

Tool Library Workshop

Marcus Lucas and Paulo Tabuada

Cyber-Physical Systems Laboratory
Department of Electrical and Computer Engineering
University of California at Los Angeles



Background

- Algorithm verification is vital to the development of cyber-physical systems
 - Safety Critical
 - Expensive to test
 - Difficult to reason about
- Verification tools are generally good for the research community
 - Aids reproducibility of results
 - Facilitates benchmarking
 - Improves quality of research/knowledge dissemination

- But verification can be a lot of extra work
 - Finding, understanding, and then using tools is non-trivial
- System designers need to better understand verification and what types of problems can be verified
- Low level of knowledge and confidence regarding existing tools
- Design process is not understood in terms of verification

Goals

- Embed verification into the engineering design process as a first-order objective
- Communicate the circumstances under which verification should be applied
- Normalize the publication and support of tools created by community members
- Grow confidence in verification's utility

How to Get There?

- Reduce barriers to entry through the CPS-VO
 - Tool Repository
 - Tutorials
 - Integrated tools
- Encourage student engagement
- Create benchmarks and standard models for evaluating tools

Verification Tool Library

Verification Tool Library

CPS-VO » CPS TOOLS AND DESIGN STUDIOS » VERIFICATION TOOL LIBRARY » SEARCH FOR TOOLS

Search for Tools

- Home
- README
- Search for Tools →
- Browse Tool Library
- Members
- Tutorials
- Tutorial - CORA (Hidden)
- Tutorial - SpaceEx
- Tutorial - S-TALIRO (Hidden)
- Bulk Edit (Hidden)
- Taxonomic Information
- Forums
- Files

[EDIT GROUP](#) [TRACK](#) [TAXONOMY](#) [BROADCAST](#) [PANELS](#) [GROUP STATS](#)

Search results for Verification Tool Library

Search for:

Types of systems handled
<Any>

Techniques used

- Abstractions
- Agent and Mode Architecture
- Ant Colony Optimization
- Approximate (bi)simulation
- Approximation of solutions to Hamilton-Jacobi partial differential equati
- Bisimulation minimization
- Boolean Equation Systems
- Bounded Model Checking
- Branching-Time Temporal Logics

Type of verification	Interface	Termination guarantees
Bisimulation generation	C++	Guaranteed to terminate
Cost optimal reachability analysis	Integrated to SpaceEx	Not guaranteed to terminate
Description/modeling language	Java	
Falsification	Matlab	
Invariant analysis	Model editor integrated	
Invariant set computation	OMNet++	
Model checking	Own interface	
Modeling	Own language	
Parameter synthesis	Python	

OS: Linux, Mac OS X, Solaris, Windows

Free: Contact the author, Free



[WIKIPAGE](#)

Submitted by [nanli](#) on Tue, 09/25/2018 - 2:53pm

COLLABORATE

Automotive Testing Simulation Validation and Verification Autonomous Driving Simulator test generation verification and validation Tool Free Java Linux Mac OS X Solaris Windows Simulation



Verification Tool Library

- VTL documents mature tools available to the CPS community
- Searchable by taxonomic terms and includes a wiki
- Links to tutorials, integrated tools, more thorough documentation

Repository Demo

Verification Tool Library

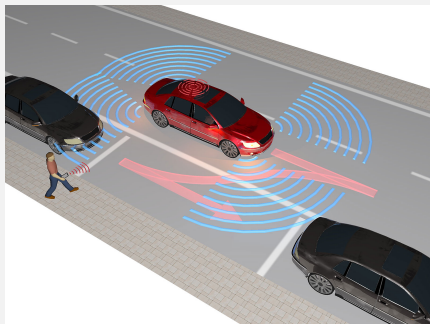
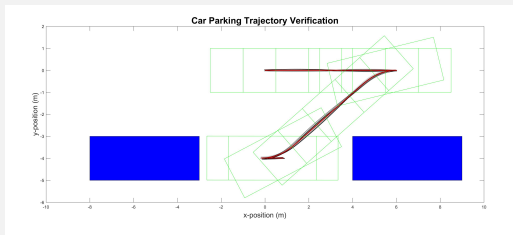
Links

- Tool Library: https://cps-vo.org/group/verification_tools/
- Submission Form: <https://cps-vo.org/group/tools/submit>

- SpaceEx
 - Only handles hybrid systems with affine continuous dynamics
 - Particularly accessible to high-school & undergraduate students
- CORA
 - Formal verification of non-linear systems
- S-TALIRO
 - Falsification of logical constraints

Tutorials

Autonomous Parking Example



- Finding and following parking trajectories is a basic requirement of autonomous cars

Tutorials

Autonomous Parking Example

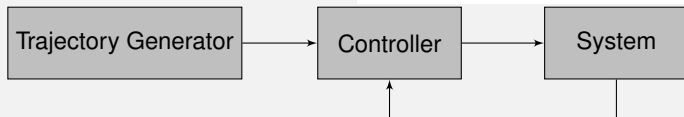
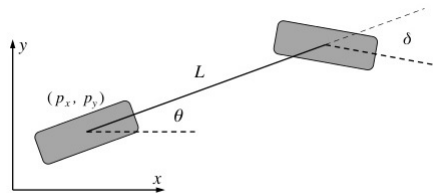
Kinematic Bicycle Model of Car

$$\dot{x} = v \cos(\theta)$$

$$\dot{y} = v \sin(\theta)$$

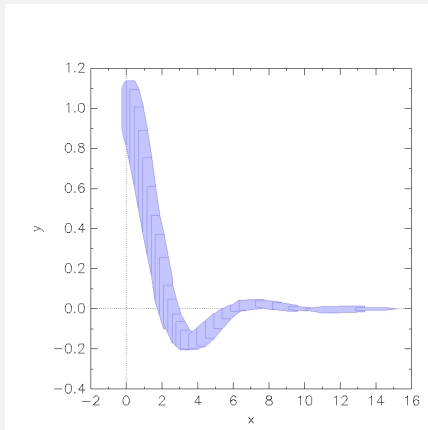
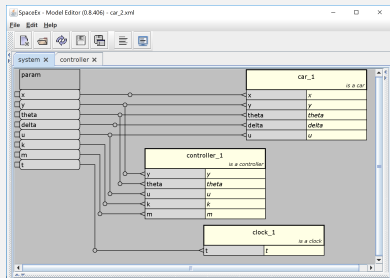
$$\dot{\theta} = \frac{v}{L} \sin(\delta)$$

$$\dot{\delta} = u$$



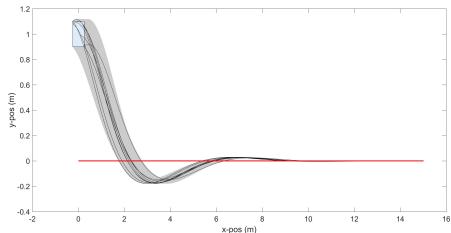
- What if you have a valid trajectory, and want to test if your controller tracks it properly?

- SpaceEx can be used to evaluate simple control problems such as stabilizing to a horizontal trajectory.



■ CORA can do the same thing

```
C:\Users\marcus\Box Sync\cyp\wfm\modelling\car_model5.m
EDITOR PUBLISH VIEW
Find Files Insert
Compare Go To Comment Breakpoints Run Run and Advance Run and Time
New Open Save Print Find Indent
FILE NAVIGATOR EDIT BREAKPOINTS RUN
1 function dx = car_model3(t, x, u)
2 %Kinematic Model with
3 % state: x=[x, y, theta, delta]
4 % input: u=[Xdot, Ydot, tdot, delta, v_p, v_d]
5
6 %parameters: get parameters from p vector
7 %body
8 m = 1370;
9 I = 2315.3;
10 ka = 1.11;
11 ab = 1.59;
12
13 vdot = 20;
14 edot = 0.1;
15 R = 30;
16
17 C_F = 1.3508 * 10^5;
18 C_r = 5.882 * 10^4;
19
20
21 %state
22 %position
23 X = x(1); %lockONAGD
24 Y = x(2); %lockONAGD
25 theta = x(3);
26 delta = x(4);
27
28 %input
29 v_p = u(5);
30 v_d = u(6);
31
32 l_R = 1;
33 l_F = 1;
34 l = 1;
35 theta = atan((l_R/l_F + l_R)*tan(x(6)));
36 sp = cos(theta);
37 ap = sin(theta);
38
```



- But it is also capable of analyzing non-linear systems, making it much more useful for verifying systems like the full parking controller.

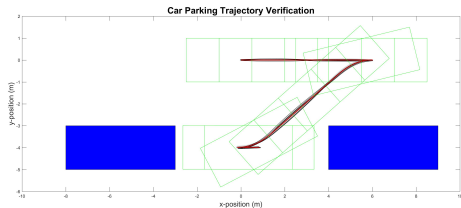
```
C:\Users\marcus\Box Sync\cops-vo\modeling\car_model5.m
EDITOR  PUBLISH  VIEW
New Open Save Print Find Files Compare Go To Comment Insert Run Section Run and Advance Run and Time
File Edit Breakpoints Run Run and Advance Run and Time
function dx = car_model3(t, x, u)
% Schematic Model with
% state x=[X,Y,theta,delta]
% input u=[Xdes,Ydes,tdes,dDes,v_p,v_d]

%parameters: get parameters from p vector
% Sbody
% m = 1370;
% I = 2315.3;
% ax = 1.1;
% xb = 1.5;
% wlen = 2;
% rlen = 0.1;
% h = 3;
% C_r = 1.3308 * 10^5;
% C_l = 9.882 * 10^4;

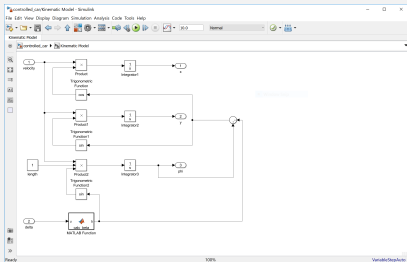
%state
% Sposition
% X = x(1); %block(0,0,0)
% Y = x(2); %block(0,0,0)
% theta = x(3);
% delta = x(4);

%input
% v_p = u(5);
% v_d = u(6);

% l_R = 1;
% l_F = 1;
% l = 1;
% theta = atan(l_R/l_F + 1_R)*tan(k(0));
% cp = cos(theta);
% sp = sin(theta);
```



- S-TALIRO does not provide formal guarantees, but can be used to falsify the same classes of problems



```
C:\Users\marcus\MATLAB\Tools\staliro\demo\Falsification\demo\staliro_demo_autotests_01.m
EDITOR FILE VIEW
New Open Save Print Compare Go To Find Breakpoints Run Run and Advance Run and Time
FILE EDITOR BREAKPOINTS RUN
22
23- disp(' ')
24- disp('The constraints on the initial conditions defined as a hypercube:');
25- init_cond = [1
26
27- disp(' ');
28- disp('The constraints on the input signal defined as a range:');
29- input_range = [0 100];
30- disp(' ');
31- disp('The number of control points for the input signal:');
32- cp_array = 7;
33
34- disp(' ');
35- disp('The specification:');
36- phi = 'ii<=1 /\ <=x2';
37
38- ii = 1;
39- precd(ii).ste='c1';
40- precd(ii).A = [-1 0];
41- precd(ii).b = -120;
42- precd(ii).loc = [1:7];
43
44- ii = ii+1;
45- precd(ii).ste='e2';
46- precd(ii).A = [0 -1];
47- precd(ii).b = -4500;
48- precd(ii).loc = [1:7];
49
50- disp(' ');
51- disp('Total simulation time:');
52- time = 30;
53
54- disp(' ');
55- disp('Create an staliro_options object with the default options:');
56- opt = staliro_options()
57
58
```


Embedded Tools

■ Currently have SpaceEx running

The screenshot displays the SpaceEx web interface in a browser window. The interface is divided into several sections:

- Model Specification:** Shows the system name 'system' and an 'Update' button. Below it, a tree view shows 'system' containing 'Controlled : L, x, y, theta, delta, u' and 'Base-components : clock_1, car_1, controller_1'. The 'Initial states' section contains the formula $t=0 \ \& \ x=0 \ \& \ 0.8 < y < 1.2 \ \& \ -0.1 < \theta < \pi \ \& \ 1 \ \& \ \delta = 0 \ \& \ u = 0$. The 'Forbidden states' section is currently empty.
- Console:** Displays the following text:

```
Computing reachable states...
Iteration 0... 1 sym states passed, 0 waiting 0.01s
Found fixpoint after 1 iterations.
Computing reachable states done after 0.011s
Output of reachable states... 0.048s
```
- Reports:** Lists the following information:

```
0.10s elapsed
1724KB memory
SpaceEx output file : output_t-x (gen).
SpaceEx output file : output_t-y (gen).
SpaceEx output file : output_x-y (gen).
Graph output file : plot_t-x (gif).
Graph output file : plot_t-y (gif).
Graph output file : plot_x-y (gif).
```
- Graphics:** Contains three plots:
 - A linear plot of x vs t showing a straight line from (0,0) to (16,16).
 - A plot of y vs t showing a noisy signal that starts at approximately 1.2 and decays towards 0 over time.
 - A plot of y vs x showing a noisy signal that starts at approximately (0, 1.2) and decays towards the origin.
- Analysis:** Features 'Start' and 'Stop' buttons. The status below indicates 'Execution terminated'.

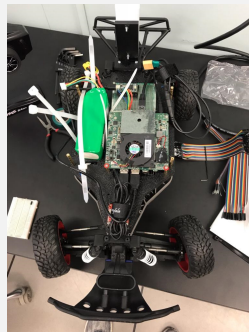
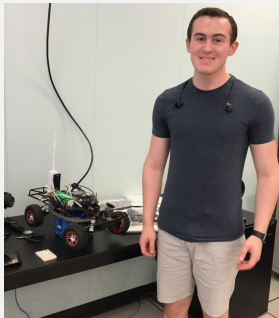
Running Tools on the CPS-VO

SpaceEx Example

SpaceEx Demo

Student Engagement

- Want to normalize the use of verification tools in design process
- Targeting undergraduate and even high-school students
- Ideal tools integrate easily with familiar software (i.e. Matlab/Simulink)



Future Work

Benchmarks

- We're announcing a new benchmarking competition!
- Expansion of the "Friendly Competition" held during the ARCH Workshop
 - <https://cps-vo.org/group/ARCH>
- Tentative structure:
 - Participants integrate verification/synthesis tools into VO
 - Test period of about a month
 - Competition organizers evaluate tools against benchmarks using VO resources
- Competition results will be advertised through the CPS-VO