Применение SAT-решателей для верификации комбинационных схем

Выполнили студенты группы 3312:
Барченков Платон Алексеевич
Лебедев Игнат Артемович
Шарапов Иван Денисович

Введение

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

Реализовать верификацию схемы с использованием Z3 или MiniSat и сравнить с традиционными методами.

Суть проблемы

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).

Суть проблемы

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.

Математические основы

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

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 → {0, 1}, такая что F(I) = 1.

Математические основы

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.

В нашей работе используется преобразование Цейтина (Tseitin transformation). Цейтин гарантирует линейный рост по размеру AST:

- на каждый узел добавляется константное число клауз
- добавляется одна новая переменная _aux

Ограничения полного перебора

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.

SAT-редукция

Одним из значительных практических применений SAT-решателей является верификация эквивалентности цифровых схем. Пусть даны две булевы 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

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). Работает исключительно с формулами в конъюнктивной нормальной форме (КНФ). 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;

Алгоритм DPLL

(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.

В худшем случае временная сложность DPLL остаётся экспоненциальной.

Алгоритм CDCL

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* анализируются причины конфликта: выполняется обход графа импликаций в обратном направлении; на основе этого анализа синтезируется новая клауза, 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;

*Конфликт в CDCL — это ситуация, когда при текущем частичном присваивании хотя бы одна клауза полностью ложна (пустая клауза).

Алгоритм CDCL

(3) VSIDS Heuristic (Variable State Independent Decaying Sum) — эвристика выбора переменной, которая приоритизирует переменные, часто появляющиеся в недавних конфликтах, с экспоненциальным затуханием старых данных;

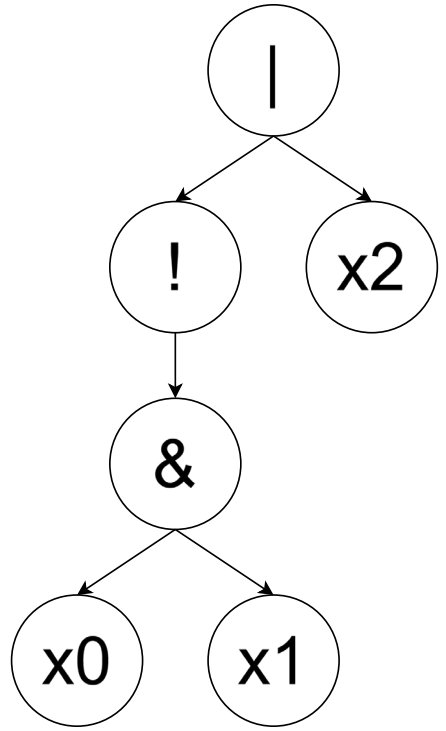
(4) Restarts — периодическая перезагрузка процесса поиска с сохранением выученных клауз, что помогает преодолевать "застревание" в неудачных ветвях;

(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.

AST

Абстрактное синтаксическое дерево (англ. abstract syntax tree, AST) — конечное помеченное ориентированное дерево, в котором внутренние вершины сопоставлены (помечены) с операторами, а листья — с соответствующими операндами. Таким образом, листья являются пустыми операторами и представляют только переменные и константы.




```
F = !(x0 & x1) | x2
```

Z3 решатель

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.

VSIDS-эвристика

Variable State Independent Decaying Sum

Каждой переменной x сопоставляется числовой параметр активности A(x). При возникновении конфликта и построении выученной клаузы активность переменных, вошедших в эту клаузы, увеличивается (операция bump).

Далее активности подвергаются затуханию (decay): вклад давних конфликтов постепенно уменьшается, а свежие конфликты оказываются важнее. Переменная для следующего решения выбирается как переменная с максимальной активностью среди не назначенных.

Интуитивно это означает, что решатель концентрируется на переменных, которые чаще всего оказываются вовлечены в противоречия и, следовательно, лучше всего «разрезают» пространство поиска.

Полный перебор

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

Пусть Var = $\{x_0, x_1, \ldots, x_{n-1}\}$ — объединение множеств переменных обеих формул. Для каждой интерпретации σ: Var → {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

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.

Использование Z3

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

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);

*Z3-выражение — это синтаксический объект (AST), описывающий терм или логическую формулу над теориями Z3, используемый для задания ограничений и рассуждений в SMT-решении.

Использование Z3

(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.

Учебный DPLL

Входом алгоритма является формула в конъюнктивной нормальной форме, полученная из AST митора с использованием преобразования Цейтина. Основные этапы работы учебного DPLL:

- разбор формул F1 и F2 в AST и построение митора $M=(F1 \wedge \neg F2) \vee (\neg F1 \wedge F2)$;
- преобразование митора в КНФ с введением вспомогательных переменных;
- unit propagation — принудительное присваивание значений переменным из единичных клауз;

Учебный DPLL

- выбор неинициализированной переменной и ветвление по значениям;
- backtracking при обнаружении конфликта (пустой клаузы).

Интерпретация результата:

SAT — найдено удовлетворяющее присваивание, схемы неэквивалентны;

UNSAT — модель отсутствует, схемы эквивалентны.

Учебный CDCL

Построение митора и перевод в КНФ выполняются аналогично варианту с DPLL. Ключевые механизмы учебного CDCL:

- unit propagation с сохранением клауз-причин для каждого принудительного присваивания;
- обнаружение конфликта при полной ложности некоторой клаузы;
- анализ конфликта и построение выученной клаузы (clause learning);
- добавление выученной клаузы в формулу;

Учебный CDCL

- нехронологический возврат (backjump) на уровень, определяемый структурой конфликта.

Использование обучения позволяет исключать повторение одинаковых конфликтных ситуаций и уменьшать пространство поиска.

Интерпретация результата:

SAT — найден контрпример, схемы неэквивалентны;

UNSAT — контрпримеров не существует, схемы эквивалентны.

Ручное сравнение

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 ($x_0 \wedge x_1$ vs $x_1 \wedge x_0$): bruteforce = 0.012 мс, DPLL = 0.209 мс, CDCL = 0.370 мс, Z3 = 60.015 мс;

Пример 2 ($x_0 \wedge x_1$ vs $x_0 \vee x_1$): bruteforce = 0.008 мс, DPLL = 0.125 мс, CDCL = 0.169 мс, Z3 = 7.522 мс;

Пример 8 ($x_0 \rightarrow x_1$ vs $\neg x_0 \vee x_1$): bruteforce = 0.011 мс, DPLL = 0.130 мс, CDCL = 0.250 мс, Z3 = 0.059 мс.

Ручное сравнение

Анализ показывает, что на малых входах брутфорс часто быстрее Z3 из-за накладных расходов инициализации Z3. DPLL и CDCL «стабильны» и показывают близкое друг к другу время.

Наблюдается высокая вариативность Z3, что объясняется сложностью конкретной формулы и эффективностью применяемых эвристик.

Автоматическое сравнение

Основной набор экспериментов включал бенчмарки на формулах с числом переменных от 2 до 100. Результаты (средние значения из 5 проходов):

n	Bruteforce, мс	DPLL, мс	CDCL, мс	Z3, мс
4	0.021	0.538	1.047	0.294
10	1.447	3.317	6.142	0.340
12	6.424	1.911	2.935	0.342
22	9175.2	18.12	30.91	0.365
70	-	202.1	347.0	0.469
90	-	658.1	987.7	0.477

Полная таблица

Автоматическое сравнение

Анализ данных выявляет критическую точку между n=10 и n=12, где брутфорс начинает демонстрировать экспоненциальный рост временной сложности, соответствующий предсказанной теоретической $O(2^n)$ сложности.

На n=22 Z3 превосходит брутфорс в тысячи раз. Z3 демонстрирует практически константное время выполнения (0.2-0.4 мс) независимо от n.

Учебные версии DPLL и CDCL также демонстрируют отличное время, но заметен рост времени с ростом n.

График сравнения

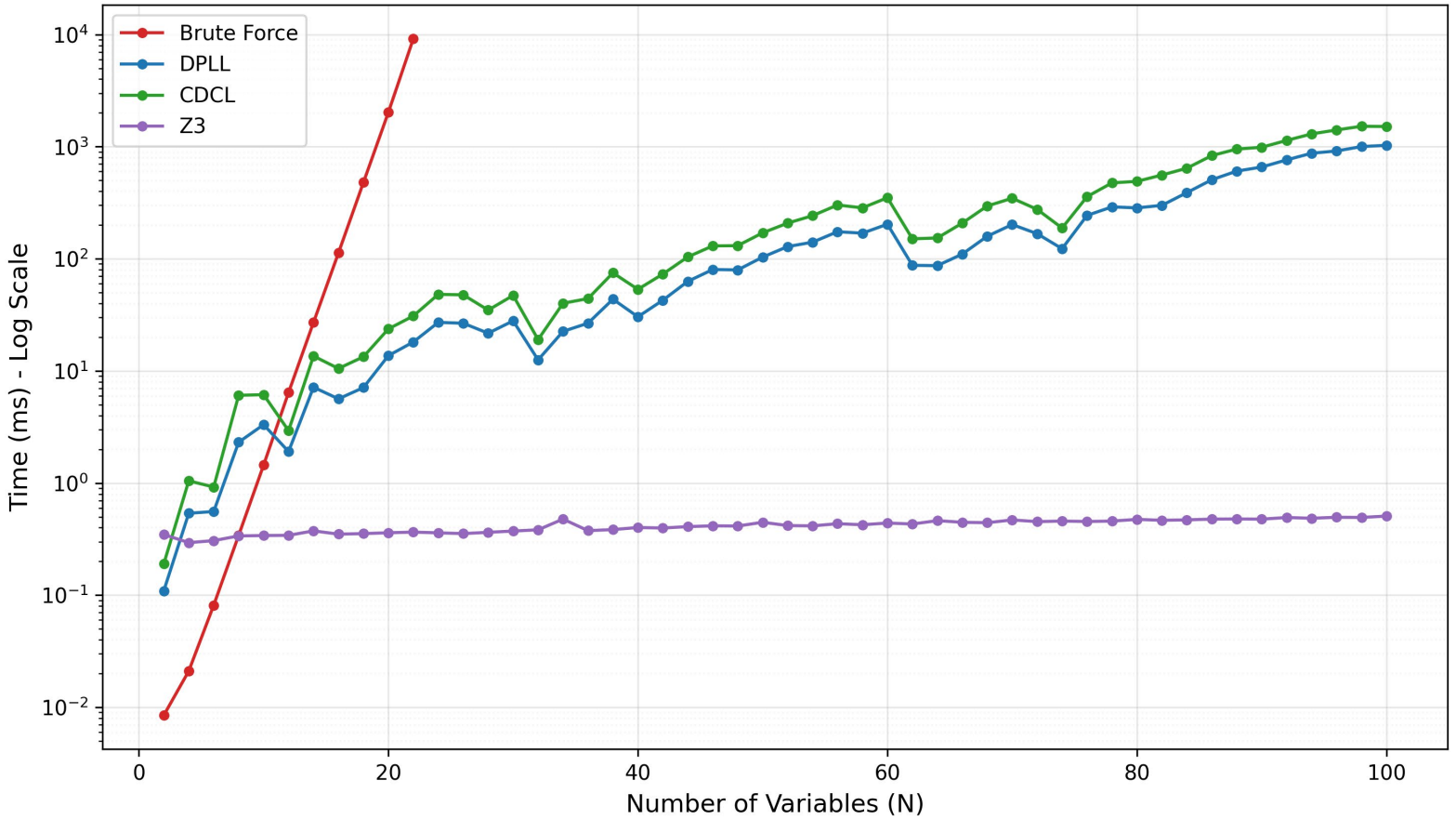


График зависимости времени выполнения от числа входовых переменных

Анализ графика

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

Кривая brute force демонстрирует характерный для $O(2^n)$ функции вид: резкое экспоненциальное возрастание.

Кривая Z3 демонстрирует слабую зависимость от размера входа и выглядит почти горизонтальной. Это связано с тем, что при малых и средних размерах формул основную долю времени составляют фиксированные накладные расходы.

Учебные DPLL и CDCL растут с небольшой скоростью.

Анализ графика

Это графически демонстрирует качественное различие между подходами: полный перебор экспоненциален по определению, тогда как Z3 показывает практическую нечувствительность к размеру входа на экземплярах со структурой.

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

Применение SAT-технологий

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;

Применение SAT-технологий

(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.

Ограничения и проблемы

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;

Ограничения и проблемы

(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.

Выводы

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;

Выводы

(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;

Выводы

(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.

Роли в команде

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

Лебедев И.А. — разработка программного комплекса

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