ТЕОРІЯ МОВ ПРОГРАМУВАННЯ

Лекція 4

Мова PCF (продовження)

Стратегії редукції
Поняття стратегії

Поскольку каждый PCF-терм может иметь не более одного результата (благодаря свойству, отмеченному выше), неважно, в каком порядке вычислять редексы, входящие в терм: если мы придём к нередуцируемому терму, то он всегда будет одним и тем же. Однако возможна ситуация, при которой одна последовательность редукций приводит к нередуцируемому терму, а другая нет. Пусть, для примера, `C` является термом вида **fun** `x` $\to$ `0`, а b_1 вида (**fix** `x` (**fun** `x` $\to$ (`f x`))) `0`. Терм b_1 сводится к b_2 = (**fun** `x` $\to$ (**fix** `f` (**fun** `x` $\to$ (`f x`)) `x`)) `0`, а затем снова к b_1. Терм `C` b_1 содержит несколько редексов и может быть сведён либо к `0`, либо к `C` b_2, который в свою очередь также содержит несколько редексов и может быть приведён либо к `0`, либо к `C` b_1 (а также и к другим термам). Редуцируя всякий раз самый внутренний редекс, можно получить бесконечную последовательность редукций `c` $b_1 \rhd$ `C` $b_2 \rhd$ `C` $b_1 \rhd \cdots$, тогда как редуцированием внешнего редекса всегда можно получить `0`.

Этот пример может показаться исключением, ведь он содержит функцию `C`, которая игнорирует свой аргумент; заметьте, однако, что условная конструкция **`ifz`** ведёт себя схожим образом. При вычислении факториала числа `3` можно наблюдать аналогичное поведение. В терме

ifz `0` **then** `1` **else** `0*((`**fix** `f` **fun** `n` $\to$ **ifz** `n` **then** `1` **else** `n*(f(n-1)))(0-1))`

имеется несколько редексов. При выборе внешнего редекса получаем `1` (остальные редексы при этом пропадают), тогда как редуцируя

fix `f` **fun** `n` $\to$ **ifz** `n` **then** `1` **else** `n*(f(n-1))`

получаем бесконечную последовательность редукций. Другими словами, терм `fact 3` можно редуцировать к `6`, но можно и получить вычисления, продолжающиеся вечно.

Оба терма `C` $\mathtt{b}_1$ и `fact 3` дают единственный результат, но не все последовательности редукций позволяют к нему прийти.

Поскольку терм `C` $\mathtt{b}_1$ имеет значение `0`, согласно семантике PCF *вычислитель*, то есть программа, которая принимает на вход PCF-терм и возвращает его значение, должна при вычислении `C` $\mathtt{b}_1$ вернуть `0`. Попробуем воспользоваться несколькими современными компиляторами для того, чтобы провести такие вычисления. В языке Caml программа

```
let rec f x = f x in let g x = 0 in g (f 0)
```

не завершается. Та же проблема и с программой на Java:

```
class Omega {
  static int f (int x) {return f(x);}
  static int g (int x) {return 0;}
  static public void main (String[] args) {
    System.out.println(g(f(0)));
  }
}
```

И только очень небольшое число компиляторов для языков, использующих *вызов по имени* или *ленивые* вычисления, таких как Haskell, Lazy-ML или Gaml, позволяют получить завершающуюся программу, которая сможет вычислить результат этого терма.

Причина проблем в том, что семантика PCF с малым шагом не соответствует семантике Java или Caml. Фактически она является слишком общей, при наличии нескольких редексов она не указывает, какой именно редекс должен быть редуцирован. По умолчанию она рассчитывает на завершение всех программ, результат которых в принципе может быть вычислен. В определении этой семантики отсутствует важный компонент: понятие стратегии, определяющей порядок выбора редексов.

Стратегией называется частичная функция, которая каждому терму из своей области определения ставит в соответствие один из его редексов. Имея стратегию $\mathtt{s}$, можно определить другую семантику, заменив отношение $\triangleright$ на новое отношение $\triangleright_{\mathtt{s}}$, определяемое следующим: $\mathtt{t} \triangleright_{\mathtt{s}} \mathtt{u}$, если определён $\mathtt{s}\ \mathtt{t}$ и терм $\mathtt{u}$ получен вычислением редекса $\mathtt{s}\ \mathtt{t}$ в $\mathtt{t}$. Теперь можно определить отношение $\triangleright_{\mathtt{s}}^{*}$ как рефлексивно-транзитивное замыкание отношения $\triangleright_{\mathtt{s}}$, а также отношение $\hookrightarrow_{\mathtt{s}}$ указанным выше способом.

Альтернативой определению стратегии может быть такое ослабление правил редукции, в особенности правила эквивалентности, чтобы можно было вычислять только некоторые специальные редексы.

Слабка редукція

Перед определением различных стратегий вычисления терма `C b1` рассмотрим ещё один пример, показывающий слишком большую свободу определённой выше операционной семантики и мотивирующий введение стратегий или ослабление правил редукции. Применим программу **fun** `x` $\rightarrow$ `x + (4 + 5)` к константе `3`. Получим терм (**fun** `x` $\rightarrow$ `x + (4 + 5)`) `3`, содержащий два редекса. Возможны два варианта редукции: `3 + (4 + 5)` и (**fun** `x` $\rightarrow$ `x + 9`) `3`. Первый вариант соответствует исполнению программы, а второй нет. Обычно говорят, что при вычислении тела функции до передачи ей аргументов происходит *оптимизация* или *специализация* программы.

Слабая стратегия редукции никогда не вычисляет редекс, находящийся внутри **fun**. Таким образом, слабая редукция не специализирует программы, она их лишь исполняет. Отсюда следует, что при слабой стратегии все термы вида $\textbf{fun}\ \mathtt{x} \to \mathtt{t}$ являются нередуцируемыми.

Другой способ определения слабой редукции заключается в ослаблении правил редукции или, точнее, в удалении правила

$$\frac{\mathtt{t} \rhd \mathtt{u}}{\textbf{fun}\ \mathtt{x} \to \mathtt{t} \rhd \textbf{fun}\ \mathtt{x} \to \mathtt{u}}.$$

Виклик за ім’ям

Снова проанализируем редукции, возможные в терме `C` $\mathtt{b}_1$. Необходимо решить, нужно ли вычислять аргументы функции `C` до того, как они будут переданы функции, или же их следует передавать функции, не вычисляя.

Стратегия *вызова по имени* требует всякий раз выбирать самый левый редекс, тогда как слабая стратегия вызова по имени требует выбирать самый левый редекс, не входящий в **fun**. Таким образом, `C` $\mathtt{b}_1$ сводится к `0`.

Эта стратегия интересна благодаря следующему свойству, называемому *полнотой*: если терм можно привести к нередуцируемому терму, то редукция с вызовом по имени всегда завершается. Другими словами, $\hookrightarrow_{\mathtt{n}} = \hookrightarrow$. Более того, при вычислении терма `(`**fun** `x` $\to$ `0) (fact 10)` с использованием стратегии вызова по имени, нам не потребуется вычислять факториал `10`. Правда, при вычислении значения терма `(`**fun** `x` $\to$ `x+x) (fact 10)` с той же стратегией, его нужно будет вычислить дважды, поскольку результатом одного шага редукции будет терм `(fact 10) + (fact 10)`. Большинство вычислителей с вызовом по имени стараются в таких случаях избегать дублирования вычислений, запоминая результат первого вычисления. Подобный способ вычислений называется *ленивым*.

Виклик за значеннями

Вызов по значению, наоборот, предполагает вычисление аргументов до их передачи функции. Он основан на следующем соглашении: терм вида `(`**fun** `x` $\to$ `t) u` редуцируется только в том случае, если `u` является значением. Таким образом, при вычислении `(`**fun** `x` $\to$ `x + x) (fact 10)` нужно начинать с редуцирования аргумента, получая `(`**fun** `x` $\to$ `x + x) 3628800`, а затем уже вычислять самый левый редекс. Это позволит вычислять факториал `10` только один раз.

К этому классу относятся все стратегии, требующие вычисления значения аргументов до их передачи. Такова, к примеру, стратегия вычисления самого левого редекса среди допустимых. Так что вызов по значению является не конкретной стратегией, а целым семейством.

Это соглашение может быть определено и ослаблением правила β-редукции: терм `(`**fun** `x` $\to$ `t) u` считается редексом, только если терм `u` является значением.

Говорят, что слабая стратегия реализует вызов по значению, если она редуцирует термы вида `(`**fun** `x` $\to$ `t) u`, только если `u` является значением и не находится внутри **fun**.

Лінь — двигун прогресу

Даже в случае реализации стратегии вызова по значению условное выражение `ifz` должно вычисляться с вызовом по имени: в терме вида **ifz** `t` **then** `u` **else** `v` никогда не следует вычислять все три аргумента. Вместо этого сначала необходимо вычислить `t`, а затем в зависимости от результата вычислить либо `u`, либо `v`.

Легко видеть, что при вычислении всех трёх аргументов `ifz` вычисление значения `fact 3` не завершается.

Операційна семантика з великим кроком

Вместо того, чтобы определять стратегию или ослаблять редукционные правила операционной семантики с малым шагом, можно контролировать порядок вычисления редексов, определив операционную семантику *с большим шагом*.

Операционная семантика с большим шагом для языка программирования даёт индуктивное определение отношения $\hookrightarrow$ без предварительного определения $\longrightarrow$ и $\triangleright$.

Виклик за ім’ям

Начнём с семантики вызова по имени. Рассмотрим терм вида $\mathtt{t\ u}$, который с учётом вызова по имени редуцируется к терму $\mathtt{V}$. Будем вычислять те редексы, которые содержатся в $\mathtt{t}$, пока не получим нередуцируемый терм. Если это терм вида $\mathbf{fun}\ \mathtt{x} \to \mathtt{t}'$, то исходный терм редуцируется к терму $(\mathbf{fun}\ \mathtt{x} \to \mathtt{t}')\ \mathtt{u}$, и самым левым редексом оказывается весь терм целиком. Он вычисляется как $\mathtt{(u/x)t}'$, что в свою очередь редуцируется к $\mathtt{V}$. Можно говорить, что терм $\mathtt{t\ u}$ сводится при вызове по имени к нередуцируемому терму $\mathtt{V}$, если $\mathtt{t}$ редуцируется к $\mathbf{fun}\ \mathtt{x} \to \mathtt{t}'$, а $\mathtt{(u/x)t}'$ к $\mathtt{V}$.

Все эти рассуждения можно выразить правилом:

$$\frac{\mathtt{t} \hookrightarrow \mathbf{fun}\ \mathtt{x} \to \mathtt{t}' \qquad \mathtt{(u/x)t}' \hookrightarrow \mathtt{V}}{\mathtt{t\ u} \hookrightarrow \mathtt{V}},$$

которое будет частью индуктивного определения отношения $\hookrightarrow$ (без предварительного определения $\longrightarrow$ и $\triangleright$).

Другие правила утверждают, что результатом вычисления терма вида **fun** является сам терм, то есть здесь мы определяем отношение слабой редукции:

$$\frac{}{\textbf{fun}\ \mathtt{x} \to \mathtt{t} \hookrightarrow \textbf{fun}\ \mathtt{x} \to \mathtt{t}},$$

и что результатом вычисления терма `n` является сам терм:

$$\frac{}{\mathtt{n} \hookrightarrow \mathtt{n}}.$$

Также есть правило для задания семантики арифметических операций:

$$\frac{\mathtt{u} \hookrightarrow \mathtt{q} \qquad \mathtt{t} \hookrightarrow \mathtt{p}}{\mathtt{t} \otimes \mathtt{u} \hookrightarrow \mathtt{n}} \text{ если } \mathtt{p} \otimes \mathtt{q} = \mathtt{n};$$

два правила для задания семантики условной конструкции `ifz`:

$$\frac{\mathtt{t} \hookrightarrow \mathtt{0} \qquad \mathtt{u} \hookrightarrow \mathtt{V}}{\textbf{ifz } \mathtt{t} \textbf{ then } \mathtt{u} \textbf{ else } \mathtt{v} \hookrightarrow \mathtt{V}},$$

$$\frac{\mathtt{t} \hookrightarrow \mathtt{n} \qquad \mathtt{v} \hookrightarrow \mathtt{V}}{\textbf{ifz } \mathtt{t} \textbf{ then } \mathtt{u} \textbf{ else } \mathtt{v} \hookrightarrow \mathtt{V}} \text{ если } \mathtt{n} \text{ — число, } \mathtt{n} \neq \mathtt{0};$$

правило для оператора неподвижной точки:

$$\frac{\texttt{(}\textbf{fix}\ \texttt{x t/x)t} \hookrightarrow \texttt{V}}{\textbf{fix}\ \texttt{x t} \hookrightarrow \texttt{V}};$$

и, наконец, правило, определяющее семантику **let**:

$$\frac{\texttt{(t/x) u} \hookrightarrow \texttt{V}}{\textbf{let}\ \texttt{x = t}\ \textbf{in}\ \texttt{u} \hookrightarrow \texttt{V}}.$$

С помощью структурной индукции по отношению вычисления можно доказать, что результат вычисления терма всегда является значением, то есть либо числом, либо замкнутым термом вида **fun**. Тупиковых термов нет. Вычисление терма `((`**fun** `x` $\to$ `x) 1) 2`, которое в семантике с малым шагом давало тупиковый терм `1 2`, теперь не даёт никакого результата, поскольку ни одно из правил не может быть к нему применено. Действительно, в семантике с большим шагом нет правила, которое объясняло бы как вычислить применение, левая часть которого сводится к числу.

Виклик за значеннями

Правила, определяющие семантику с вызовом по значению, довольно похожи, за исключением правила для применения: необходимо вычислить значение аргумента до передачи его в функцию:

$$\frac{\mathtt{u} \hookrightarrow \mathtt{W} \qquad \mathtt{t} \hookrightarrow \mathbf{fun}\ \mathtt{x} \to \mathtt{t}' \qquad (\mathtt{W}/\mathtt{x})\mathtt{t}' \hookrightarrow \mathtt{V}}{\mathtt{t}\ \mathtt{u} \hookrightarrow \mathtt{V}},$$

и правила для **let**:

$$\frac{\mathtt{t} \hookrightarrow \mathtt{W} \qquad (\mathtt{W}/\mathtt{x})\mathtt{u} \hookrightarrow \mathtt{V}}{\mathbf{let}\ \mathtt{x} = \mathtt{t}\ \mathbf{in}\ \mathtt{u} \hookrightarrow \mathtt{V}}.$$

Подытоживая, получаем следующие правила:

$$\frac{\mathtt{u} \hookrightarrow \mathtt{W} \qquad \mathtt{t} \hookrightarrow \mathbf{fun}\ \mathtt{x} \to \mathtt{t}' \qquad (\mathtt{W}/\mathtt{x})\mathtt{t}' \hookrightarrow \mathtt{V}}{\mathtt{t}\ \mathtt{u} \hookrightarrow \mathtt{V}},$$

$$\frac{}{\mathbf{fun}\ \mathtt{x} \to \mathtt{t} \hookrightarrow \mathbf{fun}\ \mathtt{x} \to \mathtt{t}},$$

$$\frac{}{\mathtt{n} \hookrightarrow \mathtt{n}},$$

$$\frac{\mathtt{u} \hookrightarrow \mathtt{q} \qquad \mathtt{t} \hookrightarrow \mathtt{p}}{\mathtt{t} \otimes \mathtt{u} \hookrightarrow \mathtt{n}} \text{ если } \mathtt{p} \otimes \mathtt{q} = \mathtt{n},$$

$$\frac{\mathtt{t} \hookrightarrow \mathtt{n} \qquad \mathtt{v} \hookrightarrow \mathtt{V}}{\textbf{ifz } \mathtt{t} \textbf{ then } \mathtt{u} \textbf{ else } \mathtt{v} \hookrightarrow \mathtt{V}} \text{ если } \mathtt{n} \text{ — число, } \mathtt{n} \neq 0,$$

$$\frac{(\textbf{fix } \mathtt{x}\ \mathtt{t}/\mathtt{x})\mathtt{t} \hookrightarrow \mathtt{V}}{\textbf{fix } \mathtt{x}\ \mathtt{t} \hookrightarrow \mathtt{V}},$$

$$\frac{\mathtt{t} \hookrightarrow \mathtt{W} \qquad (\mathtt{W}/\mathtt{x})\mathtt{u} \hookrightarrow \mathtt{V}}{\textbf{let } \mathtt{x} = \mathtt{t} \textbf{ in } \mathtt{u} \hookrightarrow \mathtt{V}}.$$

Заметим, что даже при вызове по значению правила для `ifz` сохраняются:

$$\frac{\texttt{t} \hookrightarrow \texttt{0} \qquad \texttt{u} \hookrightarrow \texttt{V}}{\textbf{ifz}\ \texttt{t}\ \textbf{then}\ \texttt{u}\ \textbf{else}\ \texttt{v} \hookrightarrow \texttt{V}},$$

$$\frac{\texttt{t} \hookrightarrow \texttt{n} \qquad \texttt{v} \hookrightarrow \texttt{V}}{\textbf{ifz}\ \texttt{t}\ \textbf{then}\ \texttt{u}\ \textbf{else}\ \texttt{v} \hookrightarrow \texttt{V}} \text{ если } \texttt{n} \text{ — число, } \texttt{n} \neq \texttt{0},$$

то есть второй и третий аргументы `ifz` не вычисляются до тех пор, пока они не потребуются.

Заметим также, что при вызове по значению сохраняется и правило для **fix**:

$$\frac{(\mathbf{fix}\ \mathtt{x}\ \mathtt{t}/\mathtt{x})\mathtt{t} \hookrightarrow \mathtt{V}}{\mathbf{fix}\ \mathtt{x}\ \mathtt{t} \hookrightarrow \mathtt{V}}.$$

Необходимо устоять перед искушением вычислить терм **fix** `x t` до значения `W` перед его подстановкой в `t`, так как правило

$$\frac{\textbf{fix}\ \texttt{x}\ \texttt{t} \hookrightarrow \texttt{W} \qquad (\texttt{W/x})\texttt{t} \hookrightarrow \texttt{V}}{\textbf{fix}\ \texttt{x}\ \texttt{t} \hookrightarrow \texttt{V}}$$

требует для вычисления **fix** `x t` начать с вычисления **fix** `x t`, что приведёт к появлению цикла, и терм `fact 3` никогда не вычислится до значения — его редуцирование запустит бесконечный вычислительный процесс.

Заметим, наконец, что возможны и другие комбинации правил. Например, некоторые варианты семантики с вызовом по имени используют в правиле **let** вызов по значению.

Обчислення PCF-програм

Вычислителем PCF называется программа, которая принимает замкнутый PCF-терм и возвращает его значение. При чтении снизу вверх правила семантики с большим шагом можно рассматривать как ядро такого вычислителя: для вычисления применения `t u` нужно начать с вычисления `u` и `t` и т. д. Это легко выразить в языке типа Caml:

```
let rec eval p = match p with
  | App(t,u) -> let w = eval u
                in let v = eval t
                in ...
  | ...
```

В случае с применением правила операционной семантики с большим шагом оставляют нам некоторую свободу в порядке вычисления термов `t` или `u` — вызов по значению является не стратегией, а семейством стратегий — однако терм $(W/x)t'$ в любом случае должен вычисляться третьим, поскольку он строится на основе результатов вычисления первых двух.

Денотационную семантику для PCF определить сложнее. Это может показаться парадоксальным, ведь PCF является функциональным языком, а значит, его программы должны легко интерпретироваться как функции. Проблема, однако, в том, что PCF допускает применение любого объекта к любому, и ничто не мешает нам, к примеру, записать терм **fun** `x` $\to$ `(x x)`. В отличие от математических функций функции PCF не имеют области определения.

ВІД ОБЧИСЛЕННЯ ДО ІНТЕРПРЕТАЦІЇ

Виклик за ім’ям

Применяя операционную семантику с большим шагом, можно построить вычислитель для языка PCF, в котором терм вида `(`**fun** `x` $\to$ `t) u` начинает вычисляться с помощью подстановки терма `u` в тело функции `t` всюду вместо переменной `x`. Например, чтобы вычислить терм `(`**fun** `x` $\to$ `(x*x) + x) 4`, подставим `4` вместо `x` в терме `(x*x) + x`, а затем вычислим полученный терм `(4*4) + 4`. Подстановка является затратной операцией; чтобы повысить производительность вычислителя, можно было бы вместо неё хранить ассоциацию `x = 4` в отдельной структуре, называемой *окружением*, и вычислять терм `(x*x) + x` в этом окружении. Программа, вычисляющая термы таким способом, называется *интерпретатором*.

Окружение это функция с конечной областью определения, действующая из переменных в термы. По сути, это то же самое, что подстановка, но в иных обозначениях. Окружение записывается в виде списка пар $\mathtt{x}_1 = \mathtt{t}_1, \ldots, \mathtt{x}_n = \mathtt{t}_n$, причём одна и та же переменная `x` может встречаться несколько раз, и в этом случае самая правая пара имеет приоритет. Таким образом, в окружении `x = 3`, `y = 4`, `x = 5`, `z = 8` используется значение `x = 5`, но не `x = 3`, которое, как говорят, *скрыто* парой `x = 5`. Наконец, если `e` это окружение и `x = t` — пара, через `e`, `x = t` обозначается список, полученный расширением `e` с помощью `x = t`.

Во время вычисления терма можно встретить свободную переменную `x`. В этом случае происходит поиск ассоциированного с ней терма в окружении. Можно показать, что если начать с замкнутого терма, такой поиск всегда будет завершаться успехом.

Фактически ситуация несколько более сложна, так как в дополнение к терму `u`, ассоциированному с переменной в окружении, необходимо также найти окружение, ассоциированное с `u`. Пара: терм и окружение — называется *задумкой* (thunk) и будет записываться $\langle$`u, e`$\rangle$.

Аналогично при интерпретации терма вида **fun** `x` $\to$ `t` в окружении `e` результат не может быть просто термом **fun** `x` $\to$ `t`, потому что может содержать свободные переменные, и во время интерпретации терма `t` понадобятся задумки, ассоциированные с этими переменными в окружении `e`. Расширим понятие значения, добавив к числам *замыкания*, состоящие из терма, который *должен иметь вид* **fun** `x` $\to$ `t`, и окружения `e`. Будем записывать такие значения следующим образом: $\langle$`x, t, e`$\rangle$. Значения больше не являются подмножеством термов, и возникает необходимость определить язык значений независимо от языка термов.

Как следствие, придётся переписать правила операционной семантики с большим шагом при вызове по имени для PCF, чтобы ввести отношение вида $\mathtt{e} \vdash \mathtt{t} \hookrightarrow \mathtt{V}$, читается «$\mathtt{t}$ интерпретируется как $\mathtt{V}$ в окружении $\mathtt{e}$», где $\mathtt{e}$ обозначает окружение, $\mathtt{t}$ — терм, $\mathtt{V}$ — значение. Когда окружение $\mathtt{e}$ пусто, это отношение записывается так: $\vdash \mathtt{t} \hookrightarrow \mathtt{V}$. К правилам, расширяющим окружение, относятся применение, которое добавляет пару, состоящую из переменной $\mathtt{x}$ и задумки $\langle \mathtt{u}, \mathtt{e} \rangle$, правило для **let**, которое добавляет пару, состоящую из переменной $\mathtt{x}$ и задумки $\langle \mathtt{t}, \mathtt{e} \rangle$, и правило для **fix**, которое добавляет пару, состоящую из переменной $\mathtt{x}$ и задумки $\langle \mathbf{fix}\ \mathtt{x}\ \mathtt{t}, \mathtt{e} \rangle$. В последнем правиле терм $\mathtt{t}$ дублируется: одна из копий интерпретируется, а другая остаётся в окружении для использования из рекурсивных вызовов, возникающих при вычислении первой.

$$\frac{\mathtt{e}' \vdash \mathtt{t} \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \mathtt{x} \hookrightarrow \mathtt{V}} \text{ если } \mathtt{e} \text{ содержит } \mathtt{x} = \langle \mathtt{t}, \mathtt{e}' \rangle,$$

$$\frac{\mathtt{e} \vdash \mathtt{t} \hookrightarrow \langle \mathtt{x}, \mathtt{t}', \mathtt{e}' \rangle \qquad (\mathtt{e}', \mathtt{x} = \langle \mathtt{u}, \mathtt{e} \rangle) \vdash \mathtt{t}' \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \mathtt{t}\ \mathtt{u} \hookrightarrow \mathtt{V}},$$

$$\frac{}{\mathtt{e} \vdash \mathbf{fun}\ \mathtt{x} \to \mathtt{t} \hookrightarrow \langle \mathtt{x}, \mathtt{t}, \mathtt{e} \rangle},$$

$$\frac{}{\mathtt{e} \vdash \mathtt{n} \hookrightarrow \mathtt{n}},$$

$$\frac{\mathtt{e} \vdash \mathtt{u} \hookrightarrow \mathtt{q} \qquad \mathtt{e} \vdash \mathtt{t} \hookrightarrow \mathtt{p}}{\mathtt{e} \vdash \mathtt{t} \otimes \mathtt{u} \hookrightarrow \mathtt{n}} \text{ если } \mathtt{p} \otimes \mathtt{q} = \mathtt{n},$$

$$\frac{\mathtt{e} \vdash \mathtt{t} \hookrightarrow 0 \qquad \mathtt{e} \vdash \mathtt{u} \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \mathbf{ifz}\ \mathtt{t}\ \mathbf{then}\ \mathtt{u}\ \mathbf{else}\ \mathtt{v} \hookrightarrow \mathtt{V}},$$

$$\frac{\mathtt{e} \vdash \mathtt{t} \hookrightarrow \mathtt{n} \qquad \mathtt{e} \vdash \mathtt{v} \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \mathbf{ifz}\ \mathtt{t}\ \mathbf{then}\ \mathtt{u}\ \mathbf{else}\ \mathtt{v} \hookrightarrow \mathtt{V}} \text{ если } \mathtt{n} \text{ — число, } \mathtt{n} \neq 0,$$

$$\frac{(\mathtt{e}, \mathtt{x} = \langle \mathbf{fix}\ \mathtt{x}\ \mathtt{t}, \mathtt{e} \rangle) \vdash \mathtt{t} \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \mathbf{fix}\ \mathtt{x}\ \mathtt{t} \hookrightarrow \mathtt{V}},$$

$$\frac{(\mathtt{e}, \mathtt{x} = \langle \mathtt{t}, \mathtt{e} \rangle) \vdash \mathtt{u} \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \mathbf{let}\ \mathtt{x} = \mathtt{t}\ \mathbf{in}\ \mathtt{u} \hookrightarrow \mathtt{V}}.$$

Виклик за значенням

Вариант с семантикой вызова по значению проще. Действительно, при интерпретации терма `(fun x → t) u` сначала интерпретируется терм `u`. Результатом является значение, то есть число или замыкание, достаточно лишь связать переменную `x` в окружении с этим значением. Аналогично при интерпретации терма `let x = t in u` вначале интерпретируется терм `t`. Результатом является значение, и вновь достаточно связать переменную `x` в окружении с этим значением. Таким образом, окружения будут сопоставлять переменным значения, а не задумки (которые заморожены до момента интерпретации). Исчезает необходимость в понятии задумки.

Однако вычислительное правило для **fix**, в отличие от правил для применения или **let**, требует подстановки вместо переменной терма в форме **fix** `x t`, который не является значением, и попытка получения значения которого перед подстановкой или сохранением в окружении привела бы к бесконечным вычислениям (как упоминалось выше). В окружение придётся включить *оснащённые значения*, которые являются либо значениями, либо задумками, содержащими терм вида **fix** `x t` и окружение `e`. При обращении к такому оснащённому значению понадобится интерпретировать его, если оно представляет из себя задумку. Это приводит к следующему набору правил:

$$\frac{}{\texttt{e} \vdash \texttt{x} \hookrightarrow \texttt{V}} \text{ если } \texttt{e} \text{ содержит } \texttt{x = V},$$

$$\frac{\mathtt{e}' \vdash \mathbf{fix}\ \mathtt{y}\ \mathtt{t} \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \mathtt{x} \hookrightarrow \mathtt{V}}\ \text{если } \mathtt{e} \text{ содержит } \mathtt{x} = \langle \mathbf{fix}\ \mathtt{y}\ \mathtt{t}, \mathtt{e}' \rangle,$$

$$\frac{\mathtt{e} \vdash \mathtt{u} \hookrightarrow \mathtt{W} \qquad \mathtt{e} \vdash \mathtt{t} \hookrightarrow \langle \mathtt{x}, \mathtt{t}', \mathtt{e}' \rangle \qquad (\mathtt{e}', \mathtt{x} = \mathtt{W}) \vdash \mathtt{t}' \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \mathtt{t}\ \mathtt{u} \hookrightarrow \mathtt{V}},$$

$$\frac{}{\mathtt{e} \vdash \mathbf{fun}\ \mathtt{x} \to \mathtt{t} \hookrightarrow \langle \mathtt{x}, \mathtt{t}, \mathtt{e} \rangle},$$

$$\frac{}{\mathtt{e} \vdash \mathtt{n} \hookrightarrow \mathtt{n}},$$

$$\frac{\mathtt{e} \vdash \mathtt{u} \hookrightarrow \mathtt{q} \qquad \mathtt{e} \vdash \mathtt{t} \hookrightarrow \mathtt{p}}{\mathtt{e} \vdash \mathtt{t} \otimes \mathtt{u} \hookrightarrow \mathtt{n}}\ \text{если } \mathtt{p} \otimes \mathtt{q} = \mathtt{n},$$

$$\frac{\mathtt{e} \vdash \mathtt{t} \hookrightarrow \mathtt{0} \qquad \mathtt{e} \vdash \mathtt{u} \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \mathbf{ifz}\ \mathtt{t}\ \mathbf{then}\ \mathtt{u}\ \mathbf{else}\ \mathtt{v} \hookrightarrow \mathtt{V}},$$

$$\frac{\mathtt{e} \vdash \mathtt{t} \hookrightarrow \mathtt{n} \qquad \mathtt{e} \vdash \mathtt{v} \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \mathbf{ifz}\ \mathtt{t}\ \mathbf{then}\ \mathtt{u}\ \mathbf{else}\ \mathtt{v} \hookrightarrow \mathtt{V}}\ \text{если } \mathtt{n} \text{ — число, } \mathtt{n} \neq \mathtt{0},$$

$$\frac{(\mathtt{e}, \mathtt{x} = \langle \mathbf{fix}\ \mathtt{x}\ \mathtt{t}, \mathtt{e} \rangle) \vdash \mathtt{t} \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \mathbf{fix}\ \mathtt{x}\ \mathtt{t} \hookrightarrow \mathtt{V}},$$

$$\frac{\mathtt{e} \vdash \mathtt{t} \hookrightarrow \mathtt{W} \qquad (\mathtt{e}, \mathtt{x} = \mathtt{W}) \vdash \mathtt{u} \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \mathbf{let}\ \mathtt{x} = \mathtt{t}\ \mathbf{in}\ \mathtt{u} \hookrightarrow \mathtt{V}}.$$

Оптимізація: індекси де Брауна

В правилах для операционной семантики с большим шагом окружение представляет из себя список пар, состоящих из переменной и оснащённого значения. Можно заменить эту структуру на пару списков той же длины, один из которых содержит переменные, а другой — значения. Так, например, список `x = 12`, `y = 14`, `z = 16`, `w = 18`, может быть заменён на список переменных `x,y,z,w` и список оснащённых значений `12`, `14`, `16`, `18`. Для поиска оснащённого значения, соответствующего переменной, необходимо просмотреть первый список для определения позиции переменной, а затем найти в другом списке элемент на этой позиции. Позиция переменной в первом списке это число, называемое *индексом де Брауна* переменной в окружении. В общем случае можно связать число `0` с последним («самым правым») элементом списка, число `1` — с предшествующим, . . . , число `n - 1` — с первым («самым левым») элементом списка.

Список переменных, которые понадобятся для интерпретации каждого подтерма, может быть вычислен до начала процесса интерпретации. Фактически, перед тем как интерпретировать терм, можно поставить в соответствие каждому вхождению переменной её индекс де Брауна. Например, если интерпретируется терм **fun** `x` $\to$ **fun** `y` $\to$ `(x + (fun z` $\to$ **fun** `w` $\to$ `(x + y + z + w))(2 * 8)(14 + 4))(5 + 7)(20 - 6)`, переменная `y` неизбежно будет интерпретирована в окружении вида `x = .`, `y = .`, `z = .`, `w = .`, то есть, чтобы найти значение, соответствующее `y`, нужно взять значение по индексу 2. Можно приписать этот индекс переменной с самого начала.

Чтобы вычислить индексы де Брауна, нужно просто обойти весь терм, поддерживая *окружение переменных*, то есть список переменных, в котором индекс $\mathtt{p}$ приписывается переменной $\mathtt{x}$ в окружении $\mathtt{e}$, если $\mathtt{p}$ обозначает позицию переменной $\mathtt{x}$ в окружении $\mathtt{e}$, считая с конца.

— $|\mathtt{x}|_{\mathtt{e}} = \mathtt{x}^{\mathtt{p}}$, где $\mathtt{p}$ это позиция $\mathtt{x}$ в окружении $\mathtt{e}$,

— $|\mathtt{t\ u}|_{\mathtt{e}} = |\mathtt{t}|_{\mathtt{e}}\ |\mathtt{u}|_{\mathtt{e}}$,

— $|\mathbf{fun}\ \mathtt{x} \to \mathtt{t}|_{\mathtt{e}} = \mathbf{fun}\ \mathtt{x} \to |\mathtt{t}|_{\mathtt{e},\mathtt{x}}$,

— $|\mathtt{n}|_{\mathtt{e}} = \mathtt{n}$,

— $|\mathtt{t} + \mathtt{u}|_{\mathtt{e}} = |\mathtt{t}|_{\mathtt{e}} + |\mathtt{u}|_{\mathtt{e}}$,

— $|\mathtt{t} - \mathtt{u}|_{\mathtt{e}} = |\mathtt{t}|_{\mathtt{e}} - |\mathtt{u}|_{\mathtt{e}}$,

— $|\mathtt{t} * \mathtt{u}|_{\mathtt{e}} = |\mathtt{t}|_{\mathtt{e}} * |\mathtt{u}|_{\mathtt{e}}$,

— $|\mathtt{t}/\mathtt{u}|_{\mathtt{e}} = |\mathtt{t}|_{\mathtt{e}}/|\mathtt{u}|_{\mathtt{e}}$,

— $|\mathbf{ifz}\ \mathtt{t}\ \mathbf{then}\ \mathtt{u}\ \mathbf{else}\ \mathtt{v}|_{\mathtt{e}} = \mathbf{ifz}\ |\mathtt{t}|_{\mathtt{e}}\ \mathbf{then}\ |\mathtt{u}|_{\mathtt{e}}\ \mathbf{else}\ |\mathtt{v}|_{\mathtt{e}}$,

— $|\mathbf{fix}\ \mathtt{x}\ \mathtt{t}|_{\mathtt{e}} = \mathbf{fix}\ \mathtt{x}\ |\mathtt{t}|_{\mathtt{e},\mathtt{x}}$,

— $|\mathbf{let}\ \mathtt{x} = \mathtt{t}\ \mathbf{in}\ \mathtt{u}|_{\mathtt{e}} = \mathbf{let}\ \mathtt{x} = |\mathtt{t}|_{\mathtt{e}}\ \mathbf{in}\ |\mathtt{u}|_{\mathtt{e},\mathtt{x}}$.

Например, терм выше будет записан так: **fun** x $\to$ **fun** y $\to$ (x^1 + (**fun** z $\to$ **fun** w $\to$ (x^3 + y^2 + z^1 + w^0))(2 * 8)(14 + 4))(5 + 7)(20 - 6).

Нетрудно показать, что вхождение подтерма, преобразованное в окружение переменных $x_1, \ldots, x_n$, всегда будет интерпретировано в окружении вида $x_1 = ., \ldots, x_n = .$, поэтому для отыскания значения переменной с индексом p достаточно взять значение p-го элемента окружения.

Это подсказывает другой путь интерпретации терма: вначале вычисляются индексы де Брауна для каждого вхождения переменной; как только все индексы известны, необходимость в хранении списка переменных в окружении отпадает. Окружение будет содержать лишь список оснащённых значений. Аналогично удаляются имена переменных из замыканий и задумок. Действительно, теперь имена переменных бесполезны, пример выше можно было бы записать так: **fun** _ → **fun** _ → (_^{1} + (**fun** _ → **fun** _ → (_^{3} + _^{2} + _^{1} + _^{0}))(2 * 8))(14 + 4))(5 + 7)(20 - 6).

Правила операционной семантики с большим шагом могут быть теперь определены следующим образом:

$$\frac{}{\mathtt{e} \vdash \_^{\mathtt{p}} \hookrightarrow \mathtt{V}} \text{ если } \mathtt{V} \text{ это } \mathtt{p}\text{-ый элемент } \mathtt{e},$$

$$\frac{\mathtt{e}' \vdash \mathbf{fix}\ \_\ \mathtt{t} \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \_^{\mathtt{p}} \hookrightarrow \mathtt{V}} \text{ если } \mathtt{p}\text{-ый элемент } \mathtt{e} \text{ это } \langle \mathbf{fix}\ \_\ \mathtt{t}, \mathtt{e}' \rangle,$$

$$\frac{\mathtt{e} \vdash \mathtt{u} \hookrightarrow \mathtt{W} \qquad \mathtt{e} \vdash \mathtt{t} \hookrightarrow \langle \mathtt{t}', \mathtt{e}' \rangle \qquad (\mathtt{e}', \mathtt{W}) \vdash \mathtt{t}' \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \mathtt{t}\ \mathtt{u} \hookrightarrow \mathtt{V}},$$

$$\frac{}{\mathtt{e} \vdash \mathbf{fun}\ \_ \to \mathtt{t} \hookrightarrow \langle \mathtt{t}, \mathtt{e} \rangle},$$

$$\frac{}{\mathtt{e} \vdash \mathtt{n} \hookrightarrow \mathtt{n}},$$

$$\frac{\mathtt{e} \vdash \mathtt{u} \hookrightarrow \mathtt{q} \qquad \mathtt{e} \vdash \mathtt{t} \hookrightarrow \mathtt{p}}{\mathtt{e} \vdash \mathtt{t} \otimes \mathtt{u} \hookrightarrow \mathtt{n}} \text{ если } \mathtt{p} \otimes \mathtt{q} = \mathtt{n},$$

$$\frac{\mathtt{e} \vdash \mathtt{t} \hookrightarrow \mathtt{0} \qquad \mathtt{e} \vdash \mathtt{u} \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \textbf{ifz } \mathtt{t} \textbf{ then } \mathtt{u} \textbf{ else } \mathtt{v} \hookrightarrow \mathtt{V}},$$

$$\frac{\mathtt{e} \vdash \mathtt{t} \hookrightarrow \mathtt{n} \qquad \mathtt{e} \vdash \mathtt{v} \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \textbf{ifz } \mathtt{t} \textbf{ then } \mathtt{u} \textbf{ else } \mathtt{v} \hookrightarrow \mathtt{V}} \text{ если } \mathtt{n} \text{ — число, } \mathtt{n} \neq \mathtt{0},$$

$$\frac{(\mathtt{e}, \langle \textbf{fix } \_ \ \mathtt{t}, \mathtt{e} \rangle) \vdash \mathtt{t} \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \textbf{fix } \_ \ \mathtt{t} \hookrightarrow \mathtt{V}},$$

$$\frac{\mathtt{e} \vdash \mathtt{t} \hookrightarrow \mathtt{W} \qquad (\mathtt{e}, \mathtt{W}) \vdash \mathtt{u} \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \textbf{let } \_ = \mathtt{t} \textbf{ in } \mathtt{u} \hookrightarrow \mathtt{V}}.$$

Преимущества такой записи, избавляющей от имён переменных, будут подчёркнуты в следующей главе при изучении компиляции.

Заметим, между прочим, что два терма имеют один и тот же вид при записи с помощью индексов де Брауна в том и только том случае, если они α-эквивалентны. Этот факт даёт ещё одно определение алфавитной эквивалентности. Замена переменных на индексы, обозначающие место их связывания, снова приводит к важной мысли о том, что связанные переменные это всего лишь «заглушки».

Побудова функцій за допомогою нерухомих точок

В большинстве языков программирования рекурсия допускается лишь при определении функций. Конструкция **fix** применима только для терма вида **fun**, поэтому можно было бы заменить символ **fix** на **fixfun** `f x` $\to$ `t`, который связывает в своём аргументе сразу две переменные. Семантическое правило с большим шагом при вызове по значению для последнего может быть выведено из приведённых выше правил для **fix** и **fun**:

$$\frac{}{\texttt{e} \vdash \textbf{fixfun}\ \texttt{f x} \to \texttt{t} \hookrightarrow \langle \texttt{x}, \texttt{t}, (\texttt{e}, \texttt{f} = \langle \textbf{fixfun}\ \texttt{f x} \to \texttt{t}, \texttt{e} \rangle) \rangle}\ .$$

В этом случае можно определить более простые версии правил для интерпретатора с вызовом по значению.

Рекурсивні замикання

Будем выделять замыкания вида $\langle \mathtt{x}, \mathtt{t}, (\mathtt{e}, \mathtt{f} = \langle \textbf{fixfun } \mathtt{f\ x} \to \mathtt{t}, \mathtt{e} \rangle) \rangle$, которые станем записывать в виде $\langle \mathtt{f}, \mathtt{x}, \mathtt{t}, \mathtt{e} \rangle$ и называть *рекурсивными замыканиями*.

Правило, данное для интерпретации конструкции **fixfun** $\mathtt{f\ x} \to \mathtt{t}$, может быть переформулировано следующим образом:

$$\frac{}{\mathtt{e} \vdash \textbf{fixfun } \mathtt{f\ x} \to \mathtt{t} \hookrightarrow \langle \mathtt{f}, \mathtt{x}, \mathtt{t}, \mathtt{e} \rangle}\ .$$

При попытке интерпретации применения $\mathtt{t\ u}$ в семантике вызова по значению, если терм $\mathtt{t}$ интерпретирован как рекурсивное замыкание $\langle \mathtt{f}, \mathtt{x}, \mathtt{t}', \mathtt{e}' \rangle$, то есть $\langle \mathtt{x}, \mathtt{t}', (\mathtt{e}', \mathtt{f} = \langle \textbf{fixfun } \mathtt{f\ x} \to \mathtt{t}', \mathtt{e}' \rangle) \rangle$, а терм $\mathtt{u}$ — как значение $\mathtt{W}$, требуется интерпретировать терм $\mathtt{t}'$ в окружении $\mathtt{e}', \mathtt{f} = \langle \textbf{fixfun } \mathtt{f\ x} \to \mathtt{t}', \mathtt{e}' \rangle, \mathtt{x} = \mathtt{W}$.

Можно ожидать интерпретации задумки $\langle \textbf{fixfun}\ \mathtt{f\ x} \to \mathtt{t'}, \mathtt{e'} \rangle$, которая возникает в таком окружении, что приведёт к использованию правила **fixfun**, являющегося рекурсивным замыканием $\langle \mathtt{f}, \mathtt{x}, \mathtt{t'}, \mathtt{e'} \rangle$. В случае рекурсивных замыканий правило для применения может быть специализировано следующим образом:

$$\frac{\begin{array}{c}\mathtt{e} \vdash \mathtt{u} \hookrightarrow \mathtt{W} \\ \mathtt{e} \vdash \mathtt{t} \hookrightarrow \langle \mathtt{f}, \mathtt{x}, \mathtt{t'}, \mathtt{e'} \rangle \\ (\mathtt{e'}, \mathtt{f} = \langle \mathtt{f}, \mathtt{x}, \mathtt{t'}, \mathtt{e'} \rangle, \mathtt{x} = \mathtt{W}) \vdash \mathtt{t'} \hookrightarrow \mathtt{V}\end{array}}{\mathtt{e} \vdash \mathtt{t\ u} \hookrightarrow \mathtt{V}}.$$

В этом правиле не используются задумки; таким образом, при вызове по значению благодаря введению рекурсивных замыканий исчезают задумки, также отпадает необходимость в правиле их интерпретации.

Последнее упрощение: обычные замыкания $\langle \texttt{x}, \texttt{t}, \texttt{e} \rangle$ могут быть заменены рекурсивными $\langle \texttt{f}, \texttt{x}, \texttt{t}, \texttt{e} \rangle$, где `f` это произвольная переменная, которая не входит в `t`. Тогда можно удалить правило применения для случая обычных замыканий.

Окончательно мы получаем следующие правила:

$$\frac{}{\mathtt{e} \vdash \mathtt{x} \hookrightarrow \mathtt{V}} \text{ если } \mathtt{e} \text{ содержит } \mathtt{x} = \mathtt{V},$$

$$\frac{\begin{array}{c}\mathtt{e} \vdash \mathtt{u} \hookrightarrow \mathtt{W} \\ \mathtt{e} \vdash \mathtt{t} \hookrightarrow \langle \mathtt{f}, \mathtt{x}, \mathtt{t}', \mathtt{e}' \rangle \\ (\mathtt{e}', \mathtt{f} = \langle \mathtt{f}, \mathtt{x}, \mathtt{t}', \mathtt{e}' \rangle, \mathtt{x} = \mathtt{W}) \vdash \mathtt{t}' \hookrightarrow \mathtt{V}\end{array}}{\mathtt{e} \vdash \mathtt{t}\ \mathtt{u} \hookrightarrow \mathtt{V}},$$

$$\frac{}{\mathtt{e} \vdash \textbf{fun}\ \mathtt{x} \to \mathtt{t} \hookrightarrow \langle \mathtt{f}, \mathtt{x}, \mathtt{t}, \mathtt{e} \rangle},$$

где $\mathtt{f}$ это произвольная переменная, отличная от $\mathtt{x}$ и не входящая ни в $\mathtt{t}$, ни в $\mathtt{e}$,

$$\frac{}{\mathtt{e} \vdash \textbf{fixfun}\ \mathtt{f}\ \mathtt{x} \to \mathtt{t} \hookrightarrow \langle \mathtt{f}, \mathtt{x}, \mathtt{t}, \mathtt{e} \rangle},$$

$$\frac{}{\mathtt{e} \vdash \mathtt{n} \hookrightarrow \mathtt{n}},$$

$$\frac{\mathtt{e} \vdash \mathtt{u} \hookrightarrow \mathtt{q} \qquad \mathtt{e} \vdash \mathtt{t} \hookrightarrow \mathtt{p}}{\mathtt{e} \vdash \mathtt{t} \otimes \mathtt{u} \hookrightarrow \mathtt{n}} \text{ если } \mathtt{p} \otimes \mathtt{q} = \mathtt{n},$$

$$\frac{\mathtt{e} \vdash \mathtt{t} \hookrightarrow \mathtt{0} \qquad \mathtt{e} \vdash \mathtt{u} \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \textbf{ifz } \mathtt{t} \textbf{ then } \mathtt{u} \textbf{ else } \mathtt{v} \hookrightarrow \mathtt{V}},$$

$$\frac{\mathtt{e} \vdash \mathtt{t} \hookrightarrow \mathtt{n} \qquad \mathtt{e} \vdash \mathtt{v} \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \textbf{ifz } \mathtt{t} \textbf{ then } \mathtt{u} \textbf{ else } \mathtt{v} \hookrightarrow \mathtt{V}} \text{ если } \mathtt{n} \text{ — число, } \mathtt{n} \neq \mathtt{0},$$

$$\frac{\mathtt{e} \vdash \mathtt{t} \hookrightarrow \mathtt{W} \qquad (\mathtt{e}, \mathtt{x} = \mathtt{W}) \vdash \mathtt{u} \hookrightarrow \mathtt{V}}{\mathtt{e} \vdash \textbf{let } \mathtt{x} = \mathtt{t} \textbf{ in } \mathtt{u} \hookrightarrow \mathtt{V}}.$$

Раціональні значення

В правиле

$$\frac{}{\texttt{e} \vdash \textbf{fixfun}\ \texttt{f x} \to \texttt{t} \hookrightarrow \langle \texttt{x, t, (e, f =} \langle \textbf{fixfun}\ \texttt{f x} \to \texttt{t, e} \rangle \texttt{)} \rangle}$$

можно ожидать будущую интерпретацию задумки $\langle \textbf{fixfun}\ \texttt{f x} \to \texttt{t, e} \rangle$. Конечно, её значением является терм $\langle \texttt{x, t, (e, f =} \langle \textbf{fixfun}\ \texttt{f x} \to \texttt{t, e} \rangle \texttt{)} \rangle$, в котором эта задумка возникает вновь. Можно было бы пытаться интерпретировать её снова и снова.

Как было сказано ранее, этот вид интерпретации терма в форме **fix f t** до подстановки или сохранения в окружение приводит к бесконечным вычислениям. В данном случае это приводит к построению бесконечного значения $\langle \texttt{x, t, (e, f =} \langle \texttt{x, t, (e, f =} \langle \texttt{x, t, (e, f =} \langle \texttt{x, t, (e, f = } \ldots \texttt{)))))} \rangle \rangle \rangle \rangle$, которое представляет из себя бесконечный, но рациональный терм. Существуют хорошо известные способы представления рациональных деревьев в памяти компьютера. В нашем случае такое значение можно представить структурой, изображённой на рис. 1.

Используя обозначение $\texttt{FIX X} \langle \texttt{x, t, (e, f = X)} \rangle$ для этого рационального значения, правило выше можно записать так:

$$\frac{}{\texttt{e} \vdash \textbf{fixfun}\ \texttt{f x} \to \texttt{t} \hookrightarrow \texttt{FIX X} \langle \texttt{x, t, (e, f = X)} \rangle}\ .$$

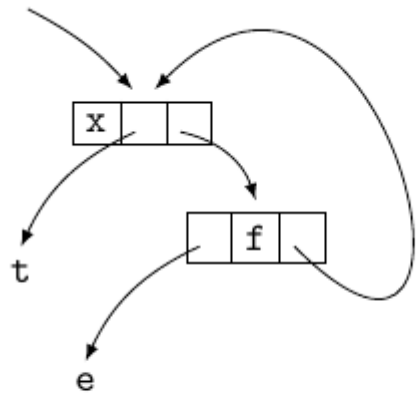

Рис. 1: представление рационального терма

Заметим, что иногда лучше представлять такое рациональное значение эквивалентным способом, изображённым на рис. 2. В этом случае следовало бы говорить о рациональных окружениях.

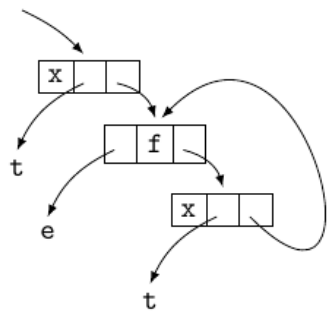

Рис. 2: эквивалентное представление рационального терма

Подведём итоги. В этом разделе было показано, что если переменная `x` входит в терм `t`, то правило редукции **fix** `x t` $\longrightarrow$ **(fix** `x t/x)t` может применяться к терму **fix** `x t` бесконечно, потому что терм **(fix** `x t/x)t` вновь содержит терм **fix** `x t` в качестве подтерма. В рекурсивном определении `f = G(f)` это соответствовало бы подстановке `G(f)` вместо `f` бесконечное число раз, что приводит к бесконечной программе `f = G(G(G(...)))`. В каком-то смысле это объясняет интуитивное представление о том, что рекурсивные программы это бесконечные программы. Например, терм `fact` мог бы быть записан так:

ifz `x` **then** `1` **else** `x*(`**ifz** `x - 1` **then** `1` **else** `(x - 1)*`
(ifz `x - 2` **then** `1` **else** `(x - 2)*...)).`

Такая замена должна производиться только по необходимости — в ленивом стиле.

Было рассмотрено несколько способов отразить это поведение в семантике PCF — и, в конце концов, в коде интерпретатора PCF: подставить **fix** `x t` вместо `x` и «заморозить» этот терм, если он оказался внутри **fun** или **ifz**, сохранить этот редекс в виде задумки или рекурсивного замыкания и выполнять его только по необходимости, представить терм `f = G(G(G(...)))` как рациональное дерево и обходить его только по необходимости. Наконец, ещё один способ основывается на кодировании **fix**, приведённом в упражнении 2.10, и состоит в том, чтобы редуцировать соответствующий терм (который предполагает дублирование подтерма) только по необходимости.

На наступній лекції розглянемо формалізацію процеса компіляції за допомогою мови PCF.