Эффективная разрешающая процедура для задачи выполнимости в теории номинальных систем типов с вариантностью

Милова Наталья Андреевна
группа 18М07-мм

научный руководитель: профессор кафедры СП, д.т.н, доцент Д. В. Кознов
консультант: старший преподаватель кафедры СП Д. А. Мордвинов
рецензент: программист ООО «Интеллиджей Лабс» Д. С. Косарев

Санкт-Петербургский государственный университет
Кафедра системного программирования

11 июня 2020 г.

Добрый день, уважаемая комиссия!
Меня зовут Милова Наталья. Я магистр направления «Программная инженерия».

- Тема моей работы: Эффективная разрешающая процедура для задачи выполнимости в теории номинальных систем типов с вариантностью.
- Научный руководитель: профессор кафедры СП, д.т.н, доцент Д. В. Кознов.
- Консультант: старший преподаватель кафедры СП Д. А. Мордвинов.
- Рецензент: программист ООО «Интеллиджей Лабс» Д. С. Косарев

Общая система типов CTS

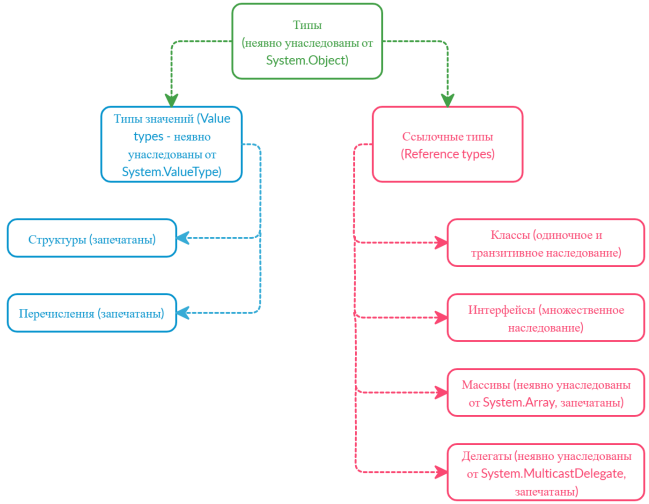

Рис.: Элементы CTS

Для платформы .NET специфицирована общая система типов (Common type system — CTS). Предполагается, что все типы в CTS неявно наследуются от `System.Object` и делятся на типы значений (структуры и перечисления) и ссылочные типы (классы, интерфейсы, массивы и делегаты).

Обобщенные типы CTS

Типовые параметры

- Инвариантные (`List<T>`)
- Ковариантные (`IEnumerable<out T>`)
- Контравариантные (`Action<in T>`)

Ограничения типовых параметров

- Ограничение базового класса `where T : <имя базового класса>`
- Ограничение интерфейса `where T : <имя интерфейса>`
- Ограничение `where T : U`
- Специальные ограничения
  - `where T : struct`
  - `where T : class`
  - `where T : new()`
  - `where T : unmanaged`

CTS поддерживает обобщенные типы (generics), типовые параметры которых уточняются при создании экземпляров данных типов. Для типовых параметров явно указывается их вариантность: инвариантность для классов и структур; инвариантность, ковариантность и контравариантность для интерфейсов и делегатов. На параметры обобщенных типов могут накладываться дополнительные ограничения.Стоит отметить, что массивы ковариантны по типу аргумента.

В целом общая система типов CTS является номинальной с вариантностью.

Влияние типов

Типы могут сохранятся во время выполнения методов и оказывать влияние на результаты их исполнения

Пример

```
private RegistryValueKind CalculateValueKind(Object value) {
    if (value is Int32)
        return RegistryValueKind.DWord;
    else if (value is Array) {
        if (value is byte[])
            return RegistryValueKind.Binary;
        else if (value is String[])
            return RegistryValueKind.MultiString;
        else
            throw new ArgumentException(value.GetType().Name));
    }
    else
        return RegistryValueKind.String;
}
```

Важной особенностью таких систем типов является то, что информация о типах сохраняется во время выполнения программ и может влиять на результат их исполнения (в отличие от языка OCaml, где информация о типах стирается на этапе компиляции). Информация о типах может появляться, например, при использовании операторов проверки типа (as, is) и выражений приведения типов.

Символьное исполнение

- Исполнение программ на символьных значениях
- Для каждой точки программы хранится условие пути и отображение переменных в символьные значения или выражения
- Использование SMT-решателя для проверки модели условия пути

Пример

```
int x, y;
if (x > y) {
    x = x + y;
    y = x - y;
    x = x - y;
    if (x - y > 0)
        assert (false);
}
```

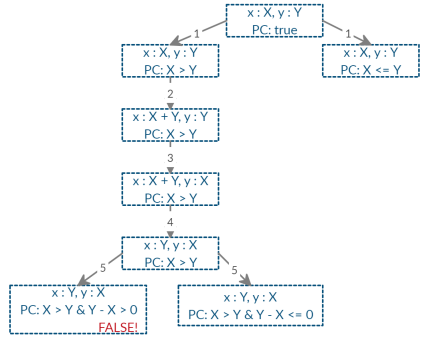

Рис.: Символьное исполнение swap двух целых чисел

Существует популярная техника анализа программ, используемая для генерации тестового покрытия и формальной верификации, которая называется символьное исполнение (symbolic execution). Она позволяет одновременно исследовать несколько путей исполнения, которые программа может пройти с разными входными данными. Основная идея заключается в том, чтобы программа исполнялась не на конкретных значениях, а на символьных. Механизм символьного исполнения для каждого пути потока управления поддерживает: (1) формулу логики первого порядка, которая описывает условия, выполненные по ветвям взятыми вдоль этого пути (path condition) и символьную память(symbolic memory store), которая отображает переменные в символьные выражения или значения. Ветка исполнения обновляет формулу, а присваивания обновляют символьную память. Средство проверки модели, обычно основанное на SMT-решателе, используется для проверки того, есть ли какие-либо нарушения свойства вдоль каждого исследуемого пути и достижим ли сам путь.

Проект V#

Проект V# направлен на создание символьного интерпретатора для платформы .NET

Пример

```
interface IComparable<in T>
class Base{}
sealed class Derived : Base, IComparable<Derived>{}

public static void F<T>(Base arg1, Derived arg2)
{
    if (arg2 is IComparable<T> && arg1 is T)
        throw new ArgumentException("Invalid arguments");
    ....
}
```

Условие корректности (невыполнимость формулы)

$$\phi \stackrel{\text{def}}{=} \exists \mathtt{T}, \mathtt{X}, \mathtt{Y}.\ \mathtt{X} <: \mathtt{IComparable<T>} \wedge \mathtt{X} <: \mathtt{Derived} \wedge \mathtt{Y} <: \mathtt{T} \wedge \mathtt{Y} <: \mathtt{Base}$$

В процессе верификации .NET-программ с помощью инструментов, основанных на символьном исполнении, например, проекта V#, требуется учитывать вышеуказанные особенности общей системы типов. Некоторые условия пути могут быть защищены ограничениями на типы, которые являются формулами теории первого порядка номинальных систем типов с вариантностью. Данная теория не поддерживается современными SMT-решателями, а для доказательства корректности программ необходимо определять выполнимость формул этой теории.

Цель и задачи

Цель

Разработать эффективную разрешающую процедуру для задачи выполнимости в теории номинальных систем типов с вариантностью для .NET в проекте V#

Задачи

1. Проанализировать основные элементы системы типов .NET
2. Исследовать задачу выполнимости в теории номинальных систем типов с вариантностью для .NET
3. Разработать и реализовать алгоритм проверки запросов на подтипирование в проекте V#
4. Провести экспериментальное исследование разработанного алгоритма

Основной целью данной работы является разработка эффективной разрешающей процедуры для задачи выполнимости в теории номинальных систем типов с вариантностью для .NET в проекте V#.
Для достижения поставленной цели требовалось

1. Проанализировать основные элементы системы типов .NET
2. Исследовать задачу выполнимости в теории номинальных систем типов с вариантностью для .NET
3. Разработать и реализовать алгоритм проверки запросов на подтипирование в проекте V#
4. Провести экспериментальное исследование разработанного алгоритма

Задача выполнимости в теории номинальных систем типов с вариантностью для .NET

Задача SUBTYPE-SAT

Дана таблица классов CT и формула первого порядка ϕ над сигнатурой $\Sigma = (\mathcal{C}, \{<:\})$, где $\mathcal{C}$ — множество конструкторов из таблицы классов, $<:$ — отношение подтипирования. Требуется найти закрытую подстановку свободных переменных, выполняющую ϕ или доказать ее отсутствие

- SUBTYPE-SAT неразрешима
- SUBTYPE-SAT полуразрешима

- Есть разрешимый фрагмент SUBTYPE-SAT

В ходе данной работы требовалось исследовать задачу выполнимости в теории номинальных систем типов с вариантностью. Данная задача предполагает, что дана таблица классов, формально описывающая систему типов, и формула первого порядка, состоящая из атомов подтипирования. Требуется найти закрытую подстановку свободных переменных, выполняющих эту формулу или доказать ее отсутствие. Данная задача сформулирована исследовательской группой проекта V#. В процессе исследований доказано, что задача неразрешима в общем случае и найден разрешимый полузакрытый фрагмент. Для разработки эффективной разрешающей процедуры для платформы .NET в ходе данной работы задача была адаптирована для общей системы типов, с учетом элементов CTS, описанных ранее.

Разрешающая процедура

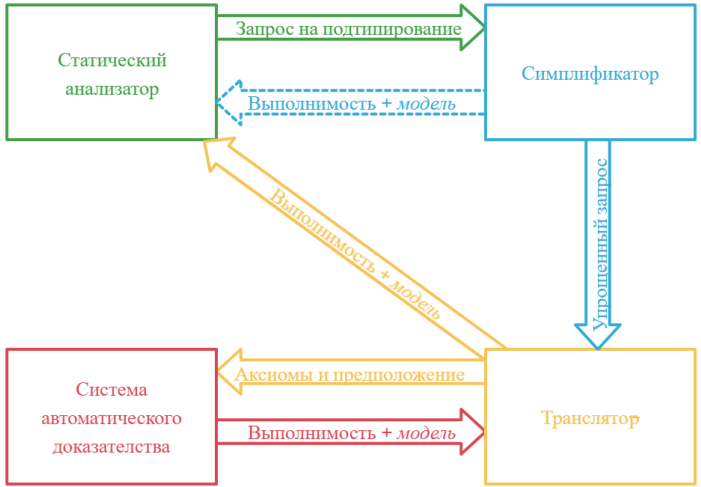

Рис.: Разрешающая процедура

В рамках данной работы разработана разрешающая процедура для задачи выполнимости ограничений на типы в .NET. Данная процедура включает несколько шагов.

1. В качестве этапа предобработки запрос на подтипирование, полученный от статического анализатора, упрощается с помощью применения правил подтипирования. В некоторых ситуациях выполнимость запроса может быть установлена уже на этой стадии.
2. При необходимости упрощенный запрос транслируется во множество дизъюнктов Хорна первого порядка.
3. Полученное в результате трансляции множество дизъюнктов используется в качестве входных данных для системы автоматического доказательства теорем.
4. Ответ системы доказательства преобразуется в заключение о выполнимости запроса и предъявляется модель (если она есть).

Пример трансляции

Таблица классов

$\texttt{Object}^R \quad <::$

$\texttt{I}^R\texttt{<}-x^R\texttt{>} \quad <:: \quad \texttt{Object}$

Формула

$\phi = x <: \texttt{I<}y\texttt{>} \wedge x \not<: y$

Пример дизъюнктов Хорна

- $\forall x.\ subtype(\texttt{I}(x), \texttt{Object})$
- $subtype(\texttt{Object}, \texttt{Object})$
- $\forall x.\ not\_subtype(\texttt{Object}, \texttt{I}(x))$
- $\forall x, y.\ subtype(\texttt{I}(x), \texttt{I}(y)) \leftarrow is\_reference(x) \wedge is\_reference(y) \wedge covar\_subtype(x, y)$
- $\forall x, y.\ not\_subtype(\texttt{I}(x), \texttt{I}(y)) \leftarrow is\_reference(x) \wedge is\_reference(y) \wedge not\_covar\_subtype(x, y)$
- $\forall x, y.\ subtype(x, \texttt{I}(y)) \wedge not\_subtype(x, y).$

Для того чтобы применить систему автоматического доказательства для определения выполнимости запроса на подтипирование, формируется множество аксиом — дизъюнктов Хорна первого порядка, из которых, в случае выполнимости, должна логически следовать формула, представляющая этот запрос. Аксиомы формируются на основании таблицы классов, которая отражает объявления типов из формулы.

Реализация

- Разрешающая процедура реализована в проекте V#
- Реализация на языках F# и C#, 1800 строк кода

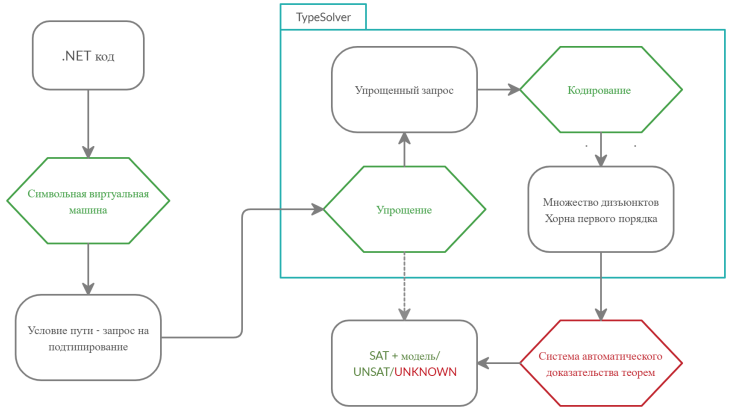

Рис.: Схема работы V# с модулем для решения запросов на подтипирование TypeSolver

Описанная процедура реализована виде модуля TypeSolver в проекте V#на языках F#и C# и включает 1800 строк кода. Данный модуль отвечает за решение условий пути, включающих запросы на подтипирование. В нем реализовано упрощение, кодирование и взаимодействие с системой автоматического доказательства теорем.

Экспериментальное исследование

- Около 30% запросов решаются на стадии упрощения
- Системы доказательств: Prolog, OCanren, CVC4, Prover9-Mace4, Vampire

Система	Выполнимые запросы (56)		Невыполнимые запросы (72)	
	Решено	Среднее время (мс.)	Решено	Среднее время (мс.)
Prolog	50	33	38	37
OCanren	55	16	38	58
Prover9-Mace4	32	88	6	20
CVC4	48	172	6	60
Vampire	56	66	70	90

Таблица: Результаты сравнительных экспериментов

Для проведения экспериментального исследования разработанного алгоритма были сгенерированы тесты, которые, отражают наиболее часто встречающиеся на практике запросы на подтипирование. В частности, с помощью символьной виртуальной машины V# исполнялись методы стандартных библиотек, например `mscorlib` и `System.Linq`, а также методы библиотеки созданной самостоятельно, включающей нетривиальные ограничения на типы и агрегировались запросы на подтипирование, порождаемые символьной виртуальной машиной.

По результатам экспериментов, около 30% запросов решаются на стадии упрощения. Для оставшихся запросов, в качестве кандидатов систем автоматического доказательства теорем логики первого порядка рассмотрены Prolog, OCanren, CVC4, Prover9-Mace4, Vampire. По результатам экспериментов, система Vampire показала лучшие результаты по количеству решенных запросов. Поэтому именно она была внедрена в проект V#. Результаты экспериментов показывают, что несмотря на то, что в общем случае задача выполнимости в теории номинальных систем типов с вариантностью неразрешима, предложенная разрешающая процедура с использованием Vampire эффективна, так как успешно определяет выполнимость 98% запросов встречающихся на практике.

Результаты

1. На основе стандарта ECMA-335 проанализированы основные элементы системы типов .NET
2. Исследована задача выполнимости в теории номинальных систем типов с вариантностью для .NET
3. В рамках проекта V# разработан и реализован на языках F# и C# алгоритм проверки выполнимости запросов на подтипирование, включающий упрощение исходной формулы с помощью применения правил подтипирования и кодирование упрощенной формулы во множество дизъюнктов Хорна первого порядка с дальнейшим применением автоматических систем доказательств теорем
4. Проведены сравнительные эксперименты систем доказательств на основе Prolog, OCanren, CVC4, систем Prover9-Mace4, и Vampire по времени работы и количеству решенных запросов на подтипирование, в ходе которых использование Vampire признано более предпочтительным для проекта V#

В ходе данной работы были получены следующие результаты.

1. На основе стандарта ECMA-335 проанализированы основные элементы системы типов .NET
2. Исследована задача выполнимости в теории номинальных систем типов с вариантностью для .NET
3. В рамках проекта V# разработан и реализован на языках F# и C# алгоритм проверки выполнимости запросов на подтипирование, включающий упрощение исходной формулы с помощью применения правил подтипирования и кодирование упрощенной формулы во множество дизъюнктов Хорна первого порядка с дальнейшим применением автоматических систем доказательств теорем
4. Проведены сравнительные эксперименты систем доказательств на основе Prolog, OCanren, CVC4, систем Prover9-Mace4, Vampire по времени работы и количеству решенных запросов на подтипирование, в ходе которых использование Vampire признано более предпочтительным для проекта V#