Introduction

- **Program verification** attempts to ensure that a computer program is correct. And a program is correct if it behaves in accordance with its specifications.
- This does not necessarily mean that the program solves the problem that it was intended to solve; the program's specifications may be at odds with or not address all aspects of a client's requirements.
- **Program validation** attempts to ensure that the program indeed meets the client's original requirements.
- **Program testing** seeks to show that particular input values produce acceptable output values.
- **Proof of correctness** uses the techniques of a formal logic system to prove that if the input variables satisfy certain specified predicates or properties, the output variables produced by executing the program satisfy other specified properties.

Assertions

- Assertion is a predicate placed in a program that is always true at that place.

```
x:=5;
assert{x>0}
x:=x+1;
assert{x>1}
```

- Assertions help to specify programs and to reason about program correctness.
  - Precondition — an assertion placed at the beginning of a section of code — determines the set of states under which the programmer expects the code to execute.
  - Postcondition — placed at the end — describes the expected state at the end of execution.

Weakest Preconditions

- WP(P,B) is "a predicate that describes the exact set of states **s** such that when program P is started in **s**, if it terminates it will terminate in a state satisfying condition B."
- Illustration

```
{x≥3}

x := x + 2;

assert{x≥5}
```

Computing Weakest Preconditions

- Sequence of Statements

  WP(S;T, B) = WP(S,WP(T,B)).

- Assignment Statement x := e:

```
{B[e/x]}                      {(x + y) > 0 ∧ y = 0}
x := e;                         z = x + y;
assert{B}                     assert{z > 0 ∧ y = 0}
```

Computing Weakest Preconditions

- if-then-else Statement **if** c **then** S1 **else** S2:

```
{(c ∧ WP(S1, B)) ∨
(¬c ∧ WP(S2, B))}
if (c)
    S1;
else
    S2;
assert{B}
```

```
{((x < y ) ∧ (y > w )) ∨
((x ≥y)∧(x >w))}
  if (x<y)
     z: = y;
  else
     z := x;
  assert{z > w}
```

Computing Weakest Preconditions

- while statement **while** b **do** S
- In general it is not possible to compute the precise WP(W , B).
- It is possible to compute an under-approximating condition WP'(W,B)such that WP'(W,B) $\Rightarrow$ WP(W,B).
    - Unroll the loop k times, for some chosen value k $\geq$ 0, and let W ' be the thus unrolled loop.
      For e.g., for k = 0

      W' =skip
    - for k = 2,

      W' =“**if**(b){S;**if**(b)S }”.
    - Now, WP'(W,B) $\equiv$ WP(W',B $\wedge(\neg$b)).
    - Higher value of k gives a better WP'(W,B).

Formal Verification of CPS

- A closed control loop consists in
  - Sensing the plant state
  - Computing required control action
  - Actuating the control action
- Discretize the plant based on the sampling period
- Simulate the closed loop as a program
  - Compute the control action `u=KX[t]`
    - Based on the current plant state and Controller used
  - Update the plant state `X[t+1]=AX[t]+Bu=AX[t]+BKX[t]`
    - Based on the control action

Illustration

- Consider a Cruise Controller in braking mode

$$\begin{bmatrix} \dot{v}_1(t) \\ \dot{v}_2(t) \\ \dot{v}_3(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6.05 & -5.29 & -0.24 \end{bmatrix} \begin{bmatrix} v_1(t) \\ v_2(t) \\ v_3(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 2.48 \end{bmatrix} u(t)$$

- $v_1(t)$ (output state) : speed of the vehicle
- $u(t)$ (control input) : engine throttle angle
- Objective: $v_1(t) = 0$
- Initial engine speed : $v_1(0) = 50$ units
- Design Requirement: speed $v_1(t) \leq 2$ units in 2sec after the brake is pressed.

Illustration

- Discretize the cruise controller with a sampling period h=10ms

$$\begin{bmatrix} v_1(t+1) \\ v_2(t+1) \\ v_3(t+1) \end{bmatrix} = \begin{bmatrix} 1.00 & 0.01 & 0.00 \\ -0.0003 & 0.9997 & 0.01 \\ -0.06.04 & -0.0531 & -0.9974 \end{bmatrix} \begin{bmatrix} v_1(t) \\ v_2(t) \\ v_3(t) \end{bmatrix} + \begin{bmatrix} 0.0001 \\ 0.0001 \\ 0.0247 \end{bmatrix} u(t)$$

- Design Requirement: $v_1(t) \leq 2$ in 2/0.1=20 closed loop iterations
- Design a Controller K (LQR, perhaps)
- How to verify ?
  - Initial condition : $v_1(0) = 50$
  - Simulate the system for 20 closed loop iterations
  - Check if $v_1(20) \leq 2$

Illustration: Simulating the system

```
V := 50,…,…; //Initialization
while(true) //Closed control loop
{
    u := -KV;
    v := AV + Bu;
}
assert (performance requirement)
```

UNROLL

```
V0:=50,…,…;
u=-KV0;
V1=AV0+Bu;
u=-KV1;
V2=AV1+Bu;
    …
V20=AV19+Bu;
```
assert$(v_1(20) \leq 2)$

Performance requirement is over 20 loop iterations. So unroll the closed loop 20 times

This is easy to verify!!! Do you need WP at all?

Need for WP

- What if the initial value of the plant came from a range ?
- For example, instead of
  - Initial engine speed : $v_1(0) = 50$ units
- We are given that the cruise controller operates in braking mode under any speed less than 200 units
  - If you want to verify directly, you have to run the program for all possible starting value combination of the plant.
  - Not feasible anymore. This is where WP comes in

Verifying with WP

V0:=[[…],[…],[…]];

u=-KV0;

V1=AV0+Bu;

u=-KV1;

$\longrightarrow$ Program Segment W

V2=AV1+Bu;

…

V20=AV19+Bu;

assert($v_1(20) \leq 2$) $\longrightarrow$ Assertion B

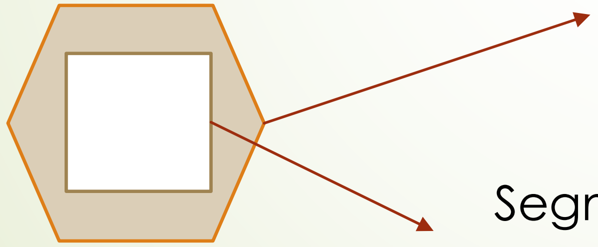
- Check if WP(W,B) satisfies V0

Verification with WP

- WP of a C program can be computed with WP plugin of Frama-C
- Frama-C has direct interface with provers like Why3, Z3 to to prove additional properties on the generated WP's
- But is that all we can do with WP ?
    - No. We can do more.
- If a prover validates a WP then there is nothing left to do.
- But, what if it fails?
    - It means that the WP holds partially with a probability.
    - How can you compute that ?

Computing probability of a WP

- Assuming uniform distribution, closed range of values of a set of variables induces a convex polytope.
- Here, the polytope captures the set of possible initial states of the plant
- The Weakest precondition in conjunction with this polytope gives the segment of initial states for which the performance requirement holds



Total set of Initial States

Segment of Initial States for which the performance requirement holds

Ratio of the volume of two regions gives the probability of the system meeting the performance requirement

Sensor Errors in CPS

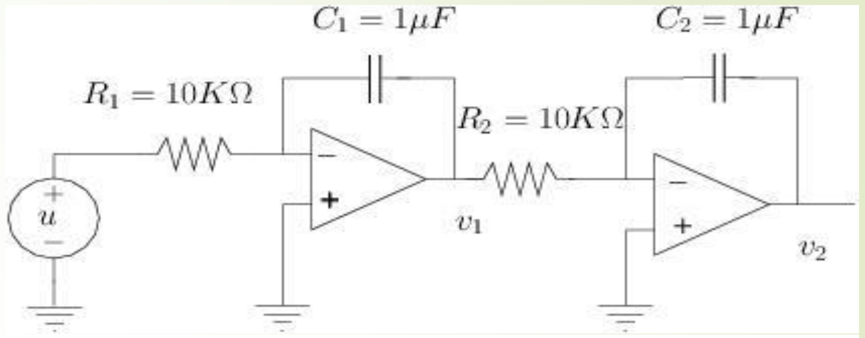
- Software controllers are prevalent in safety-critical domains
- Safety-critical systems must perform in spite of sensor faults
- Sensor faults can be caused due to
  - Transient disturbance in sensing the plant state information
  - Adversaries with the intention of destabilizing the system
- Consider an Exponential Stability requirement defined over L iterations

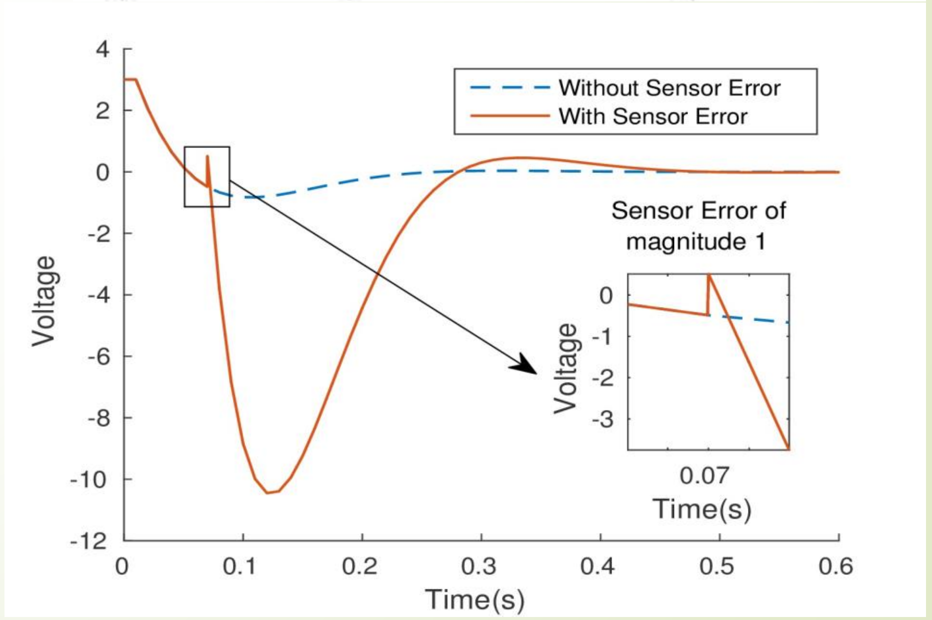
Motivating Example

- Consider a Double Integrator Circuit

$$u = -\begin{bmatrix} 200 & 20 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$$

$$\begin{bmatrix} x_1(t+1) \\ x_2(t+1) \end{bmatrix} = \begin{bmatrix} 1.00 & 0.01 \\ 0 & 1.00 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0.0001 \\ 0.01 \end{bmatrix} u$$





Closed Control Loop with Measurement Errors

```
while(true) //Closed control loop
{
    $\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} e \\ 0 \end{bmatrix}$// e is the measurement error
    $u = -\begin{bmatrix} 200 & 20 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$
    $\begin{bmatrix} x_1(t+1) \\ x_2(t+1) \end{bmatrix} = \begin{bmatrix} 1.00 & 0.01 \\ 0 & 1.00 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0.0001 \\ 0.01 \end{bmatrix} u$
}
assert (performance requirement)
```

Loops Unrolled for 2 Iterations

$$\begin{bmatrix} x_1(0) \\ x_2(0) \end{bmatrix} = \begin{bmatrix} x_1(0) \\ x_2(0) \end{bmatrix} + \begin{bmatrix} e_1 \\ 0 \end{bmatrix}$$
$$u = -\begin{bmatrix} 200 & 20 \end{bmatrix} \begin{bmatrix} x_1(0) \\ x_2(0) \end{bmatrix}$$
$$\begin{bmatrix} x_1(1) \\ x_2(1) \end{bmatrix} = \begin{bmatrix} 1.00 & 0.01 \\ 0 & 1.00 \end{bmatrix} \begin{bmatrix} x_1(0) \\ x_2(0) \end{bmatrix} + \begin{bmatrix} 0.0001 \\ 0.01 \end{bmatrix} u$$
$$\begin{bmatrix} x_1(1) \\ x_2(1) \end{bmatrix} = \begin{bmatrix} x_1(1) \\ x_2(1) \end{bmatrix} + \begin{bmatrix} e_2 \\ 0 \end{bmatrix}$$
$$u = -\begin{bmatrix} 200 & 20 \end{bmatrix} \begin{bmatrix} x_1(1) \\ x_2(1) \end{bmatrix}$$
$$\begin{bmatrix} x_1(2) \\ x_2(2) \end{bmatrix} = \begin{bmatrix} 1.00 & 0.01 \\ 0 & 1.00 \end{bmatrix} \begin{bmatrix} x_1(1) \\ x_2(1) \end{bmatrix} + \begin{bmatrix} 0.0001 \\ 0.01 \end{bmatrix} u$$
assert($x_1(2) < 5$)

Simplification

$$\begin{bmatrix} x_1'(0) \\ x_2'(0) \end{bmatrix} = \begin{bmatrix} x_1(0) \\ x_2(0) \end{bmatrix} + \begin{bmatrix} e_1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} x_1(1) \\ x_2(1) \end{bmatrix} = \begin{bmatrix} 1.00 & 0.01 \\ 0 & 1.00 \end{bmatrix} \begin{bmatrix} x_1(0) \\ x_2(0) \end{bmatrix} - \begin{bmatrix} 0.0001 \\ 0.01 \end{bmatrix} \begin{bmatrix} 200 & 20 \end{bmatrix} \begin{bmatrix} x_1'(0) \\ x_2'(0) \end{bmatrix}$$

$$\begin{bmatrix} x_1'(1) \\ x_2'(1) \end{bmatrix} = \begin{bmatrix} x_1(1) \\ x_2(1) \end{bmatrix} + \begin{bmatrix} e_2 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} x_1(2) \\ x_2(2) \end{bmatrix} = \begin{bmatrix} 1.00 & 0.01 \\ 0 & 1.00 \end{bmatrix} \begin{bmatrix} x_1(1) \\ x_2(1) \end{bmatrix} - \begin{bmatrix} 0.0001 \\ 0.01 \end{bmatrix} \begin{bmatrix} 200 & 20 \end{bmatrix} \begin{bmatrix} x_1'(1) \\ x_2'(1) \end{bmatrix}$$

assert$(x_1(2) < 5)$

Simplification

$x_1'(0) = x_1(0) + e_1$
$x_1(1) = x_1(0) + 0.01x_2(0) - 0.02x_1'(0) - 0.002x_2(0)$
$x_2(1) = x_2(0) - 2x_1(0) - 0.2x_2(0)$
$x_1'(1) = x_1(1) + e_2$
$x_1(2) = x_1(1) + 0.01x_2(1) - 0.02x_1'(1) - 0.002x_2(1)$
$x_2(2) = x_2(1) - 2x_1(1) - 0.2x_2(1)$
assert($x_1(2) < 5$)

Simplification

$x_1'(0) = x_1(0) + e_1$
$x_1(1) = x_1(0) - 0.02x_1'(0) + 0.008x_2(0)$
$x_2(1) = -2x_1(0) + 0.8x_2(0)$
$x_1'(1) = x_1(1) + e_2$
$x_1(2) = x_1(1) - 0.02x_1'(1) + 0.008x_2(1)$
assert($x_1(2) < 5$)

Simplification

$x_1(1) = 0.98x_1(0) + 0.008x_2(0) - 0.02e_1$
$x_2(1) = -2x_1(0) + 0.8x_2(0)$
$x_1(2) = 0.98x_1(1) + 0.008x_2(1) - 0.02e_2$
assert$(x_1(2) < 5)$

WP Calculation

Step – 1

$x_1(1) = 0.98x_1(0) + 0.008x_2(0) - 0.02e_1$
$x_2(1) = -2x_1(0) + 0.8x_2(0)$
assert($0.98x_1(1) + 0.008x_2(1) - 0.02e_2 < 5$)

WP Calculation

Step – 2

$x_1(1) = 0.98x_1(0) + 0.008x_2(0) - 0.02e_1$
assert$(0.98x_1(1) - 0.02e_2 + 0.008(-2x_1(0) + 0.8x_2(0)) < 5)$

WP Calculation

Step – 3

assert($0.98(0.98x_1(0) + 0.008x_2(0) - 0.02e_1) - 0.02e_2 + 0.008(-2x_1(0) + 0.8x_2(0)) <$
5)
=assert($\left(0.9604x_1(0) - 0.0196e_1 + 0.000784x_2(0)\right) - 0.02e_2 + (-0.016x_1(0) +$
$0.0064x_2(0)) < 5$)
=assert($0.9444x_1(0) + 0.007184x_2(0) - 0.0196e_1 - 0.02e_2$ <5)

Methodology

Input

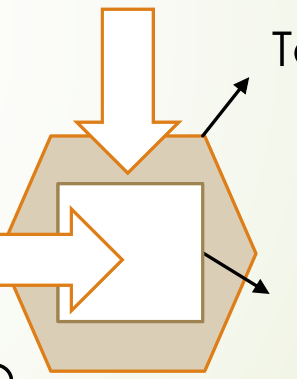
- Plant (A_p, B_p)
- Controller (K)
- (L, ε) Stability

Automatically Extract All Possible Error Scenarios

- Range of Initial Values Of the Plant $\mathcal{V}$
- Range Of Sensor Errors

```
$X[0] \in \mathcal{V}$
$u = -KX[0]$
$X[1] = A_p X + B_p u$
    …
sensor error
 … until $L$ iterations
assert(stability)
```

Weakest Precondition Q



Total Region of Operation of the Control System P

Region where the system is stable in spite of sensor errors

Probability of Error Scenario = Volume (P ∧ Q/ Volume (P))