Proof-carrying auto-coded control software

Eric Feron
Dutton/Ducoffe Professor of Aerospace Engineering
Georgia Institute of Technology
Take-Home Message

Control system proofs can be used to formally support semantics of embedded control systems.
Outline

• A simple control example
• Stability and performance analyses: Why go beyond specs and into implementation?
• Control system design chain
• What proofs or what system representations?
• Proofs in graphical system world
• Hoare logic and partial correctness
• Analysis of controller implementation
• Closed-loop system analysis
• Links to health management
• Tool development
A simple control example

\[
\begin{align*}
\frac{d}{dt} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u, \quad x(0) = x_0, \dot{x}(0) = \dot{x}_0 \\
y &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix}.
\end{align*}
\]
A simple control example (Ct’d)

\[ \tilde{y}(t) = SAT(y(t)), \]

\[ u(s) = 128 \frac{s + 1}{s + 0.1} \frac{s}{5} + 1 \tilde{y}(s), \]

Step response
Controller implementation

\[ \tilde{y}(t) = \text{SAT}(y(t)), \]

\[ u(s) = 128 \frac{s + 1}{s + 0.1} \frac{s}{50 + 1} \tilde{y}(s), \]

\[ \frac{d}{dt} x_c = \begin{bmatrix} -50.1 & -5.0 \\ 1.0 & 0.0 \end{bmatrix} x_c + \begin{bmatrix} 100 \\ 0 \end{bmatrix} \text{SAT}(y) \]

\[ u = -[564.48 \ 0] x_c + 1280 \text{SAT}(y). \]
Controller implementation

\[
x_{c,k+1} = \begin{bmatrix} 0.499 & -0.050 \\ 0.010 & 1.000 \end{bmatrix} x_{c,k} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{SAT}(y_k)
\]

\[
u_k = -\begin{bmatrix} 564.48 & 0 \end{bmatrix} x_{c,k} + 1280 \text{SAT}(y_k)
\]

Discrete time implementation

100Hz
Control system as seen by control engineers

System data → System Identification/Validation → Controller design → Controller analysis

Invalidated Controller

Not good to go

Good to go → Verification and Validation → Executable

Compiler → Source code

Matlab/Simulink/Real-time Workshop

MatrixX

Picture 2 code (UTC)
Code-level analyses of control software

- Most significant contribution is from Patrick Cousot’s group at Ecole Normale Superieure, Paris.
- Abstract interpretation aims at capturing semantics of programs
- Most important application is ASTREE analyzer for Airbus A380 control code.
- From Feret, “Static Analysis of Digital Filters”, 2004 (also with ASTREE).

A simplified second order filter relates an input stream $E_n$ to an output stream defined by:

$$S_{n+2} = aS_{n+1} + bS_n + E_{n+2}.$$  

Thus we experimentally observe, in Fig. 4, that starting with $S_0 = S_1 = 0$ and provided that the input stream is bounded, the pair $(S_{n+2}, S_{n+1})$ lies in an ellipsoid. Moreover, this ellipsoid is attractive, which means that an orbit starting out of this ellipsoid, will get closer of it. This behavior is explained by Thm. 5.

Fig. 4. Orbit.
A Paradigm Shift Enabled by Specification Analyses

(auto) Code analyzer

Controller Specifications (+proof) → Autocoder (third party) → (auto)-code → Code analyzer (proof) → Go/no Go

Credible autocoder (a la Rinard)

Controller Specifications +proof → Credible autocoder (third party) → Documented (auto)-code → Proof checker (certification Authority) → Go/no-go

Georgia Tech
College of Engineering
School of Aerospace Engineering

Daniel Guggenheim
A hierarchical control system V&V ecosystem

High-level system specification

(Control) Credible discrete-time system specification

(control) credible intermediate Code (eg esterel)

(control) source code (array variables)

Credible Executable code
Desirable attributes of “system proofs”

• Must be expressive enough to tell nontrivial statements about system
• Must speak the language of system representation
• Eg: “IEEE TAC proofs” written in natural language (one wonders…), “Simulink proofs” expressed in Simulink, “Program proofs” expressed in formal languages.
• Must be “elementary enough” to be easily checked wherever necessary.
Back to the Example

The control-systemic way
\[
\frac{dx_c}{dt} = \begin{bmatrix} -50.1 & -5.0 \\ 1.0 & 0.0 \end{bmatrix} x_c + \begin{bmatrix} 100 \\ 0 \end{bmatrix} \text{SAT}(y)
\]
\[
u = -\begin{bmatrix} 564.48 & 0 \end{bmatrix} x_c + 1280 \text{SAT}(y).
\]

Assume the controller state is initialized at \( x_{c,0} = 0 \)
What range of values could be reached by the state \( x_{c,k} \) and the control variable \( u_k \)?

There is a variety of options, including computation of -1 norms.
A Lyapunov-like proof (from Boyd et al., Poola):
\[
E_P = \{ x \in \mathbb{R}^2 \mid x^T P x \leq 1 \}.
\]
\[
P = 10^{-3} \begin{bmatrix} 0.6742 & 0.0428 \\ 0.0428 & 2.4651 \end{bmatrix}.
\]

The ellipsoid is invariant. None of the entries of \( x \) exceeds 7 in size.
The initial IEEE TAC (well Soviet Mathematics Doklady, vintage 1960) proof

∀t, xTPx ≤ 1 is equivalent to x(t)TPx(t) ≤ 1 ⇒ x(t + dt)TPx(t + dt) ≤ 1

Or xTPx + dt(xT(ATP + PA)x + 2xTPBw) ≤ 1 whenever xTPx ≤ 1 and w2 ≤ 1

True if there exists μ such that xTPx + dt(xT(ATP + PA)x + 2xTPBw) − μxTPx − (1 − μ)w2 < 0, (*) a tautology.

Indeed a linear combination of (*) and xTPx ≤ 1 and w2 ≤ 1 yields the desired property.

P that works is P = 10−3 \begin{bmatrix} 0.6742 & 0.0428 \\ 0.0428 & 2.4651 \end{bmatrix}, with μ = 0.9991 and tautology

(*) is 10−3 \begin{bmatrix} x \\ w \end{bmatrix}^{T} \begin{bmatrix} -0.5044362 & -0.0135878 & 0.3374606 \\ -0.0135878 & -0.0003759 & 0.00909 \\ 0.3374606 & 0.00909 & -0.2258 \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix} ≤ 0.
The Simulink Continuous Time Formal Proof?

Current Simulink Implementation for Online tests

\[ x_1(0) = 0 \]

\[ x_2(0) = 0 \]
Simulink, Discrete Time Formal Semantics

\[
10^{-3} \begin{bmatrix}
-0.5044362 & -0.0135878 & 0.3374606 \\
-0.0135878 & -0.0003759 & 0.00909 \\
0.3374606 & 0.00909 & -0.2258
\end{bmatrix}
\]

Quadratic form

\[
x(t) = 10^{-3} \begin{bmatrix}
0.6742 & 0.0428 \\
0.0428 & 2.4651
\end{bmatrix} x(t)
\]

\[
x_1(0) = 0
\]

\[
x_2(0) = 0
\]
Simplified Matlab program

1: A = [0.4990, -0.0500; 0.0100, 1.0000];
2: C = [-564.48, 0];
3: B = [1;0]; D = 1280;
4: x = zeros(2,1);
5: while 1
6: y = fscanf(stdin, "%f");
7: y = max(min(y,1),-1);
8: u = C*x + D*y;
9: fprintf(stdout, "%f
", u);
10: x = A*x + B*y;
11: end
Critical step

\[ \{ x \in \mathcal{E}_P, \ y^2 \leq 1 \} \Rightarrow \{ Ax + By \in \mathcal{E}_P, \ y^2 \leq 1 \}. \]

is true for the particular instances of \( A, B, P \) (ellipsoid invariance condition)/

-Use tautology for linear combination of inequalities

\[ \forall (x, y) \ (Ax + By)^T P(Ax + By) - 0.0009x^T P x - 0.9991y^2 \leq 0. \]

\[
\{ x \in \mathcal{E}_P, \ y^2 \leq 1, \ (Ax + By)^T P(Ax + By) - 0.0009x^T P x - 0.9991y^2 \leq 0 \}
\Rightarrow \{ Ax + By \in \mathcal{E}_P, \ y^2 \leq 1 \},
\]
1: \[ A = [0.4990, -0.0500; 0.0100, 1.0000] \];
2: \[ C = [-564.48, 0] \];
3: \[ B = [1;0]; D=1280 \]
4: \[ x = \text{zeros}(2,1); \]
5: \[ \text{while } 1 \]
6: \[ y = \text{fscanf(stdin,"%f")} \]
7: \[ y = \text{max(min(y,1),-1)}; \]
8: \[ u = C*x+D*y; \]
9: \[ \text{fprintf(stdout","%f\n",u)} \]
10: \[ x = A*x + B*y; \]
11: \[ \text{end} \]
Other important questions

- Other properties.
- Verifying closed-loop system.
- Incorporating existing computer-aided system analysis tools.
- Formal comment writing.
- Floating point computations.

- Many controllers are more complex than a simple lead-lag controller – nonlinear? Adaptive? Receding horizon???
Adding the controlled plant as part of the controller’s semantics
Software Health Management / Connexion to Fault Detection/Isolation

<table>
<thead>
<tr>
<th>Line of code;</th>
</tr>
</thead>
<tbody>
<tr>
<td>% Some representation of the plant dynamics</td>
</tr>
<tr>
<td>% boolean formula (code state, plant state) = true</td>
</tr>
<tr>
<td>Line of code;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line of code;</th>
</tr>
</thead>
<tbody>
<tr>
<td>If boolean formula(code state, plant state? ) == false then ABORT</td>
</tr>
<tr>
<td>Line of code;</td>
</tr>
</tbody>
</table>

From code comment …

… to code observer

Code observers monitor code properties at specific points via functions of code variables. Abort execution if observer returns “false”.

Does not extend to closed-loop code properties:

State of physical system unknown.
Fault detection/isolation

Quadratic form

\((x_c, x_p)\)
Fault detection/isolation

Estimator

Physical state Estimate
(set? Distribution?)

<1?

Quadratic form

P
Software Health Management / Connection to Fault Detection/Isolation

From code comment …

Line of code;
% Some representation of the plant dynamics
% boolean formula (code state, plant state) = true
Line of code;

To code/plant observer,
Incl. fault detection

Line of code;
physical plant state estimation step
If boolean formula(code state, estimated plant state ) == false then ABORT
Line of code;
Online test and FDI semantics....

Line of code;
physical plant state estimation step;
If boolean formula(code state, estimated plant state ) == false then ABORT
Line of code;

From code/plant observer/FDI,
Incl. fault detection ...

Line of code;
%formal physical representation of plant + fault modes
physical plant state estimation step;
%Boolean formula (code state, estimator state, faulty plant state)=true
If boolean formula(code state, estimated plant state ) == false then ABORT
Line of code;

... to code/plant observer/ FDI with semantics
Using third party system analysis tools to extract semantics automatically

- Sometimes, control system properties can be established via automated analysis; “Specification-level static analysis”
  - $\mu$-tools, IQC-$\beta$ (Balas, Megretski, Packard, Rantzer)
- Can be used to automatically generate earlier diagrams.
• ANSI/ISO C Specification Language (ACSL) can be used to formally comment C programs and can be handled by Frama-C.

• Start from initial, engineering friendly specification language: Scilab is free, open-source MATLAB-like language. Also comes with Simulink-like graphical system specification system.
SUPPORTED SCILAB INSTRUCTIONS

- If… Then… Else
- While / For loops
- Assignments: exclusively linear operations

\[ \forall j : \sum_{i=1}^{n} A_i var_i \]
- \( A_i \) are constant matrices
- The point? Being able to annotate:

\[
\begin{bmatrix}
var_1 \\
var_2 \\
\vdots \\
var_n
\end{bmatrix} \in \mathcal{E}_P
\]

\[ \forall j : \sum_{i=1}^{n} A_i var_i \]

\[
\begin{bmatrix}
var_1 \\
var_2 \\
\vdots \\
var_n \\
var_j
\end{bmatrix} \in \mathcal{E}_{P'}
\]
\[ j \in \{1, 2, \ldots, n\} \]

\[
\left\{ \begin{bmatrix}
    \text{var}_1 \\
    \text{var}_2 \\
    \vdots \\
    \text{var}_n
\end{bmatrix} \in \mathcal{E}_P \right\}
\]

\[ \text{var}_j := \sum_{i=1}^{n} A_i \text{var}_i \]

\[
\left\{ \begin{bmatrix}
    \text{var}_1 \\
    \text{var}_2 \\
    \vdots \\
    \text{var}_n
\end{bmatrix} \in \mathcal{E}_{P'} \right\}
\]

\[ P' = (M_1 P^{-1} M_1^T)^{-1} \]

\[
M_1 = \begin{bmatrix}
    I_1 & I_2 & \vdots & A_j & \ldots & A_n \\
    \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\
    A_1 & A_2 & \ldots & A_j & \ldots & A_n \\
    \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\
    \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\
    \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\
\end{bmatrix}
\]
\[ j \notin \{1, 2, \ldots, n\} \]

\[
\begin{bmatrix}
    \text{var}_1 \\
    \text{var}_2 \\
    \vdots \\
    \text{var}_n
\end{bmatrix} \in \mathcal{E}_P
\]

\[
\text{var}_j := \sum_{i=1}^{n} A_i \text{var}_i
\]

\[
\begin{bmatrix}
    \text{var}_1 \\
    \text{var}_2 \\
    \vdots \\
    \text{var}_n
\end{bmatrix} \in \mathcal{E}_P, \text{var}_j \in \mathcal{E}_{P''}
\]

\[
P'' = (M_2 P^{-1} M_2^T)^{-1}
\]

\[
M_2 = \begin{bmatrix}
    A_1 & A_2 & \cdots & A_n
\end{bmatrix}
\]
TRANSLATION PATTERN

\[
\begin{align*}
\text{\{ } & \left[ \begin{array}{c} x_c^T \\ y_c^T \end{array} \right]^T \in \mathcal{E}_P \\
\text{\}} \\
u \ := \ C_c x_c + D_c y_c
\end{align*}
\]

\[
\begin{align*}
\text{\{ } & \left[ \begin{array}{c} x_c^T \\ y_c^T \end{array} \right]^T \in \mathcal{E}_P, u \in \mathcal{E}, \left( \begin{array}{c} C_c \\ D_c \end{array} \right)^{-1} \left[ \begin{array}{c} C_c \\ D_c \end{array} \right]^T \}^{-1}
\end{align*}
\]

\[
\begin{align*}
\text{\{ } & \left[ \begin{array}{c} x_c^T \\ y_c^T \end{array} \right]^T \in \mathcal{E}_P \\
temp_1 \ := \ C_c x_c
\end{align*}
\]

\[
\begin{align*}
\text{\{ } & \left[ \begin{array}{c} x_c^T \\ y_c^T \end{array} \right]^T \in \mathcal{E}_P, temp_1 \in \mathcal{E}, \left( \begin{array}{c} C_c \\ 0 \end{array} \right)^{-1} \left( \begin{array}{c} C_c \\ 0 \end{array} \right)^T \}^{-1}, temp_1 \ := \ C_c x_c
\end{align*}
\]

\[
\begin{align*}
\text{\{ } & \left[ \begin{array}{c} x_c^T \\ y_c^T \end{array} \right]^T \in \mathcal{E}_P, temp_1 \in \mathcal{E}, \left( \begin{array}{c} C_c \\ 0 \end{array} \right)^{-1} \left( \begin{array}{c} C_c \\ 0 \end{array} \right)^T \}^{-1}, temp_1 \ := \ C_c x_c, \\
temp_2 \ := \ D_c y_c
\end{align*}
\]

\[
\begin{align*}
\text{\{ } & \left[ \begin{array}{c} x_c^T \\ y_c^T \end{array} \right]^T \in \mathcal{E}_P, temp_2 \in \mathcal{E}, \left( \begin{array}{c} 0 \\ D_c \end{array} \right)^{-1} \left( \begin{array}{c} 0 \\ D_c \end{array} \right)^T \}^{-1}, temp_2 \ := \ D_c y_c \\
u \ := \ temp_1 + temp_2
\end{align*}
\]

\[
\begin{align*}
\text{\{ } & \left[ \begin{array}{c} x_c^T \\ y_c^T \end{array} \right]^T \in \mathcal{E}_P, u \in \mathcal{E}, \left( \begin{array}{c} C_c \\ D_c \end{array} \right)^{-1} \left( \begin{array}{c} C_c \\ D_c \end{array} \right)^T \}^{-1}
\end{align*}
\]
USING TRANSLATION TEMPLATES

\[ \{ x \in \mathcal{E}_P \; \&\& \; ... \} \Rightarrow \{ x \in \mathcal{E}_P \; \&\& \; ... \} \]

\[ C = A \times x; \]

\[ \text{float** mult_vect(float** a, int rowA, int colA, float** b, int rowB, float** c) } \]
\[ \{ \]
\[ \text{int i=0, k=0;} \]
\[ \text{while (i < rowA) } \{ \]
\[ \qquad \text{k=0;} \]
\[ \qquad \text{c[i][0]=0;} \]
\[ \qquad \text{while (k<colA) } \{ \]
\[ \quad \quad \text{c[i][0] = c[i][0] + a[i][k]*b[k][0];} \]
\[ \quad \quad \text{k = k+1;} \]
\[ \quad i= i+1; \]
\[ \} \]
\[ \text{return c;} \]

Daniel Guggenheim
School of Aerospace Engineering
\{ x \in \mathcal{E}_P \ \&\& ... \} \\
float** mult vect(float** A, int rowA,int colA, float** x, int rowx, float **c ) 
{
    int i=0, k=0;
    \begin{cases}
    c_{0..i-1} \in \mathcal{E}(A_{i-1,colA}P^{-1}A_{i-1,colA}^T)^{-1} \\
    \end{cases}
    \begin{cases}
    c_{0..i} \in \mathcal{E}(A_{i,colA,k-1}P^{-1}A_{i,colA,k-1}^T)^{-1} \\
    \end{cases}
    while (i <rowA) {
        k=0;
        c[i][0]=0;
        \begin{cases}
        c_{0..i} \in \mathcal{E}(A_{i,colA,k}P^{-1}A_{i,colA,k}^T)^{-1} \\
        \end{cases}
        while (k<colA) {
            c[i][0] = c[i][0] + A[i][k]*x[k][0];
            k = k+1;
        }
        i = i+1;
    }
    return c;
}
ANNOTATION LANGUAGE

• On the Scilab side:
  – Need to be able to express:
    • Block Matrices
    • Matrix Inverse
    • Belonging to an Ellipsoid
    • Basic predicates on variables (bounds)

• On the C side
  – Same requirements of expressivity, but annotations must be readable by certification software.
  – We express everything in Frama-c
Back End: Verification of Code Semantics

Controller Specifications +proof

Credible autocoder (third party)

Documented (auto)-code (user)

Proof checker (certification Authority)

Go/no-go
FRAMA-C

• Hoare Style annotation language.

• Can interface with manual and automated proving software (e.g., PVS).

• Has the required expressivity.
INTERFACING WITH VERIFICATION TOOLS

• We use Frama-C because it can generate verification conditions for various pieces of software

• The interface with PVS allows us to use the work done at the National Institute of Aerospace on linear algebra.
Conclusion

- Stability and performance proofs of control systems are fundamentally compatible with formal static analysis and verification methods.
- Most safety-critical code is for health monitoring / fault detection-isolation. Currently developing framework for including FDI code in “credible” code toolset.
- We are developing a comprehensive safety-critical software autocoding suite with explicit semantics expression at all levels from system specification to source code.
Acknowledgements

• Army Research Office
• National Science Foundation
• NASA/Northeastern University
• Dutton/Ducoffe Professorship at Georgia Tech