Кибернетический сборник

НОВАЯ СЕРИЯ

ВЫПУСК

13

Сборник переводов

Под редакцией

О. Б. ЛУПАНОВА

ИЗДАТЕЛЬСТВО «МИР»

Москва 1976

УДК 519.95

Научный совет по кибернетике
Академии наук СССР

Продолжение новой серии кибернетических сборников, публикация которых начата издательством «Мир» в 1965 г. Сборник состоит из трех разделов: математические вопросы, математическая лингвистика и вычислительные машины и мышление. В данном выпуске большой интерес представляют статья Р. М. Карпа и Р. Е. Миллера «Параллельные схемы программ», статья В. А. Вудса «Сетевые грамматики для анализа естественных языков» и статьи по искусственному интеллекту.

Сборник рассчитан на научных работников, инженеров, аспирантов и студентов различных специальностей, занимающихся теоретической кибернетикой и ее приложением.

Редакция литературы по математическим наукам

К $\frac{20205-022}{041(01)-76}$ 22—76

Параллельные схемы программ[1])

Р. М. Карп, Р. Е. Миллер

ВВЕДЕНИЕ

Актуальной проблемой теории вычислений является проблема эффективного распознавания свойств вычислительных процессов и функций, определяемых программами. В настоящей статье вводится формальная модель программы и в ее пределах исследуются некоторые аспекты данной проблемы. На уровне абстракции этой модели игнорируются некоторые детали операторов программ и подчеркиваются свойства, сохраняющиеся независимо от способа определения этих деталей. Такая постановка сужает круг вопросов, которые можно выразить в пределах модели, однако допускает построение процедур разрешения для некоторых интересных свойств, аналоги которых в более детальных моделях были бы неразрешимы.

Схема Янова [2] (см. также [10]) является, по-видимому, наиболее простым прототипом рассматриваемой в дальнейшем модели. В подходе Янова выделены те аспекты программы, которые не зависят ни от функций, выполняемых операторами, ни от логических условий, проверяемых при выходе из логических вершин. Другими словами, здесь подчеркиваются те свойства, которые могут быть извлечены из блок-схемы программы без фиксации функций, сопоставленных ее вершинам.

Настоящая статья написана в традициях работы Янова. Однако, как и в работе [7], в ней исследуются и другие существенные элементы программы. Для каждого оператора определяются имена его операндов и результатов. Эти имена представлены в нашей модели ячейками памяти, тогда как в схеме Янова память не структурируется на ячейки [2]).

Отличие настоящей работы от [2] и [7] состоит в том, что она ориентирована на параллельные вычисления. В этом отношении она близка к более ранней работе о схемах вычислений [3]. В данной работе, как и в предварительном ее варианте [4], некоторые результаты приведены без доказательства. Ориентация на параллельные вычисления осуществляется в двух на-

---

[1]) Karp R. M., Miller R. E., Parallel Program Schemata, *Journal of Computer and System Sciences*, 3 (1969), 147—195.

[2]) Однако можно заметить, что некоторая ограниченная информация о структуре памяти может быть представлена в схеме Янова посредством «распределения сдвигов».

правлениях. Во-первых, допускается одновременное выполнение нескольких операторов. Это дает возможность представить вычислительные процессы в форме, приспособленной для их выполнения на машине с автономными процессорами, использующими общую память. Во-вторых, наряду с такими традиционными свойствами, как эквивалентность, изучаются новые свойства, характерные для параллельных вычислений. Например, подробно исследуется явление «состязания» шагов вычисления, которое может привести к неоднозначности программы.

Эти исследования возникли в связи с необходимостью выяснить роль параллельных вычислений при конструировании и эффективном использовании вычислительных систем. Вклад данной работы в решение этой проблемы имеет несколько аспектов. В частности, разработаны методы определения степени параллелизма программы и проверки ее однозначности. Однозначность программы означает независимость результатов вычислений от относительных скоростей согласованно выполняемых операторов. Эту формулировку можно применить к общей форме представления параллельных программ, к трансляторам и операционным системам для параллельных вычислительных машин, к алгоритмам для повышения степени их параллелизма.

В первом разделе данной работы вводится параллельная схема программы и определяются некоторые ее свойства, исследуемые в дальнейшем. Во втором разделе даются необходимые и достаточные условия однозначности схем некоторого определенного подкласса. В третьем разделе приводятся некоторые технические результаты, относящиеся к префиксам вычислительных процессов. В четвертом разделе исследуется класс так называемых счетчиковых схем и даются алгоритмы распознавания таких свойств этих схем как однозначность, ограниченность, свобода, замкнутость. Эти результаты получены посредством сведения данных проблем к некоторым разрешимым проблемам векторной алгебры. Установлена также неразрешимость проблемы эквивалентности в некотором подклассе схем. Этот результат, как и аналогичный результат работы [7] о неразрешимости, существенно использует память, состоящую более чем из одной ячейки. В то же время при «монолитной» памяти, как это имеет место для схем Янова, проблема эквивалентности разрешима. В пятом разделе, посредством сужения класса счетчиковых схем, определяется понятие параллельной операторной схемы. Параллельные операторные схемы можно рассматривать как обобщение последовательных операторных схем, приспособленное для представления параллельных вычислений. В шестом разделе исследуются параллельные операторные схемы без логических разветвлений (decision-free). Для этих схем проблема эквивалентности оказывается разрешимой.

1. ПАРАЛЛЕЛЬНЫЕ СХЕМЫ ПРОГРАММ

Программа может быть представлена как множество элементарных операторов, использующих ячейки памяти и воздействующих на них, для которых указаны правила включения и выключения. Такое представление легло в основу построения нашей модели.

Определение 1.1. Схема (параллельная схема программы) $\mathscr{P} = (M, A, \mathscr{T})$ определяется следующим образом:

1. M — множество *ячеек памяти.*

2. $A = \{a, b, \ldots\}$ — конечное множество *операторов*; для каждого оператора a из A заданы:

(i) $K(a)$ — *количество символов выключения* оператора a (целое положительное число),

(ii) $D(a) \subseteq M$ — множество *входных ячеек* оператора a,

(iii) $R(a) \subseteq M$ — множество *выходных ячеек* оператора a.

3. Четверка $\mathscr{T} = (Q, q_0, \Sigma, \tau)$ — *управление* схемы, где

(i) Q — множество *состояний*,

(ii) q_0 — выделенное *начальное* состояние,

(iii) каждому оператору a сопоставлены один *символ включения* $\bar{a}$ и множество *символов выключения* $\{a_1, \ldots, a_{K(a)}\}$; $\Sigma = \Sigma_i \cup \Sigma_t$, где

$$\Sigma_i = \bigcup_{a \in A} \{\bar{a}\}, \quad \Sigma_t = \bigcup_{a \in A} \{a_1, \ldots, a_{K(a)}\},$$

(iv) τ — *функция переходов* — частичная функция из $Q \times \Sigma$ в Q, всюду определенная на $Q \times \Sigma_t$.

Управление $\mathscr{T}$ формирует последовательность выполнения операторов. Элементы из Σ являются элементарными шагами вычислений. Элементы из Σ_i обозначают акты включения операторов, элементы из Σ_t — акты выключения при фиксации символов выключения (например, логическое условие в схеме Янова имеет два символа выключения 1,0. — *Перев.*). Включение оператора a допустимо лишь в таком состоянии q управления $\mathscr{T}$, что значение $\tau(q, \bar{a})$ определено. После включения оператора завершение его работы допустимо в произвольном состоянии, поскольку функция τ всюду определена на $Q \times \Sigma_t$. При включении оператор a использует значения ячеек из $D(a)$, при выключении он засылает свои результаты в ячейки из $R(a)$ и формирует символ выключения. При этом символ выключения соответствует одному из $K(a)$ направлений условной передачи.

Управление $\mathscr{T}$ можно представить в виде графа, вершины которого являются состояниями, а дуги соответствуют переходам из одних состояний в другие. Если множество состояний конечно, то граф управления отчасти похож на общепринятую

конструкцию схемы программы. Отличается же он от последней тем, что не требует чередования включений и выключений операторов.

Прежде чем дать точное описание процесса выполнения схемы, отметим, что само определение схемы не содержит интерпретации ее операторов, т. е. операторам не сопоставлены конкретные функции. Указаны лишь входные и выходные переменные операторов. В статье исследуются как раз те свойства схем, которые сохраняются при произвольной интерпретации операторов. Для установления этих свойств дадим строгое определение интерпретации и процесса выполнения схемы при фиксированной интерпретации.

Определение 1.2. Полагаем, что задана *интерпретация* $\mathcal{J}$ схемы $\mathcal{S}$, если заданы

(i) отображение C, сопоставляющее каждой ячейке $i \in M$ множество $C(i)$ допустимых значений этой ячейки,

(ii) начальное значение памяти $c_0 \in \underset{i \in M}{\mathbf{X}} C(i)$,

(iii) для каждого оператора a заданы две функции:

$$F_a\colon \underset{i \in D(a)}{\mathbf{X}} C(i) \to \underset{i \in R(a)}{\mathbf{X}} C(i),$$

$$G_a\colon \underset{i \in D(a)}{\mathbf{X}} C(i) \to \{a_1, a_2, \ldots, a_{K(a)}\}.$$

Итак, множество $C(i)$ состоит из возможных значений ячейки памяти i, c_0 — начальное значение памяти. Функция F_a определяет результат, который оператор после выполнения засылает в выходные ячейки. После выполнения оператора a символ его выключения определяется функцией G_a.

Определение 1.3. *$\mathcal{J}$-описание* α есть тройка (c, q, μ), где

(i) $C \in \underset{i \in M}{\mathbf{X}} C(i)$ — состояние памяти,

(ii) q — некоторое состояние управления,

(iii) μ — отображение, сопоставляющее каждому оператору $a \in A$ некоторую конечную последовательность (возможно, пустую) элементов из $\underset{i \in D(a)}{\mathbf{X}} C(i)$.

$\mathcal{J}$-описание соответствует некоторому моменту выполнения схемы; c — значение памяти, q — состояние управления. Для каждого оператора a значение $\mu(a)$ есть последовательность значений памяти $D(a)$. Каждое из этих значений соответствует некоторому включенному, но еще не выключенному экземпляру оператора a. Забегая вперед (определение 1.5), отметим, что экземпляры оператора a выключаются в том же порядке, в котором они включались.

Определение 1.4. *Начальным $\mathcal{J}$-описанием* назовем $\mathcal{J}$-описание $\alpha_0 = (c_0, q_0, \mu_0)$, где μ_0 сопоставляет каждому оператору a пустую последовательность.

В дальнейшем операторы будут обозначаться малыми буквами из начала латинского алфавита, элементы из Σ — буквами σ, π. Конечные или бесконечные слова над алфавитом Σ будут обозначаться буквами из конца латинского алфавита, Σ^ω — множество всех бесконечных, Σ^* — конечных слов над Σ. Длину слова $x \in \Sigma^*$ обозначим $l(x)$. Обозначим k-й элемент слова x через x_k, префикс длины k слова x через ${}_kx$, где $k \leqslant l(x)$.

Следующее определение дает правило для формирования последовательности операторов в процессе выполнения схемы.

Определение 1.5. Определим частичную функцию $\alpha \cdot \sigma$, где $\alpha(c, q, \mu)$ является произвольным $\mathcal{J}$-описанием, а $\sigma \in \Sigma$, следующим образом:

1. Если $\sigma = \bar{a}(a \in A)$, то значение $\alpha \cdot \bar{a}$ определено в том и только в том случае, когда определено $\tau(q, \bar{a})$. Пусть это значение определено. Тогда $\alpha \cdot \bar{a} = (c', q', \mu')$, где
   (i) $c' = c$,
   (ii) $q' = \tau(q, \bar{a})$,
   (iii) если $b \neq a$, то $\mu'(b) = \mu(b)$; $\mu'(a)$ получается присоединением к концу последовательности $\mu(a)$ элемента $\mathop{\mathbf{X}}\limits_{i \in D(a)} c(i)$.

2. Для $\sigma = a_j(a \in A)$ значение $\alpha \cdot a_j$ определено в том и только в том случае, когда последовательность $\mu(a)$ не пуста и $G_a(\xi) = a_j$, где ξ — первый элемент последовательности $\mu(a)$. Пусть это значение определено. Тогда $\alpha \cdot a_j = (c', a', \tilde{\mu}')$, где
   (i) для $i \notin R(a)$ выполнено $c'(i) = c(i)$,
   (ii) для $i \in R(a)$ значение $c'(i)$ равно компоненте $F_a(\xi)$, соответствующей переменной i,
   (iii) $q' = \tau(q, a_j)$,
   (iv) для $b \neq a$ выполнено $\mu'(b) = \mu(b)$ и $\mu(a) = \xi\mu'(a)$.

Из перечисленных правил следует, что выключения экземпляров оператора a осуществляются в том же порядке, что и включения.

Последовательность $\mu(a)$ можно рассматривать как очередь типа «первым пришел — первым ушел». Это формальное представление очереди для, например, включившегося, но не выключившегося оператора a. Оно достаточно для наших целей, однако допускает большое разнообразие физических интерпретаций. Например, $\mu(a)$ можно представить себе как действительную очередь, в которой наборы данных оператора a находятся в ожидании до тех пор, пока не появится доступ к соответствующим вычислительным возможностям. Совершенно другая физическая интерпретация состоит в том, что вычислительные воз-

можности имеются в наличии для любого числа одновременных выполнений каждого оператора и символы включения обозначают действительное начало выполнения операторов. При этом элементы из $\mu(a)$ соответствуют одновременным выполнениям оператора a и очередь данных по существу не нужна. Синхронизация выполнений экземпляров оператора a здесь произвольна и существен лишь порядок присваивания элементов последовательности $\mu(a)$: выполнение, соответствующее первому присваиванию, завершается первым. Легко представить и другие физические интерпретации для $\mu(a)$.

Расширим функцию τ обычным образом, полагая $\tau(q, x\sigma) = \tau(\tau(q, x), \sigma)$, где левая сторона равенства определена в том и только в том случае, когда определена правая. Аналогичным образом расширим частичную функцию $\alpha \cdot \sigma$.

Определение 1.6. Произвольное конечное или бесконечное слово x над алфавитом Σ назовем *$\mathcal{J}$-вычислением*[1]), если выполняются следующие свойства:

(i) для каждого префикса y слова x значения $\alpha_0 \cdot y$ определено;

(ii) если слово x конечно, то для каждого $\sigma \in \Sigma$ значение $\alpha_0 \cdot x\sigma$ не определено;

(iii) *свойство ограниченности задержки.* Пусть y является префиксом слова x и символ $\sigma \in \Sigma$ обладает следующим свойством: для каждого z, такого, что yz является префиксом слова x, значение $\alpha_0 \cdot yz\sigma$ определено. Тогда существует z', такое, что $yz'\sigma$ является префиксом слова x.

$\mathcal{J}$-вычисления являются допустимыми последовательностями шагов алгоритма, полученного из схемы $\mathcal{P}$ посредством интерпретации $\mathcal{J}$. Из того, что при фиксированном α значение $\alpha \cdot \sigma$ может быть определенным для нескольких элементов σ, следует, что схема может порождать более чем одно $\mathcal{J}$-вычисление. Естественно потребовать, чтобы все $\mathcal{J}$-вычисления вырабатывали, в некотором смысле, одинаковые результаты.

Определение 1.7. Пусть x есть $\mathcal{J}$-вычисление или префикс $\mathcal{J}$-вычисления. *Полная история* последовательности x есть последовательность $\psi(x)$ $\mathcal{J}$-описаний:

$$\psi(x) = \alpha_0,\ \alpha_0 \cdot {}_1x,\ \alpha_0 \cdot {}_2x,\ \ldots,\ \alpha_0 \cdot {}_kx,\ \ldots\ .$$

Пусть $\psi_i(x)$ обозначает такую подпоследовательность последовательности $\psi(x)$, которая, во-первых, содержит α_0, во-вторых, для произвольного k содержит $\alpha_0 \cdot {}_kx$ в том и только в том случае, когда x_k является символом выключения такого оператора a, что $i \in R(a)$.

[1]) Общепринят более развернутый термин: *вычислительный процесс при интерпретации $\mathcal{J}$. — Прим. перев.*

Таким образом, $\psi_i(x)$ представляет собой последовательность $\mathcal{J}$-описаний, соответствующих выключениям операторов, засылающих результаты в ячейку i.

Для выделения компонент из $\mathcal{J}$-описания $\alpha = (c, q, \mu)$ введем несколько операций проецирования [1]):

$$\Pi_i(\alpha) = c(i), \quad i \in M,$$
$$\Pi_a(\alpha) = \mu(a), \quad a \in A.$$

Пусть множество $S = \{i_1, i_2, \ldots, i_r\}$ содержится в M, причем $i_1 < i_2 < \ldots < i_r$. $\Pi_S(\alpha)$ определим как набор $\langle c(i_1), c(i_2), \ldots, c(i_r)\rangle$. Операции проецирования могут применяться и к последовательностям $\mathcal{J}$-описаний. Например, $\Pi_i(\alpha^{(0)}, \alpha^{(1)}, \ldots) = \Pi_i(\alpha^{(0)}), \Pi_i(\alpha^{(1)}), \ldots$.

Определение 1.8. Пусть x является $\mathcal{J}$-вычислением или префиксом $\mathcal{J}$-вычисления. Последовательность $\Omega_i(x) = \Pi_i(\psi_i(x))$ назовем *историей ячейки* i для последовательности x.

Таким образом, $\Omega_i(x)$ есть последовательность, состоящая из начального значения ячейки i и значений, появляющихся в этой ячейке при выключении операторов, засылающих в i.

Определение 1.9. Схему $\mathcal{S}$ назовем *однозначной*, если для произвольной интерпретации $\mathcal{J}$ и произвольных $\mathcal{J}$-вычислений x, y этой схемы для каждого $i \in M$ выполнено

$$\Omega_i(x) = \Omega_i(y).$$

Таким образом, для однозначной схемы фиксированная интерпретация однозначно определяет историю каждой ячейки, причем эта история одинакова для всех вычислений, допустимых при данной интерпретации.

Определение 1.10. Схемы $\mathcal{S} = (M, A, \mathcal{T}) = \mathcal{S}' = (M, A, \mathcal{T}')$ назовем эквивалентными [2]), если для каждой ячейки $i \in M$ и для каждой интерпретации $\mathcal{J}$ выполнено

$$\{\Omega_i(x) \mid x \text{ является } \mathcal{J}\text{-вычислением схемы } \mathcal{S}\} =$$
$$= \{\Omega_i(y) \mid y \text{ является } \mathcal{J}\text{-вычислением схемы } \mathcal{S}'\}.$$

Определение 1.11. Схему $\mathcal{S}$ назовем *ограниченной*, если существует такая константа K, что для каждой интерпретации $\mathcal{J}$ и каждого $\mathcal{J}$-описания (c, q, μ), содержащегося в исто-

[1]) Полагаем, что $A \cap M = \varnothing$. Без этого ограничения введенные обозначения нуждались бы в уточнении.

[2]) В более широком контексте теории параллельного программирования этот вид эквивалентности получил название эквивалентности по истории ячеек. — *Прим. перев.*

рии некоторого $\mathcal{J}$-вычисления, сумма длин всех последовательностей $\mu(a)$ (суммирование берется по всем $a \in A$) не превышает K. При $k = 1$ схема $\mathcal{S}$ является *последовательной* (serial) [1]).

Для ограниченной схемы ограничено количество одновременных активностей операторов, т. е. ограничен параллелизм этой схемы. Для последовательной схемы одновременные активности недопустимы.

В настоящей статье основное место будет отведено изучению условий алгоритмической разрешимости таких свойств как однозначность, эквивалентность и ограниченность схем.

2. ОДНОЗНАЧНОСТЬ

Одним из наиболее интересных явлений, связанных с параллельными вычислениями, является неопределенность, состоящая в том, что несколько операторов могут выполняться одновременно или одновременно могут иметь доступ к включению; при этом результаты выполнения схемы могут зависеть от порядка включений и выключений операторов. В настоящем разделе будет установлен критерий однозначности схемы. Полученные здесь результаты справедливы лишь для некоторого подкласса схем при определенных ограничениях на управление и операторы. Определим несколько свойств, нужных для формулировки этих ограничений.

Определение 2.1. Схему назовем *устойчивой* (persistent), если для каждой пары различных элементов $\sigma, \pi \in \Sigma$ значения $\tau(q, \sigma)$, $\tau(q, \pi)$ определены в том и только в том случае, когда определены $\tau(q, \sigma\pi)$, $\tau(q, \pi\sigma)$.

Таким образом, готовность оператора к включению в устойчивой схеме сохраняется до тех пор, пока не реализуется.

Определение 2.2. Схему назовем *коммутативной*, если всегда из определенности значений $\tau(q, \sigma\pi)$, $\tau(q, \pi\sigma)$ следует равенство этих значений.

В коммутативной схеме результирующее состояние управления не зависит от перестановки одновременно допустимых элементарных шагов.

Определение 2.3. Схему назовем *нейтральной*, если всегда для произвольных символов включения σ, π, из определенности значения $\tau(q, \sigma\pi)$ следует определенность $\tau(q, \pi)$.

[1]) Заметим, что последовательность схемы в общем случае не влечет ее однозначность, поскольку допускается альтернатива символов включения на очередном шаге. — *Прим. перев.*

В нейтральной схеме включение одних операторов не дает новых возможностей для включения других операторов, не имевших доступ к включению.

Определение 2.4. Схему $\mathscr{S}$ назовем *результативной* (lossless), если для каждого оператора $a \in A$ выполнено $R(a) \neq \varnothing$.

Таким образом, в результативной схеме каждый оператор имеет непустое множество выходных переменных.

Следующая теорема устанавливает в некотором частном случае связь между однозначностью схемы и коммутативностью элементарных шагов вычислений.

Теорема 2.1. *Пусть схема $\mathscr{S}$ устойчива, коммутативна и результативна. Тогда $\mathscr{S}$ является однозначной в том и только в том случае, когда для каждой интерпретации $\mathscr{J}$ выполняется следующее условие*:

(A) *Если α_0 — начальное $\mathscr{J}$-описание, $u \in \Sigma^*$ и σ, $\pi \in \Sigma$ таковы, что значения $\alpha_0 \cdot u\sigma\pi$, $\alpha_0 \cdot u\pi\sigma$ определены, то эти значения равны.*

Прежде чем приступить к доказательству этой теоремы, рассмотрим интерпретации специального вида.

Определение 2.5. Интерпретацию $\mathscr{J}$ назовем *взаимно однозначной*, если для произвольных операторов a, b, произвольных $s \in \mathop{\mathbf{X}}\limits_{i \in D(a)} c(i)$, $s' \in \mathop{\mathbf{X}}\limits_{i \in D(b)} c(i)$ и произвольной ячейки $i \in R(a) \cap R(b)$ из равенства

$$\Pi_i(F_a(s)) = \Pi_i(F_b(s'))$$

следует $a = b$ и $s = s'$ [1]).

Лемма 2.1. *Пусть $\mathscr{J}$ — произвольная интерпретация схемы $\mathscr{S}$, α_0 — ее начальное описание. Тогда существует взаимно однозначная интерпретация $\mathscr{J}'$ с некоторым начальным описанием α_0', порождающая такое же множество вычислений, как и $\mathscr{J}$, и удовлетворяющая следующему свойству. Если для последовательностей x, y значения $\alpha_0 \cdot x$, $\alpha_0 \cdot y$ определены, то выполнено*

$$\Pi_i(\alpha_0 \cdot x) \neq \Pi_i(\alpha_0 \cdot y) \Rightarrow \Pi_i(\alpha_0' \cdot x) \neq \Pi_i(\alpha_0' \cdot y).$$

Доказательство. Интерпретация $\mathscr{J}'$ конструируется из $\mathscr{J}$ таким образом, что каждому значению ячейки памяти ставится в соответствие пара: это значение и «формула» для вычисления этого значения. Интерпретация $\mathscr{J}$ определяется фик-

[1]) Это понятие аналогично понятию свободной интерпретации. — *Прим. перев.*

сацией $C(i)$, c_0, $\{F_a \mid a \in A\}$ и $\{G_a \mid \in A\}$. Интерпретация $\mathcal{J}'$ определяется следующим образом:

$$C'(i) = C(i) \times B^*, \quad \text{где} \quad B = A \cup \{i, (,)\},$$
$$\Pi_i(c_0') = (\Pi_i(c_0), i).$$

Опишем правила формирования формул. Пусть $D(a)$ содержит r элементов. Тогда операции F_a, F_a' имеют r аргументов, причем для каждого $i \in R(a)$ выполнено

$$\Pi_i(F_a'((\xi_1, \eta_1), (\xi_2, \eta_2), \ldots, (\xi_r, \eta_r))) =$$
$$= (\Pi_i(F_a(\xi_1, \xi_2, \ldots, \xi_r)), \mathrm{Cat}_a(\eta_1, \eta_2, \ldots, \eta_r)).$$

Здесь Cat_a определяется соотношением

$$\mathrm{Cat}_a(x_1, \ldots, x_r) = a(x_1)(x_2) \ldots (x_r);$$

для произвольных аргументов из B^* значение этой функции принадлежит B^*;

$$G_a'((\xi_1, \eta_1), (\xi_2, \eta_2), \ldots, (\xi_r, \eta_r)) = G_a(\xi_1, \xi_2, \ldots, \xi_r).$$

Индукцией по $l(x)$ легко доказать следующие утверждения.

1. Значение $(c_0, q_0, \mu_0) \cdot x$ определено относительно интерпретации $\mathcal{J}$ тогда и только тогда, когда $(c_0', q_0, \mu_0) \cdot x$ определено относительно $\mathcal{J}'$.

2. Если определены значения $(c_0, q_0, \mu_0) \cdot x$, $(c_0', q_0, \mu_0) \cdot x$, то из $\Pi_i((c_0', q_0, \mu_0) \cdot x) = (\xi, \eta)$ следует $\Pi_i((c_0, q_0, \mu_0) \cdot x) = \xi$. Из этих утверждений можно вывести взаимную однозначность интерпретации $\mathcal{J}'$.

Действительно, при выполнении

$$\Pi_i(F_a'((\xi_1, \eta_1), (\xi_2, \eta_2), \ldots, (\xi_r, \eta_r))) =$$
$$= \Pi_i(F_b'((\lambda_1, \delta_1), (\lambda_2, \delta_2), \ldots, (\lambda_t, \delta_t)))$$

из свойств Cat_a, Cat_b следуют равенства $a = b$, $r = t$, $n_j = \delta_j$ $(j = 1, 2, \ldots, r)$. Далее, если в ходе $\mathcal{J}'$-вычисления ячейка i принимала значения (ξ, η), (λ, r), то $\xi = \lambda$, поскольку формула η однозначно определяет значение ячейки i как композицию функций F_a, F_b, ... Отсюда следует взаимная однозначность интерпретации $\mathcal{J}'$.

Для доказательства теоремы 2.1 нам понадобятся несколько лемм.

Лемма 2.2. *Условие* (А) *из теоремы* 2.1 *выполняется для каждой интерпретации тогда и только тогда, когда оно выполняется для каждой взаимно однозначной интерпретации.*

Доказательство. Очевидно, что из выполнения условия (A) для каждой интерпретации следует его выполнение для взаимно однозначных интерпретаций. Обратно, пусть $\mathfrak{J}$ — произвольная интерпретация, $\mathfrak{J}'$ — взаимно однозначная интерпретация из леммы 2.1. Из выполнения условия (A) для $\mathfrak{J}'$ следует выполнение его для $\mathfrak{J}$.

Лемма 2.3. *Пусть задана устойчивая, коммутативная, результативная схема* $\mathfrak{P}$, *взаимно однозначная интерпретация* $\mathfrak{J}$, $\mathfrak{J}$*-описание* α. *Тогда для каждой пары операторов* a, b *выполнены следующие свойства*:

(a) *если определены значения* $\alpha \cdot \bar{a}\bar{b}$, $\alpha \cdot \bar{b}\bar{a}$, *то эти значения равны*;

(b) *если определены значения* $\alpha \cdot \bar{a}b_l$, $\alpha \cdot b_l\bar{a}$, *то эти значения равны в том и только в том случае, когда выполнено либо свойство*

(i) $R(b) \cap D(a) = \varnothing$,

либо свойство

(ii) b_l *является слабым* (*repetition*), *т. е.* $\Pi_{R(b)}\alpha = \Pi_{R(b)}(\alpha \cdot b_l)$;

(c) *если значения* $\alpha \cdot a_jb_l$, $\alpha \cdot b_la_j$ *определены, то они равны в том и только в том случае, когда* $R(a) \cap R(b) = \varnothing$.

Доказательство. Пусть $\alpha = (c, q, \mu)$.

(a) При $a = b$ утверждение очевидно. Пусть $a \neq b$. Тогда $(c, q, \mu) \cdot \bar{a}\bar{b} = (c, \tau(q, \bar{a}\bar{b}), \mu')$, где $\mu'(d) = \mu(d)$, $d \notin \{a, b\}$, $\mu'(a) = \mu(a)\Pi_{D(a)}(c)$; $\mu'(b) = \mu(b)\Pi_{D(b)}(c)$; значение $(c, q, \mu) \cdot \bar{b}\bar{a}$ определяется аналогично. Используя коммутативность τ, получим искомое равенство $(c, q, \mu) \cdot \bar{a}\bar{b} = (c, q, \mu) \cdot \bar{b}\bar{a}$.

(b) Рассмотрим вначале случай $a \neq b$. $(c, q, \mu) \cdot \bar{a}b_l = (c', \tau(q, \bar{a}b_l), \mu')$ и $(c, q, \mu) \cdot b_l\bar{a} = (c', \tau(q, b_l\bar{a})\mu'')$, где $\mu'(d) = \mu''(d) = \mu(d)$, $d \neq a$, b, $\mu'(b) = \mu''(b)$, $\mu'(a) = \mu(a)\Pi_{D(a)}(c)$, $\mu''(a) = \mu(a)\Pi_{D(a)}(c')$. Следовательно, в силу коммутативности τ, равенство $(c, q, \mu) \cdot \bar{a}b_l = (c, q, \mu) \cdot b_l\bar{a}$ выполняется тогда и только тогда, когда $\Pi_{D(a)}(c') = \Pi_{D(a)}(c)$. Аналогичное рассуждение можно провести и при $a = b$. Отсюда, в силу взаимной однозначности интерпретации $\mathfrak{J}$, следует, что равенство $\Pi_{D(a)}(c') = \Pi_{D(a)}(c)$ выполняется в том и только в том случае, когда либо $R(b) \cap D(a) = \varnothing$, либо символ b_l является слабым.

(c) При $a = b$ из определенности значений $(c, q, \mu) \cdot a_jb_l$ и $(c, q, \mu) \cdot b_la_j$ следует $j = l$. Пусть $a \neq b$. Тогда $(c, q, \mu) \cdot a_jb_l = (c', \tau(q, a_jb_l), \mu')$ и $(c, q, \mu) \cdot b_la_j = (c'', \tau(q, b_la_j), \mu')$, где c' не совпадает с c'' в тех и только тех ячейках, которые принадлежат $R(b) \cap R(a)$. Следовательно, равенство $\alpha \cdot a_jb_l = \alpha \cdot b_lc_j$ выполняется в том и только в том случае, когда это пересечение пусто.

Лемма 2.4. *Пусть схема $\mathcal{P}$ устойчива, коммутативна, результативна, задана взаимно однозначная интерпретация $\mathcal{J}$ с начальным $\mathcal{J}$-описанием α_0. Пусть при этом $v \in \Sigma^*$, σ, $\pi \in \Sigma$, $\alpha_0 \cdot v\sigma\pi = \alpha_0 \cdot v\pi\sigma$. Тогда для каждого $w \in \Sigma^* \cup \Sigma^\omega$ выполнены свойства*

(a) *$v\sigma\pi w$ является $\mathcal{J}$-вычислением тогда и только тогда, когда $\mathcal{J}$-вычислением является $v\pi\sigma w$;*

(b) *для произвольного $i \in M$ выполнено $\Omega_i(v\sigma\pi w) = \Omega_i(v\pi\sigma w)$.*

Доказательство. Для проверки утверждения (a) достаточно показать, что слово $v\sigma\pi w$ удовлетворяет свойствам из определения 1.6 (т. е. является $\mathcal{J}$-вычислением) в том и только в том случае, когда этим свойствам удовлетворяет слово $v\pi\sigma w$. Из доказательства леммы 2.3 видно, что из равенства $\alpha_0 \cdot v\sigma\pi = \alpha_0 \cdot v\pi\sigma$ следует $\Omega_i(v\sigma\pi) = \Omega_i(v\pi\sigma)$. Отсюда легко следует утверждение (b).

Лемма 2.5. *Пусть заданы устойчивая схема $\mathcal{P}$, интерпретация $\mathcal{J}$, начальное $\mathcal{J}$-описание α_0. Пусть u, $v \in \Sigma^*$, $w \in \Sigma^* \cup \Sigma^\omega$, $\sigma \in \Sigma$. Тогда справедливы следующие утверждения:*

(a) *если значение $\alpha_0 \cdot u\sigma$ определено и для некоторого v ($\sigma \notin v$) значение $\alpha_0 \cdot uv$ определено, то значение $\alpha_0 \cdot uv\sigma$ определено;*

(b) *если $\alpha_0 \cdot u\sigma$ определено и uw является $\mathcal{J}$-вычислением, то $\sigma \in w$.*

Доказательство. Утверждение (a) следует из определения 1.6, причем устойчивость схемы используется лишь в ситуации, когда σ — символ включения. Утверждение (b) следует из (a) и свойства ограниченности задержки.

Доказательство теоремы 2.1. Пусть $\mathcal{J}$ — взаимно однозначная интерпретация, для которой выполнено условие (A). Пусть для $\mathcal{J}$-вычислений x, y и для некоторого i выполнено $\Omega_i(x) \neq \Omega_i(y)$. Докажем, что для каждого $n \leqslant l(x)$ (для всех n, если x — бесконечная последовательность) существует такое $\mathcal{J}$-вычисление $z(n)$, что

(i) истории ячеек вычислений $z(n)$, y совпадают,

(ii) ${}_n z(n) = {}_n(x)$. Отсюда будет следовать выполнение равенства $\Omega_i(x) = \Omega_i(y)$ для всех i, что является противоречием. Доказательство проводим индукцией по n. Для $n = 0$ полагаем $z(0) = y$. Пусть для $n = k$ утверждение доказано и $l(x) = k + 1$. Из предположения индукции следует, что значение $(\alpha_0 \cdot ({}_k x)) \cdot x_{k+1}$ определено. Это равносильно определенности $(\alpha_0 \cdot ({}_k z(k))) \cdot x_{k+1}$. Пусть t обозначает ${}_k z(k)$. Из утверждения (b) леммы 2.5 следует, что $z(k)$ может быть представлено в виде $z(k) = tvx_{k+1}u$ при некотором v ($x_{k+1} \in v$). Если слово v пусто, то можно взять $z(k+1) = z(k)$. В противном случае $v = w\pi$ ($\pi \in \Sigma$). Так как $(\alpha_0 \cdot t) \cdot x_{k+1}$ определено, то из утверждения (a) леммы 2.5 следует

определенность значений $(\alpha_0 \cdot tw) \cdot x_{k+1}$, $(\alpha_0 \cdot tw\pi) \cdot x_{k+1}$. Из определенности $(\alpha_0 \cdot tw) \cdot \pi$ следует определенность $(\alpha_0 \cdot twx_{k+1}) \cdot \pi$. По предположению индукции выполнено $(\alpha_0 \cdot tw) \cdot x_{k+1}\pi = (\alpha_0 \cdot tw) \times \times \pi x_{k+1}$. Следовательно, по лемме 2.4 слово $twx_{k+1}\pi u$ является $\mathcal{J}$-вычислением, последовательность элементов которого совпадает с $z(k)$. Посредством таких перестановок можно вставлять x_{k+1} после каждого v, достигая искомого $\mathcal{J}$-вычисления $z(k+1) = (_{k+1}x)\, vu$.

Покажем, что только что полученный результат противоречит предположению о том, что $\Omega_i(x) \neq \Omega_i(y)$ для некоторого i. Действительно, из $\Omega_i(k) \neq \Omega_i(y)$ следует, что для некоторого положительного целого k выполнено одно из следующих условий:

(i) значения $[\Omega_i(x)]_k$, $[\Omega_i(y)]_k$ оба определены, но не равны;

(ii) значение $[\Omega_i(y)]_k$ определено, $[\Omega_i(x)]_k$ не определено;

(iii) $[\Omega_i(x)]_k$ определено, $[\Omega_i(y)]_k$ не определено.

Для (i) и (ii) существует достаточно большое p, такое, что $[\Omega_i(z(p))]_k \neq [\Omega_i(x)]_k$. Это противоречит свойству $\Omega_i(z(p)) = \Omega_i(y)$. Случай (iii) приводит к противоречию аналогичным образом. Итак, выполнение условия (А) влечет однозначность схемы.

Продолжим доказательство. Предположим, что $\alpha_0 \cdot u\sigma\pi \neq \neq \alpha_0 \cdot u\pi\sigma$. Тогда по лемме 2.3 либо $\pi = b_l$ и $\sigma = \bar{a}$, где $R(b) \cap D(a) \neq \varnothing$ и символ b_l не является повторным, либо $\pi = b_l$ и $\sigma = a_j$, где $R(b) \cap R(a) \neq \varnothing$. В любом случае существует такое i, что последовательности $\Omega_i(u\sigma\pi)$, $\Omega_i(u\pi\sigma)$ имеют равную длину и различные последние элементы. Если эти последовательности являются префиксами вычислений x', x'' соответственно, то $\Omega_i(x') \neq \Omega_i(x'')$. Следовательно, условие (А) является необходимым и достаточным условием однозначности.

Следствие 2.1. *Устойчивая, коммутативная, результативная схема $\mathcal{P}$ является однозначной тогда и только тогда, когда для каждой интерпретации $\mathcal{J}$ при начальном $\mathcal{J}$-описании α_0 выполнены следующие условия*:

(i) *если значения* $\alpha_0 \cdot ua\bar{b}b_l$, $\alpha_0 \cdot ub_l\bar{a}$ *определены, то либо* $R(b) \cap D(a) = \varnothing$, *либо* $\Pi_{R(b)}(\alpha_0 \cdot ub_l) = \Pi_{R(b)}(\alpha_0 \cdot u)$ (*т. е. символ* b_l *повторен*);

(ii) *если значения* $\alpha_0 \cdot ua_jb_l$, $\alpha_0 \cdot ub_la_j$ *определены, то* $R(b) \cap \cap R(a) = \varnothing$.

Это утверждение непосредственно следует из теоремы 2.1 и леммы 2.3.

Наш критерий однозначности допускает упрощение для схем, вычисления которых обладают следующим свойством: между произвольными включениями оператора происходит засылка в некоторую входную ячейку этого оператора.

Определение 2.6. Схему $\mathcal{P}$ назовем *свободной* (repetition-free), если для произвольного слова $v\bar{a}w\bar{a}x$, являющегося

$\mathcal{J}$-вычислением при некотором $\mathcal{J}$, существует такой символ выключения $c_j \in w$, что $R(c) \cap D(a) = \varnothing$.

Для свободной схемы $\mathcal{S}$ случай слабого символа b_l в критерии однозначности может быть опущен. Для нейтральных схем возможно дальнейшее упрощение. Для формулировки этого упрощения определим отношение $\rho \subseteq A \times A$, полагая $a\rho b \Leftrightarrow R(a) \neq \varnothing$, $R(b) \neq \varnothing$ и $[D(a) \cap R(b)] \cup [R(a) \cap D(b)] \cup \cup [R(a) \cap R(b)] \neq \varnothing$.

Введем полезное для дальнейшего определение. Пусть $a \in \in A$, $x \in \Sigma^* \cup \Sigma^\omega$ и m — положительное целое число. Произвольное m-е вхождение символа включения $\bar{a}$ назовем *согласованным* (mate) с m-м вхождением символа выключения этого оператора (здесь не проводится различение между различными символами выключения. — *Перев.*).

Следствие 2.2. *Пусть схема $\mathcal{S}$ является свободной, результативной, устойчивой, коммутативной и нейтральной. Тогда эквивалентны следующие утверждения:*

1. *$\mathcal{S}$ не является однозначной.*

2. *Существует интерпретация $\mathcal{J}$ с начальным $\mathcal{J}$-описанием α_0, для которой существуют $u \in \Sigma^*$, $a \in A$, $b \in A$, такие, что либо выполнено*

(i) *$R(b) \cap D(a) = \varnothing$ и существует такое l, что значения $\alpha_0 \cdot u\bar{a}b_l$, $\alpha_0 \cdot ub_l\bar{a}$ определены, либо выполнено*

(ii) *$R(a) \cap R(b) = \varnothing$ и для некоторых j, l значения $\alpha_0 \cdot ua_jb_l$, $\alpha_0 \cdot ub_la_j$ определены.*

3. *Для некоторой интерпретации $\mathcal{J}$ с начальным $\mathcal{J}$-описанием α_0 существуют $w \in \Sigma^*$, $a \in A$, $b \in A$, такие, что выполнено $a\rho b$ и значения $\alpha_0 \cdot w\bar{a}$, $\alpha_0 \cdot w\bar{b}$ определены.*

4. *Для некоторой интерпретации $\mathcal{J}$ существуют $\mathcal{J}$-вычисления x, y и операции a, b, такие, что выполнено $a\rho b$ и последовательность $\mathscr{E}_{\bar{a}\bar{b}}(x)$, являющаяся проекцией последовательности x на множестве $\{\bar{a}, \bar{b}\}$, не совпадает с $\mathscr{E}_{\bar{a}\bar{b}}(y)$.*

Доказательство. Установим цепочку (1) $\Rightarrow$ (2) $\Rightarrow$ $\Rightarrow$ (3) $\Rightarrow$ (4) $\Rightarrow$ (1).

(1) $\Rightarrow$ (2) вытекает непосредственно из следствия 2.1 и предположения о свободе схемы $\mathcal{S}$.

(2) $\Rightarrow$ (3). Пусть выполнено (2), u — слово минимальной длины, такое, что существуют a, b, такие, что выполнено (i) или (ii). Пусть выполнено (ii) (случай (i) доказывается аналогично). Тогда слово u содержит вхождения символов $\bar{a}$, $\bar{b}$, согласованные с вхождениями символов a_j, b_l соответственно. Пусть $\bar{a}$ появилось раньше, чем $\bar{b}$ (это предположение не уменьшает общности рассмотрений). Тогда u может быть представлено в

виде $u = u_1\bar{a}u_2\bar{b}u_3$. Если u_2 пусто, то (3) непосредственно следует из нейтральности схемы. Если u_2 не пусто, то оно может быть представлено в виде $u_2 = cu_2'$, где $c \in \Sigma$. Если c — символ включения, то, в силу нейтральности, значение $\alpha_0 \cdot u_1c\bar{a}$ определено. Из коммутативности и определения 1.5 следует $\alpha_0 \cdot u_1c\bar{a} = \alpha_0 \cdot u_1\bar{a}c$. Следовательно, в силу утверждения (а) леммы 2.4, слово $u_1c\bar{a}u_2'\bar{b}u_3$ является префиксом $\mathcal{J}$-вычисления. Если c — символ включения, то из нейтральности следует определенность $\alpha_0 \cdot u_1$, $c\bar{a}$. Из коммутативности и определения 1.5 следует $\alpha_0 \cdot u_1c\bar{a} = \alpha_0 \cdot u_1\bar{a}c$. Отсюда и из утверждения (а) леммы 2.4 следует, что слово $u_1c\bar{a}u_2'\bar{b}u_3$ является префиксом $\mathcal{J}$-вычисления. Если c — символ выключения, то с ним согласован некоторый символ включения u_1. Теперь из устойчивости схемы и определения $\mathcal{J}$-вычисления следует определенность $\alpha_0 \cdot u_1c\bar{a}$. Поскольку u — самое короткое слово, при котором выполняется (2), то выполнено $\alpha_0 \cdot u_1c\bar{a} = \alpha_0 \cdot u_1\bar{a}c$. Следовательно, слово $u_1c\bar{a}u_2'\bar{b}u_3$ является префиксом $\mathcal{J}$-вычисления. Таким образом, символ $\bar{a}$ можно «вставлять» после каждого элемента u_2. Отсюда следует, что $u_1u_2\bar{a}\bar{b}$ также является префиксом $\mathcal{J}$-вычисления. Из нейтральности следует, что $u_1u_2\bar{b}\bar{a}$ является префиксом $\mathcal{J}$-вычисления, а поскольку $R(a) \cap R(b) \neq \varnothing$, то $a\rho b$. Отсюда следует (3).

(3) $\Rightarrow$ (4). Из устойчивости следует, что слова $w\bar{a}\bar{b}$, $w\bar{b}\bar{a}$ являются префиксами некоторых $\mathcal{J}$-вычислений x, y. Отсюда следует $\mathscr{E}_{\bar{a}\bar{b}}(x) = \mathscr{E}_{\bar{a}\bar{b}}(y)$.

(4) $\Rightarrow$ (1). Из леммы 2.1 следует, что достаточно рассмотреть взаимно однозначную интерпретацию $\mathcal{J}$. Пусть $\mathscr{E}_{\bar{a}\bar{b}}(x) \neq \mathscr{E}_{\bar{a}\bar{b}}(y)$ и предположим (противное), что схема $\mathcal{S}$ однозначная. Рассмотрим $\mathcal{J}$-вычисление $z = u_1\bar{c}u_2c_mu_3$, в котором выделены согласованные вхождения $\bar{c}$, c_m. Используя однозначность, применяем «вставку» символа согласно теореме 2.1 и получаем, что слово $u_1\bar{c}c_mu_2u_3$ является $\mathcal{J}$-вычислением. По индукции, применяя такие вставки, можно по слову x построить $\mathcal{J}$-вычисление $\tilde{x}$:

(а) длины слов x, $\tilde{x}$ равны (если x бесконечно, то и $\tilde{x}$ бесконечно),

(b) для $k = 1, 2, \ldots$ символ $(\tilde{x})_{2k-1}$ является k-м символом включения в слове x, а $(\tilde{x})_{2k}$ согласован с k-м символом включения в слове x.

Например, если бы x был равен $\bar{a}\bar{b}\bar{c}b_2c_3\bar{a}a_1\bar{d}a_2 \ldots$, то $\tilde{x}$ был бы равен $\bar{a}a_1\bar{b}b_1\bar{c}c_3\bar{a}a_2\bar{d} \ldots$. Аналогичным образом из y может быть получено $\mathcal{J}$-вычисление $\tilde{y}$, при этом $\mathscr{E}_{\bar{a}\bar{b}}(\tilde{x}) = \mathscr{E}_{\bar{a}\bar{b}}(x) \neq \mathscr{E}_{\bar{a}\bar{b}}(y) = \mathscr{E}_{\bar{a}\bar{b}}(\tilde{y})$. В силу $a\rho b$, выполнено либо $R(a) \cap R(b) \neq \varnothing$, либо $R(b) \cap D(a) \neq \varnothing$, либо $R(a) \cap D(b) \neq \varnothing$. Эти случаи могут

не иметь аналога в предыдущих доказательствах. Остановимся подробно лишь на первом из них. Фиксируем $i \in R(a) \cap R(b)$. Пусть $S_i = \{c \mid i \in R(c)\}$. Проекцию слова $\tilde{x}$ на множество символов включения операторов из S_i обозначим $\mathscr{E}_{S_i}(\tilde{x})$. Аналогичный смысл имеет $\mathscr{E}_{S_i}(\tilde{y})$. Из $\mathscr{E}_{\bar{a}\bar{b}}(\tilde{x}) \neq \mathscr{E}_{\bar{a}\bar{b}}(\tilde{y})$ и $\{a, b\} \leqslant S_i$ следует, что для некоторого p выполнено

$$[\mathscr{E}_{S_i}(\tilde{x})]_p \neq [\mathscr{E}_{S_i}(\tilde{y})]_p.$$

В силу согласованности символов включения и выключения операторов в словах $\tilde{x}$, $\tilde{y}$, p-м элементам последовательностей $\Omega_i(\tilde{x})$ и $\Omega_i(\tilde{y})$ соответствуют различные операторы. Отсюда, в силу взаимной однозначности интерпретации $\mathscr{J}$, следует $\Omega_i(\tilde{x}) \neq \Omega_i(\tilde{y})$ и неоднозначность $\mathscr{S}$. Такой же результат получается и в двух других случаях ($R(a) \cap D(b) \neq \varnothing$, $R(b) \cap D(a) \neq \varnothing$). Отсюда следует (4) $\Rightarrow$ (1).

Пункт (3) из следствия 2.2 будет использован в разд. 5 при построении эффективного алгоритма распознавания однозначности схем некоторого класса.

3. СВОЙСТВА МНОЖЕСТВА $p(\mathscr{S})$

При исследовании свойств схем, справедливых для каждой интерпретации, мы постараемся исключить понятие интерпретации при определении этих свойств. Конечное или бесконечное слово x над алфавитом Σ назовем *вычислением*, если оно является $\mathscr{J}$-вычислением при некоторой интерпретации $\mathscr{J}$. Множество всех конечных слов, являющихся префиксами вычислений схемы $\mathscr{S}$, обозначим $p(\mathscr{S})$. Следующая лемма показывает, что множество $p(\mathscr{S})$ полностью определяет множество всех вычислений схемы $\mathscr{S}$.

Лемма 3.1. *Слово* $x \in (\Sigma^* \cup \Sigma^\omega)$ *является вычислением схемы* $\mathscr{S}$ *в том и только в том случае, когда* x *удовлетворяет следующим трем условиям*:

(1) *Для каждого* k *выполнено* ${}_kx \in p(\mathscr{S})$.

(2) *Если слово* x *конечно, то для каждого* $\sigma \in \Sigma$ *выполнено* $x\sigma \notin p(\mathscr{S})$.

(3) *Если* $x \in \Sigma^\omega$, *то для каждого* $\sigma \in \Sigma$ *и для каждого* n *выполнено следующее свойство: если для каждого* $k \geqslant n$ *выполнено* $({}_kx)\sigma \in p(\mathscr{S})$, *то существует такое* $u \in \Sigma^*$, *что* $({}_nx)u\sigma$ *является префиксом слова* x.

Доказательство этой леммы легко следует из определения 1.6.

В настоящем разделе будут рассмотрены свойства множества $p(\mathscr{S})$ без привлечения понятия интерпретации. Эти свой-

ства будут использованы в разд. 4 при построении алгоритмов распознавания.

Для $x = x_1 x_2 \ldots x_t (x \in \Sigma^*)$, $i \in M$, $1 \leqslant k \leqslant t$ определим

$$n(i, k) = \max(\{0\} \cup \{r \mid r \leqslant k,\ x_r = c_j,$$

где $c \in A$, $i \in R(c)\}$). Значением $n(i, k)$ является множество позиций слова $x_1 x_2 \ldots x_k$, содержащих символы выключения операторов, для которых i является выходной ячейкой.

Определим отношение эквивалентности ($\equiv$) между вхождениями символов включения в слово x, полагая $x_k \equiv x_l$, если для некоторого a выполнено $x_k = x_l = \bar{a}$ и выполнено хотя бы одно из следующих условий:

(i) $k = l$.

(ii) для каждого $i \in D(a)$ либо значения $n(i, k)$, $n(i, l)$ равны 0, либо оба отличны от 0 и символы, согласованные с символами $x_{n(i,k)}$, $x_{n(i,l)}$, эквивалентны в смысле $\equiv$.

Таким образом, два вхождения символа $\bar{a}$ эквивалентны в смысле $\equiv$, если соответствующие выполнения оператора a осуществляются при одинаковых значениях входных ячеек.

Множество $p(\mathscr{S})$ характеризует следующая теорема.

Теорема 3.1. *Слово $x \in \Sigma^*$ принадлежит $p(\mathscr{S})$ тогда и только тогда, когда выполнены следующие три свойства.*

Свойство 1. *Для каждого $a \in A$ в каждом префиксе слова x количество символов выключений оператора a не превышает количество его символов включений.*

Свойство 2. *Значение $\tau(q_0, x)$ определено.*

Свойство 3. *Если $x_k \equiv x_l$ и для вхождений символов включения x_k, x_l существуют согласованные с ними вхождения в слове x, то эти согласованные вхождения являются вхождениями одного и того же символа выключения.*

Необходимость выполнения этих свойств очевидна. Выполнение свойства 1 следует из того, что значение $\alpha \cdot a_j$ определено только при непустой последовательности $\mu(a)$ (используем условие (2) из определения $\alpha \cdot a_j$). Выполнение свойства 2 следует из определения $\mathscr{J}$-вычисления (условие (i) из этого определения). Свойство 3 формализует тот факт, что если оператор дважды выполняется на одних и тех же данных, то результаты этих выполнений равны. Это свойство следует из однозначности функции G_a. Достаточность следует из свойств взаимно однозначной интерпретации $\mathscr{J}$, которую можно подобрать так, чтобы x являлся префиксом $\mathscr{J}$-вычисления.

Следствие 3.1. *Для свободной схемы $\mathscr{S}$ выполнение свойств* 1 *и* 2 *теоремы* 3.1 *является как необходимым, так и достаточным условием принадлежности x множеству $p(\mathscr{S})$.*

Доказательство. Для свободной схемы невозможно выполнение отношения $\equiv$ между несовпадающими символами включения. Отсюда следует избыточность свойства 3.

Схему $\mathscr{S}$ назовем *конечной* (finite-state), если конечно множество состояний ее управления [1]).

Следствие 3.2. *Пусть $\mathscr{S}$ — конечная свободная схема. Тогда множество $p(\mathscr{S})$ регулярно в том и только в том случае, когда схема $\mathscr{S}$ ограничена.*

Доказательство. Из теоремы 2 работы [9] следует, что множество $p(\mathscr{S})$ является регулярным событием в том и только в том случае, когда следующее отношение эквивалентности над Σ^* имеет конечный индекс [2]): полагаем xEy, если для каждого w выполнено

$$xw \in p(\mathscr{S}) \Leftrightarrow yw \in p(\mathscr{S}).$$

Пусть $\Delta_a(x)$ равно разности между количеством вхождений символа $\bar{a}$ и количеством вхождений символов выключений оператора a в слово x. Тогда Δ_a принимает неограниченное множество значений в том и только в том случае, когда схема $\mathscr{S}$ не является ограниченной. Отметим следующие свойства.

(1) Если $\Delta_a(x) > \Delta_a(y)$, то не выполнено xEy. Действительно, в этом случае для последовательности w, состоящей из $\Delta_a(x)$ символов окончания оператора a, выполнено $xw \in p(\mathscr{S})$, однако $y(w) \notin p(\mathscr{S})$.

(2) Если для каждого a выполнено $\Delta_a(x) = \Delta_a(y)$ и $\tau(q_0, x) = \tau(q_0, y)$, то xEy. Это следует из того, что свойства 1,2 из теоремы 3.1 выполняются для xw в том и только в том случае, когда они выполняются для yw.

Из свойства (1) следует, что неограниченность схемы $\mathscr{S}$ влечет за собой неограниченность индекса эквивалентности E. Из свойства (2) следует, что ограниченность s влечет за собой ограниченность индекса.

Следствие 3.2 имеет самостоятельное значение, так как оно соотносит степень параллелизма схемы со структурой множества $p(\mathscr{S})$, а следовательно, со сложностью механизма порождения вычислений схемы $\mathscr{S}$.

Существенность свойства свободы схемы $\mathscr{S}$ в следствии 3.2 показывает следующий пример.

Пример 3.1. На рис. 3.1 изображено управление $\mathscr{T}$ некоторой конечной схемы $\mathscr{S}$.

---

[1]) В отличие от свойства конечности ограниченность схемы состоит в ограничении степени ее параллелизма. — *Прим. перев.*

[2]) То есть конечное количество классов эквивалентности. — *Прим. перев.*

Недостающие переходы рис. 3.1 либо не определены, либо не используются в вычислениях. Полагаем

$$A=\{a, b\}, \quad K(a)=1, \quad K(b)=2,$$
$$M=\{1, 2\}, \quad D(a)=\{1\}, \quad R(a)=D(b)=R(b)=\{2\}.$$

Схема $\mathscr{S}$ является последовательной, а следовательно, ограниченной. Значение выходной ячейки 2 после выполнения оператора определяется количеством выполнений оператора b после

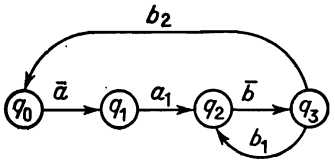


Рис. 3.1.

последнего выполнения оператора a. Следовательно, ни для какого k никакое вычисление схемы $\mathscr{S}$ не содержит пару отрезков

$$\bar{a}a_1(\bar{b}b_1)^k\bar{b}b_2,$$
$$\bar{a}a_1(\bar{b}b_1)^k\bar{b}b_1.$$

Отсюда следует, что множество $p(\mathscr{S})$ не является регулярным (оно не является и контекстно-свободным).

4. ПРОБЛЕМЫ РАЗРЕШИМОСТИ

До сих пор мы исследовали взаимосвязи между различными свойствами схем, не останавливаясь на эффективности распознавания свойств. В настоящем разделе рассматривается проблема существования алгоритмов, распознающих эти свойства. При этом мы сосредоточим внимание на некотором классе схем, названных счетчиковыми схемами. Класс счетчиковых схем (counter schemata) содержит конечные схемы и параллельные операторные схемы, определение которых дано в разд. 5.

Результаты этого раздела позволяют судить об уровне эффективности, который может быть достигнут при анализе счетчиковых схем. Большинство полученных результатов являются положительными и состоят в построении алгоритмов распознавания свойств счетчиковых схем при определенных ограничениях на схемы.

Из отрицательных результатов отметим рекурсивную неразрешимость проблемы распознавания эквивалентности для конечных последовательных схем. В конце раздела сформулированы две открытые проблемы.

При построении алгоритмов распознавания оказалось полезным привлечь векторную алгебру (система векторного сложения), перенося результаты о разрешимости некоторых свойств систем векторного сложения на схемы.

При построении алгоритмов распознавания свойств счетчиковых схем применяется следующая последовательность шагов:

1) посредством теоремы 3.1 свойство схемы $\mathscr{S}$ формулируется как свойство множества $p(\mathscr{S})$;

2) свойство множества $p(\mathscr{S})$ сводится к некоторому разрешимому свойству системы векторного сложения.

Определение 4.1. *Счетчиковой схемой* назовем схему $\mathscr{S}=(M, A, \mathscr{T})$, для которой управление $\mathscr{T}$ задается следующим образом. Пусть

k — целое неотрицательное число,

Σ — конечное множество,

S — множество с выделенным элементом s_0,

$\pi \in N^k$, где N — множество неотрицательных целых чисел,

v — функция из Σ в N^k, такая, что если $\sigma \in \Sigma_t$, то $v(\sigma) \geqslant 0$,

$\theta\colon S \times \Sigma \to S$ — частичная функция, всюду определенная на $S \times \Sigma_t$.

Полагаем $\mathscr{T}=(Q, q_0, \Sigma, \tau)$, где

$$Q = S \times N^k,$$

$$q_0 = (s_0, \pi).$$

Полагаем, что значение $\tau((s, x), \sigma)$ определено, если определено $\theta(s, \sigma)$ и выполнено $x + v(\sigma) \geqslant 0$. Если значение определено, то полагаем

$$\tau((s, x), \sigma) = (\theta(s, \sigma),\ x + v(\sigma)).$$

Таким образом, состояние управления счетчиковой схемы представлено некоторой конечной структурой и фиксированными значениями k «счетчиков» (неотрицательных чисел). Каждое включение или выключение приводит к увеличению или уменьшению этих значений.

А. *Системы векторного сложения*

Системы векторного сложения положены нами в основу алгоритмов распознавания свойств схем программ. Эти системы представляют и самостоятельный интерес как математические структуры. Они полезны не только для схем программ, но и при исследовании проблемы равенства слов в коммутативных подгруппах (эту связь активно использует М. Рабин), в теории

ограниченных контекстно-свободных языков [1], в теории однородных рекуррентных уравнений [5].

r-мерной системой векторного сложения назовем пару $\mathscr{W} = (d, W)$, где d есть r-мерный вектор, компоненты которого — неотрицательные числа, W — конечное множество r-мерных векторов, компоненты которых — целые числа. *Сферой достижимости* $R(\mathscr{W})$ назовем множество всех векторов вида $d + w_1 + w_2 + \ldots + w_s$, таких, что $w_i \in W$ $(i = 1, 2, \ldots, s)$ и $d + w_1 + w_2 + \ldots + w_i \geqslant 0$ $(i = 1, 2, \ldots, s)$. Таким образом, точка принадлежит $R(\mathscr{W})$, если она достижима из точки d последовательностью перемещений, не выводящих за пределы первого октанта (то есть все компоненты вектора, полученного в результате очередного перемещения, положительны).

При доказательстве разрешимости некоторых проблем, связанных со сферой достижимости, нам понадобится следующая терминология.

1. Определим отношение $\leqslant$ между r-мерными векторами. Полагаем $y \leqslant z$ в том и только в том случае, когда для каждого $i = 1, 2, \ldots, r$ выполнено $y_i \leqslant z_i$.

2. Вектор, все компоненты которого равны 0, иногда будем обозначать 0.

3. Полагаем, что для символа ω и для произвольного целого числа n выполнено $n < \omega$, $n + \omega = \omega$.

4. *Корневым деревом* назовем такой ориентированный граф, на одну из вершин которого (*корень* δ) не направлена ни одна дуга, на каждую из остальных вершин направлена точно одна дуга и каждая вершина которого достижима из корня.

Если ξ, η — различные вершины корневого дерева и существует путь из ξ в η, то $\xi < \eta$. Если существует дуга из ξ в η, то η — *преемник* ξ. Вершины, не имеющие преемников, назовем *концевыми*.

Каждой системе векторного сложения $\mathscr{W}$ сопоставим корневое дерево $\mathscr{T}(\mathscr{W})$ и зададим правило сопоставления каждой вершине ξ некоторого r-мерного вектора $l(\xi)$, компоненты которого принадлежат $N \cup \{\omega\}$. Дерево $\mathscr{T}(\mathscr{W})$ и функция $l(\xi)$ определяются следующим образом.

1. Фиксируется корень и помечается d.

2. Пусть η — произвольная вершина дерева.

(a) Если для некоторой вершины ξ $(\xi < \eta)$ выполнено $l(\xi) = l(\eta)$, то η — концевая вершина;

(b) если для каждой вершины ξ $(\xi < \eta)$ выполнено $l(\xi) \neq l(\eta)$, то преемникам вершины η взаимно однозначно соответствуют элементы $w \in W$, такие, что $0 \leqslant l(\eta) + w$. Преемник вершины η, которому соответствует точка w, обозначим η_w; i-я компонента точки $l(\eta_w)$ определяется следующим образом:

(i) если существует $\xi \prec \eta_\omega$, такое, что $l(\xi) \leqslant l(\eta) + w$, для которого $l(\xi)_i < (l(\eta) + w)_i$, то полагаем $l(\eta_\omega)_i = \omega$;

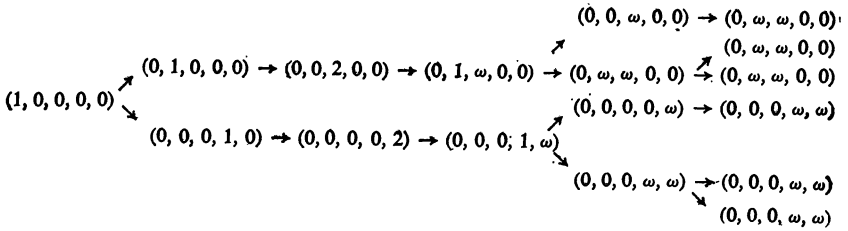
(ii) если не существует такого ξ, то полагаем $l(\eta_\omega)_i = (l(\eta) + w)_i$.

Пример 4.1. Для иллюстрации конструкции $\mathcal{T}(\mathcal{W})$ рассмотрим следующую систему векторного сложения $\mathcal{W} = (d, W)$:

$$d = (1, 0, 0, 0, 0)$$

$$W = \{(-1, 1, 0, 0, 0), \quad (-1, 0, 0, 1, 0), \quad (0, -1, 2, 0, 0),$$
$$(0, 1, -1, 0, 0), \quad (0, 0, 0, -1, 2), \quad (0, 0, 0, 1, -1)\}$$

$\mathcal{T}(\mathcal{W})$:



Теорема 4.1. *Для любой системы векторного сложения $\mathcal{W}$ дерево $\mathcal{T}(\mathcal{W})$ конечно.*

Доказательство этой теоремы использует следующие две леммы.

Лемма 4.1. *Пусть $p_0, p_1, \ldots, p_n, \ldots$ — бесконечная последовательность элементов из $(N \cup \{\omega\})^r$. Тогда существует бесконечная подпоследовательность $p_{i_1}, p_{i_2}, \ldots, p_{i_n}, \ldots$, такая, что $p_{i_1} \leqslant p_{i_2} \leqslant \ldots \leqslant p_{i_n} \leqslant \ldots$.*

Доказательство. Исходная последовательность содержит бесконечную подпоследовательность, неубывающую по первой координате, которая содержит бесконечную подпоследовательность, неубывающую по второй координате, и т. д.

Лемма 4.2. (Лемма Кёнига [6]). *Пусть T — корневое дерево, в котором каждая вершина имеет конечное число преемников и не существует бесконечного пути, исходящего из корня. Тогда дерево T конечно.*

Доказательство теоремы 4.1. Предположим (противное), что существует бесконечный путь, исходящий из корня, с последовательностью вершин $\delta, \eta_1, \eta_2, \ldots$. Тогда по лемме 4.1 существует бесконечная подпоследовательность $\eta_{i_1}, \eta_{i_2}, \ldots$ $\ldots, \eta_{i_n}, \ldots$, такая, что $l(\eta_{i_1}) \leqslant l(\eta_{i_2}) \leqslant \ldots \leqslant l(\eta_{i_n}) \leqslant \ldots$. Так

как ни одна из этих вершин не является конечной, то никогда не имеет места равенство $l(\eta_{i_n}) = l(\eta_{i_{n+1}})$ (иначе, согласно п. 2 (a) определения $\mathscr{T}(\mathscr{W})$, этот путь был бы конечен). Отсюда, согласно п. 2 (b), следует, что количество компонент, равных ω, у $\eta_{i_{n+1}}$ по крайней мере на одну больше, чем у η_{i_n}. Так как количество компонент конечно, то получили противоречие. Следовательно, предположение о существовании бесконечного пути не верно. Теперь из леммы 4.2 следует конечность $\mathscr{T}(\mathscr{W})$.

Отметим, что из индуктивного метода порождения дерева $\mathscr{T}(\mathscr{W})$ и конечности этого дерева следует эффективность его построения.

Следующая теорема играет ключевую роль для ряда проблем разрешимости, относящихся к системам векторного сложения.

Теорема 4.2. *Пусть задан произвольный r-мерный вектор x, компоненты которого — целые неотрицательные числа. Тогда эквивалентны следующие утверждения*:

(1) *существует* $y \in R(\mathscr{W})$, *такой, что* $x \leqslant y$;

(2) *существует вершина* $\eta \in \mathscr{T}(\mathscr{W})$, *такая, что* $x \leqslant l(\eta)$.

Доказательство. Докажем сначала (2) $\Rightarrow$ (1). Идея доказательства состоит в следующем. Если ξ — вершина дерева $\mathscr{T}(\mathscr{W})$, то существуют векторы из $R(\mathscr{W})$, последние координаты которых соответствуют $l(\xi)$. При этом можно добиться, чтобы как угодно много координат стали равными ω, посредством повторного использования последовательности векторов, ведущего к появлению символа ω. Предположим, что $x \leqslant l(\eta)$. Пусть путь от δ до η имеет последовательность вершин $\xi_0, \xi_1, \ldots, \xi_k$ ($\xi_0 = \delta$, $\xi_k = \eta$). Пусть вектор v_l ($l = 1, 2, \ldots, k$) соответствует дуге, направленной на ξ_l, т. е. $\xi_l = (\xi_{l-1})_{v_l}$. Не уменьшая общности предполагаем, что первые h компонент вектора $l(\xi)$ равны ω, а остальные компоненты меньше ω. При этом символ ω на пути от δ до ξ появляется сначала в первой, затем во второй координате и так далее вплоть до h. Тогда для каждого i ($1 \leqslant i \leqslant h$) существует последовательность векторов $t_i = v_{c(i)}, v_{c(i)+1}, v_{c(i)+2}, \ldots, v_{n(i)}$, таких, что вектор $u_i = v_{c(i)} + v_{c(i)+1} + \ldots + v_{d(i)}$ имеет положительную i-ю и неотрицательную $(i+1)$-ю координату (t_i — подпоследовательность для получения вектора с i компонентами, равными ω). Пусть $-\Delta$ есть нижняя граница всех (отрицательных) координат векторов $u_1, \ldots, u_h$, а $\{n_1, \ldots, n_h\}$ есть произвольное множество неотрицательных целых чисел, удовлетворяющих условию

$$n_i \geqslant (x-d)_i + \Delta(h+2-i+n_{i+1}+\ldots+n_h), \quad 1 \leqslant i \leqslant k. \qquad (1)$$

Существование такого множества следует из треугольной формы системы неравенств.

Построим последовательность $s_1, s_2, \ldots, s_{h+1}$, такую, что для каждого i $(1 \leqslant i \leqslant h)$ $s_1 s_2 \ldots s_i$ является префиксом последовательности $v_1 v_2 \ldots v_k$ вплоть до первого появления символа ω в i-й координате и выполняется $s_1 s_2 \ldots s_{h+1} = v_1 v_2 \ldots v_k$. Тогда последовательность

$$\sigma = s_1 t_1^{n_1} s_2 t_2^{n_2} \ldots s_h t_h^{n_h} s_{h+1} = u_1 u_2 \ldots u_f$$

обладает следующими свойствами:

(a) $d + u_1 + \ldots + u_f \geqslant x$,

(b) каждая частичная сумма $d + u_1 + \ldots + u_i$ неотрицательна.

Вывод свойств (a), (b) из системы неравенств (1) опускаем.

Продолжим доказательство. Предположим, что

$$d + u_1 + \ldots + u_f \in R(\mathscr{W}), \quad x \leqslant d + u_1 + \ldots + u_f,$$

и выполнено

$$d + u_1 + \ldots + u_m \geqslant 0, \quad m = 1, 2, \ldots, f,$$

где $u_m \in W$. Применим следующее преобразование к последовательности

$$d,\ d + u_1,\ d + u_1 + u_2,\ \ldots,\ d + u_1 + u_2 + \ldots + u_f$$

столько раз, сколько возможно:

(i) фиксируем первый такой элемент данной последовательности (обозначим его u'), что для некоторого предшествующего элемента u'' выполнено $u'' \leqslant u'$. При этом

(a) если $u'' = u'$, то вычеркнем из последовательности все элементы, следующие за u';

(b) в противном случае для каждого i, такого, что $(u'')_i < (u')_i$, заменим в u' и в каждом векторе, следующем за u' i-ю координату на символ ω.

В результате этого процесса получим последовательность меток вершин некоторого пути, исходящего из корня дерева $\mathscr{T}(\mathscr{W})$. Конечная метка этой последовательности представляет собой вектор, больший либо равный $d + u_1 + \ldots + u_f$. Отсюда следует выполнение п. (2) из формулировки теоремы.

Пример 4.2. Этот пример иллюстрирует построение последовательности σ из первой половины доказательства теоремы 4.2. Пусть $d = (1, 1, 1, 4)$, $W = \{(0, 0, 0, -1),\ (2, -1, 0, 0),\ (-1,$

1, 0, 0), (—1, —3, 4, 0)}. Рассмотрим следующий путь в дереве $\mathcal{T}(\mathcal{W})$:

$$(1, 1, 1, 4) \xrightarrow{0,0,0,-1} (1, 1, 1, 3) \xrightarrow{2,-1,0,0} (3, 0, 1, 3) \xrightarrow{-1,1,0,0}$$

$$\rightarrow (\omega, 1, 1, 3) \xrightarrow{-1,1,0,0} (\omega, \omega, 1, 3) \xrightarrow{-1,-3,4,0} (\omega, \omega, \omega, 3);$$

$s_1 = (0, 0, 0, -1)$, $(2, -1, 0, 0)$, $(-1, 1, 0, 0)$; $\quad t_1 = (2, -1, 0, 0)$, $(-1, 1, 0, 0)$,

$s_2 = (-1, 1, 0, 0)$; $\quad t_2 = (-1, 1, 0, 0)$;

$s_3 = (-1, -3, 4, 0)$; $\quad t_3 = (-1, -3, 4, 0)$.

Пусть $x = (22, 16, 9, 3) \leqslant (\omega, \omega, \omega, 3)$, $\Delta = 3$. Система неравенств в этом случае имеет следующий вид:

$$n_1 \geqslant 21 + 3 \cdot (4 + n_2 + n_3),$$
$$n_2 \geqslant 15 + 3 \cdot (3 + n_3),$$
$$n_3 \geqslant 8 + 3 \cdot 2.$$

Решение этой системы: $n_3 = 14$, $n_2 = 66$, $n_1 = 273$;

$\sigma = (0, 0, 0, -1)$, $(2, -1, 0, 0)$, $(-1, 1, 0, 0)$, $((2, -1, 0, 0)$, $(-1, 1, 0, 0))^{273}(-1, 1, 0, 0)(-1, 1, 0, 0)^{66}(-1, -3, 4, 0)(-1, -3, 4, 0)^{14}$.

Отсюда следует, что точка $(193, 23, 61, 3) \geqslant x$ принадлежит $R(\mathcal{W})$.

Приведем несколько следствий, полезных при построении алгоритмов распознавания некоторых свойств параллельных схем программ.

Следствие 4.1. *Существует алгоритм, который для произвольной системы векторного сложения $\mathcal{W}$ и точки x распознает, существует ли точка y $(y \geqslant x)$, принадлежащая $R(\mathcal{W})$.*

Следствие 4.2. *Существует алгоритм, который для произвольной r-мерной системы векторного сложения $\mathcal{W}$ и произвольного множества $\Theta \subseteq \{1, 2, \ldots r\}$ распознает, являются ли координаты из Θ совместно неограниченными, т. е. верно ли, что для каждого $x \in N^r$ существует такое $y \in R(\mathcal{W})$, что для каждого $i \in \Theta$ выполнено $y_i \geqslant x_i$.*

Доказательство. Это свойство выполняется в том и только в том случае, когда существует такая конечная вершина $\beta \in \mathcal{T}(\mathcal{W})$, что для каждого $i \in \Theta$ выполнено $(l(\beta))_i = \omega$.

Следствие 4.3. *Существует алгоритм распознавания бесконечности множества $R(\mathcal{W})$ для произвольной системы векторного сложения $\mathcal{W}$.*

Приведем пример неразрешимой проблемы, относящейся к множеству $R(\mathscr{W})$. Имеется в виду следующая неопубликованная теорема М. Рабина.

Теорема 4.3. *Не существует алгоритма, распознающего равенство сфер достижимости для произвольных двух систем векторного сложения* $\mathscr{W}$, $\mathscr{W}'$ (*т. е. равенство* $R(\mathscr{W}) = R(\mathscr{W}')$).

Рабин сводит к этой проблеме алгоритмически неразрешимую проблему существования решения экспоненциального диофантового уравнения.

B. *Алгоритмически разрешимые проблемы*

В этом разделе показана алгоритмическая разрешимость некоторых проблем для счетчиковых схем. При этом используются результаты, относящиеся к системам векторного сложения. Осуществляется «кодировка» каждой счетчиковой схемы некоторой системой векторного сложения.

Рассмотрим счетчиковую схему $\mathscr{S} = (M, A, \mathscr{T})$, описанную в определении 4.1. Сопоставим ей систему векторного сложения, имеющую $|S| + k + |A| + 2^{|A|}$ измерений. При определении системы $\mathscr{W}_{\mathscr{S}}$ каждый вектор удобно рассматривать как функцию, сопоставляющую каждой позиции вектора некоторое целое число. Позиция является индексом этого числа. В данном случае множество всех индексов равно $S \cup \{1, \ldots, k\} \cup A \cup 2^A$, т. е. координатами, которым присваиваются некоторые значения, являются элементы множества S, счетчики, операторы и подмножества множества операторов.

Для системы $\mathscr{W}_{\mathscr{S}} = (d, W)$ полагаем

$$
\begin{aligned}
&d(s_0) = 1,\\
&d(s) = 0, \qquad && s \in S, \quad s \neq s_0,\\
&d(i) = \pi_i, && i = 1, 2, \ldots, k,\\
&d(\varnothing) = 1,\\
&d(T) = 0, && T \subseteq A, \quad T \neq \varnothing.
\end{aligned}
$$

Элементам множества W взаимно однозначно сопоставлены тройки (s, σ, T), такие, что значение $\theta(s, \sigma)$ определено и $T = A$. Вектор w, которому сопоставлена тройка (s, σ, T), определяется следующим образом:

$$
\begin{aligned}
&w(t) = \delta(t, \theta(s, \sigma)) - \delta(t, s)\ ^{1}), && t \in S,\\
&w(i) = [v(\sigma)]_i, && i = 1, 2, \ldots, k.
\end{aligned}
$$

---

[1]) Для произвольных X, Y

$$
\delta(X, Y) = \begin{cases} 1, & \text{если } X = Y,\\ 0, & \text{если } X \neq Y. \end{cases}
$$

Если $\sigma = \bar{b} \in \Sigma_i$, то

$$w(a) = \delta(a, b), \quad a \in A;$$
$$w(U) = \delta(U, T \cup \{b\}) - \delta(U, T), \quad U \subseteq A.$$

Если $\sigma = b_j \in \Sigma_t$, то

$$w(a) = -\delta(a, b), \quad a \in A;$$
$$w(U) = \delta(U, T \cap \{a \mid R(b) \cap D(a) = \varnothing\}) - \delta(U, T), \quad U \subseteq A.$$

Для обоснования этих построений полезно ввести множество $p'(\mathscr{P}) \subseteq \Sigma^*$, состоящее из всех последовательностей, каждая из которых обладает свойствами 1, 2 из теоремы 3.1. Для свободной схемы $\mathscr{P}$ выполнено $p(\mathscr{P}) = p'(\mathscr{P})$; в общем случае $p(\mathscr{P}) \subseteq p'(\mathscr{P})$. Для произвольного слова $x \in p'(\mathscr{P})$ полагаем $\tau((s_0, \tau), x) = (s, p)$. Определим множество $T_x \subseteq A$ следующим образом: $a \in T_x$, если x можно представить в виде $x_1 \bar{a} x_2$, где слово x_2 не содержит символа $\bar{a}$, а также не содержит таких символов окончания b_j, что $R(b) \cap D(a) \neq \varnothing$. Таким образом, нарушение свободы происходит в результате выполнения оператора a и последовательности x в том и только в том случае, когда $a \in T_x$. Определим вектор w_x с множеством индексов (позиций) $S \cup \{1, 2, \ldots, k\} \cup A \cup 2^A$, полагая

$$w_x(t) = \delta(s, t), \qquad t \in S;$$
$$w_x(i) = \rho_i, \qquad i = 1, 2, \ldots, k;$$
$$w_x(a) = \Delta_a(x), \qquad a \in A;$$
$$w_x(U) = \delta(T_x, U), \qquad U \subseteq A.$$

Лемма 4.3. $R(\mathscr{W}_{\mathscr{P}}) = \{w_x \mid x \in p'(\mathscr{P})\}$.

Опускаем доказательство этой леммы. Оно имеет простой индуктивный характер. Лемма показывает, что каждый вектор, принадлежащий сфере достижимости $R(\mathscr{W}_{\mathscr{P}})$, является «кодом» информации об описании [1]), которое возникает в результате применения к схеме $\mathscr{P}$ гипотетической последовательности $x \in p'(\mathscr{P})$.

Лемма 4.4. *Для свободной схемы $\mathscr{P}$ выполнено*

$$R(\mathscr{W}_{\mathscr{P}}) = \{w_x \mid x \in p(\mathscr{P})\}.$$

Доказательство. Если $\mathscr{P}$ — свободная схема, то $p(\mathscr{P}) = p'(\mathscr{P})$.

---

[1]) Имеется в виду $\mathscr{J}$-описание при свободной интерпретации $\mathscr{J}$. — *Прим. перев.*

Теорема 4.4. *Существует алгоритм распознавания свободы произвольной счетчиковой схемы.*

Доказательство. Из определения множества $p'(\mathscr{S})$ и T_x следует, что схема $\mathscr{S}$ свободна тогда и только тогда, когда для каждого $x \in p'(\mathscr{S})$ выполнено

$$a \in T_x \Rightarrow x\bar{a} \notin p'(\mathscr{S}).$$

По лемме 4.3 это свойство равносильно следующему свойству системы $\mathscr{W}_{\mathscr{S}}$: для некоторых $a \in A$, $T \subseteq A$ $(a \in T)$, $s \in S$, таких, что значение $\theta(s, a)$ определено, существует вектор $u \in R(\mathscr{W}_{\mathscr{S}})$, удовлетворяющий следующим свойствам:

$$u(T) = 1,$$
$$u(s) = 1,$$
$$u(i) \geqslant [v(\bar{a})]_i, \quad i = 1, 2, \ldots, k.$$

Из леммы 4.3 следует, что для каждого $s \in S$ выполнено $u(s) \in \{0, 1\}$ и для каждого $T \subseteq A$ выполнено $u(T) \in \{0, 1\}$. Следовательно, вышеприведенная система эквивалентна следующей:

$$u(T) \geqslant 1,$$
$$u(s) \geqslant 1,$$
$$u(i) \geqslant [v(\bar{a})]_i, \quad i = 1, 2, \ldots, k.$$

Но из следствия 4.1 вытекает разрешимость проблемы существования такого вектора u среди конечного множества троек вида (a, T, s).

Для счетчиковых свободных схем можно упростить алгоритмы распознавания, используя систему векторного сложения $\mathscr{V}_{\mathscr{S}} = (e, V)$. Эта система получается из $\mathscr{W}_{\mathscr{S}} = (d, W)$, если вместо всех индексов вектора d и элементов из W рассматривать лишь множество $S \cup \{1, 2, \ldots, k\} \cup A$. При этом мы говорим, что $R(\mathscr{V}_{\mathscr{S}})$ является ограничением $R(\mathscr{W}_{\mathscr{S}})$ множеством индексов $S \cup \{1, 2, \ldots, k\} \cup A$.

Теорема 4.5. *Существует алгоритм для распознавания коммутативности произвольной счетчиковой схемы* $\mathscr{S}$.

Доказательство. Легко увидеть, что схема $\mathscr{S}$ не является коммутативной в том и только в том случае, когда для некоторой тройки $(s, \sigma, \sigma') \in S \times \Sigma \times \Sigma$, такой, что значения $\theta(s, \sigma, \sigma')$, $\theta(s, \sigma', \sigma)$ определены и не равны, существует такое слово x, что $\tau(q_0, x) = s$, $x\sigma\sigma' \in p(\mathscr{S})$ и $x\sigma'\sigma \in p(\mathscr{S})$. Используя лемму 4.4 и связь между $\mathscr{W}_{\mathscr{S}}$ и $\mathscr{V}_{\mathscr{S}}$, последнее условие можно переформулировать следующим образом:

для некоторой тройки $(s, \sigma, \sigma') \in S \times \Sigma \times \Sigma$, такой, что значения $\theta(s, \sigma\sigma')$, $\theta(s, \sigma'\sigma)$ определены и не равны, существует вектор $u \in R(\mathscr{V}_{\mathscr{S}})$, такой, что $u(s) = 1$, и для каждого i ($i = 1, 2, \ldots, k$) выполнено свойство

$$u(i) \geqslant \max[v(\sigma)_i,\ v(\sigma')_i,\ (v(\sigma) + v(\sigma'))_i].$$

Применяя следствие 4.1, легко показать, что это свойство алгоритмически разрешимо для заданного множества троек (s, σ, σ').

Методику использования следствия 4.1 в теоремах 4.4 и 4.5 не удается распространить на распознавание таких свойств счетчиковых свободных схем, как устойчивость и нейтральность. Однако сформулируем некоторые условия, достаточные для устойчивости и нейтральности счетчиковых схем и удобные в применении. Счетчиковая схема $\mathscr{S}$ (используем определение 4.1) является нейтральной, если для каждого $a \in A$ выполнено $v(\bar{a}) \leqslant 0$ и всякий раз, когда определено значение $\theta(s, \bar{a}, \bar{b})$, определено и $\theta(s, \bar{b})$. Счетчиковая схема $\mathscr{S}$ является устойчивой, если для произвольных различных операторов $a, b \in A$ выполнено $v(\bar{a}) < 0 \Rightarrow v(\bar{b})_i = 0$ и для произвольных различных элементов $\sigma, \pi \in \Sigma$ из определенности значений $\theta(s, \sigma)$, $\theta(s, \pi)$ следует определенность $\theta(s, \sigma\pi)$.

Дальнейшие положительные результаты можно получить, используя систему $\mathscr{V}_{\mathscr{S}}$.

Теорема 4.6. *Существует алгоритм для распознавания ограниченности*[1]) *произвольной счетчиковой свободной схемы.*

Доказательство. Из леммы 4.4 следует, что $\mathscr{S}$ ограничена тогда и только тогда, когда для каждого $a \in A$ ограничено множество $\{u(a) \mid u \in R(\mathscr{V}_{\mathscr{S}})\}$. Последнее свойство, согласно следствию 4.2, является разрешимым.

Аналогичным образом можно получить более глубокий результат, состоящий в разрешимости следующей проблемы: для произвольного a распознать, является ли множество $\{\Delta_a(x) \mid x \in p(\mathscr{P})\}$ конечным.

Следствие 4.4. *Существует алгоритм для распознавания последовательности произвольной счетчиковой свободной схемы.*

Определение 4.2. Для произвольной схемы $\mathscr{S}$ оператор a назовем *замкнутым* (*terminating*), если в каждое вычисление схемы $\mathscr{S}$ оператор a входит конечное число раз.

---

[1]) Следуя Д. Слутцу, отметим, что произвольная свободная коммутативная нейтральная устойчивая счетчиковая схема является ограниченной.

Теорема 4.7. *Существует алгоритм для распознавания замкнутости произвольного оператора $a \in A$ произвольной счетчиковой свободной схемы $\mathscr{S}$. Для замкнутого оператора эффективно находится максимальное количество вхождений оператора a в произвольное вычисление схемы $\mathscr{S}$.*

Доказательство. Присоединяя к системе $\mathscr{V}_{\mathscr{S}}$ координату, в которой накапливается количество символов включений оператора a, получим некоторую систему $\mathscr{V}'_{\mathscr{S}}$. При этом оператор замкнут в том и только в том случае, когда множество значений этой координаты ограничено. Из следствия 4.2 вытекает разрешимость последнего свойства, а максимальное значение координаты, если оно конечно, может быть вычислено по дереву $\mathscr{T}(\mathscr{V}'_{\mathscr{S}})$.

Определение 4.2 допускает ситуацию, при которой оператор счетчиковой свободной схемы является замкнутым, но не существует константы, которая являлась бы верхней границей числа вхождений символа $\bar{a}$ в вычисления. Однако, используя системы векторного сложения, можно показать невозможность такой ситуации.

Теорема 4.8. *Существует алгоритм для распознавания конечности произвольной счетчиковой свободной схемы $\mathscr{S}$ (т. е. конечности множества достижимых состояний схемы $\mathscr{S}$).*

Доказательство. Схема $\mathscr{S}$ конечна в том и только в том случае, когда каждый счетчик ограничен. В силу следствия 4.2, последняя проблема является разрешимой.

Следующий результат является одним из главных результатов настоящей работы.

Теорема 4.9. *Существует алгоритм для распознавания однозначности произвольной свободной результативной устойчивой коммутативной счетчиковой схемы.*

Доказательство. Принимая следствие 2.1 при условии свободы схемы, получим, что $\mathscr{S}$ однозначна в том и только в том случае, когда не существует такого $x \in \Sigma^*$, что для некоторых $\sigma, \sigma' \in \Sigma$ выполнено $x\sigma\sigma' \in p(\mathscr{S})$ и $x\sigma'\sigma \in p(\mathscr{S})$. Последнее свойство алгоритмически разрешимо. При этом используются система $\mathscr{V}_{\mathscr{S}}$ и метод доказательства теоремы 4.5.

Алгоритмы распознавания, построенные при доказательстве теорем 4.5—4.9, опираются на предположение о свободе рассматриваемых схем. Вместо свободы здесь можно было бы взять следующее свойство. Счетчиковую схему $\mathscr{S}$ назовем *строго ограниченной*, если в системе $\mathscr{W}_{\mathscr{S}}$ ограничены компоненты с индексами $1, 2, \ldots, k$, соответствующие k счетчикам. Это свойство

допускает следующую интерпретацию. Пусть схема $\tilde{\mathscr{P}}$ получена из $\mathscr{P}$ таким образом, что множество входных и выходных ячеек всех операторов равны и не пусты. Тогда схема $\tilde{\mathscr{P}}$ является свободной, $p(\tilde{\mathscr{P}}) = p'(\mathscr{P})$ и $\mathscr{W}_{\mathscr{P}} = \mathscr{W}_{\tilde{\mathscr{P}}}$. Применяя лемму 4.4 к $\tilde{\mathscr{P}}$, получаем, что схема $\tilde{\mathscr{P}}$ ограничена в том и только в том случае, когда для векторов из $\mathscr{W}_{\tilde{\mathscr{P}}}$ ограничены компоненты с индексами 1, 2, ..., k, т. е. в том и только в том случае, когда схема $\mathscr{P}$ строго ограничена. Отсюда следует, что проблема распознавания строгой ограниченности произвольной счетчиковой схемы $\mathscr{P}$ алгоритмически разрешима. При этом эффективно вычислимы границы значений счетчиков.

Проблемы распознавания свойств несвободных счетчиковых схем, по-видимому, в большинстве случаев неразрешимы, поскольку приходится учитывать повторение значений выходных ячеек операторов. Однако для строго ограниченных схем эта информация может быть закодирована конечным множеством компонент системы векторного сложения, поскольку ограничено количество вхождений операторов в вычислении. Не вдаваясь в подробности, констатируем, что свойство сильной ограниченности может замещать свойство свободы в следствии 4.4 и в теоремах 4.5, 4.7, 4.8 и 4.9.

Приведем еще одну разрешимую проблему для счетчиковых схем.

Теорема 4.10. *Существует алгоритм, преобразующий произвольную однозначную счетчиковую схему $\mathscr{P}$ в некоторую эквивалентную последовательную схему $\mathscr{P}'$. При этом для свободной схемы $\mathscr{P}$ схема $\mathscr{P}'$ свободна.*

Доказательство. Рассмотрим множество S из определения 4.1. Пусть $A = (a^{(1)}, \ldots, a^{(h)})$. Чтобы получить $\mathscr{P}'$ из $\mathscr{P}$, заменим S, s_0, θ на S', s_0', θ' соответственно, полагая

$$S' = S \times \{1, 2, \ldots, h\} \times \{0, 1\}, \quad s_0' = (s_0, 1, 0),$$

значение $\theta'(\langle s, \alpha, \beta\rangle, \bar{a}^{(i)})$ определено в том и только в том случае, когда выполнены следующие условия:

(i) значение $\theta(s, \bar{a}^{(i)})$ определено,

(ii) $\beta = 0$,

(iii) для каждого

$$k \in \{\alpha,\ \alpha + 1 \pmod h,\ \alpha + 2 \pmod h,\ \ldots,\ (i-1) \pmod h\}$$

значение $\theta(q, \bar{a}^{(k)})$ не определено. Если значение $\theta'(\langle s, \alpha, \beta\rangle, \bar{a}^{(i)})$ определено, то полагаем его равным $\theta(s, \bar{a}^{(i)}, \alpha + 1 \pmod h, 1)$. Наконец, полагаем

$$\theta'(s, \alpha, \beta, a_j^{(i)}) = \langle \theta(s, a_j^{(i)}), \alpha, 0\rangle.$$

Компонента β обеспечивает чередование включений и выключений операторов, так что схема $\mathscr{S}'$ является последовательной. Координата α в результате циклического пересчета гарантирует

(1) возможность включения хотя бы одного оператора на соответствующем шаге;

(2) если определены значения $\tau(s_0, x\bar{a}^{(i)})$, $\theta(s_0, x)$, то управление схемы $\mathscr{S}'$ разрешит включение оператора $\bar{a}^{(i)}$, следующего за x, не более чем через $h-1$ включений других операторов.

Из предыдущего следует, что для каждой интерпретации $\mathscr{J}$ схема $\mathscr{S}'$ имеет точно одно $\mathscr{J}$-вычисление. Более того, это $\mathscr{J}$-вычисление является также $\mathscr{J}$-вычислением схемы $\mathscr{S}$ (в частности, цикличность пересчета параметра α гарантирует выполнение свойства конечной задержки).

C. *Неразрешимые проблемы*

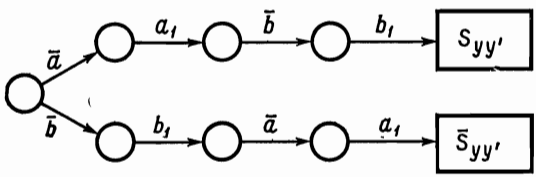
В этом разделе доказывается алгоритмическая неразрешимость проблемы эквивалентности для конечных схем. Эта проблема неразрешима и для устойчивых и для последовательных конечных схем. Поскольку класс счетчиковых схем содержит класс конечных схем, то для счетчиковых схем проблема эквивалентности также неразрешима. Разрешимость проблемы эквивалентности доказана лишь для схем без логических разветвлений, которые содержатся в классе счетчиковых схем. Этот результат обсуждается в разд. 6.

Доказательство неразрешимости эквивалентности использует неоднозначность рассматриваемых схем. Вопрос о разрешимости эквивалентности для конечных однозначных схем остается открытым [1]). Вопрос об алгоритмической разрешимости свойства однозначности произвольной конечной схемы также является открытым [2]), хотя теорема 4.9 и решает его для некоторого важного подкласса.

---

[1]) Неразрешимость проблемы распознавания эквивалентности по истории ячеек доказана в работе [11], которая добавлена переводчиком к списку литературы. — *Прим. перев.*

[2]) Неразрешимость этой проблемы можно показать, исходя из [11]. Легко подобрать операторы таким образом, чтобы схема вида



где Syy', $\bar{S}yy'$ — схемы из [11], была неоднозначной в том и только в том случае, когда она имеет хотя бы одну конечную реализацию. — *Прим. перев.*

При доказательстве неразрешимости эквивалентности будет использована неразрешимость проблемы парасочетаний Поста [8]. Аналогичное сведение провели Рабин и Скотт [9] в применении к двуленточному автомату. Проблема парасочетаний формулируется следующим образом. Для произвольных двух последовательностей слов

$$x_1, \ldots, x_n,$$
$$y_1, \ldots, y_n$$

над некоторым алфавитом Г распознать, существует ли последовательность индексов $i_1, i_2, \ldots, i_p$, такая, что выполнено равенство слов

$$x_{i_1}x_{i_2} \ldots x_{i_p} = y_{i_1}y_{i_2} \ldots y_{i_p}.$$

Эта проблема неразрешима для любого алфавита Г, содержащего более одной буквы, т. е. не существует общего алгоритма для распознавания этого свойства.

Метод сведения проблемы парасочетаний над алфавитом $\{b_1, b_2\}$ к проблеме эквивалентности схем мы проиллюстрируем на примерах, поскольку формальное доказательство легко восстанавливается.

Пусть для схемы $\mathscr{S}$

$$M = \{1, 2\}, \quad A = \{a, b\}, \quad D(a) = R(a) = \{1\}, \quad K(a) = 3,$$
$$D(b) = R(b) = \{2\}, \quad K(b) = 3.$$

Заметим, что ни один из операторов здесь не воздействует на ячейки другого оператора. Отсюда следует, что последовательность символов выключений оператора a зависит лишь от интерпретации $\mathscr{J}$, но не зависит от подпоследовательности включений и выключений оператора b; k-й элемент этой последовательности символов выключений оператора a имеет вид $G_a(F_a^{[k-1]}(\Pi_1(c_0)))$. Аналогичное рассуждение можно провести для оператора b.

Пусть $X = x_1, \ldots, x_n$ — последовательность из слов над алфавитом $\{b_1, b_2\}$; $i_1, \ldots, i_p$ — последовательность индексов. Интерпретацию $\mathscr{J}$ назовем *совместимой* с парой $(X; i_1, \ldots, i_p)$, если выполнены следующие условия:

(i) если оператор a в процессе $\mathscr{J}$-вычисления выполняется более одного раза, то проекция $\mathscr{J}$-вычисления на символы выключения оператора a имеет вид

$$a_1^{i_1-1}a_2a_1^{i_2-1}a_2 \ldots a_1^{i_p-1}a_2a_3,$$

(ii) если оператор b в процессе $\mathcal{J}$-вычисления выполняется более одного раза, то проекция $\mathcal{J}$-вычисления на символы выключения оператора b имеет вид

$$x_{i_1}x_{i_2}\ldots x_{i_p}b_3.$$

Пусть $Y = y_1, y_2, \ldots, y_n$ — вторая последовательность слов над алфавитом $\{b_1, b_2\}$ из проблемы парасочетаний. Тогда легко увидеть, что пара X, Y имеет решение (т. е. существует соответствующая последовательность индексов) в том и только в том случае, когда существует интерпретация $\mathcal{J}$, совместимая как с $(X; i_1, i_2, \ldots, i_p)$, так и с $(Y; i_1, i_2, \ldots, i_p)$.

Эта последняя формулировка является промежуточным этапом при переходе от проблемы парасочетаний к проблеме эквивалентности конечных схем.

Рассмотрим однозначные схемы $\mathcal{P}(X)$, $\overline{\mathcal{P}}(X)$, $\mathcal{P}(Y)$, $\overline{\mathcal{P}}(Y)$, каждая из которых имеет $M = \{1, 2\}$, $A = \{a, b\}$, $D(a) = R(a) = \{1\}$, $K(a) = 3$, $D(b) = R(b) = \{2\}$, $K(b) = 3$. Если $\mathcal{J}$ совместима с $(X; i_1, i_2, \ldots, i_p)$, то $\mathcal{P}(X)$ — единственное конечное $\mathcal{J}$-вычисление, для которого выполнены следующие свойства:

(i) последовательность символов выключения оператора a имеет вид

$$a_1^{i_1-1}a_2a_1^{i_2-1}a_2\ldots a_1^{i_p-1}a_2a_3,$$

(ii) последовательность символов выключения оператора b имеет вид

$$x_{i_1}x_{i_2}\ldots x_{i_p}b_3.$$

Если интерпретация $\mathcal{J}$ не совместима ни с одной парой вида $(X; i_1, i_2, \ldots, i_p)$, то каждое $\mathcal{J}$-вычисление содержит бесконечное множество вхождений символов $\bar{a}$, $\bar{b}$.

При работе схемы $\mathcal{P}(X)$ сначала некоторое количество раз выполняется оператор a и вырабатывается последовательность символов выключений $a_1^{i-1}a_2$. Затем некоторое количество раз выполняется оператор b и вырабатывается последовательность символов выключения x_i. Любое отклонение от последовательностей такого вида перед первым появлением символа a_3 переводит схему в бесконечное движение по циклу. Если после символа a_3 символом выключения оператора b является не b_3, то также происходит зацикливание.

Следующий пример иллюстрирует построение $\mathcal{P}(X)$.

Пример 4.3. Пусть $n = 3$, $x_1 = b_1$, $x_2 = b_2$, $x_3 = b_1b_1$. На рис. 4.1 изображено управление схемы $\mathcal{P}(X)$. Схема $\overline{\mathcal{P}}(X)$ обладает следующими свойствами:

(a) если интерпретация $\mathcal{J}$ совместима с $(X; i_1, i_2, \ldots, i_p)$, то каждое $\mathcal{J}$-вычисление схемы $\overline{\mathcal{P}}(X)$ содержит бесконечное множество вхождений как символа $\bar{a}$, так и $\bar{b}$. Единственное конечное $\overline{\mathcal{J}}$-вычисление схемы $\mathcal{P}(X)$ является префиксом каждого из этих вычислений;

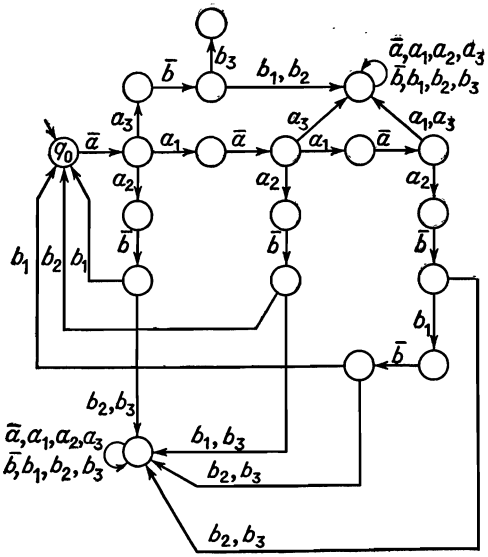


Рис. 4.1.

(b) если не существует последовательности индексов $i_1, i_2, \ldots, i_p$, такой, что $\mathcal{J}$ совместима с $(X; i_1, \ldots, i_p)$, то в любом $\mathcal{J}$-вычислении либо

(i) последовательность символов выключения оператора a бесконечна и не содержит a_3, либо

(ii) последовательность оканчивается первым вхождением символа a_3.

Аналогично для оператора b: либо последовательность символов выключения оператора b бесконечна и не содержит b_3, либо эта последовательность оканчивается первым вхождением символа b_3.

Пусть последовательности слов $X = x_1, x_2, \ldots, x_n$, $Y = y_1, y_2, \ldots, y_n$ и операторы c, d таковы, что выполнено $(R(c) \cap \cap R(d)) \cup (R(c) \cap D(d)) \cup (D(c) \cap R(d)) = \varnothing$. Рассмотрим две устойчивые схемы, изображенные на рис. 4.2 и 4.3.

Легко увидеть, что соответствующие системы $\mathscr{S}$, $\mathscr{S}''$ не эквивалентны в том и только в том случае, когда для некоторой интерпретации $\mathscr{J}$ схема $\mathscr{P}(X)$ имеет конечное $\mathscr{J}$-вычисление, а $\overline{\mathscr{P}}(X)$ не имеет. Последнее возможно в том и только в том случае, когда для некоторой последовательности $i_1, \ldots, i_p$ выполнено

$$x_{i_1}x_{i_2} \ldots x_{i_p} = y_{i_1}y_{i_2} \ldots y_{i_p}.$$

Таким образом, каждая единичная задача проблемы парасочетаний Поста над алфавитом $\{b_1, b_2\}$ сводится к вопросу о эквивалентности двух устойчивых конечных схем. Следовательно, не существует алгоритма для распознавания эквивалентности схем

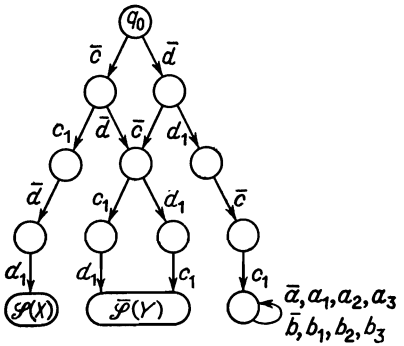


Рис. 4.2.

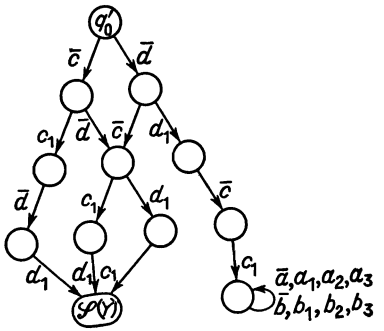


Рис. 4.3.

этого класса, так как из существования такого алгоритма следовала бы разрешимость проблемы парасочетаний.

На рис. 4.4 изображена модификация рассмотренных конструкций, посредством которой аналогичным образом устанавливается неразрешимость эквивалентности для последовательных (но не устойчивых) конечных схем. В этом примере множества $D(c)$, $D(d)$, $D(e)$, $R(c)$, $R(d)$, $R(e)$ попарно не пересекаются.

Полученные результаты сформулируем в виде следующих теорем.

Теорема 4.11. *Проблема распознавания эквивалентности устойчивых конечных схем алгоритмически неразрешима.*

Теорема 4.12. *Проблема распознавания эквивалентности последовательных конечных схем алгоритмически неразрешима.*

Поскольку приведенные выше конструкции используют неоднозначные схемы, то этот метод доказательства не распро-

страняется на эквивалентность однозначных схем и для них вопрос остается открытым. В работе [7] доказана неразрешимость проблемы эквивалентности в классе схем программ, близком к классу конечных однозначных последовательных схем. В этой

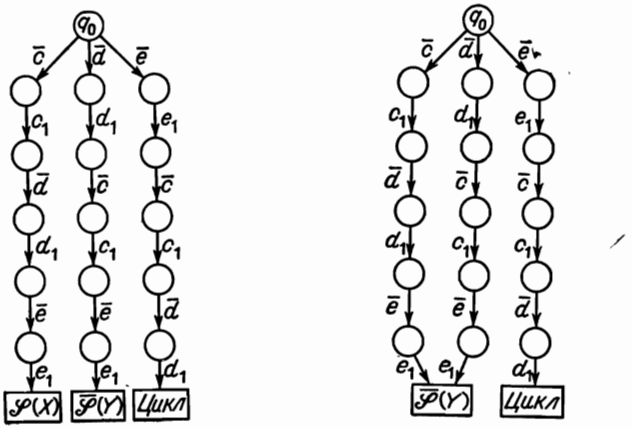


Рис. 4.4.

работе рассматриваются наряду с операторами последовательности операторов (инструкции), и различные операторы могут содержать одинаковые функциональные символы, что в наших терминах можно выразить как ограничение множества интерпретаций. Однако перенести доказательства на наш случай не удается.

5. ПАРАЛЛЕЛЬНЫЕ ОПЕРАТОРНЫЕ СХЕМЫ

В этом разделе будет рассмотрен некоторый частный случай счетчиковых схем — так называемые параллельные операторные схемы. Мы введем удобное графическое представление этих схем, а затем обсудим их адекватность параллельным алгоритмам.

Определение 5.1. *Параллельной операторной схемой* назовем произвольную счетчиковую схему, обладающую следующими свойствами:

(1) $S = \{s_0\}$;

(2) для каждого $\sigma \in \Sigma$ значение $\theta(s_0, \sigma)$ определено;

(3) если σ — символ выключения, то каждая компонента вектора $v(\sigma)$ равна 0 или 1;

(4) если σ — символ включения, то каждая компонента вектора $v(\sigma)$ равна 0 или —1;

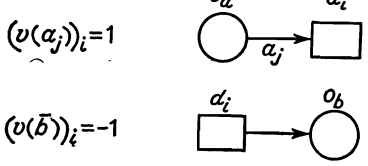
(5) для любых двух неравных символов включения σ, σ' из $v(\sigma)_i = -1$ следует $v(\sigma')_i = 0$.

Теорема 5.1. *Произвольная операторная схема является устойчивой, коммутативной и нейтральной.*

Доказательство использует коммутативность векторного сложения и достаточные условия устойчивости и перестановочности счетчиковых схем, установленные в разд. 4.

Операторную схему $\mathscr{S}$ можно представить в виде ориентированного графа $G(\mathscr{S})$ *с операторными вершинами* $o_a, o_b, o_c, \ldots$ и *вершинами-счетчиками* $d_1, d_2, \ldots, d_k$. Каждый счетчик d_i помечен числом π_i, которое является начальным значением этого счетчика. Дуги графа направлены либо от операторных вершин к счетчикам, либо от счетчиков к операторным вершинам. Каждая дуга, исходящая из операторной вершины o_a, помечена некоторым символом выключения a_j. Дуги сопоставлены ненулевым компонентам векторов $v(\sigma)$.

Если $(v(a_j))_i = 1$, то выключение оператора a с символом выключения a_j увеличивает значение счетчика i на 1. Если $(v(\bar{a}))_i = -1$, то включение оператора b уменьшает значение счетчика i на 1. Условие (5) из определения 5.1 означает, что из произвольного счетчика выходит не более чем одна дуга[1]).



Пример 5.1. На рис. 5.1 изображена некоторая интерпретированная операторная схема (изображение не формальное), задающая алгоритм нахождения первого положительного числа в последовательности $x_1, x_2, \ldots, x_m$. Результат алгоритма помещается в ячейку z. Происходит одновременное исследование подпоследовательностей $x_1, x_3, x_5, \ldots$ и $x_2, x_4, x_6, \ldots$. Исследование подпоследовательности оканчивается лишь в том случае, когда либо найден первый положительный элемент, либо когда исследованы все элементы, индексы которых меньше индекса первого положительного элемента другой последовательности.

Сделаем несколько замечаний, полезных при построении строгого формального определения интерпретированной операторной схемы рис. 5.1.

[1]) Однако дуги, исходящие из различных счетчиков, могут быть направлены на одну и ту же операторную вершину. — *Прим. перев.*

(1) Незаполненные операторные вершины соответствуют операторам с пустыми входными и выходными множествами.

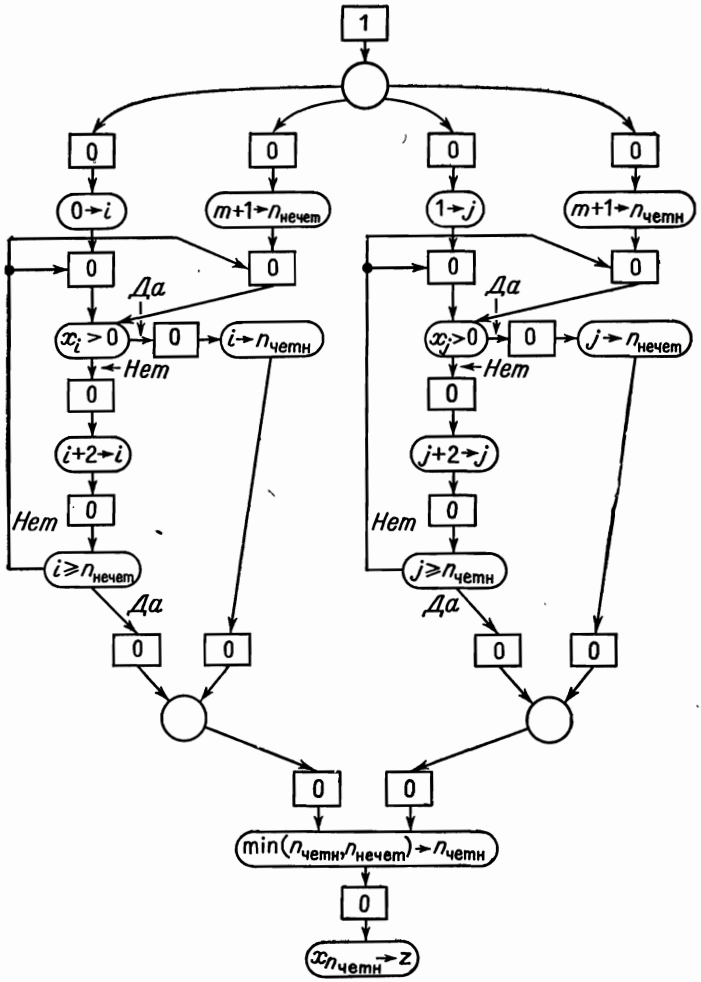


Рис. 5.1.

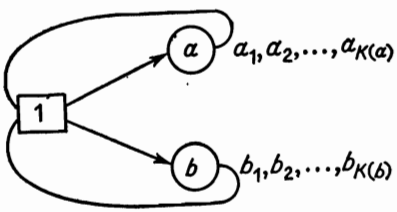
(2) Оператор, помеченный «$x_i > 0$», имеет множество $\{\mathbf{x}, i\}$ входных ячеек, где $\mathbf{x}$ представляет собой последовательность $x_1, x_2, \ldots, x_m$. Этот искусственный момент необходим, поскольку в данной модели множество входных ячеек оператора фиксируется заранее и не поддается изменению посредством перевычисления индексов.

Некоторые средства задания параллельности в языках программирования имеют аналоги в операторных схемах. Например, в результате выполнения команды *fork* происходит вызов двух или более операторов. В операторной схеме это достигается наличием в векторе $v(a_j)$ более одной компоненты, имеющей значение 1, т. е. наличием символа выключения, увеличивающего значения нескольких счетчиков.

В результате выполнения команды *join* происходит вызов некоторого оператора после завершения работы прежде включенных операторов в зависимости от результатов их выполнения. Это соответствует наличию более одной компоненты, равной —1, в некотором векторе $v(a)$, т. е. на операторную вершину o_a направлены дуги из нескольких счетчиков.

В результате выполнения команды *quit* происходит окончание работы некоторой последовательности операторов, которое может не совпадать с окончанием работы всего алгоритма. Это соответствует наличию символа выключения, не увеличивающего значения счетчиков.

Некоторые способы задания параллельности не могут быть представлены в параллельных операторных схемах. Рассмотрим, например, ситуацию, при которой два оператора a и b одновременно имеют доступ к включению, но не могут выполняться одновременно. Операторные схемы не могут полностью удовлетворить это требование, не нарушая условия (5) из ее определения (требование устойчивости). Если же нарушить это условие и разрешить счетчику «питать» несколько операторов, то возникает возможность для создания искомой ситуации. Это иллюстрирует следующий рисунок:



Хотя такое обобщение операторной схемы является естественным, в настоящей работе оно не исследуется.

Приведем еще один пример параллельного процесса: операторы a, b выполняются параллельно и осуществляют переход к выполнению одного из четырех операторов c, d, e, f. Операторная схема, отчасти соответствующая этому алгоритму, изображена на рис. 5.2.

Например, пара (a_1, b_1) осуществляет переход к оператору c. Однако если первыми двумя парами символов выключения

будут (a_1, b_1), (a_2, b_2), то в дальнейшем включатся не только операторы c, e, но и d, f. Это происходит в результате появления «избыточных» единиц в счетчиках 3 и 8, сохраняющихся вплоть до включения операторов d, e.

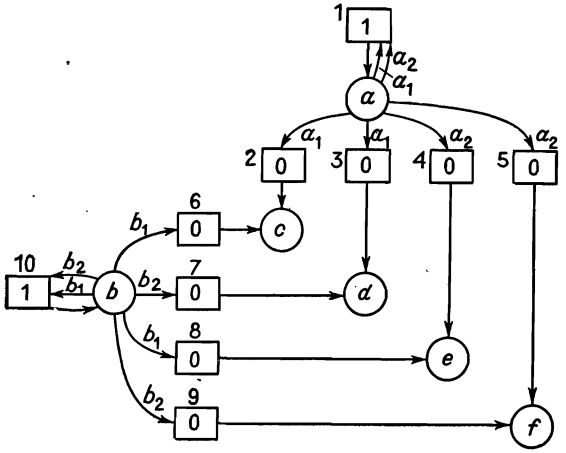


Рис. 5.2.

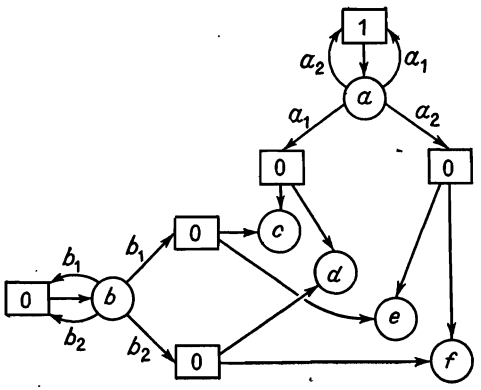


Рис. 5.3.

Указанные трудности не являются случайными, они заложены в самом определении операторной схемы. Действительно, легко показать, что для произвольной операторной схемы из определенности $\tau(q, a_1 b_2 \bar{d})$ следует определенность $\tau(q, a_1 x b_2 \bar{d})$ при $\bar{d} \notin x$. Эту трудность, как и предыдущую, можно преодо-

леть посредством ослабления п. (5) в определении 5.1 — нужно разрешить счетчику «питать» несколько операторных вершин. На рис. 5.3 изображена схема, использующая это ослабление и реализующая указанный алгоритм.

6. СХЕМЫ БЕЗ ЛОГИЧЕСКИХ РАЗВЕТВЛЕНИЙ

Схему $\mathscr{S}$ назовем *схемой без логических разветвлений*, или сокращенно *простой* схемой, если каждый ее оператор имеет точно один символ выключения, т. е. для каждого $a \in A$ выполнено $K(a) = 1$. Несмотря на ограниченность, этот класс охватывает некоторые итеративные параллельные вычисления, записанные на языке так называемых вычислительных графов [3]. Ограниченность класса простых схем по сравнению с классом счетчиковых схем позволяет получить для него более полный спектр разрешимых проблем и более простые алгоритмы распознавания. В этом разделе будет показано, что для простых схем число выполнений операторов не зависит от интерпретации и одинаково для всех вычислений. Будет построен алгоритм нахождения этого числа для произвольного оператора простой схемы. Кроме того, будут построены алгоритмы распознавания свободы и однозначности простых схем, а также алгоритм распознавания эквивалентности свободных однозначных простых схем.

Для произвольной простой схемы $\mathscr{S}$ каждое слово $x \in \Sigma^*$, удовлетворяющее свойствам 1, 2 из теоремы 3.1, принадлежит множеству $p(\mathscr{S})$. Свойство 3 для простых схем выполняется автоматически. При анализе свободных схем мы не пользовались свойством 3 из теоремы 3.1. Отсюда следует, что методы исследования свободных схем применимы к простым схемам.

А. *Вычисление* $\gamma_{\mathscr{S}}(a)$

Следующие лемма и теорема устанавливают, что для простых схем число выполнений операторов в вычислении не зависит ни от интерпретации, ни от вычисления.

Лемма 6.1. *Для произвольных интерпретаций $\mathscr{J}$, $\mathscr{J}'$ простой схемы $\mathscr{S}$ слово x является $\mathscr{J}$-вычислением тогда и только тогда, когда оно является $\mathscr{J}'$-вычислением.*

Доказательство. Из определения 1.5 следует, что слово y является префиксом $\mathscr{J}$-вычисления в том и только в том случае, когда оно является префиксом $\mathscr{J}'$-вычисления. Отсюда следует, что условия из определения 1.6, необходимые и достаточные для того, чтобы слово x являлось $\mathscr{J}$-вычислением, также не зависят от интерпретации.

Теорема 6.1. *Пусть $\mathscr{S}$ — произвольная устойчивая коммутативная простая схема. Тогда для каждого оператора $a \in A$ любые два вычисления схемы $\mathscr{S}$ содержат одинаковое число вхождений символа $\bar{a}$.*

Доказательство. Пусть x, y — произвольные вычисления схемы $\mathscr{S}$. Применяя индукцию по n, можно показать, что для любого n существует такое вычисление $z(n)$, для которого выполнены условия:

(i) для каждого $a \in A$ вычисления $z(n)$, y имеют равные количества вхождений символа $\bar{a}$ (это справедливо и для символа a_1);

(ii) ${}_n z(n) = {}_n(x)$.

Индукция проводится по значениям того же самого «следящего» аргумента, который использовался в доказательстве теоремы 2.1. Здесь мы доказательство опускаем.

Из условия (i) следует, что $z(n)$ и y имеют одинаковое число вхождений символа $\bar{a}$, а из (ii) следует, что для каждого $n \leqslant l(x)$ выполнено ${}_n z(n) = {}_n(x)$. Отсюда следует искомое.

Количество вхождений символа включения $\bar{a}$ в произвольное вычисление схемы $\mathscr{S}$ обозначим через $\gamma_{\mathscr{S}}(a)$. В теореме 4.7 построен алгоритм нахождения этого числа для произвольной свободной схемы. В этом разделе мы дадим более эффективный метод вычисления $\gamma_{\mathscr{S}}(a)$ для простых операторных схем.

Вычисление значений $\gamma_{\mathscr{S}}(a)$ осуществляется за три шага. На первом шаге используется итеративный метод нахождения таких операторов a, для которых $\gamma_{\mathscr{S}}(a) = 0$. На втором шаге находятся те операторы a, для которых $0 < \gamma_{\mathscr{S}}(a) < \infty$. При этом используются некоторые свойства подграфов графа $G(\mathscr{S})$ схемы $\mathscr{S}$. На третьем шаге строится и решается система уравнений для вычисления таких значений $\gamma_{\mathscr{S}}(a)$, что $0 < \gamma_{\mathscr{S}}(a) < \infty$.

Для определения множества $\{a \mid \gamma_{\mathscr{S}}(a) = 0\}$ рассмотрим следующую итеративную схему:

$$T_0 = \varnothing,$$
$$S_0 = \{i \mid (q_0)_i > 0\},$$
$$T_{j+1} = T_j \cup \{a \mid (d_i, o_a) \in G(\mathscr{S}) \Rightarrow i \in S_j\},$$
$$S_{j+1} = S_j \cup \{i \mid \exists a \in T_{j+1} : (o_a, d_i) \in G(\mathscr{S})\}.$$

В этой итеративной схеме S_0 — множество счетчиков, начальные значения которых положительны; T_1 — множество операторов, «питающихся» лишь счетчиками из S_0 (каждый из этих операторов может быть включен на первом шаге); S_j — множество счетчиков, которые могут позднее стать положитель-

ными; T_{j+1} — множество операторов, которые могут позднее включиться благодаря счетчикам из S_j. Итеративный процесс оканчивается, когда для некоторого положительного k выполнено $T_k = T_{k+1}$. Легко увидеть, что такое k существует, так как множество A конечно. При этом $T_k = \{a \mid \gamma_{\mathscr{S}}(a) > 0\}$, поскольку элементы из T_k являются как раз операторами, «питающимися» теми счетчиками, которые могут принимать положительные значения. Отсюда следует

$$A \cap \overline{T}_k = \{a \mid \gamma_{\mathscr{S}}(a) = 0\}.$$

Следующая теорема полезна для построения алгоритма получения множества $\{a \mid \gamma_{\mathscr{S}}(a) < \infty\}$.

Теорема 6.2. *Пусть $\mathscr{S}$ — простая операторная схема. Тогда оператор a замкнут в том и только в том случае, когда вершина o_a принадлежит некоторому подграфу G' графа $G(\mathscr{S})$, обладающему следующими свойствами:*

(i) *если вершина o_b принадлежит G', то G' содержит дугу, направленную на o_b в том и только в том случае, когда $\gamma_{\mathscr{S}}(b) \neq 0$;*

(ii) *для произвольной вершины d_i графа G' каждая дуга графа $G(\mathscr{S})$, направленная на d_i, принадлежит G';*

(iii) *G' не содержит циклов.*

Доказательство. Определим некоторый подграф H графа $G(\mathscr{S})$, который для каждого замкнутого оператора a содержит вершину o_a, и покажем, что H удовлетворяет свойствам (i)—(iii). Пусть слово x является вычислением схемы $\mathscr{S}$, число k выбрано так, что слово ${}_kx$ содержит символ выключения каждого замкнутого оператора. Подграф H определим следующим образом:

(a) вершина o_a принадлежит H в том и только в том случае, когда оператор a замкнут;

(b) вершина-счетчик d_i принадлежит H в том и только в том случае, когда

(1) каждый оператор, «питающий» счетчик i, замкнут;

(2) существует оператор b, такой, что d_i питает b, b замкнут и $\gamma_{\mathscr{S}}(b) > 0$;

(3) i-я компонента вектора-состояния $\tau(q_0, {}_kx)$ равна 0, т. е. значение счетчика d_i равно 0 в результате выполнения ${}_kx$, что препятствует дальнейшему выполнению соответствующего оператора;

(c) если некоторые две вершины принадлежат H, то и дуга графа $G(\mathscr{S})$ между этими вершинами принадлежит H.

Докажем, что H удовлетворяет свойствам (i)—(iii). Из условия (b) (2) для H следует, что если для вершины o_a выпол-

нено $\gamma_{\mathscr{S}}(a) = 0$, то не существует счетчика d_i, питающего o_a в графе $G(\mathscr{S})$. Отсюда следует, что операторы a, для которых $\gamma_{\mathscr{S}}(a) = 0$, удовлетворяют условию (i). Пусть b — замкнутый оператор, для которого выполнено $o_b \in H$, $\gamma_{\mathscr{S}}(b) > 0$. Если ни один счетчик из H не питает o_b, то каждый счетчик из $G(\mathscr{S})$, питающий o_b, либо имеет положительное значение после выполнения ${}_k x$, либо его значение увеличивает некоторый незамкнутый оператор. Однако последнее предположение влечет возможность нового включения и выключения оператора b, что противоречит свойству числа k относительно ${}_k x$. Отсюда следует, что H удовлетворяет условию (i).

В силу (b) (1), вершина d_i принадлежит H только в том случае, когда каждый оператор, питающий счетчик i, является замкнутым. Поскольку для каждого замкнутого оператора b вершина o_b принадлежит H и каждая дуга между вершинами из H принадлежит H, то для H выполняется свойство (ii).

Покажем, что для H выполнено свойство (iii). Предположим (противное), что H содержит некоторый цикл C. В цикле вместе с каждой вершиной содержится дуга, направленная на эту вершину. При этом если $o_a \in H$ и $\gamma_{\mathscr{S}}(a) = 0$, то на вершину o_a не направлена ни одна дуга. Следовательно, для каждой операторной вершины b цикла C выполнено $\gamma_{\mathscr{S}}(b) > 0$. Рассмотрим операторную вершину b цикла C, такую, что оператор b оканчивает выполнение последним в последовательности ${}_k x$. В цикле C содержится некоторая дуга (o_b, d_i). Теперь из простоты схемы $\mathscr{S}$ следует, что i-я компонента вектора $\tau(q_0, {}_k x)$ больше нуля. Однако это противоречит условию (b) (3) для H. Следовательно, H не содержит циклов.

Продолжим доказательство. Пусть G' подграф графа $G(\mathscr{S})$, удовлетворяющий условиям (i) — (iii). Покажем, что для каждой вершины $o_a \in G'$ выполнено $\gamma_{\mathscr{S}}(a) < \infty$. Поскольку G' не содержит циклов, существует вершина $v \in G'$, на которую не направлена ни одна дуга. Если $v = o_a$, то из условия (i) следует $\gamma_{\mathscr{S}}(a) = 0$. Если $v = d_i$, то из условия (ii) следует, что дуги из $G(\mathscr{S})$ не направлены на d_i и значения счетчика d_i могут лишь уменьшаться. Следовательно, если d_i питает оператор a, то $\gamma_{\mathscr{S}}(a) < \infty$. Пусть $k(c)$—длина самого длинного пути подграфа G', оканчивающегося вершиной o_c из G'. Индукцией по $k(c)$ можно показать, что для каждой вершины $o_c \in G'$ выполнено $\gamma_{\mathscr{S}}(c) < \infty$. Это следует из того, что если $(d_i, o_c) \in G'$ и для всех операторов b, для которых $(o_b, d_i) \in G'$, выполнено $\gamma_{\mathscr{S}}(b) < \infty$, то выполнено $\gamma_{\mathscr{S}}(c) < \infty$.

Определение подграфа H в доказательстве теоремы 6.2 показывает, что замкнутыми вершинами являются вершины, достижимые из счетчиков без входных дуг и из операторных вершин o_a, для которых $\gamma_{\mathscr{S}}(a) = 0$. Множество замкнутых вершин можно получить из следующей итеративной схемы:

$U_0 = \{a \mid \gamma_{\mathscr{S}}(a) = 0\}$,

$U_{l+1} = U_l \cup \{a \mid \exists d_i,$ такое, что из $(d_i, o_a) \in G(\mathscr{S})$ и $(o_b, d_i) \in G(\mathscr{S})$ следует $b \in U_l\}$.

Итеративный процесс оканчивается, если для некоторых k выполнено $U_{k+1} = U_k$. В этом случае $U = U_k = \{a \mid \gamma_{\mathscr{S}}(a) < \infty\}$. Если множество замкнутых вершин известно, то множество $\{\gamma_{\mathscr{S}}(a) \mid 0 < \gamma_{\mathscr{S}}(a) < \infty\}$ можно получить несколькими способами. Можно формировать префикс y произвольного вычисления до тех пор, пока выполняются условия для подграфа H из доказательства теоремы 6.2, а затем найти значение $\gamma_{\mathscr{S}}(a)$ на слове y для каждого замкнутого оператора. Заметим, что в общем случае H нельзя получить непосредственно из U, поскольку счетчики, «препятствующие» дальнейшим включениям операторов, не известны априори. Наиболее удобным способом вычисления величин $\gamma_{\mathscr{S}}(a)$ является решение следующей системы уравнений.

Пусть $S = \{i \mid (o_b, d_i) \in G(\mathscr{S}) \Rightarrow b \in U\}$. Тогда для произвольного $a \in U$, такого, что $\gamma_{\mathscr{S}}(a) \neq 0$, выполнено

$$\gamma_{\mathscr{S}}(a) = \min_{\{i \mid (d_i, o_a) \in G(\mathscr{S}),\ i \in S\}} \left[(q_0)_i + \sum_{(o_b, d_i) \in G(\mathscr{S})} \gamma_{\mathscr{S}}(b)\right].$$

Для быстрого решения этой системы можно использовать доказательство следующей теоремы.

Теорема 6.3. *Пусть для системы уравнений*

$$x_i = \min_{1 \leqslant k \leqslant N_i} \left(a_i^k + \sum_j b_{ij}^k x_j\right) \quad i = 1, 2, \ldots, n \qquad (*)$$

выполнены свойства

(i) *существует решение этой системы,*

(ii) *для всех* i, k *выполнено* $a_i^k \geqslant 0$,

(iii) b_{ij}^k *равно* 1 *либо* 0.

(iv) *не существует последовательности* $(i_0, k_0), (i_1, k_1), \ldots, (i_r, k_r)$, *такой, что, во-первых,*

$$b_{i_l i_{l+1}}^{k_l} = \begin{cases} 1, & \text{если } j = i_{l+1} \pmod r, \\ 0 & \text{в противном случае,} \end{cases}$$

во-вторых, $a_{i_l}^{k_l} = 0$. *Тогда для этой системы выполнены следующие свойства:*

(a) *система имеет единственное решение*;
(b) *существует хотя бы одна пара* (i, k), *такая, что*

$$\forall_j (b_{ij}^k = 0); \qquad (**)$$

(c) *если пара* (i^*, k^*) *удовлетворяет свойству* (**) *и выполнено* $a_{i^*}^{k^*} \leqslant a_i^k$ *при условии, что* (i, k) *удовлетворяет* **, *тогда* $x_{i^*} = a_{i^*}^{k^*}$.

Доказательство. Пусть y_i $(i = 1, 2, \ldots, n)$ является решением системы (*). Тогда для каждого i существует такое k_i, что выполнено

$$y_i = a_i^{k_i} + \sum_j b_{ij}^k y_j.$$

Пусть $S = \{i \mid y_i = \min_{j=1, 2, \ldots, n} (y_i)\}$. Тогда для $i \in S$ либо выполнено $b_{ij}^{k_i} = 0$ для всех j, так что уравнение для y_i имеет вид

$$y_i = a_i^{k_i},$$

либо $a_i^{k_i} = 0$ и $b_{ij}^{k_i} = 1$ для единственного $j \in S$. Это следует из минимальности y_i и условий (ii), (iii). Однако выполнение последней альтернативы для каждого $i \in S$ противоречило бы условию (iv). Отсюда следует выполнение условия (b). Для некоторого $i' \in S$ выполнено $y_{i'} = a_{i'}^{k_{i'}}$. Возьмем i^* из (c). Тогда $y_{i^*} \geqslant y_{i'} = a_{i'}^{k_{i'}} \geqslant a_{i^*}^k$. Однако из уравнений следует $y_{i^*} \leqslant a_{i^*}^{k^*}$. Последнее доказывает $y_{i^*} = a_{i^*}^{k^*}$, откуда следует (c). Заменив в системе (*) $x_{i'}$ на $a_{i'}^{k_{i'}}$, получим приведенную систему уравнений, удовлетворяющую свойствам (i) — (iv). Из приведенной системы, удовлетворяющей свойству (c), можно получить следующую компоненту решения. Повторяя этот процесс до получения полного решения, придем к очевидному заключению о единственности этого решения. Отсюда следует (a), что и завершает доказательство.

Следующий пример иллюстрирует методику вычисления $\gamma_{\mathscr{P}}(a)$ для каждого оператора a схемы $\mathscr{P}$.

Пример 6.1. На рис. 6.1 изображен граф $G(\mathscr{P})$ некоторой операторной схемы $\mathscr{P}$.

Итеративный процесс при получении множества $\{a \mid \gamma_{\mathscr{P}}(a) = 0\}$ протекает следующим образом:

$T_0 = \varnothing$;

$S_0 = \{i \mid (q_0)_i > 0\} = \{1, 2, 3, 4, 7, 9\}$;

$T_1 = T_0 \cup \{a \mid (d_i, o_a) \in G(\mathscr{P}) \Rightarrow i \in S_j\} = \{a, b, d, f\}$;

$S_1 = S_0 \cup \{i \mid \exists a \in T_1, \quad (o_a, d_i) \in G(\mathscr{P}) = \{1, 2, 3, 4, 5, 6, 7, 9, 10\}$;

$T_2 = \{a, b, c, d, f, g\}$;
$S_2 = S_1$; $T_3 = T_2$;
... ; $\{a \mid \gamma_{\mathscr{S}}(a) = 0\} = \{e\}$.

Далее порождается множество замкнутых вершин:
$U_0 = \{e\}$;
$U_1 = U_0 \cup \{a \mid (d_i, o_a) \in G(\mathscr{P}) \wedge (o_b, d_i) \in G(\mathscr{P}) \Rightarrow b \in U_0\} = \{e, a\}$;
$U_2 = \{a, e, b\}$;
$U_3 = \{e, a, b, c\}$;
$U_4 = \{a, b, c, e, f\} = U_5$.

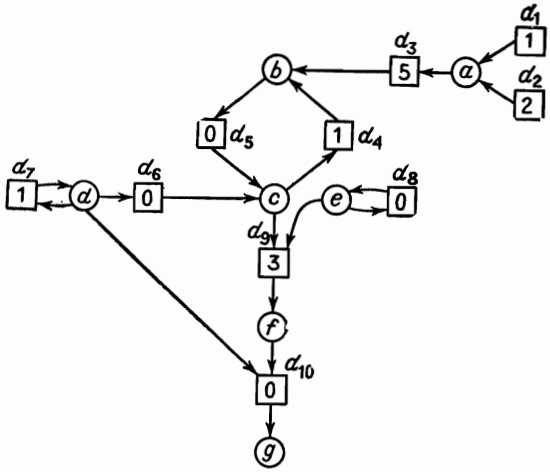


Рис. 6·1.

Получили множество замкнутых вершин $U = \{a, b, c, e, f\}$. При этом $S = \{1, 2, 3, 4, 5, 8, 9\}$. Система уравнений для нахождения $\gamma_{\mathscr{S}}(a)$ $(a \in U)$ имеет вид

$$\gamma_{\mathscr{S}}(a) = \min[1, 2];$$
$$\gamma_{\mathscr{S}}(b) = \min[5 + \gamma_{\mathscr{S}}(a),\ 1 + \gamma_{\mathscr{S}}(c)];$$
$$\gamma_{\mathscr{S}}(c) = \min[\gamma_{\mathscr{S}}(b),\ \gamma_{\mathscr{S}}(a)];$$
$$\gamma_{\mathscr{S}}(f) = \min[3 + \gamma_{\mathscr{S}}(c)].$$

Решая эту систему, получим

$$\gamma_{\mathscr{S}}(a) = 1, \quad \gamma_{\mathscr{S}}(b) = 6, \quad \gamma_{\mathscr{S}}(c) = 6, \quad \gamma_{\mathscr{S}}(f) = 9,$$
$$\gamma_{\mathscr{S}}(e) = 0.$$

В. *Разрешимые свойства простых операторных схем*

Очевидно, что для простых схем выполняется свойство 3 из теоремы 3.1. Однако в общем случае свойство свободы необходимо лишь для выполнения свойства 3 и для применимости методов, относящихся к t-счетчиковым передаточным системам (transition systems). Более того, для некоторых теорем, относящихся к простым схемам, гипотеза о результативности может быть опущена, поскольку для этих схем невозможны условные передачи, не оставляющие следов в памяти. Отсюда легко следует, что некоторые результаты разд. 4 могут быть развиты для простых схем.

Следующая теорема дает некоторую модификацию этих результатов.

Теорема 6.4. *Для произвольной простой счетчиковой схемы* $\mathscr{S}$ *разрешимы следующие проблемы*:

(1) *Конечно ли множество* $\{\Delta_a(x) \mid x \in p(\mathscr{S})\}$? *Если конечно, то чему равен максимальный элемент этого множества*?

(2) *Является ли схема* $\mathscr{S}$ *последовательной*?

(3) *Существует ли верхняя граница значений данного счетчика*?

(4) *Является ли данная свободная устойчивая коммутативная схема однозначной*?

В настоящем разделе будут построены алгоритмы распознавания свободы, однозначности простых схем, алгоритм распознавания эквивалентности свободных однозначных простых схем. Эти алгоритмы будут использовать некоторую «характеристическую последовательность» операторной схемы.

Лемма 6.2. *Пусть* $\mathscr{S}$ *— свободная простая операторная схема. Тогда* $\mathscr{S}$ *однозначна в том и только в том случае, когда для каждой пары операторов* (a, b), *таких, что* $a\rho b$, $R(a) \neq \varnothing$, $R(b) \neq \varnothing$ *и для произвольных двух вычислений* x, y *выполнено*

$$\mathscr{E}_{\bar{a}\bar{b}}(x) = \mathscr{E}_{\bar{a}\bar{b}}(y).$$

Эта лемма является ослаблением следствия 2.2 и имеет аналогичное доказательство.

В лемме 6.2 утверждается инвариантность (равенство) проекций на множество $\{\bar{a}, \bar{b}\}$ для различных вычислений схемы при $a\rho b$, что является важным структурным свойством простых операторных схем. Леммы 6.3, 6.4 вместе с теоремой 6.5 и ее следствиями дают более детальный анализ этой инвариантности. Установлено, что если проекция $\mathscr{E}_{\bar{a}\bar{b}}(x)$ является инвариантом, то она имеет некоторую «чередующуюся» структуру (теорема 6.5), и что проекция $\mathscr{E}_{\bar{a}\bar{b}}(x)$ является инвариантом в том и

только в том случае, когда первые ее четыре элемента не зависят от вычисления (следствие 6.2).

Лемма 6.3. *Предположим, что для слова* $z \in p(\mathscr{P})$, *где* $\mathscr{P}$ — *простая операторная схема, содержащая операторы* a, b, *выполнены условия*

$$\bar{b} \in z, \quad \exists v\colon\ zv\bar{a} \in p(\mathscr{P}),$$

$$zi\bar{a} \in p(\mathscr{P}) \Rightarrow \bar{b} \in u$$

(*т. е. а нельзя включить раньше* b). *Тогда* $G(\mathscr{P})$ *содержит некоторый подграф* G', *обладающий следующими свойствами*:

(1) *Подграф* G' *не содержит циклов, каждая его вершина принадлежит некоторому пути, направленному на* o_a.

(2) *Если* $o_c \in G'$ ($c = b$ *либо* $\gamma_{\mathscr{P}}(c) = 0$), *то ни одна дуга из* G' *не направлена на* o_c; *в противном случае на* o_c *направлена точно одна дуга.*

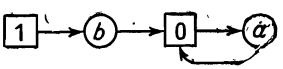


Рис. 6.2.

(3) *Если* $d_i \in G'$, *то каждая дуга графа* $G(\mathscr{P})$, *направленная на* d_i, *принадлежит* G'.

(4) *Если* $d_i \in G'$, *то* $[\tau(q_0, z)]_i = 0$; *если* $o_c \in G_i$, *то* $\Delta_c(z) = 0$.

Интуитивно содержание этой леммы сводится к тому, что при данных предположениях распространение произвольного сигнала включения вплоть до o_a осуществляется на пути, содержащем o_b. Доказательство этой леммы опущено. Оно аналогично доказательству теоремы 6.2. Отметим, что предположение $\bar{b} \in z$ здесь существенно. Например, если для операторной схемы рис. 6.2 в качестве z взять пустую последовательность, то выполняется $zi\bar{a} \in p(\mathscr{P}) \Rightarrow \bar{b} \in u$, но не существует граф G' с нужными свойствами.

Количество вхождений символа $\bar{a}$ в последовательность $w \in p(\mathscr{P})$ обозначим $\gamma_w(a)$. Отметим следующее утверждение, вытекающее из леммы 6.3.

Лемма 6.4. *При предположениях леммы* 6.3 *существуют неотрицательные целые* α, β, *такие, что, во-первых,*

$$\gamma_z(a) = a \cdot \gamma_z(b) + \beta$$

и, во-вторых, для каждого $w \in p(\mathscr{P})$

$$\gamma_w(a) \leqslant a \cdot \gamma_w(b) + \beta.$$

Доказательство. Для каждой дуги $(d_i, o_c) \in G'$ и для каждого слова $w \in p(\mathscr{S})$ *выполнено неравенство*

$$\gamma_w(c) \leqslant (q_0)_i + \sum_{\{d \mid (o_c,\, d_i) \in G(\mathscr{S})\}} \gamma_w(d),$$

переходящее в равенство при $z = w$. Неравенства, соответствующие внутренним вершинам G', можно исключить, поскольку G' не содержит циклов. Полагая $\gamma_w(c) = 0$ для каждого c, такого, что $\gamma_{\mathscr{S}}(c) = 0$, получим искомое линейное неравенство (линейное уравнение в случае $w = z$).

Заметим, что для операторной схемы рис. 6.2 не выполняется ни одно соотношение вида $\gamma_w(a) \leqslant \alpha \cdot \gamma_w(b) + \beta$, даже при условии, что a не может включиться раньше b. Однако это не приводит к противоречию, поскольку множество $p(\mathscr{S})$ не содержит слова z, для которого выполняются все предположения леммы 6.4.

Теорема 6.5. *Пусть $\mathscr{S}$ — простая операторная схема, a, b — операторы, для которых $\gamma_{\mathscr{S}}(a) > 0$, $\gamma_{\mathscr{S}}(b) > 0$, для любых двух вычислений x, y выполнено $\mathscr{E}_{\bar{a}\bar{b}}(x) = \mathscr{E}_{\bar{a}\bar{b}}(y)$. Тогда $\mathscr{E}_{\bar{a}\bar{b}}(x)$ обладает следующими свойствами:*

(i) *если $\gamma_{\mathscr{S}}(a) > 1$ и $\gamma_{\mathscr{S}}(b) > 1$, то $|\gamma_{\mathscr{S}}(a) - \gamma_{\mathscr{S}}(b)| \leqslant 1$ и вхождения символов $\bar{a}$, $\bar{b}$ чередуются;*

(ii) *если $\gamma_{\mathscr{S}}(a) = 1$, то либо первый, либо второй элемент последовательности $\mathscr{E}_{\bar{a}\bar{b}}$ равен $\bar{a}$.*

Доказательство. Предположим, что $r = \min(2, \gamma_{\mathscr{S}}(a)) + \min(2, \gamma_{\mathscr{S}}(b))$ и $\mathscr{E}_{\bar{a}\bar{b}}$ начинаются последовательностью $s_1 s_2 \ldots s_r$.

Без потери общности достаточно рассмотреть случай, когда $s_1 = \bar{a}$. Применяя несколько раз лемму 6.4, находим все варианты последовательности $s_1 s_2 \ldots s_r$:

$$s_1 s_2 = \bar{a}\bar{a} \Rightarrow \gamma_{\mathscr{S}}(b) = 0,$$

$$s_1 s_2 s_3 = \bar{a}\bar{b}\bar{b} \Rightarrow \gamma_{\mathscr{S}}(a) = 1,$$

$$s_1 s_2 s_3 s_4 = \bar{a}\bar{b}\bar{a}\bar{a} \Rightarrow \gamma_{\mathscr{S}}(b) = 1,$$

$$s_1 s_2 s_3 s_4 = \bar{a}\bar{b}\bar{a}\bar{b} \Rightarrow \forall w \gamma_w(b) \leqslant \gamma_w(a)$$
$$\text{и либо } \gamma_w(a) \leqslant \gamma_w(b) + 1,$$
$$\text{либо } \gamma_{\mathscr{S}}(a) = 2.$$

Свойства (i), (ii) непосредственно следуют из соответствующих импликаций. Для иллюстрации вывода этих импликаций рассмотрим случай $s_1 s_2 s_3 s_4 = \bar{a}\bar{b}\bar{a}\bar{b}$. Здесь существует z, удовле-

творяющее предположениям леммы 6.4, такое, что $\gamma_z(a) = 2$, $\gamma_z(b) = 1$. Таким образом, существуют неотрицательные числа α, β, такие, что $2 = \alpha \cdot 1 + \beta$, и для каждого $w \in p(\mathscr{S})$ выполнено $\gamma_w(a) \leqslant \alpha \cdot \gamma_w(b) + \beta$. Допустимые варианты: $\alpha = 0$, $\beta = 2$; $\alpha = 1$; $\beta = 1$; $\alpha = 2$, $\beta = 0$; последний вариант не возможен, поскольку $s_1 = \bar{a}$.

В следствии 6.1 будет показано, что доказательство теоремы 6.5 проходит и при более слабых предположениях.

Определение 6.1. Для произвольных $w \in p(\mathscr{S})$, $a \in A$ введем обозначение $S(w, a) = \{b \mid b \in \Sigma, \exists u: wu\bar{b} \in p(\mathscr{S}), \bar{a} \notin u\}$. Таким образом, $S(w, a)$ — множество операторов, которые могут включиться после слова w перед включением оператора a.

Следствие 6.1. *Пусть слово* $v \in p(\mathscr{S})$ *содержит* $\min(2, \gamma_{\mathscr{S}}(a))$ *вхождений символа* $\bar{a}$ *и* $\min(2, \gamma_{\mathscr{S}}(b))$ *вхождений* $\bar{b}$; z *удовлетворяет следующему условию: если* $z = z_1\sigma z_2$, *где* $\sigma \in \{\bar{a}, \bar{b}\}$, *то либо* $a \notin S(z_1, \sigma, b)$, *либо* $b \notin S(z_1\sigma, a)$. *Тогда для любых двух вычислений* x, y *выполнено* $\mathscr{E}_{\bar{a}\bar{b}}(x) = \mathscr{E}_{\bar{a}\bar{b}}(y)$.

Доказательство этого следствия почти совпадает с доказательством теоремы 6.5.

Более простой результат, составляющий основное содержание этого следствия, состоит в следующем.

Следствие 6.2. *Эквивалентны следующие два утверждения*:

(i) *для любых двух вычислений* x, y *выполнено*

$$\mathscr{E}_{\bar{a}\bar{b}}(x) = \mathscr{E}_{\bar{a}\bar{b}}(y),$$

(ii) *для любых двух вычислений* x, y *выполнено*

$$_r(\mathscr{E}_{\bar{a}\bar{b}}(x)) = {}_r(\mathscr{E}_{\bar{a}\bar{b}}(y)),$$

где $r = \min(2, \gamma_{\mathscr{S}}(a)) + \min(2, \gamma_{\mathscr{S}}(b))$.

Теорема 6.5 накладывает сильные ограничения на структуру проекции $\mathscr{E}_{\bar{a}\bar{b}}(x)$ в случае, когда она не зависит от x. На основании этого результата можно установить критерий свободы и однозначности произвольной простой операторной схемы $\mathscr{S}$.

Теорема 6.6. *Произвольная простая операторная схема* $\mathscr{S}$ *является свободной и однозначной в том и только в том случае, когда для нее выполняются следующие условия*:

(i) *из* $a\rho b$, $R(a) \neq \varnothing$, $R(b) \neq \varnothing$ *следует, что для произвольных вычислений* x, y *выполнено* $\mathscr{E}_{\bar{a}\bar{b}}(x) = \mathscr{E}_{\bar{a}\bar{b}}(y)$;

(ii) *из* $D(a) \cap R(a) \neq \varnothing$ *следует, что в каждом вычислении вхождение символов* $\bar{a}$, a_1 *чередуются*;

(iii) *из* $\gamma_{\mathscr{S}}(a) \geqslant 2$, $D(a) \cap R(a) = \varnothing$ *следует, что для некоторого* b, *такого, что* $R(b) \cap D(a) \neq \varnothing$, *вхождения символов* $\bar{a}$, $\bar{b}$ *чередуются в любом вычислении.*

Доказательство. Пусть выполнены условия (i), (ii), (iii). Предположим, что схема $\mathscr{S}$ не свободна. Тогда для некоторого a существует последовательность $v\bar{a}w\bar{a} \in p(\mathscr{S})$, такая, что $c_1 \in w \Rightarrow R(c) \cap D(a) = \varnothing$. Если выполнено $R(a) \cap D(a) = \varnothing$, то при $c = a$ получим противоречие с (ii); в противном случае из (iii) вытекает, что w можно представить в виде $w_1\bar{b}w_2$, где $R(b) \cap D(a) \neq \varnothing$, но $b_1 \in w_2$. Отсюда следует, что слово $v\bar{a}w_1$, $\bar{b}w_2\bar{a}b_1$ принадлежит $p(\mathscr{S})$. Но, в силу устойчивости и нейтральности, выполнено $v\bar{a}w_1w_2\bar{a}\bar{b} \in p(\mathscr{S})$, что противоречит условию (iii). Отсюда следует свобода схемы $\mathscr{S}$. Из леммы 6.2 следует однозначность схемы $\mathscr{S}$.

Пусть схема $\mathscr{S}$ свободна и однозначна. Из леммы 6.2 следует выполнение свойства (i). Из предположения о невыполнении (ii) следует неоднозначность схемы. Предположим, что не выполняется свойство (iii). Тогда для каждого b, такого, что $R(b) \cap D(a) \neq \varnothing$, выполнено $\gamma_{\mathscr{S}}(b) = 1$; при этом для каждого такого b либо $\gamma_{\mathscr{S}}(a) = 2$ и $\mathscr{E}_{\bar{a}\bar{b}} = \bar{b}\bar{a}\bar{a}$, либо $\gamma_{\mathscr{S}}(a) > 2$ и $\mathscr{E}_{\bar{a}\bar{b}}$ содержит одну из следующих последовательностей символов включений:

$$\bar{b}\bar{a}\bar{a}\bar{a}, \quad \bar{a}\bar{b}\bar{a}\bar{a}.$$

В первой из этих двух последовательностей первое и второе выполнения оператора используют одни и те же входные данные, во второй последовательности второе и третье выполнения оператора используют одни и те же входные данные. В обоих случаях нарушается свобода схемы, т. е. приходим к противоречию.

Теорема 6.6 и следствие 6.2 дают возможность построить очень просто алгоритм для распознавания как свободы, так и однозначности произвольной простой операторной схемы $\mathscr{S}$.

Лемма 6.5. *Существует алгоритм построения для каждой простой операторной схемы* $\mathscr{S}$ *такого слова* $v_{\mathscr{S}} \in p(\mathscr{S})$, *которое для каждого оператора* a *содержит точно* $\min(2, \gamma_{\mathscr{S}}(a))$ *вхождений символа* $\bar{a}$ *и такое же количество вхождений символа* a_1.

Доказательство. Пусть схема $\mathscr{S}'$ совпадает с $\mathscr{S}$ с точностью до некоторых дополнительных счетчиков, питающих операторы (каждому оператору — свой счетчик). Полагаем, что каждый новый счетчик имеет начальное значение 2, значения новых счетчиков никогда не увеличиваются. Тогда для произвольного оператора a выполнено $\gamma_{\mathscr{S}'}(a) = \min(2, \gamma_{\mathscr{S}}(a))$. Это сле-

дует из того, что в простой операторной схеме третье включение одного из операторов не является необходимым для второго включения другого оператора (см. теорему 6.5). Произвольное вычисление схемы $\mathscr{S}'$ является префиксом некоторого вычисления схемы $\mathscr{S}$, следовательно, его можно взять в качестве $v_{\mathscr{S}}$.

Слово $v_{\mathscr{S}}$ назовем *характеристической последовательностью* схемы $\mathscr{S}$. Его длина не более чем в 4 раза превышает количество операторов схемы $\mathscr{S}$. Это понятие играет большую роль при разработке эффективных алгоритмов распознавания свойств параллельных схем. Во-первых, как показывает следующая теорема, оно полезно при распознавании свободы и однозначности схем. Во-вторых, характеристическая последовательность и величины $\gamma_{\mathscr{S}}(a)$ могут быть основой для алгоритмов распознавания эквивалентности свободных однозначных простых операторных схем.

Теорема 6.7. *Пусть $\mathscr{S}$ — простая операторная схема, для которой последовательность $v_{\mathscr{S}}$ обладает свойствами, сформулированными в лемме* 6.5. *Тогда $\mathscr{S}$ является одновременно свободной и однозначной в том и только в том случае, когда выполняются следующие свойства*:

(a) *если $w\bar{a}$ есть префикс слова $v_{\mathscr{S}}$ и выполнено $a\rho b$, то $\bar{b} \notin S(w, \bar{a})$*;

(b) *если $D(a) \cap R(a) \neq \varnothing$ и слово $w = u\bar{a}$ является префиксом слова $v_{\mathscr{S}}$, то $\bar{a} \notin S(w, a_1)$*;

(c) *если $\gamma_{\mathscr{S}}(a) \geqslant 2$ и $D(a) \cap R(a) = \varnothing$, то для некоторого оператора b, такого, что $R(b) \cap D(a) \neq \varnothing$, вхождения символов $\bar{a}$, $\bar{b}$ чередуются в последовательности $v_{\mathscr{S}}$.*

Доказательство. Если схема $\mathscr{S}$ свободна и однозначна, то из теоремы 6.6, в силу $v_{\mathscr{S}} \in p(\mathscr{S})$, непосредственно следуют свойства (a), (b), (c). Предположим теперь, что выполнены свойства (a), (b), (c). Докажем выполнение свойств (i), (ii), (iii) из теоремы 6.6. Из следствия 6.1 вытекает $(a) \Rightarrow$ (i). Пусть b — оператор из условия (c). Тогда проекция $\mathscr{E}_{\bar{a}\bar{b}}$ постоянна для всех вычислений. Из теоремы 6.5 следует, что чередование символов $\bar{a}$, $\bar{b}$ в $v_{\mathscr{S}}$ влечет их чередование $\mathscr{E}_{\bar{a}\bar{b}}$ для произвольного вычисления x. Аналогичным образом можно показать, что (b) $\Rightarrow$ (ii).

Алгоритм распознавания свободы и однозначности схемы $\mathscr{S}$ состоит в построении последовательности $v_{\mathscr{S}}$ и проверке для нее условий из предыдущей теоремы. Эта проверка использует следующую лемму.

Лемма 6.6. *$S(w, a)$ определяется следующей итеративной схемой:*

$$T_0 = \varnothing;$$
$$S_0 = \{i \mid (\tau(q_0, w))_i > 0\};$$
$$T_{j+1} = T_j \cup \{b \mid b \neq a,\ (d_i, o_b) \in G(\mathscr{P}) \Rightarrow i \in S\};$$
$$S_{j+1} = S_j \cup \{i \mid \exists b \in T_{j+1} : (o_b, d_i) \in G(\mathscr{P})\}.$$

Итеративный процесс считаем оконченным, если для некоторого k выполнено $T_{k+1} = T_k$. При этом полагаем $S(w, a) = T_k$.

Доказательство этой леммы опущено. Оно вытекает из того, что оператор может быть включен в том и только в том случае, когда значения всех «питающих» его счетчиков положительны.

В следующей теореме построен критерий эквивалентности свободных однозначных простых операторных схем. При этом использована характеристическая последовательность $v_{\mathscr{P}}$.

Теорема 6.8. *Пусть $\mathscr{P}$, $\mathscr{P}'$ — свободные однозначные простые операторные схемы, множество операторов у них равны. Тогда эти схемы эквивалентны в том и только в том случае, когда выполнены условия*

(i) *для каждого a выполнено* $\gamma_{\mathscr{P}}(a) = \gamma_{\mathscr{P}'}(a)$,

(ii) *из $a\rho b$ следует* $\mathscr{E}_{\bar{a}\bar{b}}(v_{\mathscr{P}}) = \mathscr{E}_{\bar{a}\bar{b}}(v_{\mathscr{P}'})$.

Доказательство. Условия (i), (ii) означают (используем теорему 6.5 и следствие (6.1)), что если x, x' являются вычислениями для $\mathscr{P}$, $\mathscr{P}'$ соответственно, то $\mathscr{E}_{\bar{a}\bar{b}}(x) = \mathscr{E}_{\bar{a}\bar{b}}(x')$. Из устойчивости и определенности следует, что в качестве x, x' можно выбрать «последовательные» вычисления, в которых за выключением произвольного оператора непосредственно следует его включение. Для произвольной ячейки $i \in M$ введем обозначение $S_i = \{\bar{a} \mid i \in R(a)\}$. Фиксируем произвольную интерпретацию $\mathscr{J}$. Используя индукцию, можно показать, что для произвольных i, k справедливо следующее утверждение: если $x = x_1 \bar{c} c_1 x_2$, $x' = x'_1 \bar{d} d_1 x'_2$, где $\bar{c}$, d являются k-ми вхождениями элементов из S_i в словах x, x' соответственно, то выполнено $c = d$, $\Pi_{D(c)}(q_0 \cdot x_1) = \Pi_{D(c)}(q'_0 \cdot x'_1)$, а следовательно,

$$\Pi_{R(c)}(q_0 \cdot x_1 \bar{c} c_1) = \Pi_{R(c)}(q'_0 \cdot x'_1 \bar{c} c_1),$$

где q_0, q'_0 — начальные состояния схем $\mathscr{P}$, $\mathscr{P}'$ соответственно. Отсюда следует, что вычисления x, x' и, следовательно, произвольные $\mathscr{J}$-вычисления (это следует из однозначности) порождают равные истории ячеек.

7. НЕСКОЛЬКО ОБОБЩЕНИЙ

Модель, определенная в этой статье, нацелена на исследование природы алгоритмов и динамики их выполнения. Некоторые затронутые здесь вопросы требуют углубления. Одно из возможных направлений состоит в ограничении класса рассматриваемых интерпретаций. Это ограничение можно сформулировать как модификацию определения схемы, состоящую в детализации операторов. Например, задавая формальное отношение эквивалентности между операторами, можно достичь, чтобы эти операторы вычисляли одну и ту же функцию при произвольной интерпретации. Можно ввести оператор пересылки, всегда интерпретируя его как тождественную функцию. Имеет смысл ввести операторы, работающие с индексами, т. е. модифицирующие входные и выходные ячейки некоторых операторов. Было бы интересно обобщить полученные результаты на такие частично интерпретированные модели.

Другое направление состоит в ограничении класса рассматриваемых вычислений, формализующем некоторые «реальные» ограничения. Например, может быть учтена информация о времени выполнения операторов, об очередности их выполнения, действии приоритетов, прерываний. При этом важно, чтобы введение новых понятий оправдывалось получением новых результатов.

По-видимому, перспективным является сосредоточение на некоторых подклассах схем и некоторых их свойствах. Например, заслуживают внимания подклассы счетчиковых схем, допускающие большее разнообразие типов следования, чем параллельные операторные схемы. Можно попытаться обобщить на них результаты, полученные для простых операторных схем.

Важно также развитие предложенной модели, взятой во всей ее широте. Здесь интересны нерешенные проблемы, упомянутые в разд. 4, а для построенных алгоритмов возникает задача их улучшения. Допустимы другие понятия однозначности и эквивалентности. Например, схему можно считать однозначной, если однозначны выходные значения некоторых выделенных ячеек. Возникает также задача построения системы эквивалентных преобразований параллельных схем. Все эти проблемы могут быть преломлены в свете изучения максимально параллельных схем.

Авторы благодарны М. О. Рабину, А. Л. Розенбергу, Д. Д. Ратледжу и Д. Л. Слутцу за интересные и полезные советы.

СПИСОК ЛИТЕРАТУРЫ

1. Ginsburg S., The Mathematical Theory of Context-Free Languages, McGraw-Hill, New York, 1966.
2. Янов Ю. И., О логических схемах алгоритмов, Проблемы кибернетики, I, Физматгиз, М., 1958, стр. 75—127.
3. Karp R. M., Miller R. E., Properties of a model for parallel computations: determinacy termination, queueing, *SIAM J. Appl. Math.*, **14** (1966), 1390—1411.
4. Karp R. M., Miller R. E., Parallel program schemata: A mathematical model for parallel computation, IEEE Conference Record of the Eighth Annual Symposium on Switching and Automata Theory, October 1967, p. 55—61.
5. Karp R. M., Miller R. E., Winograd S., The organization of computations for uniform recurrence equations. *J. Assoc. Comp. Math.*, **14** (1967), 563—590.
6. König D., Theorie der Endlichen und Unendlichen Graphen, Akademische Verlagsgesellschaft, Leipzig, 1936.
7. Luckham D. C., Park D. M. R., Paterson M. S., On formalised computer programs (Preliminary Draft), Programming Research Group, Oxford University, 1967.
8. Post E., A variant of a recursively unsolvable problem, *Bull. Am. Math. Soc.* **52** (1946), 264—268.
9. Rabin M. O., Scott D., Finite automata and their decision problems, *IBM J. Res. Develop.*, **3** (1959), 198—220.
10. Rutledge J. D., On Ianov's program schemata, *J. Assoc. Comp. Mach.* **11** (1964), 1—9.
11. Itkin V. E., Zwinogrodski Z., On program Schemata Equivalence, *Journal of Comp. and Syst. Scien.*, **6**, No. 1 (1972).

Сравнительная схематология[1])

М. Патерсон и К. Хьюитт

ВВЕДЕНИЕ

Интуитивно мы можем представить, что значит, что один язык программирования мощнее другого или что некоторое подмножество языка является его адекватным «ядром». Однако попытки формализовать это понятие наталкиваются на серьезные трудности теоретического характера. Эти трудности связаны с тем, что даже крайне простые языки являются «универсальными» в следующем смысле. Если язык допускает программирование с использованием простых арифметических функций и функций для обработки списков, то можно моделировать любую эффективную структуру управления — обычно посредством подходящего кодирования работы машины Тьюринга. В частности, в простом языке с некоторой базовой арифметикой можно записать программу для любой частично рекурсивной функции. Обычно такое кодирование очень неестественно и крайне неэффективно. Поэтому, если мы хотим продолжить практическое изучение сравнительной мощности разных языков, следует отказаться от точно заданных функций и рассматривать вместо этого абстрактные, неинтерпретированные программы, или *схемы*. В следующих разделах излагаются результаты предварительных исследований в этой области.

ЯЗЫКИ

Простейший из изучаемых нами языков — это язык *стандартных схем*, в котором мы записываем схемы программ, подобные изображенной на рис. 1.

Если проинтерпретировать используемые в схеме основные функциональные и предикатные символы, то ее можно рассматривать как исполняемую программу, задающую частичную функцию. Условимся, что в качестве аргументов этой функции берутся начальные значения ячеек L_1, L_2, ... (их столько, сколько использовано в схеме), а ее значением является конечное значение ячейки L_0, если оно определено. Другие ячейки

[1]) Paterson M. S., Hewitt C. E., Comparative schematology, in Record of Project MAC Conference on concurrent systems and parallel computation, ACM, New York (December 1970), pp. 119—128.

$M_0, M_1, \ldots, N_0, N_1, \ldots$ используются только как рабочие. Будем предполагать, что ни одна из рабочих ячеек не используется в качестве аргумента прежде, чем ей присвоено значение.

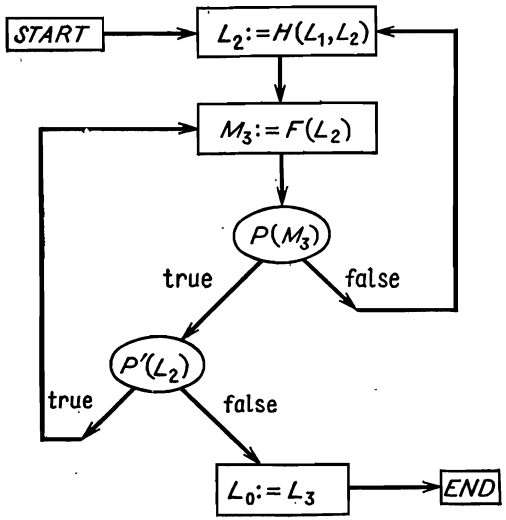


Рис. 1. Стандартная схема.

Другой язык, которым мы будем пользоваться, допускает использование условных выражений для рекурсивного определения функций. Например:

$$f(x_1, x_2) = \textbf{if } g(x_2) \textbf{ then } F(\textbf{if } P(x_1) \textbf{ then } x_2 \textbf{ else } A)$$
$$\textbf{else } (\textbf{if } P(x_2) \textbf{ then true else } f(x_2, F(x_1)))$$
$$g(x_1) = \textbf{if } P(f(x_1, F(x_1))) \textbf{ then false else } g(A)$$

Здесь две абстрактные функции определяются посредством одновременной рекурсии. Если дана интерпретация основных функциональных и предикатных символов (обозначаемых заглавными буквами), то такая система определяет частичную функцию, соответствующую каждому уравнению. Считается, что система вычисляет функцию, задаваемую первым уравнением. Такая система называется *рекурсивной схемой.* Если система состоит из одного уравнения, то она называется *простой* рекурсивной схемой, если из нескольких, — *составной.*

ИНТЕРПРЕТАЦИИ И ЭКВИВАЛЕНТНОСТЬ

Интерпретация I стандратной схемы и рекурсивной схемы задает

1) предметную область D;

2) для каждого основного предикатного символа P — предикат (всюду определенный)

$$P_I\colon D \to \{\textbf{true, false}\}$$

3) для каждого n-местного ($n \geqslant 0$) основного функционального символа F — функцию (всюду определенную)

$$F_I\colon D^n \to D$$

Обычно P, P', ... используются как предикатные символы, а другие заглавные буквы — как функциональные. Константа — это 0-местная функция.

Частичная функция, задаваемая схемой S при интерпретации I, обозначается S_I. Для двух частичных функций u, v пишем $u \simeq v$, если для любого x либо оба $u(x)$ и $v(x)$ не определены, либо оба определены и $u(x) = v(x)$. Две схемы S и S' (не обязательно одного и того же типа) называются (строго) *эквивалентными*, $S \equiv S'$, если для любой интерпретации справедливо $S_I \simeq S'_I$.

Пусть $\mathscr{P}$ и $\mathscr{P}'$ — два класса схем. Мы пишем $\mathscr{P} \leqslant \mathscr{P}'$, если $\forall S \in \mathscr{P}\, \exists S' \in \mathscr{P}'\, [S \equiv S']$; $\mathscr{P} < \mathscr{P}'$, если $\mathscr{P} \leqslant \mathscr{P}'$, но неверно, что $\mathscr{P}' \leqslant \mathscr{P}$. Пусть $\mathscr{R}$ — класс рекурсивных схем, а $\mathscr{P}$ — класс стандартных схем.

Теорема 1. $\mathscr{P} < \mathscr{R}$.

Доказательство. Проверка справедливости включения $\mathscr{P} \leqslant \mathscr{R}$ основана на громоздкой конструкции, смысл которой в общих чертах раскрывается ниже (подробности см. в [1]). Каждому оператору b_i стандартной схемы U сопоставим абстрактную частичную функцию f_i, которая при данных значениях *всех* ячеек (в том числе и рабочих. — *Перев.*) вычисляется следующим образом. Начинаем движение по схеме от оператора b_i с этими значениями ячеек; значением функции объявляется конечное значение ячейки L_0, если оно определено. Легко написать составную рекурсивную схему, которая определяет f_i рекурсивно. Если b_0 — это оператор START, то функция, вычисляемая схемой U, получается из f_0 удалением части ее аргументов (соответствующих рабочим ячейкам. — *Перев.*).

Для того чтобы показать, что включение строгое, рассмотрим следующую рекурсивную схему V:

$$\mathbf{V}\colon f(x) = \textbf{if } P(x) \textbf{ then } x \textbf{ else } H(f(L(x)),\ f(R(x)))$$

Мы покажем, что никакая стандартная схема не может быть эквивалентна V.

При этом нам будет полезно понятие *свободной* интерпретации. В свободной интерпретации предметная область — это множество всех цепочек, составленных из основных функциональ-

ных символов. Тогда, например, если H — двухместный функциональный символ, а E_1 и E_2 — такие цепочки, то

$$H_I(E_1, E_2) = HE_1E_2$$

для любой свободной интерпретации I. Интерпретация же предикатных символов никак не ограничивается. Понятно, что в большинстве случаев достаточно рассматривать лишь свободные интерпретации. Так, например, для любых S и S', $S \equiv S'$ тогда и только тогда, когда $S_I \simeq S'_I$ для всех свободных I.

В настоящем (а также в следующем) доказательстве мы используем семейство свободных интерпретаций $\{I_n\}$, $n > 0$, где

$$P_{I_n}(E) = \begin{cases} \textbf{true}, & \text{если длина } E \text{ равна } n, \\ \textbf{false} & \text{в остальных случаях.} \end{cases}$$

Каково значение $V_{I_n}(\lambda)$, где λ — пустая цепочка? Для $n = 1$ это значение равно HLR, для $n = 2$ — $HHLLRLHLRRR$ и т. д.

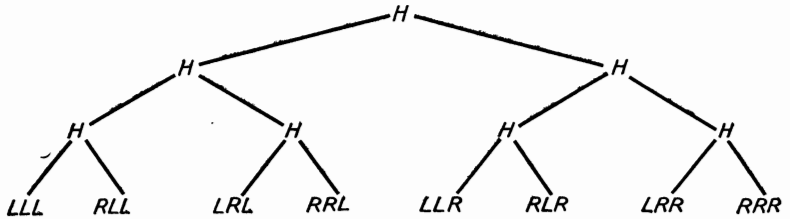


Рис. 2. Двоичное дерево при $n = 3$.

Мы покажем, что для вычисления значения при I_n требуется по крайней мере $n + 1$ ячейка. Для этого полезно представить эту же задачу в геометрической форме. Мы будем размещать фишки на конечном двоичном дереве T_n. Для $n = 3$ дерево T_n изображено на рис. 2. Правила игры таковы:

1) каждой фишкой в любой момент можно накрыть висячую вершину;

2) если непосредственные преемники данной вершины накрыты фишками, то и саму ее можно накрыть любой фишкой.

Сколько нужно фишек, чтобы можно было достичь корня дерева? Легко видеть, что $n + 1$ фишки достаточно для T_n. Следующее рассуждение показывает, что такое количество фишек является необходимым.

Дерево называется *замкнутым* на данном шаге игры, если на каждом пути от его корня до висячей вершины имеется хотя бы одна фишка. Первоначально, когда ни одна вершина дерева не накрыта фишкой, оно не замкнуто. В заключительный момент накрыт корень дерева и, значит, оно замкнуто. Рассмотрим тот шаг игры, когда дерево впервые стало замкнутым. Это

могло случиться только в результате накрытия висячей вершины, благодаря чему замкнулся последний путь. До этого шага на этом пути, таким образом, не было ни одной фишки, и, значит, каждое из n поддеревьев, непосредственно ответвляющихся от него, должно быть независимым образом замкнуто. Поскольку для того, чтобы замкнуть любое дерево, требуется хотя бы одна фишка, после рассматриваемого шага на T_n имеется по крайней мере $n + 1$ фишка.

Связь между вычислением стандартной схемы и шагами этой игры представляется нам очевидной. Предположим теперь, что некоторая стандратная схема U эквивалентна схеме V, и в ней используется r ячеек. Но это немедленно приводит к противоречию, поскольку при интерпретации I_r схема U не может вычислить требуемое значение. Это и завершает первое доказательство.

Таким образом, стандартные схемы не могут вычислять некоторые абстрактные функции по той простой причине, что фиксированного числа ячеек недостаточно для вычисления всех требуемых значений. Это не очень интересное обстоятельство, поэтому во втором доказательстве этой же теоремы будет показано, что стандартная схема может не дать нужного результата в силу более глубоких причин, отражающих неадекватность ее управляющей структуры. Для этого доказательства берется рекурсивная схема, вычисляющая частичный предикат; по сути дела, она получена из V заменой H на логическую функцию &.

Второе доказательство теоремы 1. Рассмотрим рекурсивную схему:

W : $f(x) =$ **if** $P(x)$ **then true else (if** $f(L(x))$ **then** $f(R(x))$ **else false)**

Поскольку значение функции, задаваемой этой схемой, либо истинно, либо не определено, все соображения, на которых основано предыдущее доказательство, здесь бесполезны. Предположим, что стандратная схема U эквивалентна W и в ней имеется t операторов и r ячеек. Без ограничения общности можно считать, что, кроме P, L и R, в U не используются никакие другие предикатные и функциональные символы. *Состояние* схемы U при данной интерпретации определяется фиксацией некоторого оператора и значений всех ячеек. Два состояния S_1 и S_2 называются (в данном рассмотрении) *эквивалентными*, если в ходе вычислений, начинающихся в этих состояниях, проходится одна и та же последовательность операторов.

Рассмотрим классы эквивалентности состояний схемы U при интерпретации I_n, определенной выше. Нетрудно понять, что единственное свойство значения ячейки, которое может повлиять на принадлежность состояния тому или иному классу эквива-

лентности, это длина цепочки. Кроме того, любые два слова длины большей n не различимы предикатом P. Поэтому при интерпретации I_n схема U имеет максимум $t(n+2)^r$ классов эквивалентности. Если во время вычисления в схеме U дважды встретятся состояния из одного класса эквивалентности, то она неизбежно зациклится. Вычисление $U_{I_n}(\lambda)$ по предположению должно закончиться (с результатом **true**), поэтому ни один класс эквивалентности не должен повториться дважды, и, значит, должно быть сделано не более $t(n+2)^r$ шагов. Если n выбрать достаточно большим, так что

$$2^n > t(n+2)^r,$$

то за время вычисления U не успеет проверить значения предиката P на всех цепочках

$$\underbrace{LL\ldots L}_{n},\quad \underbrace{RL\ldots L}_{n},\quad \ldots,\quad \underbrace{RR\ldots R}_{n}$$

но остановится с результатом **true**. Если немного изменить I_n, так чтобы P стал ложным на тех словах, на которых он не проверялся, то U, конечно, «не заметит» этого и снова даст ответ **true**, в то время как значение W в этом случае не определено. Таким образом, $U \not\equiv W$, и доказательство завершено. (Этот метод доказательства применим, естественно, и к схеме V.)

ЛИНЕЙНЫЕ РЕКУРСИВНЫЕ СХЕМЫ

Эффективно охарактеризовать те рекурсивные схемы, для которых существуют эквивалентные им стандартные, невозможно. В самом деле, если $\mathscr{P}^* \subseteq \mathscr{R}$ есть подмножество таких рекурсивных схем, то ни оно само, ни $\mathscr{R} \setminus \mathscr{P}^*$ не являются рекурсивно перечислимыми. Мы не приводим здесь доказательства этого факта; оно опирается на технику, описанную в [1] и [2]. Лучшее, на что можно надеяться, — эффективно охарактеризовать большие подклассы классов $\mathscr{P}^*$ и $\mathscr{R} \setminus \mathscr{P}^*$. В настоящее время описан довольно широкий эффективный подкласс класса $\mathscr{R} \setminus \mathscr{P}^*$, содержащий обе схемы V и W, который мы пытаемся расширить дальше.

С другой стороны, ниже кратко описывается класс линейных рекурсивных схем, который аппроксимирует $\mathscr{P}^*$. Рекурсивную схему можно рассматривать как описание способа вычисления значения определяемой функции, исходя из ее значений, а также значений других функций на других аргументах. Определение этих значений может в свою очередь потребовать но-

вых рекурсивных обращений и т. д. Например, в следующей схеме:

$$f(x, y) = \textbf{if}\, P(x)\, \textbf{then}\, f(x, S(y))\, \textbf{else}$$
$$(\textbf{if}\, P(g(R(x)))\, \textbf{then}\, S(x)\, \textbf{else}\, H(g(R(x)), R(y)))$$
$$g(x) = \textbf{if}\, P'(x)\, \textbf{then}\, H(g(S(x)), R(g(S(x))))\, \textbf{else}$$
$$(\textbf{if}\, P(x)\, \textbf{then}\, f(x, x)\, \textbf{else}\, A)$$

для вычисления $f(x, y)$ может понадобиться значение $f(x, S(y))$ или значение $g(R(x))$, а для вычисления $g(x) — g(S(x))$ или

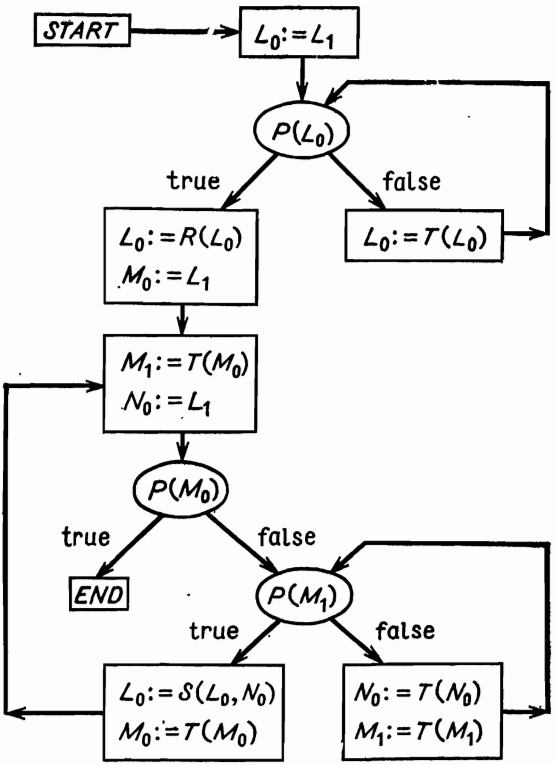


Рис. 3. Стандартная схема L, эквивалентная линейной рекурсивной схеме L^*.

$f(x, x)$. Заметим, однако, что в этом примере при любом вычислении *непосредственно* может потребоваться максимум одно новое значение определяемой функции. Это свойство и характеризует *линейные* рекурсивные схемы.

Теорема 2. *Если рекурсивная схема X линейна, то* $X \in \mathscr{P}^*$.

Мы не будем приводить здесь доказательства теоремы, а дадим вместо этого пример трансляции простой линейной рекурсивной схемы в эквивалентную стандратную схему. Доказательство же заключается в сведении случая произвольной рекурсивной схемы к схеме, изоморфной нашему примеру. Поэтому описываемую ниже трансляцию можно рассматривать как «каноническую».

$\mathbf{L}^*$: $f(x) =$ **if** $P(x)$ **then** $R(x)$ **else** $S(f(T(x)), x)$

Наличие у S второго аргумента является важной особенностью, усложняющей этот пример. Пусть $v_0, v_1, \ldots, v_m$ — последовательность значений функции f, вырабатываемых при обращении к ней во время некоторого заканчивающегося вычисления.

Заметим, что

$$v_m = R_I(T_I^{(m)}(x)),$$
$$v_r = S_I(v_{r+1}, T_I^{(r)}(x)), \quad r = 0, \ldots, m-1.$$

Поэтому если бы мы имели счетчик, позволяющий пересчитывать целые числа, не превосходящие m, то требуемое значение f можно было бы легко получить, вычисляя в обратном порядке $v_m, v_{m-1}, \ldots, v_0$. Такой счетчик, однако, можно моделировать, если заметить, что

$$P_I(T_I^{(r)}(x)) = \begin{cases} \textbf{false} & r = 0, \ldots, m-1, \\ \textbf{true} & r = m. \end{cases}$$

Стандартная схема L, эквивалентная L^*, изображена на рис. 3.

ОБСУЖДЕНИЕ

Конечно, эффективность описанной выше трансляции оставляет желать лучшего, и за возможность использовать лишь конечное число ячеек приходится платить увеличением времени вычисления. Суть проблемы в данном примере заключается в том, что требуется последовательно вычислить значения

$$T^{(m)}(x), T^{(m-1)}(x), \ldots, T(x).$$

Схема L делает это, вычисляя каждый терм независимо, всякий раз начиная с x, на что требуется (по порядку) m^2 операций. С целью более детального исследования проблемы соотношения между временем и памятью рассмотрим следующую простую комбинаторную задачу.

Предположим, что в ячейке M_0, доступной в любой момент только для считывания, хранится значение x; кроме того, имеется k ячеек $M_1, \ldots, M_k$. Задача состоит в том, чтобы для данного фиксированного m указать кратчайшую цепочку операторов вида

$$M_i := T(M_j),\ 0 < i \leqslant k,\ 0 \leqslant j \leqslant k,$$

вычисляющую последовательно значения $T^{(m)}(x), \ldots, T^{(2)}(x), T(x)$.

Значения могут появляться в любых ячейках, но должны порождаться именно в этом порядке. Пусть $L(k, m)$ — длина такой кратчайшей цепочки; например, $L(1, m) = {}^1/_2 m(m+1)$. Нами получена точная формула для $L(k, m)$, из которой следует, что

$$L(k,\ m) \sim m^{1+1/k}.$$

Таким образом, функции, задаваемые линейными рекурсивными схемами, можно вычислять достаточно эффективно, используя лишь фиксированное число ячеек. Однако, по-видимому, для реализации этой возможности требуется более гибкая структура управления, чем та, которая имеется в стандартной схеме.

В настоящее время мы исследуем соотношения между различными обогащенными формами стандартных схем, такими, как схемы со счетчиками, магазинами. Эти обогащения можно выразить либо дальнейшим усложнением понятия «схемы», либо фиксацией части интерпретации в обычной стандартной схеме. Например, счетчик, можно выразить, если обычным образом зафиксировать интерпретацию константного символа ZERO, функциональных символов ADD1, SUB1 и предикатного символа POSITIVE. При применении в этой области хорошо известных результатов из теории автоматов следует проявлять некоторую осторожность. Например, класс схем со счетчиками не является в этой теории универсальным, поскольку из первого доказательства теоремы следует, что ни одна такая схема не может быть эквивалентна рекурсивной схеме V. Однако можно показать, что при подходящем определении класса схем с магазинами он оказывается эквивалентным классу рекурсивных схем.

КЛАСС ПАРАЛЛЕЛЬНЫХ СХЕМ

Очень простое расширение класса рекурсивных схем достигается с помощью параллельного варианта условного выражения:

IF p **THEN** g **ELSE** r

принимающего значение g, если p истинно, значение r, если p ложно, и, кроме того, значение g, если p не определено, но g и r определены и равны между собой. Обозначим выражение

IF p **THEN true ELSE** g

через $p/\mathrm{OR}/g$. Понятно, что если хотя бы одно из значений p и g истинно, то и $(p/\mathrm{OR}/g)$ истинно, а если p и g оба не определены, то и $(p/\mathrm{OR}/g)$ не определено. Пусть $\mathscr{S}$ — класс схем, полученный расширением $\mathscr{R}$ за счет применения функции OR.

Теорема 3. $\mathscr{R} < \mathscr{S}$.

В доказательстве этой теоремы (слишком длинном, чтобы привести его здесь) показывается, что никакая рекурсивная схема не может быть эквивалентна схеме S:

S : $f(x) =$ **if** $p(x)$ **then true else** $(f(L(x))/OR/f(R(x)))$

Значение $S_I(\lambda)$, где I — свободная интерпретация, истинно, если существует хотя бы одна цепочка из букв L и R, на которой p истинно, и не определено в противном случае. В доказательстве рассматривается поведение рекурсивных схем при такой свободной интерпретации, что p тождественно ложно. Выясняется, что рекурсивная схема может «просмотреть» только те цепочки в двоичном дереве (L, R)-цепочек, которые находятся на ограниченном расстоянии от некоторого конечного числа путей в дереве. Поэтому, какова бы ни была рекурсивная схема, она не сможет проверить часть цепочек, а, значит, при некоторой интерпретации даст неправильное значение.

ЗАКЛЮЧЕНИЕ

Существует ясное понятие эффективного процесса вычисления, связанного с неинтерпретированными функциями, и нам хотелось бы иметь достаточно естественное обогащение стандартных схем, которое позволило бы моделировать любое такое эффективное вычисление. Хорошим кандидатом на эту наиболее важную в иерархии схем роль является, по-видимому, класс схем с двумя магазинами. Если в схеме имеется возможность запоминать связанные с управлением маркеры, а также различать их между собой, то «универсальность» этой модели обеспечена.

Всюду в этой статье мы пользовались тем упрощающим предположением, что интерпретация задает только *всюду определенные* функции и предикаты. Снятие этого ограничения меняет часть наших результатов и приводит к некоторым новым

соображениям. Например, для доказательства включения $\mathscr{R} < \mathscr{P}$ достаточно заметить, что уже для схемы

$$f(x) = (P(x)/OR/P'(x))$$

не существует эквивалентной ей рекурсивной схемы. Кроме того, понятие эффективного вычисления становится в этом случае неясным и зависит от соглашений, которые будут приняты в отношении вычисления частичных базисных функций.

Нам хотелось бы отметить работу доктора Х. Р. Стронга [3], в которой более детально обсуждаются многие из затронутых в данной статье тем, а также выразить признательность за те полезные дискуссии, которые мы с ним имели.

СПИСОК ЛИТЕРАТУРЫ

1. Luckham D. C., Park D. M. R., Paterson M. S., On formalized computer programs, *J. of Computer and System Science*, **4**, № 3 (June 1970), 220—249. (Русский перевод: Лакхэм Д., Парк Д. М., Патерсон М. С., О формализованных машинных программах, Кибернетический сборник, новая серия, вып. 12, «Мир», М., 1975, стр. 78—114.)
2. Paterson M. S., Equivalence problems in a model of computation, Ph. D. Thesis, Cambridge University, 1967.
3. Strong H. R., Translating recursion equations into flow-charts, *J. of Computer and System Science*, **5**, № 3 (June 1971), 254—285.

Стандартные схемы, рекурсивные схемы и формальные языки[1])

С. Гарлэнд и Д. Лакхэм

ВВЕДЕНИЕ

Изучение схем программ преследует по крайней мере три принципиальные цели. Во-первых, дать точную формальную модель понятия вычислительной программы в виде, который совершенно не зависел бы от особенностей или работы какой-либо (реальной или абстрактной) вычислительной машины. Рассматриваемые схемы воплощают как раз те особенности или конструкции языков программирования, которые кажутся «существенными», — наше исследование и предназначено для того, чтобы показать, являются ли они действительно существенными в каком-то смысле. Во-вторых, развить теорию, обосновывающую оптимизацию программ, опять-таки независимо от какой-либо конкретной машины или языка программирования. И третьей целью является нахождение общих методов (например, систем правил вывода или преобразований программ) для проверки правильности и контроля данной программы по ее спецификациям. Разумеется, эти цели не являются независимыми. Результаты изучения оптимизации почти несомненно найдут применение при проверке правильности программ и наоборот.

Существенным для всех трех этих целей является вопрос о трансляции программ с одним множеством свойств в программы с другими свойствами. Теорема о транслируемости, утверждающая, что любая схема из одного класса может быть транслирована в эквивалентную схему другого класса, дает нам возможность заменять одни программные средства другими. Результаты такого сорта могут дать нам нормальные формы, помочь оптимизации или свести проблему проверки правильности к некоторой другой, которую мы уже умеем решать. С другой стороны, теорема о нетранслируемости, утверждающая, что некоторые средства нельзя заменить другими, дает нам несколько неуловимое интуитивное понятие «мощности» языка программирования.

---

[1]) Garland S. J., Luckham D. C., Program Schemes, Recursion Schemes and Formal Languages, Report UCLA-ENG-7154, June 1971, School of Engineering and Applied Science, University of California, Los Angeles.

В данной статье мы изучаем транслируемость схем и связанные с ней проблемы эквивалентности, когда нужно определить, эквивалентны ли две схемы в заданном классе. В частности, мы затрагиваем вопросы развития методов решения этих проблем. Нам кажется, что должны существовать некоторые «общие методы» или по крайней мере можно задать общую форму для существующих «хитростей». По существу каждый раздел настоящей статьи содержит применения таких частных методов исследования. Имеются четыре метода: приложение результатов теории формальных языков, программистская техника, оценка сложности вычислений и метод «следов» (грубо говоря, применение метода Рабина и Скотта [7], ограничивающего поиск, необходимый для установления различий в поведении двух схем).

Хотя некоторые результаты справедливы для более общего понятия схемы, мы ради иллюстраций почти всегда имеем в виду три следующих специальных класса схем: стандартные схемы, стандартные схемы, снабженные счетчиками, и рекурсивные схемы. Бóльшая часть изложения ограничена унарными схемами (то есть схемами, содержащими только функции и предикаты от одного аргумента), а в случае рекурсивных схем мы интересуемся исключительно унарными рекурсивными схемами с одной переменной [1]), введенными де Беккером и Скоттом в [1].

Раздел 1 содержит определения, терминологию и диаграмму, резюмирующую большинство результатов о транслируемости и нетранслируемости, сформулированных в последующих разделах.

Раздел 2 имеет дело с приложениями теории формальных языков.

С данной унарной схемой ассоциируются два сорта языков и формулируется несколько теорем о нетранслируемости, непосредственно вытекающих из классических теорем теории формальных языков. Кроме того, некоторые естественные классы рекурсивных схем (например, линейные рекурсивные схемы) можно строго определить, задавая сопоставленные им языки. Точно так же некоторые результаты теории формальных языков наводят на мысль об аналогичных результатах для схем: например, теорема Хомского о нормальной форме для КС-языков приводит нас к факту, что всякая унарная рекурсивная схема эквивалентна такой рекурсивной схеме, в которой термы определяющих соотношений содержат не более двух определяемых функциональных символов.

---

[1]) Для унарных рекурсивных схем с одной переменной мы будем пользоваться сокращением dBS. — *Прим. перев.*

В разделе 3 приведены результаты, получаемые с помощью программистской техники; этот раздел содержит общие теоремы о транслируемости. Мы исследуем два определения транслируемости. Одно из них, которое кажется более естественным, говорит, что схема P является трансляцией (или результатом трансляции) схемы Q, если обе эти схемы дают один и тот же результат при любой интерпретации (т. е. P сильно эквивалентна Q). Это определение не ставит ограничений на порядок, в котором проводятся вычисления, а требует только, чтобы результаты, если они определены, были одними и теми же. Показано, что квазирациональные рекурсивные схемы (наименьший класс схем dBS, содержащий линейные рекурсивные схемы и замкнутый относительно функциональной подстановки) транслируемы в унарные стандартные схемы. Весь класс схем dBS также транслируем в класс стандартных схем, снабженных двумя счетчиками, — результат, который для бинарных рекурсивных схем оказывается неверным. Программистская техника используется также для того, чтобы показать, что класс языков значений стандартных схем в точности совпадает с классом всех рекурсивно-перечислимых языков. Это не только разрешает все наши споры, но говорит также о том, что методы раздела 2 не имеют так много приложений, как можно было бы надеяться.

Мы изучаем и более слабое понятие трансляции: говорят, что P является трансляцией Q, если Q есть «несущественное расширение» P. Оказывается, что всякая унарная рекурсивная схема транслируема в этом слабом смысле в стандартную схему.

В разделе 4 речь идет о сложности вычислений. Дан пример схемы dBS, которая не является сильно эквивалентной никакой стандартной схеме, а также никакой стандартной схеме, снабженной одним счетчиком. По существу мы пользуемся здесь элегантным примером бинарной рекурсивной схемы Патерсона и Хьюитта [6]. Кроме того, это дает нам возможность предположить, что квазирациональные схемы являются максимальным классом рекурсивных схем, транслируемых в стандартные схемы.

В разделе 5 рассматривается приложение техники следов. Показано, что проблема эквивалентности разрешима в классе линейных рекурсивных схем (что является частичным ответом на вопрос, поднятый де Беккером и Скоттом в [1]) и в классе схем Янова с постоянными функциями.

В статье мы поднимаем ряд вопросов, которые еще не поддаются нашим методам. Мы хотели бы понять, можно ли с помощью наших методов получить ответ на эти вопросы или требуются какие-то другие общие методы. Например, кажется верным, что требуется более тонкая оценка сложности вычислений,

если мы хотим показать существование унарной стандартной схемы с одним счетчиком, не эквивалентной никакой унарной стандартной схеме без счетчиков [1]). С другой стороны, кажется, что методы раздела 5 должны давать положительное решение проблемы эквивалентности для более общих классов рекурсивных схем.

1. ОПРЕДЕЛЕНИЯ

За немногими специально отмеченными исключениями мы следуем определениям и понятиям Хопкрофта и Ульмана [2], касающимся цепочек, языков и автоматов. Цепочка нулевой длины (или пустая цепочка) обозначается через λ.

Схемы являются абстрактными моделями вычислительных программ. Пусть дано множество $\mathscr{V}$ переменных (обычно обозначаемых $x, y, z, \ldots$), множество $\mathscr{F}$ *базисных* функций (обычно обозначаемых $f, g, \ldots$), которые могут быть использованы для присваивания новых значений переменным, и множество $\mathscr{P}$ предикатов (обычно обозначаемых $P, Q, \ldots$).

В терминах значений предикатов схема специфицирует тот порядок, в котором следует производить вычисления. Мы будем интересоваться преимущественно двумя классами схем, а именно: стандартными схемами и рекурсивными схемами.

Стандартная схема — это конечный список инструкций следующего вида:

a) присвоить переменной x значение $fx_1, \ldots, x_n$ и перейти к следующей инструкции (здесь $f \in \mathscr{F}$ — n-местная функция, а $x_1, \ldots, x_n \in \mathscr{V}$);

b) выполнить следующей инструкцию i, если P истинен на $x_1, \ldots, x_n \in \mathscr{V}$, иначе выполнить следующей инструкцию j (здесь $P \in \mathscr{P}$ — n-местный предикат и $x_1, \ldots, x_n \in \mathscr{V}$);

c) стоп.

Подробности, касающиеся стандартных схем, можно найти у Лакхэма, Парка и Патерсона [4]. Напоминаем читателю, что стандартные схемы имеют естественное представление в виде графа или диаграммы; например, следующая схема

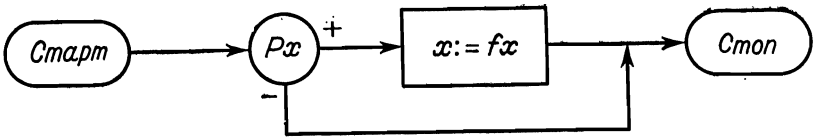
1) выполнить 2, если Px, иначе выполнить 3;

2) присвоить x значение fx;

---

[1]) Недавно Плейстед Д. А. (в работе «Стандартные схемы со счетчиками», опубликованной в Proceedings of Fourth Annual ACM Simposium on Theory of Computing, стр. 44—51) показал, что всякая стандартная схема с одним счетчиком эквивалентна некоторой стандартной схеме без счетчиков. Отсюда следует, что неквазирациональная рекурсивная схема 1.1 и из разд. 1 эквивалентна некоторой стандартной схеме (без счетчиков), что является опровержением предположения, что квазирациональные рекурсивные схемы — максимальный класс рекурсивных схем, транслируемых в стандартные схемы. — *Прим. перев.*

3) стоп
может быть представлена диаграммой



Последовательность следующих друг за другом присваиваний $x := f_1x, \ldots, x := f_nx$ мы будем изображать одним присваиванием $x := f_n \ldots f_2f_1x$, содержащим композицию функций $f_1, \ldots, f_n$. Такие произвольные термы, составленные из базисных функциональных символов, могут появляться в простых операторах присваивания наших диаграмм.

Мы не будем давать общего определения класса рекурсивных схем, а будем пока интересоваться преимущественно подклассами, которые обладают простым представлением. Назовем схему *унарной*, если она содержит только унарные функции и предикаты.

Тогда *унарная рекурсивная схема с одной переменной* (схема dBS) — это конечный список определяющих соотношений

$$F_1x := \textbf{если } P_1x \textbf{ то } \alpha_1x \textbf{ иначе } \beta_1x$$
$$\vdots$$
$$F_nx := \textbf{если } P_nx \textbf{ то } \alpha_nx \textbf{ иначе } \beta_nx,$$

где $F_1, \ldots, F_n$ — новые, *определяемые* функциональные символы, $P_1, \ldots, P_n$ — (не обязательно различные) предикаты, а $\alpha_1, \beta_1, \ldots, \alpha_n, \beta_n$ — (возможно, пустые) цепочки в алфавите определяемых и базисных функциональных символов. Например,

$$Fx := \textbf{если } Px \textbf{ то } fx \textbf{ иначе } FFfx \tag{1.1}$$

— унарная рекурсивная схема.

Схемы производят вычисления при задании интерпретаций, которые фиксируют смысл функций, предикатов и переменных. Интерпретация I задает непустое множество dom_I, называемое *областью* интерпретации I, и сопоставляет каждой $f \in \mathscr{F}$ функцию f_I над dom_I, каждому $P \in \mathscr{P}$ — характеристическую функцию отношения P_I над dom_I, а каждому $x \in \mathscr{V}$ — начальное значение $x_I \in \mathrm{dom}_I$. Интерпретация I унарной схемы называется *свободной*, если $\mathrm{dom}_I = \mathscr{F}^* \cdot \mathscr{V}$ (множество всех цепочек в алфавите базисных функциональных символов, за которыми следует символ переменной), $f_I(a) = fa$ для всех $f \in \mathscr{F}$, $a \in \mathrm{dom}_I$ и $x_I = x$ для всех $x \in \mathscr{V}$. Для всякой интерпретации I $(f_1 \ldots \ldots f_nx)_I = (f_1)_I(\ldots(f_n)_I(x_I))$.

Вычисление стандартной схемы S при интерпретации I определяется очевидным образом и может быть найдено у Лаккэма и др. [4]. *Значение* $\mathrm{val}_I(S)$ схемы S при интерпретации I равно последнему значению выделенной выходной переменной (обычно обозначаемой x), когда вычисление S при I заканчивается, и не определено в противном случае. (Другие переменные схемы S иногда называют *программными* или *рабочими* переменными.)

Вычисление унарной рекурсивной схемы можно кратко определить, позаимствовав некоторые понятия из теории формальных грамматик. Со всякой такой схемой

$$E\colon\ F_i x := \textbf{если}\ P_i x\ \textbf{то}\ \alpha_i x\ \textbf{иначе}\ \beta_i x\ (1 \leqslant i \leqslant n)$$

мы свяжем контекстно-свободную грамматику G со множеством терминальных символов, совпадающим с $\mathscr{F}$, нетерминальными символами $F_1, \ldots, F_n$ и продукциями $F_i \to \alpha_i$ и $F_i \to \beta_i$ $(1 \leqslant i \leqslant n)$. Пусть I — некоторая интерпретация. Вычисление схемы E при интерпретации I соответствует единственному правостороннему выводу в грамматике G, который *согласован* с интерпретацией I в следующем смысле: элементарный вывод $\gamma F_i w \underset{G}{\Rightarrow} \gamma\,\delta w$, где γ, $\delta \in (\mathscr{F} \cup \{F_1, \ldots, F_n\})^*$ и $w \in \mathscr{F}^* \cdot \mathscr{V}$, согласован с I, если $\delta = \alpha_i$ и $(P_i)_I(w_I) = 1$ или если $\delta = \beta_i$ и $(P_i)_I(w_I) = 0$. Если $F_1 x \overset{*}{\underset{G}{\Rightarrow}} w$ — правосторонний вывод некоторой терминальной цепочки $w \in \mathscr{F}^* \cdot \mathscr{V}$, согласованной с интерпретацией I, то $\mathrm{val}_I(E) = w_I$, в противном случае $\mathrm{val}_I(E)$ не определено.

В качестве альтернативы мы можем представить себе вычисление унарной схемы E как вычисление стандартной схемы S с одной переменной и стеком, в который может помещаться конечная цепочка символов. Вначале в стеке находится один функциональный символ F_1, и на каждом шаге вычисления стек содержит цепочку базисных и определяемых функциональных символов, которые еще нужно применить к текущему значению переменной. В схеме S запрограммировано извлечение и проверка верхнего символа в этом стеке. Если этот символ есть F_i, то S выполняет проверку P_i на текущем значении переменной; если значение предиката равно 1, то в стек помещается α_i (самый правый символ помещается в верхушку стека), если же значение P_i равно 0, то в стек помещается β_i. Если верхний символ — символ базисной функции f, то S применяет f к текущему значению своей переменной. Вычисление завершается, когда стек оказывается пустым.

Как нетрудно видеть, определяемый функциональный символ F_1 играет особую роль в рекурсивной схеме E, в то время как $F_2, \ldots, F_n$ выступают в качестве вспомогательных функций. Всякий раз, когда мы хотим подчеркнуть, какая из определяе-

мых функций F_i играет эту особую роль, мы пишем $E(F_i)$ вместо E.

Различные классы схем можно получить как подклассы классов стандартных схем и рекурсивных схем. *Схемы с одной переменной*, как об этом говорит их название, имеют только одну переменную; в частности, все схемы dBS являются таковыми. Унарные рекурсивные схемы можно классифицировать далее, исходя из сопоставленных им грамматик: линейным рекурсивным схемам соответствуют линейные контекстно-свободные грамматики (т. е. цепочки в правой части продукций содержат не более одного нетерминального символа), праволинейным схемам соответствуют праволинейные грамматики и т. д.

Мы будем исследовать относительную мощность различных классов схем. Две схемы R и S называются *(сильно) эквивалентными* (обозначение: $R \equiv S$) тогда и только тогда, когда для всякой интерпретации I $\mathrm{val}_I(R) \simeq \mathrm{val}_I(S)$ ($\simeq$ означает, что либо оба значения определены и совпадают, либо оба не определены). Класс схем $\mathscr{R}$ *транслируем* в класс схем $\mathscr{S}$ (обозначение: $\mathscr{R} \to \mathscr{S}$), если для каждой схемы $R \in \mathscr{R}$ существует сильно эквивалентная схема $S \in \mathscr{S}$. Известно, например, что класс стандартных схем с одной переменной транслируем в класс унарных праволинейных рекурсивных схем путем сопоставления всякой инструкции данной стандартной схемы определяемой функции F_i с определяющим соотношением

(a) $F_i x := F_{i+1} f x$

когда i-я инструкция — это $x := fx$ (это явно «незаконное» определяющее соотношение — просто сокращение для

$$F_i x := \textbf{если } Px \textbf{ то } F_{i+1} f x \textbf{ иначе } F_{i+1} f x)$$

(b) $F_i x :=$ **если** Px **то** $F_j x$ **иначе** $F_k x$, когда i-я инструкция — это «выполнить j, если Px — истина, иначе выполнить k» или

(c) $F_i x := x$, когда i-я инструкция — это «стоп».

Два класса $\mathscr{R}$ и $\mathscr{S}$ схем называются *взаимно транслируемыми*, если $\mathscr{R} \to \mathscr{S}$ и $\mathscr{S} \to \mathscr{R}$ (обозначение $\mathscr{R} \rightleftarrows \mathscr{S}$).

Доказательства эквивалентности и транслируемости облегчает следующая лемма.

(1.2) Лемма. *Для всяких схем R и S справедливо $R \equiv S$ тогда и только тогда, когда $\mathrm{val}_I(R) \simeq \mathrm{val}_I(S)$ для всех свободных интерпретаций I.*

Доказательство. Необходимость очевидна. Для доказательства достаточности предположим, что $R \not\equiv S$. Тогда для некоторой интерпретации I $\mathrm{val}_I(R) \not\simeq \mathrm{val}_I(S)$. Определим свободную интерпретацию I', полагая $P_{I'}(a) = P_I(a_I)$ для всех $P \in \mathscr{P}$ и $a \in \mathscr{F}^* \mathscr{V}$. Тогда $\mathrm{val}_I(R) \simeq (\mathrm{val}_{I'}(R))_I$ и $\mathrm{val}_I(S) \simeq (\mathrm{val}_{I'}(S))_I$,

так как интерпретация I и I' «изоморфны». (Формальное доказательство этого факта требует формального определения понятия схемы — мы же в данной работе хотим уклониться от абстракций. Читатель может удовлетвориться доказательствами для стандартных и рекурсивных схем.) Поэтому $\mathrm{val}_{I'}(R) \not\supseteq \mathrm{val}_{I'}(S)$.

Мощность класса схем иногда возрастает, если в них разрешается употреблять новые полностью или частично интерпрети-

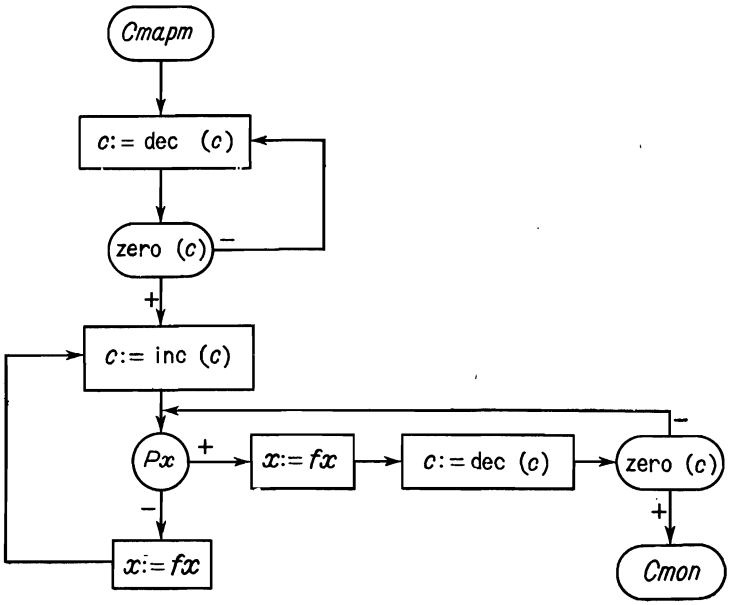


Рис. 1. Стандартная схема со счетчиком, эквивалентная рекурсивной схеме $Fx :=$ **если** Px **то** fx **иначе** $FFfx$.

рованные базисные функции или предикаты. Например, к стандартным схемам можно добавить тождественную функцию (т. е. базисную функцию f, такую, что для всякой интерпретации I и всякого $a \in \mathrm{dom}_I$ справедливо $f_I(a) = a$). Однако, как мы увидим в разд. 3, употребление тождественных функций можно элиминировать. Схемы также можно снабдить константными функциями (т. е. базисными функциями f, такими, что f_I для всякого I — одна и та же константа) или счетчиками. Константные функции иногда называют константами. *Счетчик* в стандартной схеме — это переменная, которой нельзя обычным образом присвоить значение, она не может также использоваться для присваивания значений другим переменным. Вместо этого имеются две новые базисные функции: inc (возрастание) и dec (убывание), которые используются для изменения значения

счетчика, и новый предикат zero. Интерпретация I является допустимой для схем со счетчиками, если dom_I содержит все неотрицательные целые числа, $\mathrm{inc}_I(n) = n + 1$, $\mathrm{dec}_I(n+1) = n$, $\mathrm{dec}_I(0) = 0$, $\mathrm{zero}_I(n+1) = 0$ и $\mathrm{zero}_I(0) = 1$. Стандартная схема со счетчиком, изображенная на рис. 1, сильно эквивалентна рекурсивной схеме примера 1.1.

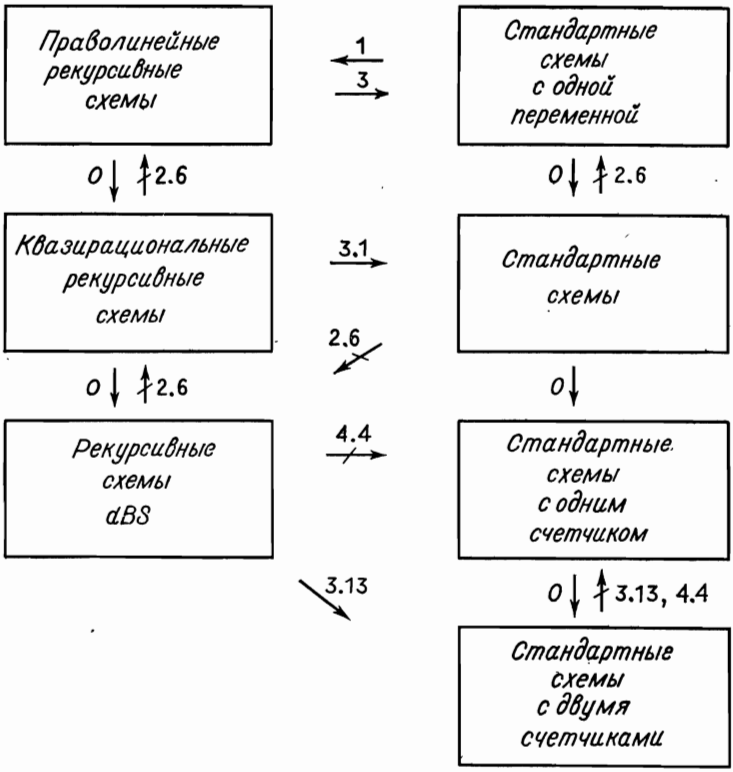


Рис. 2. Сводка результатов транслируемости для унарных схем.

Нашей главной целью в этой статье является демонстрация некоторых общепримениимых методов исследования транслируемости схем. Если не оговорено противное, все рассматриваемые схемы являются унарными, ничем не снабженными схемами. Мы не будем в дальнейшем писать слово «унарный», если только не захотим особо подчеркнуть это. Рис. 2 суммирует большинство результатов о транслируемости и нетранслируемости, установленных в разд. 2, 3 и 4; номера, проставленные рядом со стрелками, указывают номера теорем; буква O рядом со стрелкой указывает, что результат верен по определению.

2. ЯЗЫКИ ЗНАЧЕНИЙ

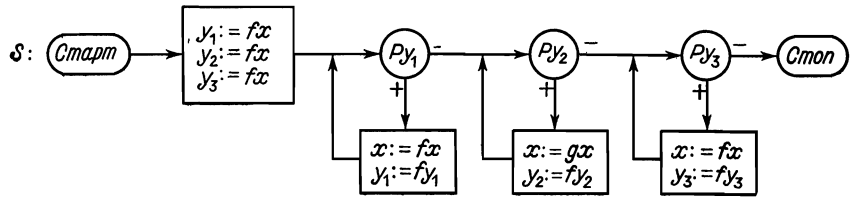
Многие результаты о разрешимости эквивалентности и транслируемости для классов унарных схем можно получить как непосредственное следствие известных результатов теории автоматов и математической лингвистики. Мы будем использовать простой способ сопоставления каждой схеме в данном классе определенного формального языка, называемого ее языком значений; затем мы используем соответствующие факты для полученных таким образом языков значений. Несмотря на то что некоторые результаты этого раздела хорошо известны, мы, однако, приводим здесь их доказательства, чтобы проиллюстрировать силу и легкость применения метода языков значений.

Для всякой унарной схемы S, содержащей функциональные символы из некоторого множества $\mathscr{F}$ и выходную переменную x, *языком значений* $L(S)$ схемы S называется подъязык $\mathscr{F}^*$, содержащий все строки a, такие. что $\mathrm{val}_I(S) = a$ для некоторой свободной интерпретации I. Например, множество $\{a\bar{a}\colon a \in \{f, g\}^*\}$ симметричных слов четной длины является языком значений линейной рекурсивной схемы

$$E\colon \begin{array}{l} F_1x :\, = \textbf{если } Px \textbf{ то } x \textbf{ иначе } F_2x \\ F_2x :\, = \textbf{если } Qx \textbf{ то } fF_1fx \textbf{ иначе } gF_1gx \end{array} \tag{2.1}$$

в то время как $\{f^n g^n f^n\colon n \geqslant 0\}$ является языком значений стандартной схемы

(2.2)



С помощью техники языков значений будет показано, что рекурсивная схема E не является сильно эквивалентной никакой стандартной схеме с одной переменной, а стандартная схема S не является сильно эквивалентной никакой рекурсивной схеме. Наконец, для всякого класса $\mathscr{S}$ схем мы рассмотрим класс

$$L(\mathscr{S}) = \{L(S)\colon S \in \mathscr{S}\}$$

языков значений схем из $\mathscr{S}$ и докажем теорему, которая, несмотря на ее тривиальность, имеет поразительное количество следствий.

2.3. Теорема. *Для всяких схем S и T, если $S \equiv T$, то*

$$L(S) = L(T).$$

Доказательство. Из-за симметрии достаточно показать, что $L(S) \subseteq L(T)$. Пусть $a \in L(S)$. Тогда $ax = \mathrm{val}_I(S)$ для некоторой интерпретации I. Так как $S \equiv T$, то $\mathrm{val}_I(S) = \mathrm{val}_I(T)$, а поэтому $a \in L(T)$.

2.4. Следствие. Для *всяких классов схем* $\mathscr{S}$ и $\mathscr{T}$
(a) *если* $\mathscr{S} \to \mathscr{T}$, *то* $L(\mathscr{S}) \subseteq L(\mathscr{T})$;
(b) *если* $\mathscr{S} \rightleftarrows \mathscr{T}$, *то* $L(\mathscr{S}) = L(\mathscr{T})$.

Для того чтобы применить следствие к схемам примеров (2.1) и (2.2), мы сначала определим $L(\mathscr{S})$ для некоторых интересных классов $\mathscr{S}$ схем.

2.5. Теорема. *Пусть $\mathscr{S}$ — класс всех*
(a) *стандартных схем с одной переменной*;
(b) *стандартных схем*;
(c) *праволинейных рекурсивных схем*;
(d) *линейных рекурсивных схем*;
(e) *рекурсивных схем,*
которые содержат функциональные символы из некоторого множества $\mathscr{F}$.

Тогда $L(\mathscr{S})$ представляет собой класс всех
(a) *регулярных*;
(b) *рекурсивно-перечислимых*;
(c) *регулярных*;
(d) *линейных контекстно-свободных и*
(e) *контекстно-свободных*
подъязыков $\mathscr{F}^*$.

Доказательство. Прямое доказательство (a) предоставлено читателю, так как (a) следует из (c) и взаимной транслируемости стандартных схем с одной переменной и праволинейных рекурсивных схем, установленной в разд. 3. Подобным же образом доказательство (b) отложено до разд. 3. Здесь мы докажем (e) и заметим, что подобным образом доказываются также (c) и (d).

Прежде всего мы покажем, что каждая рекурсивная схема E имеет контекстно-свободный язык значений. Пусть E — рекурсивная схема, содержащая функциональные символы из $\mathscr{F}$ и предикатные символы из $\mathscr{P}$, заданная системой соотношений

$$E(F_1)\colon\ F_i x :=\ \textbf{если}\ P_i x\ \textbf{то}\ \alpha_i x\ \textbf{иначе}\ \beta_i x\quad (1 \leqslant i \leqslant n),$$

где для всякого i, P_i — предикат из $\mathscr{P}$, а α_i, β_i — термы над $(\mathscr{F} \cup \{F_1, \ldots, F_n\})$. В разделе 1 схеме E было сопоставлена контекстно-свободная грамматика для того, чтобы определить

понятие вычисления E. К сожалению, язык, порождаемый этой грамматикой, в общем случае шире, чем язык значений E, поскольку не все выводы в грамматике соответствуют вычислениям схемы E. Мы устраним этот недостаток, построив несколько более сложную грамматику G следующим образом: пусть H — множество всех функций из $\mathscr{P}$ в $\{0, 1\}$. Для каждого i и каждого h из H определим терм $\gamma_{i,h}$ над $(\mathscr{F} \cup \{F_1, \ldots, F_n\})$ следующим образом:

$$\text{пусть } \gamma_{i,h,1} = \begin{cases} \text{если } h(P_i) = 1, \text{ то } \alpha_i, \\ \text{если } h(P_i) = 0, \text{ то } \beta_i, \end{cases}$$

$$\gamma_{i,h,j+1} = \begin{cases} \gamma_{i,h,j}, & \text{если } \gamma_{i,h,j} \neq \gamma F_k \text{ для всех } \gamma, k, \\ \gamma\alpha_k, & \text{если } \gamma_{i,h,j} = \gamma F_k \text{ и } h(P_k) = 1, \\ \gamma\beta_k, & \text{если } \gamma_{i,h,j} = \gamma F_k \text{ и } h(P_k) = 0, \end{cases}$$

и $\gamma_{i,h} = \gamma_{i,h,n}$.

Заметим, что если для некоторых γ и k выполняется $\gamma_{i,h} = \gamma F_k$, то для всякой свободной интерпретации I, такой, что $P_I(x) = h(P)$ для всех $P \in \mathscr{P}$, $\mathrm{val}_I(E(F_i))$ не определено, поскольку вычисление E зацикливается. Пусть теперь G — контекстно-свободная грамматика с терминальными символами из $\mathscr{F}$, нетерминальными символами $F_1, \ldots, F_n$, начальным символом F_1 и продукциями $F_i \to \gamma_{i,h}$ для всех i и h, таких, что $\gamma_{i,h} \neq \gamma F_k$ для всех γ и k. Тогда для цепочки a из $\mathscr{F}^*$ справедливо $F_1 \underset{G}{\overset{*}{\Rightarrow}} a$ тогда и только тогда, когда существует свободная интерпретация I, такая, что $ax = \mathrm{val}_I(E)$ (см. разд. 1). Следовательно, $L(E)$ — контекстно-свободный язык, порожденный грамматикой G.

Пусть, наоборот, G — контекстно-свободная грамматика с терминальными символами из $\mathscr{F}$, нетерминальными символами $F_1, \ldots, F_n$, начальным нетерминалом F_1 и продукциями $F_i \to \alpha_{ij}$, где $i = 1, \ldots, n$ и $j = 1, \ldots, p(i)$ для некоторой функции p. Пусть E — рекурсивная схема с функциональными символами из $\mathscr{F}$ и предикатными символами p_{ij} $(1 \leqslant i \leqslant n, 1 \leqslant j < p(i))$, заданная определяющими соотношениями

$$\begin{cases} F_j^i x := \textbf{если } P_{ij}x \textbf{ то } \alpha_{ij}x \textbf{ иначе } F_{j+1}^i x & (1 \leqslant j < p(i)), \\ F_{p(i)}^i x := \alpha_{ip(i)}x & (1 \leqslant i \leqslant n), \end{cases}$$

где F_1^i совпадает с F_i. Для всякой цепочки a, выводимой в грамматике G, рассмотрим ее правосторонний вывод

$$F_1 \underset{G}{\Rightarrow} \beta_1 F_{f(1)} a_1 \underset{G}{\Rightarrow} \beta_2 F_{f(2)} a_2 \underset{G}{\Rightarrow} \ldots \underset{G}{\Rightarrow} a$$

и определим свободную интерпретацию I, полагая $(P_{ij})_I(a_k x) = 1$ тогда и только тогда, когда продукция $F_i \to \alpha_{ij}$ применялась в выводе $\beta_k F_{f(k)} a_k \underset{G}{\Rightarrow} \beta_{k+1} F_{f(k+1)} a_{k+1}$ (где $\beta_0 F_{f(0)} a_0 = F_1$). Такое определение возможно при условии, что $F_{f(k)} a_k \neq F_{f(k')} a_{k'}$ для всех $k \neq k'$, которое выполняется, если $F_i \overset{*}{\underset{G}{\not\Rightarrow}} \beta F_i$ для любых β и i; последнего всегда можно добиться, например, приведением G к нормальной форме Грейбах. По определению $\mathrm{val}_I(E) = ax$, так что $a \in L(E)$.

С другой стороны, если I — свободная интерпретация, то $\mathrm{val}_I(E)$, очевидно, выводимо в грамматике G. Следовательно, G порождает язык значений схемы E.

Заметим, наконец, что если в двух приведенных выше конструкциях либо E, либо G будет вначале взята линейной или праволинейной, то конструируемая G (или E) также будет линейна или праволинейна соответственно. Отсюда следует (c) и (d).

Как уже указывалось, прямым следствием теоремы 2.5 и следствия 2.4 является следующий результат о нетранслируемости.

(2.6) Следствие.

(a) (де Беккер — Скотт). *Существует рекурсивная схема, которая не является сильно эквивалентной никакой стандартной схеме с одной переменной.*

(b) *Существует стандартная схема, которая не является сильно эквивалентной никакой рекурсивной схеме.*

(c) *Существует рекурсивная схема, которая не является сильно эквивалентной никакой линейной рекурсивной схеме.*

(d) *Существует линейная рекурсивная схема, которая не является сильно эквивалентной никакой праволинейной рекурсивной схеме.*

Доказательство. Линейная рекурсивная схема E примера 2.1 не является сильно эквивалентной никакой стандартной схеме с одной переменной, а также никакой праволинейной схеме, поскольку язык $L(E)$ нерегулярен. Стандартная схема S примера 2.2 не является сильно эквивалентной никакой рекурсивной схеме, поскольку $L(S)$ не является контекстно-свободным. Наконец, существуют рекурсивные схемы, языки значений которых не являются линейными контекстно-свободными (см. пример 4.2).

Утверждение (a) будет усилено в разд. 4 с помощью другого метода. Пример, используемый для (b), показывает также, что, допуская дополнительные переменные в рекурсивных схемах,

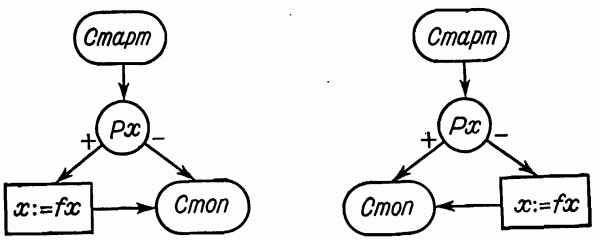
мы увеличиваем «мощность» таких схем: например, простая «бинарная» рекурсивная схема

$$F_1x :=F_2(x,\ x)$$
$$F_2(x,\ y) :=F_3(x)$$
$$F_3x :=\textbf{если } Px \textbf{ то } fF_3fx \textbf{ иначе } F_4y$$
$$F_4y :=\textbf{если } Py \textbf{ то } gF_4fy \textbf{ иначе } y$$

имеет язык значений $\{f^ng^nf^n : n \geqslant 0\}$, который не является контекстно-свободным, так что схема не является сильно эквивалентной никакой схеме dBS.

В добавление к этим следствиям, касающимся транслируемости, техника языков значений имеет также следствия, касающиеся разрешимости проблемы остановки для данного класса схем. Так как схема S останавливается тогда и только тогда, когда $L(S)$ непусто, существуют эффективные процедуры для решения проблемы остановки в классах стандартных схем с одной переменной (Янов [3]) и схем dBS, но не в классе произвольных стандартных схем (Лакхэм, Парк и Патерсон [4]). Это вытекает из того, что проблема пустоты разрешима для контекстно-свободных языков, но неразрешима для рекурсивно-перечислимых языков (Хопкрофт и Ульман [2], стр. 230), а соответствие между схемами и языками, данное теоремой 2.5, является эффективным.

Хотя следствие 2.4 дает необходимое условие для транслируемости схем, оно не дает достаточного условия, поскольку, как будет показано в разд. 4, существует рекурсивная схема, которая не является сильно эквивалентной никакой стандартной схеме, хотя ее язык значений не только рекурсивно перечислим, но и контекстно-свободен. Эта недостаточность обусловлена тем, что теорема, обратная теореме 2.3, неверна; схемы с одним и тем же языком значений не обязаны быть сильно эквивалентными, как это показывает следующий пример:



Обе приведенные выше схемы имеют язык значений $\{\lambda, f\}$, однако не являются сильно эквивалентными, так как они по-разному приходят к своим значениям.

Для определенного класса схем, а именно схем с одной переменной и без константных функций, можно доказать теорему, обратную теореме 2.3 (рассматривая некоторую модификацию техники языков значений, учитывающую не только результат вычислений, но и «историю»).

Пусть S — схема с предикатными символами $P_1, \ldots, P_k$ и функциональными символами из $\mathscr{F}$. Для свободной интерпретации I, при которой схема S останавливается, *интерпретированным значением* $\mathrm{val}_I^{\#}(S)$ схемы S при интерпретации I называется цепочка $f_n p_{n-1} f_{n-1} \ldots f_1 p_0 x$ из $(\mathscr{F} \cup \{0, 1\})^* \mathscr{V}^0$, такая, что $\mathrm{val}_I(S) = f_n f_{n-1} \ldots f_1 x$ и для всех $i < n$ p_i представляет собой цепочку $p_{i1} \ldots p_{ik}$ из k нулей и единиц, такую, что для всех j, $p_{ij} = (P_j)_I(f_i \ldots f_1 x)$. *Языком интерпретированных значений* $L^{\#}(S)$ называется множество всех интерпретированных значений $\mathrm{val}_I^{\#}(S)$ схемы S для свободных интерпретаций I. Интерпретированное значение $f_n p_{n-1} \ldots f_1 p_0 x$ называется *согласованным* со свободной интерпретацией I тогда и только тогда, когда для всех $i < n$ и $1 \leqslant j \leqslant k$ имеем $p_{ij} = (P_j)_I(f_i \ldots f_1 x)$.

Важное свойство схемы S с одной переменной, не содержащей константных функций, заключается в том, что для всякой свободной интерпретации I существует не более одной цепочки (интерпретированного значения) a из $L^{\#}(S)$, согласованной с I, и если такая цепочка существует, то она совпадает с $\mathrm{val}_I^{\#}(S)$. Причиной этого является то, что для всякой свободной интерпретации I и для всякой цепочки $a \in L^{\#}(S)$, согласованной с I, проверки предикатов производятся схемой S во время вычисления при I только над подцепочками a, а следовательно, вычисление схемы S при I должно быть тем же самым, что и при I', где $a = \mathrm{val}_{I'}^{\#}(S)$. Используя это свойство, мы получим следующее частичное обращение теоремы 2.3.

(2.7) Теорема. *Для всех схем S и T с одной переменной и без константных функций $S \equiv T$ тогда и только тогда, когда*

$$L^{\#}(S) = L^{\#}(T).$$

Доказательство. Пусть $S \equiv T$ и I — свободная интерпретация. Тогда $\mathrm{val}_I(S) \simeq \mathrm{val}_I(T)$ и, следовательно, $\mathrm{val}_I^{\#}(S) \simeq \simeq \mathrm{val}_I^{\#}(T)$ по определению $\mathrm{val}_I^{\#}$. Поэтому $L^{\#}(S) = L^{\#}(T)$. Наоборот, пусть $L^{\#}(S) = L^{\#}(T)$ и I — свободная интерпретация. Достаточно показать, что если $\mathrm{val}_I^{\#}(S)$ определено, то $\mathrm{val}_I^{\#}(T)$ также определено и $\mathrm{val}_I^{\#}(S) = \mathrm{val}_I^{\#}(T)$. Так как $L^{\#}(T) = L^{\#}(S)$, то $\mathrm{val}_I^{\#}(S) \in L^{\#}(T)$. Поскольку очевидно, что $\mathrm{val}_I^{\#}(S)$

согласовано с I, то вследствие замечаний, предшествующих теореме,

$$\mathrm{val}_I^{\#}(S) = \mathrm{val}_I^{\#}(T).$$

Отсюда и следует утверждение теоремы.

Теорема 2.7 имеет несколько следствий. Полагая $L^{\#}(\mathscr{S}) = \{L^{\#}(S) : S \in \mathscr{S}\}$ для всякого класса $\mathscr{S}$ схем, мы получим следующие аналоги 2.4 и 2.5.

(2.8) Следствие. *Для всяких классов $\mathscr{S}$ и $\mathscr{T}$ схем с одной переменной и без константных функций*

(a) $\mathscr{S} \to \mathscr{T}$ *тогда и только тогда, когда* $L^{\#}(\mathscr{S}) \subseteq L^{\#}(\mathscr{T})$;

(b) $\mathscr{S} \rightleftarrows \mathscr{T}$ *тогда и только тогда, когда* $L^{\#}(\mathscr{S}) = L^{\#}(\mathscr{T})$.

(2.9) Теорема. *Пусть $\mathscr{S}$ — класс всех*

(a) *стандартных схем с одной переменной или праволинейных рекурсивных схем*;

(b) *линейных рекурсивных схем*;

(c) *рекурсивных схем без константных функций.*

Тогда $L^{\#}(\mathscr{S})$ содержит только

(a) *регулярные*;

(b) *детерминированные линейные контекстно-свободные*;

(c) *детерминированные контекстно-свободные языки.*

Доказательство. Читатель может сам убедиться в том, что учет поведения предикатов схемы в ее интерпретированном значении устраняет недетерминизм грамматики, определенной в доказательстве теоремы 2.3. Например, язык интерпретированных значений рекурсивной схемы

$$E\colon F_i x := \textbf{если } Px \textbf{ то } \alpha_i x \textbf{ иначе } \beta_i x$$

с одним предикатом воспринимается детерминированным автоматом, изображенным на рис. 3 в виде диаграммы, где $\xrightarrow{aA/\alpha}$ означает, что если a — очередной входной символ, а A — символ в верхушке магазина, то происходит указываемый стрелкой переход в другое состояние, а A в стеке заменяется на α (если $a = \lambda$, входная лента не сдвигается). Отметим, что, пока не сделано усиление теоремы 2.9, полностью характеризующее упоминающиеся здесь языки $L^{\#}(\mathscr{S})$, следствие 2.8 является не очень полезным. Тем не менее теорема 2.9 без всякого усиления вообще позволяет сводить некоторые проблемы разрешимости для схем к проблемам разрешимости для языков.

(2.10) Теорема.

(a) (Янов) *Существует эффективная процедура, распознающая, являются ли сильно эквивалентными две стандартные схемы с одной переменной.*

(b) *Проблема эквивалентности для (линейных) рекурсивных схем сводится к проблеме эквивалентности для (линейных) контекстно-свободных языков.*

Доказательство. Так как существует процедура для распознавания, совпадают ли два регулярных языка (Хопкрофт и Ульман [2]), утверждение (a) следует из 2.8 и 2.9. Утверждение (b) также следует из 2.8 и 2.9.

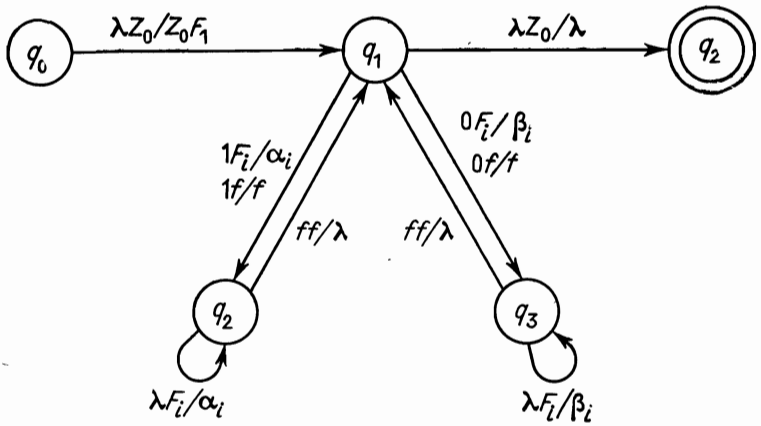


Рис. 3.

Было бы интересно выяснить, верно ли утверждение, обратное (b), так как тогда некоторые нерешенные проблемы формальных языков (например, проблему эквивалентности для линейных контекстно-свободных языков) можно было бы свести к известным проблемам разрешимости для схем (разд. 5).

3. ТРАНСЛЯЦИЯ СХЕМ

В этом разделе мы исследуем проблему трансляции рекурсивных схем в стандартные. В частности, мы покажем (теорема 3.5), что всякая квазирациональная рекурсивная схема транслируема в сильно эквивалентную стандартную схему. При несколько менее сильных условиях, которые мы называем «слабой транслируемостью», всякая рекурсивная схема оказывается слабо транслируемой в стандартную схему. Другими словами, для всякой рекурсивной схемы R существует стандартная схема S с тем же самым языком значений, что и у R, и такая, что всякий раз, когда S останавливается, R также останавливается с тем же самым результатом (т. е. $L(R) = L(S)$ и R есть расширение схемы S). Эти результаты о транслируемости получаются

с помощью ряда простых программистских приемов. Ту же самую технику мы используем для того, чтобы показать, что всякий рекурсивно-перечислимый язык является языком значений некоторой унарной стандартной схемы. Наконец, мы покажем, что подобную технику можно использовать для доказательства утверждения, что всякая схема dBS транслируема в стандартную схему с двумя счетчиками, — результат, который *неверен* для бинарных рекурсивных схем (Патерсон и Хьюитт [6]).

Формальная проверка этих утверждений затемняет содержащиеся в них простые идеи, так что доказательства, приведенные ниже, лишь описывают необходимые конструкции и сопровождаются неформальными аргументами в пользу их правильности.

Следующие примеры содержат по существу всю программистскую технику, которая будет использоваться.

Пусть E — рекурсивная схема примера 2.1. Мы покажем, что E сильно эквивалентна стандартной схеме S, изображенной на рис. 4. Рабочая переменная u «хранит» вычисляемое значение. Блок A схемы S моделирует вычисление E при интерпретации I до тех пор, пока переменной u не будет присвоено такое значение a, что $P_I(a) = 1$. Предположим, что $a = (f_n \ldots f_1 x)_I$. Тогда $P_I((f_i \ldots f_1 x)_I) = 0$ для $1 \leqslant i < n$ и

$$f_i = \begin{cases} f, & \text{если } Q(f_{i-1} \ldots f_1 x) = 1; \\ g & \text{в противном случае.} \end{cases}$$

Поскольку $\mathrm{val}_I(E) = (f_1 \ldots f_n f_n \ldots f_1 x)$, блоки B и C схемы S должны применять $f_n, \ldots, f_1$ к значению переменной u именно в таком порядке. Чтобы понять, как это происходит, рассмотрим ситуацию, когда переменной u присвоено значение $(f_{n-i+1} \ldots f_n f_n \ldots f_1 x)_I$ с некоторым $i < n$, и следующим к u должна применяться функция f_{n-i}. Это делается в блоке B, который «ожидает», когда значение v на его входе станет равным $(f_i \ldots f_1 x)_I$, так что после $n - i$ повторений цикла в B переменная v имеет значение a и проверка Pv выводит из цикла. При этом значение переменной w станет равным $(f_{n-i-1} \ldots f_1 x)_I$ (заметим, что выход происходит до $(n - i)$-го присваивания переменной w). Следовательно, функция f_{n-i} была применена к u, как это и требовалось. Блок C теперь использует w для того, чтобы вновь установить v на следующее значение, ожидаемое блоком B. Происходит $i + 1$ повторений цикла в C, прежде чем w получит значение a, так что на выходе из блока C переменная v имеет значение $(f_{i+1} \ldots f_1 x)_I$, как и требовалось. Наконец, когда мы входим в B с a как со значением переменной v, вычисление заканчивается с u и x, имеющими значение $(f_1 \ldots f_n f_n \ldots f_1 x)_I$.

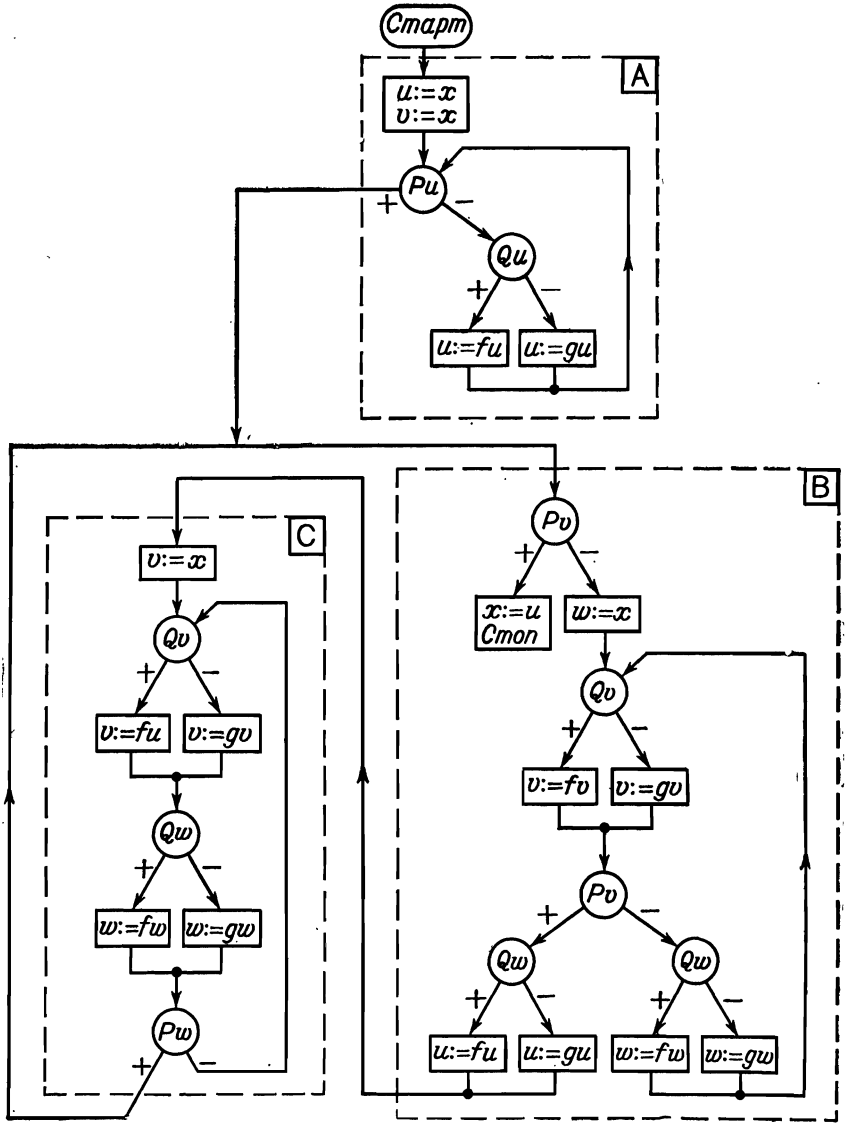

Старт
A
u:=x
v:=x
Pu
+
−
Qu
+
−
u:=fu
u:=gu
B
C
Pv
+
−
x:=u
Cmon
w:=x
Qv
+
−
v:=fv
v:=gv
Pv
+
−
Qw
Qw
+
−
+
−
u:=fu
u:=gu
w:=fw
w:=gw
v:=x
Qv
+
−
v:=fu
v:=gv
Qw
+
−
w:=fw
w:=gw
Pw
+
−

Рис. 4.

Трансляция довольно широкого подкласса рекурсивных схем в сильно эквивалентные стандартные схемы может быть достигнута путем построений, аналогичных приведенным выше. Основная идея, как и выше, — использование дополнительных рабочих переменных для «подсчета», сколько шагов вычисления рекурсивной схемы уже полностью промоделировано.

Мы рассмотрим прежде всего класс линейных рекурсивных схем, представляющих собой схемы E с n определяющими соотношениями следующего вида:

$$E(F_1): \{F_i x := \textbf{если } P_i x \textbf{ то } \alpha_i F_{l(i)} \beta_i x \textbf{ иначе } \gamma_i F_{r(i)} \delta_i x\},\ 1 \leqslant i \leqslant n,$$

где P_i — предикатный символ, а α_i, β_i, γ_i, δ_i — цепочки из базисных функциональных символов. Индексирующие функции l (левая) и r (правая) отображают $\{1, 2, \ldots, n\}$ в $\{0, 1, 2, \ldots, \ldots, n\}$ при соглашении, что F_0 — пустая цепочка, и если $l(i) = 0$, то α_i пусто, а если $r(i) = 0$, то γ_i пусто. Вычисление E будет заканчиваться всякий раз, когда оно достигает терма, «содержащего» F_0, поэтому мы называем эти термы *заключительными точками* E.

Вычисления линейных схем можно просто описать следующим образом. Пусть заключительная точка достигается после m шагов вычисления схемы E при интерпретации I. Пусть f — индексирующая функция, которая говорит нам, какое определяющее соотношение выполнялось на i-м шаге $(1 \leqslant i \leqslant m)$. Очевидно, что $f(1) = 1$, и f можно определить индуктивно в терминах l и r. Тогда существуют цепочки $a = a_m \ldots a_1$ и $b = b_1 \ldots b_{m-1}$, такие, что $\mathrm{val}_I(E) = (bax)_I$, где для всякого i

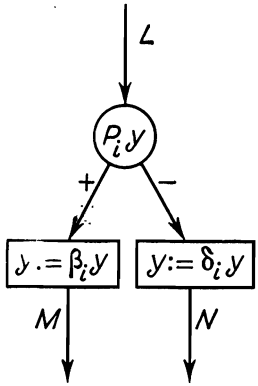
$$a_i = \begin{cases} \beta_{f(i)}, & \text{если } (P_{f(i)})_I(a_{i-1} \ldots a_1 x)_I = 1; \\ \delta_{f(i)} & \text{в противном случае} \end{cases}$$

и

$$b_i = \begin{cases} \alpha_{f(i)}, & \text{если } (P_{f(i)})_I(a_{i-1} \ldots a_1 x)_I = 1; \\ \gamma_{f(i)} & \text{в противном случае.} \end{cases}$$

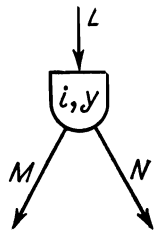
Всякую линейную рекурсивную схему E можно транслировать в сильно эквивалентную стандартную схему $S(E, x)$. Схема S имеет рабочие переменные u, v, w в дополнение к переменной x (которая одновременно является и входной, и выходной) и составлена из трех блоков. Как переменные, так и блоки служат точно для тех же целей, что и в конструкции примера на рис. 4. Блоки составлены из множества элементарных (размеченных) подсхем, которые строятся по определяющим соотношениям схемы E; i-му определяющему соотношению соответствует под-

схема вида



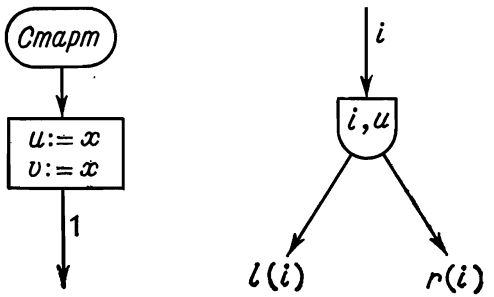
где y — одна из переменных u, v, w, а L, M, N — метки; если либо β_i, либо δ_i — пустая цепочка, соответствующий оператор не включается в подсхему. Метки являются только средством для описания, они облегчают понимание того, каким образом соединяются подсхемы, и частью схем не являются. Когда соединяется множество таких схем, каждый выход указывает на схему, которая имеет ту же самую метку на выходе.

Мы будем изображать приведенную выше подсхему, соответствующую i-му определяющему соотношению, следующим образом:



Первый блок схемы S (блок А) строится соединением следующего множества подсхем (соединения определяются исключительно метками):

$$1 \leq i \leq n$$



Выход из блока A имеет метку 0. Блок изображен на рис. 5; цикл указывает, что соединения (с метками $i > 0$) производятся внутри блока.

Должно быть понятно, что при интерпретации I вычисление достигает выхода, помеченного $i = 0$, тогда и только тогда, когда u имеет значение $(ax)_I$. В результате работы блоков B и C значение u должно стать равным $(bax)_I$. Это достигается точно так же, как и для схемы примера 2.1, однако блоки B и C устроены теперь более сложно, так как они должны «прослеживать» не только состояние вычисления (используя v и w), но «помнить» также определяющие соотношения, которые должны применяться следующими к v и w; дважды индексированные метки блоков B и C предназначены именно для этой цели.

Внутри блока B (рис. 5) выход подсхемы, вычисляющей v, соединяется с подсхемой, вычисляющей w, и наоборот, за тем исключением, что выходы с пометками «*oj*» подсхем, вычисляющих v, разрывают «цикл» в B и ведут к подсхемам, вычисляющим u. B имеет n входных точек $B_1, \ldots, B_n$. Если i-е определяющее соотношение имеет заключительную точку, вход B_i осуществляет проверку $P_i v$ и соответствующий выход ведет к СТОП.

Предположим, что переменной u присвоено значение $(b_{m-k+1} \ldots b_{m-1} a_m a_{m-1} \ldots a_1 x)_I$, так что следующим к u должно применяться b_{m-k}. Блок B «ожидает» входа по метке B_i, где i — следующее определяющее соотношение, которое нужно применить к ожидаемому значению $(a_k \ldots a_1 x)_I$ переменной v. Как и раньше, после $m - k$ повторений «цикла» в B переменная получает значение $(ax)_I$, а w — значение $(a_{m-k-1} \ldots a_1 x)_I$, так что b_{m-k} применяется к u с помощью j-го определяющего соотношения. Блок C (рис. 5) использует теперь w для того, чтобы вновь установить v на следующее значение, ожидаемое блоком B. Происходит $k + 1$ повторений «цикла» в C, прежде чем w получит значение a, так что на выходе из C переменная v будет иметь значение $(a_{k+1} \ldots a_1 x)_I$, как и требовалось. Наконец, если мы входим в B с $(a_m \ldots a_1 x)_I$ как со значением переменной v, вычисление заканчивается после присваивания переменной x значения $(bax)_I$.

Теперь должно быть ясно, что E сильно эквивалентна S. Тем самым мы получим следующую теорему.

(3.1) Теорема. *Всякая линейная рекурсивная схема транслируема в сильно эквивалентную стандартную схему. Существует алгоритм трансляции.*

Заметим, что если E — множество праволинейных определяющих соотношений, так что для каждого i, α_i и γ_i пусты, то блоки B и C схемы S и переменные u, v, w можно опустить. Таким

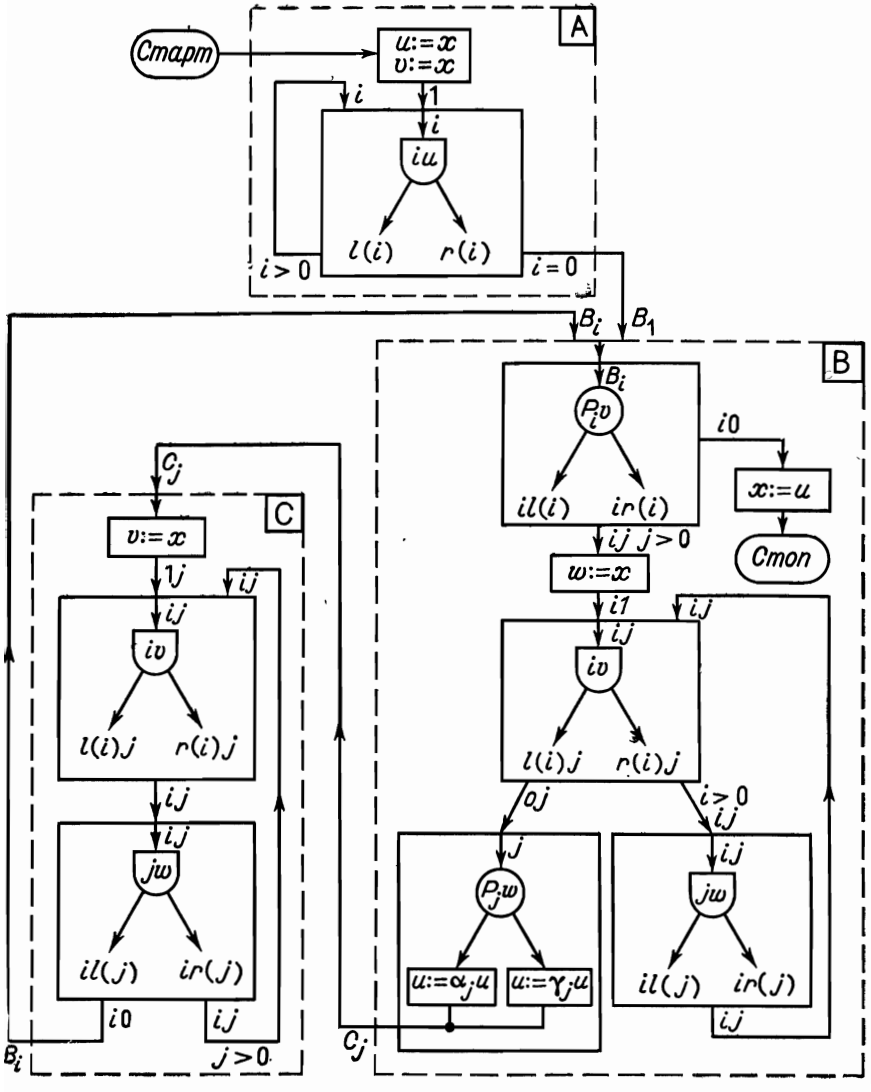

Старт
u:=x
v:=x
A
i
1
i
iu
l(i)
r(i)
i > 0
i = 0
B_i
B_1
B
P_i v
il(i)
ir(i)
i0
x:=u
Стоп
ij j > 0
w:=x
i1
iv
l(i)j
r(i)j
oj
i > 0
P_j w
u:=α_j u
u:=γ_j u
jw
il(j)
ir(j)
C_j
C
v:=x
1j
ij
B_i
j > 0

Рис. 5.

образом, праволинейные рекурсивные схемы транслируемы в сильно эквивалентные стандартные схемы с одной переменной. Обратное следует из стандартного метода трансляции стандартных схем в рекурсивные схемы со многими переменными.

Вышеупомянутая трансляция имеет дальнейшие полезные свойства.

(3.2) Замечание. Для всякой интерпретации I некоторое значение вычисляется во время выполнения схемы $S(E, x)$ при I тогда и только тогда, когда оно вычисляется во время вычисления схемы E при интерпретации I.

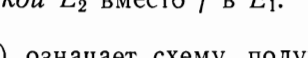
Нам хотелось бы каким-то образом охарактеризовать класс всех тех схем dBS, которые можно транслировать в стандартные схемы [1]). Естественно ожидать, что (3.1) имеет место для простых композиций линейных схем.

(3.3) Определение (функциональной подстановки). Пусть E_1 и E_2 — рекурсивные схемы, не имеющие общих определяемых функциональных символов. Предположим, что начальные определяемые функциональные символы — это F_1 и F_2 соответственно. Пусть E_1' получается заменой всех вхождений базисной функции f в E_1 на F_2. Тогда будем говорить, что рекурсивная схема с определяющими соотношениями $E_1' \cup E_2$ и начальным определяемым функциональным символом F_1 получается *функциональной подстановкой* E_2 вместо f в E_1.

Обозначение: $E_1(f; E_2)$ означает схему, получаемую функциональной подстановкой E_2 вместо f в E_1.

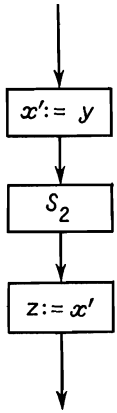
(3.4) Теорема. *Класс всех рекурсивных схем, транслируемых в стандартные схемы с выполнением условия* (3.2), *замкнут относительно функциональной подстановки.*

Доказательство. Предположим, что E_1 и E_2 транслируемы в стандартные схемы S_1 и S_2 соответственно. Можно считать, что эти стандартные схемы не имеют общих переменных, базисная функция f встречается в E_1, но не встречается в E_2 и (3.2) выполняется для обеих пар схем.

Пусть $E' = E_1(f; E_2)$. Мы построим новую стандартную схему S', заменяя все вычисления схемы S_1, содержащие f, на копии схемы S_2. Например, присваивание 

[1]) Для бинарных рекурсивных схем соответствующий класс не будет даже рекурсивно-перечислимым.

заменяется на



где x' — входная переменная схемы S_2, а оператор СТОП в S_2 заменяется на инструкцию «на выход». Пересылки, которые обеспечивают, чтобы S_2 получила нужный аргумент и верно засылала результат, могут быть элиминированы, т. е. существует эквивалентная стандартная схема, которая не содержит пересылок (и условие (3.2) при этом удовлетворяется, теорема 3.7).

Для всякой интерпретации I схем E' и S' определим ее расширение I' так, чтобы

$$f_{I'}(v) = \begin{cases} \mathrm{val}_I(E_2)(v), & \text{если вычисление } E_2 \text{ при } I \text{ с входным значением } v \text{ заканчивается,} \\ \Omega & \text{в противном случае,} \end{cases}$$

где $\Omega \notin \mathrm{dom}_I$ и $\mathrm{dom}_{I'} = \mathrm{dom}_I \cup \{\Omega\}$.

Рассмотрим сначала взаимосвязь между вычислениями S' и E' при интерпретации I и вычислениями схем S_1 и E_1 при интерпретации I'. Эти две пары вычислений идентичны, за исключением той точки, где вычисляется значение $f_{I'}$ в схеме S_1 (соответственно E_1). Если значения $f_{I'}$ отличны от Ω, то эти точки соответствуют завершающимся конечным вычислениям S_2 и E_2, дающим одинаковые результаты. Такие подвычисления не портят значений переменных схемы S_1, так как схемы имеют непересекающиеся множества переменных. Если значение $f_{I'}$ равно Ω, то эта точка соответствует бесконечному вычислению схемы S_2 (или E_2), так что вычисления S' и E' при I зацикливаются.

Далее, наше предположение (3.2) подразумевает, что при интерпретации I' схемы S_1 и E_1 вычисляют одно и то же значение $f_{I'}$. Если все такие подвычисления завершаются, то $\mathrm{val}_I(S') \simeq \mathrm{val}_{I'}(S_1) \simeq \mathrm{val}_{I'}(E_1) \simeq \mathrm{val}_I(E'.)$ Если некоторое подвычисление не заканчивается, то обе схемы S' и E' зацикливаются.

Таким образом, $E' \equiv S'$. Кроме того, легко видеть, что (3.2) сохраняется для этой пары схем.

Алгоритм трансляции для линейных схем можно усилить, используя алгоритм подстановки, приведенный в доказательстве (3.4). Зная как построить данную схему из некоторого конечного множества линейных схем посредством функциональной подстановки, мы можем построить стандартную схему, эквивалентную данной схеме.

Из теоремы 3.4 следует, что некоторые естественные классы рекурсивных схем транслируемы в класс стандартных схем. Таким классом является класс *металинейных* схем: если $E_1(F_1), \ldots, E_n(F_n)$ — линейные схемы с непересекающимися множествами определяемых функций, то схема $E(F)$ с определяющими соотношениями $\{Fx := F_1 \ldots F_n x\} \cup \bigcup_{i=1}^{n} E_i$ металинейна. Более широким является класс *квазирациональных* схем (т. е. таких, которые сопоставляются квазирациональным грамматикам; Нива [5]). Это минимальный класс, содержащий линейные схемы и замкнутый относительно функциональной подстановки.

(3.5) Теорема. *Квазирациональные рекурсивные схемы транслируемы в сильно эквивалентные стандартные схемы.*

Существуют ли неквазирациональные рекурсивные схемы, транслируемые в стандартные? В разд. 4 мы покажем, что не все рекурсивные схемы транслируются в стандартные, однако существует неквазирациональная схема, транслируемая в стандартную схему с одним счетчиком. Это побуждает нас поставить следующий вопрос.

(3.6) Вопрос. Замкнут ли класс всех схем dBS, сильно эквивалентных стандартным схемам с одним счетчиком, относительно функциональной подстановки?

Мы предполагаем, что ответ на этот вопрос *отрицательный.* Если это так, то это означало бы, что класс стандратных схем с одним счетчиком является более мощным, чем класс стандартных схем, что кажется нам верным, хотя мы не можем доказать этого [1]).

(3.7) Теорема. *Для всякой стандартной схемы S с операторами пересылки (т. е. инструкциями вида $x := y$) существует стандартная схема S', в которой все операторы пересылки содержат в правой части только входные переменные (которым*

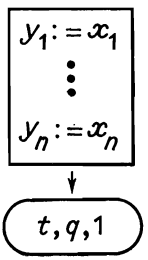
[1]) Ответ на вопрос (3.6) — положительный (см. сноску на стр. 76). — *Прим. перев.*

в схеме ничего не присваивается), *так что* $S \equiv S'$, *и свойство* (3.2) *удовлетворяется.*

Доказательство. Предположим, что S имеет переменные $x_1, \ldots, x_n$ и функции $f_1, \ldots, f_m$. Мы введем новые переменные $y_1, \ldots, y_n$ и $z_1, \ldots, z_n$, которые будут использоваться в S' следующим образом: вычисление значений x_i в S будет реализовано в S' вычислением значений y_i, так что x_i можно использовать как входные переменные, которым ничего не присваивается, в то время как z_i обеспечивают «восстановление» y_i для того, чтобы элиминировать операторы пересылки невходных значений.

Более точно, во всякий момент вычисления схемы S каждой переменной x_i сопоставлено либо входное значение, либо функция f_k и переменная x_j, такие, что текущее значение x_i равно f_k от некоторого предыдущего значения x_j. Имеется только конечное число таких сопоставлений, так что их можно закодировать в S'. Если правильное предыдущее значение x_j хранилось, скажем, в z_l, то оператор пересылки $x_h := x_i$ в S можно заменить в S' на $y_h := f_k z_l$. Пусть T — объединение всех отображений t: $\{1, \ldots, n\} \to \{0, \ldots, m\}$, Q — объединение всех отображений q: $\{1, \ldots, n\} \to \{1, \ldots, n\}$. Мы индексируем инструкции в S' индексами из множества $T \times Q \times I$, где I — индексное множество инструкций из S.

S' начинается засылками



где $t(i) = 0$, $q(i) = i$ для всех i. Остальная часть схемы S' получается соединением инструкций, указанных на рис. 6.

Простые программистские приемы позволяют нам также охарактеризовать класс языков значений стандартных схем. Хотя здесь были приведены некоторые неформальные рассуждения по поводу расширения этого класса, тем не менее он и так уже совпадает с классом всех рекурсивно-перечислимых языков. Этот результат можно считать «неудачей», поскольку он ограничивает возможности применения метода языков значений для доказательства результатов о нетранслируемости (разд. 2).

Тот факт, что мы можем построить стандартную схему, имеющую заданный рекурсивно-перечислимый язык в качестве

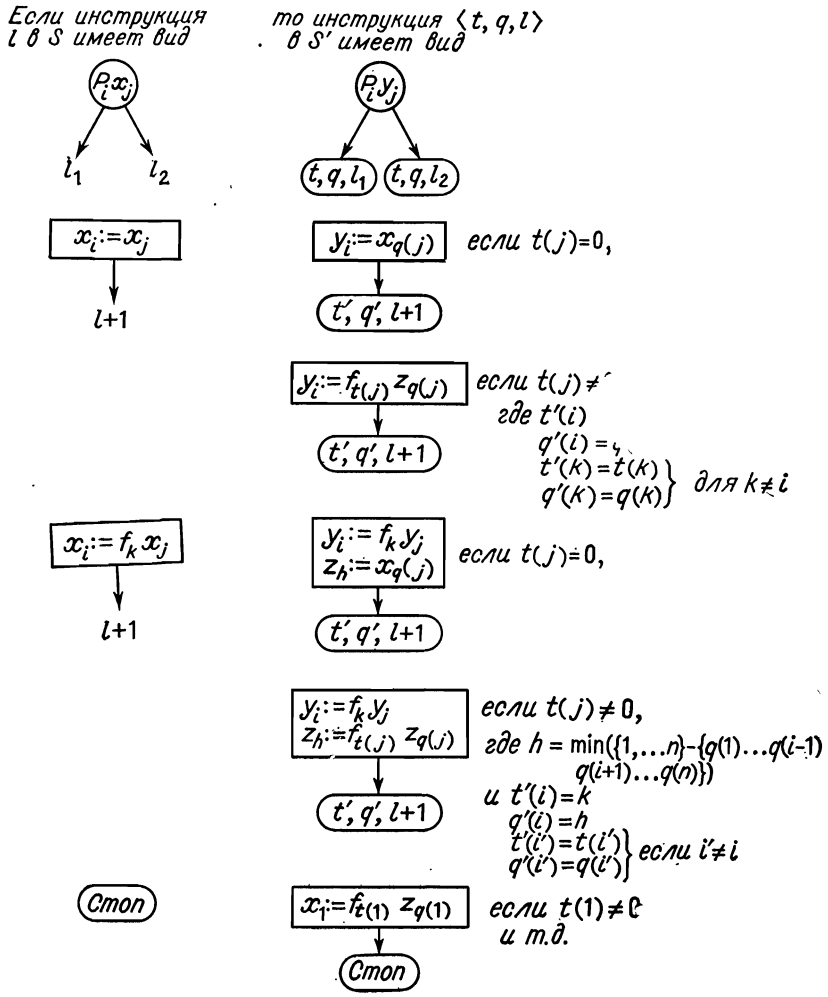


Рис. 6.

ее языка значений, является прямым следствием нашего умения моделировать программу универсальной вычислительной машины поведением стандартных схем. Продемонстрировать это легче всего на примере наиболее простых машин Шепердсона и Стургиса с регистрами [8].

Напомним, что машина с регистрами M имеет конечный набор регистров $x_1, \ldots, x_r$ и конечный список команд вида

(i) $x_i := x_i + 1$, (ii) $x_i := x_i \dot{-} 1$ (где $x \dot{-} 1 = x - 1$, если $x > 0$ и $0 \dot{-} 1 = 0$) и (iii) переход к команде l, если $x_i \neq 0$. Функция f, вычисляемая машиной M, такова, что для всякого n значение $f(n)$ — последнее значение в регистре x_1, когда M останавливается при $x_1 = n$, $x_2 = x_3 = \ldots = x_r = 0$, и $f(n)$ не определено, когда M не останавливается.

(3.8) Теорема. *Класс рекурсивно-перечислимых языков совпадает с классом языков значений унарных стандартных схем.*

Доказательство. Для всякой стандартной схемы T язык $L(T)$ является рекурсивно-перечислимым языком, поскольку множество завершающихся вычислений схемы T при свободных интерпретациях рекурсивно-перечислимо. Поэтому необходимо доказать, что если $\mathscr{F}$ — множество (функциональных) символов, а X — рекурсивно-перечислимое подмножество $\mathscr{F}^*$, то $X = L(T)$ для некоторой стандартной схемы T. Далее, существует машина с регистрами M, такая, что $\alpha \in X$ выполняется для всякой цепочки α тогда и только тогда, когда гёделевский номер [1]) α является значением функции, вычисляемой машиной M. Мы покажем сначала, что машина с регистрами M может «моделироваться» стандартной схемой S. Если M имеет регистры $x_1, \ldots, x_r$, то S будет иметь переменные $x_1, \ldots, x_r, u, v, w$, один функциональный символ f и предикатный символ P. Схема S строится из элементарных схем, моделирующих команды машины M.

Способ, которым S моделирует M, основан на следующем соображении. Для интерпретации I и элемента $\alpha \in \mathrm{dom}_I$ мы скажем, что α является «кодом» целого n, если n — наименьшее целое, такое, что $P_I\left(f_I^n(\alpha)\right) = 1$. *Содержимым* переменной x мы будем здесь называть то целое, код которого является значением этой переменной в данный момент вычисления схемы S при интерпретации I. Заметим, что переменная может иметь произвольное содержимое при интерпретации I только тогда, когда для всякого $\alpha \in \mathrm{dom}_I$ последовательность $P_I(\alpha)$, $P_I\left(f_I(\alpha)\right)$, $P_I\left(f_I^2(\alpha)\right)$, ... содержит как угодно длинные последовательности нулей, разделяемые единицами, т. е. если

$$\forall \alpha \in \mathrm{dom}_I \forall n \exists m \left[P_I\left(f_I^{m+n}(\alpha)\right) = 1 \,\&\, (\forall i < n)\, P_I\left(f_I^{m+i}(\alpha)\right) = 0\right].$$

Такие интерпретации мы называем *хорошими*; например, свободная интерпретация, для которой $P_I f x$, $P_I f^2 x$, ... совпадает с последовательностью 1010010001..., является хорошей интерпретацией. Заметим, что если содержимое переменной x со значением α равно n, то можно «прибавлять единицу» к содержи-

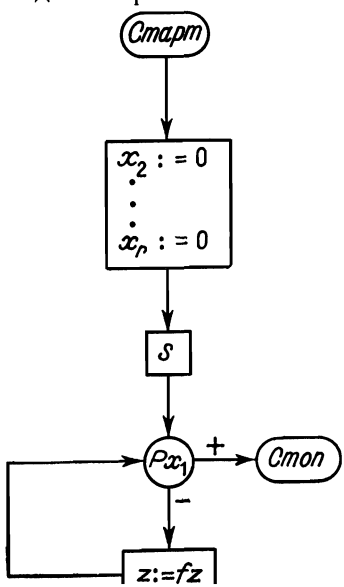
[1]) Мы предполагаем заданной некоторую стандартную вычислимую гёделевскую нумерацию цепочек символов.

мому x при хорошей интерпретации I посредством присваивания $x := f^m x$, где $P_I(f_I^{m+n+1}a)=1$ и $P_I(f_I^{m+i}a)=0$ для всех $i \leqslant n$.

Для каждой из исходных команд машины M существует элементарная схема, такая, что при всякой интерпретации она будет либо выполнять соответствующую операцию (в смысле, указанном выше) над содержимыми ее ячеек, либо зацикливаться, пытаясь выполнить нужные действия. Элементарные схемы приведены на рис. 7 (x, y — различные регистры). Предположение о том, что x и y — различные регистры, несущественно, так как чуть более сложные схемы будут моделировать, например, и команду $x := x + 1$.

Как обычно, S строится по M заменой каждой команды соответствующей схемой и соединением этих схем в S в соответствии с соединением команд машины M. При любой интерпретации и произвольном входном значении схема S либо зацикливается, либо преобразует содержимые своих переменных в точности так же, как M преобразует содержимые своих регистров. При всякой хорошей интерпретации S моделирует M.

По данной схеме S мы построим теперь схему T, такую, что $L(T) = X$. Чтобы сделать это, нам нужно декодировать результат схемы S. Это возможно, поскольку перевод из гёделевского номера в цепочку, ее представляющую, также можно выполнить с помощью машины с регистрами. В простейшем случае, когда $X = \{f^n : n \in \mathrm{range}(M)\}$, $X = L(T)$, где T — стандартная схема с выходной переменной z:



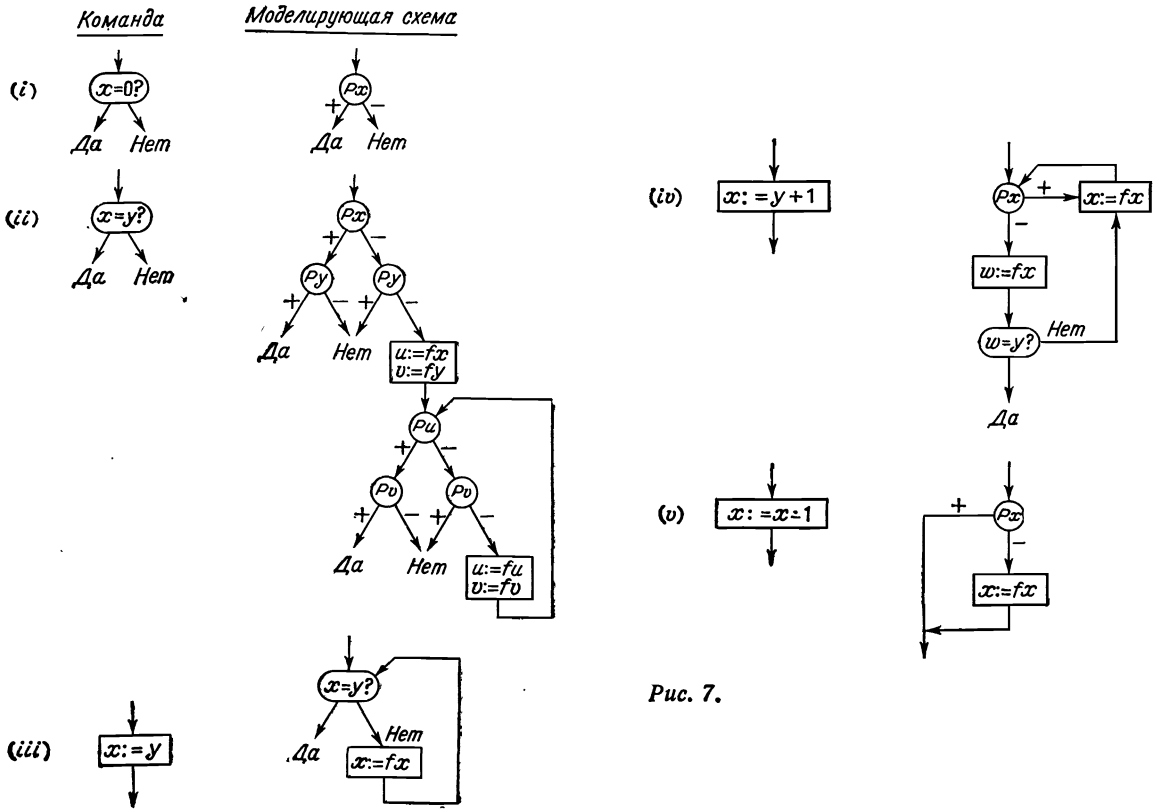

Команда
Моделирующая схема
(i)
x=0?
Да
Нет
Px
+
–
Да
Нет
(ii)
x=y?
Да
Нет
Px
Py
Py
+
–
Да
Нет
u:=fx
v:=fy
Pu
Pv
Pv
Да
Нет
u:=fu
v:=fv
(iii)
x:=y
x=y?
Нет
Да
x:=fx
(iv)
x:=y+1
Px
+
x:=fx
–
w:=fx
w=y?
Нет
Да
(v)
x:=x÷1
Px
+
–
x:=fx

Рис. 7.

Если X — рекурсивно-перечислимый язык над более мощным алфавитом, то несколько более сложное декодирование даст нам X как язык значений стандартной схемы.

(3.9) Замечание. Теорему 3.8 можно усилить за счет сужения класса стандартных схем, необходимых для получения всех рекурсивно-перечислимых языков в качестве языков значений. Всякий рекурсивно-перечислимый язык является фактически языком значений некоторой стандартной схемы с засылками констант (т. е. с вхождениями константных функций), в которой все переменные и константы независимы. То есть схема содержит переменные $x_1, \ldots, x_k$, различные константные функции $a_1, \ldots, a_k$ и унарную функцию f, а все ее присваивания имеют вид $x_i := fx_i$ или $x_i := a_i$. Доказательство мы предоставляем читателю.

Умение моделировать машины с регистрами посредством стандартных схем позволяет нам непосредственно доказывать дальнейшие результаты о транслируемости.

Прежде всего мы немного ослабим условие эквивалентности, требуемой для транслируемости.

(3.10) Определение. Схема S называется *слабой трансляцией* схемы R, если

(i) $L(R) = L(S)$ и

(ii) для всякой интерпретации I, если S останавливается при I, то R останавливается при I с тем же самым результатом.

Таким образом, схема R является (несущественным) расширением схемы S; если R что-нибудь делает, S также будет делать это при некоторой (возможно, другой) интерпретации.

(3.11) Определение. Пусть $\mathscr{R}$ и $\mathscr{S}$ — классы схем. $\mathscr{R}$ называется *слабо транслируемым* в $\mathscr{S}$ (*обозначение* $\mathscr{R} \xrightarrow[w]{} \mathscr{S}$), если $(\forall P \in \mathscr{R})\,(\exists Q \in \mathscr{S})$ [схема Q — слабая трансляция схемы P].

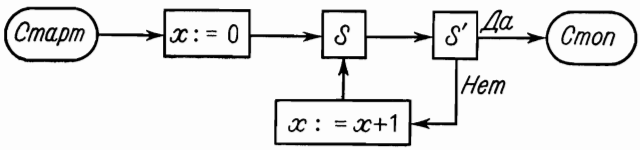
(3.12) Теорема. *Класс схем* dBS *слабо транслируем в класс стандартных схем с одной ячейкой.*

Доказательство. Мы сделаем только набросок доказательства, так как детали подобны приведенным в предыдущих доказательствах.

Пусть R — схема dBS. Тогда $L^{\#}(R)$ — рекурсивно перечислим. Следовательно (как вытекает из доказательства теоремы 3.8), существует стандартная схема S, такая, что содержимыми результата S будут гёделевские номера цепочек w_0, w_1, w_2, ... из $L^{\#}(R)$. Используя S, можно построить стандартную

схему Q, такую, чтобы для всякой интерпретации I схема Q_I порождала и проверяла каждую из цепочек w_i на согласованность с интерпретацией I и останавливалась, когда такая цепочка найдена. Более подробно, выход из схемы S, порождающей коды w_i, является входом в другую схему S'. Напомним, что w_i имеет вид $f_m p_m \ldots f_2 p_2 f_1 p_1$, где f_j — базисная функция схемы R, а p_j — двоичный набор длины n. Вычисление S' при интерпретации I декодирует w_i и проверяет его согласованность с I. Для этого S' проверяет каждый набор p_j в w_i, выполняя последовательно проверки вида $P_k f_j \ldots f_1 x = p_{jk}$? $(1 \leqslant k \leqslant n)$, где p_{jk} обозначает k-й элемент набора p_j, а P_k — k-й предикат схемы R. Заметим, что S' можно построить так, чтобы при хорошей интерпретации она всегда останавливалась. Если все проверки для всех p_j в w_i дают ответ «да», то $\mathrm{val}_I(Q) = (f_m \ldots \ldots f_1 x)_I$; если встречается хотя бы один ответ «нет», Q возвращается к схеме S для порождения w_{i+1}.

Схему Q можно представить следующим образом:



Если Q останавливается при интерпретации I, то R также останавливается при I и $\mathrm{val}_I(Q) = \mathrm{val}_I(R)$ в силу замечания, предшествующего теореме 2.7. Наоборот, если R останавливается при хорошей интерпретации I, то и Q останавливается при I. Так как всякое завершающееся вычисление схемы R выполняется при некоторой хорошей интерпретации, то $L(Q) = L(R)$. Следовательно, схема Q является слабой трансляцией схемы R.

Заметим, что слабые трансляции предыдущей теоремы можно превратить в сильные трансляции, если расширить класс стандартных схем, добавляя предикат P и функцию f с фиксированной хорошей интерпретацией. Если же разрешить использование в стандартных схемах конечного числа счетчиков, то можно обойтись и без таких P и f: поскольку машина с регистрами R представляет собой стандартную схему с конечным числом счетчиков, счетчики можно использовать для моделирования схемы R непосредственно в терминах приведенных выше конструкций. Сколько понадобится счетчиков для того, чтобы транслировать произвольную схему dBS в стандартную схему? Следующая теорема показывает, что двух счетчиков уже достаточно, в то время как результаты разд. 4 показывают, что одного счетчика еще недостаточно.

(3.13) Теорема. *Всякая схема* dBS *сильно эквивалентна унарной стандартной схеме с двумя счетчиками.*

Доказательство. Как описано в разд. 1, вычисление схемы dBS можно наглядно представить с помощью стека функций, которые еще должны быть применены к текущему значению переменной. Теорема легко следует из этого представления и стандартного кодирования стековых символов двумя счетчиками (Хопкрофт и Ульман [2], лемма 6.2 и теорема 6.4).

Следует отметить, что эта теорема оказывается неверной для бинарных рекурсивных схем (Патерсон и Хьюитт [6]), так как стек, «необходимый» для вычислений такой схемы, должен сохранять не только список операций, которые предстоит выполнить, но и вычисляемые при этом значения. В то время как первый стек можно моделировать двумя счетчиками, второй двумя счетчиками не моделируется.

4. ДЛИНА ВЫЧИСЛЕНИЙ

Техника языков значений дала нам возможность установить результаты о нетранслируемости, приведенные в разд. 2. Мы использовали там то соображение, что данная схема S не является сильно эквивалентной никакому элементу данного класса $\mathscr{S}$ схем, так как схемы в $\mathscr{S}$ не настолько «богаты», чтобы вычислять всевозможные значения схемы S. Эта техника оказывается непригодной для получения результатов о нетранслируемости, когда схемы в $\mathscr{S}$ достаточно «богаты», чтобы вычислять всевозможные значения S, но не настолько «богаты», чтобы вычислять нужное значение при той же интерпретации. Что касается рекурсивных схем и стандартных схем, то именно такой случай мы здесь и имеем; как будет видно ниже, описанную в разд. 3 слабую трансляцию рекурсивных схем в стандартные схемы нельзя улучшить с тем, чтобы получить трансляцию, сохраняющую сильную эквивалентность.

Интуитивно нетранслируемость рекурсивных схем в стандартные схемы объясняется тем, что вычисления рекурсивных схем могут быть гораздо более сложными, чем вычисления стандартных схем. Однако эта интуиция ничего не доказывает, поскольку понятие сильной эквивалентности основано только на значениях схем, а не на том, как эти значения получаются. Тем не менее интуитивное понимание того, что вычисления некоторых рекурсивных схем «слишком длинны» и их поэтому нельзя транслировать в стандартные схемы, можно формализовать с тем, чтобы показать существование схемы dBS, которая не является сильно эквивалентной никакой стандартной схеме или даже никакой стандартной схеме с одним счетчиком.

Как можно измерить длину вычисления и использовать ее для установления факта нетранслируемости схемы S в класс схем $\mathscr{S}$? Так как транслируемость имеет дело со значениями схем при данной интерпретации, поступим так: будем записывать длину вычисления схемы S при подходящих интерпретациях I в $\mathrm{val}_I(S)$. Тогда можно будет показать, что для всякой схемы T из $\mathscr{S}$ выполняется соотношение $\mathrm{val}_I(S) \neq \mathrm{val}_I(T)$ для тех интерпретаций I, при которых S имеет «слишком длинные» вычисления, чтобы T могла их выполнить.

Более точно, пусть $\mathscr{F}$ — множество функциональных символов, а $\mathscr{P}$ — множество предикатных символов. Для всякого $n > 0$ интерпретация I множеств $\mathscr{F}$ и $\mathscr{P}$ называется *подсчитывающей интерпретацией ранга* n тогда и только тогда, когда для некоторого множества A, состоящего из n символов, и некоторого символа b:

(a) $\mathrm{dom}_I = \{ab^m\colon a \in A \,\&\, m \geqslant 0\}$;

(b) для всякого $f \in \mathscr{F}$ существует функция f' из A в A и константа $k \leqslant 1$, такая, что $f_I(ab^m) = f'(a)\,b^{m+k}$ для всех $a \in A$ и любого m;

(c) $P_I(ab^m) = P_I(ab^{m'})$ для всякого $P \in \mathscr{P}$, всех $a \in A$ и любых m, m';

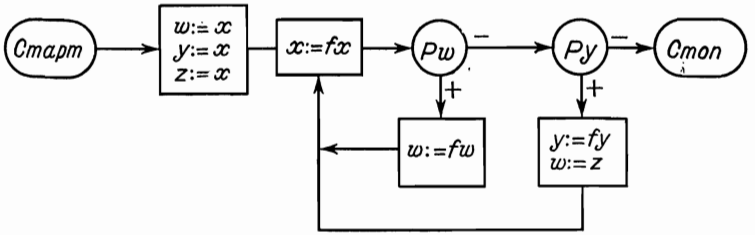
(d) $x_I = a_1$ для всякого $x \in \mathscr{V}$.

Другими словами, при подсчитывающей интерпретации ранга n, значение функции или предиката зависит только от первого символа цепочки, к которой она (он) применяется, так что в области dom_I могут различаться самое большее n объектов; однако длину вычисления можно представить длиной результата.

Для всякой схемы S и всякой подсчитывающей интерпретации I, при которой S останавливается, положим длину $\mathrm{len}_I(S)$ вычисления схемы S при интерпретации I равной тому единственному m, для которого $\mathrm{val}_I(S) = ab^m$ (с некоторым $a \in A$); для всякого $n > 0$ определим $\mathrm{len}(S, n)$ равным максимальному значению $\mathrm{len}_I(S)$ по всем подсчитывающим интерпретациям I ранга n ($\mathrm{len}(S, n)$ всегда определено, так как существует только конечное число подсчитывающих интерпретаций ранга n). Очевидно, что необходимым условием сильной эквивалентности двух схем S и T является $\mathrm{len}(S, n) = \mathrm{len}(T, n)$ для всех n (если это не очевидно, заметьте, что если $\mathrm{len}(S, n) > \mathrm{len}(T, n)$, то $\mathrm{len}_I(S) = \mathrm{len}(S, n) > \mathrm{len}(T, n) \geqslant \mathrm{len}_I(T)$ для некоторой подсчитывающей интерпретации I ранга n, так что $\mathrm{val}_I(S) \neq \mathrm{val}_I(T)$).

Несколько примеров могут помочь прояснить эти определения и предполагаемое их применение. Стандартная схема S, заданная диаграммой

(4.1)



имеет язык значений $\{f^{n^2} : n \geqslant 0\}$ и, следовательно, не является сильно эквивалентной никакой стандартной схеме с одной переменной, так как ее язык значений нерегулярен (на самом деле она не сильно эквивалентна никакой рекурсивной схеме, поскольку ее язык значений не является контекстно-свободным, но это нас здесь не интересует). Другое доказательство этого результата можно дать следующим образом. Для всякого $n > 0$ пусть I — подсчитывающая интерпретация ранга n, такая, что

$$\begin{aligned}
\mathrm{dom}_I &= \{a_i b^m : 1 \leqslant i \leqslant n \,\&\, m \geqslant 0\},\\
f_I(a_i b^m) &= \begin{cases} a_{i+1} b^{m+1}, & \text{если } 1 \leqslant i < n,\\ a_i b^{m+1}, & \text{если } i = n, \end{cases} \qquad (*)\\
P_I(a_i b^m) &= 1 \text{ тогда и только тогда, когда } 1 \leqslant i < n,\\
x_I &= a_1.
\end{aligned}$$

Тогда $\mathrm{val}_I(S) = a_n b^{n^2}$, так что $\mathrm{len}(S, n) \geqslant \mathrm{len}_I(S) \geqslant n^2$. С другой стороны, как мы покажем ниже, для всякой стандартной схемы с одной переменной T $\mathrm{len}(T, n)$ ограничена некоторой линейной функцией kn. Таким образом, для всех $n > k$ и всякой подсчитывающей интерпретации I ранга n, удовлетворяющей (*), $\mathrm{len}_I(T) \leqslant kn < n^2 = \mathrm{len}_I(S)$, так что S не сильно эквивалентна никакой стандартной схеме с одной переменной.

Более интересным примером является рекурсивная схема

(4.2) $\qquad E : Fx :=$ **если** Px **то** fx **иначе** $gFFhx$

Для всякого $n > 0$ пусть I — подсчитывающая интерпретация ранга n, такая, что

$$\begin{aligned}
\mathrm{dom}_I &= \{a_i b^m : 1 \leqslant i \leqslant n \,\&\, m \geqslant 0\},\\
f_I(a_i b^m) &= a_i b^{m+1},\\
g_I(a_i b^m) &= \begin{cases} a_{i-1} b^m, & \text{если } i > 1,\\ a_i b^m, & \text{если } i = 1, \end{cases}\\
h_I(a_i b^m) &= \begin{cases} a_{i+1} b^m, & \text{если } i < n,\\ a_i b^m, & \text{если } i = n, \end{cases}\\
P_I(a_i b^m) &= 1 \text{ тогда и только тогда, когда } i = n,\\
x_I &= a_1.
\end{aligned}$$

Тогда $\mathrm{val}_I(E) = a_1 b^{2^{n-1}}$ (поскольку $F_I(a_{n-i}b^m) = a_{n-i}b^{m+2^i}$, как можно показать индукцией по i), так что $\mathrm{len}(E, n) \geqslant \mathrm{len}_I(E) \geqslant 2^{n-1}$. Как будет показано позже, отсюда следует, что E не является сильно эквивалентной никакой стандартной схеме и даже никакой стандартной схеме с одним счетчиком, так как для такой схемы T $\mathrm{len}(T, n)$ ограничена некоторой конечной степенью n^k числа n, что для достаточно больших n меньше чем 2^n. То же самое можно сказать и о схеме

$$Fx := \textbf{если } Px \textbf{ то } x \textbf{ иначе } gFFhx$$

Границы $\mathrm{len}(T, n)$ для различных схем T устанавливаются следующей теоремой.

(4.3) Теорема. *Для всякой схемы T существует константа $k > 0$, такая, что для всякого $n > 1$*

(a) *если T — стандартная схема с одной переменной, то* $\mathrm{len}(T, n) \leqslant kn$;

(b) *если T — стандартная схема, то* $\mathrm{len}(T, n) \leqslant n^k$;

(c) *если T — стандартная схема с одним счетчиком, то* $\mathrm{len}(T, n) \leqslant n^k$;

(d) *если T — схема* dBS, *то* $\mathrm{len}(T, n) \leqslant k^n$.

Доказательство. Для доказательства утверждений (a) и (b) предположим, что T — стандартная схема с i инструкциями и ν переменными, а I — подсчитывающая интерпретация ранга n, при которой T останавливается. *Состоянием* вычисления схемы T при интерпретации I называется кортеж $\langle j, a_{n_1}, \ldots, a_{n_\nu}\rangle$, где j — номер выполняемой инструкции, и для каждого l текущее значение переменной x_l равняется $a_{n_l}b^{m_l}$ с некоторым целым m_l. Так как T останавливается при I, ни одно состояние в вычислении схемы T не может повторяться; следовательно, существует самое большее $i \cdot n^\nu$ шагов (т. е. выполняемых инструкций) в вычислении T (что будет числом различных состояний) и, таким образом, $\mathrm{len}_I(T) \leqslant i \cdot n^\nu$, поскольку каждая инструкция добавляет не более одного символа b к результату схемы. Так как I — произвольная подсчитывающая интерпретация ранга n, $\mathrm{len}(T, n) \leqslant i \cdot n^\nu$. Утверждение (a) следует отсюда непосредственно, (b) получается выбором достаточно большого k, такого, что $2^{k-\nu} \geqslant i$.

Для доказательства (c) возьмем i и ν, как выше. Пусть I — подсчитывающая интерпретация ранга n, при которой T останавливается. Прежде всего мы покажем, что максимальное значение счетчика, достигаемое во время вычисления T при I, не превосходит $i \cdot n^\nu$. Предположим, что это не так. Пусть счетчик получает свое максимальное значение $m > i \cdot n^\nu$ на некотором шаге s_m вычисления. Пусть s_0 — последний шаг перед s_m,

когда значение счетчика было нулем, и для каждого j между 0 и m пусть s_j — последний шаг перед s_m, такой, что счетчик получает значение j. Так как $m > i \cdot n^v$, должны существовать два таких значения j и j' счетчика, что $0 < j < j' \leqslant m$ и T находится в одном и том же состоянии (как оно определено выше, не принимая во внимание значение счетчика) на шагах s_j и s_j Но тогда схема T должна зацикливаться, поскольку после шага s_j счетчик никогда не обнуляется, и для всякого числа s шагов, на шаге $s_j + s$ схема T должна быть в том же состоянии, что и на шаге $s_{j'} + s$ (за тем исключением, что значение ее счетчика будет на $j' - j$ больше, чем раньше). Таким образом, учитывая значение счетчика в состоянии вычисления схемы, мы видим, что T имеет не более $(i \cdot n^v)^2$ состояний, так что $\mathrm{len}_I(T) \leqslant (i \times \times n^v)^2$. Так как I — произвольная подсчитывающая интерпретация ранга n, то $\mathrm{len}(T, n) \leqslant i^2 \cdot n^{2v}$, и (c) следует из того, что мы можем взять k достаточно большим, так чтобы $2^{k-2v} \geqslant i^2$.

Для доказательства (d) предположим, что T — рекурсивная схема с r рекурсивными определениями, q функциональными символами и термами длины не более l. Пусть I — подсчитывающая интерпретация ранга n. Представим себе вычисление T как работу со стеком, как это было описано в разд. 1.

Как сильно может расти стек во время вычисления? Пусть m — максимальная высота стека, и для каждого $j \leqslant m$ пусть s_j — последний шаг в вычислении, когда стек переходит от высоты, меньшей, чем j, к высоте j или больше (до того, как достигается высота m). Заметим, что на шаге s_j высота стека не превосходит $j + l$ и что между шагами s_j и s_m высота стека никогда не опускается ниже j. Таким образом, если на двух шагах s_j и $s_{j'}$ (при $j < j'$) последние l символов стека попарно совпадают, текущее значение переменной начинается одним и тем же символом a_i и вычисляется одно и то же определяющее соотношение, то схема T будет зацикливаться. Следовательно, $m < \\ < (r + q)^l \cdot n \cdot r$.

Если дана такая граница высоты стека, то T имеет не более $n \cdot r \cdot (r+q)^{(r+q)^l \cdot n \cdot r}$ состояний при интерпретации I, ни одно из которых не повторяется при завершающемся вычислении. Таким образом, $\mathrm{len}_I(T) \leqslant k^n$ для всякого достаточно большого k.

(4.4) Следствие. Существует рекурсивная схема, которая не является сильно эквивалентной никакой стандартной схеме и никакой стандартной схеме с одним счетчиком.

Доказательство. Для рекурсивной схемы E примера (4.2) $\mathrm{len}(E, n) = 2^n$, так что следствие вытекает из утверждений (b) и (c) теоремы, поскольку $2^n > n^k$ для всякого k и для всех достаточно больших n.

Мы заметим, что ни теорему, ни следствие из нее нельзя распространить на стандартные схемы с двумя счетчиками. Во-первых, существуют стандартные схемы T с двумя счетчиками, для которых $\mathrm{len}(T, n)$ не ограничено никакой рекурсивной функцией от n, — факт, который следует из того, что не существует рекурсивной границы для длины вычисления универсальной машины с двумя регистрами. Во-вторых, как показано в разд. 3, всякая схема dBS сильно эквивалентна некоторой стандартной схеме с двумя счетчиками. Приведенный пример 4.2 почти оптимален, так как всякая линейная рекурсивная схема сильно эквивалентна некоторой стандартной схеме. Тем не менее мы не знаем, является ли рекурсивная схема

$$Fx := \textbf{если } Px \textbf{ то } fx \textbf{ иначе } FFfx$$

которая сильно эквивалентна стандартной схеме с одним счетчиком (разд. 1) также сильно эквивалентной стандартной схеме без счетчиков. Мы подозреваем, что это не так, хотя пока не умеем доказывать, что существует стандартная схема с одним счетчиком, которая не является сильно эквивалентной никакой стандартной схеме без счетчиков [1]).

5. РАЗРЕШИМЫЕ СЛУЧАИ ПРОБЛЕМЫ ЭКВИВАЛЕНТНОСТИ

Насколько мощным должен быть класс схем для того, чтобы проблема сильной эквивалентности схем данного класса была алгоритмически неразрешима? Для простых классов схем (например, стандартных схем с одной переменной) проблема эквивалентности разрешима (Янов [3]), в то время как для более сложных классов (например, для стандартных схем) эта проблема неразрешима (Лакхэм и др. [4]); для других классов (например, рекурсивных схем) статус проблемы эквивалентности неизвестен (де Беккер и Скотт [1]). В этом разделе мы собираемся ответить на вопросы, поставленные де Беккером и Скоттом. Именно, мы покажем, что проблема эквивалентности для класса линейных рекурсивных схем разрешима. Далее, мы исследуем, насколько широкие расширения класса стандартных схем с одной переменной обладают разрешимой проблемой эквивалентности. В частности, мы покажем, что добавление к таким схемам засылок констант не влияет на разрешимость проблемы эквивалентности.

Метод, используемый здесь, лучше всего представлять себе как метод «следов»: мы покажем, что если две схемы в данном классе дают различные результаты при некоторой интерпретации I, но работают достаточно долго, то можно немного моди-

[1]) См. сноску на стр. 76. — *Прим. перев.*

фицировать интерпретацию I так, чтобы обе схемы перескакивали через часть своих предыдущих вычислений («разрезание и склеивание») и все еще давали различные результаты; тем самым мы получаем разрешающую процедуру, поскольку вопрос об эквивалентности двух схем зависит теперь от их значений не при *всех*, а только при конечном числе «ограниченных» интерпретаций. Исторически эта техника восходит к Рабину и Скотту [7], которые использовали ее при доказательстве разрешимости проблемы эквивалентности конечных автоматов.

(5.1) Теорема. *Существует эффективная процедура для распознавания, являются ли две линейные рекурсивные схемы сильно эквивалентными.*

Доказательство. В силу леммы 1.2 достаточно показать, что алгоритмически разрешима следующая проблема: для двух данных произвольных линейных рекурсивных схем R и S узнать, существует ли свободная интерпретация, которой они различаются. А для этого достаточно получить эффективную границу длины кратчайшего результата одной из схем для тех свободных интерпретаций, при которых другая схема либо зацикливается, либо дает другой результат. Если такая граница дана, то сильную эквивалентность схем R и S можно распознать, вычисляя значения обеих схем при конечном числе конечных интерпретаций, области dom_I которых состоят из всех цепочек функциональных символов длины не большей, чем эта граница.

Предположим, что R и S имеют не более e рекурсивных определений и что каждый терм в определении имеет не более l символов. Предположим также, что I — свободная интерпретация, при которой R и S различаются, и что минимум n длин термов $\mathrm{val}_I(R)$ и $\mathrm{val}_I(S)$ (по крайней мере один из которых определен) является минимальным из возможных. Мы покажем, что $n < 3e^3l$. Рассмотрим сначала тот случай, когда обе схемы R и S останавливаются при интерпретации I.

Из-за симметрии мы можем предполагать, что

$$\mathrm{val}_I(R) = w = f_n \ldots f_1,$$
$$\mathrm{val}_I(S) = w' = f'_{n'} \ldots f'_1$$

и $n < n'$, или $f_i = f'_i$, для некоторого m, $0 < i < m$, но $f_m \neq f'_m$. Предполагая $n \geqslant 3e^3l$, мы покажем, что можно определить новую интерпретацию, при которой R и S дают различные результаты, более короткие, чем w и w', и составленные из «частей» w и w'.

Поскольку обе схемы R и S останавливаются при I, каждая из них может применить ко входному значению не более e опре-

делений, прежде чем изменить его, т. е. в вычислении w и w' нет последовательности

$$u_0F_{i_0}v \Rightarrow u_1F_{i_1}v \Rightarrow \ldots \Rightarrow u_eF_{i_e}v,$$

в которой e определений выполняются без изменения аргумента, к которому применяются определяемые функции. Так как каждый терм в определении имеет не более l символов, при выполнении e определений к результату может добавиться не более el символов. Следовательно, для того, чтобы выработать результаты w и w' с длиной $\geqslant 3e^3l$, обе схемы R и S должны изменять свое входное значение по крайней мере $3e^2$ раз. Не уменьшая общности, мы можем предполагать, что каждый раз, когда входное значение изменяется, его длина возрастает ровно на один символ; такое предположение правомочно, поскольку, например, определение $F := \alpha F' f_1 \ldots f_i$ можно заменить определениями $F := F^i f_i$, $F^i := F^{i-1} f_{i-1}$, ..., $F^2 := \alpha F' f_1$, где F^i, ..., F^2 — новые определяемые функции.

Для каждого $i < 3e^2$ пусть E_i — множество всех пар $\langle F_j^R, F_k^S \rangle$ определений, которые R и S применяют при вычислении w и w' ко входным значениям длины i. Так как имеется не более e^2 пар определений, существуют определяемые функциональные символы F_j^R и F_k^S и целые числа $i_1 < i_2 < i_3 < 3e^2$, такие, что $\langle F_j^R, F_k^S \rangle \in E_{i_1} \cap E_{i_2} \cap E_{i_3}$. Другими словами, вычисления результатов w и w' должны проходить через состояния

$$\begin{aligned}
F_1^R &\overset{*}{\Rightarrow} c_1F_j^Ra_1 & F_1^S &\overset{*}{\Rightarrow} c_1'F_k^Sa_1' \\
&\overset{*}{\Rightarrow} c_1c_2F_j^Ra_2a_1 & &\overset{*}{\Rightarrow} c_1'c_2'F_k^Sa_2'a_1' \\
&\overset{*}{\Rightarrow} c_1c_2c_3F_j^Ra_3a_2a_1 & &\overset{*}{\Rightarrow} c_1'c_2'c_3'F_k^Sa_3'a_2'a_1' \\
&\overset{*}{\Rightarrow} c_1c_2c_3ba_3a_2a_1 = w & &\overset{*}{\Rightarrow} c_1'c_2'c_3'b'a_3'a_2'a_1' = w',
\end{aligned}$$

где

$$\begin{aligned}
|a_1| &= |a_1'| = i, \\
|a_2| &= |a_2'| = i_2 - i_1 > 0 \quad \text{и} \\
|a_3| &= |a_3'| = i_3 - i_2 > 0.
\end{aligned}$$

Остаток доказательства в этом случае распадается на подслучаи, в каждом из которых мы покажем, что вопреки нашему предположению w и w' не являются кратчайшими различными результатами схем R и S. В первых пяти подслучаях мы предполагаем, что w и w' различаются в первый раз на символах

f_m и f'_m, и показываем, как получить более короткие результаты, которые все еще различаются; в последнем подслучае мы делаем то же самое, предполагая $n < n'$.

Подслучай 1. $f_m \in a_1$, $f'_m \in a'_1$.

Определим по I новую свободную интерпретацию I_1, полагая для всякого предиката P, встречающегося в R и S:

$$P_{I_1}(x) = \begin{cases} P_I(ya_2a_1), & \text{если } x = ya_1, \\ P_I(ya'_2a'_1), & \text{если } x = ya'_1, \\ P_I(x) & \text{в противном случае.} \end{cases}$$

Тогда при интерпретации I_1 схема R вырабатывает результат $u_1 = c_1c_3ba_3a_1$, а S вырабатывает результат $u'_1 = c'_1c'_3b'a'_3a'_1$. Но $u_1 \neq u'_1$, поскольку $a_1 \neq a'_1$ и $|u_1| < |w|$, что противоречит выбору I.

Подслучай 2. $f_m \in a_2$, $f'_m \in a'_2$.

Определим по I новую свободную интерпретацию I_2, полагая

$$P_{I_2}(x) = \begin{cases} P_I(ya_3a_2a_1), & \text{если } x = ya_2a_1, \\ P_I(ya'_3a'_2a'_1), & \text{если } x = ya'_2a'_1, \\ P_I(x) & \text{в противном случае.} \end{cases}$$

Тогда при интерпретации I_2 схема R вырабатывает результат $u_2 = c_1c_2ba_2a_1$, а S вырабатывает результат $u'_2 = c'_1c'_2b'a'_2a'_1$. Снова $u_2 \neq u'_2$, поскольку $a_2a_1 \neq a'_2a'_1$ и $|u_2| < |w|$, что противоречит выбору I.

Подслучай 3. $f_m \in c_3ba_3$, $f'_m \in c'_3b'a'_3$.

Так как $a_1 = a'_1$ и $a_2 = a'_2$, то, как и в подслучае 1, $u_1 \neq u'_1$ при интерпретации I_1 и $|u_1| < |w|$, что противоречит выбору I.

Подслучай 4. $f_m \in c_3b$, $f'_m \in c'_1c'_2$ или $f_m \in c_1c_2$, $f'_m \in c'_3b'$.

Благодаря симметрии достаточно рассмотреть первую из альтернатив. Так как $a_1 = a'_1$, $a_2 = a'_2$ и $a_3 = a'_3$, определены интерпретации I_1 и I_2, а также интерпретация I_3, где

$$P_{I_3}(x) = \begin{cases} P_I(ya_3a_2a_1), & \text{если } x = ya_1, \\ P_I(x) & \text{в противном случае,} \end{cases}$$

При интерпретации I_3 схема R вырабатывает результат $u_3 = c_1 b a_1$, а S вырабатывает результат $u'_3 = c'_1 b' a'_1$. Поскольку все значения u_1, u_2 и u_3 короче, чем w, выбор I подразумевает, что $u_1 = u'_1$, $u_2 = u'_2$ и $u_3 = u'_3$. Так как равные цепочки должны иметь одну и ту же длину, отсюда следует, что $|c_1 b| = |c'_1 b'|$, $|c_2| = |c'_2|$ и $|c_3| = |c'_3|$. Следовательно, $n = n'$.

Рассмотрим теперь другую последовательность равенств $u_i = u'_i$. Так как $f_m \in c_3 b$, то f_m является $(m - |a_2|)$-м символом в u_1 и должен совпадать с $(m - |a_2|)$-м символом в u'_1; этим символом является $f'_{m+|c'_2|}$, поскольку $f'_m \in c'_1 c'_2$ и, следовательно, $f'_{m+|c'_2|} \in c'_1$. Далее, $f'_{m+|c'_2|}$ — это $(m - |a'_3 a'_2| - |c'_3|)$-й символ в u'_3, в то время как соответствующий символ в u_3 — это $f_{m-|c_3|}$, поскольку $f'_{m-|c'_3|} \in a'_3 a'_2 a'_1$, $|c'_3| = |c_3|$, и, следовательно, $f_{m-|c_3|} \in b$. Таким образом, $f_m = f'_{m+|c'_2|} = f_{m-|c_3|}$. Наконец, $f_{m-|c_3|}$ — $(m - |a_3| - |c_3|)$-й символ в u_2, в то время как соответствующий символ в u'_2 — это f'_m, поскольку $f'_m \in c'_1 c'_2$. Таким образом, $f_m = f'_m$, что является противоречием.

Подслучай 5. $f_m \in c_1 c_2$, $f'_m \in c'_1 c'_2$.

Так как $a_2 a_1 = a'_2 a'_1$ и $a_3 = a'_3$, то, как в подслучае 2, $u_2 \neq u'_2$ при интерпретации I_2 и $|u_2| < |w|$, что противоречит выбору I.

Подслучай 6. $n < n'$ и $f_i = f'_i$ для $0 < i \leqslant n$.

Как в подслучае 4, $n = n'$, что является противоречием. Таким образом, w и w' не могут быть кратчайшими различными результатами схем R и S, если оба имеют длину $\geqslant 3e^3 l$. Это завершает доказательство для того случая, когда оба значения $\mathrm{val}_I(R)$ и $\mathrm{val}_I(S)$ определены.

Предположим теперь, что, скажем, $\mathrm{val}_I(S)$ не определено. Доказательство по существу то же самое, что и выше, однако со следующими модификациями. Предположим, что $\mathrm{val}_I(R) = w = f_n \ldots f_1$ и что $n \geqslant 3e^3 l$. Мы покажем, что $\mathrm{val}_{I'}(R)$ для некоторых интерпретаций I' имеет длину, меньшую чем n, в то время как $\mathrm{val}_{I'}(S)$ все еще не определено. Для каждого $i < 3e^2$ пусть E_i — множество всех пар $\langle F_j^R, F_k^S \rangle$ определений, которые схемы S и R применяют при интерпретации I к аргументам длины i, или (если S не применяет никаких определений к аргументам длины i) те, которые S применяет к аргументам максимальной длины. Как и выше, вычисления схем R и S при интер-

претации I должны проходить через состояния:

$$F_1^R \overset{*}{\Rightarrow} c_1 F_j^R a_1 \qquad F_1^S \overset{*}{\Rightarrow} c_1' F_k^S a_1'$$
$$\overset{*}{\Rightarrow} c_1 c_2 F_j^R a_2 a_1 \qquad \overset{*}{\Rightarrow} c_1' c_2' F_k^S a_2' a_1'$$
$$\overset{*}{\Rightarrow} c_1 c_2 c_3 F_j^R a_3 a_2 a_1 \qquad \overset{*}{\Rightarrow} c_1' c_2' c_3' F_k^S a_3' a_2' a_1',$$
$$\overset{*}{\Rightarrow} c_1 c_2 c_3 b a_3 a_2 a_1 = w$$

где $|a_1| = |a_1'|$ или S никогда не изменяет своего входного значения после a_1'; $0 < |a_2| = |a_2'|$ или S никогда не изменяет своего входного значения после $a_2' a_1'$; $0 < |a_3| = |a_3'|$ или S никогда не изменяет своего входного значения после $a_3' a_2' a_1'$. Остаток доказательства в этом случае распадается на три подслучая, аналогичные первым трем подслучаям в доказательстве выше. В первом подслучае, если $a_1 \neq a_1'$, то для интерпретации I_1, определенной выше, $|\mathrm{val}_{I_1}(R)| < n$ и $\mathrm{val}_{I_1}(S)$ не определено. Во втором подслучае, если $a_1 = a_1'$, но $a_2 \neq a_2'$, то интерпретация I_2 дает требуемый эффект. Наконец, если $a_2 a_1 = a_2' a_1'$, то интерпретация I_1 дает требуемый эффект. Тем самым доказательство закончено.

Установив, что для некоторого подкласса класса рекурсивных схем проблема эквивалентности разрешима, мы покажем то же самое для класса стандартных схем с одной переменной и засылками констант. Проблема эквивалентности для стандартных схем с одной переменной без засылок констант разрешима, как было показано в разд. 2. Проблема эквивалентности для стандартных схем с двумя переменными неразрешима (Лаккэм и др. [4]), как и проблема эквивалентности для класса стандартных схем с двумя независимыми переменными и независимыми константами, упоминавшаяся в (3.9). Следовательно, следующая теорема является оптимальной.

(5.2) Теорема. *Существует эффективная процедура для распознавания, являются ли сильно эквивалентными две стандартные схемы с одной переменной и засылками констант.*

Доказательство. Пусть S — стандартная схема с одной переменной, в которой имеется k инструкций засылок констант, и пусть I — произвольная интерпретация. *Историей вычисления* схемы S при интерпретации I называется (возможно, бесконечная) цепочка $w = \ldots w_2 c_2 w_1 c_1 w_0$ функциональных символов, примененных схемой S к ее входному значению при I, где c_i — символы констант, а w_i — цепочки неконстантных функциональных символов. Заметим, что если S дважды выпол-

няет одну и ту же засылку константы при интерпретации I, то схема S должна зацикливаться при I. Следовательно, история вычисления w схемы S при I имеет одну из двух форм:

(a) $w = w_n c_n \dots w_1 c_1 w_0$, где $n \leqslant k$, а w_n может быть бесконечной цепочкой неконстантных функциональных символов, когда S не останавливается при I, или

(b) $w = \dots (w_n c_n \dots w_m c_m)(w_n c_n \dots w_m c_m) w_{m-1} c_{m-1} \dots$ $\dots w_1 c_1 w_0$, где $n \leqslant k$ и S зацикливается при I.

Мы покажем, что для любых данных стандартных схем R и S с одной переменной и засылками констант имеется эффективная граница, такая, что если $R \not\equiv S$, то существует интерпретация, при которой R и S различаются и при этом длина истории вычислений одной из схем меньше, чем граница. Как и раньше, этого достаточно для доказательства теоремы, поскольку эквивалентность R и S в этом случае можно проверить выполнением схем при конечном числе конечных интерпретаций.

Предположим, что R и S — схемы с не более чем e инструкциями, среди которых не более k засылок констант. Пусть I — свободная интерпретация, при которой R и S различаются, и предположим, не умаляя общности, что R и S имеют истории вычислений

$$w = \mathrm{val}_I(R) = w_n c_n \dots w_1 c_1 w_0$$

и

$$w' = w'_{n'} c'_{n'} \dots w'_1 c'_1 w'_0$$

или

$$w' = \dots (w'_{n'} c'_{n'} \dots w'_{n''} c'_{n''})(w'_{n'} c'_{n'} \dots w'_{n''} c'_{n''}) \times$$
$$\times w'_{n''-1} c'_{n''-1} \dots w'_1 c'_1 w'_0,$$

где цепочка w' может быть бесконечной, $|w| < |w'|$ и $n, n' \leqslant k$. Мы покажем, что если $|w| > k + (k+1)2^{2ke}$, то существует свободная интерпретация I', при которой R и S различаются, и при этом длина истории вычисления схемы R меньше, чем $|w|$.

Предположим, что $|w| > k + (k+1)2^{2ke}$. Тогда для некоторого $m \leqslant n$ справедливо, что $|w_m| > 2^{2ke}$. Мы обсудим случай $m > 0$, другие случаи похожи на него и еще проще. Пусть $w_m = f_{|w_m|} \dots f_1$. Для каждого $i < m$ пусть A_i — множество всех троек $\langle j, r, l \rangle$, таких, что

(a) $j = 0$, $r \leqslant n$, $c_r = c_m$, $w_r = u f_i \dots f_1$ для некоторой цепочки u, а инструкция l схемы R применялась к $f_i \dots f_1 c_r w_{r-1} \dots$ $\dots c_1 w_0$, или

(b) $j = 1$, $r \leqslant n'$, $c'_r = c_m$, $w'_r = u f_i \dots f_1$ для некоторой цепочки u, а инструкция l схемы S применялась к $f_i \dots f_1 c'_r w'_{r-1} \dots$

$\ldots c_1'w_0'$. Поскольку имеется не более 2^{2ke} таких различных множеств A_i, существуют целые $i_1 < i_2 \leqslant |w_m|$, такие, что $A_{i_1} = A_{i_2}$. Теперь для некоторых цепочек a_1 длины i_1, a_2 длины $i_2 - i_1$ и a_3 выполняется $w_m = a_3a_2a_1$. Назовем вхождение c_r (или c_r') константной функции критическим, если $c_r = c_m$ ($c_r' = c_m$) и $w_r = ua_1$ ($w_r' = ua_1$) для некоторой цепочки u. Тогда в силу выбора i_1 и i_2 для всякого критического c_r (или c_r') $w_r = va_2a_1$, и некоторая инструкция схемы R применяется к обеим цепочкам $a_2a_1c_rw_{r-1} \ldots c_1w_0$ и $a_1c_rw_{r-1} \ldots c_1w_0$ (то же для w_r').

Определим теперь новую свободную интерпретацию I' так, чтобы для всякого предиката P

$$P_{I'}(v) = \begin{cases} P_I(ua_2a_1c_m), & \text{если } v = ua_1c_m, \\ P_I(v) & \text{в противном случае.} \end{cases}$$

Для всякого r пусть

$$\hat{w}_r = \begin{cases} ua_1, & \text{если } c_r \text{ — критическое вхождение и } w_r = ua_2a_1, \\ w_r & \text{в противном случае.} \end{cases}$$

Подобным же образом определим и $\hat{w}_r'$. Тогда истории вычислений $\hat{w}$ и $\hat{w}'$ схем R и S при I' совпадают с теми цепочками, которые получаются из их историй при I посредством повсеместной замены w_r на $\hat{w}_r$ и w_r' на $\hat{w}_r'$. Поскольку $|\hat{w}_m| < |w_m|$, отсюда следует $|\hat{w}| < |w|$, так что остается показать, что $\mathrm{val}_{I'}(R) \not\simeq \mathrm{val}_{I'}(S)$.

Если S зацикливается при I, то S зацикливается и при I', а следовательно, $\mathrm{val}_I(R) \not\simeq \mathrm{val}_{I'}(S)$. Если S останавливается при I, то $w_nc_n \neq w_{n'}'c_{n'}'$, поскольку $\mathrm{val}_I(R) \neq \mathrm{val}_I(S)$. Имеются три случая:

(1) Ни c_n, ни $c_{n'}'$ — не критические. Тогда $\mathrm{val}_{I'}(R) = \hat{w}_nc_n = w_nc_n \neq w_{n'}'c_{n'}' = \hat{w}_{n'}'c_{n'}' = \mathrm{val}_{I'}(S)$.

(2) Ровно одно из вхождений c_n и $c_{n'}'$ — критическое. Тогда, скажем, $\hat{w}_nc_n = ua_1c_m$ для некоторого u, но $\hat{w}_{n'}'c_{n'}' \neq va_1c_m$ для любого v; следовательно, $\mathrm{val}_{I'}(R) \neq \mathrm{val}_{I'}(S)$.

(3) Оба вхождения c_n и $c_{n'}'$ — критические. Тогда $w_n = ua_2a_1$, $w_{n'}' = va_2a_1$ и $u \neq v$, так что $\hat{w}_n = ua_1 \neq va_1 = \hat{w}_{n'}'$ и, следовательно, $\mathrm{val}_{I'}(R) \neq \mathrm{val}_{I'}(S)$.

Итак, в каждом из случаев $\mathrm{val}_{I'}(R) \neq \mathrm{val}_{I'}(S)$, и доказательство закончено.

Сейчас мы пытаемся, комбинируя технику теорем 5.1 и 5.2, показать, что проблема эквивалентности для линейных рекурсивных схем с константными функциями является алгоритмически разрешимой.

СПИСОК ЛИТЕРАТУРЫ

1. deBakker J. W., Scott Dana, A Theory of Programs, August 1969, unpublished report.
2. Hopcroft J. E., Ullman J. D., Formal Languages and their Relation to Automata, Addison-Wesley, 1969.
3. Янов Ю. И., О логических схемах алгоритмов, сб. «Проблемы кибернетики», вып. 1, М., Физматгиз, 1958, стр. 75—127.
4. Luckham D. C., Park D. M. R., Paterson M. S., Formalized Computer Programs, *Journal Computer and System Sciences*, **4**, No. 3 (June 1970), 220—249. (Русский перевод: Лакхэм Д., Парк Д. М., Патерсон М. С., О формализованных машинных программах. Кибернетический сборник, новая серия, вып. 12, «Мир», М., 1975, стр. 78—114.)
5. Nivat M., Transductions des Languages de Chomsky, doctoral dissertation, Grenoble University (1967), Chapter 6.
6. Paterson M. S., Hewitt C. E., Comparative Schematology, Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, June 1970, published by the ACM Dec. 1970, pp. 119—128. (Русский перевод: Патерсон М., Хьюитт К., Сравнительная схематология, см. наст. сб., стр. 62—72.)
7. Rabin M., Scott D., Finite Automata and Their Decision Problems, *IBM J. Res. Develop.*, **3** (1959), 114—125.
8. Shepherdson J. C., Sturgis H. E., Computability of Recursive Functions, *Journal A. C. M.* **10**, № 2 (1963), 217—255.

Сетевые грамматики для анализа естественных языков[1])

В. А. Вудс

1. ОБОСНОВАНИЕ

Одной из ранних моделей для грамматик естественных языков был конечный граф переходов. Эта модель представляет собой сеть из узлов и связывающих их ориентированных дуг, причем узлы соответствуют состояниям конечного автомата, а дуги — переходам от состояния к состоянию. Каждая дуга помечена символом, ввод которого может вызвать переход вдоль данной дуги (в соответствии с ее ориентацией). Привлекательным свойством этой модели было то, что последовательность слов, образующая предложение, могла читаться в прямом направлении при прохождении пути от начального состояния к некоторому заключительному состоянию. К сожалению, эта модель существенно неадекватна для представления грамматик естественных языков, поскольку она не способна охватить многие закономерности языка. Наиболее явное несоответствие состоит в отсутствии магазина, позволяющего приостановить обработку составляющей на некотором уровне на время использования той же самой грамматики для обработки составляющей, вложенной в предыдущую.

Предположим, однако, что непосредственно к такой сетевой модели был бы добавлен механизм рекурсии. Другими словами, предположим, что мы взяли бы некоторую совокупность графов переходов, каждый с присвоенным ему именем, и позволили бы употреблять в качестве символов при дугах не только терминальные символы, но и нетерминальные символы, обозначающие сложные конструкции, наличие которых необходимо для того, чтобы переход по дуге был допустим. Проверка того, действительно ли такая конструкция имеется в предложении, делалась бы путем «обращения» как к «подпрограмме» к другому графу (или к тому же самому). Получающаяся модель грамматики, которую мы назовем *рекурсивной сетью переходов*, эквивалентна в смысле порождающей силы контекстно-свободной грамматике или автомату с магазинной памятью, но, как мы покажем, позволяет получить более компактное представление,

---

[1]) Woods W. A., Transition network grammars for natural language analysis, *Comm. ACM*, 13, № 10 (Oct. 1970), 591—606.

более эффективные алгоритмы анализа, а также естественное продолжение путем «расширения» до более мощной модели, которая уже допускает разной степени контекстные зависимости и более гибкое построение структур в процессе анализа. Мы покажем действительно, что с помощью «расширенной» рекурсивной сети переходов можно совершать анализ, эквивалентный трансформационному [6, 7], не вводя специальных инверсных трансформационных компонент, и что время анализа с помощью такой сети сравнимо с временем предсказуемостного анализа для КС-языков.

2. РЕКУРСИВНЫЕ СЕТИ ПЕРЕХОДОВ

Рекурсивная сеть переходов (в дальнейшем просто рекурсивная сеть) — это ориентированный граф с помеченными состояниями и дугами, выделенным состоянием, которое называется начальным, и выделенным множеством состояний, которые называются заключительными. Она имеет по существу такой же вид, как и диаграмма переходов конечного недетерминированного автомата, с тем исключением, что символы при дугах могут быть как терминальными, так и именами состояний. Дуга, помеченная именем состояния, имеет следующую интерпретацию: состояние, в которое ведет эта дуга, запоминается в магазине, а управление передается состоянию, которым помечена дуга (без продвижения входной ленты). Когда встречается заключительное состояние, разрешается «поднять» содержимое магазина, т. е. взять верхнее состояние в магазине и передать ему управление (из магазина это состояние вычеркивается). Пустота магазина, обнаруженная при попытке «поднять» его содержимое в тот момент, когда обработан последний входной символ, есть критерий допустимости входной цепочки. Имена состояний, которые в этой модели могут встречаться на дугах, — это по существу имена конструкций, которые могут быть обнаружены как самостоятельные «группы» (phrase) на входной ленте. Переход по дуге, помеченной именем состояния, может произойти, если конструкция указанного типа опознана как «группа» в соответствующем месте входной цепочки.

Пример рекурсивной сети для небольшого подмножества английского языка дан на рис. 1. Эта сеть допускает такие фразы, как "John washed the car" и "Did the red barn collapse?" По виду этой сети легко представить себе тип фраз, которые являются для нее допустимыми. Для анализа фразы "Did the red barn collapse?" работа начинается с состояния S. Первый переход — это переход по дуге AUX к состоянию Q2, возможный благодаря наличию вспомогательного глагола "did". Мы видим, что из состояния Q2 можно перейти в Q3, если следующий «эле-

мент» во входной цепочке есть NP [1]). Чтобы проверить, так ли это, обращаемся к состоянию NP. Из состояния NP мы можем пойти по дуге DET к состоянию Q6, так как присутствует детерминатив "the". Оттуда из-за наличия прилагательного "red" проходим по петле и возвращаемся к состоянию Q6, затем существительное "barn" вызывает переход к состоянию Q7. Так как состояние Q7 является заключительным, можно «подняться»

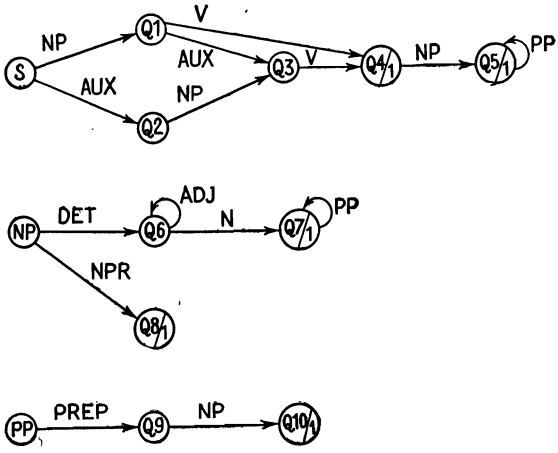


Рис. 1. Пример рекурсивной сети. S — начальное состояние, q_4, q_5, q_7, q_8 и q_{10} — заключительные состояния.

из анализа NP и продолжить движение на верхнем уровне S, начиная с состояния Q3, которое находится в конце дуги NP. От Q3 глагол "collapse" позволяет перейти к состоянию Q4, и, так как это состояние является заключительным, а "collapse" есть последнее слово цепочки, цепочка считается допустимым предложением.

В приведенном примере имеется только один путь через сеть, допускающий данное предложение, т. е. предложение однозначно по отношению к данной грамматике. Неотъемлемым свойством естественных языков, однако, является то обстоятельство, что, если речь идет не о специально выделенном подмножестве языка, всегда имеются неоднозначные предложения, для которых возможны несколько различных путей анализа в сети переходов. Поэтому сетевая модель является существенно недетерминированным механизмом и любой анализирующий алгоритм для сетевых грамматик должен уметь для любого заданного предложения проследить все возможные пути анализа.

[1]) NP (noun phrase) — именная группа. — *Прим. перев.*

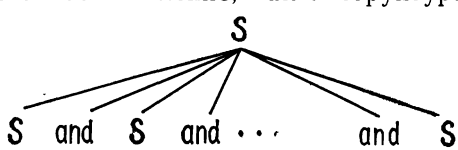
Нетрудно установить, что рекурсивная сеть эквивалентна магазинному автомату. По существу каждая рекурсивная сеть есть магазинный автомат, в котором алфавит магазина есть подмножество множества состояний. Обратное утверждение, что каждый магазинный автомат имеет эквивалентную ему рекурсивную сеть, может быть доказано непосредственно, но проще доказать это, заметив, что каждый магазинный автомат имеет эквивалентную КС-грамматику, для которой имеется эквивалентная рекурсивная сеть.

3. РАСШИРЕННЫЕ СЕТИ ПЕРЕХОДОВ

Хорошо известно (Хомский [6]), что в чистом виде КС-грамматики не представляют собой адекватный механизм для описания тонкостей естественных языков. Многие из условий, которым должны удовлетворять правильные предложения английского языка, требуют определенного согласования между различными частями предложения, которые могут быть, но могут и не быть соседними (в действительности, они могут отделяться теоретически неограниченным числом вставленных между ними слов). Контекстно-зависимые грамматики могут справиться со слабым порождением многих из таких конструкций, но только ценой потери лингвистической значимости «структуры составляющих», приписываемой грамматикой (Постал [27]).

Более того, обычная КС-грамматика не может показать регулярные соотношения, которые существуют между утвердительным предложением и соответствующим вопросительным, между предложением в активе и соответствующим предложением в пассиве и т. д. Теория трансформационных грамматик Хомского [7], различающая поверхностную и глубинную структуры предложения, преодолевает эти трудности, но и она неадекватна в других случаях (Шварц [28] или Мак-Коли [21]). В этом разделе мы опишем такую модель грамматики, основанную на понятии рекурсивной сети, которая может осуществить анализ, эквивалентный трансформационному, не нуждаясь в специальном трансформационном компоненте, и в отношении которой уже нельзя высказать многие из тех возражений, которые выдвигались против трансформационной модели грамматик.

КС-грамматики и взятые нами за основу рекурсивные сети, как мы их описали, слабо эквивалентны; эквивалентность отличается от сильной из-за возможности описывать в рекурсивной сети неограниченное ветвление, как в структурах вида



Существенным расширением возможностей трансформационных грамматик по сравнению с КС-грамматиками является способность перемещать фрагменты структуры предложения (так, что их положение в глубинной структуре отличается от положения в поверхностной), повторять и отбрасывать фрагменты структуры предложения и совершать действия над составляющими в зависимости от контекста, в котором эти составляющие находятся. Мы можем добавить эквивалентные возможности к сетевой модели, приписывая каждой дуге сети переходов дополнительное условие, которое должно быть выполнено для того, чтобы переход по дуге был возможен, и множество действий по построению структуры, которые следует осуществить, если совершается переход по данной дуге. Мы называем этот вариант модели *расширенной сетью переходов* — РСП (augmented transition network).

По мере продвижения от состояния к состоянию РСП постепенно формирует описание структуры предложения. Куски получаемых частичных описаний хранятся в *регистрах*, которые могут содержать любое дерево с корнем или множество деревьев с корнями и в которых данные автоматически передвигаются вниз при рекурсивном применении переходной сети и автоматически поднимаются, когда рекурсивное вычисление более глубокого уровня закончено. Действия по построению структур, приписанные дугам, определяют изменение содержимого этих регистров в зависимости от их предшествующего содержимого, содержимого других регистров, текущего входного символа и/или результата вычислений более глубокого уровня. Кроме того, что регистры содержат подструктуры, которые будут в конечном итоге включены в некоторые объемлющие их структуры, они могут содержать также пометки и другие указатели, к которым возможно обращение при проверке условий на дугах.

Каждому заключительному состоянию РСП сопоставлено одно или несколько условий, которые должны быть выполнены, чтобы это состояние могло вызвать «подъем» ("pop"). Каждое условие сопровождается функцией, значение которой вычисляется для выдачи в качестве результата. Когда в сети происходит возврат к дуге, вызвавшей рекурсивное вычисление, результат работы на более глубоком уровне помещается в специальный регистр * (который обычно содержит текущее входное слово). Таким образом, регистр * всегда содержит представление «элемента» (слова или группы), вызвавшего переход по дуге.

3.1. Представление РСП. Для того чтобы сделать обсуждение РСП более конкретным, мы приводим на рис. 2 спецификацию языка, на котором может быть описана РСП. Язык задается в форме расширенной КС-грамматики, где вертикальная

черта разделяет альтернативные возможности образования конструкции, а оператор * Клини используется как верхний индекс для указания произвольного числа повторений данной составляющей. Нетерминальные символы этой грамматики представляют собой записи на русском [1]) языке, заключенные в угловые скобки, а все остальные символы, кроме вертикальной черты и верхнего индекса *, являются терминальными (включая круглые

```
⟨РСП⟩ → (⟨пучок дуг⟩ ⟨пучок дуг⟩*)
⟨пучок дуг⟩ → (⟨состояние⟩ ⟨дуга⟩*)
⟨дуга⟩ → (CAT ⟨имя категории⟩ ⟨тест⟩ ⟨действие⟩* ⟨заключительное действие⟩) |
         (PUSH ⟨состояние⟩ ⟨тест⟩ ⟨действие⟩* ⟨заключительное действие⟩) |
         (TST ⟨произвольная метка⟩ ⟨тест⟩ ⟨действие⟩ ⟨заключительное дей-
                  ствие⟩) |
         (POP ⟨форма⟩ ⟨тест⟩)
⟨действие⟩ → (SETR ⟨регистр⟩ ⟨форма⟩) |
             (SENDR ⟨регистр⟩ ⟨форма⟩) |
             (LIFTR ⟨регистр⟩ ⟨форма⟩)
⟨заключительное действие⟩ → (TO ⟨состояние⟩) |
                            (JUMP ⟨состояние⟩)
⟨форма⟩ → (GETR ⟨регистр⟩) |
          * |
          (GETF ⟨свойство⟩) |
          (BUILDQ ⟨фрагмент⟩ ⟨регистр⟩*) |
          (LIST ⟨форма⟩*) |
          (APPEND ⟨форма⟩ ⟨форма⟩) |
          (QUOTE ⟨произвольная структура⟩)
```

Рис. 2. Спецификация языка для представления РСП.

скобки, которые обозначают списочную структуру). Заметим, что звездочка *, которая появляется как одна из альтернатив в правой части правила для конструкции ⟨форма⟩, является терминальным символом, ее не следует путать с верхним индексом *, означающим повторение составляющих. Верхняя строка рис. 2 означает, что РСП записывается так: левая скобка, пучок дуг, за которым следует произвольное число (нуль или более) пучков дуг, затем правая скобка. Пучок дуг в свою очередь записывается: левая скобка, затем имя состояния, затем произвольное число дуг, затем правая скобка, причем дуга может быть записана в одном из четырех видов, указанных в третьем правиле грамматики. Остальные правила интерпретируются аналогичным образом. Нетерминальные символы, развертка ко-

[1]) В оригинале на «английском». — *Прим. перев.*

торых не приведена на рис. 2, имеют имена, которые достаточно их объясняют.

Выражения, порождаемые как РСП грамматикой рис. 2, имеют форму скобочных списочных структур, где список элементов A, B, C, D представляется выражением (ABCD). РСП представляется как список пучков дуг, каждый из которых сам есть список, первый элемент которого есть имя состояния, а остальные элементы суть дуги, выходящие из этого состояния. Дуги также представляются списками, возможные формы которых указаны на рис. 2. (Условия и функции, соответствующие заключительным состояниям, представляются как «псевдодуги», никуда не ведущие и не имеющие действий.) Первый элемент каждой дуги — это слово, которое указывает тип дуги, а третий элемент — некоторый тест, который должен дать положительный результат, чтобы можно было пройти по этой дуге.

Дуга типа CAT — это дуга, по которой можно пройти, если текущий входной символ является элементом лексической категории, указанной при дуге (и если результат теста положительный). Дуга типа PUSH вызывает «спуск» (PUSHDOWN) к состоянию, указанному после символа PUSH. Дуга типа TST допускает произвольное условие, чтобы проверить, можно ли пройти по дуге. В дугах этих трех типов «действиями» являются действия по построению структуры, а заключительное действие определяет состояние, которому передается управление в результате перехода по данной дуге.

Два возможных заключительных действия TO и JUMP указывают соответственно, нужно или не нужно передвинуть указатель входа, т. е. указывают, должно ли в следующем состоянии рассматриваться следующее входное слово или то же самое входное слово. Дуга типа POP — это фиктивная дуга, указывающая, при каких условиях состояние следует рассматривать как заключительное, и определяющая форму, которая должна быть выдана как результат вычислений, если выбрана альтернатива POP. (Одно из преимуществ представления этой информации как фиктивной дуги состоит в возможности упорядочить «подъем» по отношению к другим дугам, выходящим из того же состояния.)

Действия и *формы* в РПС представлены в «кембриджско-польской» записи, где функция представлена как заключенный в скобки список, в котором первый элемент есть имя функции, а остальные элементы суть ее аргументы. Три действия, приведенные на рис. 2, заносят в указанные в них регистры значения указанных форм. SETR совершает это на текущем уровне вычислений в сети, тогда как SENDR — на следующем более глубоком уровне (используется для засылки информации вниз на более глубокий уровень вычислений), а LIFTR — на соседнем

более высоком уровне вычислений (используется для возврата дополнительной информации к более высокому уровню вычислений).

Формы, так же как условия (*тесты*) в РСП, могут быть произвольными функциями от содержимого регистров, заданными на некотором языке для описания функций (как, например, на LISP — см. Мак-Карти и др. [20] — программном языке, основанном на λ-исчислении Чёрча) и записанными в кембриджско-польской системе обозначений. Семь типов форм, перечисленные на рис. 2, являются базовым множеством, достаточным для иллюстрации основных свойств РСП-модели. GETR — это функция, значением которой является содержимое указанного регистра, * — форма, значением которой обычно является текущее входное слово, а GETF — функция, которая определяет значение указанного признака для текущего входного слова. (В действиях, которые появляются на дугах типа PUSH, значением * является результат вычисления на более глубоком уровне, успешное выполнение которого позволяет пройти до конца дугу PUSH.)

BUILDQ является полезной формой построения структуры, которая, получая на входе списочную структуру, представляющую фрагмент дерева анализа с специально помеченными узлами, выдает в качестве значения результат замещения этих специально помеченных узлов содержимым указанных регистров[1]). В частности, вместо каждого вхождения символа + в списочную структуру, являющуюся первым аргументом функции BUILDQ, подставляется содержимое очередного регистра из списка (содержимое первого регистра заменяет первый +, содержимое второго регистра — второй + и т. д.). Кроме того, BUILDQ заменяет вхождения символа * во фрагменте значением формы *.

Остальные три формы являются базисными формами для построения структур (из которых может быть составлена любая

---

[1]) Возможности функции BUILDQ, которая реализована в экспериментальной анализирующей системе (см. разд. 10), гораздо шире описанных здесь. Точно так же реализованный нами анализатор содержит дополнительные форматы для дуг и некоторые другие расширения возможностей языка по сравнению с тем, что описано здесь. Мы не ставили себе целью сделать неизбыточным базовое множество исходных условий, действий и форм, наши усилия были направлены скорее к тому, чтобы обеспечить гибкость в отношении добавления «естественных» исходных элементов, которые облегчили бы написание компактных грамматик. Поэтому множество возможных условий, действий и форм было оставлено открытым, позволяющим экспериментальное определение полезных основных элементов. Однако форматы дуг и действия, описанные здесь, вместе с произвольными выражениями на LISP в качестве условий и форм дают модель, которая по мощности эквивалентна машине Тьюринга и, следовательно, в теоретическом смысле полна.

BUILDQ). Эти формы соответственно составляют список значений перечисленных аргументов, объединяют два списка в один и дают в качестве значения форму, являющуюся аргументом.

```
(S / (PUSH NP/T
        (SETR SUBJ *)
        (SETR TYPE (QUOTE DCL))
        (TO Q1)
    (CAT AUX T
        (SETR AUX *)
        (SETR TYPE (QUOTE Q))
        (TO Q2)))
(Q1 (CAT V T
        (SETR AUX NIL)
        (SETR V *)
        (TO Q4))
    (CAT AUX T
        (SETR AUX *)
        (TO Q3)))
(Q2 (PUSH NP/T
        (SETR SUBJ *)
        (TO Q3)))
(Q3 (CAT V T
        (SETR V *)
        (TO Q4)))
(Q4 (POP (BUILDQ (S + + + (VP +)) TYPE SUBJ AUX V) T)
    (PUSH NP/T
        (SETR VP (BUILDQ (VP (V +) *) V))
        (TO Q5)))
(Q5 (POP (BUILDQ (S+ + + + +) TYPE SUBJ AUX VP) T)
    (PUSH PP/T
        (SETR VP (APPEND (GETR VP) (LIST *)))
        (TO Q5)))
```

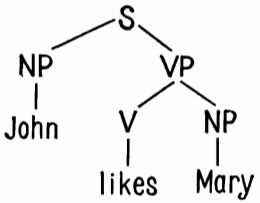
Рис. 3. Иллюстративный фрагмент РСП.

Для иллюстрации на рис. 3 дан фрагмент РСП. В разд. 3.2 описывается работа этой сети и обсуждаются некоторые свойства РСП-модели.

3.2. Иллюстративный пример. На рис. 3 приведен фрагмент РСП, записанный на языке, который дан на рис. 2. Этот фрагмент является расширением той части сети рис. 1, которая содержит состояния S/, Q1, Q2, Q3, Q4 и Q5. РСП строит структурное представление предложения, в котором первая составляющая есть *тип* (либо DCL (declarative), либо Q), указывающий, является ли предложение повествовательным или вопро-

сительным, вторая составляющая есть группа подлежащего, третья — это вспомогательный глагол (или NIL, если вспомогательного глагола нет), четвертая составляющая есть группа глагола. Такое представление получается независимо от того, в каком порядке стояли подлежащее и вспомогательный глагол в предложении. Сеть дает также представление группы глагола, хотя в сети и нет магазина, соответствующего этой группе. Как для понимания обозначений, так и для понимания работы РСП полезно сейчас разобрать один пример, используя фрагмент РСП, изображенный на рис. 3.

Прежде чем приступать к рассмотрению примера, необходимо объяснить представление дерева анализа, которое используется этим фрагментом. Деревьям анализа сопоставляются скобочные записи, в которых узел представлен списком, где первый элемент есть имя узла, а остальные элементы суть представления составляющих этого узла. Например, дереву анализа



будет сопоставлено выражение

(S (NP John) (VP (V likes) (NP Mary)))

Это представление можно также рассматривать как размеченное разбиение предложения на группы, в котором левая граница группы типа X представлена левой скобкой и символом X, а соответствующая правая граница — просто правой скобкой.

Рассмотрим теперь работу фрагмента РСП, представленного на рис. 3, для входного предложения "Does John like Mary?"

1. Мы начинаем процесс с состояния S/ и рассмотрения первого слова предложения "does". Так как это слово является вспомогательным глаголом, в его словарной статье будет указано, что он является элементом категории AUX, и следовательно (так как T (true) обозначает тождественно истинное условие), можно пойти по дуге (CAT AUX T...). (Другая дуга, которая ведет к спуску для поиска именной группы, не годится.) Проходя по выбранной дуге, мы выполняем следующие действия: (SETR AUX*), которое заносит рассматриваемое слово "does" в регистр, названный AUX; (SETR TYPE (QUOTE Q)), которое заносит символ Q в регистр, названный TYPE, и (TO Q2), которое переводит сеть в состояние Q2 для рассмотрения следующего слова предложения "John".

2. Из состояния Q2 выходит только одна дуга — спуск к состоянию NP/. Работа на более глубоком уровне будет выполнена и выдаст представление структуры именной группы, которое станет значением специального регистра *. Мы допустим, что выданное представление есть выражение (NP John). Теперь, опознав конструкцию типа NP, мы осуществляем действия, соответствующие дуге. Действие (SETR SUBJ*) заносит значение (NP John) в регистр SUBJ, а действие (TO Q3) переводит сеть в состояние Q3 для рассмотрения следующего слова "like". В этот момент регистры заполнены следующим образом:

TYPE: Q
AUX : does
SUBJ: (NP John)

3. Из состояния Q3 глагол "like" делает возможным переход к состоянию Q4 с засылкой "like" в регистр V, а входной указатель перемещается к слову "Mary".

4. Так как Q4 — заключительное состояние, из него выходит дуга POP, указывающая, что рассмотренная цепочка может быть полным предложением в соответствии с грамматикой, но поскольку предложение еще не кончилось, то выбрать эту альтернативу нельзя. Из этого состояния выходит также дуга, которая спускает нас к состоянию NP/. Эта альтернатива допустима, в результате обработки получается значение (NP Mary). Теперь действие (SETR VP (BUILDQ (VP (V+) * V)) берет фрагмент структуры (VP(V+)*) и подставляет текущее значение * вместо вхождения * во фрагмент, а также заменяет вхождение знака + содержимым указанного регистра V. Полученная структура (VP (V like) (NP Mary)) помещается в регистр VP, а действие (TO Q5) вызывает переход к состоянию Q5 и выход за конец входной цепочки. Содержимое регистров в этот момент таково:

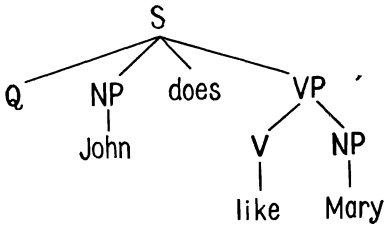
TYPE: Q
AUX: does
SUBJ: (NP John)
V: like
VP: (VP (V like) (NP Mary))

5. Так как предложение окончилось и Q5 есть заключительное состояние (т. е. имеет дугу типа POP), а условие T выполнено, данное предложение считается допустимым. Форма (BUILDQ (S + + + +)TYPE SUBJ AUX VP) дает значение, которое должно быть выдано как результат анализа предложе-

ния. Это значение получается в результате подстановки содержимого регистров TYPE, SUBJ, AUX и VP вместо последовательных вхождений символа + во фрагмент (S + + + +), что дает результат анализа предложения в виде записи

(S Q (NP John) does (VP (V like) (NP Mary)))

которая представляет дерево анализа:



В обычном КС-анализе структура предложения является более или менее прямым представлением хода передач управления в процессе анализа предложения. Структурное представление, которое дают правила построения структур в РСП, как мы видели на примере, сравнительно независимо от порядка работы алгоритма. Это не означает, что оно не зависит от хода работы анализатора, поскольку оно им определяется; мы скорее хотим подчеркнуть, что оно не является изоморфным отражением порядка передач управления, как в обычных алгоритмах КС-анализа. Так, составляющая, найденная в процессе анализа, может появиться в окончательном структурном представлении несколько раз или ни разу, и ее место может быть совсем не там, где эта составляющая найдена в поверхностной структуре. Кроме того, структурное представление составляющей может быть в ходе анализа изменено или преобразовано прежде, чем эта структура войдет в окончательное структурное представление предложения в целом. Эти возможности плюс возможность проверки произвольных условий позволяют построить структуру, эквивалентную глубинной трансформационной структуре, в то время как анализатор совершает переходы, которые изоморфны поверхностной структуре предложения.

4. ТРАНСФОРМАЦИОННЫЙ АНАЛИЗ

Обычная модель трансформационной грамматики — это порождающая модель, состоящая из контекстно-свободной (*базовой*) грамматики и множества трансформационных правил, которые отображают синтаксические деревья в новые (*производные*) синтаксические деревья. Порождение предложения такой

грамматикой состоит в построении сначала *глубинной структуры* с помощью базовой грамматики и в преобразовании затем этой глубинной структуры в *поверхностную* путем последовательного применения трансформаций. Концевые узлы (или листья) дерева поверхностной структуры дают окончательную форму предложения. Эта модель трансформационной грамматики полностью ориентирована на порождение предложений, а не на анализ, и, хотя существует очевидный алгоритм использования такой грамматики для анализа предложения, а именно процедура «анализа через синтез» (Мэтьюз [23]), этот алгоритм настолько неэффективен, что не может быть речи о его практическом применении. (Анализ через синтез состоит в применении правил в «прямом» (порождающем) направлении всеми возможными способами для порождения всех возможных предложений языка и в наблюдении за тем, не появилось ли среди порожденных то предложение, которое мы хотим проанализировать.)

Две попытки сформулировать более практичные алгоритмы для трансформационного анализа (Петрик [26] и Митре [24]) дали алгоритмы, которые либо для многих предложений тратят слишком много времени на анализ, либо являются формально неполными. Оба эти алгоритма пытаются анализировать предложения, применяя правила трансформаций в обратном направлении, — процедура, которая осуществляется далеко не так просто, как формулируется. Применение трансформаций в обратном направлении наталкивается на трудности двух видов. Во-первых, трансформации применяются к древовидным структурам и дают в результате тоже древовидные структуры. В прямом направлении они начинают с дерева глубинной структуры и дают дерево поверхностной структуры. Для того чтобы этот процесс пошел в обратном направлении, надо сначала получить дерево поверхностной структуры для входного предложения. Однако в трансформационной модели нет компонента, который характеризовал бы допустимые поверхностные структуры (имеется только их неявная характеристика указанием изменений, которые могут быть сделаны в глубинной структуре трансформациями).

Обе процедуры анализа, как Митре, так и Петрика, решают эту проблему путем построения «расширенной грамматики», состоящей из правил исходной базовой грамматики плюс дополнительные правила, характеризующие структуры, которые могут быть добавлены трансформациями. В анализе Митре эта поверхностная грамматика строится вручную и нет формальной процедуры для ее построения по исходной трансформационной грамматике. В анализе Петрика есть формальная процедура для построения расширенной грамматики, однако она может не за-

кончиться, если только длина возможных входных предложений не будет заранее ограничена. Когда появляются предложения, длина которых больше выбранной, для расширенной грамматики нужно построить новые правила.

В анализе Митре расширенная грамматика используется для получения полной «пробной» поверхностной структуры, к которой затем применяются в обратном направлении трансформации. В анализе Петрика обратные трансформации применяются к частично построенной поверхностной структуре, и процессы применения трансформаций и построения структуры переплетаются. В обеих системах обратные трансформации могут дать, но могут и не дать допустимую глубинную структуру. Если они дают таковую, то данное предложение приемлемо, но если не дают — значит, пробная поверхностная структура является ложной и она отбрасывается. Нельзя построить поверхностную КС-грамматику, которая будет приписывать предложению все допустимые поверхностные структуры и только их. Приходится удовлетворяться такой, которая приписывает все допустимые (правильные) поверхностные структуры плюс некоторое количество ложных. Более того, единственный способ отделить эти два вида структур состоит в том, чтобы применить обратные трансформации и проверить полученные «пробные» глубинные структуры.

Вторая трудность с алгоритмом Петрика состоит в росте числа возможных последовательностей обратных трансформаций, которые могут быть применены к заданному поверхностному дереву. Хотя многие из трансформаций, когда они применяются в прямом направлении, являются обязательными, так что только одно возможное действие может быть проделано, почти все обратные трансформации факультативны. Причина этого состоит в том, что даже тогда, когда заданная структура выглядит так, как если бы она была получена определенной прямой трансформацией, т. е. можно применять соответствующую обратную трансформацию, нет гарантии, что та же самая структура не могла быть получена другим трансформационным выводом. Поэтому, когда какая-то обратная трансформация может быть применена, должны быть опробованы обе альтернативы: как применение обратной трансформации, так и неприменение ее. С ростом числа применяемых трансформаций число возможных активных путей может расти экспоненциально. Более того, прямые трансформации часто дают очень мало сведений о структуре, получающейся в результате их применения (даже если лингвисты много знают о том, как должна выглядеть получающаяся структура). Поэтому обратные трансформации не так избирательны, как соответствующие прямые, и поэтому становятся возможными гораздо больше ложных приме-

нений. Это значит, что, в то время как большинство последовательностей прямых трансформаций ведут к правильным поверхностным структурам, многие последовательности обратных трансформаций не ведут к допустимым глубинным структурам, и много напрасных усилий тратится на тупики. Анализ Митре преодолевает недетерминированность обратного трансформационного процесса путем построения ad hoc для той или иной частной грамматики детерминированных множеств обратных трансформационных правил. Этот метод, однако, не гарантирует получения всех допустимых глубинных структур, и не существует формальной процедуры для построения необходимого множества обратных трансформаций.

5. РСП И ТРАНСФОРМАЦИОННЫЙ АНАЛИЗ

В 1965 г. Куно [18] высказал предположение, что можно расширить в трансформационной грамматике ту ее часть, которая дает поверхностную структуру, таким образом, чтобы она «помнила» соответствующие глубинные конструкции и могла бы построить глубинную структуру предложения в процессе поверхностного анализа, тогда отпадала бы необходимость в специальном обратном трансформационном компоненте. Модель, которую он тогда предложил, однако, оказалось неадекватной для работы с некоторыми из мощных трансформационных механизмов, как, например, экстрапозиция составляющих из вставления произвольной грубины. С другой стороны, РСП дает модель, которая может сделать все то же, что и трансформационная грамматика, и есть, следовательно, реализация части предположения Куно. Остается выяснить, возможна ли совершенно механическая процедура, позволяющая взять трансформационную грамматику, заданную обычным формализмом, и перевести ее в эквивалентную РСП. Я полагаю, что дело обстоит именно так.

Однако, даже если такая механическая процедура осуществима, может оказаться, что для развития оригинальных лингвистических исследований и построения грамматик лучше использовать РСП-модель прямую. Для этого имеются разные причины. Во-первых, нельзя ожидать, что РСП, которые могут быть получены механически из традиционных трансформационных грамматик, будут столь же эффективны, как созданные вручную. Во-вторых, РСП-модель дает механизм, против которого уже нельзя выдвинуть некоторые из возражений, высказывавшихся лингвистами против трансформационных грамматик как языковых моделей (такие, как несовместимость со многими психолингвистическими фактами, которые, как известно, характеризуют использование языка человеком).

Третья причина предпочтения РСП-модели обычным формулировкам трансформационных грамматик — это возможности, которые дает использование произвольных условий и произвольных действий по построению структур. Модель по мощности эквивалентна машине Тьюринга и в то же время действия, которые она совершает, «естественны» для анализа языка. Многие из лингвистических исследований, касающихся изучения структуры языков и механизмов грамматик, специально имели целью построение моделей, которые не обладали бы мощностью машины Тьюринга и в которых использовались бы очень слабые грамматические механизмы, т. е. тем самым делались очень сильные предположения о механизмах языка, к которым такие грамматики применимы.

В результате этого подхода было предложено много вариантов модели трансформационной грамматики с различными наборами базисных трансформационных механизмов. Одни из них содержат циклические трансформационные правила; другие имеют специальные «послециклические» правила, которые действуют после того, как все циклические правила применены. Имеются различные типы условий: некоторые модели имеют двойное описание структур, некоторые имеют упорядоченные правила, некоторые — обязательные правила, некоторые — блокирующие правила и т. д. Короче говоря, существует не одна модель трансформационной грамматики, а много моделей, которые более или менее несравнимы. Если одна из таких моделей может обработать одни явления языка, а вторая модель — другие, то не существует регулярной процедуры объединения обеих в одну модель.

В РСП есть возможность добавлять к модели любые средства, которые нужны и кажутся естественными для решения задачи. Можно добавить новый механизм простым введением нового базисного предиката, используемого в условиях, или новой функции, используемой в правилах построения структур. Можно также сделать сильные предположения о типах условий и действий, которые требуются, но если выясняется, что надо выполнить действия, для которых основная модель не имеет «естественного» механизма, то не представляет труда дополнить РСП, включив такой механизм.

Для этого требуется лишь ослабление ограничений на типы условий и действий, но не нужно перестраивать базовую модель.

6. ПРЕДШЕСТВУЮЩИЕ МОДЕЛИ СЕТЕЙ ПЕРЕХОДОВ

Основная идея рекурсивной сети — идея объединения правых частей правил КС-грамматики, имеющих общую левую

часть, в единую диаграмму переходов была известна разработчикам синтаксически управляемых трансляторов и искусственных языков программирования по крайней мере с 1963 г., когда она была описана в работе М. Конвея [8]. Однако в то время интерес представляла не столько полная общность недетерминированного механизма, сколько множество условий, достаточных для того, чтобы гарантировать детерминированность диаграммы. Конвей описал элементарную форму действий, сопоставляемых дугам диаграммы, но эти действия ограничиваются командами записи информации на специальной выходной ленте, которая служит входом для компонента, порождающего коды. (Эта модель очень близка к обычной модели преобразователя с конечным числом состояний и отличается дополнительной возможностью рекурсии.) В модели Конвея не имеется аналога временной записи информации в регистры или последующего использования такой информации в условиях при дугах.

Впоследствии в литературе были описаны две системы анализа для естественных языков, основанные на рекурсивной сети переходов. Торне, Брэтли и Дьюар [29] описали процедуру анализа естественного языка, основную на сети переходов с конечным числом состояний (которая применяется рекурсивно), а Бобров и Фразер [1] описали систему, которая является «развитием процедуры, описанной Торне, Брэтли и Дьюаром». Хотя эти системы имеют значительное сходство с описанной нами, они обладают рядом существенных отличий, которые мы укажем ниже. Однако сначала коротко опишем эти две системы.

6.1. Система Торне. Система Торне [29] использует синтаксическое представление, в котором автор пытается одновременно отразить и глубинную, и поверхностную структуры предложения. Конструкции перечисляются в том порядке, в котором они обнаруживаются в поверхностной структуре, а их функции в глубинной структуре обозначаются метками. Инверсии в порядке слов указываются пометкой при структурах, которые найдены «не на месте» (т. е. в позициях, отличных от их позиций в глубинной структуре), без перемещения их с позиций, занимаемых ими в поверхностной структуре. Дальше в цепочке позиции, в которых они должны были бы появиться в глубинной структуре, отмечаются меткой, соответствующей глубинной функции, за которой следует звездочка. (Авторы не описывают процедуру для обработки составляющих, которые обнаруживаются в поверхностной структуре правее, чем их позиция в глубинной структуре. По-видимому, их грамматика не учитывает такие конструкции.)

Торне рассматривает свою грамматику как форму трансформационной грамматики, базовым компонентом которой яв-

ляется автоматная грамматика, и допускает рекурсии только для трансформаций. Согласно Торне, большая часть трансформационных правил может рассматриваться как «метаправила» в том смысле, что они скорее «оперируют с другими правилами с тем, чтобы дать производные правила, чем с описаниями структур с тем, чтобы дать новое описание структуры». Он использует расширенную сеть переходов, содержащую как исходные правила построения глубинной структуры, так и эти производные правила, в качестве грамматической таблицы, которая направляет анализирующий алгоритм, но не может таким способом обработать трансформации, дающие инверсии в порядке слов, или трансформации для сочинительных конструкций. Вместо этого он добавляет к своему анализирующему алгоритму эти случаи как исключения.

6.2. Система Боброва и Фразера. Бобров и Фразер [1] описали анализирующую систему, которая есть развитие системы Торне. Аналогично системе Торне основная форма результата анализа у них «подобна поверхностной структуре с добавочными указаниями о перемещенных составляющих и их месте в глубинной структуре». Эта модель грамматики также есть разновидность расширенной сети переходов, действия которой включают расстановку пометок и меток функций, а условия сети включают проверку ранее поставленных пометок. Однако в отличие от системы Торне система Боброва предусматривает средства для пересылки информации «назад» некоторым ранее проанализированным составляющим. Вообще говоря, условия при дугах могут быть произвольными функциями LISP (система запрограммирована на LISP) и действия для передачи информации могут также быть произвольными функциями LISP. Однако условия и действия, реализованные в настоящее время в системе, ограничиваются проверкой пометок и приписыванием новых меток функций в глубинной структуре частям структуры, проанализированным ранее.

Согласно утверждению Боброва [1]), основное различие между его системой и системой Торне состоит в использовании символьных имен пометок (вместо закрепленных разрядов), средств для использования мнемонических имен состояний, средств для передачи информации «назад» ранее проанализированным составляющим и средств для записи в словаре «активных» свойств (эти последние представляют собой подпрограммы, которые записываются в словарных статьях слов, а не просто приводятся в действие признаками, записанными в словаре).

[1]) Устное сообщение.

6.3. Сравнение с предлагаемой моделью. При сопоставлении РСП-модели, описанной в настоящей работе, с системами Боброва и Фразера [1] и Торне и др. [29] следует различать два направления, по каждому из которых может идти сравнение: формальное описание модели и реализация анализирующей системы. Одно из основных отличий предлагаемой анализирующей системы от систем Боброва и Торне состоит в степени проведения этого различия. В [29] Торне не описал концепцию расширенной сети переходов, которая им используется, за исключением того, что он указал, что грамматическая таблица, которую использует анализирующая программа, «имеет вид сети с конечным числом состояний или ориентированного графа — форму, принятую для представления регулярных грамматик». Эта сетевая модель, по-видимому, формально определена только в той форме, в какой она фактически фигурирует в анализирующей программе (которая не описана). Условия при дугах, видимо, ограничены проверками согласования признаков, приписанных лексическим единицам и составляющим, а действия ограничиваются записью текущей составляющей в выходном представлении, приписыванием меток составляющим или вставкой фиктивных узлов и маркеров. Средства для обработки инверсии в порядке слов или сочинительных конструкций не представлены в сети, но «включены в программу».

В системе Боброва и Фразера [1] заметно повышается мощность основной сетевой модели, использованной Торне и др. В ней добавляются средства для произвольных условий и действий при дугах, что увеличивает мощность модели, делая ее эквивалентной машине Тьюринга Однако в этой системе, как и в системе Торне, не проведено различие между моделью и ее реализацией. Хотя в реализации условия и действия произвольны, нет отдельной формальной модели, которая характеризовала бы структуру данных, с которыми они работают. То есть для того чтобы добавить произвольное условие, надо знать, как работает реализация анализирующего алгоритма на языке LISP и где и как записываются его промежуточные результаты. Условия и действия, которые можно использовать без такой информации, т. е. подпрограммы условий и действий, предусмотренные в настоящее время в реализации, состоят в записи и проверке пометок и перемещении меток функций назад к ранее проанализированным составляющим. В обеих системах Боброва и Торне фактическая структура составляющих изоморфна рекурсивной структуре анализа, как эта последняя определяется историей рекурсивного применения сети. Структура составляющих автоматически порождается анализирующим алгоритмом.

РСП, как мы ее описали, дает формальную сетевую модель, имеющую мощность машины Тьюринга, *независимо от реализа-*

ции. В модели проводится явное разделение различных частичных результатов в регистрах с именами и допускаются произвольные условия и действия, *которые используют содержимое этих регистров.* Поэтому составителю грамматики не обязательно знать детали реализации анализирующего алгоритма для того, чтобы воспользоваться преимуществами произвольных условий и действий [1]). Построение структуры составляющих в этой модели не делается автоматически анализирующим алгоритмом, оно должно быть определено явно заданными правилами построения структур. В результате этого структура, сопоставляемая предложению, не обязана больше быть изоморфным представлением рекурсивной истории анализа, и составляющие в структуре могут переставляться. Поэтому представление, даваемое анализатором, может быть представлением глубинной структуры того типа, который дают обычные трансформационные грамматики (или также это может быть представление поверхностной структуры, или представление, отражающее обе структуры, как в системах Боброва и Торне, или любое из множества других представлений, как, например, представление зависимостей). Явное указание действий по построению структур при дугах вместе с использованием регистров для хранения фрагментов структуры предложения (функции и места которых могут еще не быть определены) дает исключительно гибкое и эффективное средство для перемешения составляющих в представлении глубинной структуры и для изменения интерпретации составляющих в соответствии с ходом анализа. Можно даже построить структуру с несколькими уровнями вставления, оставаясь на одном и том же уровне переходной сети, и, обратно, пройти несколько уровней рекурсии в сети при построении структуры, которая имеет только один уровень. Подобных возможностей нет в системах Боброва и Торне.

Другая особенность РСП, предложенной здесь, которая отличает ее от систем Торне и Боброва, — это язык для спецификации грамматики переходной сети. Этот язык ориентирован скорее на то, чтобы быть удобным и естественным для составителя грамматики, чем для машины или для программиста. Можно на нескольких страницах полностью определить допустимые синтаксические формы для представления РСП. Каждая дуга представлена мнемоническим именем типа дуги, меткой дуги, произвольным условием и списком действий, которые сле-

[1]) В экспериментальной анализирующей системе иногда выгодно использовать условия и действия, которые основаны на особенностях реализации, не отраженных в формальной модели. Действия такого рода рассматриваются как *расширение* базовой модели, и особенности реализации, позволяющие легко их добавить, — это в значительной степени свойства системы BBN LISP [2], с помощью которой реализована система.

дует осуществить при движении по этой дуге. Условия и действия представляются как выражения в кембриджско-польской записи с мнемоническими именами функций. Мы позаботились о том, чтобы создать базисный набор таких функций, который является «естественным» для проблемы анализа естественных языков. Одной из целей экспериментальной анализирующей системы, основанной на РСП, которую я реализовал, было именно создание такого набора естественных операций путем накопления опыта при написании грамматик для этой системы, и многие из базисных операций, описанных в настоящей работе, являются результатом происходившей эволюции. Одной из ценных характеристик РСП-модели является то, что она легко допускает подобного рода эволюцию.

7. ПРЕИМУЩЕСТВА РСП-МОДЕЛИ

РСП-модель имеет много преимуществ как модель грамматики для естественных языков, некоторые из которых распространяются также и на языки программирования. В этом разделе мы рассмотрим и резюмируем некоторые из основных черт РСП-модели, которые определяют ее привлекательность в качестве модели для естественных языков.

7.1. Прозрачность. КС-грамматики как модель для описания естественных языков были чрезвычайно удачны (или, во всяком случае, популярны), несмотря на их формальную неадекватность в отношении некоторых характерных свойств естественных языков. КС-грамматики обладают определенной степенью «прозрачности», так как составляющие, которые дают конструкцию некоторого заданного типа, могут быть прочтены непосредственно из контекстно-свободного правила. То есть по виду правила КС-грамматики сразу виден результат применения этого правила к разрешенным конструкциям. С другой стороны, магазинный автомат, хотя он и эквивалентен КС-грамматике по порождающей мощности, не обладает такой прозрачностью. Поэтому не удивительно, что лингвисты при создании грамматик для естественных языков работали с формализмом КС-грамматик, а не прямо с магазинными автоматами, хотя последние благодаря их конечно-автоматному механизму управления допускают некоторую экономию в представлении и дают в результате более эффективные анализирующие алгоритмы.

Теория трансформационных грамматик, предложенная Хомским [6], является одним из наиболее мощных средств для описания предложений, возможных в естественных языках, и связей между ними, но эта теория при ее теперешней формализации (в тех ограниченных пределах, в которых она формализована) те-

ряет прозрачность КС-грамматик. В этой модели невозможно, глядя на одиночное правило, осознать немедленно последствия его применения к допустимым на входе конструкциям. Результат применения отдельного правила тесно связан с его взаимоотношениями с другими правилами, и может потребоваться очень сложный анализ для того, чтобы по фрагменту трансформационной грамматики естественного языка установить результат и назначение некоторого определенного правила. РСП имеет мощность трансформационной грамматики, но сохраняет в значительной степени прозрачность модели КС-грамматики. Если бы РСП была реализована на машине с графико-построителем для вывода сети на дисплее, она была бы одной из наиболее прозрачных (и мощных) моделей грамматики, доступных в настоящее время.

7.2. Порождающая мощность. Даже без условий и действий при дугах рекурсивная сетевая модель имеет большую порождающую мощность, чем обычные КС-грамматики (нет сильной эквивалентности). Это достигается благодаря возможности характеризовать конструкции, которые имеют неограниченное число непосредственных составляющих. Обычные КС-грамматики не позволяют определить дерево с неограниченным ветвлением иначе, чем допуская бесконечное множество правил. В некотором смысле можно рассматривать рекурсивную сетевую модель как конечное представление КС-грамматики с возможно бесконечным (но регулярным) множеством правил. Когда добавляются условия и действия при дугах, модель достигает мощности машины Тьюринга, хотя основные операции, которые она осуществляет, являются «естественными» для анализа языка. С использованием этих условий и действий модель дает возможность осуществить эквивалент трансформационного анализа и при этом она не нуждается в отдельном трансформационном компоненте.

Другое привлекательное свойство РСП-модели состоит в том, что эффективность не приносится в жертву мощности. При переходе от КС-грамматик к НС-грамматикам и трансформационным грамматикам время работы соответствующих алгоритмов анализа чрезвычайно возрастает. Вместе с тем РСП-модель достигает мощности трансформационной грамматики, не требуя, по-видимому, существенно больше времени, чем требует предсказуемостный анализ для КС-языков. (Это показывает в некоторой степени пример в разд. 8.)

Дополнительное преимущество РСП-модели по сравнению с трансформационной моделью состоит в том, что она гораздо ближе к двусторонней модели, чем трансформационная. Это значит, что хотя мы описываем ее как распознающую, или ана-

лизирующую модель, которая *анализирует* предложения, реально нет ограничений, которые помешали бы модели действовать в направлении синтеза для *порождения*, или *генерирования* предложений. Единственное изменение в работе, которое потребуется, состоит в том, что условия, которые проверяют следующие элементы в предложении, должны интерпретироваться в порождающем алгоритме как решение, которое следует принять и которое, если оно принято, налагает ограничения на порождение последующих фрагментов предложения. Трансформационная модель, напротив, есть почти исключительно порождающая модель. Проблема анализа для трансформационной грамматики столь исключительно сложна, что для трансформационных грамматик пока не найдено разумно эффективного распознающего алгоритма.

7.3. Эффективность представления. Одно из существенных преимуществ РСП-модели перед обычными КС-грамматиками — это возможность объединять общие части многих контекстно-свободных правил, что дает большую компактность записи. Например, одно регулярно записанное правило S → → (Q) (NEG)NP VP заменяет четыре правила:

$$S \rightarrow NP\ VP$$
$$S \rightarrow Q\ NP\ VP$$
$$S \rightarrow NEG\ NP\ VP$$
$$S \rightarrow Q\ NEG\ NP\ VP$$

в обычных контекстно-свободных обозначениях. РСП-модель часто может достичь и еще большей компактности путем объединения правил, благодаря отсутствию ограничений, связанных с линейностью записи, которые имеются при записи регулярных выражений.

Объединение избыточных частей правил не только дает более компактное представление, но и исключает избыточную работу в процессе анализа. То есть, сокращая объем записи грамматики, мы сокращаем также число проверок, которые надо проделать при анализе. Дело в том, что независимо от того, применялось правило или нет в обычной КС-грамматике, в процессе его применения (или попытки применения) часто получается информация, которая может оказать воздействие на применимость или неприменимость последующих правил. Поэтому, когда два правила имеют общие части, при проверке применимости первого уже осуществляется часть тестов, требующихся для проверки применимости второго. Объединение общих частей

правил позволяет выиграть за счет использования этой информации, исключив избыточную работу при применении второго правила.

В дополнение к прямому объединению общих частей различных правил, которое происходит при построении нерасширенной сети, РСП благодаря использованию пометок допускает объединение *сходных* частей сети путем запоминания информации в регистрах и обращения к этой информации в условиях при дугах. Так, можно запомнить в регистрах некоторую информацию, которая иначе неявно запоминалась бы состоянием сети, и объединить состояния, которые имеют переходы, одинаковые во всем, кроме условий на содержимое регистров. Например, рассмотрим два состояния, переходы которых одинаковы с тем исключением, что одно из них «помнит», что в предложении уже была найдена отрицательная частица, тогда как другое позволяет переход, который допускает отрицательную частицу. Эти два состояния можно объединить, введя пометку, указывающую на то, что отрицательная частица уже была ранее, и поместив при дуге, которая допускает отрицательную частицу, условие, блокирующее эту дугу, при наличии пометки.

Процесс объединения сходных частей сети с использованием пометок, хотя и дает более компактное представление, но не уменьшает время работы и обычно даже требует небольшого увеличения времени. Причина в том, что требуется дополнительное время для проверки условий и появляются дополнительные дуги, которые приходится пробовать, даже если потом условия запретят пройти по ним. В пределе (доводя до абсурда) можно свести любую сеть к сети с одним состоянием, используя пометку для каждой дуги и помещая при дугах условия, которые запрещают проход по этой дуге, если не появилась соответствующая пометка на одной из дуг, предшествующих данной. Явная неэффективность здесь состоит в том, что на каждом шаге нужно будет рассматривать каждую дугу сети и применять сложные тесты для выяснения того, можно ли пройти по этой дуге. Поэтому имеется «обратная пропорциональность» между компактностью представления, которую можно получить, используя пометки, и возможным возрастанием времени работы. Представляется, что это еще один пример постоянного конфликта «место — время», который возникает почти во всех проблемах, программируемых для ЭВМ.

Во многих случаях использование регистров для хранения частичных анализов автоматически порождает пометки, так что нет необходимости заводить специальные регистры для их запоминания. Например, на наличие где-то раньше в предложении отрицательной частицы может указывать непустота регистра NEG, который содержит эту частицу. Точно так же на на-

личие вспомогательного глагола указывает непустота регистра AUX, который содержит вспомогательный глагол.

7.4. Улавливание закономерностей. Целью лингвистики при создании грамматик естественных языков является улавливание *закономерностей* в языке. Это значит, что если существует регулярный процесс, который работает в ряде контекстов, грамматика должна описать этот процесс единым механизмом или правилом, а не многими независимыми копиями одного и того же процесса для разных контекстов, в которых он может проявиться. Простым примером этого принципа является представление предложной группы как составляющей предложения, поскольку конструкция, состоящая из предлога, за которым следует именная группа, появляется часто в английских предложениях в самых разных окружениях. Поэтому модель, в которой предложные группы не будут рассматриваться как составляющие, упустит закономерное явление. Этот принцип есть разновидность *принципа экономии*, который говорит, что наилучшая грамматика — это та, в которой язык характеризуется описанием с наименьшим числом символов. Грамматика, в которой будут независимо существовать копии одной и той же информации, затратит лишние символы при описании языка, и та модель, которая объединит эти многочисленные копии в одну, даст лучшую грамматику, так как истратит меньше символов. Поэтому принцип экономии ведет к отдаче предпочтения грамматикам, которые охватывают закономерности.

Сетевая модель с добавлением произвольных условий при дугах и использованием регистров, содержащих пометки и частичные конструкции, дает средства для опознания и улавливания закономерностей. Всякий раз, когда в грамматике оказываются два или более подграфов любой размерности, которые являются по существу копиями друг друга, это свидетельствует об упущенной закономерности. Это значит, что имеются две по существу идентичные части в грамматике, которые отличаются только тем, что управляющая часть машины помнит некоторые сведения за счет различия состояний, но, за исключением этого, работа этих двух частей графа идентична. Чтобы охватить эту закономерность, достаточно явно запомнить различающие данные в регистре (например, пометкой) и оставить только один экземпляр подграфа.

7.5. Эффективность работы. В дополнение к выигрышу, который получается благодаря объединению общих частей разных правил, РСП-модель дает ряд других преимуществ в отношении эффективности работы. Одно из них — это возможность отложить принятие решения, используя переработку сети. Во мно-

гих грамматиках естественных языков очень неэффективна процедура, при помощи которой грамматика пытается угадать некоторые основные свойства конструкции очень рано в процессе анализа, например пытается угадать, является ли предложение активным или пассивным, прежде, чем начался анализ предложения. Из-за этого в анализе прослеживаются несколько альтернатив, пока не дойдут до того места в предложении, где имеется достаточно информации, чтобы отбросить ошибочный вариант. Гораздо привлекательнее подход, при котором можно отложить решение, пока не дойдем до места, где есть необходимая информация для принятия решения. РСП-модель дает эту возможность.

Используя стандартные методы оптимизации конечных автоматов (Вудс [32]), можно «оптимизировать» сеть переходов, сделав ее детерминированной во всем, кроме магазинных операций (где недетерминизм может быть уменьшен, но не исключен). Это значит, что, если несколько дуг с одним и тем же символом выходят из одного состояния, может быть построена модифицированная сеть, в которой из любого состояния выходит не более одной дуги с данным символом. Это дает повышение эффективности работы благодаря сокращению числа «активных» конфигураций, которые надо проследить в процессе работы. В детерминированной сети варианты анализа, которые выглядят одинаково, объединены до момента, когда они перестают быть идентичными; т. е. решение о том, на каком пути мы находимся, откладывается до момента, когда два пути расходятся, причем в этом пункте символ на входе обычно определяет, какой путь является правильным. РСП, возможно, не допускают полностью автоматическую оптимизацию, которую допускают нерасширенные сети, но и к ним можно применить общий метод сокращения числа активных конфигураций путем уменьшения недетерминизма сети, тем самым откладывая решение о выборе варианта до того места входной цепочки, где они действительно различаются. Хранение фрагментов анализа в регистрах, пока их функция не будет определена, позволяет подождать момента принятия такого решения прежде, чем строить синтаксическое представление, которое может зависеть от того, какое решение принято. Эти возможности позволяют отложить принятие решения даже тогда, когда строится глубинная структура того типа, какие даются трансформационной грамматикой.

Необходимость прослеживать несколько активных конфигураций в процесс анализа есть следствие потенциальной неоднозначности естественного языка. Источник неоднозначности в сети лежит в рекурсивности ее работы, так как без рекурсии сеть была бы конечным автоматом, и ее можно было бы сделать полностью детерминированной. Мы показали в другой статье

[32], что можно убрать многие рекурсии из сети переходов (в действительности можно исключить все рекурсии, кроме тех, которые получаются от самовставляющихся символов), и тем самым сократить еще больше число активных конфигураций, которые надо прослеживать. В РСП-модели можно, используя условия при дугах, определить однозначно, когда нужно перейти на более глубокий уровень рекурсии и оставить в качестве источника неоднозначности и причины множественности активных конфигураций только операцию подъема к более высокому уровню. Использование соответствующих условий (включая семантические) при POP-дугах сети дает возможность еще больше сократить здесь неоднозначность.

Одним из наиболее интересных свойств использования регистров в РСП является возможность принимать временные гипотетические решения о структуре предложения и затем менять их при дальнейшем рассмотрении предложения, не совершая при этом возврата. Например, когда тот, кто анализирует предложение, в месте, где он ожидает глагол, наталкивается на глагол "be", он может временно счесть его основным глаголом предложения, поместив его в регистр основного глагола. Если он затем встретит второй глагол, указывающий, что "be" был не основным, а вспомогательным глаголом, глагол "be" может быть перемещен из регистра основного глагола в регистр вспомогательного, а новый основной глагол встанет на его прежнее место. Эта методика, как и другие, ведет к сокращению числа активных конфигураций, которые должны быть прослежены в процессе анализа. В разд. 8 мы даем пример, где многократно демонстрируется методика принятия гипотетических решений, которые потом заменяются другими.

7.6. Легкость экспериментирования. Может быть наиболее важным преимуществом РСП-модели является то, что она удобна для экспериментальных лингвистических работ. Открытое множество базисных операций, которые могут быть использованы при дугах, дает возможность сформировать фундаментальное множество «естественных» операций для анализа естественных языков на основе опыта, получаемого при написании грамматик. Мощная функция BUILDQ была найдена таким путем и оказалась чрезвычайно полезной на практике. В разд. 8 есть другие примеры формирования специальных «естественных» операций, в которых встретилась необходимость: использование «накопителя» (hold list) и «виртуальных» переходов.

Другая область, где экспериментирование облегчается благодаря использованию РСП-модели, — это исследование разных типов представлений структур. Явно заданные операции построения структур при дугах сети позволяют экспериментиро-

вать с различными синтаксическими представлениями, такими, какие дают грамматики зависимостей, тагмемные формулы (tagmemic formula) или грамматика падежей Филмора [11]. По-видимому, с помощью действий по построению структур при дугах можно даже получить некоторые виды семантических представлений.

Наконец, можно использовать условия при дугах для экспериментов с разного рода семантическими условиями, позволяющими направить анализ и сократить число «бессмысленных» синтаксических структур. В рамках РСП можно начинать пользоваться многими внесинтаксическими сведениями, которыми, видимо, пользуется человек в процессе анализа. Многие полезные идеи в этой области остались неопробованными из-за отсутствия подходящего формализма.

8. ВТОРОЙ ПРИМЕР

В этом разделе мы приведем пример, иллюстрирующий некоторые из преимуществ РСП, которые мы уже обсуждали, — в особенности возможность принимать гипотетические решения, которые меняются при дальнейшем анализе. Рис. 4 дает фрагмент сети переходов, который описывает поведение вспомогательных глаголов "be" и "have", указывающих на пассивность конструкции или перфектность. Мы рассмотрим анализ, который дает этот образец сети для предложения "John was believed to have been shot" — предложение с довольно сложной синтаксической структурой. При этом мы увидим, что РСП ясно характеризует изменение предположений в ходе анализа и что это делается без необходимости возвратов или рассмотрения различных альтернатив.

Рис. 4 разделен на три части: (a) — графическое представление сети с нумерованными дугами, (b) — описание условий и форм, сопоставленных заключительным состояниям, (c) — список условий и действий, сопоставленных дугам сети. На рис. 4 (a) S, NP и VP — нетерминальные символы, AUX и V — названия лексических категорий; по дугам, помеченным TO или BY, можно пройти, только если на входе появилось слово "to" или "by" соответственно. Пунктирная дуга с символом NP есть особая «виртуальная» дуга, по которой можно пройти, если ранее командой HOLD в специальный «накопитель» была помещена группа существительного. После ее использования при проходе по виртуальной дуге эта группа из накопителя удаляется. Накопитель — это звено экспериментальной анализирующей системы, дающее естественное средство для обработки составляющих, которые найдены не на месте и должны быть поставлены на их обычное место прежде, чем анализ будет закон-

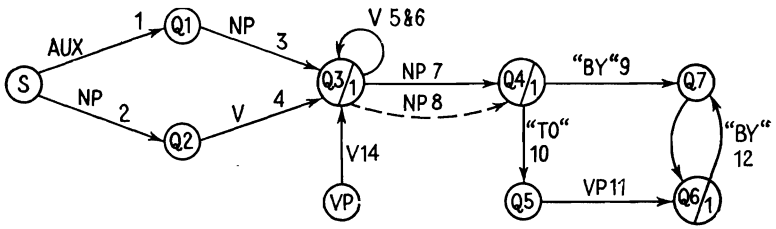


Рис. 4 (a). Фрагмент сети — графическое представление.

Q3: Условие: (INTRANS (GETR V))
Форма: (BUILDQ (S + + (TNS +) (VP (V +))) TYPE SUBJ TNS V)
Q4 и Q6: Условие: T
Форма: (BUILDQ (S + + (TNS +) (VP (V +) +)) TYPE SUBJ TNS V OBJ)

Рис. 4 (b). Фрагмент сети — условия и формы для заключительных состояний.

Условия	*Действия*
1. T	(SETR V *) (SETR TNS (GETF TENSE)) (SETR TYPE (QUOTE Q))
2. T	(SETR SUBJ *) (SETR TYPE (QUOTE DCL))
3. T	(SETR SUBJ *)
4. T	(SETR V *) (SETR TNS (GETF TENSE))
5. (AND (GETF PPRT) (EQ (GETR V) (QUOTE BE)))	(HOLD (GETR SUBJ)) (SETR SUBJ (BUILDQ (NP (PRO SOMEONE))) (SETR AGFLAG T) (SETR V *)
6. (AND (GETF PPRT) (EQ (GETR V) (QUOTE HAVE)))	(SETR TNS (APPEND (GETR TNS) (QUOTE PERFECT))) (SETR V *)
7. (TRANS (GETR V))	(SETR OBJ *)
8. (TRANS (GETR V))	(SETR OBJ *)
9. (GETR AGFLAG)	(SETR AGFLAG NIL)
10. (S-TRANS (GETR V))	(SENDR SUBJ (GETR OBJ)) (SENDR TNS (GETR TNS)) (SENDR TYPE (QUOTE DCL))
11. T	(SETR OBJ *)
12. (GETR AGFLAG)	(SETR AGFLAG NIL)
13. T	(SETR SUBJ *)
14. (GETF UNTENSED)	(SETR V *)

Рис. 4 (c). Фрагмент сети — условия и действия при дугах.

чен. При элементах, помещаемых в накопитель, помечается уровень, на котором они заносятся в накопитель, и алгоритм не может подняться с этого уровня, пока элемент не будет «использован» виртуальным переходом на этом уровне или на некотором более глубоком уровне.

Заключительные состояния отмечены на рис. 4 (а) косой чертой с индексом 1, обозначение, которое является обычным в конечных автоматах. Необходимые условия для подъема из заключительного состояния и выражение, которое определяет выдаваемое при этом значение, указаны на рис. 4 (b). Скобочная запись дерева такая же, как в разд. 3.2. Условия TRANS и INTRANS проверяют, является ли глагол переходным (транзитивным) или непереходным соответственно, и условие S-TRANS проверяет наличие таких глаголов, как "believe" (верить) и "want" (хотеть), каждый из которых может иметь «объектом» вставленное номинализованное предложение. Признаки PPRT и UNTENSED соответственно указывают на форму причастия прошедшего времени (past participle) и на глагольные формы, не выражающие времени.

Мы начинаем анализ предложения "John was believed to have been shot" с состояния S, рассматривая первое слово предложения John. Так как "John" есть имя собственное, то поиск именной группы со спуском с дуги 2 окончится успешно, и будет выполнено действие, приписанное этой дуге, тем самым именная группа (NP (NPR JOHN)) будет записана в регистр SUBJ и будет отмечено, что предложение является повествовательным, тем, что в регистр TYPE будет занесено DCL. Второе слово предложения "was" разрешает переход по дуге 4, при этом в регистр V записывается неопределенная форма глагола "be", а время предложения записывается в регистр TNS (tense). Содержимое регистров в этот момент соответствует гипотетическому решению о том, что глагол "be" есть основной глагол предложения, и если бы дальше следовала именная группа или прилагательное (не показанное на схеме сети в примере), то это решение осталось бы неизменным.

В состоянии Q3 ввод причастия прошедшего времени "believed" указывает, что предложение имеет пассивную форму и что глагол "be" есть лишь вспомогательный глагол, означающий пассив. В данном случае будет пройдена дуга 5, поскольку входное слово есть причастие прошедшего времени и текущее значение регистра глагола есть глагол "be". Это приводит к пересмотру гипотетического решения, причем прежнее гипотетическое подлежащее заносится в специальный накопитель, создается новое гипотетическое подлежащее (неопределенное) и записывается пометка AGFLAG, которая указывает, что субъектом может быть стоящая дальше агентивная именная группа, вво-

димая предлогом "be". Основной глагол теперь изменяется с "be" на "believe", и сеть возвращается в состояние Q3 для рассмотрения слова "to". Содержимое регистров в этот момент следующее:

SUBJ: (NP) PRO SOMEONE))
TYPE: DCL
V: BELIEVE
TNS: PAST
AGFLAG: T

и в накопителе содержится именная группа (NP (NPR JOHN)).

Ни одна из дуг, выходящих из состояния Q3, не соответствует входному слову "to". Однако наличие именной группы "John" в накопителе позволяет виртуальный переход по дуге 8 точно так же, как если бы эта именная группа была обнаружена в данном месте предложения. (Переход разрешен, поскольку глагол "believe" является транзитивным.) Результатом является гипотетическое рассмотрение именной группы (NP (NPR JOHN)) как объекта глагола "believe". Если бы это был конец предложения и мы закончили бы разбор в заключительном состоянии Q4, то мы получили бы правильный анализ "Someone believed John".

Ввод слова "to" в состоянии Q4 говорит нам, что объектом глагола "believe" является не просто именная группа "John", но номинализованное предложение, для которого "John" является предположительным субъектом. Результатом прохода дуг 10 и 11 является засылка необходимой информации для вычисления на более глубоком уровне, которое выявит полностью вставное предложение и выдаст результат в качестве объекта глагола "believe". Дуга 10 подготовляет засылку именной группы (NP (NPR JOHN)) как субъекта вставного предложения, времени PAST (прошедшее) и типа DCL (повествовательное). С дуги 11 происходит затем спуск к состоянию VP к рассмотрению слова "have".

В этот момент мы ведем обработку вставного предложения со следующим содержимым регистров:

SUBJ: (NP (NPR JOHN))
TYPE: DCL
TNS: PAST

По дуге 14 возможен переход, если текущее входное слово есть глагол в форме, не выражающей времени и наклонения, так как, например, нельзя сказать "John was believed to *has* been shot". Поскольку "have" именно такая форма, переход возмо-

жен, и основным глаголом вставного предложения предположительно считается "have", что годилось бы для предложения "John was believed to have money".

Причастие прошедшего времени "been", следующее за "have", вызывает переход 6, выявляющий тот факт, что вставное предложение стоит в перфекте (результат наличия вспомогательного глагола "have"), и предлагающий новый гипотетический основной глагол "be", который годился бы для предложения "John was believed to have been a druggist". Содержимое регистров для вычисления глубокого уровня в этот момент следующее:

SUBJ: (NP (NPR JOHN))
TYPE: DCL
TNS: PAST PERFECT
V: BE

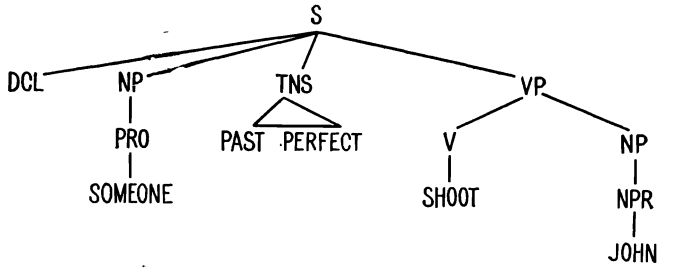
Снова в состоянии Q3 ввод причастия прошедшего времени shot при предположительном глаголе "be" в регистре глагола указывает, что предложение стоит в пассиве и переход 5 помещает именную группу (NP (NPR JOHN)) в накопитель и формирует неопределенное подлежащее (NP (PRO SOMEONE)). Хотя мы теперь пришли к концу предложения, но и наличие именной группы в накопителе, и тот факт, что глагол shoot является переходным, препятствуют окончанию работы алгоритма. Вместо этого происходит виртуальный переход по дуге 8, приписывающий именную группу "John" в качестве объекта глаголу "shoot". Содержимое регистров после вычисления глубокого уровня теперь следующее:

SUBJ: (NP (PRO SOMEONE))
TYPE: DCL
TNS: PAST PERFECT
V: SHOOT
AGFLAG: T
OBJ: (NP (NPR JOHN))

В этот момент мы находимся в конце предложения и в заключительном состоянии Q4 с пустым накопителем, так что от вычисления глубокого уровня можно вернуть управление на более высокий уровень вычисления, с которого мы спустились. Выдаваемое значение, как это определяет форма, приписанная состоянию Q4, есть

(S DCL (NP (PRO SOMEONE)) (TNS PAST PERFECT) (VP (V SHOOT) (NP (NPR JOHN))))

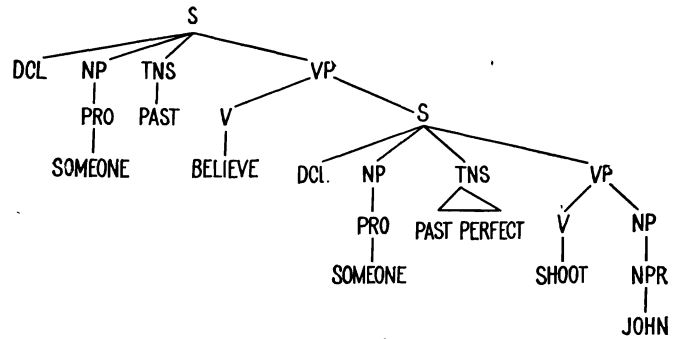
что соответствует дереву:



Вычисление более высокого уровня продолжается действием при дуге 11, засылающим в регистр OBJ результат вычисления глубокого уровня. Так как высокий уровень работы также привел к заключительному состоянию Q6, предложение принимается и приписываемая ему структура (как это определяет форма, соответствующая состоянию Q6) есть

(S DCL (NP (PRO SOMEONE)) (TNS PAST) (VP (V BELIEVE)
(S DCL (NP (PRO SOMEONE)) (TNS PAST PERFECT) (VP (V
SHOOT) (NP (NPR JOHN))))))

которая в виде дерева представляется следующим образом:



Эта структура может соответствовать перифразе: "Someone believed that someone had shot John". Если бы за предложением следовала бы группа "by Harry", то были бы возможны две интерпретации в зависимости от того, была ли эта дополнительная группа воспринята как допустимая при обработке на глубоком или на верхнем уровне. В каждом случае одно из неопределенных подлежащих SOMEONE было бы заменено определенным подлежащим "Harry". Структура, получающаяся в одном случае, представляется предложением: "Someone believed that Harry had shot John", тогда как в другом случае имеем: "Harry believed that someone had shot John".

9. АНАЛИЗ ПРИ ПОМОЩИ СЕТЕВЫХ ГРАММАТИК

Поскольку грамматика с нерасширенной сетью переходов есть в действительности не более чем другая организация элементов автомата с магазинной памятью, ряд существующих алгоритмов анализа, сделанных для КС-грамматик, применимы более или менее непосредственно к сетевой модели. Две основные стратегии анализа «сверху-вниз» и «снизу-вверх» имеют аналоги для рекурсивных сетей переходов, и противопоставляемые стратегии обработки неоднозначных предложений — прослеживание всех вариантов анализов «параллельно» или прослеживание вариантов анализа по одному (с запоминанием информации в местах выбора пути) — обе применимы для этого типа модели. В частности, один из наиболее мощных и эффективных анализирующих алгоритмов для КС-грамматик — распознающий алгоритм Эрли ([9, 10]) — может быть приспособлен с минимальным изменением к использованию для сетевых грамматик, при этом может возрасти эффективность его работы.

Алгоритм Эрли, тщательно прослеживая все варианты параллельно, получает представление всех возможных вариантов анализа цепочки по отношению к КС-грамматике за время, которое может быть ограничено величиной Kn^3, где n — длина цепочки, а K — константа, зависящая только от грамматики и не зависящая от входной цепочки. Более того, для некоторых подклассов КС-грамматик можно показать, что граница требуемого времени может быть снижена до n^2 для линейных грамматик и некоторых других и до n для $LR(k)$-грамматик со считыванием вперед k символов (Кнут [17]) и некоторых других. Многие из этих результатов были получены с другими алгоритмами, использующими специальные свойства различных классов грамматик [15, 16, 33], но алгоритм Эрли — единственный алгоритм, который для любой КС-грамматики (без ограничений на форму, отсутствие левой рекурсии и т. п.) работает со временем, ограниченным n^3. Кроме того, он *автоматически* достигает более низких границ для специальных классов грамматик, не нуждаясь в указании о том, что грамматика принадлежит специальной разновидности (т. е. он не использует специальных приемов для этих случаев).

Мы дали в другой работе (Вудс [32]) модифицированную версию алгоритма Эрли, которая может быть использована для анализа предложений по отношению к грамматике (нерасширенной) сети переходов с той же границей для времени, и показали там, что можно применять к грамматике сети переходов ряд механических «оптимизирующих» приемов, которые уменьшат коэффициент пропорциональности в оценке времени.

Более сложна проблема анализа для *расширенных* сетей переходов с использованием алгоритма Эрли из-за занесения информации в регистры и использования явно заданных действий по построению структур. Возможность переходов, обусловленных содержимым регистров, затрудняет установление того факта, что конфигурации «эквивалентны» и могут быть объединены для дальнейшей обработки, а использование регистров и явно заданных действий по построению структур усложняет задачу выбора подходящей формы представления для объединенных конфигураций. Можно довольно просто расширить алгоритм Эрли так, чтобы преодолеть эти трудности, но так как граница времени n^3 существенно зависит от объединения эквивалентных конфигураций и фиксированной границы времени для осуществления каждого перехода из (объединенного) состояния, не ясно, какая оценка времени получится для такого алгоритма.

Если мы различаем регистры «пометок», которые могут содержать только пометки из фиксированного конечного набора и регистры «произвольного содержимого», которые могут содержать любую структуру, и если мы ограничиваем условия и действия при дугах так, что (1) условия могут касаться только регистров пометок и символов входной цепочки (например, со считыванием вперед), (2) сами условия и действия требуют ограниченного времени, (3) есть только один регистр, к которому можно добавлять в конец и заполнять при помощи функции BUILDQ, но в который нельзя «заглядывать» внутрь, тогда мы можем построить разновидность распознающего алгоритма Эрли, который будет работать в пределах границы n^3 для времени (или границ n^2 и n в специальных случаях). Однако если мы заметно ослабим эти ограничения, то возрастание границы времени неизбежно, например условия или действия могут сами требовать больше чем n^3 шагов.

9.1. О последовательном анализе. Во многих приложениях анализа естественных языков не является необходимым (или хотя бы желательным) получение представления всех возможных вариантов анализа входного предложения. В приложениях, где естественный язык является средством общения человека с машиной, важнее выбрать анализ, «наиболее правдоподобный» в данном контексте, и сделать это как можно скорее. Несомненно, будут случаи, когда имеется несколько «одинаково правдоподобных» анализов или где «наиболее правдоподобный» анализ оказывается «неправильным» (т. е. не тем, который имел в виду говорящий); поэтому все-таки нужно иметь недетерминированный алгоритм и нужно иметь способ для нахождения любого конкретного варианта анализа, если это потребуется.

Однако вовсе не обязателен параллельный подход, который растрачивает время, строя все варианты анализа сразу. В таких приложениях последовательный подход (снабженный механизмом для выбора варианта анализа, который надо рассматривать первым) подходит больше, чем параллельный подход, так как в большинстве случаев он позволит избежать прослеживания других альтернатив. Можно (при желании) сохранять запись результата анализа правильно построенных подцепочек, найденных предыдущими альтернативными вариантами (чтобы исключить повторный анализ тех же самых подцепочек), но выигрыш во времени анализа при таком подходе не всегда оправдывает расход памяти, требующейся для запоминания всех результатов анализа частичных подцепочек.

Успех последовательного анализа, описанного выше, зависит, конечно, от наличия средств для выбора семантически «наиболее правдоподобного» варианта, который нужно проследить первым. РСП-грамматика дает несколько таких средств. Во-первых, упорядочивая дуги, которые выходят из некоторого состояния сети, можно тем самым соответственно упорядочить получающиеся анализы. Составитель грамматики может так подобрать этот порядок с целью максимизировать «априорное правдоподобие» (зависящее только от структуры предложения, как она получается при данной грамматике, и ни от чего больше), что анализ, выбранный первым, будет правильным. Далее, повторяя некоторые дуги с разными условиями, можно сделать этот порядок зависящим от специфических особенностей анализируемого предложения, в частности можно поставить его в зависимость от семантических свойств слов, входящих в предложение. Два дополнительных средства для выбора «наиболее правдоподобного» анализа были добавлены к модели в реализованной экспериментальной анализирующей системе: специальный регистр «веса», содержащий оценку «правдоподобности» текущего анализа (который может быть использован для откладывания просмотра неправдоподобно выглядящих путей и рассмотрения в первую очередь более правдоподобных), и специальные приемы для подчинения обстоятельств, которые используют семантическую информацию с целью определения «наиболее правдоподобного» управляющего для обстоятельств в предложении.

10. РЕАЛИЗАЦИЯ

Экспериментальная анализирущая система, основанная на изложенных здесь идеях, была реализована на BBN LISP с помощью системы разделения времени SDS 940 в Гарвардском университете и в компании «Болт, Беранек и Ньюмен», и ис-

пользуется для многочисленных экспериментов по построению грамматик и исследованию стратегий анализа для естественных языков. Целью этой реализации было получение достаточно гибких средств для экспериментирования и усовершенствования системы, и по этой причине был использован модульный принцип организации, который сам допускает развитие и расширение без больших изменений в общей структуре системы. Система уже прошла несколько циклов эволюции, при этом у нее появилось большое число новых свойств и еще больше ожидается при продолжении исследований.

Реализованная система содержит общие средства для семантической интерпретации (описанные у Вудса [30, 31]), и основным мотивом для реализации было желание исследовать взаимодействие синтаксического и семантического аспектов в процессе «понимания» предложения. Особое значение придавалось использованию семантической информации для управления анализом, минимизации числа тупиковых путей, которые приходится просматривать, и упорядочиванию вариантов анализа предложения в отношении некоторой меры «правдоподобия». Сейчас экспериментальная система включает специальные приемы для подчинения обстоятельств, использующие семантическую информацию, несколько подходов к проблеме сочинительных конструкций (в том числе сочиненных фрагментов предложения), и средств для лексического и морфологического анализа. При помощи этой системы были построены и опробованы несколько различных грамматик и были исследованы и исследуются в настоящее время большое число английских конструкций и стратегий анализа. Готовится отчет о деталях этой реализации и проведенных экспериментах.

СПИСОК ЛИТЕРАТУРЫ

1. Bobrow D. G., Fraser J. B., An augmented state transition network analysis procedure, Proc. Internat. Joint Conf. on Artificial Intelligence, Washington, D. C., 1969, pp. 557—567.
2. Bobrow D. G., Murphy D., Teitelman W., BBN Lisp System. Bolt, Beranek and Newman Inc., Cambridge, Mass., 1968.
3. Book R., Even S., Greibach S., Ott G., Ambiguity in graphs and expressions, Mimeo. rep., Aiken Computat. Lab., Harvard U., Cambridge, Mass., 1969.
4. Cheatham T. E., Sattley K., Syntax-directed compiling, Proc. AFIPS 1964 Spring Joint Comput. Conf., v. 25, Spartan Books, New York, pp. 31—57.
5. Chomsky N., Formal properties of grammars, *in* Handbook of Mathematical Psycology, v. 2, R. D. Luce, R. R. Bush and E. Galanter (Eds.), Wiley, New York, 1963. (Русский перевод: Хомский Н. Формальные свойства грамматик. Кибернетический сборник, новая серия, вып. 2, «Мир», М., стр. 121—230.)
6. Chomsky N. A transformational approach to syntax, *in* The Structure of Language, J. A. Fodor and J. J. Katz (Eds.), Prentice-Hall, Englewood Cliffs, N. J., 1964.

7. Chomsky N., Aspects of Theory of Syntax, MIT Press, Cambridge, Mass., 1965.
8. Conway M. E., Design of a separable transition-diagram compiler, *Comm. ACM* **6**, 7 (July 1963), 396—408.
9. Earley J., An efficient context-free parsing algorithm, Ph. D. th. Dep. Computer Sci., Carnegie-Mellon U., Pittsburgh, Pa., 1968.
10. Earley J., An efficient context-free parsing algorithm, *Comm. ACM* **13**, 2 (Feb. 1970), 94—102. (Русский перевод: Эрли Дж., Эффективный алгоритм анализа контекстно-свободных языков, в сб. «Языки и автоматы», «Мир», М., 1975, стр. 47—70.)
11. Fillmore C. J., The case for case, *in* Universals in Linguistic Theory, E. Bach and R. Harms (Eds.), Holt, Rinehart and Winston, New York, 1968.
12. Ginsburg S., The Mathematical Theory of Context-Free Languages, McGraw-Hill, New York, 1966. (Русский перевод: Гинзбург С., Математическая теория контекстно-свободных языков, «Мир», М., 1970.)
13. Greibach S. A., A simple proof of the standard-form theorem for context-free grammars, *in* Mathematical linguistics and automatic translation, Rep. NSF-18, Comput. Lab., Harvard U., Cambridge, Mass., 1967.
14. Herringer J., Weiler M., Hurd E., The immediate constituent analyzer, *in* Rep. NSF-17, Aiken Comput. Lab., Harward U., Cambridge, Mass., 1966.
15. Kasami T., An efficient recognition and syntax-analysis algorithm for context-free languages, Sci. Rep. AFCRL-65-558, Air Force Cambridge Res. Lab., Bedford, Mass., 1965.
16. Kasami T., A note on computing time for recognition of languages generated by linear grammars, *Inform. Contr.*, **10** (1964), 209—214.
17. Knuth D. E., On the translation of languages from left to right, *Inf. Contr.*, **8** (1965), 607—639. (Русский перевод: Кнут Д., О переводе (трансляции) языков слева направо, в сб. «Языки и автоматы», «Мир», М., 1975, стр. 9—42.)
18. Kuno S., A system for transformational analysis, *in* Rep. NSF-15, Comput. Lab. Harvard U., Cambridge, Mass., 1965.
19. Kuno S., Oettinger A. G., Multiple path syntactic analyzer, *in* Information Processing 1962, North-Holland Publishing Co., Amsterdam, 1963. (Русский перевод: Куно С. и Эттингер А., Многовариантный синтаксический анализатор, в сб. «Автоматический перевод», «Прогресс», М., 1971, стр. 102—120.)
20. McCarthy J., et al., LISP 1.5 programmer's manual, MIT Comput. center, Cambridge Mass., 1962.
21. McCawley J. D., Meaning and the description of languages, *in* Kotoba No Ucho, Tec. Co. Ltd., Tokyo, 1968.
22. McNaughton R. F., Yamada H., Regular expressions and state graphs for automata, *IRE Trans.* **EC-9** (Mar. 1960), 39—47.
23. Matthews G. H., Analysis by synthesis of natural languages, Proc. 1961 Internat. Conf. on Machine Translation and Applied Language Analysis, Her Majesty's Stationary Office, London, 1962.
24. MITRE. English preprocessor manual, Rep. SR-132, The Mitre Corp., Bedford, Mass., 1964.
25. Ott G., Feinstein N. H., Design of sequential machines from their regular expressions, *J. ACM* **8**, 4 (Oct. 1961), 585—600.
26. Petrick S. R., A recognition procedure for transformational grammars, Ph. D. th., Dep. Modern Languages, MIT, Cambridge, Mass., 1965.
27. Postal P. M., Limitations of phrase structure grammars, *in* The Structure of Language, J. A. Fodor and J. J. Katz (Eds.), Prentice-Hall, Englewood Cliffs, N. J., 1964.

28. Schwarcz R. M., Steps toward a model of linguistic performance: A preliminary sketch, *Mechanical Translation*, **10** (1967), 37—52.
29. Thorne J., Bratley P., Dewar H., The syntactic analysis of English by machine, *in* Machine Intelligence 3, D. Michie (Ed.), American Elsevier, New York, 1968.
30. Woods W. A., Semantics for a question-answering system, Ph. D. th., Rep. NSF-19, Aiken Comput. Lab., Harvard U., Cambridge, Mass., 1967.
31. Woods W. A., Procedural semantics for a question-answering machine, Proc. AFIPS 1968 Fall Joint. Comput. Conf., 33, Pt. 1, MDI Publications, Wayne, Pa., pp. 457—471.
32. Woods W. A., Augmented transition networks for natural language analysis, Rep. CS-1, Comput. Lab., Harvard U., Cambridge, Mass., 1969.
33. Younger D. H., Context free language processing in time n^3, G. E. Res. and Devel. Center, Schenectady, N. Y., 1966

Искусственный интеллект как психология[1])

М. Б. Клауз

Слово «психология» как в обыденной речи, так и в «интеллектуальном» научном контексте имеет сугубо «человеческий» смысл, резко отличаясь от коннотаций термина «искусственный интеллект», которые должны обязательно включать вычислительные машины, роботов и разного рода системы «механических» устройств. Тем не менее многие специалисты, работающие в этой области, начали рассматривать искусственный интеллект как новый подход к сугубо психологическим проблемам и настаивают на том, что он должен составлять важную, а может быть, и основную часть любого действительно научного исследования в области психологии.

Каким же образом методология, в которой машина играет основную роль, может способствовать изучению явлений, присущих исключительно живым биологическим системам? Для того чтобы ответить на этот вопрос, следует в первую очередь определить «заботы» искусственного интеллекта. Это можно сделать, рассмотрев несколько разновидностей машинных программ, в частности шахматные программы, программы, отвечающие на вопросы на английском языке, программы для доказательства теорем, программы для решения головоломок и программы, способные воспринимать «трехмерные ситуации». Нет необходимости (и места) обсуждать все упомянутые виды программ, поскольку интересующие нас качества являются общими для них всех. В связи с этим мы сосредоточимся на одном функциональном типе, а именно восприятии трехмерных ситуаций или, как его часто называют, зрении роботов.

Представьте себе, что телевизионная камера направлена на стол, на котором рассыпаны детские кубики, и мы можем, преобразовав выходной сигнал камеры в цифровую форму, ввести его в вычислительную машину. Существует несколько программ, которые, используя в качестве входной информации телевизионный сигнал, пытаются решать такие задачи, как подсчет числа кубиков, определение их формы (т. е. кубы или треугольные призмы) или взаимного расположения (А лежит на В, С распо-

[1]) Clowes M. B., Artificial Intelligence as Psychology, *European AISB Bulletin*, № 1, 1972.

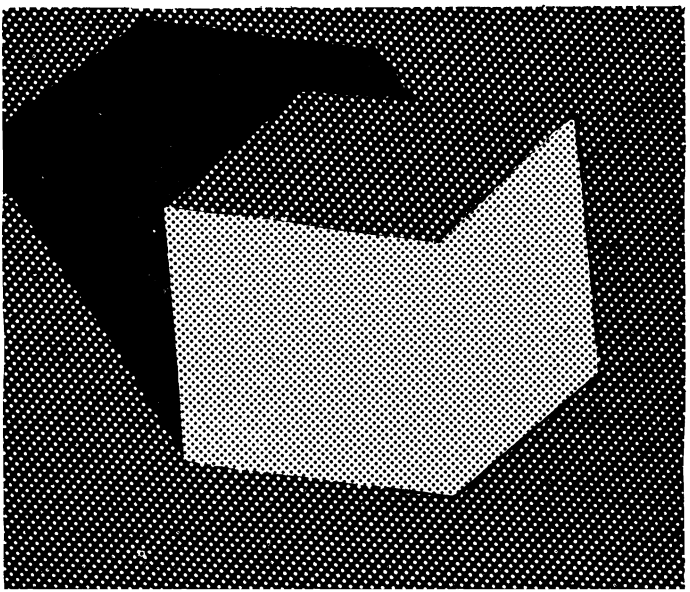
ложен перед А). Эти программы олицетворяют нынешний этап эволюции систем зрительного восприятия, охватывающий более десяти лет напряженных усилий их создателей. Первые варианты, ориентированные в основном на распознавание символов алфавита, а не на идентификацию форм твердых тел, имели программную память, хранящую в более или менее необработанном виде «прошлый опыт» этих систем — телевизионные изображения, названия которых («А», «В» и т. д.) сообщались системе. Распознавание (наименование) предъявленного изображения осуществлялось посредством подбора соответствующего изображения из памяти системы. Такой подход имеет миллион недостатков, и для их преодоления разрабатывались все более сложные формы организации прошлого опыта. Важной модификацией, в частности, было введение *структурного описания*, обеспечившее оценку организации изображения на основе «содержательных» принципов, скажем, разделение изображения на собственно фигуру и фон, разбиение фигуры на группы символов, отдельные символы, штрихи и т. д.

Подобные описания имеют очень много общего со структурными описаниями предложений естественного языка, предложенными Хомским и его сотрудниками, за исключением одного очень существенного отличия. По Хомскому, изучение форм, которые подобные описания принимают, может быть отделено от способа постановки их в соответствие собственно предложениям. Дело в том, что он занят не столько «употреблением» языка, сколько «языковой компетенцией». Необходимость создания программы, способной реагировать на внешние воздействия, заставляет разработчика определить *процесс*, посредством которого соответствующие структурные описания реализуются. Требуя, чтобы реакция системы была осмысленной, мы сталкиваемся с семантическими проблемами, которыми лингвисты, вообще говоря, пренебрегают.

Этот *процесс*, имеющий решающее значение для понимания нами феномена восприятия, «высвечивается» затруднениями, возникающими при разработке программ, способных адекватно «реагировать» на вышеупомянутые трехмерные ситуации. Допустим, что в программу заложены модели всех многогранников, которые потенциально могут быть предъявлены ей для опознания; структурные описания этих моделей представляют собой множества связанных вершин, координаты которых, заданные в некоторой трехмерной системе, известны. Для того чтобы распознать на входном телевизионном изображении аналоги этих моделей, требуется всего лишь отыскать на изображении точки, соответствующие вершинам многогранника, и посредством алгебраических преобразований, воспроизводящих обычные закономерности начертательной геометрии, совместить с ни-

ми соответствующую трехмерную модель. Естественно, при этом, по всей вероятности, требуется испытать массу моделей всевозможных размеров и расположенных всеми мыслимыми способами (т. е. привязанных ко всем потенциально возможным системам координат), однако можно предложить изощренные методы для сокращения объема подобного поиска. Еще несколько лет назад программа, основанная на таких принципах, успешно распознавала на фотоизображениях довольно значительное количество сложных сцен, образованных различными комбинациями детских кубиков. Последующие попытки перейти на основе этих принципов к «реальным» сценам, т. е. к условиям, когда положение камеры и освещение не контролируются столь тщательно, оказались в общем безуспешными по причинам, в высшей степени поучительным.

Для того чтобы иметь возможность применить проективные преобразования, обеспечивающие переход от трехмерной к двумерной системе координат, следует в первую очередь найти на изображении точки, соответствующие вершинам многогранника. Это всегда очень трудная задача, за исключением тех случаев, когда освещение очень хорошее, т. е. такое, которое может обеспечить лишь квалифицированный фотограф. В реальных условиях не менее сложно обнаружить на фотографии все линии, соответствующие ребрам, и части изображения, соответствующие граням многогранника.



Изучение этой фотографии свидетельствует о том, что мы в состоянии различить ребра и отдельные грани фигуры, несмотря на практическое отсутствие на изображении локальных признаков существования соответствующего разделения. Это происходит таким образом, что мы как бы воображаем грани, ребра и вершины, *руководствуясь механизмом, который можно было бы определить как трехмерное восприятие мира*, существующего вне данного изображения. Важным компонентом этого механизма восприятия, естественно, является знание законов перспективы, а также роли освещения и, в частности, особенностей образования теней, не говоря уже о предположительных типах объектов, с которыми мы можем столкнуться. Подразумевается и наличие предположений более общего характера, как, например, матовость, цветовая и текстурная однородность, а также и более специфических характеристик — кубы, треугольные призмы, цилиндры и т. п. Разработка программ для восприятия трехмерных объектов в настоящее время полностью определяется идеей, состоящей в задании такого набора ограничений некоему *механизму «решения задач»*, который будет использовать характерные признаки изображения (контрастные границы, равномерно освещенные области) для выдвижения гипотез и проверять их внутреннюю непротиворечивость с помощью этого набора ограничений. Содержательные выводы из этих гипотез следует привязать к участкам изображения, указываемым подобной «геометрически образованной» системой логического вывода, и таким образом обеспечивается распознавание ребер, вершин и граней, ни в коем случае не обнаруживаемых при обычном осмотре.

Неисчислимы последствия принятия такого подхода к реализации *процесса* восприятия трехмерных объектов. Хотя алгебраически многогранник можно задать, указав значения координат всех его вершин в трехмерной системе, предварительные гипотезы, формируемые на основе существенных признаков изображения, формулируются на совершенно ином языке. Обычно они сводятся к сравнительно приблизительным характеристикам типа «данная контрастная граница представляет собой ребро выпуклой фигуры, соединяющее две видимые грани, одна из которых освещена», «данная контрастная граница является тенью ребра, представленного на изображении контрастной границей», «данная контрастная граница образована тенью, лежащей на плоской поверхности», «данные контрастные границы являются ребрами одной и той же грани» и т. д. Описание, содержащее сведения о перспективе, освещенности и объектах, должно для того, чтобы абсорбировать как такой язык, так и более точные формулировки, выражающие столь тонкие нюансы, как различия между прямоугольниками и квадратами, включать все виды

побочных образований типа «теневых поверхностей» (плоскость, образованная источником света, ребром грани и соответствующей тенью), «поверхностей интерпретации» (плоскость, образованная точкой обзора, контрастной границей и гранью фигуры, которую она представляет). Более того, программа, обрабатывающая всю эту информацию, использует при этом способы, которые в большей степени отражают собственно систему знаний в данной области, чем это, вероятно, предполагается в какой-либо универсальной системе вывода (системе, выполняющей грамматический разбор), например в некоторых разновидностях лингвистических моделей. Язык программы свидетельствует больше о целях и их следствиях — гипотезах и их значении, чем о жестко расписанных последовательностях операций, которые должны неукоснительно исполняться.

В результате способы, которыми программа «обрабатывает» предъявленную ей сцену, подвержены значительным вариациям, поскольку одна и та же сцена, освещенная по-разному или рассматриваемая со слегка измененной позиции, будет представлена такими телевизионными изображениями, которые будут порождать абсолютно различные исходные гипотезы. Избыточность, имеющаяся в системе (точные и приблизительные описания сведений, закладываемых в нее), приводит к появлению более чем одного путей достижения цели, и большая оперативная память, требующаяся для реализации ведущегося методом проб и ошибок на различных множествах конкурирующих гипотез поиска, поставит систему решения задач в зависимость от других потребностей в памяти, возникающих параллельно. Объем и организация «знаний», сообщаемых программе, существенно влияют на процедуру интерпретации, обеспечивая таким образом потенциальную возможность проявления индивидуальных особенностей в работе программы. Можно предположить, что процесс, называемый обучением, глубоко связан с внесением дополнительных сведений, а в особенности с реорганизацией вышеупомянутого исходного массива «знаний».

Этот подход заостряет внимание на *«творческом»* характере процесса, называемого восприятием. «Творчество» возникает в связи с тем обстоятельством, что программа вносит в процесс решения задачи (и обязана делать это для того, чтобы надеяться на успех) сложную и обширную систему знаний, методов и процедур их использования. Было бы, однако, ошибочно предполагать, что в настоящее время существуют программы, которые могут убедительно продемонстрировать наличие таких творческих способностей. Самое большее, чем мы располагаем, — недавно созданная программа, предназначенная для понимания текста на английском языке и использующая сведения о контексте и английской грамматике для выделения «смысла» из

сложных предложений [1]). Существенно, что эта программа написана на языке, ориентированном на процесс решения задач и обладающем как раз теми характеристиками, о которых шла речь выше.

Эта «система, понимающая естественный язык» [2]), как и описанная нами программа зрительного восприятия, характеризует определенный этап в эволюционном процессе разработки программ, потребовавший около десяти лет напряженных усилий. Каждая очередная программа должна была справляться с такими ситуациями и уметь вырабатывать адекватные реакции в тех случаях, которые были недоступны предыдущим вариантам. Неполноценность новой разработки может быть установлена только в процессе экспериментальных исследований ее реализации. Последнее же представляет собой столь сложную задачу, что лишь цифровая вычислительная машина, предназначенная для обработки символьной информации, может справиться с ее реализацией. Этим и определяется смысл методологии искусственного интеллекта: вычислительная машина играет центральную роль, обеспечивая экспериментальное изучение все более сложных и «творчески одаренных» систем интеллекта и восприятия. Объектом такого изучения являются в сущности самые обычные факты: мы не пытаемся выискивать тончайшие нюансы или хвататься за секундомер, когда кто-то утверждает, что «время летит, как стрела». Именно использование системы знаний позволяет облечь повседневный опыт в чрезвычайно сложную, но все же целостную форму, придавая ему в то же время индивидуальный характер. Программы искусственного интеллекта, которые мы обсуждали начинают демонстрировать как собственно возникновение подобного процесса, так и логическую необходимость в нем.

Традиционная психология, сталкиваясь с индивидуальными особенностями подобных процессов, прибегает к различным способам. Экспериментальная психология для того, чтобы минимизировать

---

[1]) Winograd T. S., Procedures as a representation for data in a computer program for understanding natural language, MAC TR-84, Massachusetts Institute of Technology: Project MAC.

Название проекта расшифровывается как Machine-Aided Cognition, что приблизительно звучит как «познание с помощью вычислительной машины». Работы, посвященные отдельным темам, выполняемым в рамках этого проекта, можно на русском языке найти в двух выпусках сборника статей «Интегральные роботы», «Мир», М., 1973 (1), 1975 (2), в выпусках «Труды IV Международной объединенной конференции по искусственному интеллекту», АН СССР, Научный совет по комплексной проблеме «Кибернетика», Институт кибернетики АН Грузинской ССР, М., 1975. — *Прим. перев.*

[2]) В издательстве «Мир» в 1976 г. выходит русский перевод книги Винограда Т. «Системы, понимающие естественный язык», излагающей проблемы, связанные с построением систем, понимающих естественный язык, равно как и описывающей примеры таковых. — *Прим. перев.*

зировать индивидуальные различия и максимизировать воспроизводимость результатов эксперимента, последовательно уменьшала объем знаний, которые испытуемый должен был бы использовать при решении задачи. Следствием этого явилось постепенное снижение значимости результатов подобных исследований для объяснения закономерностей поведения в обычных условиях. Широкое использование изощренных методов статистической обработки в социальной психологии — пример еще одного способа «преодоления» неотъемлемых специфических особенностей «осмысленного» поведения. Те же, кто не может или не хочет жертвовать индивидуальными особенностями (психиатры, специалисты в области психологии обучения и другие), вынуждены удовлетворяться такими объяснениями неврозов, срывов процесса обучения и тому подобных расстройств, которые не допускают строгой проверки и часто являются не более чем произвольной апостериорностью.

Эти далеко идущие обобщения в сущности подвержены многим ограничениям. История психологии указывает много примеров, свидетельствующих о полезности попыток систематического объяснения поведения, определяемого системой знаний индивидуума. Классические работы Макса Вертгаймера, посвященные теории мышления, показывают, каким образом восприятие образует основу процесса мышления (еще один слой на готовый портрет процесса, основанного на восприятии) и, в частности, как различные схемы представления способствуют реализации процесса творческого мышления. Его концепция примечательна как объяснением роли графических схем в усвоении алгебраических понятий, так и (аналогично работам Пиаже) привлечением «диагностических» процедур для выявления системы знаний, обеспечивающей основу для формирования поведения. Полный отказ от количественных оценок, например, таких, как процент детей, выполнивших задание согласно определенному критерию, время, затраченное на получение решения, и т. п., который для столь многих исследователей эквивалентен потере научной строгости, является непосредственным результатом необходимости справиться со структурной сложностью поведения.

Заботы, связанные с проблемами представления, аналогичными изучавшимся Вертгаймером, в течение длительного времени играли первостепенную роль в программах искусственного интеллекта, предназначенных для решения головоломок[1]). Основная трудность здесь заключается в том, каким образом придать программам достаточно универсальную способность переводить задачу, сформулированную алгебраически или вербаль-

[1]) См., например, работу Amarel S., On representation of problems of reasoning about actions, Machine Intelligence, 3, Michie (ed.), pp. 131—172, 1968, Edinburgh Univ. Press.

но, скажем, в графическое представление: процесс, весьма родственный «галлюцинаторным» аспектам восприятия и понимания. Хотя работы Вертгаймера не снимают этого затруднения, они содержат подробно разобранные примеры, характеризующие сложность того поведения, которое называют «решением задач».

Исследования Бартлетта, относящиеся к несколько другой области — запоминанию, указывают на творческий характер этого процесса и явно связывают его со специфическими свойствами памяти, занятой обыденными текущими событиями. Его уверенность в том, что «любой новый входной импульс должен становиться не просто командой, задающей последовательность выполняемых в жестком хронологическом порядке действий, но призван быть раздражителем, позволяющим нам обращаться непосредственно к той части массива упорядоченного прошлого опыта, которая в наибольшей степени соответствует текущей потребности», ассоциируется скорее с разрабатываемыми в рамках искусственного интеллекта фактографическими информационно-поисковыми системами, чем с нынешними экспериментальными исследованиями кратковременной памяти, простого ассоциативного обучения, акустической спутанности и т. п.

Перечень параллелей между основными проблемами искусственного интеллекта и тем, что можно было бы определить как психологию поведения, формируемого на основе некоторой системы знаний, допускает практически неограниченное расширение, но уже и так должно быть очевидно значение этой машинно-ориентированной методологии для изучения личности. Современное состояние искусственного интеллекта — благоговейный трепет перед великолепной загадочностью волшебной тайны — поведения, обусловленного предшествующим знанием. Именно это чувство благоговения вместе с прагматическим оптимизмом побуждает все возрастающее число специалистов, действующих в области искусственного интеллекта, считать свои исследования скорее психологическими, чем техническими. Интеллектуальные потенции этих исследователей, особенно молодых специалистов — математиков, логиков, инженеров и других, начинающих сейчас работать в этой области, не могут не привести к быстрому, хотя быть может неравномерному прогрессу на пути к этим целям. Если мы не хотим стать свидетелями возникновения «бикультурной» (или «бипарадигматической») ситуации, необходимо обеспечить установление непосредственной связи между искусственным интеллектом и исследователями и студентами, специализирующимися в области методологии, определенной выше как личностно-ориентированной.

Последнее по меньшей мере обеспечит моральную поддержку возврату к «диагностическому» подходу при исследова-

нии индивидуального поведения в проблемной среде, являющейся хорошей аппроксимацией «повседневного опыта».

Такие среды включают разного рода «учебные материалы» — иллюстрированные истории, текстовые задачи и т. п., обеспечивая естественные условия для реализации большого объема знаний и навыков. Приложение этой системы знаний и навыков к конкретной задаче состоит в синтезе *соответствующего* сложного процесса, образованного последовательно упорядоченным набором актов, который мы называем осмысленным поведением. Задачей психолога является установление связи сформированного поведения с материалом «учебного задания» способами, которые идут дальше объяснений «на уровне здравого смысла», например, предсказав, каким образом будет протекать процесс научения, каков будет результат переструктуризации проблемного материала и т. д. Для установления подобных связей необходимо обратиться к понятиям и формальным достижениям, развитым и продолжающим усовершенствоваться в процессе рождения методологии искусственного интеллекта.

Сделать эту методологию основной для классической психологии означает придать психологическим исследованиям ту строгость при изучении «повседневного опыта», которая в течение столь долгих лет от них ускользала. Использование психологами этой методологии приведет, кроме всего прочего, к тому, что к искусственному интеллекту перестанут относиться лишь как к новейшим техническим достижениям, угрожающим праву человеческих существ заниматься деятельностью, являющейся их неотъемлемой прерогативой.

Автор выражает признательность профессору С. Сатерланду и доктору Р. Стантону за советы и практические замечания, высказанные ими в процессе подготовки данной работы.

Недостатки логики[1])

Д. Брус Андерсон и Патрик Дж. Хейз

1. ВВЕДЕНИЕ

Среди аргументов в пользу важности работ по автоматическому доказательству теорем для искусственного интеллекта приводятся и такие: вычислительные машины наделяются способностью выполнять формальные математические построения (посредством логики), и соответствующие методы (доказательства теорем) представляют интерес с точки зрения организации процессов «мышления» у роботов. Мы считаем, что методы, разработанные в этой области, как, впрочем, и любые методы, которые могут быть созданы на основе принятой в ней в настоящее время методологии (столь удачно определяемой как «вычислительная логика»), бесполезны для обоих этих направлений, хотя в настоящей статье мы рассматриваем в основном второе. Мышление роботов составляет основу нашей концепции искусственного интеллекта — и как будто не должно быть сомнений что знание, как построить «машину», способную рассуждать о реальном мире и действовать в нем, а также «разговаривать» о себе на естественном языке, необходимо (но не достаточно!) для создания системы искусственного интеллекта, действительно заслуживающей это название.

2. ЧТО ТАКОЕ «СИСТЕМА ДЛЯ ДОКАЗАТЕЛЬСТВА ТЕОРЕМ»?

Обычно на вход программы предназначенной для доказательства теорем, подается множество высказываний узкого исчисления предикатов, представленных множеством формул. Программа воспроизводит последовательности таких формул некоторым систематическим образом, довольно часто используя правило вывода, называемое резолюцией, до тех пор, пока не будет удовлетворено требование какого-либо критерия остановки. В качестве такового может выступать ограничение по «пространству» или по времени, воспроизведение пустой дизъюнкции или «нулевой (пустой) формулы». В последнем случае програм-

---

[1]) Anderson D. B., Hayes P. J., The Logicians Folly, European AISB Bulletin, № 1, 1972.

ма демонстрирует невыполнимость исходного множества формул и, изучая процесс вывода пустой формулы, можно сравнительно легко получить разного рода полезную информацию (в частности, последовательности операций, приводящие к достижению простых целей и т. п.).

Совершенно очевидно, что реализация такого рода программ сталкивается с проблемами, связанными с большими объемами поиска. Это затруднение преодолевается, как правило, с помощью двух методов. Во-первых, в программу вводится критерий, который, априори исключая некоторые выводы, ограничивает пространство поиска; этот метод называется *уточнением* (очищением). Во-вторых, используется определенная *стратегия поиска*, управляющая порядком воспроизведения ограниченного вышеупомянутым образом пространства. Обычно эта стратегия поиска определяется некоторой эвристической оценочной функцией, присваивающей фиксированный вес каждой формуле (или каждому множеству формул). Оценочные функции всегда основываются на локальных синтаксических признаках, таких, как длина формулы (число компонент), глубина вывода или (в более сложных случаях) глубина вложения функциональных символов и т. д.

Основное положение, которое мы хотели бы подчеркнуть, состоит в том, что независимо от совершенства или сложности программы ее разработка базируется на следующих двух важнейших моментах. Во-первых, практически неизбежно в программу закладывается процедура уточнения, использование которой, таким образом, не зависит от типа задачи. Во-вторых, стратегия поиска локально определяется простыми синтаксическими признаками формул. В сущности другого способа и нет, поскольку синтаксис формул — это единственный вид «готовой» информации специфической для конкретной задачи, которым может оперировать программа. Программа обрабатывает единообразно любые формулы независимо от того, представляют ли они высказывания из области универсальной алгебры или формирования планов у роботов. Она воспринимает лишь *логическую* структуру формул. В частности, если в некотором множестве формул единообразно переименовать все предикатные буквы, то система доказательства теорем окажется не в состоянии различить эти два входные множества. Ее поведение будет совершенно идентичным при обработке обоих множеств.

До сих пор у нас шла речь о стандартных современных системах доказательства теорем, однако наш тезис не сводится к непригодности этих программ для построения систем искусственного интеллекта, но утверждает, что методология «вычислительной логики» не позволяет создавать программы, которые были бы адекватны задачам этого направления. Эта методология,

которая, как мы надеемся, была достаточно освещена выше, рассматривает «механическое» доказательство теорем как сугубо *комбинаторную* задачу. Основное внимание уделяется не вопросам типа «что значит эта аксиома?», а таким моментам, как «какова вероятность того, что данное символическое выражение может быть решено за десять шагов?». В рамках такого подхода, естественно, практикуются очень сложные процедуры, например, такие, как компоненты эвристического поиска типа планирования, выдвижения подцелей, механического заучивания и т. д. И, естественно, нет ничего удивительного в том, что подобные исследования приносят важные результаты — в области задач комбинаторного характера.

3. ДЕМОНСТРАЦИЯ НЕСОСТОЯТЕЛЬНОСТИ

Система доказательства теорем чарует и влечет. В самом деле, она универсальна, упорядоченна и ее достоинства имеют подтверждение; имея правильные входные данные, она весело и непринужденно приведет Вас к искомому результату. Почему же мы не хотим пользоваться ею?

3.1. Часть знаний остается за пределами аксиом

Значительная часть сведений, которыми мы хотели бы наделить «разумную» систему, представляет собой описания простейших закономерностей внешнего мира, например, «если делается то-то, то в результате происходит следующее», «всякое то-то и то-то является тем-то» и т. п. Некоторые из такого рода описаний действительно можно рационально сформулировать в виде исходных аксиом для системы доказательства теорем. Существуют, однако, категории знания, которые совершенно не укладываются в рамки аксиом: информация, заключенная в советах и догадках, т. е. мы имеем в виду эвристики. Например, «этот план стоит попробовать применить при таких-то обстоятельствах» или «если вы использовали такой-то факт, то вам, вероятно, потребуется следующее». Этот вид информации присутствует в стратегии, определяющей характер тех процедур логического вывода, которыми следует воспользоваться, — можно было бы сказать «если данное символическое выражение решено, примените такую-то вычислительную процедуру и попытайтесь получить решение, используя формулу такого-то типа». Информация такого характера необходима при работе в любой проблемной области.

Иногда утверждают, что чистота, в частности, математики не запятнана подобными приемами. Однако достаточно хотя бы раз взглянуть в любой учебник для того, чтобы убедиться в том, как такого рода информация включается в процесс изучения

и понимания математики. Следующие примеры получены посредством случайного выбора из учебника по топологии:

«Остается наиболее тонкая часть доказательства — подтверждение того, что...»

«Тот факт, что ..., непосредственно следует из рассмотрения определяющих базисов. Для того чтобы убедиться в том, что ..., покажем, что ...»

«Доказательства следующих предложений не требуют применения новых методов».

«Исключительно важная лемма ...», «примечательное свойство...»

В настоящее время язык, на котором можно выразить эти стратегические сведения в системе доказательства теорем, является исключительно скудным. В лучшем случае он включает эвристические оценочные функции и упорядочения формул (функции выбора), а они, как мы убедились выше, должны определяться локальными синтаксическими признаками формул. Это означает, что просто невозможно «привязать» *советы* к формулам, т. е. использовать какую-либо стратегическую информацию, имеющую *значение* в контексте задачи, решаемой системой доказательства теорем.

Использование только синтаксических признаков может легко привести к неприятностям. Если, например, изменить оценочную функцию таким образом, чтобы уменьшить степень вложенности, то в результате можно отказаться от использования выражения

выделить (выделить (выделить (выделить (объект))))

при необходимости выделить какой-либо объект, и это хорошо, но это в то же время приведет к исключению выражения

повернуть (повернуть (повернуть (повернуть (пробку (бутылки)))))

как способа открыть бутылку. *Логические* структуры в обоих случаях одинаковы. Поскольку эти оценочные функции столь слабо связаны с содержательной стороной задачи, их восприятие (и реализация!) сопровождаются очень большими трудностями, связанными с тем, что невозможно предсказать результаты их использования. Как можно определить веса, которые следует присвоить вложениям функции с тем, чтобы избежать бесполезных затрат мускульных усилий при открывании бутылки? Быть может, следует ввести устройство слежения за вложениями, которое будет определять, какой предикат участвует в этом процессе.

Прибегнув к такому средству, мы немедленно столкнемся со множеством проблем. Откуда могут появиться сведения о функциях, подверженных появлению вложений? Являются ли они константами, либо представляют собой часть описания задачи? Как они вводятся в систему? На каком языке? Какого рода управление следует предусмотреть для придания этой схеме единообразия? Из обсуждения, проведенного в разд. 2, должно быть ясно, что в данном случае мы вышли за пределы стандартных представлений о системах доказательства теорем. А ведь мы ввели лишь крошечное изменение. Введение в систему тех сведений, которые действительно необходимы для решения задач, встречающихся в реальной жизни, потребует массы расширений такого рода.

Существует обстоятельство, которое вызывает недоразумения (и побуждает некоторых к сверхоптимизму по поводу автоматического доказательства теорем); дело в том, что в некоторых так называемых «игрушечных задачах» большую часть *структуры* задачи удается отобразить в *структуре* формул. Кроме того, можно обнаружить, что использование простой универсальной синтаксической эвристики типа единичного предпочтения позволяет получить доказательство посредством реализации экономной процедуры поиска.

Так бывает, однако, лишь в «игрушечных задачах». В узком исчислении просто нет структуры, достаточной для отображения сложности реальных задач. Более того, даже в тех случаях, когда такие логические процедуры «действуют», они выводят нас за пределы идеологии автоматического доказательства теорем. Для того чтобы иметь возможность *использовать* подобную структуру, мы должны располагать исчерпывающими сведениями о стратегии поиска и прочих особенностях системы доказательства теорем, необходимых для соответствующего формулирования аксиом. Итак, входной язык не является узким исчислением, а является скорее его некоторым расширением, в котором, например, порядок записи формул имеет стратегический смысл. Однако до сих пор ничего не известно о существовании системы доказательства теорем, предусматривающей использование подобного гипотетического языка. Обладает ли такой язык семантикой? Что означает термин «невыполнимость» в этом расширенном языке? и т. д. и т. п. Ясно, что мы опять оказались в новой области, поскольку изменили основные правила автоматического доказательства теорем.

Наше утверждение о том, что известные системы доказательства теорем неспособны воспроизводить все те виды знания, обладание которыми было бы желательно, на самом деле обосновано в большей степени, чем это может показаться с первого взгляда. Это положение не следует смешивать с доводом, сво-

дящимся к тому, что правильный подход к построению разумных машин требует «запрограммировать» всю нашу систему знаний. Наша точка зрения состоит в том, что если мы *действительно хотим* обладать разумной машиной, то эта машина *должна обладать* таким знанием, а ее внутренний язык должен быть пригоден для его воспроизведения. Должна ли такая машина добывать это знание в процессе обучения или оно будет просто вводиться в нее — это уже другой вопрос. Известные системы доказательства теорем не позволяют ни того, ни другого, поскольку в них не выполняется основное требование, сформулированное в 1958 г. Маккарти в виде следующего афоризма: «Для того чтобы программа имела возможность чему-нибудь научиться, надо в первую очередь иметь возможность сообщать ей что-нибудь».

Можно, конечно, утверждать, что при наличии хороших эвристик логический вывод будет выполняться не системой доказательства теорем, а какой-то другой программой, однако при этом мы сталкиваемся с проблемами организации взаимодействия, которые обсуждаются ниже. Некая программа будет решать, что должно использоваться — эвристическая программа или система доказательства теорем, которые должны находиться во взаимодействии друг с другом.

3.2. Большая часть знаний не может быть выражена с помощью исчислений предикатов

Кроме неизбежных недостатков, связанных с использованием аксиоматики для определения задачи, серьезные трудности возникают при попытке выразить сведения о проблемной области через высказывания узкого исчисления предикатов.

Главной темой нашего обсуждения является приложение методов автоматического доказательства теорем к различным классам задач, рассматриваемых в рамках искусственного интеллекта, особенно задачам, связанным с организацией мышления роботов. И именно в этой области становятся особенно очевидны недостатки узкого исчисления предикатов.

Задача создания формальных логических языков, даже на самом элементарном уровне эквивалентная задаче воспроизведения событий повседневной жизни, является животрепещущей темой иследований в области искусственного интеллекта и традиционной логики, но отнюдь *не* «вычислительной логики». И в настоящее время эту деятельность можно определить скорее как постановку фундаментальных задач, чем решительное продвижение вперед по пути успеха. Приведем краткий перечень примеров задач такого рода.

1. С задачами, включающими время и разного рода изменения, в настоящее время можно справляться только с помощью

введения фиксированных моментов времени (известных иначе как *ситуации*) и рассмотрения их по отдельности (другой способ предусматривает использование модальной логики, абсолютно не исследованной с вычислительной точки зрения). Язык, получаемый в результате таких построений, обременен труднопреодолимыми проблемами, разрешение части которых требует выхода за пределы языка узкого исчисления. Другие же в настоящее время просто не поддаются разрешению.

2. Задачи, включающие стратегии поведения, требуют использования трехзначной логики или другого языка подобного типа.

3. Часто возникает необходимость воспроизведения количественной информации (особенно в связи с использованием дедуктивного вывода при обработке визуальной информации — в этом случае очень важна информация о геометрических свойствах объекта). Соответствующие средства, которые могли бы быть вычислительно реализованы, в логике первого порядка отсутствуют.

4. «Бытовые» факты («снег — белый», «этот воздушный шар наполнен газом» и т. п.) обычно вызывают непреодолимые трудности.

5. Дедуктивные выводы, включающие оценки вероятностей или трудоемкости вычислительного процесса (например, в области зрительного восприятия: дедукции, предполагающие, что определенная зона поля зрения представляет интерес и должна быть обработана более тщательно), не могут быть адекватно формализованы средствами логики первого порядка.

Мы могли бы продолжить этот перечень, однако наша точка зрения и так освещена достаточно подробно.

Довольно забавно, что профессор Д. Миши, выступив некоторое время назад в защиту «вычислительной логики» [1]), снабдил нас отличным примером. Он приводит воспроизводимую ниже картинку (рис. 1) и пишет в связи с этим:

> «современный уровень развития вычислительной логики вполне позволяет аксиоматизировать набор сведений, представленных простой картинкой (например, сведения о плюшевых медвежатах, воздушных шарах, флажках, лестницах, тяготении и прочем), таким образом, что машинные процедуры доказательства теорем могут ими оперировать, т. е. делать это с помощью средств узкого исчисления предикатов».

Отлично, обратимся к рисунку. Задачи 3, 4 и 5 возникают немедленно. Понятие «падение» (ведь передний медвежонок падает?) вводит задачу 1. Упоминаемая профессором Миши эври-

[1]) Michie D., On Not Seeing Things, Department of Machine Intelligence, Edinburgh University, 1971.

стика «падение» основана на предположении о том, что флажки сделаны из мягкого материала и изгибаются под действием ветра. Совершенно не очевидно, что медвежонок А встает: флажок может висеть вниз из-за того, что он попал в нисходящий поток

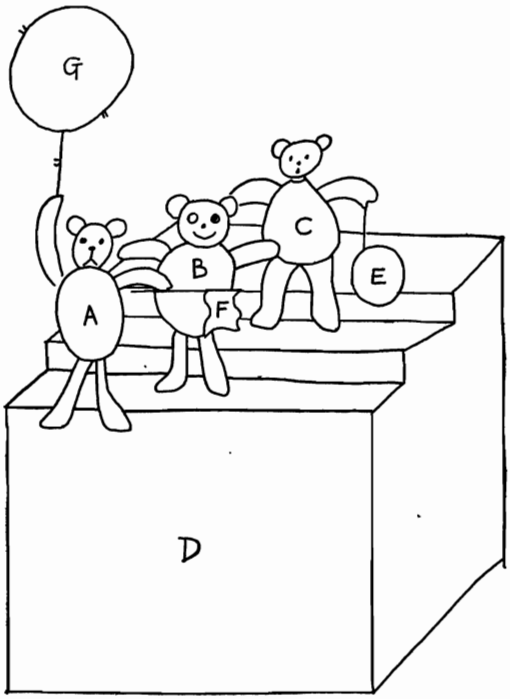


Рис. 1. Эпистемологические плюшевые медвежата.

воздуха или сделан из металла. Нам придется аксиоматизировать сведения относительно тяготения, зрительной окклюзии, причинности и т. д. и т. п. Дать аксиоматическое описание, адекватное куску реального внешнего мира, представленному на картинке, — задача интересная и достойная, однако ее разрешение при нынешнем уровне развития данной науки абсолютно ей не по силам. Еще сложнее создать действующую систему доказательства теорем, оказывающуюся полезной при работе с аксиоматикой подобного типа. И предоставим еще раз слово профессору Миши:

> «до недавнего времени представление процессов высшего уровня, обеспечивающих зрение роботов, как процессов доказательства теорем на основе метода резолюции встретило

бы самое сдержанное отношение, поскольку вычислительная неэффективность стратегий резолюции служила серьезным препятствием. После появления таких усовершенствований, как процедуры SL и G, перспективы кажутся значительно более обнадеживающими».

3.3. Система доказательства теорем не работает в режиме взаимодействия

Система доказательства теорем может являться *частью* «разумной» системы, но она совершенно очевидно не представляет собой *всю* систему. Она, в частности, не управляет двигателями робота и не может реагировать непосредственно на входную информацию, предъявляемую в виде изображений. Она не может определить, какие контрпримеры нужны для опровержения предложенной теоремы. Какой частью должна она быть? Сколь сложно организовать ее взаимодействие с остальными программами? Допустим, мы спросили у нее: «находится ли A на B?». Вполне возможно, что вывести ответ не удастся и система выдаст «не знаю». После этого какая-то другая программа должна просмотреть не приведший к искомому результату процесс вывода и каким-то известным ей способом принять решение о том, что делать дальше: то ли использовать какой-либо другой метод для поиска ответа на заданный вопрос, то ли предпринять какие-либо действия, которые устранят необходимость отвечать на него. Допустим, что наше обращение к системе доказательства теорем увенчалось успехом и она сформировала соответствующий план поведения. Другая подсистема получит его и попытается реализовать. Какой контроль нужно организовать для того, чтобы знать, что все идет по плану? Что следует делать, если в процессе исполнения плана выясняется, что допущена ошибка? Вероятно, план не может быть столь подробным, чтобы учитывать *любую* случайность. Необходимо принимать решение — вырабатывать ли новый план, исправлять ли ошибки на ходу и т. п. Допустим, что нам даже удалось ввести в систему доказательства теорем аксиоматическое описание *картинки* или *сцены. Целой* сцены? Естественно, при этом потребуется какая-либо подсистема для принятия решений. Как организовать взаимодействие системы доказательства теорем с этими подсистемами? Ведь программа, предназначенная для доказательства теорем, в сущности тоже является подсистемой, на вход которой подается множество формул, а на выходе воспроизводится (законченное или незаконченное) доказательство. Не может быть и речи, по крайней мере на нынешнем этапе развития искусственного интеллекта, об использовании этой подсистемы для управления вызовом и выполнением программ, входящих в другие подсистемы. Способность своевременно менять

направление «атаки» на задачу является прерогативой какой-то другой части системы. Профессор Лонге-Хиггинс отмечает, что чрезмерное упрощение базиса искусственного интеллекта может привести к тому, что все проблемы взаимодействия этого базиса с реальным миром ускользнут от нас. Упоминавшиеся выше вспомогательные программы осуществляют некоторые разновидности этого взаимодействия — и становится ясно, что они принимают очень важные решения относительно того, какую информацию следует сообщить системе доказательства теорем. Одна из этих программ даже анализирует выводы, не приведшие к получению искомого результата, с тем, чтобы определить, что следует делать дальше. Создается впечатление, что методы, используемые *этими* подсистемами, являются значительно более «мощными», чем методы, имеющиеся в распоряжении системы доказательства теорем. В состоянии ли последняя проанализировать причины, не позволившие какой-либо другой системе доказательства теорем найти искомое доказательство?

Кажется странным, что столь мощные процедуры вынуждены обращаться к системе доказательства теорем как подпрограмме. Так, например, программа, диагностирующая ошибки системы доказательства теорем, на самом деле будет полностью занята определением возможных путей развития доказательства.

Совершенно не ясно, могут ли эти программы работать в режиме взаимодействия с системой доказательства теорем. Какого рода информацию могла бы одна из них вводить в систему доказательства теорем с тем, чтобы управлять ее поведением? Больше или меньше аксиом? Интересно, что попытка ввести в систему доказательства теорем «чужую» процедуру натолкнулась на серьезные затруднения именно из-за этого обстоятельства. Единственный способ, посредством которого детектор аналогий Клинга в состоянии передать системе доказательства теорем типа QA3.5 предпочтение, отданное им определенному методу доказательства, заключается в коррекции опорного множества.

4. ФАКТЫ

В предыдущем разделе мы привели целый ряд причин, по которым система доказательства теорем не в состоянии удовлетворительным образом воспроизводить необходимую систему знаний и потому не удовлетворяет требованиям, предъявляемым к системе искусственного интеллекта. Однако подтверждаются ли наши критические замечания на практике? Быть может, все-таки можно аксиоматизировать приемлемым образом значительный объем знаний, найти соответствующую эвристическую функцию и «запустить» весь этот «механизм», не прибегая к ис-

пользованию указаний о дедуктивном выводе такого рода, как обсуждались выше.

4.1. Где же пользователи, заинтересованные в системе доказательства теорем?

Создается впечатление, что никто еще никогда не пытался получить интересную теорему, выведенную *собственно* системой доказательства теорем (хотя была масса разговоров на эту тему), не считая случаев работы с такой системой в режиме взаимодействия. Множество людей, использовавших или пытавшихся использовать их в качестве систем дедуктивного вывода, жаловались на неспособность систем доказательства теорем производить доказательства. Препятствием служила и их непригодность для работы с множествами и «арифметикой». Ни один специалист, связанный непосредственно с проектом, посвященным разработке робота, не использовал систему доказательства теорем каким-либо существенным образом. На самом деле нам известна всего одна занимающаяся искусственным интеллектом группа, в которой хоть кто-нибудь использовал бы или вообще собирался использовать систему доказательства теорем. Это исключение представляет группа, работающая в Станфордском исследовательском институте.

4.2. Станфордский исследовательский институт

Группа Станфордского исследовательского института была первой, применившей систему доказательства теорем к задачам, возникающим в рамках искусственного интеллекта. Их робот управлялся системой доказательства теорем (типа QA3), и существует фильм, демонстрирующий способность этого робота принять решение об использовании наклонной плоскости для того, чтобы взобраться на ящик. Естественно, система доказательства теорем лишь часть всей системы, однако она служит основным инструментом логического вывода (в программах зрительного восприятия, например, процедуры вывода не используются). Трудности, обнаружившиеся при попытках решать такие и подобные им тривиальные задачи с помощью систем доказательства теорем, привели к выполнению целого ряда исследований, кульминационным пунктом которых явилось рождение новой системы, названной STRIPS. Мы не будем вдаваться здесь в подробности, однако отметим, что эта новая система использовала новый вид формализма для описания ее деятельности и система доказательства теорем играла в ней значительно менее важную роль[1]). Робот, управляемый системой STRIPS,

[1]) STRIPS — Stanford Research Institute Problem Solver (Решатель задач Станфордского исследовательского института). Система реализована на вычислительной машине типа PDP-10 на языке LISP. Для поиска новых моде-

дополненной системой ILA (Intermediate Level Actions), представляющей собой набор программ, вырабатывающих операторы, регулирующие их работу, и системой PLANEX, программой-диспетчером, обеспечивающей обращение к STRIPS, реализует планы, выработанные с помощью ILA и других подсистем. Итак, в единственном иследовательском центре, обладающем реальным опытом применения системы доказательства теорем для управления роботом, роль последней становится все менее и менее значительной.

5. УНИВЕРСАЛЬНОСТЬ АРГУМЕНТОВ

Хотя до сих пор мы рассматривали системы комбинаторно-логического характера, использованные доводы с тем же успехом можно использовать для оценки возможностей систем решения задач иных типов.

5.1. Как следует ими пользоваться

Основная проблема здесь состоит в том, чтобы оценить мощность тех языков, с помощью которых пользователь может сообщить системе свою задачу и применить специализированные подпрограммы для ее решения. При этом возникают три важных вопроса:

1. Сколь сложно воспроизвести сведения, касающиеся процеса решения задачи? Сложно ли вводить в систему необходимые указания?
2. Сколь хорошо подходит данный язык для описания задач?
3. Как осуществляется работа системы в режиме взаимодействия? Располагает ли система действенной управляющей структурой?

5.2. Примеры

5.2.1. Эвристический поиск

Обычно в системах, обладающих процедурами эвристического поиска, в качестве языка описания задач используется стандартный язык программирования, предусматривающий использование структур данных для представления состояний и

лей в пространстве моделей используется так называемый метод анализа целей и средств, а в пределах конкретной модели — методы формального доказательства. Более подробные сведения о системе STRIPS на русском языке можно найти в статьях Дж. Мансона «Робот планирует, выполняет и контролирует в неопределенной среде» и Р. Файкса и Н. Нильсона «Система STRIPS — новый подход к применению методов доказательства теорем при решении задач», опубликованных в сб. статей «Интегральные роботы», «Мир», М., 1973, стр. 355—403. — *Прим. перев.*

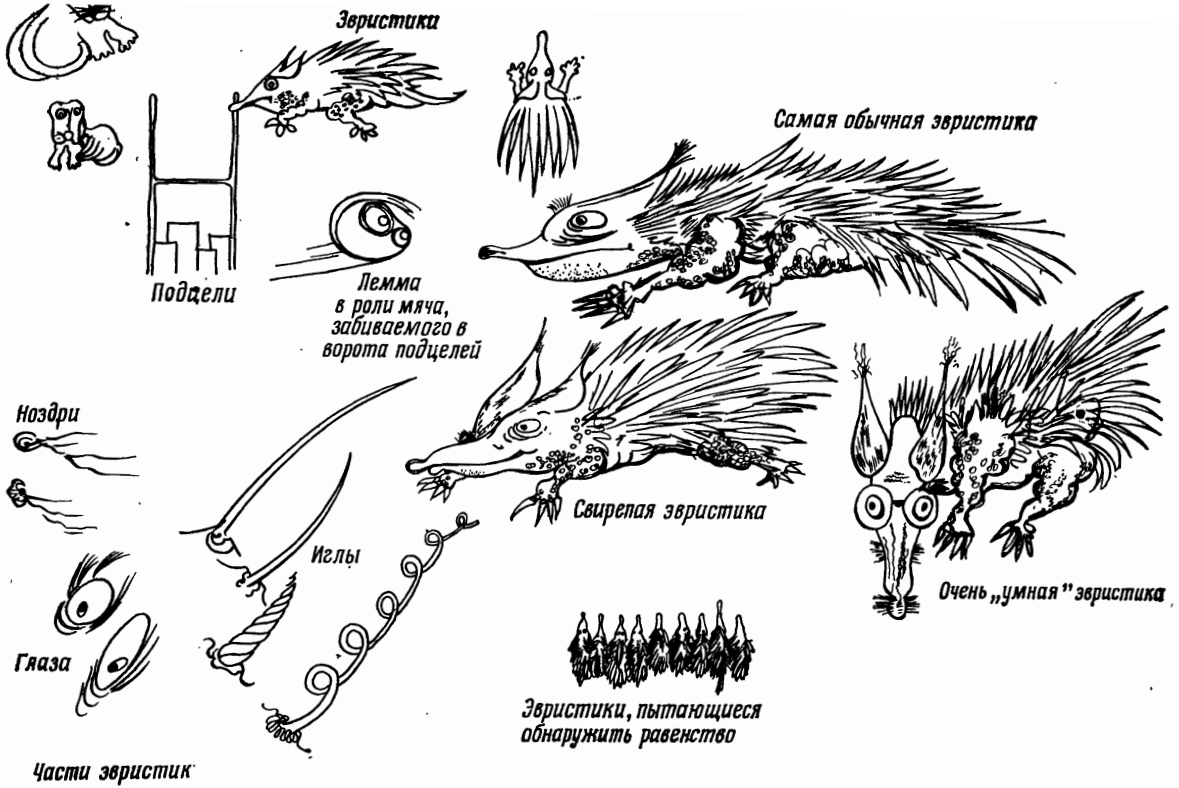


Рис. 2. **«... как работать с равенствами: с помощью лемм и подцелей или «умных» эвристик ...»**

функций для выполнения преобразований; точность описания полностью определяется возможностями языка. Однако, для того чтобы управлять процессом логического вывода, необходимо обратиться к языку эвристических функций. Если мы хотим, чтобы применение принципов эвристического поиска оказалось продуктивным, соответствующая функция должна определяться локально и ее приложение к анализу конкретной ситуации не должно требовать продолжительных вычислений. Она не может предложить следующий шаг или запретить применение какого-либо оператора. Известно, что при решении целого ряда головоломок и игр использование процедур эвристического поиска не приносит успеха, за исключением систем, обладающих большим объемом знаний, например, таких, как системы, обеспечивающие мышление роботов.

5.2.2. Система GPS и система STRIPS

Со времен системы GPS [1]) создание универсальных «систем решения задач» (именно так) является привлекательной областью для приложения усилий специалистов, занимающихся искусственным интеллектом. Основная идея таких разработок сводилась к попытке сформулировать некие универсальные механизмы или принципы интеллекта, причем применение их к решению конкретных задач трактовалась как чисто техническая проблема создания комплекта программного обеспечения, ориентированного на частные потребности пользователя, однако только разработчики системы GPS уделили внимание публичному обсуждению психологических аспектов этой проблемы. В системе GPS используется весьма ограниченный язык таблиц

[1]) GPS — General Problem Solver (Универсальный решатель задач). Программа, разработанная в 1960 г. Ньюэллом, Шоу и Саймоном, представляющая собой попытку синтезировать в одной схеме набор понятий, методов и стратегий, которые по предположению лежат в основе общих действий человека при решении задач, отвлекаясь от специфических черт конкретной задачи. Наиболее подробно на русском языке описание принципов, заложенных в систему GPS, и самой системы можно найти в следующих работах: 1) две статьи Ньюэлла, переводы которых помещены в сб. «Самоорганизующиеся системы», «Мир», М., 1964; 2) А. Ньюэлл, Г. Саймон, Моделирование человеческого мышления на вычислительной машине, в сб. «Кибернетика и живой организм», «Наукова думка», Киев, 1964; 3) А. Ньюэлл, Г. А. Саймон, Имитация мышления человека с помощью электронно-вычислительной машины, в сб. «Психология мышления», «Прогресс», М., 1965; 4) А. Ньюэлл, Дж. С. Шоу, Г. А. Саймон, Процессы творческого мышления, в сб. «Психология мышления», «Прогресс», М., 1965; 5) А. Ньюэлл и Г. Саймон, GPS — программа, моделирующая процесс творческого мышления, в сб. «Вычислительные машины и мышление» (под редакцией Э. Фейгенбаума и Дж. Фельдмана), «Мир», М., 1967, стр. 281—301. Результаты работ по усовершенствованию системы GPS, в частности описание систем GPS-2-1 ÷ GPS-2-6, можно найти в монографии G. W. Ernst, A. Newell, GPS: A Case Study in Generality and Problem Solving, Academic Press, New York, London, 1969.— *Прим. перев.*

различий между объектами и т. п. признаков для описания процесса решения задачи; представление самой задачи осуществляется также с помощью довольно маломощного языка. Можно указать ситуации, когда описание задачи для введения в машину будет занимать больше времени, чем решение ее собственными силами!

Если систему GPS использовали только для решения головоломок, то система STRIPS применялась уже для выработки планов поведения робота. Ее операторы, подобные операторам, использовавшимся в системе GPS, образуют язык описания поведения робота, который является сильно ограниченным, и соответственно управление процессом решения осуществляется довольно приблизительно. Поскольку эта подсистема занимается исключительно формированием планов, ее взаимодействие с остальными компонентами системы является очень слабым: подсистема не в состоянии запросить помощь или установить, почему ей не удалось получить ответ.

5.2.3. LISP и PLANNER

Из сказанного, очевидно, следует, что только система, обладающая возможностями языка программирования, может удовлетворить нас. Как в таком случае обстоят дела с языком LISP? С ним сейчас все в порядке, но *выразителем* искусственного интеллекта он никак не является. Может быть и верно говорить «представлять все в виде списочных структур», но это едва ли поможет делу. На языке LISP написано много интересных программ, но это не делает его более интересным! PLANNER лучше LISP не только потому, что является более мощным языком программирования, но и в связи с наличием в нем указания о возможностях его использования. Информационный массив обычно представляет «состояние» внешнего мира; при воспроизведении внешнего мира предпочтение отдается теоремам, а не стандартным функциям. В PLANNER систему знаний о внешнем мире можно представлять по-разному и проводить эксперименты, вводя и исключая связи между различными частями этого информационного массива, что совершенно немыслимо в рамках систем «решения задач».

6. ВЫВОДЫ

Сейчас стало очевидным, что подходы к организации мышления роботов (и к высшим интеллектуальным процессам), отличающиеся от классической «автоматизации доказательства теорем», утратили черты методологии для данного случая и обладают высшим совершенством. Мы указывали выше, что этот процесс неизбежен в связи с нынешним характером методоло-

гии «вычислительной логики». Естественно, из неприемлемости этой методологии для автоматизации процессов мышления отнюдь не следует ее принципиальная непригодность. Эффективные методы синтаксического поиска и пр., разработанные в области «доказательства теорем», весьма полезны в соответствующих областях, например в таких задачах исследования операций, как составление расписаний и распределение ресурсов. Приемы, специфические для «вычислительной логики», могут оказаться как раз теми, что необходимы в таких областях, как теория сложности. Если, однако, «вычислительные логики» все еще собираются поразить провозглашенную ими цель автоматизации мышления, не говоря уже о создании программ хотя бы приближающихся к возможностям «живых» математиков, то они должны полностью изменить свой подход к решению этой проблемы.

СОДЕРЖАНИЕ

КИБЕРНЕТИЧЕСКИЙ СБОРНИК № 13

Редактор Л. Н. Бабынина
Художник Н. К. Сапожников Художественный редактор В. И. Шаповалов
Технический редактор Н. Б. Панфилова Корректор Л. В. Байкова

Сдано в набор 17.03.76. Подписано к печати 20.08.76. Бумага № 2 60 × 90¹/₁₆ = 5,75 бум. л. 11,50 печ. л. Уч.-изд. л. 11,01. Изд. № 1/8643. Цена 1 р. 31 к. Зак. 125.

ИЗДАТЕЛЬСТВО «МИР»
Москва, 1-й Рижский пер., 2

Ордена Трудового Красного Знамени Ленинградская типография № 2
имени Евгении Соколовой Союзполиграфпрома при Государственном комитете
Совета Министров СССР по делам издательств,
полиграфии и книжной торговли.
198052, Ленинград, Л-52, Измайловский проспект, 29.