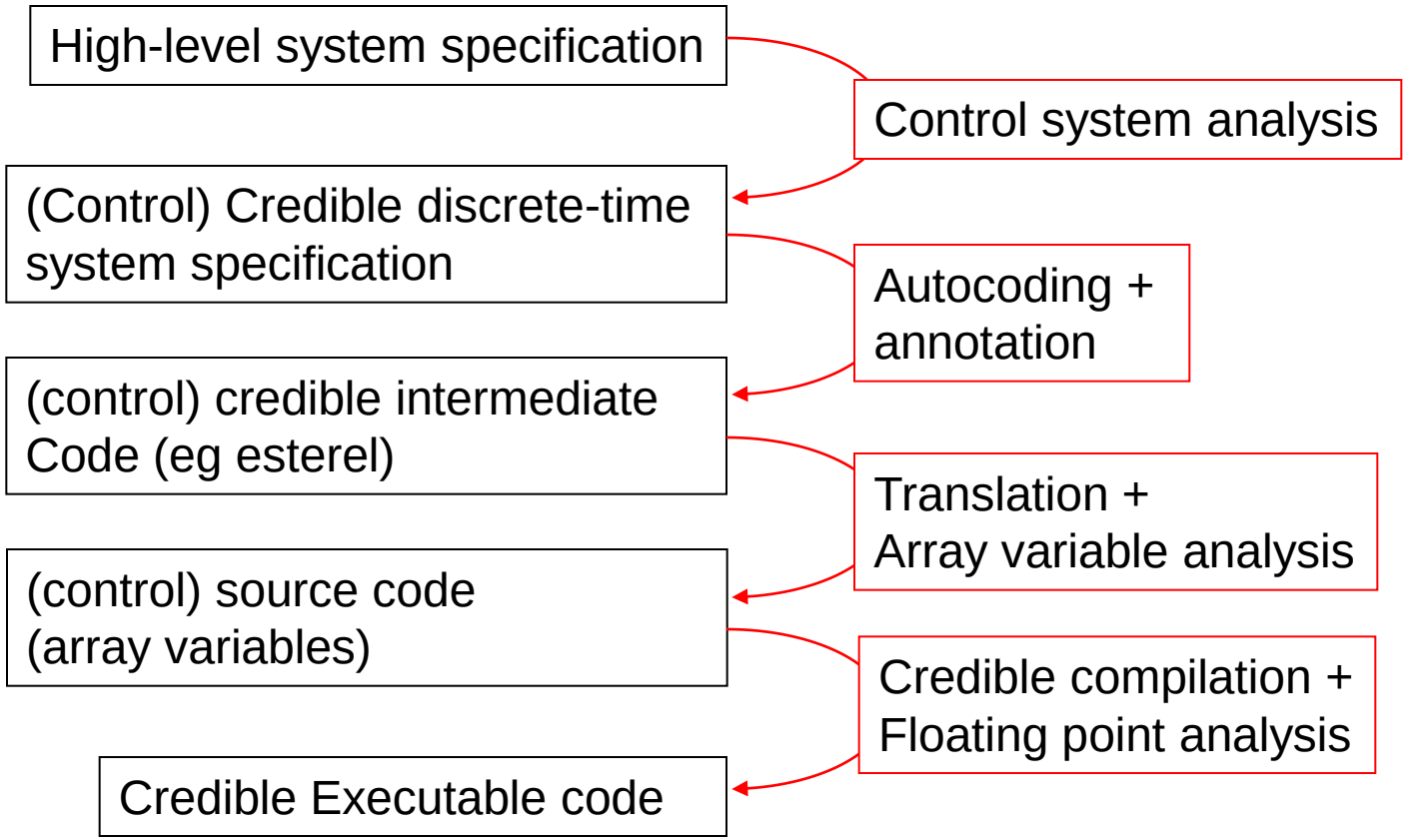
A cascaded approach to control system software design and verification

Darren Cofer, Eric Feron, Vlad Gavrilets, Arnaud Venet

Rockwell Collins ATC, Georgia Tech, Rockwell-Collins Controls Technologies, Carnegie-Mellon University West

Main message

- We propose an hierarchical approach to control system V&V.



Some Background: Control systems

- Much ongoing progress in control system analysis and design:
  - Nonlinear control systems
  - Adaptive controls
  - Robustness analyses
  - Decentralized control systems

- Novel Control systems often come with rigorous proofs and spectacular application.
- Control specs flow down to software design, proofs do not and must be reconstructed from scratch (see ASTREE example next)

Some background: Software analysis

- Many static software analyzers over past few decades.
- Industrial implementation of model checkers.
- Application of abstract interpretation techniques (ASTREE analyzer) for Airbus A380 control code.

From Feret, 2004

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.

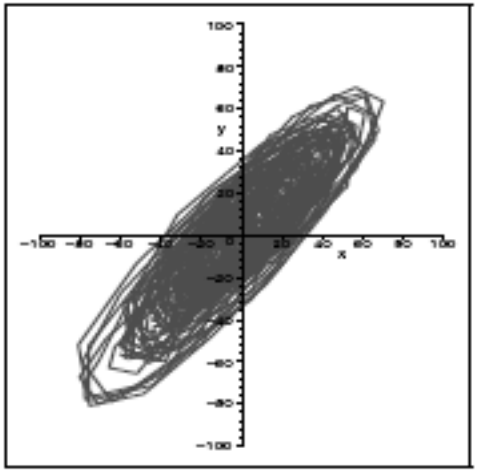


Fig. 4. Orbit.

Control specifications semantics trickle down to software implementation.

Core idea

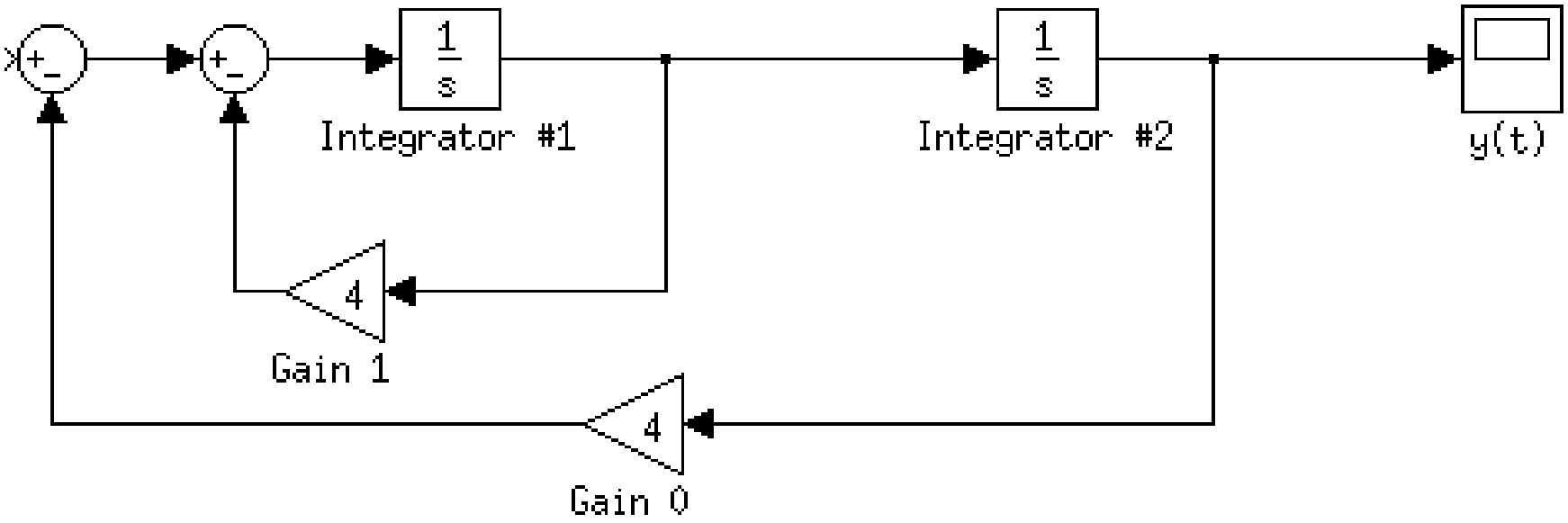
- Develop an environment where control/software engineers can implement control laws (and other, many other control-oriented products) and flow them down seamlessly to executable software, together with formal proofs of correctness.
- Call that “credible autocoder” to emphasize need to produce formally proven software.
- Properties of specifications/software are extracted as soon as possible and flow down with successive implementations.

Process

- We begin with a control systems, user-friendly interface (eg simulink-like).

- We end with a commented source code.

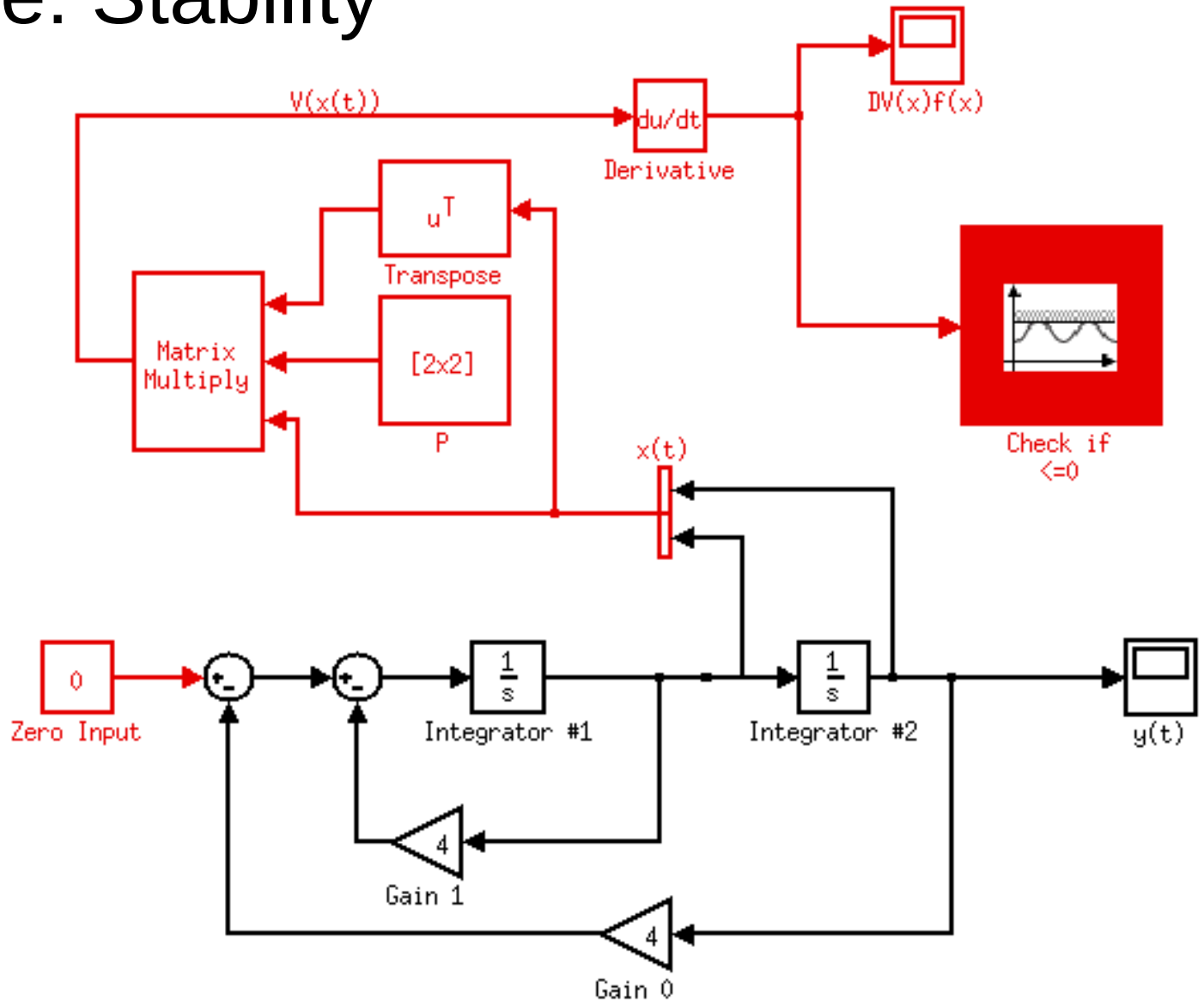
Specification level: Graphical interfaces



Typical simulink diagram comes with no comments/semantics

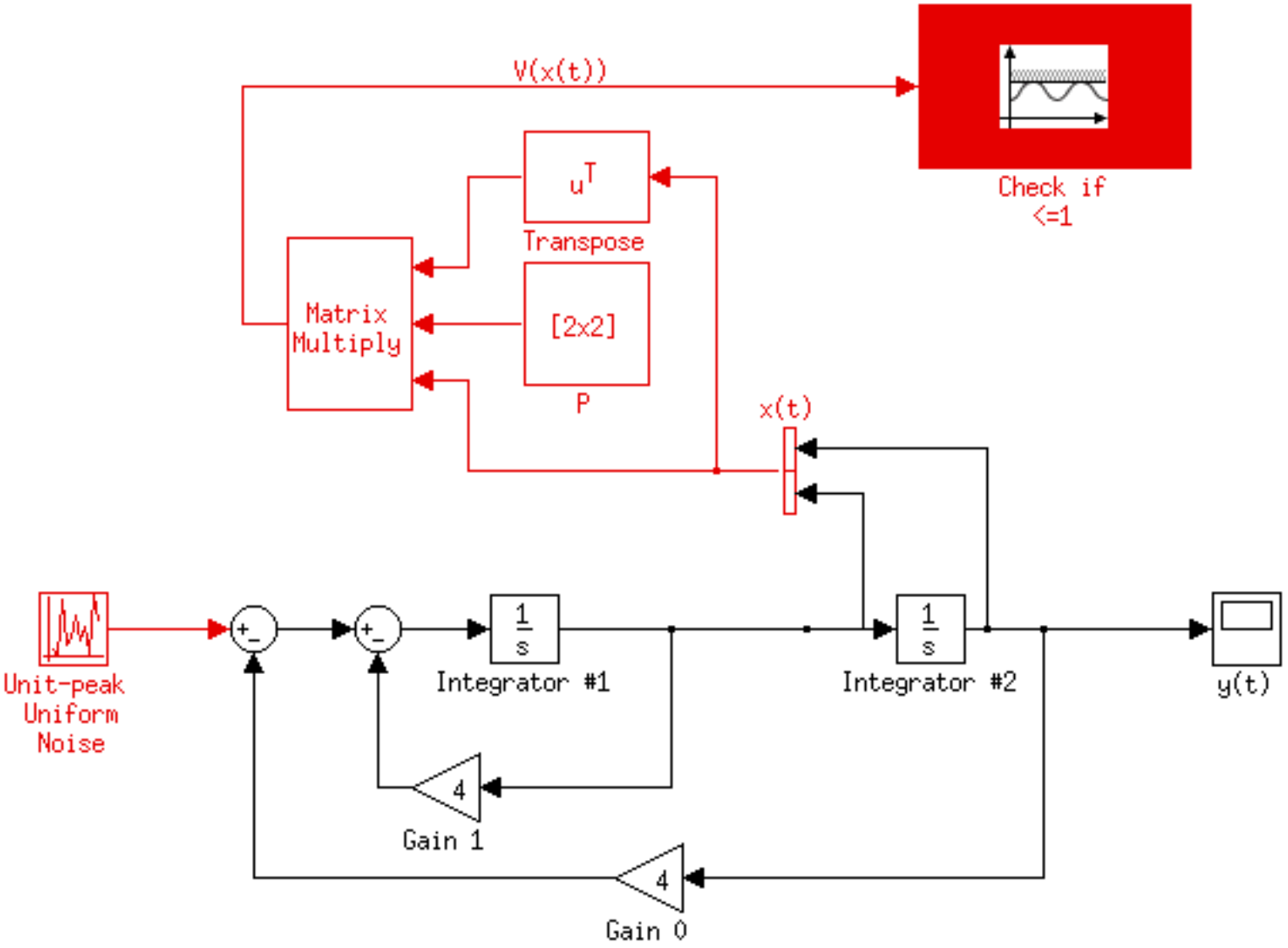
Can use commenting capabilities of Simulink to iilllustrate design process

Example: Stability



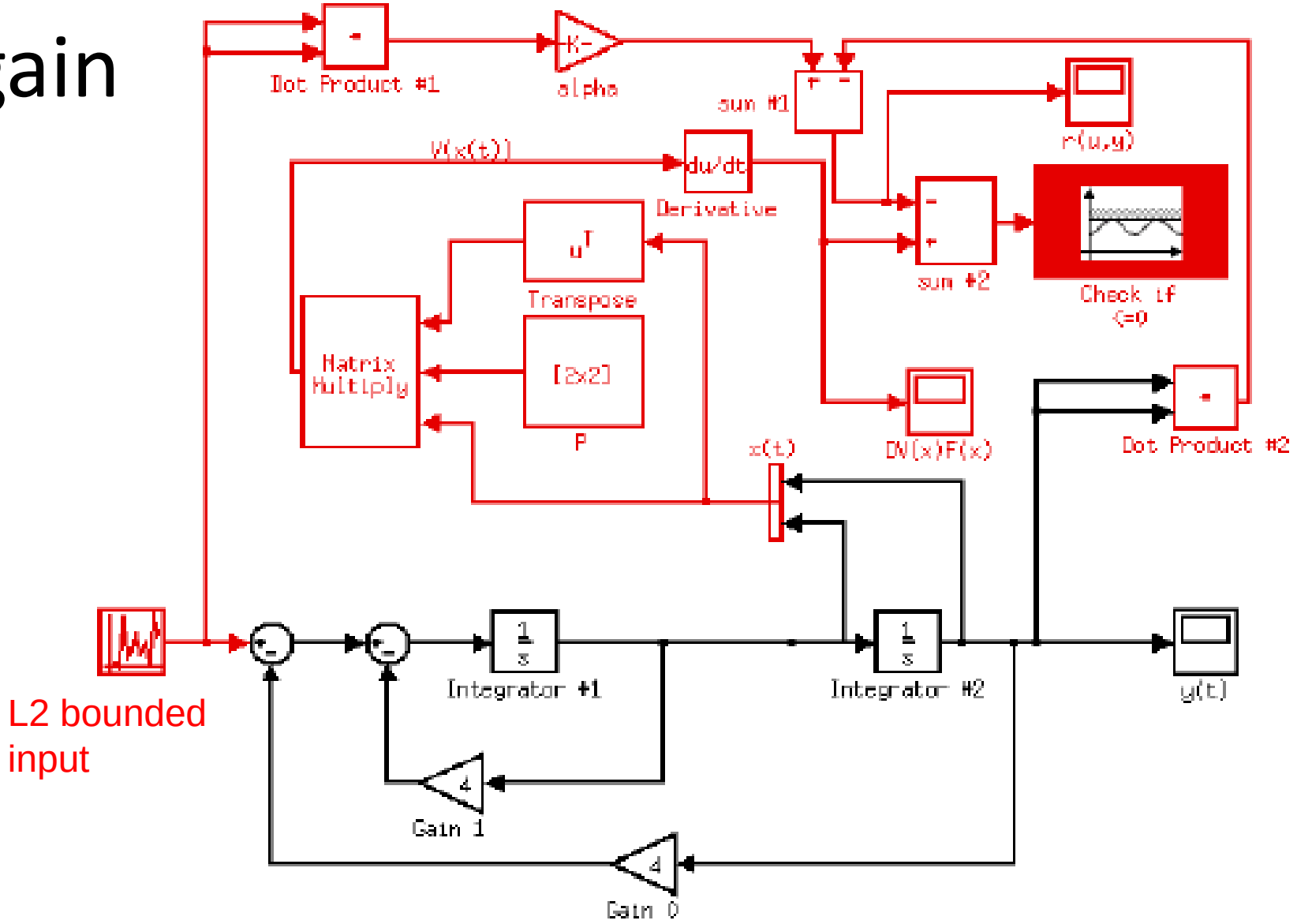
$$\frac{d}{dt}x^T P x < 0, \quad P > 0$$

State boundedness under bounded inputs



$$x^T P x \leq 1, \quad P > 0$$

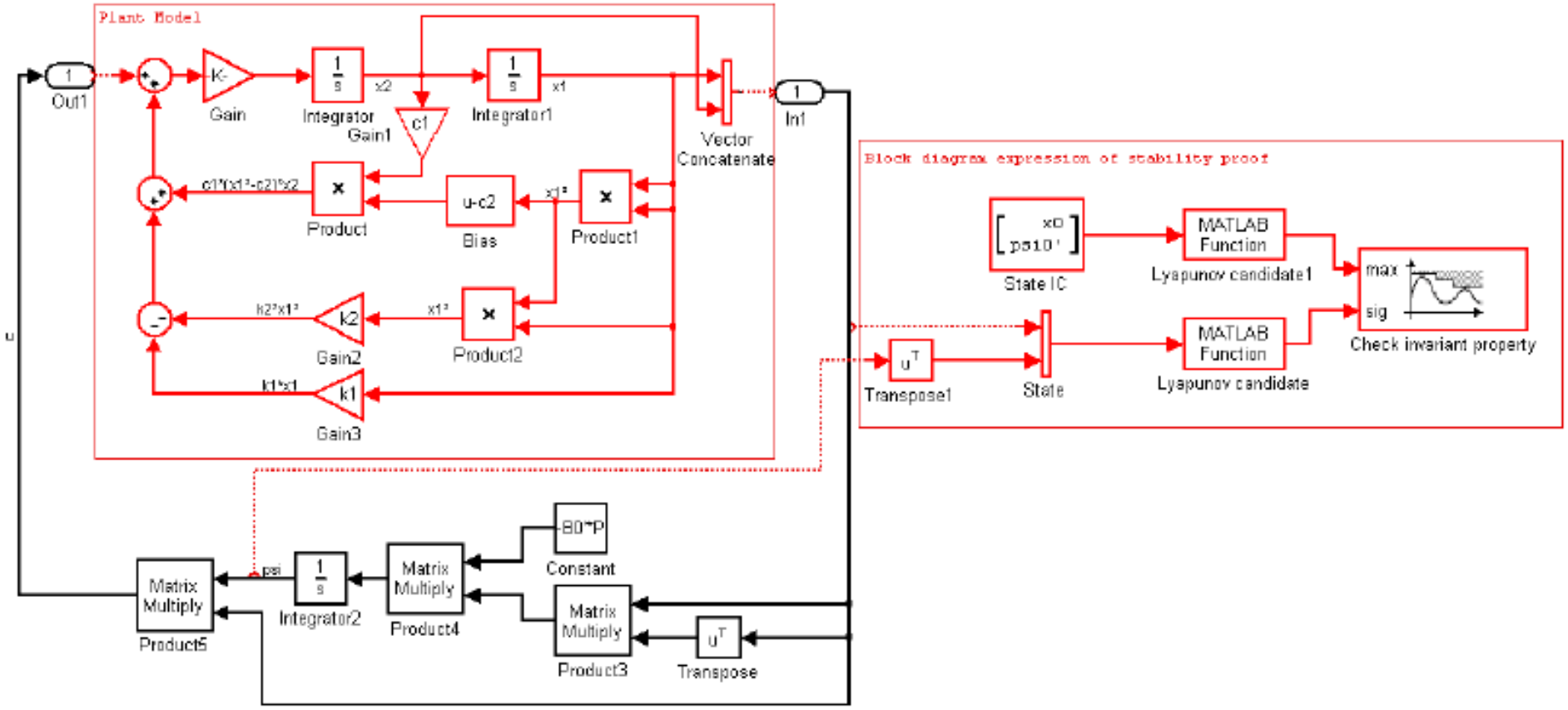
L2 gain



$$\frac{d}{dt}x^T Px - \alpha^2 w^T w + y^T y \leq 0 \rightarrow \int_0^\infty y^T y \leq \alpha^2 \int_0^\infty w^T w$$

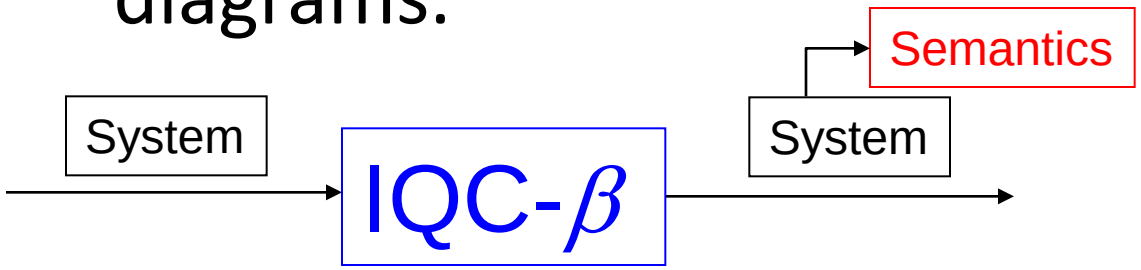
Adding the controlled plant as part of the controller's semantics

Closed-loop adaptive control system stability



Using third party system analysis tools

- Sometimes, control system properties can be established via automated analysis; "Specification level static analysis"
- μ–tools, IQC-β
- Automatic generation of Lyapunov functions that establish (stability, boundedness, performance) without human intervention.
- Can be used to automatically generate earlier diagrams.



Further steps

- Discretization, expression in intermediate language, expression in source code all introduce additional analysis requirements.
- Time discretization: Lyapunov stability proofs that hold for continuous systems may hold for discretized system (most of the time). Often fails (adaptive systems, low sampling rate). Solution may require doing discrete Lyapunov-like analysis from scratch (easy for the most part).

Further steps (ct'd)

- Expression in intermediate language: eg Matlab or SCADE.

$\{true\}$
1: Ac = [0.4990, -0.0500; 0.0100, 1.0000];
$\{true\}$
2: Cc = [564.48, 0];
$\{true\}$
3: Bc = [1;0];D=-1280
$\{true\}$
4: xc = zeros(2,1);
$\{x_c \in \mathcal{E}_P\}$
5: receive(y);
$\{x_c \in \mathcal{E}_P\}$
6: receive(yd);
$\{x_c \in \mathcal{E}_P\}$
7: yc = y-yd;
$\{x_c \in \mathcal{E}_P\}$
8: while 1
$\{x_c \in \mathcal{E}_P\}$
9: yc = max(min(y,1),-1);
$\{x_c \in \mathcal{E}_P,\ y_c^2 \leq 1\}$
10: u = Cc*xc+Dc*yc;
$\{x_c \in \mathcal{E}_P,\ y_c^2 \leq 1,$
$(A_c x_c + B_c y_c)^T P (A_c x_c + B_c y_c)$
$-0.01 x_c^T P x_c - 0.99 y_c^2 \leq 0,$
$u^2 \leq 2(C_c P^{-1} C_c^T + D_c^2)\}$
skip
$\{A_c x_c + B_c y_c \in \mathcal{E}_P,$
$y_c^2 \leq 1\},\ u^2 \leq 2(CP^{-1}C^T + D^2)$
11: xc = Ac*xc + Bc*yc;
$\{x_c \in \mathcal{E}_P,\ u^2 \leq 2(CP^{-1}C^T + D^2)\}$
12: send(u)
$\{x_c \in \mathcal{E}_P\}$
13: receive(y);
$\{x_c \in \mathcal{E}_P\}$
14: receive(yd);
$\{x_c \in \mathcal{E}_P\}$
15: yc = y-yd;
$\{x_c \in \mathcal{E}_P\}$
16: end
$\{false\}$

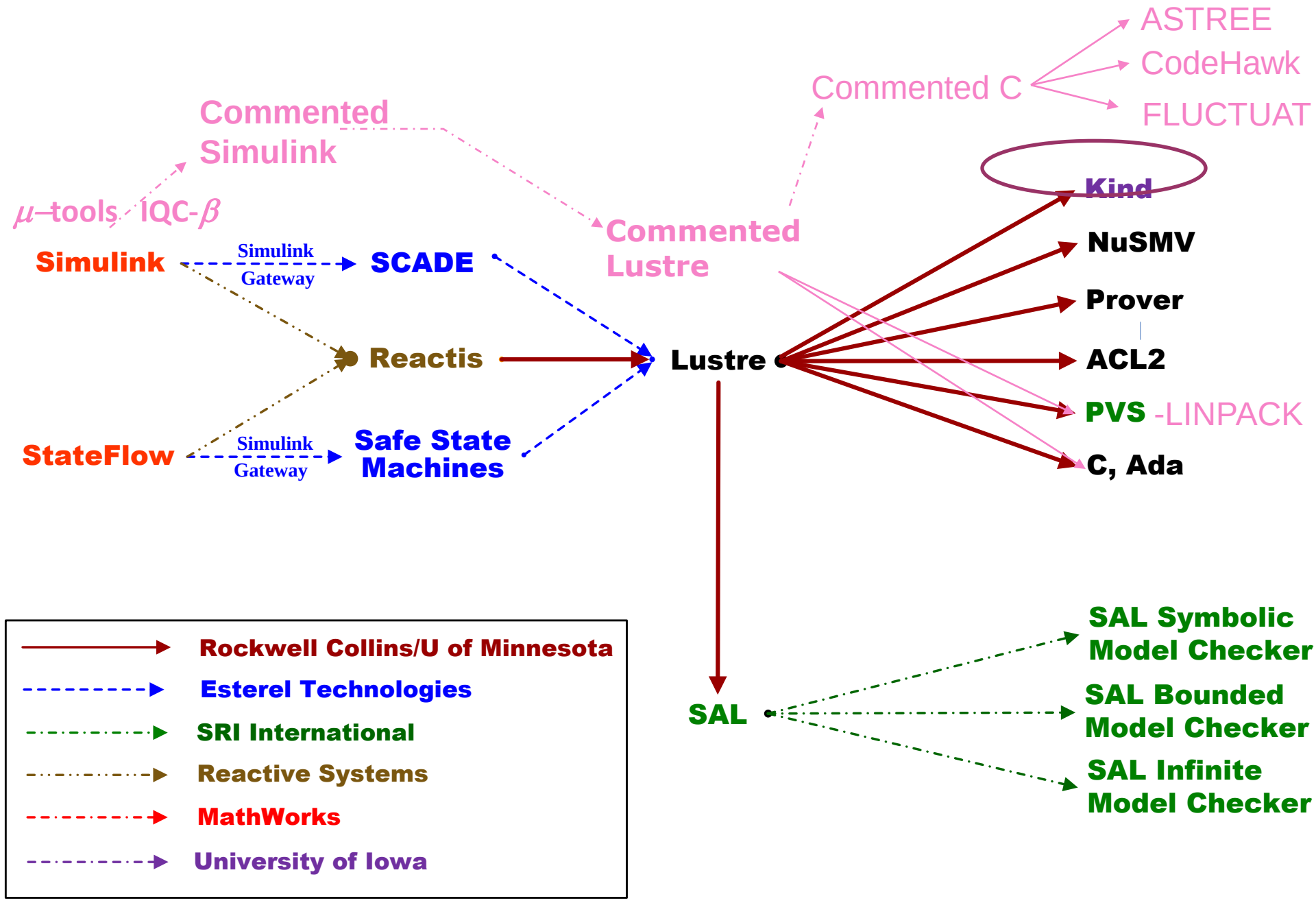
Translation to source code

- Must preserve semantics identified in block diagrams, carried into MATLAB: Translator currently under development at Georgia Tech.
- Translation to source code introduces many new variables. Eg Matlab has highly vectorized operations and vector variables, C uses arrays, array indices, pointers, casts etc. Coherence of these variables (range…) must be verified independently. Use static analyzers such as global C surveyor (Developed at NASA by Carnegie-Mellon West)

From source code to binary

- Detailed structure of operations is known; enables detailed analysis of numerical computations with floats.
- Begin effort with executable version of FLUCTUAT (CEA/LIST --- Goubault/Putot)
- Research issue: Need to revisit high-level invariants determined earlier (quadratic inequalities), based on real computations.

Insertion in Rockwell Collins Translation Framework



Academic Example

- Quanser 3 DOF Helicopter

- Easy to fly and demonstrate advanced control concepts. Currenty undergoing task of implementing credible controller written in C

Industry Example

- F-18 R/C subscale model, with Rockwell Collins Damage Tolerant Flight Control Technology

•Use verifiable autocoder to generate source code for a subset of damage tolerant control laws, e.g. Lyapunov-based model reference adaptive control for lateral-directional dynamics