Formal Correctness and Stability of Dense Linear Algebra Algorithms

Paolo Bientinesi **Robert van de Geijn**
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
{pauldj,rvdg}@cs.utexas.edu

Abstract - The Formal Linear Algebra Methods Environment (FLAME) project at UT-Austin pursues the mechanical derivation of algorithms for linear algebra operations. Rather than proving loop based algorithms correct *a posteriori*, a systematic methodology is employed that determines loop invariants from a mathematical specification of a given linear algebra operation. Algorithms and their correctness proofs are then constructively derived from this loop invariant.
The process has been made mechanical via a prototype system implemented with Mathematica. Once an algorithm has been determined, a similarly systematic *a posteriori* process is used to determine the correctness in the presence of roundoff error (stability properties) of the algorithm. In this paper, we report progress towards the ultimate goal of full automation of the system.

Keywords—**formal derivation, linear algebra algorithms, stability analysis, automated system, mechanical derivation**

I. INTRODUCTION

In Fig. 1, we give a generic "worksheet" for deriving a large class of loop based linear algebra algorithms. The worksheet resembles the skeleton of a linear algebra algorithm in the sense that it consists of an initialization step (Step 4) and a while loop (Step 3). In the body of the while loop updates with operands are performed (Steps 5a, 8 and 5b). The skeleton is also annotated with predicates, in curly-brackets, that describe the status of output variables at different stages in the algorithm (Steps 1a, 2, 2,3, 6, 7, 1b). The steps number refer to the first column of Fig. 1 and the numbering follows the order in which the boxes are filled to yield a valid algorithm.

The basic idea behind the FLAME approach to deriving algorithms is that the predicates in the worksheet are systematically derived, after which the statements between the predicates are chosen to make those predicates correct. In other words, the approach is goal oriented: every command between two predicates is derived in such a way that the Hoare's triple $\{\mathrm{predicate}_{\mathrm{before}}\}\ \mathrm{command}\ \{\mathrm{predicate}_{\mathrm{after}}\}$ is satisfied. If it is possible to derive commands for all the predicates, then the derived algorithm will be guaranteed to be formally correct.

In Section II we describe, by means of a practical example, the steps necessary to derive a family of algorithms from the mathematical specification of a target operation. Key to this result is the fact that the loop invariant for the algorithms solving the target operation can be identified *a priori*. While the example is simple, the methodology has been shown to apply to most of the operations supported by the Basic Linear Algebra Subprograms, LAPACK and the RECSY library [LAW 79], [DON 88], [DON 90], [AND 92], [BIE 05], [QUI 03], [JON 02b], [JON 02a].

Step	Annotated Algorithm: $[C, D, \ldots] = \mathrm{op}(A, B, C, \ldots)$
1a	$\{\ P_{\mathrm{pre}}\ \}$
4	**Partition** **where**
2	$\{\ P_{\mathrm{inv}}\ \}$
3	**while** G **do**
2,3	$\{(\ P_{\mathrm{inv}}\ ) \wedge (\ G\ )\}$
5a	**Repartition** **where**
6	$\{\ P_{\mathrm{before}}\ \}$
8	S_U
7	$\{\ P_{\mathrm{after}}\ \}$
5b	**Continue with**
2	$\{\ P_{\mathrm{inv}}\ \}$
	enddo
2,3	$\{(\ P_{\mathrm{inv}}\ ) \wedge \neg (\ G\ )\}$
1b	$\{\ P_{\mathrm{post}}\ \}$

Fig. 1. Worksheet for developing linear algebra algorithms.

Although the procedure that we describe in Sect. II ensures the derived algorithm to be formally correct, it is the application of the procedure itself that is error prone, due to tedious algebraic manipulations. As the complexity of the target operation increases, it becomes difficult to perform the procedure by hand. In Section III we illustrate how, starting from the loop invariant for a target operation, the steps of the procedure can be automated and performed by a symbolic system.

Our methodology generates many algorithms for the same target operation. The numerical stability properties of each algorithm needs to be ascertained, since often new algorithms are derived. And given that often times more than half a dozen algorithms are derived, it becomes a necessity to develop tools to help the user assert numerical properties for these algorithms. In Section IV we present an extended worksheet that allows the stability analysis to be performed by following a sequence of steps similar to the ones for deriving the algorithm itself.

II. A Concrete Example: $A := A + UU^T$

We describe here the eight-step procedure for filling out the worksheet in Fig. 1. We consider the SYmmetric Rank-K operation $A := A + UU^T$ (SYTRRK), where the $n \times n$ matrices A and U are, respectively, symmetric and upper triangular. We will assume that only the upper triangular part of A is stored. The completed worksheet is presented in Fig. 2, while the final algorithm appears in Fig. 3.

Steps 1a,1b: Determine $P_{\rm pre}$ and $P_{\rm post}$. The generic target operation is given by [C, D,...]=op(A, B, C,...). Some operands may be both input and output variables. The structure and properties of the input variables are given by the predicate $P_{\rm pre}$, the *precondition*. The *postcondition*, $P_{\rm post}$, is the predicate that describes the desired state upon completion of the algorithm. Predicates $P_{\rm pre}$ and $P_{\rm post}$ for the operation $A := A + UU^T$ are, respectively,

$A = \hat{A} \wedge A = A^T \wedge \text{UpperTri}(U)$ and $A = \hat{A} + UU^T$ as shown in Fig. 2. The predicates $P_{\rm pre}$ and $P_{\rm post}$ dictate how the remainder of the worksheet must be filled.

Step 2: Determine loop invariant $P_{\rm inv}$. The loop invariant is a predicate that expresses the contents of the variables during the computation. In order to determine possible intermediate contents of the output variable, one starts by partitioning the input and output operands. The partitioning corresponds to an assumption that algorithms progress through data in a systematic fashion.

In this example the matrices A and U are partitioned into quadrants to capture the symmetric and upper triangular structure:

$$A \rightarrow \left(\begin{array}{c|\|c} A_{TL} & A_{TR} \\ \hline\hline A_{TR}^T & A_{BR} \end{array}\right), \quad U \rightarrow \left(\begin{array}{c|\|c} U_{TL} & U_{TR} \\ \hline\hline 0 & U_{BR} \end{array}\right).$$

Here the indices T, B, L, and R stand for Top, Bottom, Left, and Right, respectively.

Now, the partitioned matrices are substituted into the postcondition after which algebraic manipulation expresses the final result in terms of operations on the original contents of those quadrants. We refer to this expression as the Partitioned Matrix Expression (PME):

$$\left(\begin{array}{c|\|c} A_{TL} & A_{TR} \\ \hline\hline \star & A_{BR} \end{array}\right) =$$

$$\left(\begin{array}{c|\|c} \hat{A}_{TL} + U_{TL}U_{TL}^T + U_{TR}U_{TR}^T & \hat{A}_{TR} + U_{TR}U_{BR}^T \\ \hline\hline \star & \hat{A}_{BR} + U_{BR}U_{BR}^T \end{array}\right).$$

Here $\hat{A}$ denotes the original content of the matrix A and $\star$ is used to indicate that the matrix is symmetric. At an intermediate stage (at the top of the loop body) only some of the operations will have been performed. For example, the intermediate state

$$\left(\begin{array}{c|\|c} A_{TL} & A_{TR} \\ \hline\hline \star & A_{BR} \end{array}\right) = \left(\begin{array}{c|\|c} \hat{A}_{TL} + U_{TL}U_{TL}^T & \hat{A}_{TR} \\ \hline\hline \star & \hat{A}_{BR} \end{array}\right) \tag{1}$$

comes from assuming that A_{TL} has been partially updated while the other parts of the matrix have not yet been touched. Let us use this loop invariant for the remainder of this discussion: it becomes $P_{\rm inv}$ in the worksheet in Fig. 1 as illustrated in Fig. 2.

Step 3: Determine loop guard G. The loop guard is the condition under which the program remains in the loop. Thus, when the loop completes, $\neg G$ holds, and so does $P_{\rm inv}$. We are seeking a loop guard so that $(P_{\rm inv} \wedge \neg G)$ implies $P_{\rm post}$; this condition dictates G. In our example, by comparing the loop invariant (1) and the postcondition $\{A = \hat{A} + UU^T\}$ it's easy to determine that $G = \text{SameSize}(\text{A}, \text{A}_{\text{TL}})$, where the predicate $\text{SameSize}(\text{A}, \text{A}_{\text{TL}})$ is *true* iff the dimensions of A and A_{TL} are equal.

Step 4: Determine initialization. Since the loop invariant must hold before the loop commences, the *initialization*, Step 4 in Fig. 1, must have the property that starting in the state $P_{\rm pre}$, it sets the variables to a state in which $P_{\rm inv}$ holds: $\{P_{\rm pre}\}$ Step 4 $\{\text{P}_{\text{inv}}\}$. Ideally, only the partitioning of operands is required to attain this state. Notice that the initialization in Step 4 in Fig. 2 has this property.

Steps 5a, 5b: Determine how to traverse the operands. In our notation, the double lines indicate progress through the operands. For the SYTRRK operation, initially the top-left quadrant of matrix A is empty, and ultimately that quadrant must encompass all of the matrix. The repartitioning of the operands in Step 5a and the moving of the double lines in Step 5b expose and move rows and columns from the bottom-right quadrant to the top-left one. Respectively,

$$\left(\begin{array}{c|\|c} A_{TL} & A_{TR} \\ \hline\hline \star & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|\|c} A_{00} & A_{01} & A_{02} \\ \hline \star & A_{11} & A_{12} \\ \hline\hline \star & \star & A_{22} \end{array}\right) \text{ and}$$

$$\left(\begin{array}{c|\|c} A_{TL} & A_{TR} \\ \hline\hline \star & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|\|c} A_{00} & A_{01} & A_{02} \\ \hline \star & A_{11} & A_{12} \\ \hline\hline \star & \star & A_{22} \end{array}\right).$$

Step 6: Determine $P_{\rm before}$. Step 5a exposes submatrices of the operands. The predicate in Step 6 merely expresses the contents of those submatrices. This predicate is determined by substituting the exposed submatrices into the loop invariant. Formally: $\{P_{\rm inv}\}$ Step 5a $\{\text{P}_{\text{before}}\}$. Mathematically $P_{\rm before}$ is given by:

$$\textbf{Simplify}\left(\text{P}_{\text{inv}}\Big|_{\textbf{Repartitioning}}\right),$$

Step	Annotated Algorithm: $A := A + UU^T$
1a	$\left\{ A = \hat{A} \wedge A = A^T \wedge \text{UpperTri}(U) \right\}$
4	**Partition** $A \rightarrow \left(\begin{array}{c\|\|c} A_{TL} & A_{TR} \\ \hline\hline \star & A_{BR} \end{array}\right), U \rightarrow \left(\begin{array}{c\|\|c} U_{TL} & U_{TR} \\ \hline\hline 0 & U_{BR} \end{array}\right), \hat{A} \rightarrow \left(\begin{array}{c\|\|c} \hat{A}_{TL} & \hat{A}_{TR} \\ \hline\hline \star & \hat{A}_{BR} \end{array}\right),$ **where** A_{TL} and $\mathrm{U_{TL}}$ are 0×0
2	$\left\{ \left(\begin{array}{c\|\|c} A_{TL} & A_{TR} \\ \hline\hline \star & A_{BR} \end{array}\right) = \left(\begin{array}{c\|\|c} \hat{A}_{TL} + U_{TL}U_{TL}^T & \hat{A}_{TR} \\ \hline\hline \star & \hat{A}_{BR} \end{array}\right) \right\}$
3	**while** $\neg\mathrm{SameSize(A, A_{TL})}$ **do**
2,3	$\left\{ \left(\left(\begin{array}{c\|\|c} A_{TL} & A_{TR} \\ \hline\hline \star & A_{BR} \end{array}\right) = \left(\begin{array}{c\|\|c} \hat{A}_{TL} + U_{TL}U_{TL}^T & \hat{A}_{TR} \\ \hline\hline \star & \hat{A}_{BR} \end{array}\right) \right) \wedge (\neg\mathrm{SameSize(A, A_{TL})}) \right\}$
5a	**Repartition** $\left(\begin{array}{c\|\|c} A_{TL} & A_{TR} \\ \hline\hline \star & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|\|c} A_{00} & A_{01} & A_{02} \\ \hline \star & A_{11} & A_{12} \\ \hline\hline \star & \star & A_{22} \end{array}\right), \left(\begin{array}{c\|\|c} U_{TL} & U_{TR} \\ \hline\hline 0 & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|\|c} U_{00} & U_{01} & U_{02} \\ \hline 0 & U_{11} & U_{12} \\ \hline\hline 0 & 0 & U_{22} \end{array}\right)$ **where** A_{11} and $\mathrm{L_{11}}$ are $\mathrm{b} \times \mathrm{b}$
6	$\left\{ \left(\begin{array}{c\|c\|\|c} A_{00} & A_{01} & A_{02} \\ \hline \star & A_{11} & A_{12} \\ \hline\hline \star & \star & A_{22} \end{array}\right) = \left(\begin{array}{c\|c\|\|c} \hat{A}_{00} + U_{00}U_{00}^T & \hat{A}_{01} & \hat{A}_{02} \\ \hline \star & \hat{A}_{11} & \hat{A}_{12} \\ \hline\hline \star & \star & \hat{A}_{22} \end{array}\right) \right\}$
8	$A_{00} := A_{00} + U_{01}U_{01}^T$ $A_{01} := A_{01} + U_{01}U_{11}^T$ $A_{11} := A_{11} + U_{11}U_{11}^T$
7	$\left\{ \left(\begin{array}{c\|c\|\|c} A_{00} & A_{01} & A_{02} \\ \hline \star & A_{11} & A_{12} \\ \hline\hline \star & \star & A_{22} \end{array}\right) = \left(\begin{array}{c\|c\|\|c} \hat{A}_{00} + U_{00}U_{00}^T + U_{01}U_{01}^T & \hat{A}_{01} + U_{01}U_{11}^T & \hat{A}_{02} \\ \hline \star & \hat{A}_{11} + U_{11}U_{11}^T & \hat{A}_{12} \\ \hline\hline \star & \star & \hat{A}_{22} \end{array}\right) \right\}$
5b	**Continue with** $\left(\begin{array}{c\|\|c} A_{TL} & A_{TR} \\ \hline\hline \star & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|\|c} A_{00} & A_{01} & A_{02} \\ \hline \star & A_{11} & A_{12} \\ \hline\hline \star & \star & A_{22} \end{array}\right), \left(\begin{array}{c\|\|c} U_{TL} & U_{TR} \\ \hline\hline 0 & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|\|c} U_{00} & U_{01} & U_{02} \\ \hline 0 & U_{11} & U_{12} \\ \hline\hline 0 & 0 & U_{22} \end{array}\right)$
2	$\left\{ \left(\begin{array}{c\|\|c} A_{TL} & A_{TR} \\ \hline\hline \star & A_{BR} \end{array}\right) = \left(\begin{array}{c\|\|c} \hat{A}_{TL} + U_{TL}U_{TL}^T & \hat{A}_{TR} \\ \hline\hline \star & \hat{A}_{BR} \end{array}\right) \right\}$
	enddo
2,3	$\left\{ \left(\left(\begin{array}{c\|\|c} A_{TL} & A_{TR} \\ \hline\hline \star & A_{BR} \end{array}\right) = \left(\begin{array}{c\|\|c} \hat{A}_{TL} + U_{TL}U_{TL}^T & \hat{A}_{TR} \\ \hline\hline \star & \hat{A}_{BR} \end{array}\right) \right) \wedge \neg\,(\neg\mathrm{SameSize(A, A_{TL})}) \right\}$
1b	$\left\{ A = \hat{A} + UU^T \right\}$

Fig. 2. Worksheet for developing an algorithm for the symmetric rank-k update.

where the vertical bar signifies the application of textual substitution rules and the function **Simplify** indicates algebraic simplifications.

Step 7: Determine P_{after}. The Hoare's triple $\{P_{\text{after}}\}$ Step 5b $\{\mathrm{P_{inv}}\}$ must hold. Therefore the predicate P_{after} denotes the loop invariant before the **Continue with** (Step 5b) execution. Since P_{inv} is known, P_{after} can be determined by executing the converse of the **Continue** statement, i.e. executing the textual substitution rules backwards. Mathematically, P_{after} is determined by:

$$\textbf{Simplify}\left(\mathrm{P_{inv}}\Big|_{\textbf{Continue}^{-1}}\right).$$

Both the predicates P_{before} and P_{after} are shown in Fig.2.

Step 8: Determine the updates S_U. The statements S_U must be such that the triple $\{P_{\text{before}}\}\ S_U\ \{P_{\text{after}}\}$ holds. The required updates S_U are determined by comparing the state in Step 6 with the desired state in Step 7.

Predicates and commands for the example that we are carrying on are presented in Fig. 2. Once the predicates are removed the resulting algorithm appears as shown in Fig. 3.

Correctness. If it is possible to successfully execute the eight steps we described, the resulting algorithm is guaranteed to be formally correct. It is in fact possible to state a sequence of valid Hoare's triples, the first one beginning with the predicate $\{P_{\text{pre}}\}$ and the last one ending with the predicate $\{P_{\text{post}}\}$. Thus the triple $\{P_{\text{pre}}\}\mathrm{algorithm}\{\mathrm{P_{post}}\}$ is also valid.

Family of Algorithms. Among the aforementioned eight steps, only one is not fully determined: the choice of the loop invariant P_{inv} (Step 2). Often times for a given PME, there are many loop invariants yielding valid algorithms.

$$A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline \star & A_{BR} \end{array}\right), U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right),$$
$$\hat{A} \rightarrow \left(\begin{array}{c|c} \hat{A}_{TL} & \hat{A}_{TR} \\ \hline \star & \hat{A}_{BR} \end{array}\right)$$

while $\neg\mathrm{SameSize}(\mathrm{A}, \mathrm{A_{TL}})$ **do**

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline \star & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline \star & A_{11} & A_{12} \\ \hline \star & \star & A_{22} \end{array}\right),$$
$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} U_{00} & U_{01} & U_{02} \\ \hline 0 & U_{11} & U_{12} \\ \hline 0 & 0 & U_{22} \end{array}\right)$$

where A_{11} and $\mathrm{L_{11}}$ are $\mathrm{b} \times \mathrm{b}$

$$A_{00} := A_{00} + U_{01}U_{01}^T$$
$$A_{01} := A_{01} + U_{01}U_{11}^T$$
$$A_{11} := A_{11} + U_{11}U_{11}^T$$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline \star & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline \star & A_{11} & A_{12} \\ \hline \star & \star & A_{22} \end{array}\right),$$
$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} U_{00} & U_{01} & U_{02} \\ \hline 0 & U_{11} & U_{12} \\ \hline 0 & 0 & U_{22} \end{array}\right)$$

enddo

Fig. 3. Algorithm for the symmetric rank-k update.

Repeating Steps 3-8 for different loop invariants results in a family of algorithms solving the same operation. As an example, the following predicate is also a valid P_{inv} for the SYTRRK operation:

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline \star & A_{BR} \end{array}\right) =$$
$$\left(\begin{array}{c|c} \hat{A}_{TL} + U_{TL}U_{TL}^T + U_{TR}U_{TR}^T & \hat{A}_{TR} \\ \hline \star & \hat{A}_{BR} \end{array}\right)$$

III. Mechanical Generation of Algorithms

In this section we cover the eight steps forming the derivation procedure again, looking at how automation can be achieved. It will be apparent that once a loop invariant has been selected, all the other steps in the procedure are completely determined and therefore computable by a mechanical system.

Step 1: Determine P_{pre} and P_{post}. These two predicates are given as part of the specifications for the operation we want to implement. We also assume the knowledge of the expression for P_{post} when the input operands are partitioned: the PME. **The expressions P_{pre}, P_{post} and PME are the input to a mechanical system.**

Step 2: Determine P_{inv}. Loop invariants are obtained by selecting a subset of the operations that appear in the PME. Notice that not every subset leads to a viable algorithm.

Steps 3, 4, 5: Determine loop guard G, the initialization, and how to move the boundaries. These three steps can be performed by comparing the selected loop invariant P_{inv} and the expressions for P_{pre} and P_{post}. By means of repartitionings only, we want P_{inv} to hold before the loop is entered (initialization) and after the loop is completed (loop guard). Then the boundaries are moved in such a way to make progress towards rendering G false.

Steps 6, 7: Determine P_{before} and P_{after}. It is important to realize that the two predicates P_{before} and P_{after} are nothing more than the loop invariant expressed in terms of newly exposed parts of the operands, respectively at the top and the bottom of the loop. The expressions for P_{before} and P_{after} can therefore be computed by applying textual substitution, exploiting the knowledge of the PME and performing algebraic manipulations. It is this computation that becomes difficult to perform manually for complex operations. Fortunately it can be entirely automated.

Step 8: Determine the update S_U. The updates are determined by comparing of the states P_{before} and P_{after}. A symbolic system with pattern matching capabilities, like Mathematica, can be programmed to identify S_U.

Implementation of the Mechanical System. We have implemented a prototype mechanical system. Evidence suggests that the system is as general as the FLAME methodology itself: it has been successfully applied to complex operations like the triangular Sylvester equation ($LX + XU = C$) and the triangular Lyapunov equation ($LX + XL^T = C$), operations frequently encountered in control theory.

We now show how the system is applied to the SYTRRK operation. The complete output from the system is a worksheet identical to the one shown in Fig. 2. In Fig. 4 we instead display the output as produced by the mechanical system without the annotations (predicates). The resulting algorithm is equivalent to the algorithm we derived step-by-step in Sect. II.

We briefly comment on how the mechanical system performs Steps 6, 7 and 8. First the predicate P_{before} is computed. This predicates displays the current contents of the output variables. The result matches exactly with Step 6 in Fig. 2 (the empty triangles are used to represent a symmetric matrix):

```
loop invariant before the updates:
```

$$\left(\begin{array}{c|c|c} \mathtt{A_{00}} & \mathtt{A_{01}} & \mathtt{A_{02}} \\ \hline \triangledown & \mathtt{A_{11}} & \mathtt{A_{12}} \\ \hline \triangledown & \triangledown & \mathtt{A_{22}} \end{array}\right) = \left(\begin{array}{c|c|c} \mathtt{U_{00}.T[U_{00}]} + \hat{\mathtt{A}}_{\mathtt{00}} & \hat{\mathtt{A}}_{\mathtt{01}} & \hat{\mathtt{A}}_{\mathtt{02}} \\ \hline \triangledown & \hat{\mathtt{A}}_{\mathtt{11}} & \hat{\mathtt{A}}_{\mathtt{12}} \\ \hline \triangledown & \triangledown & \hat{\mathtt{A}}_{\mathtt{22}} \end{array}\right)$$

Then the predicate P_{after} is computed. This predicate displays what the variables need to contain at the bottom of the loop. The result matches exactly with Step 7 in Fig. 2:

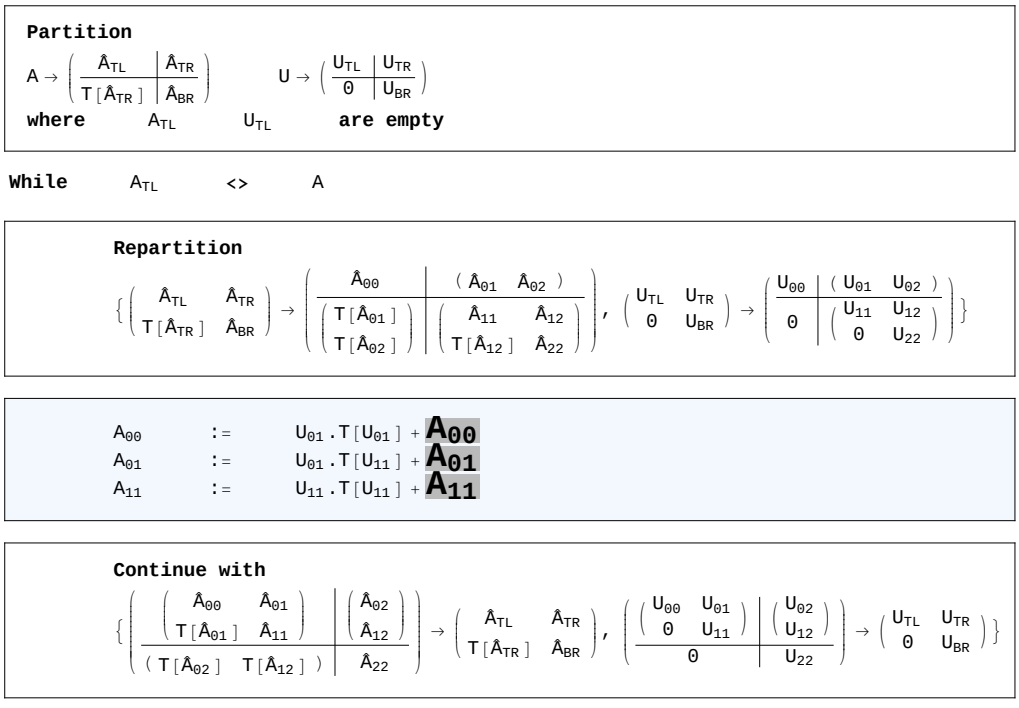
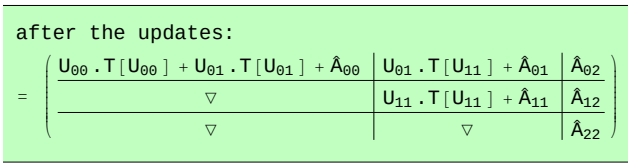
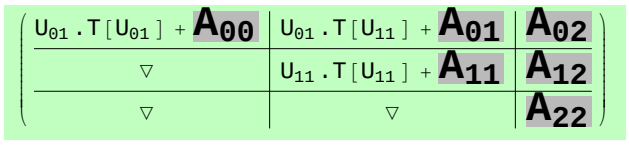


Fig. 4. Mechanical Derivation of Algorithms: SYTRRK



This last predicate is then simplified by performing pattern matching to identify expressions from P_{before} that also appear in P_{after}. For instance the quadrant A_{00} in P_{after} equals $\hat{A}_{00} + U_{00}U_{00}^T + U_{01}U_{01}^T$, but the expression $\hat{A}_{00} + U_{00}U_{00}^T$ appears as A_{00} in P_{before}, therefore can be replaced by $\boxed{\text{A}_{00}}$. Once this step has been performed, any quadrant X_{ij} in P_{after} containing only the expression $\boxed{\text{X}_{ij}}$ requires no updates. In our example, the simplified predicate P_{after} is (quadrants A_{02}, A_{12} and A_{22} require no operation):



This final expression for P_{after} is then transformed into the traditional assignments: $A_{00} := A_{00} + U_{01}U_{01}^T$, $A_{01} := A_{01} + U_{01}U_{11}^T$ and $A_{11} := A_{11} + U_{11}U_{11}^T$ which have to be performed at each iteration of the loop in order to maintain the loop invariant true.

IV. Systematic Error Analysis

Thanks to the mechanical system we presented in the former section, it is possible to obtain a family of algorithms for one operation in matter of seconds. Therefore the users can select the best algorithm for their needs, according to the target given system and environment (sequential, distributed memory, shared memory,..). It is then important to be able to assert numerical properties of the generated algorithms. A numerically unstable algorithm is useless, regardless of performance considerations. Coming up with numerical stability analysis by hand for each algorithm in the family is a difficult and tedious effort, especially when facing operations like the Sylvester equation, for which more than 15 different algorithms are produced.

In Fig. 5 we propose an extended version of the worksheet in Fig. 1 that facilitates the derivation of stability results via induction. Associated with the extended worksheet we introduce a three-stage procedure to identify stability properties of a derived algorithm. The first stage of the procedure is the standard FLAME derivation of algorithms. In the second stage a formula for the error analysis is determined. In the third and last stage error bounds for this formula are generated. The focus of this section is the second stage of

LA Op.		Stability formula	Step
Partition Operands **where**		Error Operands	4
{ Loop Invariant }		{ Error Invariant }	2
while G **do**			3
Repartition Operands **where**		Error Operands	5a
{ Loop Invariant }		{ Error Invariant }	6
Updates	Error Analysis for the Updates	Error Updates	8
{ Loop Invariant }		{ Error Invariant }	7
Continue with Operands		Error Operands	5b
enddo			

Fig. 5. Extended worksheet for deriving linear algebra algorithms and proving their stability properties.

the procedure.

Space constraints require us to limit the discussion to a very simple problem: the determination of the backward stability of the inner product $\kappa := x^T y$. We must show that the computed quantity $\check{\kappa}$ is the exact result for slightly perturbed inputs. This can be expressed as $\check{\kappa} = x^T \Delta y$, where the diagonal matrix Δ is given by the sum of the identity matrix I and the matrix Θ, whose entries have magnitude of the order of the unit roundoff ($\Delta = I + \Theta$).

In the worksheet (Fig. 6), the notation $[\text{expression}]$ is used to indicate the result that is computed in the presence of roundoff error (other texts often use the notation $fl(\text{ expression })$). Also, notice the difference between the operator $:=$ which represents the assignment, and $=$ which signifies the equality relation.

Looking at Fig. 6, the left side of the worksheet is used to derive an algorithm for the operation $\kappa := x^T y$. On the right side, the operation $\check{\kappa} = x^T \Delta y$ is considered, where the matrix Δ is the unknown that we want to compute. It is important to stress that the operation on the right side of the worksheet corresponds to the error analysis for the algorithm on the left side. The difference between the derivation process in the left and right columns is that while the updates (Step 8) for the operation on the left are strictly dictated by the operation, the updates on the right side are dictated by the operation itself and by the error introduced by the updates in the presence of roundoff error.

In our example, the inner product, we choose the loop invariant $\check{\kappa} := x_T^T y_T$ (Step 2 on the left). The loop invariant determines the update $\check{\kappa} := [\check{\kappa} + \chi_1 \psi_1]$, as shown in the left-most column of Step 8. In the middle column of Step 8 the error analysis for the update is performed, using the standard computation model

$$\big[x \text{ op } y\big] = (x \text{ op } y)(1+\epsilon), \quad |\epsilon| \leq \mathbf{u}, \text{ and } \text{op} = +, -, *, /$$

and the resulting analysis is:

$$\check{\kappa} = \big(\check{\kappa} + \chi_1 \psi_1 (1+\epsilon_*)\big)(1+\epsilon_+), \tag{2}$$

where ϵ_* is the error due to the multiplication $\chi_1 \psi_1$ and ϵ_+ is the error due to the sum $\kappa + \chi_1 \psi_1$. Recall that we want to prove that $\check{\kappa} = x^T \Delta y$ holds, where Δ is a diagonal matrix. Equivalently we can show that at each iteration of the algorithm, the equality $\check{\kappa} = x_T^T \Delta_T y_T$ holds (Step 2), where Δ_T is the top-left quadrant of Δ (Step 4). For this, repartitioning (Step 5a) the matrix Δ as $\left(\begin{array}{c|c|c} \Delta_0 & 0 & 0 \\ \hline 0 & \delta_1 & 0 \\ \hline 0 & 0 & \Delta_2 \end{array}\right)$, we obtain the inductive hypothesis $\check{\kappa} = x_0^T \Delta_0 y_0$, (Step 6) that can be exploited to prove the theorem. Combining formula (2) with the inductive hypothesis we obtain

$$\check{\kappa} = x_0^T \Delta_0 (1+\epsilon_+) y_0 + \chi_1 (1+\epsilon_*)(1+\epsilon_+)\psi_1$$

which dictates how the matrix Δ_0 and the scalar δ_1 have to be updated to maintain the loop invariant $\check{\kappa} = x_T^T \Delta_T y_T$ (Step 8):

$$\Delta_0 := \Delta_0 (1+\epsilon_+)$$
$$\delta_1 := (1+\epsilon_+)(1+\epsilon_*)$$

It is then easy to find bounds on the entries of the matrix Δ. The reader may have the feeling that this approach introduced unnecessary complications to prove a simple fact like the backward stability of the inner product. Instead, the strength of the approach relies on the modularity. Even for more complex operations the structure of the proof remains the same and results for simpler operations can be used incrementally. As an example, the extended worksheet yields a proof for the backward stability of the LU factorization that is as concise, in our opinion, as is the proof that appears in [HIG 02].

V. Conclusions

We have presented a methodology for deriving linear algebra algorithms. The methodology requires only the mathematical description for the operation for which an algorithm is to be found. It relies on the loop invariant concept to generate formally correct algorithms. The proof of correctness is inherently embedded in the derivation process. The methodology is so systematic that can be performed mechanically by a system. In the paper we introduced a prototype system written in Mathematica that achieves this.

$\kappa := x^T y$	$\check{\kappa} = x^T \Delta y$	Step
Partition $x \rightarrow \left(\begin{array}{c} x_T \\ \hline x_B \end{array}\right), y \rightarrow \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right)$ **where** x_T and y_T are 0×1 and Δ_T is 0×0	$\Delta \rightarrow \left(\begin{array}{c\|\|c} \Delta_T & 0 \\ \hline\hline 0 & \Delta_B \end{array}\right)$	4
$\left\{\check{\kappa} = \left\lfloor x_T^T y_T \right\rfloor\right\}$	$\left\{\check{\kappa} = x_T^T \Delta_T y_T\right\}$	2
while $m(x_B) > 0$ **do**		3
Repartition $\left(\begin{array}{c} x_T \\ \hline x_B \end{array}\right) \rightarrow \left(\begin{array}{c} x_0 \\ \hline\hline \chi_1 \\ \hline\hline x_2 \end{array}\right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right) \rightarrow \left(\begin{array}{c} y_0 \\ \hline\hline \psi_1 \\ \hline\hline y_2 \end{array}\right)$ **where** x_T, y_T and δ_1 are 1×1	$\left(\begin{array}{c\|\|c} \Delta_T & 0 \\ \hline\hline 0 & \Delta_B \end{array}\right) \rightarrow \left(\begin{array}{c\|\|c\|c} \Delta_0 & 0 & 0 \\ \hline\hline 0 & \delta_1 & 0 \\ \hline 0 & 0 & \Delta_2 \end{array}\right)$	5a
$\left\{\check{\kappa} = \left\lfloor x_0^T y_0 \right\rfloor\right\}$	$\left\{\check{\kappa} = x_0^T \Delta_0 y_0\right\}$	6
$\check{\kappa} := [\check{\kappa} + \chi_1 \psi_1]$	$\check{\kappa} := \big(\check{\kappa} + \chi_1\psi_1(1+\epsilon_*)\big)(1+\epsilon_+)$ $= x_0^T \Delta_0 (1+\epsilon_+) y_0 +$ $\chi_1(1+\epsilon_*)(1+\epsilon_+)\psi_1$	$\Delta_0 := \Delta_0(1+\epsilon_+)$ $\delta_1 := (1+\epsilon_+)(1+\epsilon_*)$	8
$\left\{\check{\kappa} = \left[x_0^T y_0 + \chi_1 \psi_1\right]\right\}$	$\left\{\begin{array}{r} \check{\kappa} = \left(\begin{array}{c} x_0 \\ \hline \chi_1 \end{array}\right)^T \left[\begin{array}{c\|c} \Delta_0 & \\ \hline & \delta_1 \end{array}\right] \left(\begin{array}{c} y_0 \\ \hline \psi_1 \end{array}\right) = \\ = x_0^T \Delta_0 y_0 + \chi_1 \delta_1 \psi_1 \end{array}\right\}$	7
Continue with $\left(\begin{array}{c} x_T \\ \hline x_B \end{array}\right) \leftarrow \left(\begin{array}{c} x_0 \\ \hline\hline \chi_1 \\ \hline\hline x_2 \end{array}\right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right) \leftarrow \left(\begin{array}{c} y_0 \\ \hline\hline \psi_1 \\ \hline\hline y_2 \end{array}\right)$	$\left(\begin{array}{c\|\|c} \Delta_T & 0 \\ \hline\hline 0 & \Delta_B \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|\|c} \Delta_0 & 0 & 0 \\ \hline 0 & \delta_1 & 0 \\ \hline\hline 0 & 0 & \Delta_2 \end{array}\right)$	5b
enddo		

Fig. 6. Extended worksheet used to proof the backward stability of the inner product

Experimental results suggest that the system is as general as the derivation methodology itself.

Correctness in the usual sense is not enough in the presence of roundoff error. In this paper we discussed how to extend the methodology to prove facts about the stability of the generated algorithms. The resulting process facilitates inductive proofs and modularity in the error analyses.

REFERENCES

[AND 92] ANDERSON E., BAI Z., DEMMEL J., DONGARRA J. E., DUCROZ J., GREENBAUM A., HAMMARLING S., MCKENNEY A. E., OSTROUCHOV S., SORENSEN D., *LAPACK Users' Guide*, SIAM, Philadelphia, 1992.

[BIE 05] BIENTINESI P., GUNNELS J. A., MYERS M. E., QUINTANA-ORTÍ E. S., VAN DE GEIJN R. A., *The Science of Deriving Dense Linear Algebra Algorithms, ACM Transactions on Mathematical Software*, vol. 31, n1, March 2005.

[DON 88] DONGARRA J. J., DU CROZ J., HAMMARLING S., HANSON R. J., *An Extended Set of FORTRAN Basic Linear Algebra Subprograms, ACM Transactions on Mathematical Software*, vol. 14, n1, p. 1-17, March 1988.

[DON 90] DONGARRA J. J., DU CROZ J., HAMMARLING S., DUFF I., *A Set of Level 3 Basic Linear Algebra Subprograms, ACM Transactions on Mathematical Software*, vol. 16, n1, p. 1-17, March 1990.

[HIG 02] HIGHAM N. J., *Accuracy and Stability of Numerical Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second dition, 2002.

[JON 02a] JONSSON I., KÅGSTRÖM B., *Recursive Blocked Algorithms for Solving Triangular Systems: Part II: Two-Sided and Generalized Sylvester and Lyapunov Matrix Equations, ACM Transactions on Mathematical Software*, vol. 28, n4, p. 416–435, December 2002.

[JON 02b] JONSSON I., KÅGSTRÖM B. B., *Recursive Blocked Algorithms for Solving Triangular Systems: Part I: One-Sided and Coupled Sylvester-type Matrix Equations, ACM Transactions on Mathematical Software*, vol. 28, n4, p. 392–415, ACM Press, 2002.

[LAW 79] LAWSON C. L., HANSON R. J., KINCAID D. R., KROGH F. T., *Basic Linear Algebra Subprograms for Fortran Usage, ACM Trans. Math. Soft.*, vol. 5, n3, p. 308-323, Sept. 1979.

[QUI 03] QUINTANA-ORTÍ E. S., VAN DE GEIJN R. A., *Formal Derivation of Algorithms: The Triangular Sylvester Equation, ACM Transactions on Mathematical Software*, vol. 29, n2, p. 218–243, June 2003.