Глава 4
Линейные упорядочения

В настоящей главе описаны свойства бинарных отношений, такие как транзитивность и симметричность. В частности, мы вводим понятия полного и слабого линейных упорядочений, а также концепцию стабильности функций на основе линейного упорядочения: сохранения порядка, присутствующего в аргументах, для эквивалентных элементов. Мы обобщаем min и max до функций выбора порядка, таких как медиана трех элементов, и вводим способ управления сложностью их реализации путем приведения к ограниченным подзадачам.

4.1. Классификация отношений

Отношение — это предикат, принимающий два параметра одного и того же типа:

$$\begin{aligned} &\mathit{Relation}(\mathrm{Op}) \triangleq \\ &\qquad \mathit{HomogeneousPredicate}(\mathrm{Op}) \\ &\quad \wedge\ \mathsf{Arity}(\mathrm{Op}) = 2 \end{aligned}$$

Отношение *транзитивно*, если из того, что оно выполняется между a и b и между b и c, следует, что оно выполняется между a и c:

$$\begin{aligned} &\textbf{property}(\mathrm{R} : \mathit{Relation}) \\ &\mathsf{transitive} : \mathrm{R} \\ &\quad \mathrm{r} \mapsto (\forall \mathrm{a}, \mathrm{b}, \mathrm{c} \in \mathsf{Domain}(\mathrm{R}))\,(\mathrm{r}(\mathrm{a}, \mathrm{b}) \wedge \mathrm{r}(\mathrm{b}, \mathrm{c}) \Rightarrow \mathrm{r}(\mathrm{a}, \mathrm{c})) \end{aligned}$$

Примеры транзитивных отношений — равенство, равенство первых элементов пар, достижимость в орбите и делимость.

Отношение *строго*, если оно никогда не выполняется между элементом и им самим; отношение *рефлексивно*, если это всегда выполняется между элементом и им самим:

property(R : *Relation*)
strict : R
$r \mapsto (\forall a \in \mathsf{Domain}(R))\, \neg r(a, a)$

property(R : *Relation*)
reflexive : R
$r \mapsto (\forall a \in \mathsf{Domain}(R))\, r(a, a)$

Транзитивные отношения во всех предыдущих примерах рефлексивны; отношение "собственный делитель" является строгим.

Упражнение 4.1. Приведите пример отношения, которое не является ни строгим, ни рефлексивным.

Отношение *симметрично*, если из того, что оно выполняется в одном направлении, следует, что оно выполняется в другом; отношение *асимметрично*, если оно никогда не выполняется в обоих направлениях одновременно:

property(R : *Relation*)
symmetric : R
$r \mapsto (\forall a, b \in \mathsf{Domain}(R))\, (r(a, b) \Rightarrow r(b, a))$

property(R : *Relation*)
asymmetric : R
$r \mapsto (\forall a, b \in \mathsf{Domain}(R))\, (r(a, b) \Rightarrow \neg r(b, a))$

Пример симметричного транзитивного отношения — "брат"; пример асимметричного транзитивного отношения — "предок".

Упражнение 4.2. Приведите пример симметричного отношения, которое не является транзитивным.

Упражнение 4.3. Приведите пример симметричного отношения, которое не является рефлексивным.

Если дано отношение $r(a, b)$, то существуют *производные отношения* с тем же доменом:

$$\begin{aligned} \mathsf{complement}_r(a, b) &\Leftrightarrow \neg r(a, b) \\ \mathsf{converse}_r(a, b) &\Leftrightarrow r(b, a) \\ \mathsf{complement\_of\_converse}_r(a, b) &\Leftrightarrow \neg r(b, a) \end{aligned}$$

Что касается симметричного отношения, то единственным интересным отношением, которое может быть получено на его основе, является дополнение, поскольку обратное отношение эквивалентно исходному.

Отношение *эквивалентно*, если оно транзитивно, рефлексивно и симметрично:

property(R : *Relation*)
equivalence : R
$r \mapsto \mathsf{transitive}(r) \wedge \mathsf{reflexive}(r) \wedge \mathsf{symmetric}(r)$

Примеры отношений эквивалентности — равенство, геометрическая конгруэнтность и целочисленная конгруэнтность по модулю n.

Лемма 4.1. Если r — отношение эквивалентности, то $a = b \Rightarrow r(a, b)$.

Отношение эквивалентности разделяет свой домен на множество *классов эквивалентности*: подмножеств, содержащих все элементы, эквивалентные данному элементу. Мы часто можем реализовать отношение эквивалентности, определяя *ключевую функцию*, которая возвращает уникальное значение для всех элементов в каждом классе эквивалентности. Применение равенства к результатам ключевой функции определяет эквивалентность:

property(F : *UnaryFunction*, R : *Relation*)
requires(Domain(F) = Domain(R))
key_function : F × R
$(f, r) \mapsto (\forall a, b \in \mathsf{Domain}(F))\,(r(a, b) \Leftrightarrow f(a) = f(b))$

Лемма 4.2. $\mathsf{key\_function}(f, r) \Rightarrow \mathsf{equivalence}(r)$

4.2. Полные и слабые упорядочения

Отношение является *полным упорядочением*, если оно транзитивно и подчиняется *закону трихотомии*, согласно которому для каждой пары элементов выполняется одно и только одно из следующего: отношение, его обращение или равенство:

property(R : *Relation*)
total_ordering : R
$r \mapsto \mathsf{transitive}(r) \wedge$
$(\forall a, b \in \mathsf{Domain}(R))$ выполняется одно и только одно из следующего:
$r(a, b), r(b, a),$ или $a = b$

Отношение является *слабым упорядочением*, если оно транзитивно и существует отношение эквивалентности на том же домене, такое, что исходное отношение подчиняется *закону слабой трихотомии*, согласно которому для каждой пары элементов выполняется одно и только одно из следующего: отношение, его обращение или эквивалентность:

property(R : *Relation*, E : *Relation*) **requires**(Domain(R) = Domain(E))
weak_ordering : R
$r \mapsto \mathsf{transitive}(r) \wedge (\exists e \in E)\,\mathsf{equivalence}(e) \wedge$
$(\forall a, b \in \mathsf{Domain}(R))$ выполняется одно и только одно из следующего: $r(a, b), r(b, a)$ или $e(a, b)$

Если дано отношение r, то отношение $\neg r(a,b) \wedge \neg r(b,a)$ называется *симметричным дополнением* для r.

Лемма 4.3. Симметричное дополнение слабого упорядочения — это отношение эквивалентности.

Примеры слабого упорядочения — пары, упорядоченные по их первым элементам, и служащие, упорядоченные по зарплате.

Лемма 4.4. Полное упорядочение — слабое упорядочение.

Лемма 4.5. Слабое упорядочение асимметрично.

Лемма 4.6. Слабое упорядочение строго.

Ключевая функция f на множестве T, наряду с полным упорядочением r на кодомене f, определяет слабое упорядочение $\widetilde{r}(x,y) \Leftrightarrow r(f(x),f(y))$.

Мы называем полные и слабые упорядочения *линейными* упорядочениями, поскольку на них распространяются соответствующие законы трихотомии.

4.3. Выбор порядка

Если дано слабое упорядочение r и два объекта, a и b, из домена r, то приобретает смысл вопрос: какой из них является минимумом? Очевидно, как определить минимум, когда выполняется r или его обращение между a и b, но не столь очевидно, как это сделать, если элементы эквивалентны. Аналогичная проблема возникает, если вопрос состоит в том, какой из элементов является максимумом.

Свойство, позволяющее справиться с этой проблемой, известно как *стабильность*. Неформально алгоритм является *стабильным*, если он сохраняет исходный порядок эквивалентных объектов. Итак, если рассматривать минимум и максимум как выбор в списке из двух аргументов, соответственно наименьшего и второго наименьшего по счету, то стабильность требует, чтобы при вызове с эквивалентными элементами операция минимума возвратила первый элемент, а максимума — второй[1].

Мы можем обобщить минимум и максимум на выбор (j,k)-го порядка, где $k > 0$ указывает количество аргументов, а $0 \leqslant j < k$ указывает, что должен быть выбран j-й наименьший аргумент. Чтобы формализовать применяемое нами понятие стабильности, предположим, что каждый из аргументов k связан с уникальным натуральным числом, называемым его *индексом стабильности*.

[1] В следующих главах мы распространим понятие стабильности на другие категории алгоритмов.

При условии, что дано исходное слабое упорядочение r, определяем *усиленное* отношение $\widehat{r}$ на парах (объект, индекс стабильности):

$$\widehat{r}((a, i_a), (b, i_b)) \Leftrightarrow r(a, b) \vee (\neg r(b, a) \wedge i_a < i_b)$$

Если мы реализуем алгоритм выбора порядка в терминах $\widehat{r}$, то неоднозначные случаи, вызванные эквивалентными аргументами, исключаются. Естественным стандартным значением для индекса стабильности аргумента является его порядковая позиция в списке аргументов.

Разумеется, это усиленное отношение $\widehat{r}$ предоставляет нам мощное инструментальное средство проведения рассуждений о стабильности, но несложно определить простые процедуры выбора порядка, не прибегая явно к использованию индексов стабильности. Следующая реализация минимума возвращает a, если a и b эквивалентны, удовлетворяя принятому нами определению стабильности[2]:

```
template<typename R>
    requires(Relation(R))
const Domain(R)& select_0_2(const Domain(R)& a,
                            const Domain(R)& b, R r)
{
    // Предусловие: weak_ordering(r)
    if (r(b, a)) return b;
    return a;
}
```

Аналогично следующая реализация максимума возвращает b, если a и b эквивалентны, также в соответствии с принятым определением стабильности[3]:

```
template<typename R>
    requires(Relation(R))
const Domain(R)& select_1_2(const Domain(R)& a,
                            const Domain(R)& b, R r)
{
    // Предусловие: weak_ordering(r)

    if (r(b, a)) return a;
    return b;
}
```

В оставшейся части главы подразумевается выполнение предусловия weak_ordering(r).

[2]Описание принятого нами соглашения об именовании приведено ниже в этом разделе.

[3]STL необоснованно требует, чтобы функция $\max(a, b)$ возвращала a, если a и b эквивалентны.

Несомненно, было бы полезно иметь другие процедуры выбора порядка для k аргументов, но сложность составления подобной процедуры выбора порядка быстро возрастает с увеличением k, а количество различных процедур, которые могли бы потребоваться, весьма велико. Поэтому мы применяем подход, названный нами *сведением к ограниченным подзадачам*, который позволяет разрешить обе проблемы. Мы разрабатываем семейство процедур, которые принимают определенный объем информации об относительном упорядочении их аргументов.

При этом важно определить систему обозначения этих процедур. Каждое имя начинается с select_j_k, где $0 \leqslant j < k$, указывая на выбор j-го элемента из из k аргументов в соответствии с заданным упорядочением. Затем добавляется последовательность символов, указывающая предусловие упорядочения параметров, которое выражено в виде наращиваемых цепочек. Например, суффикс _ab означает, что по порядку расположены первые два параметра, а _abd показывает, что по порядку расположены первый, второй и четвертый параметры. Больше чем один подобный суффикс появляется, когда имеются предусловия, относящиеся к различным цепочкам параметров.

Например, несложно реализовать минимум и максимум для трех элементов:

```
template<typename R>
    requires(Relation(R))
const Domain(R)& select_0_3(const Domain(R)& a,
                            const Domain(R)& b,
                            const Domain(R)& c, R r)
{
    return select_0_2(select_0_2(a, b, r), c, r);
}

template<typename R>
    requires(Relation(R))
const Domain(R)& select_2_3(const Domain(R)& a,
                            const Domain(R)& b,
                            const Domain(R)& c, R r)
{
    return select_1_2(select_1_2(a, b, r), c, r);
}
```

Легко также найти медиану трех элементов, если известно, что первые два элемента находятся в порядке по возрастанию:

```
template<typename R>
    requires(Relation(R))
const Domain(R)& select_1_3_ab(const Domain(R)& a,
                               const Domain(R)& b,
```

```
                                const Domain(R)& c, R r)
{
    if (!r(c, b)) return b;       // a, b, c отсортированы
    return select_1_2(a, c, r);   // b не является медианой
}
```

Определение предусловия для select_1_3_ab требует только одного сравнения. Параметры передаются по постоянной ссылке, поэтому перемещение данных не происходит:

```
template<typename R>
    requires(Relation(R))
const Domain(R)& select_1_3(const Domain(R)& a,
                            const Domain(R)& b,
                            const Domain(R)& c, R r)
{
    if (r(b, a)) return select_1_3_ab(b, a, c, r);
    return              select_1_3_ab(a, b, c, r);
}
```

В наихудшем случае в select_1_3 выполняются три сравнения. В этой функции производятся два сравнения, только если c является максимумом среди a, b, c, а поскольку это происходит в одной трети случаев, среднее количество сравнений равно $2\frac{2}{3}$, при условии, что распределение входных данных равномерно.

Для поиска второго из наименьших среди n элементов требуется по крайней мере $n+\lceil\log_2 n\rceil-2$ сравнений[4]. В частности, для поиска второго элемента из четырех требуются четыре сравнения.

Второй аргумент из четырех выбрать несложно, если известно, что обе пары аргументов, и первая, и вторая, заданы в порядке возрастания:

```
template<typename R>
    requires(Relation(R))
const Domain(R)& select_1_4_ab_cd(const Domain(R)& a,
                                  const Domain(R)& b,
                                  const Domain(R)& c,
                                  const Domain(R)& d, R r) {
    if (r(c, a)) return select_0_2(a, d, r);
    return              select_0_2(b, c, r);
}
```

Предусловие для select_1_4_ab_cd может быть установлено с применением одного сравнения, если известно, что первая пара аргументов приведена в порядке возрастания:

[4]Этот результат был предсказан Йозефом Шрейером и доказан Сергеем Кислицыным [Knuth 1998, теорема S, стр. 209].

```
template<typename R>
    requires(Relation(R))
const Domain(R)& select_1_4_ab(const Domain(R)& a,
                               const Domain(R)& b,
                               const Domain(R)& c,
                               const Domain(R)& d, R r) {
    if (r(d, c)) return select_1_4_ab_cd(a, b, d, c, r);
    return              select_1_4_ab_cd(a, b, c, d, r);
}
```

Предусловие для select_1_4_ab может быть установлено с помощью одного сравнения:

```
template<typename R>
    requires(Relation(R))
const Domain(R)& select_1_4(const Domain(R)& a,
                            const Domain(R)& b,
                            const Domain(R)& c,
                            const Domain(R)& d, R r) {
    if (r(b, a)) return select_1_4_ab(b, a, c, d, r);
    return              select_1_4_ab(a, b, c, d, r);
}
```

Упражнение 4.4. Реализуйте select_2_4.

Поддержка стабильности сетей выбора порядка вплоть до порядка 4 оказалась не слишком затруднительной. Но начиная с порядка 5 возникают ситуации, в котором процедура, соответствующая ограниченной подзадаче, вызывается из исходной вызывающей программы с аргументами, не приведенными в требуемом порядке, что приводит к нарушению стабильности. Успешно справляться с такими ситуациями позволяет систематический подход, который состоит в том, что наряду с фактическими параметрами передаются индексы стабильности и используется усиленное отношение $\widehat{\mathrm{r}}$. Мы избегаем дополнительного увеличения времени выполнения благодаря использованию параметров в виде целочисленных шаблонов.

Мы именуем индексы стабильности как ia, ib, ... в соответствии с параметрами a, b и т. д. Для получения усиленного отношения $\widehat{\mathrm{r}}$ используется шаблон функционального объекта compare_strict_or_reflexive, принимающий параметр шаблона bool, значение true которого показывает, что индексы стабильности его аргументов находятся в порядке возрастания:

```
template<bool strict, typename R>
    requires(Relation(R))
struct compare_strict_or_reflexive;
```

При создании экземпляра compare_strict_or_reflexive мы предоставляем соответствующий булев аргумент шаблона:

```
template<int ia, int ib, typename R>
    requires(Relation(R))
const Domain(R)& select_0_2(const Domain(R)& a,
                            const Domain(R)& b, R r)
{
    compare_strict_or_reflexive<(ia < ib), R> cmp;
    if (cmp(b, a, r)) return b;
    return a;
}
```

Мы специализируем compare_strict_or_reflexive для следующих двух случаев: 1) индексы стабильности находятся в порядке возрастания, и в этом случае используется исходное строгое отношение r; и 2) имеет место порядок убывания; при этом используется соответствующая рефлексивная версия r:

```
template<typename R>
    requires(Relation(R))
struct compare_strict_or_reflexive<true, R> // строгая
{
    bool operator()(const Domain(R)& a,
                    const Domain(R)& b, R r)
    {
        return r(a, b);
    }
};

template<typename R>
    requires(Relation(R))
struct compare_strict_or_reflexive<false, R> // рефлексивная
{
    bool operator()(const Domain(R)& a,
                    const Domain(R)& b, R r)
    {
        return !r(b, a); // complement_of_converse_r(a, b)
    }
};
```

Когда процедура выбора порядка с индексами стабильности вызывает другую такую процедуру, тогда передаются индексы стабильности, соответствующие параметрам, в том же порядке, в каком они появляются в вызове:

```
template<int ia, int ib, int ic, int id, typename R>
    requires(Relation(R))
const Domain(R)& select_1_4_ab_cd(const Domain(R)& a,
                                  const Domain(R)& b,
                                  const Domain(R)& c,
                                  const Domain(R)& d, R r)
```

```
{
    compare_strict_or_reflexive<(ia < ic), R> cmp;
    if (cmp(c, a, r)) return
        select_0_2<ia,id>(a, d, r);
    return
        select_0_2<ib,ic>(b, c, r);
}

template<int ia, int ib, int ic, int id, typename R>
    requires(Relation(R))
const Domain(R)& select_1_4_ab(const Domain(R)& a,
                               const Domain(R)& b,
                               const Domain(R)& c,
                               const Domain(R)& d, R r)
{
    compare_strict_or_reflexive<(ic < id), R> cmp;
    if (cmp(d, c, r)) return
        select_1_4_ab_cd<ia,ib,id,ic>(a, b, d, c, r);
    return
        select_1_4_ab_cd<ia,ib,ic,id>(a, b, c, d, r);
}

template<int ia, int ib, int ic, int id, typename R>
    requires(Relation(R))
const Domain(R)& select_1_4(const Domain(R)& a,
                            const Domain(R)& b,
                            const Domain(R)& c,
                            const Domain(R)& d, R r)
{
    compare_strict_or_reflexive<(ia < ib), R> cmp;
    if (cmp(b, a, r)) return
        select_1_4_ab<ib,ia,ic,id>(b, a, c, d, r);
    return
        select_1_4_ab<ia,ib,ic,id>(a, b, c, d, r);
}
```

Теперь мы готовы реализовать процедуры выбора порядка 5:

```
template<int ia, int ib, int ic, int id, int ie, typename R>
    requires(Relation(R))
const Domain(R)& select_2_5_ab_cd(const Domain(R)& a,
                                  const Domain(R)& b,
                                  const Domain(R)& c,
                                  const Domain(R)& d,
                                  const Domain(R)& e, R r)
{
    compare_strict_or_reflexive<(ia < ic), R> cmp;
```

```
    if (cmp(c, a, r)) return
        select_1_4_ab<ia,ib,id,ie>(a, b, d, e, r);
    return
        select_1_4_ab<ic,id,ib,ie>(c, d, b, e, r);
}

template<int ia, int ib, int ic, int id, int ie, typename R>
    requires(Relation(R))

const Domain(R)& select_2_5_ab(const Domain(R)& a,
                               const Domain(R)& b,
                               const Domain(R)& c,
                               const Domain(R)& d,
                               const Domain(R)& e, R r)
{
    compare_strict_or_reflexive<(ic < id), R> cmp;
    if (cmp(d, c, r)) return
        select_2_5_ab_cd<ia,ib,id,ic,ie>(
                        a, b, d, c, e, r);
    return
        select_2_5_ab_cd<ia,ib,ic,id,ie>(
                        a, b, c, d, e, r);
}

template<int ia, int ib, int ic, int id, int ie, typename R>
    requires(Relation(R))
const Domain(R)& select_2_5(const Domain(R)& a,
                            const Domain(R)& b,
                            const Domain(R)& c,
                            const Domain(R)& d,
                            const Domain(R)& e, R r)
{
    compare_strict_or_reflexive<(ia < ib), R> cmp;
    if (cmp(b, a, r)) return
        select_2_5_ab<ib,ia,ic,id,ie>(b, a, c, d, e, r);
    return
        select_2_5_ab<ia,ib,ic,id,ie>(a, b, c, d, e, r);
}
```

Лемма 4.7. select_2_5 выполняет шесть сравнений.

Упражнение 4.5. Найдите алгоритм определения медианы для 5 аргументов, который в среднем выполняет немного меньше сравнений.

Мы можем оформить процедуру выбора порядка с внешней процедурой, которая предоставляет в качестве индексов стабильности любой строго возрастающий ряд целочисленных постоянных; обычно используются подряд идущие целые числа начиная с 0:

```
template<typename R>
    requires(Relation(R))
const Domain(R)& median_5(const Domain(R)& a,
                          const Domain(R)& b,
                          const Domain(R)& c,
                          const Domain(R)& d,
                          const Domain(R)& e, R r)
{
    return select_2_5<0,1,2,3,4>(a, b, c, d, e, r);
}
```

Упражнение 4.6. Докажите стабильность каждой процедуры выбора порядка, приведенной в этом разделе.

Упражнение 4.7. Проверьте правильность и стабильность каждой процедуры выбора порядка в этом разделе с помощью исчерпывающего тестирования.

Проект 4.1. Спроектируйте набор необходимых и достаточных условий сохранения стабильности при составлении процедур выбора порядка.

Проект 4.2. Создайте библиотеку процедур проверки на минимум для стабильной сортировки и слияния[5]. Минимизируйте не только количество сравнений, но и количество перемещений данных.

4.4. Естественное полное упорядочение

Равенство применительно к типу определяется уникальным образом, поскольку равенство значений конкретного типа показывает, что эти значения представляют одну и ту же сущность. Зачастую нельзя уникально определить естественное полное упорядочение применительно к некоторому типу. Что касается конкретных видов, то часто можно определить много полных и слабых упорядочений, причем ни одно из них не играет какую-то особую роль. Применительно к абстрактным видам может быть задано одно специальное полное упорядочение, в котором поддерживаются все фундаментальные операции для конкретного вида. Такое упорядочение именуется *естественным полным упорядочением* и обозначается символом $<$, как показано ниже.

$$\begin{aligned} &\mathit{TotallyOrdered}(\mathsf{T}) \triangleq \\ &\quad \mathit{Regular}(\mathsf{T}) \\ &\quad \wedge\ <\colon \mathsf{T} \times \mathsf{T} \to \mathsf{bool} \\ &\quad \wedge\ \mathsf{total\_ordering}(<) \end{aligned}$$

Например, фундаментальные операции поддерживаются в естественном полном упорядочении на целых числах:

[5] См. [Knuth 1998, Section 5.3: Optimum Sorting].

$$a < \mathsf{successor}(a)$$
$$a < b \Rightarrow \mathsf{successor}(a) < \mathsf{successor}(b)$$
$$a < b \Rightarrow a + c < b + c$$
$$a < b \wedge 0 < c \Rightarrow ca < cb$$

Иногда тип не имеет естественного полного упорядочения. Например, комплексные числа и личные дела служащих не имеют естественных полных упорядочений. Мы требуем, чтобы регулярные типы обеспечивали *стандартное полное упорядочение* (иногда сокращенно называемое *стандартным упорядочением*) для предоставления возможности логарифмического поиска. Примером стандартного полного упорядочения значений в условиях отсутствия естественного полного упорядочения может служить лексикографическое упорядочение для комплексных чисел. Если естественное полное упорядочение существует, оно совпадает со стандартным упорядочением. Мы используем следующую систему обозначений:

	Определения	C++
Стандартное упорядочение для `T`	less_T	`less<T>`

4.5. Семейства производных процедур

Некоторые процедуры естественным образом объединяются в семейства. Если некоторые процедуры в семействе уже определены, то на этой основе могут быть легко получены определения других процедур. Всякий раз, когда определено равенство, становится определенным неравенство — дополнение равенства; операторы $=$ и $\neq$ должны быть определены непротиворечивым способом. Для каждого полностью упорядоченного типа должны быть определены одновременно все четыре оператора, $<$, $>$, $\leqslant$ и $\geqslant$, таким образом, чтобы имело место следующее:

$$a > b \Leftrightarrow b < a$$
$$a \leqslant b \Leftrightarrow \neg(b < a)$$
$$a \geqslant b \Leftrightarrow \neg(a < b)$$

4.6. Расширение процедур выбора порядка

Процедуры выбора порядка в этой главе не возвращают объект, который может быть изменен, поскольку они работают с постоянными ссылками. Но удобнее и проще, если предусмотрены версии, которые возвращают изменяемый объект, что позволяет использовать их в левой стороне операции присваивания или в качестве изменяемого аргумента для действия или процедуры накопления. Перегруженная изменяемая версия процедуры выбора порядка может быть реализована путем удаления в неизменяемой версии ключевого

слова const в объявлении каждого типа параметра и типа результата. Например, применяемая нами версия select_0_2 может быть дополнена следующим образом:

```
template<typename R>
    requires(Relation(R))
Domain(R)& select_0_2(Domain(R)& a, Domain(R)& b, R r)
{
    if (r(b, a)) return b;
    return a;
}
```

Кроме того, в любой библиотеке должны быть предусмотрены версии для полностью упорядоченных типов (с операцией $<$), так как это — обычный случай. Это означает, что имеются четыре версии каждой процедуры.

Согласно законам трихотомии и слабой трихотомии, которым подчиняется полное и слабое упорядочение, вместо двузначного отношения можно использовать трехзначную процедуру сравнения, в связи с тем, что в некоторых ситуациях это позволяет избежать дополнительного вызова процедуры.

Упражнение 4.8. Перепишите алгоритмы в этой главе с использованием трехзначного сравнения.

4.7. Резюме

Аксиомы полного и слабого упорядочения обеспечивают интерфейс для подключения определенных упорядочений к универсальным алгоритмам. Систематический поиск решений небольших задач позволяет облегчить декомпозицию крупных задач. Процедуры объединяются в семейства со взаимосвязанной семантикой.