



Применение SAT-решателей для верификации комбинационных схем

Выполнили студенты группы 3312:

Барченков Платон Алексеевич

Лебедев Игнат Артемович

Шарапов Иван Денисович

Цель исследования



СПбГЭТУ «ЛЭТИ»
ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ

Проблема проверки выполнимости булевых формул (SAT) представляет собой одну из фундаментальных NP-полных (полный перебор). Несмотря на отсутствие известных полиномиальных алгоритмов решения в общем случае, современные SAT-решатели, основанные на алгоритме CDCL (Conflict-Driven Clause Learning), демонстрируют эффективное решение формул с сотнями и тысячами переменных.

Цель исследования



СПБГЭТУ «ЛЭТИ»
ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ

Целью данного исследования является систематическое изучение эволюции алгоритмических подходов от классического DPLL (Davis-Putnam-Logemann-Loveland) к современному CDCL, анализ архитектуры промышленного SMT-решателя Z3 от Microsoft Research, и эмпирическое сравнение производительности полного перебора с CDCL-алгоритмом на примере задачи проверки эквивалентности булевых схем.

Суть проблемы



СПБГЭТУ «ЛЭТИ»
ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ

Проблема SAT формулируется следующим образом: дана булева формула F , составленная из пропозициональных переменных, логических операторов (конъюнкция, дизъюнкция, отрицание) и скобок. Требуется определить, существует ли такое назначение значений всем переменным из множества $\{0,1\}$, при котором формула F принимает значение истинно (1). Если такое назначение существует, формула называется выполнимой (SAT); в противном случае — невыполнимой (UNSAT).

Проблема SAT является первой NP-полной задачей, доказанной Стивеном Куком в 1971 году (теорема Кука). Хотя в худшем случае проблема требует экспоненциального времени вычисления $O(2^n)$, где n — число переменных, эмпирические исследования показывают, что для большинства практических экземпляров современные SAT-решатели находят решение значительно быстрее.

Булева переменная принимает значения из множества {0, 1}. Литерал — это либо переменная x , либо её отрицание $\neg x$. Клауза представляет собой дизъюнкцию литералов: $(l_1 \vee l_2 \vee \dots \vee l_k)$. Конъюнктивная Нормальная Форма (КНФ) — это конъюнкция клауз: $(C_1 \wedge C_2 \wedge \dots \wedge C_m)$.

КНФ является стандартным представлением для входных данных SAT-решателей, поскольку она позволяет эффективно применять методы, основанные на *unit propagation* и *resolution*. Любая булева формула может быть преобразована в эквивалентную КНФ-формулу посредством распределительного закона, хотя это преобразование может экспоненциально увеличить размер формулы.

Основные логические операции определяются таблицами истинности: конъюнкция (AND), дизъюнкция (OR) и отрицание (NOT).

Формальное определение выполнимости: формула F называется выполнимой, если существует интерпретация $I: F \rightarrow \{0, 1\}$, такая что $F(I) = 1$.

Метод полного перебора (brute force) основан на систематическом тестировании всех возможных назначений значений переменным формулы. Практическая применимость этого метода ограничивается следующим образом: при $n = 10$ переменных требуется проверить 1024 комбинации; при $n = 20$ — 1048576 комбинаций; при $n = 30$ — 1073741824 комбинаций.

Экспоненциальный рост функции 2^n делает полный перебор практически неприменимым для $n > 25$ на современных вычислительных системах. А такой запрос есть.

Одним из значительных практических применений SAT-решателей является верификация эквивалентности цифровых схем. Пусть даны две булевые функции F_1 и F_2 , реализованные различными схемотехническими реализациями. Вопрос верификации формулируется следующим образом: выполняется ли условие $\forall x \in \{0,1\}^n: F_1(x) = F_2(x)$? Классический подход к решению этой задачи посредством SAT-решателя использует конструкцию, называемую miter: $D(x) = (F_1(x) \wedge \neg F_2(x)) \vee (\neg F_1(x) \wedge F_2(x))$.

Формула D истинна тогда и только тогда, когда F_1 и F_2 выдают различные значения при некотором входном векторе x . SAT-решатель решает задачу о выполнимости D . Если D выполнима (SAT), то найдено свидетельство неэквивалентности (контрпример). Если D невыполнима (UNSAT), функции эквивалентны.

Алгоритм DPLL, предложенный Davis, Putnam, Logemann и Loveland в 1960-х годах, представляет собой первый систематический метод решения SAT-задач, использующий принцип поиска с возвратом (backtracking). Основные компоненты алгоритма включают:

- (1) Unit Propagation — если клауза содержит единственный литерал, этот литерал обязательно должен быть установлен в истину; в результате клауза удаляется, остальные клаузы упрощаются путём удаления противоположного литерала;
- (2) Pure Literal Elimination — если переменная встречается в формуле только в одном полярном виде, ей присваивается значение, удовлетворяющее все содержащие её клаузы;
- (3) Decision — выбор неустановленной переменной и рекурсивная попытка обоих её значений;
- (4) Backtracking — при обнаружении противоречия (пустой клаузы) возврат к последней точке выбора и попытка альтернативного значения.

DPLL значительно снижает эффективный размер пространства поиска по сравнению с полным перебором, однако не использует информацию из предыдущих конфликтов для оптимизации будущего поиска. Этого недостатка лишён следующий алгоритм.

Алгоритм CDCL (Conflict-Driven Clause Learning) представляет собой современное расширение DPLL, определившее развитие SAT-решателей за последние два десятилетия. Ключевые инновации CDCL включают:

- (1) Clause Learning — при обнаружении конфликта выполняется анализ причин конфликта посредством backward traversal по цепочке причинно-следственных связей (implication graph); на основе этого анализа синтезируется новая клауда, которая добавляется в формулу и предотвращает повторение идентичного конфликта;
- (2) Non-chronological Backtracking — возврат выполняется не на один уровень, а на уровень, определяемый анализом конфликта (asserting level), что позволяет более эффективно сужать пространство поиска;

- (3) VSIDS Heuristic (Variable State Independent Decaying Sum) — эвристика выбора переменной, которая приоритизирует переменные, часто появляющиеся в недавних конфликтах, с экспоненциальным затуханием старых данных;
- (4) Restarts — периодическая перезагрузка процесса поиска с сохранением выученных клауз, что помогает преодолевать "застрение" в неудачных ветвях;
- (5) In-processing — оптимизация формулы во время решения путём упрощения и удаления избыточных клауз.

CDCL-решатели демонстрируют способность решать практические экземпляры SAT с тысячами переменных и миллионами клауз.

Z3 – SMT-решатель, разработанный в Microsoft Research. В отличие от классических SAT-решателей, Z3 поддерживает не только булеву логику, но и теории целых/вещественных чисел, массивов и функций.

Архитектура основана на CDCL-алгоритме, дополненном слоями: парсинг и построение AST, применение теоретических разрешителей, ядро SAT-решателя с оптимизациями, управление конфликтами и обучение клаузам.

Производительность обеспечивается благодаря VSIDS-эвристике, эффективному управлению памятью, стратегиям рестарта и параллельному поиску. Решатель распространяется открытым исходным кодом на GitHub с поддержкой привязок для Python, Java, C++, Go и Rust.

Реализованная система верификации состоит из следующих основных компонентов:

- (1) Tokenizer — лексический анализатор, преобразующий входную строку булева выражения в последовательность токенов (переменные, операторы, скобки), с обработкой ошибок синтаксиса;
- (2) Parser — синтаксический анализатор, конструирующий Абстрактное Синтаксическое Дерево (AST) с соблюдением приоритета операций ($! > \wedge > \vee > \rightarrow$);

- (3) AST Representation — иерархическое представление формулы с использованием узлов типов Var (переменная), NotNode (отрицание), BinNode (бинарная операция);
- (4) Brute Force Verifier — модуль полного перебора, перечисляющий все 2^n интерпретаций и сравнивающий результаты вычисления обеих формул;
- (5) Z3 Integration Layer — преобразование AST в Z3-выражения и использование Z3 Solver для проверки выполнимости miter;
- (6) Benchmarking Framework — система сбора и анализа временных метрик, включающая измерение производительности обоих подходов на формулах возрастающей сложности с сохранением результатов в CSV-формате.

Алгоритм полного перебора для проверки эквивалентности двух булевых формул F_1 и F_2 основан на следующей процедуре:

Пусть $\text{Var} = \{x_0, x_1, \dots, x_{n-1}\}$ — объединение множеств переменных обеих формул. Для каждой интерпретации $\sigma: \text{Var} \rightarrow \{0, 1\}$ (всего 2^n интерпретаций) вычисляются значения $F_1(\sigma)$ и $F_2(\sigma)$. Если существует интерпретация σ , для которой $F_1(\sigma) \neq F_2(\sigma)$, формулы не эквивалентны — процедура возвращает `False`. Если все интерпретации удовлетворяют $F_1(\sigma) = F_2(\sigma)$, формулы эквивалентны — возвращает `True`.

Полный перебор



СПБГТУ «ЛЭТИ»
ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ

Временная сложность алгоритма: $T = O(2^n \cdot m)$, где m — время вычисления формулы. Пространственная сложность: $O(n)$.

Практическая верхняя граница применимости полного перебора составляет $n \leq 25$ переменных на современных ПК, при этом время вычисления может достигать нескольких минут.

Преимущество метода — простота реализации и гарантированная корректность. Недостаток — невозможность масштабирования на более крупные задачи.

Подход верификации с использованием Z3 основан на редукции к задаче выполнимости. Для проверки эквивалентности $F_1 \equiv F_2$ строится формула miter: $M(x) = (F_1(x) \wedge \neg F_2(x)) \vee (\neg F_1(x) \wedge F_2(x))$.

Формула M выражает XOR функций: M истинна тогда и только тогда, когда F_1 и F_2 выдают различные значения. Процедура:

- (1) Парсинг обеих формул в AST;
- (2) Трансляция AST в Z3 выражения с использованием рекурсивного обхода дерева и преобразования узлов в соответствующие Z3 примитивы (And, Or, Not);

- (3) Конструирование miter как булево выражение в Z3;
- (4) Добавление miter в Z3 Solver: `solver.add(M);`
- (5) Вызов `solver.check()`: если результат SAT, найдено свидетельство различия — формулы не эквивалентны; если UNSAT — формулы эквивалентны. Z3 при наличии свидетельства (SAT) может извлечь значение-модель, которое служит контрпримером.

Временная сложность в среднем случае практически не зависит от n , что демонстрирует эффективность CDCL-стратегий.

Для анализа производительности на малых формулах ($n \leq 10$ переменных) были проведены измерения на наборе тестовых примеров с различными типами логических связей. Результаты показали:

Пример 1 ($x_0 \vee x_1$ vs $x_1 \vee x_0$): brute force = 0.016 мс, Z3 = 0.576 мс;

Пример 2 ($x_0 \vee x_1$ vs $x_0 \vee x_1$): brute force = 0.011 мс, Z3 = 0.650 мс;

Пример 8 ($x_0 \rightarrow x_1$ vs $\neg x_0 \vee x_1$): brute force = 0.012 мс, Z3 = 0.040 мс.

Анализ показывает, что на малых входах брутфорс часто быстрее Z3 из-за накладных расходов инициализации Z3 (парсинг, инициализация Solver, построение внутренних структур данных). Однако наблюдается высокая вариативность Z3, что объясняется сложностью конкретной формулы и эффективностью применяемых эвристик.

На примере № 8 видно, что при простой логической эквивалентности Z3 показывает превосходство. Эти результаты иллюстрируют существование точки пересечения, после которой CDCL-подход становится доминирующим.

Основной набор экспериментов включал бенчмарки на формулах с числом переменных от 2 до 25. Результаты (средние значения из 30 проходов):

n=10: brute force = 0.117 мс, Z3 = 0.345 мс;

n=15: brute force = 1.164 мс, Z3 = 0.331 мс;

n=17: brute force = 6.220 мс, Z3 = 0.346 мс;

n=19: brute force = 19.309 мс, Z3 = 0.321 мс;

n=20: brute force = 15.168 мс, Z3 = 0.360 мс;

n=25: brute force = 124.952 мс, Z3 = 0.291 мс.

Анализ данных выявляет критическую точку между n=13 и n=15, где брутфорс начинает демонстрировать экспоненциальный рост временной сложности, соответствующий предсказанной теоретической $O(2^n)$ сложности.

На n=25 Z3 превосходит брутфорс в 400 раз. Z3 демонстрирует практически константное время выполнения (0.2-0.4 мс) независимо от n, что подтверждает эффективность CDCL-алгоритма при работе с реальными формулами.

График сравнения



СПБГЭТУ «ЛЭТИ»
ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ

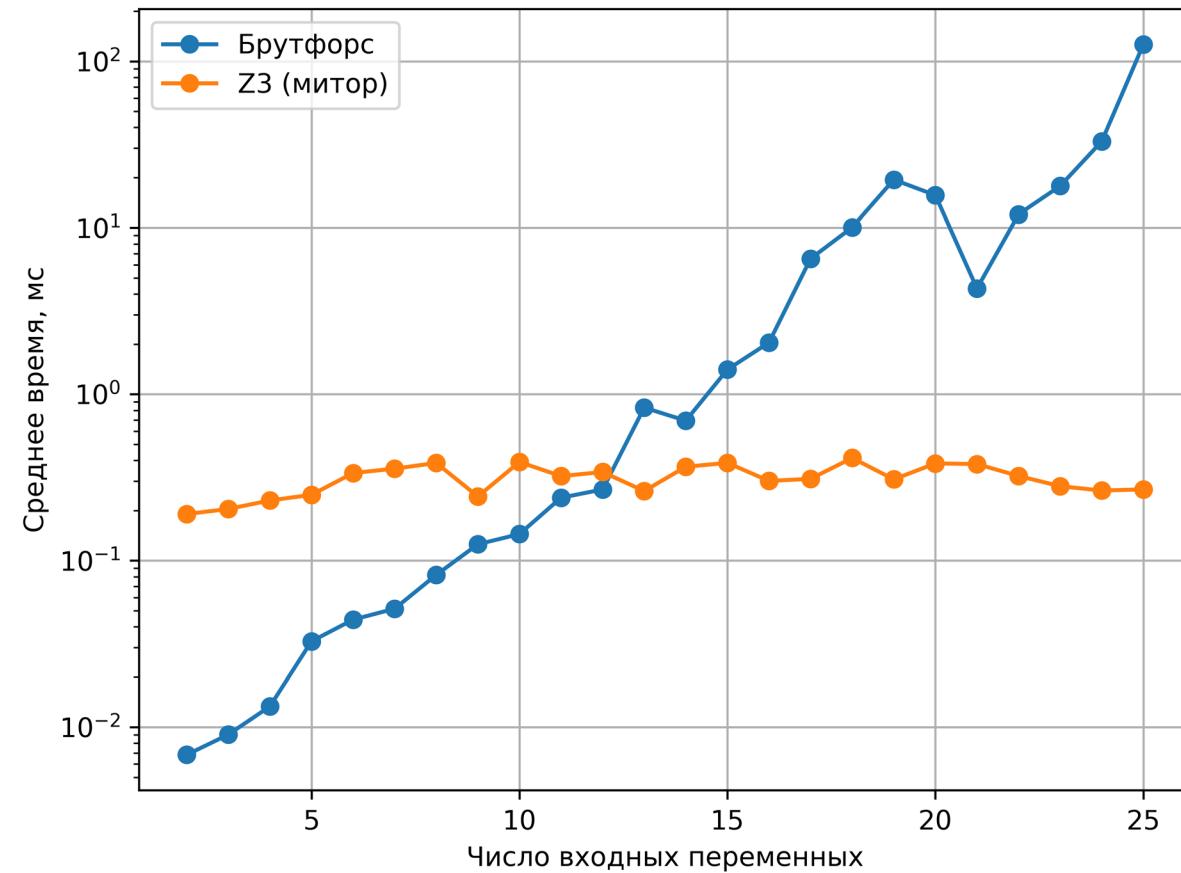


График зависимости времени выполнения от числа входовых переменных

Визуализация результатов на графике с логарифмической шкалой времени выявляет контрастные характеристики двух подходов.

Кривая brute force демонстрирует характерный для $O(2^n)$ функции вид: практически горизонтальная для $n \leq 13$, затем резкое экспоненциальное возрастание.

Кривая Z3 проявляет субэкспоненциальное поведение: практически горизонтальная на всём диапазоне с малой вариативностью.

Это графически демонстрирует качественное различие между подходами: полный перебор экспоненциален по определению, тогда как CDCL показывает практическую нечувствительность к размеру входа на экземплярах со структурой.

Область пересечения кривых находится при $n \approx 10-12$ переменных, что соответствует точке уравновешивания накладных расходов Z3 и превосходства эффективности алгоритма при возрастании сложности.

SAT-решатели используются в критических приложениях современной цифровой индустрии:

- (1) Верификация аппаратуры (Hardware Verification) – SAT является основным инструментом при проверке эквивалентности схем (Equivalence Checking) в процессе проектирования СБИС, что обеспечивает корректность микропроцессоров и других цифровых систем;
- (2) Формальная верификация программного обеспечения (Model Checking) – использование SAT для автоматической проверки моделей систем против спецификаций безопасности;

(3) Оптимизация и планирование — решение задач удовлетворения ограничений (CSP) при планировании производства, составлении расписаний, размещении ресурсов;

(4) Криптографический анализ — использование SAT-решателей для анализа криптографических примитивов, поиска коллизий, обнаружения уязвимостей;

(5) Синтез программ — автоматическое синтезирование кода, удовлетворяющего заданным спецификациям посредством SAT-редукции.

Глобальный рынок средств верификации аппаратуры, опирающихся на SAT-технологию, оценивается в миллиарды долларов, что подтверждает критическую значимость этой технологии.

Несмотря на значительные успехи, CDCL-решатели сталкиваются с рядом трудностей:

- (1) Масштабируемость на очень большие экземпляры — формулы с сотнями тысяч клауз остаются вызовом, требуя часов вычисления;
- (2) Зависимость от структуры экземпляра — случайные формулы часто решаются быстро, тогда как структурированные экземпляры (возникающие из реальных приложений) могут демонстрировать непредсказуемо долгое время решения;
- (3) Качество эвристик выбора переменной — неправильный выбор эвристики может замедлить алгоритм в 1000 раз;

- (4) Фазовые переходы — для некоторого класса случайных формул наблюдается резкий переход от простого к сложному при определённом соотношении клауз к переменным, создающий "пиковую сложность";
- (5) Проблема несбалансированного поиска — алгоритм может зависнуть в неудачной ветви, несмотря на наличие других решающих путей.

Указанные ограничения остаются активной областью исследований, включая разработку адаптивных эвристик, параллельных SAT-решателей и гибридных подходов.

Проведённое исследование подтвердило следующие ключевые положения:

- (1) Проблема SAT остаётся NP-полной, и методы полного перебора экспоненциальны по определению, практически применимы только для $n \leq 25$ переменных;
- (2) Алгоритм DPLL, базирующийся на unit propagation и backtracking, обеспечивает значительное сокращение пространства поиска по сравнению с полным перебором, но не использует информацию из конфликтов;

- (3) Алгоритм CDCL, расширяющий DPLL посредством clause learning, non-chronological backtracking и интеллектуальных эвристик, достигает практических решений задач со сложностью, казалось бы, непреодолимой для методов полного перебора;
- (4) Z3 SMT-решатель, реализующий CDCL с многочисленными оптимизациями, демонстрирует ускорение на 100-400 раз по сравнению с брутфорсом на формулах масштаба 20-25 переменных;

(5) SAT-технология является критической инфраструктурой для верификации аппаратуры, тестирования ПО и множества других приложений в современной цифровой индустрии

Результаты подтверждают гипотезу о том, что интеллектуальные алгоритмические подходы преодолевают теоретические барьеры экспоненциальной сложности и обеспечивают практическую разрешимость задач, неразрешимых наивными методами.

Барченков П.А. — Главы 1,2, управление проектом

Лебедев И.А. — разработка программного комплекса

Шарапов И.Д. — тестирование ПК, Глава 3, презентация