EESTI NSV TEADUSTE AKADEEMIA TOIMETISED.
FÜÜSIKA * MATEMAATIKA
ИЗВЕСТИЯ АКАДЕМИИ НАУК ЭСТОНСКОЙ ССР. ФИЗИКА * МАТЕМАТИКА
PROCEEDINGS OF THE ACADEMY OF SCIENCES OF THE ESTONIAN SSR.
PHYSICS * MATHEMATICS

1985, 34, 2

УДК 510.649

А. ЛОМП

СТРУКТУРНЫЙ СИНТЕЗ ПРОГРАММ С ПОМОЩЬЮ СИСТЕМЫ ПРОЛОГ

(Представил Э. Тыугу)

Введение

Описывается структурный синтез программ средствами исчисления предикатов первого порядка [1] и системы программирования ПРОЛОГ [2].

Под структурным синтезом программ мы понимаем автоматический синтез, применяемый в системе программирования ПРИЗ [3] и основанный на том, что можно построить программу, зная мало о свойствах подпрограмм, вызываемых в программе. Предполагается, что спецификация содержит только информацию о возможности вычислить значения переменных, встречающихся в спецификации, с помощью данных подпрограмм. Эта информация представляется в вычислительной модели и задаче [4, 5].

В [6] рассмотрен математический аппарат, приспособленный для описания работы системы ПРИЗ в терминах интуиционистского исчисления высказываний и не направленный на его реализацию средствами системы ПРОЛОГ. Однако реализация этого метода синтеза программ средствами системы ПРОЛОГ приводит к простому способу программирования систем подобных ПРИЗу и к простому способу проведения экспериментов с этими системами. Известно, что задачи, описанные в исчислении предикатов первого порядка, можно решить в системе ПРОЛОГ. До сих пор задачу синтеза программ удалось описать только в некотором слабом исчислении предикатов второго порядка [6]. В данной работе рассматривается синтез программ в исчислении предикатов первого порядка и в системе программирования ПРОЛОГ.

1. Описание вычислительной модели и задачи на ней на языке предикатов первого порядка

В разделе 1.1 дается описание подъязыка языка исчисления предикатов, в разделах 1.2 и 1.3 показывается, как с помощью этого подъязыка можно описать вычислительную модель и вычислительную задачу.

Напомним некоторые свойства модели и задачи [3–5]. **Вычислительная модель** (сокращенно модель) содержит информацию о переменных и подпрограммах, где подпрограммы могут применяться для вычисления значений некоторых переменных по значениям других переменных. **Подпрограмма применима**, если ее входные переменные вычислимы и ее подзадачи разрешимы. **Переменная вычислима**, если она является входной переменной задачи или выходной переменной подпрограммы. **В вычислительной задаче** указываются входные и вы-

ходные переменные. Задача разрешима, если из входных переменных вычислима выходная переменная. **В подзадаче** также указываются входные и выходные переменные. Имеются независимые и зависимые подзадачи. Они отличаются друг от друга критериями разрешимости. **Независимая подзадача разрешима**, если из входных переменных подзадачи вычислима выходная переменная подзадачи. Разрешимость зависимой подзадачи зависит от контекста, т. е. мы говорим, что в контексте первой подзадачи решается вторая, если в ходе решения первой подзадачи решается и вторая. Данная **зависимая подзадача разрешима** в контексте других подзадач, если выходная переменная данной подзадачи вычислима из входных переменных данной подзадачи и из входных переменных других подзадач, в контексте которых решается данная подзадача.

1.1. Символы, термы и атомы

Обозначим: константы строчными буквами, переменные прописными буквами и множества большими греческими буквами.

Алфавит языка следующий:

1) константы:
   индивидные константы: $a, b, \ldots, x, y,$
   функциональные константы: $\lambda, f, g, \ldots$,
   предикатная константа: c,
2) свободные переменные: $A, B, \ldots, X, Y$,
3) вспомогательные символы: $\rightarrow$, ', ·, (,).

Из символов языка конструируются термы и атомы. В наш язык входят термы трех типов: «выходная переменная», «входные переменные» и «программа».

Термы типа «выходная переменная» (сокращенно т.т. вых.) — это индивидные константы и свободные переменные.

Термы типа «входные переменные» (сокращенно т.т. вх.) определяются следующим образом:

1) индивидная константа и свободная переменная есть т. т. вх.;
2) если $t_1, t_2, \ldots, t_k$ — т. т. вх., то выражение $t_1 . t_2 . \ldots . t_k$ есть т. т. вх. $(k \geqslant 1)$.

Термы типа «программа» (сокращенно т.т. пр.) определяются следующим образом:

1) любая индивидная константа и свободная переменная есть т. т. пр.;
2) любая 0-местная функциональная константа есть т. т. пр.;
3) если $t_1, t_2, \ldots, t_k$ — т. т. пр. и f есть k-местная функциональная константа, то выражение $f(t_1, t_2, \ldots, t_k)$ есть т. т. пр. $(k \geqslant 1)$;
4) если s есть т. т. вх., t — т. т. пр. и λ — имя 2-местной функциональной константы, то выражение $\lambda(s, t)$ есть т. т. пр.

В языке имеется только один 3-местный предикат c, который характеризует вычислимость выходной переменной (первого аргумента) из входных переменных (второго аргумента) с помощью программы (третьего аргумента).

Из термов и предикатов конструируются атомы вычислимости. Если $t_{\text{вых}}$ есть т. т. вых., $t_{\text{вх}}$ — т. т. вх., $t_{\text{пр}}$ — т. т. пр., то выражение $c(t_{\text{вых}}, t_{\text{вх}}, t_{\text{пр}})$ есть **атом вычислимости**. В терминах вычислительной модели атом вычислимости $c(t_{\text{вых}}, t_{\text{вх}}, t_{\text{пр}})$ выражает вычислимость переменной $t_{\text{вых}}$ по входным переменным задачи и подзадач $t_{\text{вх}}$ с помощью подпрограммы $t_{\text{пр}}$. Например, атом вычислимости $c(z, x. y, f(x, y))$ читается следующим образом: из входных переменных x и y вычислима z с помощью подпрограммы f.

Используем секвенциальный вид записи аксиом. **Секвенция** есть выражение вида $\Gamma \rightarrow A$, где Γ — последовательность формул (т. н. антецедент), а A — формула (т. н. сукцедент). Секвенция $A_1, \ldots, A_m \rightarrow A_0$

интерпретируется как выражение $\neg A_1 \vee \ldots \vee \neg A_m \vee A_0 \vee 0$, где $\neg$ — отрицание, $\vee$ — дизъюнкция, A_i — формула, **0** — ложь [1].

Атомы вычислимости являются единственными формулами нашего языка, а во всех рассматриваемых нами теориях **аксиомы** будут иметь вид секвенций $c_1, c_2, \ldots, c_k \rightarrow c_0$, где $c_0, c_1, \ldots, c_k$ являются атомами вычислимости. Секвенция вида $\rightarrow c$ называется **фактом**, а секвенция вида $c \rightarrow$ — **задачей**. Конечное множество аксиом называется **теорией**, конечное множество одноименных аксиом — **процедурой** (см. в разделе 4 процедуру u).

На описываемом языке свободная переменная I означает любой терм типа «входные переменные» и свободные переменные отличные от I означают любые термы типа «программа». Они заменяются термами в ходе решения задачи (см. раздел 2). Конструкциям без переменных соответствуют понятия на языке вычислительных моделей. Они приведены ниже.

Язык исчисления предикатов	Язык вычислительной модели
Индивидная константа	Имя переменной
Функциональная константа	Имя подпрограммы
Терм типа «выходная переменная»	Выходная переменная задачи
Терм типа «входные переменные»	Входные переменные задачи
Атом вычислимости	Задача, подзадача

1.2. Описание вычислительной модели

Вычислительная модель — это теория следующего вида. Рассмотрим правила записи аксиом на примерах, причем отдельно подпрограммы без подзадач, подпрограммы с независимыми подзадачами и подпрограммы с зависимыми подзадачами.

а) Подпрограмма без подзадач.

Пример:

$$c(x, I, X), c(y, I, Y) \rightarrow c(z, I, f(X, Y)).$$

Аксиома описывает подпрограмму f, которая вычисляет из переменных x, y переменную z.

б) Подпрограмма с независимыми подзадачами.

Пример:

$$c(y, x, Y), c(z, I, Z) \rightarrow c(w, I, f(\lambda(x, Y), Z)).$$

Аксиома описывает подпрограмму f, которая вычисляет из переменной z переменную w и вызывает при этом подпрограмму, вычисляющую из переменной x переменную y. Вызываемая подпрограмма синтезируется в ходе решения независимой подзадачи «вычислить y из x».

в) Подпрограмма с зависимыми подзадачами.

Пример:

$$c(y, x.I, Y), c(z, I, Z) \rightarrow c(w; I, f(\lambda(x.I, Y), Z)).$$

Аксиома описывает подпрограмму f, которая вычисляет из переменной z переменную w и вызывает при этом подпрограмму, вычисляющую из переменной x и переменных I переменную y. Вызываемая подпрограмма синтезируется в ходе решения зависимой подзадачи «вычислить y из x».

Во всех рассматриваемых нами теориях аксиомой считается секвенция $\rightarrow c(A, L.A.R, A)$, где A, L, R являются свободными переменными и аргумент $L.A.R$ служит для указания на тривиальный алгоритм вычис-

лений переменной A, т. е. выделения ее значения из списка данных значений. Говоря неформально в терминах вычислительной модели, аксиома $\rightarrow c(A, L.A.R, A)$ выражает очевидный факт: «если A входная переменная, то A вычислима». Мы называем эту секвенцию **аксиомой вычисления.**

1.3. Описание вычислительной задачи

В вычислительной задаче требуется сконструировать программу $t_{\text{пр}}$, которая вычисляет из входных переменных задачи $t_{\text{вх}}$ выходные переменные задачи $t_{\text{вых}}$ [3].

На описанном выше языке задачу можно сформулировать в виде секвенции $\rightarrow c(t_{\text{вых}}, t_{\text{вх}}, t_{\text{пр}})$. Для решения этой задачи нужно доказать теорему $A_1 \wedge \ldots \wedge A_n \vdash T$, где A_i — аксиома и T — задача. Для доказательства этой теоремы методом резолюции надо показать, что аксиомы $A_1 \wedge \ldots \wedge A_n$ и отрицание задачи $\neg T$ противоречивы (т. е. $A_1 \wedge \ldots \ldots \wedge A_n \wedge \neg T$ тождественно ложно) [7]. Мы запишем задачу не в положительной форме $\rightarrow c(t_{\text{вых}}, t_{\text{вх}}, t_{\text{пр}})$, а в отрицательной $c(t_{\text{вых}}, t_{\text{вх}}, t_{\text{пр}}) \rightarrow$.

Отметим дополнительно, что задачами будем называть также выражения $c(t_{\text{вых}}, t_{\text{вх}}, t_{\text{пр}}), \ldots \rightarrow$, которые генерируются в ходе решения.

1.4. Пример

Опишем задачу вычисления двойного интеграла $s = \int_0^{ry} \int_0^{rx} f(x, y)\,dx\,dy$, где ry и rx — пределы интегрирования. Предположим, что заданы подпрограмма интегрирования *simps* и подпрограмма вычисления функции f:

$$\rightarrow c(A, L.A.R, A), \tag{1}$$

$$c(x, I, X),\ c(y, I, Y) \rightarrow c(z, I, f(X, Y)), \tag{2}$$

$$c(z, x.I, Z),\ c(rx, I, RX) \rightarrow c(w, I, simps(\lambda(x.I, Z), RX)), \tag{3}$$

$$c(w, y.I, W),\ c(ry, I, RY) \rightarrow c(s, I, simps(\lambda(y.I, W), RY)), \tag{4}$$

$$c(s, rx.ry, S) \rightarrow. \tag{5}$$

Здесь

(1) — аксиома вычисления (см. раздел 1.2.),
(2)—(4) — вычислительная модель,
(5) — вычислительная задача.

2. Решение задач в исчислении предикатов первого порядка

Для решения задачи надо доказать или опровергнуть описывающую ее теорему. Мы используем метод резолюции и стратегию линейной резолюции, которые используются и в системе ПРОЛОГ [7]. Для применения вышеуказанного метода аксиомы и задачу преобразуют в секвенциальную форму, а затем пытаются вывести из них противоречие (пустую секвенцию), используя при этом правило резолюции (сечения).

Теперь приведем решение задачи вычисления двойного интеграла, описанной в предыдущем разделе. После доказательства теоремы приведем вспомогательную таблицу, где показаны все шаги унификации (говорят, что **термы унифицируемы**, если их можно делать одинаковыми, подставляя вместо свободных переменных новые термы [7]). В ходе решения **поставленной задачи** $c(s, rx.ry, S) \rightarrow$ генерируются **вспомога-**

тельные задачи: $c(w, y.rx.ry; W)\rightarrow$, $c(ry, rx.ry, RY)\rightarrow$, $c(z, x.y.rx.ry, Z)\rightarrow$, $c(rx, y.rx.ry, RX)\rightarrow$, $c(x; x.y.rx.ry, X)\rightarrow$ и $c(y, x.y.rx.ry, Y)\rightarrow$.
Пример:

1) Доказательство теоремы.

Шаг доказательства	Номер аксиомы	Поставленная задача
1	4	$c(s, rx.ry, S)\rightarrow$
2	3	$c(w, y.rx.ry, W)$, $c(ry, rx.ry, RY)\rightarrow$
3	2	$c(z, x.y.rx.ry, Z)$, $c(rx, y.rx.ry, RX)$, $c(ry, rx.ry, RY)\rightarrow$
4	1	$c(x, x.y.rx.ry, X)$, $c(y, x.y.rx.ry, Y)$ ... $\rightarrow$
5	1	$c(y, x.y.rx.ry, Y)$, $c(rx, y.rx.ry, RX)$, $c(ry, rx.ry, RY)\rightarrow$
6	1	$c(rx, y.rx.ry, RX)$, $c(ry, rx.ry, RY)\rightarrow$
7	1	$c(ry, rx.ry, RY)\rightarrow$
		$\rightarrow$

2) Шаги унификаций [7].

Шаг доказательства	Номер аксиомы	Подстановки
1	4	$\{rx.ry/I,\ simps(\lambda(y.rx.ry, W), RY)/S\}$
2	3	$\{y.rx.ry/I,\ simps(\lambda(x.y.rx.ry, Z), RX)/W\}$
3	2	$\{x.y.rx.ry/I,\ f(X, Y)/Z\}$
4	1	$\{x/A,\ nil/L,\ y.rx.ry/R,\ x/X\}$
5	1	$\{y/A,\ x/L,\ rx.ry/R,\ y/Y\}$
6	1	$\{rx/A,\ x.y/L,\ ry/R,\ rx/RX\}$
7	1	$\{ry/A,\ x.y.rx/L,\ nil/R,\ ry/RY\}$

3) Программа.

$$simps(\lambda(y.rx.ry, simps(\lambda(x.y.rx.ry, f(x, y)), rx), ry)).$$

3. Описание и решение вычислительной задачи средствами системы ПРОЛОГ

Мы спроектировали язык таким образом, что описание модели является одновременно текстом на языке исчисления предикатов первого порядка и почти текстом на языке программирования ПРОЛОГ. При переводе текста с языка исчисления предикатов на язык программирования ПРОЛОГ требуется уточнить только аксиому вычислений $\rightarrow c(A, L.A.R, A)$, заменяя ее на следующие три:

$$\rightarrow c(A, A, A),$$
$$\rightarrow c(A, A.R, A),$$
$$c(A, R; P)\rightarrow c(A, L.R, P).$$

4. Избежание зацикливания интерпретатора системы ПРОЛОГ

Приведенный выше материал может создать впечатление, что в системе ПРОЛОГ проблема синтеза программ решена (мы ведь показали как описывать и как решать вычислительную задачу). На самом же деле при поиске доказательства интерпретатор ПРОЛОГа может зациклиться. Например, если в теории имеются аксиомы $x\rightarrow y$, $y\rightarrow x$ и задача $y\rightarrow$, то интерпретатор будет по очереди решать задачи $y\rightarrow$ и $x\rightarrow$ (и по очереди использовать аксиомы $x\rightarrow y$ и $y\rightarrow x$).

Для устранения этого недостатка нужно реализовать процедуру, ко-

торая проверяет предикат «решается указанная вычислительная задача». Интерпретатор должен вычислять этот предикат на каждом шаге поиска доказательства, и если он истинен — не применять аксиому, которая генерировала повторную задачу. Следовательно, надо:

— реализовать процедуру проверки,
— упорядочить входные переменные так, чтобы на каждом шаге поиска доказательства можно было легко осуществлять проверку,
— преобразовать аксиомы так, чтобы на каждом шаге поиске доказательства происходило обращение к процедуре проверки.

Во избежание зацикливания необходимо, во-первых, реализовать базовые процедуры *order*, *branch* и, во-вторых, с помощью этих процедур реализовать процедуры *u* и *P*. ***Order*** преобразует неупорядоченный список констант (первый аргумент) в упорядоченный список констант (второй аргумент). ***Branch*** осуществляет поиск атома (единственный аргумент) из ветви доказательства (т. е. из числа атомов, которые доказываются в момент вызова). Например, вызов процедуры $branch(c(z, x.y, —))$ заканчивается успешно, если в ветви доказательства найдется атом, который унифицируем с атомом $c(z, x.y, —)$. Отметим отдельно, что третий аргумент этого атома не влияет на ход поиска доказательства (это утверждение следует из конструкции аксиом). При поиске атома из ветви этот аргумент не учитывается. Эти базовые процедуры легко реализуются через стандартные процедуры системы МПРОЛОГ [8] и мы не будем их подробно описывать.

Процедура *u* записывается двумя секвенциями:

$$u(O, I, P) \leftarrow branch(c(O, I, —)), !, fail,$$
$$u(O, I, P) \leftarrow c(O, I, P), !.$$

Она работает следующим образом: «если в данной ветви доказательства обнаружится вызов $c(O, I, —)$, то вызов процедуры будет неудачным». Отметим, что процедура ! является управляющей процедурой ПРОЛОГа, которая иногда называется процедурой *cut* [2, 8].

Проверка *branch* осуществляется легко, если в данном доказательстве во всех атомах входные переменные упорядочены одинаково. **Процедура *P*** вначале упорядочивает входные переменные, а затем вызывает процедуру *u*, т. е. $p(O, I1, P) \leftarrow order(I1, I2), u(O, I2, P)$.

Для вызова на каждом шаге доказательства реализованных процедур *u* и *p* следует переписать аксиомы. Мы перепишем антецеденты аксиом так, чтобы символ *c* был заменяем символом *p* (при постановке подзадачи) или *u* (без постановки подзадачи).

5. Сохранение и использование информации о разрешимости вспомогательных задач

В ходе решения поставленной задачи решается множество вспомогательных задач (см. пункт 2). Интерпретатор ПРОЛОГа не запоминает доказанные или опровергнутые вспомогательные задачи (он решает их каждый раз заново). Это приводит к экспоненциальной сложности поиска доказательства.

Для устранения этого недостатка надо реализовать процедуру, которая запоминает факт «задача разрешима» или факт «задача не разрешима». Когда задача появляется первый раз, то ее доказывают или опровергают и запоминают соответствующий факт. Когда задача появляется следующий раз, то ее не доказывают и не опровергают, а используют факт, который был запомнен ранее.

Факт $\rightarrow f(T, O, I, P)$ имеет четыре аргумента: первый указывает тип, т. е. «+» — задача была разрешима и «—» — задача была неразрешима; следующие три аргумента совпадают с аргументами процедур *c*, *u* и *p*.

Процедура *add* добавляет указанный факт в базу знаний. Она легко реализуется через стандартные процедуры МПРОЛОГа [8] и вызывается из процедуры *u*.

Ниже приводится описание задачи вычисления двойного интеграла в МПРОЛОГе (см. пример в разделе 1.4). В процедуре *u* на 2-й и 3-й строке используют соответственно положительный и отрицательный факт, а на 4-й и 5-й строке запоминают соответственно положительный и отрицательный факт. При использовании фактов учитываются свойства вычислительных задач «если разрешима задача $c(z, x.y, p)\rightarrow$, то разрешима и задача $c(z, x.y.w, p)\rightarrow$» и «если неразрешима задача $c(z, x.y, p)\rightarrow$, то неразрешима и задача $c(z, x, p)\rightarrow$». Вызов процедуры *inclusion*$(I1, I2)$ реализует проверку $I1 \subseteq I2$. В МПРОЛОГе «$c_1, c_2, \ldots, \ldots, c_n \rightarrow c_0$» записывается «$c_0$: $-c_1, c_2, \ldots, c_n$.».

Пример:

$u(O, I, P) :- branch(c(O, I, -)), !, fail.$ (1)

$u(O, I2, P) :- f(-, O, I1, P), inclusion(I2, I1), !, fail.$ (2)

$u(O, I2, P) :- f(+, O, I1, P), inclusion(I1, I2), !.$ (3)

$u(O, I, P) :- c(O, I, P), !, add(f(+, O, I, P)), !.$ (4)

$u(O, I, P) :- add(f(-, O, I, P)), fail.$ (5)

$p(O, I1, P) :- order(I1, I2), u(O, I2, P).$ (6)

$c(A, A, A) :-.$ (7)

$c(A, A.R, A) :-.$ (8)

$c(A, L.R, A) :- c(A, R, P).$ (9)

$c(z, I, f(X, Y)) :- u(x, I, X), u(y, I, Y).$ (10)

$c(w, I, simps(\lambda(x.I, Z), RX)) :- p(z, x.I, Z), u(rx, I, RX).$ (11)

$c(s, I, simps(\lambda(y.I, w), RY)) :- p(w, y.I, W), u(ry, I, RY).$ (12)

$:- p(s, rx.ry, S).$ (13)

Здесь

(1)—(9) — процедуры интерпретатора,
(10)—(12) — вычислительная модель,
(13) — задача.

Заключение

Показано, во-первых, что каждую вычислительную задачу системы ПРИЗ можно описать и решить в системе ПРОЛОГ, и, во-вторых, как для их эффективного решения надо усовершенствовать интерпретатор ПРОЛОГа.

Показан, в принципе, как переформулировать задачу из ПРИЗа в ПРОЛОГ. Это, однако, не значит, что вычислительную задачу целесообразно сформулировать в ПРОЛОГе. Одной конструкции входного языка ПРИЗ соответствуют в общем случае сотни или тысячи конструкций языка ПРОЛОГ. Трансляция текста задачи из ПРИЗа в ПРОЛОГ мыслима только с помощью ЭВМ.

Автор признателен Э. Тыугу за поддержку и интерес к данной работе и Г. Минцу, П. Лоренцу, Л. Мытусу, М. Мацкину за ценные советы и замечания.

ЛИТЕРАТУРА

1. *Такеути Г.* Теория доказательств. М., «Мир», 1978.
2. *Gallaire, H.* A Study of PROLOG — Computer Program Synthesis Methodologies. Proc. of the NATO Advanced Study Institute held of Bonas, France, Sept. 28 — Okt. 10, 1981, 173—212.
3. *Кахро М. И., Калья А. П., Тыугу Э. Х.* Инструментальная система программирования ЕС ЭВМ (ПРИЗ). М., Финансы и статистика, 1981.
4. *Тыугу Э.* Ж. вычисл. матем. и матем. физ., **10**, № 3, 716—733 (1970).
5. *Тыугу Э.* Ж. вычисл. матем. и матем. физ., **11**, № 4, 993—1004 (1971).
6. *Минц Г. Е., Тыугу Э. Х.* В кн.: Автоматический синтез программ. Таллин, АН ЭССР, 1983, 3—40.
7. *Чень Ч., Ли Р.* Математическая логика и автоматическое доказательство теорем. М., «Наука», 1983.
8. MPROLOG Language Reference Manual, Version 1.3., Institute for Co-ordination of Computer Techniques. Budapest, 1983.

Институт кибернетики
Академии наук Эстонской ССР

Поступила в редакцию
21/XI 1984

A. LOMP

PROGRAMMIDE STRUKTUURNE SÜNTEES SÜSTEEMIS PROLOG

Artiklis on näidatud, kuidas süsteemi PROLOG vahenditega realiseerida programmide struktuurset sünteesi.

A. LOMP

IMPLEMENTATION OF PROGRAM SYNTHESIS IN PROLOG

Programming system PRIZ is based on the automatic synthesis of programs from computational models. In the paper an implementation of this synthesis in PROLOG is discussed.