

Экстракция кода из Agda в Haskell

Шабалин Александр

научный руководитель
доц. Москвин Д. Н.

Академический университет
2014 г.

Формальная верификация

- ▶ Необходимо уметь убеждаться, что написанная программа решает поставленную задачу.
- ▶ Тестирование не может показать, что программа верна для всех случаев (если, конечно, нельзя сделать полный перебор).
- ▶ Формальная верификация позволяет сравнить программу с формальной математической моделью и доказать их эквивалентность на всех входных данных.

- ▶ Один из способов формально верифицировать — строить формулы достаточно мощной логики над элементами программы и проверять их на этапе компиляции.
- ▶ Agda позволяет строить формулы на языке предикативной конструктивной логики.

Использование верифицированного кода

Написание верифицированного алгоритма недостаточно — необходимо еще использовать этот код из «реальных» приложений. Подходы:

1. Использовать Agda для написания приложений целиком.
 - + Можно верифицировать больше кода.
 - Не Тьюринг-полный язык.
2. По коду на Agda генерировать код на другом языке
 - + Удобнее писать «реальный» код.
 - Необходимо поддерживать корректность кода при трансляции.

Второй пункт называется «экстракция программ» и используется в системе Coq.

Постановка задачи

Задача

По коду на Agda получить код на Haskell, который можно использовать из программы на Haskell, не нарушая внутренние инварианты, поддерживаемые Agda.

Haskell выбран по нескольким причинам:

- ▶ Языки синтаксически похожи
- ▶ Типы в Haskell — подмножество типов в Agda
- ▶ Часть работы уже сделана

Поскольку на Agda можно потребовать от аргументов функций свойств, не представимых в Haskell, то необходимо уметь запрещать давать прямой доступ к ним. Иначе, можно будет передать неправильный (с точки зрения Agda) аргумент, который пройдет проверку типов на стороне Haskell, и получить падение программы на этапе исполнения.

Существующие решения

Coq Экстракция программ¹. Генерируется код, из которого стираются все доказательства. Но это значит, что некоторые функции, требовавшие инварианты на этапе компиляции, теперь будут их требовать на этапе исполнения.

Agda 2.3.2.2 Компилятор MAlonzo². Фокусируется на генерировании исполняемых файлов через трансляцию в Haskell. Генерирует имена вида буква+число, теряет всю информацию о типах (кроме арности функций).

Agda 2.3.4 Появилась возможность давать пользовательские имена функциям и генерировать для них разумные типы.

¹P. Letouzey. *A New Extraction for Coq*. 2002

²<http://thread.gmane.org/gmane.comp.lang.agda/62>

Цели и задачи

Цель работы

Разработать механизм для MAlonzo, генерирующий интерфейс на Haskell к коду на Agda, использование которого не позволит нарушить инварианты, поддерживаемые Agda.

Задачи:

1. Разобраться с принципом кодогенерации в MAlonzo.
2. Разработать способ генерировать интерфейс из выделенных функций и типов данных и на этапе компиляции проверять, что он не нарушает инварианты.
3. Доказать корректность выполняемых трансформаций.
4. Реализовать и протестировать.

Реализация

- ▶ Выставляемый интерфейс представляет собой обертки над кодом, сгенерированным *MAlonzo*, которые имеют типы, поддерживающие те же инварианты, что требует код на Agda.
- ▶ Язык Agda расширен прагмой `{-# EXPORT AgdaName HaskellName #-}`, которой передается имя из Agda и желаемое имя в Haskell. Если сущность *AgdaName* представима в Haskell и *HaskellName* — разрешенное имя для этой сущности, то во время компиляции генерируется соответствующая обертка.
- ▶ Для модуля *AgdaModuleName* код интерфейса помещается в модуль *MAlonzo.Export.AgdaModuleName*, чтобы отделить код, сгенерированный *MAlonzo* (находящийся в *MAlonzo.Code.AgdaModuleName*) от безопасного интерфейса.

Выводы

Таким образом, разработан способ генерировать безопасный интерфейс на Haskell к коду на Agda и доказана его корректность.

EXTRA - Выполняемые трансляции

Объявления типов

EXTRA - Выполняемые трансляции

Типы функций

EXTRA - Выполняемые трансляции

Термы функций

EXTRA - Agda 2.3.4 COMPILED_EXPORT

TODO: What does it generate?