ИЗВЕСТИЯ
ПЕНЗЕНСКОГО ГОСУДАРСТВЕННОГО
ПЕДАГОГИЧЕСКОГО УНИВЕРСИТЕТА
имени В. Г. БЕЛИНСКОГО
ФИЗИКО-МАТЕМАТИЧЕСКИЕ И ТЕХНИЧЕСКИЕ НАУКИ
№ 13 (17) 2009

IZVESTIA
PENZENSKOGO GOSUDARSTVENNOGO
PEDAGOGICHESKOGO UNIVERSITETA
imeni V. G. BELINSKOGO
PHYSICAL, MATHEMATICAL AND TECHNICAL SCIENCES
№ 13 (17) 2009

УДК 004.424

ДЕДУКТИВНЫЙ СИНТЕЗ ФУНКЦИОНАЛЬНЫХ И ИМПЕРАТИВНЫХ ПРОГРАММ

© В. В. ДРОЖДИН, М. В. ЖУКОВ
Пензенский государственный педагогический университет имени В.Г. Белинского,
кафедра прикладной математики и информатики
e-mail: drozhdin@spu-penza.ru

Дрождин В. В., Жуков М. В. – Дедуктивный синтез функциональных и императивных программ // Известия ПГПУ им. В. Г. Белинского. 2009. № 13 (17). С. 89–94. – *Для предметной области, описанной на языке конструктивной логики, дедуктивный метод позволяет автоматически находить решение новых задач. Рассматривается способ реализации дедуктивного метода, основанный на дедуктивных таблицах, приводятся примеры дедуктивного синтеза функциональных и императивных программ.*

Ключевые слова: дедуктивный синтез, дедуктивная таблица, синтез транзакций.

Drozhdin V. V., Zhukov M. V. – The deductive synthesis of functional and imperative programs // Izv. Penz. gos. pedagog. univ. im.i V. G. Belinskogo. 2009. № 13 (17). P. 89–94. – *The deductive method enables to find the solutions of new tasks for the domain, described at the language of constructive logic. The article considers one of deductive methods of realization based on the deductive tables. It also gives examples of deductive synthesis both functional and imperative programs.*

Keywords: deductive synthesis, deductive table, synthesis of transactions.

Одной из наиболее сложных проблем в области программирования является разработка метода автоматической генерации программы компьютером, для задачи, которая ранее им не решалась и для которой у него отсутствует алгоритм решения. Одним из возможных способов решения этой проблемы является дедуктивный синтез программ, основанный на дедуктивных таблицах [1].

Синтез программы начинается с ее спецификации: задания отношения между входными и выходными данными. Спецификация должна описывать цель или ожидаемое поведение программы и в общем виде задается следующим образом:

$$f(a) <= \text{найти } z \text{ такой, что } Q[a,z],$$

где z – выходные данные, a – входные, Q – условия, заданные на языке конструктивной логики, которым должны удовлетворять входные данные a и соответствующий им результат z. Следовательно, необходимо найти функцию f такую, чтобы для любого значения a, ее результат z удовлетворял ограничению Q.

Согласно дедуктивному методу данная задача интерпретируется как доказательство теоремы существования:

$$\forall(a)\exists(z)Q[a, z].$$

Для ее доказательства достаточно найти функцию f получения z по a, которую будем строить с помощью метода дедуктивных таблиц.

Дедуктивные таблицы. Дедуктивная таблица состоит из колонки утверждений (assertions), в которой записываются известные аксиомы, колонки целей (goals), которая содержит цели для доказательства, и выходной колонки, в которой формируется текст программы. Если в одной спецификации задано несколько функций, то в таблице появляется несколько выходных колонок. Для синтеза программы её спецификация добавляется в дедуктивную таблицу в качестве цели для доказательства. При этом в таблицу может быть уже включен набор аксиом, описывающих предметную область задачи, свойства предикатов и функций, использованных в спецификации. Ниже приведен вариант начального состояния дедуктивной таблицы:

assertions	goals	f(a)
	Q[a, z]	z
A_1		
A_2		
⋮		
A_n		

где A_1, A_2,.. A_n – известные аксиомы предметной области.

Процесс доказательства представляет собой последовательное преобразование дедуктивной таб-

лицы, до тех пор, пока в ней не появится одна из строк:

assertions	**goals**	**f(a)**
	true	t

или

assertions	**goals**	**f(a)**
false		t

При этом программа t будет являться искомым решением, т.е. f(a) <= t. Построение осуществляется посредством ***эквивалентных преобразований***, в основу которых положены эквивалентные преобразования конструктивной логики и свойства дедуктивных таблиц, либо посредством ***подобных*** преобразований. В первом случае дедуктивная таблица (строка дедуктивной таблицы) заменяется эквивалентной, во втором результирующая таблица (строка) представляет собой объединение исходной и подобной таблиц (строк).

Свойства дедуктивных таблиц

1 Двойственность (слева расположена исходная таблица, справа – эквивалентная ей)

A		s	↔		¬A	s

и

	G	s	↔	¬G		s

2 Подстановка

A		s	↔	A		s
				Aθ		sθ

и

	G	s	↔		G	s
					Gθ	sθ

где θ – подстановка, т.е. множество $\{x_1 \leftarrow t_1, ..., x_n \leftarrow t_n\}$ пар вида $x_i \leftarrow t_i$, где x_i – переменная в исходном выражении, t_i – заменяющая x_i подстановка.

Дедуктивные правила

1 Правила разделения

1.1 And-split (в верхней части таблицы, до двойной черты, расположена исходная таблица, в нижней – строки, которые будут добавлены к ней после применения дедуктивного правила)

$A_1 \wedge A_2$		s
A_1		s
A_2		s

1.2 Or-split

	$G_1 \vee G_2$	s
	G_1	s
	G_2	s

1.3 If-split

	if A **then** G	s
A		s
	G	s

2 Правило резолюции (GG – goal, goal)

	G_1 [P]	s
	G_2 [P']	t
	$G_1\theta$[false] ∧ $G_2\theta$[true]	**if** Pθ **then** tθ **else** sθ

где θ – наибольший общий унификатор подвыражений P и P'. С учетом свойства двойственности и свойств логических операций, можно получить и другие правила резолюции: AA, AG, GA.

Заметим, что при применении правила резолюции зачастую в таблице могут появиться строки вида:

	false	t

или

true		t

которые не представляют никакой ценности для дальнейшего процесса доказательства и могут быть удалены из таблицы. Чтобы избежать появления таких строк можно использовать полярность выражений. Выражение, расположенное в столбце goals имеет положительную полярность, если количество знаков отрицания ¬ влияющих на него чётно, в противном случае полярность отрицательная. Для выражений расположенных в столбце assertions все с точностью наоборот.

При применении правила резолюции подвыражение P(P') следует заменить true, если его полярность положительная, иначе – false.

3 Правило эквивалентности

A_1[l = r]		s
A_2 [l']		t
$A_1\theta$[false] ∨ $A_2\theta$[rθ]		**if** (l = r)θ **then** tθ **else** sθ

где θ – наибольший общий унификатор подвыражений l и l'.

4 Правило индукции

Если Q[a, z] – исходная цель (спецификация), то в таблицу может быть добавлена гипотеза индукции, утверждающая истинность исходного утверждения для меньшего, чем a аргумента и содержащая обращение к искомой функции f:

	Q[a, **z**]	z
if $x <_w a$ **then** Q[x, f(x)]		

где $<_w$ – отношение полной фундированности.

Дедуктивный синтез функциональных программ. Процедуру синтеза функциональных программ рассмотрим на примере построения программы со следующей спецификацией:

$$<\text{front}(s), \text{last}(s)>$$
$$<=$$
найти $<z_1, z_2>$ такие, что
$$[\textbf{if } \neg(s = \Lambda) \textbf{ then } (\text{char}(z_2) \wedge s = z_1 \bullet z_2)],$$

где s – произвольный набор символов (строка), Λ – строка, не содержащая ни одного символа, char(s) – предикат, принимающий значение true, если строка s состоит только из одного символа, • – операция конкатенации строк. Следовательно, необходимо разработать программу <front(s), last(s)>, декомпозирующую строку s на две подстроки z_1 и z_2, конкатенация которых дает исходную строку s, причем z_2 – последний символ строки s.

Программы для функций front(s) и last(s), полученные в результате применения метода дедуктивной таблицы и аксиом домена строк будут иметь вид:

front(s) <= [**if** char(s) **then** Λ
else head(s) • front(tail(s))]

last(s) <= [**if** char(s) **then** s
else last(tail(s))].

Дедуктивная таблица, с комментариями к каждому шагу построения приведена ниже (заметим, что все аксиомы в столбце assertion, которым не присвоен порядковый номер – это известные аксиомы домена строк):

assertion	goal	front(s)	last(s)	Комментарии
	1) if $\neg(s = \Lambda)$ then $\text{char}(z_2) \wedge s = z_2 \bullet z_2$	z_1	z_2	По правилу if-split получим 2), 3)
2) $\neg(s = \Lambda)$				z_1, z_2 удалены из результирующих столбцов, т.к. не являются свободными переменными в 2)
	3) $\text{char}(z_2) \wedge [s = z_1 \bullet z_2]^+$	z_1	z_2	
$[\Lambda \bullet y = y]^-$				Аксиома конкатенации
	4) $\text{char}(z_2) \wedge [s = z_2]^+$	Λ	z_2	Получено по правилу эквивалентности и подстановке $\{z_1 \leftarrow \Lambda, y \leftarrow z_2\}$
$[x = x]^-$				
	5) char(s)	Λ	s	Правило резолюции и подстановка $\{x \leftarrow s, z_2 \leftarrow s\}$
if char(u) then $[[(u \bullet y_1) \bullet y_2] = u \bullet (y_1 \bullet y_2)]^-$				Аксиома конкатенации
	6) $\text{char}(u) \wedge \text{char}(y_2) \wedge s = u \bullet (y_1 \bullet y_2)$	$u \bullet y_1$	y_2	Получено применением к 3) правила эквивалентности и подстановки $\{z_2 \leftarrow u \bullet y_1, z_2 \leftarrow y_2\}$
7) if $x <_{string} s$ then if $\neg(x = \Lambda)$ then $\text{char}(\text{last}(x)) \wedge x = \text{front}(x) \bullet \text{last}(x)$				Применение правила индукции к 1)
if $\neg(y = \Lambda)$ then $y = \text{head}(y) \bullet \text{tail}(y)$				Аксиома декомпозиции
Из 6), 7) и аксиомы декомпозиции можно получить:				
	8) $\neg(s = \Lambda) \wedge \neg\text{char}(s)$	head(s) • front(tail(s))	last(tail(s))	
Из 5), 8) и аксиомы: $(y = \Lambda)$ $\text{char}(y) \vee \neg(\text{tail}(y) = \Lambda)$ выводится:				
	9) true	if char(s) then Λ else head(s) • front(tail(s))	if char(s) then s else last(tail(s))	

Дедуктивный синтез императивных программ на примере синтеза транзакций [2]. Рассмотрим базу данных (БД) кадрового агентства, схема которой приведена на рис. 1, со следующими ограничениями целостности (ОЦ) (приведены только необходимые в дальнейшем ограничения):

1. В таблице Applicant (Соискатель) не может быть 2-х записей с одинаковым атрибутом name, т.к. name является первичным ключом;

2. Для любой строки r таблицы Interview (Собеседование) в таблице Applicant существует строка, первичный ключ которой равен атрибут applicant из r;

3. Applicant не может быть Employee (Работником), т.е. множество значений атрибута name из таблицы Applicant не пересекается с множеством значений атрибута name из таблицы Employee.

С учетом изложенных ОЦ БД процедура трудоустройства нового работника employee будет состоять из этапов:

1. Если в таблице Applicant есть соискатель applicant с именем работника employee, то:

1.1 Удалить все собеседования applicant;

1.2 Удалить applicant из таблицы Applicant;

2. Добавить employee в Employee.

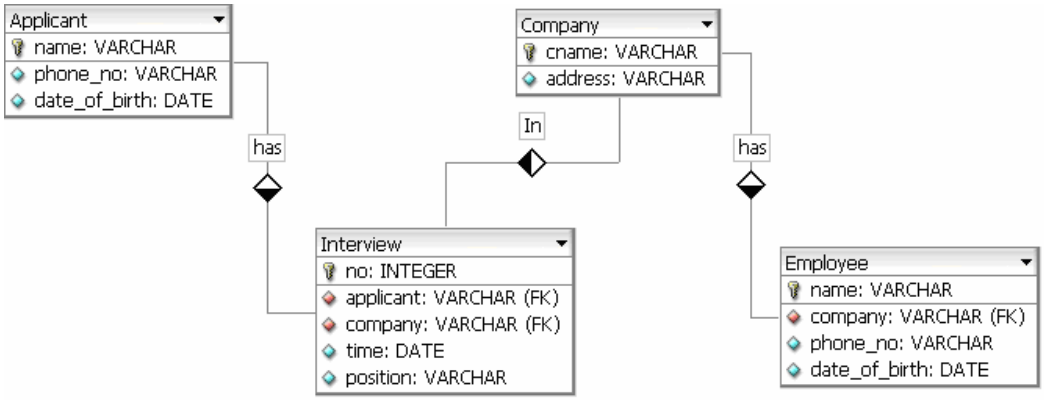

Рис. 1. Схема БД кадрового агентства

Прежде чем приступать к синтезу кода транзакции трудоустройства введем необходимые обозначения:

1. R(e) – предикат, принимающий значение true если отношение R содержит кортеж e. В рассматриваемом примере R может принимать одно из значений Employee, Interview, Company, Applicant сокращенно обозначаемые как E, I, C, A соответственно;

2. $s_i^n(e)$ – селектор, извлекающий значение i-го атрибута из кортежа e длинной n.

Любая транзакция представляет собой композицию одной из атомарных транзакций:

1. $insert_R(e)$ – добавление e в R;

2. $delete_R(e)$ – удаления e из R;

3. $modify_R(x, e)$ – замена x на e в R.

Новые транзакции могут образовываться следующими способами:

1. композиция $t_1 ;; t_2$;

2. условный переход **if** p **then** t_1 **else** t_2 **fi**, где p – предикат;

3. цикл **foreach** x **in** R|p **do** t **od**, где p – предикат, заданный для отношения R и позволяющий выбрать кортежи, к которым будет применена транзакция, размещенная в теле цикла.

Чтобы перейти от синтеза функциональных к синтезу императивных программ, в конструктивную логику необходимо ввести понятие состояния. Будем различать два типа объектов: обычные объекты (s-object), такие как состояние, атрибут или кортеж, и функциональные объекты (f-object), которые возвращают состояние, значение атрибута или кортеж как результат вычисления. Представители первого класса объектов – это результаты вычислений, полученные для конкретного состояния БД. Рассмотренные выше предикаты и транзакции – это f-object-ы, каждому из которых можно поставить в соответствие s-object:

f-object	s-object	Альтернативное обозначение s-object	Комментарии к s-object
e	e’	w:e	некоторое конкретное значение кортежа e
R(e)	R’(w, e’)	w:: R(e)	true, если e’ находится в R при состоянии БД w
$s_i^n(e)$	$S'^n_i(w, e')$	w: $s_i^n(e)$	значение атрибута s_i^n кортежа e’ при состоянии БД w
t(e)	t’(w, e’)	w; t(e)	состояние, в которое перейдет БД из w после выполнения транзакции t с входными данными e’

Автоматический синтез транзакции заключается в поиске такого кода транзакции, который, не нарушая ограничений целостности, переводит БД в состояние, удовлетворяющее ограничениям, заданным в спецификации транзакции. Следовательно, необходимо найти доказательство теоремы:

$$(\forall s_0)(\forall x)(\exists \tau)(C(s_0) \rightarrow Q(s_0, s_0;\tau, x) \wedge C(s_0;\tau)),$$

где s_0 – исходное состояние БД, $s_0;\tau$ – состояние БД после выполнения транзакции τ, **x – вектор входных** аргументов транзакции τ, **C(s) – предикат проверки** ограничений целостности БД в состоянии s, **Q – предикат**, специфицирующий задачу.

Для процедуры трудоустройства нового работника **а теорема будет иметь вид:**

$$(\forall s_0)(\forall a)(\exists \tau)$$
$$((\forall x)(E'(s_0, x) \rightarrow \neg A'(s_0, x)) \wedge$$
$$(\forall x)(\ I'(s_0, x) \rightarrow (\exists y)(A'(s_0, y) \wedge$$
$$applicant(x) = name(y))) \wedge$$
$$(\forall x)(\forall y)((A'(s_0, x) \wedge A'(s_0, y) \wedge$$
$$name(x) = name(y)) \rightarrow (x=y))) \rightarrow (E'(s_0;\tau, a) \wedge$$
$$(\forall x)(E'(s_0;\tau, x) \rightarrow \neg A'(s_0;\tau, x)) \wedge$$
$$(\forall x)(\ I'(s_0;\tau, x) \rightarrow (\exists y)(A'(s_0;\tau, y) \wedge$$
$$applicant(x) = name(y))) \wedge$$
$$(\forall x)(\forall y)((A'(s_0;\tau, x) \wedge A'(s_0;\tau, y) \wedge$$
$$name(x) = name(y)) \rightarrow (x = y)))$$

Теперь, когда сформулирована спецификация задачи, можно приступать к дедуктивному синтезу (см. таблицу ниже):

assertion	goal	transaction	Комментарии
	1) $C(s_0) \rightarrow E'(s_0;\tau, a) \wedge C(s_0;\tau)$	$s_0;\tau$	
2) $C(s_0)$		$s_0;\tau$	
	3) $[E'(s_0;\tau, a)]^+ \wedge C(s_0;\tau)$	$s_0;\tau$	
$[E'(insert_E'(w, x), x)]^-$			Свойство транзакции insert
	4) $C(s_0;\tau_1;insert_E(a))$	$s_0;\tau_1;insert_E(a)$	$\{w \leftarrow s_0;\tau, \tau \leftarrow \tau_1;;insert_E(a), x \leftarrow a\}$
	...	…	
	5) $[\neg A'(s_0;\tau_1, a)]^+ \wedge C(s_0;\tau_1)$	$s_0;\tau_1;insert_E(a)$	
$A'(w, x) \rightarrow [\neg A'(delete_A'(w,x),x)]^-$			Свойство транзакции delete
	6) $A'(s_0;\tau_2, a) \wedge C(s_0;\tau_2;delete_A(a))$	$s_0;\tau_2;delete_A(a);$ $insert_E(a)$	$\{w \leftarrow s_0;\tau_2, \tau_1 \leftarrow \tau_2;;delete_A(a), x \leftarrow a\}$
	7) $C(s_0;\tau_2) \wedge C(s_0;\tau_2;$ $delete_A(a))$	**if** $A'(s_0;\tau_2, a)$ **then** $s_0;\tau_2;delete_A(a);$ $insert_E(a)$ **else** $s_0;\tau_2;insert_E(a)$ **fi**	Из 6) и 5)
	8) $C(s_0;\tau_2) \wedge C(s_0;\tau_2;$ $delete_A(a))$	$s_0;\tau_2;$(**if** A(a) **then** $delete_A(a)$ **else** Λ **fi**); $insert_E(a)$	Преобразование транзакции (алгоритм преобразований описан в [2])
	9) $A'(s_0;\tau_2, a) \wedge C(s_0;\tau_2) \wedge (\forall x)(I'(s_0;\tau_2, x) \rightarrow \neg p(x, a))$	$s_0;\tau_2;$ $delete_A(a);$ $insert_E(a)$	Из 6) (из фрейма удаления) $p(a, x) = (applicant(x) = name(a))$
	10) $A'(s_0;\tau_3;t, a) \wedge C(s_0;\tau_3;t)$	$s_0;\tau_3;$ (**foreach** x **in** I|p(x, a) **do** $delete_I(x)$ **od**); $delete_A(a);$ $insert_E(a)$	Здесь t – это цикл с вложенной транзакцией, квантор всеобщности заменен на true после сколемизации
	11) $A'(s_0;\tau_3,a) \wedge C(s_0;\tau_3;t)$	$s_0;\tau_3;$ (**foreach** x **in** I|p(x,a) **do** $delete_I(x)$ **od**); $delete_A(a);$ $insert_E(a)$	т.к. t не влияет на предикат A
	12) $C(s_0;\tau_3)$	$s_0;\tau_3;$ (**if** A(a) **then** (**foreach** x **in** |p(x, a) **do** $delete_I(x)$ **od**); $delete_A(a);$ **else** Λ **fi**); $insert_E(a)$	Из 5) и учитывая то, что t не нарушает никаких ограничений целостности
$C(s_0)$			
	13) true	$s_0;$ (**if** A(a) **then** (**foreach** x **in** I|p(x, a) **do** $delete_I(x)$ **od**); $delete_A(a);$ **else** Λ **fi**); $insert_E(a)$	$\{ s_0;\tau_3 \leftarrow s_0\}$

Таким образом, код транзакции трудоустройства нового работника будет иметь вид:

```
if A(a)
  then
    foreach x in I|p(x, a)
      do delete_I (x)
      od;
    delete_A (a);
  else Λ
fi;
insert_E (a).
```

Рассмотренный метод синтеза программ на основе дедуктивных таблиц позволяет автоматически генерировать программы решения задач в предметных областях, описанных на языке конструктивной логики.

СПИСОК ЛИТЕРАТУРЫ

1. Manna Z., Waldinger R., Fundamentals of deductive program synthesis. Computer and Systems Sciences, Springer-Verlag, Berlin, 1991, 62p.
2. Qian X. The deductive synthesis of database transactions. PhD Dissertation, Technical Report STAN-CS-89-1291, Department of Computer Science, Stanford University, November 1989, 48p.