Верификация функциональных программ методом построения диаграмм состояний

Андрей Миронов

ФИЦ "Информатика и Управление" Российской Академии Наук

amironov66@gmail.com

Аннотация

В настоящей статье рассматривается проблема верификации функциональных программ (ФП) над символьными строками, где спецификации свойств ФП определяются другими ФП, и ФП Σ_1 удовлетворяет спецификации, определяемой ФП Σ_2, если композиция функций, определяемых ФП Σ_1 и Σ_2, принимает значение 1 на всех аргументах. Мы вводим понятие диаграммы состояний ФП, и сводим проблему верификации ФП к проблеме анализа диаграмм состояний ФП. Предложенный подход иллюстрируется примером верификации ФП сортировки.

Ключевые слова: функциональная программа, диаграмма состояний, верификация.

1 Введение

Проблема верификации программ является одной из основных проблем теоретической информатики. Для различных классов программ используются различные методы верификации. Например, для верификации последовательных программ используются метод индуктивных утверждений Флойда [1], логика Хоара [2], и т.д. Для верификации параллельных и распределённых программ используются методы, основанные на исчислении взаимодействующих систем (CCS) и π-исчислении [3], [4], теории взаимодействующих последовательных процессов (CSP) и его обобщений [5], [6], темпоральной логике и методе model checking [7], процессной алгебре [8], сетях Петри [9], и т.д.

Методы верификации функциональных программ (ФП) развиты не так удовлетворительно, как методы верификации последовательных и параллельных программ. Одними из основных методов верификации ФП являются вычислительная индукция и структурная индукция [10]. Недостаток этих методов связан с трудностью построения доказательств правильности программ. Среди других методов верификации ФП следует отметить методы, основанные на рассуждениях с типами данных и абстрактной интерпретации через уточнение вывода типов [12], метод model checking для верификации ФП [13], [14], методы основанные на потоковом анализе [11], методы основанные на понятии мультипараметрического преобразователя деревьев [15].

В настоящей статье мы рассматриваем ФП как системы алгебраических уравнений над символьными строками, вводим понятие диаграммы состояний таких ФП и излагаем метод верификации, основанный на использовании диаграмм состояний ФП. Главные преимущества нашего подхода по сравнению со всеми вышеперечисленными подходами к верификации ФП заключаются в том, что наш подход позволяет представлять доказательства корректности ФП в виде простых свойств их диаграмм состояний.

Основная идея нашего подхода заключается в следующем. Мы предполагаем, что спецификация свойств верифицируемой ФП Σ_1 определяется другой ФП Σ_2, вход которой является выходом Σ_1, т.е. мы рассматриваем ФП $\Sigma_1 \circ \Sigma_2$, которая является последовательной композицией Σ_1 и Σ_2. Мы будем говорить, что Σ_1 корректна относительно спецификации Σ_2, если функция $f_{\Sigma_1 \circ \Sigma_2}$, которая соответствует $\Sigma_1 \circ \Sigma_2$ (т.е. композиция функций, определяемых Σ_1 и Σ_2) принимает значение 1 на всех аргументах. Мы сводим проблему доказательства соотношения $f_{\Sigma_1 \circ \Sigma_2} = 1$ к проблеме анализа диаграммы состояний ФП $\Sigma_1 \circ \Sigma_2$.

Предложенный метод верификации ФП иллюстрируется примером верификации ФП сортировки. Сначала мы излагаем полное доказательство корректности ФП сортировки методом структурной индукции (это сделано для сравнения сложности метода структурной индукции и сложности предложенного метода). Затем мы излагаем доказательство корректности ФП сортировки методом построения диаграммы состояний. Доказательство вторым методом существенно короче, и кроме того, оно может быть порождено автоматически. Это свидетельствует о преимуществах предложенного метода верификации ФП по сравнению с "ручной" верификацией, основанной на построении доказательства с применением метода математической индукции.

2 Основные понятия

2.1 Термы

Мы предполагаем, что заданы множества

- $\mathcal{D}$ **значений**, представляющее собой объединение $\mathcal{D}_{\mathbf{C}} \cup \mathcal{D}_{\mathbf{S}}$, причём
  - элементы $\mathcal{D}_{\mathbf{C}}$ называются **символами**, и
  - элементы $\mathcal{D}_{\mathbf{S}}$ называются **строками символов** (или просто **строками**), каждая строка из $\mathcal{D}_{\mathbf{S}}$ представляет собой конечную (возможно пустую) последовательность элементов $\mathcal{D}_{\mathbf{C}}$,
- $\mathcal{X}$ **переменных по данным** (называемых также просто **переменными**),
- $\mathcal{C}$ **констант**,
- $\mathcal{F}$ **функциональных символов (ФС)**, и
- Φ **функциональных переменных**,

причём каждому элементу m какого-либо из вышеперечисленных множеств сопоставлен некоторый **тип**, обозначаемый записью $type(m)$, и

- если $m \in \mathcal{D} \cup \mathcal{X} \cup \mathcal{C}$, то $type(m) \in \{\mathbf{C}, \mathbf{S}\}$,
- если $m \in \mathcal{F} \cup \Phi$, то $type(m)$ является записью вида $t_1 \times \ldots \times t_n \to t$, где $t_1, \ldots, t_n, t \in \{\mathbf{C}, \mathbf{S}\}$.

Если $d \in \mathcal{D}_{\mathbf{C}}$, то $type(d) = \mathbf{C}$, и если $d \in \mathcal{D}_{\mathbf{S}}$, то $type(d) = \mathbf{S}$.

Каждой константе $c \in \mathcal{C}$ соответствует элемент множества $\mathcal{D}_{type(c)}$, называемый **значением** этой константы. Запись ε обозначает константу типа $\mathbf{S}$, значением которой является пустая строка. Мы будем предполагать, что ε является единственной константой типа $\mathbf{S}$.

Каждому ФС $f \in \mathcal{F}$ сопоставлена частичная функция вида $\mathcal{D}_{t_1} \times \ldots \times \mathcal{D}_{t_n} \to \mathcal{D}_t$, где

$$type(f) = t_1 \times \ldots \times t_n \to t.$$

Данная функция обозначается тем же символом f.

Ниже мы перечисляем некоторые ФС, входящие в $\mathcal{F}$. Рядом с каждым ФС мы указываем (через двоеточие) его тип.

1. $head : \mathbf{S} \to \mathbf{C}$. Функция $head$ определена только на непустых строках, она сопоставляет каждой непустой строке её первый символ.
2. $tail : \mathbf{S} \to \mathbf{S}$. Функция $tail$ тоже определена только на непустых строках, она сопоставляет каждой непустой строке u строку, получаемую из u удалением первого символа (называемую **хвостом** u).
3. $conc : \mathbf{C} \times \mathbf{S} \to \mathbf{S}$. Для каждой пары $(a, u) \in \mathcal{D}_{\mathbf{C}} \times \mathcal{D}_{\mathbf{S}}$ строка $conc(a, u)$ представляет собой строку, получаемую приписыванием символа a спереди к строке u.
4. $empty : \mathbf{S} \to \mathbf{C}$. Функция $empty$ сопоставляет пустой строке символ 1, и каждой непустой строке – символ 0.
5. $=: \mathbf{C} \times \mathbf{C} \to \mathbf{C}$. Значение функции $=$ на паре (u, v) равно 1, если $u = v$, и 0 – иначе.
6. $\leq: \mathbf{C} \times \mathbf{C} \to \mathbf{C}$. Мы предполагаем, что на $\mathcal{D}_{\mathbf{C}}$ задано отношение линейного порядка, и значение функции $\leq$ на паре (u, v) равно 1, если $u \leq v$, и 0 – иначе.
7. Булевы ФС: $\neg : \mathbf{C} \to \mathbf{C}$, $\wedge : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$, и т.д., соответствующие им функции действуют стандартным образом на аргументы, состоящие из 0 и 1 (т.е. $\neg(1) = 0$, и т.д.) и не определены на других аргументах.
8. $if\_then\_else : \mathbf{C} \times t \times t \to t$, где $t = \mathbf{C}$ или $\mathbf{S}$ (т.е. записью if_then_else обозначены два ФС), и функции, соответствующие обоим ФС, определяются одинаково:

$$if\_then\_else\ (a, u, v) \stackrel{\text{def}}{=} \begin{cases} u, & \text{если } a = 1 \\ v, & \text{иначе.} \end{cases}$$

Понятие **терма** определяется индуктивно. С каждым термом e связан некоторый тип $type(e) \in \{\mathbf{C}, \mathbf{S}\}$. Каждая переменная по данным или константа является термом, тип которого совпадает с типом этой переменной или константы. Если $e_1, \ldots, e_n$ – список термов, и g – ФС или функциональная переменная, причем

$$type(g) = \ type(e_1) \times \ldots \times type(e_n) \to t$$

то запись $g(e_1, \ldots, e_n)$ является термом типа t.

Мы будем записывать термы

$$head(e),\ tail(e),\ conc(e_1, e_2),\ empty(e),$$
$$= (e_1, e_2),\ \leq (e_1, e_2),\ if\_then\_else\ (e_1, e_2, e_3)$$

в виде

$$e_h,\ e_t,\ e_1 e_2,\ e = \varepsilon,\ e_1 = e_2,\ e_1 \leq e_2,\ e_1\ ?\ e_2 : e_3$$

соответственно. Термы, содержащие булевы ФС, записываются так, как это принято в математических текстах (т.е. в виде $e_1 \wedge e_2$, и т.п.). Термы вида $e_1 \wedge \ldots \wedge e_n$ могут записываться также в виде $\{e_1, \ldots, e_n\}$.

2.2 Понятие функциональной программы над символьными строками

Функциональная программа над символьными строками (называемая ниже просто **функциональной программой (ФП)**) – это совокупность Σ

функциональных уравнений вида

$$\begin{cases} \varphi_1(x_{11},\ldots,x_{1n_1}) = e_1 \\ \ldots \\ \varphi_m(x_{m1},\ldots,x_{mn_m}) = e_m \end{cases} \quad (1)$$

где $\varphi_1,\ldots,\varphi_m$ – различные функциональные переменные, и для каждого $i = 1,\ldots,m$ $\varphi_i(x_{i1},\ldots,x_{in_i})$ и e_i – термы одинакового типа, причём

$$X_{e_i} = \{x_{i1},\ldots,x_{in_i}\}, \quad \Phi_{e_i} \subseteq \{\varphi_1,\ldots,\varphi_m\}.$$

Мы будем обозначать записью Φ_Σ совокупность всех функциональных переменных, входящих в Σ.

ФП (1) определяет список

$$(f_{\varphi_1},\ldots,f_{\varphi_m}) \quad (2)$$

функций, соответствующих функциональным переменным из Φ_Σ, являющийся наименьшим (в смысле порядка на списках частичных функций, описанного в [10]) решением системы функциональных уравнений (1) (данный список называется **наименьшей неподвижной точкой (ННТ) ФП** (1)). Значения этих фунций м.б. вычислены стандартной рекурсией. Мы будем предполагать, что для каждой рассматриваемой ФП все компоненты ННТ, соответствующей этой ФП, являются тотальными функциями. Первая функция из списка (2) (т.е. f_{φ_1}) обозначается записью f_Σ и называется **функцией, вычисляемой ФП** Σ. Если ФП Σ имеет вид (1), то $type(\Sigma)$ обозначает тип $type(e_1)$.

3 Пример спецификации и верификации ФП

3.1 Пример ФП

Рассмотрим в качестве примера ФП, определяющую функцию сортировки строк. Эта ФП имеет вид

$$\begin{aligned} &\mathbf{sort}(x) = (x = \varepsilon)?\ \varepsilon : \mathbf{insert}(x_h, \mathbf{sort}(x_t)) \\ &\mathbf{insert}(a,y) = (y = \varepsilon) \quad ?\ a\varepsilon \\ &\qquad\qquad\qquad\quad : (a \le y_h) \quad ?\ ay \\ &\qquad\qquad\qquad\qquad\quad : y_h\ \mathbf{insert}(a, y_t) \end{aligned} \quad (3)$$

В данной ФП определяются две функции:

- $\mathbf{sort} : \mathbf{S} \to \mathbf{S}$ – основная функция, и
- $\mathbf{insert} : \mathbf{C} \times \mathbf{S} \to \mathbf{S}$ – вспомогательная функция, которая отображает пару $(a, y) \in \mathbf{C} \times \mathbf{S}$ в строку, получаемую вставкой символа a в строку y, так чтобы если строка y упорядочена, то строка $\mathbf{insert}(a, y)$ тоже была бы упорядоченной. (мы называем строку упорядоченной, если ее компоненты образуют неубывающую последовательность)

3.2 Пример спецификации ФП

Одно из свойств корректности ФП, изложенной в предыдущем пункте, заключается в следующем: для каждой строки $x \in \mathbf{S}$ строка $\mathbf{sort}(x)$ упорядочена. Это свойство можно формально выразить следующим образом. Рассмотрим ФП, определяющую функцию **ord** проверки упорядоченности строки:

$$\begin{aligned} &\mathbf{ord}(x) = \\ &= (x = \varepsilon) \quad ?\ 1 \\ &\qquad : (x_t = \varepsilon) \quad ?\ 1 \\ &\qquad\qquad : (x_h \le (x_t)_h) \quad ?\ \mathbf{ord}(x_t) \\ &\qquad\qquad\qquad\quad : 0 \end{aligned} \quad (4)$$

Функция **ord** позволяет выразить вышеупомянутое свойство корректности в виде следующего математического утверждения:

$$\forall\, x \in \mathbf{S} \quad \mathbf{ord}(\mathbf{sort}(x)) = 1 \quad (5)$$

3.3 Пример верификации ФП

Задача верификации свойства корректности ФП, описанного в предыдущем пункте, заключается в построении формального доказательства утверждения (5). Данное утверждение может быть доказано как обычная математическая теорема, например с использованием метода математической индукции. Доказательство этого утверждения может иметь следующий вид.

Если $x = \varepsilon$, то, согласно первому уравнению системы (3), верно равенство $\mathbf{sort}(x) = \varepsilon$, и поэтому

$$\mathbf{ord}(\mathbf{sort}(x)) = \mathbf{ord}(\varepsilon) = 1$$

Пусть $x \neq \varepsilon$. Докажем равенство (5) для этого случая методом математической индукции. Предположим, что верно равенство, получаемое из равенства в (5) заменой x на любую строку, длина которой меньше длины x. Докажем, что в этом случае равенство в (5) также будет верным.

Равенство в (5) можно переписать в виде

$$\mathbf{ord}(\ \mathbf{insert}(x_h, \mathbf{sort}(x_t))) = 1 \quad (6)$$

По индуктивному предположению, верно равенство

$$\mathbf{ord}(\mathbf{sort}(x_t)) = 1$$

из которого следует (6) по нижеследующей лемме.

Лемма.
Имеет место импликация

$$\mathbf{ord}(y) = 1 \quad \Rightarrow \quad \mathbf{ord}(\mathbf{insert}(a, y)) = 1 \quad (7)$$

Доказательство.
Доказываем лемму индукцией по длине y.

Если $y = \varepsilon$, то правая часть в (7) имеет вид

$$\mathbf{ord}(a\varepsilon) = 1$$

что верно по определению **ord**.

Пусть $y \neq \varepsilon$, и для каждой строки z, длина которой меньше длины y, верна импликация

$$\mathbf{ord}(z) = 1 \quad \Rightarrow \quad \mathbf{ord}(\mathbf{insert}(a, z)) = 1 \qquad (8)$$

Обозначим $c \stackrel{\text{def}}{=} y_h$, $d \stackrel{\text{def}}{=} y_t$.
(7) имеет вид

$$\mathbf{ord}(cd) = 1 \quad \Rightarrow \quad \mathbf{ord}(\mathbf{insert}(a, cd)) = 1 \qquad (9)$$

Для доказательства импликации (9) нужно доказать, что при условии $\mathbf{ord}(cd) = 1$ верны импликации

(a) $a \leq c \quad \Rightarrow \quad \mathbf{ord}(a(cd)) = 1$,

(b) $c < a \quad \Rightarrow \quad \mathbf{ord}(c\ \mathbf{insert}(a, d)) = 1$.

(a) верно потому, что из $a \leq c$ следует

$$\mathbf{ord}(a(cd)) = \mathbf{ord}(cd) = 1.$$

Докажем (b).

- $d = \varepsilon$. В этом случае правая часть в (b) имеет вид
$$\mathbf{ord}(c(a\varepsilon)) = 1 \qquad (10)$$
(10) следует из $c < a$.

- $d \neq \varepsilon$. Обозначим $p \stackrel{\text{def}}{=} d_h$, $q \stackrel{\text{def}}{=} d_t$.
В этом случае надо доказать, что при $c < a$
$$\mathbf{ord}(c\ \mathbf{insert}(a, pq)) = 1 \qquad (11)$$

  1. Если $a \leq p$, то (11) имеет вид
  $$\mathbf{ord}(c(a(pq))) = 1 \qquad (12)$$
  Т.к. $c < a \leq p$, то (12) следует из равенств
  $$\mathbf{ord}(c(a(pq))) = \mathbf{ord}(a(pq)) = \mathbf{ord}(pq) = \\ = \mathbf{ord}(c(pq)) = \mathbf{ord}(cd) = 1$$

  2. Если $p < a$, то (11) имеет вид
  $$\mathbf{ord}(c(p\ \mathbf{insert}(a, q))) = 1 \qquad (13)$$
  Поскольку по предположению
  $$\mathbf{ord}(cd) = \mathbf{ord}(c(pq)) = 1$$
  то $c \leq p$, и поэтому (13) можно переписать в виде
  $$\mathbf{ord}(p\ \mathbf{insert}(a, q)) = 1 \qquad (14)$$

  При $p < a$
  $$\mathbf{insert}(a, d) = \mathbf{insert}(a, pq) = p\ \mathbf{insert}(a, q)$$
  поэтому (14) можно переписать в виде
  $$\mathbf{ord}(\mathbf{insert}(a, d)) = 1 \qquad (15)$$
  (15) следует по индуктивному предположению для леммы (т.е. из импликации (8), в которой $z \stackrel{\text{def}}{=} d$) из равенства
  $$\mathbf{ord}(d) = 1$$
  которое обосновывается цепочкой равенств
  $$1 = \mathbf{ord}(cd) = \mathbf{ord}(c(pq)) = \quad \text{(т.к. } c \leq p) \\ = \mathbf{ord}(pq) = \mathbf{ord}(d). \quad \blacksquare$$

Из рассмотренного примера мы видим, что даже для простейшей ФП из нескольких строк доказательство ее корректности представляет собой довольно нетривиальное математическое рассуждение, его сложно проверить и гораздо сложнее построить. Кроме того, в настоящее время не существует математического подхода на основе которого данное доказательство могло бы быть порождено автоматически.

Ниже мы излагаем другой метод для верификации ФП, основанный на построении диаграмм состояний функциональных программ, и иллюстрируем данный метод путем доказательства на его основе утверждения (5). Это доказательство существенно короче доказательства, приведённого в настоящем пункте. Кроме того, данное доказательство может быть порождено автоматически, что является убедительным свидетельством преимущества предложенного метода для верификации ФП.

4 Диаграммы состояний ФП

4.1 Понятия и обозначения, связанные с термами

Ниже будут использоваться следующие обозначения.

- Мы будем обозначать записью $\mathcal{E}$ множество всех термов, и записью $\mathcal{E}_0$ – множество всех термов, не содержащих функциональных переменных.

- Для каждого $e \in \mathcal{E}$ записи X_e и Φ_e обозначают множество всех переменных по данным и функциональных переменных соответственно, входящих в e.

- Если $e \in \mathcal{E}$, $x_1, \ldots, x_n$ – список различных переменных, и $e_1, \ldots, e_n$ – термы, причём $\forall\, i = 1, \ldots, n$ $type(e_i) = type(x_i)$, то запись
$$e(e_1/x_1, \ldots, e_n/x_n) \qquad (16)$$

обозначает терм, получаемый из e заменой для каждого $i \in \{1,\dots,n\}$ каждого вхождения x_i в e на терм e_i.

- Если e и e' – термы, то для каждого терма e'', тип которого равен типу терма e', запись $e(e''/e')$ обозначает терм, получаемый из e заменой всех вхождений e' в e на терм e''.

- Если $X \subseteq \mathcal{X}$, то **означиванием** переменных из X называется соответствие ξ, сопоставляющее каждой переменной $x \in X$ некоторое значение $x^\xi \in \mathcal{D}_{type(x)}$. Множество всех означиваний переменных из X будем обозначать записью $X^\bullet$.

- Для каждого $e \in \mathcal{E}_0$, каждого $X \supseteq X_e$ и каждого $\xi \in X^\bullet$ запись e^ξ обозначает объект, называемый **значением** e на ξ и определяемый стандартным образом (т.е. если $e \in \mathcal{C}$, то e^ξ равно значению константы e, если $e \in \mathcal{X}$, то e^ξ равно значению означивания ξ на переменной e, и если $e = f(e_1,\dots,e_n)$, то $e^\xi = f(e_1^\xi,\dots,e_n^\xi)$).

- Мы будем считать термы $e_1, e_2 \in \mathcal{E}_0$ равными, если для каждого $\xi \in (X_{e_1} \cup X_{e_2})^\bullet$ верно равенство $e_1^\xi = e_2^\xi$, которое мы понимаем в следующем смысле: значения e_1^ξ и e_2^ξ либо оба не определены, либо оба определены и совпадают.

- Терм $e \in \mathcal{E}_0$ называется **формулой**, если все переменные из X_e имеют тип $\mathbf{C}$ и $\forall\, \xi \in X_e^\bullet \quad e^\xi \in \{0,1\}$. Символ $\mathcal{B}$ обозначает множество всех формул. Символы $\top$ и $\bot$ обозначают формулы, принимающие на произвольном означивании значение 1 и 0 соответственно.

4.2 Понятие состояния ФП

Пусть задана ФП Σ.

Обозначим записью $\mathcal{E}_{conc}$ множество термов из $\mathcal{E}_0$, в которых нет других ФС кроме *conc*, и записью $\mathcal{E}_\Sigma$ – множество термов, каждый из которых либо является переменной, либо имеет вид $\varphi(u_1,\dots,u_n)$, где $\varphi \in \Phi_\Sigma$ и $u_1,\dots,u_n \in \mathcal{E}_{conc}$.

Будем называть **присваиванием** запись вида

$$u := e \tag{17}$$

где $u \in \mathcal{E}_{conc}$, $e \in \mathcal{E}_\Sigma$, $type(u) = type(e)$.

Состоянием ФП Σ называется запись s вида

$$b.u\,(\theta_1,\dots,\theta_m) \tag{18}$$

компоненты которой определяются следующим образом:

- b – формула из $\mathcal{B}$, называемая **условием** s,
- u – терм из $\mathcal{E}_{conc}$, называемый **значением** s, и
- $\theta_1,\dots,\theta_m$ – присваивания.

Будем использовать следующие обозначения.

- S_Σ обозначает совокупность всех состояний Σ.
- Если состояние $s \in S_\Sigma$ имеет вид (18), то мы будем обозначать через b_s, u_s, Θ_s и $type(s)$ формулу b, терм u, последовательность присваиваний (которая м.б. пустой) в (18), и тип $type(u)$ соответственно.

  Если $b_s = \top$, то формулу b в (18) мы будем опускать.
- Если $s \in S_\Sigma$, то
  - X_s обозначает множество всех переменных по данным, входящих в s,
  - переменные из X_s, входящие в левую часть какого-либо присваивания из Θ_s, называются **внутренними переменными** s, все остальные переменные из X_s называются **входными переменными** s,
  - $s^\bullet$ обозначает множество всех $\xi \in X_s^\bullet$, таких, что $b_s^\xi = 1$, и $\forall\,(u_i := e_i) \in \Theta_s$
    - если $e_i \in \mathcal{E}_{conc}$, то $u_i^\xi = e_i^\xi$, и
    - если $e_i = \varphi(v_1,\dots,v_n)$, то

      $$u_i^\xi = f_\varphi(v_1^\xi,\dots,v_n^\xi),$$

      где f_φ – соответствующая компонента НИТ ФП Σ.

Состояние $s \in S_\Sigma$ называется **терминальным**, если Θ_s не содержит функциональных переменных.

Пусть задана пара состояний $s_1, s_2 \in S_\Sigma$. Будем обозначать записью $s_1 \subseteq s_2$ следующее утверждение: множества входных переменных s_1 и s_2 совпадают, и

$$\forall\, \xi_1 \in s_1^\bullet\ \exists\, \xi_2 \in s_2^\bullet : u_{s_1}^{\xi_1} = u_{s_2}^{\xi_2}.$$

Наряду с состояниями ФП мы будем рассматривать также **псевдосостояния**, которые отличаются от состояний ФП лишь тем, что присваивания в них имеют вид $u := e$, где $u \in \mathcal{E}_{conc}$, $e \in \mathcal{E}$. Для каждого псевдосостояния s записи b_s, u_s и Θ_s имеют тот же смысл, что и для состояний ФП.

4.3 Раскрытия состояний ФП

Пусть заданы ФП Σ, состояние $s \in S_\Sigma$, и присваивание $\theta \in \Theta_s$, причём θ имеет вид $u := \varphi(v_1,\dots,v_n)$, и уравнение в ФП Σ, соответствующее φ, имеет вид $\varphi(x_1,\dots,x_n) = e_\varphi$.

Обозначим записью s^θ множество, называемое **раскрытием** состояния s относительно θ, и определяемое процедурой его построения, которая состоит из перечисляемых ниже действий.

Действие 1.

s^θ полагается равным одноэлементному множеству, которое состоит из псевдосостояния, получаемого из s заменой θ на присваивание

$$u := e_\varphi(v_1/x_1, \ldots, v_n/x_n).$$

Действие 2.

(Данное действие может выполняться несколько раз, до тех пор пока есть возможность его выполнять.)
Если все элементы множества s^θ являются состояниями из S_Σ, то выполнение данного действия завершается, иначе s^θ модифицируется следующим образом.

Выберем произвольный элемент $s' \in s^\theta$, который не является состоянием из S_Σ, и обозначим записью θ' первое из присваиваний, входящих в $\Theta_{s'}$, которое имеет вид $u := e$, где $e \notin \mathcal{E}_\Sigma$. Рассмотрим все возможные варианты вида терма e, и для каждого из этих вариантов укажем правило модификации множества s^θ, соответствующее этому варианту. Ниже словосочетание "новая переменная" означает "переменная, не имеющая вхождений в рассматриваемое псевдосостояние".

- $e \in \mathcal{C}$, в этом случае
  - если $u = e$, то удаляем θ' из s',
  - если $u \in \mathcal{X}$, то заменяем все вхождения u в s' на e, и удаляем θ' из s',
  - иначе удаляем s' из s^θ.
- $e = e'_h$, в этом случае заменяем θ' на присваивание
  - $u := e_1$, если e' имеет вид $e_1 e_2$,
  - $ux := e'$, где x – новая переменная, иначе.
- $e = e'_t$, в этом случае заменяем θ' на присваивание
  - $u := e_2$, если e' имеет вид $e_1 e_2$,
  - $xu := e'$, где x – новая переменная, иначе.
- $e = e_1 e_2$, в этом случае
  - если $u = u_1 u_2$, то заменяем θ' на пару присваиваний $u_1 := e_1$, $u_2 := e_2$,
  - если $u \in \mathcal{X}$, то заменяем все вхождения u в s' на терм xy (где x и y – новые переменные), и θ' – на пару присваиваний $x := e_1$, $y := e_2$,
  - иначе – удаляем s' из s^θ.
- $e = (e_1 = \varepsilon)$, в этом случае
  - добавляем к s^θ копию состояния s' (обозначим её s''),
  - заменяем
    - θ' в s' на пару $u := 1$, $\varepsilon := e_1$, и
    - θ' в s'' – на пару $u := 0$, $xy := e_1$, где x и y – новые переменные.
- $e = (e_1 = e_2)$, $e = (e_1 \le e_2)$, $e = (e_1 \wedge e_2)$ и т.п., в этом случае
  - заменяем θ' на пару $x_1 := e_1$, $x_2 := e_2$, где x_1, x_2 – новые переменные, и
  - добавляем к $b_{s'}$ коньюнктивный член $u = e'$, где e' получается из e заменой e_i на x_i $(i = 1, 2)$.
- $e = (e_1 ? e_2 : e_3)$, в этом случае добавляем к s^θ копию s' (обозначим её s''), и заменяем все вхождения
  - θ' в s' на пару $1 := e_1$, $u := e_2$,
  - θ' в s'' на пару $0 := e_1$, $u := e_3$.
- $e = \varphi(e_1, \ldots, e_k)$, $\exists\, i : e_i \notin \mathcal{E}_{conc}$, в этом случае заменяем e_i в θ' на новую переменную x, и добавляем $x := e_i$ перед θ'.

Действие 3.

Для каждого $s' \in s^\theta$

- если в $\Theta_{s'}$ есть пара вида $u := x$, $v := x$, где $x \in \mathcal{X}$, и u, v имеют вид $u_1 \ldots u_n$, $v_1 \ldots v_m$ соответственно, то выполняется алгоритм, состоящий из следующих шагов:
(в результате выполнения каждого из этих шагов вид данных присваиваний меняется, но мы будем обозначать изменившиеся присваивания теми же записями что и исходные присваивания):
  - если $n < m$, то в случае $u_n \in \mathcal{X}$ каждое вхождение переменной u_n в s' заменяем на терм $v_n \ldots v_m$, а в случае $u_n = \varepsilon$ удаляем s' из s^θ,
  - аналогично – при $m < n$,
  - $\forall\, i = 1, \ldots, n$:
    - если $u_i \in \mathcal{X}$, то заменяем все вхождения u_i в s' на v_i, а если $u_i \notin \mathcal{X}$, но $v_i \in \mathcal{X}$, то заменяем все вхождения v_i в s' на u_i,
    - если $u_i \ne v_i$, то удаляем s' из s^θ,
  - удаляем одно из рассматриваемых присваиваний,
- если $b_{s'} = \{b', x = u\}$, где $x \in \mathcal{X}$, $u \in \mathcal{X} \cup \mathcal{C}$, то $b_{s'}$ заменяется на b', и все вхождения x в s' заменяются на u,
- $b_{s'}$ упрощается путем
  - замены подтермов без переменных на равные им константы, и

– применения упрощающих преобразований, связанных булевскими тождествами и свойствами отношений равенства и линейного порядка,

- если $b_{s'} = \bot$, то s' удаляется из s^θ.

Теорема 1.
Описанная выше процедура построения множества s^θ всегда завершается. ■

Состояние $s \in S_\Sigma$ называется **противоречивым**, если оно нетерминально, и $\exists\, \theta \in \Theta_s$: либо $s^\theta = \emptyset$, либо все состояния из s^θ противоречивы.

4.4 Подстановка состояний в термы

Пусть заданы ФП Σ терм $e \in \mathcal{E}$, список $x_1, \ldots, x_n$ различных переменных из $\mathcal{X}$, и $s_1, \ldots, s_n$ – список состояний из S_Σ, причём $\forall i = 1, \ldots, n \quad type(s_i) = type(x_i)$. Будем обозначать записью

$$e(s_1/x_1, \ldots, s_n/x_n) \tag{19}$$

состояние $s_e \in S_\Sigma$, определяемое индукцией по структуре e:

- если $e = x_i \in \{x_1, \ldots, x_n\}$, то $s_e \stackrel{\text{def}}{=} s_i$,
- если $e \in \mathcal{X} \setminus \{x_1, \ldots, x_n\}$ или $e \in \mathcal{C}$, то $s_e \stackrel{\text{def}}{=} e\,()$,
- если $e = g(e_1, \ldots, e_k)$, где $g \in \mathcal{F} \cup \Phi$, и состояния $s_{e_1}, \ldots, s_{e_k}$ вида (19), соответствующие термам $e_1, \ldots, e_k$, уже определены, то s_e определяется следующим образом:
  - внутренние переменные состояний s_{e_i} заменяются на новые переменные стандартным образом, так, чтобы все внутренние переменные этих состояний были различными, обозначим получившиеся состояния записями $b_i.u_i(\Theta_i)$ $(i = 1, \ldots, k)$,
  - s_e является результом применения действий 2 и 3 из пункта (4.3) к состоянию

$$\{b_1, \ldots, b_k\}.g(u_1, \ldots, u_k)\,(\Theta_1, \ldots, \Theta_k).$$

Терм (19) обозначается записью $e(s_1, \ldots, s_n)$, в том случае, когда список переменных $x_1, \ldots, x_n$ ясен из контекста.

4.5 Понятие диаграммы состояний ФП

Пусть Σ – ФП, и левая часть первого уравнения в Σ имеет вид $\varphi(x_1, \ldots, x_n)$.

Диаграммой состояний (ДС) ФП Σ называется граф G с выделенной вершиной n_0 (называемой **начальной вершиной**), удовлетворяющий следующим условиям.

- Каждой вершине n графа G сопоставлено некоторое состояние $s_n \in S_\Sigma$, причём s_{n_0} имеет вид

$$y\,(y := \varphi(x_1, \ldots, x_n)), \quad \text{где } y \notin \{x_1, \ldots, x_n\}.$$

- Для каждой вершины n графа G верно одно из следующих утверждений.
  1. Из n не выходит ни одного ребра, и s_n терминально.
  2. Из n выходят два ребра, и состояния s', s'', соответствующие концам этих рёбер, обладают следующим свойством: $\exists x \in X_{s_n}$: $type(x) = \mathbf{S}$, в Θ_{s_n} нет присваиваний вида $u := x$, и s', s'' получаются из s_n путём
     - замены всех вхождений x на константу ε и на терм yz соответственно (где y и z – переменные, не входящие в X_{s_n}), и
     - если x не входит в левую часть ни одного присваивания из Θ_{s_n}, то – добавления присваиваний $\varepsilon := x$ и $yz := x$ к $\Theta_{s'}$ и $\Theta_{s''}$ соответственно.
  3. $\exists \theta \in \Theta_{s_n}$: множество состояний, соответствующих концам рёбер, выходящих из n, совпадает с множеством всех непротиворечивых состояний из s_n^θ.
  4. u_{s_n} имеет вид $u_1 u_2$, и из n выходит одно ребро с меткой $tail$, конец n' которого удовлетворяет условию: $tail(s_n) \subseteq s_{n'}$.
  5. Из n выходит одно ребро, с меткой $<$, конец n' которого удовлетворяет условию:
     - $\exists\, n_1, n_2$: G содержит ребро из n_1 в n_2 с меткой $tail$, и
     - $\exists\, e \in \mathcal{E}_\Sigma$, $\exists\, x \in X_e$:

$$s_n \subseteq e(tail(s_1)/x), \quad e(s_2/x) \subseteq s_{n'}.$$

Опишем неформальный смысл понятия ДС. Каждое состояние s можно рассматривать как описание процесса вычисления значения u_s на конкретных значениях входных переменных этого состояния (путем выполнения присваиваний из Θ_s, проверки условия b_s и вычисления значения терма u_s на вычисленных значениях переменных, входящих в этот терм). Если из состояния n выходят рёбра без меток, то концы этих ребер соответствуют возможным вариантам вычисления значения u_{s_n} (путем различной детализации структуры значения некоторой переменной из X_{s_n}, или путем эквивалентного преобразования какого-либо присваивания из Θ_{s_n}). Если из состояния n выходит ребро с меткой $tail$ в состояние n', то это ребро выражает сведение задачи вычисления хвоста значения u_{s_n} к вычислению значения $u_{s_{n'}}$. Если из состояния n выходит ребро с меткой $<$ в состояние n', то это ребро выражает сведение задачи вычисления значения u_{s_n} к вычислению значения $u_{s_{n'}}$ на аргументах меньшего размера.

Будем говорить, что ФП Σ **имеет конечную ДС**, если существует ДС ФП Σ, множество вершин которой является конечным.

Теорема 2.
Пусть Σ_1 и Σ_2 имеют конечные ДС, $\Phi_{\Sigma_1} \cap \Phi_{\Sigma_2} = \emptyset$, и левые части первых уравнений в Σ_1 и Σ_2 имеют вид $\varphi_1(x_1, \ldots, x_n)$ и $\varphi_2(y_1, \ldots, y_m)$ соответственно, причём $type(\Sigma_1) = type(y_1)$. Тогда ФП Σ, первое уравнение в которой имеет вид

$$\varphi(x_1, \ldots, x_n, y_2, \ldots, y_m) = \\ = \varphi_2(\varphi_1(x_1, \ldots, x_n), y_2, \ldots, y_m)$$

а совокупность остальных уравнений равна $\Sigma_1 \cup \Sigma_2$, тоже имеет конечную ДС. ■

Мы не даем описание алгоритма построение ДС для Σ ввиду ограничений на размер статьи. Отметим лишь, что данная ДС представляет собой объединение ДС для Σ_1, ДС для Σ_2, и ДС, являющейся декартовым произведением двух предыдущих ДС.

Теорема 3.
Пусть ФП Σ имеет конечную ДС Σ, причем значения состояний, соответствующих тем терминальным вершинам этой ДС, которые достижимы из начального состояния, равны 1. Тогда f_Σ принимает значение 1 на всех своих аргументах. ■

Сформулированные выше теоремы являются теоретическим обоснованием метода применения ДС для верификации функциональных программ. Суть данного метода заключается в попытке построения конечных ДС для анализируемой функции, описываемой ФП Σ_1, и проверочной функции, описываемой ФП Σ_2. Если для этих ФП удалось построить конечные ДС, то, согласно теореме 2, для ФП, соответствующей суперпозиции этих функций, тоже можно построить конечную ДС. Если эта ДС обладает свойством, указанным в теореме 3, то суперпозиция функций, соответствующих Σ_1 и Σ_2, принимает значение 1 на всех своих аргументах. В следующем параграфе мы приводим пример использования этого метода.

Для построения ДС используется метод обоснования соотношений вида $s_1 \subseteq s_2$, который мы не излагаем здесь ввиду ограничений на размер статьи. Отметим лишь, что, этот метод использует понятие унификации термов.

При изображении ДС ФП мы будем использовать следующее соглашение: если состояние s, сопоставленное какой-либо вершине, имеет вид $b.u(\theta_1, \ldots, \theta_n)$, то данная вершина обозначается в виде овала, над которым нарисована запись $b.u$ (или u, если $b = \top$), внутри которого перечислены компоненты списка Θ_s, и слева от которого м.б. нарисован символ, являющийся идентификатором этой вершины.

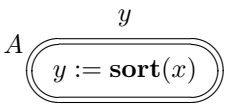
5 Пример верификации ФП методом построения ДС

В этом пункте мы иллюстрируем изложенный выше метод верификации ФП примером верификации ФП сортировки, в данном случае $\Sigma_1 = (3)$ и $\Sigma_2 = (4)$. Будем использовать следующее соглашение: если в ДС вершины n_1 и n_2 таковы, что n_2 можно получить из n_1 путем выполнения последовательности действий 2 и 3 из определения ДС, то из n_1 в n_2 можно нарисовать ребро без метки (т.е. рёбра без меток в новом понимании ДС соответствуют путям из рёбер без меток в исходном понимании ДС).

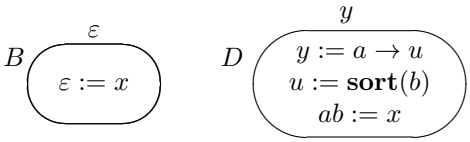
5.1 Диаграмма состояний ФП сортировки

В этом пункте мы описываем процесс построения ДС ФП (3). Термы вида $\mathbf{insert}(a, y)$ мы будем обозначать записью $a \to y$.

Начальная вершина ДС ФП (3) (обозначим эту вершину идентификатором A) имеет вид



Из этой вершины можно нарисовать два ребра без меток (соответствующих замене x на ε и ab, и раскрытию одного присваивания) с концами в вершинах



Из D тоже можно нарисовать два ребра без меток (соответствующих замене переменной u на константу ε и на терм cd): конец первого ребра имеет метку

$$y\,(y := a \to cd, cd := \mathbf{sort}(b), ab := x), \qquad (20)$$

а конец второго ребра имеет метку

$$y\,(y := a\varepsilon, \varepsilon := \mathbf{sort}(b), ab := x). \qquad (21)$$

Из вершины с меткой (20), согласно пункту 3 в определении ДС (раскрытие первого присваивания), можно нарисовать два ребра с концами

$$C: \quad \{a \le c\}.acd\,(cd := \mathbf{sort}(b), ab := x),$$
$$G: \quad \{c < a\}.cz\,(z := a \to d, cd := \mathbf{sort}(b), ab := x).$$

Из C можно провести ребро с меткой $tail$ в начальную вершину (мы не приводим описание метода, на основании которого устанавливается существование такого ребра, отметим лишь, что в данном случае существование такого ребра усматривается непосредственно).

Из вершины с меткой (21), согласно пункту 2 в определении ДС (замена b на ε и на pq) можно получить две вершины, одна из которых терминальна и имеет вид

$$E: \quad a\varepsilon\ (a\varepsilon := x),$$

а вторая вершина будет противоречивой (что можно установить путем дополнительных раскрытий, которые мы здесь не приводим).

Из G можно нарисовать два ребра без меток (соответствующих замене b на ε и на pq), первая вершина будет противоречивой, а вторая имеет метку

$$\{c < a\}.\, cz \left(\begin{array}{l} z := a \to d, \\ cd := p \to w, \\ w := \mathbf{sort}(q), \\ apq := x \end{array} \right). \tag{22}$$

Из (22) можно нарисовать два ребра без меток (соответствующих замене w на ε и на ij):

- из конца первого из этих ребер можно нарисовать несколько ребер без меток, но среди концов этих ребер непротиворечивой будет только вершина с меткой

  $$\{c < a\}.\, cz \left(\begin{array}{l} z := a \to \varepsilon, \\ c := p, \\ d := \varepsilon, \\ ap\varepsilon := x \end{array} \right),$$

  из которой существует единственное ребро без метки в терминальную вершину

  $$H: \quad \{c < a\}.\, ca\varepsilon\ (ac\varepsilon := x),$$

- конец второго из этих ребер имеет метку

  $$\{c < a\}.\, cz \left(\begin{array}{l} z := a \to d, \\ cd := p \to ij, \\ ij := \mathbf{sort}(q), \\ apq := x \end{array} \right). \tag{23}$$

Из (23) можно нарисовать пару ребер, концы которых имеют метки

$$F: \quad \{c < a, c \le i\}.\, cz \left(\begin{array}{l} z := a \to ij, \\ ij := \mathbf{sort}(q), \\ acq := x \end{array} \right),$$

$$I: \quad \{c < a, c < p\}.\, cz \left(\begin{array}{l} z := a \to d, \\ d := p \to j, \\ cj := \mathbf{sort}(q), \\ apq := x \end{array} \right).$$

Из F можно провести ребро с меткой *tail* в начальную вершину (существование такого ребра усматривается непосредственно).

Нетрудно заметить, что пара вершин (D, I) связана с парой вершин (A, G) следующими соотношениями:

$$tail(I) = e(tail(G)/h), \quad D = e(A/h) \tag{24}$$

где $e = a \to h$. Другими словами, метки вершин вершин I, D можно получить из меток вершин G, A путем приписывания дополнительного присваивания сверху. Данный факт может быть использован для обоснования существования ребра с меткой *tail* из G в A. Мы не излагаем детального описания метода обоснования существования такого ребра, опишем лишь схему такого обоснования. Обозначим записью $\rho(x)$ частичную функцию, обладающую следующим свойством: если ρ определена на некотором значении α переменной x, то она преобразует α в такую строку β, которая обладает свойством

$$u^{x \mapsto \alpha}_{tail(G)} = u^{x \mapsto \beta}_{A}.$$

Из соотношения (24) непосредственно выводится следующее свойство функции ρ:

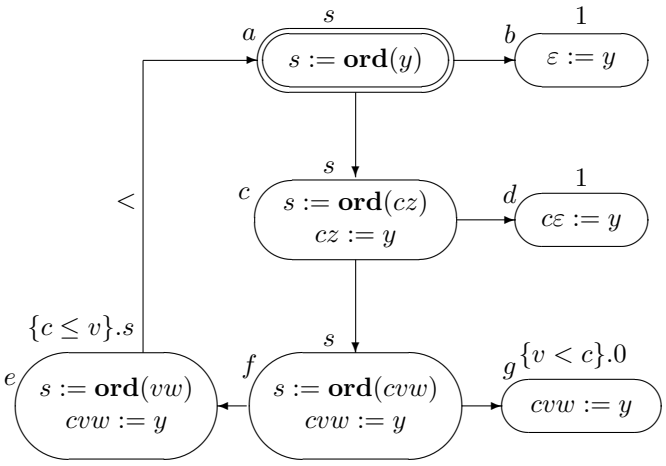
$$x \neq \varepsilon \ \Rightarrow\ \rho(x) \sqsupseteq x_h \rho(x_t) \tag{25}$$

где неравенство $\sqsupseteq$ понимается как отношение порядка на множестве частично определенных функций: если для некоторого значения x правая часть (25) определена, то левая часть на этом значении x тоже определена и значения обоих частей совпадают. Тотальность функции ρ обосновывается неравенством (25) и анализом уже построенного фрагмента ДС для (3). Отметим, что это обоснование может быть порождено автоматически. Доказательство корректности данного обоснования использует понятие унификации пар состояний, его описание занимает большой объем и мы в данном тексте его не приводим.

Построенная ДС для ФП (3) представлена на рис. 1, её можно упростить до ДС на рис. 2.

5.2 Диаграмма состояний ФП проверки упорядоченности строки

Фрагмент ДС для ФП Σ_2 (см. (4)) (состоящий из вершин, достижимых из начального состояния) имеет вид



5.3 Диаграмма состояний для суперпозиции ФП сортировки и ФП проверки упорядоченности строки

Существует алгоритм, основанный на теореме 3, который можно применить к ДС для (3) и (4), и получить ДС, изображенную на рис. 3. Данная ДС имеет две терминальные вершины, метки обоих этих вершин имеют значение 1. Согласно теореме 3, отсюда следует утверждение о том, что функция $\mathbf{ord} \circ \mathbf{sort}$ принимает значение 1 на всех своих аргументах.

В заключение отметим, что несмотря на сложность всех вышеприведенных преобразований и обоснований, все они могут быть порождены автоматически. Попытка обоснования существования ребер с метками *tail* и $<$ производится автоматически для каждой пары вершин, возникающих в процессе построения ДС. Как видно из рассмотренного примера построения ДС для программы сортировки, процесс построения ДС заканчивается достаточно быстро.

6 Заключение

Мы предложили понятие диаграммы состояний функциональных программ (ФП) и основанный на этом понятии метод верификации ФП. Одна из проблем для дальнейших исследований, связанных с понятием ДС, имеет следующий вид: найти необходимое и достаточное условие (или как можно более сильное достаточное условие), которому должна удовлетворять ФП, чтобы эта ФП имела конечную ДС.

Список литературы

[1] R.W. Floyd: Assigning meanings to programs. In J.T. Schwartz, editor, Proceedings Symposium in Applied Mathematics, Mathematical Aspects of Computer Science, pages 19-32. AMS, 1967.

[2] C. A. R. Hoare: An axiomatic basis for computer programming. Communications of the ACM, 12(10): 576–580, 583, October 1969.

[3] R. Milner: A Calculus of Communicating Systems. Number 92 in Lecture Notes in Computer Science. Springer Verlag, 1980.

[4] R. Milner: Communicating and Mobile Systems: the π-Calculus. Cambridge University Press, 1999.

[5] Hoare, C. A. R.: Communicating sequential processes. Communications of the ACM 21 (8): 666–677, 1978.

[6] Separation Logic: A Logic for Shared Mutable Data Structures. John C. Reynolds. LICS 2002.

[7] Clarke, E.M., Grumberg, O., and Peled, D.: Model Checking. MIT Press, 1999.

[8] J.A. Bergstra, A. Ponse, and S.A. Smolka, editors: Handbook of Process Algebra. North-Holland, Amsterdam, 2001.

[9] C.A. Petri: Introduction to general net theory. In W. Brauer, editor, Proc. Advanced Course on General Net Theory, Processes and Systems, number 84 in LNCS, Springer Verlag, 1980.

[10] Z. Manna: Mathematical Theory of Computation. McGraw-Hill Series in Computer Science,1974.

[11] N. D. Jones and N. Andersen. Flow analysis of lazy higher-order functional programs. Theoretical Computer Science, 375:120–136, 2007.

[12] Ranjit Jhala, Rupak Majumdar, Andrey Rybalchenko: HMC: Verifying Functional Programs Using Abstract Interpreters, `http://arxiv.org/abs/1004.2884`

[13] N. Kobayashi and C.-H. L. Ong. A type theory equivalent to the modal mu- calculus model checking of higher-order recursion schemes. In Proceedings of LICS 2009. IEEE Computer Society, 2009.

[14] C.-H. L. Ong. On model- checking trees generated by higher order recursion schemes. In Proceedings 21st Annual IEEE Symposium on Logic in Computer Science, Seattle, pages 81–90. Computer Society Press, 2006.

[15] N. Kobayashi, N. Tabuchi, and H. Unno. Higher-order multiparameter tree transducers and recursion schemes for program verification. In POPL, pages 495–508, 2010.

Рис. 1:

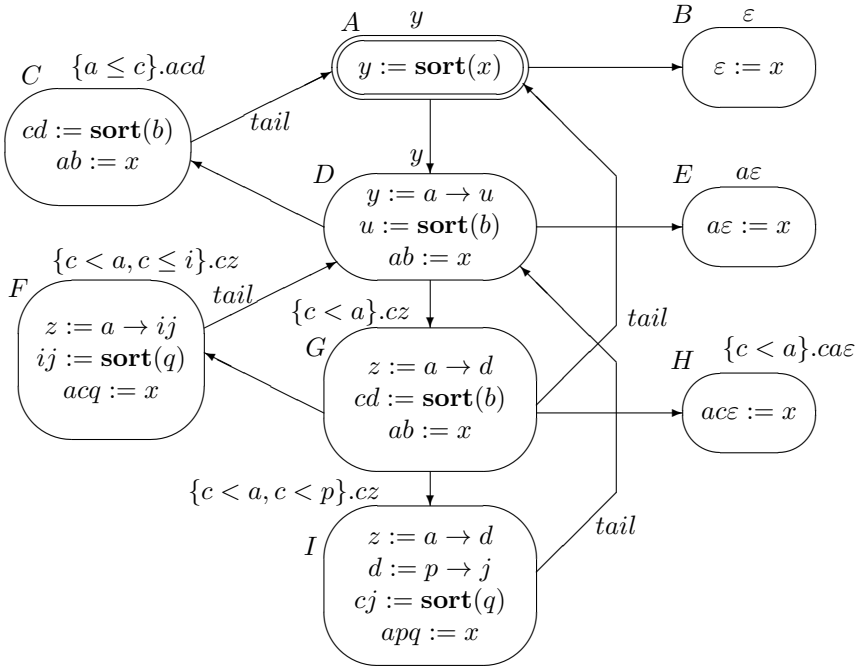


Рис. 2:

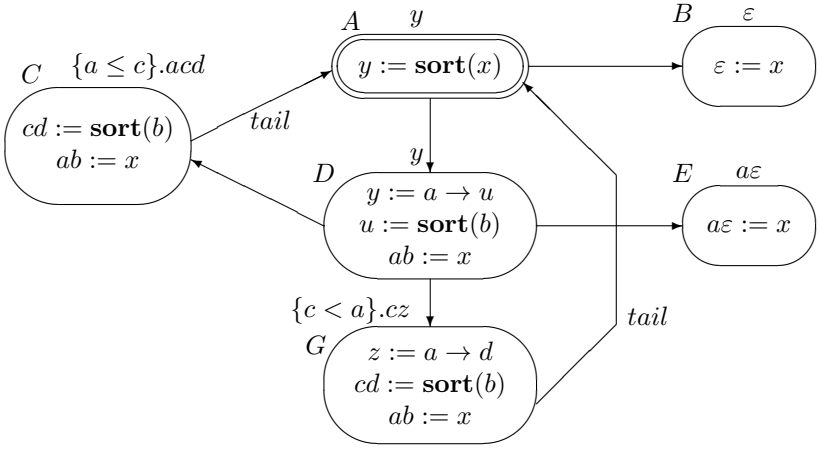


Рис. 3: