Модел. и анализ информ. систем. Т. **18**, № **4** (2011) 168–180

УДК 519.681+519.683.8+519.688

Верификация шаблонов алгоритмов для метода отката и метода ветвей и границ

Шилов Н.В.[1]

Институт систем информатики им. А.П. Ершова СО РАН,
Новосибирский государственный университет и
Новосибирский государственный технический университет

e-mail: shilov@iis.nsk.su

получена 11 октября 2011 года

Ключевые слова: метод отката, метод ветвей и границ, абстрактные типы данных, частичная корректность, тотальная корректность, метод верификации Флойда, задача о расстанове ферзей, дискретная задача о ранце

Курс проектирования и анализа алгоритмов является обязательной составляющей учебных программ по информатике всех уровней. В университетах этот курс обязательно включает изучение структур данных, методы проектирования алгоритмов, теорию сложности и т.д. Этот курс знакомит с такими методами проектирования алгоритмов, как *жадные алгоритмы*, *динамическое программирование*, *метод разделяй и властвуй*, *метод отката*, *метод ветвей и границ*. Обычно знакомство с этими методами происходит на примерах. Но (как показано в данной статье) они могут быть (полу)формализованы в виде "паттернов", специфицированы условиями частичной и/или тотальной корректности и обоснованы (доказаны) методом Флойда верификации алгоритмов. Такой формализованный подход может привести к новому, более глубокому пониманию этих методов. В статье демонстрируется перспективность такого подхода на примере метода отката и метода ветвей и границ. В частности, мы доказываем, что разработанные шаблоны корректны, если граничное условие — монотонная функция, а решающее условие — антимонотонная функция от множества уже проверенных вершин.

1. Введение

Графовая задача перечисления вершин состоит в перечислении всех вершин входного (ор)графа (в дальнейшем — просто графа) с целью найти примеры (или все) вершины, которые удовлетворяют *целевому условию* (или *целевой функции*) C; если необходимо найти все вершины графа, удовлетворяющие целевому условию, то

[1]Работа поддержана проектом РФФИ 09-01-00361-а.

говорят о *полном перечислении*, а если надо найти хотя бы одну такую вершину, то говорят о *перечислении примеров*. *Обход в глубину* (*поиск в глубину*, *depth-first search*, DFS) — это подход к задаче перечисления конечного графа, при котором наследник вершины посещается раньше, чем его "братья" (наследники предка этой вершины). *Обход в ширину* (*поиск в ширину*, *breadth-first search*, BFS) — альтернативный подход к задаче перечисления конечного графа, когда братья вершины посещаются раньше, чем наследники вершины.

Однако зачастую для решения задачи перечисления графа нет необходимости обходить все вершины графа, чтобы найти те, которые удовлетворяют целевой функции, т. к. в нашем распоряжении есть дополнительное *граничное условие* B, которое гарантирует, что среди всех наследников текущей вершины нет удовлетворяющих целевому условию C: *метод отката* (*backtracking*, BTR) — это поиск в глубину с использованием граничного условия, а *метод ветвей и границ* (*branch-and-bound*, B&B) — это поиск в ширину также с использованием граничного условия.

Метод отката стал популярным после 1965 г. благодаря работам S.W. Golomb и L.D. Baumert [9], однако он был предложен ранее D. H. Lehmer. Метод ветвей и границ был разработан A.H. Land и A.G. Doig в 1960 [10].

К сожалению, дальнейшее развитие методов BTR и B&B свелось только к накоплению "примеров" и "рецептов" использования (прежде всего для решения комбинаторных и оптимизационных задач). В результате (на наш взгляд) сложилась достаточно плачевная ситуация в сфере образования, когда разделы, посвященные проектированию алгоритмов в соответствующей учебной литературе (например, [1, 4]), очень напоминают кулинарные книги.

Данная статья представляет вариант (частичной) формализации, спецификации и верификации шаблонов проектирования алгоритмов методом отката и методом ветвей и границ с целью повысить надёжность и корректность конкретных алгоритмов, спроектированных этими методами. По своему духу данная работа близка к работе [5], посвященной новым шаблонам циклов в языках программирования: шаблоны циклов *while*, *until* и *for* известны с начала 1960-х гг., но, как показано в [5], введение новых шаблонов может существенно повлиять на эффективность программирования, спецификации и верификации программ.

Статья имеет следующий план. Часть 2 описывает абстрактный тип данных, унифицирующий как очереди, так и стек. Затем следует часть 3, в которой описан унифицированный шаблон для алгоритмов отката и ветвей и границ; в этой же части приведён простой пример использования этого унифицированного шаблона для решения классической задачи о расстановке ферзей на шахматной доске методом отката. А вот следующая часть 4 посвящена более сложному примеру применения унифицированного шаблона для решения классической задачи о дискретном ранце методом ветвей и границ.

2. Абстрактный тип данных “Teque”

Определим генерический темпоральный абстрактный тип данных (АТД) *тека* (*theque* — (греч.) хранилище[2]). Пусть *type* — произвольный тип данных со множеством значений D. Тогда в качестве значений типа $Thequeoftype$ могут выступать конечные множества значений из D, помеченных различными *метками времени*. Эти метки времени — показания некоторых *глобальных часов*, которые отсчитывают время в *тиках*. Единица времени тик является неделимым квантом времени, и мы предполагаем, что любое действие занимает целое число тиков (возможно, разное в разные моменты времени). Метки времени у значений в теке соответствуют времени (по глобальным часам), когда эти значения попали в теку, они не могут быть изменены никаким образом и всегда не больше текущего показания глобальных часов. Мы будем представлять значение $x \in D$, помеченное меткой времени t, в виде пары (x, t). Показания глобальных часов и временные метки не доступны для непосредственного наблюдения.

АТД тека наследует некоторые теоретико-множественные операции: *empteq* (т.е. *пустая тека*) — это просто пустое множество ($\emptyset$), равенство ($=$) и неравенство ($\neq$), включения ($\subset$, $\subseteq$). Но, кроме того, АТД тека имеет свои собственные специфические операции, часть из которых зависит от времени исполнения, часть — не зависит, а часть “чувствительна” ко времени.

Начнём с перечисления операций, которые не зависят от времени.

- Операция *Set*: для любой теки T пусть $Set(T)$ — это $\{x \ : \ \exists t((x,t) \in T)\}$, т. е. множество всех значений, которые встречаются в T (с какой-либо меткой времени).

- Операции In и Ni: для любой теки T и произвольного значения $x \in D$ пусть $In(x, T)$ обозначает $x \in Set(T)$, а $Ni(x, T)$ — $x \notin Set(T)$.

- Операция *Spec специализации*: для любой теки T и одноместного предиката $\lambda x \in D.Q(x)$ пусть $Spec(T, Q)$ — это тека $\{(x,t) \in T \ : Q(x)\}$.

Единственная зависимая от времени операция на теках — это операция синхронного добавления $AddTo$ новых элементов в несколько тек сразу. Для любого конечного списка тек T_1, ... T_n $(n \geq 1)$ и произвольного конечного множества элементов $\{x_1, \ ... \ x_m\} \subseteq D$ $(m \geq 0)$, пусть результатом выполнения $AddTo(\{x_1, \ldots x_m\}, T_1, \ldots T_n)$ в момент времени t (т. е. в текущий момент времени, когда показания глобальных часов есть t) будут n тек T'_1, ... T'_n таких, что существуют m моментов времени (т. е. показаний глобальных часов) $t = t_1 < ... < t_m = t'$ для которых t' — это время завершения этой операции, а для каждого $1 \leq i \leq n$ тека T'_i есть $T'_i = \ T_i \cup \ \{(x_1, t_1), \ ... \ (x_m, t_m)\}$. Заметим, что эта операция является недетерминированной, т. к. множество $\{x_1, \ ... \ x_m\}$ может быть отсортировано $m!$ различными способами и, кроме того, моменты времени $t_1 < ... < t_m$ могут быть произвольными. В случае необходимости мы будем писать $AddTo(x, T_1, \ldots T_n)$ вместо $AddTo(\{x\}, T_1, \ldots T_n)$.

[2]например, библиотека, фильмотека и т. д.

Существует три пары операций, чувствительных ко времени: Fir и $ReMFir$, Las и $RemLas$, Elm и $RemElm$.

- Пусть $Fir(T)$ (от first)— это значение (типа *type*), которое входит в T с наименьшей меткой времени (т. е. хронологически первое), а $RemFir(T)$ — это результат удаления этого элемента (вместе с его минимальной меткой времени) из T.

- Пусть $Las(T)$ (от last) — это значение (типа *type*), которое входит в T с наибольшей меткой времени (т.е. хронологически последнее), а $RemLas(T)$ — это результат удаления этого элемента (вместе с его максимальной меткой времени) из T.

Результат операции $Elm(T)$ — это *некоторый* элемент теки T (без временного штампа), выбранный согласно некоторой предопределённой процедуре, а $RemElm(T)$ — результат удаления этого же элемента (вместе со своим штампом времени) из T.

3. Унифицированный шаблон

Введём некоторые вспомогательные обозначения и понятия для удобства представления и унификации алгоритмов (спроектированных по методу BTR или B&B) в виде единого унифицированного шаблона.

Во-первых, пусть FEL и REM обозначают или пару операций Fir и $ReMFir$, или пару операций Las и $RemLas$, или пару операций Elm и $RemElm$. Это означает, например, что если в унифицированном шаблоне в одном месте мы подставили Fir вместо FEL, то и всюду в этом шаблоне мы должны подставить Fir вместо FEL, а $RemFir$ — вместо REM. Подстановка Fir и $RemFir$ приводит к дисциплине FIFO (first-in, first-out), "превращает" теку в *очередь* и трансформирует унифицированный шаблон в шаблон метода B&B; подстановка же Las и $RemLas$ приводит к дисциплине FILO (first-in, last-out), "превращает" теку в *стек* и трансформирует унифицированный шаблон в шаблон метода BTR. Подстановка Elm и $RemElm$ "превращает" теку в очередь или стек *с приоритетами* и трансформирует унифицированный шаблон в шаблон метода глубокого отката или ветвей и границ с приоритетами.

Во-вторых, будем говорить что граф *виртуальный*, если он задан следующим образом:

- определён тип его вершин $Node$ и выделена его *начальная* вершина ini (типа $Node$) такая, что любая вершина этого графа достижима из ini;

- имеется операция $Neighb : \ Node \rightarrow 2^{Node}$, которая для каждой вершины вычисляет (возвращает) множество всех её соседей.

Таким образом виртуальный граф задан как бы неявно, не списком своих вершин и рёбер или матрицей смежности и т. д.

Теперь мы готовы представить унифицированный шаблон для перечисления виртуального графа G с помощью "легко вычислимых"

- *граничного условия* $B: 2^{Node} \times Node \to BOOLEAN$,
- *критерия* $D: 2^{Node} \times Node \to BOOLEAN$

с целью собрать все вершины этого графа, которые удовлетворяют "трудно вычислимому"

- *целевому условию* $C: Node \to BOOLEAN$.

VAR U: Node;
VAR S: set of Node;
VAR Visit, Live, Out: theque of Node;
Live, Visit:= AddTo(ini, empteq, empteq);
Out:= empteq;
IF D({ini}, ini) THEN Out:= AddTo(ini, Out);
WHILE Live ≠ empteq
DO U:= FEL(Live); Live:= REM(Live);
S:= {W ∈ Neighb(U) : Ni(W, Visit) ∧ ¬B(Set(Visit), W)};
Live, Visit:= AddTo(S, Live, Visit);
Out:= Spec(Out, λx.D(Set(Visit), x));
IF D(Set(Visit), U) THEN Out:= AddTo(U,Out);
OD

В качестве простого примера использования этого унифицированного шаблона рассмотрим решение методом отката задачи о расстановке $n \geq 4$ ферзей на шахматной доске размером $n \times n$ так, чтобы они не били друг друга (т. е. не стояли на одной линии, одном ряду, одной диагонали).

Для этого в качестве виртуального графа G примем дерево *частичных расстановок* (p-placements). Каждая частичная расстановка — это безопасная расстановка k $(0 \leq k \leq n)$ ферзей на первых k рядах. Пусть начальная вершина ini — это пустая расстановка *em-placement* (т. е. расстановка 0 ферзей в 0 первых рядах). Пусть функция $Neighb$ "вычисляет" по заданной частичной расстановке k ферзей на первых k рядах $(0 \leq k < n)$ все возможные n вариантов постановки еще одного ферзя на очередной ряд $(k+1)$.

Граничное условие, критерий и целевая функция B, D и C достаточно очевидны и естественны: для каждой частичной расстановки x из k ферзей на первых k рядах $(0 \leq k \leq n)$ пусть

- $B(x)$ означает "x небезопасно";
- $D(x)$ означает "x есть полная расстановка" (т. е. $k = n$);
- $C(x)$ означает "x есть полная и безопасная расстановка".

Тогда в результате специализации унифицированного шаблона получается следующий алгоритм:
VAR U: p-placement;
VAR S: set of p-placements;
VAR Live, Out: theque of p-placements;

Live:= AddTo(em-placement, empteq);
Out:= empteq;
WHILE Live $\neq$ *empteq*
DO U:= Las(Live); Live:= RemLas(Live);
S:= $\{$*W extends U : W is safe*$\}$*;*
Live:= AddTo(S, Live);
IF U is complete THEN Out:= AddTo(U,Out);
OD

4. Пример: дискретная задача о ранце

Рассмотрим более интересный пример использования унифицированного шаблона для решения методом ветвей и границ дискретной задачи о ранце [4].

4.1. Постановка задачи

Пусть нам дано $n \geq 0$ неделимых товаров, вещественные числа $p_1 \geq 0$, ... $p_n \geq 0$ и $w_1 \geq 0$, ... $w_n \geq 0$ — цены и вес этих товаров, а $W \geq 0$ — максимальный вес, который можно уложить в ранец. Допустимый набор товаров — это любой набор товаров, суммарный вес которых не превосходит W. Задача состоит в том, что надо вычислить все допустимые наборы товаров, суммарная цена каждого из этих наборов — максимально возможная среди всех допустимых наборов.

Давайте отождествим наборы товаров с их характеристическими векторами, т. е. векторами $(c_1, \ldots c_n) \in \{0,1\}^n$. Пусть L и P обозначают следующие функции веса (load) и цены (price): для любого набора $c_1, \ldots c_n \in \{0,1\}$ пусть $L(c_1, \ldots c_n) = \sum_{1 \leq k \leq n} c_k \times w_k$ и $P(c_1, \ldots c_n) = \sum_{1 \leq k \leq n} c_k \times p_k$. Тогда дискретная задача о ранце может быть формализована следующим образом: требуется найти все $(c_1, \ldots c_n) = \arg\max\{P(c_1, \ldots c_n) : L(c_1, \ldots c_n) \leq W\}$.

4.2. Применение метода ветвей и границ

Пусть наш виртуальный граф G — это (лексикографически) упорядоченное полное бинарное дерево T_n *частичных наборов* (p-collections) $(c_1, \ldots c_m) \in \{0,1\}^m$, где $0 \leq m \leq n$. Примем пустой частичный набор (корень T_n) в качестве начальной вершины ini. Функция $Neighb$ определяется естественным образом: для каждого частичного набора $c = (c_1, \ldots c_m)$, $(0 \leq m < n)$ пусть $Neighb$ возвращает два частичных набора $c \circ 0 = (c_1, \ldots c_m, 0)$ и $c \circ 1 = (c_1, \ldots c_m, 1)$.

Для удобства введём вспомогательные понятия и обозначения. Длину частичного набора $(c_1, \ldots\ c_m)$ определим как $|c_1, \ldots c_m| = m$. Разумеется, что длина пустого частичного набора 0, а частичные наборы длины n — это в точности все наборы. Функции L и P можно естественным образом распространить на частичные наборы: для любого частичного набора $(c_1, \ldots c_m)$, $0 \leq m \leq n$ пусть $L(c_1, \ldots c_m) = \sum_{1 \leq k \leq m} c_k \times w_k$ и $P(c_1, \ldots c_m) = \sum_{1 \leq k \leq m} c_k \times p_k$. Для любого множества частичных наборов Q пусть $ext(Q)$ — это множество всех *допустимых пополнений* векторов из

Q, т. е. множество $\{z \in T_n \ : \ L(z) \leq W$ и некоторый $x \in Q$ является префиксом $z\}$; будем писать $ext(x)$ в случае, когда Q есть одноэлементное множество $\{x\}$.

Определим граничное условие B как функцию двух аргументов: в качестве первого аргумента может стоять произвольное множество X частичных наборов, а в качестве второго — произвольный частичный набор y; для такой пары аргументов значение $B(X, y)$ — это

$$\begin{gathered}\text{некоторым фиксированным способом оцененная нижняя граница} \\ \max\{P(x) : x \in ext(X)\} \ > \\ > \text{некоторым фиксированным способом оцененная верхняя граница} \\ \max\{P(x) : x \in ext(y)\},\end{gathered}$$

где

- способ оценки нижней границы монотонно зависит от X (т. е. значение не может убывать при разрастании X как множества);
- способ оценки верхней границы антимонотонно зависит от y (т. е. значение не может возрастать при увеличении y в соответствии с лексикографическим порядком).

Примерами таких способов может служить следующая пара функций:

- $\max\{P(x) : x \in X \text{ and } L(x) \leq W\}$ — способ определения нижней границы для $\max\{P(x) : x \in ext(X)\}$,
- $(P(y) + \sum_{|y|<k\leq n} p_k)$ — способ определения верхней границы для $\max\{P(x) : x \in ext(y)\}$.

Множество других подходящих примеров можно найти в [4].

Определим критерий D как функцию таких же как и B двух аргументов, значение которой равно конъюнкции следующих двух условий:

- $P(y) = \max\{P(x) : x \in X$ и $L(x) \leq W\}$;
- y является набором (т. е. $|y| = n$) и $L(y) \leq W$.

Целевая функция $C(x)$ определяется совершенно естественным образом: x является набором (т. е. $|x| = n$), $L(x) \leq W$ и $P(x) = \max\{P(y) : |y| = n$ и $L(y) \leq W\}$.

Тогда унифицированный шаблон для решения дискретной задачи о ранце методом ветвей и границ выглядит следующим образом:

VAR U: p-collection;
VAR S : set of p-collections;
VAR Visit, Live, Out : theque of p-collections;
Live, Visit:= AddTo(empty p-collection, empteq, empteq);
Out:= empteq;
WHILE Live $\neq$ *empteq*
DO U:= Fir(Live); Live:= RemFir(Live);
S:= $\{U \circ b : |U| < n,\ b \in \{0, 1\}$ *and*

lower bound for $\max\{P(x) : x \in ext(Set(Visit))\} \leq$
$\leq$ *upper bound for* $\max\{P(x) : x \in ext(U \circ b)\}$;
Live, Visit:= AddTo(S, Live, Visit);
IF $(|U| = n$, $L(U) \leq W$ *and* $P(U) > \max\{P(x) : x \in Set(Out)\})$
THEN Out:= AddTo(U, empteq);
IF $(|U| = n$, $L(U) \leq W$ *and* $P(U) = \max\{P(x) : x \in Set(Out)\})$
THEN Out:= AddTo(U,Out);
OD

5. Спецификация и корректность

Алгоритм (или программа) без спецификации (того, что он/она делает) — это инструмент без инструкции по использованию: он может быть использован неправильно. Специфицированный алгоритм (специфицированная программа) без доказательства корректности (т. е. соответствия спецификации) — это несертифицированный инструмент: он может быть потенциально опасным. Следовательно, нам необходимо специфицировать и доказать корректность нашего унифицированного шаблона — и тем самым специфицировать и доказать корректность сразу всех возможных частных алгоритмов, которые можно получить по этому шаблону. Мы будем использовать для этого подход Флойда — Хоара [6, 7, 2]. Этот подход предполагает спецификацию алгоритма посредством предусловия на входные данные, постусловия на выходные данные и индукционное доказательство корректности с использованием инвариантов циклов.

Постусловие достаточно естественно и очевидно: тека Out состоит из тех и только тех вершин графа G (с метками времени, разумеется), которые удовлетворяют целевому условию C, причём каждая такая вершина имеет единственное вхождение в Out.

Предусловие значительно сложнее и может быть представлено как конъюнкция следующих условий.

1. G — виртуальный граф с начальной вершиной ini и функцией вычисления соседних вершин $Neighb$.

2. Для каждой вершины x из G граничное условие $\lambda S.B(S,x)$ является монотонной функцией: $B(S_1,x) \Rightarrow B(S_2,x)$ для любых множеств вершин $S_1 \subseteq S_2$ (т. е. если какой-либо узел отвергнут при одном множестве обойдённых вершин, то он отвергнут “навсегда”).

3. Для любых вершин x и y из G, для любого множества вершин S, если y достижимо из x без посещения вершин из S, тогда $B(S,x)$ влечёт $B(S,y)$ (т. е. если какая-либо вершина отвергнута, то и все её наследники отвергнуты).

4. Для каждой вершины x из G, критерий $\lambda S.D(S,x)$ является антимонотонной функцией: $D(S_2,x) \Rightarrow D(S_1,x)$ для любых множеств вершин $S_1 \subseteq S_2$ (т. е. вершина-кандидат может быть отвергнута после обхода бóльшего множества вершин).

5. Для любого множества вершин S, если множество $S \cup \{x \in G : B(S, x)\}$ совпадает с множеством всех вершин графа G, то $D(S, x) \Leftrightarrow C(x)$ (т. е. критерий D, применённый к множеству, которое можно "полнить" до множества всех вершин, эквивалентен целевому условию C).

Утверждение 1. *Унифицированный шаблон частично корректен по отношению к приведённым выше предусловию и постусловию, т. е. для любого алгоритма, который получается в результате специализации унифицированного шаблона, и любых значений входных данных этого алгоритма, если эти значения удовлетворяют предусловию и алгоритм завершает работу на этих входных данных, то значения выходных данных удовлетворяют постусловию.*

Доказательство. В соответствии с методом Флойда доказательства частичной корректности императивных алгоритмов [8, 2, 7, 6] представим шаблон в виде блок-схемы (Рис. 1).

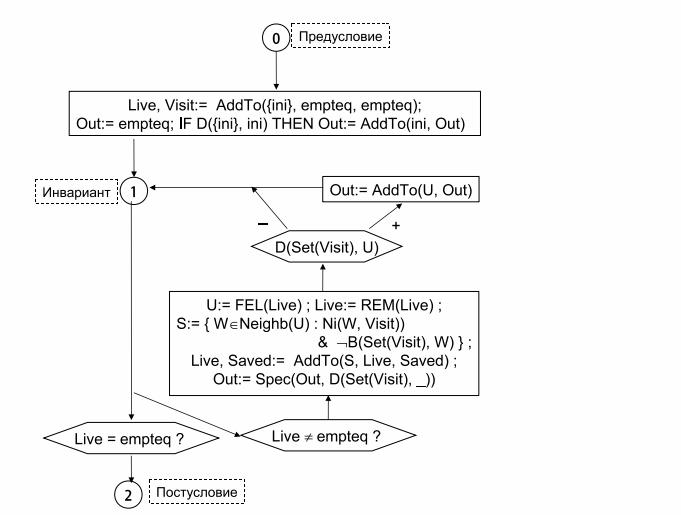

Рис. 1. The flowchart of the unified template

Легко видеть, что эта блок-схема имеет единственный цикл (соответствующий циклу *while*), следовательно, нам достаточно аннотировать начальную (0), заключительную (2) и какую-нибудь одну *контрольную* точки внутри этого цикла. Выберем в качестве контрольной точки цикла точку (1) входа в цикл. Аннотации точек (0) и (2) — это предусловие и постусловие, а в качестве аннотации (инварианта) точки (1) примем конъюнкцию предусловия и следующих условий.

1. Условие на Out и $Visit$: $Out = Spec(Visit,\ \lambda x.D(Set(Visit),\ x))$.

2. Условие на $Visit$ и $Live$: $Live \subseteq Visit$ и для каждой вершины $z \in G$, если $Ni(z, Visit)$ и $C(z)$, то z достижима из $Set(Live)$ без посещения вершин из $Visit$.

3. Условие на $Visit$: каждая вершина $x \in G$ имеет не более одного вхождения $Visit$, а множество $Set(Visit) \cup Neighb(Set(Visit))$ совпадает с множеством

всех вершин G, которые были порождены во время выполнения алгоритма вплоть до текущего момента времени.

Остаётся показать, что любой ациклический путь по блок-схеме между любой парой контрольных точек, не содержащий внутри других контрольных точек, является "консервативным" по отношению к аннотациям, т. е.

- если предусловие верно для каких-либо значений переменных перед исполнением простого пути от точки (0) к точке (1), то инвариант верен для достигнутых значений переменных в конце этого пути (при условии завершения вычислений);
- если инвариант верен для каких-либо значений переменных перед исполнением простого пути от точки (1) к точке (1), то инвариант верен для достигнутых значений переменных в конце этого пути (при условии завершения вычислений);
- если инвариант верен для каких-либо значений переменных перед исполнением простого пути от точки (1) к точке (2), то постусловие верно для достигнутых значений переменных в конце этого пути (при условии завершения вычислений).

Доказательство всех этих пунктов очевидно. ■

Утверждение 2. *Унифицированный шаблон тотален на конечных графах при выполнении приведённого выше предусловия, т. е. для любого алгоритма, который получается в результате специализации унифицированного шаблона, и любых значений входных данных этого алгоритма, если эти значения удовлетворяют предусловию и входной виртуальный граф конечен, то этот алгоритм обязательно завершает свою работу на этих входных данных.*

Доказательство. Пусть N — число вершин G, а `Checked` — новая переменная типа *set of Node*. Добавим следующие два присваивания в унифицированный шаблон: "`Checked := ∅`" прямо перед входом в цикл, а "`Checked := Checked ∪ {U}`" — в тело цикла. Такая модификация не изменит поведение алгоритма (полученного специализацией из унифицированного шаблона), т. к. новая переменная не участвует ни в одном условии.

Добавим следующее условие:

- $Checked \subseteq Set(Visit)$, $Checked \cap Set(Live) = \emptyset$,
  $Set(Live) \subseteq Set(Visit)$, $Set(Visit)$ — множество вершин G

в инвариант точки (1) из доказательства предыдущего утверждения. Легко видеть, что

- если предусловие верно для каких-либо значений переменных перед исполнением простого пути от точки (0) к точке (1), то модифицированный инвариант верен для достигнутых значений переменных в конце этого пути (при условии завершения вычислений);

- если модифицированный инвариант верен для каких-либо значений переменных перед исполнением простого пути от точки (1) к точке (1), то модифицированный инвариант верен для достигнутых значений переменных в конце этого пути (при условии завершения вычислений).

Следовательно, значение выражения $(N - |Checked|)$ уменьшается после всякой "легальной" (т. е. при верном условии цикла) итерации цикла. Согласно методу потенциалов Флойда [8, 2, 7, 6], любой алгоритм, полученный специализацией из унифицированного шаблона, обязательно завершает свою работу (после не более чем N итераций цикла). ■

Из этих утверждений тривиально следует

Теорема 1.
Если конкретные граничное условие, критерий и целевая функция B, D и C удовлетворяют описанному выше предусловию, а виртуальный граф G конечен, то алгоритм, полученный из унифицированного шаблона в результате данной специализации завершает свою работу, выполнив не более $O(|G|)$ итераций цикла таким образом, что по его завершении множество $Set(Out)$ состоит из тех и только тех вершин графа G, которые удовлетворяют C.

Приведём пример использования этой теоремы. Как видно из описаний граничного условия, критерия и целевой функции, выбранных нами в части 4 для метода ветвей и границ для решения дискретной задачи о ранце, эти функции удовлетворяют предусловию унифицированного шаблона. Следовательно, в соответствии с теоремой 1, наш алгоритм, полученный в результате специализации унифицированного шаблона, тотально корректен по отношению к следующим предусловию и постусловию:
Предусловие:

- нижняя оценка монотонна по X;
- верхняя оценка антимонотонна по y.

Постусловие: $Set(Out)$ состоит из всех $\arg\max\{P(c_1, \dots c_n) : L(c_1, \dots c_n) \leq W\}$.

6. Заключение

В этой статье описан, аннотирован и верифицирован унифицированный шаблон проектирования алгоритмов перечисления вершин графа методом отката и методом ветвей и границ. Мы также привели примеры специализации этого шаблона для решения задачи о расстановке ферзей и дискретной задачи о ранце. Однако в этой статье не предлагается знакомить учащихся старших классах школы и студентов младших курсов с этими методами посредством предложенного шаблона и его спецификации, разумеется, что начинающих надо знакомить с методами проектирования алгоритмов на примерах; однако, мы считаем, что на старших курсах изучение шаблона, его спецификации и доказательства корректности полезно, т. к. позволяет единообразно представить большое семейство конкретных алгоритмов и

получать конкретные корректные алгоритмы путём специализации унифицированного шаблона.

Может показаться, что подход, принятый в этой статье, напоминает парадигму Э. Дейкстры конструирования алгоритмов одновременно со спецификацией и доказательством [3]. Однако разница состоит в том, что автор шёл не от спецификаций, а от конкретных алгоритмов, реализующих методы отката или ветвей и границ, пытаясь выявить то общее, что поддаётся обобщению в виде унифицированного шаблона, формализуя в виде спецификаций желаемые свойства граничного условия, критерия и целевой функции, и только потом переходя к доказательству корректности специфицированного таким образом унифицированного шаблона.

В то же время мы предвидим несколько направлений для исследований. В частности, возникает вопрос о возможности формализации, спецификации и верификации шаблонов проектирования алгоритмов другими методами, например методом динамического программирования. После этого возможна постановка задачи разработки автоматизированной системы проектирования алгоритмов на основе шаблонов для методов отката, ветвей и границ, динамического программирования и других методов. Такое направление исследований нельзя считать абсолютно новым, т. к. в период с 1980 по 2000 г. несколько исследовательских групп ставили задачу разработки подобной системы из *неформализованных и неверифицированных шаблонов* на основе алгебраических спецификаций [12] и с использованием методов искусственного интеллекта [11]. Однако эти проекты не увенчались успехом (о чём говорит отсутствие дальнейших публикаций), использование неформализованных и неверифицированных шаблонов при проектировании алгоритмов приводило потом к необходимости полномасштабной верификации получающихся алгоритмов, в то время как в нашем случае речь будет идти только о специализации уже готового доказательства корректности.

Список литературы

1. Ахо А., Хопкрофт Дж., Ульман Дж. *Построение и анализ вычислительных алгоритмов.* М.: Мир, 1979.

2. Грис Д. *Наука программирования.* М.: Мир, 1984.

3. Дейкстра В.Э. *Дисциплина программирования.* М.: Мир, 1978.

4. Корнмэн Т, Лейзерсон Ч, Ривест Р. *Алгоритмы: построение и анализ.* М.: МЦНМО, 2000.

5. Непомнящий В.А. *Верификация финитных итераций над наборами изменяемых структур данных.* Кибернетика и системный анализ. Киев. 2007. №3. С.33–46.

6. Шилов Н.В. *Основы синтаксиса, семантики, трансляции и верификации программ.* Новосибирск: Новосибирский государственный университет, 2011.

7. Apt K.R., de Boer F.S., Olderog E.-R. *Verification of Sequential and Concurrent Programs.* Third edition. Springer, 2009.

8. Floyd R.W. *Assigning meanings to programs.* Proc. of a Symposium in Applied Mathematics. Mathematical Aspects of Computer Science // American Math. Society, Providence, R. I. 1967. Vol. 19. P. 19–32.

9. Golomb S.W. and Baumert L.D. *Backtrack Programming* // Journal of ACM. 1965. 12(4). P. 516–524.

10. Land A. H. and Doig A. G. *An automatic method of solving discrete programming problems* // Econometrica. 1960. 28(3). P. 497–520.

11. Kant E. *Understanding and Automating Algorthm Design* // IEEE Transactions on Software Engineering. 1985. SE-11(11). P. 1361–1374.

12. Smith D. R. *Automating the design of algorithms* // Formal Program Development. 1993. Lecture Notes in Computer Science. 755. P. 324–354.

Verification of Backtracking and Branch and Bound Design Templates

Shilov N.V.

Keywords: backtracking, branch-and-bound, abstract data type, partial correctness, total correctness, Floyd proof method, n-queen puzzle, knapsack problem

The Design and Analysis of Computer Algorithms is *a must* of Computer Curricula. It covers many topics that group around several core themes. These themes range from data structures to the complexity theory, but one very special theme is algorithmic design patterns, including greedy method, divide-and-conquer, dynamic programming, backtracking and branch-and-bound. Naturally, all the listed design patterns are taught, learned and comprehended by examples. But they can be semi-formalized as design templates, semi-specified by correctness conditions, and semi-formally verified by means of Floyd method. Moreover, this approach can lead to new insights and better comprehension of the design patterns, specification and verification methods. In this paper we demonstrate an utility of the approach by study of backtracking and branch-and-bound design patterns. In particular, we prove correctness of the suggested templates when the boundary condition is monotone, but the decision condition is anti-monotone on sets of "visited" vertices.

Сведения об авторе:
Шилов Николай Вячеславович,
кандидат физико-математических наук, старший научный сотрудник
Института систем информатики им. А.П. Ершова СО РАН,
доцент Новосибирского государственного университета
и Новосибирского государственного технического университета.