Verification of Functional Programs
Introduction

Andrés Sicard-Ramírez

EAFIT University

Semester 2014-1

Administrative Information

Course web page

`https://asr.github.io/courses/verification-of-functional-programs/`

Evaluation

Homework	30%
Presentation	30%
Final project	40%

Preliminaries

Notation

Sometimes we write $\forall x \alpha$ or $\forall x.\alpha$ instead of $\forall x(\alpha)$. In $\forall x.\alpha$, the scope of the quantifier extends as far as possible, e.g. $\forall x.\alpha \wedge \beta$ means $\forall x(\alpha \wedge \beta)$. Similar for $\exists$.

Source code

All code in the examples have been tested with Agda 2.6.0.1, Coq 8.9.1 and Isabelle 2019 (June 2019).

U$22.2 to U$59.5 billion!*

*Source: Tassey [2002].

Motivational Example

'Every functional programmer worth his salt knows how to reverse a list, debug the code, and prove that list reversal is its own inverse.' [Swierstra and Altenkirch 2007, p. 25]

Motivational Example

'Every functional programmer worth his salt knows how to reverse a list, debug the code, and prove that list reversal is its own inverse.' [Swierstra and Altenkirch 2007, p. 25]

Let's go (Haskell code) ...

```
(++) :: [a] → [a] → [a]
[]       ++ ys = ys
(x : xs) ++ ys = x : (xs ++ ys)

rev :: [a] → [a]
rev []       = []
rev (x : xs) = rev xs ++ [x]
```

To prove that the rev function is an involution.

Motivational Example

Example

Proving `rev (rev xs) = xs`.

Case `[]`.

```
rev (rev []) = rev []  (rev.1)
             = []      (rev.1)
```

Motivational Example

Example

Proving `rev (rev xs) = xs`.

Case `[]`.

```
rev (rev []) = rev []  (rev.1)
             = []      (rev.1)
```

Case `x:xs`.

```
rev (rev (x : xs)) = rev (rev xs ++ [x])  (rev.2)
                   = x : rev (rev xs)     (auxiliary thm.)
                   = x : xs               (IH)
```

Auxiliary theorem: `rev (ys ++ [x]) = x : rev ys`.

Motivational Example

Observation

The auxiliary theorem

```
rev (ys ++ [x]) = x : rev ys
```

is a generalisation of the required result

```
rev (rev xs ++ [x]) = x : rev (rev xs).
```

'A standard method of generalisation is to look for a sub-expression that appears on both sides of the equation and replace it by a variable.' [Bird and Wadler 1988, p. 124]

Observations from the Motivational Example

- Inductive data types $\Rightarrow$ Structural induction for reasoning about them.

Observations from the Motivational Example

- Inductive data types $\Rightarrow$ Structural induction for reasoning about them.
- Equational reasoning (process of replacing like for like using the substitutivity property and the equivalence properties of the equality) based on the referential transparency.

Observations from the Motivational Example

- Inductive data types $\Rightarrow$ Structural induction for reasoning about them.
- Equational reasoning (process of replacing like for like using the substitutivity property and the equivalence properties of the equality) based on the referential transparency.
- Generalisation of auxiliary theorem (including the inductive hypothesis) $\Rightarrow$ Proofs by induction are difficulty to automatise.

Questions from the Motivational Example

- What about ⊥?

$$\texttt{rev (rev } \bot\texttt{)} \stackrel{?}{=} \bot$$

Questions from the Motivational Example

- What about $\bot$?

$$\texttt{rev (rev } \bot\texttt{)} \stackrel{?}{=} \bot$$

  - Extend structural induction for handling $\bot$.

Questions from the Motivational Example

- What about $\bot$?

$$\texttt{rev (rev } \bot\texttt{)} \stackrel{?}{=} \bot$$

  - Extend structural induction for handling $\bot$.
  - Choose a programming logic to behaviours of programs on total and finite elements of data structures [Bove, Dybjer and Sicard-Ramírez 2009; Dybjer 1985].

Questions from the Motivational Example

- What about $\bot$?

$$\texttt{rev (rev } \bot\texttt{)} \stackrel{?}{=} \bot$$

  - Extend structural induction for handling $\bot$.
  - Choose a programming logic to behaviours of programs on total and finite elements of data structures [Bove, Dybjer and Sicard-Ramírez 2009; Dybjer 1985].
  - 'Morally' correct reasoning [Danielsson, J. Hughes, Jansson and Gibbons 2006].

Questions from the Motivational Example

- What about if xs is an infinite list?

$$\texttt{rev (rev xs)} \stackrel{?}{=} \texttt{xs}$$

Questions from the Motivational Example

- What about if xs is an infinite list?

$$\texttt{rev (rev xs)} \stackrel{?}{=} \texttt{xs}$$

  - Co-inductive data types $\Rightarrow$ Co-induction for reasoning about them [Gibbons and Hutton 2005].

Questions from the Motivational Example

- What about if xs is an infinite list?

$$\texttt{rev (rev xs)} \stackrel{?}{=} \texttt{xs}$$

  - Co-inductive data types $\Rightarrow$ Co-induction for reasoning about them [Gibbons and Hutton 2005].
  - Choose a programming logic to behaviours of programs on total (finite or potentially unbounded) elements of data structures [Bove, Dybjer and Sicard-Ramírez 2012; Dybjer and Sander 1989].

Questions from the Motivational Example

- The rev function is $O(n^2)$. Why are we reasoning about it?

```
GHCi> rev [1..10^7]
*** Exception: stack overflow
```

Questions from the Motivational Example

- The `rev` function is $O(n^2)$. Why are we reasoning about it?

```
GHCi> rev [1..10^7]
*** Exception: stack overflow
```

The `reverse` function in the `Data.List` library (GHC 7.8.2) is $O(n)$:

```
reverse l = rev l []
where
  rev []     a = a
  rev (x:xs) a = rev xs (x:a)
```

Questions from the Motivational Example

- In relation to the formal verification of `find` or `gcd` algorithms versus the verification of real programs:

  'They are differences in kind. Babysitting for a sleeping child for one hour does not scale up to raising a family of ten—the problems are essentially, fundamentally different.' [De Millo, Lipton and Perlis 1979, p. 278]

Verification of Functional Programs: Research Areas

Area	Research focuses on
Semantics definitions	Defining new concepts
Transformation rules	Programming transformations
Functional properties verification	The input and output correspondence of programs
Non-functional properties verification	Properties such as memory consumption or parallel performance

Source: Achten, van Eekelen, Koopam and Morazán [2010].

(Incomplete) Time Line

1949 Turing, Alan [1949]. Checking a Large Routine. In: Report of a Conference on High Speed Automatic Calculating Machines, pp. 67–69.

1957 Backus, J. W., Beeber, R. J., Best, S., Goldberg, R., Haibt, L. M., Herrick, H. L., Nelson, R. A., Sayre, D., Sheridan, P. B., Stern, H., Ziller, I., Hughes, R. A. and Nutt, R. [1957]. The FORTRAN Automatic Coding System. In: Proceedings Western Joint Computer Conference, pp. 188–198. (FORTRAN)

1958 McCarthy, John [1960]. Recursive Functions of Symbolic Expressions and their Computation by Machine, Part I. Communications of the ACM 3.4, pp. 184–195. DOI: 10.1145/367177.367199. (Lisp)

1960 Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Perlis, A. J., Rutishauser, H., Samelson, K., Vauquois, B., Wegstein, J. H., Wijngaarden, A. van and Woodger, M. [1960]. Report on the Algorithmic Language ALGOL 60. Communications of the ACM 3.5. Ed. by Naur, Peter, pp. 299–314. DOI: 10.1145/367236.367262. (ALGOL 60)

(Incomplete) Time Line

1961 McCarthy, John [1961]. A Basis for a Mathematical Theory of Computation. In: Proceedings Western Joint Computer Conference, pp. 225–238.

1966 Naur, Peter [1966]. Proof of Algorithms by General Snapshots. BIT 6.4, pp. 310–316.

1967 Floyd, Robert W. [1967]. Assigning Meanings to Programs. In: Mathematical Aspects of Computer Science. Ed. by Schwartz, Jacob T. Vol. 19. Proceedings of Symposia in Applied Mathematics, pp. 19–32.

1968 'In 1968, a NATO Conference on Software Engineering was held in Garmisch, Germany, ...For the first time, a consensus emerged that there really was a software crisis, that programming was not very well understood.' [Gries 1981, p. 296]

1969 Hoare, C. A. R. [1969]. An Axiomatic Basis for Computer Programming. Communications of the ACM 12.10, 576–580(3). DOI: 10.1145/363235.363259.

(Incomplete) Time Line

1971 Martin-Löf, Per [1971]. A Theory of Types. Tech. rep. University of Stockholm.

1973 Martin-Löf, Per [1975]. About Models for Intuitionistic Type Theories and the Notion of Definitional Equality. In: Proceedings of the Third Scandinavian Logic Symposium. Ed. by Kanger, Stig. Vol. 82. Studies in Logic and the Foundations of Mathematics. Elsevier, pp. 81–109.

1979 Martin-Löf, Per [1982]. Constructive Mathematics and Computer Programming. In: Logic, Methodology and Philosophy of Science VI (1979). Ed. by Cohen, L. J., Los, J., Pfeiffer, H. and Podewski, K.-P. Vol. 104. Studies in Logic and the Foundations of Mathematics. North-Holland Publishing Company, pp. 153–175. DOI: 10.1016/S0049-237X(09)70189-2.

1981 Nordström, Bengt [1981]. Programming in Constructive Set Theory: Some Examples. In: Proceedings of the 1981 Conference on Functional Programming Languages and Computer Architecture (FPCA 1981). ACM, pp. 141–154.

References

Achten, Peter, van Eekelen, Marko, Koopam, Pieter and Morazán, Marco T. (2010). Trends in *Trends in Functional Programming* 1999/2000 versus 2007/2008. Higher-Order Symbolic Computation 23.4, pp. 465–487. DOI: 10.1007/s10990-011-9074-z (cit. on p. 23).

Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Perlis, A. J., Rutishauser, H., Samelson, K., Vauquois, B., Wegstein, J. H., Wijngaarden, A. van and Woodger, M. (1960). Report on the Algorithmic Language ALGOL 60. Communications of the ACM 3.5. Ed. by Naur, Peter, pp. 299–314. DOI: 10.1145/367236.367262 (cit. on p. 24).

Backus, J. W., Beeber, R. J., Best, S., Goldberg, R., Haibt, L. M., Herrick, H. L., Nelson, R. A., Sayre, D., Sheridan, P. B., Stern, H., Ziller, I., Hughes, R. A. and Nutt, R. (1957). The FORTRAN Automatic Coding System. In: Proceedings Western Joint Computer Conference, pp. 188–198 (cit. on p. 24).

Bird, Richard and Wadler, Philip (1988). Introduction to Functional Programming. Series in Computer Sciences. Prentice Hall International (cit. on p. 9).

Bove, Ana, Dybjer, Peter and Sicard-Ramírez, Andrés (2009). Embedding a Logical Theory of Constructions in Agda. In: Proceedings of the 3rd Workshop on Programming Languages Meets Program Verification (PLPV 2009), pp. 59–66 (cit. on pp. 13–16).

References

Bove, Ana, Dybjer, Peter and Sicard-Ramírez, Andrés (2012). Combining Interactive and Automatic Reasoning in First Order Theories of Functional Programs. In: Foundations of Software Science and Computation Structures (FoSSaCS 2012). Ed. by Birkedal, Lars. Vol. 7213. Lecture Notes in Computer Science. Springer, pp. 104–118 (cit. on pp. 17–19).

Danielsson, Nils Anders, Hughes, John, Jansson, Patrik and Gibbons, Jeremy (2006). Fast and Loose Reasoning is Morally Correct. In: Proceedings of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2006), pp. 206–217. DOI: 10.1145/1111037.1111056 (cit. on pp. 13–16).

De Millo, Richard A., Lipton, Richard J. and Perlis, Alan J. (1979). Social Processes and Proofs of Theorems and Programs. Communications of the ACM 22.5, pp. 271–280. DOI: 10.1145/359104.359106 (cit. on p. 22).

Dybjer, Peter (1985). Program Verification in a Logical Theory of Constructions. In: Functional Programming Languages and Computer Architecture. Ed. by Jouannaud, Jean-Pierre. Vol. 201. Lecture Notes in Computer Science. Springer, pp. 334–349. DOI: 10.1007/3-540-15975-4_46 (cit. on pp. 13–16).

References

Dybjer, Peter and Sander, Herbert P. (1989). A Functional Programming Approach to the Specification and Verification of Concurrent Systems. Formal Aspects of Computing 1, pp. 303–319 (cit. on pp. 17–19).

Floyd, Robert W. (1967). Assigning Meanings to Programs. In: Mathematical Aspects of Computer Science. Ed. by Schwartz, Jacob T. Vol. 19. Proceedings of Symposia in Applied Mathematics, pp. 19–32 (cit. on p. 25).

Gibbons, Jeremy and Hutton, Graham (2005). Proof Methods for Corecursive Programs. Fundamenta Informaticae XX, pp. 1–14 (cit. on pp. 17–19).

Gries, David (1981). The Science of Programming. Monographs in Computer Science. Springer-Verlag. DOI: 10.1007/978-1-4612-5983-1 (cit. on p. 25).

Hoare, C. A. R. (1969). An Axiomatic Basis for Computer Programming. Communications of the ACM 12.10, 576–580(3). DOI: 10.1145/363235.363259 (cit. on p. 25).

Martin-Löf, Per (1971). A Theory of Types. Tech. rep. University of Stockholm (cit. on p. 26).

— (1975). About Models for Intuitionistic Type Theories and the Notion of Definitional Equality. In: Proceedings of the Third Scandinavian Logic Symposium. Ed. by Kanger, Stig. Vol. 82. Studies in Logic and the Foundations of Mathematics. Elsevier, pp. 81–109 (cit. on p. 26).

References

Martin-Löf, Per (1982). Constructive Mathematics and Computer Programming. In: Logic, Methodology and Philosophy of Science VI (1979). Ed. by Cohen, L. J., Los, J., Pfeiffer, H. and Podewski, K.-P. Vol. 104. Studies in Logic and the Foundations of Mathematics. North-Holland Publishing Company, pp. 153–175. DOI: 10.1016/S0049-237X(09)70189-2 (cit. on p. 26).

McCarthy, John (1960). Recursive Functions of Symbolic Expressions and their Computation by Machine, Part I. Communications of the ACM 3.4, pp. 184–195. DOI: 10.1145/367177.367199 (cit. on p. 24).

— (1961). A Basis for a Mathematical Theory of Computation. In: Proceedings Western Joint Computer Conference, pp. 225–238 (cit. on p. 25).

Naur, Peter (1966). Proof of Algorithms by General Snapshots. BIT 6.4, pp. 310–316 (cit. on p. 25).

Nordström, Bengt (1981). Programming in Constructive Set Theory: Some Examples. In: Proceedings of the 1981 Conference on Functional Programming Languages and Computer Architecture (FPCA 1981). ACM, pp. 141–154 (cit. on p. 26).

Swierstra, Wouter and Altenkirch, Thorsten (2007). Beauty in the Beast. A Functional Semantics for the Awkward Squad. In: Proceedings of the ACM SIGPLAN 2007 Haskell Workshop, pp. 25–36 (cit. on pp. 5, 6).

References

Tassey, Gregory (2002). The Economic Impacts of Inadequate Infrastructure for Software Testing. Tech. rep. National Institute of Standards and Technology. US Department of Commerce (cit. on p. 4).

Turing, Alan (1949). Checking a Large Routine. In: Report of a Conference on High Speed Automatic Calculating Machines, pp. 67–69 (cit. on p. 24).