Предварительный отчёт

Никита Костливцев

20 июня 2018 г.

1 Описание задачи

Здесь представлен обзор статьи Джада Хэмза и Виктора Кункака «Minimal Synthesis of String To String Functions From Examples». В данной статье описывается один из способов решения задачи генерации программы по предоставленным входным/выходным данным. Грубо говоря, эта задача составления программы, когда известны результаты, которые должна выдавать программа на некоторое множество входов. Причём получившаяся программа должна хорошо обобщать примеры, поданные ей на вход.

Входные и выходные данные представляются в виде строк. Авторы статьи решают задачу путем генерации автомата, детерминированного по выходу (Output-deterministic automaton или ODA), согласующегося с входными/выходными данными. ODA представляет собой детерминированный конечный автомат, алфавит которого составляет декартово произведение входного и выходного алфавитов, а также имеющий следующие дополнительные ограничения. Находясь в некотором состоянии, данный автомат принимает следующие символы входной и выходной строки и переходит в следующее состояние. Нужно иметь ввиду, что данный автомат является детерминированным только относительно своего алфавита. Если же алфавит ODA сузить на входной алфавит, то он будет недетерминированным. Из соображений здравого смысла в ODA по любой входной строке хотелось бы получать единственную выходную строку. Таким образом описанный выше автомат также влечёт некоторые ограничения на входные данные: входная и выходная строки должны иметь одинаковую длины, а также каждая входная строка должна быть сопоставлена единственной выходной строке. Далее, авторы статьи предлагают генерировать минимальный полный ODA, согласующийся с входными/выходными данными, что является хорошим способом их обобщения.

Однако генерация минимального ODA, согласующегося с входными/выходными данными, как было доказано в статье, является NP-полной задачей. А именно, данную задачу очевидно можно свести к проблеме выбора: существует ли ODA, согласующийся с входными/выходными данными, в котором ровно k состояний, к которой, как было показано, сводится по Карпу задача об удовлетворимости булевой формулы? В статье было показано, что количество состояний в минимальном автомате не может пре-

высить суммарной длины входных данных, увеличенной на два. Из этого следует, что исходная задача тоже является NP-полной.

2 Формальная постановка задачи

Исходя из вышенаписанного, дадим формальные определения.

Определение 1. *Алфавит* Σ *– непустое конечное множество.*

Определение 2. Σ^n *– множество слов длины* n *из алфавита* Σ*.*

Определение 3. $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$*.*

Определение 4. *Длина слова - функция* $|| : \Sigma^* \to \mathbb{Z}^+$*.* $|u| = n$*, если* $u \in \Sigma^n$*.*

Определение 5. *Недетерминированный конечный автомат – это кортеж (*Σ*,* Q*,* q_{init}*,* δ*,* F*), где* Σ *– алфавит автомата,* Q *– конечное множество состояний,* $q_{init} \in Q$ *– начальное состояние,* $\delta \subseteq Q \times \Sigma \times Q$ *– функция перехода,* F *– допускающие состояния.*

Определение 6. $\mathcal{L}(A)$ *– множество всех слов, которые допускает автомат* A*.*

Определение 7. *Автомат называется недвойственным, если* $\forall n \forall w \in \Sigma^n$ $\exists! q_1..q_{n+1}, q_i \in Q : (q_i, w_i, q_{i+1}) \in \delta \ \forall i \in 1..n$

Определение 8. *Автомат называется детерминированным, если* $\forall q_1 \in Q, \forall a \in \Sigma \ \exists! q_2 \in Q$*:* $(q_1, a, q_2) \in \delta$

Определение 9. $|A| = n$*, если автомат* A *имеет* n *состояний.*

Определение 10. *Пусть* $u = u_1...u_n$*,* $v = v_1...v_n$*.*
Тогда $u * v = (u_1, v_1)...(u_n, v_n)$*.*

Определение 11. *Пусть* Σ *– алфавит входных слов,* Γ *– алфавит выходных слов. Автомат, детерминированный по выходу (ODA) – конечный детерминированный автомат над алфавитом* $\Sigma \times \Gamma$*, для которого выполняется условие:* $\forall u \in \Sigma^*, \forall v_1, v_2 \in \Gamma^*$*, если* $u * v_1 \in \mathcal{L}(A), u * v_2 \in \mathcal{L}(A)$*, то* $v_1 = v_2$*.*

Определение 12. *Можно заметить, что для ODA существует частичная функция* $\overline{A} : \Sigma^* \to \Gamma*$*, что* $\forall u * v \in \mathcal{L}(A)$ $\overline{A}(u) = v$*. Тогда ODA называется полным, если полна* $\overline{A}$*.*

Формально обозначим следующие задачи:

Задача 1. *Пусть* $E \subseteq \Sigma^* \times \Gamma^*$ *– множество входных/выходных строк. Найти полный ODA, согласующийся с* E*, размер которого минимален.*

Задача 2. *Пусть* $E \subseteq \Sigma^* \times \Gamma^*$ *– множество входных/выходных строк. Найти полный ODA, согласующийся с* E*, размер которого не привышает заданного* n*.*

В статье было приведено три доказательства NP-полноты для трех разных случаев. Все эти случаи доказывались по одному принципу – сведением этих задач к задаче One-In-Three SAT, которая является NP-полной.

Задача 3. *Пусть дано множество переменных V и множество троек переменных $C \subseteq V^3$. Существует ли отображение $f : V \to 0, 1$ такое, что для каждой тройки $(x, y, z) \in C$ только одна из переменных x, y, z отображается в 1 под действием отображения f? Данная задача называется One-In-Three SAT задачей.*

Теорема 1. *Задача 2 является NP-полной, когда количество состояний фиксировано, выходной алфавит Γ фиксирован, и существует единственный входной/выходной пример.*

Теорема 2. *Задача 2 является NP-полной, когда входной алфавит Σ, выходной алфавит Γ фиксированы, и существует единственный входной/выходной пример.*

Заметим, что оставшийся случай, когда фиксированы оба алфавита и количество состояний является тривиальным и не является NP-полной задачей. В данном случае необходимо просто перебрать всевозможные автоматы и проверить их на согласование с примерами и полноту, что можно сделать за полиномиальное время.

Также в статье авторы доказали даже более сильное утверждение: было доказано, что даже если входные/выходные примеры имеют одинаковую длину, задача все равно остается NP-полной.

Определение 13. *Назовем недетерминированный конечный автомат $A = (\Sigma, Q, q_{init}, \delta, F)$ l-слоистым для $l \in \mathbb{N}$, если автомат допускает только слова длины l, то есть $\mathcal{L}(A) \subseteq \Sigma^l$. Тогда такой автомат будет являться l-полным, если областью определения функции $\overline{A}$ будет являться Σ^l.*

Задача 4. *Пусть Σ – входной алфавит, Γ – выходной алфавит и $l \in \mathbb{N}$. Тогда $u_1 * v_1, u_2 * v_2 ... u_k * v_k$ – множество входных/выходных примеров, таких что $u_i \in \Sigma^l$ и $v_i \in \Gamma^l$*

Существует ли l-слоистый и l-полный ODA, который допускает все $u_i * v_i$ $1 \leq i \leq k$ и имеет не более n состояний?

Теорема 3. *Задача 4 является NP-полной, если входной алфавит Σ и выходной алфавит Γ являются фиксированными.*

Далее, чтобы полностью закончить доказательство, что и Задача 1 является NP-полной авторы доказали лемму.

Лемма 1. *Пусть $E \subseteq (\Sigma \times \Gamma)*$ – корректные множество входных/выходных примеров. Тогда существует полный ODA, согласующийся с E, в котором не более $2 + \Sigma_{w \in E}|w|$ состояний.*

Таким образом, чтобы решить Задачу 1, мы можем решить Задачу 2 не более линейного от входных данных количества раз. Это означает, что Задача 1 тоже является NP-полной.

3 Решение с помощью SMT-решателя

Теперь рассмотрим решение, предложенное авторами статьи, основанное на применении SMT-решателя. SMT - задача разрешимости логических формул с учетом лежащих в них теорий, выраженных в логике первого порядка с равенством. Пусть E – набор входных/выходных примеров. Пусть $\phi_{E,k}$ – формула, которая выполняется тогда и только тогда, когда существует полный ODA с k состояниями, согласующийся с E. Будем проверять с помощью SMT-решателя выполнимость формулы $\phi_{E,k}$. Свободными переменными в $\phi_{E,k}$ будут функции, описывающие детерминированный конечный автомат A с k состояниями, а именно: $\delta : Q \times (\Sigma \times \Gamma) \to Q$ – функция, описывающая переходы в A, $isFinal : Q \to 0, 1$ – функция, определяющая допускающие состояния в A, и δ_{in} – отображение, которое будет являться проекцией δ на входной алфавит Σ.

Разобьем формулу $\phi_{E,k}$ на несколько составных частей.

$$\phi_{E,k} = AcceptExamples \wedge Projection \wedge Unambiguous \wedge Total. \tag{1}$$

Формула $AcceptExamples$ проверяет, что каждый входной/выходной пример w будет допускаться автоматом.

$$isFinal(\delta(\delta(..\delta(q_0, w_1).., w_{n-1}), w_n)) = 1 \tag{2}$$

Формула $Projection$ проверяет, что δ_{in} действительно является проекцией δ на алфавит Σ. Формула $Unambiguous$ проверяет, что проекция автомата на алфавит Σ является недвойственной. Сделать это, не прибегая к использованию кванторов, можно следующим образом. Будем строить множество пар, замкнутое в себе относительно операции добавления новой пары. А именно, эта операция берет любую пару (q_1, q_2) из множества, просматривает все символы a из алфавита Σ, добавляет в множество новую пару (q_3, q_4), такую что $\delta_{in}(q_1, a, q_3)$ и $\delta_{in}(q_2, a, q_4)$. Данное множество начинаем строить следующим образом: просмотрим все достижимые состояния q в графе, просмотрим все символы перехода a из них и добавим в множество пары (q_1, q_2) такие, что $\delta_{in}(q, a, q_1)$ и $\delta_{in}(q, a, q_2)$. Таким образом, чтобы проверить недвойственность, нужно просмотреть все пары множества и проверить, что для любой пары (q_1, q_2) не более чем одно состояние из пары является допускающим или должно выполняться

$$\neg(isFinal(q_1) = 1 \wedge isFinal(q_2) = 1). \tag{3}$$

Формула $Total$ проверяет полноту автомата A. Обозначим за A' проекцию A на алфавит Σ. Утверждается следующее: чтобы A был полным необходимо и достаточно, чтобы A' допускал все слова из Σ^l для всех $0 \leq l \leq |Q|$. Можно посчитать, сколько слов длины l допускает каждое из состояний. Тогда автомат будет являться полным, если для каждого $0 \leq l \leq |Q|$.

$$\Sigma_{q \in Q} isFinal(q) * c_{l,q} = |\Sigma|^l, \tag{4}$$

где $c_{l,q}$ – число слов длины l, оканчивающихся в состоянии q.

4 Решение с помощью SAT-решателя

Отметим, что применение SMT-решателя для решения поставленной задачи не является обоснованным. Теперь приведем решение, основанное на применении SAT-решателя. Как и раньше, обозначим за Q множество состояний, за $E = \{ex_1, ex_2...ex_{|E|}\}$ – множество входных/выходных примеров, определим переменные $\delta(i, a, j)$, где $i, j \in Q$, $a \in (\Sigma \times \Gamma)$, означающие, что между состоянием i и состоянием j есть переход по символу a, если $\delta(i, a, j) = 1$, переменные $\delta_{in}(i, b, j)$, где $i, j \in Q$, $a \in \Sigma$, которые являются проекциями переходов δ на алфавит Σ, и переменные $isFinal(i)$, где $i \in Q$, означающие, что i – допускающее состояние, если $isFinal(i) = 1$. (В дальнейшем «добавление булевой формулы» будет означать, что следующая формула будет присоединена к окончательной формуле при помощи конъюнкции «$\wedge$». Изначально окончательная формула будет равна $\mathbf{1}$).

Для начала добавим следующие булевы формулы.

Пусть $i, j, k \in Q, j \neq k$, $a \in (\Sigma \times \Gamma)$. Тогда

$$\neg\delta(i, a, j) \vee \neg\delta(i, a, k) \tag{5}$$

Эти формулы ограничивают автомат таким образом, чтобы по символу a из состояния i было не более одного перехода в другие состояния.

Попробуем перенести формулу (1), приведённую в решении авторов для SMT-решетеля, в SAT-решатель.

$$\phi_{E,k} = AcceptExamples \wedge Projection \wedge Unambiguous \wedge Total \tag{6}$$

Каждую из составных частей $\phi_{E,k}$ приведем к булевой формуле.

4.1 AcceptExamples

Введем булевы переменные $t_{i,l,k}$, которые означают, что после просмотра первых l символов примера с номером i, ODA окажется в состоянии с номером k. Тогда, чтобы проверить, что ODA допускает все примеры, добавим следующие булевы выражения:

Пусть $k \in Q$, $ex_i \in E$, $0 \leq l < |ex_i|$, $a \in (\Sigma \times \Gamma)$ – l-ый символ примера ex_i. Тогда добавим формулы:

$$t_{i,l,k} \implies \bigvee_{j \in Q} (\delta(k, a, j) \wedge t_{i,l+1,j}) \tag{7}$$

(Если автомат, после промотра его первых l символов примера находился в состоянии k, то оно после перехода по символу a, автомат должен находиться в состоянии j)

Пусть $k \in Q$, $ex_i \in E$, $l = |ex_i|$. Тогда

$$t_{i,l,k} \implies isFinal(k) \tag{8}$$

(Это условие заставляет допускать примеры).

Пусть $ex_i \in E$. Тогда необходимо добавить формулы:

$$t_{i,0,1} \tag{9}$$

(Не умаляя общности, считаем, что стартовое состояние – 1. Тогда ODA заканчивает в стартовом состоянии, просмотрев в любом из примеров 0 символов).

4.2 Projection

Проверить, что δ_{in} является проекцией δ не составляет особых усилий:

Пусть $i, j \in Q$, $a \in \Sigma$, $b \in \Gamma$, $(a, b) \in (\Sigma \times \Gamma)$. Тогда

$$\delta(i, (a, b), j) \implies \delta_{in}(i, a, j) \tag{10}$$

$$\delta_{in}(i, a, j) \implies \bigvee_{b \in \Sigma} \delta(i, (a, b), j) \tag{11}$$

4.3 Unambiguous

Перед тем, как проверить автомат на недвойственность, введем переменные r_i, где $i \in Q$. $r_i = 1$, если состояние i достижимо из стартового состояния. Тогда добавим формулы:

$$r_1 \tag{12}$$

$$r_i \iff \bigvee_{j \in Q, a \in \Sigma, i \neq j} \delta_{in}(j, a, i) \tag{13}$$

Проверим получающийся автомат на недвойственность. Будем действовать способом, описанным в главе «Решение с помощью SMT-решателя». Введем переменные: $\tau_{i,j}$. Если $\tau_{i,j} = 1$, то существует слово w, что пробег слова по δ_{in} мог закончиться и в состоянии i, и в состоянии j (т.к. δ_{in} – функция перехода недетерминированного конечного автомата), пройдя по различным путям. Тогда:

Пусть $i, j, k \in Q, a \in \Sigma, b, c \in \Gamma, b \neq c$. Добавим формулы:

$$r_i \wedge \delta(i, (a, b), j) \wedge \delta(i, (a, c), k) \implies \tau_{j,k} \tag{14}$$

(Если состояние i достижимо и из этого состояния есть переходы в состояния j и k, то добавляем пару (i, j). Это является «инициализацией» в описанном выше алгоритме).

Пусть $i, j, q, w \in Q, a \in \Sigma$. Тогда добавим формулы:

$$\tau_{i,j} \wedge \delta_{in}(i, a, q) \wedge \delta_{in}(j, a, w) \implies \tau_{q,w} \tag{15}$$

(Формируем и добавляем новые пары. Заметим, что в данном случае нет требований $q \neq w$ и $i \neq j$, т.к. возможна ситуация, когда слово прошло разными путями и оказалось в одном и том же состоянии).

Пусть $i, j \in Q$. Добавим последнюю булеву формулу:

$$\neg(\tau_{i,j} \wedge isFinal(i) \wedge isFinal(j)) \tag{16}$$

(У слова не должно быть двух путей, оканчивающихся в терминальном состоянии).

4.4 Total

Самое сложное – проверка, что автомат получился полный. Будем использовать конструкцию, предложеную авторами статьи, с одним лишь тем изменением, что вместо $c_{l,q}$ будем хранить список переменных, которые будут соответствовать битам в двоичной записи $c_{l,q}$. Также придется реализовать сложение двух двоичных чисел. Как упоминалось раньше, достаточно проверить, что подходят все слова из Σ^l, где $0 \leq l \leq |Q|$. Определим $Size$:

$$Size = |Q| * log_2(max(|Q|, |\Sigma|)) + 1 \tag{17}$$

Будем обозначать за $\mathbf{a}$ список переменных $a_1, a_2 ... a_{Size}$

Будем считать, что + принимает $\mathbf{a}$ и $\mathbf{b}$ и возвращает некоторый $\mathbf{d}$, где $\mathbf{d}$ – новый список перемнных(то есть переменные из этого списка еще не встречались в окончательной формуле), а также добавляет следующие формулы:

Пусть **rest** –список переменных, которые еще не встречались в окончательной формуле, $i \in [1..Size - 1]$. Тогда добавим формулы:

$$d_i \iff a_i \oplus b_i \oplus rest_i \tag{18}$$

$$rest_{i+1} \iff med(a_i, b_i, rest_i) \tag{19}$$

$$rest_1 \iff \mathbf{0} \tag{20}$$

Данные операции описывают сложение двух бинарных чисел. Напомним, что $\oplus$ от трех аргументов можно записать следующим образом:

$$q \oplus w \oplus e \equiv (q \wedge w \wedge e) \vee (\neg q \wedge \neg w \wedge e) \vee (\neg q \wedge w \wedge \neg e) \vee (q \wedge \neg w \wedge \neg e) \tag{21}$$

$med(q, w, e)$ можно записать так:

$$med(q, w, e) \equiv (q \wedge w) \vee (q \wedge e) \vee (w \wedge e) \tag{22}$$

Функцию + также можно обобщить на n списков переменных, положив

$$+ (\mathbf{a}^1, \mathbf{a}^2, ...\mathbf{a}^n) = (..((\mathbf{a}^1 + \mathbf{a}^2) + \mathbf{a}^3) + ...) + \mathbf{a}^n) \tag{23}$$

Теперь опишем операцию =, которая принимает $\mathbf{a}$ и $\mathbf{b}$. Эта операция исключительно добавляет булевы формулы:

Пусть $i \in [1..Size - 1]$. Тогда добавим формулы:

$$a_i \iff b_i \tag{24}$$

И еще потребуется операция $And(q, \mathbf{a})$, которая принимает булеву переменную и список переменных, возвращает новый список переменных $\mathbf{d}$ и добавляет формулы:

$$d_i \iff (q \wedge a_i) \tag{25}$$

Пересчитывать $\mathbf{c}_{l,q}$ будем следующим образом. Вызовем функцию + со следующими аргументами

$$And(\delta_{in}(j, a, q), \mathbf{c}_{l-1,j}) \forall j \in Q, \forall a \in \Sigma \tag{26}$$

А дальше возьмем результат этой функции **res** и применим операцию

$$\mathbf{res} = \mathbf{c}_{l,q} \tag{27}$$

Когда сделали операции из предыдущего пункта для всех l и q, необходимо проделать следующие операции для любого l:

Применим операцию + ко всем переменным вида

$$And(isFinal(q), \mathbf{c}_{l,q}) \forall q \in Q \tag{28}$$

Возьмем результат **res** предыдущей операции и применим операцию

$$\mathbf{res} = FromIntToBooleanList(|\Sigma|^l) \tag{29}$$

Функция $FromIntToBooleanList(x)$ возвращает новый список $\mathbf{d}$ и добавляет булевы формулы:

Если в двоичной записи числа x на позиции i стоит 1:

$$d_i \iff \mathbf{1} \tag{30}$$

Иначе:

$$d_i \iff \mathbf{0} \tag{31}$$

Последнее, о чем необходимо позаботиться – об «инициаллизации»: применим следующие операции.

$$\mathbf{c}_{0,1} = FromIntToBooleanList(1) \tag{32}$$

$$\mathbf{c}_{0,i} = FromIntToBooleanList(0) \tag{33}$$

где $i \in Q$ и $i \neq 1$

Таким образом мы проверим, что автомат является полным.

5 Решение с помощью QBF-решателя

Приведём также сведение задачи к QBF. QBF - задача проверки на выполнимость булевой формулы, в которой разрешено использование кванторов "для любого"($\forall$) и "существует"($\exists$), а также каждая переменная в формуле связана (находится под действием какого-либо квантора). Так как любая задача, сводящаяся к SAT, также сводится и к QBF (просто свяжем все

свободные переменные внешним квантором существования), то сведение, предложенное в прошлой секции является корректным и в данной секции. Но у нас стояла задача проверить, что реализация естественной проверки автомата на недвойственность и полноту проигрывает реализации, предложенной авторами статьи. Таким образом, возьмём часть формул, приведенных в предыдущей секции, а именно формулы из AcceptExamples и Projection. Далее приведём остальные формулы, которые нужны для корретного построения ODA.

Нам осталось добавить формулы, которые наложат на автомат ограничения недвойственности и полноты. С помощью кванторов эти ограничения могут выражены как:

$$\forall u \in \Sigma^* \exists! v \in \Gamma^* : u * v \in \mathcal{L}(A) \tag{34}$$

Теперь нужно избавиться от $\exists!$ и записать его в обычных кванторах. Тогда мы имеем:

$$\forall u \in \Sigma^* \exists v \in \Gamma^* \forall w \in \Gamma^* : u * v \in \mathcal{L}(A) \wedge (u * w \notin \mathcal{L}(A) \vee w = v) \tag{35}$$

Но теперь встаёт проблема с множеством, откуда берутся слова u, v, w, так как они получаются неограниченными по длине. Можно показать, что достаточно брать слова длиной $|Q|^2$, где $|Q|$ - количество состояний в автомате. Действительно, пусть нашлось слово длиной $l > |Q|^2$, которое показывает, что автомат либо неполный, либо двойственный, причём оно минимальное по длине. Рассмотрим ситуацию, когда автомат оказался двойственным. Тогда существует минимум два различных пути в проекции автомата, которые допускают слово. В этих путях $l + 1$ состояние. Рассмотрим в каких парах состояний оказывается префикс слова некоторой длины, следуя данными путями. Среди данных пар одна пара повторится хотя бы дважды, так как $l + 1 > |Q^2|$. Пусть эти повторяющиеся пары встретились на префиксе длины i и j. Тогда можно вырезать часть слова с i невключая по j включая, и слово продолжит показывать двойственность автомата. Тогда слово не было минимальным по длине. Противоречие. С неполнотой действуют аналогичные рассуждения.

Теперь формула приняла следующий вид:

$$\begin{aligned} \forall u \in \bigcup_{i \in 0..|Q|^2} \Sigma^i \exists v \in \bigcup_{i \in 0..|Q|^2} \Gamma^i \forall w \in \bigcup_{i \in 0..|Q|^2} \Gamma^i : \\ u * v \in \mathcal{L}(A) \wedge (u * w \notin \mathcal{L}(A) \vee w = v) \end{aligned} \tag{36}$$

Последнее, что осталось сделать - перевести все в булевы переменные. Слово можно сгенерировать таким образом: просто заменить все символы, находящиеся в нём на последовательность бит. Тогда на генерацию слова уйдёт $|Q|^2 * \lceil log_2(|\Sigma|) \rceil$ переменных для входного или $|Q|^2 * \lceil log_2(|\Gamma|) \rceil$ переменных для выходного слова. Но при таком способе генерации слова могут возникнуть проблемы из-за того, что может сгенерироваться мусор, так как количество символов, которые могут сгенерироваться, равно $2^{\lceil log_2(|\Sigma|) \rceil}$, что

может быть больше, чем $|\Sigma|$. Пусть $l = \lceil log_2(|\Sigma|) \rceil$ или $l = \lceil log_2(|\Gamma|) \rceil$. Пусть $e_l, e_{l-1}..e_2e_1$ - битовое представление $|\Sigma|$. Пусть $q_{il}, q_{il-1}..q_{i1} \forall i \in [1..|Q|^2]$ - битовые переменные, которые задают слово. Тогда добавим следующую формулу, чтобы ограничить каждый символ в слове.

$$\begin{gathered} \bigwedge_{i \in 1..|Q|^2} \neg f(q_{il}, e_l, f(q_{i(l-1)}, e_{(l-1)}, f(.., f(q_{i1}, e_1, \mathbf{0})..))) \\ f(a, b, prev) = \begin{cases} a \wedge prev, & \text{if } b = 1 \\ a \vee prev, & \text{if } b = 0 \end{cases} \end{gathered} \tag{37}$$

Рассмотрим формулу $f(a, b, prev)$ повнимательнее. Когда бит b равен единице, то, чтобы число превысило верхнюю границу, необходимо, чтобы бит a был равен единице и хвост числа превысил хвост верхней границы. Иначе, если бит b равен 0, то необходимо либо, чтобы бит a был равен единице, либо хвост числа превысил хвост верхней границы. Таким образом, если $f(q_{il}, e_l, f(q_{i(l-1)}, e_{(l-1)}, f(.., f(q_{i1}, e_1, \mathbf{0})..)))$ удовлетворимо, то число превысило верхнюю границу. Следовательно, нужно взять отрицание.

Конечная формула в очередной раз примет вид:

$$\begin{gathered} \forall l \in 1..|Q|^2 \forall \mathbf{u}_{1..l*\lceil log_2(|\Sigma|) \rceil} \exists \mathbf{v}_{1..l*\lceil log_2(|\Gamma|) \rceil} \forall \mathbf{w}_{1..l*\lceil log_2(|\Gamma|) \rceil} : \\ \neg(\mathbf{u} \leq |\Sigma|) \vee (\mathbf{u} * \mathbf{v} \in \mathcal{L}(A) \wedge (\neg(\mathbf{w} \leq |\Gamma|) \vee \mathbf{u} * \mathbf{w} \notin \mathcal{L}(A) \vee \mathbf{w} = \mathbf{v})) \end{gathered} \tag{38}$$

Теперь возникли проблемы с длиной. Количество переменных, которые генерируют слово неизменно, то есть каждый раз будет генерироваться слово длины $|Q|^2$, но необходимо, чтобы было проверено каждое слово из промежутка $0..|Q|^2$. Этого добиться не так сложно: введём переменные $l_1..l_{|Q|^2+1}$. Наложим следующие ограничения на данные переменные. Назовём эту формулу «correctLength».

$$\neg l_1 \wedge l_{|Q|^2+1} \wedge l_1 \implies l_2 \wedge .. \wedge l_{|Q|^2} \implies l_{|Q|^2+1} \tag{39}$$

Из формулы видно, что только некоторый префикс $\{l_i\}$ будет равен нулю. Будем считать длиной слова количество переменных l_i, равных нулю. Теперь все символы, которые будут иметь индекс больше, чем длина слова, будем считать мусором. Заметим, что длина слов будет из промежутка $1..|Q|^2$. Чтобы слово нулевой длины тоже допускалось, достаточно добавить формулу:

$$isFinal(0) \tag{40}$$

Добавим в формулу дополнительные переменные $\mathbf{p}athV, \mathbf{p}athW$, означающие, какой путь прошли слова $u * v$ и $u * w$ перед допуском автомата. Генерироваться пути будут способом аналогичным способу генерации слова.

Таким образом окончательно конечная формула будет иметь вид:

$$\begin{aligned}
&\forall \mathbf{l}_{1..|Q|^2+1}, \mathbf{u}_{1..|Q|^2*\lceil log_2(|\Sigma|)\rceil} \\
&\exists \mathbf{v}_{1..|Q|^2*\lceil log_2(|\Gamma|)\rceil}, \mathbf{pathV}_{1..(|Q|^2+1)*\lceil log_2(|Q|)\rceil} \\
&\forall \mathbf{w}_{1..(|Q|^2+1)*\lceil log_2(|\Gamma|)\rceil} \mathbf{pathW}_{1..|Q|^2*\lceil log_2(|Q|)\rceil} : \\
&\neg correctLength(\mathbf{l}) \vee \neg(\mathbf{u} \leq |\Sigma|) \vee ((\mathbf{u} * \mathbf{v}, \mathbf{pathV}) \in \mathcal{L}(A) \wedge \\
&(\neg(\mathbf{w} \leq |\Gamma|) \vee \neg(\mathbf{pathW} \leq |Q|) \vee (\mathbf{u} * \mathbf{w}, \mathbf{pathW}) \notin \mathcal{L}(A) \vee \\
&(\mathbf{w} = \mathbf{v} \wedge \mathbf{pathV} = \mathbf{pathW})))
\end{aligned} \tag{41}$$

Для начала реализуем функцию «equalSym», которая сравнивает два символа a и b. a_i, b_i - их битовые записи. Предполагаем, что $|a| = |b|$

$$a_1 \iff b_1 \wedge a_2 \iff b_2 \wedge .. \wedge a_{|a|} \iff b_{|a|} \tag{42}$$

Реализация функции $\mathbf{w} = \mathbf{v}$ очевидна.

$$(l_1 \vee equalSym(w_1, v_1)) \wedge .. \wedge (l_{|Q|^2} \vee equalSym(w_{|Q|^2}, v_{|Q|^2})) \tag{43}$$

Если индекс символов меньше длины слова, то $l_i = \mathbf{0}$, тогда нужно чтобы a_i было равно b_i. Иначе $l_i = \mathbf{1}$ и все такие скобки обращаются в $\mathbf{1}$. Реализация $\mathbf{pathV} = \mathbf{pathW}$ похожа на преализацию предыдущей функции за одним лишь тем исключением, что нужно сдвинуть индексы в $\mathbf{pathV}$ и $\mathbf{pathW}$ на единицу относительно индексов $\mathbf{l}$, то есть:

$$\begin{aligned}
&(l_1 \vee equalSym(pathW_2, pathV_2)) \wedge .. \\
&\wedge (l_{|Q|^2} \vee equalSym(pathW_{|Q|^2+1}, pathV_{|Q|^2+1})) \wedge \\
&equalSym(pathW_1, pathV_1)
\end{aligned} \tag{44}$$

Теперь попытаемся сконструировать функцию $(\mathbf{u} * \mathbf{v}, \mathbf{pathV}) \in \mathcal{L}(A)$. Пусть «bin» - функция, которая сопоставляет числу/символу его бинарное представление. Добавим следующие формулы:

$$\begin{aligned}
&\forall k \in 1..|Q|^2 \\
l_k \vee \bigvee_{i \in 1..|Q|, j \in 1..|Q|, a \in \Sigma, b \in \Gamma} &(\sigma_{i(a,b)j} \wedge \\
&equalSym(pathV_k, bin(i)) \wedge \\
&equalSym(u_k, bin(a)) \wedge \\
&equalSym(v_k, bin(b)) \wedge \\
&equalSym(pathV_k, bin(j)))
\end{aligned} \tag{45}$$

Эта формула проверяет путь на корректность. Действительно, как всегда отсекаем мусор с помощью $(l_k \vee ..)$. Чтобы путь был корректный, между любыми двумя состояниями $pathV_k, pathV_{k+1}$ на пути должно быть ребро с корректным символом (u_k, v_k), что собственно и проверяем. Последнее,

что осталось, - проверить, что последнее состояние на пути допускающее. Добавим формулы:

$$\forall k \in 1..|Q|^2$$
$$l_k \vee \neg l_{k+1} \vee \bigvee_{i \in 1..|Q|} equalSym(pathV_k, bin(i)) \wedge isFinal(i) \tag{46}$$

Первые 2 дизъюнкта в этой формуле будут равны нулю только в момент перехода переменных l_i с нулей на единицы. Следовательно, в этот момент нужно проверить, что состояние, соответствующее концу пути, является допускающим.

6 Реализация

Сведение данной задачи к SAT и QBF осуществлялось на языке Kotlin. Также использовались SAT-решатель BumbleBee (http://amit.metodi.me/research/bee/) и QBF-решатель Quabs (https://www.react.uni-saarland.de/tools/quabs/). Для реализации работы с целыми числами использовался инструмент, переводящий операции с ними в формат, приемлемый для SAT-решателя (https://github.com/ctlab/beepp2bee). Также пришлось добавить предикаты, реализующие метод нарущения симметрии в автомате на основе BFS (https://arxiv.org/abs/1602.05028). Реализацию можно увидеть по ссылке http://github.com/slelaron/odaAutomaton/.

7 Результаты

Сведение к QBF оказалось бесполезной тратой времени, так как уже при поиске автомата с минимальным количеством вершин равным трем, QBF-решатель не завершался, что не удивительно. По сравнению со сведением к SAT, в QBF используется четыре поверхностных квантора. Ощутимого прироста к скорости предикаты нарушения симметрии не дали.