A Foundational Proof Carrying Code For 'Elimination' of an Isolated Numerical Value

Abstract— **A constructive approach to real types pervades the environment without introducing logical errors or runtime exceptions due, for instance, to system overflow. For conversion of IEEE 754 formats into these alternative types, a higher order arithmetic logic is discussed which injects a subtype equivalent to the mlTON language intinf, and the general idea is presented in a synopsis of a single line of INRIA's FLOCQ library. The exported modular types for the CM build tree for sml NJ are mentioned.**

Keywords— **Proof carrying code, double double word, proof assistant**

INTRODUCTION

In a foundational proof carrying code system what is the pervasive structure of the logic? Should it be looked at as an inductive logic system? And if so, why did Alonzo Church insist in his famous thesis on a recursive lambda calculus instead of just the infinite Turing model?

Regardless, most real time proof carrying codes are build around inductive logic. These verifiers need to implement exact arithmetic, and cannot optimally implement floating point in their intermediate representations. This constructive real arithmetic, when utilized in these foundational units of the logical structure, does not come without challenges and hence is traditionally left out of these environment's primitive constructive kinds and their data type elements. One way that traditional proof assistants choose to isolate these issues is by demoting them, or refusing these types and their side effects equality status, meaning only the basis environment can define them through a specification, and not at an implementation level. Subsequently, the application is left with the task to reinterpret the values of these extended precision types as real values in the local use environment.

In a recursive lambda logic, the iterative construction of types proceeds in a manner best exemplified by the factorial function. One element, the predicate constraint, expands while the other component of the term contracts, or descends until the last iteration is reached. So the value of zero, or sometimes simply the empty set, becomes the limit value. As this again poses an issue for systems defined by rounding in addition to absolute error, double double word types have been introduced by Jean Michel-Muller and an INRIA team where the composed unit values become a sum term, as opposed to an atomic real double type.

Such constructions of numerical constraints on the system also inductively defines higher level structures in abstract syntax layers, and these constraints can be implemented in extensional type systems elaborated by functional languages built axiomatically, i.e. from embedded proof-assistants. In such environments, the primitive type which pervades and is contained in almost all set operations is the integer. And the subtype in a dialect of Standard ML that is constructed from the integer and is also typically defined as a derived type of the core is the intinf subtype. When unchecked by the proper assert, recursion on integer types incur a risk of non termination on the term that is contained in Z. The natural course of logic then is to implement induction, in the traditional mathematical sense. And while integers are not nearly prone to the number of issues of real types, for the obvious reasons stated above, they also do not permit the extension of the base layer logic of the system to higher orders, for rarely is a recursion on multiple integer terms encountered in them, as much as in their real counterparts.

The main contribution of this idea for asserting types is the promulgation of a goal implementing a logical subtype to a tactical one, and then subsequently to one included into a hierachy of definitional modules. The goal, if lifted to enough levels of the hierarchical logic, will implement a reversed sequential set, in the same constructive manner as a list is reversed in a primitive operation of the Haskell functional programming language. The containing module Z is constructively bisected into the traditional sub intervals, isolating the limit point of the set. Subsequently a non-computable subtype value analogous to intinf for the real-types is defined in the Red-PRL prover lexeme, and the introduction into the universe of a dialect of Standard ML will be tested for the development of the subtype. As an example, and merely an example, of the simple form this proposition injects into the verification condition generator, a small definition from Jean-Michel Muller's team's FLOCQ library is shown with an extra propositional term ‘injected’ into the functional logic. In such a hypothetical insertion the tactics at the top level will reflect this mathematical goal, but in a flattened non- cubical type-theory.

```
Definition SF_inf x :=
  match x with
  | S754_finite true (#float64_ldshiftexp).
```

FIGURE 1. IN THE RED-PRL ELIM-EQUAL.PRL AS WELL AS THE REFINER-TYPES.SML TACTIC DECLARATION STATEMENTS, THE SYNTHESIZED INTINF TYPE IS TRIVIALLY VERIFIED FOR THE LOGIC, AND THIS WILL MINIMIZE THE NECESSARY PROPOSITIONAL VALUES FOR THE EXPORTED MODULE STRUCTURE. (JUST A NOTE, THE TERMS IN THE DOUBLE DOUBLE TYPE, WHEN SCALED TO FIT THE CHOSEN FORMAT, ARE NECESSARILY BOUND BY THE TERM OF A NEGATIVE INTEGER POWER OF 2 MULTIPLIED BY A CONSTANT. [3])

REFERENCES

[1] Andrew W. Appel and David McAllester. 2001 "An Indexed Model of Recursive Types for Foundational Proof-Carrying Code." ACM Trans. Program. Lang. Syst. 23, 5 (September 2001), 657-683. https://doi.org/10.1145/504709.504712

[2] Andrej Bauer and Alex Simpson. "Two Constructive Embedding-Extension Theorems with Applications to Continuity Principles and to Banach-Mazur Computability." Mathematical Logic Quarterly. Volume 50, Issue 4-5. (September 2004) 351-369.

[3] H.J. Boem "Constructive Interpretation of Numerical Programs." SIGPLAN Not. 22,7 (July 1987) 214-221. https://doi.org/0.1145/960114.29673

[4] Boldo, Sylvie and Jeannerod, Claude-Pierre and Melquiond, Guillaume and Muller, Jean-Michel. (2023) Floating Point Arithmetic. Acta Numerica. 32. 203-290. 10.1017/S0962492922000101.

[5] Crary, K. Harper, Robert Murphy, T. Swasey, David "A Separate Compilation Extension to Standard ML." Carnegie Mellon University Computer Science Division. 2006

[6] Dreyer, Derek Gaher, Lennard Sammler, Michael Spies, Simon "Quiver: Guided Abductive Inference of Separation Logic Specifications in Coq." Proc. ACM Program. Lang., Vol. 8, No. PLDI, Article 183. June 2024

[7] Gustafson, John L. The Posit System Annotated. In John L. Gustafson Every Bit Counts: Posit Computing (1st ed., pp97-146). Chapman and Hall 2025

[8] Shawn McLaughlin and John Harrison. 2005. A proof-producing decision procedure for real arithmetic. In Proceedings of the 20th international conference on Automated Deduction (CADE' 20) Springer-Verlag, Berlin. Heidelberg, 295-314. https://doi.org/10.1007/11532231_22

[9] Quine, N.V. "Concepts of Negative Degree." Proceedings of the National Academy of Science. Vol 22 (1936) pp40-45.

[10] Webb, Donald L. "Definition of Post's Generalized Negative and Maximum in Terms of One Binary Operation." American Journal of Mathematics. Vol. 58 (1936) pp.193-194.