МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное образовательное
учреждение высшего образования
ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

Институт математики, механики и компьютерных наук
имени И. И. Воровича

Направление подготовки
01.03.02 — Прикладная математика
и информатика

ВЕРИФИКАЦИЯ СТРУКТУРЫ ДАННЫХ "ЗИППЕР"

Выпускная квалификационная работа
на степень бакалавра

Студента 4 курса
П. А. Грахова

Научный руководитель:
ст. п. В. Н. Брагилевский

Допущено к защите:
руководитель направления _____ В. С. Пилиди

Ростов-на-Дону
2019

Содержание

Введение 3

1. Структура данных Зиппер 4
   1.1. Зиппер в общем смысле 4
   1.2. Зиппер для обобщенного дерева 5
   1.3. Описание в Coq 5

2. Тестирование с использованием QuickChick 6
   2.1. Общие замечания 6
   2.2. Описание тестов 7

3. Формальная верификация с использованием Coq 7
   3.1. Формулировка желаемых свойств 7

Заключение 8

Список литературы 8

Приложение А. Coq-код доказательства 8

Введение

В процессе разработки программного обеспечения неизбежно возникает вопрос, удовлетворяет ли написанный программистом код некоторым заранее заданным спецификациям, согласно которым должны были быть разработаны алгоритмы? Этот процесс проверки корректности кода, или, иначе, верификации, может быть построен как с использованием так называемых unit-тестов, проверяющих код на большом количестве разнообразных входных данных, так и с помощью формального доказательства корректности интересующих нас участков кода. Несмотря на то, что второй вариант тестирования может формально гарантировать абсолютную корректность, в силу своей сложности он используется сравнительно редко, в промышленности куда более популярен первый способ. Однако в последнее время, с развитием систем автоматической верификации теорем, приобретает популярность использование комбинации этих двух методов, когда некоторые критически важные участки кода либо алгоритмы формулируются в виде теорем и верифицируются формально в подобных средах, в то время как корректность второстепенных частей "доказывается"посредством прогонки на тестах, возможно, в этих же средах, с помощью некоторых их расширений.

Одной из самых популярных систем автоматической верификации теорем является система Coq, разработанная в INRIA. В этой среде для тестирования написанных определений и теорем может использоваться плагин QuickChick, позволяющий сгенерировать набор случайных входных данных и проверить, является ли интересующая нас теорема либо свойство верными на этом наборе. Подобная проверка не гарантирует абсолютной правильности проверенных теорем (то есть то, что они будут верны для любых входных данных), но вероятность этого весьма высока, если был успешно пройден подобный unit-тест, что может вдохновить программиста либо исследователя на поиск настоящего доказательства.

В данной работе ставится цель исследовать одну структуру данных с позиций подобного стиля верификации. Более точно, исследуется вариант структуры данных "Зиппер"для обобщенных деревьев, описывается сама структура данных и набор функций для работы с ней, формулируются желаемые свойства, которым должна подобная структура данных удовлетворять, составляются тесты и доказывается корректность подобных свойств в нашей формулировке. Похожий процесс при разработке ПО выполняется часто, цель данной работы — показать, что Coq может являться удобной средой для подобных манипуляций, когда важен высокий уровень доверия к полученным алгоритмам.

1. Структура данных Зиппер

1.1. Зиппер в общем смысле

Строго говоря, Zipper/Зиппер — не есть структура данных в том смысле, который в него вкладывается исследователями и специалистами в области компьютерных наук, но конструкция, позволяющая на базе некоторой сложной структуры данных построить новую структуру, позволяющую при работе с данными обращаться лишь к той части исходных данных, которые требуют модификации либо обращения, оставляя оставшуюся часть нетронутой. Особенно это актуально при реализации алгоритмов на чисто функциональных языках программирования, где прямая работа с интересующей нас частью данных может быть сложной задачей.

Как пример, Зиппер для бинарного дерева может быть построен следующим образом: что-то там

1.2. Зиппер для обобщенного дерева

Зиппер был описан Gerard Huet [**Huet1997**] на примере обобщенного дерева, приведем здесь ту часть его описания, которая будет интересовать нас.

что-то там

1.3. Описание в Coq

Важно корректно описать изначальные структуры на внутреннем языке Coq , чтобы в дальнейшем избежать проблем с формальным доказательством интересующих нас свойств. Функциональная реализация алгоритмов и структур данных в этой работе базируется в основном на идеях Окасаки [**Okasaki1996**].

Тип, описывающий Зиппер, может быть рассмотрен как произведение типов “дерево” и “список контекстов”, где “контекст”, в свою очередь, тоже тип, описанный выше. Возможности стандартной библиотеки Coq позволяют описать Зиппер как модуль, который можно затем использовать повторно

```
Module TreeZipper (Import T: Typ).

Definition A := T.t.

Inductive Tree : Type :=
  | T_nil: Tree
  | T_tr: A → list Tree → Tree.

Inductive Context : Type :=
  | T_move: nat → A → list Tree → Context.

Definition ZipperTree := prod Tree (list Context).
...
```

что-то там про алгоритмы

2. Тестирование с использованием QuickChick

Как уже было сказано, есть возможность проводить тесты алгоритмов в самом Coq на случайных данных, используя QuickChick .

2.1. Общие замечания

Ограничения, которые мы накладываем на Зиппер (например, контекст не может быть вида $cntn, A, []$, не позволяют нам использовать стандартные механизмы QuickChick , автоматически выводящие генераторы для заданных типов. В связи с этим стоит детальнее рассмотреть то, как можно вручную описать генераторы для интересующих нас структур.

Самым важным (и сложным) является описание функции, возвращающей генератор случайного дерева высоты sz. Функция работает следующим образом: в случае $sz = 0$ возвращается лист (nil), иначе генерируется случайная величина типа A, затем рекурсивно строится список деревьев высоты $sz - 1$ и из полученных данных составляется исходное дерево.

```
Fixpoint genTreeSized (sz: nat) (g: G A) : G (Tree) :=
 match sz with
 | 0 ⇒ returnGen T_nil
 | S sz' ⇒
  freq [ (1, returnGen T_nil) ;
   (sz, bindGen g (fun x ⇒
       bindGen (@listOf (Tree) (genTreeSized sz' g)) (fun l ⇒
     returnGen (T_tr x l)))) ]
 end.
```

Проблема в том, что, так как у нас имеется лишь два варианта выбора ($sz = 0$ или $sz > 0$), то QuickChick будет генерировать лист в примерно 50% случаев, что не есть желаемый результат. В связи с этим используется стандартный комбинатор freq, позволяющий снизить вероятность появления nil до $\frac{1}{sz+1}$.

что-то про Context

2.2. Описание тестов

Опять же, в силу фундаментальных ограничений `QuickChick` , нельзя напрямую проверить сформулированные в Coq леммы на случайных данных. Требуется их переформулировать в терминах, доступных для проверки.

Описание пары тестов

3. Формальная верификация с использованием Coq

3.1. Формулировка желаемых свойств

Модифицируя или обозревая элементы дерева с использованием Зиппер, мы неявно предполагаем, что модификация будет затрагивать лишь ту часть дерева, которая не содержится в контексте (поскольку это есть определение Зиппер), и что некоторые свойства дерева, его “целостность”, не будут нарушаться при выполнении стандартных операций. Было бы неплохо убедиться в том, что наши ожидания действительно будут выполняться.

Пусть у нас имеется дерево T и некоторый предикат P над деревом. Тогда будем говорить, что P применимо к T, если P верно для T и всех его поддеревьев. Подобное обобщение хорошо тем, что позволяет при необходимости определить сколь угодно сложную функцию проверки “правильности” дерева. Это определение “применимости” может быть сформулировано в Coq следующим образом:

```
Inductive PropertyOverTree : (Tree → Prop) → Tree → Prop :=
  | P_nil: forall (P: Tree → Prop),
    P T_nil → PropertyOverTree P T_nil
```

```
| P_tree: forall (P: Tree → Prop) T,
  P T → (forall T', IsSubtreeOf T' T → P T') →
  PropertyOverTree P T.
```

Заключение

Список литературы

1. Cowgod's CHIP-8 Technical Reference. — URL: http://devernay.free.fr/hacks/chip8/C8TECH10.HTM#8xy6 (дата обр. 09.08.2018).
2. Revised Report on the Algorithmic Language ALGOL 68 / A. van Wijngaarden [и др.]. — Amsterdam : IFIP, 1973.
3. *Gunter C. A.* Semantics of Programming Languages. — The MIT Press, 1992. — С. 441. — ISBN 9780262570954.
4. *Plotkin G. D.* The Origins of Structural Operational Semantics. — 2004. — URL: http://homepages.inf.ed.ac.uk/gdp/publications/Origins_SOS.pdf (дата обр. 09.08.2018).
5. *Hennessy M.* Semantics of Programming Languages. — Wiley, 1990.