Bachelor Project 2012

Analysis of the Euclidean Feature Transform algorithm

Sebastian Verschoor

Supervisors:

› prof. dr. Gerard R. Renardel de Lavalette
› prof. dr. Wim H. Hesselink
Table of Contents

› Project Goal
› Explaining the EFT
› Mechanical Verification
› Project Progress
› Evaluation
› Questions
The project goal

› The goal of this bachelor project is to mechanically verify (or even disprove) that the algorithm as posed by Hesselink [1] correctly calculates the Euclidean Feature Transform (EFT), and does so in linear time complexity.

› Mechanical Verification > Mathematical Proof
The Euclidean Feature Transform (EFT)
The EFT algorithm

- The algorithm uses some clever tricks
  - Iterating the dimensions, using the same algorithm for solving the base case and the inductive step

- Reduces the problem to finding the one-dimensional EFT
- $O(n)$ ($n$ number of "pixels")
The EFT algorithm

OneFT(n, h):

\[
\begin{align*}
q &\leftarrow 0; t[0] \leftarrow 0; at[0] \leftarrow 0 \\
\text{for } (k \leftarrow 1; k < n; k++) \\
\quad \text{while } (q \geq 0 \land f(t[q], at[q]) > f(t[q], k)) \\
\quad \quad q \leftarrow q - 1 \\
\quad \text{if } (q < 0) \\
\quad \quad q \leftarrow 0; at[0] \leftarrow k \\
\text{else} \\
\quad w \leftarrow 1 + g(at[q], k) \\
\quad \text{if } (w < n) \\
\quad \quad q \leftarrow q + 1 \\
\quad t[q] \leftarrow w; at[q] \leftarrow k \\
\end{align*}
\]

\[
\begin{align*}
t[q+1] &\leftarrow n; at[q+1] \leftarrow n - 1 \\
\text{for } (j \leftarrow 0; j = q; j++) \\
\quad x_1 \leftarrow t[j+1] - 1 \\
\quad \text{for } (x \leftarrow t[j]; x = x_1; x++) \\
\quad \quad FT[x] \leftarrow \{at[j]\} \\
\quad \text{for } (p \leftarrow at[j] + 1; p = at[j+1]; p++) \\
\quad \quad \text{if } (f(x_1, p) = f(x_1, at[j])) \\
\quad \quad \quad FT[x_1] \leftarrow FT[x_1] \cup \{p\}
\end{align*}
\]
Mechanical Verification

Prototype Verification System (PVS 5.0)
  • SRI International, Computer Science Laboratory

Specification Language

Interactive Prover
PVS Specification Language

› Based upon simple typed logic
› Formal specification of the problem
  • Types
  • Definitions
  • Theorems / Lemmas
PVS Prover

› Proof obligation
  • Logical sentence:
    \[ P_0 \land P_1 \land \ldots \land P_m \Rightarrow Q_0 \lor Q_1 \lor \ldots \lor Q_n \]

› Proof commands
  • Rewrite proof obligation to a logical equivalent statement

› The Prover does not prove anything!
  • It is merely keeps a "smart" administration
PVS Prover - Example
Program Correctness

› programs.pvs

• Hoare-Triples:
  • \{P\} S \{Q\}

• While loops
  • 5 steps
  • Prove correctness and termination
Project Progress (done)

› Learning PVS
  • Basics of the master course Automated Reasoning
› Understanding the algorithm
› Verified the mathematics
› The algorithm
  • Proved on paper
  • Specified in PVS
› 118 theorems/lemmas
  • 91 proven
Project Progress (todo)

› Prove the algorithm
  • With PVS
› Optional: prove the mathematics behind iterating the dimensions
› Write thesis
Evaluation

› Mechanically verifying a problem does not *result* in a deeper understanding of a problem
  • It does *require* a full understanding of the problem
› PVS is a great tool for proving complex mathematical theorems
  • But, often it feels like you do a lot of trivial work that could somehow be automated
Thank you for your attention

Are there any questions?
References