4. *Вагнер В.В.* Теория отношений и алгебра частичных отображений // Теория полугрупп и её приложения. Саратов: Изд-во Сарат. ун-та, 1965. Вып. 1. С. 3–178.

УДК 519.682.1

А.А. Орел

ФАНТОМНЫЕ ТИПЫ ДАННЫХ НА ОСНОВЕ ОТНОШЕНИЯ ПРЕДПОРЯДКА

В работе [1] рассмотрена технология функционального программирования на языке Haskell, основанная на применении так называемых фантомных типов (phantom types), построенных на основе отношения эквивалентности между типами. Фантомные типы определяются [2] как типы данных, применяемые только для того, чтобы создавать другие типы данных, при этом их значения никогда не используются.

В данной статье предлагается при конструировании фантомных типов в ряде случаев использовать отношение предпорядка, которое проще реализуется средствами языка Haskell, чем отношение эквивалентности.

Для иллюстрации применения этого отношения рассмотрим задачу построения интерпретатора для простого языка выражений, предложенного в работе [3]. Выражения этого языка должны представлять операции сложения целых чисел, конъюнкции и условные операторы, а также удовлетворять требованию статического контроля типов. Следующий алгебраический тип данных `Exp` на языке Haskell дает представление синтаксиса этих выражений.

```
data Exp = LitInt  Int
         | LitBool Bool
         | Plus    Exp Exp
         | And     Exp Exp
         | If      Exp Exp Exp
```

Проблема, связанная с данным представлением, состоит в том, что формальные параметры выражений не содержат информацию о типе. Таким образом, статический контроль типов успешно пройдут выражения вида

```
Plus(LitInt 3)(LitBool False)
```

Однако в процессе выполнения программы будет обнаружена ошибка несоответствия типов.

Решить проблему можно с помощью расслоения грамматики выражений так, чтобы она отражала информацию об используемых типах данных. Однако расслоение грамматики влечет за собой функциональное расслоение программы, которое, в свою очередь, приводит к большому количеству почти повторяющегося кода, что недопустимо с технологической точки зрения. В работе [3] для решения данной проблемы предложено использовать в определении `Exp` фантомный типовой параметр `a`, назначение которого состоит в том, чтобы аннотировать тип `Exp` на уровне мета описаний. В этом случае модифицированное определение `Exp` принимает вид

```
data Exp a = Lit  a
           | Plus (Exp a) (Exp a)
           | And  (Exp a) (Exp a)
           | If   (Exp a) (Exp a) (Exp a)
```

Построенный тип позволяет решить проблему статического контроля типов выражений с помощью определения функций вида

```
plus :: Exp Int -> Exp Int -> Exp Int
plus = Plus
```

Однако не удается создать универсальный интерпретатор выражений, подобный следующему:

```
evalExp :: Exp a -> a
evalExp (Lit i)    = i
evalExp (Plus x y) = evalExp x + evalExp y
evalExp (And  x y) = evalExp x && evalExp y
evalExp (If c t e) = if(evalExp c) then (evalExp t)
                                   else (evalExp e)
```

При компиляции этих выражений система контроля типов Haskell сообщает о невозможности совместить тип переменной `a` с базовыми типами `Int` или `Bool`.

Решить возникшую проблему можно, включив в определение типа `Exp` выражение, требующее при выборе варианта конструктора данных использовать только определенное значение параметра типа. Таким выражением может быть конструктор типа реализующий отношение эквивалентности между типами. В данном случае, однако, можно использовать реализацию более простого отношения, а именно отношения предпорядка. Выбрав в качестве соответствующего конструктора типа стандартный конструктор функциональной зависимости (`->`), получим модифицированное определение типа `Exp`:

```
data Exp a = Lit             a
           | Plus (Int -> a) (Exp Int) (Exp Int)
           | And  (Bool -> a)(Exp Bool)(Exp Bool)
           | If              (Exp Bool)(Exp a)   (Exp a)
```

Поскольку отношение предпорядка не симметрично, следует определить правило выбора позиций типовых аргументов. Будем всегда выбирать в качестве первого аргумента более ограниченный тип данных, чем в качестве второго аргумента. Например, правильным будет выбор `Int -> a`, а не `a -> Int`. Заметим, что первая конструкция в отличие от второй представляет фантомный тип, поскольку сигнатуру `Int -> a` может иметь только всюду неопределенная функция. Свойство рефлексивности будем моделировать с помощью тождественной функции `id`, а свойство транзитивности с помощью операции композиции `(.)`. Необходимо определить также правило замены выражений, содержащих модифицированные конструкторы данных. Считая, что отношение предпорядка задает первый параметр в определении модифицированного конструктора данных `C`, и, обозначив реализующую его функцию как `p`, будем заменять выражение

```
f (C p1, ... pn) = e
```

на

```
f (C p, p1, ... pn) = p e
```

Данное правило полностью согласуется с правилом модификации выражений, использующих параметр, реализующий отношение эквивалентности (см. [1]).

Применяя рассмотренные процедуры для модификации функции сложения и для интерпретатора выражений типа `Exp`, получим

```
plus :: Exp Int -> Exp Int -> Exp Int
plus = Plus id
```

и

```
evalExp :: Exp a -> a
evalExp (Lit  i)     = i
evalExp (Plus f x y) = f $ evalExp x + evalExp y
evalExp (And  f x y) = f $ evalExp x && evalExp y
evalExp (If   c t e) = if (evalExp c) then (evalExp t)
                                      else (evalExp e)
```

В результате могут быть вычислены выражения, подобные следующему:

```
evalExp (plus (Lit (1 :: Int)) (Lit (2 :: Int)))
```

В то же время выражения вида

```
plus (Lit (1 :: Int)) (Lit True)
```

не пройдут статического контроля типов.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Hinze R.* Fun with phantom types // The Fun of Programming, Cornerstones in Computing. Palgrave, 2003. P. 245–262.

2. http://www.haskell.org/haskellwiki/Phantom_type

3. http://babel.ls.fi.upm.es/~pablo/Papers/Notes/GADTs.html

УДК 517.95

Д.В. Поплавский

О НЕЛИНЕЙНОЙ ЭВОЛЮЦИИ МАТРИЦЫ ВЕЙЛЯ СОГЛАСНО НАЧАЛЬНО-КРАЕВОЙ ЗАДАЧИ НА ПОЛУОСИ ДЛЯ ВЕКТОРНОГО МОДИФИЦИРОВАННОГО УРАВНЕНИЯ КДФ

Рассмотрим следующую задачу:

$$\begin{cases} u_t + 2u_x(3u^2 + v^2) + 4uvv_x + u_{xxx} = 0, \\ v_t + 2v_x(3v^2 + u^2) + 4vuu_x + v_{xxx} = 0, \quad x \geq 0,\ t \geq 0, \end{cases} \tag{1}$$

$$u(x,0) = u_0(x), \quad v(x,0) = v_0(x), \tag{2}$$

$$\begin{cases} u(0,t) = u_1(t), \quad u_x(0,t) = u_2(t), \quad u_{xx}(0,t) = u_3(t), \\ v(0,t) = v_1(t), \quad v_x(0,t) = v_2(t), \quad v_{xx}(0,t) = v_3(t). \end{cases} \tag{3}$$

где $u_k,\ v_k,\ k = \overline{0,3}$, – непрерывные комплекснозначные функции ($u_0(x)$, $v_0(x) \in L(0,\infty)$, $u_0(0) = u_1(0)$, $v_0(0) = v_1(0)$). Известно, что система (1) допускает эквивалентное представление нулевой кривизны [1], что дает возможность для нахождения ее решения применить метод обратной спектральной задачи [2], в котором нелинейная задача (1)–(3) сводится к обратной спектральной задаче на полуоси для дифференциальной системы с кратными корнями характеристического многочлена [3]. При этом соответствующей спектральной характеристикой выступает матрица Вейля. В [2] получены эволюционные уравнения на элементы матрицы Вейля и алгоритм решения задачи (1)–(3). То обстоятельство, что данные уравнения носят нелинейный характер, в определенной степени затрудняет реализацию метода обратной спектральной задачи именно в случае «полуоси». В настоящей статье приводится один