СИСТЕМОЛОГИЯ И МЕТОДОЛОГИЧЕСКИЕ ПРОБЛЕМЫ
ИНФОРМАЦИОННО-ЛОГИЧЕСКИХ СИСТЕМ
(Вычислительные системы)
1990 год Выпуск 135

УДК 519.685

К ОПЕРАЦИОННОЙ ИНТЕРПРЕТАЦИИ Σ^+-ПРОГРАММ. r-ПРОГРАММЫ И ИХ СЕМАНТИКА.

С.В. Котов

В в е д е н и е

Цель данной работы - попытаться развить намеченный в [3] подход к операционной семантике языка Σ^+-программ (см.[1,2]).

В [3] излагался язык r-процедур, представляющий собой процедурное расширение языка Σ^+-формул. Отличительной особенностью этого языка является то, что он обладает двумя взаимо - увязанными семантиками, одна из которых **(S)** распространяет теоретико-множественный смысл Σ^+-формул, которые есть подкласс r-процедур, на конструкторы "внешне императивной" природы. Другая, императивная **(F)** семантика определяет r-процедуры как объекты, меняющие состояние некоторой модели вычислитель - ной среды. Таким образом, императивную семантику можно рассматривать в качестве прообраза реализации языка Σ^+-формул и, следовательно, Σ^+-программ. Данная работа посвящена определению семантики Σ^+-программ в терминах r-формализма. Делается это следующим образом:

а) выделяется подкласс r-процедур, состоящий из выражений "наиболее императивного" вида;

б) вводится понятие r-программы, определяется семантика r-программ (используется выделенный подкласс);

в) каждой Σ^+-программе ставится в соответствие r-программа.

Действительно, весь класс r-процедур не может быть интересен при исследовании вопросов операционного характера, так как он включает в себя, например, Σ^+-формулы, которые практически не несут в себе информации, полезной для решения такого рода вопросов. Естественно выделить подкласс r-процедур, императивный смысл которых наиболее явно представлен синтаксисом. Другими словами, это должны быть процедуры, содержащие как можно меньше информации в виде r-формул и как можно больше конструкторов ";" и "*", имеющих естественные операционные интерпретации.

Такой подкласс мы назовем классом fr-процедур ("плоских r-процедур"). Аналогичные плоские конструкции были использованы в работе [4] по логическому программированию. Приведение хорновых дизъюнктов к плоской форме имело в этой работе следующую цель: позволить обращаться с предикатом равенства, как и с любым другим, т.е. с помощью варианта SLD-резолюции. В нашем случае уплощение позволяет, во-первых, избавиться от вложенных термальных конструкций, как одного из проявлений декларативной постановки задачи, и, во-вторых, элиминировать и логические связки. Тем самым осуществляется переход к r-процедурам, атомарные подпроцедуры которых есть r-формулы простейшего вида. Как правило, такой переход от логических конструкторов к императивным может быть выполнен неоднозначно, и, следовательно, существует возможность выбора варианта операционной трактовки декларативного описания.

Чтобы распространить такого рода преобразования на Σ^+-программы, необходимо ввести понятие r-программы (и соответственно r-определения и r-схемы), расширяющее Σ^+-формализм аналогично тому, как r-процедуры обогащают язык Σ^+-формул. Семантика r-программ будет определена посредством дополнительного построения вычислительных схем специального вида. Введение

такого построения дает нам повод считать семантику не чисто денотационной, а имеющей частично операционный характер.

Основным результатом работы является доказательство адекватности семантики, построенной с помощью вычислительных схем, денотационной семантике Σ^+-программ. Отсюда следует,что описанный при определении семантики r-программ процесс трансляции r-схемы в вычислительную схему может рассматриваться как первый шаг трансляции Σ^+-программ в императивные формализмы.В перспективе вторым шагом мы видим трансляцию вычислительных схем в динамические сети (недетерминированных) процессов. В этом направлении и будет развиваться работа по операционной интерпретации Σ^+-формализма.

Для определения семантики r-программ нам понадобятся дополнительные определения. Они носят довольно технический характер. Как указывалось, вводимая семантика в известной мере операционная, и мы сталкиваемся соответственно с естественными практическими проблемами, такими как перечисленные ниже.

1. Существование локальных переменных в подпрограммах и при реализации языковых конструкций высокого уровня (формулы) посредством более императивных конструкций (плоские процедуры) требует ввести понятие сужения изменения состояния системы, локализующего изменения неглобальных переменных.

2. Существует необходимость переименования переменных - сопоставление формальных и фактических параметров при "вызовах" подпрограмм (понятие переименования).

3. Предложенная в [3] трактовка понятия предикат предполагала возможность перехода от предиката к объектам, реализующим его частные случаи. Понятие допустимой подстановки определяет синтаксическое преобразование r-процедур, соответствующее переходу к частному случаю.

Перечисленным вопросам посвящены три пункта §1. Там же дано определение плоских r-процедур (fr-процедур) и сформулирована лемма о возможности представления произвольной r-процедуры посредством плоской. В приложении к работе указан пример алгоритма получения такого плоского представления. Заключает §1 необходимое, как нам кажется, краткое изложение синтаксиса и денотационной семантики Σ^+-программ.

В §2 дано определение r-программ, вычислительных схем, семантик вычислительных схем и r-программ, а также сформулирован основной результат данной работы.

§1. Определения

1.1. Понятие представления. В данном параграфе мы введем несколько необходимых в дальнейшем определений. Некоторые из них связаны с императивной семантикой $(\mathbf{F})$ r-процедур. Напомним, что при зафиксированных значениях внешних параметров семантика $\mathbf{F}$ сопоставляет r-процедуре элемент пространства $\mathbf{Func} = \mathbf{Val} \to [\mathbf{States} \to \mathbf{States}]$, где $\mathbf{Val}$ - пространство означиваний нефиксированных переменных r-процедур, $\mathbf{States}$ - пространство состояний, являющееся полной решеткой относительно упорядочения $\mathbf{I} \subseteq \mathbf{J}$ ("I - состояние с меньшей информацией, чем J"). Квадратные скобки $[\ldots \to \ldots]$ обозначают полную решетку непрерывных отображений, а круглые скобки $(\ldots \to \ldots)$ - пространство всех отображений.

Введем частичный порядок на $\mathbf{Func}$ обычным образом:

$$\mathbf{f} \subseteq \mathbf{g} \Leftrightarrow \forall \mathbf{u} \in \mathbf{Val}\ \ \forall \mathbf{I} \in \mathbf{States}\ \ \mathbf{f}\langle \mathbf{u}\rangle(\mathbf{I}) \subseteq \mathbf{g}\langle \mathbf{u}\rangle(\mathbf{I}).$$

Известно, что $\langle \mathbf{Func}, \subseteq \rangle$ есть полная решетка. Обозначим ее наименьший элемент через "o". Отображение "o" ставит в соответствие любому состоянию при любом означивании $\mathbf{u} \in \mathbf{Val}$ пустое состояние 0.

ОПРЕДЕЛЕНИЕ 1. Для $\mathbf{W} \subseteq \mathbf{Var}$ и $\mathbf{f} \in \mathbf{Func}$ определим $\mathbf{N_W(f)}$ как элемент $\mathbf{Val} \to (\mathbf{States} \to \mathbf{States})$, такой

что для любых $I \in States$ и $u \in Val$ состояние $J = N_W(f)\langle u\rangle(I)$ определяется условиями:

$$x \notin W \Rightarrow [x]_J = [x]_I \setminus W \;\&\; J[x] = I[x] \,|\, [x]_J,$$

$$x \in W \Rightarrow [x]_J = W \;\&\; J[x] = S_W(f\langle u\rangle(I)).$$

Таким образом, $N_W(f)$ отличается от f тем, что оно изменяет интерпретацию только переменных из множества W и делает все элементы W синхронизированными.

УТВЕРЖДЕНИЕ 1.

а) *Образ любого* $f \in Func$ *при отображении* N_W *принадлежит* $Func$,

б) *отображение* $N_W: Func \to Func$ *непрерывно.*

Следующее определение объясняет смысл введенной операции.

ОПРЕДЕЛЕНИЕ 2. Для $P, R \in Proc$ будем говорить, что R <u>представляет</u> P, если $Ext(P) = Ext(R)$, $Unf(P) = Unf(R)$, $Out(P) \subseteq Out(R)$, $In(P) \subseteq In(R)$ и для любого $e \in PVal$ выполняется $F[P]\langle e\rangle = N_{Out(P)}(F[r]\langle e\rangle)$.

Введем соответствующее обозначение $R \in Rep(P)$. Содержательно: $Rep(P)$ есть класс всех r-процедур, вычисляющих те же результаты, что и P, используя, может быть, дополнительные входные и выходные переменные.

<u>1.2. Переименование</u>.

ОПРЕДЕЛЕНИЕ 3. <u>Переименование</u> есть взаимно-однозначное отображение множества Var на себя. Действие переименования θ на означивание $u: Var \to M$ есть означивание $u\theta$, такое что $u\theta(\theta(z)) = u(z)$. Аналогично действие переименования на состояние $I \in States$ определяется формулой:

$$\forall z \in Var\ [\theta(z)]_{I\theta} = [z]_I \;\&\; I\theta[\theta(z)] = I[z].$$

Ясно, что отображение, обратное к переименованию θ, также есть переименование. Будем обозначать его $\tilde{\theta}$.

ОПРЕДЕЛЕНИЕ 4. Вариант r-процедуры есть такая r-процеду - ра, что она может быть получена из первой синтаксической заменой всех вхождений переменных согласно некоторому переименованию.

ОПРЕДЕЛЕНИЕ 5. Для $f \in \mathbf{Func}$ обозначим через $f\theta$ отображение типа $\mathbf{Val} \to (\mathbf{States} \to \mathbf{States})$ такое, что $f\langle u\rangle(I) = f\theta\langle u\theta\rangle(I\theta)$ для всех $u \in \mathbf{Val}$ и $I \in \mathbf{States}$. Содержательно $f\theta$ есть, то же отображение, что и f, только выполненное на других входных, выходных и нефиксированных переменных.

УТВЕРЖДЕНИЕ 2.

а) *Для произвольного переименования* θ *верно* $\forall f \in \mathbf{Func}\ f\theta \in \mathbf{Func}$;

б) *отображение* $f \to f\theta$ *непрерывно для любого переименования* θ.

1.3. Допустимые подстановки. В [3] была предложена система видов портов, частично упорядоченная согласно их гибкости (нефиксированности). Гибкость представляющего предикат объекта позволяет изменить режим его работы - перейти к более частному случаю взаимодействия объекта с "внешним миром". В самых частных случаях смысл объекта близок к обычной процедуре (более подробно и точно см. в [3]).

Сейчас мы хотим распространить технику перехода к частному случаю на r-процедуры. Сведение процедуры к ее частному случаю, а также переименование нефиксированных переменных могут выполняться посредством применения к процедуре операции допустимой подстановки.

ОПРЕДЕЛЕНИЕ 6. Пусть $R \in \mathbf{Proc}$; $x \neq y$; $x \notin \mathbf{Var}(R) \setminus \mathbf{In}(R)$; $y \notin \mathbf{Var}(R) \setminus \mathbf{Out}(R)$; $z, w \notin \mathbf{Var}(R) \setminus \mathbf{Unf}(R)$ и m есть порт одного из видов w, $w!y$, $w?x$, $?x$, $?x!y$, $!y$. Тогда запись $[z \Leftarrow m]$ есть допустимая для процедуры R подстановка m вместо z (z может совпадать с w, x или y).

ОПРЕДЕЛЕНИЕ 7. Результат применения допустимой подстановки $[z \Leftarrow m]$ к процедуре R, обозначаемый $R[z \Leftarrow m]$, определяется правилами:

а) если z встречается в R в контексте порта z, то z заменяется на m;

б) если z встречается в порту $z?u$, то z заменяется частью m, полученной удалением $?x$;

в) аналогично, если z встречается в порту $z!u$, то z заменяется частью m, полученной удалением $!y$.

Например, подстановки $[x \Leftarrow ?x]$ $[z \Leftarrow w!y]$ $[z \Leftarrow ?x!y]$ преобразуют $A(z?u,z!v)$ в $A(?u,?x!v)$, $A(?u!y,w!v)$, $A(?u!y,?x!v)$ соответственно.

УТВЕРЖДЕНИЕ 3. *Если* $[z \Leftarrow m]$ *есть допустимая подстановка для r-процедуры* R, *то результат подстановки* $R[z \Leftarrow m]$ *также есть r-процедура.*

ОПРЕДЕЛЕНИЕ 8. Если r-процедура R может быть получена из процедуры P посредством применения конечной последовательности допустимых подстановок, то говорим, что R есть пример (или частный случай) r-процедуры P.

Полезными являются следующие сокращения записи. Если $z \in Unf(R)$, то $?z.R$ и $!z.R$ обозначает $R[z \Leftarrow ?z]$ и $R[z \Leftarrow !z]$ соответственно.

1.4. Плоские r-процедуры.

ОПРЕДЕЛЕНИЕ 9. Простейшей r-формулой назовем атомарную r-формулу, в которой встречается не более одного сигнатурного символа, т.е. имеющую вид $A(m_1,\ldots,m_n)[m_{n+1},\ldots,m_k]$ или $m_1 = f(m_2,\ldots,m_n)[m_{n+1},\ldots,m_k]$, причем порты m_i таковы, что никакая выходная переменная не встречается в $m_1,\ldots,m_k$ более одного раза.

ОПРЕДЕЛЕНИЕ 10. Базовой r-процедурой назовем простейшую r-формулу, или ее отрицание, или процедуру $empty(y)$.

ОПРЕДЕЛЕНИЕ 11. r-процедуру, каждая атомарная подформула которой есть базовая r-процедура, назовем плоской r-процедурой.

Самым существенным для нас свойством плоских r-процедур является следующее

УТВЕРЖДЕНИЕ 4. *Для каждой* r*-процедуры* P *существует* fr*-процедура, ее представляющая.*

Доказательство утверждения естественно вести непосредственным построением искомой fr-процедуры. В приложении описаны правила получения плоской процедуры, представляющей P. (Строгие доказательства правильности представления опущены.) Важно заметить, что совокупность правил фактически определяет алгоритм получения плоского представления r-процедуры. Не менее важно, что способ уплощения r-процедур, как правило, неединствен.

1.5. Синтаксис и денотационная семантика языка Σ^+-программ. Кратко изложим определения синтаксиса и денотационной семантики Σ^+-формализма, немного отличные от полного описания в [2].

Σ^+-определение есть выражение вида $L(\bar{x})$ def $R(\bar{x},\bar{s})$, где $L(\bar{x})$ имеет вид $p(x_1,\ldots,x_n)$ или $x_1 = f(x_2,\ldots,x_n)$, причем x_i - различные переменные, символы p и f не входят в сигнатуру базовой модели, и R есть Σ^+-формула, такая что $\{x_1,\ldots,x_n\} \subseteq FV(R)$; $\bar{s}$ есть совокупность внешних параметров ("-" обозначает набор символов).

Σ^+-схема есть конечное множество Σ^+-определений с различными предикатными переменными в левых частях:

$$L_1(\bar{x}_1) \text{ def } R_1(\bar{x}_1,\bar{h},\bar{s}_1),$$
$$\ldots \qquad\qquad \ldots$$
$$L_k(\bar{x}_k) \text{ def } R_k(\bar{x}_k,\bar{h},\bar{s}_k).$$

Сигнатурой схемы мы называем $\bar{h} = \{L_1,\ldots,L_k\}$. Объединение компонент наборов $\bar{s}_i$ есть множество внешних параметров схемы.

Σ^+-программа - это тройка $\langle Sch, \varphi, V \rangle$, где Sch есть Σ^+-схема, φ есть Σ^+-формула, $V \subseteq FV(\varphi)$.

Множество V естественно назвать множеством выходных переменных программы, а $W \equiv FV(\varphi) \setminus V$ - множеством входных переменных. Формулу φ обычно называют запросом.

Содержательный смысл схемы состоит в том, что она определяет новую (возможно, параметрическую) модель $\mathcal{M}_2()$ сигнатуры σ_2 над базовой моделью $\mathcal{M}_1$. Другими словами, при зафиксированных означиванием e внешних параметров схемы модель $\mathcal{M}_2(e)$ интерпретирует сигнатуру σ так же, как $\mathcal{M}_1$, а сигнатура схемы интерпретируется в новой модели следующим образом.

Введем обозначение $M_i \equiv M(\bar{x}_i)$. Для каждой формулы $R_i(\bar{x}_i, \bar{h}, \bar{s}_i)$ отображение $\Gamma_i\langle e \rangle: M_1 \times \ldots \times M_k \to M_i$ при заданном означивании e внешних параметров схемы определяется равенством

$$\Gamma_i[A_1, \ldots, A_k]\langle \bar{e} \rangle = \{\bar{a} \in M_i \mid \mathcal{M}_1 \models R_i(\bar{a}, \bar{A}, \bar{e})\}$$

(A_j означивают символы сигнатуры схемы в формуле R_i).

Набор из k отображений Γ_i образует оператор Γ над $M_1 \times \ldots \times M_k$. Для "хорошо организованной" базовой модели Γ непрерывен над решеткой $\langle M_1 \times \ldots \times M_k, \subseteq \rangle$ (подробней см. [1, 2]). Следовательно, существует наименьшая неподвижная точка оператора $LEP(\Gamma)$. Определим семантику сигнатуры схемы в модели $\mathcal{M}_2(e)$ компонентами $LEP(\Gamma\langle e \rangle)$.

Пусть определена семантика схемы, т.е. модель $\mathcal{M}_2$ сигнатуры σ_2. Семантика программы $Prog \equiv \langle Sch, \varphi, V \rangle$ есть $S[Prog]$-параметрическое отображение из $M(W)$ в множество подмножеств $M(V)$: $PVal \to (M(W) \to P(M(V)))$.

За определение возьмем равенство $(b \in M(W))$

$$S[Prog]\langle e \rangle(b) = \{c \in M(V) \mid \mathcal{M}_2(e) \models \varphi[e, b, c]\}.$$

В частности, при $V = \emptyset$:

$$S[Prog]\langle e\rangle(b) = \begin{cases} \text{true}, & \text{если } \mathcal{M}_2(e) \models \varphi[e,b], \\ \text{false} - & \text{в противном случае.} \end{cases}$$

Для упрощения технических выкладок существен тот факт, что любую Σ^+-программу можно заменить на эквивалентную ей с простейшей Σ^+-формулой в качестве запроса. Для этого схему Sch надо дополнить определением $"Q(x_1,\ldots,x_n)\ \text{def}\ \varphi"$, где Q - новый сигнатурный символ, а $\{x_1,\ldots,x_n\}$ есть $FV(\varphi)$. Новая программа $NProg \equiv \langle Sch \cup \{"Q(x_1,\ldots,x_n)\ \text{def}\ \varphi"\}, Q(x_1,\ldots,x_n), V\rangle$ эквивалентна исходной. Действительно,

$$S[Prog]\langle e\rangle(b) = \{c \in M(V) \mid \mathcal{M}_2(e) \models \varphi[e,b,c]\} =$$

$$= \{c \in M(V) \mid \mathcal{M}_3(e) \models Q(x_1,\ldots,x_n)[b,c]\} =$$

$$= S[NProg]\langle e\rangle(b).$$

§2. r-программы и их семантика

2.1. Синтаксис r-программ. Аналогично тому, как определяются понятия Σ^+-формализма, введем понятия r-определения, r-схемы, r-программы и их семантику.

ОПРЕДЕЛЕНИЕ 12. Выражение вида "A def R" есть r-определение, если A - простейшая r-формула и R - r-процедура такие, что $Ext(A) = Ext(R)$, $Unf(A) = Unf(R)$, $Out(A) \subseteq Out(R)$, $In(A) \subseteq In(R)$.

Смысл r-определения отличен или, можно сказать, шире, чем смысл его Σ^+-аналога. В последнем разделитель def означал "по определению тождествен", подразумевая семантическую эквивалентность правой и левой частей. Части r-определения в общем случае не являются семантически эквивалентными, def фактически декларирует, что правая часть есть представление левой части определения. Действительно, если R содержит больше

переменных, чем **A** , то трудно надеяться на эквивалентность r-процедур в смысле семантики **F** .

ОПРЕДЕЛЕНИЕ 13. Конечное множество r-определений составляет r-схему, если все сигнатурные символы в левых частях определений различны. Совокупность этих символов назовем сигнатурой схемы.

Заметим, что любые Σ^+-определения и Σ^+-схемы есть r-определения и r-схемы соответственно.

2.2. Вычислительные схемы и их семантика.

ОПРЕДЕЛЕНИЕ 14. Вычислительная схема есть конечное множество r-определений, таких что

а) их правые части есть fr-процедуры;

б) их левые части есть простейшие r-формулы без нефиксированных переменных; совокупность их сигнатурных символов есть сигнатура вычислительной схемы;

в) атомарные подпроцедуры правых частей с символом сигнатуры вычислительной схемы назовем вызовами; среди вызовов должны встречаться только такие, варианты которых встречаются среди левых частей, определений; соответствующее вызову в этом смысле определение назовем вызываемым; кроме того, все вызовы должны входить в определения позитивно, т.е. без символа отрицания.

Заметим, что правые части определений из вычислительной схемы не имеют нефиксированных переменных.

Определим семантику вычислительной схемы **CSch**. Для этого проведем следующие преобразования схемы **CSch** в схему **CSchN** .

1. Всем r-определениям поставим в соответствие различные процедурные переменные, не встречающиеся в схеме **CSch** .

2. Заменим каждый вызов в схеме на предикатную переменную, соответствующую определению, левая часть которой есть вариант

вызова. При этом для каждой такой переменной h будем отмечать, какое переименование превращает эту левую часть в замещенный вызов $\theta(h)$ и каков номер определения с подходящей левой частью $\pi(h)$. Необходимо наложить следующее требование на $\theta(h)$: переименование должно быть таким, чтобы ни одна переменная из вызывающего определения не совпадала с переименованием какой-либо переменной из правой части вызываемого определения, что далее и будет предполагаться.

Обозначим правые части полученной схемы через R_i, $1 \le i \le k$; пусть вызовы в этой схеме есть h_j, $1 \le j \le m$. Поставим в соответствие схеме **CSch** оператор типа **PVal** → → (**Func**^{k} → **Func**^{k}), определение которого мы дадим покомпонентно. Для заданного набора $\langle f_1,\dots,f_k\rangle \in$ **Func**^{k} определим означивание переменных $h_1,\dots,h_m$ следующим образом:

$$\tilde{h}(\langle f_1,\dots,f_k\rangle) \equiv \langle h_1 \leftarrow f_{\pi(h_1)}\theta(h_1),\dots,h_m \leftarrow f_{\pi(h_m)}\theta(h_m)\rangle.$$

ОПРЕДЕЛЕНИЕ 15. Определим i-ю компоненту оператора Γ^C через равенство:

$$\Gamma^C_i\langle\theta\rangle(f_1,\dots,f_k) = N_{Out(A_i)}(f[R_i]\langle\theta,h(\langle f_1,\dots,f_k\rangle)\rangle).$$

Для доказательства корректности определения необходимо показать, что правая часть принадлежит **Func**. Это действительно так в силу того, что семантика **F** всегда определяет элемент из **Func** (см. [3]) и утверждений 1 и 2.

Имеет место

УТВЕРЖДЕНИЕ 5. *Оператор* Γ^C *непрерывен.*

Таким образом, оператор Γ^C имеет наименьшую неподвижную точку $LFP(\Gamma^C)$, которая и определяет семантику вычислительной схемы.

2.3. Семантика r-программ. Опишем семантику r-программы $Prog \equiv \langle RSch, Q \rangle$ через семантику вычислительных схем. Заметим сразу, что, как и для Σ^+-программ, без ограничения общности мы можем рассматривать только случай, когда r-процедура запроса Q простейшая.

Поставим в соответствие r-схеме $RSch$ вычислительную схему $CSch$. Построение последней проведем в несколько этапов. Исходным, конечным и промежуточными пунктами построения являются конечные множества r-определений. Пусть r-определения схемы $RSch$ имеют вид:

$$L_i \ def \ R_i . \tag{1}$$

Рассмотрим все различные примеры этих определений, в которых нет нефиксированных переменных, получая совокупность определений вида:

$$B_j \ def \ E_j . \tag{2}$$

Далее, заменим все правые части E_j на какие-то их плоские представления F_j:

$$B_j \ def \ F_j . \tag{3}$$

В правых частях этих определений, возможно, есть атомарные подформулы вида $B_j \in [m_1, \ldots, m_m]$. Для каждого такого вхождения добавим к нашей схеме определения:

$$C_j \ def \ G_j , \tag{4}$$

где G_j есть $(F_j^1; TRUE[?U^1 U, m_1 \tilde{\in}, \ldots, m_m \tilde{\in}])$, а C_j - $B_j[m_1 \tilde{\in}, \ldots, m_m \tilde{\in}]$. Здесь F_j^1 - r-процедура, полученная переименованием всех выходных переменных из C_j новыми переменными, а $?U^1 !U$ - набор портов вида $?y^1 !y$, где y - выходная C_j, а y^1 - ее заменившая в F_j^1. Таким образом,

мы получили вычислительную схему, семантика которой должна определить семантику r-программ. Напомним, что семантика вычислительной схемы получается при помощи еще одного дополнительного построения, вводящего в правые части новые процедурные переменные:

$$C_j \text{ def } H_j . \qquad (5)$$

Перейдем от построений непосредственно к определению. Ясно, что запрос Q есть вариант левой части какого-то определения из $CSch$. Обозначим необходимое переименование θ, а номер этого определения j.

ОПРЕДЕЛЕНИЕ 16. Определим семантикой r-программы $RProg$ при означивании внешних переменных $\theta \in PVal$ переименование компоненты неподвижной точки $LFP_j(\Gamma^C)\theta$. (Семантикой можно считать как элемент пространства $Func$, так и элемент $[States \to States]$, так как множество нефиксированных параметров запроса пусто.)

УТВЕРЖДЕНИЕ 6. *Определение семантики r-программы корректно, т.е.* а) *для всякой r-программы существует хотя бы одна вычислительная схема, подходящая в указанном смысле, и* б) *семантика r-программы не зависит от выбора вычислительной схемы.*

Существование подходящей вычислительной схемы вытекает из конструктивности описанного построения и утверждения 4. Независимость семантики от конкретной выбранной вычислительной схемы вытекает из того, что каковы бы ни были правые части определений, они представляют пример левой части, а оператор M в определении оператора Γ^C элиминирует различия двух представлений.

2.4. Связь Σ^+- и r-программ. Уже отмечалось, что любая Σ^+-схема есть также r-схема. Это позволяет легко преобразовать Σ^+-программу в такую r-программу, что декларативная се-

мантика Σ^+-программы может быть выражена через семантику последней.

ТЕОРЕМА. *Пусть* $\mathbf{Prog} \equiv \langle \mathbf{Sch}, \varphi, V \rangle$ *есть* Σ^+-*программа и* $g \in \mathbf{Func}$ *есть семантика* r-*программы* $\mathbf{RProg} \equiv \langle \mathbf{Sch}, ?W!V.\varphi \rangle$, $W \equiv FV(\varphi) \setminus V$, *при означивании внешних переменных* $e \in \mathbf{PVal}$, *тогда для любого* $b \in M(W)$ *верно* $S[Prog]\langle e \rangle(b) = S_V(g\langle u \rangle(O \setminus b))$, *где* O - *пустое состояние,* $O \setminus b$ - *состояние, которое интерпретирует переменные из* W *множеством* $\{b\}$, *а остальные переменные - как пустое состояние,* u - *произвольное означивание из* $\mathbf{Val}$.

ДОКАЗАТЕЛЬСТВО. Предполагая, что запросы r- и Σ^+-программ есть простейшие r-формулы, построим (согласно описанным выше шагам (1)-(5)) по r-схеме $\mathbf{RSch}$ вычислительные схемы $\mathbf{CSch}$ и $\mathbf{CSchN}$. По последней определяются оператор Γ^C и семантика r-программы $\mathbf{RProg}$. Для доказательства равенства в формулировке теоремы покажем корреляцию неподвижных точек операторов Γ и Γ^C. Введем обозначения:

$$A^0 = \emptyset,\ A^l = \Gamma(A^{l-1}),\ A^l \subseteq M_1 \times \ldots \times M_k,\ l > 0;$$

$$f^0 = \langle o, \ldots, o \rangle,\ f^l = \Gamma^C(f^{l-1}),\ f^l \in \mathbf{Func}^m,\ l > 0.$$

В силу непрерывности операторов Γ и Γ^C их наименьшие неподвижные точки можно задать как $\cup \{A^l \mid l \in \omega\}$ и $\cup \{f^l \mid l \in \omega\}$ соответственно. Для описанных последовательностей имеет место следующая

ЛЕММА. *Пусть* i *и* j *таковы, что левая часть* fr-*определения* C_j **def** H_j *схемы* $\mathbf{CSchN}$ *есть пример* $L_i[z_1, \ldots, z_n]$, *где* L_i *есть левая часть* Σ^+-*определения* L_i **def** R_i *схемы* $\mathbf{Sch}$. *Тогда для всякого* l *верно равенство:*

$$A_i^l \times M(\{z_1,\dots,z_n\}) = \{a \in M_i \times M(\{z_1,\dots,z_n\}) \mid$$

$$\forall I \in States\ a|In(C_j) \in S_{In\langle C_j\rangle}(I) \to$$

$$\to a|Out(C_j) \in S_{Out\langle C_j\rangle}(f_j^l\langle u\rangle(I))\}.$$

ДОКАЗАТЕЛЬСТВО проведем индукцией по l. Для $l = 0$ требуемое равенство очевидно имеет место. Для $l > 0$ установим равенство левой и правой частей, основываясь на том, что определение $C_j\ def\ H_j$ было получено из $I_i def\ R_i$ посредством преобразований (1)-(5).

Пусть $a' \in A_i^l\ M(\{z_1,\dots,z_n\})$ и $a \in A_i^l = \Gamma_i(A^{l-1})$ есть его сужение на M_i. По определению оператора Γ условие $a \in \Gamma_i(A^{l-1})$ равносильно $\mathcal{M}_2 \models R_i[e, \bar{h} \leftarrow A^{l-1}, a]$. Согласно последнему утверждению из [3] это эквивалентно тому, что верна импликация (I - произвольное состояние, B_j и E_j см. (2)):

$$a'|In(B_j) \in S_{In\langle E_j\rangle}(I) \to$$

$$\to a'|Out(B_j) \in S_{Out\langle E_j\rangle}(F[E_j]\langle e, \bar{h} \leftarrow A^{l-1}\rangle(I)).$$

Так как F_j из (3) есть плоское представление E_j, то по определению представления можно заключить, что утверждение

$$a'|In(B_j) \in S_{In\langle E_j\rangle}(I) \to$$

$$\to a'|Out(B_j) \in S_{Out\langle E_j\rangle}(F[F_j]\langle e, \bar{h} \leftarrow A^{l-1}\rangle(I))$$

эквивалентно предыдущему. Далее, можно показать, что

$$a'|In(C_j) \in S_{In\langle E_j\rangle}(I) \to$$

$$\to a'|Out(C_j) \in S_{Out\langle E_j\rangle}(F[G_j]\langle e, \bar{h} \leftarrow A^{l-1}\rangle(I)).$$

Легко заметить, что согласно предположению индукции формула H_j, полученная из $G_j \equiv (F_j^1; \mathrm{TRUE}[?U^1!U, m_1\tilde{\Theta}, \ldots, m_m\tilde{\Theta}])$ при определении семантики вычислительной схемы, имеет ту же семантику F, что и последняя, если процедурные параметры H_j означить элементами пространства $\mathbf{Func}$ из набора f^{1-1}. Для завершения доказательства леммы достаточно увидеть, что все утверждения из предложенной цепочки эквивалентны.

Из леммы вытекает, что верно равенство

$$\mathrm{LFP}(\Gamma)_i = \{a \mid \forall I \in \mathrm{States}\ a|\mathrm{In}(C_j) \in S_{\mathrm{In}\langle C_j\rangle}(I) \rightarrow$$

$$\rightarrow a|\mathrm{Out}(C_j) \in S_{\mathrm{Out}\langle C_j\rangle}(f_j\langle u\rangle(I))\},$$

где i - номер определения "$Q(x_1, \ldots, x_n)$ def φ", а j - номер r-определения из вычислительной схемы с левой частью, являющейся переименованием Q. По определению семантики r-программ f_i и есть g из условия теоремы. Из определения семантики Σ^+-программ $a \in \mathrm{LFP}_i(\Gamma)$ тогда и только тогда, если $a|V \in S[\mathrm{Prog}]\langle e\rangle(a|W)$. С другой стороны, $a \in \mathrm{LFP}_i(\Gamma)$ равносильно тому, что $a|W \in S_V(g\langle u\rangle(O\backslash(a|V)))$. Последнее замечание и доказывает теорему.

ПРИМЕР. Приведем пример простой Σ^+-схемы и преобразования ее по правилам (1)-(5) п. 2.3. Пусть схема состоит из одного определения $\mathrm{even}(X)$ def $(\neg X = 1)\ \&\ ((X = 2) \vee \mathrm{even}(\mathrm{minus}(X,2)))$, где X имеет тип натуральных чисел и minus есть базовая функция усеченной разности. Ясно, что это рекурсивное определение задает предикат, истинный на четных числах.

Допустимая подстановка $[X \Leftarrow ?X]$ в данном случае сводится к синтаксической замене: $\mathrm{even}(?X)$ def $(\neg ?X = 1)\ \&\ ((?X = 2) \vee (\mathrm{even}(\mathrm{minus}(?X,2))))$.

Аналогично применяются подстановки $[X \Leftarrow !Y]$ и $[X \Leftarrow ?X!Y]$. Таким образом мы получаем три примера исходного определения без нефиксированных переменных. Заметим, что для двухместного предиката $a(X,Y)$ число таких примеров не есть 3^2, так как возможны примеры типа $a(?X!Y,?X!Z)$ и т.п.

Следующим шагом преобразуем правые части определений. Приведение к плоской форме, использованное в этом примере, более сложно, чем описанное в приложении. Оно дает более красивый результат. Итак, имеем три определения:

```
even(?X) def ((¬?X!X1 = 1); (?X1 = 2); (¬?X1!X2 = 2);
                    (!Z = minus(?X2,2)); even(?Z)),
even(!Y) def ((¬!Y1 = 1); (?Y1!Y = 2); (¬?Y1!Y2 = 2);
          (!Z = minus(?Y2!Y3,2)); even(?Z)[?Y3!Y]),
even(?X!Y) def ((¬?X!Y1 = 1); (?Y1!Y = 2); (¬?Y1!Y2 =
    = 2); (!Z = minus(?Y2!Y3,2)); even(?Z)[?Y3!Y]).
```

Так как в правых частях встречается простейшая процедура вида $even(?X)[?Z!Y]$, добавим четвертое определение:

```
even(?X)[?Z!Y] def ((¬?X!X1=1);(?X1=2);(¬?X1!X2=2);
(!Z=minus(?X2,2));even(?Z);TRUE[?TruthValue,?Z!Y]).
```

Наконец, введя процедурные переменные f_1, f_2, f_3, f_4, получаем схему:

```
f1: even(?X) def ((¬?X!X1=1); (?X1=2);
          (¬?X1X2=2); (!Z=minus(?X2,2)); f1),

f2: even(!Y) def ((¬!Y1=1); (?Y1!Y=2);
        (¬?Y1!Y2=2); (!Z=minus(?Y2!Y3,2)); f4),

f3: even(?X!Y) def ((¬?X!Y1=1); (?Y1!Y=2);
        (¬?Y1!Y2=2); (!Z=minus(?Y2!Y3,2)); f4),

f4: even(?X)[?Z!Y] def ((¬?X!X1=1); (?X1=2);
            (¬?X1!X2=2); (!Z=minus(?X2,2)); f1;
                        TRUE[?TruthValue,?Z!Y]).
```

Заключение

В заключении хотелось бы мотивировать определение семантики r-программ посредством построения вычислительных схем. На первый взгляд вычислительная схема отличается от r-схемы только тем, что для нее легче строится оператор, неподвижная точка которого и фиксирует семантику. Однако использование вычислительных схем потребовало сложного доказательства основного утверждения работы. Таким образом, предложенная конструкция вводилась не с целью удобства изложения или с целью элегантности определения семантики r-программ. Основной задачей было ввести конструкцию, обладающую определенным операционным смыслом, и получить тем самым семантику, в известной степени операционную. Для того чтобы показать это, т.е. перейти от вычислительной схемы к реализации, необходимо сделать, как минимум, еще один промежуточный шаг, "заземляющий" нашу цепочку преобразований. Таким шагом мы видим формализацию вычислений в стиле потоков данных, а именно трансляцию вычислительных схем в вариант динамических сетей процессов, впервые предложенных в [6]. Эта модель вычислений хорошо изучена как с теоретической, так и с практической стороны, предложенные ей идеи вычислений имеют ряд реализаций программного и аппаратного характера. Работа по операционной интерпретации Σ^+-программ будет продолжена именно в направлении трансляции вычислительных схем в сети (недетерминированных) процессов, а также изучения на этой основе различных аспектов "исполнения" Σ^+-программ, таких как параллелизм и недетерминизм вычислений.

Автор благодарит Д.И.Свириденко и М.Ю.Трифонова за полезные замечания по содержанию работы.

Литература

1. ГОНЧАРОВ С.С., СВИРИДЕНКО Д.И. Σ-программирование //Логико-математические проблемы МОЗ. - Новосибирск, 1985.-Вып.107: Вычислительные системы. - С.3-29.

2. Их же. Σ-программы и их семантики //Логические методы в программировании. - Новосибирск, 1987. -Вып. 120: Вычисли - тельные системы. - С. 24-51.

3. КОТОВ С.В. К операционной интерпретации Σ^{+}-программ. Язык r-процедур //Настоящий сборник. -С. 131-159.

4. COX P.T., PIETRZYKOWSKI T. Incorporating Equality into Logic Programming via Surface Deduction //Annals of Pure and Applied Logic. - 1986. - Vol. 31. -P. 177-189.

5. KAHN G. The Semantics of a Simple Language for Paral - lel Programming //Proceedings of IEIP'74, North-Holland, Amsterdam, 1974. -P. 471-475.

6. KAHN G., MacQUEEN D.B. Coroutines and Networks of Parallel Processes //Proceeding of IFIP'77, North-Holland, Amster - dam, 1977. -P. 993-998.

Поступила в ред.-изд.отд.

15 мая 1990 года

Алгоритм уплощения (пример)

Для изложения правил определения плоского представления нам необходимы дополнительные обозначения. Для данной $y \in Var$ через $y^1, y^2, \ldots \in Var$ обозначим различные новые переменные, ранее не встречавшиеся в формулах или рассуждениях. Если m - порт, то через $m^1, m^2, \ldots$ обозначим новые порты, полученные из m заменой всех вхождений некоторой указанной переменной y (или набора переменных) на $y^1, y^2, \ldots$ соответственно.

Зададим алгоритм в виде набора правил переписывания, применимых к подпроцедурам данной процедуры или подформул данной r-формулы. Доказательство корректности алгоритма и соответственно доказательство утверждения 4 должно состоять из а) доказательства корректности каждого правила относительно определения представления и б) доказательства успешной завершаемости алгоритма (для любой данной r-процедуры число возможных применений правил конечно и результатом является fr-процедура). Мы опускаем оба пп."а" и "б", так как доказательство заключается в рутинной непосредственной проверке. Итак, приводим пример набора правил уплощения, предлагающих для данной процедуры P процедуру R, ее представляющую.

R1. Если в P есть подпроцедуры, являющиеся небазовыми r-формулами, то эти вхождения необходимо заменить на их негативные нормальные формы (определение таких форм для r-формул полностью аналогично определению для логических формул, см. [2]).

R2. Предположим, что некоторые выходные переменные имеют более одного вхождения в r-формулу P. Пусть это будут $y_1, \ldots, y_l$. Выходные переменные, имеющие ровно одно вхождение, обозначим $z_1, \ldots, z_m$. Обозначим через $y_1^1, \ldots, y_1^{k_1}, \ldots, y_l^1, \ldots, y_l^{k_l}, z_1^1, \ldots, z_m^1$ новые переменные $(k_1, \ldots, k_l$

есть количества вхождений в P $y_1,\ldots,y_l$ соответственно). Определим G как r-процедуру, полученную из P синтаксической заменой всех вхождений всех выходных переменных на новые переменные так, чтобы каждая выходная переменная встречалась ровно один раз. Введем

$$T \equiv \mathop{\&}_{i=1,l} (?y_i^1 ! y_i = \ldots = ?y_i^{k_i})[\, z_1^1 ! z_1, \ldots, ?z_m^1 ! z_m\,].$$

Очевидно, что $(empty(y_1^1,\ldots,z_m^1);G;T) \in Rep(P)$ и всякая выходная переменная в подпроцедурах, являющихся r-формулами, встречается один раз.

R3. Атомарная r-формула

$$P \equiv A(t_1,\ldots,t_n)[m_1,\ldots,m_k]$$

$$(\text{или}\quad t_1 = f(t_2,\ldots,t_n)[m_1,\ldots,m_k]),$$

не являющаяся базовой, может быть упрощена следующим образом. Вводятся новые переменные $w_1,\ldots,w_n$ - по числу термов в формуле и $y_1^1,\ldots,y_l^1$ - по числу выходных переменных формулы. Обозначим через n_i, $1 \le i \le l$, порты вида $?y_i^1 ! y_i$, а через s_j, $1 \le j \le n$, - термы, полученные заменой в t_j всех выходных переменных y_i на y_i^1. Тогда имеем

$$(empty(y_1^1,\ldots,y_l^1);\ \mathop{\&}_{j=1,n} (!w_j = s_j)\ \&$$

$$\&\ A(?w_1,\ldots,?w_n)[n_1,\ldots,n_i,m_1,\ldots,m_k]) \in Rep(P)$$

и все подформулы конъюнкции имеют меньшую (по глубине) вложенность термов, чем исходная формула.

R4. Аналогичная операция проделывается для отрицания атомарной r-формулы. Результат почти идентичен:

$$(empty(y_1^1,\ldots,y_l^1);\ \mathop{\&}_{j=1,n} (!w_j = s_j)\ \&\ \mathop{\&}_{j=1,n} (!w_j = s_j)\ \&$$

$$\&\ \neg A(?w_1,\ldots,?w_n)[n_1,\ldots,n_i,m_1,\ldots,m_k]) \in Rep(P).$$

R5. Пусть $P \equiv F(m_1,\ldots,m_k) \vee G(m_{k+1},\ldots,m_l) \times$ $\times[m_{l+1},\ldots,m_n]$. Обозначим через U и V непересекающиеся множества выходных переменных P, встречающихся исключительно в портах $m_1,\ldots,m_k$ и $m_{k+1},\ldots,m_l$ соответственно. Тогда

$$(F(m_1,\ldots,m_k)[!V,m_{l+1},\ldots,m_n];$$

$$G(m^1_{k+1},\ldots,m^1_l)[!U,m_{l+1},\ldots,m_n]) \in Rep(P).$$

Здесь и ниже $!W$ обозначает $!y_1,\ldots,!y_m$, $y_i \in W$, а $?W^1!W$ обозначает $?y^1_1!y_1,\ldots,?y^1_m!y_m$, $y_i \in W$.

R6. Пусть P будет $F(m_1,\ldots,m_k)\,\&\,G(m_{k+1},\ldots,m_l)\times$ $\times[m_{l+1},\ldots,m_n]$ и подпроцедуры F и G не имеют общих выходных переменных. Тогда

$$(F(m^1_1,\ldots,m^1_k);$$

$$G(m_{k+1},\ldots,m_l)[?W^1!W,m_{l+1},\ldots,m_n]) \in Rep(P),$$

если W есть $Out(F(m_1,\ldots,m_k))$.

R7. Пусть P имеет тот же вид, что и в предыдущем пункте, но F и G имеют общие выходные переменные. Зададим U, V, W так же, как и в п. R5, тогда

$$(F(m^1_1,\ldots,m^1_k);G(m^2_{k+1},\ldots,m^2_l);\ \&_{y\in W}(!y?y^1 = ?y^2)\times$$

$$\times[?U^1!U,?V^2!V,m_{l+1},\ldots,m_n]) \in Rep(P),$$

а появившаяся конъюнкция не содержит подпроцедур, имеющих общие выходные переменные.

R8. Пусть P будет $\exists z\ F(m_1,\ldots,m_l,z)[m_{l+1},\ldots,m_n]$ и $\{y_1,\ldots,y_m\} = Out(F(m_1,\ldots,m_l,z)[m_{l+1},\ldots,m_n])$, очевидно, что

$$(F(m^1_1,\ldots,m^1_l,!z);\ TRUE[?y^1_1!y_1,\ldots$$

$$\ldots, ?y_m^1 ! y_m, m_{l+1}, \ldots, m_n] \;)$$

представляет P .

R9. Если P есть $\exists z \in l \;\; F(z)[m_1, \ldots, m_n]$ (l- это r-терм), мы воспользуемся логической эквивалентностью формул $\exists\, z \in l \;\; \varphi(z)$ и $\exists z \; (z \in l \;\&\; \varphi(z))$ и сведем случай с ограниченной квантификацией к случаю с неограниченной в предположении, что двухместный предикат принадлежности объекта списку $(\in)$ реализован в базовой модели.

R10. Пусть $P \equiv \forall z \in l \; (m_1, \ldots, m_k) \;\; \varphi(m_1, \ldots, m_k, z) \times$ $\times [m_{k+1}, \ldots m_n]$ (l есть r-терм). Обозначим y_i выходную переменную из m_i, все выходные переменные будут $y_{p_1}, \ldots$ $\ldots, y_{p_l}$, $1 \le p_j \le n$. Пусть m_i^0 будет порт, полученный из m_i заменой y_i на y_i^1, а если порт не имеет выходной переменной, то добавлением $'y_i^1$. Таким образом, в любом порту m_i^0 встречается выходная переменная y_i^1. Наше представление таково:

$$(empty(h,t,y_1^1, \ldots, y_n^1, y_1^2, \ldots, y_n^2);$$

$$(cons(!h,!t) = l(m_1^0, \ldots, m_k^0) \&$$

$$\& \varphi(m_1^0, \ldots, m_k^0, !h))[m_{k+1}^0, \ldots, m_n^0];$$

$$*z \leftarrow t.(\varphi(?y_1^1 ! y_1^2, \ldots, ?y_k^1 ! y_k^2, z)[?y_{k+1}^1 ! y_{k+1}^2, \ldots$$

$$\ldots, ?y_n^1 ! y_n^2];$$

$$empty(y_1^1, \ldots, y_n^1);$$

$$TRUE[?y_1^2 ! y_1^1, \ldots, ?y_n^2 ! y_n^1]);$$

$$TRUE[?y_{p_1}^1 ! y_{p_1}, \ldots, ?y_{p_l}^1 ! y_{p_l}]).$$

Здесь cons обозначает конструктор списков, первый аргумент соответствует голове, второй аргумент - хвосту списка.

Важно отметить, что в частном случае, когда r-процедура имеет пустое множество выходных переменных, таковой следует считать переменную TruthValue и производить с ней анало - гичные манипуляции. Отличие состоит в том, что постоянно встречающийся в правилах фрагмент $[?y_1^1 y_1, \ldots, ?y_k^1 y_k, \ldots]$ в конце выражения, представляющего P , следует в данном случае заменить на [?TruthValue,...].

На этом описание алгоритма заканчивается. Утверждается, что применение конечного числа правил вида R1-R10 позволит получить плоское представление для любой r-процедуры.