Integrated simulation and formal verification of a simple autonomous vehicle

A. Domenici[1] A. Fagiolini[2] M. Palmieri[3] [1]

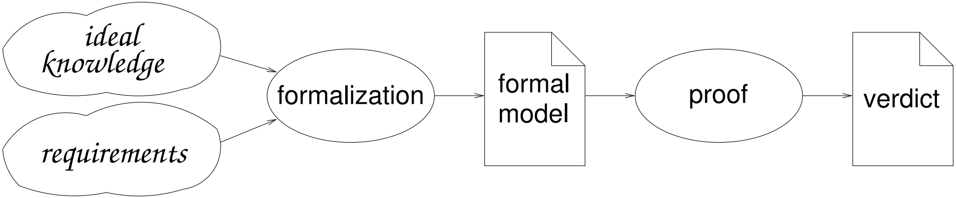
[1]Department of Information Engineering
University of Pisa, Italy

[2]Dipartimento di Energia, Ingegneria dell'Informazione e Modelli Matematici (DEIM), University of Palermo, Italy

[3]DINFO, University of Florence, Italy

1st Workshop on Formal Co-Simulation of Cyber-Physical Systems
A satellite event of SEFM2017
September 5, 2017, Trento, Italy

Verification

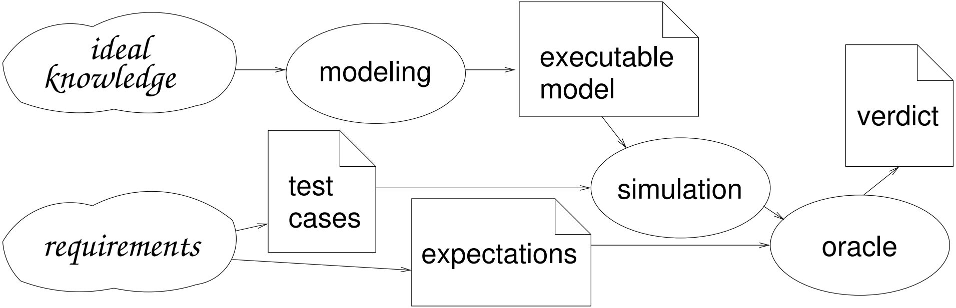


The verdict is valid under all circumstances.

But what if

- knowledge is not complete and accurate?
- requirements are fuzzy?
- formalization is incorrect?
- proof is faulty?

Simulation



The verdict is valid for each test case.

But what if

- knowledge is not complete and accurate?
- requirements are fuzzy?
- expected results are incorrect?
- the test cases do not cover all possible circumstances?

Integrated co-simulation and verification

Simulation and verification complement each other.

Verification provides results of **general validity**, **if** assumptions and inferences are correct.

Simulation validates assumptions under **given scenarios**, and also shows **explicitly** system behaviors that are only **implied** by formal models.

Co-simulation makes it possible to combine formal (declarative) models with executable ones.

Note:

- a formal model is expressed in a language oriented to verification, e.g., higher-order logic, temporal logic, dynamic logic, process algebras ...
- an executable model is usually expressed in a block-based graphical language, such as Simulink, Modelica, 20-Sim ...

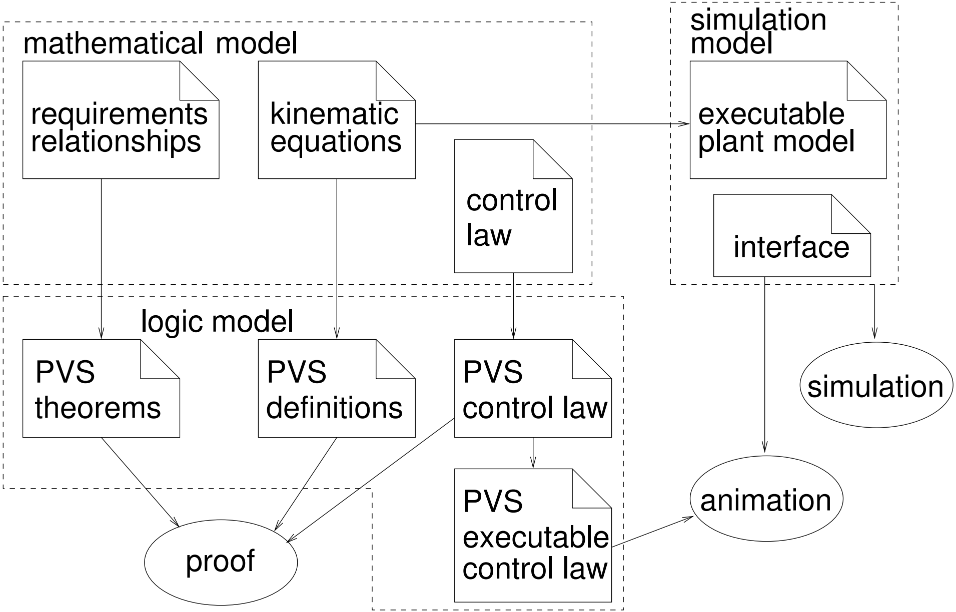
An approach to control system verification (1)

Most systems are composed of a plant subsystem and a control subsystem.

The approach proposed in this work assumes that

- the plant subsystem's behavior is given;
- developers of the control subsystem must verify that it performs its tasks correctly;
- both the plant and the control are defined by mathematical equations;
- the plant's equations have been implemented as an executable model.

An approach to control system verification (2)



An approach to control system verification (3)

Mathematical model: differential/algebraic equations from control theory.

Logic model: the mathematical model expressed in the higher-order logic PVS language (see below).

Simulation model: executable model of the plant's kinematics, plus an interface mechanism enabling co-simulation of logic models.

Animation: execution (simulation) of a logic model, using the PVSio ground evaluator (see below).

Background: The Prototype Verification System

Proving:

The PVS is an **interactive theorem prover** environment based on:

- A **typed higher-order logic** language
- a **sequent calculus** deduction system.

A PVS **theory** is a collection of definitions and statements, including axioms.

A PVS model is a collection of theories describing a system.

System requirements are expressed as **theorems** to be proved wrt the theory.

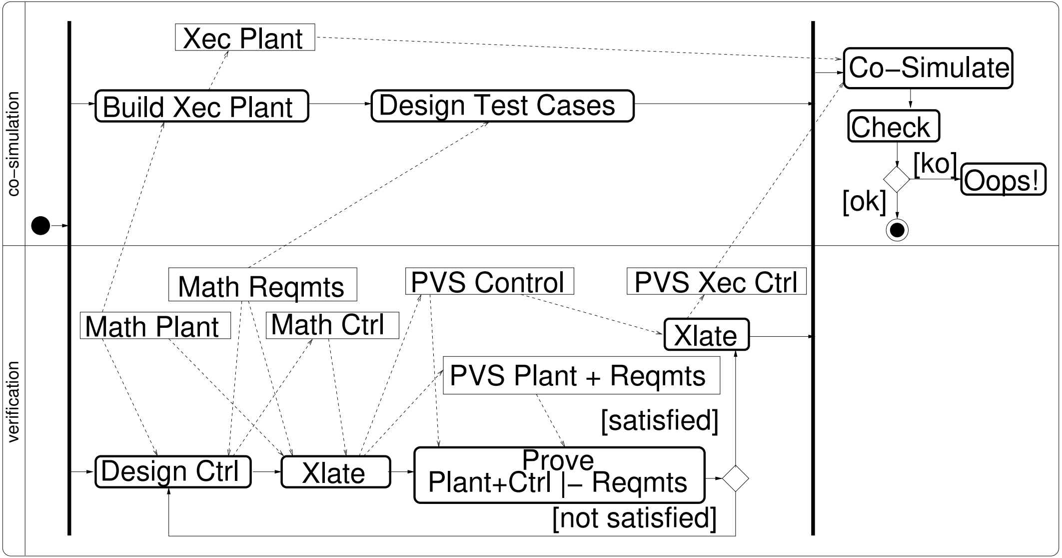
Animating:

The PVSio extension is a **ground evaluator** that translates PVS function definitions into LISP code.

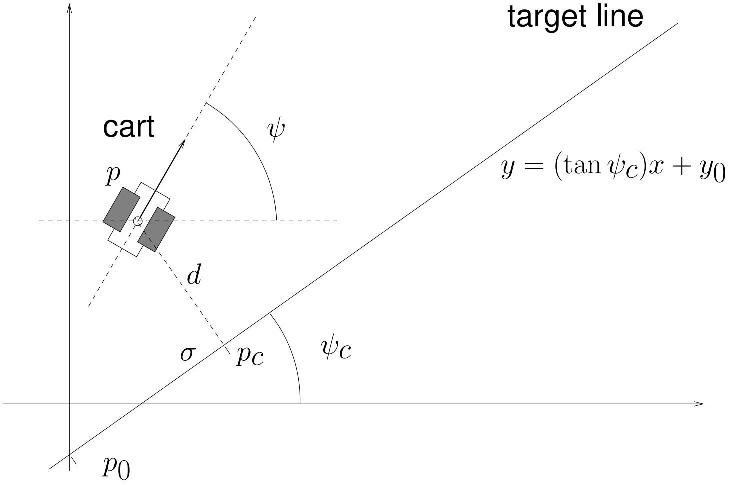
A PVS function definition may contain applications of **extra-logical** functions, providing, e.g., input and output.

A PVS model can then be animated, i.e., **simulated**.

A workflow


co–simulation
verification
Xec Plant
Build Xec Plant
Design Test Cases
Co–Simulate
Check
[ko]
Oops!
[ok]
Math Reqmts
PVS Control
PVS Xec Ctrl
Math Plant
Math Ctrl
Xlate
PVS Plant + Reqmts
[satisfied]
Design Ctrl
Xlate
Prove
Plant+Ctrl |– Reqmts
[not satisfied]

Example: a simple autonomous vehicle



kinematics:

$$\sigma = | p_c - p_0 |$$
$$\theta = \psi - \psi_c$$
$$d = | p - p_c |$$

$$\begin{cases} \dot{\sigma} = \mathrm{V} \cos\theta \\ \dot{d} = \mathrm{V} \sin\theta \\ \dot{\theta} = \dot{\psi} = \omega \end{cases}$$

control law:

$$\omega = -dv \operatorname{sinc}\theta - \mathrm{k}\theta$$

Requirement: reach and follow the target line without oscillations.

From math model to logic (1)

Partial derivatives of the generating functions:

$$\frac{\partial f_\sigma}{\partial \sigma} = 0 \quad \cdots \quad \frac{\partial f_\sigma}{\partial \theta} = -V \sin\theta \quad \cdots \quad \frac{\partial f_\theta}{\partial \theta} = -Vd(\frac{\theta\cos\theta - \sin\theta}{\theta^2}) - k$$

```
t: VAR nnreal
V, k: posreal
sigma(t), d(t), theta(t): real

dfsigma_dsigma(sigma, d, theta, t): real = 0
    ...
dfsigma_dtheta(sigma, d, theta, t): real = -V*sin(theta(t))
    ...
dftheta_dtheta(sigma, d, theta, t): real =
  -V*d(t)*(cos(theta(t))/theta(t) - sin(theta(t))/(theta(t))^2)
  - k
```

From math model to logic (2)

Jacobian, characteristic polynomial, and eigenvalues:

$$J_0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & V \\ 0 & -V & -k \end{bmatrix} \qquad P(\lambda) = -\lambda^3 - k\lambda^2 - V^2\lambda$$

$$\lambda_1 = -\frac{\sqrt{k^2-4V^2}+k}{2} \qquad \lambda_2 = \frac{\sqrt{k^2-4V^2}-k}{2} \qquad \lambda_3 = 0$$

```
J_0(sigma, d, theta, t) : bool =
    J(1, 1, sigma, d, theta, t) = 0 and
    ...  and J(3, 3, sigma, d, theta, t) = -k

char_J(lam: real) : real = -lam^3 - k*lam^2 - (V^2)*lam

lam_1: real = - (sqrt(k^2 - 4*V^2) + k)/2
lam_2: real =   (sqrt(k^2 - 4*V^2) - k)/2
lam_3: real =   0
```

From math model to logic (3)

Requirement: reach and follow the target line without oscillations, i.e.,
the eigenvalues are real and nonpositive

$$P(\lambda_1) = P(\lambda_2) = P(\lambda_3) = 0 \qquad \lambda_1, \lambda_2, \lambda_3 \in \mathbb{R}$$

$$\Re(\lambda_1) \leq 0 \qquad \Re(\lambda_2) \leq 0 \qquad \Re(\lambda_3) \leq 0$$

```
eigenvals: LEMMA
  k > 2*V implies
  char_J(lam_1) = 0 and char_J(lam_2) = 0 and char_J(lam_3) = 0

local_stability: THEOREM
  k > 2*V implies
  char_J(lam_1) = 0 and char_J(lam_2) = 0 and char_J(lam_3) = 0
  and lam_1 < 0 and lam_2 < 0
```

A (very) simple proof

```
  |-------
{1}   k > 2*V IMPLIES lam_1 < 0

Rerunning step: (flatten)

{-1}  k > 2*V
  |-------
{1}   lam_1 < 0

Rerunning step: (expand "lam_1")

[-1]  k > 2*V
  |-------
{1}   -(sqrt(k^2 - 4*V^2) + k)/2 < 0

Rerunning step: (assert)

Q.E.D.
```

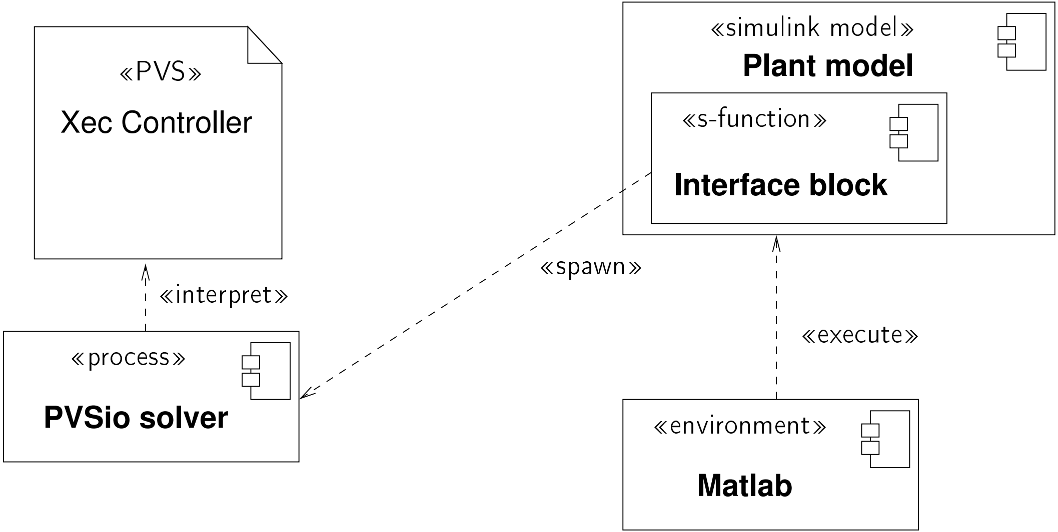
Animating the controller theory

To simulate and visualize the logic model of the controller:

- transform the control law to the fixed reference frame:
  $\omega = -((y - y_0)\cos\psi_c - x\sin\psi_c)v\,\mathrm{sinc}\,\theta - k\theta$
- define a simple state machine, with the transformed control law as its transition (*step*) function:

```
State : TYPE = [# y, x, psi: real,        % inputs
                  y_0, psi_c: real,       % target line
                  k, v: real,             % parameters
                  omega: real #]          % output
theta(s:State): real = psi(s) - psi_c(s)
step(s:State): State = s WITH [
    omega := -((y(s) - y_0(s))*COS(psi_c(s))
                  - x(s)*SIN(psi_c(s))))
        *v(s)*SINC(theta(s)) - k(s)*theta(s) ]
```

Co-simulation



Conclusions

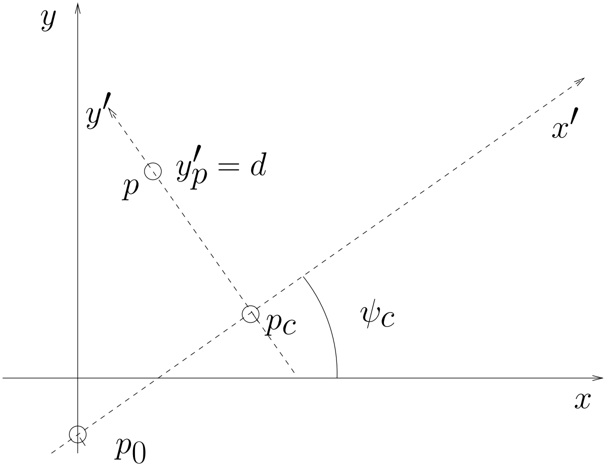
A general approach to integrated simulation and verification of control systems has been presented, using higher-order logic to model and verify both plant and controller subsystems, and co-simulation of plant and controller with different modeling languages for validation and visualization.

- Using the same controller model for verification and simulation avoids the effort both of producing two models of the same controller and of proving their equivalence;
- having different plant models for verification and simulation makes it possible to cross-check the two models;
- dfferent co-simulation frameworks can be used, e.g., the INTO-CPS tool (based on the FMI) or the PVSio-web framework.

Thank you

Grazie

Appendix: coordinate transformation



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} \cos\psi_c & -\sin\psi_c \\ \sin\psi_c & \cos\psi_c \end{bmatrix} + \begin{bmatrix} 0 \\ -y_0 \cos\psi_c \end{bmatrix}$$