Формальная верификация методом проверки модели

Захаров А.В.

Санкт-Петербургский государственный политехнический университет

2011

План

1. Формулировка задачи SAT, SMT
2. Абстракция
3. SatAbs

SAT - Satisfiability

Дана булевская функция $f(x_1, x_2, ...x_n)$, требуется

- проверить, выполнима ли эта функция, т.е. существует ли интерпретация, при которой функция истинна
- (если выполнима, то привести такую интерпретацию)

Пример

- Выполнима ли функция $f(x_1, x_2, x_3) = \bar{x_1} \vee x_2 \wedge x_3$?

SMT - Satisfiability Modulo Theories

SMT problem – задача выполнимости логических функций с учетом лежащих в их основе теорий

- предикаты могут быть определены в теории, отличной от алгебры логики
- например, в арифметике целых чисел, в арифметике вещественных чисел, в бит-векторной арифметике

Пример

- Выполнима ли функция $f(x_1, x_2, x_3) = (x_1 > 5) \wedge (x_1 < x_2 + x_3)$?

План

1 Формулировка задачи SAT, SMT

2 Абстракция

3 SatAbs

Экзистенциальная абстракция

- Каждая трасса в модели M является трассой в модели A
- A является аппроксимацией сверху (over-approximation) M: $A \models \phi \Rightarrow M \models \phi$
- Некоторые трассы в A могут не быть трассами в B

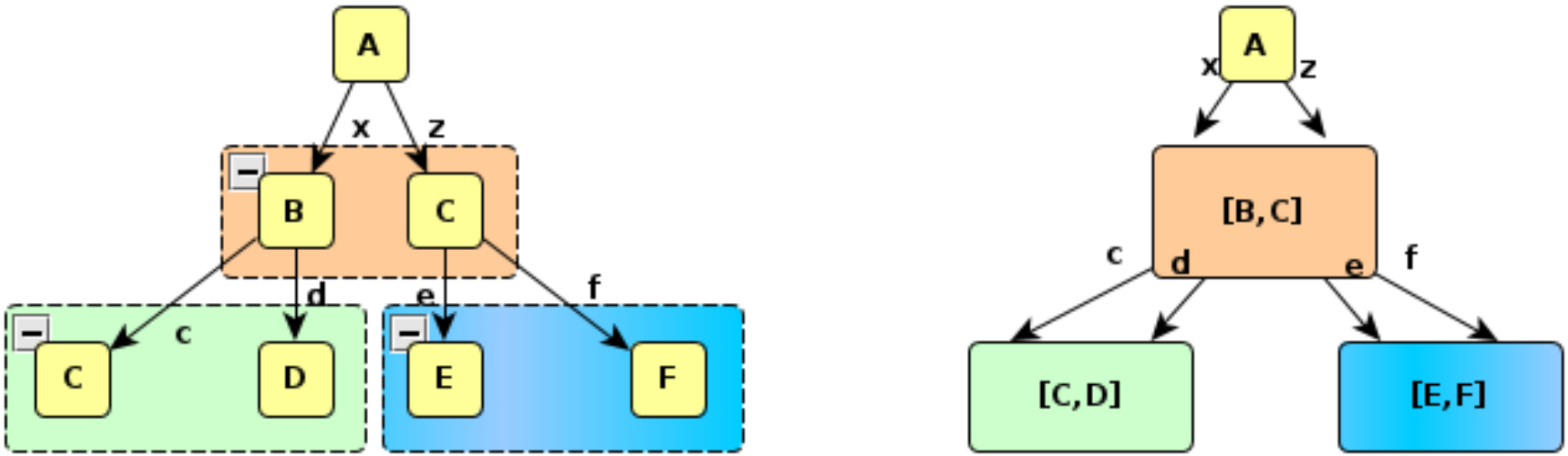

Figure: Экзистенциальная абстракция

Формализация абстракции

Даны: модель M и свойство ϕ, необходимо выбрать:

- множество абстрактных состояний S_h
- множество атомарных предикатов AP (для состояний из M и A)
- отображение $h : S \to S_h$, такое что $h(s) = h(t) \Leftrightarrow L(s) = L(t)$

План

1. Формулировка задачи SAT, SMT
2. Абстракция
3. SatAbs

Основные идеи

- Верифицируемые программы – ANSI C, без динамического выделения памяти и рекурсии
- Верифицируемые свойства – достижимость недопустимого состояния
- Используется подход CEGAR с экзистенциальной абстракцией (абстракция предикатов)
- Модель программы – программа над булевскими переменными
- Задача построения модели программы формулируется как задача SAT
- Если находится контр-пример, то выполняется ограниченный modelchecking

Представление программы

- Используется SSA-представление программы
- Каждое присваивание порождает новую версию переменной
- Множество линейных блоков операторов, соединенных между собой операторами передачи управления
- Каждый линейный блок – последовательность операторов $\{s_i\}$.
- Введены функции $lhs(s_i)$, $rhs(s_i)$, $vars(s_i)$

Отношение переходов

- Анализируется блок программы
- Строится *конкретное* отношение переходов τ
- Используется символьное представление отношения переходов
- $\tau(\bar{v}, \bar{v'}) = \bigwedge_{i=1..n} \sigma(s_i)$
  - $\bar{v}$ - вектор всех переменных v
  - $\sigma(s_i)$ – формула для оператора s_i

Формулы для операторов

- $\sigma(s_i)$ – формула для оператора s_i
- $\sigma(s_i)$ оперирует переменными из $\bar{V}$
- $\sigma(s_i)$ может содержать побитовые, арифметические операторы, оператор *with*
- Для построения используется функция трансляции $l(ls, rs)$
- $\sigma(s_i) = l(lhs(s_i), rhs(s_i))$

Функция трансляции l

Функция $l(ls, rs)$ определена рекурсивно по структуре ls:

- $l(v, rs) := v = rs$
- $l(g[i], rs) := l(g, g\ with\ [i] := rs)$
- $l(s.f, rs) := l(s, s\ with\ .f := rs)$

Оператор *with*:

- $g'[j] = g\ with\ [i] := e$
  - $g'[j] = g[j], j \neq i$
  - $g'[j] = e, j = i$
- $s' = s\ with\ .f := e$
  - $s'.j = e, j = f$
  - $s'.j = s.j, j \neq f$

Построение абстракции

- Дано множество предикатов $\mathcal{P}$ над переменными исходной программы
- Каждому предикату $\pi_i \in \mathcal{P}$ сопоставляется булевская переменная b_i
- Предикаты являются функциями абстракции: $\pi : \bar{V} \to 0, 1$
- Требуется построить *абстрактное* отношение переходов $\mathcal{B}(\bar{b}, \bar{b}')$

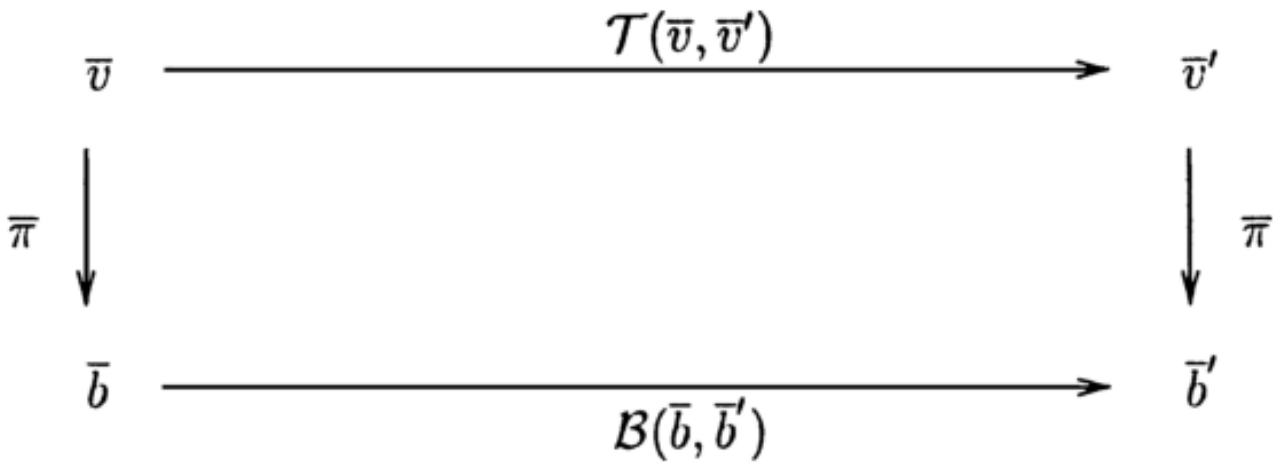

Figure: Абстракция

Построение абстракции

- Используется функция Γ:

$$\Gamma(\bar{b}, \bar{b'}, \bar{v}, \bar{v'}) = (\bar{\pi}(\bar{v}) = \bar{b}) \wedge \tau(\bar{v}, \bar{v'}) \wedge (\bar{\pi}(\bar{v'}) = \bar{b'}) \quad (1)$$

- Построение $\mathcal{B}(\bar{b}, \bar{b'})$ – задача SMT (в satabs реализована как SAT)

$$\mathcal{B}(\bar{b}, \bar{b'}) \Leftrightarrow \exists \bar{v}, \bar{v'} : \Gamma(\bar{b}, \bar{b'}, \bar{v}, \bar{v'}) \quad (2)$$

Анализ контр-примера

- Верификатор (SMV) выдает контр-пример в виде трассы $(\bar{b}^0 \rightarrow \bar{b}^1 \rightarrow \dots)$ для программы над булевскими переменными b_i
- Для проверки, существует ли соотвествующая трасса $(\bar{v}^0 \rightarrow \bar{v}^1 \rightarrow \dots)$ в программе, используется Γ:
  - для каждого перехода $\bar{b_c} \rightarrow \bar{b_c}'$ контр-примера проверить выполнимость SMT-формулы
  - $\Gamma(\bar{b}, \bar{b}', \bar{v}, \bar{v}') \wedge (\bar{b} = \bar{b_c}) \wedge (\bar{b}' = \bar{b_c}')$

Пример

```
int main(void)
{
    // ...
    int a = 2;
    int b = 3;
    int c = a + (b - a)/2;
    assert((c > a) && (c < b));
    // ...
}
```

Конкретное отношение переходов

- $V = a, b, c$
- $\sigma(s0) = a = 2$
- $\sigma(s1) = b = 3$
- $\sigma(s2) = c = a + (b - a)/2$
- $\tau(a, b, c, a', b', c') = (a' = 2) \wedge (b' = 3) \wedge (c' = a' + (b' - a'))/2$

Предикаты

- $\pi_0(a, b, c) = (c > a)$
- $\pi_1(a, b, c) = (c < b)$

Абстрактное отношение переходов

- $b_0 \sim \pi_0(a, b, c)$
- $b_1 \sim \pi_1(a, b, c)$
- Функция Γ:

$$\begin{aligned}\Gamma(b_0, b_1, a, b, c, a', b', c', b_0', b_1') = &(b_0 \leftrightarrow (c > a)) \wedge (b_1 \leftrightarrow (c < b)) \wedge \\ &\tau(a, b, c, a', b', c') \wedge \\ &(b_0' \leftrightarrow (c' > a')) \wedge (b_1' \leftrightarrow (c' < b'))\end{aligned}$$

- $\mathcal{B}(b_0, b_1, b_0', b_1') = b_0' \wedge b_1'$

Пример

```
int main(void)
{
    // ...
    int b0 = TRUE;
    int b1 = TRUE;
    assert(b0 && b1);
    // ...
}
```

Источники

1. E. Clarke, D. Kroening, N.Sharygina, K. Yorav. Predicate Abstraction of ANSI-C Programs Using SAT // Formal Methods for System Design (2004) 105–127
2. E. Clarke, D. Kroening, N.Sharygina, K. Yorav. SatAbs: SAT-Based Predicate Abstraction for ANSI-C // Tools and Algorithms for the Construction and Analysis of Systems, LNCS 2005 570-574
3. `www.cprover.org/satabs/`