Applications of control to formal software semantics

Eric Feron
Aerospace Engineering
Georgia Tech

(with Fernando Alegre, Alwyn Goodloe, Romain Jobredeaux, Tim Wang + a growing number of guests of Alwyn's at NIA)

July 15, 2010
NSV 3

Take-Home Message

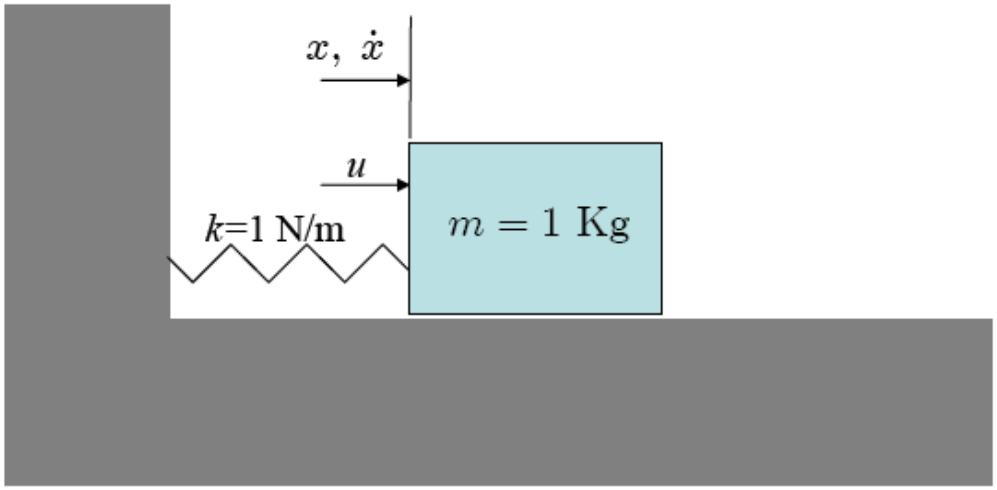
We've got proofs for our control systems, so let's use them beyond cool research papers, inside software

(Probable NSV interpretation:
Yet another guy who's discovered static analyzers and formal proofs)

Outline

- A simple control example
- Stability and performance analyses
- Why code-level analyses?
- Hoare logic and partial correctness
- Analysis of controller implementation
- Closed-loop system analysis

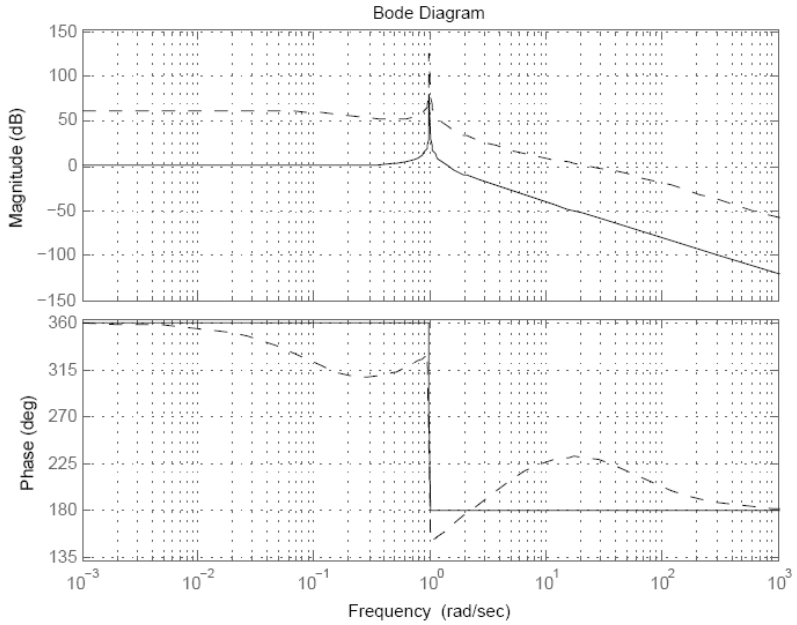
A simple control example



$$\frac{d}{dt}\begin{bmatrix} x \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u, \quad x(0) = x_0, \dot{x}(0) = \dot{x}_0$$

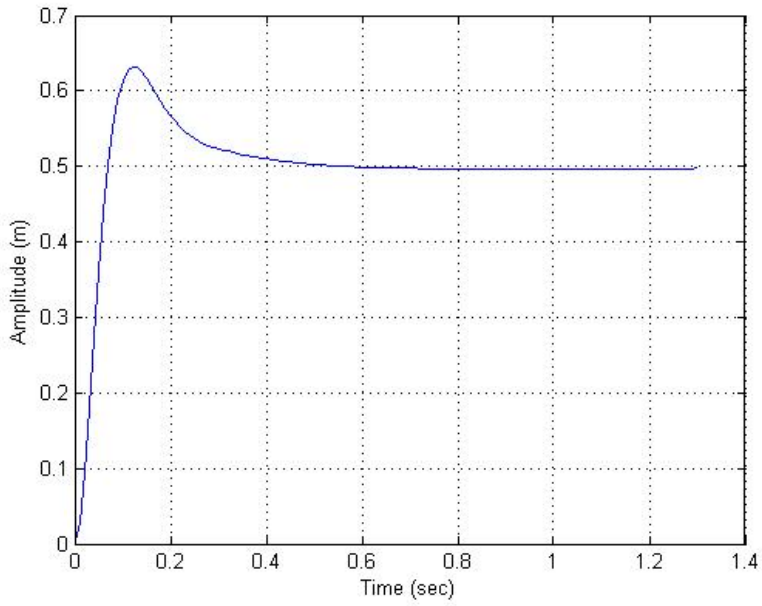
$$y = [1 \ \ 0]\begin{bmatrix} x \\ \dot{x} \end{bmatrix}.$$

A simple control example (Ct'd)



$$\tilde{y}(t) = \mathbf{SAT}(y(t)),$$
$$u(s) = 128\frac{s+1}{s+0.1}\frac{s/5+1}{s/50+1}\tilde{y}(s),$$

Step response



Controller implementation

$$\tilde{y}(t) = \mathbf{SAT}(y(t)),$$
$$u(s) = 128 \frac{s+1}{s+0.1} \frac{s/5+1}{s/50+1} \tilde{y}(s),$$

$$\begin{aligned} \tfrac{d}{dt} x_c &= \begin{bmatrix} -50.1 & -5.0 \\ 1.0 & 0.0 \end{bmatrix} x_c + \begin{bmatrix} 100 \\ 0 \end{bmatrix} \mathbf{SAT}(y) \\ u &= -\begin{bmatrix} 564.48 & 0 \end{bmatrix} x_c + 1280\, \mathbf{SAT}(y). \end{aligned}$$

State-space realization

$$\begin{aligned} x_{c,k+1} &= \begin{bmatrix} 0.499 & -0.050 \\ 0.010 & 1.000 \end{bmatrix} x_{c,k} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \mathbf{SAT}(y_k) \\ u_k &= -\begin{bmatrix} 564.48 & 0 \end{bmatrix} x_{c,k} + 1280\, \mathbf{SAT}(y_k) \end{aligned}$$

Discrete time Implementation 100Hz

Controller 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\n",u);
10:  x = A*x + B*y;
11: end
```

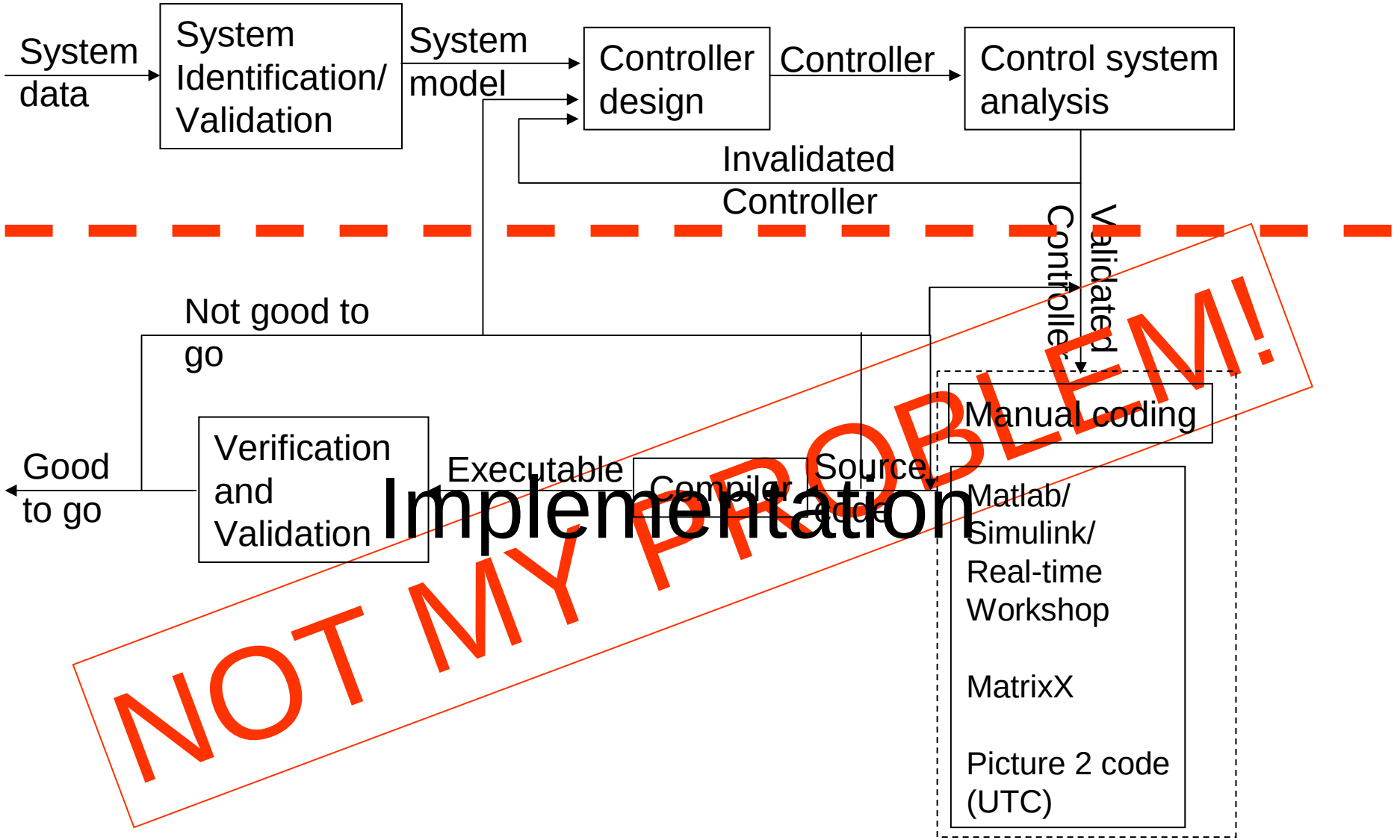
MATLAB

Controller Program

```
1: int main(int argc, char *argv[])
2: {
3:   double *x[2], *xb[2], y, u;
4:   x[0] = 0;
5:   x[1] = 0;
6:   while(1){
7:     fscanf(stdin,"%f",&y);
8:     if (y >1){
9:       y=1
10:    }
11:    if (y<-1){
12:      y=-1
13:    }
14:    u = -564.48*x[0]+1280*y;
15:    fprintf(stdout,"%f\n",u);
16:    xb[0]=x[0];
17:    xb[1]=x[1];
18:    x[0]:= 0.4990*xb[0]-0.0500*xb[1]+y;
19:    x[1]:= 0.01*xb[0]+xb[1];
20:  }
21:}
```

C

Control system as seen by control engineers



Code-level analyses of control software

- Significant contribution from Patrick Cousot's group at Ecole Normale Superieure, Paris and NYU.
- Abstract interpretation supports 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.

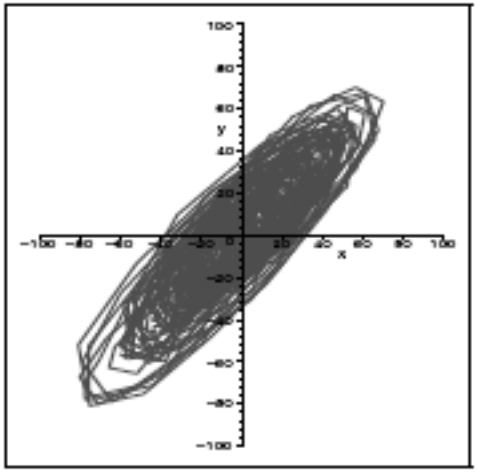


Fig. 4. Orbit.

Making sense out of computer programs

- Consists of asking: “Where can the program go? Does it do what it’s supposed to do?”
- Well-known limits to what can be done.
- Several options are available: “Formal Methods”
  - Model checking
  - Abstract interpretation

Desirable attributes of “program proofs”

- Must be expressive enough to tell nontrivial statements about program
- Must be “elementary enough” so that each “proof element” can be verified using only local program elements. “Line-by-line verification”.

Hoare logic: While programs

- Assignments y:=t
- Composition S1; S2
- If-then-else if e then S1 else S2 fi
- While e do S od
- Hoare logic consist of instrumenting each line with “what it does to the state-space”

$$\begin{array}{rl} & \{\text{state-space statement } 1\} \leftarrow precondition \\ \texttt{assignment} \Rightarrow & \texttt{assignment} \\ & \{\text{state-space statement } 2\} \leftarrow postcondition \end{array}$$

Compositionality:

$$\{pre_1\}\texttt{loc 1}\{post_1\}; \{pre_2\}\texttt{loc 2}\{post_2\} \wedge (\{post_1\} \rightarrow \{pre_2\}) \Rightarrow \{pre_1\}\texttt{loc 1;loc 2}\ \{post_2\}$$

See Peled, 2001, Monin, 2000

Example: Capturing controller behavior

The control-systemic way

$$\begin{aligned} x_{c,k+1} &= \begin{bmatrix} 0.499 & -0.050 \\ 0.010 & 1.000 \end{bmatrix} x_{c,k} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \mathbf{SAT}(y_k) \\ u_k &= -\begin{bmatrix} 564.48 & 0 \end{bmatrix} x_{c,k} + 1280\, \mathbf{SAT}(y_k) \end{aligned}$$

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*):*

The ellipsoid $\mathcal{E}_P = \left\{ x \in \mathbf{R}^2 \mid x^T P x \leq 1 \right\}. P = 10^{-3} \begin{bmatrix} 0.6742 & 0.0428 \\ 0.0428 & 2.4651 \end{bmatrix}$

is invariant. None of the entries of *x* exceeds 40 in size.

How do you say the same thing about the *program* that implements the controller?

A 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\n",u);
10:   x = A*x + B*y;
11: end
```

$\{true\}$
4: x = zeros(2,1)
$\{x \in \mathcal{E}_P\}$.
5: while 1
$\{x \in \mathcal{E}_P\}$
6: y = fscanf(stdin,"%f")
$\{x \in \mathcal{E}_P\}$
7: y = max(min(y,1),-1);
$\{x \in \mathcal{E}_P,\ y^2 \leq 1\}$
8: u = C*x+D*y;
$\{x \in \mathcal{E}_P,\ u^2 \leq 2(CP^{-1}C^T + D^2),\ y^2 \leq 1\}$
9: fprintf(stdout,"%f\n",u)
$\{x \in \mathcal{E}_P,\ y^2 \leq 1\}$

?

$\{Ax + By \in \mathcal{E}_P,\ y^2 \leq 1,$
$u^2 \leq 2(CP^{-1}C^T + D^2)\}$
9: fprintf(stdout,"%f\n",u)
$\{Ax + By \in \mathcal{E}_P,\ y^2 \leq 1\}$
10: x = A*x + B*y;
$\{x \in \mathcal{E}_P\}$
11:end
$\{false\}$

Critical step

$$\left\{x \in \mathcal{E}_P,\ y^2 \leq 1\right\} \Rightarrow \left\{Ax + By \in \mathcal{E}_P,\ y^2 \leq 1\right\}.$$

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

-Either leave it as is to check via automated checker that can handle systems of polynomial inequalities (eg Zumkeller, Thery 2008)

-Or provide additional proof element:

$$\forall (x, y)\ (Ax + By)^T P (Ax + By) - 0.01 x^T P x - 0.99 y^2 \leq 0.$$

$$\left\{x \in \mathcal{E}_P,\ y^2 \leq 1,\ (Ax + By)^T P (Ax + By) - 0.01 x^T P x - 0.99 y^2 \leq 0\right\} \Rightarrow \left\{Ax + By \in \mathcal{E}_P,\ y^2 \leq 1\right\},$$

Commented code

$\{\text{true}\}$
```
1:  A = [0.4990, -0.0500; 0.0100, 1.0000];
```
$\{\text{true}\}$
```
2:  C = [-564.48, 0];
```
$\{\text{true}\}$
```
3:  B = [1;0];D=1280
```
$\{\text{true}\}$
```
4:  x = zeros(2,1);
```
$\{x \in \mathcal{E}_P\}$
```
5:  while 1
```
$\{x \in \mathcal{E}_P\}$
```
6:  y = fscanf(stdin,"%f")
```
$\{x \in \mathcal{E}_P\}$
```
7:  y = max(min(y,1),-1);
```
$\{x \in \mathcal{E}_P,\ y^2 \leq 1\}$
```
8:  u = C*x+D*y;
```
$\{x \in \mathcal{E}_P,\ u^2 \leq 2(CP^{-1}C^T + D^2),\ y^2 \leq 1\}$
```
9:  fprintf(stdout,"%f\n",u)
```
$\{x \in \mathcal{E}_P,\ y^2 \leq 1,\ (Ax+By)^T P(Ax+By) - 0.01x^T Px - 0.99y^2 \leq 0\}$

skip

$\{Ax + By \in \mathcal{E}_P,\ y^2 \leq 1\}$
```
10:  x = A*x + B*y;
```
$\{x \in \mathcal{E}_P\}$
```
11:  end
```

Forward constraint propagation

```
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\n",u);
10:   x = A*x + B*y;
11: end
```

$$Q = \begin{bmatrix} 0.01P & 0 \\ 0 & 0.99 \end{bmatrix}$$

$\{true\}$
1: A = [0.4990,-0.0500;0.0100,1.0000];
$\{true\}$
2: C = [-564.48, 0];
$\{true\}$
3: B = [1;0];D = 1280;
$\{true\}$
4: x = zeros(2,1);
$\{x \in \mathcal{E}_P\}$
5: while 1
$\{x \in \mathcal{E}_P\}$
6: y = fscanf(stdin,"%f");
$\{x \in \mathcal{E}_P\}$
7: y = max(min(y,1),-1);
$\{x \in \mathcal{E}_P,\ y^2 \leq 1\}$
$\{(\ x\ \ y\ ) \in \mathcal{E}_Q\}$
8: u = C*x + D*y;
$\{(\ x\ \ y\ ) \in \mathcal{E}_Q,\ u^2 \leq [C\ D]\, Q^{-1}\, [C\ D]^T\}$
9: fprintf(stdout,"%f\n",u);
$\{(\ x\ \ y\ ) \in \mathcal{E}_Q\}$
10: x = A*x + B*y;
$\{x \in \mathcal{E}_{(\frac{1}{0.01}AP^{-1}A^T + \frac{1}{0.99}BB^T)^{-1}}\}$
$\{x \in \mathcal{E}_P\}$
11: end
$\{false\}$

C code analysis

```
1: int main(int argc, char *argv[])
2: {
3:   double *x[2], *xb[2], y, u;
4:   x[0] = 0;
5:   x[1] = 0;
6:   while(1){
7:     fscanf(stdin,"%f",&y);
8:     if (y >1){
9:       y=1
10:    }
11:    if (y<-1){
12:      y=-1
13:    }
14:    u = -564.48*x[0]+1280*y;
15:    fprintf(stdout,"%f\n",u);
16:    xb[0]=x[0];
17:    xb[1]=x[1];
18:    x[0]:= 0.4990*xb[0]-0.0500*xb[1]+y;
19:    x[1]:= 0.01*xb[0]+xb[1];
20:  }
21:}
```

$$\begin{bmatrix} x[1] \\ x[2] \\ y \end{bmatrix} \in \mathcal{E}_P \}$$

$$\{x \in \mathcal{E}_P,\ y > 1\}$$

$$\{x \in \mathcal{E}_P,\ y \leq 1\}$$

$$\mathcal{E}_{R^{-1}} = \{x \mid x^T R^{-1} x \leq 1\} =$$

$$\mathcal{G}_R = \left\{ x \in \mathcal{E}_P, \begin{bmatrix} 1 & x^T \\ x & R \end{bmatrix} \geq 0 \right\}$$

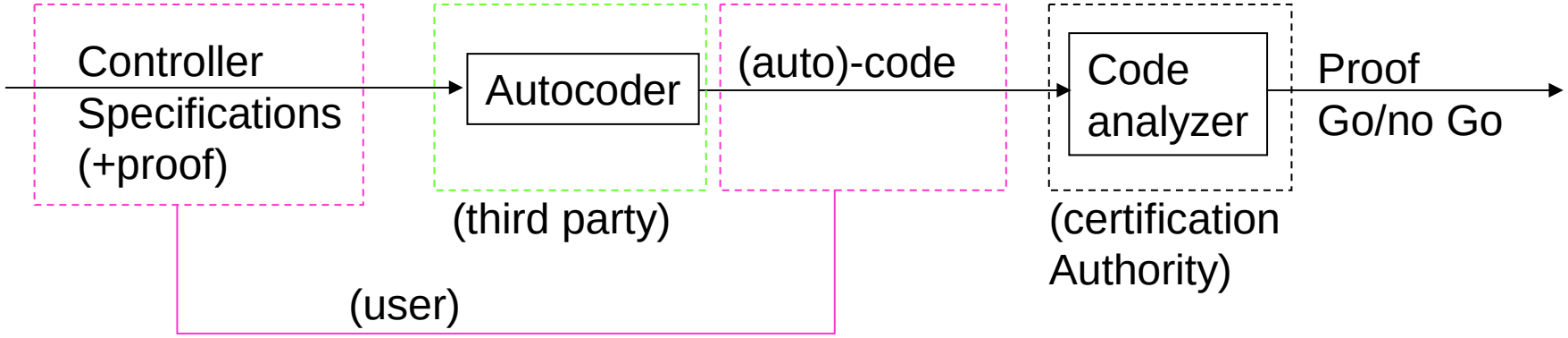
$$\left\{ \begin{bmatrix} x \\ y \end{bmatrix} \in \mathcal{E}_Q \right\} \quad Q = \begin{bmatrix} 0.01P & 0 \\ 0 & 0.99 \end{bmatrix}$$

$$\left\{ \begin{bmatrix} x \\ y \\ xb[1] \end{bmatrix} \in \mathcal{G}_R \right\}$$

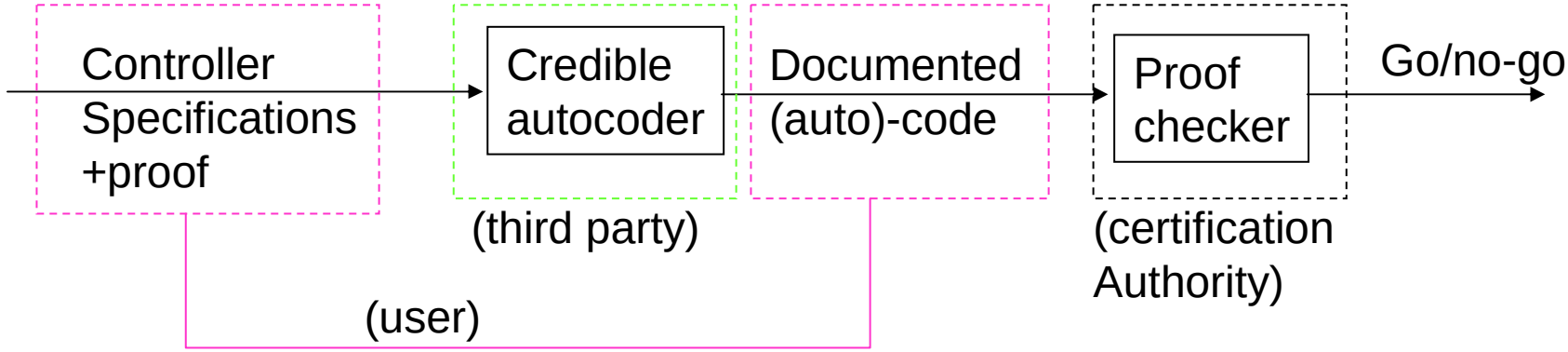
$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} Q^{-1} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}^T$$

Applications

(auto) Code analyzer



Credible autocoder (a la Rinard)



Other important questions

- Verifying closed-loop system
- So far made assumption that analyzer could handle real numbers like you and me….
- Many controllers are more complex than a simple lead-lag controller – nonlinear? Adaptive? Receding horizon???

More Complex Control Systems

Adaptive systems

$$x_{p,k+1} = x_p + \delta\left(ax_{p,k} + bu_k\right), \quad x(0) = x_0$$
$$y_{p,k} = x_{p,k}$$

a, b unknown

$$x_{m,k+1} = x_{m,k} + \delta\left(a_m x_{m,k} + b_m r_k\right), \quad x_{m,0} = x_{m0}$$
$$y_{m,k} = c_m x_{m,k}$$

Reference model

$$\rho_k = \frac{1}{1 + \varphi_k^{\mathrm{T}} \varphi_k}$$

$$\varphi(t) \triangleq \begin{bmatrix} e(t) & r(t) \end{bmatrix}^{\mathrm{T}}$$

$$\tilde{k}_{x,k+1} = \tilde{k}_{x,k} - \delta \rho_k \tilde{k}_{x,k}(y_{p,k}^2 + y_{p,k} r_k)$$
$$\tilde{k}_{r,k+1} = \tilde{k}_{r,k} - \delta \rho_k \tilde{k}_{r,k}(r_k^2 + y_{p,k} r_k)$$

Adaptive control architecture

$$u_{k+1} = (K_k + K^*)\,\varphi_k$$

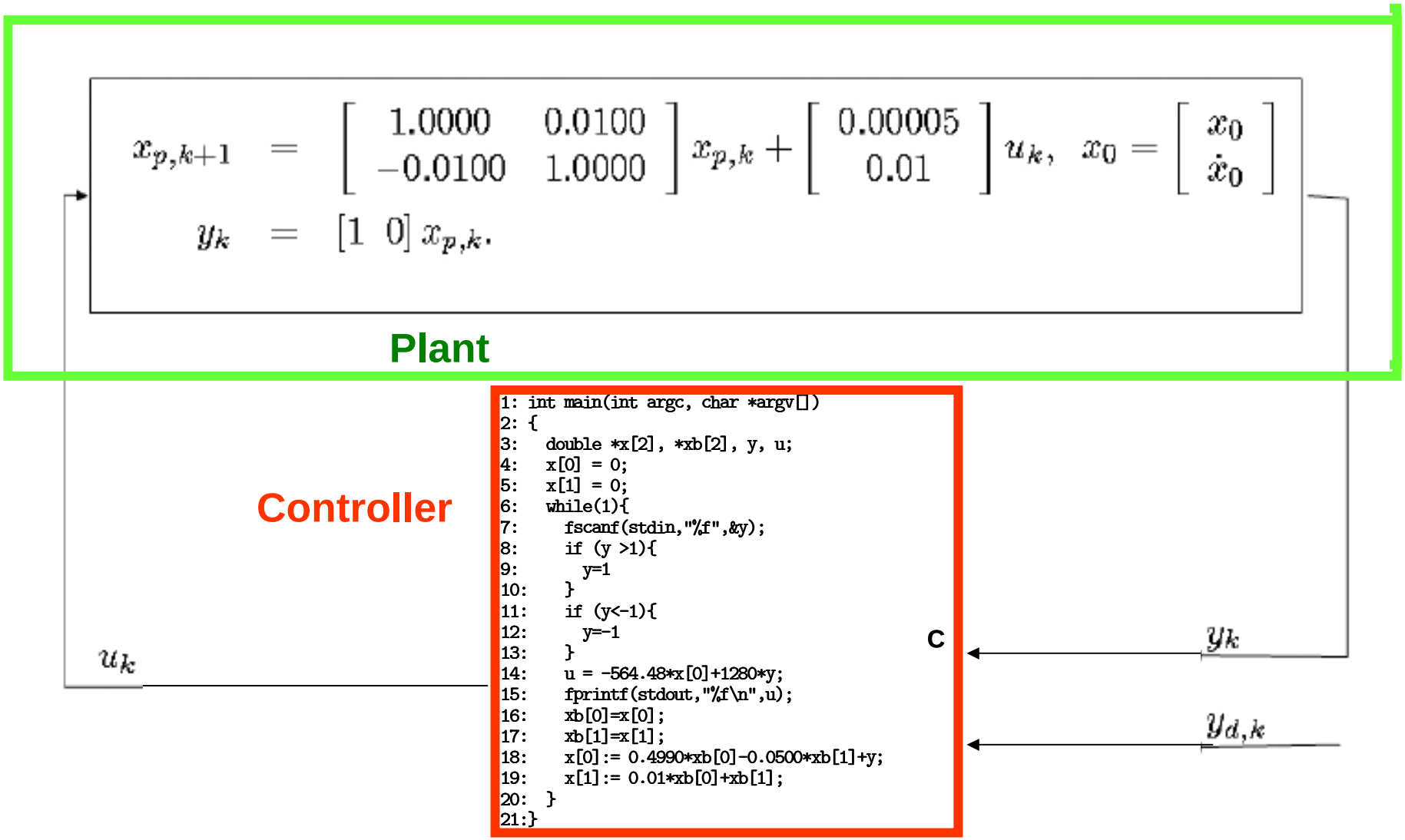
$$K_k \triangleq \begin{bmatrix} \tilde{k}_{x,k} & \tilde{k}_{r,k} \end{bmatrix}$$

$$V(e, K) = \ln\left(1 + e^{\mathrm{T}} \mathbf{P} e\right) + \alpha \mathrm{Tr}\left(K^{\mathrm{T}} K\right)$$

$$e(t) \triangleq x_p(t) - x_m(t)$$

Nonquadratic invariant structure!

Closed-loop system representations



Concurrent Representations of System and Controller

%Controller dynamics

```
01: int main(int argc, char *argv[])
02: {
03:   double xc [2], xb [2], y, yd, yc, u;
04:   xc[0] = 0;
05:   xc[1] = 0;
06:   receive(y,2);
07:   receive(yd,3);
08:   yc = y-yd;
09:   while(1){
10:     if (yc > 1){
11:       yc=1
12:     }
13:     if (yc < -1){
14:       yc=-1
15:     }
16:     u = 564.48*xc[0]-1280*yc;
17:     xb[0]=xc[0];
18:     xb[1]=xc[1];
19:     xc[0]:= 0.4990*xb[0]-0.0500*xb[1]+yc;
20:     xc[1]:= 0.01*xb[0]+xb[1];
21:     send(u,1);
22:     receive(y,2);
23:     receive(yd,3);
24:     yc = y-yd;
25:   }
26: }
```

```
% Plant Dynamics
1p: Ap = [1.0000,0.0100;
-0.0100,1.0000];
2p: Cp=[1,0];
3p: Bp = [0.00005; 0.01];
4p: while (1)
5p:    yp = Cp*xp;
6p:    send(yp,2);
7p:    receive(up,1);
8p:    xp = Ap*xp + Bp*up;
9p: end;
```

Actual Computations

- Computations do not follow real arithmetic
  - Fixed/Floating-point computations are often neglected during specification phase, comes back as “thorn in foot” during implementation.
  - Neglect at specification level makes typical “leaky abstraction”
  - Justified because detailed computations are typically left aside when spec-ing system
- Computation model that’s agreeable/manageable by everybody

Additive/multiplicative uncertainty model

(Discussions with Eric Goubault & Sylvie Putot)

x real

$$x^{\circ}{}_{\mathrm{oat}} = x(1 + \Delta_1(x)) + \Delta_2(x)$$

IEEE 754: x, y floats

$$(x + y)^{\circ}{}_{\mathrm{oat}} = (x + y)(1 + \Delta_1(x + y)) + \Delta_2(x + y)$$

$$\text{Same for } -,\ \times,\ /$$

$$\text{Only upper bounds are known on } \Delta_1(.), \Delta_2(.)$$

$$\text{eg } |\Delta_1(.)| \leq M_1,\ |\Delta_2(.)| \leq M_2$$

Consequence on invariance conditions

Ellipsoidal invariance condition

$$\left\{x \in \mathcal{E}_P,\ y^2 \leq 1\right\} \Rightarrow \left\{Ax + By \in \mathcal{E}_P,\ y^2 \leq 1\right\}.$$

becomes

$$\begin{aligned} &\left\{x \in \mathcal{E}_P,\ y^2 \leq 1\right\} \\ \Rightarrow &\left\{F(x, y, \Delta_1, \Delta_2, \ldots, \Delta_M) \in \mathcal{E}_P,\ y^2 \leq 1\right\} \end{aligned}$$

Back to proof checker / proof assistant that can handle systems of polynomial inequalities (eg Zumkeller, Thery 2008)

Conclusion

- Stability and performance proofs of control systems fundamentally compatible with formal static analysis and verification methods – must feed available information in “system assembly process”.
- Must write software development tools that run across layers of implementation and are compatible with verification preferences/levels of rigor favored by different communities.