Proving Properties of Programs by Structural Induction: Summary

Oliver Vecchini

17th February 2019

Abstract

The following is a summary of the contents, context, and consequences of a seminal paper in theoretical computer science, namely '*Proving Properties of Programs by Structural Induction*', written by R. M. Burstall and published in 1969 [1]. It is submitted as a part of the assessed coursework component of the COMP0007 'Directed Reading' module at UCL.

1 Background

As expressed in the previous paper in this series, program correctness has long been a focus in both academic and industrial programming, with the costs of program failure rising into the millions even by the 1960s (as in the case of the destructive abortion of the *Mariner 1* spacecraft in 1962). Formal program verification has been of academic interest since the work of J. von Neumann & H. H. Goldstine in 1947 and A. M. Turing in 1949, in examining correctness of flowchart-based imperative programs; however, while progress did continue for this paradigm (in particular under P. Naur and R. W. Floyd), focus was also diverted towards correctness in the context of functional programming [2].

The concepts found in functional programming languages are mostly derived from those of λ-calculus, introduced by A. Church in the 1930s [3], and combinatory logic, introduced by M. Schönfinkel in 1924 [4], two near-equivalent notations which model computation in terms of the binding and evaluation of functions. Focus was diverted from explicit mutability of data (e.g. assignment) and control flow 'jumps', and towards modelling each program as the evaluation of a single function. Programming languages were beginning to express these concepts in the decade preceding Burstall's paper, with languages that treat functions as first-class objects including IPL [5], ISWIM (used in Burstall's own paper [1]), and LISP [6]. The last of these was originally designed by J. McCarthy, who continued on to make developments in the mathematical theory of computation; one such contribution was *recursion induction*, a system for proving equivalence of two functions g and h by showing that, if there exists some function f defined recursively by a particular equation E, and g and h both satisfy E by replacing f, then g is equivalent to h (this is illustrated in the domains of both generalised computable functions and LISP). McCarthy also defines *S-expressions*, akin to inductively defined data structures, where each S-expression is either an atom or an expression of S-expressions; McCarthy demonstrates application of recursion induction to structures of this form as well [7].

While Burstall's paper was another step in the direction of program verification in functional programming, it should be noted that the concepts expressed had appeared in literature before. H. B. Curry and R. Feys demonstrated the use of structural induction for proofs concerning functions in combinatory logic in 1958 [8], and McCarthy and J. Painter used structural induction (as an unnamed induction principle) to prove correctness of a compiler in 1967 [9].

2 Contents of the article

Burstall first discusses the variety of induction principles available for program verification, and notes that structural induction is simply a reformulated special case of recursion induction; he notes that reasoning about programs with the former is generally easier than the latter (an opinion that appears to be generally accepted [10]). Burstall briefly describes the general 'roadmap' of the introduction to structural induction (specifically aiming to be a middle-ground between an easily automatable tool and an easily understood principle), as well as noting the programming language used to illustrate proofs (ISWIM).

Burstall then describes the notion of inductively-defined data structures (McCarthy's s-expressions are absent), where each *object* of some type is either an *atom* or a *structure*. An atom can be one of any 'base objects', and a structure is an object returned by a *constructor* function evaluated with some number of objects as arguments; it is said that these objects are the *components* of the structure. Also defined are the *destructor* function, returning the components of a structure, and the *predicate* function, returning the truth value of whether a given object was made with a given constructor. Note that these definitions match the description of a general recursively-defined algebraic data type.

Burstall then prepares to state the principle of structural induction. He defines the *constituency* relation, where if an object is a constituent of another object, then either the former is equal to the latter or the former is a constituent of one of the components of the latter (i.e. the former is involved in the construction of the latter); these objects also satisfy the relation of *proper constituency* if they are not equal. The principle of structural induction is thus stated: for a set of objects, if, for all objects, the fact that all proper constituents of an object satisfy a property implies that the object itself satisfies the property, then all objects in the set satisfy that property. This definition makes it clear as to why proofs using structural induction often match the structure of the functions they concern; similarly to how structures are 'built up' by constructors with objects and ultimately atoms, the fact that a structure satisfies a property is 'built up' from the fact that its proper constituents (i.e. the objects and ultimately atoms used in constructing it) satisfy the property.

Of note is that Burstall arrives the principle of structural induction as a specialization of Noetherian induction using the set of structures and the relation of proper constituency. Only well-founded (i.e. all non-empty subsets have a minimal element) partial ordering on a set is required for Noetherian induction, as opposed to the well-ordering (and thus total ordering) required by regular induction. This is required as proper constituency is *not* totally-ordered (for instance, two atoms are not proper constituents of one another).

Burstall proceeds to introduce the ISWIM language, and adapt its syntax to make written proofs even more transparent and clear; however, it should be noted that this is only syntactic sugar and strictly is unrelated to the actual proof mechanism of structural induction. He gives rules α and β for variable name substitution to convert between equivalent program expressions, for variable name changing and variable value forward-substitution respectively. He groups several ISWIM operators into the single operator *cons*, which he changes to an infix *::* operator; he further advises this procedure for any general constructor operation. Burstall compares McCarthy's axioms for proving LISP programs using recursion induction [7] with his own substitution rules, noting both their generality across constructor functions (examples are given in terms of list operations and the *cons* operator, but are not specific to these) and their flexibility for adding rules later. The pattern-matching idiom is proposed here (although not named as such) to avoid redundancy in writing constructor/destructor cases, again as syntactic sugar to make proof-writing less tedious (although the formal rules for selecting the appropriate case are omitted). Also proposed is the notion of the constructor type signature, consisting of the types of the objects taken as arguments by the constructor.

To illustrate the use of structural induction and the adapted syntax of ISWIM, Burstall proves statements regarding a *fold*-style expression, a tree-sorting function, and an expression-to-machine-code compiler function. In each instance, he proves the property for each function argument *case*, akin to the proving of the property for each of atoms and structures. The compiler example in particular shows that expressions can be evaluated by working with a stack, in particular that executing a compiled expression is equivalent to loading the value of that expression onto the stack. Using other auxiliary functions defined earlier, this is proved completely using structural induction.

3 Legacy

The influences of Burstall's paper are numerous. Firstly, the structural induction principle has had considerable impact, with research continuing in its automation conducted by R. Aubin [11], Brotz [12], and R. S. Boyer & J. S. Moore. The practical impact of the paper is also evident; the work of Boyer & Moore in particular lead to the creation of the Nqthm/ACL2 automated theorem provers [13], and examples of others include Zeno and Hipspec [14]. Together with McCarthy's work, Burstall's paper motivated the further general research of functional program verification, with contributions by McCarthy, Z. Manna, and A. Pneuli appearing in the 1970s [15] [16], to name a few. Functional programming is now the dominant paradigm of programming verification, with common tools such as Coq and Isabelle serving as functional programming languages [17]. Type theory has also been impacted, with the notion of coinductive types stemming from structural induction [18]. Finally, the augmented notation of Burstall's paper has been widely adopted, with the function type signature convention, the *::* notation for *cons*, and the pattern-matching idiom finding their ways into popular languages [19].

4 In Review

Overall, the paper is extremely clear in its description of the concepts it presents. Despite taking multiple concepts from earlier work [7] [8], everything is presented without requiring prior knowledge in the field, gradually and thoroughly enough that one unacquainted with the topic can easily approach the paper. Despite a few inconsistencies of terminology (using 'constituent' to refer to 'proper constituents' such as in the phrase 'no structure is identical to a constituent of itself', and mixing up of the phrases 'object' and 'structure' when strictly a structure is just one possible form of an object), Burstall clearly (and explicitly) recognizes the benefit of clarity in order to ensure future development of the subject. He attributes the previous work of others clearly, while also highlighting areas in which progress could continue. Burstall's paper is an excellent groundwork for further research into both the specific topic of structural induction and the general area of program verification.

References

[1] Rod M Burstall. Proving properties of programs by structural induction. *The Computer Journal*, 12(1):41–48, 1969.

[2] Oliver Vecchini. An axiomatic basis for computer programming: Summary.

[3] Alonzo Church. An unsolvable problem of elementary number theory. *American journal of mathematics*, 58(2):345–363, 1936.

[4] Moses Schönfinkel. Über die bausteine der mathematischen logik. *Mathematische annalen*, 92(3):305–316, 1924.

[5] Herbert A Simon. *Models of my life*. MIT press, 1996.

[6] John McCarthy. Recursive functions of symbolic expressions and their computation by machine. 1959.

[7] John McCarthy. A basis for a mathematical theory of computation. In *Studies in Logic and the Foundations of Mathematics*, volume 35, pages 33–70. Elsevier, 1963.

[8] HB Curry, R Feys, and W Craig. Combinatory logic, vol. 1 north-holland pub. *Co., Amsterdam*, 1958.

[9] John McCarthy and James Painter. Correctness of a compiler for arithmetic expressions. *Mathematical aspects of computer science*, 1, 1967.

[10] JM Brady. Hints on proofs by recursion induction. *The Computer Journal*, 20, 1977.

[11] Raymond Aubin. Mechanizing structural induction part i: Formal system. *Theoretical Computer Science*, 9(3):329–345, 1979.

[12] Douglas K Brotz. Embedding heuristic problem solving methods in a mechanical theorem prover. Technical report, 1974.

[13] Introduction to acl2. `http://www.cs.utexas.edu/users/moore/acl2/v2-7/INTRODUCTION.html`.

[14] Koen Claessen, Moa Johansson, Dan Rosén, and Nicholas Smallbone. Hipspec: Automating inductive proofs of program properties. In *ATx/WInG@ IJCAR*, 2012.

[15] Zohar Manna and John McCarthy. Properties of programs and partial function logic. Technical report, STANFORD UNIV CALIF DEPT OF COMPUTER SCIENCE, 1969.

[16] Zohar Manna and Amir Pnueli. Formalization of properties of functional programs. *Journal of the ACM (JACM)*, 17(3):555–569, 1970.

[17] Artem Yushkovskiy and Stavros Tripakis. Comparison of two theorem provers: Isabelle/hol and coq. *arXiv preprint arXiv:1808.09701*, 2018.

[18] Eduardo Giménez. Structural recursive definitions in type theory. In *International Colloquium on Automata, Languages, and Programming*, pages 397–408. Springer, 1998.

[19] Jeffrey D Ullman. *Elements of ML programming*, volume 2. Prentice Hall Englewood Cliffs, NJ, 1994.