УДК 51:681.3

Р.С. Шевченко, А.Е. Дорошенко

η-ИСЧИСЛЕНИЕ – РЕАЛИСТИЧНАЯ ФОРМАЛИЗАЦИЯ КЛАССА ПЕРЕПИСЫВАЮЩИХ СИСТЕМ

Предложен новый формализм типизированного η-исчисления в качестве теоретической основы для построения специальных классов систем программирования на основе переписывающих правил. Формализм использует упорядоченные неконфлюэнтные множества правил переписывания и взаимодействие с программным окружением, что позволяет расширить возможности программирования динамических приложений.

Введение

Техника символьных преобразований, основанная на правилах переписывания термов, является мощным средством разработки программных систем. Системы переписывания известны давно и существует довольно много формализмов для их описания, такие как эквациональная логика [1], ρ-исчисление [2] или λ-исчисление с образцом [3, 4], снабженные разнообразной системой типов [5]. Однако при разработке и использовании систем переписывания становится очевидной неполнота описания математическими моделями существующей практики техники переписывания и соответствующие ограничения, а именно:

- математические модели в большинстве своем описывают семантику неупорядоченных конфлюэнтных систем правил без приоритетов, хотя достаточно удобными оказываются системы с использованием упорядоченного набора правил, где порядок применения определяется либо отношением конкретизации, либо приоритетом, а наборы правил формально неконфлюэнтны;
- в качестве ввода и вывода обычно используется множества термов, при этом существуют применения для системы с интерактивным взаимодействием;
- кроме действий по сопоставлению с определенным образцом, также используются действия по неудачному сопоставлению; обычно это сопровождается расширением существующих моделей, достаточно сильно меняющим первоначальную семантику [6].

В данной работе строится формализм типового исчисления, названный η-исчислением, как возможное теоретическое обоснование для класса систем переписывания с упорядоченными правилами и взаимодействием со внешней средой, предоставляющих разработчику полный набор возможностей по определению средств проектирования приложений. В частности, рассмотрен способ встраивания средств ввода/вывода в структуру типов языка, чтобы анализ типов мог включать в себя анализ взаимодействия с окружающей средой, что может быть полезно с точки зрения оптимизации программ. η-исчисление строится на замкнутых термах без использования свободных переменных, что дает некоторый новый взгляд на спектр возможностей программной реализации. Реализацию η-исчисления с системой типов и анализом взаимодействий планируется встроить в следующую версию авторской системы Termware [7, 8].

1. Модель переписывающих систем

1.1. Основные обозначения. Рассмотрим некоторое множество термов T над алфавитом A с помощью некоторой индуктивной схемы, содержащей множество переменных X.

Для термов $g, h \in T$ обозначение $g \blacktriangleleft h$ означает, что g содержится в h. Мно-

жество свободных переменных терма t обозначается $fv(t)$. Подстановка – это множество пар, состоящих из свободных переменных и термов. Результат применения подстановки S к терму t будем обозначать как $t[S]$. Синтаксическую эквивалентность термов будем обозначать $t1=t2$, а эквивалентность экземпляров — $t1\equiv t2$. Для множеств будем использовать декартово произведение $X\times Y$ и проекции $X|i$, а именно: $X\times Y$ – это множество пар (x,y) таких, что $x\in X$, $y\in Y$; $(x,y)|1 = x$, $(x,y)|2 = y$. Для множества пар S определим $S|i = \{s|i: s\in S\}$. Последовательность элементов $seq(x_1,\ldots, \ldots,x_n)$ будет иногда обозначаться как $(x_1,\ldots,x_n)$. Проекция $(x_1,\ldots,x_n)|i$ также может быть применена к последовательности и обозначает i-й элемент этой последовательности.

1.2. Нетипизированное η-исчисление. Пусть наш алфавит состоит из констант $c_i\in C$, функциональных символов $f_i\in F$ и пропозициональных переменных $x_i \in X$. Теперь множество неограниченных η-термов T_η^S может быть построено индуктивно:

- константы C: если $c_i\in C$, то $c_i\in T_\eta^S$. Также будем предполагать, что C содержит выделенную константу для ошибки, которую будем обозначать как $\perp$, и булевские значения true и false;
- переменные X: если $x_i \in X$, то $x_i \in T_\eta^S$;
- функции 1-го порядка F: если $f_i\in F$, $t_1,\ldots,t_n\in T_\eta^S$, то $f(t_1,\ldots,t_n)\in T_\eta^S$; предполагается, что множество функциональных символов включает, по крайней мере, конструктор последовательности $seq(x_1,\ldots,x_n)$, проекции $proj_i$, условное выражение $cond(x,y,z)$ и тождественное преобразование id;
- композиции функций высших порядков: если $t \in$, $t_1,\ldots,t_n \in T_\eta^S$, то $t(t_1,\ldots,t_n) \in T_\eta^S$;
- η-термы: если $v=\{x_1\ldots x_n: x_i\in X\}$, $f,g,h\in T_\eta^S$, то $(\eta v.f \to g|h)\in T_\eta^S$; также будет использоваться написание $\eta\ v.f \to g$ как сокращение для $\eta\ v.f \to g|id$;
- let-конструкция: $let\ x \leftarrow f.g$ (иногда будем использовать альтернативный синтаксис $g\ where\ x \leftarrow f$).

Пусть $r = (\eta\ v.f \to g|h) \in T_\eta^S$. Назовем $X(t)$ множество переменных, связанных с самым верхним η-термом (т. е. v); $P(t)$, $T(t)$ и $F(t)$ обозначим, соответственно, образец, результат в случае успеха в сопоставлении и в случае неуспеха (т. е. f, g, h, соответственно).

Теперь выберем из T_η^S подмножество T_η с полностью связанными переменными, включающее:

- константы: если $c_i \in C$, то $c_i \in T_\eta$;
- функции первого порядка: если $f\in F$, $t_1,\ldots,t_n \in T_\eta$, то $f(t_1,\ldots,t_n)\in T\eta$;
- композиции функций высших порядков: если $t\in T_\eta$, $t_1,\ldots,t_n\in T_\eta$, то $t(t_1,\ldots,t_n)\in T_\eta$;
- полностью связанные η-термы: если $v=\{x_1,\ldots,x_n: x_i \in X\}$, $f,g,h\in T_\eta$, и $t=(\eta v.f\to g|h)$, то $t\in T_\eta$ при одновременном выполнении следующих условий:
  1) $\forall\ x \in X : x \blacktriangleleft f \Rightarrow x \in X(t)$ или $x\blacktriangleleft\eta\ z_1\ldots z_n\ .fz\to gz|hz \blacktriangleleft g$;
  2) $\forall\ x \in X: x \blacktriangleleft g \Rightarrow x\in X(t)\wedge x \blacktriangleleft f$ или $x \blacktriangleleft \eta\ z_1\ldots z_n\ .fz\to gz|hz \blacktriangleleft g$;
  3) $\forall\ x \in X: x \blacktriangleleft h \Rightarrow x\in X(t) \wedge x\blacktriangleleft f$ или $x \blacktriangleleft \eta\ z_1\ldots z_n\ .fz\to gz|hz \blacktriangleleft h$.

Иными словами, все η-термы закрыты, любая переменная из g находится в какой-либо η-конструкции (либо самой верхней, либо внутри g); то же относится и к переменными из h. Для каждой переменной x можно определить η-конструкцию, в которой она определена. Будем обозначать эту конструкцию как $\eta t(x)$.

1.3. Операционная семантика. Основная операция семантической модели – это редукция η-правила:

$$(\eta v.f \to g\ |\ h)\ q = \begin{cases} g[S], \text{если } match\ (f,q)\,|_1 \\ \text{и } S = match\ (f,q)\,|_2; \\ hq, \text{если } \neg match\ (f,q). \end{cases}$$

Таким образом, успех правила переписывания зависит от успеха сопоставления образца в левой части правила с аргументом. Будем преобразовывать η-выражения в условные выражения:

$$(\eta v\ .f \to g|h)\ q = let\ m \leftarrow$$
$$\leftarrow match(f,q).cond(m|_1, q[m|_2], h\ q)$$
$$(\eta{:}match).$$

Условные выражения могут быть редуцированы, когда первый аргумент принимает значение true или false, а именно: $cond(\mathbf{true},f,g)=f$; $cond(\mathbf{false},f,g)=g$. Let-выражения используются в основном для совместного использования подтермов и могут быть полностью элиминированы: $let\ x \leftarrow y.z = z\ [\{(x,y)\}]$.

Собственно, суть формализации состоит в разрешении проблемы сопоставления. Результатом сопоставления будет пара из булевского значения и подстановки: два терма x и y совместимы, если существует подстановка S, такая что $x[S]=y$. Определим процесс сопоставления таким образом, что поведение сопоставления зависит от уровня вложенности переменных. При сопоставлении переменных, определенных в разных η-выражениях, соответствующая пара добавляется в результирующую подстановку. Но если сравниваются две переменные, определенные в одном η-выражении, то они должны быть синтаксически эквивалентны.

Пусть теперь $x,y \in X$. Тогда:

$$match\,(x,y) =$$
$$= \begin{Bmatrix} (\mathbf{true}, (x \leftarrow y)), \\ \text{если } \neg\eta t(x) \equiv \eta t(y) \\ (x = y, \varnothing), \\ \text{если} \quad \eta t(x) \equiv \eta t(y) \end{Bmatrix} (match : \text{var}).$$

Если $x \in X$ и $y \notin X$, то просто добавляем пару в подстановку: $match(x,y) = (\mathbf{true}, (x \leftarrow y))$ $(match{:}var)$.

Сопоставление структурных термов можно определить, как обычно, по индукции. Пусть x и y – функциональные термы, $x=x_0(x_1,...,x_n)$, $y=y_0(y_1,...,y_m)$. Тогда:

$$match\,(x,y) =$$
$$= \begin{Bmatrix} \prod_{i=0}^{n} match\,(x_i, y_i), \\ \text{если } m = n \\ (\mathbf{false}, \varnothing), \\ \text{если} \quad m \neq n \end{Bmatrix} (match : \text{fun}),$$

где произведение на парах определено следующим образом:

$$(b_1, s_1) \wedge (b_2, s_2) =$$
$$= \begin{cases} (b_1 \wedge b_2, s_1 \cup s_2 \cup (s_1 \oplus s_2)), \\ \text{если } s_1 \text{ и } s_2 \text{ совместимы} \\ (x = y, \varnothing), \\ \text{если они несовместимы}, \end{cases}$$

а два множества совместимы, если выполняются условия

$$\forall\ x,y,z{:}\ (x,y) \in s_1 \wedge (x,z) \in s_2 \Rightarrow$$
$$\Rightarrow match(y,z) \text{ и}$$
$$(s_1 \oplus s_2) = \bigcup_{\mathrm{p}} (y,z),$$
$$p = \{x,y,z{:}(x,y) \in s_1, (x,z) \in s_2\}.$$

Заметим, что сопоставление констант получается тривиальным образом:

$$match(c_1, c_2) =$$
$$= \begin{Bmatrix} (\mathbf{true}, \varnothing), \\ \text{если} \quad c_1 = c_2 \\ (\mathbf{false}, \varnothing), \\ \text{если} \quad c_1 \neq c_2 \end{Bmatrix} (match{:}c).$$

При этом термы с разными заглавными конструкциями не могут быть сопоставлены и, соответственно, сравнимы.

Вышеописанные правила неконфлюэнтны, если образец содержит выражения из $T\eta$. К примеру, терм $match(cond(\mathbf{true},1,2),1)$ будет редуциро-

ваться к **true** или **false** в зависимости от того, будет ли вначале вычисляться внешний или внутренний подтерм.

Обычно во избежание подобной неоднозначности в формализмах фиксируется порядок редукций. Предложим несколько другой подход, основанный на том, что выявление неоднозначности может иметь определенный смысл с точки зрения программирования и с точки зрения прагматики лучше выявить и локализовать противоречие в системе правил, а не придавать четко фиксированный результат неоднозначной системе. Более подробно предотвращение неоднозначности будет рассмотрено в разделе 3.

2. Интенсиональное равенство как замена переименования переменных

Построенное исчисление переписывающих правил высшего порядка основано на замкнутых термах, где пропозициональные переменные действуют в разных областях разных η-выражений. Основанные на этой семантике процессы вычислений могут избегать переименования переменных во время применения правил переписывания, так как переменные должны быть разными только внутри одного η-терма. Однако вместо такого определения области действия можно использовать две операции, не входящие в традиционную операционную семантику языков и использующие:

- возможность по переменной определить η-терм, в котором эта переменная определена;
- интенсиональное равенство $t1 \equiv t2$, если $t1$ и $t2$ – это один и тот же терм (подразумевая, что реализация не использует неявно различаемые подтермы).

Таким образом, необходимость и "магия" переименования переменных компенсируется необходимостью разрыва идентичности η-термов при копировании. Естественной интерпретацией равенства ≡ может быть сравнение машинных адресов (с тем ограничением, что система должна заменять адрес во время η-переписывания). Можно переопределить η-исчисление и без интенсионального равенства, используя модель "термов с метками". Пусть для каждого терма t у нас определена его метка $L(t)$ такая, что:

- разные термы помечены разными метками: $t1 \neq t2 \Rightarrow L(t1) \neq L(t2)$;
- во время создании η-терма при переписывании метка новой копии должна не совпадать с уже существующими.

Достаточно теперь определить метки как явные компоненты η-термов и пропозициональных переменных и изменить определение структурной эквивалентности так, чтобы не принимать метки во внимание при сравнении η -термов.

Пусть η-терм выглядит, как $<L1, L2>\ \eta\ X.f \to g|h,$ а связанные переменные, как $<L1, L2>v_i$. Метка состоит из двух частей: $L1$ – уникальный идентификатор для начального η-терма, который выделяется при создании начального множества термов и $L2$ – переменная часть, которая изменяется во время каждой подстановки. Тогда все, что нужно сделать для переформулирования η-исчисления – это изменить правило (ρ*:match*) так, что бы при подстановке инкрементировалась переменная часть меток переменных:

$$apply((\eta v.f \to g \mid h)\ \ q = \begin{cases} incrL(q[S]), \\ \text{если} \quad match(f,h)|_1 \\ \text{и} \quad S = match(f,h)|_2; \\ apply(hq), \\ \text{если} \quad \neg match(f,h), \end{cases}$$

где *incrL* каким-то образом изменяет метки переменных.

Можно сказать, что использование интенсионального равенства и уникального машинного представления эквивалентно явному процессу переименования пере-

менных. Использование меток для эмулирования интенсионального равенства может быть использовано при реализации системы на платформах без доступа к вычислению адресов (таким, например, как виртуальная машина JVM).

3. Комбинирование правил в упорядоченные множества правил

Еще одним упрощением в языках, основанных на переписывающих правилах, является предположение, что вычислительный процесс не должен зависеть от порядка применения набора правил. Однако в типичной программисткой практике чаще встречается другой тип рассуждений, сводящийся к выбору между частными случаями какого-то общего образца.

Определим частичное упорядочивание на множестве правил по "степени конкретности" входящих образцов. То есть будем говорить, что $t_1 << t_2$, если t_1 является "более конкретным", чем t_2. Для нашего случая можем определить, что $f << g$, если $\forall h.match(g,h)|_1 \Rightarrow match(f,h)|_1$.

Теперь пусть $p = (f_p \to g_p)$ и s = $(f_s \to \to g_s)$ – два правила с пересекающимися образцами и $f_p << f_s$. Как правило, наличие в системе двух таких правил говорит о намерении сначала провести более частную проверку, а потом более общую. Тогда можно ввести правило вида $f_p \to g_p \mid (f_s \to \to g_s|id)$ (в традиционной интерпретации такие правила образуют неконфлюэнтную критическую пару [9]). Соответственно, комбинирование множества правил переписывания $\{p_1,...,p_N\}$ сводится к разбиению этого множества правил на последовательность упорядоченных правил со взаимно-непересекающиеся образцами и построению упорядоченного локально-конфлюэнтного терма внутри таких сегментов. Естественно, построить отношения << не всегда возможно, что иллюстрируется следующим примером:

1) $\eta\ x,y: f(x,a(y)) \to 1$

2) $\eta\ x,y: f(a(x),y) \to 2$

3) $\eta\ x,y: f(x,y) \to 3$.

Здесь невозможно построить упорядочение образцов, так как редукция терма $f(a(x),a(x))$ неоднозначна и зависит от стратегии (или порядка) применения правил переписывания.

Заметим, что неконфлюэнтность системы с упорядоченным множеством правил всегда может быть разрешена добавлением дополнительных разрешающих правил. То есть, если у нас есть система переписывающих правил $r_1,\ldots,r_N$, дающая неоднозначный результат при переписывании, упорядоченном по степени конкретности, то мы можем построить систему правил $r_1,\ldots,r_N, r_{N+1},...,r_{N+K}$, включающую в себя первоначальную и в свою очередь конфлюэнтную. Действительно, если система неконфлюэнтна, значит найдутся два правила r_i и r_j, образующую критическую пару. Это означает, что r_i и r_j имеют общий унификатор образцов (пусть это будет l: $l << left(r_i)$ и $l << left(r_j)$. Тогда можем ввести новое правило $r_{N+1}= l \to x$, которое при упорядоченно-частном применении будет всегда применяться перед r_i или r_j для образцов, образующих критическую точку в $r_1,...,r_N$. Для того, чтобы x не вводил новых критических точек, достаточно выбрать его совпадающим с каким-либо из уже существующих результатов подстановок правых частей r_i или r_j. Итого, получим новую систему, где r_i и r_j не являются более критической парой. Продолжая этот процесс до полного исчерпания критических пар получаем конфлюэнтную систему.

Так, вышеприведенном примере критическую пару составляют правила 1) и 2), общий унификатор образцов − $\eta\ x,y: :f(a(x),a(y))$. Следовательно, систему можно превратить в конфлюэнтную по отношению к упорядоченному переписыванию, добавив правило: $\eta\ x,y: f(a(x),a(y)) \to 1$.

Такое построение работает для систем переписывания первого порядка. Для систем высших порядков дело обстоит несколько сложнее, так как редукция может образовывать критическую пару с собственной подстановкой, процесс определения дополнительных правил тогда становится бесконечным и фактически сводит-

ся к определению порядка вычисления. Стандартный пример, который обычно иллюстрирует неконфлюэнтность правил переписывания второго порядка [2], в η-исчислении можно переписать как (η $x.x\ a \to x$) ((η $y.y \to y$) a), или, в полной форме: $apply(\eta\ x.apply(x,a) \to x,\ apply((\eta\ y.y \to y),\ a))$. В работе [10] рассмотрены примеры ограничений вывода для получения локальной конфлюэнтности систем высшего порядка, основанных на λ-исчислении (такие как линейность образцов).

4. Взаимодействие с внешней средой – получение и посылка сигналов

Теперь дополним конструкции построенного языка операциями взаимодействия с внешней средой. В предыдущей версии системы Termware [7, 8] обмен информацией с внешним миром осуществлялся с помощью установки значения свободной переменной из процедурной части базы данных (БД) фактов. В настоящем подходе с использованием связующих переменных естественно ввести специальные конструкции для получения и передачи информации. Будем представлять БД фактов как пару:

- множество входных событий, которое может быть представлено множеством функциональных символов или атомов T_{in};
- множество выходных действий T_{out}, также представленных функциональными символами или атомами.

Для учета передачи информации дополним наш язык θ-конструкциями, получив множество термов $T\eta\theta$ с помощью следующего рекурсивного определения. Выделим подмножества $T\theta_{in}$ и $T\theta_{out}$ следующим образом:

- любой η-терм входит в дополненный язык: $t \in T\eta \Rightarrow t \in T\theta_{in}$ и $t \in T\theta_{out}$;
- можно комплектовать входные и выходные функциональные термы:

$f_{in} \in T_{in}, t_1 ... t_N \in T\theta_{in} \Rightarrow f_{in}(t_1,..t_N) \in T\theta_{in}$,

$f_{out} \in T_{out}, t_1 ... t_N \in T\theta_{out} \Rightarrow f_{out}(t_1,..t_N) \in T\theta_{out}$;

- можно связывать входные переменные с помощью θ-конструкций ввода:

$f \in T\theta_{in}, g \in T\eta^S \wedge fv(g)=\{x\} \Rightarrow \theta x \leftarrow_{\text{in}} f.g \in T\theta_{in}$;

- можно передавать информацию с помощью θ-конструкций вывода:

$f \in \text{T}\theta_{out}, \text{g}, \text{h} \in T\eta \Rightarrow \theta g.f \leftarrow_{out} \text{h} \in T\theta_{out}$;

- правило взаимодействия корректно, если в левой части находятся конструкции ввода, а в правой – конструкции вывода: $f \in T\theta_{\text{in}}, g,h \in T\theta_{out} \Rightarrow \eta(f \to g|h) \in T\eta\theta$.

Операционная семантика таких правил может быть определена как проведение операций ввода при сопоставлении и вывода при подстановке.

5. Типизация η-исчисления

Типизация построенного η-исчисления имеет целью улучшить свойства анализируемости формул исчисления и эффективности его реализации. Определение правил вывода типов для η-конструкций, θ-конструкций и правил взаимодействия начинается с принятия сигнатуры типов для констант и функциональных символов. Начнем со случая без взаимодействий: пусть имеем некоторую упорядоченно сортную сигнатуру $(T, <, \sum)$ и требуется обогатить ее возможностью построения η-термов. Можно построить рекурсивно полиморфную структуру типов, в точности следуя подходу, принятому в [5], т. е. построив систему типов $V=T+(V \to V)+(V+V)+(V \times V)+W$ путем прямой трансляции конструкций λ2μ∪ в контекст η-исчисления. Единственное изменение, которое надо будет внести в собственно исчисление – это типизация связанных переменных. Правила вывода тогда будут иметь следующий вид, где символ ▹ означает свойство выводимости:

1) $\dfrac{x : \delta \in \Gamma}{\Gamma \triangleright x : \delta}$ – начало;

2) $\dfrac{\Gamma \triangleright M : \varphi \to \delta, \Gamma \triangleright N : \varphi}{\Gamma \triangleright apply(M, N) : \delta}$ – элиминация стрелки;

3) $\dfrac{\Gamma \triangleright M : \varphi, \Gamma \triangleright N : \varphi, \Gamma \triangleright Q : \tau}{\Gamma \triangleright \eta M \to N \,|\, Q : \varphi \to \delta \,|\, \tau}$ – внесение стрелки;

4) $\dfrac{\Gamma \triangleright M : \varphi \cap \delta}{\Gamma \triangleright M : \varphi, \Gamma \triangleright M : \delta}$ – элиминация $\cap$;

5) $\dfrac{\Gamma \triangleright M : \varphi, \Gamma \triangleright M : \delta}{\Gamma \triangleright M : \varphi \cap \delta}$ – внесение $\cap$;

6) $\dfrac{\Gamma \triangleright M : \varphi \times \delta}{\Gamma \triangleright proj_1(M) : \varphi, \Gamma \triangleright proj_2(M) : \delta}$ – элиминация $\times$;

7) $\dfrac{\Gamma \triangleright M : \varphi, \Gamma \triangleright N : \delta}{\Gamma \triangleright pair(M, N) : \varphi \times \delta}$ – внесение $\times$.

Теперь добавим к рассмотрению θ-конструкции, определив на них типизацию. Добавим указание типа к переменным ввода и пусть для каждого входного события f_{in}, совместимого со входной переменной типа τ, имеется определенный идентификатор события *Id*(*f*). Будем обозначать такой факт как *Id*(*f*):τ.

Введем специальный тип $\theta_{in}(\{Id_i{:}\varphi_i\},\delta)$, содержащий в себе множество входных событий и результирующий тип, с интерпретацией правила, которое воспринимает на входе событие с такими *Id* и выдает на выход результирующий терм типа δ. Также можем определить, что $\theta_{in}(\varnothing,\delta)=\delta$.

Соответствующие правила конструирования и правила элиминации имеют вид:

1) $\dfrac{\Gamma \triangleright M : \varphi, \Gamma \triangleright Id(f) : \delta}{\Gamma \triangleright \theta(x \leftarrow_{in} f)\ M : \theta_{in}(\{Id(f) : \delta\}, \varphi)}$ – внесение θ_{in};

2) $\dfrac{\Gamma \triangleright M : \theta_{in}(Id(f) : \delta, \theta_{in}(Id(g) : \tau, \varphi))}{\Gamma \triangleright M : \theta_{in}(\{Id(f) : \delta\}, Id(g) : \tau\}, \varphi)}$ – секвенция θ_{in};

3) $\dfrac{\Gamma \triangleright M : \varphi}{\Gamma \triangleright M : \theta_{in}(\varnothing, \varphi)}$ – пустой сигнал;

4) $\dfrac{\Gamma \triangleright M : \theta_{in}(\delta, \varphi)}{\Gamma \triangleright M : \varphi}$ – ослабление зависимости;

5) $\dfrac{\Gamma \triangleright M : \theta_{in}(Id(f) : \delta, \varphi)}{\exists N, f.M = \theta(x \leftarrow_{in} f)\ N, \Gamma \triangleright N : \varphi, \Gamma \triangleright Id(f) : \delta)}$ – элиминация θ_{in};

6) $\dfrac{\Gamma \triangleright M : \theta_{in}(Id(f) : \delta, Id(g) : \tau, \varphi)}{\Gamma \triangleright M : \theta_{in}(Id(f) : \delta\}, \theta_{in}(Id(g) : \tau\}, \varphi))}$ – десеквенция θ_{in}.

Точно также можно определить $\theta_{out}(\{Id_i{:}\varphi_i\},\delta)$, содержащий в себе набор идентификаторов выходных событий и ввести аналогичные правила конструирования. Например, правило вывода для внесения в формулу θ_{out} будет иметь вид: $\dfrac{\Gamma \triangleright M : \varphi, \Gamma \triangleright N : \delta, \Gamma \triangleright Id(f) : \delta}{\Gamma \triangleright \theta M\,(f \leftarrow_{out} N) : \theta_{out}(Id(f) : \delta, \varphi)}$ (остальные правила полностью аналогичны соответствующим правилам для θ_{in}, поэтому здесь не приводятся).

Отношение между θ_{in}-типами и остальными можно записать покомпонентно (правила для θ_{out} – аналогичны):

$\theta_{in}(A,\varphi) \cup \theta_{in}(B,\tau) = \theta_{in}(A \cup B, \varphi \cup \tau)$
$\theta_{in}(A,\varphi) \cap \theta_{in}(B,\tau) = \theta_{in}(A \cap B, \varphi \cap \tau)$
$\theta_{in}(A,\varphi) \times \theta_{in}(B,\tau) = \theta_{in}(A \cup B, \varphi \times \tau)$
$\theta_{in}(A,\varphi \to \tau) \to \theta_{in}(B,\tau) = \theta_{in}(A \cup B, \tau)$.

Анализ типов взаимодействий может использоваться системой выполнения правил для определения корректности оптимизационных преобразований. Например, если система правил не включает в себя взаимодействия, то применение кэширования промежуточных результатов

(memoizing) не изменит семантику вычислений и может быть применено.

Выводы

Построенное исчисление описывает применения правил переписывания, пригодных к широкому спектру задач. Сопоставление η-исчисления и ρ-исчисления позволяет сделать вывод как об области применения, так и об архитектуре разрабатываемых систем. Если в η-исчислении мы работаем с одним термом, который сопоставляется с правилами на основе выбранного порядка применение, то в ρ-исчислении система имеет дело сразу со множеством термов. И, таким образом, структурный оператор ρ-исчисления позволяют применить сразу множество правил, при этом как в η-исчислении все время выбирается одно правило, на основе порядка и стратегии. Недетерминированный выбор и спецификация результата в случае отказа изначально не были предусмотрены в ρ-исчислении, но для этой цели существуют его расширения [4, 6]. Интересно, что реализация обработки отказа также потребовала фиксации стратегии редукции методом call-by-value. Намного ближе в этом отношении находятся вариации λ-исчисления с образцами [3, 9] и исчислениями образцов [4].

В перспективе возможно построение объединяющего формализма, включающего в себя редукцию как одиночных термов, так и множеств так, что бы подобное исчисление в программировании, основанном на правилах переписывания термов, заняло роль, подобную роли System F [11] в функциональном программировании.

Типизация правил взаимодействия в принципе не зависит от собственно η-исчисления и может быть применена к любому типизированному декларативному языку с побочными эффектами.

В будущем планируется исследовать возможности ослабления требований к стратегии редукций [12] в отдельных областях и встроить типизированное η-исчисление в следующую версию системы TermWare [13].

Дальнейшие исследования в данном направлении предполагают формулирование дополнительных условий, обеспечивающих корректность преобразований, а также реализацию переписывающих правил для проверки данных условий. Также возможна интеграция с системами для автоматического доказательства теорем, с целью более полной автоматизации процесса доказательства корректности.

1. *C. de Oliveira Braga*. Rewriting Logic as a Semantic Framework for Modular Structural Operational Semantics. Pontificia Universidade Catolica do Rio de Janeiro. 2001.
2. *Cirstea H., Liquori L., Wack B*. Rewriting calculus with fixpoints: Untyped and first-order systems. Vol. 3085, Springer, 2003.
3. *Oostrom V*. van. Lambda Calculus with Patterns. 1990.
4. *Jay C.B.* The pattern calculus. ACM Trans. Program. Lang. Syst. 2005.–Vol. 26, N 6. P. 911 – 937.
5. *Abadi M., Pierce B., Plotkin G.* Faithful Ideal Models for Recursive Polymorphic Types // International J. of Foundations of Computer Science. –1989. – Vol. 2. –P. 1 – 21.
6. *Faure G., Kirchner C.* Exceptions in the rewriting calculus / Proceedings of the RTA conference, Copenhagen, July 2002. – Lect. Notes in Comp. Sci. – Vol. 2378. – P. 66 – 82.
7. *Doroshenko A., Shevchenko R.* TermWare: A Rewriting Framework for Rule-Based Programming Dynamic Applications // Fundamenta Informaticae. – 2006. –Vol. 72, Issue 1-3. – P. 95 – 108.
8. *Дорошенко А.Е., Шевченко Р.С.* Система символьных вычислений для программирования динамических приложений // Проблеми програмування. – 2005. – №4. – С. 23 – 33.
9. *Baader F. and Nipkow T.* Term Rewriting and All That. Cambridge, England: Cambridge University Press, 1999. – 316 p.
10. *Mayr R., Nipkow T.* High Order Rewrite Systems and their confluence. Theoretical Com-

puter Science. – 1998. – Vol. 192, Issue 1. – P. 3 – 29.
11. *Girard J.-Y., Taylor P., Lafont Y.* Proofs and Types, Cambridge University Press, 1989. – 183 p.
12. *Visser E.*, A survey of rewriting strategies in program transformation systems, Proceedings of the Workshop on Reduction Strategies in Rewriting and Programming (B. Gramlichand, S.L.Alba, eds.), 2001.
13. *TermWare:* http://www.gradsoft.ua

Получено 03.03.2011

Об авторах:

Шевченко Руслан Сергеевич,
архитектор ПО,

Дорошенко Анатолий Ефимович,
доктор физико-математических наук,
профессор, заведующий отделом теории
компьютерных вычислений Института
программных систем НАН Украины.

***Место работы авторов*:**

ООО Gradsoft,
e-mail: ruslan@shevchenko.kiev.ua

Институт программных систем
НАН Украины,
03680, Киев-187,
Проспект Академика Глушкова, 40.
тел.: (044) 526 3559.
e-mail: dor@isofts.kiev.ua