Formal verification of programs in molecular models with random access memory

Mario J. Pérez–Jiménez, Fernando Sancho–Caparrini

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
E.T.S. Ingeniería Informática. Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: {marper,fsancho}@us.es

1 Introduction

Since the first result in molecular computing at the end of 1994 [1], some solutions to several **NP**-complete problems in this framework have been offered. In 1995 the first molecular models appeared. They are universal models, that is, with the same computational power as Turing Machines. Since then, the possibility of designing molecular programs having as input the instance of the problem to solve (until then, molecular computing was limited to solve particular instances, with no description of molecular operations in general cases) solving that problems appears. However, this brings about the necessity of a greater flexibility in the design of the programs and the necessity of a formal verification of the fact that those programs effectively solve the problem for which they were designed.

Usually, the first step to solve a concrete problem in computing models is to encode the input of the problem into the type of data that the model deals with (in this case, encoding will be made by means of tubes containing DNA molecules); next step consists in applying a finite sequence of basic operations of the model to the input data to get an output encoding the solution of the problem.

The goal of this paper is the presentation of a methodology to establish a formal verification of programs in molecular models with memory. The method is applied to molecular programs solving several well-known numerical **NP**-complete problems: the *Generating Cover Families* problem, the *Set Covering* problem, and the *Minimal Set Cover* problem.

The problems we solve in this work are associated with a finite family, $\mathcal{F}$, of subsets of a finite set. And they are the following ones: (1) generate every ordered pair $(\mathcal{F}', B)$ such that $\mathcal{F}'$ is a subfamily of $\mathcal{F}$ and $B = \cup\mathcal{F}'$; (2) generate all pairwise disjoint subfamilies of $\mathcal{F}$.

The paper is organized as follows: in section 2 a methodology to establish a formal verification of programs in molecular models with memory is given. Section 3 summarizes the main characteristics of the molecular model we use in this work: the Sticker Model. Sections 4 and 5 apply this methodology to two numerical problems. The main reason why we have chosen the above problems is because there exists a qualitative difference in the way their formal verification

have been established. On one hand, all the molecules of the input tube are kept (possibly modified) along the execution of the program; on the other hand, a filtering process is made, hence some molecules are thrown away because of they don't encode any of the valid solutions to the problem.

References

1. Adleman L. Molecular Computation of Solutions to Combinatorial Problems, *Science*, **268** (November 1994), 1021–1024.
2. Garey M.R.; Johnson D.S. *Computers and intractability*, W.H. Freeman and Company, New York, 1979.
3. Pérez-Jiménez, M.J.; Sancho-Caparrini, F. Solving Knapsack Problems in a Sticker Based Model, in N. Jonoska and N. Seeman (eds.) *DNA Computing, Lecture Notes in Computer Science*, **2340** (2002), 161-171.
4. Hoare, C.A.R. An axiomatic basis for computer programming, *Communications of the ACM*, **12** (1969), 576–583.
5. Pérez-Jiménez, M.J.; Sancho-Caparrini, F. Minimal Set Cover Problem: On a DNA solution of selection stage, in *Pre-Proceedings of Workshop on Membrane Computing*, Curtea de Arges, Romania, August 20-25, 2001 (C. Martín-Vide, Gh. Păun eds.), Report 17/01 of the Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain, 2001, 251–258.
6. Roweis, S.; Winfree, E.; Burgoyne, R.; Chelyapov, N.; Goodman, M.; Rothemund, P.; Adleman, L. A Sticker–Based Model for DNA Computation, *Journal of Computational Biology*, **5** (4) (1998), 615–629.