Automatic Software Verification for High-Confidence Cyber-Physical Systems

Miroslav Pajic

Department of Electrical and Computer Engineering
Department of Computer Science
Duke University

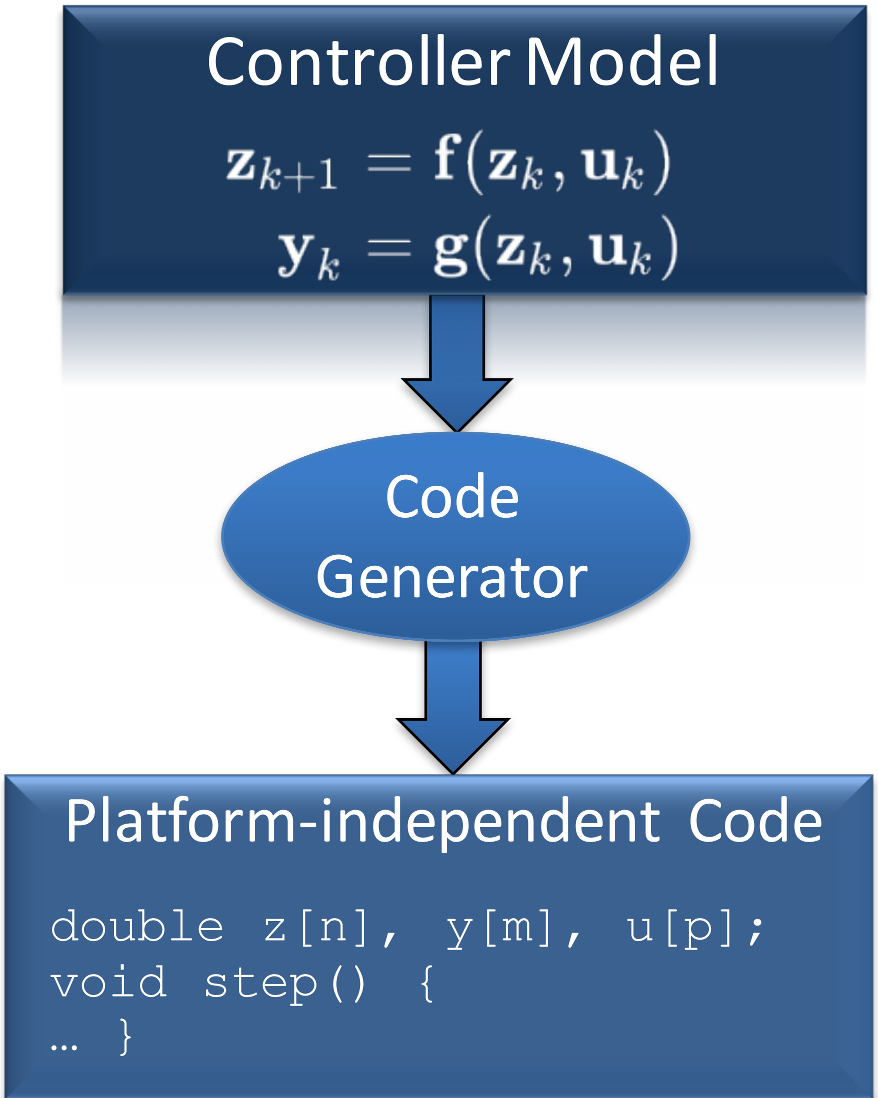
joint work with

Junkil Park   Insup Lee   Oleg Sokolsky
University of Pennsylvania

Duke PRATT SCHOOL OF Engineering

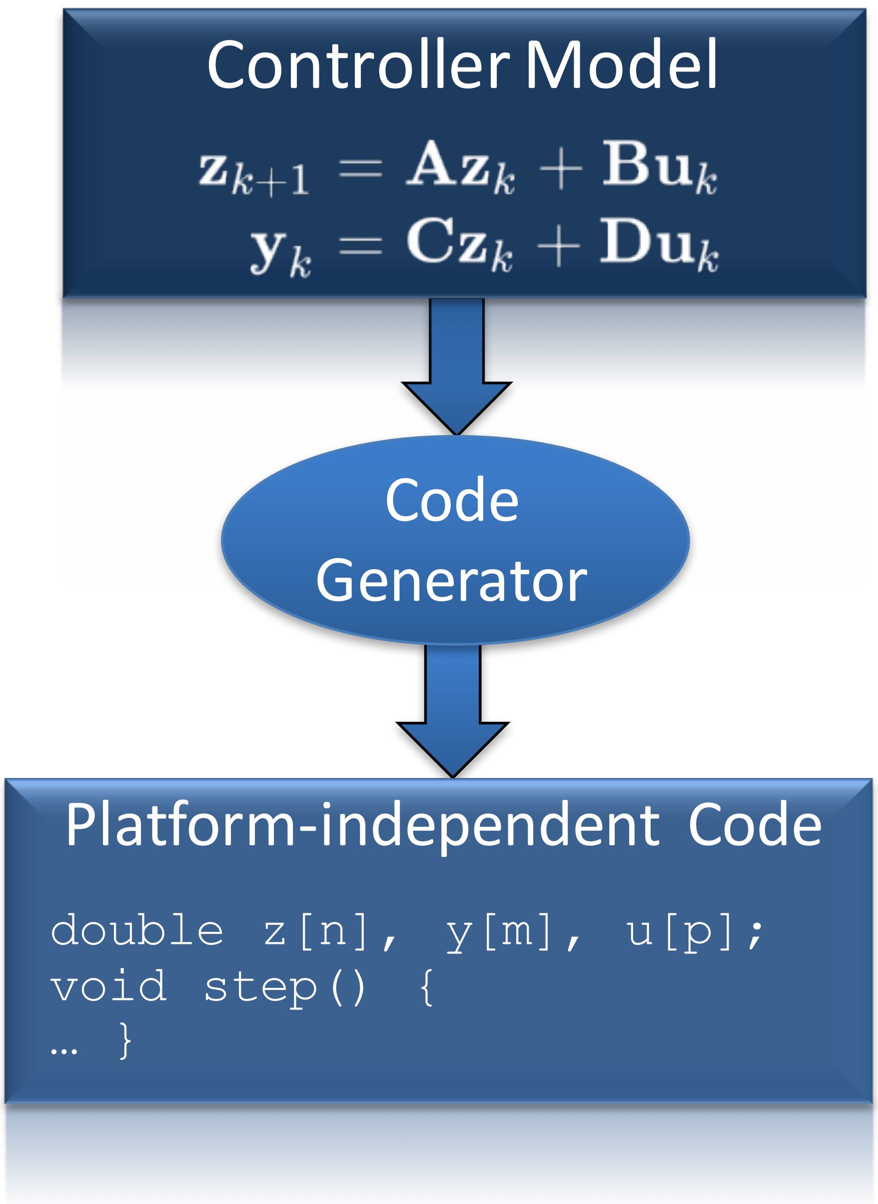
Code Generation for Embedded Control

Duke
PRATT SCHOOL OF
Engineering



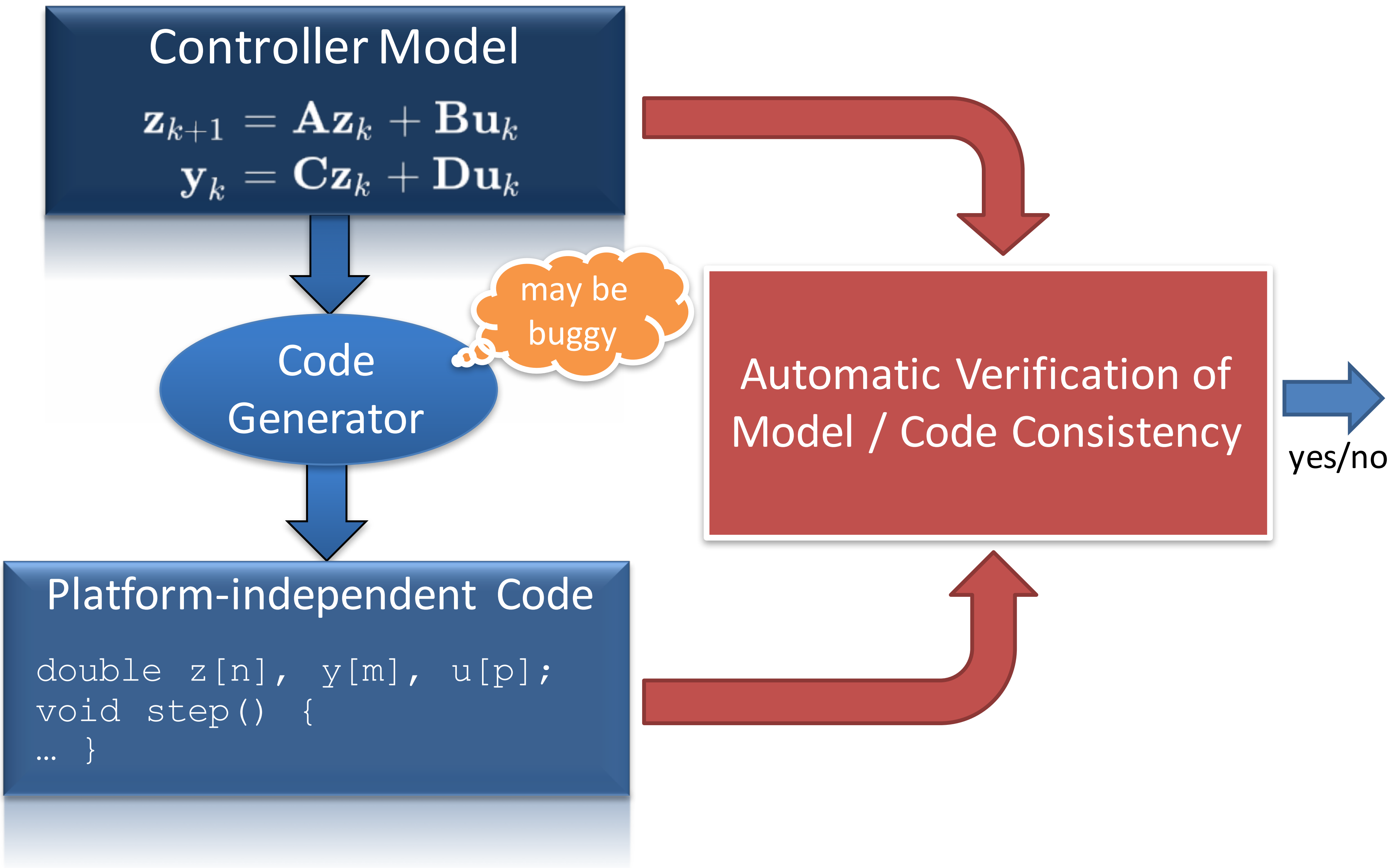
- From a closed-loop system model
  - Controller model $\Sigma(\mathbf{f}, \mathbf{g})$ (i.e., controller parameters)
- Code generator
  - `step` function
  - may employ optimization that affects the controller state

Code Generation for Embedded Control



- From a closed-loop system model
  - Controller model $\Sigma(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ (i.e., controller parameters)
- Code generator
  - `step` function
  - may employ optimization that affects the controller state
- Goal – verification of the generated code
  - Linear controllers - a very large class of embedded controllers

Code Generation for Embedded Control



A straightforward approach…

Defining Invariants for Linear Controllers based on input-output and state invariants

Annotating Input-Output and State Invariants

Duke PRATT SCHOOL OF Engineering

- Exploit the ACSL's notion of the function contract
  - effectively a Hoare triple

Controller Model

$$\mathbf{z}_{k+1} = \mathbf{A}\mathbf{z}_k + \mathbf{B}\mathbf{u}_k$$
$$\mathbf{y}_k = \mathbf{C}\mathbf{z}_k + \mathbf{D}\mathbf{u}_k$$

- Running example:

$$\mathbf{A} = \begin{bmatrix} 0.8147 & 1.1534 \\ 2.6413 & 3.6411 \end{bmatrix},$$

$$\mathbf{B} = \begin{bmatrix} 3.1019 \\ 2.1432 \end{bmatrix},$$

$$\mathbf{C} = \begin{bmatrix} 1.7121 & 0.1351 \end{bmatrix}$$

```
double x[2], u, y;
/*@ requires \valid(x+(0..1));
  @ ensures x[0] == 0.8147*\old(x[0]) +
  @         1.1534*\old(x[1]) + 3.10191*\old(u);
  @ ensures x[1] == 2.6413*\old(x[0]) +
  @         3.6411*\old(x[1]) + 2.1432*\old(u);
  @ ensures y == 1.7121*\old(x[0]) +
  @         0.1351*\old(x[1]) + 0*\old(u);
*/
void step() {
  double t1, t2;
  y = 1.7121*x[0] + 0.1351*x[1];
  t1 = 0.8147*x[0] + 1.1534*x[1] + 3.1019*u;
  t2 = 2.6413*x[0] + 3.6411*x[1] + 2.1432*u;
  x[0] = t1;
  x[1] = t2;
}
```

A straightforward approach…

… does not always work

Example

A simple linear integrator

$$y_k = \sum_{i=0}^{k-1} \alpha u_i, \quad k \geq 1,$$
$$y_0 = 0$$

$\Sigma(1, \alpha, 1, 0)$

$$z_{k+1} = z_k + \alpha u_k,$$
$$y_k = z_k$$

$\hat{\Sigma}(1, 1, \alpha, 0)$

$$z_{k+1} = z_k + u_k,$$
$$y_k = \alpha z_k$$

- Both functionally correct but the maintained states are different
  - The latter could introduce a lower computational error when finite precision computations are taken into account

Example

MIMO control of a batch reactor

$$\mathbf{z}_{k+1} = \underbrace{\begin{bmatrix} 0.942 & 0.006888 & 0.04187 & -0.02319 \\ -0.01543 & 0.7965 & -0.03386 & 0.001563 \\ -0.1537 & 0.0137 & 0.7417 & 0.2006 \\ -0.03841 & 0.05637 & -0.02116 & 0.9949 \end{bmatrix}}_{\mathbf{A}} \mathbf{z}_k + \underbrace{\begin{bmatrix} 0.0774 & -0.0103 \\ -0.0022 & 0.0227 \\ 0.0267 & 0.0398 \\ 0.0356 & 0.0001 \end{bmatrix}}_{\mathbf{B}} \mathbf{y}_k$$

$$\mathbf{u}_k = \underbrace{\begin{bmatrix} 0.0583 & 0.9093 & 0.3258 & 0.08721 \\ -2.464 & -0.0504 & -1.71 & 1.165 \end{bmatrix}}_{\mathbf{C}} \mathbf{z}_k$$

$n(n+p)$ multiplications

$$\hat{\mathbf{z}}_{k+1} = \underbrace{\begin{bmatrix} 0.7636 & 0 & 0 & 0 \\ 0 & 0.8393 & 0 & 0 \\ 0 & 0 & 0.9595 & 0 \\ 0 & 0 & 0 & 0.9127 \end{bmatrix}}_{\hat{\mathbf{A}}} \hat{\mathbf{z}}_k + \underbrace{\begin{bmatrix} -0.2867 & -0.2581 \\ -0.3964 & -0.04506 \\ -0.07256 & 0.03278 \\ 0.5478 & -0.003331 \end{bmatrix}}_{\hat{\mathbf{B}}} \mathbf{y}_k,$$

$$\mathbf{u}_k = \underbrace{\begin{bmatrix} -0.1318 & 0.03834 & 0.02127 & -0.01226 \\ 0.147 & 0.08209 & -0.08674 & -0.2307 \end{bmatrix}}_{\hat{\mathbf{C}}} \hat{\mathbf{z}}_k$$

$n(1+p)$ multiplications

There exists a non-singular matrix ***T***: $\hat{\mathbf{A}} = \mathbf{T}^{-1}\mathbf{A}\mathbf{T},\ \hat{\mathbf{B}} = \mathbf{T}^{-1}\mathbf{B},\ \hat{\mathbf{C}} = \mathbf{C}\mathbf{T}$

If the same inputs then: $\forall k \geq 0,\ \hat{\mathbf{z}}_k = \mathbf{T}^{-1}\mathbf{z}_k \quad \Leftrightarrow \quad \hat{\mathbf{z}}_0 = \mathbf{T}^{-1}\mathbf{z}_0$

Example

MIMO control of a batch reactor

$$\mathbf{z}_{k+1} = \underbrace{\begin{bmatrix} 0.942 & 0.006888 & 0.04187 & -0.02319 \\ -0.01543 & 0.7965 & -0.03386 & 0.001563 \\ -0.1537 & 0.0137 & 0.7417 & 0.2006 \\ -0.03841 & 0.05637 & -0.02116 & 0.9949 \end{bmatrix}}_{\mathbf{A}} \mathbf{z}_k + \underbrace{\begin{bmatrix} 0.0774 & -0.0103 \\ -0.0022 & 0.0227 \\ 0.0267 & 0.0398 \\ 0.0356 & 0.0001 \end{bmatrix}}_{\mathbf{B}} \mathbf{y}_k$$

$$\mathbf{u}_k = \underbrace{\begin{bmatrix} 0.0583 & 0.9093 & 0.3258 & 0.08721 \\ -2.464 & -0.0504 & -1.71 & 1.165 \end{bmatrix}}_{\mathbf{C}} \mathbf{z}_k$$

$n(n+p)$ multiplications

$$\hat{\mathbf{z}}_{k+1} = \underbrace{\begin{bmatrix} 0.7636 & 0 & 0 & 0 \\ 0 & 0.8393 & 0 & 0 \\ 0 & 0 & 0.9595 & 0 \\ 0 & 0 & 0 & 0.9127 \end{bmatrix}}_{\hat{\mathbf{A}}} \hat{\mathbf{z}}_k + \underbrace{\begin{bmatrix} -0.2867 & -0.2581 \\ -0.3964 & -0.04506 \\ -0.07256 & 0.03278 \\ 0.5478 & -0.003331 \end{bmatrix}}_{\hat{\mathbf{B}}} \mathbf{y}_k,$$

$$\mathbf{u}_k = \underbrace{\begin{bmatrix} -0.1318 & 0.03834 & 0.02127 & -0.01226 \\ 0.147 & 0.08209 & -0.08674 & -0.2307 \end{bmatrix}}_{\hat{\mathbf{C}}} \hat{\mathbf{z}}_k$$

$n(1+p)$ multiplications

When same inputs are applied, the controllers' outputs will be identical!

- The controllers provide the same control functionality – ***input-output conformance***

How to verify LTI controllers when the maintained state is now known?

We need a specification of the controller that is insensitive to the representation of control state

Invariant-Checking Approach (IC)

Duke
PRATT SCHOOL OF
Engineering

Original Model

$$\mathbf{z}_{k+1} = \underbrace{\begin{bmatrix} -0.500311 & 0.16751 & 0.028029 & -0.395599 & -0.652079 \\ 0.850942 & 0.181639 & -0.29276 & 0.481277 & 0.638183 \\ -0.458583 & -0.002389 & -0.154281 & -0.578708 & -0.769495 \\ 1.01855 & 0.638926 & -0.668256 & -0.258506 & 0.119959 \\ 0.100383 & -0.432501 & 0.122727 & 0.82634 & 0.892296 \end{bmatrix}}_{\mathbf{A}} \mathbf{z}_k + \underbrace{\begin{bmatrix} 1.1149 & 0.164423 \\ -1.56592 & 0.634384 \\ 1.04856 & -0.196914 \\ 1.96066 & 3.11571 \\ -3.02046 & -1.96087 \end{bmatrix}}_{\mathbf{B}} \mathbf{u}_k$$

$$\mathbf{y}_k = \underbrace{\begin{bmatrix} 0.283441 & 0.032612 & -0.75658 & 0.085468 & 0.161088 \\ -0.528786 & 0.050734 & -0.681773 & -0.432334 & -1.17988 \end{bmatrix}}_{\mathbf{C}} \mathbf{z}_k$$

$\overset{?}{=}$

Code

```c
void LTIS_step(void)
{
  {
    {
      static const int_T colCidxRow0[5] = { 0, 1, 2, 3, 4 };
      const int_T *pCidx = &colCidxRow0[0];
      const real_T *pC0 = LTIS_ConstP.Internal_C;
      const real_T *xd = &LTIS_DW.Internal_DSTATE[0];
      real_T *y0 = &LTIS_Y.y[0];
      int_T numNonZero = 4;
      *y0 = (*pC0++) * xd[*pCidx++];
      while (numNonZero--) {
        *y0 += (*pC0++) * xd[*pCidx++];
      }
    }
    {
      static const int_T colCidxRow1[5] = { 0, 1, 2, 3, 4 };
      const int_T *pCidx = &colCidxRow1[0];
      const real_T *pC5 = &LTIS_ConstP.Internal_C[5];
      const real_T *xd = &LTIS_DW.Internal_DSTATE[0];
      real_T *y1 = &LTIS_Y.y[1];
      int_T numNonZero = 4;
      *y1 = (*pC5++) * xd[*pCidx++];
      while (numNonZero--) {
        *y1 += (*pC5++) * xd[*pCidx++];
      }
    }
  }
  {
    real_T xnew[5];
    int_T i;
    xnew[0] = (0.87224)*LTIS_DW.Internal_DSTATE[0];
    xnew[0] += (0.822174)*LTIS_U.u[0]+(-0.438008)*LTIS_U.u[1];
    xnew[1] = (0.366378)*LTIS_DW.Internal_DSTATE[1];
    xnew[1] += (-0.278536)*LTIS_U.u[0]+(-0.824313)*LTIS_U.u[1];
    xnew[2] = (-0.540795)*LTIS_DW.Internal_DSTATE[2];
    xnew[2] += (0.874484)*LTIS_U.u[0]+(0.858857)*LTIS_U.u[1];
    xnew[3] = (-0.332664)*LTIS_DW.Internal_DSTATE[3];
    xnew[3] += (-0.117628)*LTIS_U.u[0]+(-0.506362)*LTIS_U.u[1];
    xnew[4] = (-0.204322)*LTIS_DW.Internal_DSTATE[4];
    xnew[4] += (-0.955459)*LTIS_U.u[0]+(-0.622498)*LTIS_U.u[1];
    for(i=0; i<5; i++) LTIS_DW.Internal_DSTATE[i] = xnew[i];
  }
}
```

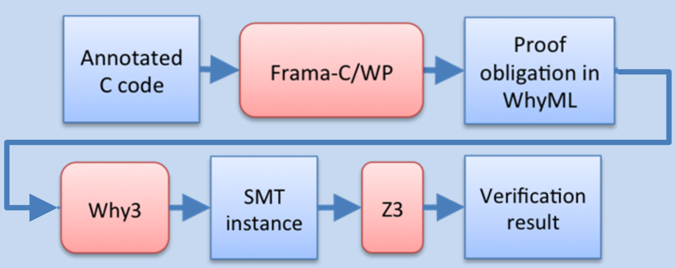
(1) Loop unrolling

(2) Automatic Annotation

$$y_k = \sum_{i=0}^{n} \beta_i u_{k-i} - \sum_{i=1}^{n} \alpha_i y_{k-i}$$

@ assert \at(y, k_n) + α_1\at(y, k_{n-1})+...
@ α_n*\at(y, k_0) == β_0*\at(u, k_n) +...
@ β_n*\at(u, k_0)

Frama/C-based Toolchain



M. Pajic, J. Park, I. Lee, G. J. Pappas, and O. Sokolsky, “Automatic Verification of Linear Controller Software", 12th ACM SIGBED International Conference on Embedded Software (EMSOFT), pp. 217-226, October 2015

A More Scalable Approach (SC)

Duke PRATT SCHOOL OF Engineering

Original Model

$$\mathbf{z}_{k+1} = \underbrace{\begin{bmatrix} -0.500311 & 0.16751 & 0.028029 & -0.395599 & -0.652079 \\ 0.850942 & 0.181639 & -0.29276 & 0.481277 & 0.638183 \\ -0.458583 & -0.002389 & -0.154281 & -0.578708 & -0.769495 \\ 1.01855 & 0.638926 & -0.668256 & -0.258506 & 0.119959 \\ 0.100383 & -0.432501 & 0.122727 & 0.82634 & 0.892296 \end{bmatrix}}_{\mathbf{A}} \mathbf{z}_k + \underbrace{\begin{bmatrix} 1.1149 & 0.164423 \\ -1.56592 & 0.634384 \\ 1.04856 & -0.196914 \\ 1.96066 & 3.11571 \\ -3.02046 & -1.96087 \end{bmatrix}}_{\mathbf{B}} \mathbf{u}_k$$

$$\mathbf{y}_k = \underbrace{\begin{bmatrix} 0.283441 & 0.032612 & -0.75658 & 0.085468 & 0.161088 \\ -0.528786 & 0.050734 & -0.681773 & -0.432334 & -1.17988 \end{bmatrix}}_{\mathbf{C}} \mathbf{z}_k$$

?

$=$

(2) Input-Output Similarity Checking

Extracted Model

$$\hat{\mathbf{z}}_{k+1} = \underbrace{\begin{bmatrix} 0.87224 & 0 & 0 & 0 & 0 \\ 0 & 0.366378 & 0 & 0 & 0 \\ 0 & 0 & -0.540795 & 0 & 0 \\ 0 & 0 & 0 & -0.332664 & 0 \\ 0 & 0 & 0 & 0 & -0.204322 \end{bmatrix}}_{\hat{\mathbf{A}}} \hat{\mathbf{z}}_k + \underbrace{\begin{bmatrix} 0.822174 & -0.438008 \\ -0.278536 & -0.824313 \\ 0.874484 & 0.858857 \\ -0.117628 & -0.506362 \\ -0.955459 & -0.622498 \end{bmatrix}}_{\hat{\mathbf{B}}} \mathbf{u}_k,$$

$$\mathbf{y}_k = \underbrace{\begin{bmatrix} -0.793176 & 0.154365 & -0.377883 & -0.360608 & -0.142123 \\ 0.503767 & -0.573538 & 0.170245 & -0.583312 & -0.56603 \end{bmatrix}}_{\hat{\mathbf{C}}} \hat{\mathbf{z}}_k$$

(1) Model Extraction

Code

```
void LTIS_step(void)
{
  {
    {
      static const int_T colCidxRow0[5] = { 0, 1, 2, 3, 4 };
      const int_T *pCidx = &colCidxRow0[0];
      const real_T *pC0 = LTIS_ConstP.Internal_C;
      const real_T *xd = &LTIS_DW.Internal_DSTATE[0];
      real_T *y0 = &LTIS_Y.y[0];
      int_T numNonZero = 4;
      *y0 = (*pC0++) * xd[*pCidx++];
      while (numNonZero--) {
        *y0 += (*pC0++) * xd[*pCidx++];
      }
    }
    {
      static const int_T colCidxRow1[5] = { 0, 1, 2, 3, 4 };
      const int_T *pCidx = &colCidxRow1[0];
      const real_T *pC5 = &LTIS_ConstP.Internal_C[5];
      const real_T *xd = &LTIS_DW.Internal_DSTATE[0];
      real_T *y1 = &LTIS_Y.y[1];
      int_T numNonZero = 4;
      *y1 = (*pC5++) * xd[*pCidx++];
      while (numNonZero--) {
        *y1 += (*pC5++) * xd[*pCidx++];
      }
    }
  }
  {
    real_T xnew[5];
    int_T i;
    xnew[0] = (0.87224)*LTIS_DW.Internal_DSTATE[0];
    xnew[0] += (0.822174)*LTIS_U.u[0]+(-0.438008)*LTIS_U.u[1];
    xnew[1] = (0.366378)*LTIS_DW.Internal_DSTATE[1];
    xnew[1] += (-0.278536)*LTIS_U.u[0]+(-0.824313)*LTIS_U.u[1];
    xnew[2] = (-0.540795)*LTIS_DW.Internal_DSTATE[2];
    xnew[2] += (0.874484)*LTIS_U.u[0]+(0.858857)*LTIS_U.u[1];
    xnew[3] = (-0.332664)*LTIS_DW.Internal_DSTATE[3];
    xnew[3] += (-0.117628)*LTIS_U.u[0]+(-0.506362)*LTIS_U.u[1];
    xnew[4] = (-0.204322)*LTIS_DW.Internal_DSTATE[4];
    xnew[4] += (-0.955459)*LTIS_U.u[0]+(-0.622498)*LTIS_U.u[1];
    for(i=0; i<5; i++) LTIS_DW.Internal_DSTATE[i] = xnew[i];
  }
}
```

J. Park, M. Pajic, I. Lee, and O. Sokolsky, “Scalable Verification of Linear Controller Software",
Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 2016

Defining Invariants for Linear Controllers

- Annotating input-output and state invariants
- Annotating input-output only invariants
- Inexact controller implementations
- Instantiation-based input-output invariants

Problem: How to check input-output conformance when state conformance is violated?

Duke PRATT SCHOOL OF Engineering

- Input-output invariants obtained from controllers ***transfer functions***

$$\Sigma = (\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}) \qquad \mathbf{G}(z) = \mathbf{C}(z\mathbf{I}_n - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}$$

- In the general case for Single-Input-Single-Output controllers

$$G(z) = \frac{\beta_0 + \beta_1 z^{-1} + \cdots + \beta_n z^{-n}}{1 + \alpha_1 z^{-1} + \cdots + \alpha_n z^{-n}},$$

and the controllers inputs and outputs satisfy

$$y_k = \sum_{i=0}^{n} \beta_i u_{k-i} - \sum_{i=1}^{n} \alpha_i y_{k-i}$$

with $y_k = 0, k < 0$ because $\mathbf{z}_0 = 0$ and $u_k = 0,\ k < 0$

Annotating Input-Output Only Invariants

- Cannot be specified using pre- and post-conditions for every execution of the `step` function
  - relates the last *n*+1 executions of the step function

$$y_k = \sum_{i=0}^{n} \beta_i u_{k-i} - \sum_{i=1}^{n} \alpha_i y_{k-i}$$

- Perform execution unrolling of the `step` function
  - construct the `verif_driver` function invoking the `step` function exactly *n*+1 times

$$\backslash{*}@\ \texttt{assert}\ \backslash\texttt{at}(\texttt{y}, k_n) + \alpha_1 * \backslash\texttt{at}(\texttt{y}, k_{n-1}) + \ldots$$
$$@\ \alpha_n * \backslash\texttt{at}(\texttt{y}, k_0) == \beta_0 * \backslash\texttt{at}(\texttt{u}, k_n) + \ldots$$
$$@\ \beta_n * \backslash\texttt{at}(\texttt{u}, k_0)$$

Annotating Input-Output Only Invariants

Duke PRATT SCHOOL OF Engineering

- Cannot be specified using pre- and post-conditions for every execution of the `step` function
  - relates the last *n*+1 executions of the step function
- Perform execution unrolling of the `step` function
  - construct the `verif_driver` function invoking the `step` function exactly *n*+1 times

$$y_k = \sum_{i=0}^{n} \beta_i u_{k-i} - \sum_{i=1}^{n} \alpha_i y_{k-i}$$

```
extern double input();

void verif_driver() {
  u = input();    step();
  k0::

  u = input();    step();
  k1::

  u = input();    step();
  k2::

  /*@assert \at(y,k2) - 4.4558*\at(y, k1)
    @ - 0.08007125*\at(y, k0)
    @ == 5.60030931*\at(u, k1)
    @ - 14.233777166248*\at(u, k0);
    @ */
}
```

$$\backslash{*}@\ \texttt{assert}\ \backslash\texttt{at}(y,k_n) + \alpha_1 * \backslash\texttt{at}(y,k_{n-1}) + ..$$
$$@\ \alpha_n * \backslash\texttt{at}(y,k_0) == \beta_0 * \backslash\texttt{at}(u,k_n) + ...$$
$$@\ \beta_n * \backslash\texttt{at}(u,k_0)$$

$$\mathbf{A} = \begin{bmatrix} 0.8147 & 1.1534 \\ 2.6413 & 3.6411 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 3.1019 \\ 2.1432 \end{bmatrix}, \mathbf{C} = \begin{bmatrix} 1.7121 & 0.1351 \end{bmatrix}$$

$$G(z) = \frac{5.60030931z^{-1} - 14.233777166248z^{-2}}{1 - 4.4558z^{-1} - 0.08007125z^{-2}}$$

Errors from optimization in code generation

Duke PRATT SCHOOL OF Engineering

- Back to the running example
  - a more efficient controller obtained in Matlab using the function canon for the modal type of decomposition

$$\mathbf{A} = \begin{bmatrix} 0.8147 & 1.1534 \\ 2.6413 & 3.6411 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 3.1019 \\ 2.1432 \end{bmatrix}, \mathbf{C} = \begin{bmatrix} 1.7121 & 0.1351 \end{bmatrix}$$

$$G(z) = \frac{5.60030931z^{-1} - 14.233777166248z^{-2}}{1 - 4.4558z^{-1} \boxed{- 0.08007125z^{-2}}}$$

$$\hat{\mathbf{A}} = \begin{bmatrix} -0.0179 & 0 \\ 0 & 4.474 \end{bmatrix}, \hat{\mathbf{B}} = \begin{bmatrix} -1.051 \\ -1.055 \end{bmatrix}, \hat{\mathbf{C}} = \begin{bmatrix} -3.037 & -2.283 \end{bmatrix}$$

$$G(\hat{z}) = \frac{5.600452z^{-1} - 14.2373891245z^{-2}}{1 - 4.4561z^{-1} \boxed{- 0.0800846z^{-2}}}$$

Defining Invariants for Linear Controllers

Duke PRATT SCHOOL OF Engineering

- Annotating input-output and state invariants
- Annotating input-output only invariants
- Inexact controller implementations
- Instantiation-based input-output invariants

Inexact Controller Implementations

- There is a need to extend our input-output invariants for the case with imprecise specification of the transfer functions

- Start by assuming that the transfer function could take the form

$$G(z) = \frac{\hat{\beta}_0 + \hat{\beta}_1 z^{-1} + \cdots + \hat{\beta}_n z^{-n}}{1 + \hat{\alpha}_1 z^{-1} + \cdots + \hat{\alpha}_n z^{-n}},$$

such that $i = 0, 1, \ldots, n$

$$\beta_i - \epsilon_\beta \leq \hat{\beta}_i \leq \beta_i + \epsilon_\beta, \quad \alpha_i - \epsilon_\alpha \leq \hat{\alpha}_i \leq \alpha_i + \epsilon_\alpha.$$

- `Inexact' invariant

$$\exists \Delta\beta_i, \Delta\alpha_i \in \mathbb{R}, i = 0, \ldots, n, \;\; |\Delta\beta_i| \leq \epsilon_\beta \wedge |\Delta\alpha_i| \leq \epsilon_\alpha \wedge$$

$$y_k = \sum_{i=0}^{n} (\beta_i + \boxed{\Delta\beta_i) u_{k-i}} - \sum_{i=1}^{n} (\alpha_i + \Delta\alpha_i) y_{k-i}$$

nonlinear

Inexact Controller Implementations

Duke PRATT SCHOOL OF Engineering

- There is a need to extend our input-output invariants for the case with imprecise specification of the transfer functions

- Start by assuming that the transfer function could take the form

$$G(z) = \frac{\hat{\beta}_0 + \hat{\beta}_1 z^{-1} + \cdots + \hat{\beta}_n z^{-n}}{1 + \hat{\alpha}_1 z^{-1} + \cdots + \hat{\alpha}_n z^{-n}},$$

such that $i = 0, 1, \ldots, n$

$$\beta_i - \epsilon_\beta \leq \hat{\beta}_i \leq \beta_i + \epsilon_\beta, \quad \alpha_i - \epsilon_\alpha \leq \hat{\alpha}_i \leq \alpha_i + \epsilon_\alpha.$$

- \`Inexact' ***linear*** invariant

$$\exists \tilde{u}_{k-i}, \tilde{y}_{k-i} \in \mathbb{R}, i = 0, 1, \ldots, n,$$

$$|\tilde{u}_{k-i}| \leq \epsilon_\beta |u_{k-i}| \wedge |\tilde{y}_{k-i}| \leq \epsilon_\alpha |y_{k-i}| \sim \wedge$$

$$y_k = \sum_{i=0}^{n} (\beta_i u_{k-i} + \tilde{u}_{k-i}) - \sum_{i=1}^{n} (\alpha_i y_{k-i} + \tilde{y}_{k-i})$$

$$\tilde{u}_{k-i} = \Delta\beta_i u_{k-i}, \quad \tilde{y}_{k-i} = \Delta\alpha_i y_{k-i}$$

Inexact Controller Implementations

Duke PRATT SCHOOL OF Engineering

- Start by assuming that the transfer function could take the form

$$G(z) = \frac{\hat{\beta}_0 + \hat{\beta}_1 z^{-1} + \cdots + \hat{\beta}_n z^{-n}}{1 + \hat{\alpha}_1 z^{-1} + \cdots + \hat{\alpha}_n z^{-n}},$$

such that $i = 0, 1, \ldots, n$

$$\beta_i - \epsilon_\beta \leq \hat{\beta}_i \leq \beta_i + \epsilon_\beta, \quad \alpha_i - \epsilon_\alpha \leq \hat{\alpha}_i \leq \alpha_i + \epsilon_\alpha.$$

- \`Inexact' ***linear*** invariant – ***for all*** u_{k-i} y_{k-i}

$$\exists \tilde{u}_{k-i}, \tilde{y}_{k-i} \in \mathbb{R}, i = 0, 1, \ldots, n,$$
$$|\tilde{u}_{k-i}| \leq \epsilon_\beta |u_{k-i}| \wedge |\tilde{y}_{k-i}| \leq \epsilon_\alpha |y_{k-i}| \sim \wedge$$
$$y_k = \sum_{i=0}^{n} (\beta_i u_{k-i} + \tilde{u}_{k-i}) - \sum_{i=1}^{n} (\alpha_i y_{k-i} + \tilde{y}_{k-i})$$

$$\tilde{u}_{k-i} = \Delta\beta_i u_{k-i}, \quad \tilde{y}_{k-i} = \Delta\alpha_i y_{k-i}$$

A mixture of both universal and existential quantifiers

Defining Invariants for Linear Controllers

Duke PRATT SCHOOL OF Engineering

- Annotating input-output and state invariants
- Annotating input-output only invariants
- Inexact controller implementations
- Instantiation-based input-output invariants

Instantiation-based Input-Output Invariants

Duke PRATT SCHOOL OF Engineering

$$\mathbf{D}_N = \begin{bmatrix} y_n & y_{n-1} & \dots & y_1 & y_0 & u_n & u_{n-1} & \dots & u_1 & u_0 \\ y_{n+1} & y_n & \dots & y_2 & y_1 & u_{n+1} & u_n & \dots & u_2 & u_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ y_{n+N-1} & y_{n+N-1} & \dots & y_N & y_{N-1} & u_{n+N-1} & u_{n+N-2} & \dots & u_N & u_{N-1} \end{bmatrix}$$

$$\mathbf{D}_N \cdot \theta = \mathbf{0}$$

$$\theta = [1 \quad \alpha_1 \quad \dots \quad \alpha_n \quad \beta_0 \quad \beta_1 \quad \dots \quad \beta_n]^T$$

$$y_k = \sum_{i=0}^{n} \beta_i u_{k-i} - \sum_{i=1}^{n} \alpha_i y_{k-i}$$

PROPOSITION 1. *Consider LTI controller Σ of size n. Then the rank of any matrix $\mathbf{D}_N$ cannot be larger than $2n+1$. Furthermore, when the rank of $\mathbf{D}_N$ is $2n+1$, then linear conditions $\mathbf{D}_N \cdot \theta = \mathbf{0}$ are satisfied if and only if the condition (8) is satisfied for all k.*

$$N = 2n+1$$

Instantiation-based Input-Output Invariants

$$\mathbf{D}_N = \begin{bmatrix} y_n & y_{n-1} & \dots & y_1 & y_0 & u_n & u_{n-1} & \dots & u_1 & u_0 \\ y_{n+1} & y_n & \dots & y_2 & y_1 & u_{n+1} & u_n & \dots & u_2 & u_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ y_{n+N-1} & y_{n+N-1} & \dots & y_N & y_{N-1} & u_{n+N-1} & u_{n+N-2} & \dots & u_N & u_{N-1} \end{bmatrix}$$

$$\mathbf{D}_N \cdot \theta = \mathbf{0}$$

$$\theta = [1 \quad \alpha_1 \quad \dots \quad \alpha_n \quad \beta_0 \quad \beta_1 \quad \dots \quad \beta_n]^T$$

$$y_k = \sum_{i=0}^{n} \beta_i u_{k-i} - \sum_{i=1}^{n} \alpha_i y_{k-i}$$

Code annotations

```
\*@ assert ((\at(y,k_0)==y_0)&&...&&(\at(y,k_{n-1})==y_{n-1}) && (\at(u,k_0)==u_0 )&&...&&(\at(u,k_{3n})==u_{3n}))
 @ ⇒ ((\at(y,k_n) == y_n) && ... && (\at(y,k_{3n})==y_{3n}))
```

Allows us to specify a set of $2n + 1$ linear invariants

Instantiation-based Input-Output Invariants

$$\mathbf{D}_N = \begin{bmatrix} y_n & y_{n-1} & \dots & y_1 & y_0 & u_n & u_{n-1} & \dots & u_1 & u_0 \\ y_{n+1} & y_n & \dots & y_2 & y_1 & u_{n+1} & u_n & \dots & u_2 & u_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ y_{n+N-1} & y_{n+N-1} & \dots & y_N & y_{N-1} & u_{n+N-1} & u_{n+N-2} & \dots & u_N & u_{N-1} \end{bmatrix}$$

$$\mathbf{D}_N \cdot \theta = \mathbf{0}$$

$$\theta = [1 \quad \alpha_1 \quad \dots \quad \alpha_n \quad \beta_0 \quad \beta_1 \quad \dots \quad \beta_n]^T$$

$$y_k = \sum_{i=0}^{n} \beta_i u_{k-i} - \sum_{i=1}^{n} \alpha_i y_{k-i}$$

Code annotations for inexact controllers

$$\exists \Delta\beta_i, \Delta\alpha_i \in \mathbb{R}, i = 0, ..., n, |\Delta\beta_i| \le \epsilon_\beta \wedge |\Delta\alpha_i| \le \epsilon_\alpha \wedge$$
$$\mathbf{D}^y_{2n+1}\Delta\alpha + \mathbf{D}^u_{2n+1}\Delta\beta = \mathbf{v} \ \wedge$$
$$y_{n+i} = \mathbf{D}^y_{2n+1}(n+i), \ \ i = 0, ..., 2n,$$

*@ assert \exists real $a_0, ..., a_{n-1}, b_0, ..., b_n$

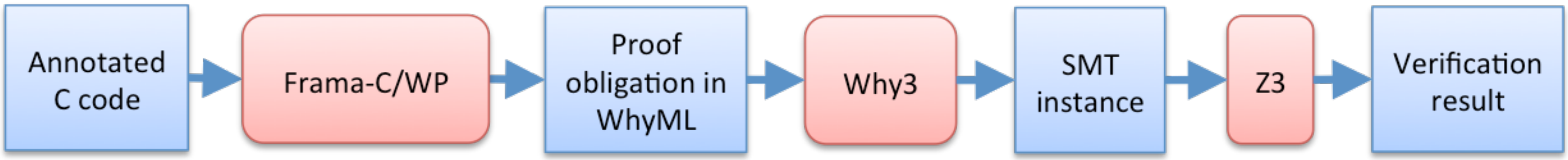
@ $(a_0 \le \epsilon_\alpha)$ && $(a_0 \ge -\epsilon_\alpha)$ && . . . && $(a_{n-1} \le \epsilon_\alpha)$ && $(a_{n-1} \ge -\epsilon_\alpha)$ && $(b_0 \le \epsilon_\beta)$ && $(b_0 \ge -\epsilon_\beta)$ && . . . && $(b_n \le \epsilon_\beta)$ && $(b_n \ge -\epsilon_\beta)$

*@ ((\at(y,k_0)==y_0) &&...&& (\at(y,k_{n-1})==y_{n-1}) && (\at(u,k_0) ==u_0) &&...&& (\at(u,k_{3n})==u_{3n}))

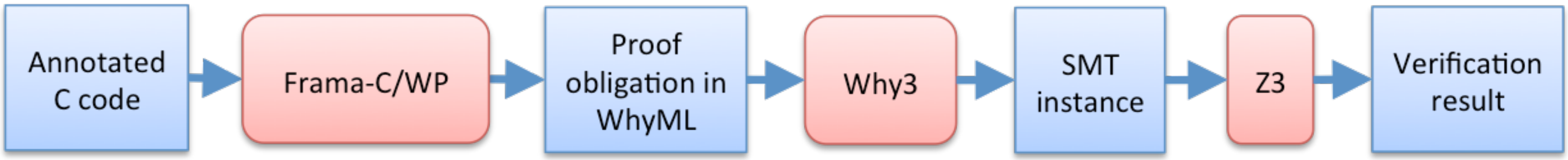
@ $\Rightarrow$ ((\at(y,k_n) == y_n) && ... && (\at(y,k_{3n})==y_{3n}) &&

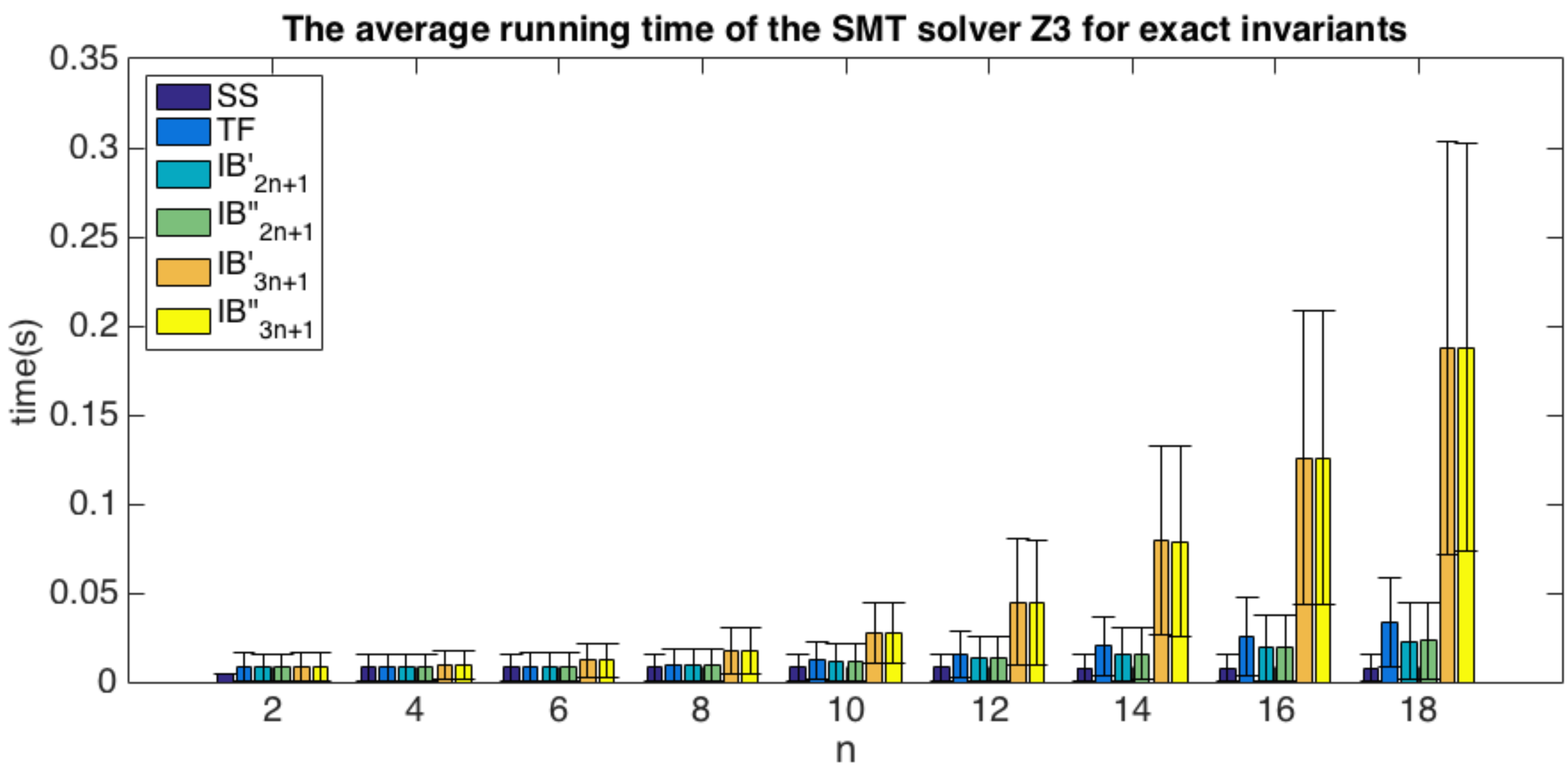
*@ vector_equal((lin_comb($Dy, 1, a_0, ..., a_{n-1}$) + lin_comb($Du, b_0, ..., b_n$)),v))

Framework For Automatic Verification

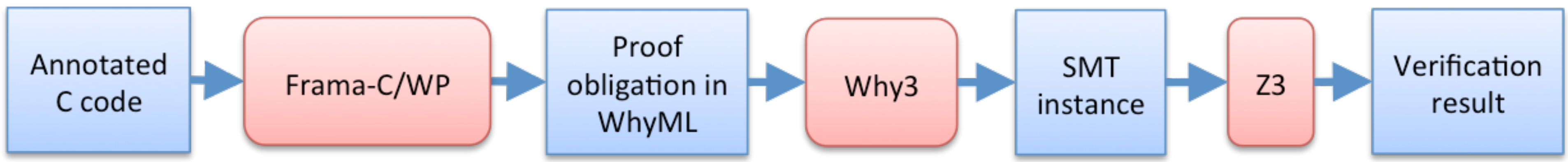


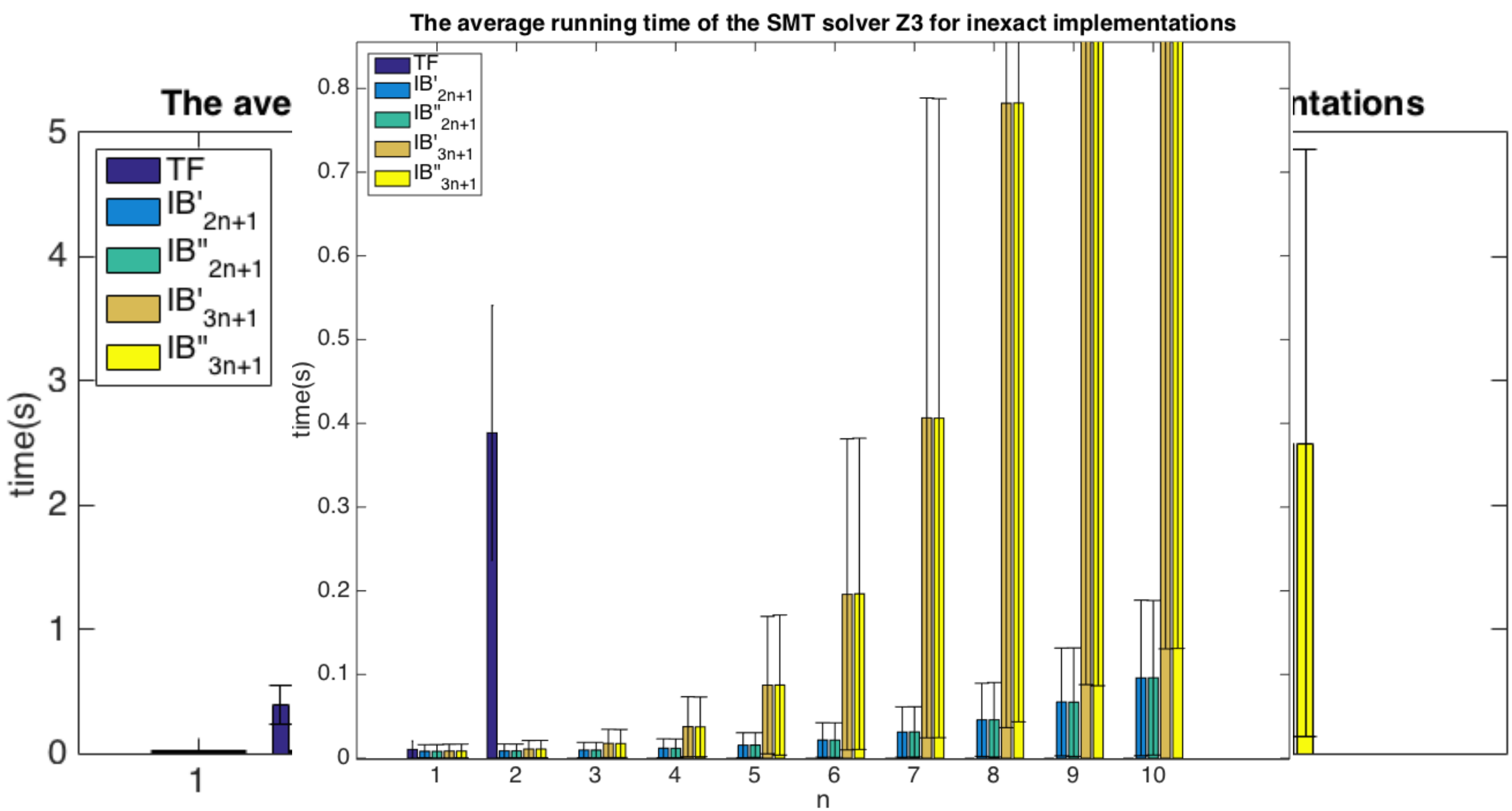
Automatic Verification for Exact Invariants





Automatic Verification – Inexact Invariants





A More Scalable Approach (SC)

Duke
PRATT SCHOOL OF
Engineering

Original Model

$$\mathbf{z}_{k+1} = \underbrace{\begin{bmatrix} -0.500311 & 0.16751 & 0.028029 & -0.395599 & -0.652079 \\ 0.850942 & 0.181639 & -0.29276 & 0.481277 & 0.638183 \\ -0.458583 & -0.002389 & -0.154281 & -0.578708 & -0.769495 \\ 1.01855 & 0.638926 & -0.668256 & -0.258506 & 0.119959 \\ 0.100383 & -0.432501 & 0.122727 & 0.82634 & 0.892296 \end{bmatrix}}_{\mathbf{A}} \mathbf{z}_k + \underbrace{\begin{bmatrix} 1.1149 & 0.164423 \\ -1.56592 & 0.634384 \\ 1.04856 & -0.196914 \\ 1.96066 & 3.11571 \\ -3.02046 & -1.96087 \end{bmatrix}}_{\mathbf{B}} \mathbf{u}_k$$

$$\mathbf{y}_k = \underbrace{\begin{bmatrix} 0.283441 & 0.032612 & -0.75658 & 0.085468 & 0.161088 \\ -0.528786 & 0.050734 & -0.681773 & -0.432334 & -1.17988 \end{bmatrix}}_{\mathbf{C}} \mathbf{z}_k$$

?
=

(2) Input-Output Similarity Checking

Extracted Model

$$\hat{\mathbf{z}}_{k+1} = \underbrace{\begin{bmatrix} 0.87224 & 0 & 0 & 0 & 0 \\ 0 & 0.366378 & 0 & 0 & 0 \\ 0 & 0 & -0.540795 & 0 & 0 \\ 0 & 0 & 0 & -0.332664 & 0 \\ 0 & 0 & 0 & 0 & -0.204322 \end{bmatrix}}_{\hat{\mathbf{A}}} \hat{\mathbf{z}}_k + \underbrace{\begin{bmatrix} 0.822174 & -0.438008 \\ -0.278536 & -0.824313 \\ 0.874484 & 0.858857 \\ -0.117628 & -0.506362 \\ -0.955459 & -0.622498 \end{bmatrix}}_{\hat{\mathbf{B}}} \mathbf{u}_k,$$

$$\mathbf{y}_k = \underbrace{\begin{bmatrix} -0.793176 & 0.154365 & -0.377883 & -0.360608 & -0.142123 \\ 0.503767 & -0.573538 & 0.170245 & -0.583312 & -0.56603 \end{bmatrix}}_{\hat{\mathbf{C}}} \hat{\mathbf{z}}_k$$

(1) Model Extraction

Code

```
void LTIS_step(void)
{
  {
    {
      static const int_T colCidxRow0[5] = { 0, 1, 2, 3, 4 };
      const int_T *pCidx = &colCidxRow0[0];
      const real_T *pC0 = LTIS_ConstP.Internal_C;
      const real_T *xd = &LTIS_DW.Internal_DSTATE[0];
      real_T *y0 = &LTIS_Y.y[0];
      int_T numNonZero = 4;
      *y0 = (*pC0++) * xd[*pCidx++];
      while (numNonZero--) {
        *y0 += (*pC0++) * xd[*pCidx++];
      }
    }
    {
      static const int_T colCidxRow1[5] = { 0, 1, 2, 3, 4 };
      const int_T *pCidx = &colCidxRow1[0];
      const real_T *pC5 = &LTIS_ConstP.Internal_C[5];
      const real_T *xd = &LTIS_DW.Internal_DSTATE[0];
      real_T *y1 = &LTIS_Y.y[1];
      int_T numNonZero = 4;
      *y1 = (*pC5++) * xd[*pCidx++];
      while (numNonZero--) {
        *y1 += (*pC5++) * xd[*pCidx++];
      }
    }
  }
  {
    real_T xnew[5];
    int_T i;
    xnew[0] = (0.87224)*LTIS_DW.Internal_DSTATE[0];
    xnew[0] += (0.822174)*LTIS_U.u[0]+(-0.438008)*LTIS_U.u[1];
    xnew[1] = (0.366378)*LTIS_DW.Internal_DSTATE[1];
    xnew[1] += (-0.278536)*LTIS_U.u[0]+(-0.824313)*LTIS_U.u[1];
    xnew[2] = (-0.540795)*LTIS_DW.Internal_DSTATE[2];
    xnew[2] += (0.874484)*LTIS_U.u[0]+(0.858857)*LTIS_U.u[1];
    xnew[3] = (-0.332664)*LTIS_DW.Internal_DSTATE[3];
    xnew[3] += (-0.117628)*LTIS_U.u[0]+(-0.506362)*LTIS_U.u[1];
    xnew[4] = (-0.204322)*LTIS_DW.Internal_DSTATE[4];
    xnew[4] += (-0.955459)*LTIS_U.u[0]+(-0.622498)*LTIS_U.u[1];
    for(i=0; i<5; i++) LTIS_DW.Internal_DSTATE[i] = xnew[i];
  }
}
```

Model Extraction

Duke PRATT SCHOOL OF Engineering

- Use symbolic execution to identify transition relation

fragment of step function

```
const ConstP_LTIS_T LTIS_ConstP = {
  { -0.793176, 0.154365, -0.377883, -0.360608, -0.142123,
    0.503767, -0.573538, 0.170245, -0.583312, -0.56603 } };
static const int_T colCidxRow0[5] = { 0, 1, 2, 3, 4 };
const int_T *pCidx = &colCidxRow0[0];
const real_T *pC0 = LTIS_ConstP.Internal_C;
const real_T *xd = &LTIS_DW.Internal_DSTATE[0];
real_T *y0 = &LTIS_Y.y[0];
int_T numNonZero = 4;
*y0 = (*pC0++) * xd[*pCidx++];
while (numNonZero--) {
  *y0 += (*pC0++) * xd[*pCidx++];
}
```

symbolic execution

big-step transition relation

$$
\begin{aligned}
\mathrm{y}[0]^{(new)} = {} & (((((-0.793176 \cdot \mathrm{x}[0]) + (0.154365 \cdot \mathrm{x}[1])) + (-0.377883 \cdot \mathrm{x}[2])) \\
& +(-0.360608 \cdot \mathrm{x}[3])) + (-0.142123 \cdot \mathrm{x}[4]))
\end{aligned}
$$

y stands for **LTIS_Y.y**, and **x** stands for **LTIS_DW.Internal_DSTATE**

Model Extraction (cont.)

Duke PRATT SCHOOL OF Engineering

- Identify the set of state variables V_{state}

$$V_{state} = (V_{updated} \setminus V_{output}) \cup (V_{used} \setminus V_{input})$$

- Transform into matrix form

transition relation

$$\mathtt{y}[0]^{(new)} = (((((-0.793176 \cdot \mathtt{x}[0]) + (0.154365 \cdot \mathtt{x}[1])) + (-0.377883 \cdot \mathtt{x}[2])) + (-0.360608 \cdot \mathtt{x}[3])) + (-0.142123 \cdot \mathtt{x}[4]))$$

canonical form

$$= -0.793176 \cdot \mathtt{x}[0] + 0.154365 \cdot \mathtt{x}[1] + -0.377883 \cdot \mathtt{x}[2] + -0.360608 \cdot \mathtt{x}[3] + -0.142123 \cdot \mathtt{x}[4]$$

vector form

$$= [-0.793176, 0.154365, -0.377883, -0.360608, -0.142123] \cdot \mathbf{x} \quad + [0, 0] \cdot \mathbf{u}$$

matrix form

$$\mathbf{x}^{(new)} = \hat{\mathbf{A}}\mathbf{x} + \hat{\mathbf{B}}\mathbf{u}$$
$$\mathbf{y}^{(new)} = \hat{\mathbf{C}}\mathbf{x} + \hat{\mathbf{D}}\mathbf{u}$$

Input-output equivalence checking

Duke PRATT SCHOOL OF Engineering

- Check similarity between two models

Two minimal LTI models $\Sigma(A, B, C, D)$ and $\hat{\Sigma}(\hat{A}, \hat{B}, \hat{C}, \hat{D})$ are input-output equivalent iff there exists a non-singular matrix T such that

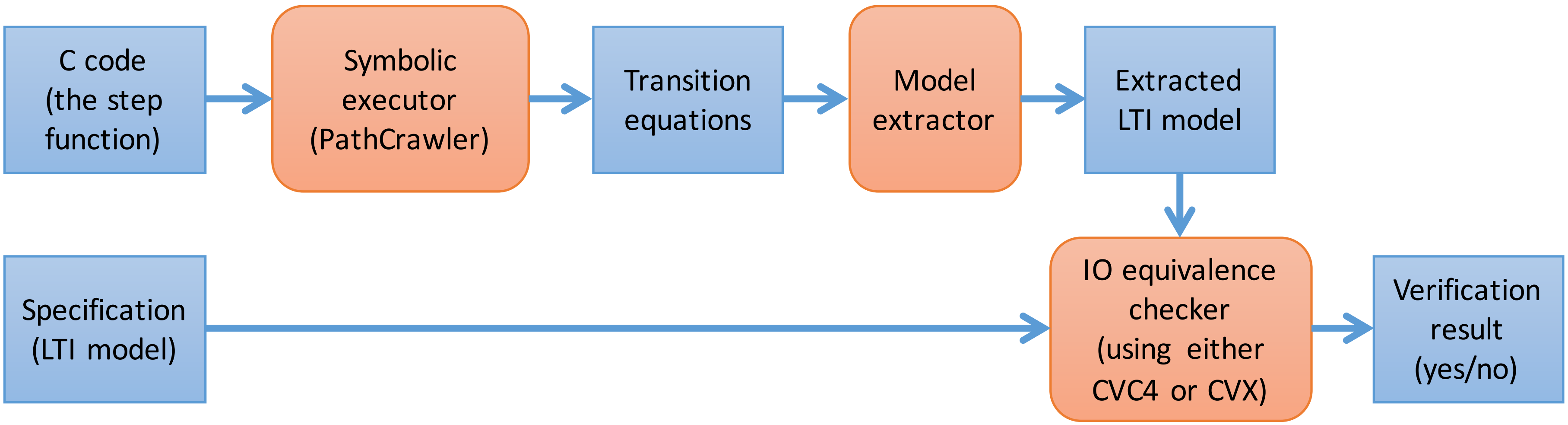
$$\hat{\mathbf{A}} = \mathbf{TAT}^{-1}, \qquad \hat{\mathbf{B}} = \mathbf{TB}, \qquad \hat{\mathbf{C}} = \mathbf{CT}^{-1}, \qquad \text{and} \qquad \hat{\mathbf{D}} = \mathbf{D}$$

- Find the existence of similarity transformation matrix using
  - SMT formulation approach
  - Convex optimization formulation approach

- Need to tolerate the numerical errors on the model parameters

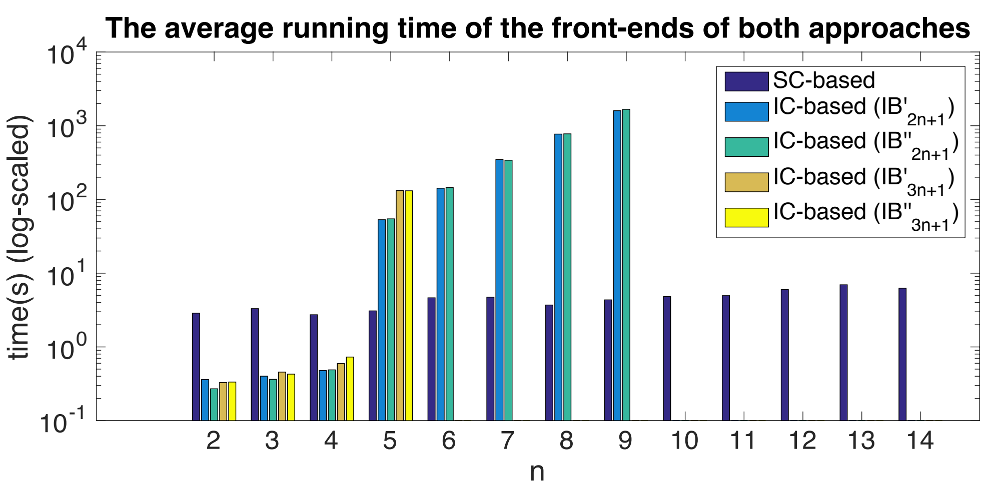
Verification Toolchain

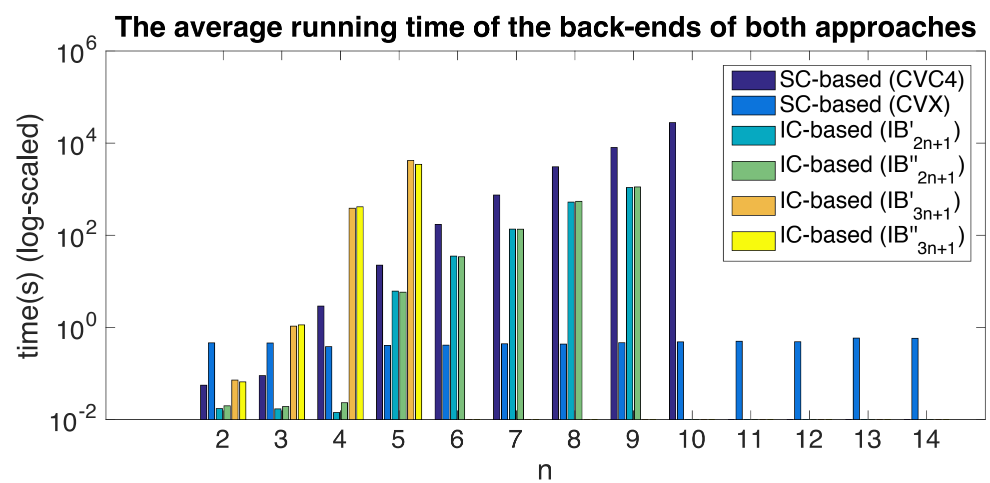
- Similarity Checking (SC)-based approach



Evaluation

- Compare scalability of the two approaches
  - Random LTI models with a range of state sizes
  - Code obtained by Simulink Coder
- Similarity-checking approach (SC) dramatically outperforms invariant-checking approach (IC)





Current work

Duke PRATT SCHOOL OF Engineering

- Focus on more complex controllers
  - Convex optimization-based controllers

Thank You