Сравнение объектов в Arageli
ЧЕРНОВОЙ ВАРИАНТ

С. С. Лялин

3 августа 2006 г.

Здесь под сравнимостью подразумевается возможность установить порядок на некотором множестве объектов. То есть для любых двух объектов, типы которых могут участвовать во взаимном сравнении, однозначно определяется какой из них "меньше" или устанавливается, что они равны (эквивалентны). Будут рассмотрены принципы, по которым происходит сравнения различных объектов, и существующие функции, в том или ином виде предоставляющие сервис сравнения.

Заметим, что объекты не любой пары типов могут участвовать в сравнении. Правила, которые регулируют *возможность сравнения* пары объектов разных типов, не являются предметом данного описания и принадлежат общим принципам формирования смешанных выражений в Arageli. Здесь же рассматривается *механизм сравнения* объектов, если уже известно, что их можно сравнивать. Например, мы можем сравнивать полиномы с различными типами коэффициентов, матрицы с различными типами элементов, матрицы и скаляры и т. п., но не целочисленный вектор и полином над целыми числами и проч.

Длинные целые числа `big_int` и рациональные `rational`. Числа сравниваются так, как это принято в математике; поясним лишь некоторые технические детали. Если при сравнении двух объектов `big_int` вопросов не возникает, то при сравнении двух `rational` или `rational` и какого-либо другого числового типа нужно сделать несколько пояснений. Во-первых, два объекта типа `rational<`T_1`>` и `rational<`T_2`>` всегда сравниваются через сравнение с нулём разности между ними. В этом случае информация об одном из типов T_1 или T_2 теряется на этапе вычисления разности[1]. Это не влияет на конечный результат сравнения, если

[1]Описываемая последовательность действий и возможные эффекты, которые при этом могут происходить, верны для настоящей реализации библиотеки. В следующих версиях при сравнении не будет выполняться вычитание и потери информации об

вычитание происходит точно и результат точно представляет разность. Последнее может не выполняться, если T_1 и T_2 являются ограниченными типами и при вычитании происходит переполнение.

При сравнении рационального числа с *нерациональным* последнее приводится к рациональному одним из существующих способов преобразования, и сравниваются два рациональных числа, как это описано выше.

Векторы `vector`**.** Сравнение векторов — это лексикографическое сравнение[2]. Причём, лексикографическое сравнение расширено так, что можно сравнивать вектора разного размера (по сути это есть лексикографическое сравнение произвольных строк символов). Таким образом, этот способ работает для любых двух векторов `vector<`T_1`>` и `vector<`T_2`>`. Пусть v_1 типа `vector<`T_1`>` и v_2 типа `vector<`T_2`>`, s_1 и s_2 их размеры соответственно, тогда $v_1 < v_2$, если

1) при одновременном просмотре этих векторов с начала к концу первая пара различающихся элементов под номером i находится в отношении $v_1[i] < v_2[i]$ или,

2) если различающихся элементов не найдено (при просмотре закончился один из векторов), но $s_1 < s_2$.

В частности пустой вектор (нулевого размера, не содержащий элементов) меньше любого другого не пустого вектора и любые два пустых вектора (с любыми различными или одинаковыми типами элементов) равны между собой.

При сравнении вектора и скаляра, скаляр представляется как вектор того же размера из элементов равных данному скаляру. Далее происходит обычное сравнение двух векторов.

Заметим, что выключатель счётчика ссылок никак не влияет на результат сравнения и может быть любым (в том числе и различным в v_1 и в v_2), поэтому мы не ввели его в сигнатуру шаблона. Т. е. то, что мы его не указывали при записи `vector<`T`>` не означает, что он должен быть обязательно включён (`true`). Это верно для всех остальных структур в данном обзоре.

одном из типов не будет. Вообще, далее при описании точных операций, которые происходят при сравнении (преобразование из одного типа в другой и проч.) имеется ввиду лишь общий принцип, по которому идёт сравнение; реализация может быть более эффективной и не требовать некоторых затратных с точки зрения расходуемой памяти и времени выполнения операций.

[2]Так же реализованы несколько вариантов покомпонентного сравнения. Об этом см. ниже.

Матрицы `matrix`**.** Хотелось бы сделать сравнение матриц таким, чтобы можно было, как и для векторов, сравнивать матрицы разного размера. Предложенный метод сравнения сначала проверяет размеры матриц: если они различны, то результат определяется сравнением этих размеров, если размеры одинаковы, то матрицы рассматриваются как вектора и между ними происходит лексикографическое сравнение.

Более формально процедура сравнения выглядит так. Пусть M_1 имеет тип `matrix<`T_1`>`, состоит из m_1 строк и n_1 столбцов, а M_2 имеет тип `matrix<`T_2`>`, состоит из m_2 строк и n_2 столбцов. Тогда $M_1 < M_2$ если

1) $m_1 < m_2$, либо

2) $m_1 = m_2$ и $n_1 < n_2$, либо

3) $m_1 = m_2$, $n_1 = n_2$ и матрица M_1 развёрнутая по строкам лексикографически меньше чем так же представленная M_2.

Подобно векторам, пустая матрица с нулевыми размерами будет меньше любой не пустой матрицы. К тому же, как не трудно видеть, рассмотренный алгоритм согласован со сравнением векторов. То есть, если взять две матрицы в виде строк (столбцов) одинаковых размеров, то результат сравнения будет таким же, как если бы сравнивались соответствующие вектора. Если одна из матриц — непустая строка, а другая — непустой столбец, то первая всегда не больше второй.

При сравнении матрицы и скаляра, скаляр представляется как матрица тех же размеров из элементов равных данному скаляру. Далее происходит обычное сравнение двух матриц.

Полиномы `sparse_polynom`**.**[3] Для полиномов выбран такой способ сравнения, который полезен в различных математических контекстах. Как известно, полиномы хранятся в виде упорядоченной последовательности ненулевых мономов. Сейчас для мономов в полиноме нельзя указать различные способы сравнения; существует единственный способ — это сравнение для класса `monom<T>`. Оно определяется естественным образом: моном t представляется как упорядоченная пара $\langle deg(t), \ell(t)\rangle$[4] и такие пары сравниваются лексикографически.

Два объекта `sparse_polynom<T>` сравниваются лексикографически как цепочки мономов, из которых они состоят; начинается сравнение

[3]Наверное, для шаблона “плотных” полиномов `polynom` нужно сделать так же.

[4]В данном обзоре $\ell(P)$ обозначает старший коэффициент полинома (то, что возвращает функция `sparse_polynom::leading_coef_cpy`); если $P = 0$, то $\ell(P) = 0$. Для монома — это просто его коэффициент.

со старших членов (т.е. с максимальных, в смысле реализованного сравнения, мономов). Таким образом нулевой моном и нулевой полином являются минимальными элементами в множествах всех мономов и полиномов соответственно. Результат сравнения для двух данных мономов такой же, как для двух полиномов, каждый из которых состоит из соответствующего монома.

При сравнении полинома (монома) и скаляра, скаляр представляется как полином (моном). Далее происходит обычное сравнение двух полиномов (мономов).

Для `sparse_polynom` мы не указывали тип степеней (второй параметр шаблона). Он не влияет на алгоритм сравнения (и может различаться для двух полиномов).

Функции сравнения. Библиотека предоставляет ряд взаимосвязанных функций, которые выполняют сравнение теми методами, которые описаны выше. Далее перечислены все функции.

`cmp(`a`, `b`)` Основная функция сравнения, сравнивает a, b и возвращает $+1$ если $a > b$, 0 если $a = b$ и -1 если $a < b$.

`sign(`a`)` Определяет знак объекта путём сравнения с нулём. Ноль получают путём вызова `null(`a`)` (т.е. создание нуля по a как по образцу). Возвращаемые значения: $+1$ если $a > 0$, 0 если $a = 0$ и -1 если $a < 0$.

`is_positive(`a`)` Определяет больше ли элемент нуля. Возвращает `true` если $a > 0$.

`is_negative(`a`)` Определяет меньше ли элемент нуля. Возвращает `true` если $a < 0$.

операторы сравнения `operator<(`a`, `b`)`, `operator>(`a`, `b`)`, `operator<=(`a`, `b`)`, `operator>=(`a`, `b`)` `operator==(`a`, `b`)`, `operator!=(`a`, `b`)`. Они имеют привычный смысл в программах на C++, реализуются через `cmp`.

Все эти функции связаны следующими соотношениями:

а) `cmp(`a`, `b`)`$= -$`cmp(`b`, `a`)`

б) `sign(`a`)`=`cmp(`a`, null(`a`))`

в) `is_positive(`a`)`=(`sign(`a`)`$= +1$)

г) `is_negative(`a`)`=(`sign(`a`)`$= -1$)

д) `operator<`$(a,\ b) = (\texttt{cmp}(a,\ b) < 0)$

е) `operator>`$(a,\ b) = (\texttt{cmp}(a,\ b) > 0)$

ж) `operator<=`$(a,\ b) = (\texttt{cmp}(a,\ b) \leq 0)$

з) `operator>=`$(a,\ b) = (\texttt{cmp}(a,\ b) \geq 0)$

и) `operator==`$(a,\ b) = (\texttt{cmp}(a,\ b) = 0)$

к) `operator!=`$(a,\ b) = (\texttt{cmp}(a,\ b) \neq 0)$

Покомпонентное сравнение векторов. Наряду с *лексикографическим* сравнением для векторов библиотека предоставляет *покомпонентное* сравнение (*each-сравнение*). Данный способ сравнения выделяется среди других сравнивающих функций и доступен только для векторов. При покомпонентном сравнении двух векторов их размеры должны совпадать. Имена этих функций следующие (связь с основными функциями сравнения можно понять по имени):

- `each_cmp`
- `each_sign`
- `each_is_positive`
- `each_is_negative`
- `each_less`
- `each_greater`
- `each_less_equal`
- `each_greater_equal`
- `each_equal`
- `each_not_equal`.

Результатом является вектор того же размера с элементами, тип которых зависит от конкретной функции сравнения: для `each_cmp` и `each_sign` это `int`, а для всех остальных — `bool`. Значением каждого элемента результирующего вектора является значение соответствующей функции, применённой к соответствующей паре элементов исходных векторов.

Если нужно узнать, удовлетворяют ли все элементы данного вектора некоторому условию или находятся ли все соответствующие пары

двух данных векторов в некотором отношении друг с другом, то следует воспользоваться одной из следующих специальных функций (*all-сравнения*):

- `all_cmp`,
- `all_sign`,
- `all_is_positive`,
- `all_is_negative`,
- `all_less`,
- `all_greater`,
- `all_less_equal`,
- `all_greater_equal`,
- `all_equal`,
- `all_not_equal`.

Данные функции принимают один или два векторных параметра в зависимости от того, сколько аргументов принимает оригинальная функция для скалярных параметров (для `all_sign`, `all_is_positive`, `all_is_negative` — один аргумент, для всех остальных — два). Тип возвращаемого значения у всех функций: `bool`. Они возвращают истинное значение, если для *всех* элементов данного вектора или всех пар двух данных векторов выполняется условие сравнения, и ложное значение в противном случае.