# Контролёры алгоритмов в Arageli
# РАБОЧИЙ ВАРИАНТ

С. С. Лялин

3 августа 2006 г.

Типичный алгоритм, реализованный в библиотеке, оформлен в виде функции, которая принимает некоторые входные значения и передаёт результат своей работы через аргументы-ссылки или возвращаемое значение. Для некоторых алгоритмов требуется организовать более жёсткий контроль. Например, если функция может выполняться достаточно долгое время, то было бы неплохо иметь механизм, который позволит аварийно завершить работу и выйти из функции. К тому же, в процессе работы процедуры может понадобиться сделать вывод промежуточных результатов или узнать о степени завершённости выполняемой работы. Это актуально, например, для учебных интерактивных сред или при отладке.

Такой механизм в библиотеке Arageli имеется. Это специальным образом оформленные объекты-контролёры, которые передаются в "сложную" функцию как дополнительный параметр. Требования к интерфейсу для типа этого объекта полностью определяются функцией; тип должен поддерживать все методы, которые вызывает для него целевая функция.

Например, функция `rref` (приведение матрицы к ступенчатому виду) из `‘‘gauss.hpp’’` принимает последним аргументом такой объект. Она ожидает присутствия метода `preamble`, которому она передаёт в качестве параметра входную матрицу; в начале и конце каждой гауссовой итерации `rref` вызывает для этого объекта методы `before_iter` и `after_iter` соответственно, которым передаёт текущую расширенную матрицу, базис, значение детерминанта; и т.д — есть ещё ряд методов, которые вызываются в различных ключевых местах алгоритма.

Что делает объект контролёра, когда вызываются эти его методы, зависит от создателя контролёра и может быть весьма разнообразным. Например, он может формировать вывод в виде исходного текста документа для LaTeX, который представляет собой подробное пошаговое

решение задачи. Или на каком-то этапе будет принято решение досрочно остановить выполнение функции (без завершения вычисления всех выходных параметров), тогда контролёр в одном из своих методов может сгенерировать специальное исключение, которое укажет функции на необходимость аварийного выхода (контролёр может управляться интерактивным интерфейсом пользователя в параллельном потоке).

Т.е. контролёр функции — это механизм передачи дополнительной информации в/из исполняемой функции, *в процессе её работы*. Тип контролёра всегда задаётся как параметр шаблона, и может быть сконструирован, например, таким образом, что ничего не будет делать по сигналам, поступающим из функции, и никак не будет влиять на исполнение. Для контролёра функции `rref` это означает, что все методы будут с пустым телом. *Нормальный* компилятор не оставит в получаемом бинарном коде и следа вызова таких методов. Таким образом, данный механизм не лишает возможности вызвать целевую функцию как раньше, без дополнительных затрат на контроль, если важна в первую очередь эффективность получаемой программы.

Функцию, сконструированную так, что она принимает контролёра, будем называть *контролируемой функцией*.

Проиллюстрируем сказанное на примере уже рассмотренной функции `rref`. Следующий код просто находит ступенчатую форму матрицы; в конце выводится лишь окончательный результат. [1]

```
template <typename T, bool REFCNT>
void print_rref (const matrix<T, REFCNT>& a)
{
    matrix<T, false> b, q;
    vector<size_t, false> basis;
    T det;

    rref(a, b, q, basis, det);

    output_aligned(cout, b);
}
```

[1] Здесь и везде далее в отрывках кода предполагается, что подключены все необходимые заголовки и сделаны видимыми все используемые имена, т.е. в начале кода есть, как минимум, следующее:

```
#include <iostream>
#include <arageli/arageli.hpp>
using namespace std;
using namespace Arageli;
```

По умолчанию, `rref`, как контролируемая функция, молчалива. Это объясняется тем, что по умолнчанию она вызывается с аргументом-контролёром `Arageli::ctrl::rref_idler`, который ничего не делает, т.е. вызов `rref` в примере эквивалентен

```
rref(a, b, q, basis, det, ctrl::rref_idler());
```

Если мы вызовем `print_rref` так

```
print_rref(matrix<rational<> >
    ("((1, 2, 3), (4, 5, 6), (7, 8, 9))"));
```

то получим вывод

```
|| 1 0 -1 ||
|| 0 1 2  ||
|| 0 0 0  ||
```

Подставив в качестве последнего аргумента `rref` контролёр `Arageli::ctrl::rref_slog` можно получить все промежуточные результаты вычислений. Заменим

```
rref(a, b, q, basis, det);
```

в нашем примере на

```
rref(a, b, q, basis, det, ctrl::rref_slog<ostream>(cout));
```

тогда после исполнения `print_rref` для той же матрицы получим такой вывод в стандартный поток вывода [2]:

```
Producing of reduced row echelon form for matrix
|| 1 2 3 ||
|| 4 5 6 ||
|| 7 8 9 ||
Lets write additionaly an unitary matrix and produce gauss
iterations.
|| 1 2 3 | 1 0 0 ||
|| 4 5 6 | 0 1 0 ||
```

[2]Заметим, что “inverse of input matrix” в выводе — это не обратная матрица, конечно, а та матрица, на которую нужно домножить оригинальную матрицу слева, что бы получить результат (за более подробными разъяснениями обращайтесь к документации по `rref`). Название (“обратная матрица”) объясняется тем, что для квадратной невырожденной матрицы в том месте будет обратная для неё.

```
|| 7 8 9 | 0 0 1 ||
Basis is ()
Determinant is 1
Find the biggest (in absolute value) entry in a column 1
Pivot item is (3, 1)
Swap rows 1, 3
|| 7 8 9 | 0 0 1 ||
|| 4 5 6 | 0 1 0 ||
|| 1 2 3 | 1 0 0 ||
Determinant is -1
Eliminate in column 1
|| 1 8/7 9/7  | 0 0 1/7  ||
|| 0 3/7 6/7  | 0 1 -4/7 ||
|| 0 6/7 12/7 | 1 0 -1/7 ||
Basis is (1)
Determinant is -7
Find the biggest (in absolute value) entry in a column 2
Pivot item is (3, 2)
Swap rows 2, 3
|| 1 8/7 9/7  | 0 0 1/7  ||
|| 0 6/7 12/7 | 1 0 -1/7 ||
|| 0 3/7 6/7  | 0 1 -4/7 ||
Determinant is 7
Eliminate in column 2
|| 1 0 -1 | -4/3 0 1/3  ||
|| 0 1 2  | 7/6  0 -1/6 ||
|| 0 0 0  | -1/2 1 -1/2 ||
Basis is (1, 2)
Determinant is 6
Find the biggest (in absolute value) entry in a column 3
Column 3 is negligible
|| 1 0 -1 | -4/3 0 1/3  ||
|| 0 1 2  | 7/6  0 -1/6 ||
|| 0 0 0  | -1/2 1 -1/2 ||
The task has been solved.  Found reduced row echelon form is
|| 1 0 -1 ||
|| 0 1 2  ||
|| 0 0 0  ||
Inverse of input matrix is
|| -4/3 0 1/3  ||
|| 7/6  0 -1/6 ||
```

```
|| -1/2 1 -1/2 ||
Basis is (1, 2)
Determinant is 0
|| 1 0 -1 ||
|| 0 1 2  ||
|| 0 0 0  ||
```

В конце дописан результат, который выводится, как и раньше, самой `print_rref`. Подставив `ctrl::rref_latexlog` вместо `ctrl::rref_slog` мы можем получить вывод LaTeX. Наконец, можно сконструировать свой котролёр, который сделает ещё что-нибудь полезное.

Для любой контролируемой функции рекомендуется иметь как минимум две её перегруженные версии: с явно указываемым объектом контролёра и без оного. Вторая версия просто вызывает первую с контролёром по умолчанию, который ничего не делает (idler-контролёр). Т.е. для контролируемой шаблонной функции `f` у нас есть следующие определения:

```
template <параметры-шаблона-без-контроля, typename Ctrler>
тип-возвращаемого-значения f
(
    параметры-функции-без-контроля,
    Ctrler ctrler // контролёр
)
{ что-то-с-вызовами-методов-контролёра }

template <параметры-шаблона-без-контроля>
inline тип-возвращаемого-значения f
(
    параметры-функции-без-контроля
)
{ f(передача-параметров-функции-без-контроля, ctrl::f_idler()); }
```

Аргументы-контролёры рекомендуется размещать в конце основного списка аргументов и придерживаться продемонстрированной здесь системе именования.

Все контролёры определяются в `namespace Arageli::ctrl` в подпространстве, имя которого соответсвует имени подпространства, в котором определена контролируемая функция. Например, если функция определена в `Arageli::simplex_method`, то контролёры для неё будут реализовываться в `Arageli::ctrl::simplex_method`. (В этом случае, возможно, будет удобным сделать отображение определений контролёров в `Arageli::simplex_method::ctrl` из `Arageli::ctrl::simplex_method`.)

Контролёры, которые ничего не далают имеют суффикс `_idler` в имени, которые выводят в простой поток — `_slog`, оформляют в виде LaTeX— `_latexlog`.

Класс контролёра должен иметь вложенный класс с именем `abort`, наследник `Arageli::ctrl::abort`. Это тип исключения, которое может быть сгенерировано методами контролёра в качестве сообщения, что он прекращает работу вызванной контролируемой функции. При завершении работы какого-либо метода контролёра по этому исключению, функция должна завершить свою работу и перегенерировать исключение.

Основным критерием, по которому определяется, делать ли некоторую функцию контролируемой или неконтролируемой, является объём промежуточных результатов и предполагаемая продолжительность работы функции. Например, сложение, умножение векторов и матриц — это неконтролируемая операция, а приведение матрицы к нормальной диагональной форме Смита — контролируемая. При принятии решения нужно учитывать множество факторов в каждом конкретном случае.

К примеру, для основной контролируемой функции `simplex_method::primal_row_iters` (итерации прямого строчного симплекс-метода) контолёр может понадобится для выхода при зацикливании алгоритма симплекс-метода. Данная функция позволяет установить любые правила выбора текущего элемента в симплекс таблице, и некоторые из них могут привести к зацикливанию. Сохраняя список баз, контролёр может прервать исполнение, когда появится база уже рассматривавшаяся в ходе решения.

Справедливости ради нужно заметить, что на втором уровне отладки и выше (`ARAGELI_DEBUG_LEVEL > 2`) библиотека отлавливает зацикливание стандартной отладочной проверкой. Но, данное поведение не перекрывает возможности контролёра и срабатывает лишь тогда, когда контролёр не прервал исполнение в нужный момент. Т.е. вызывающей программе даётся шанс узнать о зацикливании без задействия механизма отладочных проверок библиотеки. Вызов симплекс-метода с таким правилом выбора ведущего элемента, который может привести к зацикливанию, очевидно, полезен в иллюстративных целях в образовательном процессе.

Опыт реализации контролёров показывает, что почти всегда код простых контролёров, которые выводят все промежуточные результаты в поток (с несложным оформлением), занимает больше места, чем сам алгоритм. Это вызвано с одной стороны громоздкостью шаблонных определений методов контролёра, а с другой — многословностью определений, делающих “красивый” вывод. Но, отсюда вовсе не следует сложность реализации этих методов.

Заметим, что объект контролёра всегда передаётся по значению. Поэтому создателю контролёра нужно заботится о том, что бы его можно было скопировать и что бы копирование не занимало много времени. Для контролёра, который имеет какие-то поля и обладает обратной связью с вызывающей стороной, типичным является офрмление его в виде промежуточного объекта, держащего ссылку на реальное представление.

Конечно, можно предложить альтернативу выбранному методу контроля алгоритмов. Если реализовать какой-нибудь алгоритм в виде, который уже предполагает некоторый вполне определённый и зашитый в коде вид контроля, то строчек кода будет заметно меньше, чем в нашем решении с разделением кода алгоритма и контролирующего кода. Не будет вызова функций контолёра, всё будет в одном месте: и код алгоритма и код контоля. Т.е. вместо создания одной реализации алгоритма и $n$ реализаций контролёров (по одному на каждый вид контроля), мы будем реализовывать $n$ алгоритмов с зашитым контролем, но избавимя от общих громоздких определений отдельных контролёров.

У подобной схемы есть один большой недостаток, — для ликвидации которого и создан механизм контроля алгоритмов в библиотеке, — каждый раз придётся переписывать один и тот же алгоритм, но по разному. Это повлечёт за собой большее количество ошибок, не говоря уже о том, что сопровождение такого кода превратится в кошмар. К тому же, выделение основных ортоганальных по отношению друг к другу концепций в независимые механизмы, представляется более элегантным.