УДК 004.056.55; 004.832.25 doi 10.26089/NumMet.v20r106

ДУБЛИКАТЫ КОНФЛИКТНЫХ ОГРАНИЧЕНИЙ В CDCL-ВЫВОДЕ И ИХ ИСПОЛЬЗОВАНИЕ В ЗАДАЧАХ ОБРАЩЕНИЯ НЕКОТОРЫХ КРИПТОГРАФИЧЕСКИХ ФУНКЦИЙ

В. С. Кондратьев[1], А. А. Семенов[2], О. С. Заикин[3]

Изучен феномен повторного порождения конфликтных ограничений SAT-решателями в процессе работы с трудными экземплярами задачи о булевой выполнимости. Данный феномен является следствием применения эвристических механизмов чистки конфликтных баз, которые реализованы во всех современных SAT-решателях, основанных на алгоритме CDCL (Conflict Driven Clause Learning). Описана новая техника, которая позволяет отслеживать повторно порождаемые дизъюнкты и запрещать их последующее удаление. На базе предложенных технических решений построен новый многопоточный SAT-решатель (SAT, SATisfiability), который на ряде SAT-задач, кодирующих обращение криптографических хеш-функций, существенно превзошел по эффективности многопоточные решатели, занимавшие в последние годы высокие места на специализированных соревнованиях.

Ключевые слова: задача о булевой выполнимости (SAT), алгоритм CDCL, конфликтные дизъюнкты, криптографические хеш-функции.

1. Введение. Проблема булевой выполнимости (Boolean Satisfiability Problem), сокращенно обозначаемая SAT, является одной из центральных комбинаторных проблем ввиду своей тесной связи со многими областями современной теории алгоритмов и теории вычислительной сложности. В общей постановке SAT является NP-трудной задачей (NP-hard, Non-deterministic Polynomial-time hard); поэтому для ее решения не может существовать полиномиальных алгоритмов в предположении, что $P \neq NP$. Однако вопрос разработки алгоритмов, эффективных на подклассах SAT, связанных с различными практическими областями, не теряет своей актуальности вот уже более 20 лет. Сказанное обусловлено тем фактом, что SAT очень удобна с точки зрения сведения к ней обширнейшего класса комбинаторных проблем. Так, в силу теоремы Кука–Левина [1, 2] любая задача из класса NP сводится к SAT в ее распознавательном варианте. В последние годы был предложен целый ряд концепций решения SAT, однако лишь немногие из них получили всеобщее признание. В этом контексте уместно выделить CDCL (Conflict Driven Clause Learning) [3, 4], SLS (Stochastic Local Search) [5, 6] и LA-SAT (Look-Ahead based SAT) [7]. Следует отметить, что SLS-алгоритмы относятся к "неполным" методам, поскольку решение о невыполнимости КНФ (конъюнктивная нормальная форма), выдаваемое таким алгоритмом, в лучшем случае будет верным лишь с некоторой вероятностью. Алгоритмы, основанные на Look-Ahead-идеологии, сами по себе являются весьма слабыми и используются, как правило, в связке с CDCL, например для осуществления декомпозиции исходной трудной SAT-задачи. Такой подход используется, в частности, при реализации известного принципа Cube-and-Conquer [8, 9]. Подавляющее большинство современных SAT-решателей, используемых при работе с "индустриальными" тестами, основаны на концепции CDCL. Особо отметим, что на таких аргументированно трудных тестах, которыми являются задачи криптоанализа, нетривиальных результатов удается добиться только с использованием CDCL-решателей [10, 11].

В настоящей статье основным объектом исследований являются SAT-задачи, кодирующие обращение криптографических хеш-функций. Отталкиваясь от этих задач, мы исследуем феномен повторного порождения конфликтных ограничений в процессе CDCL-вывода. Данный феномен возникает вследствие использования процедур чистки конфликтных баз. Такие процедуры применяются во всех современных CDCL-решателях. Основной результат работы состоит в новой технике использования повторно порождаемых конфликтных ограничений. Применение этой техники позволило существенно ускорить решение SAT-задач, кодирующих проблемы поиска прообразов некоторых криптографических хеш-функций.

[1] Иркутский национальный исследовательский технический университет, ул. Лермонтова, 83, 664074, г. Иркутск; магистрант, e-mail: vikseko@gmail.com

[2] Институт динамики систем и теории управления имени В. М. Матросова СО РАН, ул. Лермонтова, 134, 664033, г. Иркутск; зав. лабораторией, e-mail: biclop.rambler@yandex.ru

[3] Институт динамики систем и теории управления имени В. М. Матросова СО РАН, ул. Лермонтова, 134, 664033, г. Иркутск; ст. науч. сотр., e-mail: zaikin.icc@gmail.com

Кратко опишем структуру статьи. Следующий раздел (раздел 2) является вводным — в нем формулируются базовые понятия, дается краткий обзор подходов и алгоритмов, используемых как при решении SAT, так и при сведении к SAT различных комбинаторных задач. В разделе 3 описывается феномен повторного порождения конфликтных дизъюнктов, который наблюдается при использовании современных SAT-решателей, основанных на алгоритме CDCL. Данный феномен является следствием необходимости периодической чистки базы конфликтных дизъюнктов — соответствующие механизмы предусмотрены во всех современных CDCL-решателях. Кроме того, в разделе 3 мы описываем структуру многопоточного CDCL-решателя RADAR (Repeatable clAuses Driven strAtegy solveR), в котором предусмотрен механизм выявления повторно порождаемых дизъюнктов. В дальнейшем эти дизъюнкты становятся частью исходной КНФ и не удаляются на этапах чистки конфликтной базы. В разделе 4 мы приводим результаты вычислительных экспериментов. Мы демонстрируем существенное превосходство RADAR перед несколькими многопоточными решателями, занимавшими высокие места на соревнованиях из серии SAT-competition в последние годы. Сравнение проводилось на классах SAT-задач, кодирующих криптоанализ некоторых криптографических хеш-функций.

2. Базовые понятия, подходы и алгоритмы. Переменные, принимающие значения в множестве из двух элементов, обычно обозначаемом как $\{0,1\}$, называются булевыми. Пусть $X = \{x_1, \ldots, x_n\}$ — множество булевых переменных. Произвольное определенное всюду на X отображение $\alpha : X \to \{0,1\}$ задает *набор значений* переменных из X. Все возможные наборы значений переменных из X образуют множество, которое обозначается через $\{0,1\}^n$. Данное множество — это множество, которое состоит из 2^n различных двоичных слов длины n. Элементы $\{0,1\}^n$ часто называют n-мерными *булевыми векторами*. Произвольное всюду определенное отображение $f : \{0,1\}^n \to \{0,1\}$ называется *булевой функцией* (иногда добавляют "от n переменных"). Любую булеву функцию от n переменных можно задать построенным по определенным правилам словом, которое помимо переменных $x_1, \ldots, x_n$ включает в себя специальные символы, называемые *логическими связками*. Такие слова называются *булевыми формулами*, формулами исчисления высказываний [12] или формулами алгебры логики [13]. Везде далее мы используем термин "булева формула".

Пусть F — произвольная булева формула, задающая булеву функцию f над множеством булевых переменных $X = \{x_1, \ldots, x_n\}$. Формула F называется *выполнимой*, если найдется такой набор $\alpha \in \{0,1\}^n$ значений переменных из X, что $f(\alpha) = 1$. Набор α с таким свойством называется набором, *выполняющим формулу* F. Если не существует наборов, выполняющих F, то данная формула называется *невыполнимой*. Задача о булевой выполнимости (SAT) заключается в следующем: по произвольной булевой формуле F решить, является ли данная формула выполнимой. Можно показать, что SAT в отношении произвольной формулы может быть эффективно сведена к SAT в отношении формулы в конъюнктивной нормальной форме (КНФ). Напомним, что КНФ — это конъюнкция булевых формул, называемых дизъюнктами. Произвольный дизъюнкт — это формула, состоящая из литералов над множеством X, соединенных связкой "$\vee$" (дизъюнкция). Литерал же — это простейшая булева формула, которая может быть двух видов: x или $-x$ (здесь x — переменная из X, $-x$ — формула, принимающая значение 0 при $x = 1$ и принимающая значение 1 при $x = 0$). С учетом сказанного, чаще всего под "SAT" понимается проблема выполнимости в отношении формулы в КНФ.

SAT — классическая NP-полная задача (см. [1, 14]), следовательно, в предположении, что $P \neq NP$, она не может быть решена за полиномиальное время в общем случае. Однако подобно многим другим NP-полным и NP-трудным задачам, SAT решается весьма эффективно для обширных классов своих частных случаев. Алгоритмы решения SAT обычно построены на простой основе, которая естественным образом дополняется различными стратегиями и эвристиками. Это сделало алгоритмику SAT направлением, которое бурно развивается в последние 20 лет. Перечисление только ключевых статей по данной тематике потребовало бы слишком много места, поэтому сошлемся здесь на весьма объемное руководство по SAT-проблематике [15].

Прогресс в разработке практических алгоритмов решения SAT привел к развитию основанных на этих алгоритмах вычислительных методов, широко используемых на сегодня в таких областях, как символьная верификация [16, 17], биоинформатика [18, 19], построение комбинаторных структур [20], а также в криптоанализе (см., например, [21]). Настоящая статья посвящена применению алгоритмов решения SAT к задачам обращения некоторых криптографических функций.

Функции, которые преобразуют двоичные слова в двоичные слова, будем называть дискретными функциями. Нас будут интересовать дискретные функции, которые задаются алгоритмами, т.е. программами для любой из известных формальных вычислительных моделей. Далее везде в роли такой модели будем рассматривать машину Тьюринга (МТ). Через $\{0,1\}^*$ обозначим множество всех двоичных слов

произвольной конечной длины. Пусть $f : \{0,1\}^* \to \{0,1\}^*$ — произвольная всюду определенная (тотальная) дискретная функция, заданная МТ-программой M_f. Данная программа задает счетное семейство тотальных дискретных функций вида $f_n : \{0,1\}^n \to \{0,1\}^*$, $n \in N$.

Проблема поиска прообраза (или кратко "проблема обращения") произвольной функции f_n — это проблема в следующей формулировке: по произвольному $\gamma \in \mathrm{Range}\, f_n$ требуется найти такой $\alpha \in \{0,1\}^n$, что $f(\alpha) = \gamma$.

В контексте сформулированной только что проблемы можно рассматривать многие задачи криптоанализа. Действительно, для примера приведем задачу криптоанализа генератора ключевого потока (*keystream generator*). Такой генератор — это функция вида $f_n : \{0,1\}^n \to \{0,1\}^*$ для некоторого фиксированного n, называемого длиной секретного ключа. Получая на вход произвольный секретный ключ $\alpha \in \{0,1\}^n$, генератор f_n строит ключевой поток $\gamma \in \{0,1\}^m$ (как правило, $m \gg n$), который затем используется для шифрования открытого сообщения — слова $v \in \{0,1\}^m$. Шифрующее преобразование — побитовое сложение по модулю 2 булевых векторов γ и v. В современных системах поточного шифрования (например, при шифровании GSM-трафика) секретный ключ $\alpha \in \{0,1\}^n$ может использоваться для шифрования информации, объем которой многократно превышает n. Бывает так, что часть слова v становится известной противнику. В этой ситуации он может найти соответствующую часть γ, которую обозначим через γ', и рассмотреть задачу обращения функции f_n "в точке γ'". Если противнику удастся решить данную задачу, он найдет ключ α и сможет прочитать любое сообщение, созданное на данном ключе. Такого сорта атаки принято называть атаками на основе известного открытого текста (*known plaintext attacks*). Известно довольно много примеров успешных атак данного типа. В частности, генератор A5/1, использовавшийся на протяжении целого ряда лет для шифрования GSM-трафика, подвержен таким атакам [11, 22–25].

Итак, с учетом очевидного практического интереса, мы будем рассматривать далее задачи обращения функций вида $f_n : \{0,1\}^n \to \{0,1\}^m$. Известен ряд подходов к решению таких задач, объединенных следующей общей идеей: свести задачу обращения функции к задаче поиска решений некоторой системы алгебраических или булевых уравнений и использовать для решения полученной задачи алгоритмы из соответствующей области (эксплуатируя при этом различные особенности исходной задачи). В настоящей статье мы фактически работаем с системами булевых уравнений, представленными в форме SAT, и используем для их решения современные алгоритмы из соответствующей области. Данный подход относится к области, за которой в последние годы (после выхода книги [21]) закрепился термин "алгебраический криптоанализ". Основополагающий результат в данном направлении состоит в том, что задачу обращения произвольной криптографической функции можно эффективно свести к SAT. Строго мы данный факт здесь не устанавливаем, поскольку соответствующее доказательство весьма громоздко, хотя и строится на известных принципах — по сути, этот факт основан на используемой при доказательстве теоремы Кука–Левина [1] технике, позже названной символьным исполнением (*symbolic execution*) [26]. На практике для сведения к SAT задач обращения дискретных функций можно использовать специализированные трансляторы, такие, например, как Cryptol [27, 28] или URSA [29]. В наших экспериментах мы использовали транслятор Transalg [30, 31], при создании которого были учтены различные аспекты криптографической специфики.

После того как задача обращения рассматриваемой функции сведена к SAT, к полученной SAT-задаче применяется какой либо-алгоритм решения SAT. На основании большого числа экспериментов, проводившихся на протяжении ряда лет, мы заключаем, что единственная концепция, которая обеспечивает относительно эффективное решение SAT-задач, кодирующих обращение криптографических функций, — это CDCL (*Conflict Driven Clause Learning*). Кратко опишем здесь основные составляющие архитектуры современных CDCL-решателей.

В основе концепции CDCL лежит алгоритм Девиса–Патнема–Лоджмана–Лавленда (DPLL) [32, 33], являющийся одним из старейших алгоритмов в автоматическом доказательстве теорем. DPLL — это алгоритм направленного обхода дерева поиска, который использует простой бэктрекинг для выхода из тупиковых точек. Алгоритм DPLL использовался, по сути, в его исходном виде около 30 лет. Прорыв произошел в 1996 г., когда в работе [34] была реализована весьма элегантная процедура запоминания тупиковых ветвей дерева поиска в виде новых ограничений, являющихся дизъюнктами (так называемые "конфликтные дизъюнкты"). Эта техника позже получила название Conflict Driven Clause Learning (CDCL). Принципиальное отличие алгоритмов DPLL и CDCL заключается в использовании последним памяти для хранения истории поиска. Информация, содержащаяся в конфликтных дизъюнктах, в ряде случаев позволяет совершать существенно более глубокие откаты, чем это происходит при бэктрекинге. Данный эффект получил название нехронологического бэктрекинга или бэкджампинга (*backjumping*). Перечисленные факты

детально проанализированы в статье [3] с использованием специальных структур данных, названных графами вывода (*implication graphs*). Следующий важный шаг сделан в работах [35, 36], в которых описаны быстрые структуры данных для представления КНФ в памяти компьютера (*watched literals*), а также были введены процедуры выбора переменных в процессе поиска, использующие эвристические меры "полезности" выбираемых переменных. Введенные меры представляли собой динамически пересчитываемые показатели конфликтности переменных. Соответствующая техника получила название VSIDS (*Variable State Independent Decaying Sum*). SAT-решатель zchaff, появившийся в 2001 г., стал первым решателем, основанным на концепции CDCL, в котором были реализованы перечисленные техники. Именно zchaff был первым решателем, который позволял осуществлять криптоанализ некоторых генераторов ключевого потока. Довольно быстро, впрочем, выявилась негативная черта CDCL: огромная скорость генерации конфликтных дизъюнктов приводит к быстрому переполнению оперативной памяти. Использование же дисковой памяти ведет к резкому падению эффективности поиска. В этих условиях еще одной важной составной частью современных CDCL-решателей становятся процедуры чистки баз конфликтных дизъюнктов. Впервые такие процедуры были реализованы в SAT-решателе minisat [37], архитектура которого на долгие годы стала де-факто стандартом архитектуры эффективных CDCL-решателей. На сегодняшний день процедуры чистки конфликтных баз используются во всех CDCL-решателях, претендующих на высокие места в проходящих ежегодно соревнованиях SAT-решателей. Авторы некоторых решателей [38] подчеркивают "агрессивный характер" используемых процедур чистки. Поскольку все меры, оценивающие "полезность" дизъюнктов, являются эвристическими, то нет никаких гарантий, что удаленные из конфликтной базы дизъюнкты не будут выведены в ходе дальнейшего поиска. Строго говоря, данный факт противоречит заявлениям о полноте алгоритмов, основанных на CDCL (полнота CDCL легко доказывается, однако при этом не делается никаких ограничений на объем используемой памяти).

В последующих разделах мы детально исследуем эффект повторного порождения конфликтных дизъюнктов современными CDCL-решателями и предлагаем специальный механизм обнаружения и последующего использование таких дизъюнктов.

3. Феномен повторного порождения конфликтных дизъюнктов в CDCL и его анализ. В данном разделе мы исследуем феномен повторного порождения конфликтных дизъюнктов в CDCL-выводе. Мы исходим из тезиса, что порождаемые повторно дизъюнкты представляют важную информацию для поиска и их учет может существенно повысить эффективность решения рассматриваемой SAT-задачи. Мы продемонстрируем, что данный тезис справедлив в отношении некоторых аргументированно трудных SAT-задач. В этом направлении мы покажем, что учет повторно порождаемых конфликтных дизъюнктов способен многократно увеличивать эффективность решения задач обращения некоторых криптографических хеш-функций.

3.1. Повторное порождение конфликтных дизъюнктов. Итак, пусть C — произвольная КНФ над множеством булевых переменных $X = \{x_1, \ldots, x_n\}$, задача выполнимости которой рассматривается. Будем решать ее при помощи SAT-решателей, основанных на концепции CDCL.

Как уже было сказано выше, в основе CDCL лежит известный алгоритм DPLL [33], представляющий из собой обход бинарного дерева (которое интерпретирует пространство поиска) с выходом из тупиковых ветвей при помощи процедуры, названной бэктрекингом (*backtracking*). В рамках DPLL происходит угадывание значений переменных из X с последующим применением (когда это возможно) правила единичного дизъюнкта (*Unit Propagation rule*, *UP*, [39]) в роли механизма распространения ограничений (концепция, известная как *Boolean Constraint Propagation*). Грубо говоря, UP используется для порождения логических следований из исходной формулы C и угаданных значений переменных. В этом контексте дизъюнкты, образующие КНФ C, а также единичные дизъюнкты, соответствующие угаданным литералам, можно рассматривать как аксиомы, которые проверяются на совместную непротиворечивость. Правило UP в этом случае является единственным правилом вывода. Соответственно, далее будем использовать термины "UP-вывод", "DPLL-вывод" и "CDCL-вывод".

Итак, как уже было сказано, фундамент концепции CDCL составляет идея записи информации о тупиковых путях в дереве DPLL-поиска в форме конфликтных дизъюнктов. Конфликтный дизъюнкт хранит информацию о некоторых переменных и их значениях, одновременный выбор которых привел к конфликту. Легко показать, что любой конфликтный дизъюнкт D является логическим следствием (импликацией) исходной КНФ C, и, соответственно, C выполнима тогда и только тогда, когда выполнима $C' = C \wedge D$. Несложно доказать, что CDCL является полным алгоритмом при отсутствии ограничений на объем используемой памяти.

Современные CDCL SAT-решатели порождают в процессе работы колоссальные по мощности множества конфликтных дизъюнктов. С другой стороны, используемые в современных SAT-решателях струк-

туры данных подразумевают работу только с оперативной памятью, поскольку при использовании дисковой памяти резко падает производительность. Соответственно, при чрезмерном количестве конфликтных дизъюнктов возникает риск переполнения памяти. Во всех CDCL SAT-решателях, начиная с minisat, некоторые конфликтные дизъюнкты, признанные нерелевантными, периодически удаляются из оперативной памяти. Все используемые меры релевантности носят эвристический характер. Соответственно, полнота данной версии CDCL уже не очевидна, поскольку, строго говоря, алгоритм может многократно попадать в одни и те же тупиковые области поиска.

Далее мы используем следующую простую идею: если некоторый конфликтный дизъюнкт D был выведен дважды в процессе работы алгоритма CDCL, применяющего чистку конфликтной базы, то D представляет собой ценную информацию — решатель неоднократно пытается пойти по соответствующему пути. В этой ситуации мы предлагаем запрещать удаление такого рода "ценных дизъюнктов". Для этого мы предлагаем специальную технику сбора таких дизъюнктов и их учета в последующем поиске.

Итак, однопоточный алгоритм вывода, в котором учитываются повторно порождаемые дизъюнкты, имеет следующий общий вид. Изначально запускается обычный CDCL, дополненный процедурой чистки конфликтной базы. Через небольшие временны́е промежутки фрагменты конфликтной базы записываются в отдельный файл (`solverout`), не имеющий непосредственного отношения к процессу вывода. Произвольный конфликтный дизъюнкт в данном файле представлен в виде слова над некоторым алфавитом. Через некоторое время файл `solverout` анализируется на предмет присутствия в нем повторяющихся слов. Все выявленные повторяющиеся дизъюнкты конъюнктивно приписываются к исходной КНФ. Обозначим через C' полученную КНФ и подадим ее на вход SAT-решателю как исходную. В этом случае решатель не удалит приписанные дизъюнкты в ходе дальнейшей работы.

Описанная простая версия CDCL, использующего повторно порождаемые дизъюнкты, была изначально реализована на основе решателя MapleCOMSPS [46]. В рамках полученного решателя все генерируемые конфликтные дизъюнкты сохранялись на жесткий диск в виде файла `mapleout`, имеющего структуру хеш-таблицы. При чтении файла `mapleout` относительно каждого очередного дизъюнкта D проверялось наличие в `mapleout` копии D. Использование техники работы с хеш-таблицами позволило сделать данную процедуру весьма эффективной (соответствующие затраты для рассмотренных задач составляют доли процента от общего времени их решения). Все повторно порожденные дизъюнкты конъюнктивно приписывались к исходной КНФ, после чего рассматривалась проблема выполнимости полученной КНФ.

Построенный однопоточный решатель был протестирован на формулах Дирихле (они же Pigeon Hole Principle formulas, PHP_m^{m+1}, m — число клеток) [41]. Это один из самых популярных классов тестов для алгоритмов решения SAT. Хорошо известно (см. [42, 43]), что CDCL на формулах Дирихле имеет экспоненциальную сложность. Соответственно, кардинально повысить эффективность CDCL на данном классе формул в принципе невозможно. Однако формулы Дирихле можно использовать для оценивания эффективности различных эвристик, за счет применения которых время решения исходной задачи иногда удается снизить на несколько десятков процентов, а иногда (при небольшом числе клеток m) и в несколько раз. Результаты применения техники учета повторно порождаемых дизъюнктов к формулам Дирихле приведены в табл. 1.

Таблица 1

Учет повторно порождаемых конфликтных дизъюнктов применительно к формулам Дирихле. Время указано в секундах

Число клеток	Время решения обычным решателем MapleCOMSPS	Время решения решателем MapleCOMSPS, использующим повторно порождаемые дизъюнкты
8	2	2
9	9	9
10	58	18
11	596	25
12	> 100000	137
13	> 100000	3497

Комментарии к табл. 1. Можно заметить, что оригинальный решатель MapleCOMSPS (использовались значения его входных параметров "по умолчанию") демонстрирует резкий скачок сложности при переходе от 11 к 12 клеткам (доказательство невыполнимости PHP_{12}^{13} было прервано через 100 000 секунд).

Добавление повторно порожденных конфликтных дизъюнктов позволяет решить за разумное время даже задачу с 13 клетками (при этом тоже используется MapleCOMSPS с параметрами "по умолчанию").

3.2. Структура параллельного SAT-решателя, использующего механизм учета повторно порождаемых конфликтных дизъюнктов. Описанные выше идеи использования повторно порождаемых конфликтных дизъюнктов при решении трудных экземпляров SAT были реализованы в виде параллельного SAT-решателя RADAR (Repeatable clAuses Driven strAtegy solveR). Ниже мы описываем его базовые компоненты и их взаимодействие.

Решатель RADAR имеет модульную архитектуру и включает в себя перечисленные ниже компоненты.

1. Управляющий модуль — MPI-программа, которая обеспечивает сбор и анализ информации, получаемой в процессе решения SAT, а также взаимодействие между другими модулями.

2. Решатель minisat2.2 — основная программа, используемая для решения SAT.

3. Решатель minisat-mod — модифицированный решатель minisat2.2, сохраняющий всю конфликтную информацию, накопленную за некоторый фиксированный временной интервал.

4. Программа hash_clauses, выполняющая анализ конфликтной базы и выбирающая повторно порожденные конфликтные дизъюнкты.

5. Программа clause_decompose, в которой реализуется параллельная портфолио-стратегия на основе использования повторно порожденных дизъюнктов.

Процесс решения SAT включает в себя перечисленные далее этапы. На стадии инициализации исходная КНФ C подается на вход решателю RADAR. Далее полагаем, что RADAR работает с k ядрами.

I. На каждое из доступных ядер управляющий модуль передает C, на которой запускается решатель minisat-mod (обозначаемый далее через M) с ограничением времени работы в t секунд (значение t является входным параметром). Копии minisat-mod $M_1, \ldots, M_k$ запускаются с разными значениями параметра rnd-freq, заданными случайным образом — шаг, который стандартно используется в портфолио-стратегии (см., например, [44]) для обеспечения различных направлений поиска. Стадия I завершается через время t.

II. Пусть $D_1, \ldots, D_k$ — конфликтные базы, накопленные на стадии I решателями $M_1, \ldots, M_k$ соответственно. Данные базы подаются на вход k копиям программы hash_clauses, обозначенным через $H_1, \ldots, H_k$. Результатом применения H_i к базе D_i, $i \in \{1, \ldots, k\}$, является множество различных дизъюнктов, каждый из которых встретился в D_i более одного раза. Обозначим данное множество через R_i. Управляющий модуль собирает R_i, $i \in \{1, \ldots, k\}$, соединяет их в общую базу R, к которой еще раз применяется hash_clauses. Итог: множество различных дизъюнктов R^*, каждый из которых был повторно порожден какой-либо из k копий minisat-mod. Программа hash_clauses представляет произвольный набор конфликтных дизъюнктов в виде хеш-таблицы: повторно порожденные дизъюнкты получают при этом одинаковое хеш-значение. Программа hash_clauses основана на реализации алгоритмов работы с хеш-таблицами из стандартной библиотеки C++.

III. Управляющий модуль передает исходную КНФ C и множество дизъюнктов R^* программе clause_decompose. Эта программа организует решение SAT в соответствии с портфолио-стратегией при помощи специальной эвристики, использующей конфликтные дизъюнкты для задания различных направлений поиска. Детально работа этой эвристики описана в [45]. Результатом данного этапа является k различных КНФ $C_1, \ldots, C_k$.

IV. Управляющий модуль передает КНФ $C_1, \ldots, C_k$ рабочим процессам, в каждом из которых используется решатель minisat2.2. Поскольку произвольная КНФ C_i, $i \in \{1, \ldots, k\}$, имеет вид $C \wedge D'$, где D' — конъюнкция конфликтных дизъюнктов, выведенных из C в соответствии с алгоритмом CDCL, то из базовых свойств CDCL следует, что C выполнима тогда и только тогда, когда выполнимы все КНФ $C_1, \ldots, C_k$. Таким образом, если копия minisat2.2 с произвольным номером i, $i \in \{1, \ldots, k\}$, найдет набор, выполняющий C_i, то данный набор будет выполнять и исходную КНФ C. Если же будет доказана невыполнимость C_i, то это будет означать, что и C невыполнима.

4. Вычислительные эксперименты: решение задач обращения некоторых криптографических хеш-функций с применением решателя RADAR. В этом разделе мы приводим результаты

применения решателя RADAR, конструкция которого описана в предыдущем разделе, к задачам обращения некоторых криптографических хеш-функций.

Напомним, что хеш-функцией называется произвольная дискретная функция вида $h : \{0,1\}^* \to \{0,1\}^C$, где C — некоторая константа, не зависящая от длины входного слова. Из практических соображений рассматриваются только алгоритмически вычислимые за полиномиальное время хеш-функции. К криптографическим хеш-функциям предъявляются дополнительные требования, которые можно кратко суммировать следующим образом: задачи, так или иначе предполагающие вычисление прообраза произвольного хеш-значения, должны быть вычислительно трудными. Также требуется, чтобы вычислительно трудной была задача поиска произвольной коллизии криптографической хеш-функции. Напомним, что для $n > C$ коллизией длины n называется такая пара $x_1, x_2 \in \{0,1\}^n$, $x_1 \neq x_2$, что $h(x_1) = h(x_2)$.

В вычислительных экспериментах мы рассматривали перечисленные ниже задачи.

1. Для полнораундовой хеш-функции MD4 найти произвольное такое 512-битное сообщение, хеш которого содержит k нулевых старших бит. Задачи подобного типа в отношении хеш-функций более поздних поколений, таких как SHA-256 и Keccak, используются в некоторых протоколах криптовалют (Bitcoin, Ethereum) для реализации концепции “Proof-of-Work”. Конкретно, мы рассматривали задачи такого типа для MD4 со значениями $k = 20, 21, \ldots, 27$.

2. Задача обращения неполнораундовых версий хеш-функции SHA-1. Даже для 22 раундов из 80 эта задача оказывается трудной для всех известных SAT-решателей (см., например, [46, 47]).

3. Поиск второго блока двухблоковой коллизии для хеш-функции MD5 при разностных соотношениях из работы [48]. Эти задачи трудны для последовательных SAT-решателей, однако многопоточные решатели при работе в 36 потоков обычно тратят на один такой тест от одного до нескольких часов (см. [49]).

Таблица 2

Результаты применения RADAR к обращению MD4-хеша с k нулевыми старшими битами. Время указано в секундах

№ запуска	Число k старших нулевых бит MD4-хеша							
	20	21	22	23	24	25	26	27
1	550	1058	1032	893	1435	1432	1079	4509
2	675	629	1477	694	4447	1813	1972	1407
3	572	213	603	1627	501	22368	5191	17851
4	745	918	1728	2502	625	1232	3367	53476
5	477	513	1172	1594	370	1637	5021	473
6	617	460	1222	721	5883	1441	571	2341
7	451	468	763	2994	3059	2437	556	39033
8	477	970	1854	2479	361	3196	4931	29819
9	555	696	842	949	3497	599	5099	71768
10	890	1031	647	532	1377	1151	1397	2169
Ср. время	600	725	1161	1498	2155	3730	2918	22284

В вычислительных экспериментах принимали участие следующие многопоточные решатели: Plingeling, Treengeling [50], Painless-mapleCOMSPS [51], RADAR. Все эти решатели являются рандомизированными. Поэтому в каждом конкретном эксперименте рассматривается 10 копий одного теста. Далее время работы по этим 10 тестам усредняется и полученное значение является численным результатом эксперимента. Время решения одной копии задачи ограничивалось величиной 200 000 секунд. В каждом эксперименте в роли вычислительной платформы использовался один узел кластера “Академик В.М. Матросов”, установленного в ИДСТУ СО РАН [52]. Конфигурация узла: два 18-ядерных процессора Intel Xeon E5-2695 v4. Итого было задействовано 36 ядер. В табл. 2 приведены результаты работы решателя RADAR на задачах обращения хеш-функции MD4 с k нулевыми старшими битами хеша.

Таблица 3

Результаты применения различных решателей к обращению MD4-хешей с k нулевыми старшими битами. Для каждого значения k указано среднее время в секундах по 10 независимым запускам

Решатель	Количество подставленных нулевых бит хеша							
	20	21	22	23	24	25	26	27
RADAR	600	725	1161	1498	2155	3730	2918	22284
Painless	4724	7401	11262	27223	66078	135097	127941	75167
Plingeling	9054	2527	11846	12573	4444	20807	52793	>200000

Таблица 4

Результаты обращения случайных значений хеш-функции SHA-1-22 (для каждого теста указано среднее время в секундах по 10 независимым запускам)

Решатель	Тест № 1	Тест № 2	Тест № 3	Тест № 4	Тест № 5
RADAR	812	1424	1218	1718	1235
Painless	1346	1852	922	936	1205
Plingeling	30702	58890	18085	93547	3384
Treengeling	36605	118195	147424	44439	>200000

В табл. 3 приведено среднее время работы всех тестируемых решателей на задачах обращения MD4-хешей с k нулевыми старшими битами. Дополнительно отметим, что при $k = 24$ Painless решил 8 задач из 10, для $k = 26$ Painless решил 7 задач из 10, для $k = 27$ этим решателем были решены только 3 задачи из 10. Решатель Plingeling для $k = 27$ не решил ни одной задачи. В каждом эксперименте среднее время вычислялось только по тем задачам, на решение которых потребовалось не более 200 000 секунд.

Решатель Treengeling не справился ни с одной из копий рассматриваемых задач за отведенное время.

Таблица 4 содержит результаты применения SAT-решателей к задаче обращения 22-раундовой версии хеш-функции SHA-1 (SHA-1-22). Были сгенерированы 5 случайных хеш-значений данной функции, после чего для каждого хеш-значения строилось 10 копий SAT-задачи, кодирующей его обращение. На построенных 10 копиях запускались тестируемые SAT-решатели. Время решения усреднялось, причем учитывались только те копии, на решение которых было затрачено не более 200 000 секунд. Отметим, что решатель Treengeling не дорешал ни одной копии теста № 5 за отведенное время.

Таблица 5

Поиск вторых блоков для двухблоковой коллизии MD5 с учетом дифференциальных соотношений из [48] (среднее время в секундах по 10 независимым запускам)

Решатель	Тест № 1	Тест № 2
RADAR	734	821
Painless	18318	4742
Plingeling	32763	11620
Treengeling	26430	25909

Таблица 5 содержит результаты поиска вторых блоков для двухблоковой коллизии хеш-функции MD5 с учетом дифференциальных соотношений из статьи [48]. Так же, как и в случае с рассмотренной выше задачей обращения функции MD4, здесь каждый решатель запускался на 10 копиях рассматриваемой задачи. Получаемое время усреднялось. Именно это значение и приведено в таблице.

Заключение. В настоящей статье описана новая техника, дополняющая стандартные реализации алгоритма CDCL в современных SAT-решателях. Данная техника основана на отслеживании конфликтных дизъюнктов, которые удаляются в процессе чистки конфликтных баз, но затем порождаются снова. Данный факт мы рассматриваем как свидетельство важности таких конфликтных ограничений для CDCL-вывода. Представлены механизмы отслеживания таких дизъюнктов и последующего запрета на

их удаление. На основании предложенной техники был построен новый многопоточный SAT-решатель RADAR (Repeatable clAuses Driven strAtegy solveR). На SAT-задачах, кодирующих проблемы обращения криптографических хеш-функций, решатель RADAR продемонстрировал существенное превосходство над многопоточными решателями, занимавшими в последние годы высокие места на специализированных соревнованиях.

Заикин Олег Сергеевич поддержан Советом по грантам Президента Российской Федерации (грант № МК-4155.2018.9).

СПИСОК ЛИТЕРАТУРЫ

1. *Cook S.A.* The complexity of theorem-proving procedures // Proc. 3rd Annual ACM Symposium on Theory of Computing. New York: ACM Press, 1971. 151–159.
2. *Левин Л.А.* Универсальные задачи перебора // Пробл. передачи информ. 1973. **9**, N 3, 115–116.
3. *Marques-Silva J.P., Sakallah K.A.* GRASP: a search algorithm for propositional satisfiability // IEEE Transactions on Computers. 1999. **48**, N 5. 506–521.
4. *Marques-Silva J., Lynce I., Malik S.* Conflict-driven clause learning SAT solvers // Handbook of Satisfiability. Amsterdam: IOS Press, 2009. 131–153.
5. *Selman B., Kautz H.A., Cohen B.* Noise strategies for improving local search // Proc. 12th National Conf. on Artificial Intelligence (AAAI'94). Cambridge: MIT Press, 1994. 337–343.
6. *Schöning U.* A probabilistic algorithm for k-SAT and constraint satisfaction problems // Proc. 40th Annual Symp. on Foundations of Computer Science. Washington, DC: IEEE Press, 1999. 410–414.
7. *Heule M.J.H., van Maaren H.* Look-ahead based SAT solvers // Handbook of Satisfiability. Amsterdam: IOS Press, 2009. 155–184.
8. *Heule M.J.H., Kullmann O., Wieringa S., Biere A.* Cube and conquer: guiding CDCL SAT solvers by lookaheads // Lecture Notes in Computer Science. Vol. 7261. Heidelberg: Springer, 2011. 50–65.
9. *Heule M.J.H., Kullmann O., Marek V.W.* Solving and verifying the Boolean pythagorean triples problem via cube-and-conquer // Lecture Notes in Computer Science. Vol. 9710. Heidelberg: Springer, 2016. 228–245.
10. *Mironov I., Zhang L.* Applications of SAT solvers to cryptanalysis of hash functions // Lecture Notes in Computer Science. Vol. 4121. Heidelberg: Springer, 2006. 102–115.
11. *Semenov A., Zaikin O., Bespalov D., Posypkin M.* Parallel logical cryptanalysis of the generator A5/1 in BNB-Grid system // Lecture Notes in Computer Science. Vol. 6873. Heidelberg: Springer, 2011. 473–483.
12. *Мендельсон Э.* Введение в математическую логику. М.: Наука, 1971.
13. *Яблонский С.В.* Введение в дискретную математику. М.: Наука, 1986.
14. *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982.
15. *Biere A., Heule M., van Maaren H., Walsh T.* Handbook of Satisfiability. Amsterdam: IOS Press, 2009.
16. *Biere A.* Bounded model checking // Handbook of Satisfiability. Amsterdam: IOS Press, 2009. 457–481.
17. *Kröning D.* Software verification // Handbook of Satisfiability. Amsterdam: IOS Press, 2009. 505–532.
18. *Lynce I., Marques-Silva J.* SAT in bioinformatics: making the case of haplotype inference // Lecture Notes in Computer Science. Vol. 4121. Heidelberg: Springer, 2006. 136–141.
19. *Kochemazov S., Semenov A.* Using synchronous Boolean networks to model several phenomena of collective behavior // PLoS ONE. 2014. **9**. doi 10.1371/journal.pone.0115156.
20. *Zhang H.* Combinatorial designs by SAT solvers // Handbook of Satisfiability. Amsterdam: IOS Press, 2009. 533–568.
21. *Bard G.* Algebraic cryptanalysis. Berlin: Springer, 2009.
22. *Gendrullis T., Novotny M., Rupp A.* A real-world attack breaking A5/1 within hours // Lecture Notes in Computer Science. Vol. 5154. Heidelberg: Springer, 2008. 266–282.
23. *Посыпкин М.А., Заикин О.С., Беспалов Д.В., Семенов А.А.* Решение задач криптоанализа поточных шифров в распределенных вычислительных средах // Труды ИСА РАН. 2009. № 46. 119–137.
24. *Nohl K.* Attacking phone privacy // Black hat lecture notes. Las-Vegas, 2010. 1–6.
25. *Bulavintsev V., Semenov A., Zaikin O., Kochemazov S.* A bitslice implementation of Anderson's attack on A5/1 // Open Engineering. 2018. **8**. 7–16.
26. *King J.C.* Symbolic execution and program testing // Commun. ACM. 1976. **19**, N 7. 385–394.
27. *Erkök L., Matthews J.* Pragmatic equivalence and safety checking in Cryptol // Proceedings of the 3rd Workshop on Programming Languages Meets Program Verification. New York: ACM Press, 2008. 73–82.
28. *Erkök L., Carlsson M., Wick A.* Hardware/software co-verification of cryptographic algorithms using Cryptol // Proceedings of 9th International Conference on Formal Methods in Computer-Aided Design. New York: IEEE Press, 2009. 188–191.
29. *Janicic P.* URSA: a system for uniform reduction to SAT // Logical Methods in Computer Science. 2012. **8**, N 3. 1–39.
30. *Отпущенников И.В., Семенов А.А.* Технология трансляции комбинаторных проблем в булевы уравнения // Прикладная дискретная математика. 2011. № 1. 96–115.

31. *Otpuschennikov I., Semenov A., Gribanova I., Zaikin O., Kochemazov S.* Encoding cryptographic functions to SAT using TRANSALG system // Frontiers in Artificial Intelligence and Applications. Vol. 285. Amsterdam: IOS Press, 2016. 1594–1595.
32. *Davis M., Putnam H.* A computing procedure for quantification theory // Journal of the ACM. 1960. **7**, N 3. 201–215.
33. *Davis M., Logemann G., Loveland D.* A machine program for theorem-proving // Communication of the ACM. 1962. **5**, N 7. 394–397.
34. *Marques-Silva J.P., Sakallah K.A.* GRASP — a new search algorithm for satisfiability // Proceedings of Int. Con. on Computer-Aided Design. Piscataway: IEEE Press, 1996. 220–227.
35. *Moskewicz M.W., Madigan C.F., Zhao Y., Zhang L., Malik S.* Chaff: engineering an efficient SAT solver // Proc. 38th Annual Design Automation Conference (DAC'01). New York: IEEE Press, 2001. 530–535.
36. *Zhang L., Madigan C.F., Moskewicz M.H., Malik S.* Efficient conflict driven learning in a Boolean satisfiability solver // Proc. 2001 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'01). Piscataway: IEEE Press, 2001. 279–285.
37. *Eén N., Sörensson N.* An extensible SAT-solver // Lecture Notes in Computer Science. Vol. 2919. Heidelberg: Springer, 2004. 502–518.
38. *Audemard G., Simon L.* Predicting learnt clauses quality in modern SAT solvers // Proceedings of 21st International Joint Conference on Artificial Intelligence. San Francisco: Morgan Kaufmann, 2009. 399–404.
39. *Dowling W.F., Gallier J.H.* Linear-time algorithms for testing the satisfiability of propositional Horn formulae // The Journal of Logic Programming. 1984. **1**, N 3. 267–284.
40. *Liang J.H., Ganesh V., Poupart P., Czarnecki K.* Learning rate based branching heuristic for SAT solvers // Lecture Notes in Computer Science. Vol. 9710. Heidelberg: Springer, 2016. 123–140.
41. *Cook S.A., Reckhow R.A.* The relative efficiency of propositional proof systems // Journal of Symbolic Logic. 1979. **44**, N 1. 36–50.
42. *Haken A.* The intractability or resolution // Theoretical Computer Science. 1985. **39**. 297–308.
43. *Beame P., Kautz H., Sabharwal A.* Understanding the power of clause learning // Proceedings of 18th International Joint Conference on Artificial Intelligence. San Francisco: Morgan Kaufmann, 2003. 1194–1201.
44. *Hyvärinen A.E.J.* Grid based propositional satisfiability solving. PhD Thesis. Aalto: Aalto Univ., 2011.
45. *Zaikin O.* A parallel SAT solving algorithm based on improved handling of conflict clauses // Procedia in Computer Science. 2017. **119**. 103–111.
46. *Nejati S., Newsham Z., Scott J., Liang J.H., Gebotys C., Poupart P., Ganesh V.* A propagation rate based splitting heuristic for divide-and-conquer solvers // Lecture Notes in Computer Science. Vol. 10491. Heidelberg: Springer, 2017. 251–260.
47. *Nejati S., Liang J.H., Gebotys C., Czarnecki K., Ganesh V.* Adaptive restart and CEGAR-based solver for inverting cryptographic hash functions // Lecture Notes in Computer Science. Vol. 10712. Heidelberg: Springer, 2017. 120–131.
48. *Wang X., Yu H.* How to break MD5 and other hash functions // Lecture Notes in Computer Science. Vol. 3494. Heidelberg: Springer, 2005. 19–35.
49. *Богачкова И.А., Заикин О.С., Кочемазов С.Е., Отпущенников И.В., Семенов А.А., Хамисов О.О.* Задачи поиска коллизий криптографических хеш-функций семейства MD как варианты задачи о булевой выполнимости // Вычислительные методы и программирование. 2015. **16**. 61–77.
50. *Biere A.* CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT entering the SAT competition 2017 // In Tomás Balyo, Marijn J.H. Heule, and Matti Järvisalo, Editors, SAT Competition 2017. 2017. **B-2017-1**. 14–15.
51. *Le Frioux L., Baarir S., Sopena J., Kordon F.* PaInleSS: a framework for parallel SAT solving // Lecture Notes in Computer Science. Vol. 10491. Heidelberg: Springer, 2017. 233–250.
52. Иркутский суперкомпьютерный центр СО РАН [Электронный ресурс]: сайт. Иркутск: ИДСТУ СО РАН. URL: http://hpc.icc.ru (дата обращения: 15.02.2019).

Поступила в редакцию
14.01.2019

Duplicates of Conflict Clauses in CDCL Derivation and Their Usage to Invert Some Cryptographic Functions

V. S. Kondratiev [1], **A. A. Semenov** [2], **and O. S. Zaikin** [3]

[1] *Irkutsk National Research Technical University; ulitsa Lermontova 83, Irkutsk, 664074, Russia; Master's Student, e-mail: vikseko@gmail.com*

[2] *Matrosov Institute for System Dynamics and Control Theory, Siberian Branch of Russian Academy of Sciences; ulitsa Lermontova 134, Irkutsk, 664033, Russia; Ph.D., Associate Professor, Head of Laboratory, e-mail: biclop.rambler@yandex.ru*

[3] *Matrosov Institute for System Dynamics and Control Theory, Siberian Branch of Russian Academy of Sciences; ulitsa Lermontova 134, Irkutsk, 664033, Russia; Ph.D., Senior Scientist, e-mail: zaikin.icc@gmail.com*

Received January 14, 2019

Abstract: A phenomenon of conflict clauses generated repeatedly by SAT solvers is studied. Such clauses may appear during solving hard Boolean satisfiability problems (SAT). This phenomenon is caused by the fact that the modern SAT solvers are based on the CDCL algorithm that generates conflict clauses. A database of such clauses is periodically and partially cleaned. A new approach for practical SAT solving is proposed. According to this approach, the repeatedly generated conflict clauses are tracked, whereas their further generation is prohibited. Based on this approach, a multithreaded SAT solver was developed. This solver was compared with the best multithreaded SAT solvers awarded during the last SAT competitions. According to the experimental results, the developed solver greatly outperforms its competitors on several SAT instances encoding the inversion of some cryptographic hash functions.

Keywords: Boolean satisfiability problem (SAT), CDCL algorithm, conflict clause, cryptographic hash functions.

References

1. S. A. Cook, "The Complexity of Theorem-Proving Procedures," in *Proc. 3rd Annu. ACM Symp. on Theory of Computing, Shaker Heights, USA, May 3–5, 1971* (ACM Press, New York, 1971), pp. 151–158.

2. L. A. Levin, "Universal Sequential Search Problems," Probl. Peredachi Inf. **9** (3), 115–116 (1973) [Problems Inform. Transmission **9** (3), 265–266 (1973)].

3. J. P. Marques-Silva and K. A. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability," IEEE Trans. Comput. **48** (5), 506–521 (1999).

4. J. Marques-Silva, I. Lynce, and S. Malik, "Conflict-Driven Clause Learning SAT Solvers," in *Handbook of Satisfiability* (IOS Press, Amsterdam, 2009), pp. 131–153.

5. B. Selman, H. A. Kautz, and B. Cohen, "Noise Strategies for Improving Local Search," in *Proc. 12th National Conf. on Artificial Intelligence (AAAI'94), Seattle, USA, July 31–August 4, 1994* (MIT Press, Cambridge, 1994), pp. 337–343.

6. U. Schöning, "A Probabilistic Algorithm for k-SAT and Constraint Satisfaction Problems," in *Proc. 40th Annual Symp. on Foundations of Computer Science, New York, USA, October 17–19, 1999* (IEEE Press, Washington, DC, 1999), pp. 410–414.

7. M. J. H. Heule and H. van Maaren, "Look-Ahead Based SAT Solvers," in *Handbook of Satisfiability* (IOS Press, Amsterdam, 2009), pp. 155–184.

8. M. J. H. Heule, O. Kullmann, S. Wieringa, and A. Biere, "Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2011), Vol. 7261, pp. 50–65.

9. M. J. H. Heule, O. Kullmann, and V. W. Marek, "Solving and Verifying the Boolean Pythagorean Triples Problem via Cube-and-Conquer," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2016), Vol. 9710, pp. 228–245.

10. I. Mironov and L. Zhang, "Applications of SAT Solvers to Cryptanalysis of Hash Functions," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2006), Vol. 4121, pp. 102–115.

11. A. Semenov, O. Zaikin, D. Bespalov, and M. Posypkin, "Parallel Logical Cryptanalysis of the Generator A5/1 in BNB-Grid System," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2011), Vol. 6873, pp. 473–483.

12. E. Mendelson, *Introduction to Mathematical Logic* (Van Nostrand, Princeton, 1964; Nauka, Moscow, 1971).

13. S. V. Yablonskii, *Introduction to Discrete Mathematics* (Nauka, Moscow, 1986) [in Russian].

14. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, New York, 1979; Mir, Moscow, 1982).

15. A. Biere, M. Heule, H. van Maaren, and T. Walsh, *Handbook of Satisfiability* (IOS Press, Amsterdam, 2009).

16. A. Biere, "Bounded Model Checking," in *Handbook of Satisfiability* (IOS Press, Amsterdam, 2009), pp. 457–481.

17. D. Kröning, "Software Verification," in *Handbook of Satisfiability* (IOS Press, Amsterdam, 2009), pp. 505–532.

18. I. Lynce and J. Marques-Silva, "SAT in Bioinformatics: Making the Case with Haplotype Inference," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2006), Vol. 4121, pp. 136–141.

19. S. Kochemazov and A. Semenov, "Using Synchronous Boolean Networks to Model Several Phenomena of Collective Behavior," PLoS ONE **9** (2014). doi 10.1371/journal.pone.0115156

20. H. Zhang, "Combinatorial Designs by SAT Solvers," in *Handbook of Satisfiability* (IOS Press, Amsterdam, 2009), pp. 533–568.

21. G. V. Bard, *Algebraic Cryptanalysis* (Springer, Berlin, 2009).

22. T. Gendrullis, M. Novotný, and A. Rupp, "A Real-World Attack Breaking A5/1 within Hours," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2008), Vol. 5154, pp. 266–282.

23. M. A. Posypkin, O. S. Zaikin, D. V. Bespalov, and A. A. Semenov, "Solving the Cryptanalysis Problems for Stream Ciphers in Distributed Computing Environments," Tr. Inst. Sistem. Analiza, No. 46, 119–137 (2009).

24. K. Nohl, "Attacking Phone Privacy,"
https://media.blackhat.com/bh-us-10/whitepapers/Nohl/BlackHat-USA-2010-Nohl-Attacking.Phone.Privacy-wp.pdf. Cited February 15, 2019.

25. V. Bulavintsev, A. Semenov, O. Zaikin, and S. Kochemazov, "A Bitslice Implementation of Anderson's Attack on A5/1," Open Eng. **8** (1), 7–16 (2018).

26. J. C. King, "Symbolic Execution and Program Testing," Commun. ACM **19** (7), 385–394 (1976).

27. L. Erkök and J. Matthews, "Pragmatic Equivalence and Safety Checking in Cryptol," in *Proc. 3rd Workshop on Programming Languages Meets Program Verification, Savannah, USA, January 20, 2009* (ACM Press, New York, 2009), pp. 73–82.

28. L. Erkök, M. Carlsson, and A. Wick, "Hardware/Software Co-Verification of Cryptographic Algorithms Using Cryptol," in *Proc. 9th Int. Conf. on Formal Methods in Computer-Aided Design, Austin, USA, November 15–18, 2009* (IEEE Press, New York, 2009), pp. 188–191.

29. P. Janicic, "URSA: A System for Uniform Reduction to SAT," Log. Meth. Comput. Sci. **8** (3), 1–39 (2012).

30. I. V. Otpushchennikov and A. A. Semenov, "Technology for Translating Combinatorial Problems into Boolean Equations," Prikl. Diskr. Mat., No. 1, 96–115 (2011).

31. I. Otpuschennikov, A. Semenov, I. Gribanova, et al., "Encoding Cryptographic Functions to SAT Using TRANSALG System," in *Frontiers in Artificial Intelligence and Applications* (IOS Press, Amsterdam, 2016), Vol. 285, 1594–1595.

32. M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory," J. ACM **7** (3), 201–215 (1960).

33. M. Davis, G. Logemann, and D. Loveland, "A Machine Program for Theorem-Proving," Commun. ACM **5** (7), 394–397 (1962).

34. J. P. Marques-Silva and K. A. Sakallah, "GRASP — a New Search Algorithm for Satisfiability," in *Proc. 1996 IEEE/ACM Int. Con. on Computer-Aided Design, San Jose, USA, November 10–14, 1996* (IEEE Press, Piscataway, 1996), pp. 220–227.

35. M. W. Moskewicz, C. F. Madigan, Y. Zhao, et al., "Chaff: Engineering an Efficient SAT Solver," in *Proc. 38th Annual Design Automation Conference, Las Vegas, USA, June 22–22, 2001* (IEEE Press, New York, 2001), pp. 530–535.

36. L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik, "Efficient Conflict Driven Learning in a Boolean Satisfiability Solver," in *Proc. 2001 IEEE/ACM Int. Conf. on Computer-Aided Design, San Jose, USA, November 4–8, 2001* (IEEE Press, Piscataway, 2001), pp. 279–285.

37. N. Eén and N. Sörensson, "An Extensible SAT-Solver," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2004), Vol. 2919, pp. 502–518.

38. G. Audemard and L. Simon, "Predicting Learnt Clauses Quality in Modern SAT Solvers," in *Proc. 21st Int. Joint Conf. on Artificial Intelligence. Pasadena, USA, July 11–17, 2009* (Morgan Kaufmann, San Francisco, 2009), pp. 399–404.

39. W. F. Dowling and J. H. Gallier, "Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae," J. Logic Programm. **1** (3), 267–284 (1984).

40. J. H. Liang, V. Ganesh, P. Poupart, and K. Czarnecki, "Learning Rate Based Branching Heuristic for SAT Solvers," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2016), Vol. 9710, pp. 123–140.

41. S. A. Cook and R. A. Reckhow, "The Relative Efficiency of Propositional Proof Systems," J. Symbolic Logic **44** (1), 36–50 (1979).

42. A. Haken, "The Intractability of Resolution," Theor. Comp. Sci. **39**, 297–308 (1985).

43. P. Beame, H. Kautz, and A. Sabharwal, "Understanding the Power of Clause Learning," in *Proc. 18th Int.*

Joint Conf. on Artificial Intelligence. Acapulco, Mexico, August 9–15, 2003 (Morgan Kaufmann, San Francisco, 2003), pp. 1194–1201.

44. A. E. J. Hyvärinen, *Grid Based Propositional Satisfiability Solving*, Ph.D. Thesis (Aalto Univ., Aalto, 2011).

45. O. Zaikin, "A Parallel SAT Solving Algorithm Based on Improved Handling of Conflict Clauses," Procedia Comput. Sci. **119**, 103–111 (2017).

46. S. Nejati, Z. Newsham, J. Scott, et al., "A Propagation Rate Based Splitting Heuristic for Divide-and-Conquer Solvers," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2017), Vol. 10491, pp. 251–260.

47. S. Nejati, J. H. Liang, C. Gebotys, et al., "Adaptive Restart and CEGAR-Based Solver for Inverting Cryptographic Hash Functions," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2017), Vol. 10712, pp. 120–131.

48. X. Wang and H. Yu, "How to Break MD5 and Other Hash Functions," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2005), Vol. 3494, pp. 19–35.

49. I. A. Bogachkova, O. S. Zaikin, S. E. Kochemazov, et al., "Problems of Search for Collisions of Cryptographic Hash Functions of the MD Family as Variants of Boolean Satisfiability Problem," Vychisl. Metody Programm. **16**, 61–77 (2015).

50. A. Biere, "CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT Entering the SAT Competition 2017," http://fmv.jku.at/papers/Biere-SAT-Competition-2017-solvers.pdf. Cited February 15, 2019.

51. L. Le Frioux, S. Baarir, J. Sopena, and F. Kordon, "PaInleSS: A Framework for Parallel SAT Solving," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2017), Vol. 10491, pp. 233–250.

52. Irkutsk Supercomputing Center, Siberian Branch of the Russian Academy of Sciences. http://hpc.icc.ru. Cited February 15, 2019.