Static Stability Analysis of Autocoded Software for Aviation Systems

Eric Feron and Arnaud Venet
Georgia Tech and Kestrel Technologies

October 6, 2006

Core message

Modern model-based design environments such as Matlab/Simulink have auto coding capabilities. These auto coding capabilities can substantially modify the properties of the initial design.

We are working on static analyzers aimed at proving that essential safety and functional properties of the design are not substantially affected by the auto coding process.

Outline

- Problem statement
- Abstract interpretation
- Autocode verification challenges
- Simple design example
  - Closed-loop stability: Spec. level
  - From spec-level to implementation-level
  - Implementation artifacts
- Need for good collecting semantics
- Conclusion

Problem statement

- Autocoding tools have made real-time executable embedded software available seconds away from design specifications
- There remains need for certification
- “Certified autocoder” concept is nearly impossible concept (“forall correct specification inputs, autocoded output is correct”)
- Static analysis, eg using Cousot’s abstract interpretation techniqes, can help answer more reasonable statement “for a given correct specification input, autocoded output is correct”

Abstract interpretation

- Proposed by Cousot in late 70’s
- Based on building conservative, but easier to analyze approximations of program behavior
- Similar in spirit to *robust control system analysis*
- ASTRÉE static analyzer (Cousot / Ecole Normale Supérieure, Paris) used to verify consistency of floating point arithmetic on A380
- C Global Surveyor (Kestrel Technology LLC) used to verify absence of pointer manipulation errors on NASA’s Mars rovers

Autocode verification challenges

1. How do we translate a property of the high-level model into a property of the code generated from a model? What are the features of the autocoder we must know to effectively build this translator?

   The autocoder of the model-based design environment generates code from the model in a predictable way. There are rich, structured libraries of basic components for building systems but properties of interest may greatly vary across systems.

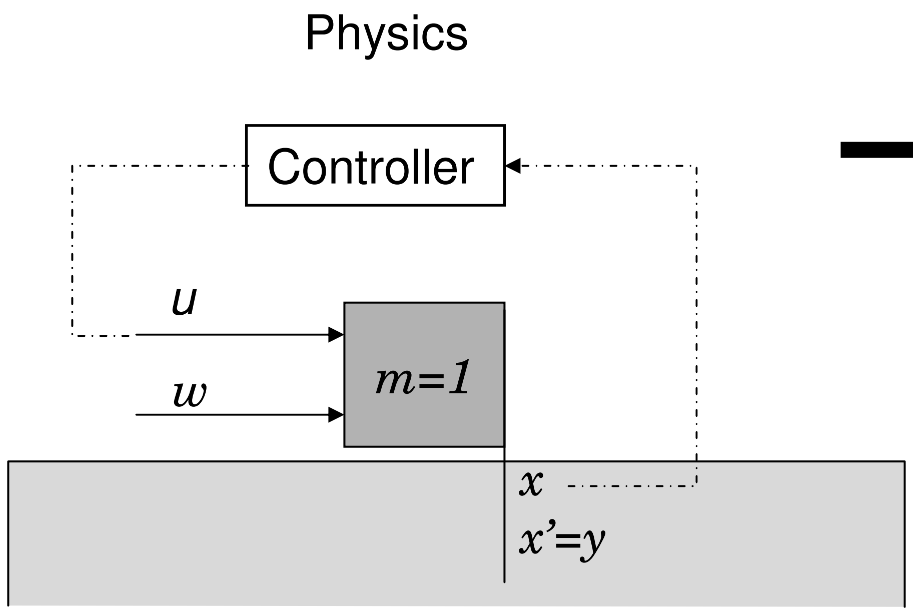
2. How do we specify the basic components of the static analyzer required to verify the desired property? How do we specialize the analyzer to the code generated by a given autocoder?

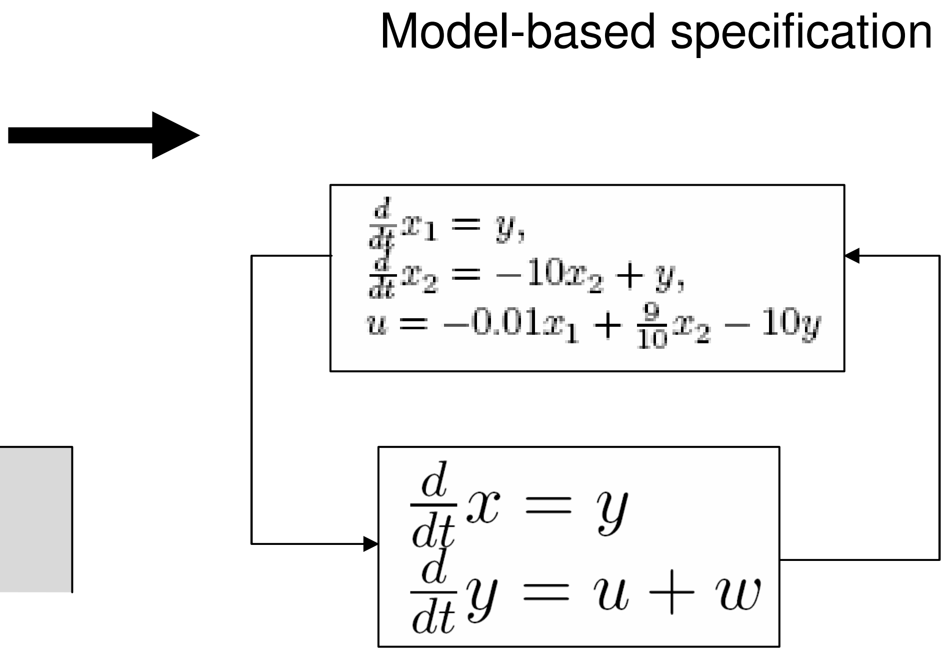
   Use Matlab/SimulinkTMtool suite as exprimental test-bed; study family of closed-loop dynamical systems: Differential equations that capture the system's physics + closed-loop control algorithm & code.

   Functional properties of interest: closed-loop systems stability, closed-loop system performance

Simple Design Example

- Simple dynamical system





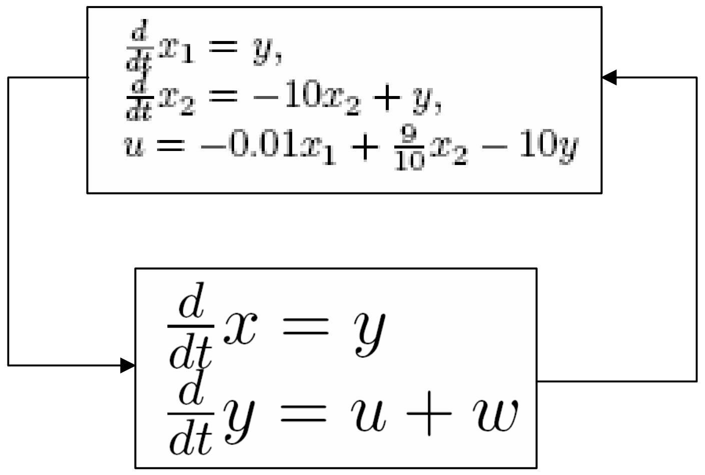
$$\frac{d}{dt}x_1 = y, \quad \frac{d}{dt}x_2 = -10x_2 + y, \quad u = -0.01x_1 + \frac{9}{10}x_2 - 10y$$

$$\frac{d}{dt}x = y, \quad \frac{d}{dt}y = u + w$$

(x, y)= position / velocity
u = controlled input force
w = exogenous disturbance force

Simple Design Example (ct'd)

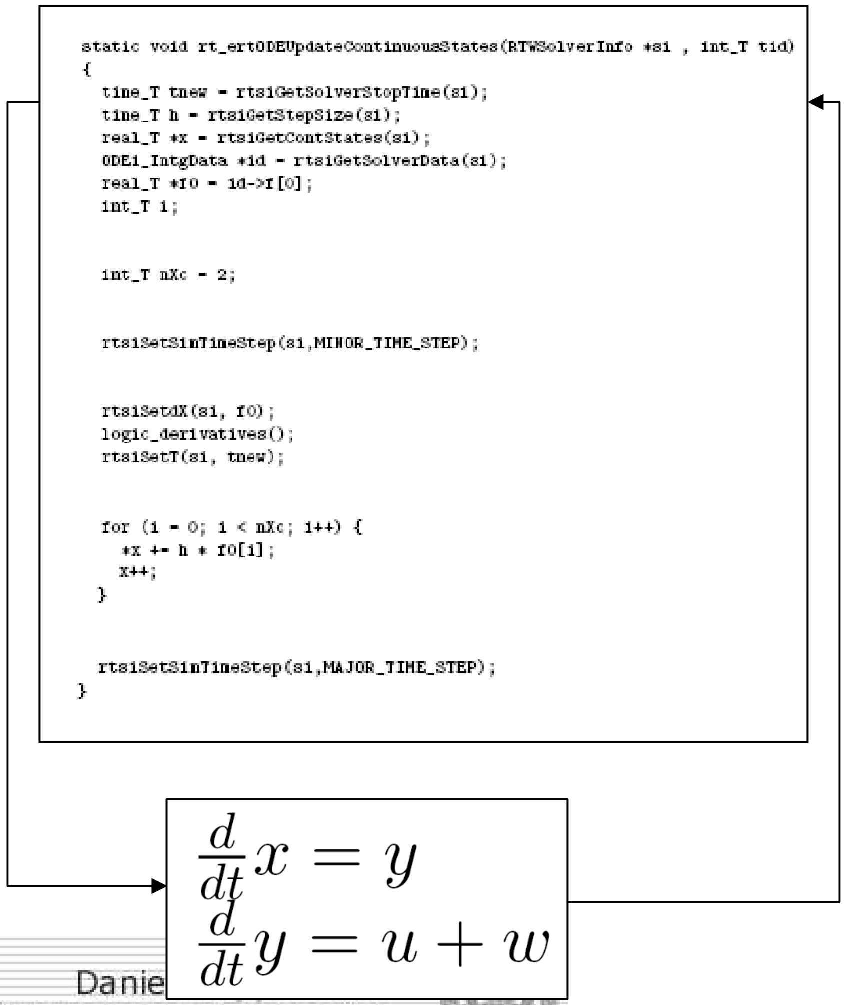
Model-based specification



$$\frac{d}{dt}x_1 = y,$$
$$\frac{d}{dt}x_2 = -10x_2 + y,$$
$$u = -0.01x_1 + \frac{9}{10}x_2 - 10y$$

$$\frac{d}{dt}x = y$$
$$\frac{d}{dt}y = u + w$$

Implementation



```
static void rt_ertODEUpdateContinuousStates(RTWSolverInfo *si , int_T tid)
{
  time_T tnew = rtsiGetSolverStopTime(si);
  time_T h = rtsiGetStepSize(si);
  real_T *x = rtsiGetContStates(si);
  ODE1_IntgData *id = rtsiGetSolverData(si);
  real_T *f0 = id->f[0];
  int_T i;


  int_T nXc = 2;


  rtsiSetSimTimeStep(si,MINOR_TIME_STEP);


  rtsiSetdX(si, f0);
  logic_derivatives();
  rtsiSetT(si, tnew);


  for (i = 0; i < nXc; i++) {
    *x += h * f0[i];
    x++;
  }


  rtsiSetSimTimeStep(si,MAJOR_TIME_STEP);
}
```

$$\frac{d}{dt}x = y$$
$$\frac{d}{dt}y = u + w$$

Closed loop stability (*w=0)* Specification level

- Use invariant function

$$V(x,y,x_1,x_2)=\begin{bmatrix}x\\y\\x_1\\x_2\end{bmatrix}^T\begin{bmatrix}768.5818 & -0.5000 & 0.8254 & -6.3254\\ -0.5000 & 82.5378 & 50.0000 & 6.7959\\ 0.8254 & 50.0000 & 495.5018 & 4.4932\\ -6.3254 & 6.7959 & 4.4932 & 0.6616\end{bmatrix}\begin{bmatrix}x\\y\\x_1\\x_2\end{bmatrix}$$

We have *V>0 (except at 0) and* $\frac{d}{dt}V \le 0$

To be exact: V positive-definite

$\rightarrow$ Stability

From system-level properties to implementation-level properties

- Ellipsoidal invariant tailored to continuous-time specification – Does it prove sampled-data implementation closed-loop stability?
- Impact of roundoff errors due to floating-point arithmetic?
- Designing efficient *abstract domains*

Implementation artifacts

- Many auxiliray static analyses to recover data structures, from layers and layers of pointers (probably "cover letters" due to successive Matlab/Simulink upgrades)
- Must perform pointer / variable range analysis to distinguish elements of arrays and index variables used to manipulating model data

What collecting semantics?

- Closed-loop system feels like

```
for(; {
  $x_+ = Ax$;
  $x = x_+$;
}
```

Invariant to prove sability i $V(x) = x^T P x$

Stability iff $A^T P A - P \leq 0$

(P symmetric, positive-definite matrix)

What collecting semantics? (ct'd)

Code implementation requires several lines of code, eg, instruction $x = x_+;$

is implemented as

```
for (i =0; i < n; i++) {
  *(x + i) = *(x_+ + i);
}
```

But evolution of *V*(*x*) may not be monotonic when looking at line-by-line evolution. In fact it is NEVER monotonic in most cases of interest. Substantially constrains the design of the collecting semantics and the static analysis: Execution traces of some loops must be collected and represented by one semantic object.

Conclusions

- Outlined a research program aimed at static analysis of autocoded software
- Outlined some of the issues and approaches
  - "Invariant invariance" across autocoding step
  - Collecting semantics must be adapted to properties of interest

Thanks to

- NSF: Embedded and Hybrid Systems program, GTech: Dutton/Ducoffe Professorship in Aerospace Software Engineering
- Patrick Cousot (ENS), Alexandre Megretski (MIT), Cesar Munoz (NASA-Langley), Rene Valenzuela (GTech)