Система типов для чистого статически типизированного функционального конкатенативного языка программирования с поддержкой функций первого класса

2025

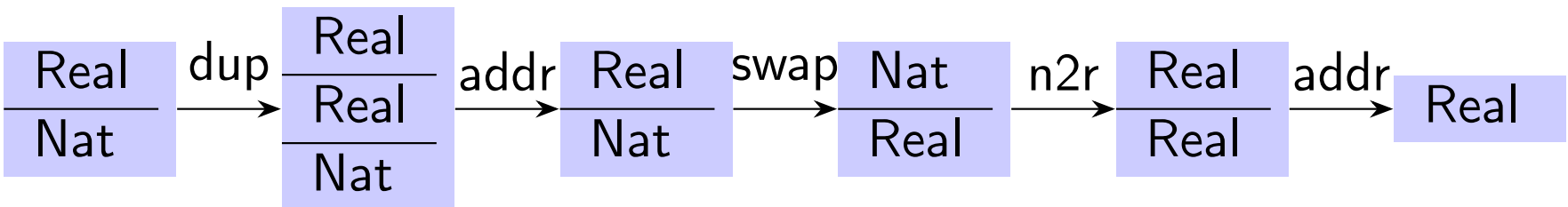
Конкатенативные языки программирования

Аппликативные ЯП [**Haskell**]

```
foo :: Real -> Nat -> Real
foo r n = (r `addr` r) `addr` (n2r n)
```

Конкатенативные ЯП [**IV**]

```
define [Real, Nat] foo [Real]:
  dup addr swap n2r addr.
```



Конкатенативные и стековые языки программирования

Динамическая типизация

- Forth
- PostScript
- Joy
- Factor

Статическая типизация

- WASM
- Cat
- Kitten
- IV

IV и IV++

Две версии созданного ЯП отличаются поддержкой функций первого класса (ФПК)

IV

- Пользовательские структуры
- Пользовательские операторы
- **Нет ФПК**
- Полный по Тьюрингу

IV++

- Надмножество IV
- **Поддержка ФПК**

IV

- Объявление структуры

```
data Maybe a:
  nothing,
  [a] just.
```

- Объявление оператора

```
define [Nat, Nat] add [Nat]:
  case { zero { }, suc { add suc } }.

define [Nat] addThree [Nat]:
  zero suc suc suc add.
```

Правила вывода типов (без поддержки ФПК)

Разделение между типом и типом оператора

Система типов IV разделяет между типами и типами операторов.

- **Тип** обозначает тип значения значения на стеке.
- **Тип оператора** обозначает тип части тела оператора.

Тип

Тип может быть

- **Mono** - мономорфный тип, e.g., `Int`, `String`.
- **Var** - переменная типа, e.g., `a`, `b`.
- **App** - применение типа на другой тип, e.g., `Maybe a`, `Either a b`.

Тип оператора (нотация)

$$[\mathrm{pre}_1, \mathrm{pre}_2, \ldots, \mathrm{pre}_m][\mathrm{post}_1, \mathrm{post}_2, \ldots, \mathrm{post}_n]$$

- pre - входные аргументы
- post - выходные аргументы

Тип оператора (нотация)

$$[]\texttt{foo bar baz}[\text{Baz}, \text{Bar}, \text{Foo}]$$

Является сокращением

$$\texttt{foo bar baz} : [][\text{Baz}, \text{Bar}, \text{Foo}]$$

Тип оператора (нотация)

$$[]\texttt{foo bar baz}[\text{Baz}, \text{Bar}, \text{Foo}]$$

Самый левый тип в списке выходных аргументов обозначает тип самого недавно добавленного в стек значения.

Система типов

- Все операторы должны иметь аннотацию с типом оператора.
- Γ содержит информацию об объявленных операторах и структурах в программе.
- Γ не меняется в процессе выведения типов.

Empty rule

$$\frac{}{\Gamma \vdash [][]}\ \text{Empty}$$

Позволяет использовать пустую последовательность операторов, которая не меняет стек.

Operator specialization rule

$$\frac{[\alpha][\beta] = \{a' \mapsto a\}[\alpha'][\beta']}{[\alpha][\beta] \sqsupseteq [\alpha'][\beta']}\ \text{Spec}$$

Тип считается специализацией другого общего типа, если существует замена, которая превращает общий тип в частный.

Name rule

$$\frac{[\alpha]\mathsf{op}[\beta] \in \Gamma}{\Gamma \vdash [\alpha]\mathsf{op}[\beta]}\ \mathsf{Name}$$

Позволяет использовать объявленные операторы по их именам.

Specialization and extension rule

$$\frac{\Gamma \vdash [\alpha']\mathsf{x}[\beta'] \qquad [\alpha][\beta] \sqsupseteq [\alpha'][\beta']}{\Gamma \vdash [\alpha \cdot \gamma]\mathsf{x}[\beta \cdot \gamma]}\ \mathsf{SpecExt}$$

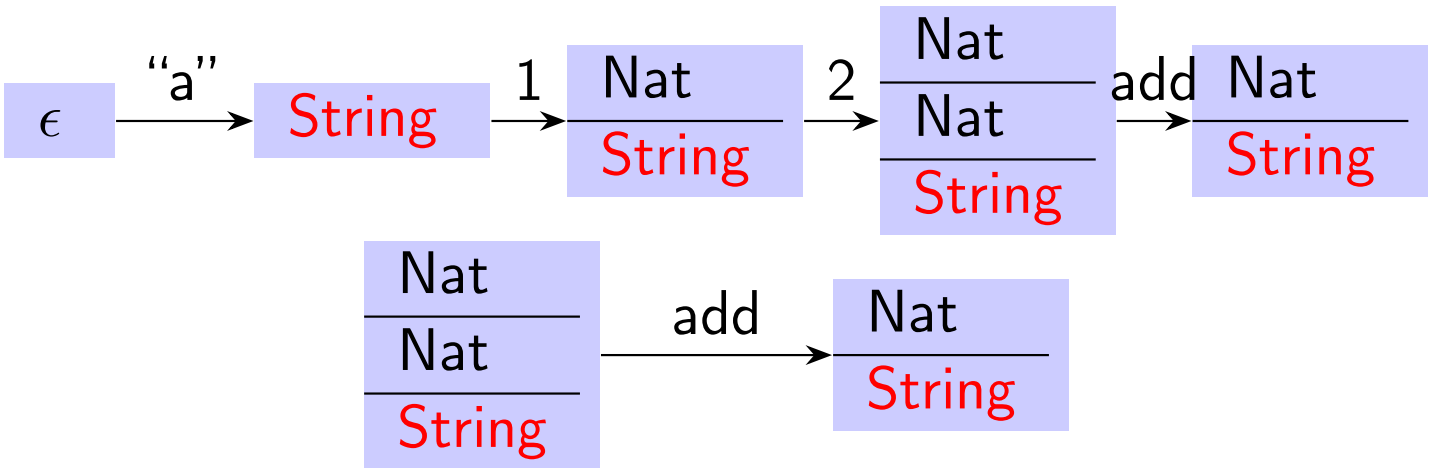
Позволяет специализировать и/или дополнить тип оператора

- Специализация
  Позволяет использовать [*a*]`id`[*a*] вместо [Nat]`inc`[Nat]
- Дополнение
  Позволяет использовать [Nat] `inc` [Nat] вместо [Nat, Nat] `inc2` [Nat, Nat]

Более о дополнении типа оператора на следующем слайде.

Дополнение типа оператора

- Дополнение типа оператора подразумевает добавление дополнительных пар входных-выходных аргументов.
- Дополнение типа оператора позволяет вызывать оператор в контексте стека с большем количеством элеметнов чем количество входных аргументов оператора.
- Созданные пары входных-выходных аргументов не меняют значения элементов.



Chain rule

$$\frac{\Gamma \vdash [\alpha]\mathsf{x}[\beta] \qquad \Gamma \vdash [\psi]\mathsf{y}[\omega] \qquad \beta = \psi}{\Gamma \vdash [\alpha]\mathsf{x}\ \ \mathsf{y}[\omega]}\ \text{Chain}$$

Позволяют производить композицию операторов. Для этого необходимо чтобы кол-во выходных арг. левого оператора было равно кол-ву входных арг. правого оператора.

О композиции операторов с разным кол-вом входных-выходных арг на следующем слайде.

Композиция с дополнением

Для того чтобы произвести композицию двух операторов с разным кол-вом входных-выходных арг используется SpecExt правило.

- Выходные левого $>$ Входные правого (Overflow chain)

$$\frac{\Gamma \vdash [A]\mathtt{foo}[B, C] \qquad \dfrac{\Gamma \vdash [B]\mathtt{bar}[D]}{\Gamma \vdash [B, C]\mathtt{bar}[D, C]} \qquad [B, C] = [B, C]}{\Gamma \vdash [A]\mathtt{foo}\ \ \mathtt{bar}[D, C]}$$

- Выходные левого $<$ Входные правого (Underflow chain)

$$\frac{\dfrac{\Gamma \vdash [A]\mathtt{foo}[B]}{\Gamma \vdash [A, C]\mathtt{foo}[B, C]} \qquad \Gamma \vdash [B, C]\mathtt{bar}[D] \qquad [B, C] = [B, C]}{\Gamma \vdash [A, C]\mathtt{foo}\ \ \mathtt{bar}[D]}$$

Case rule

Выражение сопоставления с образцом

$$\frac{\{\texttt{constr1}, \ldots\} = \text{constrs}(t) \qquad \Gamma \vdash [t, \alpha]\texttt{constr1}^{-1}\ \texttt{body1}[\beta] \qquad [t, \alpha][\beta] \sqsupseteq [t, \alpha'][\beta'] \qquad \ldots}{\Gamma \vdash [t, \alpha]\texttt{case}\{\texttt{constr1}\{\texttt{body1}\}, \ldots\}[\beta]}$$

У каждого конструктора есть деструктор, разбивающий сконструированное значение на аргументы использованные в конструкторе.

$$\frac{\Gamma \vdash [\alpha]\texttt{constr}[t]}{\Gamma \vdash [t]\texttt{constr}^{-1}[\alpha]}$$

Стандартные операторы

- Dup - дублировать верхнее значение на стеке

$$\frac{}{\Gamma \vdash [a]\mathsf{dup}[a, a]}$$

- Pop - удалить верхнее значение со стека

$$\frac{}{\Gamma \vdash [a]\mathsf{pop}[]}$$

- Bury - переместить самый верхний элемент на n позицию в стеке

$$\frac{||\alpha|| = n}{\Gamma \vdash [b, \alpha]\mathsf{br\text{-}n}[\alpha, b]}$$

- Dig - переместить элемент с позиции n на верх стека

$$\frac{||\alpha|| = n}{\Gamma \vdash [\alpha, b]\mathsf{dg\text{-}n}[b, \alpha]}$$

Пример доказательства (композиция с мономорфными типами)

```
define [] example [Baz, Bar, Foo]:
  foo bar baz.
```

Композиция foo и bar

$$\dfrac{\dfrac{[]\texttt{foo}[\text{Foo}] \in \Gamma}{\Gamma \vdash []\texttt{foo}[\text{Foo}]}\,\text{Name} \qquad \dfrac{\dfrac{[]\texttt{bar}[\text{Bar}] \in \Gamma}{\Gamma \vdash []\texttt{bar}[\text{Bar}]}\,\text{Name} \qquad \dfrac{[][\text{Bar}] = [][\text{Bar}]}{[][\text{Bar}] \sqsupseteq [][\text{Bar}]}\,\text{OpSpec}}{\Gamma \vdash [\text{Foo}]\texttt{bar}[\text{Bar}, \text{Foo}]}\,\text{SpecExt} \qquad [\text{Foo}] = [\text{Foo}]}{\Gamma \vdash []\texttt{foo bar}[\text{Bar}, \text{Foo}]}\,\text{Chain}$$

Композиция foo bar и baz

$$\dfrac{\Gamma \vdash []\texttt{foo bar}[\text{Bar}, \text{Foo}] \qquad \dfrac{\dfrac{[]\texttt{baz}[\text{Baz}] \in \Gamma}{\Gamma \vdash []\texttt{baz}[\text{Baz}]}\,\text{Name} \qquad \dfrac{[][\text{Baz}] = [][\text{Baz}]}{[][\text{Baz}] \sqsupseteq [][\text{Baz}]}\,\text{OpSpec}}{\Gamma \vdash [\text{Bar}, \text{Foo}]\texttt{baz}[\text{Baz}, \text{Bar}, \text{Foo}]}\,\text{SpecExt} \qquad [\text{Bar}, \text{Foo}] = [\text{Bar}, \text{Foo}]}{\Gamma \vdash []\texttt{foo bar baz}[\text{Baz}, \text{Bar}, \text{Foo}]}\,\text{Chain}$$

Пример доказательства (композиция с типами переменных)

```
define [] example [Foo, Foo, Foo]:
  foo dup dup.
```

Композиция foo и dup

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash [a]\mathsf{dup}[a,a]}\,\mathsf{Dup} \quad \dfrac{[\mathsf{Foo}][\mathsf{Foo},\mathsf{Foo}] = \{a \mapsto \mathsf{Foo}\}[a][a,a]}{[\mathsf{Foo}][\mathsf{Foo},\mathsf{Foo}] \sqsupseteq [a][a,a]}\,\mathsf{OpSpec}}{\Gamma \vdash [\mathsf{Foo}]\mathsf{dup}[\mathsf{Foo},\mathsf{Foo}]}\,\mathsf{SpecExt} \quad \dfrac{[]\mathtt{foo}[\mathsf{Foo}] \in \Gamma}{\Gamma \vdash []\mathtt{foo}[\mathsf{Foo}]}\,\mathsf{Name} \quad [\mathsf{Foo}] = [\mathsf{Foo}]}{\Gamma \vdash []\mathtt{foo}\ \ \mathsf{dup}[\mathsf{Foo},\mathsf{Foo}]}$$

Композиция foo dup и dup

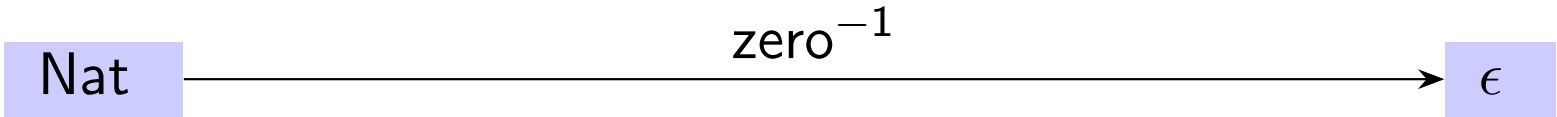
$$\dfrac{\Gamma \vdash []\mathtt{foo}\ \ \mathsf{dup}[\mathsf{Foo},\mathsf{Foo}] \quad \dfrac{\dfrac{\dfrac{}{\Gamma \vdash [a]\mathsf{dup}[a,a]}\,\mathsf{Dup} \quad \dfrac{[\mathsf{Foo}][\mathsf{Foo},\mathsf{Foo}] = \{a \mapsto \mathsf{Foo}\}[a][a,a]}{[\mathsf{Foo}][\mathsf{Foo},\mathsf{Foo}] \sqsupseteq [a][a,a]}\,\mathsf{OpSpec}}{\Gamma \vdash [\mathsf{Foo},\mathsf{Foo}]\mathsf{dup}[\mathsf{Foo},\mathsf{Foo},\mathsf{Foo}]}\,\mathsf{SpecExt} \quad [\mathsf{Foo},\mathsf{Foo}] = [\mathsf{Foo},\mathsf{Foo}]}{\Gamma \vdash []\mathtt{foo}\ \ \mathsf{dup}\ \ \mathsf{dup}[\mathsf{Foo},\mathsf{Foo},\mathsf{Foo}]}$$

Пример доказательства (case)

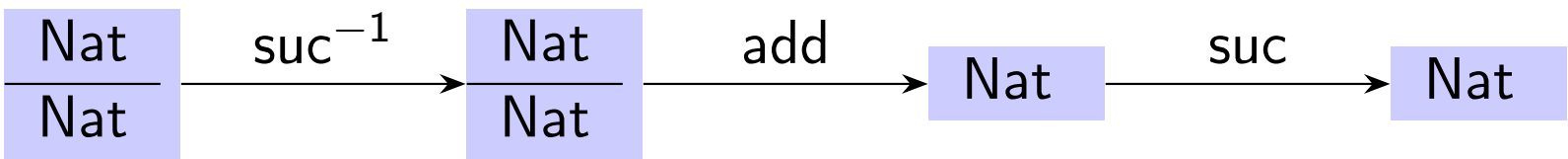
```
data Nat: zero, [Nat] suc.
define [Nat, Nat] add [Nat]:
  case { zero { },
         suc { add suc } }.
```

Диаграммы типов значений на стеке для каждого case arm:

- `zero{}`



- `suc{add suc}`

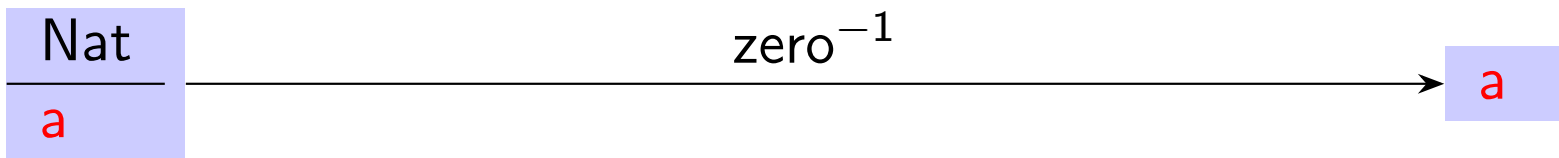


Пример доказательства (case)

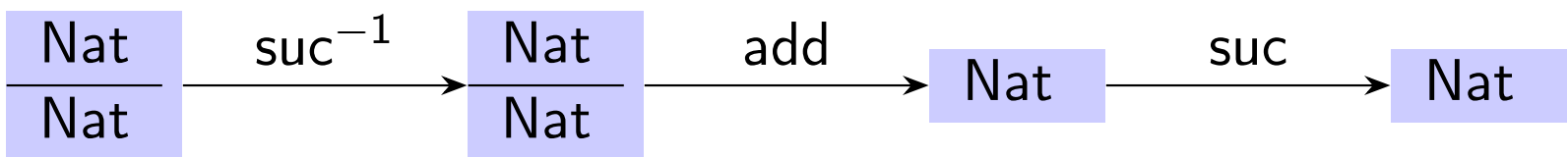
```
data Nat: zero, [Nat] suc.
define [Nat, Nat] add [Nat]:
  case { zero { },
         suc { add suc } }.
```

Диаграммы типов значений на стеке для каждого case arm:

- `zero{}`



- `suc{add suc}`



Пример доказательства (case)

- `zero{}` arm

$$\dfrac{\dfrac{\dfrac{[]\texttt{zero}[\mathsf{Nat}] \in \Gamma}{\Gamma \vdash []\texttt{zero}[\mathsf{Nat}]}\,\mathsf{Name}}{\Gamma \vdash [\mathsf{Nat}]\texttt{zero}^{-1}[]}\,\mathsf{Destr} \qquad \dfrac{[\mathsf{Nat}][] = [\mathsf{Nat}][]}{[\mathsf{Nat}][] \sqsupseteq [\mathsf{Nat}][]}\,\mathsf{OpSpec}}{\dfrac{\Gamma \vdash [\mathsf{Nat}, a]\texttt{zero}^{-1}[a]}{\Gamma \vdash [\mathsf{Nat}, a]\texttt{zero}\{\}[a]}\,\mathsf{CaseArm}}\,\mathsf{SpecExt}$$

Пример доказательства (case)

- SpecExt of a destructor suc^{-1}

$$\dfrac{\dfrac{[\mathsf{Nat}]\mathsf{suc}[\mathsf{Nat}] \in \Gamma}{\Gamma \vdash [\mathsf{Nat}]\mathsf{suc}[\mathsf{Nat}]}\,\mathsf{Name} \quad \dfrac{[\mathsf{Nat}][\mathsf{Nat}] = [\mathsf{Nat}][\mathsf{Nat}]}{[\mathsf{Nat}][\mathsf{Nat}] \sqsupseteq [\mathsf{Nat}][\mathsf{Nat}]}\,\mathsf{OpSpec}}{\Gamma \vdash [\mathsf{Nat},\mathsf{Nat}]\mathsf{suc}^{-1}[\mathsf{Nat},\mathsf{Nat}]}\,\mathsf{SpecExt}$$

- $\mathsf{suc}\{\mathsf{add}\ \ \mathsf{suc}\}$ arm

$$\dfrac{\Gamma \vdash [\mathsf{Nat},\mathsf{Nat}]\mathsf{suc}^{-1}[\mathsf{Nat},\mathsf{Nat}] \quad \dfrac{\dfrac{\cdots}{\Gamma \vdash [\mathsf{Nat},\mathsf{Nat}]\mathsf{add}\ \ \mathsf{suc}[\mathsf{Nat}]}\,\mathsf{Chain} \quad [\mathsf{Nat},\mathsf{Nat}] = [\mathsf{Nat},\mathsf{Nat}]}{\Gamma \vdash [\mathsf{Nat},\mathsf{Nat}]\mathsf{suc}^{-1}\ \ \mathsf{add}\ \ \mathsf{suc}[\mathsf{Nat}]}\,\mathsf{Chain}}{\Gamma \vdash [\mathsf{Nat},\mathsf{Nat}]\mathsf{suc}\{\mathsf{add}\ \ \mathsf{suc}\}[\mathsf{Nat}]}\,\mathsf{CaseArm}$$

Пример доказательства (case)

- $\{\mathtt{zero}, \mathtt{suc}\} = \mathsf{constrs}(\mathsf{Nat})$
- $\Gamma \vdash [\mathsf{Nat}, a]\mathtt{zero}\{\}[a]$

  $\Gamma \vdash [\mathsf{Nat}, \mathsf{Nat}]\mathtt{suc}\{\mathtt{add}\ \ \mathtt{suc}\}[\mathsf{Nat}]$
- $$\frac{[\mathsf{Nat}, \mathsf{Nat}][\mathsf{Nat}] = [\mathsf{Nat}, \mathsf{Nat}][\mathsf{Nat}]}{[\mathsf{Nat}, \mathsf{Nat}][\mathsf{Nat}] \sqsupseteq [\mathsf{Nat}, \mathsf{Nat}][\mathsf{Nat}]}$$
- $$\frac{[\mathsf{Nat}, \mathsf{Nat}][\mathsf{Nat}] = \{a \mapsto \mathsf{Nat}\}[\mathsf{Nat}, a][a]}{[\mathsf{Nat}, \mathsf{Nat}][\mathsf{Nat}] \sqsupseteq [\mathsf{Nat}, a][a]}$$

$\Gamma \vdash [\mathsf{Nat}, \mathsf{Nat}]\mathtt{case}\{\ldots\}[\mathsf{Nat}]$

Ассоциативность процесса выведения типов

Ассоциативность процесса выведения типов

Все правила выведения типов в IV(++) независимы от контекста, в котором они используются.

- В языке все операторы должны иметь явную аннотацию типов, каждая аннотация типа оператора проверяется на соответствие выведенной аннотации.
- Замены в одной последовательности операторов не влияют на другие последовательности
- Специализация применяет замены локально
- Chain rule влияет только на два соседних оператора.

Ассоциативность позволяет

- Тип любых двух частей программы могут быть проверены параллельно.
- Инкрементальная проверка типов по мере изменения программы

Ассоциативность вывода типов и поддержка ФПК

Есть множество способов добавления поддержки ФПК в систему типов языка. Какие-то способы сохраняют свойство ассоциативности, некоторые нет.

Добавление поддержки ФПК

Функции первого класса

Для поддержки ФПК необходимо следующее

- Захват переменных
- Создание ФПК
- Выполнение ФПК
- Композиция ФПК

Функции первого класса

В IV++

- Захват переменных `quote`
- Создание ФПК `(foo bar baz)`
- Выполнение ФПК `exec-x-y`
- Композиция ФПК `comp-x-y-z-w`

Quote rule

$$\frac{}{\Gamma \vdash [a]\texttt{quote}[[][a]]}$$

Lambda rule

$$\frac{\Gamma \vdash [\alpha]\texttt{op1 op2 ...}[\beta]}{\Gamma \vdash []\texttt{(op1 op2 ...)}[[\alpha][\beta]]}$$

Exec rule

$$\frac{||\alpha|| = x \qquad ||\beta|| = y}{\Gamma \vdash [[\alpha][\beta], \alpha]\texttt{exec-x-y}[\beta]}$$

Comp rule

Правило для композиции ФПК разделено на два правила.

- левый оператор возвращает больше чем правый принимает

$$\frac{||\alpha|| = x \qquad ||\beta \cdot \gamma|| = y \qquad ||\beta|| = z \qquad ||\omega|| = w}{\Gamma \vdash [[\alpha][\beta \cdot \gamma], [\beta][\omega]]\texttt{comp-x-y-z-w}[[\alpha][\omega \cdot \gamma]]}$$

- левый оператор возвращает меньше чем правый принимает

$$\frac{||\alpha|| = x \qquad ||\beta|| = y \qquad ||\beta \cdot \gamma|| = z \qquad ||\omega|| = w}{\Gamma \vdash [[\alpha][\beta], [\beta \cdot \gamma][\omega]]\texttt{comp-x-y-z-w}[[\alpha \cdot \gamma][\omega]]}$$

Где

$$\alpha = [a_1, a_2, ..., a_{|\alpha|}] \qquad \beta = [b_1, b_2, ..., b_{|\beta|}],$$
$$\gamma = [c_1, c_2, ..., c_{|\gamma|}] \qquad \omega = [d_1, d_2, ..., d_{|\omega|}]$$

Сложности с добавлением поддержки ФПК

Операторы связанные с ФПК могут иметь разные типы операторов в зависимости от типов элементов на стеке

- `exec` может принимать и возвращать разное кол-во аргументов в зависимости от типа ФПК которая выполняется
  Пример: `exec` на [a, b] `foo` [c, d, e] берет $1 + 2$ арг и возвращает 3 арг.
- `comp` разные типы операторов на входе и выходе, в зависимости от двух входных ФПК.
- `(foo bar baz)` ...
- `quote` ...

Разные способы добавления ФПК

- Параметрический - все операторы связанные с ФПК принимают дополнительные параметры - натуральные числа в своем имени. `comp-1-2-3-4`; ассоциативность.
- Отслеживание состояния - следить за тем, какие типы данных хранятся на стеке по мере вывода типа; нет ассоциативности.
- Переменные типа разной длины - специальные переменные типа которые могут стоять в конце стека и обозначают список типов; нет ассоциативности.
- Кортежи - все ФПК функции принимают одно значение и возвращают одно значение; ассоциативность.

Разные способы добавления ФПК

- Параметрический - все операторы связанные с ФПК принимают дополнительные параметры - натуральные числа в своем имени. `comp-1-2-3-4` Требует явно указывать параметры при каждом выводе
- Отслеживание состояния - следить за тем, какие типы данных хранятся на стеке по мере вывода типа. Сложно вывести тип оператора без знаний об изначальном состоянии стека. Пример: `(foo bar)`
- Переменные типа разной длины - специальные переменные типа которые могут стоять в конце стека и обозначают список типов. Самый сложный вариант, требует rank-n полиморфную систему типов
- Кортежи - все ФПК функции принимают одно значение и возвращают одно значение. При работе с ФПК программист все равно должен использовать параметрические операторы по типу pack-n и unpack-n для запаковки и распаковки значений

Существующие системы типов

- **Cat** - переменные типа разной длины.
- Kitten - отслеживание состояния.

Проблема с Cat

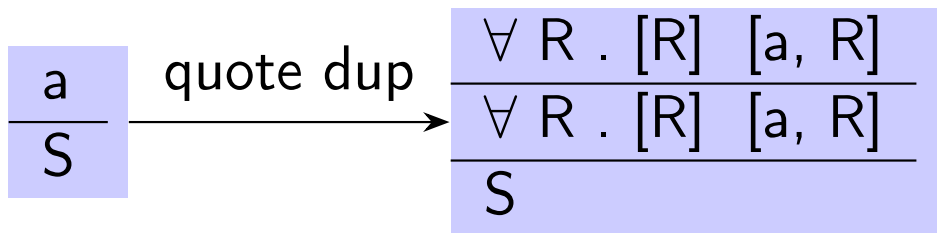
- Очень сложная система типов.
- Для того, чтобы вызывать операторы в разных стеках, типы операторов имеют переменные типа разной длины в конце стеков.
- Система типов Cat не работает без rank-n полиморфизма.

Аннотации операторов ФПК в Cat

```
quote:
  !S!a.((a S)->(!R.(R->(a R))S))
dup:
  !S!a.((a S)->(a (a S)))
quote dup:
  !S!a.((a S)->(!R.(R->a R)(!R.(R->a R)S)))
```

Обратите внимание на кванторы всеобщности ! в типах. Вложенные кванторы необходимы для того чтобы вызывать операторы в разных стеках и работать с ФПК.

Проблема с Cat



Проблема с Cat

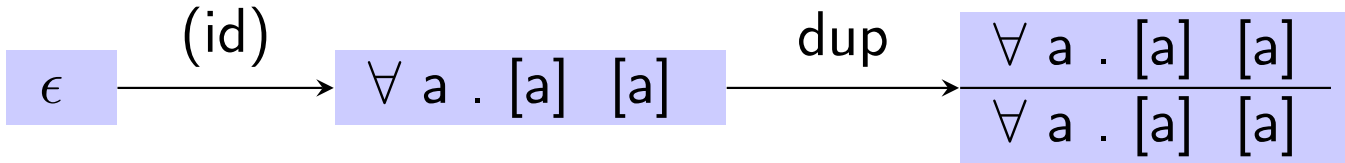
Система типов IV++ способна описать следующее без необходимости в rank-n полиморфизме.

```
[a] quote dup [[][a], [][a]]
```

Позволяет выполнить обе ФПК в разных стеках при помощи дополнения типов оператора.

Отсутствие rank-n полиморфизма в IV++

Несмотря на то, что система типов IV++ способна описать предыдущую проблему, система типов IV++ не rank-n полиморфная.
Следующее не может быть описанно в системе типов IV++.



Результат

- Был создан язык программирования и собственная система типов с формализованными правилами вывода типов.
- Была создана имплементация языка на Rust с проверкой типов и интерпретацией.

Спасибо за внимание

Extra slides: Implementation

Body inference

```
infer(ops : [Op]) -> OpType
  acc = nop
  for op in ops
    t = infer_single op
    acc = chain t acc
  acc
```

or

```
infer(ops : [Op]) -> OpType
  foldl chain nop (map infer_single acc)
```

Most General Unifier of a list

```
listmgu(a : [Type], b : [Type]) -> Subst
  case a, b of
    [], _ -> empty,
    _, [] -> empty,
    (x::xs), (y::ys) ->
      s = mgu x y
      substCompose
        (listmgu (s `apply` xs) (s `apply` ys))
        s
```

Chain

```
chain(lhs : OpType, rhs : OpType) -> OpType
  s = listmgu lhs.post rhs.pre
  if lhs.post >= rhs.pre then
    -- overflow & exact chain
    tail = skip_n lhs.post (len rhs.pre)
    s `apply` OpType (lhs.pre) (rhs.post <> tail)
  else
    -- underflow chain
    tail = skip_n rhs.pre (len lhs.post)
    s `apply` OpType (lhs.pre <> tail) (rhs.post)
```

Infer single

```
infer_single(op : Op) -> OpType
  case op of
    Name name -> lookup_optype name
    Quote ops -> OpType [] [infer ops]
    Case arms -> infer_case arms
```

Infer case

```
infer_arm(arm: (Constr, [Op])) -> OpType
  destr constr `chain` body

infer_case(arms : [(Constr, [Op])]) -> OpType
  totality_check arms
  (head_arm::rest_arms) = arms
  head_ot = infer_arm head_arm
  for arm in rest_arms
    arm_ot = infer_arm arm
    head_ot = head_ot `extend_towards` arm_ot
    arm_ot = arm_ot `extend_towards` head_ot
    s = head_ot `subst_to_match` arm_ot
    head_ot = s `apply` head_ot
  head_ot
```