Лекция 8. Добавление новых типов и функций

Цель лекции

Научиться описывать новые типы данных и новые функции, для которых нет теории в SMT-солвере. Понять, как моделировать типы языков программирования при создании модели программы. Увидеть отличия между языком спецификации и языком программирования. ((в одном неисполнимые термы и символьные преобразования и невыводимость, в другом исполнимые конструкции и ошибки с исключениями))

Добавление типов

- Пока мы использовали только целые числа, но этого недостаточно для моделирования памяти программы, например.
- Можно ли обойтись только теми типами, которые заранее определены в библиотеке языка, и не определять свои типы?
- Мы будем определять типы самостоятельно. Что такое тип?

Для чего нужны типы?

- В языке программирования (для компилятора): определить, сколько памяти надо выделить под переменную, определить нужные коды операций (они могут зависеть от типа операндов)
- В модели программы? В языке спецификации? Исключительно для методов Флойда: домены, выражения, предикаты. Это символьная природа в отличие от исполнимой природы типов данных в языках программирования.

Содержание

1 Немного матлогики

2 Теория неинтерпретируемых функций

3 UF в WhyML

4 UF в блок-схемах

Типы-символы, функциональные символы

Вспомним из курса математической логики (там были символы):

- Есть функциональные символы и символы-константы. Из них можно составлять термы.
- Есть типы-символы. Они ограничивают допустимые термы. Каждому символу-константе сопоставлен тип-символ. Каждому функциональному символу сопоставлены типы-символы для аргументов и результата.
- Есть предикатные символы, логические связки и кванторы. Из них (используя переменные) можно составить формулы.

Аксиомы и правила вывода

Чисто символьная задача – проверка выводимости:

- Задаются правила получения некоторых формул из некоторых других – правила вывода.
- Задается множество замкнутых формул – аксиомы.
- Задается замкнутая формула – цель.
- Надо проверить, можно ли использовать только правила вывода, чтобы из аксиом получить цель.

Значения формул, модели

- Но пока это лишь символьные преобразования. Оставаясь только с символьными преобразованиями, мы не можем использовать солверы (можем только пруверы) и трудно установить связь с типами данных в языках программирования. Нужны значения формулам.
- Модель – это сопоставление множеств типам-символам, функций – функциональным символам и предикатным символам, констант – символам-константам. Задав модель, у терма и формул появляются значения.
- Общезначимая формула – истинная при всех моделях.
- Выполнимая формула – истинная хотя бы при одной модели.

Логическое следствие

- Формула-цель является логическим следствием формул-аксиом, если в любой модели, где истинны все формулы-аксиомы, истинна формула-цель.
- Пример цели – условие верификации.
- SMT-солверы умеют «имитировать» пруверы, т.е. SMT-солвер может решать задачу определения логического следствия.
- Для этого ищется модель для формул-аксиом и отрицания формулы-цели. Есть модель – нет логического следствия. Нет модели – есть логическое следствие.

Содержание

1 Немного матлогики

2 Теория неинтерпретируемых функций

3 UF в WhyML

4 UF в блок-схемах

Теория неинтерпретируемых функций

- Обычно в формулах для SMT-солверов используются числа, битовые вектора и т.п. То есть домены с операциями, определения (модель) которых известны солверу. А для функциональных и предикатных символов модели нет.
- Если SMT-солвер умеет строить модели для символов, про него говорят, что он поддерживает теорию неинтерпретируемых функций (UF).
- Продемонстрируем, как может работать такой солвер.

Пример работы UF-солвера

Надо доказать или опровергнуть следующее логическое следствие:

- Символы: `a, b, c` из домена $\mathbb{Z}$, `f` из домена $\mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$.
- Аксиомы: `f(a, f(b, c)) != f(a, b)`, `f(b, b) = c`
- Цель: `f(a, b) = a`

Пример::шаг 1

Делаем отрицание цели и получаем множество формул, для которых надо искать модель (противоречие):

- `f(a, f(b, c)) != f(a, b)`
- `f(b, b) = c`
- `f(a, b) != a`

Пример::шаг 2

Введем новые переменные, чтобы убрать термы с более чем одним функциональным символом и чтобы функциональные символы входили только в равенства:

- `v1 = f(b, c), v2 = f(a, v1), v3 = f(a, b), c = f(b, b)`
- `v2 != v3, v3 != a`

Пример::шаг 3

Уберем функциональные символы совсем. Для этого надо применить следующее простейшее правило функции: $\forall x \forall y\ x = y \Rightarrow f(x) = f(y)$ и перебрать только те переменные вместо x и y, которые являются операндами функциональных символов.

- `a = b ∧ c = v1 -> v1 = v2`
- `a = b ∧ b = c -> v3 = v1`
- `b = c -> c = v1`
- `b = v1 -> v2 = v3`
- `a = b ∧ v1 = b -> c = v2`
- `a = b -> c = v3`
- `v2 != v3`
- `v3 != a`

Пример::шаг 4

Осталось перебрать все варианты разбиения множества символов `a`, `b`, `c`, `v1`, `v2`, `v3` на классы эквивалентности по равенству, чтобы в одном классе не оказались переменные, не равные друг другу. Если таких разбиений нет, то исходное логическое следствие доказано. Если разбиение есть, то найден контрпример, исходное логическое следствие не имеет места. Обычно эту задачу сводят к задаче SAT, обозначая равенство каждой пары переменных отдельной булевской переменной, добавляя аксиомы равенства в виде еще одних конъюнктов.

Пример::ответ

В этом примере есть разбиение на классы эквивалентности: `a = b, c = v3, v1 = v2`. Значит, логического следствия нет.

Содержание

1 Немного матлогики

2 Теория неинтерпретируемых функций

3 UF в WhyML

4 UF в блок-схемах

Синтаксис

- Тип-символ: `type T` (нет равенства после имени типа)
- Функциональный символ: `function sqrt (x: int): int` (нет равенства после этого заголовка)
- Предикатный символ: `predicate empty (x: T)`
- Аксиома: `axiom sqrtDef: forall x. sqrt x >= x`

Аксиомы

- Нужно быть крайне осторожными при определении аксиом!
- Пример аксиомы для символа квадратного корня:

```
function sqrt (x: int): int
axiom sqrtDef: forall x.
      sqrt x * sqrt x <= x
           < (sqrt x + 1) * (sqrt x + 1)
```

Проверка аксиомы

- Любая модель для sqrt – это функция из $\mathbb{Z}$ в $\mathbb{Z}$.
- Чему равно (sqrt 0) во всех моделях? Используем аксиому: sqrt 0 * sqrt 0 <= 0 < (sqrt 0 + 1) * (sqrt 0 + 1). Т.к. квадрат sqrt 0 не может быть отрицательным числом, то во всех моделях sqrt 0 равно 0. Аксиома правильная.
- Чему равно (sqrt 10) во всех моделях? (sqrt 10) * (sqrt 10) <= 10 < (sqrt 10 + 1) * (sqrt 10 + 1). Значит, значение (sqrt 10) может равняться только 3.

Продолжаем проверять аксиому

- Чему равно (sqrt -1) во всех моделях? Аксиома утверждает, что: (sqrt -1) * (sqrt -1) <= -1 < (sqrt -1 + 1) * (sqrt -1 + 1). Но таких (sqrt -1) не существует. То есть нет ни одной модели для символа sqrt с такой аксиомой. Аксиома противоречива. Это плохо.
- Исправляем аксиому:

```
function sqrt (x: int): int
axiom sqrtDef: forall x. x >= 0 ->
      sqrt x * sqrt x <= x
           < (sqrt x + 1) * (sqrt x + 1)
```

- Теперь аксиома для (sqrt -1) истинна при любой модели. Что это дает?
- Может быть надо считать, что (sqrt -1) равно некой ω? Ведь нельзя определить квадратный корень от -1. И считать, что в каждом типе есть ω.
- Но добавление ω усложняет формулы, а для символьных операций необходимы как можно более простые формулы.
- Более того, ω не нужна, т.к. и без нее при такой аксиоме никакие интересные формулы про квадратный корень от отрицательных чисел доказать нельзя. Далее смотрите примеры.

Примеры формул с недоопределенными символами

Являются ли логическим следствием указанной выше аксиомы следующие формулы?

- `forall x. sqrt x >= 0`? Нет.
- `forall x. sqrt x < 0`? Тоже нет.
- `forall x. sqrt x = sqrt x`? Да, но она неинтересна.
- `forall x. x >= 0 -> sqrt x >= 0`? Да, но она не использует sqrt от отрицательных чисел.

Интересный вопрос

- Можно ли алгоритмически определить выполнимость аксиом? То есть что существует хотя бы одна модель, в которой все аксиомы истинны.
- Если модели нет, то любые формулы будут логическими следствиями этого набора аксиом. Но какой смысл в таком следствии?..

Содержание

1 Немного матлогики

2 Теория неинтерпретируемых функций

3 UF в WhyML

4 UF в блок-схемах

Частично-определенные символы в формулах

- Нет ли ошибки в этой формуле? Если такая формула будет целью (например, это условие верификации), то надо ли генерировать дополнительную цель, что все «вызовы» sqrt делаются только для неотрицательных аргументов?
  `forall x a. a = sqrt x -> a * a <= x`
- Ответ: в формуле ошибки нет, формула – это просто «текст». Ошибка в формуле может быть лишь синтаксической (например, аргументов больше, чем в заголовке функционального символа sqrt). Эта формула не является логическим следствием аксиомы. Генерировать дополнительные условия не нужно, т.к. нет какой-либо последовательности действий, вычислений.

Частично-определенные символы в блок-схемах

- А вот в блок-схемах есть последовательность действий. Например, если в операторе ASSIGN надо использовать квадратный корень. Тогда в блок-схеме надо перед этим ASSIGN добавить CALL со взятием квадратного корня. Сопоставить квадратному корню спецификацию с предусловием. И тогда согласно методам Флойда будет построено условие верификации о том, что квадратный корень не берется от отрицательных чисел.
- Это же касается машинной арифметики (предотвращение арифметического переполнения), адресной арифметики (разыменование невалидного указателя) и т.п.

val-примитивы

- Что надо написать, чтобы в блок-схеме (let-функции) пользоваться типами данных, которые мы дополнительно определяем?
- Наличия `type`, `function`, `predicate` недостаточно, если надо доказывать корректность использования операций над типом данных
- Тогда надо написать val объявления (*val-примитивы*) - простейшие операции, необходимые для программирования с этим типом данных
- У каждого val-примитива должна быть спецификация – указываем у нее предусловие – и будут генерироваться условия верификации о корректности каждого использования этого примитива (предусловие val-объявления выполнено)