Экстракция кода из Agda в Haskell

Шабалин Александр Леонидович

научный руководитель

к. ф.-м. н. Москвин Денис Николаевич

Академический университет

2014 г.

Формальная верификация

- Способ доказать, что программа решает поставленную задачу. Тестирование только может показать это на ограниченном наборе данных.
- Agda — функциональный язык программирования с системой типой, позволяющей строить и проверять логические формулы над элементами языка.

$$zip : (lst_1\ lst_2 : List\ Nat) \to length\ lst_1 \equiv length\ lst_2 \to List\ (Nat \times Nat)$$

Использование верифицированного кода

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 («Экстракция программ»).
   - \+ Удобнее писать «реальный» код.
   - – Необходимо поддерживать корректность кода при трансляции.

Экстракция программ

- Уже есть транслятор: MAlonzo компилирует Agda через трансляцию в Haskell.
- Необходимо уметь запрещать прямой доступ к функциям со сложными предусловиями на аргументы.

Существующие решения

Coq 8.4pl3 Экстракция программ[1]. Из кода стираются все доказательства и инварианты.

Agda 2.3.2.2 Компилятор MAlonzo[2]. Написан для генерации исполняемых файлов. Генерирует имена вида буква+число, теряет почти всю информацию о типах.

Agda 2.4.0 Появилась возможность давать пользовательские имена функциям и генерировать для них разумные типы.

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

[2] M. Benke. *Alonzo — a compiler for Agda*. 2007

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. Провести анализ методов трансляции и принципов генерации кода в компиляторе MAlonzo.
2. Разработать механизм генерации интерфейса на Haskell к сгенерированному MAlonzo коду, не позволяющий нарушить инварианты, поддерживаемые Agda.
3. Доказать корректность генерируемого интерфейса.
4. Реализовать поддержку механизма экстракции в компиляторе MAlonzo и провести тестирование этой реализации.

Обзор реализации

- Генерируемый интерфейс — обертки над кодом, сгенерированным MAlonzo, которые имеют согласованные с Agda типы.
- Введена прагма `{-# EXPORT` *AgdaName* *HaskellName* `#-}`. Если сущность *AgdaName* представима в Haskell и *HaskellName* — разрешенное имя, то генерируется соответствующая обертка.
- Модуль с генерируемым интерфейсом помещается в поддерево *MAlonzo.Export*, отдельно от MAlonzo-генерируемого кода в *MAlonzo.Code*.

Подробности реализации

Тип данных и все его конструкторы выразимы

Agda:

$$\begin{aligned}&\textbf{data}\ \mathit{List}\ (A : \mathit{Set}) : \mathit{Set}\ \textbf{where}\\ &\quad \mathit{nil} : \mathit{List}\ A\\ &\quad \mathit{cons} : A \to \mathit{List}\ A \to \mathit{List}\ A\end{aligned}$$

Haskell:

$$\textbf{data}\ \mathit{List}\ (a :: *) = \mathit{Nil} \mid \mathit{Cons}\ a\ (\mathit{List}\ a)$$

- При реализации требуется, чтобы терм MAlonzo имел идентичную внутреннюю структуру терму интерфейса.
- Необходимо подменять тип, генерируемый MAlonzo.

Подробности реализации

Тип данных выразим, но хотя бы один конструктор не выразим

Agda:

$$\begin{aligned}&\textbf{data}\ \mathit{IsEqual}\ (A : \mathit{Set}) : \mathit{Set}\ \textbf{where}\\ &\quad \mathit{yes} : (x\ y : A) \to x \equiv y \to \mathit{IsEqual}\ A\\ &\quad \mathit{no} : (x\ y : A) \to x \not\equiv y \to \mathit{IsEqual}\ A\end{aligned}$$

Haskell:

$$\begin{aligned}&\textbf{newtype}\ \mathit{IsEqual}\ (a :: *) =\\ &\quad \mathit{IsEqual}\ (\forall b_0 \cdots b_3.\ \mathit{MAlonzoType}\ b_0 \cdots b_3)\end{aligned}$$

- Позволяет использовать `unsafeCoerce` для трансформации между этим типом и типом из MAlonzo.
- Такой же подход используется сейчас и для простых типов.

Подробности реализации

Функции

Agda:

$$identity : \{A : Set\} \to A \to A$$
$$identity\ x = x$$

Haskell:

$$identity :: \mathbf{forall}\ (a :: *).\ a \to a$$
$$identity = \lambda x \to \texttt{unsafeCoerce}$$
$$(mAlonzoTerm\ ()\ (\texttt{unsafeCoerce}\ x))$$

- MAlonzo оставляет типовой параметр аргументом функции.
- Из генерируемого интерфейса они убираются.

Доказательство корректности

- Выполнено методом структурной индукции.
- Вызов функции из интерфейса дает такой же результат, что и вызов в Agda.

Выводы

Таким образом:

1. Экстракция кода из Agda в Haskell, сохраняющая семантику, возможна для широкого класса типов данных и функций Agda.
2. Разработан способ генерировать безопасный интерфейс на Haskell к коду на Agda.
3. Доказана его корректность.

Agda 2.4.0

- Прагма `{-# COMPILED_EXPORT` *AgdaName HaskellName* `#-}`.
- Подменяется имя функций (переименовывание типов данных не поддерживается), генерируемых MAlonzo.
- Функции требуют *Unit* на место параметров типов.

Что дальше

- Необходимо сделать экспортирования типов данных целиком: подменять код MAlonzo для типов данных и конструкторов.
- В Agda есть способ симулировать классы типов из Haskell с помощью **record** — можно попробовать генерировать соответствующие классы типов и их реализации.
- Существует множество расширений системы типов Haskell, которые позволяют в той или иной степени реализовывать зависимые типы — их использование позволит выражать больше типов Agda в Haskell.

data вместе с биекциями

newtype *List a*, **data** $T = In\ Int \mid Ch\ Char$

$trTtoT1 :: T \to T1$, $trT1toT :: T1 \to T$, $empty :: List\ a$

$add :: T \to List\ T \to List\ T$

$add = \lambda x\ xs.\ \texttt{unsafeCoerce}\ (d7\ (trTtoT1\ x)\ (\texttt{unsafeCoerce}\ xs))$

$head :: List\ a \to a$

$head = \lambda xs.\ \texttt{unsafeCoerce}\ (d8\ (\texttt{unsafeCoerce}\ xs))$

$test = head\ (add\ (In\ 3)\ empty)$

Для корректной работы *head* обязан был вызвать $trT1toT$ вместо `unsafeCoerce`.