

Automatic Program Optimization via the Transformation of Nuprl Synthesis Proofs; *A summary for the application to the 1988 Alvey Conference.**

Peter Madden

August 13, 1992

We investigate program optimization by the transformation of Nuprl program synthesis proofs.¹ Synthesis proofs which yield inefficient programs are transformed into *analogous* proofs which yield more efficient programs. The idea that transformation of synthesis proof induction schemas is the key to program optimization is pursued.

There exists a few systems which preport to be *program transformation systems* (cf. Burstall and Darlington [1977] and Manna and Waldinger [1979]). These systems generally apply re-write rules to sets of expressions which comprise the recursive equations of a program. Transformation consists in symbolically evaluating the equations by applying sequences of *foldings* and *unfoldings* together with *instantiations*.

In more recent work Darlington has manually derived a cross-section of sorting algorithms with an eye to eventually being able to automatically generate the more efficient versions from the remainder ((cf. Darlington [1978] and Darlington [1981]). There are many steps in their derivations which clearly involve human ingenuity, or, at least, for which no mechanical explanation is provided. Furthermore, as with their more general program transformation model, Darlington and Burstall are not concerned with the *transformation of synthesis proofs*, but rather with the transformation of program structure itself. Unlike Burstall and Darlington, who apply transformation rules directly to the program's specification, we wish to apply transformation rules directly to complete *Nuprl refinement proofs* in order to map alternative proofs from which more efficient programs can be extracted.²

This means that their transformation mechanism cannot directly make use of the extensive knowledge of the properties of constructive proofs, in particular, the role mathematical induction plays in introducing recursive calls into the function definitions extracted from the synthesis proofs [Bundy, Stevens, Boyer-Moore].

Within Nuprl, a program specification, comprising the desired input-output relationship, is represented by a statement in *constructive* logic. By finding a constructive proof of this statement we can routinely extract an algorithm from the proof which satisfies the desired input-output relation. Hence the techniques of theorem proving can be brought to bear on the program synthesis and transformation domains.

The logic of *both* proof and the program extracted therefrom is based on Martin-Löf type theory thus closely relating proofs to programs. This means that synthesis and optimization can be treated uniformly, or, in other words, for each transformation operation performed on a synthesis proof there will be a one-to-one corresponding transformation in the target program language.

In order to synthesise *recursive* programs we must employ mathematical induction in the construction of the Nuprl proofs from which the programs are extracted. Furthermore, the type of recursion employed is dependant on the type of induction employed in the corresponding Nuprl proof. So, the way in which an algorithm recurses on its input can be *controlled* by the way in which mathematical induction is

*This research is supported by S.E.R.C. grant GR/D/44270, and a studentship to the author. This work is a summary of [Madden 88] and has benefited from discussions with my supervisor Alan Bundy.

¹Nuprl is version "nu" of the Edinburgh *Proof Refinement Logic* system.

²There may, of course, be considerable patching of the target plan required.

employed in the algorithms synthesis. The latter amounts to the type of induction schema employed in the proof and what, amongst the possible many alternatives, is the correct variable(s) to induce on.

The clues that the recursive argument position and structure of a function give us as to the best induction schema and induction variable to use have been incorporated, in the form of heuristics, into the Nuprl system. Boyer and Moore have done extensive work on heuristics for inductive proofs (cf. Boyer and Moore [1973, 1979]). The relationships between induction and recursion which they established have been generalized such that most recursive structures have a corresponding induction schema which can be employed to synthesise programs exhibiting the desired recursive behaviour [Stevens87]. It is such knowledge that, due to the proof-program language uniformity, can be exploited by a Nuprl transformation system.

We believe that the key to program optimization lies in the transformation of the induction schemas employed in the respective synthesis proofs. For any particular recursive function, f , we may construct any number of inductive proofs, each employing a certain induction schema and/or induction variable(s), and each yielding a program which satisfies the input-output relation corresponding to f (ie. satisfies the same specification). Hence program optimization does not lie merely in the transformation of induction schemas but in the indirect transformation of recursive structures such that the *end result* of a transformation is a program which exhibits more efficient recursive behaviour

To sum up so far, Boyer and Moore type heuristics are used to guide the employment of Nuprl tactics, *refinement rules*, in the synthesis of recursive algorithms. We also foresee them having a role to play in the transformation and optimization of Nuprl proofs. Inductive proofs are transformed by transforming the induction schemas employed within; exactly what is transformed into what being determined by the sorts of considerations inherent in the Boyer-Moore heuristics *and* by further domain specific investigation into the types of changes that can be induced in the recursive structure of the extract term by altering the nature of the Nuprl constructive proof.

In order to develop a transformation system which contains *proof transformation tactics* we require some means of *high level comparison* between the proofs such that similarities in proof strategy can be exploited. That is, we wish to draw analogies between proofs at a level of abstraction which, much as in the spirit of Plummer's Gazing technique, ignores superfluous details but records important (or useful) similarities (and, indeed, differences): Although proofs may share similar (sub)goals they may also diverge in the *syntax* used to the extent that any underlying(or rather overlying) similarities in strategy (or tactics) is rendered somewhat opaque. What we need to have access to is some sort of "skeleton" structure of the proofs (i.e a *proof plan* of sorts).

Work is currently underway on the formalization of the types of meta-level reasoning inherent in the Boyer and Moore heuristics for guiding inductive proofs: Proof-plans which comprise the reasoning/heuristics employed in the construction of an inductive proof are represented by the formulae of a meta-logic [Bundy87].³

It is such a meta-logic, and the proof-plans represented therein, which will form the target of our transformations. In this sense we will be transforming the *reasoning* behind inductive proofs in order to come up with proofs which yield more efficient recursive structures. So, in other words, we require tactics for manipulating meta-level proof plan formulae.

If we can incorporate meta-level knowledge into formal proof plans and use these to guide the construction of inductive proofs then we should be able to formalize *transformation tactics* which take (sub)proof-plans as arguments and return (sub)proof-plans which can guide the synthesis of more efficient programs.⁴ In this way we will be exploiting past proof-plans in order to construct new proof-plans.

At present, we have synthesized all the main sorting algorithms, are studying the relations between the inductive proofs, investigating the possibility of re-synthesizing all the algorithms from a common specification (similar in rationale to Darlington's approach) and investigating the representation of proof-

³This is just one particular aspect of on going research into the use of meta-level inference as a tool for controlling implementations of mathematical activities (cf. Progress report of the Mathematical Reasoning Group in the Edinburgh department of AI, submitted to Alvey conference 1988).

⁴There is already *provision* in Nuprl for "transformation tactics" which take object-level proofs as arguments so, given a suitable meta-logic, the extension to transformation tactics which take (sub)proof-plans as arguments should be possible.

plans as the specifications of LCF-like tactics.

There is clearly scope for extending this methodology beyond the optimization of several sorting algorithm's domain: We intend to investigate the analogical mapping of proof-plans which yield completely new programs, as opposed to more efficient programs which compute the same function. In a similar spirit to Carbonell's *Analogical Transformation model*, the transformation tactics would be applied to a past successful proof-plan, the *source*, in order to map a proof plan, the *target*, which guides the construction of a new proof yielding a new program. Indeed, the optimization of several sorting algorithms is a sub-domain of this wider application of transformative analogy (cf. Carbonell [1983, 1985] and Madden [1987]).

References

[Boyer and Moore 73]: R.S Boyer and J.S. Moore. Proving theorems about lisp functions. In N. Nilson, ed., *Proceedings of the third IJCAI*, pp. 486-493, International Joint Conference on Artificial Intelligence, August 1973.

[Carbonell 85]: J.G. Carbonell. *Learning by analogy: forming and generalizing plans from past experience* in Michalski, R.S., Carbonell, J.G., Mitchell, T.M., (eds): *Machine Learning: An Artificial Intelligence Approach* pp.137-160 (Tioga Press 1983).

[Darlington 78]: J. Darlington. *A Synthesis of Several Sorting Algorithms*. Acta Informatica **11** (1978).

[Darlington 81]: J. Darlington. *An Experimental Program Transformation and Synthesis System*. Artificial Intelligence **16**, pp. 1-46 (1981).

[Madden 87]: P. Madden. *The Application of Analogy to Program Optimization*. Discussion paper 51, DAI, September 1987.

[Madden 88]: P. Madden. *Automatic Program Optimization via the Transformation of Nuprl Synthesis Proofs*. Discussion paper 63, DAI, January 1988.

[Manna and Waldinger 78]. Z. Manna and R. Waldinger. *The Logic of Computer Programming*. IEEE Trans. Software Eng. **SE-4**(3), 199-229 (1978).

[Stevens 87]: A.Stevens. *A Rational Reconstruction of Boyer-Moore Recursion Analysis*. Research Paper (forthcoming) submitted to CADE-9 379, Dept. of Artificial Intelligence, Edinburgh, 1987.

Address for correspondence

Peter Madden,
Mathematical Reasoning Group,
Dept. of Artificial Intelligence,
University of Edinburgh,
80 South Bridge, Edinburgh,
Scotland.