SMT-решатели и с чем их едят

или

Как перестать беспокоиться и полюбить NP-полные задачи

Михаил Беляев

12 декабря 2016

Задача SAT

Задача о выполнимости булевых формул

1. Есть некая формула в логике 1-го порядка
2. Нужно определить, существует ли такой набор значений входящий в неё переменных, при котором она становится истинной

Два возможных исхода: SAT (т.е. да) и UNSAT (т.е. нет)

Задача SAT

Одна из первых NP-полных задач (первая, для которой доказана NP-полнота)

Задача SAT

Одна из первых NP-полных задач (первая, для которой доказана NP-полнота)

Одна из первых NP-полных задач, **которые научились эффективно решать**

Почему это так важно?

Задача SAT

Одна из первых NP-полных задач (первая, для которой доказана NP-полнота)

Одна из первых NP-полных задач, **которые научились эффективно решать**

Почему это так важно?

Потому, что все NP-полные задачи сводимы друг к другу

Задача SAT: пример

Формула:

$$(x_1 \vee \neg x_5 \vee x_4) \wedge (\neg x_1 \vee x_5 \vee x_3 \vee x_4)$$

Представление:

```
1 -5 4 0
-1 5 3 4 0
```

Задача SAT: пример

Формула:

$$(x_1 \vee \neg x_5 \vee x_4) \wedge (\neg x_1 \vee x_5 \vee x_3 \vee x_4)$$

Представление:

```
1 -5 4 0
-1 5 3 4 0
```

Решение:
SAT

$$x_1 = True \quad x_3 = False \quad x_4 = True \quad x_5 = False$$

SAT-солверы

- Средства эффективного решения задачи SAT
- С появлением SAT-солверов NP-сложность задачи перестала быть приговором

SAT-солверы: что можно использовать

- MiniSAT / CryptoMiniSAT
- Lingeling (PicoSAT)
- Sat4j
- Glucose
- Есть куча проприетарных решений

Входной формат: обычно КНФ в формате CNF-DIMACS

Задача выполнимости vs задача доказательства

Задача выполнимости: существуют такие значения переменных, для которых формула верна

Задача доказательства: для любых значений переменных формула верна

Какая из задач сложнее?

Задача выполнимости vs задача доказательства

Задачи выполнимости и доказательства *дуальны* и сводятся друг к другу:

- Формула $F(X)$ выполнима, **iff** $\neg F(X)$ неверна
- Формула $G(X)$ верна, **iff** $\neg G(X)$ невыполнима

В дальнейшем все средства решения этих задач будут называться (SAT/SMT)-солверами, хотя многие из них считают себя «автоматическими доказателями теорем»

Вопросы терминологии

- К результатам типа SAT прилагается *модель*:
  Набор **каких-то** значений, на которых формула выполняется
- К результатм типа UNSAT может прилагаться т.н. unsat core:
  Куски формулы, конъюнкция которых тоже UNSAT

Задача SMT

Satisfiability Modulo Theories

Задача выполнимости булевых формул **с учётом лежащих в их основе теорий**

Задача SMT

Что такое теории?

- Набор возможных значений и операций над ними
- Теории можно комбинировать в новые теории
- **В новых теориях задача разрешимости может перестать быть NP-полной или вообще разрешимой**

Задача SMT: пример

Формула:

$$x_1 \in \mathbb{Z} \quad x_2 \in \mathbb{Z}$$
$$x_1^2 + 2x_1 + 4 = 2x_2$$

Представление:

```
(declare-const x_1 Int)
(declare-const x_2 Int)
(assert (= (+ (* x_1 x_1) (* 2 x_1) 4) (* 2 x_2)))
(check-sat)
(get-model)
```

Решение:

$$x_1 = -4 \quad x_2 = 6$$

Теории

Часто приводят формулу:

SMT-solver = SAT-solver (boolean core) + *T-solvers*

На практике это корректно только для солверов на основе разновидностей DPLL

Булева теория

Код: нет

Поддерживается: всеми

Операции: те же, что и в задаче SAT

Пустая теория

Код: UF
Поддерживается: почти всеми
Операции: объявления функций, применения функций

Теория «пустых» функций, содержащих только объявления, работает за счёт определения понятия «функция»

При комбинировании теорий функции могут принимать на вход и возвращать объекты из других теорий

Разрешима, в чистом виде сводится к задаче SAT

Теория целых чисел

Код: (L|NL)IA
Поддерживается: Z3, CVC4, MathSAT, etc.
Операции: линейной (нелинейной) арифметики над целыми числами

Теория целых чисел в общем случае **неразрешима**

Теория вещественных чисел

Код: (L|NL)RA
Поддерживается: Z3, CVC4, MathSAT, etc.
Операции: линейной (нелинейной) арифметики над действительными числами

По факту иррациональные числа не поддерживаются ни в задачах, ни в результатах из-за невозможности их представления

Теория вещественных чисел, как ни странно, разрешима с поправкой на вышесказанное

Теория битовых векторов

Код: BV
Поддерживается: почти всеми
Операции: машинная арифметика, в том числе битовая, сравнение машинных слов

Представляют собой набор бит ограниченного размера, очевидным образом сводится к задаче SAT, поэтому разрешима. Процесс сведения называется bit-blasting.

Теория массивов

Код: A(X)
Поддерживается: Z3, MathSAT, CVC4, Boolector, STP, etc.
Операции: создание массива, чтение массива, запись в массив

Массив это сложный объект, привязанный к теории индексов (объекты которой используются как индексы) и теории элементов (объекты которой используются как элементы).

Интересный факт: очевидным образом выражается через теорию UF, наоборот тоже, но не столь очевидно.

Разрешима, если разрешимы входящие в неё теории.

Теория AX отличается аксиомой:

$$\forall a, b : (\forall i : a[i] = b[i]) \implies a = b$$

Теория кортежей и структур данных

Код: нет
Поддерживается: Z3, ???
Операции: другие теории + создание и чтение алгебраических структур данных

Сводится к неинтерпретированным функциям, но плохо совместима с другими теориями

Теория чисел с плавающей точкой

Код: FP (будет)
Поддерживается: Z3, ???
Операции: машинные операции над числами с плавающей точкой

Сводится к задаче SAT, поэтому должна быть разрешима.

Теория строк

Код: пока нет, вероятно S
Поддерживается: Z3, CVC4, ???
Операции: валидируемые операции над строками символов

Разрешима ли?

SAT modulo SAT

Код: отсутствует
Поддерживается: ModSAT, IC3
Операции: булевы + запросы выполнимости

Теория, которая рассматривает сам процесс решения задачи SAT как теорию внутри задачи SAT. Интересна в основном теоретикам построения SMT-солверов.

Кванторы

- Любая версия задачи SAT содержит над собой неявный квантор существования
- Любая версия задачи доказательства содержит над собой неявный квантор общности
- Кванторы **внутри** формул очень сильно усложняют решение задачи
- Для булевых формул это почти не имеет значения
- Для более сложных теорий всё может стать очень плохо очень быстро
- Проблема решается засчёт алгоритмов *ликвидации кванторов*

Кванторы

- Ликвидация кванторов — очень долгий и сложный процесс
- Может потребовать множества запусков процедуры решения с проверкой модели
  (MBQI — model-based quantifier instantiation)

Когда кванторы можно игнорировать

- Кванторы существования на верхнем уровне

```
(assert (= x 5))
(assert (exists ((y Int) (z Int)) (/= y (+ x z))))
```

Это ровно то же самое, что и

```
(assert (= x 5))
(declare-const y Int)
(declare-const z Int)
(assert (/= y (+ x z)))
```

Этот процесс называется **сколемизацией**

Теории с кванторами

По умолчанию код любых теорий позволяет использовать кванторы.

Теории без кванторов имеют префикс QF_.

Примеры:

- QF_ABV — теория бит-векторов и массивов, без кванторов
- AUFLIA — теория линейной арифметики над целыми числами с массивами и функциями

Какие бывают солверы

Bit-blasting solvers: сводят всё к задаче SAT, которую решают SAT-солвером
Примеры: Boolector, STP
Как правило, поддерживают только QF_AUFBV и ниже

T-solver-based solvers: используют интеллектуальные решения для теорий
Примеры: Z3, MathSAT, CVC4
Могут поддерживать всё, что угодно, проблемы возникают с комбинациями теорий

Что такое хорошо и что такое плохо

- Что солверы делают хорошо?
  - Находят случаи, о которых вы даже не думали
  - Выбирают единственный вариант в условиях сложных ограничений
- Что солверы делают плохо?
  - Решают задачи на индукцию (ооочень плохо)
  - Работают с «тяжелыми» операциями типа целочисленного деления
  - Делают поиск в бесконечном пространстве

Что не нужно делать с солверами

- Пытаться строить абстракции на конструкциях самого солвера
  - Использовать функции как функции, кортежи как кортежи и т.д.
  - Чем тупее формула, тем лучше
- «Утаивать» от солвера критическую информацию об ограничениях
  - Использовать переменные там, где хватило бы констант
  - Давать сложные подформулы, не имеющие отношения к задаче

Что нужно делать

- Строить абстракции в своём языке, храня в них элементарные формулы
- Передавать любые нетривиальные факты в явном виде

Что делать, если солвер тормозит?

- Ставить таймауты
- Подумать, что вы можете выкинуть из задачи
- Подумать, что вы ещё знаете о задаче, о чем солверу догадаться сложно
- Не переоценивать его мыслительные способности

Формулировка задачи и чтение результатов

- Формат входа: SMTLIB или SMTLIB2
  http://smtlib.cs.uiowa.edu
  - Диалект LISP
  - Теории описываются на нём же
- Единых форматов выходных данных не существует
  - Многие солверы умеют выдавать результат в формате SMTLIB2, но читать его очень сложно

Формулировка задачи и чтение результатов

- Формат входа: SMTLIB или SMTLIB2
  http://smtlib.cs.uiowa.edu
  - Диалект LISP
  - Теории описываются на нём же
- Единых форматов выходных данных не существует
  - Многие солверы умеют выдавать результат в формате SMTLIB2, но читать его очень сложно
- На нашей практике даже задачи, распечатанные Z3, не работали на том же Z3

Интерполирующие солверы

Решают задачу **интерполяции по Крейгу**:

> *Для любых формул A и B, таких, что $A \implies B$, существует такая формула C, состоящая только из символов, общих для и B, что $A \implies B$ и $B \implies C$. C называется интерполянтом Крейга*

Крейг доказал только существование интерполянта и только для логики первого порядка.

Работает не на всех теориях и не всегда хорошо.

Примеры: iZ3, MathSAT

Оптимизирующие солверы

Помимо обычных ограничений, можно вводить:

- Мягкие ограничения (soft assertions), которые выполнять не обязательно
- Оптимизационные ограничения
  «Найди мне решение системы уравнений над X, желательно так, чтобы $(X * 2 - 15)$ было максимальным»

Примеры: Z3opt

Солверы неподвижной точки

Позволяют решать задачу SMT над стандартным логическим программированием а-ля Datalog. Расширяют эту задачу на поддержку теорий и бесконечных областей решений.

Примеры: μZ3, Duality

Взаимодействие с солвером на практике

Есть два варианта:

- SMTLIB2
  - Плюс: легко заменять солверы
  - Минус: дополнительную информацию не достать
- Bindings
  - Плюс: достать можно всё
  - Минусы:
    - У каждого солвера свои тараканы
    - Обычные проблемы с FFI

Библиотеки

- MetaSMT — C++, абстрация над разными C API
  - Очень полезен, если вам нужно сделать то же самое
  - Там уже наступили на все грабли за вас
- SBV — Haskell, работает через внешний интерфейс
- ScalaSMT — Scala, тоже внешний интерфейс

Использование SMT-солверов в анализе

- Дедуктивная верификация
  - Как основа логического «движка»
  - Boogie/Dafny/VCC
  - Why3
- Верифицируемые языки
  - Аналогично
  - F*, chalice, Spec#, P
- Model Checking
  - «Нетрадиционный» model checking
  - SLAM
- Надстройки над Prolog/Datalog
  - Formula

Dynamic symbolic execution

- Символическое выполнение + SMT-солвер
- Как правило, анализируются только условные операторы
- Пытаемся покрыть программу тестами обычным фаззингом
- Где не получается — просим SMT-солвер подобрать входные данные

Примеры: SAGE, PEX

CEGAR

- Итеративный анализ на основе «предикатных абстракций»
- Как правило, анализируются только условные операторы
- Если ошибка не найдена — производится «refinement» модели программы на основе контрпримера из солвера
- С помощью интерполянтов Крейга можно «абстрагировать» куски кода в виде коротких формул

Примеры: CPAchecker, частично SLAM

Bounded model checking

- Программа преобразуется к конечному виду
- Конечная программа превращается в формулу
- Над этой формулой проверяется выполнимость формулы ошибки
- Возможно, процесс повторяется до получения «хорошей» конечной программы
- С помощью интерполянтов Крейга можно «абстрагировать» куски кода в виде коротких формул

Примеры: SBMC, CBMC, LLBMC, Borealis :-)

Анализ на основе систем типов

- В основном в игрушечных и/или функциональных языках
- Вычисляются только уравнения для систем типов
- Универсальный способ (vs более эффективные, но сложные алгоритмы вывода)

Примеры: Фреймворк «жидких» типов и зависимые типы вообще

Солвер как вспомогательное средство

- Анализ производится каким-то более простым методом(-ами)
- Если требуется более детальный анализ условий в ветках — их проверяют солвером

Примеры: Svace, Aegis, ???

Проблемы всех методов анализа на основе солверов

- Что делать с циклами и функциями типа `memcpy`?
  - Ничего
  - Что-то
- Что делать с памятью?
  - Ничего
  - Моделировать только часть семантики (например, «одинаковые указатели указывают на одинаковые значения»)
  - Моделировать память поблочно
  - Моделировать память побайтово

Проблемы всех методов анализа на основе солверов

- Что делать с межпроцедурностью?
  - Ничего
  - Контракты
  - Аппроксимации
  - Подставлять функцию полностью
- Что делать если солвер просто тормозит на ваших формулах?
  - Ничего
  - Агрессивный слайсинг
  - Оптимизации

Как выбрать нужный вам солвер

Disclaimer: на текущий момент, декабрь 2016

- Вам нужны только бит-вектора и массивы? Boolector
- Не нравится лицензия? STP
- Нужен солвер общего пользования со всеми фичами? Z3
- Не любите Microsoft? CVC4
- Специфические вещи, типа интерполяции над бит-векторами, требуют специфических решений

Но ведь кто-то их сравнивает?

- SMT-COMP — соревнование SMT-солверов
  http://www.smtcomp.org
- Внимание! Синтетические тесты
- Достаточно легко понять, из чего сейчас выбирать
- Демонстрируют недюженный прогресс в области

Почему стоит использовать SMT-солвер

- Задача SMT **очень сложна**
- Прогресс в этой области не стоит на месте
- Лучше переобобщить, чем недообобщить :-)

Почему не стоит использовать SMT-солвер

- Он почти неминуемо станет самой затратной частью вашего средства
  - Если это уже так — вы всё сделали правильно)
- Возможно, вам нужно более специфическое средство?
  - В каком-то смысле SMTLIB становится «ассемблером» методов решения логических задач
  - Солвер решает задачу в очень общем виде
  - Примеры: Boogie или Formula

А что с альтернативами SMT-солверам?

- SAT-солверы без теорий
- Datalog в его классическом виде и с расширениями
- ASP (Answer Set Programming) — альтернативный взгляд на почти ту же проблему

И как мне попробовать решать свою задачу?

Проще всего пойти на rise4fun http://rise4fun и пройти туториалы по Z3 прямо в браузере

Или скачать Z3 и попробовать примеры на Python.
Хоть это и не самый быстрый солвер, но он является ледоколом области.

Ссылки

- Эта презентация:
  https://bitbucket.org/belyaev.mikhail/smt-talk
- Z3:
  http://rise4fun.com/Z3
  http://rise4fun.com/iZ3
- CVC4: http://cvc4.cs.nyu.edu
- Boolector: http://fmv.jku.at/boolector
- MathSAT: http://mathsat.fbk.eu
- STP: http://stp.github.io/
- Boogie: http://rise4fun.com/Boogie
- Formula2: http://rise4fun.com/Formula2

belyaev@kspt.icc.spbstu.ru