TYPICAL TEMPLATE VERIFICATION FOR LIST EDITING IN HASKELL LANGUAGE

Archvadze Natela[1], Nizharadze Mziana[2]
1-Ivane Javakhishvili Tbilisi State University,
2-Georgian Technical University

Summary

Programming language Haskell represents a functional language and it has all the characteristics that defines functional paradigm. In the functional paradigm of programming, methods used for building data structure gives ability to create simultaneously templates of typical functions to edit these structures. The classic module of language Haskell defines template of the function for editing lists. Our goal is following: the algorithm that we have used for Lisp functional programs to use for the Haskell typical template as well. This paper describes the method for structural induction that is used for verification of those Haskell programs that can present functions for editing lists.

Keywords: functional programming. Program verification. Structural induction method

1. Introduction

Programming language Haskell represents a functional language and it has all the characteristics that defines functional paradigm. We can list following major characteristics of functional languages: a) simplicity and shortness, b) strict standardization, c) module characteristics, d) functions as values and calculation object, e) absence of side effects and indeterminacy, f) lazy calculations. Typical tasks that are solved using functional programming methods include the tasks for dynamic structure descriptions and automatic construction of programs and verification for given structures. We will describe these structures using Haskell language and compare with Lisp language capabilities [1].

2. Automatic construction of major part of the program according to the data structure description

Given example is related to typical example for describing dynamic structure of data. The latter is solved not only by means of functional programming, but in other paradigms it is very complicated to construct the major part (unity of functions) of the program for editing certain data structures.

The methods used to build data structure in functional paradigm of programming gives ability to create typical function templates for editing these structures. Therefore, syntax oriented construction has capability to build automatically certain function descriptive foundation for constructed data types that will be used to edit corresponding data types. These foundations can be discussed as templates that will complete necessary functionality. The general construction of this template remains unchanged, only the content changes that depends on the function requirements defined by the user's target.

Only functional languages are characterized by building functional templates for editing data structure.

Let's consider a data structure as an example where the list of A type elements is given and let's represent it by using syntax oriented construction method. This method constructs data types by two simple operations – decartian product and unification. The structure "list" will be defined as follows [2]:

$$\begin{gathered} List(A)=NIL+(A\times List(A)) \\ prefix=constructor\ List(A) \\ head,tail=selector\ List(A) \\ isNil,\ isNonNil=predicate\ List(A) \\ nil,\ nonNil=parts\ List(A) \end{gathered} \qquad (1)$$

The syntax oriented construction method was introduced by British mathematician Charles Hoaro. He introduced the meta language that is able to describe data structure of any complexity, even the ones that are defined recursively using itself as well.

This method gives ability not only to edit dynamic structures of data, but to solve the task to create automatically data editing function templates as well. It is also used to solve another typical task of functional programming – to prove the function characteristics.

For the given definition formula (1) is necessary to build the typical function that will make the function List(A) work. The standard module - Prelude [2] of Haskell language defines function template to edit lists.

The typical function template that edits lists (it is assumed that the function takes one argument – the list at the beginning) has the following form:

$$f\,[\,] = g1\,[\,]$$
$$f\,(x : xs) = g2\,(g3\,x)\,(g4\,(f\,(g5\,xs))) \qquad (2)$$

In the formula given, *g1, g2, g3, g4, g5* functions represent those functions that are dependent on the goals of the task: *g1* is a function to edit empty list, *g2* is a function that unifies the results of function's head and tail editing, *g3* is a function that edits the head of the list, *g4 i*s a function that edits recursive call of the non-empty list tail, *g5* is a function that edits in advance the tail of the non-empty list before the recursive call.

3. Generalized form of programming language Lisp

[4] discusses the forms of abstract programs in programming language Lisp that enables to edit lists. One of the forms has the following form:

```
(DEFINE FUN(F F^0.L)
  (COND((MEMBER NIL L)a)                    ( 3 )
    (T(g(f(M F L))
  (APPLY 'FUN(CONS 'F(CONS 'F^0(M F^0 L)))))) ))
```

[5] describes the verification algorithm for functional languages. It shows how to prove by means of structural induction method.

Formula (3) resembles the definitions given by formula (1). Our goal is to use the algorithm that was used for Lisp functional program also for formula (1) in order to prove its verification. It is worth mentioning, that Lisp program edits lists.

4. Verification of typical template

Let us prove the verification of the equation described with formula (2) by using structural induction method.

Structural induction method is used for such type of recursive functions which has structures, not numbers, as arguments. The correctness of this type of program can be proved as follows:

1. Let's prove that the program works correctly for simple data (function arguments).

2. Let's prove that the program works correctly for more complicated data (function arguments) with the assumption that it works well for simpler data (function arguments).

For verification of formula (2) we will assume that the induction is conducted in accordance with list length; in other words, we assume that the function's argument is "simple" if it contains less elements than the "complicated" argument. Let's assume that *g5* function is equality. Analysis of the function shows that *f* function's recursive directing - *f(g5 xs)* on the right side is more simple because the argument is the result of *g5* function's interaction on *xs* list that has one less element than the original list *(x:xs)* as its first element was excluded.

The proof is as follows:

1. Let's prove that for any list with 0 elements, function *f* works correctly. Truly, *f[]=g1[]*. In general, *g1* function, that edits empty list is an equality function; in other words, it returns empty list as a result from the empty list: *f[]=[]*. That's what we wanted to show.

2. Let's prove that *f* function works well for lists with *N* elements (on upper level): *L=(x:xs)*. Then it will work correctly for all lists *L1=(x1 : xs1)* that has N+1 elements on upper level. The induction hypothesis is as follows:

$$f\,(x : xs) = g2\,(g3\,x)\,(g4\,(f\,(g5\,xs1)))$$

Let's write non-zero part of formula *(2)* for *L1* list that has *N+1* elements:

$$f\,(x1 : xs1) = g2\,(g3\,x)\,(g4\,(f\,(g5\,xs1))).$$

Here, *xs1* is a list with *N* elements as it is derived from *L1* list by excluding the first element. It is influenced by function *g5* that edits in advance *L1* tail before the recursive calling. Then *f (g5 xs1)* is called recursively. Often *g5* function is equality but it does not have a value because *xs1* is already a list with *N* elements for which the *f* function works correctly by induction hypothesis. That's what we wanted to prove.

5. Conclusion

The given research describes the structural induction method that is used for Haskell program verification and that can be used to derive function for editing lists.

References:

1. Archvadze N., Shetsiruli L.. Programing Language Lisp. Publisher "Universal" Tbilisi, 2008
2. Dushkin R.V. Functional Programming in Haskell Language. Moscow. Publish.DMK, 2007
3. Dushkin R.V. Guide of Haskell Language. Moscow DMK, 2008.
4. Archvadze N., Pkhovelishvili M., Shetsiruli L. Construction of generalized recursive forms for functional languages and their application in program verification tasks. El. Scientific Journal: "Computer Sciences and Telecommunications", http://gesj.internet-academy.org.ge , 2010, No. 3(26), pp. 133-141
5. Archvadze N., Pkhovelishvili M., Shetsiruli L., Nizharadze. A recursion forms and their verification by using the inductive methods. Computing and Computational Intelligence. Proceeding of the 3nd European Computing Conference (ECC'09), Tbilisi, 2009, pp.357-361.

http://www.wseas.org/conferences/2009/tbilisi/Program.pdf

http://www.wseas.us/e-library/conferences/2009/georgia/CCI/CCI58.pdf

სიების დამუშავების ტიპური შაბლონის ვერიფიკაცია Haskell ენისთვის

ნათელა არჩვაძე[1], მზიანა ნიჟარაძე[2]

1-ივ.ჯავახიშვილის სახ. თბილისის სახელმწიფო უნივერსიტეტი,

2-საქართველოს ტექნიკური უნივერსიტეტი,

რეზიუმე

Haskell დაპროგრამირების ფუნქციონალურ ენაა და ახასიათებს ყველა ის თვისება, რასაც ფუნქციონალური პარადიგმა განსაზღვრავს. პროგრამირების ფუნქციონალურ პარადიგმაში მონაცემთა სტრუქტურის ასაგებად გამოყენებული მეთოდიკები საშუალებას იძლევა პარალელურად შეიქმნას ტიპური ფუნქციის შაბლონები ამ სტრუქტურების დასამუშავებლად. Haskell ენის სტანდარტულ მოდულში განსაზღვრულია ფუნქციის შაბლონი სიების დასამუშავებლად. ჩვენი მიზანია, ვერიფიკაციკაციის ის ალგორითმი, რომელიც გამოყენებულ იყო Lisp-ის ფუნქციონალური პროგრამებისთვის, ასევე გამოვიყენოთ Haskell ენის ტიპური შაბლონისათვის. წინამდებარე ნაშრომში აღწერილია სტრუქტურული ინდუქციის მეთოდი, რომლის საშუალებითაც მოხდა Haskell -ის იმ პროგრამების ვერიფიკაცია, რომლებითაც შეიძლება სიების დასამუშავებელი ფუნქციების ასახვა.

ВЕРИФИКАЦИЯ ТИПИЧНОГО ШАБЛОНА ОБРАБОТКИ СПИСКОВ ЯЗЫКА HASKELL

Арчвадзе Н.[1], Нижарадзе М.[2]

1-Тбилисский Гос. Униветситет им. И.Джавахишвили,

2-Грузинский Технический Университет

Резюме

Язык программирования Haskell представляет собой функциональный язык, и его характеризуют все те свойства, которые определяет функциональная парадигма. В функциональной парадигме программирования примененные методы построения структур данных позволяют параллельно создавать шаблоны типичных функций для обработки этих структур. В стандартном модуле языка Haskell определен шаблон функции для обработки списков. Наша цель – примененный для функционального программирования Лиспа алгоритм верификации использовать также для типичного шаблона языка Haskell. В данной работе описан метод структурной индукции, с помощью которого проведена верификация тех программ Haskell, которыми можно представить функции обработки списков.