УДК 512

ОСТАНОВКА АЛГОРИТМА F5

В. В. Галкин[1]

Алгоритм F5, предложенный Фожером, принимает в качестве входных данных произвольное множество однородных многочленов и корректность результата доказана для всех случаев, когда алгоритм останавливается. Однако остановка алгоритма за конечное число шагов доказана лишь для случая, когда на вход алгоритма подаётся регулярная последовательность многочленов. В этой работе показано, что алгоритм останавливается на любых входных данных без какого-либо использования регулярности

Ключевые слова: базис Грёбнера, алгоритм F5, доказательство остановки

The original F5 algorithm introduced by Faugère is formulated for any homogeneous polynomial set input. The correctness of output is shown for any input that terminates the algorithm, but the termination itself is proved only for the case of input being regular polynomial sequence. This article shows that algorithm correctly terminates for any homogeneous input without any reference to regularity

Key words: Groebner basis, F5 algorithm, termination proof

1 Введение

Алгоритм Фожера F5 известен как эффективный алгоритм вычисления базисов Грёбнера, но одной из главных проблем в его практическом использовании является отсутствие доказательства остановки для всех входных данных. Первоисточник [1] и детальные исследования в [2] показывают остановку алгоритма только для случая отсутствия редукций к нулю, что с практической точки зрения означает доказательство для тех случаев, когда входное множество многочленов представлено регулярной последовательностью. Но для большинства входных последовательностей их регулярность неизвестна, и доказанного факта оказывается недостаточно для утверждения остановки реализации алгоритма на конкретных входных данных. Один из подходов к решению проблемы – добавление в алгоритм дополнительных проверок и критериев, гарантирующих остановку алгоритма. Этот подход даёт строгое доказательство остановки, однако получаемый результат есть доказательство остановки модифицированной версии F5, содержащей дополнительные проверки, которая в силу этого может быть более сложна в реализации или иметь большее время работы на некоторых входных данных. Этот подход применяется в работах [3, 4, 5, 6, 7]. Другой подход состоит в доказательстве остановки некоторого семейства алгоритмов, основанных на идеях F5 с последующей попыткой переформулировать F5 таким образом, чтоб он являлся представителем этого семейства. Основная проблема этого подхода появляется в процессе переформулировки: описание F5 в других терминах может привести к незаметному внесению различий в поведение алгоритма, которые трудно заметить, но которые требуют дополнительных рассуждений для доказательства эквивалентности с F5. К примеру, [8] доказывает остановку алгоритма F5GEN, который отличается от исходного F5 отсутствием проверки критериев при выборе редуктора. Работа [9] даёт доказательство остановки алгоритма TRB-F5, который, как удалось осознать автору в процессе плодотворных дискуссиях с Джоном Перри, имеет два существенных отличия от F5. Первое отличие – другая схема построения правил, приводящая в конце концов к тому, что в процессе выполнения TRB-F5 правила в массивах *Rule* оказываются отсортированными по возрастания сигнатуры. Второе отличие – отсутствие в TRB-F5 применения оператора нормальной формы φ перед редукцией, что приводит к эффекту противоположному отличиям F5 от F5GEN: алгоритм TRB-F5 при выборе редуктора проверяет критерии для элементов с бóльшими индексами, которые в F5 используются неявно в операторе нормальной формы и за счёт этого не подвергаются проверки критериев. Автор полагает, что эти алгоритмы могут быть изменены таким образом, чтоб в точности повторять поведение алгоритма F5, и доказательство остановки может быть перенесено на изменённые версии. Однако подход с алгоритмами, эквивалентными F5 имеет недостаток: он усложняет понимание того, как теоремы, используемые для доказательства остановки могут быть сформулированы в терминах поведения исходного алгоритма F5. Подход к доказательству остановки, предлагаемый в данной работе, применяется к F5 без каких-либо модификаций. Первый шаг доказательства основан на предлагаемой ниже идеи цепей S-пар. Второй шаг основывается на методе, использованном в теореме 21 работы [10] для доказательства корректности алгоритма F5C: представление S-многочлена в виде

[1] *Галкин Василий Витальевич* — асп. каф. алгебры мех.-мат. ф-та МГУ, e-mail: galkin-vv@yandex.ru.

суммы домноженных многочленов из множества, вычисленного F5C может быть модифицировано последовательностью замен S-пар и их отброшенных частей, и за конечное число таких шагов приведено к состоянию, когда выполняются определённые «хорошие» свойства. Вторая часть этой статьи показывает, что предусловия этого метода могут быть ослаблены для его применения к множеству на любом промежуточном шаге вычислений F5, и что получаемые следствия могут быть усилены для их использования в доказательстве остановки. Работа оформлена как альтернативное доказательство остановки для алгоритма, описанного в статье [1], доступной на сайте её автора, поэтому читатель предполагается хорошо знакомым с ней. Большинство используемой терминологии, включая названия этапов алгоритма, взято оттуда.

2 Потенциально бесконечные циклы в F5

2.1 Процедура `AlgorithmF5`: увеличение d

Утверждение 1. Если при некоторых входных данных цикл **while** внутри процедуры `AlgorithmF5` выполняется бесконечное число раз, то значение d неограниченно возрастает.

Доказательство. Предположим существование последовательности многочленов $\{f_1, \ldots, f_m\}$ в кольце $\mathcal{K}[x_1, \ldots, x_n]$ для которой алгоритм F5 не завершается. Без ограничение общности будем считать что это самая короткая последовательность такого рода – алгоритм завершается на более короткой последовательности $\{f_2, \ldots, f_m\}$. Это означает, что не завершается последняя итерация цикла внутри `incrementalF5`, то есть не завершается последний вызов процедуры `AlgorithmF5`. Для исследования данной ситуации необходимо понять, как ведёт себя значение полной степени d в процессе выполнения цикла внутри процедуры `AlgorithmF5`. Обозначим за d_j значение d на j-ой итерации цикла и положим $d_0 = -1$. Простейшее свойство d_j – неубывание: $d_j \geqslant d_{j-1}$. Оно выполняется, поскольку на $j-1$-ой итерации все многочлены в R_d имеют степень d_{j-1}, и поэтому все вновь создаваемые критические пары имеют степень не менее d_{j-1}. Предположим теперь, что j фиксированный номер некоторой итерации. В начале итерации j все критические пары степени d_j извлекаются из P. После вызова процедуры `Reduction` в P добавляются некоторые новые критические пары в цикле, итерирующем по R_d. Существует возможность что некоторые из них будут иметь полную степень равную d_j. Нижеследующие рассуждения призваны показать, что все критические пары такой полной степени будут отброшены на следующей итерации алгоритма и ни одна из них не породит S-многочлен. Для каждой из критических пар $[t, u_1, r_1, u_2, r_2]$ порождённых на итерации j как минимум один из многочленов пары принадлежит R_d и не более чем один многочлен из пары мог принадлежать G_i на момент начала итерации. Все многочлены R_d генерируются процедурой `Reduction` путём добавления по одному многочлену к множеству *Done*. Поэтому, среди одного или двух многочленов критической пары, принадлежащих R_d, мы можем выбрать многочлен r_k добавленный в *Done* позже. Тогда про другой многочлен S-пары r_{3-k} можно утверждать, что он уже присутствовал в $G \cup Done$ к моменту добавления r_k в *Done*. Поэтому процедура `TopReduction` пыталась редуцировать r_k по r_{3-k}, но не сделала этого, поскольку в функции `IsReducible` одна из проверок (a) - (d) запретила это. При этом для критических пар с полной степенью равной d_j мы имеем $u_k = 1$, поскольку полная степень критической пары равна полной степени её члена r_k. Это означает, что значение u_{3-k} равно $\frac{\mathrm{HM}(r_k)}{\mathrm{HM}(r_{3-k})}$, поэтому в `IsReducible` правило (a) разрешает редукцию r_k по r_{3-k}. Получается, что только правила (b) - (d) могли запретить редукцию. Предположим, что редукция была запрещена правилом (b). Это означает, что в G_{i+1} существует многочлен, редуцирующий $u_{3-k}\mathcal{S}(r_{3-k})$. Для нашего случая отсюда следовало бы, что в функции `CritPair` эквивалентная проверка $\varphi(u_{3-k}\mathcal{S}(r_{3-k})) = u_{3-k}\mathcal{S}(r_{3-k})$ запретила бы создание критической пары. Получается, что правило (b) также не могло запретить редукцию. Предположим, что редукция была запрещена правилом (c). Это означает существование перезаписи для домноженного редуктора. В нашем случае это значит, что `Rewritten?`(u_{3-k}, r_{3-k}) показала наличие перезаписи в процессе выполнения `TopReduction`, и будет продолжать возвращать значение «Истина» на всех последующих этапах алгоритма, поскольку перезаписывающие многочлены не могут исчезнуть. Предположим, что редукция была запрещена правилом (d). Псевдокод в [1] несколько неясен в этом месте, но исходный код процедуры `FindReductor`, приложенный к [2] достаточно ясен и утверждает, что редуктор отбрасывается, если одновременно моном и индекс сигнатуры совпадают для редуктора и редуцируемого многочлена (в коде `r[k0][1]` – моном сигнатуры, а `r[k0][2]` – её индекс):

```
if (ut eq r[k0][1]) and (r[j][2] eq r[k0][2]) then
 continue;
end if;
```

В нашем случае это обозначает, что сигнатуры r_k и $u_{3-k}r_{3-k}$ равны. Значит вызов `Rewritten?`(u_{3-k}, r_{3-k}) будет возвращать «Истину» после добавления правила, соответствующего r_k, поскольку $u_{3-k} \cdot r_{3-k}$ перезаписывается $1 \cdot r_k$. Получается, что как и в случае с правилом (c), `Rewritten?`(u_{3-k}, r_{3-k}) возвращает «Истину» при выполнении `TopReduction` и позже. Теперь рассмотрим функцию `Spol`, выполняемую для некоторой S-пары полной степени d_j,

сгенерированной на итерации j. Она выполняется внутри $j+1$ итерации цикла в `AlgorithmF5`, то есть уже после того, как закончилось выполнение `TopReduction` для r_k. Поэтому для случаев (c) и (d) вызов `Rewritten?`(u_{3-k}, r_{3-k}) внутри `Spol` вернёт «Истину». Значит на итерации $j+1$ ни одна из S-пар полной степени d_j не добавит многочлена в F. Итого, получено:

- первая возможность относительного расположения d_{j+1} и d_j – их равенство: $d_{j+1} = d_j$. В этом случае F оказывается пустым на итерации $j+1$, и, таким образом, P не содержит ни одной пары с полной степенью d_j после того как итерация $j+1$ завершится. Значит $d_{j+2} > d_{j+1}$.
- Другая возможность относительного расположения – строгое возрастание $d_{j+1} > d_j$.

Вместе эти факты дают $\forall j\ d_{j+2} > d_j$, что доказывает утверждение 1.

2.2 Процедура `Reduction`: конечность $ToDo$

Утверждение 2. Каждая итерация цикла внутри процедуры `AlgorithmF5` останавливается, в частности останавливаются все вызовы процедуры `Reduction`.

Доказательство. Факт остановки известен для вызовов `AlgorithmF5`, соответствующих многочленам $f_2, \ldots, f_m$, поэтому будет рассматриваться лишь один оставшийся вызов `AlgorithmF5`, обрабатывающий первый элемент входного набора многочленов f_1. Вначале покажем несколько общих утверждений о многочленах в множествах $ToDo$ и $Rule$ в процессе j-ой итераций цикла внутри этого вызова `AlgorithmF5`. Старшая по сигнатуре часть S-пары всех критических пар, добавляемых функцией `CritPair` вначале выполнения `AlgorithmF5`, обладает индексом сигнатуры, равным 1. Все прочие критические пары порождаются с индексом сигнатуры, соответствующим некоторому отмеченному многочлену, перемещённому из множества $ToDo$ в множество $Done$. В свою очередь все элементы $ToDo$ создаются или на основе критических пар или внутри процедуры `TopReduction`. Многочлены, генерируемые `TopReduction` имеют сигнатуру, превышающую сигнатуру многочлена, который редуцирует данный вызов процедуры, и который является элементом $ToDo$. Поэтому многочлен или критическая пара с индексом сигнатуры, отличным от 1 не могут появиться в рассматриваемом вызове `AlgorithmF5`. С другой стороны, все многочлены в $ToDo$ имеют одну и ту же полную степень d_j. Вместе с единичностью индексов это позволяет заключить, что полная степень мономов сигнатур равна $d_j - \deg(f_1)$ для всех элементов $ToDo$. Каждое добавление элемента в массив $Rule$ соответствует добавлению в множество $ToDo$. Поэтому, элементы добавляемые в $Rule$ на итерации j имеют полную степень равную d_j. Вспоминая неубывание d_j получаем, что на j-ой итерации все элементы $Rule$ с индексом сигнатуры 1 имеют полную степень $\leqslant d_j$ и полную степень монома сигнатуры $\leqslant d_j - \deg(f_1)$. Также это даёт информацию о порядке элементов $Rule$ с сигнатурой 1: их полная степень не убывает.

Определение 1. Редукция отмеченного многочлена r_k по отмеченному многочлену r_m называется *сигнатурной редукцией,* если $\mathcal{S}(r_k) \succ t \cdot \mathcal{S}(r_m)$, где $t = \frac{\mathrm{HM}(r_k)}{\mathrm{HM}(r_m)}$ – моном, на который умножается редуктор. Редуктор, соответствующий такой редукции называется *сигнатурным редуктором.*

Алгоритм производит только сигнатурные редукции: процедура `TopReduction` производит редукцию по неотброшенному редуктору, если он сигнатурный, и добавляет элемент в $ToDo$ в противном случае. Элементы $ToDo$ обрабатываются в порядке возрастания сигнатур, поэтому ни один из элементов $G \cup Done$ не может иметь сигнатуры, превышающей сигнатуру многочлена r_k, редуцируемого в `TopReduction`. Рассмотрим неотброшенный проверками редуктор r_m. Если он имеет полную степень $\deg(r_m) = \deg(r_k)$, мы получаем свойства $t = 1$ и $\mathcal{S}(r_k) \succ \mathcal{S}(r_m)$, гарантирующие что редукция по нему будет сигнатурной. Случай $\mathcal{S}(r_k) = \mathcal{S}(r_m)$ невозможен, поскольку такие редукторы отбрасываются в правиле (d) процедуры `IsReducible`. Таким образом, ситуация $\mathcal{S}(r_k) \prec t \cdot \mathcal{S}(r_m)$ возможна только при $\deg(r_m) < \deg(r_k)$ и все добавления в $ToDo$ в процессе `TopReduction` соответствуют этому случаю. Моном сигнатуры многочлена, добавляемого таким образом, равен $t \cdot \mathcal{S}(r_m)$ и тот факт что r_m не был отброшен проверкой правила `Rewritten?` внутри `IsReducible` гарантирует, что ни одного многочлена с сигнатурой $t\mathcal{S}(r_m)$ не было порождено, потому что иначе такой многочлен имел бы связанное с ним правило в $Rule$ с большей полной степенью, чем правило, соответствующее r_m, и r_m был бы отброшен в `IsReducible`. Мы хотим показать, что единственная возможность отсутствия остановки алгоритма соответствует случая неограниченного возрастания d_j. Мы показали, что отсутствие остановки алгоритма происходит в случае, когда не завершается вызов `AlgorithmF5`, и что он не может зациклиться обрабатывая бесконечное число итераций итерации с одним и тем же значением d. Остаются два варианта: неограниченное возрастание d и зацикливание внутри одной из итераций с фиксированным значением. Далее показано, что такое зацикливание невозможно. Процедура `AlgorithmF5` содержит 3 цикла помимо главного:

- **for**-цикл внутри `Spol` завершается, поскольку число его итераций ограничено числом критических пар к моменту начала выполнения цикла;

- **for**-цикл внутри `AlgorithmF5` проходит по элементам R_d и также завершается, поскольку число элементов R_d зафиксировано к моменту начала выполнения цикла;
- наиболее сложный случай соответствует **while**-циклу внутри процедуры `Reduction`, который выполняется до тех пор, пока множество $ToDo$ не станет пустым. Множество $ToDo$ изначально заполняется процедурой `Spol` и потом дополняется новыми элементами в процессе выполнения `TopReduction`. Процедура `Spol` порождает конечное число элементов, поскольку она завершается, а все элементы добавляемые `TopReduction` имеют различные сигнатуры индекса 1, поэтому их число ограничено числом различных сигнатур полной степени $d_j - \deg(f_1)$, поэтому в $ToDo$ добавляется лишь конечное число элементов. Теперь мы покажем, что все типы шагов, происходящих внутри `Reduction` могут быть выполнены лишь конечное число раз:
  - шаг, на котором `IsReducible` возвращает пустое множество, соответствует переносу элемента множества $ToDo$ в $Done$ и число таких шагов ограничено числом элементов, добавляемых в $ToDo$
  - шаг, на котором `IsReducible` возвращает не сигнатурный редуктор, соответствует добавлению нового элемента в $ToDo$ и число таких шагов ограничено числом возможных добавлений
  - шаг, на котором `IsReducible` возвращает сигнатурный редуктор, соответствует редукции одного из элементов $ToDo$. Это может произойти лишь конечное число раз, поскольку в $ToDo$ добавляется конечное число многочленов и не может существовать бесконечной цепочки редукций для одного многочлена, поскольку в процессе редукции его старший моном HM строго $\prec$-убывает, а множество мономов вполне упорядочено по $\prec$.

Мы получили, что все циклы внутри процедуры `AlgorithmF5`, кроме главного, завершаются, что доказывает утверждение 2.

Отсюда получается следующий факт о поведении алгоритма в ситуации, когда он не завершается:

Утверждение 3. Если алгоритм не завершается на некоторых входных данных, то значение d неограниченно возрастает в процессе итераций.

Доказательство. Следует из комбинирования утверждений 1 и 2.

3 Цепи S-пар

Утверждение 3 показывает, что в случае отсутствия остановки работа алгоритма приводит к появлению бесконечной последовательности ненулевых отмеченных многочленов с неограниченно возрастающей полной степенью, добавляемых в G_i. То есть, в этом случае алгоритм порождает бесконечную последовательность отмеченных многочленов $\{r_1, r_2, \ldots, r_m, \ldots, r_l, \ldots\}$, в которой $r_1, \ldots, r_m$ соответствуют m исходным многочленам, а остальные были получены в процедурах `Spol` и `TopReduction`. В обоих случаях новый элемент r_l порождается как S-многочлен двух уже ранее добавленных в последовательность многочленов. Будем обозначать за l^* и l_* позиции многочленов, использовавшихся для генерации l-го многочлена последовательности и за $\overline{u_l}$, $\underline{u_l}$ мономы, на которые они умножались. При этом l^* соответствует части с большей сигнатурой: $\mathrm{poly}(r_l) = \overline{u_l}\mathrm{poly}(r_{l^*}) - \underline{u_l}\mathrm{poly}(r_{l_*})$ и $\mathcal{S}(r_l) = \overline{u_l}\mathcal{S}(r_{l^*}) \succ \underline{u_l}\mathcal{S}(r_{l_*})$. Значение $\mathrm{poly}(r_l)$ может меняться в процедуре `TopReduction` на многочлен с меньшим HM, но $\mathcal{S}(r_l)$ нигде далее не меняется после добавления многочлена в последовательность. Далее, будем пытаться найти бесконечную подпоследовательность $\{r_{k_1}, r_{k_2}, \ldots, r_{k_n}, \ldots\}$ в этой последовательности, обладающую свойством, что r_{k_n} является S-многочленом $r_{k_{n-1}} = r_{k_n^*}$ и некоторого другого многочлена меньшей сигнатуры. то есть $\mathcal{S}(r_{k_n}) = \overline{u_{k_n}}\mathcal{S}(r_{k_{n-1}})$ и

$$\mathcal{S}(r_{k_{n-1}}) | \mathcal{S}(r_{k_n}). \tag{3.1}$$

Определение 2. Конечную или бесконечную последовательность, соседние элементы которой удовлетворяют свойству 3.1 будем называть *цепью S-пар*.

Каждый порождаемый многочлен r_l имеет конечную цепь S-пар, оканчивающуюся этим многочленом. Эта цепь может быть последовательно построена, начиная с последнего элементаr_l, если на каждом шаге переходить от текущего многочлена r_n к многочлену r_{n^*}, который использовался при генерации r_n как S-многочлена. Результирующая цепь S-пар имеет вид $\{r_q, \ldots, r_{l^{**}}, r_{l^*}, r_l\}$, где все многочлены имеют одинаковый индекс сигнатуры $q = \mathrm{index}(r_l)$ и первый элемент является входным многочленом этого индекса. Первое свойство цепей S-пар основано на критерии перезаписи и заключается в следующей теореме.

Теорема 3. *Любой отмеченный многочлен может являться начальным элементом лишь конечного числа различных цепей S-пар длины 2.*

Доказательство. Алгоритм `AlgorithmF5` считает S-полиномы в двух местах: процедуре `SPol` и процедуре `TopReduction`. Важно заметить, что в обоих случаях проверка `Rewritten?` для части S-полинома с большей

сигнатурой выполняется непосредственно перед созданием S-многочлена. В первом случае такая проверка производится в самой `SPol`, а в `TopReduction` проверка присутствует в вызове `IsReducible`. И в обоих случаях получаемый S-многочлен немедленно добавляется в список *Rule* последним элементом. Поэтому, в момент построения S-многочлена с сигнатурой s можно утверждать, что старая часть S-пары соответствует последнему из правил с сигнатурой, делящей s – она даже может быть найдена исходя лишь из списка *Rule* и сигнатуры s без знания какой-либо ещё информации об алгоритме. Рассмотрим произвольный отмеченный многочлен r_L с сигнатурой $\mathcal{S}(r_L) = s$ и упорядоченное по порядку добавления подмножество $\{r_{l_1}, \ldots, r_{l_i}, \ldots\}$ отмеченных многочленов с сигнатурами удовлетворяющими условию $\mathcal{S}(r_{l_i}) = v_i\mathcal{S}(r_L)$. С точки зрения делимости любая из потенциально бесконечного числа пар $\{r_L, r_{l_i}\}$ может быть цепью S-пар длины 2. Но идеал (v_i) в T является конечно порождённым по лемме Диксона, поэтому после некоторого шага i_0 будет выполняться$\forall i > i_0\, \exists j \leqslant i_0$ такое что $v_j | v_i$. Поэтому при $\forall i > i_0$ последовательность $\{r_L, r_{l_i}\}$ не может являться цепью S-пар, поскольку $\mathcal{S}(r_L) \cdot v_i$ перезаписывается $\mathcal{S}(r_{l_j}) \cdot \frac{v_i}{v_j}$ и существует не более чем i_0 цепей S-пар длины 2, начинающихся с многочлена r_L.

Определение 4. Конечное множество концов цепей S-пар длины 2, начинающихся с r_L будет называться *множеством S-порождённых* r_L.

Теорема 5. *Если алгоритм не останавливается на некоторых входных данных, то он порождает бесконечную цепь S-пар* $\{h_i\}$.

Доказательство. Поскольку при работе с понятием бесконечности требуется некоторая строгость, дадим следующее определение.

Определение 6. Отмеченный многочлен r_l называется *генератором цепи S-пар*, если существует бесконечное множество различных конечных цепей S-пар, начинающихся с r_l.

Если алгоритм не останавливается, то многочлен входного множества $r_1 = (f_1, 1F_1)$ является генератором цепи S-пар, поскольку каждый многочлен r_l, порождаемый в последнем не завершающемся вызове `AlgorithmF5` имеет индекс сигнатуры 1 и является концом цепи S-пар$\{r_1, \ldots, r_{l^{**}}, r_{l^*}, r_l\}$. Теперь предположим, что про некоторый отмеченный многочлен r_l известно, что он является генератором цепи S-пар. Один из конечного множества S-порождённых r_l также должен являться генератором цепи S-пар, поскольку в противном случае число различных цепей, исходящих из r_l, было бы ограничено конечной суммой конечных количеств цепей, выходящих из S-порождённых плюс конечным количеством цепей длины 2, выходящих из r_l. Поэтому, если отмеченный многочлен r_l является генератором цепи S-пар, среди его S-порождённых всегда может быть выбран другой генератор цепи S-пар. Таким образом может быть построена бесконечная цепь S-пар, начинающаяся r_1 и состоящая из генераторов, что доказывает теорему.

Для следующей теоремы необходимо ввести порядок на частных, образованных мономами, путём транзитивного расширения порядка на мономах: $\frac{m_1}{m_2} >_q \frac{m_3}{m_4} \Leftrightarrow m_1 m_4 > m_3 m_2$.

Теорема 7. *Если алгоритм не останавливается на некоторых входных данных, то после некоторого конечного шага множество* G *содержит пару отмеченных многочленов* f', f*, причём* f *сгенерирован после* f' *и выполняются следующие 3 свойства:*

$$\mathrm{HM}(f') | \mathrm{HM}(f),$$
$$\frac{\mathrm{HM}(f')}{\mathcal{S}(f')} >_q \frac{\mathrm{HM}(f)}{\mathcal{S}(f)},$$
$$\mathcal{S}(f') | \mathcal{S}(f).$$

Доказательство. При работе с цепями S-пар является важным тот факт, что многочлен никогда не редуцируется дальше, после того как он был использован для создания S-пары в качестве старей по сигнатуре части. Факт выполняется, поскольку все многочлены, которые ещё могут быть подвергнуты редукции находятся в множестве *ToDo*, а все многочлены, используемые как старшая часть S-пары, находятся в G или в *Done*. Поэтому можно утверждать, что многочлен h_n, предшествующий многочленуh_{n+1} в цепи S-пар, сохраняет одно и то же значение $\mathrm{poly}(h_n)$ после того как был использован для создания какой-либо S-пары. и можно утверждать, что

$$\mathrm{poly}(h_{n+1}) = c\frac{\mathcal{S}(h_{n+1})}{\mathcal{S}(h_n)}\mathrm{poly}(h_n) + g_n,$$

где g_n многочлен, соответствующей младшей части S-пары, использованный при генерации h_{n+1} из h_n, и удовлетворяет следующему:

$$\mathrm{HM}(h_{n+1}) < \mathrm{HM}\left(\frac{\mathcal{S}(h_{n+1})}{\mathcal{S}(h_n)}h_n\right) = \mathrm{HM}(g_n),\ \mathcal{S}(h_{n+1}) = \mathcal{S}\left(\frac{\mathcal{S}(h_{n+1})}{\mathcal{S}(h_n)}h_n\right) \succ \mathcal{S}(g_n). \tag{3.2}$$

Из первого неравенства в 3.2 получаем, что $\frac{\mathrm{HM}(h_n)}{\mathcal{S}(h_n)} >_q \frac{\mathrm{HM}(h_{n+1})}{\mathcal{S}(h_{n+1})}$, поэтому в цепи S-пар частные $\frac{\mathrm{HM}(h_i)}{\mathcal{S}(h_i)}$ строго убывают в смысле порядка на частных. Этот факт не может быть напрямую использован для доказательства

конечности цепей, поскольку порядок на частных, в отличии от порядка на мономах, не даёт вполне упорядоченности: к примеру последовательность $\frac{x}{x} >_q \frac{x}{x^2} >_q \cdots >_q \frac{x}{x^n} >_q \cdots$ является бесконечно убывающей. Существует две возможности для отношения между НМ соседних элементов. Известно, что $\mathcal{S}(h_n)|\mathcal{S}(h_{n+1})$, поэтому они или имеют равные сигнатуры, или $\deg(h_n) < \deg(h_{n+1})$. В первом случае $\mathrm{HM}(h_{n+1}) < \mathrm{HM}(h_n)$ при равенстве полной степени, а во втором – $\mathrm{HM}(h_{n+1}) > \mathrm{HM}(h_n)$, поскольку полные степени НМ отличаются. Поэтому, последовательностьНМ элементов бесконечной цепи S-пар состоит из блоков с фиксированной полной степенью, где НМ внутри блока строго убывают. Длтны блоков могут быть равными единице, и полные степени блоков возрастают. Это приводит к следующим свойствам: цепь S-пар $\{h_i\}$ не может содержать элементов с равными НМ и $\mathrm{HM}(h_i)|\mathrm{HM}(h_j)$ возможно только в случае $i < j$ и $\deg(h_i) < \deg(h_j)$. Это позволяет использовать метод аналогичный используемому в Предложении 14 работы [11]: рассмотрим НМ бесконечной цепи S-пар $\{h_i\}$. Они порождают бесконечную последовательность в T, поэтому по лемме Диксона существует два многочлена, с $\mathrm{HM}(h_i)|\mathrm{HM}(h_j)$. Из предыдущего абзаца следует, что при этом $i < j$, а при помощи свойств цепи S-пар мы получаем, что $\mathcal{S}(h_i)|\mathcal{S}(h_{i+1})|\cdots|\mathcal{S}(h_j)$ и $\frac{\mathrm{HM}(h_i)}{\mathcal{S}(h_i)} >_q \frac{\mathrm{HM}(h_{i+1})}{\mathcal{S}(h_{i+1})} >_q \cdots >_q \frac{\mathrm{HM}(h_j)}{\mathcal{S}(h_j)}$, поэтому можно взять $f' = h_i$ и $f = h_j$.

Последнее свойство о делимости сигнатур из утверждения теоремы является побочным эффектом от использования цепей S-пар и не используется в дальнейшем. При этом первые два свойства используются для построения сигнатурного редуктора.

Факт 4. *Если никакие многочлены не были отброшены проверками критериев (b) и (c) в* `IsReducible`*, рассматриваемый алгоритм завершается.*

Доказательство. Данное выше доказательство теоремы 7 не опирается на соответствие между порядками на сигнатурах и мономах многочленов. Но алгоритм F5 использует один и тот же порядок в обоих случаях, и теперь мы можем воспользоваться этим фактом и переформулировать отношение на частных мономов из теоремы 7 в отношение на сигнатурах:

$$\mathcal{S}(f) \succ t \cdot \mathcal{S}(f'), \text{ где } t = \frac{\mathrm{HM}(f)}{\mathrm{HM}(f')} \in T.$$

Это неравенство вместе с делимостью НМ из утверждения теоремы показывает, что tf' является редуктором для f в `TopReduction` с точки зрения сигнатуры – он проходит проверки (a) и (d) внутри `IsReducible` и его сигнатура меньше. При отсутствии проверки критериев (b) и (c) это напрямую приводило бы к противоречию, так как в момент добавления f в G отмеченный многочлен f' уже был там и процедура `TopReduction` должна была бы редуцировать f по f'.

Но существование критериев делает возможной ситуацию, в которой tf' отбрасывается проверками (b) или (c) процедуры `IsReducible`. Идея дальнейших рассуждений состоит в том, чтоб показать что в этом случае может быть найден другой сигнатурный редуктор f, который не будет отброшен проверками и таким образом прийти к противоречию. Последующие главы работы посвящены этому.

4 S-пары с сигнатурами, меньшими $\mathcal{S}(g)$

В этой и последующих частях g подразумевается некоторым фиксированным многочленом с индексом сигнатуры 1, добавленный на некоторой итерации алгоритма в *Done*. Мы будем анализировать состояние алгоритма в момент непосредственно предшествующий добавлению g в *Done* в вызове `AlgorithmF5` с $i = 1$. Рассмотрим в этот момент конечное множество отмеченных многочленов $G_1 \cup Done$. Оно состоит из чисел, являющихся позиций отмеченных многочленов в R, поэтому его элементы могут быть упорядочены в соответствии с позицией в R и оно окажется записанным в виде упорядоченной последовательности целых чисел $G_g = \{b_1, \ldots, b_N\}$ с $b_j < b_{j+1}$. Необходимо отметить, что этот порядок соответствует порядку отмеченных многочленов в последовательности, получаемой склеиванием массивов правил $Rule[m] : Rule[m-1] : \cdots : Rule[1]$, поскольку добавление нового многочлена в R всегда сопровождается добавлением соответствующего правила. Но этот порядок может отличаться от порядка в котором многочлены добавлялись в множество $G_1 \cup Done$, поскольку многочлены одной полной степени добавляются в *Done* в порядке возрастания сигнатуры, при том что добавление многочленов одной полной степени в R производится в довольно случайном порядке в процедурах `Spol` и `TopReduction`. Далее для простоты мы будем говорить о отмеченных многочленах b_j в G_g, подразумевая что G_g является не упорядоченным списком позиций, а упорядоченным списком отмеченных многочленов, расположенных на этих позициях. В этой терминологии можно сказать, что все входные многочлены $\{f_1, \ldots, f_m\}$ присутствуют в G_g, поскольку они присутствуют в G_1 в момент его создания. S-пары могут обрабатываться в алгоритме различным путями, но главный факт, описывающий порядок их обработки выражается следующими свойствами, соответствующими свойствам, использованным в Теореме 21 работы [10], но рассматриваются на произвольной итерации алгоритма, а не после его остановки.

Теорема 8. *К моменту добавления g в Done каждая S-пара элементов G_g, сигнатура которой меньше $\mathcal{S}(g)$ удовлетворяет одному из трёх свойств:*

1. S-пара имеет часть, которая была отброшена критерием проверки нормальной формы φ (в `CritPair` или в `IsReducible`). Такие S-пары будут называться *S-парами с частью, удовлетворяющей критерию F5.*

2. S-пара имеет часть, которая была отброшена проверкой `Rewritten?` (в `SPol` или в `IsReducible`). Такие S-пары будут называться *S-парами с частью, удовлетворяющей критерию Перезаписи.*

3. S-пара не была отброшена, поэтому её S-многочлен был сигнатурно редуцирован по некоторым элементам G_g и результат был добавлен как элемент G_g. Такие S-пары будут называться *S-парами с известным G_g-представлением.*

Доказательство. S-пары элементов G_g обрабатываются в алгоритме двумя основными путями. Основной путь используется для S-пар, полная степень которых больше чем полная степень породивших их многочленов. Такие S-пары обрабатываются в следующем порядке:

- в процедуре `AlgorithmF5` они рассматриваются функцией `CritPair` при перемещении элементов G_i из $R_d = Done$ и при обработке входного многочлена r_i
- функция `CritPair` или отбрасывает пару после проверки нормальной формы φ или добавляет пару в P
- S-пара извлекается из P и передаётся в функцию `SPol`
- функция `SPol` или отбрасывает пару после проверки `Rewritten?` или добавляет S-многочлен в $F = ToDo$
- на некоторой итерации процедура `Reduction` берёт S-многочлен из $ToDo$, производит некоторые сигнатурные редукции и добавляет результат в $Done$.

Второй путь обработки используется для S-пар, соответствующих редукциям, запрещённым алгоритмом – соответствующие S-пары порождаются многочленами r_{l^*} и r_{l_*}, такими что $\mathrm{HM}(r_{l^*})|\mathrm{HM}(r_{l_*})$, и S-многочлен им соответствующий имеет вид $\overline{u_l} \cdot \mathrm{poly}(r_{l^*}) - 1 \cdot \mathrm{poly}(r_{l_*})$. Такая ситуация возможна, если для элементов G_g редукция r_{l_*} по r_{l^*} была запрещена сравнением сигнатур в `TopReduction` или проверками в `IsReducible`. Для этого случая порядок «обработки» S-пары такой:

- часть S-пары $\overline{u_l} \cdot r_{l^*}$ проверяется в `IsReducible`. (a) выполнено, поскольку $\mathrm{HM}(r_{l^*})|\mathrm{HM}(r_{l_*})$. Она может быть отброшена другими проверками:
  - отбрасывание пунктом (b) соответствует проверке нормальной формы φ для $\overline{u_l} \cdot r_{l^*}$
  - отбрасывание пунктом (c) соответствует проверке `Rewritten?` для $\overline{u_l} \cdot r_{l^*}$
  - отбрасывание пунктом (d) означает, что один из многочленов $\overline{u_l} \cdot r_{l^*}$ и $1 \cdot r_{l_*}$ может быть перезаписано другим, поэтому, если S-пара не была отброшена проверкой (c), данный тип отбрасывания означает, что часть S-пары $1 \cdot r_{i_1}$ не проходит проверку `Rewritten?`
- S-пары, не отброшенные в `IsReducible` возвращаются в `TopReduction`. Сравнение сигнатур в `TopReduction` запрещает редукцию r_{l_*} по r_{l^*} и помещает вычисленный S-многочлен, соответствующий S-паре, в множество $ToDo_1$
- процедура `Reduction` добавляет этот многочлен в $ToDo$
- последний шаг совпадает для обоих путей обработки S-пар: на некоторой итерации процедура `Reduction` берёт S-многочлен из $ToDo$, производит некоторые сигнатурные редукции и добавляет результат в $Done$.

Из путей обработки S-пар видно, что после окончания обработки каждая S-пара или редуцирована и добавлена в $Done$ или одна из частей S-пары была отброшена проверкой нормальной формы φ или критерием `Rewritten?`. Некоторые S-пары могут оказываться на путях обработки несколько раз, к примеру это происходит на итерации в `AlgorithmF5` со значением d, не изменившимся с прошлой итерации. Если S-пара была отброшена при первой попытке обработки, то она будет точно также отброшена и на следующей попытке. Если первая обработка добавила редуцированный многочлен в $Done$, то пара будет отбрасываться при следующих попытках обработки проверкой `Rewritten?` за счёт этого многочлена. Поэтому все попытки обработки, кроме первой, не дают ничего нового. Путь обработки не является одной процедурой, и в случае, если алгоритм не останавливается, некоторые S-пары всегда находятся в середине обработки, при этом или соответствующая S-пара находится в очереди P или S-многочлен в очереди $ToDo$. Поэтому необходимо понять, обработка каких S-пар уже завершилась в рассматриваемый нами момент. Элементы P и $ToDo$ извлекаются в процедурах `AlgorithmF5` и `Reduction` в порядке возрастания сигнатур. S-пары с сигнатурами, меньшими $\mathcal{S}(g)$, могут быть разделены на 3 класса:

- S-пары с сигнатурой w, такой что $\mathrm{index}(w) > \mathrm{index}(\mathcal{S}(g)) = 1$. Они обрабатывались на предыдущих вызовах `AlgorithmF5`.
- S-пары с сигнатурой w, такой что $\mathrm{index}(w) = \mathrm{index}(\mathcal{S}(g)) = 1, \deg(w) < \deg(\mathcal{S}(g))$. Они обрабатывались на предыдущих итерациях цикла внутри того же вызова `AlgorithmF5`, который обрабатывает g.
- S-пары с сигнатурой w, такой что $\mathrm{index}(w) = \mathrm{index}(\mathcal{S}(g)) = 1, \deg(w) = \deg(\mathcal{S}(g)), w \prec \mathcal{S}(g)$. Они обрабатывались на предыдущих итерациях цикла внутри того же вызова `Reduce`, который обрабатывает g.

S-пары из этих классов не могут находиться в середине пути обработки, потому что в рассматриваемом состоянии алгоритма обработка только что завершена для g, поэтому ни P ни $ToDo$ не содержат необработанных элементов с сигнатурами, меньшими $\mathcal{S}(g)$. Осталось показать, что для всех S-пар из утверждения теоремы обработка начиналась хотя бы один раз. Это просто проверить для первых двух классов: обработка соответствующих S-пар была начата по крайней мере один раз путём вызова `CritPair` в `AlgorithmF5` непосредственно перед тем, как наибольший из порождающих S-пару был добавлен в G. Для S-пар третьего класса ситуация зависит от полной степени её порождающих. Если оба порождающих S-пары имеют полную степень $< \deg(g)$, то её обработка была начата в `CritPair` аналогично S-парам первых двух классов. Но некоторые S-пары третьего класса могут иметь старший по сигнатуре порождающий r_l, такой что $\deg(r_l) = \deg(g)$, $\mathcal{S}(r_l) \prec \mathcal{S}(g)$. Они обрабатываются вторым из рассмотренных путей обработки S-пар, поэтому обработка таких S-пар ещё не стартовала к моменту последнего вызова `Reduction`. Однако, их обработка начинается внутри `Reduction` до изучаемого нами момента: процедура выбирает многочлены из $ToDo$ в порядке возрастания сигнатуры, поэтому r_l редуцируется до g и в процессе редукции r_l непосредственно перед добавлением r_l в $Done$ вызов `TopReduction` начинает обработку всех таких S-пар.
Понятие *удовлетворять критерию F5* и *удовлетворять критерию Перезаписи* могут быть расширены но произвольные умноженные на моном отмеченные многочлены sh, $h \in G_g$:

Определение 9. Умноженный на моном отмеченный многочлен sr_i, $r_i \in G_g$ называется *удовлетворяющим критерию F5*, если $\varphi_{index(r_i)+1}(s\mathcal{S}(r_i)) \neq s\mathcal{S}(r_i)$, где $\varphi_{index(r_i)+1}$ – оператор нормальной формы по отношению к $G_{index(r_i)+1}$.

Это определение эквивалентно тому, что sr_i является не-нормализованным отмеченным многочленом с точки зрения определения 2 в части 5 работы [1].

Определение 10. Умноженный на моном отмеченный многочлен sr_i, $r_i \in G_g$ называется *удовлетворяющим критерию Перезаписи*, если $\exists j > i$ такое что $\mathcal{S}(r_j) | s\mathcal{S}(r_i)$.

В случае, если sr_i является частью S-пары, эти определения эквиваленты проверкам, производимым в алгоритме, в том смысле, что часть S-пары отбрасывается алгоритмом тогда и только тогда, когда она удовлетворяет данному определению как умноженный на моном отмеченный многочлен. Для обоих критериев выполняется важное свойство, утверждающее, что если sr_i удовлетворяет критерию, то то и дополнительно домноженный многочлен $s_1 sr_i$ также ему удовлетворяет.

5 Представления

5.1 Определение

Идея представлений, определённых ниже, приходит из [10], где подобный метод используется в доказательстве Теоремы 21. Представления используются для описания способов, которыми многочлен p может быть записан как элемент идеала (G_g). Одно представление соответствует записи отмеченного многочлена p в виде конечной суммы вида

$$p = \sum_k m_k \cdot b_{i_k},\ b_{i_k} \in G_g \tag{5.1}$$

с коэффициентами $m_k = c_k t_k \in \mathcal{K} \times T$.

Определение 11. Сумма вида 5.1, в которой все пары (t_k, b_{i_k}) различны, называется G_g-*представлением* p. Символические произведения $m_k \cdot b_{i_k}$ называются *элементами* представления. Если рассмотреть это символическое произведение как умножение многочленов, то мы получим отмеченный многочлен $m_k b_{i_k}$, соответствующий элементу представления. Тогда p оказывается равным сумме отмеченных многочленов, соответствующих элементам его представления. Понятие *сигнатура элемента* будет использоваться для обозначения сигнатура многочлена, соответствующего элементу. Два представления равны, если наборы их элементов совпадают как множества.

Большинство представлений, которые будут нам интересны также имеют следующее свойство, ограничивающее сигнатуру элементов:

Определение 12. G_g-представление p называется *сигнатурным*, если $\forall k\, \mathcal{S}(m_k b_{i_k}) \preccurlyeq \mathcal{S}(p)$.

5.2 Примеры

Пример 5. Первый важный пример G_g-представления тривиален: отмеченный многочлен из G_g равен сумме одного слагаемого – самого себя с единичным коэффициентом:

$$b_j = 1 \cdot b_j.$$

Это G_g-представление сигнатурно. Запрет на наличие в представлении нескольких элементов, имеющих одинаковый моном t_k и многочлен b_{i_k} гарантирует, что все элементы представления, отличающиеся только коэффициентом из поля c_k скомбинированы вместе путём суммирования коэффициентов из поля. Поэтому выражения вида $b_j = -1 \cdot b_j + 2 \cdot b_j$ и $b_j = 2x \cdot b_k + 1 \cdot b_j - 2x \cdot b_k$ не являются корректными G_g-представлениями.

Пример 6. Отмеченный многочлен $b_j \in G_g$, домноженный на произвольный многочлен h также имеет простое G_g-представление, получающееся из записи h в виде суммы термов: $h = \sum_k m_k$, $m_k \in \mathcal{K} \times T$. Это G_g-представление имеет форму

$$b_j h = \sum_k m_k \cdot b_j \tag{5.2}$$

и также является сигнатурным.

Отмеченный многочлен может иметь бесконечное множество различных представлений: к любому представлению можно добавить элементы, соответствующие сизигии, сгруппировать элементы с одновременно равными мономами и многочленами и получить другое корректное представление. Это будет представление того же многочлена, поскольку сумма многочленов, соответствующих элементам сизигии равна 0.

Пример 7. Произведение двух многочленов из G_g имеет 2 представления вида (5.2), которые отличаются прибавлением сизигии:

$$b_j b_i = \sum_k m_{i_k} \cdot b_j = \sum_k m_{i_k} \cdot b_j + 0 = \sum_k m_{i_k} \cdot b_j + \left(\sum_k m_{j_k} \cdot b_i - \sum_k m_{i_k} \cdot b_j \right) = \sum_k m_{j_k} \cdot b_i,$$

где m_{i_k} – термы b_i, а m_{j_k} – термы b_j.

Пример 8. Нулевой многочлен имеет пустое представление и представление, соответствующее каждой сизигии:

$$0 = \sum_{\emptyset} (\text{empty sum}) = \sum_k m_{j_k} \cdot b_i + \sum_k (-m_{i_k}) \cdot b_j,$$

где m_{i_k} и m_{j_k} взяты из предыдущего примера. Все непустые представления нулевого многочлена не сигнатурны. Другой важный пример G_g-представлений получается из определений сигнатуры и идеала. Все отмеченные многочлены, вычисляемые алгоритмом, принадлежат идеалу $(f_1, \ldots, f_m)$. Поэтому любой отмеченный многочлен p может быть записан в виде $\sum_i f_i g_i$, где g_i – однородные многочлены. Все входные многочлены f_i принадлежат G_g, поэтому $f_i g_i$ имеют G_g-представления вида (5.2).

Пример 9. Сумма этих представлений даёт следующее сигнатурное представление:

$$p = \sum_k m_k \cdot b_{i_k},\ m_k \in \mathcal{K} \times T,\ b_{i_k} \in \{f_1, \ldots, f_m\} \subset G_g.$$

Определение 13. Частный случай G_g-представления, при котором b_{i_k} ограничены лишь входными многочленами, будет называться *входным представлением.*

Входные представления всегда имеют единственный элемент максимальной сигнатуры. Произвольные G_g-представления не всегда обладают этим свойством, поскольку могут иметь несколько разных элементов с одинаковой максимальной сигнатурой, так как для них возможна ситуация $m_1 \mathcal{S}(b_{i_1}) = m_2 \mathcal{S}(b_{i_2})$ при $i_1 \neq i_2$. Следующее утверждение устанавливает связь между входными представлениями и понятием сигнатуры:

Утверждение 10. Допустимый отмеченный многочлен p с известной сигнатурой $\mathcal{S}(p)$ имеет входное представление, состоящее из элемента $c\mathcal{S}(p) \cdot f_{index(p)}$ и некоторых других элементов с меньшими сигнатурами.

Доказательство. Утверждение вытекает из определения допустимого многочлена в [1], ссылающегося на функцию v, которая соответствует суммированию элементов входного представления.

Теорема 1 из [1] утверждает, что все многочлены, порождаемые алгоритмом, допустимы, поэтому мы будем применять предыдущее утверждение ко всем таким многочленам.

Пример 11. Последний пример восходит к S-парам с известным G_g-представлением. S-многочлен, порождённый b_{l^*} и b_{l_*} из G_g, имеет вид $p = \overline{u_l}\mathrm{poly}(b_{l^*}) - \underline{u_l}\mathrm{poly}(b_{l_*})$. Из процесса редукции вытекает, что p сигнатурно редуцируется и результат добавляется в G_g как некоторый отмеченный многочлен b_l. Отсюда его G_g-представление имеет вид:

$$p = \sum_k m_k \cdot b_{n_k} + 1 \cdot b_l,$$

где сигнатуры элементов $m_k \cdot b_{n_k}$ меньше чем $\mathcal{S}(b_l) = \mathcal{S}(p)$. Значение l – позиция b_l в упорядоченном списке G_g. В данном представлении l больше чем l^* и l_*, поскольку соответствующий отмеченный многочлен b_l был добавлен в R в момент подсчёта S-многочлена в процедурах `Spol` или `TopReduction`, поэтому многочлены b_{l^*} и b_{l_*}, использованные для его создания не отброшенной S-пары, уже присутствовали в R к тому моменту, а порядок G_g соответствует порядку добавления элементов в R.

5.3 Порядок на представлениях

Определение 14. Для введения порядка на G_g-представлениях мы начнём с *порядка на элементах представления* $\succcurlyeq_1$: будем говорить, что $c_i t_i \cdot b_i \succcurlyeq_1 c_j t_j \cdot b_j$ если выполняется один из следующих случаев:

- $t_i\mathcal{S}(b_i) \succ t_j\mathcal{S}(b_j)$
- $t_i\mathcal{S}(b_i) = t_j\mathcal{S}(b_j)$ и $i < j$ (сравнение позиций – в обратную сторону).

Этот порядок основан лишь на сравнение сигнатур и позиций отмеченных многочленов в упорядоченном списке G_g, но не зависит от коэффициента из поля. Единственный случай, в котором два элемента не могут быть упорядочены – одновременное равенство сигнатур $t_i\mathcal{S}(b_i) = t_j\mathcal{S}(b_j)$ и позиций в списке $i = j$. Равенство позиций означает равенство многочленов $b_i = b_j$, что вместе с равенством сигнатур даёт $t_i = t_j$. Поэтому любые два элемента, принадлежащие одному корректному G_g представлению являются сравнимыми с точки зрения порядка $\preccurlyeq_1$, поскольку они имеют различные (t_k, b_k) по определению. Ниже приведены некоторые примеры сравнений элементов по $\preccurlyeq_1$ для списка $G_g = \{b_1, b_2, b_3\}$ из 3-х элементов с порядком на сигнатурах $x\mathbf{F}_i \succ y\mathbf{F}_i$ и сигнатурами $\mathcal{S}(b_1) = \mathbf{F}_1, \mathcal{S}(b_2) = \mathbf{F}_2, \mathcal{S}(b_3) = x\mathbf{F}_1$.

- $y \cdot b_1 \succcurlyeq_1 100y \cdot b_2$ поскольку сигнатура левой части $\succ$
- $x \cdot b_1 \succcurlyeq_1 y \cdot b_1$ поскольку сигнатура левой части $\succ$
- $-x \cdot b_1$ и $2x \cdot b_1$ не сравнимы, поскольку сигнатуры и позиции в списке равны
- $y^2 \cdot b_1 \preccurlyeq_1 y \cdot b_3$ поскольку сигнатура левой части $\prec$
- $x^2 \cdot b_1 \succcurlyeq_1 x \cdot b_3$ поскольку сигнатуры равны, а позиция в списке отмеченного многочлена левой части равна 1, что меньше чем позиция многочлена правой части, равная 3.

Для расширения этого порядка на G_g-представления целиком будем рассматривать *упорядоченную форму* представления, состоящую из всех его элементов, записанных в $\succcurlyeq_1$-убывающий список. Эта форма может быть использована для проверки на равенство, поскольку представления равны тогда и только тогда, когда равны их упорядоченные формы.

Определение 15. При помощи упорядоченных форм введём *порядок на G_g-представлениях*: представление $\sum_k m'_k \cdot b_{i'_k}$ является $\preccurlyeq$-меньшим, чем $\sum_k m_k \cdot b_{i_k}$, если упорядоченные формы для первого и второго представления удовлетворяют лексикографически расширенному на формы отношению $\preccurlyeq_1$. Для особого случая, когда упорядоченные формы отличаются лишь длиной, более короткая форма будет называться $\preccurlyeq$-меньшей. Если наибольшие различные элементы в упорядоченных формах двух представлений отличаются лишь коэффициентов поля, то представления являются не сравнимыми.

Далее даны некоторые примеры этого порядка для того же, трёх-элементного списка G_g. Все G_g-представления записаны в упорядоченных формах:

- $x^2 \cdot b_1 + xy \cdot b_1 + y^2 b_1 \succcurlyeq x^2 \cdot b_1 + 100y^2 \cdot b_1$ поскольку $xy \cdot b_1 \succcurlyeq_1 y^2 \cdot b_1$
- $x^2 \cdot b_1 + 100y^2 \cdot b_1 \succcurlyeq x^2 \cdot b_1$ поскольку правая часть является началом левой
- $x^2 \cdot b_1 \succcurlyeq xy \cdot b_1 + y^2 \cdot b_1 + x^2 \cdot b_2$ поскольку $x^2 \cdot b_1 \succcurlyeq_1 xy \cdot b_1$
- $xy \cdot b_1 + y^2 \cdot b_1 + x^2 \cdot b_2 \succcurlyeq y \cdot b_3 + y^2 \cdot b_1 + x^2 \cdot b_2$ поскольку $xy \cdot b_1 \succcurlyeq_1 y \cdot b_3$

- $y \cdot b_3 + y^2 \cdot b_1 + x^2 \cdot b_2$ и $2y \cdot b_3 + y^2 \cdot b_2$ не сравнимы, поскольку наибольшие различные элементы – это $y \cdot b_3$ и $2y \cdot b_3$.

Порядок на представлениях совместим с понятием сигнатурности представления:

Теорема 16. *Если для пары представлений отмеченного многочлена p выполняется отношение* $\sum_k m'_k \cdot b_{i'_k} \lessdot \sum_k m_k \cdot b_{i_k}$ *и правое представление сигнатурно, то и левое представление сигнатурно.*

Доказательство. Теорема легко следует из того, что сигнатуры элементов $\lessdot$-меньшего представления не могут быть $\succ$-больше, чем максимальная сигнатура $\geqslant$-большего представления.

Важным фактом, позволяющим брать $\lessdot$-минимальный элемент, является вполне упорядоченность:

Теорема 17. *Представление вполне упорядочены порядком* $\lessdot$.

Доказательство. Количество различных отмеченных многочленов, входящих в элементы представления, конечно, поскольку оно равно $|G_g|$ для некоторого фиксированного g. Поэтому существование бесконечной $\geqslant_1$-убывающей последовательности элементов представлений привело бы к существованию бесконечной $\succ$-убывающей последовательности сигнатур. Учитывая вполне упорядоченность сигнатур по $\prec$ мы получаем доказательство мы получаем доказательство вполне упорядоченности элементов представлений по $\lessdot_1$. Прямое доказательство вполне упорядоченности представлений по $\lessdot$ на основе вполне упорядоченности их элементов по $\lessdot_1$ не является очень сложным, однако дабы не вдаваться в детали, используем теорему 2.5.5 книги [12]. Она утверждает вполне упорядоченность конечных наборов с порядком, являющимся лексикографическим расширением порядка на элементах набора. Это применимо к предмтавлениямм, так как они являются наборами своих элементов.

5.4 Последовательность представлений

Идея этой части состоит в построении конечной последовательности строго $\lessdot$-убывающих сигнатурных G_g-представлений для некоторого отмеченного многочлена mh, $m \in \mathcal{K} \times T$, $h \in G_g$ со свойством $\mathcal{S}(mh) \prec \mathcal{S}(g)$. Первым сигнатурным представлением в последовательности является $mh = m \cdot h$, а последнее представление имеет вид $mh = \sum_k m_k \cdot b_{i_k}$ со свойствами, выполняющимися для $\forall k$:

1. $m_k b_{i_k}$ не удовлетворяет критерию F5.
2. $m_k b_{i_k}$ не удовлетворяет критерию Перезаписи.
3. $\mathrm{HM}(m_k b_{i_k}) \leqslant \mathrm{HM}(mh)$

Доказательство существования такой последовательности довольно похоже на доказательство Теоремы 21 из [10] и основано на том, что если некоторое сигнатурное представление mh содержит не удовлетворяющий одному из свойств элемент $m_K \cdot b_{i_K}$, то может быть найдено $\lessdot$-меньшее представление. Метод построения отличается для разных свойств, однако схема замены одинакова:

- выбирается некоторый элемент $m_{K'} \cdot b_{i_{K'}}$ в представлении mh. Следует отметить, что K' может как совпадать, так и отличаться от K
- строится некоторое представление $m_{K'} b_{i_{K'}} = \sum_l m_l \cdot b_{i_l}$ для этого элемента
- показывается, что построенное представление $\lessdot$-меньше, чем представление $m_{K'} b_{i_{K'}} = m_{K'} \cdot b_{i_{K'}}$

Построение такого представления для $m_{K'} \cdot b_{i_{K'}}$ позволяет применить следующую лемму:

Лемма 18. *Если элемент* $m_{K'} \cdot b_{i_{K'}}$ *сигнатурного представления* $mh = \sum_k m_k \cdot b_{i_k}$ *имеет представление* $m_{K'} b_{i_{K'}} = \sum_l m_l \cdot b_{i_l}$, $\lessdot$*-меньшее чем представление* $m_{K'} b_{i_{K'}} = m_{K'} \cdot b_{i_{K'}}$, *то* mh *имеет сигнатурное представление,* $\lessdot$*-меньшее, чем* $mh = \sum_k m_k \cdot b_{i_k}$.

Доказательство. Заменим элемент $m_{K'} \cdot b_{i_{K'}}$ в представлении $mh = \sum_k m_k \cdot b_{i_k}$ на $\sum_l m_l \cdot b_{i_l}$ и скомбинируем коэффициенты при элементах с одновременно одинаковыми мономами и многочленами, получив таким образом изменённое представления для mh. Оно будет $\lessdot$-меньше, чем $mh = \sum_k m_k \cdot b_{i_k}$ поскольку: все элементы $\geqslant_1$-большие $m_{K'} \cdot b_{i_{K'}}$ идентичны в исходном и изменённом представлениях; элемент $m_{K'} \cdot b_{i_{K'}}$ содержится в исходном, но не в изменённом представлении; все остальные элементы в представлениях являются $\lessdot_1$-меньшими, чем $m_{K'} \cdot b_{i_{K'}}$ поэтому они не играют роли при сравнении. Сравнение выполняется даже в случае, когда при комбинировании коэффициентов все элементы обнулились. Этот случай может возникнуть, если исходное представление было равно $mh = m_{K'} \cdot b_{i_{K'}} + \sum_l (-m_l) \cdot b_{i_l}$, что привело к изменённому представлению вообще не содержащему элементов: $mh = 0$. Оно является $\lessdot$-меньшим, чем любое непустое представление.

Теперь покажем, что представление элемента по вышеуказанной схеме может быть построено, если исходное представление содержит элемент, не удовлетворяющий по крайней мере одному из свойств.

Лемма 19. *Если сигнатурное G_g-представление $mh = \sum_k m_k \cdot b_{i_k}$ содержит элемент, не удовлетворяющий свойству 1, то найдётся элемент $m_{K'} \cdot b_{i_{K'}}$, обладающий G_g-представлением $m_{K'} b_{i_{K'}} = \sum_l m_l \cdot b_{i_l}$ $\lessdot$-меньшим, чем представление $m_{K'} b_{i_{K'}} = m_{K'} \cdot b_{i_{K'}}$.*

Элемент, не обладающий первым свойством, удовлетворяет критерию F5, что позволяет позволяет использовать соображение о том, что $m_K \mathcal{S}(b_{i_K})$ не является его сигнатурой $m_K b_{i_K}$ аналогично теореме 20 работы [10]. Для этого случая берётся $K' = K$. **Доказательство.** Рассмотрим входное представление $m_K b_{i_K}$ с сигнатурой $\succcurlyeq_1$-максимального элемента, равной $m_K \mathcal{S}(b_{i_K}) = s_0 \mathbf{F}_{j_0}$:

$$m_K b_{i_K} = c_0 s_0 \cdot f_{j_0} + \sum_l m_l \cdot f_{i_l}. \tag{5.3}$$

Из удовлетворения критерия F5 следует, что s_0 может быть представлено как $s_0 = s_1 \mathrm{HM}(f_{j_1})$, $j_1 > j_0$, откуда $s_0 f_{j_0} = s_1 f_{j_0} f_{j_1} - s_1 (f_{j_1} - \mathrm{HM}(f_{j_1})) f_{j_0}$. На основе этого выражения можно получить другое представление для $m_K b_{i_K}$, обозначая за m_{0i} упорядоченные термы f_{j_0}, за m_{1i} – упорядоченные термы f_{j_1} и за N_0, N_1 число термов в этих многочленах:

$$m_K b_{i_K} = \sum_{i=1}^{N_0} c_0 s_1 m_{0i} \cdot f_{j_1} + \sum_{i=2}^{N_1} -c_0 s_1 m_{1i} \cdot f_{j_0} + \sum_l m_l \cdot f_{i_l}.$$

Это представление $\lessdot$-меньше, чем $m_K \cdot b_{i_K}$ поскольку сигнатуры всех его элементов меньше чем $s_0 \mathbf{F}_{j_0}$. Для элементов третьей суммы $\sum_l m_l \cdot f_{i_l}$ это следует из (5.3), где эти элементы являются меньшими элементами входного представления. Для элементов первой суммы $\sum_{i=1}^{N_0} c_0 s_1 m_{0i} \cdot f_{j_1}$ это следует из неравенства индексов сигнатур $j_1 > j_0$. А для второй суммы используем совпадение порядка на сигнатурах и на термах: все термы m_{1i}, $i \geqslant 2$ меньше, чем m_{11}, поэтому и выполняется неравенство для сигнатур: $s_1 m_{1i} \mathbf{F}_{j_0} \prec s_1 m_{11} \mathbf{F}_{j_0} = s_0 \mathbf{F}_{j_0}$.

Лемма 20. *Если сигнатурное G_g-представление $mh = \sum_k m_k \cdot b_{i_k}$ с $\mathcal{S}(mh) \prec \mathcal{S}(g)$ содержит элемент, не удовлетворяющий свойству 2, то найдётся элемент $m_{K'} \cdot b_{i_{K'}}$, обладающий G_g-представлением $m_{K'} b_{i_{K'}} = \sum_l m_l \cdot b_{i_l}$ $\lessdot$-меньшим, чем представление $m_{K'} b_{i_{K'}} = m_{K'} \cdot b_{i_{K'}}$.*

Для элементов, не удовлетворяющих критерию Перезаписи $\lessdot$-меньшее представление сроится методом, аналогичным Предложению 17 из [10]. В этом случае также берётся $K' = K$. **Доказательство.** Предположим, что сигнатура $\mathcal{S}(m_K b_{i_K}) = s_0 \mathbf{F}_{j_0}$ и она перезаписывается отмеченным многочленом $b_{i'}$ из R. Поскольку представление сигнатурно, мы имеем $\mathcal{S}(b_{i'}) \preccurlyeq s_0 \mathbf{F}_{j_0} \preccurlyeq \mathcal{S}(mh) \prec \mathcal{S}(g)$. Поэтому $b_{i'}$ обрабатывался в процедуре `TopReduction` раньше g. Значит $b_{i'}$ был редуцирован и результат его редукции или был нулевым или присутствует в G_g, поэтому он может использоваться как многочлен элемента G_g-представления. Из критерия Перезаписи известно, что $i' > i_K$ и существует $s' \in T$, такое что $s' \mathcal{S}(b_{i'}) = s_0 \mathbf{F}_{j_0}$. Для $m_K b_{i_K}$ есть входное представление (5.3) а для $s' b_{i'}$ входным представлением является:

$$s' b_{i'} = c' s_0 \cdot f_{j_0} + \sum_{l'} m_{l'} \cdot f_{i_{l'}}.$$

Преобразованием этого выражения получим G_g-представление для $c_0 s_0 f_{j_0}$:

$$c_0 s_0 f_{j_0} = c'^{-1} c_0 s' \cdot b_{i'} + \sum_{l'} -c'^{-1} c_0 m_{l'} \cdot f_{i_{l'}}.$$

Используя его для замены первого элемента в (5.3) получим желаемый результат:

$$m_K b_{i_K} = c'^{-1} c_0 s' \cdot b_{i'} + \sum_{l'} -c'^{-1} c_0 m_{l'} \cdot f_{i_{l'}} + \sum_l m_l \cdot f_{i_l}$$

Он $\lessdot$-меньше, чем $m_K b_{i_K} = m_K \cdot b_{i_K}$ поскольку элементы обоих сумм имеют сигнатуры, меньшие $s_0 \mathbf{F}_{j_0}$, а для первого элемента, если $b_{i'}$ не нулевой, имеем $\mathcal{S}(c'^{-1} c_0 s' \cdot b_{i'}) = \mathcal{S}(m_K \cdot b_{i_K}) = s_0 \mathbf{F}_{j_0}$ с $i' > i_K$, откуда применяя правило $\lessdot_1$-сравнения для равных сигнатур и различных позиций в списке, получаем, что элемент $c'^{-1} c_0 s' \cdot b_{i'}$ также $\lessdot_1$-меньше, чем $m_K \cdot b_{i_K}$. Если же $b_{i'} = 0$, то он отбрасывается, и желаемое представление состоит только из оставшихся 2-х сумм.

Лемма 21. *Если все элементы сигнатурного представления $mh = \sum_k m_k \cdot b_{i_k}$ с $\mathcal{S}(mh) \prec \mathcal{S}(g)$ удовлетворяют свойствам 1 и 2, но один из них не удовлетворяет свойству 3, то найдётся элемент $m_{K'} \cdot b_{i_{K'}}$, имеющий представление $m_{K'} b_{i_{K'}} = \sum_l m_l \cdot b_{i_l}$ $\lessdot$-меньшее, чем представление $m_{K'} b_{i_{K'}} = m_{K'} \cdot b_{i_{K'}}$.*

Доказательство. Существует по крайней мере один элемент $m_K \cdot b_{i_K}$, не удовлетворяющий свойству 3. Обозначим за $m_{\max}$ максимальный НМ отмеченных многочленов, соответствующих элементам представления, и за $H_{\max}$ список элементов, на котором $m_{\max}$ достигается. Возьмём в качестве K' номер элемента представления, являющегося $\succcurlyeq_1$-максимальным в $H_{\max}$. Имеем $\mathrm{HM}(m_{K'} b_{i_{K'}}) = m_{\max} \geqslant \mathrm{HM}(m_K b_{i_K}) > \mathrm{HM}(mh)$, поэтому НМ суммы всех элементов, кроме K'-го равен $\mathrm{HM}(mh - m_{K'} b_{i_{K'}}) = \mathrm{HM}(m_{K'} b_{i_{K'}}) = m_{\max}$, поэтому существует другой

элемент K'', обладающий $\mathrm{HM}(m_{K''}b_{i_{K''}}) = m_{\max}$. Отсюда $m_{K''} \cdot b_{i_{K''}} \in H_{\max}$ и $m_{K''} \cdot b_{i_{K''}} \preccurlyeq_1 m_{K'} \cdot b_{i_{K'}}$ поскольку $m_{K'} \cdot b_{i_{K'}}$ $\succcurlyeq_1$-максимален в $H_{\max}$. Равенство $\mathrm{HM}(m_{K''}b_{i_{K''}}) = \mathrm{HM}(m_{K'}b_{i_{K'}})$ означает, что S-пара $b_{i_{K'}}$ и $b_{i_{K''}}$ имеет вид $[m'^{-1}m_{\max}, m'^{-1}m_{K'}, b_{i_{K'}}, m'^{-1}m_{K''}, b_{i_{K''}}]$ где $m' = \gcd(m_{K'}, m_{K''})$. Пусть q – соответствующий S-многочлен. Тогда $m'\mathcal{S}(q) \preccurlyeq \mathcal{S}(mh) \prec \mathcal{S}(g)$ поскольку $m'\mathcal{S}(q) = \mathcal{S}(m_{K'}b_{i_{K'}})$ и представление сигнатурно. Части S-многочлена $m'^{-1}m_{K'}b_{i_{K'}}$ и $m'^{-1}m_{K''}b_{i_{K''}}$ не удовлетворяют критериям F5 и Перезаписи, поскольку они же, умноженные на m' равны $m_{K'}b_{i_{K'}}$ и $m_{K''}b_{i_{K''}}$ – отмеченным многочленам, соответствующим элементам, про которые известно, что они не удовлетворяют критериям по предположению леммы. Из $m'\mathcal{S}(q) \prec \mathcal{S}(g)$ имеем $\mathcal{S}(q) \prec \mathcal{S}(g)$, а значит можно применить теорему 8 и получить, что $(b_{i_{K'}}, b_{i_{K''}})$ – S-пара с известным G_g-представлением, что означает существование представления, описанного в примере 11:

$$q = 1 \cdot b_{i'} + \sum_l m_l \cdot b_{i_l},$$

причём выполняются свойства, показанные после примера: $\mathcal{S}(q) = \mathcal{S}(b_{i'})$, $\forall l\, \mathcal{S}(q) \succ \mathcal{S}(m_l b_{i_l})$ и $i' > K'$. С другой стороны по определению S-многочлена мы имеем $m'q = c_0 m_{K'} b_{i_{K'}} - c_1 m_{K''} b_{i_{K''}}$, откуда получаем следующее представление:

$$m_{K'}b_{i_{K'}} = c_0^{-1}c_1 m_{K''} \cdot b_{i_{K''}} + c_0^{-1}m' \cdot b_{i'} + \sum_l c_0^{-1}m'm_l \cdot b_{i_l}.$$

Оно $\lessdot$-меньше, чем $m_{K'}b_{i_{K'}} = m_{K'} \cdot b_{i_{K'}}$: $m_{K''} \cdot b_{i_{K''}}$ уже сравнивалось с $m_{K'} \cdot b_{i_{K'}}$ $m' \cdot b_{i'}$ имеет ту же сигнатуру, но больший номер позиции $i' > i_{K'}$ последняя сумма состоит из элементов с сигнатурами, меньшими чем $m'\mathcal{S}(b_{i'}) = \mathcal{S}(m_{K'} \cdot b_{i_{K'}})$.

Теорема 22. *Или сигнатурное представление $mh = \sum_k m_k \cdot b_{i_k}$ с $\mathcal{S}(mh) \prec \mathcal{S}(g)$ удовлетворяет свойствам 1-3 или существует другое сигнатурное представление $mh = \sum_l m_l \cdot b_{i_l}$ $\lessdot$-меньшее, чем $\sum_k m_k \cdot b_{i_k}$.*

Доказательство. Теорема немедленно следует из комбинации четырёх предыдущих лемм.

Это привод на с к следующему ключевому результату:

Теорема 23. *Для любого многочлена mh, $m \in \mathcal{K} \times T$, $h \in G_g$ с $\mathcal{S}(mh) \prec \mathcal{S}(g)$ существует удовлетворяющее свойствам 1-3 сигнатурное G_g-представление $mh = \sum_k m_k \cdot b_{i_k}$.*

Доказательство. Начнём с представления $mh = m \cdot h$ и будем заменять текущее представление на $\lessdot$-меньшее из теоремы 22 до тех пор пока текущее представление не будет удовлетворять свойствам 1-3. Конечность процесса гарантируется вполне упорядоченностью представлений по $\lessdot$.

Этот результат может представлять интерес сам по себе, однако для целей доказательства остановки нужно лишь одно следствие из него:

Вывод 12. *Рассмотрим произвольный многочлен f без ограничений на его сигнатуру. Если существует сигнатурный редуктор $f' \in G_g$ для f с $\mathcal{S}(f')\frac{\mathrm{HM}(f)}{\mathrm{HM}(f')} \prec \mathcal{S}(g)$ то G_g содержит сигнатурный редуктор для f, который не отбрасывается критериями F5 и перезаписи.*

Доказательство. Пусть mf', $m = \frac{\mathrm{HM}(f)}{\mathrm{HM}(f')} \in \mathcal{K} \times T$, $f' \in G_g$ умноженный редуктор с $\mathcal{S}(mf') \prec \mathcal{S}(g)$. Из предыдущей теоремы мы можем найти удовлетворяющее свойствам 1-3 представление $mf' = \sum_k m_k \cdot b_{i_k}$. Свойство 3 означает отсутствие элементов с HM, большим чем у mf', откуда, поскольку сумма всех элементов этого представления имеет HM равный $\mathrm{HM}(mf')$, существует элемент K, на котором достигается равенство HM: $\mathrm{HM}(m_K \cdot b_{i_K}) = \mathrm{HM}(mf') = \mathrm{HM}(f'_1)$. Поскольку представление сигнатурно, имеем $\mathcal{S}(m_K \cdot b_{i_K}) \preccurlyeq \mathcal{S}(mf') \prec \mathcal{S}(f)$, а значит $m_K b_{i_K}$ – сигнатурный редуктор для f и выполнение свойств 1-2 гарантирует, что $m_K b_{i_K}$ не отбрасывается критериями.

6 Обнаружение противоречия с учётом критериев

Теперь мы можем вернуться к результаты теоремы 7, утверждающей, что в случае. если алгоритм не останавливается, существуют многочлены $f', f \in G$, такие что $\mathrm{HM}(f')|\mathrm{HM}(f)$, $\frac{\mathrm{HM}(f')}{\mathcal{S}(f')} >_q \frac{\mathrm{HM}(f)}{\mathcal{S}(f)}$. Используя этот результат и последнее следствие мы построим два многочлена, приводящие к противоречию в случае отсутствия остановки алгоритма.

Теорема 24. *Если алгоритм не останавливается на некоторых входных данных, то найдётся шаг, после которого конечное множество $G \cup Done$ содержит пару отмеченных многочленов f'_1, f, для которых выполняется:*

- *f'_1 было добавлено в $G \cup Done$ до f*
- *$t_1 f'_1$ не удовлетворяет критериям F5 и Перезаписи, где $t_1 = \frac{\mathrm{HM}(f)}{\mathrm{HM}(f'_1)}$*

- f_1' – *сигнатурный редуктор для* f.

Доказательство. Пусть f', f – многочлены из теоремы 7 и $t = \frac{\mathrm{HM}(f)}{\mathrm{HM}(f')}$. Многочлен $f \in G$, поэтому построенная выше теория о представлениях может быть рассмотрена применительно к случаю g равного f, и можно говорить о множестве G_f и G_f-представлениях. Поскольку tf' – сигнатурный редуктор для f – мы имеем $\mathcal{S}(f')t \prec \mathcal{S}(f)$ и следствие 12 может быть применено для нахождения сигнатурного редуктора $t_1 f_1'$ для f, который не удовлетворяет критериям. Помимо этого известно, что он принадлежит G_f, поэтому выполняется и первое свойство: в процессе работы алгоритма f_1' был добавлен в $G \cup Done$ раньше f.

Теорема 25. *Алгоритм F5, описанный в [1], останавливается на любых входных данных.*

Доказательство. Покажем, что существование многочленов f_1', f из теоремы 24 приводит к противоречию. Рассмотрим вызов `TopReduction` после которого многочлен f был добавлен в $Done$. Этот вызов вернул многочлен f как первую половину значения, возвращаемого `TopReduction`, что означает что предшествующий вызов `IsReducible` вернул пустое множество. Это значит, что для каждого многочлена в $G \cup Done$, в том числе f_1', хотя бы одно из условий (a) - (d) не выполнилось. Это невозможно, поскольку:

- (a) удовлетворяется, так как f_1' – редуктор для f по теореме 24
- (b) и (c) удовлетворяются, поскольку $\frac{\mathrm{HM}(f)}{\mathrm{HM}(f_1')} f_1'$ не удовлетворяет критериям F5 и Перезаписи по теореме 24
- (d) удовлетворяется, поскольку f_1' – сигнатурный редуктор для f по теореме 24.

7 Выводы

Данная работа показывает, что исходный алгоритм F5 завершается на любых однородных входных данных, не вводя промежуточных алгоритмов. При этом не даётся никакого ограничения на количество операций. Простейшее доказательство остановки алгоритма Бухбергера основано на свойстве Нётеровости и тоже не даёт такого ограничения. Однако доказательство остановки, приведённое здесь, значительно отличается по структуре от доказательства остановки алгоритма Бухбергера, поэтому не может быть использовано для сравнения эффективности алгоритма F5 с алгоритмом Бухбергера. В отличие от этого, остановка изменённых вариаций алгоритма F5 в работах [3, 4, 5] показывается методом, близким к доказательству остановки алгоритма Бухбергера, за счёт чего остаётся возможность сравнения их эффективности с алгоритмом Бухбергера. С точки зрения практической компьютерной алгебры существует вопрос эффективности изменённых вариаций по сравнению с исходным F5. Изменённые вариации могут проводить больше времени в дополнительных проверках, инициирующих остановку. Но в некоторых случаях возможна ситуация, когда за счёт этих проверок остановка в изменённых версиях инициируется раньше, чем в исходной, за счёт чего изменённые версии проводят меньше редукций. Поэтому становится возможным, что на некоторых входных данных быстрее оказывается исходный алгоритм, а на других – модифицированный. Экспериментально измеренные времена работы в Таблице 1 работы [3] показывают, что оба случая действительно встречаются на практике, но разница во временах работы незначительна. Поэтому, хотя данная работа и показывает, что останавливаются не только модифицированные версии, но и исходный алгоритм, вопрос о сравнения эффективности исходной и модифицированной версий F5 остаётся открытым. В доказательстве используются три свойства исходного алгоритма F5, которые отсутствуют или опциональны в других F5-подобных алгоритмах: однородность входных многочленов, наличие критерия перезаписи и совпадения порядка на мономах $<$ с порядком на сигнатурах $\prec$. Возможность расширения доказательства на F5-подобные алгоритмы, не обладающие этими свойствами, остаётся открытым. Автор предполагает, что доказательство может быть модифицировано таким образом, что зависимость от первых двух свойств исчезнет, однако от третьего избавиться не получится, поскольку оно является ключевым моментом при получении противоречия на основе результата теоремы 7. Автор благодарит Christian Eder, Jean-Charles Faugère, Amir Hashemi, John Perry, Till Stagers и Алексея Зобнина за их работы и комментарии, воодушевившие автора на исследования в этой области. Спасибо!

Список литературы

[1] *Faugère J. C.* A new efficient algorithm for computing Gröbner bases without reduction to zero (F5) // Proceedings of the 2002 international symposium on Symbolic and algebraic computation. 2002. 75–83.

[2] *Stegers T.* Faugere's F5 Algorithm Revisited. // IACR Cryptology ePrint Archive. 2006. **404**.

[3] *Eder C., Gash J., Perry J.* Modifying Faugère's F5 Algorithm to ensure termination // ACM Commun. Comput. Algebra. 2011. **45**, №1/2. 70–89.

[4] *Ars G.* Applications des bases de Gröbner à la cryptographie. Rennes, France: Université de Rennes 1, 2005.

[5] *Gash J. M.* On efficient computation of Gröbner bases. Indianapolis, IN, USA: Indiana University, 2008.

[6] *Hashemi A., Ars G.* Extended F5 criteria // J. Symb. Comput. 2010. **45**, № 12. 1330–1340.

[7] *Зобнин А. И.* Обобщение алгоритма F5 вычисления базиса Грёбнера полиномиальных идеалов // Программирование. 2010. № 2. 21–30.

[8] *Pan S., Hu Y., Wang B.* The Termination of Algorithms for Computing Gröbner Bases // ArXiv e-prints. 2012. **1202**, №3524.

[9] *Huang L.* A new conception for computing Gröbner basis and its applications // ArXiv e-prints. 2010. **1012**, №5425.

[10] *Eder C., Perry J.* F5C: a variant of Faugère's F5 algorithm with reduced Gröbner bases // J. Symb. Comput. 2010. **45**, № 12. 1442–1458.

[11] *Arri A., Perry J.* The F5 Criterion revised // J. Symb. Comput. 2011. **46**, № 9. 1017–1029.

[12] *Baader F., Nipkow T.* Term Rewriting and All That. United Kingdom: Cambridge University Press, 1998.