Developing Proof Carrying Code to Formally Assure Termination in Fault Tolerant Distributed Controls Systems

Romain Jobredeaux, Heber Herencia-Zapana, Natasha Neogi, and Eric Feron

Abstract—We address the semantic gap between the model and the implementation of control algorithms for distributed systems in a formal fashion: we provide an interactive (partially automated) method by which to translate the global, model level theoretical properties, such as stability and convergence, into code level assertions and invariants which assure termination of the relevant implementations and validity of the result. We outline a simple flight control system example for altitude holding, and describe a fault-tolerant distributed agreement protocol among \(n\) processor nodes, in the presence of a varying network topology. We develop a Lyapunov-like function for the distributed algorithm, and attain a polynomial time theoretical bound on the convergence time \(T_{\text{conv}} \propto n^3\). We then exploit the Lyapunov function and convergence proof, in order to derive requisite ANSI-C Specification Language (ACSL) annotations for the C code distributed implementation, in order to guarantee termination. The ACSL annotations can then be used to automatically generate formal proof obligations via the Frama-C tool. These proof obligations also encompass assertions relating to timing, memory allocation, type checking and processor specific issues. The proof obligations can then be automatically discharged using a theorem prover, and their success guarantees that the behavior of the corresponding distributed code will evince the global properties proven for the model. Thus, the verification process spans efforts from the high-level control theory to the low-level implementation as a C program.

I. INTRODUCTION

The distributed control of networked systems impacts critical infrastructures such as transportation, telecommunications, power management, emergency response and food and water distribution. The verification and validation of the global emergent properties of a distributed, reactive control system poses several challenges. Issues related to global and local timing, interprocess communication protocols, allowable fault models, and processor nondeterminism all give rise to subtle complications in the assessment of system level safety, stability and convergence. Furthermore, the ability to ensure that the properties guaranteed by the theoretical analysis of distributed system will be demonstrated by its actual hardware and software implementation is a critical piece in the verification, validation and eventual certification of such systems.

Correctness guarantees on the emergent behavior of distributed agents in the presence of faults is vital in ensuring critical system properties. Much work has been done regarding the problem of consensus in distributed systems, in both the computer science and controls community. Controls oriented frameworks outlining notions of robustness [1],[2],[3], based on fundamental results from the fault-tolerant distributed consensus problem [4],[5], have proven to be extremely successful in analyzing convergence and termination properties [1]. However, there does not exist a simple means by which to ensure that the high level, theoretically-proven properties of the controller are realized by the coded implementation.

In this paper, we develop theoretical stability and termination guarantees for a fault-tolerant distributed algorithm employed over a networked control system. The structure of the stability and termination proofs is then used to generate annotations which are embedded in proof carrying code, in order to enable formal verification of the implemented software. A brief survey of the state of the art follows in the next section, along with a simplified problem statement. We outline and formally model a fault-tolerant distributed agreement algorithm, then employ Lyapunov-style arguments to theoretically prove stability and a bounded convergence time in Sections III and IV, respectively. We then employ Floyd-Hoare logic in order to create annotational invariants which allow for the local expression of the relevant global properties (stability and termination) in Sections V-VI. We build on current work in bounded-input, bounded-state Lyapunov stability proofs for software implementation [6], [7], [8] by extending the notions provided to encompass timing in distributed systems. We also highlight the issues of both convergence in the face of faults at the model level, and the novel agreement and validity results ensured upon termination at the code level, previously unaddressed in this framework, in Section VII.

II. STATE OF THE ART AND PROBLEM STATEMENT

A. Brief Background

Although stability is distinct from convergence and termination, constructing proofs for both these properties can employ the existence of Lyapunov-like functions. An approach to multi-agent coordination utilizing a Lyapunov function for control purposes is posited in [9], and convergence bounds on the flocking problem is developed via Lyapunov-like functions in [10]. Olshesky et. al. develop a theoretical bound on convergence in the consensus problem for both fixed and varying topology networks, and benchmark several consensus algorithms via simulation [11]. Kim, Mitra and Kumar develop Lyapunov-like conditions for both stability and termination of a restricted class of linear hybrid systems in [12].
To our knowledge, apart from the work of [6], [7], [8], no other research endeavor addresses the issue of proving in the C code the high-level correctness properties of control systems such as stability. Some successful attempts have been made at extracting quadratic invariants from the code, in [13] and [14].

There is a large body of literature on proving termination of programs and recursive functions using theorem provers [15], [16]. For example a recent project, Coqelicot, develops real functional analysis, Gaussian elimination and basic properties of matrices and determinants for the Coq proof assistant [17]. Generic design patterns were proposed to define algebraic structures [18]. Formalization and instrumentation of Euclidean spaces also appear to be a new concern for Isabelle/HOL [19]. Automatic decision procedures for floating point arithmetic, such as Gappa [20] are in progress. In general however, none of these recent extensions of theorem provers are able to deal with the properties of interest in this paper.

B. Problem Statement

In the following sections, we use the motivating problem of an aircraft attempting to hold its altitude. The aircraft possesses a collection of processors, each with access to a separate altimeter, which then attempt to agree on the aircraft’s altitude, and use this agreed upon value to actuate the pitch angle of the aircraft to maintain steady, level flight. At any given moment, a limited number of the processors, altimeters and communications channels may be faulty.

Adopting the notation in [21] graph $G = (N, E)$ is comprised of a set $N = \{1, 2, 3, \ldots, n\}$ of processor nodes, with $(i, j) \in E$ edges, where an edge is a symmetric communication channel between a subset of processor nodes $i,j \in N$. We assume that each node $i$ starts with the scalar value $x_i(0)$; the vector with the values of all the nodes at time $t$ is denoted by $\vec{x}(t) = (x_1(t), \ldots, x_n(t))$. For a model that encompasses communication failures between nodes, we have a sequence of time varying graphs $G(t)$ which describe the network topology.

Termination Problem Definition. For the distributed network of $N$ processor nodes described by $G(t)$, with each $i^{th}$ node possessing the initial value $V_i$ of altitudes, does there exist a distributed algorithm $A$ such that all non-faulty processors eventually deciding on a value for the altitude, to within the tolerance $\varepsilon$?

The correctness conditions for termination of the distributed algorithm $A$ is further defined by the following two properties.

Agreement. All non-faulty processors eventually decide on the same value for the altitude, to within a tolerance $\varepsilon$.

Validity. If all non-faulty processors start with the same initial value $V = V$ for the altitude, then all non-faulty processors decide on a value in $[V - \varepsilon, V + \varepsilon]$.

C. Assumptions

We now state two necessary assumptions which we must employ in order to render the termination problem tractable.

Assumption 1 At any given time $t$, there are no more than $3f + 1 < N$ faulty nodes.

In this asynchronous setting, we require one further strong assumption on validation communication between nodes; that is, the notion of bounded communication delay and interconnectivity. For each non-faulty node pair $(j,i)$, following some arbitrary bounded time $B$, there exists a sequence of communications through which node $j$ influences the value held by node $i$.

Assumption 2 There is some $B$ such that for all $k$, the graph $(N, E(kB) \cup E(kB+1) \cup \ldots \cup E(kB+(B-1)))$ is strongly connected.

This assumption proves critical in obtaining a polynomial time convergence bound for the algorithm we outline in the next section.

III. ALGORITHM FOR DISTRIBUTED AGREEMENT WITH BOUNDED COMMUNICATION DELAYS

For asynchronous communication and control (with a bounded delay on communications), a good node can communicate to another good node, but that message may arrive too late to affect the computation at a fixed round. Using the assumptions above (see Assumptions 1,2), we modify a distributed dynamic end-balanced sharing (EBS) multiprocessor scheduling algorithm, that achieves approximately equal makespans, in order to attain approximate agreement among nodes. Specifically, the algorithm that runs on all processor nodes $N$, at time $t$, is outlined in Algorithm 1. As a small technicality, we let $N_i(t) = \{j \neq i : (i,j) \in E(t)\}$, so that $i$ is not considered to be a neighbor of itself.

Algorithm 1. Distributed Agreement over Fault Tolerant Network

Input: Set $N_A$, neighbor nodes of $i$.
Vector $x(0)$; vector of initial values for neighbor nodes.

Constants: time $T_{converge}$.

1. Initialize $x_A = x_i(0), t = 0;$
2. While $t < T_{converge}$ {
3. If $(N_x(t) \neq 0)$ {
4. Node A broadcasts current value $x_A$ to neighbors $j \in N_A(t)$
5. Node A finds neighbor node B with smallest broadcast value: $x_B = \min \{x_j : j \in N_A(t)\}$
6. If $x_B < x_A$
7. Node A sends an offer of $(x_A - x_B)/2$ to Node B;
8. If (Node A receives offers) {
9. Node A sends acceptance to Node C where: $(x_C - x_j)/2 = \max \{x_j - x_A\}, j \in N_A(t)$;
10. Node A sends rejection to all other offer senders;
11. Node A updates values $x_A$:
12. $x_A := x_A + (x_C - x_A)/2$
13. $t := t+1$;
14. If (Node A receives acceptance from node B of offer) {
15. Node A updates its current value $x_A$:
16. $x_A = x_A - (x_A - x_B)/2$;
17. $t := t+1$;
18. Terminate with $x_i(t) = x_A$;
}
19. Terminate with $x_i(t) = x_A$;
}

Intuitively, the algorithm acts to drive the state value of
the processors in the graph \( G_i(t) = (N_A, E_A) \) toward the
average value of all of their initial values. At each iteration of
the while loop, the host process finds the process in the graph
which has the lowest value, and offers to average values with
that process. Simultaneously, the processor receives offers
from other nodes for which it is the lowest value. The node
keeps exchanging its value and making and accepting offers
with its neighbors until all of the values are within the
specified tolerances, the network topology changes or the
maximum time for convergence, derived in the next section,
is exceeded.

IV. CONVERGENCE AND TERMINATION

The correct execution of the above algorithm keeps the
value of \( \sum_{i=1}^{n} x_i(t) \) constant, as nodes only change values
in pairs. Thus, the value reached is a valid average value.
If each \( x_i(t) \) is symmetric, and the bounded delay and
interconnectivity assumption holds, then:

\[
\forall i, \lim_{t \to \infty} x_i(t) = \frac{1}{n} \sum_{k=1}^{n} x_k(0).
\]

Consider the following Lyapunov-like distance metric for
the global state vector \( \vec{x}(t) \) from the desired average:

\[
V(t) = \| \vec{x}(t) - \frac{1}{n} \sum_{i=1}^{n} x_i(0) \|_2^2,
\]

where \( x_i(0) \) is the starting value of the \( i \)th node and \( I_{1 \times n} \)
represents the row vector of ones.

Given a sequence of graphs \( G(t) \) on \( n \) nodes, and an initial vector \( x(0) \), approximate
agreement is reached after some time \( T \), which is the first
time \( t \) after which \( V(\cdot) \) remains smaller than \( \epsilon V(0) \), for
a fixed tolerance \( \epsilon \). Exact agreement would occur as \( \lim \epsilon \to 0 \).
We will use the function \( V(t) \) to prove that the algorithm
is guaranteed to terminate with approximate agreement on
all non-faulty nodes after a worst case execution time of
\( T_n(B, \epsilon) \), over all possible graph sequences \( G(t) \) on \( n \) nodes,
where assumption 2 holds for that particular \( B \).

Theorem 1 \( V(t) \) is nonincreasing for any single accepted
offer.

Proof: We observe from the algorithm that the value of
\( x_i \) changes only upon an accepted offer (accepted either by
the node \( i \) or from the node \( i \)). Consider a single accepted
offer, involving the nodes \( i \) and \( j \) beginning at time \( t_i \) at
the start of the \( k \)th iteration of the while loop and ending
at time \( t_j \). At the end of the time period, both nodes
accepting, the values at both nodes are equal to \( (x_i + x_j)/2 \),
their average value. Thus the Lyapunov function \( V \) decreases by:

\[
V(t_i) - V(t_i + \tau) = x_i^2 + x_j^2 - 2(x_i + x_j)^2 / 2
\]

\[
V(t_i) - V(t_i + \tau) = \frac{1}{2} (x_i - x_j)^2 \geq \frac{1}{2n^2}.
\]

In order to prove that \( V(t) \) is globally nonincreasing
we utilize the following Lemma, adapted from concurrent
programming, which states that any correct parallel execution
can be serialized into a set of sequential executions [22].

Lemma 1 Any set of parallel accepted offers at time \( t \) can
be serialized into a set of sequentially accepted offers.

The serializability property, equivalent to checking that one
regular language is a proper subset of another regular
language which is shuffled using a commutative alphabet, is
PSPACE. For a full proof of this Lemma, see [22].

Theorem 2 \( V(t) \) is globally non-increasing.

Proof: By induction. The value of \( V(t) \) only changes
upon an accepted offer. If a single offer is accepted at time
\( t \), \( V(t) \) is decreasing, as a result of Theorem 1. If \( m \) multiple
offers are accepted in parallel at time \( t \), we use Lemma 1 to
serialize this set into a sequence of \( m \) accepted offers, and
apply Theorem 1 inductively \( m \) times.

Lemma 2 If \( \vec{x} = (x_1, \ldots, x_i, \ldots, x_j, \ldots, x_1) \), \( \vec{y} =
(x_1, \ldots, x_i-1, y_1, x_{i+1}, \ldots, x_j-1, y_j, x_{j+1}, \ldots, x_1) \), where
\( 1 \leq N \) and \( y_i := x_i + [x_j - x_i]/2 \) and \( y_j := x_j - [x_j - x_i]/2 \)
Then \( ||\vec{y}|| \leq ||\vec{x}|| \)

Now, from II-B Assumption 2, we have that the network
is strongly connected. We can define the convergence time
formally as:

\[
T_{G_s(\ldots)}(x(0), \epsilon) = \min \{ t | V(\tau) \leq \epsilon V(0), \forall \tau \geq t \}
\]

where \( G_s(\ldots) \) is a sequence of graphs \( G(t) \) on \( n \) nodes
which possess the initial vector \( x(0) \). Note that \( V(t) \) does not
increase above \( \epsilon V(t) \) after this time. The worst case
execution time (WCET) becomes the maximum value of
\( T_{G_s(\ldots)}(B, \epsilon) \) over all initial vectors on all graph sequences
of \( n \) nodes that satisfy Assumption 2 for a particular value
of \( B \). We now demonstrate that the algorithm possesses a
polynomial time convergence bound \( T_{conv}(B, \epsilon) \propto n^3 \).

Theorem 3 \( \forall n, \epsilon \in [0, 1], \exists \beta \geq 0 | T_n(B, \epsilon) \leq \beta Bn^3 \log \frac{1}{\epsilon} \)

Proof: By induction. Without loss of generality (WLOG), take \( \sum_{i=1}^{n} x_i(0) = 0 \). \( \therefore \sum_{i=1}^{n} x_i(0) = 0 \Rightarrow \min_i x_i(0) \leq 0 \).

If we can show that after \( (k + 1) \) rounds of length \( B \), the
value of \( V(t) \) is strictly less than that at the \( k \)th round of
length \( B \), by a factor of \( (1 - \frac{1}{2n^2}) \), for every natural number
\( k \), then by using Theorem 2, we have the correct result by
implication, as follows:

If

\[
\frac{V((k+1)B)}{V(kB)} \leq (1 - \frac{1}{2n^2})
\]

this means \( V(t) \) decreases by a factor of \( 1 - \frac{1}{2n^2} \) for
every bounded time \( B \), then \( V(t) \) decrease by a factor of
\( \Theta(1) \) in \( Bn^3 \) steps, that is, the rate of decrease of \( V(t) \)
is then asymptotically bounded from above and below by
a constant. Thus, \( V(t) \) first becomes less than \( \varepsilon V(0) \) at 
\( T_{\text{conv}} \propto O(Bn^3 \log \frac{1}{\varepsilon}) \), and we terminate the algorithm at this point. Theorem 2 proves that \( V(t) \) is non-increasing, and it remains below this value for all subsequent time \( t \geq T_{\text{conv}} \).

To prove Equation 6 is true, we reason as follows. Let us discretize the time bound \( B \) into intervals \( 0, 1, \ldots, B - 1 \). We argue that there exists a \( t \) in this interval at which at least one offer is accepted, thereby reducing \( V(t) \) by \( \frac{1}{2n^2} V(0) \), at least. Again, WLOG, we assume that \( \max_i |x_i(0)| = 1 \), so \( \forall i, x_i(0) \in [-1, 1] \Rightarrow V(0) \leq n \). Hence, the \( \max_i (x_i(0)) = \min_i (x_i(0)) \geq \frac{1}{2} \). Thus, \( \{3u, v, v, \in R | v - u \geq \frac{1}{2}\} \), so that the all nodes \( i \in N \) can be partitioned into disjoint subsets \( U_\ldots = \{i \in N | x_i(0) \leq a\} \) and \( U^+ = \{i \in N | x_i(0) \geq b\} \). Since the sequence of graphs \( \bigcup_{p=0, \ldots, B-1} E(\eta) \) is connected, due to Assumption 2, \( \exists \zeta \in 0, 1, \ldots, B - 1 \) where \( \zeta \) is the first time at which there is a communication between nodes \( (i, j) \), \( i \in U_\ldots, j \in U^+ \), resulting in an offer from \( j \) to \( i \). Note that until \( \zeta \) there is no communication between nodes in \( U_\ldots \) and \( U^+ \), so that \( x_q(\zeta) < a \), \( \forall q \in U_\ldots \) and \( x_q(\zeta) > b \), \( \forall q \in U^+ \); thus \( x_i(\zeta) \geq b \) and \( x_i(\zeta) \leq a \).

There are two possible actions: \( i \) accepts the offer from \( j \), or it accepts a higher offer. Regardless of which, a node from \( U_\ldots \) accepts an offer from a node in \( U^+ \) at \( \zeta \). Thus, in the worst case we have, by Theorem 1:

\[
V(0) - V(\zeta) \geq \frac{(x_i - x_j)^2}{2} \tag{7}
\]

using the fact that \( x_j - x_i \geq v - u \) we have

\[
V(0) - V(\zeta) \geq \frac{(v - u)^2}{2} \tag{8}
\]

\[
V(0) - V(\zeta) \geq \frac{1}{2n^2} \tag{9}
\]

At every other time prior, \( V \) is nonincreasing, and \( V(0) \leq n \) so:

\[
V(B) \leq V(0) - \frac{1}{2n^2} \tag{10}
\]

\[
V(B) \leq V(0)(1 - \frac{1}{2n^2}) \tag{11}
\]

By inductively repeating this argument over the interval \( kB, \ldots, (k+1)B \), we establish the desired result:

\[
\frac{V(k+1)B}{V(kB)} \leq (1 - \frac{1}{2n^3}) \tag{12}
\]

Note that this is a nonlinear algorithm which possesses a polynomial time convergence bound \( T_{\text{conv}}(B, \varepsilon) = \beta Bn^3 \log(\frac{1}{\varepsilon}) \propto n^3 \).

However, correctness of a theoretical formal model does not ensure the correctness of its code implementation, as different implementations of the same model are often possible. The process of translating a model into code instructions requires semantic knowledge concerning the specified model, and optimizations may be made for performance and efficiency. Numerical approximations in the coding process, such as the discretization of continuous states, the conversion to floating point representations, lacking of a strongly typed language, the substitution of interval equality for exact equality as well as the processor dependent nature of computation all result in deviations from theoretically obtained results.

V. HOARE LOGIC AND CODE ANNOTATIONS

Our current approach does not consist in automating the proof of stability, but rather, given a stability proof, to check the proof automatically. As was suggested in [6], [7], [8], we consider a line-by-line annotation of the code, allowing a Hoare-like reasoning approach to the program.1 The main approach to Lyapunov-like stability and termination arguments is to relate quadratic invariants and affine or linear combinations of variables under the ellipsoid affine combination theorem. Stability can then expressed as a predicate stating that the system state remains in a given ellipsoid. An instruction \( S: \tilde{y} = G(x) \) is annotated:

\[
\{\tilde{x} \in E_\lambda \} \tilde{y} = G(x) \{\tilde{y} \in E_\lambda \} \tag{13}
\]

where the pre- and post- conditions are predicates expressing that the vectors \( x \) and \( y \) are both contained in ellipsoid \( E_\lambda \).

A. Defining Quadratic Invariants as Code Annotations

The ANSI/ISO C Specification Language (ACSL) [23] allows its user to specify the properties of a C program within comments, in order to be able to formally verify that the implementation respects these properties. This language was proposed as part of the Fram-C platform [24], which provides a set of tools to reason on both C programs and their ACSL annotations. ACSL offers the means to extend its internal logic with any user-defined theory, i.e., types, constructors, functions, predicates and axioms.

In order to axiomatize ACSL to fit our needs, we wish to express ellipsoid-based Hoare triples over C code. We first outline the axiomatisation of temporal elements in ACSL. Then we develop the Hoare triple annotations in ACSL.

B. Temporal Relations in ACSL predicates

\[
//@ logic real Time_to_Receive_Message;
//@ logic real Time_to_Send_Message;
//@ logic real Time_to_Ack_Acceptance;
\]

The above ACSL functions return the maximum time taken in order to send, receive and acknowledge a message, respectively, from the host node to the node furthest away.

\[
//@ logic real matrix_of_array{t}(float Vx, integer row, integer col);
//@ logic real float time_lapsed();
\]

The previous ACSL predicates enable the expression of a matrix and of an elapsed time counter, since the code begins execution, with respect to the local clock.

Complex constructions or relations can be defined as uninterpreted predicates, i.e., with no associated axiom. The following predicate is meant to express that vector \( x \) belongs to the ellipsoidal reach set \( E_\rho \):

\[
//@ predicate in_ellipsoid (matrix P, vector x);
//@ predicate Lyapunov_condition (vector x, vector y);
\]

Similarly, we can declare the following timing predicate:

\[
//@ predicate Time_to_Ack (Real t, t_before_send);
\]

which evaluates to true if the time taken to send the message falls within the required bound.

1 or equivalently a limited-depth Dijkstra weakest precondition.
C. Linear Algebra Code Example Annotations: If Statement and While Loop

A function contract can be expressed as a Hoare triple for a whole function. The key word `requires` is used to introduce the pre-conditions of the triple, and the key word `ensures` is used to introduce its post-conditions. Dealing with a low-level language has its disadvantages: we need to deal with memory issues. In general, we want all functions to be called with valid pointers as arguments, i.e., valid array and therefore valid matrices. The built-in ACSL predicate valid enables this functionality. The following is a reachability annotation, which illustrates how the convergence of the Lyapunov-like function in the model is transferred into the code to assure that it terminates, even though the returned value may not reach a fixed point. For any accepted offer, there is always a pair of values which are averaged, bringing the program closer to termination.

```c
//@ requires IsValidRange[x, n];
//@ ensures abs( result - average(x,0,n)) < epsilon; */
float consensus.protocol(int sock, int j, float xA, float x[0], int n, int T_conv_cell, float epsilon)

The previous box establishes the contract of what is required and ensured through a single loop execution.

The following is the actual loop invariant, which holds throughout all executions, as well as the strictly decreasing loop variant which guarantees termination.

```c
//@ loop invariant 0<=time_round < T_conv_cell;
//@ loop invariant x*x < x_0*x_0;
//@ loop variant (ceil(2n^3*Vx_0 - Vx) - time_round);
//@ while (time_round < T_conv_cell && Vx > epsilon*Vx_0)
Update values x_i and x_j in state vector x via broadcast;
Average values xA and x_j in state vector x, calculate Vx;
time_round++;
//@ end while*/ return x_i;
```

VI. DISCHARGING PROOF OBLIGATIONS

A. Generation

The Frama-C toolset, i.e. the Jessie plug-in and the Why platform, are used to generate verification conditions (VCs) from the annotated code described in the previous section. Following the work in [26] we choose PVS as the output language in which these VCs are generated. They then take the forms of theorems which, when proven, ascertain the validity of the annotations with respect to the code.

Some of the symbols used in the previous section have not been given meaning, that is, they express properties that have not been described in ACSL. To fill in this semantic gap, we use theory interpretation. This technique enables one to map these uninterpreted symbols in ACLS to existing PVS types and functions. For example:

\[ \text{lyapunov} \text{(vector } x, \text{ vector } y) = y^*y^* x \]

\[ \text{Time to Receive Message (Reals } t, t_{beforereceive}) = \]

\[ t \geq t_{beforereceive} + \delta_{message latency} \]

\[ \text{Time to Send Offer (Reals } t, t_{beforeoffer}, n) = \]

\[ t \geq (n-1)*(\delta_{message latency}) + t_{sort} + t_{beforeoffer} \]

Note, however, that the notion of time itself is not implemented in PVS. Rather, this enables us to derive time bounds on execution time, assuming other time bounds on lower level primitives.

B. Examples

A proof obligation relative to a loop invariant takes the following form:

\[ \text{in_ellipsoid(matrix } I, \text{ vector } x \text{before}) \]

\[ \text{AND } x \text{after= F(x \text{before}) IMPLIES} \]

\[ \text{in_ellipsoid(matrix } I, \text{ vector } x \text{after}) \]

where function F has been obtained from the semantics of one loop execution, and generated automatically by the toolset. Because all the operations carried out on x in the loop are linear, we can use the linear algebra library developed in PVS to discharge that kind of property, as was done in [26].

The loop variant described in the previous section yields, in essence, 2 proof obligations: that the variant provided be non-negative, and that it also be strictly decreasing at each loop execution. The use of ghost variables in Frama-C enables us to track the value of the Lyapunov function which, for all intents and purposes is our loop variant. At this point the proof of termination has not yet been completed in PVS.

VII. CONCLUSIONS

This paper draws from the prior work expressed in [6], [7], [8], employing the idea of carrying a high level systems proof down its implementation, aiming at making the certification of the software a simple matter of checking the correctness of the annotations, with proof techniques known in control theory. In [26], a more concrete annotation and checking process is described. We have develop a polynomial time theoretical bound on the convergence time \(T_{conv} \propto n^3\) of the nonlinear agreement algorithm used. This proof structure is
then translated into ACSL annotations in the C code used to implement the distributed algorithm, and can then be used to automatically generate formal proof obligations via Frama-C. Thus, the verification effort spans from the high-level control theory to the low-level implementation as a C program. The combination of tools used to carry this out form a coherent story, with Frama-C taking care of the low-level C implementation details leaving the core linear algebra problems to be handled by interactive proof in PVS.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the following funding sources: NSF Grant CNS - 1135955 CPS: Medium: Collaborative Research: Credible Autocoding and Verification of Embedded Software (CrA VES), NSF Grant 1136008, ARO MURI W911NF-11-1-0046, AFRL CertaAMOR program, the Dutton/Ducatte Professorship at Georgia Tech, and NASA Cooperative Agreement NNL09AA00A, activity 2736.

REFERENCES


