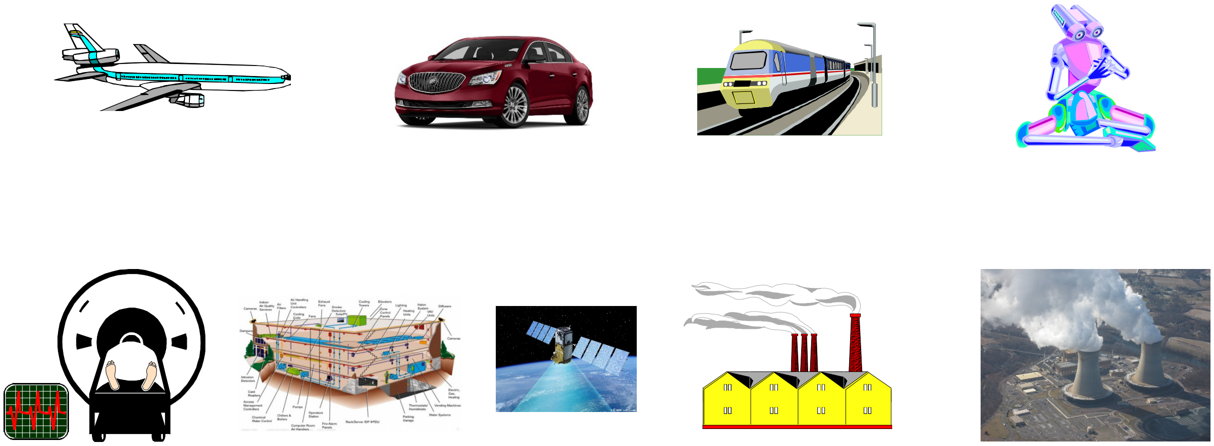
Ed Clarke’s Impact on Automotive Systems

Prof. Raj Rajkumar

Carnegie Mellon University

Cyber-Physical Systems

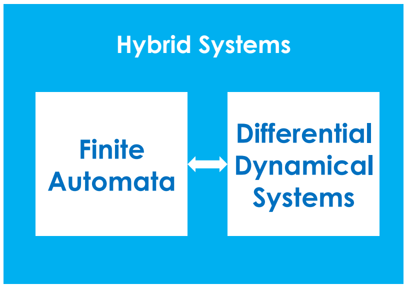


Cyber-Physical Systems as *Stochastic Systems*

- Due to uncertainties in the environment, faults, etc.
- *Transient property* specification:
  - “What is the probability that the system shuts down within 0.1 ms”?

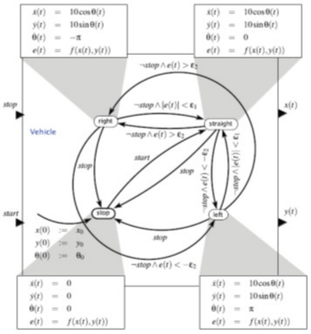
Hybrid Systems

- Hybrid systems combine continuous and discrete components.
- They suitably model automotive control systems.

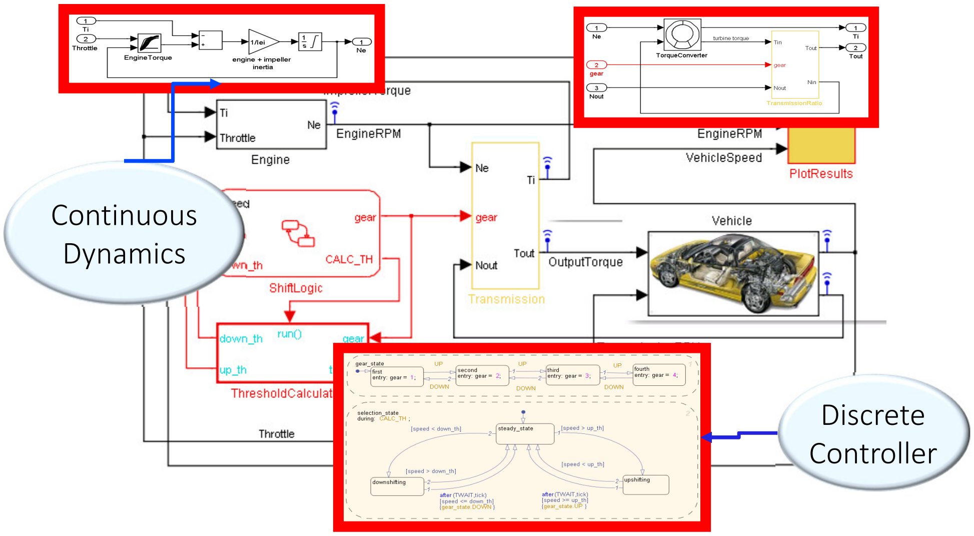


Hybrid Systems

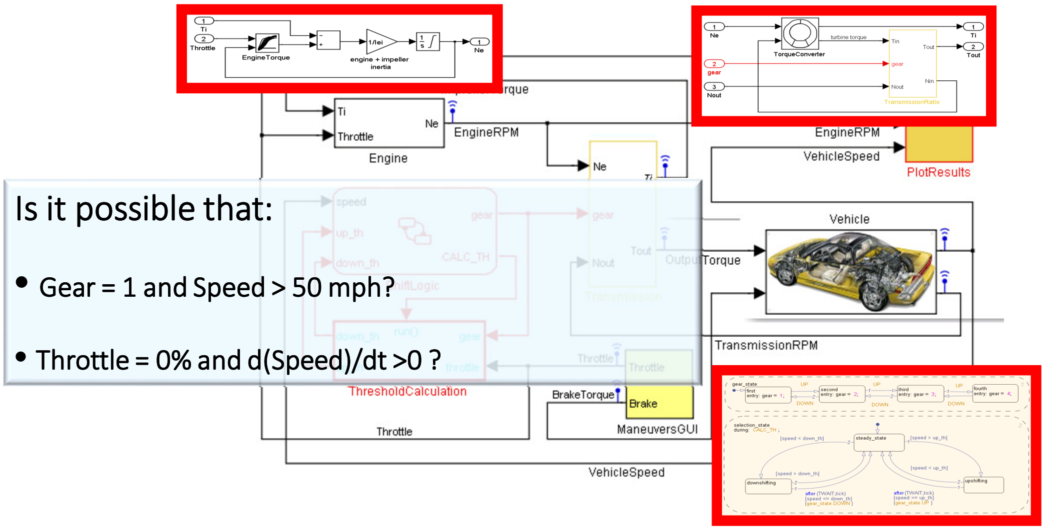
Finite Automata ⟷ **Differential Dynamical Systems**



Safety Verification of a Hybrid System



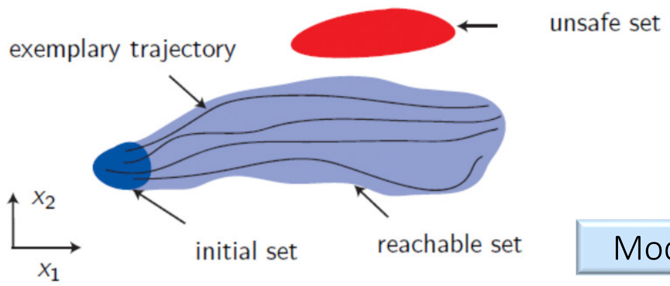
Safety Verification of Hybrid System



Is it possible that:

- Gear = 1 and Speed > 50 mph?
- Throttle = 0% and d(Speed)/dt >0 ?

Safety Verification



Model Checking

Goal:

- Show that UNSAFE states are NOT reachable.
- When UNSAFE is reachable, ALWAYS report the problems and provide *DIAGNOSIS*.
- Fully automated.

Bounded Model Checking

Developed a bounded model checker for non-linear hybrid systems

- Debug systems up to a bounded depth.

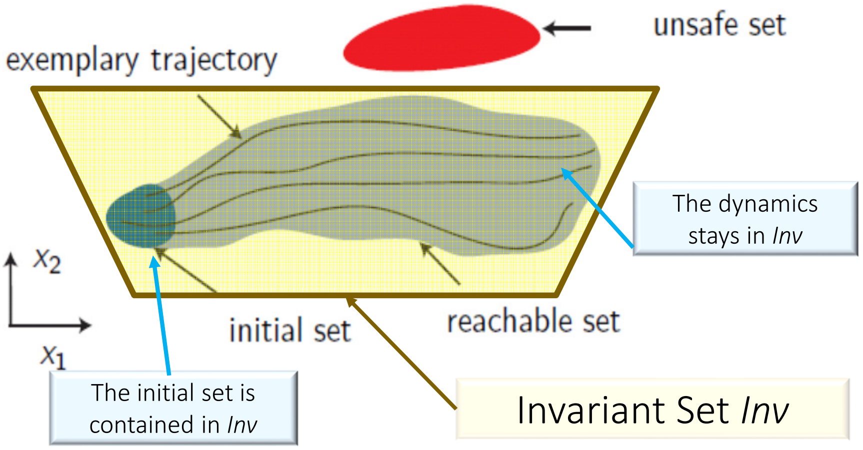
However, typically users ask to verify models that are supposed to be correct!

Can Hybrid Systems Be Verified?

- Bounded model checking and reachable-set computation do not suffice.
  - Bounds on time
  - Bounds on depth
- In general, an undecidable problem.

Clarke & Co: But there is a way!

Invariant Set → Safety



The dynamics stays in *Inv*

The initial set is contained in *Inv*

Invariant Set *Inv*

Inductive Invariants

- Suppose a region *Inv* of the state space satisfies:
  - The system starts within *Inv*;
  - Dynamics never takes the system outside of *Inv*;

Result

- Decision solvers can be used for *invariant-based verification* of hybrid systems.
- The method is complementary to bounded model checking

  debugging vs. verifying
- *Possible to verify realistic designs now!*

dReach Tool

Used for the safety verification of:

- Model-Level Properties
- Code-Level Properties

---

dReach's Verification Techniques

- *Visualization*: Reachable Set Computation
- *Debugging*: Bounded Model Checking
- *Certifying*: Invariant-based Verification

The first model-checking tool that can handle non-linear hybrid systems.

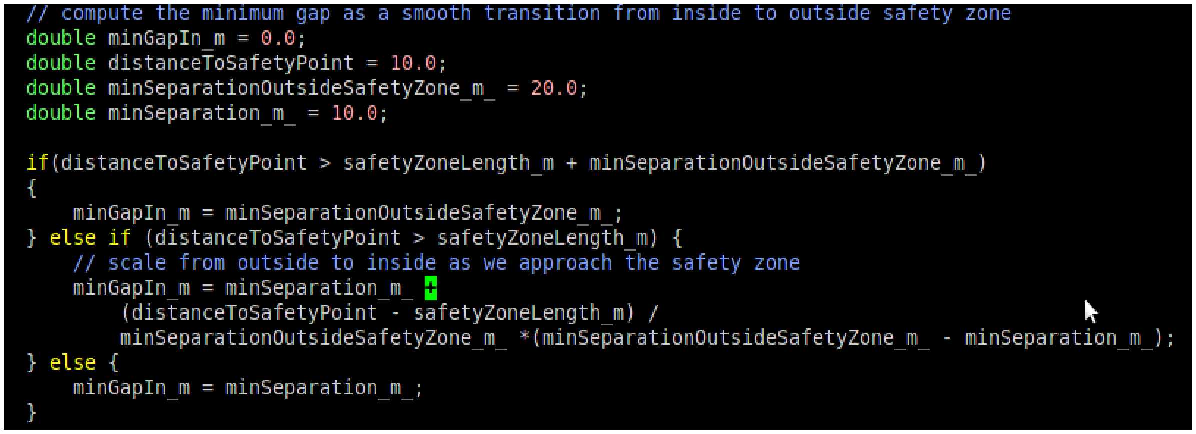
Code in CMU's Autonomous Cadillac SRX

- About 500K lines of C++ code in total. A very complex system: perception, planning, behaviors, ...
- Hybrid system (combining continuous and discrete controls) in nature.
- Control part: The implementation of control should be right.
- Logical part: The logical framework should not have bugs.
- Run-time errors: Division by zero, overflow, ...

Formal Studies of Programs

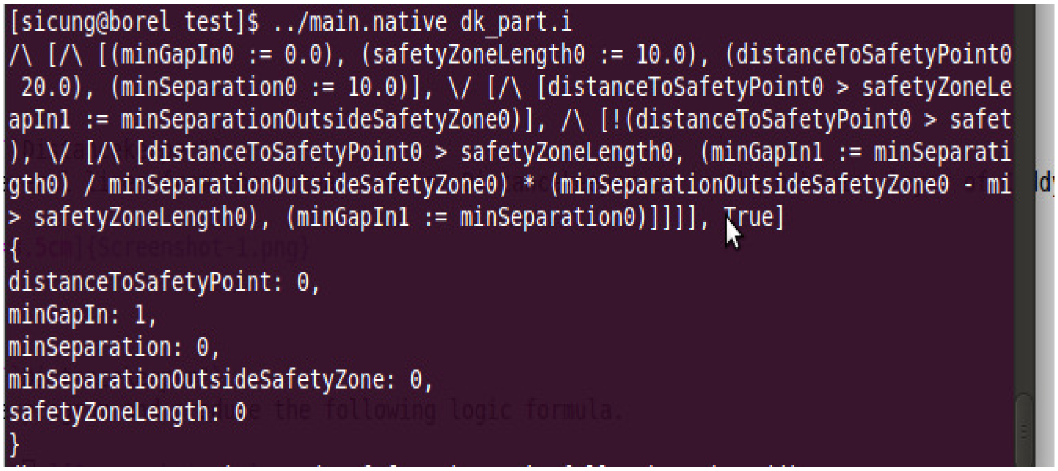
- Programs are state transformers:
  - All the possible values for variables in a piece of code form a state space
  - They define a transitional system?
- Safety Properties
  - Does there exist an E_0 such that after some n, $E_n \in$ {Unsafe states}?

Example: “Distance Keeper”



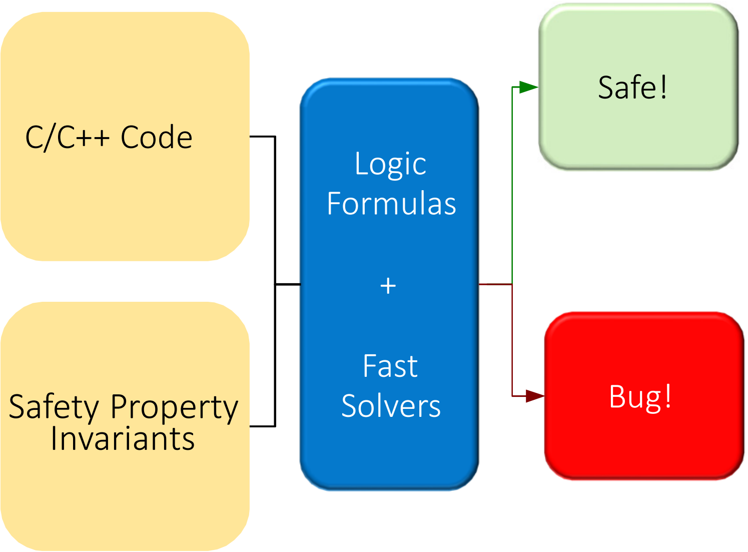
```
// compute the minimum gap as a smooth transition from inside to outside safety zone
double minGapIn_m = 0.0;
double distanceToSafetyPoint = 10.0;
double minSeparationOutsideSafetyZone_m_ = 20.0;
double minSeparation_m_ = 10.0;

if(distanceToSafetyPoint > safetyZoneLength_m + minSeparationOutsideSafetyZone_m_)
{
    minGapIn_m = minSeparationOutsideSafetyZone_m_;
} else if (distanceToSafetyPoint > safetyZoneLength_m) {
    // scale from outside to inside as we approach the safety zone
    minGapIn_m = minSeparation_m_ +
        (distanceToSafetyPoint - safetyZoneLength_m) /
        minSeparationOutsideSafetyZone_m_ *(minSeparationOutsideSafetyZone_m_ - minSeparation_m_);
} else {
    minGapIn_m = minSeparation_m_;
}
```

“Distance Keeper”: Code to Logic Formula

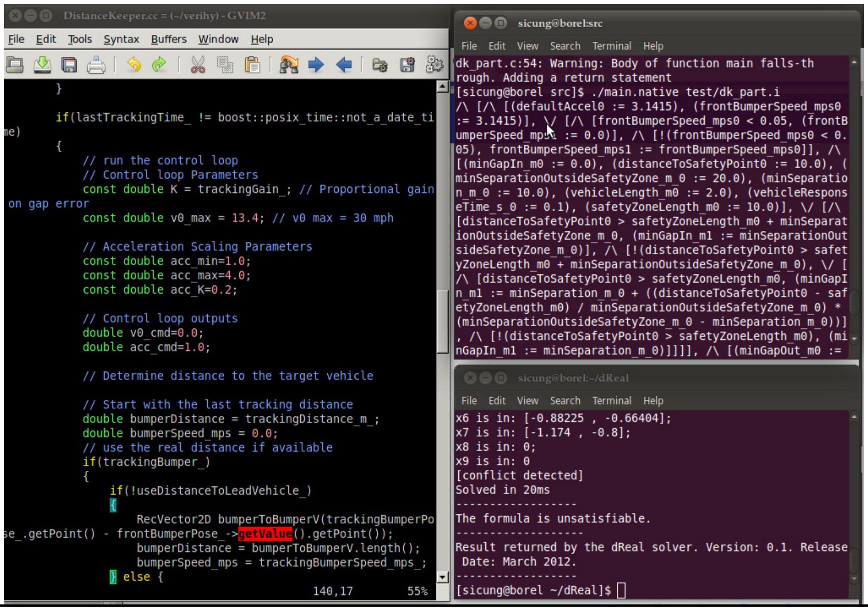


```
[sicung@borel test]$ ../main.native dk_part.i
/\ [/\ [(minGapIn0 := 0.0), (safetyZoneLength0 := 10.0), (distanceToSafetyPoint0
 20.0), (minSeparation0 := 10.0)], \/ [/\ [distanceToSafetyPoint0 > safetyZoneLe
apIn1 := minSeparationOutsideSafetyZone0)], /\ [!(distanceToSafetyPoint0 > safet
), \/ [/\ [distanceToSafetyPoint0 > safetyZoneLength0, (minGapIn1 := minSeparati
gth0) / minSeparationOutsideSafetyZone0) * (minSeparationOutsideSafetyZone0 - mi
> safetyZoneLength0), (minGapIn1 := minSeparation0)]]]], True]
{
distanceToSafetyPoint: 0,
minGapIn: 1,
minSeparation: 0,
minSeparationOutsideSafetyZone: 0,
safetyZoneLength: 0
}
```

The *dReach* Approach



The Implementation



```
        }

        if(lastTrackingTime_ != boost::posix_time::not_a_date_ti
me)
        {
            // run the control loop
            // Control loop Parameters
            const double K = trackingGain_; // Proportional gain
 on gap error
            const double v0_max = 13.4; // v0 max = 30 mph

            // Acceleration Scaling Parameters
            const double acc_min=1.0;
            const double acc_max=4.0;
            const double acc_K=0.2;

            // Control loop outputs
            double v0_cmd=0.0;
            double acc_cmd=1.0;

            // Determine distance to the target vehicle

            // Start with the last tracking distance
            double bumperDistance = trackingDistance_m_;
            double bumperSpeed_mps = 0.0;
            // use the real distance if available
            if(trackingBumper_)
            {
                if(!useDistanceToLeadVehicle_)
                {
                    RecVector2D bumperToBumperV(trackingBumperPo
se_.getPoint() - frontBumperPose_->getValue().getPoint());
                    bumperDistance = bumperToBumperV.length();
                    bumperSpeed_mps = trackingBumperSpeed_mps_;
                } else {
                                                  140,17        55%
```

```
dk_part.c:54: Warning: Body of function main falls-th
rough. Adding a return statement
[sicung@borel src]$ ./main.native test/dk_part.i
/\ [/\ [(defaultAccel0 := 3.1415), (frontBumperSpeed_mps0
:= 3.1415)], \/ [/\ [frontBumperSpeed_mps0 < 0.05, (frontB
umperSpeed_mps1 := 0.0)], /\ [!(frontBumperSpeed_mps0 < 0.
05), frontBumperSpeed_mps1 := frontBumperSpeed_mps0]], /\
[(minGapIn_m0 := 0.0), (distanceToSafetyPoint0 := 10.0), (
minSeparationOutsideSafetyZone_m_0 := 20.0), (minSeparatio
n_m_0 := 10.0), (vehicleLength_m0 := 2.0), (vehicleRespons
eTime_s_0 := 0.1), (safetyZoneLength_m0 := 10.0)], \/ [/\
[distanceToSafetyPoint0 > safetyZoneLength_m0 + minSeparat
ionOutsideSafetyZone_m_0, (minGapIn_m1 := minSeparationOut
sideSafetyZone_m_0)], /\ [!(distanceToSafetyPoint0 > safet
yZoneLength_m0 + minSeparationOutsideSafetyZone_m_0), \/ [
/\ [distanceToSafetyPoint0 > safetyZoneLength_m0, (minGapI
n_m1 := minSeparation_m_0 + ((distanceToSafetyPoint0 - saf
etyZoneLength_m0) / minSeparationOutsideSafetyZone_m_0) *
(minSeparationOutsideSafetyZone_m_0 - minSeparation_m_0))]
, /\ [!(distanceToSafetyPoint0 > safetyZoneLength_m0), (mi
nGapIn_m1 := minSeparation_m_0)]]]], /\ [(minGapOut_m0 :=
```

```
x6 is in: [-0.88225 , -0.66404];
x7 is in: [-1.174 , -0.8];
x8 is in: 0;
x9 is in: 0
[conflict detected]
Solved in 20ms
------------------
The formula is unsatisfiable.
------------------
Result returned by the dReal solver. Version: 0.1. Release
 Date: March 2012.
------------------
[sicung@borel ~/dReal]$ 
```

Conclusions

- To produce reliable automobile with any safety-critical automated features, it is impossible to do without complete formal verification on the code.
- Clarke & Co. have developed the technology that suits the verification needs of this domain.
- It is based on established theories of program verification and their new powerful solvers for non-linear problems.
- Tools ready for use by developers of makers of automated vehicles.

Thank you, Ed!