Using Automated Reasoning Systems on Molecular Computing

Carmen Graciani Díaz and Mario J. Pérez-Jiménez

Research Group on Natural Computing,
Dpto. Ciencias de la Computación e Inteligencia Artificial,
Universidad de Sevilla (Spain)
{cgdiaz, marper}@us.es

Abstract. This paper is focused on the interplay between automated reasoning systems (as theoretical and formal devices to study the correctness of a program) and DNA computing (as practical devices to handle DNA strands to solve classical hard problems with laboratory techniques). To illustrate this work we have proven in the PVS proof checker, the correctness of a program, in a sticker based model for DNA computation, solving the pairwise disjoint families problem. Also we introduce the formalization of the Floyd–Hoare logic for imperative programs.

1 Introduction

One of the most active areas of research in Computer Science is the study and use of formal methods (applications of primarily discrete mathematics to software engineering problems). Its widely development and the complexity of interesting problems have given rise to automated reasoning. In this area, one of the main problems is the correctness [2]: developing specifications and proofs that ensures a program meets its specification. There is a previous work of formalization: expressing all definitions, theorems and proofs in a formal language without semantic ambiguity. This approximation has especial relevance in new computing paradigms such as the DNA based molecular computing. In many molecular models the data are tubes over an alphabet whose content encodes a collection of DNA strands. The operations considered are abstraction of different laboratory techniques to manipulate DNA strands.

This paper is organized as follows. It begins with a short presentation of the Prototype Verification System (PVS) and the sticker model. Then, how this model can be formalized in PVS, is briefly described. Section 4 introduces imperative programs and gives an overview of how we deal with them in PVS. Finally, as an example, a molecular solution of the pairwise disjoint families problem and a description of its formal verification obtained with PVS, is presented. The set of developed theories in PVS for this paper are available on the web at http://www.cs.us.es/~cgdiaz/investigacion.
2 The Prototype Verification System

The Prototype Verification System (PVS) is a proof checker based on higher-order logic where types have semantics according to Zermelo-Fraenkel set theory with the axiom of choice \cite{8}. In such a logic we can quantify over functions which take functions as arguments and return them as values.

Specifications are organized into theories. They can be parameterized with semantic constructs (constant or types). Also they can import other theories. A prelude for certain standard theories is preloaded into the system. As an example we include in figure 1 the PVS theory suc_finitas_def which provides an alternative definition (to the type finseq given in the prelude) for sequences of a given length \( n \) for elements of a given type \( V \).

\[
suc\textunderscore finitas\textunderscore def[V: TYPE, n: nat]: THEORY
\begin{align*}
\text{BEGIN} \\
\text{\% Finite sequence: } S = \{s_k\}_{k<n}^{n+1} \\
\text{SUC\_FINITAS: TYPE} = [\text{below}[n] \rightarrow V] \\
\text{SF: TYPE} = \text{SUC\_FINITAS} \\
\text{END suc\textunderscore finitas\textunderscore def}
\end{align*}
\]

Fig. 1. A PVS Theory

Before a theory may be used, it must be typechecked. The PVS typechecker analyzes the theory for semantic consistency and adds semantic information to the internal representation built by the parser. Since this is an undecidable process, the checks which cannot be resolved automatically are presented to the user as assertions called type-correctness conditions.

The PVS prover is goal-oriented. Goals are sequents consisting of antecedents and consequents, e.g. \( A_1, \ldots, A_n \vdash B_1, \ldots, B_m \). The conjunction of the antecedents should imply the disjunction of consequents, i.e. \( A_1 \land \cdots \land A_n \rightarrow B_1 \lor \cdots \lor B_m \). The proof starts with a goal of the form \( \vdash B \), where \( B \) is the theorem to be proved. The user may type proof commands which either prove the current goal, or result in one or more new goals to prove. In this manner a proof tree is constructed. The original goal is proved when all leaves of the proof tree are recognized as true propositions. Basic proof commands can also be combined into strategies.

3 The Sticker Model: A Description Through PVS

The sticker model used in this paper was introduced by S. Roweis et al. \cite{9} (this model is completely different from the sticker systems introduced by L. Kari et al in \cite{6}). It is an abstract model of DNA based molecular computing with random access memory in the following sense: some operations could modify the structure of the DNA molecules and so the information codified by them changes during the execution.