерпретатор**.

“Магическая функция” — генератор интерпретаторов

```
intgen(P, C) := generalize(extract(P, C))
```

Синтез интерпретатора с учетом архитектурных ограничений:

1. **extract**: синтез команд.
2. **generalize**: обобщение синтезированных команд.

Функция extract

Извлечение из графов программ P **повторяющихся подграфов** — кандидатов в команды входного языка.

Желательно найти множество команд, дающее лучшую компрессию исходных графов программ.

Известные подходы:

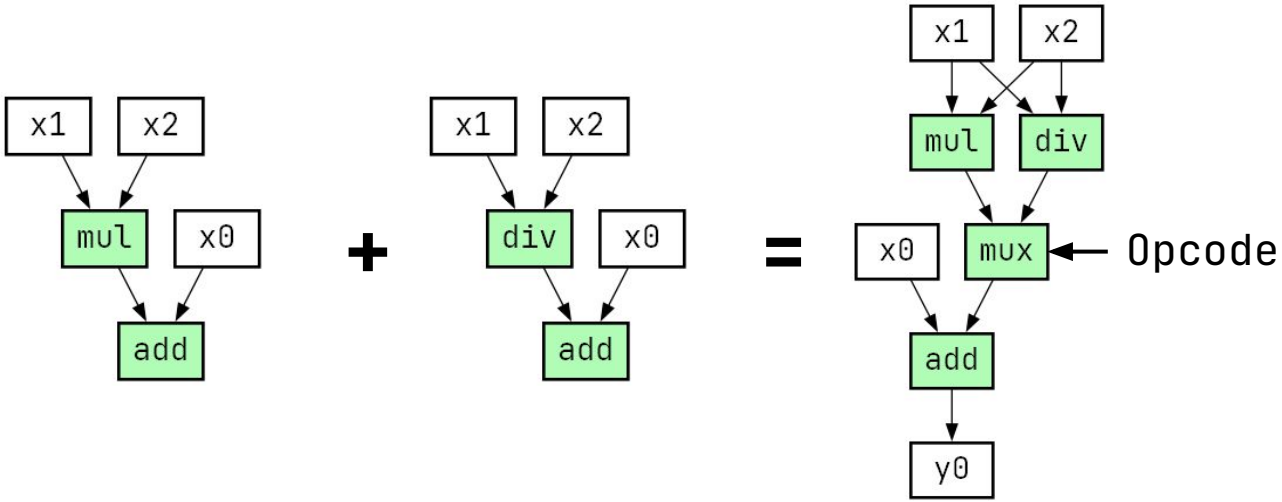
1. **Синтез предметно-ориентированных команд для спецпроцессоров**.
2. Библиотечное обучение (library learning).

> “Library learning compresses a given corpus of programs by extracting common structure from the corpus into reusable library functions”

Bowers, Matthew, et al. "Top-down synthesis for library learning." Proceedings of the ACM on Programming Languages 7.POPL (2023): 1182-1213.

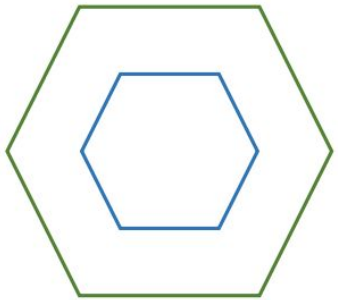
Функция generalize

1. Удаление из множества синтезированных команд семантических дублей и частных случаев.
2. Обобщение команд с разделением общих операций.



Nuts & Bolts: модельный DSL для графики

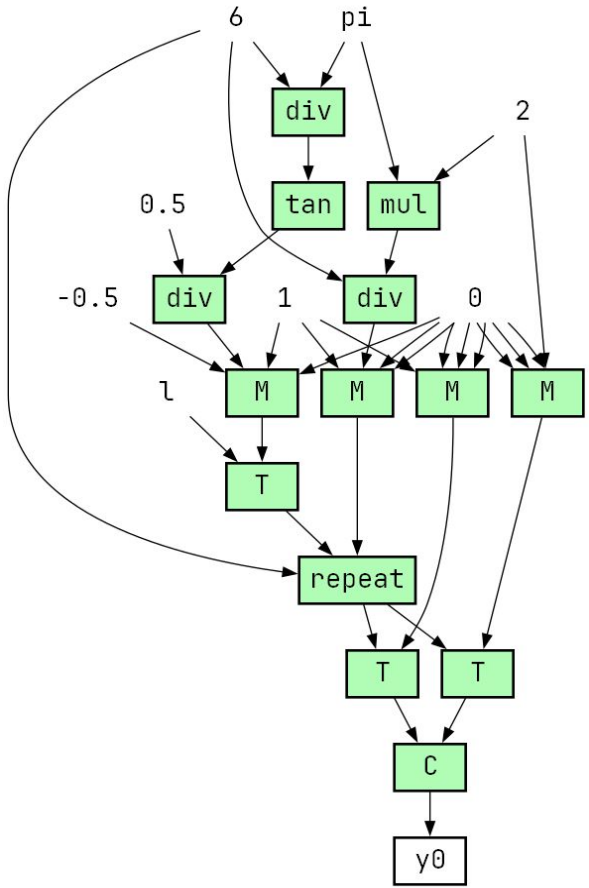
```
C(T(repeat(T(l, M(1, 0, -0.5, div(0.5, tan(div(pi, 6))))), 6,
           M(1, div(mul(2, pi), 6), 0, 0)), M(2, 0, 0, 0)),
  T(repeat(T(l, M(1, 0, -0.5, div(0.5, tan(div(pi, 6))))), 6,
           M(1, div(mul(2, pi), 6), 0, 0)), M(1, 0, 0, 0)))
```



Wang, H., Polikarpova, N., & Fan, J. E. (2021). Learning part-based abstractions for visual object concepts. In Proceedings of the Annual Meeting of the Cognitive Science Society (Vol. 43, No. 43).

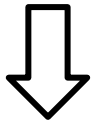
Перевод в графовое представление

```
C(T(repeat(
        T(l,
          M(1, 0, -0.5,
            div(0.5, tan(div(pi, 6))))),
        6,
        M(1, div(mul(2, pi), 6), 0, 0)),
    M(2, 0, 0, 0)),
  T(repeat(
        T(l,
          M(1, 0, -0.5,
            div(0.5, tan(div(pi, 6))))),
        6,
        M(1, div(mul(2, pi), 6), 0, 0)),
    M(1, 0, 0, 0)))
```



Автоматический рефакторинг

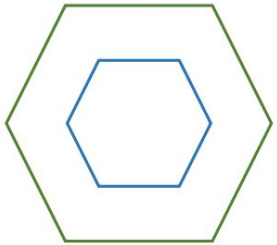
```
C(T(repeat(T(l, M(1, 0, -0.5, div(0.5, tan(div(pi, 6))))), 6,
           M(1, div(mul(2, pi), 6), 0, 0)), M(2, 0, 0, 0)),
  T(repeat(T(l, M(1, 0, -0.5, div(0.5, tan(div(pi, 6))))), 6,
           M(1, div(mul(2, pi), 6), 0, 0)), M(1, 0, 0, 0)))
```



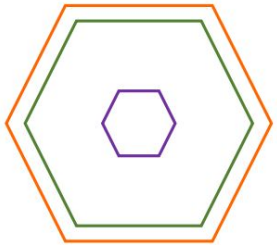
```
f9() = M(1, 0, -0.5, div(0.5, tan(div(pi, 6))))
f2(x0) = repeat(T(l, x0), 6, M(1, div(mul(2, pi), 6), 0, 0))
f19(x0) = C(T(x0, M(2, 0, 0, 0)), T(x0, M(1, 0, 0, 0)))

f19(f2(f9()))
```

3 программы

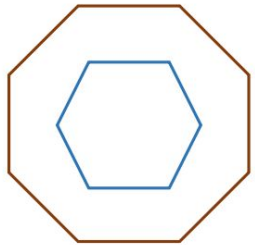
```
y0 = C(T(repeat(T(l, M(1, 0, -0.5, div(0.5, tan(div(pi, 6))))), 6,
                M(1, div(mul(2, pi), 6), 0, 0)), M(2, 0, 0, 0)),
       T(repeat(T(l, M(1, 0, -0.5, div(0.5, tan(div(pi, 6))))), 6,
                M(1, div(mul(2, pi), 6), 0, 0)), M(1, 0, 0, 0)))
```



```
y0 = C(C(T(repeat(T(l, M(1, 0, -0.5, div(0.5, tan(div(pi, 6))))), 6,
                  M(1, div(mul(2, pi), 6), 0, 0)), M(2.25, 0, 0, 0)),
         T(repeat(T(l, M(1, 0, -0.5, div(0.5, tan(div(pi, 6))))), 6,
                  M(1, div(mul(2, pi), 6), 0, 0)), M(2, 0, 0, 0))),
       T(repeat(T(l, M(1, 0, -0.5, div(0.5, tan(div(pi, 6))))), 6,
                M(1, div(mul(2, pi), 6), 0, 0)), M(0.5, 0, 0, 0)))
```



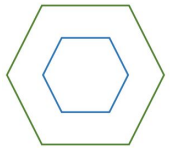
```
y0 = C(T(repeat(T(l, M(1, 0, -0.5, div(0.5, tan(div(pi, 8))))), 8,
                  M(1, div(mul(2, pi), 8), 0, 0)), M(2, 0, 0, 0)),
       T(repeat(T(l, M(1, 0, -0.5, div(0.5, tan(div(pi, 6))))), 6,
                  M(1, div(mul(2, pi), 6), 0, 0)), M(1, 0, 0, 0)))
```



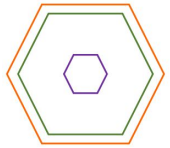
Извлечение команд из 3 программ

```
f0(x0, x1) = repeat(T(l, M(1, 0, -0.5, div(0.5, tan(div(pi, x0))))), x0,
                    M(1, div(x1, x0), 0, 0))
f713(x0) = mul(2, x0)
f68(x0, x1) = C(T(x0, M(2, x1, x1, x1)), T(x0, M(1, x1, x1, x1)))
f29(x0, x1) = C(C(T(x0, M(2.25, x1, x1, x1)), T(x0, M(2, x1, x1, x1))),
                T(x0, M(0.5, x1, x1, x1)))
```

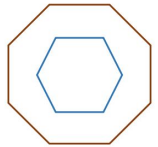
```
y0 = f68(f0(6, f713(pi)), 0)
```



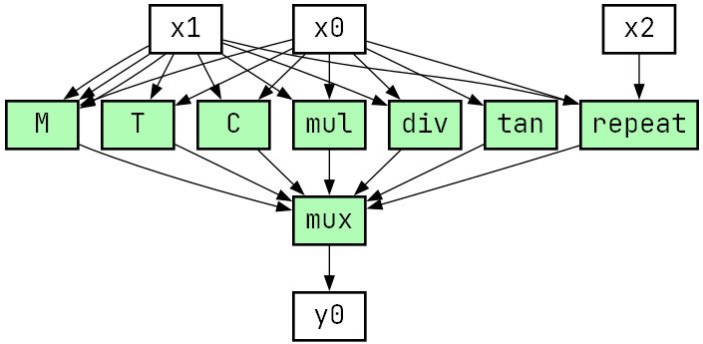
```
y0 = f29(f0(6, f713(pi)), 0)
```



```
y0 = f68(f0(8, f713(pi)), 0)
```

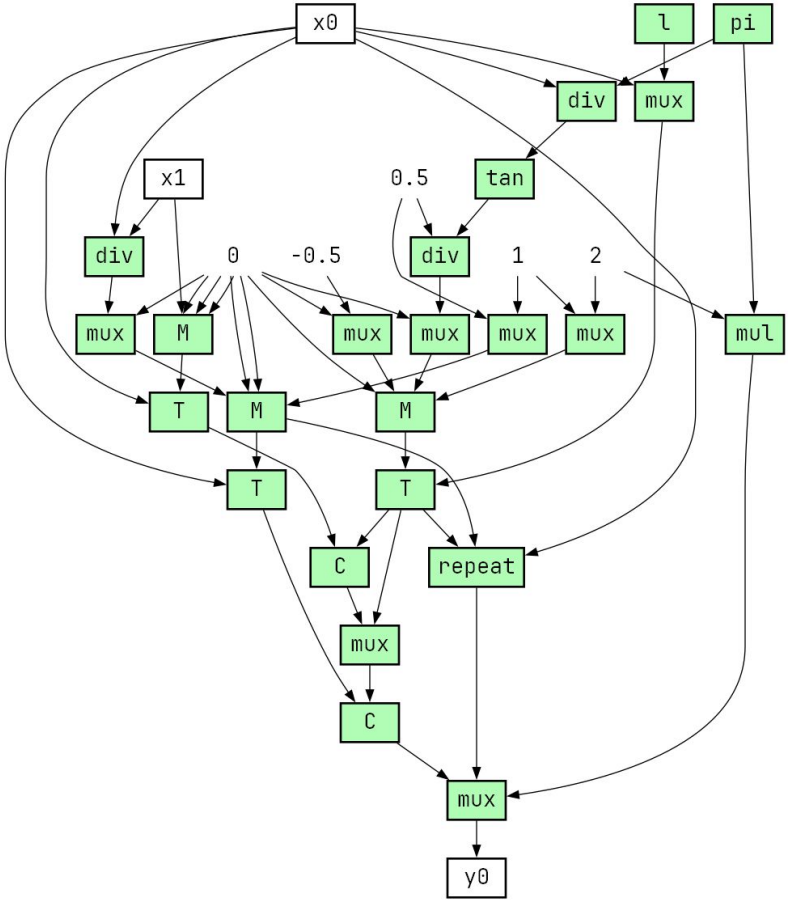


Слияние базового набора команд



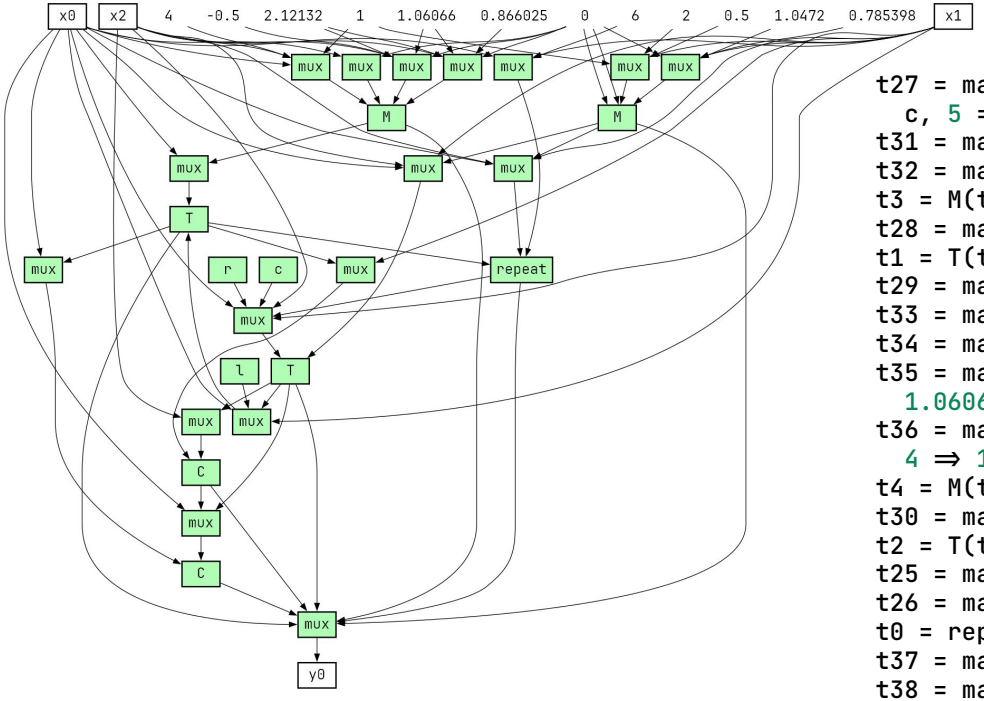

```
t2 = M(x[0], x[1], x[1], x[1]);
t3 = div(x[0], x[1]);
t4 = T(x[0], x[1]);
t5 = C(x[0], x[1]);
t7 = repeat(x[0], x[1], x[2]);
t8 = tan(x[0]);
t9 = mul(x[0], x[1]);
y[0] = match op[10] {0 ⇒ t2, 1 ⇒ t3, 2 ⇒ t4,
                     3 ⇒ t5, 4 ⇒ t7, 5 ⇒ t8, 6 ⇒ t9};
```

Синтез интерпретатора для 3 программ (4 команды)




```
t2 = match op[2] {0 ⇒ l, 1 ⇒ x[0]};
t5 = match op[5] {0 ⇒ 1, 1 ⇒ 2};
t8 = match op[8] {0 ⇒ 0, 1 ⇒ -0.5};
t11 = div(pi, x[0]);
t12 = tan(t11);
t13 = div(0.5, t12);
t14 = match op[14] {0 ⇒ 0, 1 ⇒ t13};
t15 = M(t5, 0, t8, t14);
t16 = T(t2, t15);
t17 = match op[17] {0 ⇒ 1, 1 ⇒ 0.5};
t19 = div(x[1], x[0]);
t20 = match op[20] {0 ⇒ t19, 1 ⇒ 0};
t21 = M(t17, t20, 0, 0);
t22 = repeat(t16, x[0], t21);
t23 = mul(2, pi);
t24 = M(x[1], 0, 0, 0);
t25 = T(x[0], t24);
t26 = C(t25, t16);
t27 = match op[27] {0 ⇒ t16, 1 ⇒ t26};
t28 = T(x[0], t21);
t29 = C(t27, t28);
y[0] = match op[30] {0 ⇒ t22, 1 ⇒ t23, 2 ⇒ t29};
```

Синтез интерпретатора для 250 программ (29 команд)




```
t27 = match op[27] {0 ⇒ x[2], 1 ⇒ x[0], 2 ⇒ x[1], 3 ⇒ t0, 4 ⇒
  c, 5 ⇒ r};
t31 = match op[31] {0 ⇒ 1, 1 ⇒ 0.5, 2 ⇒ x[1]};
t32 = match op[32] {0 ⇒ 0, 1 ⇒ x[1], 2 ⇒ 1.0472, 3 ⇒ 0.785398};
t3 = M(t31, t32, 0, 0);
t28 = match op[28] {0 ⇒ t3, 1 ⇒ x[0], 2 ⇒ x[1], 3 ⇒ x[2]};
t1 = T(t27, t28);
t29 = match op[29] {0 ⇒ t1, 1 ⇒ x[0], 2 ⇒ l, 3 ⇒ x[1]};
t33 = match op[33] {0 ⇒ 1, 1 ⇒ x[2], 2 ⇒ x[0], 3 ⇒ 2, 4 ⇒ 4};
t34 = match op[34] {0 ⇒ 0, 1 ⇒ x[0]};
t35 = match op[35] {0 ⇒ 2.12132, 1 ⇒ 0, 2 ⇒ -0.5, 3 ⇒ x[2], 4 ⇒
  1.06066};
t36 = match op[36] {0 ⇒ 2.12132, 1 ⇒ 0, 2 ⇒ x[2], 3 ⇒ 0.866025,
  4 ⇒ 1.06066};
t4 = M(t33, t34, t35, t36);
t30 = match op[30] {0 ⇒ t4, 1 ⇒ x[0]};
t2 = T(t29, t30);
t25 = match op[25] {0 ⇒ x[0], 1 ⇒ 6, 2 ⇒ x[1]};
t26 = match op[26] {0 ⇒ x[1], 1 ⇒ t3, 2 ⇒ x[0], 3 ⇒ x[2]};
t0 = repeat(t2, t25, t26);
t37 = match op[37] {0 ⇒ x[1], 1 ⇒ t2};
t38 = match op[38] {0 ⇒ t1, 1 ⇒ x[2]};
t12 = C(t37, t38);
t39 = match op[39] {0 ⇒ t12, 1 ⇒ t1, 2 ⇒ x[0]};
t40 = match op[40] {0 ⇒ x[0], 1 ⇒ t2};
t13 = C(t39, t40);
y[0] = match op[41] {0 ⇒ t0, 1 ⇒ t13, 2 ⇒ t2, 3 ⇒ t4, 4 ⇒ t3, 5
  ⇒ t1, 6 ⇒ t12};
```

1. Обратная проекция Футамуры.
2. **Новая реализация Рефала.**

RefalPy

Написан специально для доклада “Автоматизация программирования в СССР”.

- Для обучения студентов.
- Для формализации семантики языков программирования.

Особенности реализации

- Миниатюрная реализация: 325 строк на Python.
- Компилятор в байткод виртуальной машины.
- Неизменяемые кортежи вместо списков.
- DSL, интегрированный в Python.
- Синтаксис (поневоле) вдохновлен Mathematica.

Реализация алгоритма RLE

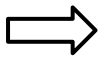
```
@refal({
    'add': lambda arg: (arg[0] + arg[1],)
})
def rules():
    rle[s.x, e.tail] = rle[{s.x, 1}, e.tail]
    rle[{s.x, s.c}, s.x, e.tail] = rle[{s.x, add[s.c, 1]}, e.tail]
    rle[{s.x, s.c}, s.y, e.tail] = {s.x, s.c}, rle[{s.y, 1}, e.tail]
    rle[e.x] = e.x
```

Получение верхней новости с HN

```
TOP_URL = 'https://hacker-news.firebaseio.com/v0/topstories.json'

ITEM_URL = 'https://hacker-news.firebaseio.com/v0/item/'

first[s.x, e.y] = s.x

get[{e.x, {s.key, t.val}, e.y}, s.key] = t.val

item_url[s.x] = concat[ITEM_URL, s.x, '.json']

top_index = first[tuple[download_json[TOP_URL]]]

msg[s.x] = get[{dict[download_json[item_url[s.x]]]}, 'title']

top = msg[top_index]
```

Компилятор и интерпретатор

```
(('n' '=' 10)
 ('s' '='
  (('n' '*' ('n' '+' 1)) '/' 2)))
```



```
('push' 10)
('store' 'n')
('load' 'n')
('load' 'n')
('push' 1)
'add'
'mul'
('push' 2)
'div'
('store' 's')
```

```
(('n' 10) ('s' 55))
```

Компилятор и интерпретатор: реализация

```
op['int'] = 'push'
op['str'] = 'load'

comp[{s.var, '=', t.expr}, e.prog] = comp[t.expr], {
    'store', s.var}, comp[e.prog]
comp[{t.a, '+', t.b}] = comp[t.a], comp[t.b], 'add'
comp[{t.a, '-', t.b}] = comp[t.a], comp[t.b], 'sub'
comp[{t.a, '*', t.b}] = comp[t.a], comp[t.b], 'mul'
comp[{t.a, '/', t.b}] = comp[t.a], comp[t.b], 'div'
comp[s.val] = {op[type[s.val]], s.val}
comp = _
```

```
get[{e.x, {s.key, t.val}, e.y}, s.key] = t.val
set[{e.x, {s.key, t.val}, e.y}, s.key, t.new] = {e.x, {s.key, t.new}, e.y}
set[{e.x}, s.key, t.val] = {e.x, {s.key, t.val}}

interp[{t.cmd, e.code}, t.stack, t.env] = interp[{
    e.code}, step[t.cmd, t.stack, t.env]]
interp[{}, {}, t.env] = t.env

step['add', {e.stack, s.x, s.y}, t.env] = {e.stack, add[s.x, s.y]}, t.env
step['sub', {e.stack, s.x, s.y}, t.env] = {e.stack, sub[s.x, s.y]}, t.env
step['mul', {e.stack, s.x, s.y}, t.env] = {e.stack, mul[s.x, s.y]}, t.env
step['div', {e.stack, s.x, s.y}, t.env] = {e.stack, div[s.x, s.y]}, t.env
step[{'push', s.x}, {e.stack}, t.env] = {e.stack, s.x}, t.env
step[{'load', s.var}, {e.stack}, t.env] = {
    e.stack, get[t.env, s.var]}, t.env
step[{'store', s.var}, {e.stack, s.val}, t.env] = {
    e.stack}, set[t.env, s.var, s.val]

sem[e.prog] = interp[{comp[e.prog]}, {}, {}]
```

Компилятор и интерпретатор: форматирование кода

```
<op 'int'> = 'push'
<op 'str'> = 'load'

<comp (s.var '=' t.expr) e.prog> = <comp t.expr>
('store' s.var) <comp e.prog>
<comp (t.a '+' t.b)> = <comp t.a> <comp t.b> 'add'
<comp (t.a '-' t.b)> = <comp t.a> <comp t.b> 'sub'
<comp (t.a '*' t.b)> = <comp t.a> <comp t.b> 'mul'
<comp (t.a '/' t.b)> = <comp t.a> <comp t.b> 'div'
<comp s.val> = (<op <type s.val>> s.val)
<comp> = _
```

```
<get (e.x (s.key t.val) e.y) s.key> = t.val

<set (e.x (s.key t.val) e.y) s.key t.new> = (e.x (s.key t.new) e.y)
<set (e.x) s.key t.val> = (e.x (s.key t.val))

<interp (t.cmd e.code) t.stack t.env> = <interp (e.code) <step t.cmd t.stack
t.env>>
<interp () () t.env> = t.env

<step 'add' (e.stack s.x s.y) t.env> = (e.stack <add s.x s.y>) t.env
<step 'sub' (e.stack s.x s.y) t.env> = (e.stack <sub s.x s.y>) t.env
<step 'mul' (e.stack s.x s.y) t.env> = (e.stack <mul s.x s.y>) t.env
<step 'div' (e.stack s.x s.y) t.env> = (e.stack <div s.x s.y>) t.env
<step ('push' s.x) (e.stack) t.env> = (e.stack s.x) t.env
<step ('load' s.var) (e.stack) t.env> = (e.stack <get t.env s.var>) t.env
<step ('store' s.var) (e.stack s.val) t.env> = (e.stack) <set t.env s.var
s.val>

<sem e.prog> = <interp (<comp e.prog>) () ()>
```

PLT Redex: стековый интерпретатор

```
(define stack-red
  (reduction-relation stack-lang
   #:domain prog
   (⟶ [(lit_1 cmd ...) (lit ...)]
        [(cmd ...) (lit_1 lit ...)]
        "literal")
   (⟶ [(dup cmd ...) (lit_1 lit ...)]
        [(cmd ...) (lit_1 lit_1 lit ...)]
        "dup")
   (⟶ [(drop cmd ...) (lit_1 lit ...)]
        [(cmd ...) (lit ...)]
        "drop")
   (⟶ [(swap cmd ...) (lit_1 lit_2 lit ...)]
        [(cmd ...) (lit_2 lit_1 lit ...)]
        "swap")
   (⟶ [(+ cmd ...) (lit_1 lit_2 lit ...)]
        [(cmd ...) (lit_3 lit ...)]
        (where lit_3 ,(+ (term lit_2) (term lit_1)))
        "+")
   (⟶ [(- cmd ...) (lit_1 lit_2 lit ...)]
        [(cmd ...) (lit_3 lit ...)]
        (where lit_3 ,(- (term lit_2) (term lit_1)))
        "-")
   (⟶ [(* cmd ...) (lit_1 lit_2 lit ...)]
        [(cmd ...) (lit_3 lit ...)]
        (where lit_3 ,(* (term lit_2) (term lit_1)))
        "*")
   (⟶ [(/ cmd ...) (lit_1 lit_2 lit ...)]
        [(cmd ...) (lit_3 lit ...)]
        (where lit_3 ,(/ (term lit_2) (term lit_1)))
        "/")))
```

K Framework: стековый интерпретатор

```
module STACK
  imports STACK-SYNTAX
  imports DOMAINS

  configuration <T> <k> $PGM:Prog </k> <stack> .List </stack> </T>

  rule C1:Cmd C2:Prog ⇒ C1 ~> C2

  rule <k> I:Int ⇒ . ... </k> <stack> .List ⇒ ListItem(I) ... </stack>

  rule <k> dup ⇒ . ... </k> <stack> ListItem(I) ⇒ ListItem(I) ListItem(I) ...</stack>
  rule <k> drop ⇒ . ... </k> <stack> ListItem(I) ⇒ .List ...</stack>
  rule <k> swap ⇒ . ... </k> <stack> ListItem(I) ListItem(J) ⇒ ListItem(J) ListItem(I) ...</stack>

  rule <k> + ⇒ . ... </k> <stack> ListItem(I) ListItem(J) ⇒ ListItem(J +Int I) .List ... </stack>
  rule <k> - ⇒ . ... </k> <stack> ListItem(I) ListItem(J) ⇒ ListItem(J -Int I) .List ... </stack>
  rule <k> * ⇒ . ... </k> <stack> ListItem(I) ListItem(J) ⇒ ListItem(J *Int I) .List ... </stack>
  rule <k> / ⇒ . ... </k> <stack> ListItem(I) ListItem(J) ⇒ ListItem(J /Int I) .List ... </stack>
endmodule
```

Спасибо за внимание!

https://github.com/true-grue/refalpy