УДК 519.681

ЛОГИКА СЕКВЕНЦИАЛЬНЫХ СХЕМ АЛГОРИТМОВ

С.П. Кашубин

Институт проблем машиностроения НАН Украины,
Харьков, Украина, s.kashubin@gmail.com

Описана новая теория, названная *логикой секвенциальных схем алгоритмов*, которая предназначена главным образом для программирования. В рамках теории исследуются вопросы доказательства правильности алгоритмов, их эквивалентные преобразования и распараллеливание.

АЛГОРИТМ, ПРЕДСТАВЛЕНЕ ЗНАНИЙ, ДОКАЗАТЕЛЬСТВО ПРАВИЛЬНОСТИ АЛГОРИТМА, ЭКВИВАЛЕНТНЫЕ ПРЕОБРАЗОВАНИЯ, РАСПАРАЛЛЕЛИВАНИЕ, ТЕОРИЯ ПРОГРАММИРОВАНИЯ, СЕКВЕНЦИАЛЬНАЯ СХЕМА

Введение

Чтобы решить задачу, необходимо знать ее *алгоритм*, под которым понимается точная конечная система правил, определяющая совокупность действий над некоторыми объектами (исходными и промежуточными данными) для решения задачи. Иногда вместо слова «алгоритм» мы будем употреблять словосочетание «*схема алгоритма*».

Общее понятие алгоритма было известно еще в средние века, но его точное математическое определение было сформировано лишь в тридцатых годах XX столетия. Дело в том, что в начале XX века накопилось большое количество задач, которые не поддавались решению, несмотря на значительные усилия. Возникло подозрение, что для некоторых из этих задач не существует алгоритма вообще. В связи с такой ситуацией появилась необходимость в точном определении алгоритма. Осознавая эту ситуацию, а также опираясь на исследования в области оснований математики и математической логики, было выработано сразу несколько стандартных математических определений алгоритма. Первые стандартные варианты этого понятия были сформулированы практически одновременно и независимо друг от друга А.Тьюрингом (машина Тьюринга), А.Черчем (лямбда-исчисление) и Э.Постом (машина Поста). Чуть позже С.Клини ввел понятие рекурсивной функции, а А.А.Марков – нормальные алгорифмы. Как оказалось, все эти определения эквивалентны между собой. В настоящее время «алгоритм» является одним из основных и употребительных понятий математики и не только ее.

Процесс программирования обычно начинается с описания области знаний промежуточным языком, дающим возможность построения алгоритмов с последующей реализацией этих алгоритмов в программный код. В этом промежуточном языке понятие «алгоритм» более приспособлено для целей программирования, чем определения, рассмотренные выше. Такие алгоритмы будем называть прикладными алгоритмами, а область знаний, которая изучает их свойства – *прикладной теорией алгоритмов*.

В направлении развития прикладной теории алгоритмов сделаны важные шаги. В 1958 году А.А. Ляпунов [1] и Ю.А. Янов [2] разработали понятие логической схемы алгоритма, а годом позже Л.А. Калужнин ввел понятие граф-схемы алгоритма [3], затем И.Д. Заславский [4] продолжил работы в данном направлении. В.С. Королюк [5] и Е.Л. Ющенко [6] разработали понятие адресного алгоритма. Еще одна модель была предложена И.В. Вельбицким [7] и получила название Р-схемы. Для моделирования программного обеспечения и других систем в 1994 году Г. Бучем, Дж. Румбахом и А. Джекобсоном был создан UML – универсальный язык моделирования [8]. Основная парадигма для современной информатики, характерная для интеллектуальных систем, опирается на идею представления знаний. Сложность и многообразие структур знаний породили ряд способов их представления, таких как логическая модель, фреймовская и продукционная системы, семантические сети. Новая концепция Web также названа семантическими сетями, где знания должны быть описаны при помощи онтологий. Под онтологией (в информатике) понимается описание предметной области в виде, удобном для компьютерной обработки. Данная концепция была предложена и активно развивается комитетом W3C, создателем и руководителем которой является Тим Бернарс-Ли – автор сети World Wide Web. Базовыми языками для онтологий сейчас являются языки RDF, RDFS и OWL.

В 1983 году автором была опубликована работа [9], где описан язык секвенциальных схем (ЯСС). Этот язык использовался при создании и документировании систем программного обеспечения, например, система *konofo,* которая предназначена для приведения формул исчисления высказываний к конъюнктивно нормальной форме [10]. В результате исследований ЯСС был сделан вывод о том, что этот язык необходимо уточнить и доработать. В настоящей работе представлена новая уточненная и дополненная версия языка. Среди дополнений имеются средства логического вывода, что дает основание считать ЯСС логикой. Полное название этой логики – «*логика секвенциальных схем алгоритмов*», а краткое – «*логика* S_2» или просто «S_2».

1. Структура области знаний

Мы будем придерживаться точки зрения, согласно которой знания ориентированы на возможность решения задач, а поскольку задачи решаются по алгоритмам, то и алгоритмы являются знаниями. Языки описания знаний бывают естественными и искусственными (формальными). На естественных языках мы разговариваем с детства, а языки искусственные включают языки программирования, языки моделирования, алгоритмические языки и т.д. Естественные языки могут содержать специфическую символику: математическую, химическую и т.д. Многие языки используются в различных формах, например, в виде устной речи, системы жестов, письменности и т.д., однако, любая форма языка сводится к текстам. Знания, как правило, хранятся в различных библиотеках и Интернете.

Для описания отдельного языка используется метаязык. *Метаязык* – это язык для исследования какого-либо другого языка, который в данном случае называется *объектным языком*. К метаязыкам в информатике относится стандарты типа Dublin Core, XML, RDF и т.д. Для логики S_2 планируется создание своего собственного метаязыка.

Пусть A и B алгоритмы, где B манипулирует объектами, которые являются схемой алгоритма A или ее частями. Тогда A по отношению к B называется *алгоритмом-объектом*, а B по отношению к A – *метаалгоритмом*. Примерами метаалгоритмов являются программы ИКС-технологии [11], а по отношению к ним программное обеспечение, созданное этой технологией – алгоритмами-объектами.

Все алгоритмы выполняются последовательно или параллельно в среде *абстрактного компьютера (АК)*, который имеет для этих целей необходимое количество исполнителей и памяти. *Исполнителем* может быть как человек, так и некоторое автоматическое устройство. Память состоит из отдельных ячеек. *Ячейка памяти* – это место для записи и хранения информации. *АК* обладает *вычислительным ресурсом*, состоящим из набора алгоритмов, правильно решающих свои задачи.

1.1. Предметная область

Объектами *предметной области* является все то, что может быть как-то воспринято и на что может быть направлена наша мысль. Объекты используются как данные при решении задач. Предметная область логики S_2 структурирована на основе понятия *типа*, который представляет собой множество объектов. Любые данные, которыми манипулируют алгоритмы, относятся к тому или иному типу.

Для того чтобы объекты и типы можно было использовать в рассуждениях, им следует дать имена. Имена, обозначающие конкретные объекты, называются *константами*, а их совокупность определяет тип. *Переменные* – это имена ячеек памяти, куда разрешено записывать константы одного типа. Другими словами, тип переменной – это область ее определения.

Тип вводится выражения вида:

$$type\ T\colon c_1, c_2, \ldots, c_n,$$

где *type* – ключевое слово; T – имя типа; c_1, c_2, ..., c_n – константы.

Переменная определяется следующим образом:

$$variable\ x_1, x_2, \ldots, x_m : T,$$

где *variable* – ключевое слово; T – имя типа; x_1, x_2, ..., x_m – переменные.

Обычно логика предикатов имеет одну предметную область, которая является областью определения для всех возможных переменных. Однако с точки зрения здравого смысла лучше рассматривать предметную область, состоящую из нескольких типов объектов, как это сделано в классической математике и в программировании. Кванторы $\forall x$ и $\exists x$ не обязаны теперь пробегать всю полную предметную область, а только тот тип объектов, какому соответствует переменная x.

1.2. Высказывания и предикаты

Тексты, описывающие некоторую область знаний, включают совокупность *высказываний*, под которым подразумевается смысл некоторого повествовательного предложения. Высказывание *верно (истинно)*, если то, о чем оно повествует, действительно имеет место; и высказывание *неверно (ложно)*, если то, о чем оно повествует, не соответствует действительности.

Простые высказывания объединяются в более сложные при помощи логических связок *«или», «и», «нет», «если»* и т.д. Самыми простыми высказываниями являются *атомные высказывания* – нерасчленимые высказывания для порождения других.

Высказывания, соединенные связкой *«или»*, называются *дизъюнкцией*. Дизъюнкции бывают двухчленные, трехчленные, четырехчленные и т.д. Если один из членов дизъюнкции является истинным, то и вся дизъюнкция считается истинной, иначе она является ложной.

Конъюнкцией называются высказывания, соединенные связкой *«и»*. По аналогии с дизъюнкцией конъюнкция тоже может быть многочленной. Если все члены конъюнкции является истинным, то и вся конъюнкция считается истинной, иначе она является ложной.

Высказывание вида *«не А»* называется *отрицанием*, где A – высказывание, *«не»* – унарная логическая связка. Отрицание истинное, если A – ложное и отрицание ложное, если A – истинное.

Импликацией называется высказывание вида *«если А, то В»,* где A, B – высказывания, *«если ..., то ...»* – логическая связка. Импликация является

ложной, если A есть истина, а B – ложь. Во всех остальных сочетаниях истинностных значениях A и B импликация принимает значение истина.

Эквивалентность (читается «*A эквивалентно B*») имеет значение «истина» тогда, когда совпадают значения высказываний A и B. Во всех остальных случаях эквивалентность считается ложной.

С высказываниями тесно связаны *предикаты*, под которыми будем понимать шаблоны текста, состоящие из постоянной части и имен типов для подстановки других имен. Предикаты дают осмысленные высказывания при замене имен типов константами того же типа. Примеры предикатов: *«число ЧИСЛА простое», «2 + ЧИСЛА = 8», «ЧЕЛОВЕК любит ЧЕЛОВЕК»*, где тип *ЧИСЛА* – обозначает целые числа, тип *ЧЕЛОВЕК* – обозначает людей. Предикаты можно обозначать с помощью постоянной части текста, за которой сразу идет перечень имен типов в круглых скобках, например, для предыдущих примеров: *«простое число(ЧИСЛА)», «два плюс нечто равно восемь(ЧИСЛА)», «любит(ЧЕЛОВЕК, ЧЕЛОВЕК)»*. Предикат истинен, если при замене имен типов на константы таких же типов получается истинное высказывание, и предикат ложен, если при аналогичных заменах получается ложное высказывание.

Предикат мы будем определять при помощи следующего выражения:

$$predicate: P(T_1, T_2, ..., T_n),$$

где P – имя постоянной части предиката; T_1, T_2, ..., T_n – имена типов; $n \geq 0$ – порядок (арность) предиката.

Бинарный предикат $P(T_1, T_2)$ можно изобразить в виде $(T_1\ P\ T_2)$ или же вообще без скобок. *0*-арный предикат $P( )$ обозначает некоторое высказывание, которое, как правило, записывается в виде P.

По определению произвольный предикат $P(T_1, T_2, ..., T_n)$ разрешим на основе связанной с ним таблицы 1.

Таблица 1

T_1	T_2		T_n
a_{11}	a_{12}	...	a_{1n}
a_{21}	a_{22}	...	a_{2n}
⋮	⋮		⋮
a_{k1}	a_{k2}	...	a_{kn}

В колонках таблицы записаны константы соответствующих типов, а в строках – константы, при которых предикат превращается в истинное высказывание, если заменить этими константами соответствующие типы в $P(T_1, T_2, ..., T_n)$. Если типы заменить значениями, не являющимися строками таблицы, то получившееся высказывание будет ложным.

Предикаты изучаются в логике предикатов, которая представляет собой универсальный абстрактный язык описания знаний и является неотъемлемой частью фундамента современной математики. В этом смысле логика S_2 строится как специальная логика предикатов для описания, изучения и манипулирования алгоритмами.

2. Формулы

Алфавит языка S_2 содержит основные, специальные и вспомогательные символы. Основные символы – это буквы естественного языка, специальные символы – логические связки, вспомогательные – это разделители типа скобок и запятых. Из основных символов составляются слова, обозначающие константы, переменные, предикаты и другие понятия S_2.

Пусть $P(T_1, T_2, ..., T_n)$ – предикат, а $v_1, v_2, ..., v_n$ – константы и переменные, относящиеся соответственно к типам $T_1, T_2, ..., T_n$, тогда выражение $P(v_1, v_2, ..., v_n)$, полученное подстановкой $v_1, v_2, ..., v_n$ вместо $T_1, T_2, ..., T_n$, называется *атомарной формулой* или *атомом*. Атомы служат «кирпичиками» для построения более сложных формул при помощи логических связок и скобок. Связками могут быть: $\neg$ – «*нет*», $\&$ – «*и*», $\vee$ – «*или*», $\supset$ – «*если ..., то ...*», $\sim$ – «*эквивалентно*».

Логические формулы, или просто формулы определим по индукции:

1. Атомарная формула есть формула.
2. Если A, B – формулы, то выражения

$$\neg A, (A \vee B), (A \,\&\, B), (A \supset B), (A \sim B)$$

также являются формулами.

Совокупность различных переменных формулы называются ее *параметрами* или *аргументами*. Любая формула выражает некоторый предикат, зависящий от своих параметров.

Иногда для удобства восприятия большую и сложную формулу необходимо изобразить в более коротком виде. Сделать это можно при помощи следующего формального определения:

$$A(x_1, x_2, ..., x_n) ::= \text{«определяемая формула»},$$

где A – имя определяемой формулы; $x_1, x_2, ..., x_n$ – параметры определяемой формулы, $::=$ – символ «равно по определению». Введенное обозначение формулы можно записывать без параметров и скобок, т.е. в виде A, если это не приведет к недоразумениям.

Разрешение формулы осуществляется по индукции:

1. Если формула является атомарной, то она разрешима по определению.
2. Если истинностные значения формул A и B известны, то истинность формул

$$\neg A, (A \vee B), (A \,\&\, B), (A \supset B), (A \sim B)$$

определяются по правилам для связок.

Формула соответствует *логической функции*, областью определения которой являются все возможные значения ее параметров, а область значений – истина или ложь. В матлогике логическая функция обычно задается таблицей истинности.

Пусть имеем *type T*: c_1, c_2, ..., c_k и *variable x*: *T,* тогда допустимы следующие два формальных определения:

$$\forall xA(x) ::= A(c_1)\ \&\ A(c_2)\ \&\ ...\ \&\ A(c_k),$$

$$\exists xA(x) ::= A(c_1) \vee A(c_2) \vee ... \vee A(c_k).$$

Символ $\forall$ называется *квантором общности*, а символ $\exists$ – *квантором существования*. Формулы $\forall xA(x)$ и $\exists xA(x)$ могут быть использованы для построения других формул, при этом вхождение переменной x в построенную формулу называется *связанным*.

Формула A называется *общезначимой* (символически $\vdash A$), тогда когда она является истинной для всех допустимых распределений значений ее параметров. При помощи выражения $\Rightarrow A$ мы будем обозначать тот факт, что формула A является истинной для текущих значений своих параметров.

Мы говорим, что формула B является следствием из формул $A_1, A_2, \cdots A_m$ (это символически записывается в виде $A_1, A_2, \cdots A_m \models B$) тогда, когда выполняется следующее условие: если все $A_1, A_2, \cdots A_m$ принимают значение «истина», то B также принимает значение «истина». Выражение $A_1, A_2, \cdots A_m \models B$ назовем *логической секвенцией*. Понятие секвенции в математической логике впервые ввел Г. Генцен [12] для формализации логического вывода.

Можно доказать, что для логики S_2 секвенция $A_1, A_2, \cdots A_m \models B$ разрешима. Другими словами – можно построить алгоритм, который определяет истинностное значение этой секвенции.

Пусть $\Gamma \models B_1, \Gamma \models B_2, \ldots, \Gamma \models B_n$ – логические секвенции, тогда по отношению к ним можно применять такое определение как

$$\Gamma \models B_1, B_2, \ldots, B_n ::= \Gamma \models B_1, \Gamma \models B_2, \ldots, \Gamma \models B_n.$$

Для списка формул введем следующее определение:

$$\Delta(x_1, x_2, \ldots, x_n) ::= A_1, A_2, \cdots, A_k,$$

где $\Delta(x_1, x_2, \ldots, x_n)$ – обозначение списка формул; $A_1, A_2, \cdots, A_k$ – список формул для $k \geq 1$; $x_1, x_2, \ldots, x_n$ – объединение параметров формул $A_1, A_2, \cdots, A_k$. Параметры $x_1, x_2, \ldots, x_n$ называются *параметрами списка формул* Δ.

Если $\Rightarrow A_1, \Rightarrow A_2, \ldots \Rightarrow A_k$ – список истинных формул для $k \geq 1$, то тогда имеем определение:

$$\Rightarrow A_1, A_2, \ldots A_k ::= \Rightarrow A_1, \Rightarrow A_2, \ldots \Rightarrow A_k.$$

Пусть $\Delta(x_1, x_2, ..., x_n)$ – список формул с параметрами $x_1, x_2, ..., x_n$. Тогда будем считать, что выражение $\Rightarrow \Delta(x_1, x_2, ..., x_n)$ определяет *состояние памяти* для $x_1, x_2, ..., x_n$. В S_2 списку $\Delta(x_1, x_2, ..., x_n)$ соответствует таблица с колонками $x_1, x_2, ..., x_n$ и со строками, для которых справедливо $\Rightarrow \Delta(x_1, x_2, ..., x_n)$.

3. Вычислительные секвенции

Наиболее общей моделью алгоритма (также как и другой системы) является модель черного ящика, который изображен на рис.1. В этом случае алгоритм представляется в виде прямоугольника, внутреннее устройство которого скрыто или неизвестно для внешнего наблюдателя. Алгоритм взаимодействует с внешней средой посредством своих входов и выходов так, что состояние выходов функционально зависит от состояний входов.

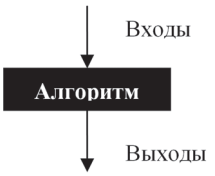

Рис. 1

Алгоритм кратко описывается своей *спецификацией*, под которой будем понимать совокупность *вычислительных секвенций*, представляющих собой выражения вида:

$$\Lambda(x_1, x_2, \ldots, x_n) \xrightarrow{r} \Delta(y_1, y_2, \ldots, y_m),$$

где r – имя спецификации алгоритма; список формул $\Lambda(x_1, x_2, \ldots, x_n)$ называется *условием секвенции*; список формул $\Delta(y_1, y_2, \ldots, y_m)$ называется *заключением секвенции*. Параметры условия $x_1, x_2, \ldots, x_n$ называются *входами секвенции (алгоритма)*, а параметры заключения $y_1, y_2, \ldots, y_m$ – *выходами секвенции (алгоритма)*. Условие и заключение можно рассматривать как конъюнкции их формул. *Условие секвенции выполняется*, если имеет место $\Rightarrow \Lambda(x_1, x_2, \ldots, x_n)$; это влечет за собой дальнейшее выполнение алгоритма. Если же условие не выполняется, то алгоритм также не выполняется.

Имя спецификации состоит из двух частей: первая часть написана строчными буквами, вторая – начинается с заглавной буквы, а все остальные строчные. Первая часть имени совпадает с именем алгоритма, а вторая часть – выбирается произвольно. Иногда имя спецификации над стрелкой мы будем опускать, если это не приведет к недоразумениям. Введем определение:

$$\rightarrow (y_1, y_2, \ldots, y_m) := r(x_1, x_2, \ldots, x_n) ::=$$

$$\Lambda(x_1, x_2, \ldots, x_n) \xrightarrow{r} \Delta(y_1, y_2, \ldots, y_m),$$

где символ $:=$ называется *символом присвоения*. Такую секвенцию допускается изображать без стрелки, т.е. в виде выражения

$$(y_1, y_2, \ldots, y_m) := r(x_1, x_2, \ldots, x_n).$$

Вычислительную секвенцию, представленную в форме с символом присвоения, назовем *секвенцией присвоения*.

В логике S_2 имеются секвенция такая как

$$\Lambda(x) \xrightarrow{copy} \Delta(y) \text{ или } \rightarrow y := copy(x),$$

которая копирует значение x в y. Для подобных секвенций будем использовать следующее определение:

$$\rightarrow y := x ::= \rightarrow y := copy(x).$$

Вычислительная секвенция с одним выходом называется *функциональной секвенцией*.

Пусть $\rightarrow y := f(x_1, x_2, \ldots, x_n)$ – функциональная секвенция присвоения, тогда $f(x_1, x_2, \ldots, x_n)$ назовем *функцией*. Считается, что 0-арная функция – это константа [13].

В классической логике предикатов используется понятие *терма*, который по смыслу соответствует математическому выражению. Терм в S_2 вводится по индукции следующим образом:

1. Переменные, константы и функции являются термами.

2. Если x – переменная, t и v – произвольные термы, $t|_v^x$ – результат замены x на v в t, то тогда $t|_v^x$ также является термом.

4. Секвенциальные схемы алгоритмов

Пара вычислительных секвенций называется *независимой*, если она обладает следующими свойствами:

1. Входы одной из них не содержатся среди выходов другой, однако, входы и выходы одной и той же секвенции могут совпадать.

2. Все выходы секвенций различаются между собой.

Совокупность вычислительных секвенций $r_1, r_2, \ldots, r_n$ является *независимыми секвенциями*, если любая пара из них независима.

Пусть $r_1, r_2, \ldots, r_n$ – независимые секвенции, тогда выражение

$$p : [r_1, r_2, \ldots, r_n]$$

называется *блок-объединением*, где p – имя этого объединения, $n \geq 0$. Блок-объединение будет записываться в одну строку (по необходимости с переносом). Считаем, что блок-объединение определяет совокупность алгоритмов, выполняемых параллельно.

При выполнении блок-объединения его секвенции могут быть выполнены все, частично или даже ни одна из них. Те секвенции, которые выполняются до конца, назовем *активными секвенциями*. Блок-объединение, содержащий активные секвенции, называется *активным блок-объединением*. Состояние памяти AK может изменяться только под воздействием активных блок-объединений. Совокупность активных секвенций активного блок-объединения получила название *активной строки*.

4.1. Алгоритмические блоки

Полное описание алгоритма в логике S_2 представлено *секвенциальной схемой алгоритма*, которая состоит из отдельных частей, названных *(алгоритмическими) блоками*. Вся секвенциальная схема также есть блок, а те блоки, которые в него входят, называются *внутренними блоками* или *подблоками*. Для подблока тот блок, куда он входит, называется *внешним блоком*. В общем случае блок обозначается выражением вида

$$p : [T],$$

где p – имя блока, T – внутренние блоки. Квадратные скобки и имена отдельных блоков можно опускать, если это не приведет к недоразумениям. Блок может быть пустым.

В состав некоторых блоков кроме вычислительных секвенций войдут еще и *управляющие секвенции*, которые представляют собой такие выражения как

$$(\Delta \downarrow p) \text{ или } (\Delta \uparrow p),$$

где p – имя блока, Δ – список формул (он может быть пустым). Выражение $(\Delta \downarrow p)$ получило название *break-секвенции*, а $(\Delta \uparrow p)$ – *continue-секвенции*. Список формул Δ является *условием управляющей секвенции*, понятие которой совпадает с понятием условия вычислительной секвенции. Совокупность параметров списка формул Δ являются *входами управляющей секвенции*.

Алгоритмический блок (секвенциальную схему алгоритма) определим так:

1. Блок-объединение есть блок.

2. Если $[K], [M]$ – блоки, тогда выражение $p : \begin{bmatrix} [K] \\ [M] \end{bmatrix}$ также является блоком, и называется он *композицией блоков*.

3. Если после одного из подблоков блока p вставить управляющую секвенцию, то полученное выражение есть блок, и называется он *управляемым блоком*.

Из определения блока следует, что секвенциальная схема состоит из последовательности строк вычислительных и управляющих секвенций, структурированных квадратными скобками. Управляющая секвенция всегда образует одну строку.

Выполнение алгоритмического блока осуществляется в соответствии со следующим правилом:

1. Объединения секвенций – его выполнение рассмотрено выше.

2. Композиция выполняется последовательно сверху вниз.

3. Если при выполнении управляемого блока p очередь дошла до break-секвенции $(\Delta \downarrow p)$ и Δ есть истина, то тогда произойдет выход из блока p. Если Δ ложное, то тогда будет выполняться компонент секвенциальной схемы, следующий за $(\Delta \downarrow p)$.

4. Если при выполнении управляемого блока p очередь дошла до continue-секвенции $(\Delta \uparrow p)$ и Δ есть истина, то тогда блок p начнет выполняться с самого начала. Если Δ ложное, то тогда будет выполняться компонент секвенциальной схемы, следующий за $(\Delta \uparrow p)$.

Пример 1 (алгоритмический блок)

$$\left[\begin{array}{c} y_1 := x + y;\ \ y_2 := z - m\ \ \ y_3 := x - z \\ y_4 := y_3 + y;\ \ y_5 := y_1 + y_2 \\ y_6 := y_4 * y_5 \end{array}\right].$$

Этот блок вычисляет выражение

$$(((x + y) + (z - m)) * ((x - z) + y))$$

и относится к типу *линейных блоков*, под которыми понимаются блоки без управляющих секвенций.

4.2. Информационный граф блока

Для анализа информационных взаимосвязей линейного блока можно использовать орграф со взвешенными дугами. *Орграф* или *ориентированный граф* – это упорядоченная пара $G = (V, D)$, где V – множество *вершин* или *узлов*, D – множество дуг. *Дуга* – это упорядоченная пара вершин графа (v, w), где вершина v называется началом, а w – концом дуги. Можно сказать, что дуга $v\ \ w$ ведет от вершины v к вершине w. Граф является взвешенным, если множеству дуг поставлена в соответствие некоторая величина, называемая *весом* дуги. Тот факт, что дуга (v, w) имеет вес u, запишем с помощью упорядоченной тройки (u, v, w). Таким образом, орграф с взвешенными дугами – это упорядоченная пара $G = (V, D)$, где V – множество *вершин* или *узлов*, D – множество взвешенных дуг (u, v, w).

Интерпретация алгоритмического блока взвешенным орграфом осуществляется по следующему правилу:

1. Блок индексируется так, что между его вычислительными секвенциями и индексами установится взаимно-однозначное соответствие.

2. Индексы секвенций – это вершины орграфа.

3. Пусть s и r – две секвенции, а u – выход секвенции s, который одновременно является входом секвенции r. Тогда тройка (u, i_s, i_r) – взвешенная дуга орграфа, где i_s, i_r – индексы секвенций s и r.

Граф, построенный по этому правилу, назовем *информационным графом блока*, а его пример показан на рис. 2.

Граф на рис. 2 имеет три уровня: {1, 2, 3}, {4, 5}, {6}, каждый из которых соответствует одному блок-объединению.

4.3. Производные блоки

Те блоки, которые были введены выше, а именно объединение, композиция и управляемый блок, назовем *основными блоками*, а блоки, которые мы сейчас определим, назовем *производными*. Производные блоки, как правило, моделируют известные конструкции программирования, такие как оператор цикла, условный оператор и другие.

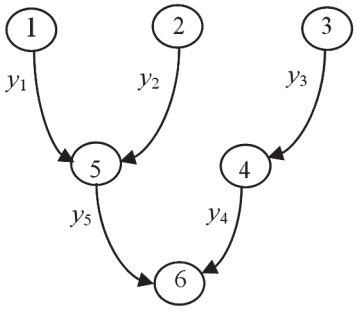

Рис. 2. Информационный граф для блока из примера 1

While-цикл с постусловием определяется следующим образом:

$$[T](A) ::= p : \left[\begin{array}{c} [T] \\ (A \uparrow p) \end{array}\right],$$

где A – формула под названием «*условие цикла*»; $[T]$ – блок под названием «*тело цикла*».

Пусть $[T](A)$ – *while*-цикл с постусловием, тогда *while-цикл с предусловием* определится так:

$$(A)[T] ::= r : \left[\begin{array}{c} (\neg A \downarrow r) \\ [T](A) \end{array}\right].$$

Цикл *for* соответствует такому блоку как

$$\left[\begin{array}{c} i := a \\ A(i)\left[\begin{array}{c} T \\ i := f(i) \end{array}\right] \end{array}\right],$$

где $i := a$ – инициализация цикла; $A(i)$ – условие цикла; $i := f(i)$ – шаг цикла; T – тело цикла.

Условный блок if-then вводится следующим образом:

$$r : [A \to [T]] ::= r : \left[\begin{array}{c} [\ ] \\ (\neg A \downarrow r) \\ [T] \end{array}\right],$$

где $[T]$ – блок, $A \to [T]$ – условный блок *if-then*; A – формула, которая называется *условием блока*. Блок $[T]$ выполняется, когда условие A есть истина.

Оператор *if-then-else* моделируется при помощи следующей композиции двух условных блоков *if-then*:

$$p : \left[\begin{array}{c} A \to \left[\begin{array}{c} [K] \\ (\downarrow p) \end{array}\right] \\ \neg A \to [M] \end{array}\right].$$

Эту композицию можно сократить на основе такого определения:

$$A \to [K][M] ::= p : \left[\begin{array}{c} A \to \left[\begin{array}{c} [K] \\ (\downarrow p) \end{array}\right] \\ \neg A \to [M] \end{array}\right].$$

Возникает вопрос, какие алгоритмы можно представить секвенциальными схемами? Ответ на него дает теорема, приведенная ниже.

Теорема 1 (о полноте). При помощи секвенциальной схемы можно реализовать любой алгоритм.

Доказательство:

1. «Алгоритм любой сложности можно реализовать, используя только три конструкции: следования, выбора и повторения» (Теорема Дейкстры).

2. Конструкция следования соответствует композиции блоков S_2, конструкции выбора и повторения определены как производные блоки (см. выше).

3. Утверждение теоремы (пп 1, 2).

5. О доказательстве в логике S_2

Обычно в программировании правильность программ проверяется путем тестирования, однако это не дает нужного эффекта – ошибки могут выявляться на протяжении всей жизни программы. Выдающийся ученый в области информатики Э. В. Дейкстра говорил, что программист должен использовать теорию, необходимую для подтверждения алгоритма. В этом смысле можно сказать, что такой теорией является логика S_2, где делается попытка представить рассуждения относительно алгоритмов в виде формального доказательств.

Алгоритм является правильным, если его секвенциальная схема построена из правильных вычислительных секвенций, и существует доказательство спецификации этого алгоритма. Доказательство вычислительных секвенций строится на основе *схемных правил*, а спецификации – на основе *правила изменения памяти* и *логических правил*. Все виды правил являются фигурами заключения генценовского типа, приспособленные для логики S_2. Последовательность построения секвенциальной схемы правильного алгоритма такова: вначале доказываются все секвенции схемы, затем из них строится сама схема, и наконец доказываются ее спецификация.

5.1. Доказательство секвенций схемы алгоритма

Рассмотрим некоторые из схемных правил.

Пусть $\Lambda \xrightarrow{r} \Delta$ – вычислительная секвенция, $\to x := t$ – функциональная секвенция присвоения, t – терм, $\Lambda\big|_t^x$ – результат замены переменной x на t в списке формул Λ. Тогда справедливо правило замены входа секвенции, сформулированное ниже.

Правило замены входа секвенции

$$\frac{\to x := t;\ \Lambda \xrightarrow{r} \Delta}{\Lambda\big|_t^x \xrightarrow{r} \Delta}.$$

На основе этого правила можно получить секвенцию, где в формулах условия содержатся произвольные термы.

Пусть $\Lambda \xrightarrow{r} \Delta$ – вычислительная секвенция, x – параметр списка формул Δ, v – переменная такого же типа, как и x, $\Delta\big|_v^x$ – результат замены x на v в Δ. Тогда имеет место правило замены выхода (см. ниже).

Правило замены выхода

$$\frac{\Lambda \xrightarrow{r} \Delta}{\Lambda \xrightarrow{r} \Delta\big|_v^x}.$$

В схемных правилах, приведенных ниже, используются следующие обозначения: Δ, Γ, Ψ – списки формул, A, B – одиночные формулы.

Правило перестановки в условии

$$\frac{\Delta, A, B, \Gamma \xrightarrow{r} \Psi}{\Delta, B, A, \Gamma \xrightarrow{r} \Psi}$$

Правило перестановки в заключении

$$\frac{\Delta \xrightarrow{r} \Psi, A, B, \Gamma}{\Delta \xrightarrow{r} \Psi, B, A, \Gamma}$$

Правило сокращения в условии

$$\frac{\Delta, A, A, \Gamma \xrightarrow{r} \Psi}{\Delta, A, \Gamma \xrightarrow{r} \Psi}$$

Правило сокращения в заключении

$$\frac{\Delta \xrightarrow{r} \Psi, A, A, \Gamma}{\Delta \xrightarrow{r} \Psi, A, \Gamma}$$

Правило введения конъюнкции в условии

$$\frac{\Delta, A, B, \Gamma \xrightarrow{r} \Psi}{\Delta, A \,\&\, B, \Gamma \xrightarrow{r} \Psi}$$

Правило введения конъюнкции в заключении

$$\frac{\Delta \xrightarrow{r} \Psi, A, B, \Gamma}{\Delta \xrightarrow{r} \Psi, A \,\&\, B, \Gamma}$$

Правило введения дизъюнкции в условии

$$\frac{A, \Delta \xrightarrow{r} \Psi;\ B, \Delta \xrightarrow{r} \Psi}{A \vee B, \Delta \xrightarrow{r} \Psi}$$

Правило удаления импликации в заключении

$$\frac{\Delta \xrightarrow{r} A;\ \Delta \xrightarrow{r} A \supset B}{\Delta \xrightarrow{r} B}$$

1-е правило утончения

$$\frac{\Gamma \xrightarrow{r} \Psi}{\Delta, \Gamma \xrightarrow{r} \Psi}$$

2-е правило утончения

$$\frac{\to (y_1, y_2, \ldots, y_m) := r(t_1, t_2, \ldots, t_n)}{\Delta \to (y_1, y_2, \ldots, y_m) := r(t_1, t_2, \ldots, t_n)}$$

где $t_1, t_2, \ldots, t_n$ – термы.

Выражение под чертой 2-го правила утончения также является секвенцией присвоения. Для этого вида секвенций может быть полезным следующее определение:

$$\Delta \to (y_1, y_2, \ldots, y_n) := (t_1, t_2, \ldots, t_n) ::=$$
$$\Delta \to y_1 := t_1, \Delta \to y_2 := t_2, \ldots, \Delta \to y_n := t_n,$$

где $t_1, t_2, \ldots, t_n$ – термы.

Конечная последовательность вычислительных секвенций $s_1, s_2, \ldots, s_n$ называется *доказательством*

секвенции схемы алгоритма s_n, если эта последовательность удовлетворяет следующему условию: каждая секвенция $s_i, i \le n$ либо является спецификацией алгоритма из вычислительного ресурса *AK*, либо получена по одному из схемных правил и некоторых секвенций $s_j, j < n$.

5.2. Доказательство спецификации алгоритма

Алгоритм изменяет свою память по шагам. На очередном шаге находится активный блок-объединение, который сразу выполняется, что приводит к изменению состояния памяти. Работу алгоритма со спецификацией $\Delta \to \Omega$ можно описать последовательностью состояний памяти

$$\Rightarrow \Delta_1$$
$$\Rightarrow \Delta_2$$
$$\dots$$
$$\Rightarrow \Delta_m,$$

где $\Rightarrow \Delta_1$ – начальное состояния памяти (Δ_1 совпадает с Δ), $\Rightarrow \Delta_m$ – конечное состояние памяти. *Алгоритм удовлетворяет спецификации*, если имеет место $\Delta_m \models \Omega$. *Алгоритм является правильным*, если он удовлетворяет всем своим спецификациям.

Активный блок-объединение изменяет состояние памяти по правилу, приведенному ниже.

Правило изменения памяти

$$\frac{\Rightarrow \Delta; \Delta \vdash \Lambda_1, \Lambda_2, \dots, \Lambda_n; \Lambda_1 \to \Gamma_1; \Lambda_2 \to \Gamma_2; \dots; \Lambda_n \to \Gamma_n}{\Rightarrow \exists\, y_1 \exists\, y_2 \dots \exists\, y_m(\Delta), \Gamma(y_1, y_2, \dots, y_m)},$$

где $\Rightarrow \Delta$ – состояние памяти до выполнения блока, $\Lambda_1 \to \Gamma_1, \Lambda_2 \to \Gamma_2, \dots, \Lambda_n \to \Gamma_n$ – активные секвенции, $\Gamma(y_1, y_2, \dots, y_m)$ – объединение формул из заключений активных секвенций, $y_1, y_2, \dots, y_m$ – объединение выходов активных секвенций, $\exists y_1 \exists y_2 \dots \exists y_m(\Delta)$ – формулы, полученные из списка Δ приписыванием перед каждой его формулой кванторов $\exists y_1 \exists y_2 \dots \exists y_m$, выражение под чертой правила – это состояние памяти после выполнения блока. Лишние кванторы в списке формул $\exists y_1 \exists y_2 \dots \exists y_m(\Delta)$ можно будет сократить, и в результате останутся только кванторы перед формулами, содержащими связанные ними переменные. Если список формул Δ не содержат переменных $y_1, y_2, \dots, y_m$, то тогда это правило запишется в виде правила изменения памяти.

Правило изменения памяти

$$\frac{\Rightarrow \Delta; \Delta \vdash \Lambda_1, \Lambda_2, \dots, \Lambda_n; \Lambda_1 \to \Gamma_1; \Lambda_2 \to \Gamma_2; \dots; \Lambda_n \to \Gamma_n}{\Rightarrow \Delta, \Gamma_1, \Gamma_2, \dots, \Gamma_n}.$$

Доказательство спецификации алгоритма $\Delta \to \Omega$ представляет собой конечную последовательность

$$\Rightarrow \Delta_1$$
$$\Rightarrow \Delta_2$$
$$\dots$$
$$\Rightarrow \Delta_m$$
$$\Delta_m \models \Omega,$$

где $\Rightarrow \Delta_1, \Rightarrow \Delta_2, \dots, \Rightarrow \Delta_m$ – состояния памяти, которые пробегает алгоритм в процессе его выполнения, $\Rightarrow \Delta_1$ – начальное состояния памяти, $\Rightarrow \Delta_m$ – конечное состояние памяти. Последовательность состояний памяти должна удовлетворять следующим условиям:

1. Δ_1 совпадает с Δ.
2. Каждое состояние памяти $\Rightarrow \Delta_i, 1 < i \le m$ получено по правилу изменения памяти.
3. Логическая секвенция $\Delta_m \models \Omega$ доказывается при помощи логических правил классической математической логики.

6. Эквивалентность алгоритмов

Секвенция $\Lambda(x_1, x_2, \dots, x_n) \to \Delta(y_1, y_2, \dots, y_m)$ и секвенция $\Gamma(x_1, x_2, \dots, x_n) \to \Omega(y_1, y_2, \dots, y_m)$ называются *эквивалентными*, если для одинаковых значений входов $x_1, x_2, \dots, x_n$ их выходам $y_1, y_2, \dots, y_m$ всегда будут присваиваться одинаковые значения. Формально эквивалентность секвенций изобразится при помощи такого выражения как

$$\Lambda(x_1, x_2, \dots, x_n) \to \Delta(y_1, y_2, \dots, y_m) \approx$$
$$\Gamma(x_1, x_2, \dots, x_n) \to \Omega(y_1, y_2, \dots, y_m),$$

где $\approx$ – символ эквивалентности вычислительных секвенций (спецификаций).

Из определения понятия эквивалентности секвенций вытекает следующая теорема.

Теорема 2. Если для двух вычислительных секвенций справедливо $\Lambda \to \Delta \approx \Gamma \to \Omega$, то тогда $\Lambda \models \Gamma$ и $\Gamma \models \Lambda$.

Следствие. Если для двух вычислительных секвенций справедливо $\Lambda \to \Delta \approx \Gamma \to \Omega$, то тогда $\Lambda \to \Omega$ и $\Gamma \to \Delta$.

Два алгоритма эквивалентны, если между множествами их спецификаций существует такое взаимно-однозначное соответствие, где каждая пара – эквивалентные секвенции. Эквивалентность алгоритмов обозначается выражением $P \approx R$, где *P*, *R* – алгоритмы, $\approx$ – символ эквивалентности алгоритмов, который совпадает с символом эквивалентности спецификаций.

Приведем без доказательства несколько теорем, которые могут быть использованы для эквивалентных преобразований секвенциальных схем.

Теорема 3. Пусть *T, R* – блок-объединения такие, что *R* отличается от *T* лишь порядком расположения секвенций, тогда $T \approx R$.

Теорема 4. Пусть T_A – секвенциальная схема, в которой выделено вхождение формулы *A*, а T_B – секвенциальная схема, полученная из T_A заменой *A* на формулу *B*. Тогда, если $A \sim B$, то $T_A \approx T_B$.

Теорема 5. Пусть T_A – секвенциальная схема, в которой выделено вхождение блока *A*, а T_B – секвенциальная схема, полученная из T_A заменой *A* на блок *B*. Тогда, если $A \approx B$, то $T_A \approx T_B$.

Теорема 6. Пусть $[\Psi_S]$ – блок-объединение, в котором выделена последовательность вычислительных секвенций S. Тогда $[\Psi_S] \approx \begin{bmatrix} [\Psi] \\ [S] \end{bmatrix}$ и $[\Psi_S] \approx \begin{bmatrix} [S] \\ [\Psi] \end{bmatrix}$, где $[\Psi]$ – блок-объединение, в котором удалена последовательность секвенций S, $[S]$ – блок-объединение, состоящий из последовательности секвенций S.

Следствие. Пусть $[S_1, S_2, \ldots, S_n]$ – блок-объединение, который состоит из вычислительных секвенций $S_1, S_2, \ldots, S_n$, тогда $[S_1, S_2, \ldots, S_n] \approx \begin{bmatrix} S_1 \\ S_2 \\ \ldots \\ S_n \end{bmatrix}$.

Теорему 6 и следствие из нее можно использовать для распараллеливания алгоритмов.

Выводы

Новая логика S_2 является специальной логикой предикатов для изучения и манипулирования прикладными алгоритмами. Ее можно использовать для проектирования знаниеориентированных интеллектуальных систем программного обеспечения. В логике создается механизм формального доказательства правильности алгоритмов и их эквивалентности. Алгоритмы можно распараллеливать. Схемы алгоритмов удобно редактировать в стандартных редакторах формул, таких как MathType.

Список литературы: 1. *Ляпунов, А.А.* О некоторых общих вопросах кибернетики [Текст] / А.А. Ляпунов // Сб. «Проблемы кибернетики». – М. : Физматгиз, 1958. – Вып. 1. – С. 5-22. 2. *Янов, Ю.И.* О логических схемах алгоритмов [Текст] / Ю.И. Янов // Сб. «Проблемы кибернетики». – М. : Физматгиз, 1958. – Вып. 1. – С. 46-74. 3. *Калужнин, Л.А.* Об алгоритмизации математических задач // Сб. «Проблемы кибернетики». – М.: Физматгиз,1959. – Вып. 2. – С. 51-58. 4. *Заславский, И.Д.* Граф-схемы с памятью [Текст] / И.Д. Заславский // Сб. «Труды математического института им. В.А. Стеклова». – М.-Л. : изд-во «Наука», 1964. – Вып. LXXII. – С. 99-192. 5. *Королюк, В.С.* О понятии адресного алгоритма [Текст] / В.С. Королюк // Сб. «Проблемы кибернетики». – М. : Физматгиз, 1960. – Вып. 4. – С. 95-110. 6. *Ющенко, Е.Л.* Адресное программирование [Текст] / Е.Л. Ющенко. – К. : Изд-во «Техническая литература», 1963. – 288 с. 7. *Вельбицкий, И.В.* Технология программирования [Текст] / И.В. Вельбицкий. – К. : Техника, 1984. – 279 с. 8. *Рамбо, Дж.* UML: специальный справочник [Текст] / Дж. Рамбо, А. Якобсон, Г. Буч. – СПб.: Питер, 2002. – 656 с. 9. *Кашубин, С.П.* Язык секвенциальных схем алгоритмов [Текст] / *С.П. Кашубин*; Ин-т пробл. Машиностроения АН УССР. – Харьков, 1983. – 20 с. – Деп. В ВИНИТИ 05.07.83, №3963. 10. *Кашубин, С.П.* Приведение формул исчисления высказываний к конъюнктивно нормальной форме [Текст] / *С.П. Кашубин*; Ин-т пробл. Машиностроения АН УССР. – Харьков, 1986. – 30 с. – Деп. В ВИНИТИ 18.06.86, №4483-В86. 11. *Кашубин, С.П.* ИКС-технология (создание интегрированных компьютерных систем) [Текст] / *С.П. Кашубин* // Автоматизированные системы управления и приборы автоматики. 2006. Вып. 135. С. 47-50. 12. Математическая теория логического вывода: Сб. переводов. – М.: изд-во «Наука», 1967. – 352 с. 13. *Шенфильд, Дж.* Математическая логика: Пер. с англ. – М.: Изд-во «Наука», 1975. – 528 с. 14. *Дейкстра, Э.* Дисциплина программирования: Пер. с англ. – М.: Мир, 1978. – 280 с.

Поступила в редколлегию 10.09.2010.

УДК 519.681

Логіка секвенціальних схем алгоритмів / С.П. Кашубін // Біоніка інтелекту: наук.-техн. журнал. – 2010. – № 3 (74). – С. 120–128.

Запропоновано нову теорію під назвою «Логіка секвенційних схем алгоритмів», яка призначена головним чином для програмування. В статті досліджуються питання доказу правильності алгоритмів, їх еквівалентні перетворення та розпаралелювання.

Табл. 1. Іл. 2. Бібліогр.: 14 найм.

UDC 519.681

The logic of sequential schemes of algorithms / S.P. Kashubin // Bionics of Intelligence: Sci. Mag. – 2010. – № 3 (74). – P. 120–128.

A new theory called “The logic of sequential schemes of algorithms”, designed mainly for programming, is proposed. The article investigates the proof of the correctness of algorithms, their equivalent transformations and parallelization.

Tabl. 1. Fig. 2. Ref.: 14 item.