Автоматизация доказательств в LiquidHaskell с помощью реляционных типов

Василенко Е.Г.

Под руководством
Prof. Niki Vazou, IMDEA Software Institute
Prof. Gilles Barthe, Max-Planck Institute

НИУ ВШЭ, Санкт-Петербург
11 июня 2021 г.

LiquidHaskell

Пусть τ – тип исходной системы, p – булевское выражение.
Синтаксис refinement типа:

$$\varphi ::= \{\, \tau \mid p \,\}$$

LiquidHaskell – refinement types для Haskell:

```
safeDiv :: Int -> {x:Int|x /= 0} -> Int
safeDiv = div

safeAt :: xs:[a] -> {i:Nat|i < len xs} -> a
safeAt = (!!)
```

SMT-солверы

Этапы проверки типов

- Тайпчекер сводит задачу проверки типов к решению набора предикатов
- Если SMT-солвер решает набор, то программа корректна

SMT-солвер решает формулы пропозициональной логики и её разрешимых расширений, таких как линейная арифметика:

```
foo :: Int -> Int
foo x = safeDiv x y
  where y = (-2) * (x `mod` 2) + 1
```

Подсказки для доказательств

Не любая корректная программа тайпчекается, могут требоваться подсказки:

```
-- Проверка на чётность
isEven :: n:Int -> {v:Bool|v <=> n mod 2 = 0}
isEven 0 = True
isEven n = if isEven n' then False else True
  where n' = if n < 0 then n + 1 else n - 1

-- Сравнимые числа имеют одинаковую чётность
theorem :: n:Int -> m:Int ->
  {()|n mod 2 = m mod 2 => isEven n = isEven m}
theorem n m = () -- наивная попытка не сработала
theorem n m = lemma n m -- ПОДСКАЗКА
```

Реляционные типы

Синтаксис реляционного типа:

$$\psi ::= \{\ \tau_1 \sim \tau_2 \mid p\ \}$$

Такой тип приписывается паре выражений:

$$e_1 \sim e_2$$

```
foo, bar :: Bool -> Int
foo a = if a then 0 else 2
bar b = if b then 1 else 3
```

```
foo ~ bar
  :: {a:_ -> i:_ ~ b:_ -> j:_ | a = b => i < j}
```

Существующие реляционные системы
I. RHOL[1]

Предоставляет правила сведения реляционных утверждений к утверждениям логики высшего порядка.

Примеры правил для типизации $\lambda \mathrm{x}_1.\mathrm{E}_1 \sim \lambda \mathrm{x}_2.\mathrm{E}_2$:

Синхронное	$\lambda \mathrm{x}_1.\mathrm{E}_1$	$\sim$	$\lambda \mathrm{x}_2.\mathrm{E}_2$
Асинхронные	$\lambda \mathrm{x}.\mathrm{E}$	$\sim$	$*$
Структурные	$*$	$\sim$	$*$

Не предоставляет алгоритм применения правил.

[1] A Relational Logic for Higher-Order Programs. / A. Aguirre [и др.] // Proc. ACM Program. Lang. New York, NY, USA, 2017. Авг. Т. 1, ICFP. DOI: 10.1145/3110265.

Существующие реляционные системы
II. RelCost[2]

Предоставляет алгоритм применения правил.

Предикаты p в типе $\{ \tau_1 \sim \tau_2 \mid p \}$ ограничены высказываниями о времени исполнения и количестве попарно различных элементов, например:

$$costDiff\,(pow(x_1, p_1), pow(x_2, p_2)) = bitDiff\,(p_1, p_2)$$

[2] Relational Cost Analysis. / E. Çiçek [и др.] // SIGPLAN Not. New York, NY, USA, 2017. Янв. Т. 52, № 1. С. 316—329. ISSN 0362-1340. DOI: 10.1145/3093333.3009858.

Цель и задачи

Цель

Расширить класс автоматически доказываемых свойств в LiquidHaskell за счёт добавления реляционных правил типизации.

Задачи

- Сформулировать алгоритмическую реляционную систему
- Реализовать её для Haskell
- Оценить степень достигнутой автоматизации

Формулировка алгоритмической системы
I. Модификация RHOL

Проверка, что условные выражения связаны отношением p:

$$\texttt{if x1 then a1 else b1} \overset{p}{\sim} \texttt{if x2 then a2 else b2}$$

В RHOL было бесконечно много способов сведения, в т.ч.

- Синхронным правилом к
  $\texttt{x1 <=> x2},\ \texttt{a1} \overset{p}{\sim} \texttt{a2},\ \texttt{b1} \overset{p}{\sim} \texttt{b2}$
- Структурным и двумя асинхронными к
  $\texttt{q} \vdash \texttt{a1} \overset{p}{\sim} \texttt{a2},\ \texttt{b1} \overset{p}{\sim} \texttt{b2},\ \texttt{a1} \overset{p}{\sim} \texttt{b2},\ \texttt{b1} \overset{p}{\sim} \texttt{a2}$

Теперь обобщённым синхронным правилом сводится к

$$\texttt{x1} \overset{q}{\sim} \texttt{x2},\ \texttt{a1} \overset{q \Rightarrow p}{\sim} \texttt{a2},\ \texttt{b1} \overset{q \Rightarrow p}{\sim} \texttt{b2},\ \texttt{a1} \overset{q \Rightarrow p}{\sim} \texttt{b2},\ \texttt{b1} \overset{q \Rightarrow p}{\sim} \texttt{a2}$$

Формулировка алгоритмической системы
II. Пример

Проверка, что условные выражения связаны отношением p:

```
if isEven n' then False else True
  ~ if isEven m' then False else True
```

```
p := n mod 2 = n mod 2 =>
    (if isEven n' then False else True <=>
        if isEven m' then False else True)
```

```
q := n' mod 2 = m' mod 2 =>
        (isEven n' <=> isEven m')
```

SMT-солверу под силу доказать `q => p`

Формулировка алгоритмической системы
III. Полученная система

Зафиксирован порядок применения правил:

1. ***Синхронные*** правила применяются для термов с одинаковой структурой
2. ***Асинхронные*** применяются когда нет подходящих синхронных
3. ***Не реляционные*** правила исходной системы типов применяются когда термы примитивны
4. ***Структурное*** правило применяется для сведения проверки типа к синтезу

Алгоритм гарантирует проверку строго большего числа свойств по сравнению с не реляционной системой LiquidHaskell.

Реализация для Haskell

GHC Core – синтаксис термов, для которых были реализованы правила:

```
data Expr b
  = Var   Id
  | App   (Expr b) (Arg b)
  | Lam   b (Expr b)                    -- λ-исчисление
  | Case  (Expr b) b Type [Alt b]       -- + типы данных
  | Type  Type                          -- + полиморфизм
  | Tick  (Tickish Id) (Expr b)         -- + комментарии
  | ...
```

- Релизованы реляционные правила для GHC Core
- Написан парсер реляционных сигнатур
- Вывод ошибок сопровождается локациями в файле

Оценка достигнутой автоматизации

Улучшения по сравнению с предшественниками:

- ***LiquidHaskell*** $\rightarrow$ ***меньше подсказок***
  Минимальность сравнений в сортировке вставками доказывается без подсказок:
  $$sorted(xs)\ \&\ len(xs) = len(ys) \Rightarrow cost(isort(xs)) \leqslant cost(isort(ys))$$
- ***RelCost*** $\rightarrow$ ***расширена логика***
  Может быть доказана алгоритмическая стабильность градиентного спуска:
  $$\|gd(xs_1, ws, \alpha, f) - gd(xs_2, ws, \alpha, f)\| \leqslant 2L\alpha \cdot diff(xs_1, xs_2)$$
- ***RHOL*** $\rightarrow$ ***предъявлен алгоритм проверки типа***

Результаты

Реализация: https://github.com/oquechy/liquidhaskell.

- Получена алгоритмическая реляционная система на основе RHOL
- Реализована для Haskell и протестирована
- Увеличен автоматизм LiquidHaskell за счёт меньшего количества необходимых подсказок при доказательстве реляционных свойств

Про реляционные типы для LiquidHaskell было рассказано на YOW! Lambda Jam 2021.

Производительность реляционной системы

Время компиляции тестовых примеров			
Название примера	Число строк кода	Время компиляции (сек)	Количество булевых формул на выходе
Монотонность фибоначчи	18	2.8	83
Быстрое возведение в степень	24	4.6	417
Map fusion	158	12.1	276
Сортировка вставками	165	6.6	154
Градиентный спуск	390	16.2	478

Рис. 12: Время компиляции тестовых примеров на процессоре 2,8 ГГц Dual-Core Intel Core i5 с оперативной памятью 8 Гб 1600 МГц DDR3

Реляционная сигнатура для map

```
{- reflect diff -}
{-@ diff :: xs:[Int] ->
                ys:{[Int]|len ys == len xs} -> Int @-}
diff :: [Int] -> [Int] -> Int
diff (x : xs) (y : ys) | x == y = diff xs ys
diff (x : xs) (y : ys) | x /= y = 1 + diff xs ys
diff _ _                        = 0

map :: (Int -> Int) -> [Int] -> [Int]
map _ [] = []
map f (x:xs) = f x : map f xs

{-@ map ~ map :: f1:(x1:_ -> _) -> xs1:_ -> _
              ~ f2:(x2:_ -> _) -> xs2:_ -> _
              | f1 == f2 => true => diff xs1 xs2 >=
                  diff (r1 f1 xs1) (r2 f2 xs2) @-}
```

Вывод типа

Вывод реляционного типа $\boxed{\Gamma \mid \Psi \vdash e_1 \sim e_2 \Rightarrow t_1 \sim t_2 \mid \phi}$

$$\frac{\Gamma \vdash e_1 \Rightarrow t_1 \quad \Gamma \vdash e_2 \Rightarrow t_2 \quad e_1, e_2 \text{ – переменные или константы}}{\Gamma \mid \Psi \vdash e_1 \sim e_2 \Rightarrow t_1 \sim t_2 \mid \mathsf{inst}(\Psi,\ e_1,\ e_2)}\ \textsc{RelSyn-Unary}$$

$$\frac{\begin{array}{c}\mathsf{inst}(\Psi,\ e_1\ x_1,\ e_2\ x_2) \doteq \forall v_1\ v_2.\ p \Rightarrow \phi \\ \Gamma \vdash e_1 \Leftarrow x_1 : s_1 \to t_1 \quad \Gamma \vdash e_2 \Leftarrow x_2 : s_2 \to t_2 \\ \Gamma \mid \Psi \vdash x_1 \sim x_2 \Rightarrow s_1 \sim s_2 \mid q\end{array}}{\Gamma \mid \Psi \vdash e_1\ x_1 \sim e_2\ x_2 \Rightarrow t_1 \sim t_2 \mid \forall v_1\ v_2.\ p \wedge q[\mathtt{r}_1 := v_1][\mathtt{r}_2 := v_2] \Rightarrow \phi}\ \textsc{RelSyn-App}$$

inst	:	$(\Psi \times E \times E) \to \Pi$
$\mathsf{inst}(\Psi,\ f_1\ x_1,\ f_2\ x_2)$ $\mid\ \langle f_1,\ f_2,\ \forall v_1\ v_2.\ p \Rightarrow \phi\rangle \in \Psi$	$\doteq$	$(p \Rightarrow \phi)[v_1 := x_1][v_2 := x_2]$
$\mathsf{inst}(\Psi,\ x_1,\ x_2)$ $\mid\ \langle x_1,\ x_2,\ p\rangle \in \Psi$	$\doteq$	p
$\mathsf{inst}(\Psi,\ e_1,\ e_2)$	$\doteq$	true

Проверка типа I

Relational Type Checking $\boxed{\Gamma \mid \Psi \vdash e \sim d \Leftarrow t \sim s \mid \phi}$

$$\frac{\Gamma \mid \Psi \vdash e \sim d \Rightarrow s_1 \sim s_2 \mid \psi \quad \Gamma; x : t_\Psi \vdash s_1 \prec: t_1 \quad \Gamma; x : t_\Psi \vdash s_2 \prec: t_2 \quad \Gamma \mid \Psi \vdash s_1 \sim s_2 \mid \psi \prec: \phi}{\Gamma \mid \Psi \vdash e \sim d \Leftarrow t_1 \sim t_2 \mid \phi}\textsc{RelChk-Syn}$$

$$\frac{\phi_s \doteq \phi[\mathtt{r}_1 := \mathtt{r}_1\ v_1][\mathtt{r}_2 := \mathtt{r}_2\ v_2] \quad \Gamma; x_1 : s_1; x_2 : s_2 \mid \Psi; p \vdash e_1 \sim e_1 \Leftarrow t_1[v_1 := x_1] \sim t_2[v_2 := x_2] \mid \phi}{\Gamma \mid \Psi \vdash \lambda x_1.\ e_1 \sim \lambda x_2.\ e_2 \Leftarrow v_1 : s_1 \to t_1 \sim v_1 : s_2 \to t_2 \mid \forall v_1\ v_2.\ p \Rightarrow \phi_s}\textsc{RelChk-Lam}$$

$$\frac{\phi_s \doteq \phi[\mathtt{r}_1 := \mathtt{r}_1\ x_1][\mathtt{r}_2 := \mathtt{r}_2\ x_2] \quad \Gamma_r \doteq \Gamma; f_1 : (x_1 : s_1 \to t_1); f_2 : (x_2 : s_2 \to t_2); x_1 : s_1; x_2 : s_2 \quad \Gamma_r \mid \Psi; p; \langle f_1,\ f_2,\ \forall x_1\ x_2.\ p \Rightarrow \phi_s \rangle \vdash e_1 \sim e_2 \Leftarrow t_1 \sim t_2 \mid \phi}{\Gamma \mid \Psi \vdash \mathtt{rec}\ f_1 = \lambda x_1.\ e_1 \sim \mathtt{rec}\ f_2 = \lambda x_2.\ e_2 \Leftarrow t_1 \sim t_2 \mid \forall x_1\ x_2.\ p \Rightarrow \phi_s}\textsc{RelChk-Rec}$$

Проверка типа II

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash d_1 \sim d_2 \Rightarrow s_1 \sim s_2 \mid \psi \\ \Gamma; x_1 : s_1; x_2 : s_2 \mid \Psi; \psi[\mathtt{r}_1 := x_1][\mathtt{r}_2 := x_2] \vdash e_1 \sim e_2 \Leftarrow t_1 \sim t_2 \mid \phi\end{array}}{\Gamma \mid \Psi \vdash \mathtt{let}\ x_1 = d_1\ \mathtt{in}\ e_1 \sim \mathtt{let}\ x_2 = d_2\ \mathtt{in}\ e_2 \Leftarrow t_1 \sim t_2 \mid \phi}\ \textsc{RelChk-Let}$$

$$\frac{\begin{array}{c}\psi_s \doteq \psi[\mathtt{r}_1 := x_1][\mathtt{r}_2 := x_2] \\ \Gamma \vdash s_1 \prec: \mathtt{bool} \quad \Gamma \vdash s_2 \prec: \mathtt{bool} \quad \Gamma \mid \Psi \vdash x_1 \sim x_2 \Rightarrow s_1 \sim s_2 \mid \psi \\ \Gamma \mid \Psi; \psi_s; x_1; x_2 \vdash d_1 \sim d_2 \Leftarrow t_1 \sim t_2 \mid \phi \\ \Gamma \mid \Psi; \psi_s; x_1; \neg x_2 \vdash d_1 \sim e_2 \Leftarrow t_1 \sim t_2 \mid \phi \\ \Gamma \mid \Psi; \psi_s; \neg x_1; \neg x_2 \vdash e_1 \sim e_2 \Leftarrow t_1 \sim t_2 \mid \phi \\ \Gamma \mid \Psi; \psi_s; \neg x_1; x_2 \vdash d_1 \sim e_2 \Leftarrow t_1 \sim t_2 \mid \phi\end{array}}{\Gamma \mid \Psi \vdash \mathtt{if}\ x_1\ \mathtt{then}\ d_1\ \mathtt{else}\ e_1 \sim \mathtt{if}\ x_2\ \mathtt{then}\ d_2\ \mathtt{else}\ e_2 \Leftarrow t_1 \sim t_2 \mid \phi}\ \textsc{RelChk-If}$$

Проверка типа III

$$\frac{\Gamma \vdash x_1 \Leftarrow \texttt{bool} \quad \Gamma \mid \Psi; x_1 \vdash d_1 \sim e_2 \Leftarrow t_1 \sim t_2 \mid \phi \qquad \Gamma \mid \Psi; \neg x_1 \vdash e_1 \sim e_2 \Leftarrow t_1 \sim t_2 \mid \phi}{\Gamma \mid \Psi \vdash \texttt{if}\ x_1\ \texttt{then}\ d_1\ \texttt{else}\ e_1 \sim e_2 \Leftarrow t_1 \sim t_2 \mid \phi}\ \textsc{RelChk-IF-L}$$

$$\frac{\Gamma \vdash x_2 \Leftarrow \texttt{bool} \quad \Gamma \mid \Psi; x_2 \vdash e_1 \sim d_2 \Leftarrow t_1 \sim t_2 \mid \phi \qquad \Gamma \mid \Psi; \neg x_2 \vdash e_1 \sim e_2 \Leftarrow t_1 \sim t_2 \mid \phi}{\Gamma \mid \Psi \vdash e_1 \sim \texttt{if}\ x_2\ \texttt{then}\ d_2\ \texttt{else}\ e_2 \Leftarrow t_1 \sim t_2 \mid \phi}\ \textsc{RelChk-IF-R}$$