Progress in the Semantic Analysis of Scientific Code

Mark Stewart
Dynacs Engineering Company, Inc., Brook Park, Ohio

Prepared for the
Computational Aerosciences Workshop
sponsored by the High Performance Computing and Communications Program
Moffett Field, California, February 15–17, 2000

Prepared under Contract NAS3–98008

National Aeronautics and
Space Administration

Glenn Research Center

November 2000
This report contains preliminary findings, subject to revision as analysis proceeds.

Available from

NASA Center for Aerospace Information
7121 Standard Drive
Hanover, MD 21076
Price Code: A03

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22100
Price Code: A03

Available electronically at http://gltrs.grc.nasa.gov/GLTRS
Existing software analysis tools use the semantics of the programming language to check our codes: Are variables declared and initialized? Do variable types match? Where do memory leaks and memory errors occur? However, the meaning or semantics that a code developer builds into his/her code extends far beyond programming language semantics. Scientific code developers use variables to represent physical and mathematical quantities (mass, derivative), expressions of quantities to represent physical formulae (Navier-Stokes equation), loops to apply these formulae in a domain, and conditional expressions to control execution. These semantic details are crucial when developers and users try to understand and check their scientific and engineering codes; further, their analysis is manual, time-consuming, and error-prone.

This paper reports progress in an experiment to automatically recognize and check these physical and mathematical semantics. The experimental procedure combines semantic declarations with a pattern recognition capability; the code (1)

\[
\begin{align*}
C \& MA &= mass, ACC &= acceleration \\
FF &= MA \times ACC 
\end{align*}
\]

contains two semantic declarations for MA and ACC, and with Newton's law among the recognizable patterns, the procedure recognizes this code as force assigned to FF. These formula patterns are represented in and recognized by parsers\(^1\). The conclusions of this procedure are displayed for the user as shown in Figure 1. A more detailed explanation of this procedure and its extensions is given in Reference 2.

This experiment's objective is to understand the limits of this automatic recognition procedure: Does it apply to a wide range of scientific and engineering codes? Can it reduce the time, risk, and effort required to develop and modify scientific code?

Previous work\(^2\) demonstrated that scientific concepts and formulae could be represented and recognized. In fact, for part of one reacting flow code (Figure 2), 50% of the operations can be recognized. However, this preliminary work posed several more questions: Can additional semantic details be represented and recognized? How well do the recognition rules work in blind test cases? What are the limitations of this procedure?
Determine inlet static temperature from isentropic relations:
\[ \text{tsrat} = \text{gasin}(\text{emach} \cdot \text{Z}, \text{gam}) \]
\[ \text{tsin} = \text{tsrat} \cdot \text{t0ln} \]

Inlet temperature absolute:
\[ \text{TSIN} = \text{tsin} - \text{dtodr} \]

Determine inlet density, velocity, viscosity, Reynolds number:
\[ \text{rhoin} = \text{emach}^2 \cdot \sqrt{(\text{gam} \cdot \text{rgas} \cdot \text{tsin})} \]
\[ \text{auln} = \text{rhoin} \cdot \text{uin} \]
\[ \text{visin} = \text{visref} \cdot (\text{tsin} / \text{visref})^2 \cdot \text{vispwr} \]
\[ \text{areox} = \text{rhoin} \cdot \text{uin} \cdot \text{chord} \cdot \text{visin} \]

Determine inlet thermal conductivity and Prandtl number:
\[ \text{conln} = \text{conref} \cdot (\text{tsin} / \text{conref})^2 \cdot \text{conpwr} \]
\[ \text{prndll} = \text{visln} \cdot \text{cepe} / (\text{conln} \cdot \text{visln}^2 / \text{conln} \cdot \text{visln}^2) \]

Figure 1: GUI display for the semantic analysis program. The top window displays a user’s code; variables and expressions may be selected for explanation. The middle region explains this selected text. In this case, the physical quantity is density, it does not have a grid location, and it has the displayed dimensions, units, and derivation. The bottom region displays the semantic dictionary/lexicon.

To answer these questions, the procedure’s representation and recognition of semantic details has been significantly extended, including expert parsers for vector analysis, object analysis (the object of the formula), array reference/assignment analysis. Also, existing expert parsers have been refined and extended. A measure of the expert parsers is given in Table 1. Table 2 samples the rules represented in these parsers.
Subroutine Understanding vs. Semantic Declarations

20 Allspd Routines; 5278 FORTRAN Sima [No Comments]; 3431 Operations to Understand

Figure 2: Graph showing the increase in expression understanding as semantic declarations are added to twenty subroutines from the ALLSPD code. The subroutines contain 5278 non-comment FORTRAN statements and 3431 operations to understand. Further work will increase the understanding fraction. The analysis results reflect the analysis code's quality and not the quality or ability of the ALLSPD code.

<table>
<thead>
<tr>
<th>Aspect Analyzed</th>
<th>Parsers</th>
<th>Parser Rules</th>
<th>Fundamental Equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantity-Math</td>
<td>5</td>
<td>772</td>
<td>72</td>
</tr>
<tr>
<td>Quantity-Physical</td>
<td>3</td>
<td>766</td>
<td>114</td>
</tr>
<tr>
<td>Value / Interval</td>
<td>2</td>
<td>223</td>
<td>27</td>
</tr>
<tr>
<td>Grid Location</td>
<td>4</td>
<td>1801</td>
<td>235</td>
</tr>
<tr>
<td>Geometrical Entity</td>
<td>1</td>
<td>447</td>
<td>20</td>
</tr>
<tr>
<td>Vector Entity</td>
<td>1</td>
<td>300</td>
<td>15</td>
</tr>
<tr>
<td>Non-Dimensional</td>
<td>1</td>
<td>72</td>
<td>5</td>
</tr>
<tr>
<td>Dimensions</td>
<td>1</td>
<td>59</td>
<td>10</td>
</tr>
<tr>
<td>Units</td>
<td>1</td>
<td>71</td>
<td>14</td>
</tr>
<tr>
<td>Object Analysis</td>
<td>1</td>
<td>128</td>
<td>10</td>
</tr>
<tr>
<td>Array Analysis</td>
<td>2</td>
<td>121</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 1: Aspect analyses performed by the semantic analysis procedure including number of parsers for each aspect, number of Yacc^1 parser rules, and fundamental equations. Equation (1) corresponds to a fundamental equation; some equations require several parser rules.
Table 2: A sampling of expert parser rules used in the semantic analysis method. Many rules are condensed. Due to decomposition a single operation may involve multiple independent aspects (units, grid location and quantity for \(x\) coordinate – \(x\) coordinate), and several rules from this table can apply to it.

To understand the procedure’s generality, that is, if the rules and recognition capability can apply to a range of codes, the procedure’s performance was tested on large blind test cases. Semantic declarations for solution variables and coordinates were included in the ADPAC code (a 3D Navier-Stokes, curvilinear coordinate, turbomachinery code with 86k lines of code (Ioc)) and the ENG10 code (an axisymmetric, curvilinear coordinate, engine simulation code with 20k Ioc). The fraction of operations recognized is shown in Figure 3. These baseline results provide some initial evidence of generality, however, how these measurements improve as the procedure develops further is most important.
Assessing the future of this procedure is problematic, however experience indicates that three issues will determine success. First, the large number of formulae used in scientific codes—even within a field—makes it difficult, but not a priori impossible, to capture the knowledge necessary for recognition. Second, although one rule application or inference is necessary to recognize equation (1), and the formula $\sqrt{u_x^2 + u_y^2 + u_z^2}$ involves six inferences, $O(10^5)$ inferences are often required as expressions are evaluated and combined. Needing many inferences to find a result magnifies the risk of failure since an unknown inference, a limitation of this procedure, or a coding error will terminate the inference chain and leave the result unidentified. Hence, success of this method depends on good coverage of the domain knowledge, a robust semantic analysis procedure, and stable procedure coding. Third, representation of semantic details has not been a major problem, however continued success in representing knowledge is important.

Future work will pursue two questions. First, can formulae be added to the expert parsers so that the knowledge domain is sufficiently covered for good recognition of general codes? Second, can the procedure be perfected to a useful scientific software tool? The best way to answer these questions is to develop the procedure further while testing it on more codes.

**Figure 3**: Graph showing the increase in expression understanding as semantic declarations are added to two blind test cases. The ADPAC codes contain 86k loc, and the ENG10 code contains 20k loc. Further work will increase the understanding fraction. The analysis results reflect the analysis code's quality and not the quality or abilities of the ADPAC or ENG10 codes.
This paper concerns a procedure that analyzes aspects of the meaning or semantics of scientific and engineering code. This procedure involves taking a user's existing code, adding semantic declarations for some primitive variables, and parsing this annotated code using multiple, independent expert parsers. These semantic parsers encode domain knowledge and recognize formulae in different disciplines including physics, numerical methods, mathematics, and geometry. The parsers will automatically recognize and document some static, semantic concepts and help locate some program semantic errors. These techniques may apply to a wider range of scientific codes. If so, the techniques could reduce the time, risk, and effort required to develop and modify scientific codes.