Concurrent program extraction in computable analysis

Ulrich Berger and Hideki Tsuiki

July 21, 2017

In constructive logic and mathematics the meaning of a proposition is defined by describing how to prove it, that is, how to construct evidence for it. This is called the Brouwer-Heyting-Kolmogorov interpretation. For example,

- evidence for a conjunction, $A \wedge B$, is a pair (d, e) where d is evidence for A and e is evidence for B,
- evidence for a disjunction, $A \vee B$, is a pair (i, d) where i is 0 or 1 such that if $i = 0$ then d is evidence for A and if $i = 1$ then d is evidence for B,
- evidence for an implication, $A \to B$, is a computable procedure that transforms evidence for A into evidence for B.

Formalising this interpretation of propositions and the corresponding constructive proof rules leads to a method of program extraction from constructive proofs: From every constructive proof of a formula one can extract a program that computes evidence for it. The extracted programs are functional and possibly higher-order and can be conveniently coded in programming languages such as ML, Haskell or Scheme.

If one attempts to develop program extraction into a method of synthesising 'correct-by-construction' software, one realizes that one misses out an indispensable element of modern programming: *Concurrency*, that is, the composition of independently executing computations.

Our work is an attempt to fill this gap. We present an extension of constructive logic by a new formula construct $\mathbf{S}_n(A)$ with the following BHK interpretation:

- Evidence for $\mathbf{S}_n(A)$ is tuple of at most n computations running concurrently, at least one of which terminates, and each of which, if it terminates, computes evidence for A.

It turns out that the operator $\mathbf{S}_n$ becomes useful only in conjunction with a strong form of implication, $A \parallel B$, to be read 'A restricted to B'. The BHK interpretation of restriction is as follows:

- Evidence for $A \parallel B$ is a computation a such that
  - if there is evidence for B, then a terminates;
  - if a terminates, then it does so with a result that provides evidence for A.

We present proof rules for $\mathbf{S}_n(A)$ and $A \parallel B$ and give examples of proofs that give rise to concurrent extracted programs. Somewhat surprisingly, the two operators validate a concurrent version of the Law of Excluded Middle,

$$\frac{A \parallel B \quad A \parallel \neg B}{\mathbf{S}_2(A)}$$

Indeed, assuming evidence a for $A \parallel B$ and b for $A \parallel \neg B$, one obtains evidence for $\mathbf{S}_2(A)$ by executing a and b concurrently.

We look at two examples of proofs with concurrent computational content in the area of computable analysis.

The first example is concerned with infinite Gray code, an extension of the well-known Gray code for integers to a representation of the real numbers, introduced by Tsuiki [3]. One can prove that the (coinductive) predicate characterising this representation implies a concurrent version of the predicate characterising the signed digit representation and extract from this a concurrent program that translates infinite Gray code into signed digit representation. The extracted program is the same as the one given in [3].

The second example is about finding in a non-zero vector of real numbers an entry that is apart from zero. A concurrent program solving this problem can be extracted from a proof in the new logic. This can be further used to prove the invertibility of non-singular quadratic matrices and hence to extract a program for matrix inversion using a concurrent version of Gaussian elimination.

Currently, program extraction in this extended logic is done informally and the extracted programs are implemented in a concurrent extension of Haskell. It is future work to integrate the concurrent proof rules in a suitable interactive proof system (for example, Minlog) and to implement the corresponding program extraction procedure to make it fully automatic.

Prior to this work, a (non-concurrent) program translating an intensional version of infinite Gray code into signed digit representation has been extracted from a proof implemented in the Minlog system [1]. A precursor of our logical system is presented in [2]. It allows for the extraction of non-determinism and concurrent programs, however, without control over the number of threads, that is, processes running concurrently at the same time.

Acknowledgments

This work was supported by the International Research Staff Exchange Scheme (IRSES) Nr. 612638 CORCON and Nr. 294962 COMPUTAL of the European Commission, the JSPS Core-to-Core Program, A. Advanced Research Networks and JSPS KAKENHI Grant Number 15K00015.

The latest results were obtained while the authors were visiting the University of Canterbury, Christchurch, New Zealand, in April 2017. We are grateful to Douglas Bridges and Hannes Diener and the Mathematics Department of UC for hosting our visits which were part of the CORCON project.

References

[1] U. Berger, K. Miyamoto, H. Schwichtenberg, and H. Tsuiki. Logic for Gray-code computation. In *Concepts of Proof in Mathematics, Philosophy, and Computer Science*, Ontos Mathematical Logic 6. de Gruyter, 2016.

[2] Ulrich Berger. Extracting Non-Deterministic Concurrent Programs. In *CSL 2016*, volume 62 of *LIPIcs*, pages 26:1–26:21, 2016.

[3] H. Tsuiki. Real Number Computation through Gray Code Embedding. *Theoretical Computer Science*, 284(2):467–485, 2002.