

# ОСТАНОВКА АЛГОРИТМА F5

ВАСИЛИЙ ГАЛКИН

Аннотация. 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. Схема доказательства состоит из двух частей: в первой части показано, что если алгоритм не останавливается, то рано или поздно он получит пару многочленов, в которой первый может редуцировать второй. При этом, однако, не утверждается, что такая редукция будет разрешена критериями, предложенными в F5. Вторая часть показывает, что при существовании такой пары также будет существовать пара в которой редукция разрешена всеми критериями. Существование такой пары приводит к противоречию.

## 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 [6] и детальные исследования в [11] 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 [5, 2, 7, 8, 12].

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, [10] 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 возрастанию сигнатуры. Второе отличие – отсутствие в TRB-F5 применения оператора нормальной формы  $\varphi$  перед редукцией, что приводит к эффекту противоположному отличиям F5 от F5GEN: алгоритм TRB-F5 при выборе редуктора проверяет критерии для элементов с большими 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 [4] для доказательства корректности алгоритма F5C: представление S-многочлена в виде 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 [6], 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  $\{f_1, \dots, f_m\}$  в кольце  $\mathcal{K}[x_1, \dots, x_n]$  для которой алгоритм F5 не завершается. Без ограничения общности будем считать что это самая короткая последовательность такого рода – алгоритм завершается на более короткой последовательности  $\{f_2, \dots, f_m\}$ . 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:  $d_j \geq d_{j-1}$ . 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  $\frac{HM(r_k)}{HM(r_{3-k})}$ , 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  $u_{3-k}S(r_{3-k})$ . Для нашего случая отсюда следовало бы, что в функции **CritPair** эквивалентная проверка  $\varphi(u_{3-k}S(r_{3-k})) = u_{3-k}S(r_{3-k})$  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 [6] несколько неясен в этом месте, но исходный код процедуры **FindReductor**, приложенный к [11] 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
  // discard reductor by criterion (d)
  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.  $\square$

## 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  $f_2, \dots, f_m$ , 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  $\leq d_j$  и полную степень монома сигнатуры  $\leq d_j - \deg(f_1)$ . Также это даёт информацию о порядке элементов *Rule* с сигнатурой 1: их полная степень не убывает.

**Определение 3.** 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  $t = \frac{\text{HM}(r_k)}{\text{HM}(r_m)}$  – 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  $G \cup \text{Done}$  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 случаю неограниченного возрастания  $d_j$ . Мы показали, что отсутствие остановки алгоритма происходит в случае, когда не завершается вызов **AlgorithmF5**, и что он не может заиклиться обрабатывая бесконечное число итераций итерации с одним и тем же значением  $d$ . Остаются два варианта: неограниченное возрастание  $d$  и заикливание внутри одной из итераций с фиксированным значением. Далее показано, что такое заикливание 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 НМ 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.  $\square$

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

*Утверждение 4.* Если алгоритм не завершается на некоторых входных данных, то значение  $d$  неограниченно возрастает в процессе итераций.

*Доказательство.* Следует из комбинирования утверждений 1 и 2.  $\square$

### 3. ЦЕПИ S-ПАР

Утверждение 4 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  $\{r_1, r_2, \dots, r_m, \dots, r_l, \dots\}$ , в которой  $r_1, \dots, r_m$  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  $\bar{u}_l, u_l$  мономы, на которые они умножались. При этом  $l^*$  соответствует части с большей сигнатурой:  $\text{poly}(r_l) = \bar{u}_l \text{poly}(r_{l^*}) - u_l \text{poly}(r_{l_*})$  и  $\mathcal{S}(r_l) = \bar{u}_l \mathcal{S}(r_{l^*}) \succ u_l \mathcal{S}(r_{l_*})$ . Значение  $\text{poly}(r_l)$  может меняться в процедуре **TopReduction** на многочлен с меньшим НМ, но  $\mathcal{S}(r_l)$  нигде далее не меняется после добавления многочлена в последовательность. Далее, будем пытаться найти бесконечную подпоследовательность  $\{r_{k_1}, r_{k_2}, \dots, r_{k_n}, \dots\}$  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  $\mathcal{S}(r_{k_n}) = \bar{u}_{k_n} \mathcal{S}(r_{k_{n-1}})$  и

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

**Определение 5.** 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 элементарного  $r_l$ , 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  $\{r_q, \dots, r_{l^*}, r_{l_*}, r_l\}$ , где все многочлены имеют одинаковый индекс сигнатуры  $q = \text{index}(r_l)$  и первый элемент является входным многочленом этого индекса.

Первое свойство цепей S-пар основано на критерии перезаписи и заключается в следующей теореме.

**Теорема 6.** *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  $\{r_{l_1}, \dots, r_{l_i}, \dots\}$  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  $\forall i > i_0 \exists j \leq i_0$  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  $r_L$ .  $\square$

**Определение 7.** Конечное множество концов цепей S-пар длины 2, начинающихся с  $r_L$  будет называться *множеством S-порождённых  $r_L$* .

**Теорема 8.** *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 9.** 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, порождаемый в последнем не завершающемся вызове `AlgorithmF5` имеет индекс сигнатуры 1 и является концом цепи S-пар  $\{r_1, \dots, r_{l^*}, r_l\}$ .

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.  $\square$

Для следующей теоремы необходимо ввести порядок на частных, образованных мономах, 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 10.** 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:

$$\text{HM}(f') | \text{HM}(f),$$

$$\frac{\text{HM}(f')}{\mathcal{S}(f')} >_q \frac{\text{HM}(f)}{\mathcal{S}(f)},$$

$$\mathcal{S}(f') | \mathcal{S}(f).$$

*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 многочлену  $h_{n+1}$  в цепи S-пар, сохраняет одно и то же значение  $\text{poly}(h_n)$  после того как был использован для создания какой-либо S-пары. и можно утверждать, что

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

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

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

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

в отличие от порядка на мономах, не даёт вполне упорядоченности: к примеру последовательность  $\frac{x}{x} >_q \frac{x}{x^2} >_q \dots >_q \frac{x}{x^n} >_q \dots$  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  $\text{НМ}(h_{n+1}) < \text{НМ}(h_n)$  при равенстве полной степени, а во втором –  $\text{НМ}(h_{n+1}) > \text{НМ}(h_n)$ , поскольку полные степени НМ отличаются. Поэтому, последовательность НМ элементов бесконечной цепи S-пар состоит из блоков с фиксированной полной степенью, где НМ внутри блока строго убывают. Длты 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  $\text{НМ}(h_i) | \text{НМ}(h_j)$  возможно только в случае  $i < j$  и  $\deg(h_i) < \deg(h_j)$ .

Это позволяет использовать метод аналогичный используемому в Предложении 14 работы [1]: рассмотрим НМ бесконечной цепи S-пар  $\{h_i\}$ . Они порождают бесконечную последовательность в  $T$ , поэтому по лемме Диксона существует два многочлена, с  $\text{НМ}(h_i) | \text{НМ}(h_j)$ . Из предыдущего абзаца следует, что при этом  $i < j$ , а при помощи свойств цепи S-пар мы получаем, что  $\mathcal{S}(h_i) | \mathcal{S}(h_{i+1}) | \dots | \mathcal{S}(h_j)$  и  $\frac{\text{НМ}(h_i)}{\mathcal{S}(h_i)} >_q \frac{\text{НМ}(h_{i+1})}{\mathcal{S}(h_{i+1})} >_q \dots >_q \frac{\text{НМ}(h_j)}{\mathcal{S}(h_j)}$ , поэтому можно взять  $f' = h_i$  и  $f = h_j$ .  $\square$

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

*Доказательство.* Данное выше доказательство теоремы 10 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 мономах из теоремы 10 в отношении на сигнатурах:

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

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  $f'$ .  $\square$

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-PAIRS WITH SIGNATURES SMALLER THAN $\mathcal{S}(g)$

In this and following sections  $g$  is treated as some fixed labeled polynomial with signature index 1 added to *Done* in some algorithm iteration. Let us work with algorithm state just before adding  $g$  to *Done* during call to *AlgorithmF5* with  $i = 1$ . Consider a finite set  $G_1 \cup \text{Done}$  of labeled polynomials at that moment. This set contains positions of labeled polynomials in  $R$ , so it's elements can be ordered according to position in  $R$  and written as an ordered integer sequence



$G_g = \{b_1, \dots, b_N\}$  with  $b_j < b_{j+1}$ . It should be noted that this order does correspond to the order of labeled polynomials in the sequence produced by concatenated rule arrays  $Rule[m] : Rule[m-1] : \dots : Rule[1]$  because addition of new polynomial to  $R$  is always followed by addition of corresponding rule. But this order may differ from the order polynomials were added to  $G_1 \cup Done$  because polynomials with same total degree are added to  $Done$  in the increasing signature order, while the addition polynomials with same total degree to  $R$  is performed in quite random order inside **SPol** and **TopReduction** procedures. For the simplicity we will be speaking about labeled polynomials  $b_j$  from  $G_g$ , assuming that  $G_g$  is not the ordered positions list but the ordered list of labeled polynomials themselves corresponding to those positions. In this terminology we can say that all input polynomials  $\{f_1, \dots, f_m\}$  do present in  $G_g$ , because they all present in  $G_1$  at the moment of its creation.

S-pairs can be processed in a different ways inside the algorithm but the main fact we need to know about their processing is encapsulated in the following properties which correspond to the properties used in Theorem 21 in [4] but are taken during arbitrary algorithm iteration without requirements of termination.

**Теорема 12.** *At the moment of adding  $g$  to  $Done$  every S-pair of  $G_g$  elements which signature is smaller than  $S(g)$  satisfies one of three properties:*

- (1) S-pair has a part that is rejected by the normal form check  $\varphi$  (in **CritPair** or in **IsReducible**). Such S-pairs will be referenced as *S-pairs with a part that satisfies F5 criterion*.
- (2) S-pair has a part that is rejected by the **Rewritten?** check (in **SPol** or in **IsReducible**). Such S-pairs will be referenced as *S-pairs with a part that satisfies Rewritten criterion*.
- (3) S-pair was not rejected, so it's S-polynomial was signature-safe reduced by  $G_g$  elements and the result is stored in  $G_g$ . Such S-pairs will be referenced as *S-pairs with a computed  $G_g$ -representation*.

*Доказательство.* S-pairs of  $G_g$  elements are processed in two paths in the algorithm. The main path is for S-pairs with total degree greater than total degrees of polynomials generated it. Such S-pairs are processed in the following order:

- in the **AlgorithmF5** they are passed in **CritPair** function while moving elements to  $G_i$  from  $R_d = Done$  or while processing input polynomial  $r_i$ .
- The **CritPair** function either discards them by normal form check  $\varphi$  or adds to  $P$
- The S-pair is taken from  $P$  and passed to **SPol** function
- The **SPol** function either discards them by **Rewritten?** check or adds S-polynomial to  $F = ToDo$
- At some iteration the **Reduction** procedure takes S-polynomial from  $ToDo$ , performs some signature-safe reductions and adds result to  $Done$ .

The other processing path is for special S-pairs corresponding to reductions forbidden by algorithm – the case when S-pair is generated by polynomials  $r_{l^*}$  and  $r_{l^*}$  such that  $HM(r_{l^*})|HM(r_{l^*})$  so that S-polynomial has a form  $\overline{u_l} \cdot \text{poly}(r_{l^*}) - 1 \cdot \text{poly}(r_{l^*})$ . Such situation is possible for two  $G_g$  elements if reduction of  $r_{l^*}$  by  $r_{l^*}$  was forbidden by signature comparison in **TopReduction** or by checks in **IsReducible**. So for this case the path of S-pair “processing” is the following:

- The S-pair part  $\overline{u_l} \cdot r_{l^*}$  is checked in **IsReducible**. (a) is satisfied because  $HM(r_{l^*})|HM(r_{l^*})$ . It can be rejected by one of the other checks:
  - Rejection by check (b) correspond to the normal form check  $\varphi$  for  $\overline{u_l} \cdot r_{l^*}$
  - Rejection by check (c) correspond to the **Rewritten?** check for  $\overline{u_l} \cdot r_{l^*}$

- Rejection by check (d) means that either  $\overline{u_l} \cdot r_{l^*}$  or  $1 \cdot r_{l^*}$  can be rewritten by other, so if S-pair was not rejected by check (c) this type of rejection means that S-pair part  $1 \cdot r_{i_1}$  fails to pass **Rewritten?** check.
- The non-rejected in **IsReducible** S-pair is returned in the **TopReduction**. Signature comparison in **TopReduction** forbids the reduction of  $r_{l^*}$  by  $r_l$  and returns computed S-polynomial corresponding to the S-pair in the set  $ToDo_1$
- The **Reduction** procedure add this polynomial in  $ToDo$
- The last step is equal to for both processing paths: at some iteration the **Reduction** procedure takes S-polynomial from  $ToDo$ , performs some signature-safe reductions and adds result to *Done*

It can be seen that after S-pair processing termination every S-pair is either reduced and added to *Done* or one of its S-pair parts is rejected by normal form  $\varphi$  or **Rewritten?** check. Some S-pairs can be processed by processing paths multiple times, for example this is done in the second iteration inside **AlgorithmF5** with same  $d$  value. If the pair was rejected during first processing it will be rejected the same way during next attempt. If the first time processing adds polynomial to *Done* the pair will be rejected in next attempts by the **Rewritten?** check with that polynomial. So all but the first processing attempts are insignificant.

The processing path is not a single procedure and for the case of algorithm infinite cycling some S-pairs are always staying in the middle of the path having S-pair queued in  $P$  or S-polynomial in  $ToDo$ . So we have to select S-pairs which processing is already finished at the fixed moment we studying. The elements from  $P$  and  $ToDo$  in **AlgorithmF5** and **Reduction** procedures are taken in the order corresponding to growth of their signatures. So S-pairs with signature smaller than  $\mathcal{S}(g)$  can be split in the following classes:

- S-pairs with signature  $w$  such that  $\text{index}(w) > \text{index}(\mathcal{S}(g)) = 1$ . They were processed on previous calls of **AlgorithmF5**.
- S-pairs with signature  $w$  such that  $\text{index}(w) = \text{index}(\mathcal{S}(g)) = 1, \deg(w) < \deg(\mathcal{S}(g))$ . They were processed on previous iterations inside the call to **AlgorithmF5** that is processing  $g$ .
- S-pairs with signature  $w$  such that  $\text{index}(w) = \text{index}(\mathcal{S}(g)) = 1, \deg(w) = \deg(\mathcal{S}(g)), w \prec \mathcal{S}(g)$ . They were processed on previous iterations inside the call to **Reduce** that is processing  $g$ .

S-pairs from these classes can't be at the middle of processing path because at the studied state of algorithm the processing is just finished for  $g$  so  $P$  and  $ToDo$  sets does not contain any non-processed elements with signatures smaller  $\mathcal{S}(g)$ . The only left thing to show is proof that processing was started at least one time for all S-pairs from theorem claim. This is true for first two classes: the processing of corresponding S-pairs was started at least one time with the call to **CritPair** inside **AlgorithmF5** just before the greatest of S-pair generators was added to  $G$ . For S-pairs of third class the situation depend on the total degrees of its generators. If both generators of S-pair have total degrees  $< \deg(g)$  then its processing is started in **CritPair** like for the S-pairs from first two classes. But some S-pairs from the third class can have a signature-greater generator polynomial  $r_l$  such that  $\deg(r_l) = \deg(g), \mathcal{S}(r_l) \prec \mathcal{S}(g)$ . They are processed with the second mentioned processing path so the processing for such S-pairs is not yet started at the beginning of last **Reduction** call. Fortunately, their processing starts inside **Reduction** before fixed moment we studying: the procedure selects polynomials from  $ToDo$  in the signature increasing order, so  $r_l$  is reduced before  $g$  and during  $r_l$  reduction just before putting  $r_l$  to *Done* a call to **IsReducible** starts processing for all such S-pairs.  $\square$

The ideas of *satisfying F5 criterion* and *satisfying Rewritten criterion* can be extended to arbitrary monomial-multiplied labeled polynomial  $sh, h \in G_g$ :

**Определение 13.** The monomial-multiplied labeled polynomial  $sr_i$ ,  $r_i \in G_g$  is called *satisfying F5 criterion* if  $\varphi_{\text{index}(r_i)+1}(s\mathcal{S}(r_i)) \neq s\mathcal{S}(r_i)$ , where  $\varphi_{\text{index}(r_i)+1}$  is operator of normal form w.r.t  $G_{\text{index}(r_i)+1}$ .

This definition is equivalent to  $sr_i$  being non-normalized labeled polynomial according to definition 2 in part 5 of [6].

**Определение 14.** The monomial-multiplied labeled polynomial  $sr_i$ ,  $r_i \in G_g$  is called *satisfying Rewritten criterion* if  $\exists j > i$  such that  $\mathcal{S}(r_j) | s\mathcal{S}(r_i)$ .

For the case  $sr_i$  is the S-pair part these definitions are equivalent to the checks in the algorithm in a sense that S-pair part is rejected by the algorithm if and only if it satisfies the definition as monomial-multiplied labeled polynomial. Note that for both criteria holds important property that if  $sr_i$  satisfies a criteria then a further multiplied  $s_1sr_i$  satisfies it too.

## 5. REPRESENTATIONS

**5.1. Definition.** The idea of representations comes from [4], where a similar method is used in the proof of Theorem 21. Representations are used to describe all possible ways how a labeled polynomial  $p$  can be written as an element of  $(G_g)$  ideal. The single representation corresponds to writing a labeled polynomial  $p$  as any finite sum of the form

$$(5.1) \quad p = \sum_k m_k \cdot b_{i_k}, \quad b_{i_k} \in G_g$$

with coefficients  $m_k = c_k t_k \in \mathcal{K} \times T$ .

**Определение 15.** Sum of the form 5.1 with all pairs  $(t_k, b_{i_k})$  distinct is called  $G_g$ -representation of  $p$ . The symbolic products  $m_k \cdot b_{i_k}$  are called the *elements* of representation. If we treat this symbolic product as multiplication we get an labeled polynomial  $m_k b_{i_k}$  corresponding to the representation element. So  $p$  is equal to sum of labeled polynomials, corresponding to elements of its representation. Also the term *element signature* will be used for signature of labeled polynomials corresponding to the element. Two representations are equal if the sets of their elements are equal.

Most representations we are interested in have the following additional property limiting elements signature:

**Определение 16.** The  $G_g$ -representation of  $p$  is called *signature-safe* if  $\forall k \mathcal{S}(m_k b_{i_k}) \preceq \mathcal{S}(p)$ .

### 5.2. Examples.

**Пример 17.** The first important example of a  $G_g$ -representation is trivial: the labeled polynomial from  $G_g$  is equal to sum of one element, identity-multiplied itself:

$$b_j = 1 \cdot b_j.$$

This  $G_g$ -representation is signature-safe. The prohibition of two elements which have same monomial  $t_k$  and polynomial  $b_{i_k}$  ensures that all elements of representation that differ only in field coefficient  $c_k$  are combined together by summing field coefficients. So expressions like  $b_j = -1 \cdot b_j + 2 \cdot b_j$  and  $b_j = 2x \cdot b_k + 1 \cdot b_j - 2x \cdot b_k$  are not valid  $G_g$ -representations.

**Пример 18.** A labeled polynomial  $b_j \in G_g$  multiplied by arbitrary polynomial  $h$  also have a simple  $G_g$ -representation arising from splitting  $h$  into terms:  $h = \sum_k m_k$ ,  $m_k \in \mathcal{K} \times T$ . This  $G_g$ -representation has form

$$(5.2) \quad b_j h = \sum_k m_k \cdot b_j$$

and is signature-safe too.

A labeled polynomial can have arbitrary number of representations: for example we can add elements corresponding to a syzygy to any representation and combine elements with identical monomials and polynomials to get the correct representation. The result will be representation of the same polynomial because sum of syzygy elements is equal to 0.

**Пример 19.** The product of two polynomial from  $G_g$  has two representations of the form (5.2) which differs in syzygy addition:

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

where  $m_{i_k}$  are terms of  $b_i$  and  $m_{j_k}$  are terms of  $b_j$ .

**Пример 20.** The zero polynomial has an empty representation and an representation for every syzygy:

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

where  $m_{i_k}$  and  $m_{j_k}$  are same as above.

Another important example of  $G_g$ -representation comes from ideal and signature definitions. All labeled polynomials computed by the algorithm are elements of ideal  $(f_1, \dots, f_m)$ . So any labeled polynomial  $p$  can be written as  $\sum_i f_i g_i$ , where  $g_i$  are homogeneous polynomials. All input polynomials  $f_i$  belong to  $G_g$ , so  $f_i g_i$  has  $G_g$ -representations of the form (5.2).

**Пример 21.** Those representations sum give the following signature-safe representation:

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

**Определение 22.** This particular case of  $G_g$ -representation where  $b_{i_k}$  are limited to input polynomials will be called *input-representation*.

Input representations always has the only element with maximal signature. This property is special to input-representations because generic  $G_g$ -representations can have multiple elements with same maximal signature – it is possible to have  $m_1 \mathcal{S}(b_{i_1}) = m_2 \mathcal{S}(b_{i_2})$  while  $i_1 \neq i_2$ .

The following claim makes important connection between signatures and input-representations:

**Утверждение 23.** An admissible labeled polynomial  $p$  with known signature  $\mathcal{S}(p)$  has an input-representation consisting of an element  $c\mathcal{S}(p) \cdot f_{index(p)}$  and some other elements with smaller signatures.

*Доказательство.* The claimed fact follows from the admissible polynomial definition in [6] referring to function  $v$  which correspond to summing representation elements.  $\square$

The theorem 1 of [6] states that all polynomials in the algorithm are admissible, do the above claim will be applied to all appeared polynomials.

**Пример 24.** The last example comes from S-pairs with a computed  $G_g$ -representation. S-polynomial of  $b_{l^*}$  and  $b_{l_*}$  from  $G_g$  is  $p = \overline{u_l} \text{poly}(b_{l^*}) - \underline{u_l} \text{poly}(b_{l_*})$ . It is known from reduction process that for such S-pairs  $p$  is signature-safe reduced and the result is added to  $G_g$  as some labeled polynomial  $b_l$ . So the  $G_g$ -representation is:

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

where signatures of  $m_k \cdot b_{n_k}$  elements are smaller than  $\mathcal{S}(b_l) = \mathcal{S}(p)$ . The value of  $l$  is position of  $b_l$  in ordered list  $G_g$ . In this representation  $l$  is greater than  $l^*$  and  $l_*$  because corresponding labeled polynomial  $b_l$  is added to  $R$  at the moment of S-polynomial computation in **Spol** or **TopReduction** so the polynomials  $b_{l^*}$  and  $b_{l_*}$  used to create S-pair already present in  $R$  at that moment and the order of  $G_g$  correspond to order of  $R$ .

### 5.3. Ordering representations.

**Определение 25.** To order  $G_g$ -representations we start from *representation elements ordering*  $\succ_1$ : we say that  $c_i t_i \cdot b_i \succ_1 c_j t_j \cdot b_j$  if one of the following cases holds:

- $t_i \mathcal{S}(b_i) \succ t_j \mathcal{S}(b_j)$
- $t_i \mathcal{S}(b_i) = t_j \mathcal{S}(b_j)$  and  $i < j$  (note the opposite order).

This ordering is based only on comparison of signatures and positions of labeled polynomials in the ordered list  $G_g$  but does not depend on the field coefficient. The only case in which two elements can't be ordered is equality of both signatures  $t_i \mathcal{S}(b_i) = t_j \mathcal{S}(b_j)$  and positions in list  $i = j$ . Position equality means  $b_i = b_j$  which in conjunction with signature equality gives  $t_i = t_j$ . So any two elements that belong to a single  $G_g$ -representation are comparable with  $\leq_1$  order because they have distinct  $(t_k, b_k)$  by definition. Below are given some examples of  $\leq_1$  element ordering for the 3-element list  $G_g = \{b_1, b_2, b_3\}$  with ordering  $x\mathbf{F}_i \succ y\mathbf{F}_i$  and signatures  $\mathcal{S}(b_1) = \mathbf{F}_1, \mathcal{S}(b_2) = \mathbf{F}_2, \mathcal{S}(b_3) = x\mathbf{F}_1$ .

- $y \cdot b_1 \succ_1 100y \cdot b_2$  because signature of left side is  $\succ$
- $x \cdot b_1 \succ_1 y \cdot b_1$  because signature of left side is  $\succ$
- $-x \cdot b_1$  and  $2x \cdot b_1$  are not comparable because signatures and list indexes are equal
- $y^2 \cdot b_1 \leq_1 y \cdot b_3$  because signature of left side is  $\prec$
- $x^2 \cdot b_1 \succ_1 x \cdot b_3$  because signatures are equal and the list position of left side's labeled polynomial is 1 which is smaller than right side's position 3.

To extend this order to entire  $G_g$ -representations consider *ordered form* of representation consisting of all its elements written in a list with  $\succ_1$ -decreasing order. This form can be used for equality testing because if two representations are equal then they have exactly equal ordered forms.

**Определение 26.** With ordered forms the  $G_g$ -representations ordering can be introduced: the representation  $\sum_k m'_k \cdot b_{i'_k}$  is  $\leq$ -smaller than  $\sum_k m_k \cdot b_{i_k}$  if the ordered form of the first representation is smaller than second's according to lexicographical extension of  $\leq_1$  ordering on elements. For the corner case of the one ordered form being beginning of the other the shorter form is  $\leq$ -smaller. If the greatest different elements of ordered forms differ only in field coefficient the representations are not comparable.

Some examples of this ordering are given for the same as above 3-element  $G_g$  list. Note that all  $G_g$ -representations are already written in ordered forms:

- $x^2 \cdot b_1 + xy \cdot b_1 + y^2 b_1 \succ x^2 \cdot b_1 + 100y^2 \cdot b_1$  because  $xy \cdot b_1 \succ y^2 \cdot b_1$
- $x^2 \cdot b_1 + 100y^2 \cdot b_1 \succ x^2 \cdot b_1$  because the right ordered form is beginning of the left
- $x^2 \cdot b_1 \succ xy \cdot b_1 + y^2 \cdot b_1 + x^2 \cdot b_2$  because  $x^2 \cdot b_1 \succ xy \cdot b_1$
- $xy \cdot b_1 + y^2 \cdot b_1 + x^2 \cdot b_2 \succ y \cdot b_3 + y^2 \cdot b_1 + x^2 \cdot b_2$  because  $xy \cdot b_1 \succ y \cdot b_3$
- $y \cdot b_3 + y^2 \cdot b_1 + x^2 \cdot b_2$  and  $2y \cdot b_3 + y^2 \cdot b_2$  are not comparable because the greatest different elements are  $y \cdot b_3$  and  $2y \cdot b_3$ .

The ordering is compatible with signature-safety:

**Теорема 27.** *If two representations of  $p$  has a relation  $\sum_k m'_k \cdot b_{i'_k} \leq \sum_k m_k \cdot b_{i_k}$  and the second one is signature-safe representation then the first one is signature-safe too.*

*Доказательство.* This theorem quickly follows from a fact that elements of a  $\leq$ -smaller representation can't has signatures  $\succ$ -greater than signatures of  $\succ$ -greater representation.  $\square$

The key fact allowing to take  $\leq$ -minimal element is well-orderness:

**Теорема 28.** *The representations are well-ordered with  $\leq$  ordering.*

*Доказательство.* The number of different labeled polynomial positions is finite because it is equal to  $|G_g|$  which is finite for fixed  $g$ . So the existence of infinite  $\succ_1$ -descending sequence of representation elements would lead to existence of infinite  $\succ$ -descending sequence of signatures. Combining this with well-orderness of signatures with ordering  $\prec$  we get the proof for well-orderness of elements with ordering  $\leq_1$ .

The straightforward proof for  $\leq$ -well-orderness of representations following from  $\leq_1$ -well-orderness of elements is not very complex but to skip its strict details the theorem 2.5.5 of [3] will be referenced. It states well-orderness of finite multiset with an lexicographically extended ordering of well-ordered elements. This applies to the representations because they form a subset in the finite multiset of representation elements.  $\square$

**5.4. Sequence of representations.** The idea of this part is constructing a finite sequence of strictly  $\leq$ -descending signature-safe  $G_g$ -representations for a given labeled polynomial  $mh$ ,  $m \in \mathcal{K} \times T$ ,  $h \in G_g$  with  $\mathcal{S}(mh) \prec \mathcal{S}(g)$ . The first signature-safe representation in the sequence is  $mh = m \cdot h$ , the last representation is  $mh = \sum_k m_k \cdot b_{i_k}$  with elements having the following properties  $\forall k$ :

- (1)  $m_k b_{i_k}$  does not satisfy F5 criterion.
- (2)  $m_k b_{i_k}$  does not satisfy Rewritten criterion.
- (3)  $\text{HM}(m_k b_{i_k}) \leq \text{HM}(mh)$

The proof of such sequence existence is very similar to Theorem 21 of [4] and is based on a fact, that if a some signature-safe representation of  $mh$  contains an element  $m_{K'} \cdot b_{i_{K'}}$  not having one of the properties then a  $\leq$ -smaller representation can be constructed. The exact construction differ for three cases but the replacement scheme is the same:

- a some element  $m_{K'} \cdot b_{i_{K'}}$  in  $mh$  representation is selected. Note that  $K'$  in some cases is not equal to  $K$
- some representation  $m_{K'} b_{i_{K'}} = \sum_l m_l \cdot b_{i_l}$  is constructed for this element.
- it is shown that constructed representation is  $\leq$ -smaller than representation  $m_{K'} b_{i_{K'}} = m_{K'} \cdot b_{i_{K'}}$

Construction of such representation for  $m_{K'} \cdot b_{i_{K'}}$  allows application of the following lemma:

**Лемма 29.** *If an element  $m_{K'} \cdot b_{i_{K'}}$  of signature-safe representation  $mh = \sum_k m_k \cdot b_{i_k}$  has an representation  $m_{K'} b_{i_{K'}} = \sum_l m_l \cdot b_{i_l}$  which is  $\leq$ -smaller than representation  $m_{K'} b_{i_{K'}} = m_{K'} \cdot b_{i_{K'}}$ , then  $mh$  has a signature-safe representation  $\leq$ -smaller than  $mh = \sum_k m_k \cdot b_{i_k}$ .*

*Доказательство.* We replace  $m_{K'} \cdot b_{i_{K'}}$  in  $mh = \sum_k m_k \cdot b_{i_k}$  by  $\sum_l m_l \cdot b_{i_l}$  and combine coefficients near elements with both monomial and polynomial equal, so a modified representation for  $mh$  appears. Is is  $\leq$ -smaller than  $mh = \sum_k m_k \cdot b_{i_k}$  because all elements  $\succ_1$ -greater than  $m_{K'} \cdot b_{i_{K'}}$  are identical in both representations if they present but the element  $m_{K'} \cdot b_{i_{K'}}$  is contained in original representation but not in the modified. And all other elements in representations are  $\leq_1$ -smaller than  $m_{K'} \cdot b_{i_{K'}}$  so they does not influence the comparison. The comparison holds even in a corner case when all elements are discarded while combining coefficients. This case can appear if the original representation is equal to  $mh = m_{K'} \cdot b_{i_{K'}} + \sum_l (-m_l) \cdot b_{i_l}$  what leads

to modified representation  $mh = 0$  with zero elements which is  $\ll$ -smaller than any non-empty representation.  $\square$

Now it will be shown that replacement scheme can be performed if the representation contains an element not satisfying at least one of three properties.

**Лемма 30.** *If a signature-safe  $G_g$ -representation  $mh = \sum_k m_k \cdot b_{i_k}$  does not satisfy property 1 then there exists an element  $m_{K'} \cdot b_{i_{K'}}$  having  $G_g$ -representation  $m_{K'} b_{i_{K'}} = \sum_l m_l \cdot b_{i_l}$  which is  $\ll$ -smaller than representation  $m_{K'} b_{i_{K'}} = m_{K'} \cdot b_{i_{K'}}$ .*

An element not having the first property does satisfy the F5 criterion and the idea is to use that  $m_K \mathcal{S}(b_{i_K})$  is not the minimal signature of  $m_K b_{i_K}$  like in Theorem 20 of [4].  $K' = K$  is taken for this case.

*Доказательство.* Consider input-representation of  $m_K b_{i_K}$  with signature of  $\geq_1$ -maximal element equal to  $m_K \mathcal{S}(b_{i_K}) = s_0 \mathbf{F}_{j_0}$ :

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

From the satisfying F5 criterion  $s_0$  can be expressed like  $s_0 = s_1 \text{HM}(f_{j_1})$ ,  $j_1 > j_0$  so  $s_0 f_{j_0} = s_1 f_{j_0} f_{j_1} - s_1 (f_{j_1} - \text{HM}(f_{j_1})) f_{j_0}$ . From this we can write another representation for  $m_K b_{i_K}$ , assuming  $m_{0i}$  are sorted terms of  $f_{j_0}$ ,  $m_{1i}$  are sorted terms of  $f_{j_1}$  and  $N_0, N_1$  are number of terms in those polynomials:

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

This representation is  $\ll$ -smaller than  $m_K \cdot b_{i_K}$  because signatures of all elements are smaller than  $s_0 \mathbf{F}_{j_0}$ . For the elements of the third sum  $\sum_l m_l \cdot f_{i_l}$  this follows from 5.3, where those elements are smaller elements of input-representation. For the elements of the first sum  $\sum_{i=1}^{N_0} c_0 s_1 m_{0i} \cdot f_{j_1}$  this follows from the position inequality  $j_1 > j_0$ . And for the second sum we use the equality in term and signature orderings: all terms  $m_{1i}$ ,  $i \geq 2$  are smaller than  $m_{11}$ , so the signatures are:  $s_1 m_{1i} \mathbf{F}_{j_0} \prec s_1 m_{11} \mathbf{F}_{j_0} = s_0 \mathbf{F}_{j_0}$ .  $\square$

**Лемма 31.** *If a signature-safe  $G_g$ -representation  $mh = \sum_k m_k \cdot b_{i_k}$  with  $\mathcal{S}(mh) \prec \mathcal{S}(g)$  does not satisfy property 2 then there exists an element  $m_{K'} \cdot b_{i_{K'}}$  having  $G_g$ -representation  $m_{K'} b_{i_{K'}} = \sum_l m_l \cdot b_{i_l}$  which is  $\ll$ -smaller than representation  $m_{K'} b_{i_{K'}} = m_{K'} \cdot b_{i_{K'}}$ .*

For the elements not satisfying case 2 the  $\ll$ -smaller representation is created in a way used in Proposition 17 of [4].  $K' = K$  is taken for this case too.

*Доказательство.* Assume that  $\mathcal{S}(m_K b_{i_K}) = s_0 \mathbf{F}_{j_0}$  and it is rewritten by labeled polynomial  $b_{i'}$  from  $R$ . Because the representation is signature-safe we have  $\mathcal{S}(b_{i'}) \preceq s_0 \mathbf{F}_{j_0} \preceq \mathcal{S}(mh) \prec \mathcal{S}(g)$ . So  $b_{i'}$  was processed in **TopReduction** before  $g$ . Since  $b_{i'}$  is rewriter we have  $b_{i'} \neq 0$ . All this gives the fact that  $b_{i'}$  does present not only in  $R$  but in  $G_g$  too so it can be used as a polynomial of  $G_g$ -representation element. From the Rewritten criterion definition we know that  $i' > i_K$  and the existence of  $s' \in T$  such that  $s' \mathcal{S}(b_{i'}) = s_0 \mathbf{F}_{j_0}$ . So, for the  $m_K b_{i_K}$  there is an input-representation 5.3 and for the  $s' b_{i'}$  the input-representation is:

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

A  $G_g$ -representation for  $c_0 s_0 f_{j_0}$  can be acquired with transformation of the above expression:

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

Using this to replace the first element in 5.3 we get the wanted result:

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

It is  $\ll$ -smaller than  $m_K b_{i_K} = m_K \cdot b_{i_K}$  because elements of both sums has signatures smaller than  $s_0 \mathbf{F}_{j_0}$ , and for the first element  $\mathcal{S}(c'^{-1} c_0 s' \cdot b_{i'}) = \mathcal{S}(m_K \cdot b_{i_K}) = s_0 \mathbf{F}_{j_0}$  but  $i' > i_K$ , so applying the  $\ll_1$ -comparison rule for equal signatures and different list positions we get that element  $c'^{-1} c_0 s' \cdot b_{i'}$  is  $\ll_1$ -smaller than  $m_K \cdot b_{i_K}$  too.  $\square$

**Лемма 32.** *If a signature-safe representation  $mh = \sum_k m_k \cdot b_{i_k}$  with  $\mathcal{S}(mh) \prec \mathcal{S}(g)$  satisfies properties 1 and 2 but does not satisfy property 3 then there exists an element  $m_{K'} \cdot b_{i_{K'}}$  having representation  $m_{K'} b_{i_{K'}} = \sum_l m_l \cdot b_{i_l}$  which is  $\ll$ -smaller than representation  $m_K b_{i_K} = m_{K'} \cdot b_{i_{K'}}$ .*

*Доказательство.* There exists at least one element  $m_K \cdot b_{i_K}$  that does not satisfy property 3. Let  $m_{\max}$  be the maximal HM of labeled polynomials corresponding to representation elements and  $H_{\max}$  be a list of elements where  $m_{\max}$  is achieved. Select  $K'$  to be the index of the  $\succ_1$ -greatest representation element in  $H_{\max}$ . We have  $\text{HM}(m_{K'} b_{i_{K'}}) = m_{\max} \geq \text{HM}(m_K b_{i_K}) > \text{HM}(mh)$ , so the HM of sum of all elements except  $K'$  is equal to  $\text{HM}(mh - m_K b_{i_K}) = \text{HM}(m_{K'} b_{i_{K'}}) = m_{\max}$ , so there is another element  $K''$  having  $\text{HM}(m_{K''} b_{i_{K''}}) = m_{\max}$ . So,  $m_{K''} \cdot b_{i_{K''}} \in H_{\max}$  and  $m_{K''} \cdot b_{i_{K''}} \ll_1 m_{K'} \cdot b_{i_{K'}}$  because of  $m_{K'} \cdot b_{i_{K'}}$   $\succ_1$ -maximality in  $H_{\max}$ .

The  $\text{HM}(m_{K''} b_{i_{K''}}) = \text{HM}(m_{K'} b_{i_{K'}})$  means that a critical pair of  $b_{i_{K'}}$  and  $b_{i_{K''}}$  has the form  $[m'^{-1} m_{\max}, m'^{-1} m_{K'}, b_{i_{K'}}, m'^{-1} m_{K''}, b_{i_{K''}}]$  where  $m' = \gcd(m_{K'}, m_{K''})$ . Let  $q$  be corresponding S-polynomial. Then  $m' \mathcal{S}(q) \prec \mathcal{S}(mh) \prec \mathcal{S}(g)$  because  $m' \mathcal{S}(q) = \mathcal{S}(m_{K'} b_{i_{K'}})$  and the representation is signature-safe. The S-polynomial parts  $m'^{-1} m_{K'} b_{i_{K'}}$  and  $m'^{-1} m_{K''} b_{i_{K''}}$  does not satisfy F5 and Rewritten criteria because their forms multiplied by  $m'$  are  $m_{K'} b_{i_{K'}}$  and  $m_{K''} b_{i_{K''}}$  – labeled polynomials corresponding to elements which are known not to satisfy both criteria by assumption. Therefore  $m' \mathcal{S}(q) \prec \mathcal{S}(g)$  and  $\mathcal{S}(q) \prec \mathcal{S}(g)$ . It follows from this with theorem 12 that the S-pair  $(b_{i_{K'}}, b_{i_{K''}})$  is S-pair with computed  $G_g$ -representation, what means that there is an representation described in example 24 :

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

satisfying the properties shown after that example:  $\mathcal{S}(q) = \mathcal{S}(b_{i'}), \forall l \mathcal{S}(q) \succ \mathcal{S}(m_l b_{i_l})$  and  $i' > K'$ .

From the other hand we have  $m' q = c_0 m_{K'} b_{i_{K'}} - c_1 m_{K''} b_{i_{K''}}$ , so we get the following representation:

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

It is  $\ll$ -smaller than  $m_{K'} b_{i_{K'}} = m_{K'} \cdot b_{i_{K'}}$ :

$m_{K''} \cdot b_{i_{K''}}$  was already compared to  $m_{K'} \cdot b_{i_{K'}}$

$m' \cdot b_{i'}$  has the same signature but greater position  $i' > i_{K'}$

the last sum contains elements with signatures smaller than  $m' \mathcal{S}(b_{i'}) = \mathcal{S}(m_{K'} \cdot b_{i_{K'}})$ .  $\square$

**Теорема 33.** *A signature-safe representation  $mh = \sum_k m_k \cdot b_{i_k}$  with  $\mathcal{S}(mh) \prec \mathcal{S}(g)$  either satisfies properties 1-3 or there exists a signature-safe representation  $mh = \sum_l m_l \cdot b_{i_l}$  which is  $\ll$ -smaller than  $\sum_k m_k \cdot b_{i_k}$ .*

*Доказательство.* This theorem quickly follows from four previous lemmas together  $\square$



This leads to main result:

**Теорема 34.** *For any labeled polynomial  $mh$ ,  $m \in \mathcal{K} \times T$ ,  $h \in G_g$  with  $\mathcal{S}(mh) \prec \mathcal{S}(g)$  there exists a signature-safe  $G_g$ -representation  $mh = \sum_k m_k \cdot b_{i_k}$  that satisfies properties 1-3.*

*Доказательство.* Start with representation  $mh = m \cdot h$  and begin replacing it by  $\prec$ -smaller representation from theorem 33 until the representation satisfying properties 1-3 appears. The finiteness of the process is guaranteed by  $\prec$ -well-orderness.  $\square$

This result may be interesting by itself, but for the purposes of proving termination only one corollary is needed:

**Вывод 35.** *Consider an arbitrary polynomial  $f$  without any restrictions on its signature. If there exists a signature-safe reductor  $f' \in G_g$  for  $f$  with  $\mathcal{S}(f') \frac{\text{HM}(f)}{\text{HM}(f')} \prec \mathcal{S}(g)$  then  $G_g$  contains a signature-safe reductor for  $f$  that is not rejected by F5 and Rewritten criteria.*

*Доказательство.* Let  $mf', m = \frac{\text{HM}(f)}{\text{HM}(f')} \in \mathcal{K} \times T$ ,  $f' \in G_g$  be a multiplied reductor with  $\mathcal{S}(mf') \prec \mathcal{S}(g)$ . From the previous theorem we can find representation  $mf' = \sum_k m_k \cdot b_{i_k}$  that satisfies properties 1-3. Property 3 means that there is no elements with HM's greater than  $mf'$  so because sum of all elements has HM equal to  $\text{HM}(mf')$  there exists an element  $K$  that achieves HM equality:  $\text{HM}(m_K \cdot b_{i_K}) = \text{HM}(mf') = \text{HM}(f'_1)$ . Since the representation is signature-safe  $\mathcal{S}(m_K \cdot b_{i_K}) \preceq \mathcal{S}(mf') \prec \mathcal{S}(f)$  so  $m_K b_{i_K}$  is a signature-safe reductor for  $f$  and properties 1-2 ensure that  $m_K b_{i_K}$  does not satisfy criteria.  $\square$

## 6. FINDING CONTRADICTION WITH THE CRITERIA ENABLED

Now return to the result of theorem 10 which states for the case of algorithm non-termination existence of a polynomials  $f', f \in G$  such that  $\text{HM}(f') | \text{HM}(f)$ ,  $\frac{\text{HM}(f')}{\mathcal{S}(f')} >_q \frac{\text{HM}(f)}{\mathcal{S}(f)}$ . Using this result and last corollary we construct two polynomials leading to contradiction for the case of algorithm non-termination.

**Теорема 36.** *If the algorithm does not terminate for some input then after some finite step the set  $G \cup \text{Done}$  contains a pair of labeled polynomials  $f'_1, f$  where:*

- $f'_1$  is added to  $G \cup \text{Done}$  before  $f$
- $t_1 f'_1$  does not satisfy F5 and Rewritten criteria, where  $t_1 = \frac{\text{HM}(f)}{\text{HM}(f'_1)}$
- $f'_1$  is signature-safe reductor for  $f$ .

*Доказательство.* Let  $f', f$  be polynomials from the theorem 10 and define  $t = \frac{\text{HM}(f)}{\text{HM}(f')}$ . We have  $f \in G$  so the above theory about representations can be applied to the fixed value of  $g$  equal to  $f$  and we can speak about  $G_f$  set and  $G_f$ -representations. Because  $tf'$  is a signature-safe reductor for  $f$  we have  $\mathcal{S}(f')t \prec \mathcal{S}(f)$  and the corollary 35 can be applied to find a signature-safe reductor  $t_1 f'_1$  for  $f$  which does not satisfy criteria. Also it is known to belong to  $G_f$ , so during the algorithm execution  $f'_1$  was appended to  $G \cup \text{Done}$  before  $f$ .  $\square$

**Теорема 37.** *The original F5 algorithm as described in [6] does terminate for any input.*

*Доказательство.* We are going to show that the existence of polynomials  $f'_1, f$  from the theorem 36 leads to contradiction. Consider the call to `TopReduction` after which the polynomial  $f$  was inserted in *Done*. That call returns polynomial  $f$  as first part of `TopReduction` return value, so the value returned by `IsReducible` is empty set. It means that one of conditions (a) - (d) was not satisfied for all polynomials in  $G \cup \text{Done}$  including  $f'_1$ . This is not possible because:

- (a) is satisfied because  $f'_1$  is a reductor for  $f$  from the theorem 36

- (b) and (c) are satisfied because  $\frac{HM(f)}{HM(f'_1)}f'_1$  does not satisfy F5 and Rewritten criteria from the theorem 36
- (d) is satisfied because  $f'_1$  is a signature-safe reductor for  $f$  from the theorem 36.

□

## 7. CONCLUSIONS

This paper shows that original F5 algorithm terminates for any homogeneous input without introducing intermediate algorithms. However, it does not give any limit on number of operations. The simplest proof of the termination of Buchberger algorithm is based on Noetherian property and does not give any such limit too. Unfortunately the termination proof given here is quite different in structure compared to the proof of Buchberger algorithm termination, so this proof does not show that F5 is more efficient than Buchberger in any sense. Unlike this the termination of the modified versions of F5 algorithm in [5, 2, 7] is shown in a way analogous to Buchberger algorithm and there is room for comparison of their efficiency with Buchberger's one.

From the point of view of practical computer algebra computations there is a question about efficiency of the modified versions compared to original F5. The modified versions can spend more time in additional termination checks. But for some cases it is possible that those checks can allow the termination of modified versions before original so the modified version performs smaller number of reductions. So it is possible that for some inputs the original algorithm is faster and for others the modified version. Some experimental timings in Table 1 in [5] shows that both cases are possible in practice but the difference in time is insignificant. So the question about efficiency of original F5 compared to modified versions is open.

This proof uses three properties of original F5 that are absent or optional in some F5-like algorithms: the homogeneity of input polynomials, the presence of Rewritten criterion and the equality of monomial order  $<$  and signature order  $\prec$ . The possibility of extending the termination proof to the modified algorithms without these properties is open question. There is an unproved idea that the proof can be modified to remove reliance on the first two properties but not on the third property of orders equality because it is key point of coming to a contradiction from the result of theorem 10.

The author would like to thank Christian Eder, Jean-Charles Faugère, Amir Hashemi, John Perry, Till Stagers and Alexey Zobnin for inspiring me on investigations in this area by their papers and comments. Thanks!

## СПИСОК ЛИТЕРАТУРЫ

- [1] A. Arri and J. Perry. The F5 Criterion revised. *ArXiv e-prints*, December 2010.
- [2] Gwenole Ars. Applications des bases de Gröbner à la cryptographie. Technical report, 2005.
- [3] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, United Kingdom, 1998.
- [4] C. Eder and J. Perry. F5C: a variant of Faugère's F5 algorithm with reduced Gröbner bases. *ArXiv e-prints*, June 2009.
- [5] Christian Eder, Justin Gash, and John Perry. Modifying Faugère's F5 Algorithm to ensure termination. *ACM Communications in Computer Algebra, Issue 176, vol. 45, no. 2 (June 2011), pgs. 70-89*, December 2010.
- [6] Jean Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *Proceedings of the 2002 international symposium on Symbolic and algebraic computation, ISSAC '02*, pages 75–83, New York, NY, USA, 2002. ACM.
- [7] Justin M. Gash. *On efficient computation of Gröbner bases*. PhD thesis, Indiana University, Indianapolis, IN, USA, 2008.
- [8] Amir Hashemi and Gwénolé Ars. Extended F5 criteria. *J. Symb. Comput.*, 45(12):1330–1340, 2010.
- [9] L. Huang. A new conception for computing Gröbner basis and its applications. *ArXiv e-prints*, December 2010.

- [10] S. Pan, Y. Hu, and B. Wang. The Termination of Algorithms for Computing Gröbner Bases. *ArXiv e-prints*, February 2012.
- [11] Till Stegers. Faugere's F5 Algorithm Revisited. *IACR Cryptology ePrint Archive*, 2006:404, 2006.
- [12] A. I. Zobnin. Generalization of the F5 algorithm for calculating Gröbner bases for polynomial ideals. *Programming and Computing Software*, 36(2):75–82, March 2010.

МГУ

*E-mail address:* galkin-vv@ya.ru