Constructive Proofs or Constructive Statements?*

Julio Rubio

Dpto. de Matemáticas y Computación. Univ. de La Rioja. 26004 Logroño (Spain)
julio.rubio@dmc.unirioja.es

Abstract In this work the following question is considered: is Sergeraert's "Constructive Algebraic Topology" (CAT, in short) really constructive (in the strict logical sense of the word "constructive")? We have not an answer to that question, but we are interested in the following: could have a positive (or negative) answer to the previous question an influence in the problem of proving the correctness of CAT programs (as Kenzo)? Studying this problem, we have observed that, in fact, many CAT programs can be extracted from statements (that is, from the specification of certain objects and constructions), without needing an extraction from proofs. This remark shows that the logic used in the proofs could be uncoupled with respect to the correctness of programs. Thus, the first question posed could be unimportant from the practical point of view. These rather speculative ideas will be illustrated by means of some elementary examples, where the Isabelle code extraction tool can be successfully applied.

This research is part of an ongoing project devoted to use the proof assistant Isabelle [6] to explore the correctness of programs in Sergeraert's theory for Constructive Algebraic Topology (CAT) [7]. The main realization of the CAT framework is a Common Lisp program called Kenzo [5]. Due to the intrinsic difficulty of the task we decided to simplify the problem in two ways: first, restricting our attention to a small (but relevant) fragment of Kenzo (namely, the BPL or Basic Perturbation Lemma [3]); second, tackling the problem of the mechanized proof of the theorems, and not of the correctness of the programs (our primary goal). Once the feasibility of this objective has been established (see [1]), our next goal was to apply the Isabelle code extraction tools [2] to our proofs in order to bridge the gap between theorems and programs. The extracted code would produce ML programs; another gap would be then to reach the Kenzo Common Lisp programs.

For example, we show here a Kenzo fragment from [5]: the following Lisp function encodes essentially the BPL (more concretely, its *series*; see below). The complexity of its formal analysis should be clear.

* Partially supported by SEUI-MEC, project TIC2002-01626

Dagstuhl Seminar Proceedings 05021
Mathematics, Algorithms, Proof
http://drops.dagstuhl.de/opus/volltexte/2006/289

```
(DEFUN BPL-*-sigma (homotopy perturbation)
   (declare (type morphism homotopy perturbation))
   (the morphism
      (let ((cmpr (cmpr (sorc perturbation)))
            (h-delta (cmps homotopy perturbation)))
         (declare
            (type cmprf cmpr)
            (type morphism h-delta))
         (flet
            ((sigma-* (degr gnrt)
                (declare
                   (fixnum degr)
                   (type gnrt gnrt))
                (do ((rslt (zero-cmbn degr)
                           (2cmbn-add cmpr rslt iterated))
                     (iterated (term-cmbn degr 1 gnrt)
                               (cmbn-opps (cmbn-? h-delta
                                                  iterated))))
                    ((cmbn-zero-p iterated) rslt)
                   (declare (type cmbn rslt iterated)))))
            (build-mrph
               :sorc (sorc homotopy) :trgt (sorc homotopy)
               :degr 0 :intr #'sigma-* :strt :gnrt
               :orgn `(bpl-*-sigma ,homotopy ,perturbation))))))
```

The statement of the BPL is the following (for the necessary definitions and notations, we refer to [1]).

Theorem 1. Basic Perturbation Lemma — *Let* $(f, g, h)\colon D_* \Rightarrow C_*$ *be a chain complex reduction and* $\delta_{D_*}\colon D_* \to D_*$ *a* perturbation *of the differential* d_{D_*} *satisfying the nilpotency condition with respect to the reduction* (f, g, h). *Then a new reduction* $(f', g', h')\colon D'_* \Rightarrow C'_*$ *can be obtained where the underlying graded groups of* D_* *and* D'_* *(resp.* C_* *and* C'_**) are the same, but the differentials are perturbed:* $d_{D'_*} = d_{D_*} + \delta_{D_*}, d_{C'_*} = d_{C_*} + \delta_{C_*}$, *and* $\delta_{C_*} = f\phi\delta_{D_*}g$; $f' = f\phi$; $g' = (1 - h\phi\delta_{D_*})g$; $h' = h\phi$, *where* $\phi = \sum_{i=0}^{\infty}(-1)^i(\delta_{D_*}h)^i$.

Even in more elementary examples, as:

```
(DEFMETHOD CMPS ((mrph1 morphism) (mrph2 morphism) &optional strt)
   ;;; ... lines skipped
  (build-mrph :sorc sorc2 :trgt trgt1 :degr (+ degr1 degr2)
              :intr #'(lambda (cmbn)
                        (declare (type cmbn cmbn))
                          (the cmbn
                              (cmbn-? mrph1 (cmbn-? mrph2 cmbn))))
              :strt :cmbn :orgn `(2mrph-cmps ,mrph1 ,mrph2 ,strt))
   ;;; ... lines skipped
```

that implements the composition of two morphisms, the differences with the formalization in Isabelle are quite big. One fragment of such a formalization can be found here[1]:

```
constdefs
   group_mrp_comp :: "[ ('b, 'c) group_mrp_type,
                        ('a, 'b) group_mrp_type] =>
                        ('a, 'c) group_mrp_type"
   "group_mrp_comp g f  == \<lparr> src = src f, trg = trg g,
                            morph = (morph g) \<circ> (morph f),
      src_comm_gr = src_comm_gr f, trg_comm_gr = trg_comm_gr g
                           \<rparr>"

lemma group_mrp_composition:
  assumes A1: "group_mrp A"
  and B1: "group_mrp B"
  and C1: "trg_comm_gr A = src_comm_gr B"
  and D1: "trg A = src B"
  shows "group_mrp (B \<circ> A)"
```

Thus, in order to link the formalization with a *real* program, more efforts are needed. One well-known strategy is to establish the formalization in a (mechanized) *constructive* logic, and then extract a program from the proof. Nevertheless, one can observe that the previous Isabelle lemma has a *constructive statement*. That is to say: a new object is defined (the composite of two morphisms, in the example) and some property of this object is asserted (namely, it is a morphism). If code can be extracted from the definition or specification appearing in the statement, it is quite clear that the logic underlying the proof of the statement could be unrestricted (see [4]). In the example, we can apply in this way Berghofer's extraction tool [2], obtaining the following ML program (only the most relevant part is shown here):

```
fun comp f g = (fn x => f (g x));

fun group_mrp_comp g f =
  group_mrp_type_ext (src f) (trg g) (comp (morph g) (morph f))
                     (src_comm_gr f) (trg_comm_gr g) Unity;
```

In this case, the proof of the previous Isabelle Lemma is to be considered as a proof of correctness for the ML program (assuming, as usual, the soundness of Berghofer's translation). This (certified correct) ML program should be compared with the *real* corresponding Kenzo program (partially displayed before).

The scope and practical interest of this *"constructive statements"* approach need additional investigations (in particular, in order to formalize it). It is even

[1] The Isabelle scripts used in this paper have been written by J. Aransay, with the help of C. Ballarin.

unclear if each function in Kenzo has an equivalent in ML extracted from such a constructive statement. In this vein, it is worth noting that the occurrence in the BPL statement of the series $\phi = \sum_{i=0}^{\infty}(-1)^i(\delta_{D*}h)^i$ implies further difficulties from a constructive point of view.

References

1. J. Aransay, C. Ballarin and J. Rubio, *Four approaches to automated reasoning with differential algebraic structures*, AISC 2004, LNAI vol. 3249, pp. 222-235.
2. S. Berghofer, *Program Extraction in Simply-Typed Higher Order Logic*, TYPES 2002, LNCS vol. 2646, pp. 21-38.
3. R. Brown, *The twisted Eilenberg-Zilber theorem*, Celebrazioni Arch. Secolo XX, Simp. Top. (1967), pp. 34-37.
4. J. M. Davenport, *Effective Mathematics: the Computer Algebra viewpoint*, Lecture Notes in Mathematics 873 (1981), pp. 31-43.
5. X. Dousson, F. Sergeraert and Y. Siret, *The Kenzo program*, `http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/`
6. T. Nipkow, L. C. Paulson and M. Wenzel, *Isabelle/HOL: A proof assistant for higher order logic*, Lecture Notes in Computer Science, 2283, 2002.
7. J. Rubio and F. Sergeraert, *Constructive Algebraic Topology*, Bulletin des Sciences Mathématiques 126 (2002) 389-412.