ПРЕПРИНТЫ ПОМИ РАН

Учредитель: Санкт-Петербургское отделение Математического института им. В. А. Стеклова Российской академии наук

Свидетельство о регистрации средства массовой информации: ЭЛ №ФС 77-33560 от 16 октября 2008 г. Выдано Федеральной службой по надзору в сфере связи и массовых коммуникаций

Контактные данные: 191023, г. Санкт-Петербург, наб. реки Фонтанки, дом 27

телефоны:(812)312-40-58; (812) 571-57-54

e-mail: admin@pdmi.ras.ru

http://www. pdmi.ras.ru /preprint/

Заведующая информационно-издательским сектором Симонова В.Н

ПРЕПРИНТ ПОМИ — 10/2009

ВЕРХНЯЯ ОЦЕНКА ДЛЯ ЗАДАЧИ CIRCUIT SAT

С. Нурк

Санкт-Петербургский государственный университет

http://sergey.nurk.ru

01 декабря 2009 г.

Аннотация

В данной работе приводится алгоритм для решения задачи *Circuit SAT*, время работы которого составляет $O(2^{0.4058m})$, где m – количество гейтов во входной схеме.

НИР проведена в рамках реализации ФЦП “Научные и научно-педагогические кадры инновационной России” на 2009–2013 годы, госконтракт П265 от 23.07.2009.

1 Введение

Задача о выполнимости булевой схемы в полном бинарном базисе (*Circuit SAT*), наряду с ее частным случаем – задачей о выполнимости формулы в конъюнктивной нормальной форме (*SAT*), является одной из наиболее фундаментальных задач теоретической информатики. Но если на время решения различных разновидностей задачи *SAT* было доказано большое количество верхних оценок (см. [2]), то ни одной нетривиальной верхней оценки на время решения *Circuit SAT*, насколько известно автору, до сих пор не доказано. [1]

Определим *Circuit SAT* более формально. *Булева схема* представляет собой конечный ориентированный ациклический граф с вершинами, имеющими входящую степень 0 или 2 и единственной вершины исходящей степени 0. Вершины, имеющие входящую степень 0, соответствуют переменным $\{x_1, \ldots, x_n\}$ и называются *входами*. Каждая из вершин с входящей степенью 2 помечена некоторой булевой функцией двух переменных и называется *гейтом*. Вершина с исходящей степенью 0 соответствует *выходу* схемы. *Размером схемы* называется число гейтов, из которых она состоит (в дальнейшем будем обозначать его через m). Схема называется *выполнимой*, если существует набор значений переменных, на котором значение на выходе схемы будет 1. Решить задачу *Circuit SAT* – значит ответить на вопрос, является ли данная схема выполнимой.

Очевидная оценка на время решения поставленной задачи составляет (с точностью до полиномиального сомножителя) 2^n, где n – число входов. Так же, как и для задачи *SAT*, подходов к получению верхних оценок вида c^n $(c < 2)$ в общем случае неизвестно. Однако, если размер схемы невелик относительно количества входных переменных, то в этом случае интерес представляют и оценки вида c^m, где m – количество гейтов. Мы докажем такую оценку посредством предъявления рекурсивного алгоритма, время работы которого составляет $O(2^{0.4058m})$.

Очевидно, что существует 2^4 различных типов бинарных связок:

- две константные функции и четыре функции, зависящие только от одной из переменных, мы будем называть вырожденными;
- восемь функций вида $((x \oplus a) \wedge (y \oplus b)) \oplus c$, где $a, b, c \in \{0, 1\}$, назовем функциями $\wedge$-типа;
- две функции вида $x \oplus y \oplus a$, где $a \in \{0, 1\}$, будем называть функциями $\oplus$-типа.

Основное различие последних двух типов функций заключается в том, что функцию $\wedge$-типа всегда можно сделать константной, присвоив константное значение любому ее входу (например, подставив a на входе x, на выходе гейта мы получим константу c; в таком случае мы говорим, что гейт стал тривиальным), а для функций $\oplus$-типа это невозможно.

На рисунках $\oplus$ и $\wedge$ обозначают любую связку из соответствующего класса.

Будем называть гейт, помеченный костантной функцией, *константным гейтом*.

Предложение 1. *Любой гейт, вычисляющий вырожденную функцию, может быть удалён из схемы (исключение составляет лишь костантный гейт, если он является выходом схемы, но этот случай в дальнейшем будет учтен отдельно).*

Доказательство. Из соображений удобства введем новую фиктивную переменную z, которую в дальнейшем будем использовать в качестве незначимого входа для гейтов, помеченных вырожденными функциями.

Чтобы удалить из схемы константный гейт B, не являющийся выходом схемы, нужно заменить функции, вычисляемые в его потомках, "подходящими" вырожденными функциями. Теперь можно удалить гейт B, соединив при этом фиктивную переменную z со всеми его потомками.

Теперь рассмотрим гейты, помеченные функциями, зависящими лишь от одной переменной. Назовем *существенным предком* такого гейта ту вершину схемы, от которой зависит вычисляемое в

[1] Когда автор заканчивал работу над статьей, в частной переписке ему стало известно о недавней публикации Calabro, Impagliazzo and Paturi [1]. В своей статье авторы приводят вероятностный алгоритм, который решает задачу *Circuit SAT* для схем глубины d от n переменных и не более cn AND/OR/NOT гейтов произвольной входящей степени. Время его работы ограничено величиной $2^{(1-\delta)n}$, где $\delta \geq 1/O(c^{2^{d-2}-1} \lg^{3*2^{d-2}-2} c)$, а постоянная в O зависит только от d.

данном гейте значение. Допустим, интересующий нас гейт не является выходом схемы. Тогда мы можем удалить этот гейт, соединив его существенного предка со всеми его непосредственными потомками, при необходимости поменяв функции (может потребоваться "внести" в них отрицание), вычисляющиеся в последних. Теперь предположим, что мы имеем дело с выходом схемы. Обозначим его существенного предка через G. Удалим все гейты схемы, в которые есть путь из G, после чего сделаем G новым выходом схемы. При этом, если вершина G соответствует входной переменной, то полученная схема также, как и исходная, является выполнимой, а если G является гейтом, то нам, возможно, придется поменять функцию, вычисляемую в нем (как и в предыдущем случае, может потребоваться "внести" отрицание). □

Для схемы C и ее переменной x мы через $C[x = a]$, $a \in \{0, 1\}$, будем обозначать схему, полученную подстановкой константного гейта, соответствующего a, на место переменной x в схему C.

Теперь неформально опишем наш алгоритм, подробно изложенный в следующем разделе.

Данный алгоритм имеет много общего с типичными расщепляющими алгоритмами решения задачи SAT (см. [2], [3]), а также использует технику $\oplus$-цепей, впервые предложенную Полом для доказательства нижних оценок схемной сложности некоторых булевых функций (см. [6]).

- Получив на вход булеву схему, алгоритм в первую очередь упрощает ее, избавляясь от гейтов, не участвующих в вычислениях, и гейтов, вычисляющих вырожденные функции.
- Если ответ не был найден уже на этапе упрощения, алгоритм пытается найти переменную, по которой было бы "выгодно" расщепляться (здесь "выгода" определяется количеством гейтов, от которых нам удалось бы избавиться при подстановке конкретных значений вместо выбранной переменной).

  Если такую переменную удается найти, то происходят рекурсивные вызовы алгоритма на схемах, полученных подстановкой конкретных значений на место этой переменной. После чего алгоритм возвращает дизъюнкцию результатов этих вызовов. Если подходящей переменной найти не удалось, то алгоритм перестраивает схему таким образом, чтобы в новой схеме такая переменная гарантировано нашлась бы, после чего повторяется поиск переменной для расщепления, а затем и рекурсивные вызовы алгоритма.

2 Алгоритм

Алгоритм Circuit SAT
Вход: схема C.
Выход: **True**, если схема выполнима, и **False** в противном случае.

1: **Return** Split(Reduce(C)).

Функция Split
Вход: схема C, полученная в результате работы функции Reduce.
Выход: **True**, если схема выполнима, и **False** в противном случае.

1: Если выход схемы C является константным гейтом, то вернуть его значение.
2: Выбрать переменную x, удовлетворяющую одному из следующих условий:

- ее исходящая степень не меньше трех;
- ее исходящая степень – два, при этом она имеет потомка $\wedge$-типа.

Замечание. О том, что такая переменная будет найдена, заботится функция Reduce.
3: **Return** Split(Reduce($C[x = 0]$)) $\vee$ Split(Reduce($C[x = 1]$))

Перед описанием функции Reduce(C) дадим следующее определение, которое было впервые введено Полом [6]. Для гейта G_0 схемы C мы говорим, что существует $\oplus$-*цепь* длины k в G_0 тогда и только тогда, когда есть k гейтов $G_1, \ldots, G_k$ в C таких, что

1. Для $1 \leq i \leq k$, G_i является гейтом $\oplus$-типа;
2. Для $1 \leq i \leq k$, G_i является единственным потомком G_{i-1};
3. В G_k нет $\oplus$-цепи, то есть G_k либо является выходом схемы, либо имеет исходящую степень не менее чем 2, либо его единственный потомок является гейтом $\wedge$-типа.

Функция Reduce
Вход: схема C.
Выход: схема C', которая:

- является выполнимой в том и только в том случае, когда схема C выполнима;
- является константным гейтом или содержит переменную, удовлетворяющую условию из второго шага функции Split(C);
- имеет размер не больше, чем размер C.

Шаг 1. Пока выход C не является константным гейтом, и в C есть гейт, соответствующий вырожденной функции, исключить из схемы этот гейт.

Шаг 2. Последовательно удалить из схемы все гейты, не являющиеся выходом схемы и имеющие исходящую степень 0.

Шаг 3. Если выход C есть константный гейт, то вернуть схему, состоящую только из него. Если выход C соответствует входящей переменной , то вернуть константный гейт **True**.

Шаг 4. Если в C есть переменная исходящей степени больше двух, то вернуть C.

Шаг 5. Если в C есть переменная исходящей степени два, имеющая потомка $\wedge$-типа, то вернуть C.

Шаг 6. Если в C есть переменная x исходящей степени два, то оба ее потомка – гейты $\oplus$-типа. Обозначим их через P_0 и R_0, после чего рассмотрим $\oplus$-цепи $P_1, \ldots, P_p$ и $R_1, \ldots, R_r$, начинающиеся в P_0 и R_0 соответственно (в случае, если соответствующие $\oplus$-цепи пусты, p и r могут оказаться равными нулю).

Возможны два случая:

- **Случай 1.** Множества гейтов $P_0, P_1, \ldots, P_p$ и $R_0, R_1, \ldots, R_r$ не имеют общих элементов (см. рис.1а). Так как в C нет циклов, то мы можем считать, что в схеме нет пути из R_r в P_p (в противном случае переобозначим $\oplus$-цепи). В этом предположении покажем, что из x нет пути в L_i $(0 \leq i \leq p)$. Действительно, путь из x в L_i должен бы был пройти через R_r или P_p, что в первом случае привело бы к наличию пути из R_r в P_p, а во втором – к существованию цикла в схеме.

  Покажем, что гейт P_p не является выходом схемы. Предположим, что это не так, но тогда, в силу отсутствия пути из R_r в P_p, в схеме должен существовать гейт исходяшей степени 0, не являющийся выходом схемы. На **шаге 2** все такие гейты были исключены из схемы, а значит, мы пришли к противоречию.

  Итак, либо гейт P_p имеет исходящую степень не менее чем 2, либо его единственный потомок является гейтом $\wedge$-типа. Обозначим значение выхода гейта P_p через y. Тогда

$$y = x \oplus L_0 \oplus L_1 \oplus \cdots \oplus L_p \oplus a, \tag{1}$$

  где a равно 0 или 1 в зависимости от того, какие функции $\oplus$-типа находятся в гейтах $\oplus$-цепи. авенство (1) равносильно

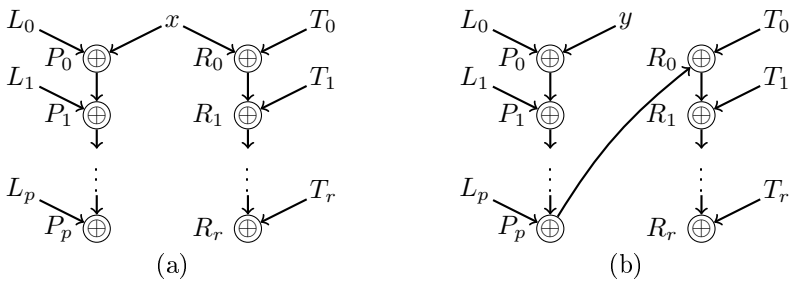

Рис. 1: Случай непересекающихся $\oplus$-цепей.

$$x = y \oplus L_0 \oplus L_1 \oplus \cdots \oplus L_p \oplus a. \tag{2}$$

Так как x не входит в L_i, то правая часть равенства (2) не зависит от x. Это позволяет преобразовать схему следующим образом:

1. Заменить переменную x на переменную y (делаем этот шаг просто чтобы соответствовать нашим обозначениям).
2. Убрать все ребра, выходящие из гейта P_p, а также ребро между переменной y и гейтом R_0 (см. рис.1б).
3. Соединить y со всеми гейтами, в которые раньше вели ребра из гейта P_p.
4. Соединить гейт P_p с гейтом R_0 (см. рис.1б).

Теперь остается вернуть получившуюся схему в качестве результата функции $\textsc{Reduce}(C)$, так как она удовлетворяет всем требованиям на схему C', заявленным в начале описания функции $\textsc{Reduce}$. Ее размер не больше размера исходной схемы C, C' выполнима, только если C выполнима, и содержит переменную y исходящей степени не меньше 2, которая либо имеет потомка $\wedge$-типа, либо ее исходящая степень больше 2 (поскольку либо исходящая степень гейта P_p была не меньше 2, либо его единственный потомок был гейтом $\wedge$-типа).

- **Случай 2.** P_k совпадает с R_m при некоторых $0 \le k \le p$ и $0 \le m \le r$ (см. рис.2а).

  В этом случае значение на выходе гейта P_k равно

  $$x \oplus x \oplus L_0 \oplus \cdots \oplus L_{k-1} \oplus T_0 \oplus \cdots \oplus T_{m-1} \oplus a = L_0 \oplus \cdots \oplus L_{k-1} \oplus T_0 \oplus \cdots \oplus T_{m-1} \oplus a,$$

  где a равно 0 или 1 в зависимости от того, какие функции $\oplus$-типа находятся в гейтах $\oplus$-цепей. Таким образом значение на выходе гейта P_k (а вместе с ним и значение выхода схемы C) не зависит от значения переменной x. Это дает нам возможность заменить рассматриваемую подсхему схемой, вычисляющей $L_0 \oplus \cdots \oplus L_{k-1} \oplus T_0 \oplus \cdots \oplus T_{m-1} \oplus a$, которая будет содержать по крайней мере на 1 гейт меньше исходной (см. рис.2б). Вернуться на **шаг 1**.

Шаг 7. Заменить произвольный гейт верхнего уровня схемы C (гейт, в который входят лишь переменные) на новую входную переменную. Вернуться на **шаг 1**.

Действительно, раз мы дошли до этого шага, значит в схеме не осталось переменных, имеющих исходящую степень больше 1. Вместе с этим заменяемый гейт не константный, так как все вырожденные гейты из нашей схемы были исключены на **шаге 1** (к появлению в схеме новых вырожденных гейтов мог привести только лишь **шаге 6**, но после него мы либо выходим из функции, либо вновь возвращаемся на **шаг 1**). Значит описанное выше преобразование не повлияет на выполнимость схемы, уменьшив при этом ее размер.

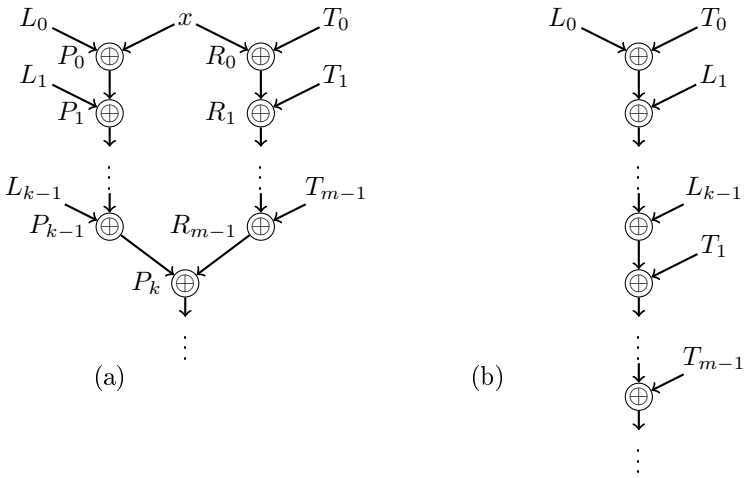

Рис. 2: Случай пересекающихся $\oplus$-цепей.

3 Верхняя оценка

Проведем анализ приведенного выше алгоритма.

Вначале схема C размера m передается на вход функции Reduce, в которой происходит ее упрощение. Ясно, что время работы функции Reduce оценивается некоторым полиномом от m. Стоит отметить, что в результате работы функции Reduce в схеме не остается заведомо "бесполезных" гейтов (гейтов исходящей степени 0, не являющихся выходом). Если в процессе выполнения функции Reduce устанавливается факт выполнимости/невыполнимости схемы, то мы сигнализируем об этом, возвращая тривиальную схему из одного константного гейта, соответствующего ответу да/нет.

Полученная в результате работы функции Reduce схема передается в функцию Split, где происходит сведение задачи к аналогичной задаче для двух более "простых" схем. А именно: предположим, что нами для "расщепления" выбрана переменная x. Мы составляем две новые схемы: $C[x = 0]$ и $C[x = 1]$, рекурсивно вызываем на них наш алгоритм (функции Reduce и Split) и возвращаем дизъюнкцию результатов. При подстановке на место переменной x константного гейта, размер схемы увеличивается на 1, с другой стороны на первом же шаге функции Reduce этот гейт будет исключен из схемы, поэтому далее считается, что этот гейт просто не учитывается при подсчете размера схемы.

Если исходящая степень переменной x больше двух, то можно утверждать, что перед каждым из рекурсивных вызовов функции Split, на первом же шаге работы функции Reduce($C[x = a]$) ($a \in \{0, 1\}$), либо размер соответствующей схемы уменьшится хотя бы на три, либо ее выход окажется константным гейтом.

Теперь рассмотрим случай, когда исходящая степень x равна двум, при этом один из ее потомков – гейт $\wedge$-типа, а другой – гейт $\oplus$-типа (обозначим их A и O соответственно). Из того, что исходящая степень x равна двум, следует, что на первом же шаге работы функции Reduce($C[x = a]$), либо размер соответствующей схемы уменьшится хотя бы на два, либо ее выход окажется константным гейтом. Чтобы добиться лучшей оценки, придется разобрать несколько случаев.

Случай 1. Гейт O не является непосредственным потомком гейта A.

Случай 1.1. Гейт A не является выходом схемы. Подставляя вместо переменной x константу, делающую гейт A тривиальным, мы избавляемся по меньшей мере от трех гейтов.

Случай 1.2. Гейт A является выходом схемы. Подставляя вместо переменной x константу, делающую гейт A тривиальным, мы также однозначно определяем выход схемы.

Случай 2. Гейт O является непосредственным потомком гейта A (см.рис.3).

Случай 2.1. Гейт O не является выходом схемы. Подставляя вместо переменной x константу,

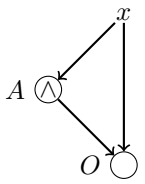

Рис. 3:

делающую гейт A тривиальным, мы также однозначно определяем выход гейта O и вновь избавляемся по меньшей мере от трех гейтов.

Случай 2.2. Гейт O является выходом схемы. Подставляя вместо переменной x константу, делающую гейт A тривиальным, мы также однозначно определяем выход гейта O, а вместе с ним и выход схемы.

В каждом из описанных случаев при подстановке некоторой константы либо размер схемы уменьшается по крайней мере на три, либо схема становится тривиальной.

Теорема 3.1. *Время работы приведенного алгоритма ограничено сверху величиной* $O(2^{0.4058m})$*, где* m *– размер входной схемы.*

Доказательство. Обозначим через $f(m)$ максимальное среди всех схем размера m количество листьев в дереве рекурсивных вызовов описанного алгоритма.

Приведенные выше соображения приводят нас к следующему рекуррентному соотношению:

$$f(m) \leq f(m-2) + f(m-3), \tag{3}$$

для $m \geq 3$.

Чтобы оценить $f(m)$, воспользуемся стандартной для подобных задач техникой, предложенной Куллманном и Лукхардтом (см. [5], [4]). Составим уравнение, соответствующее рекуррентному соотношению (3):

$$1/x^3 + 1/x^2 = 1 \Longleftrightarrow x^3 - x - 1 = 0. \tag{4}$$

Его единственный вещественный корень $\tau \approx 1.32472$. Утверждается (соответствующий факт доказан в [4]), что тогда

$$f(m) \leq poly(m)\tau^m \leq O(2^{0.4058m}).$$

Так как мы имеем дело с двоичным деревом, то количество всех его вершин не превосходит $2f(m)$. Нетрудно понять, что время, проведенное в каждой из вершин дерева вызовов, также как и время работы функции Reduce, оценивается некоторым полиномом от m. Все вместе приводит нас к заявленной верхней оценке. □

Благодарности

Автор благодарит своего научного руководителя Эдуарда Алексеевича Гирша, обратившего его внимание на рассматриваемую в работе задачу, а также Александра Куликова и Криса Калабро за их ценные замечания.

Список литературы

[1] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. 2009.

[2] Evgeny Dantsin and Edward A. Hirsch. Worst-case upper bounds. In *Handbook of Satisfiability*. IOS Press, 2009.

[3] E. A. Hirsch. New worst-case upper bounds for SAT. *Journal of Automated Reasoning*, 24(4):397–420, 2000.

[4] O. Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science*, 223 (1-2):1–72, 1999.

[5] O Kullmann and H Luckhardt. Deciding propositional tautologies: Algoriithms and their complexity. 1997.

[6] Wolfgang J. Paul. A $2.5n$-lower bound on the combinational complexity of Boolean functions. *SIAM Journal of Computing*, 6(3):427–433, 1977.