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.
Safety Verification of a Hybrid System

Is it possible that:

• Gear = 1 and Speed > 50 mph
• Throttle = 0% and d(Speed)/dt > 0
Safety Verification

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 $\xrightarrow{\text{Safety}}$

- The initial set is contained in $\mathcal{Inv}$
- The dynamics stays in $\mathcal{Inv}$
- Exemplary trajectory
- Unsafe set
- Initial set
- Reachable set

Invariant Set $\mathcal{Inv}$
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 \( n \), \( E_n \in \{\text{Unsafe states}\} \)?
Example: “Distance Keeper”

```c
// 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 1
\( \land [(\text{minGapIn}0 = 0.0), (\text{safetyZoneLength}0 = 10.0), (\text{distanceToSafetyPoint}0 > 20.0), (\text{minSeparation}0 = 10.0)], \lor [(\text{distanceToSafetyPoint}0 > \text{safetyZoneLength}0 \land \text{minGapIn}1 = \text{minSeparationOutsideSafetyZone}0)], \lor [(\text{distanceToSafetyPoint}0 > \text{safetyZoneLength}0 \land \text{minGapIn}1 = \text{minSeparationOutsideSafetyZone}0 \land \text{minSeparation}0 > \text{safetyZoneLength}0), (\text{minGapIn}1 = \text{minSeparation}0)])], \text{true}
```

```
distanceToSafetyPoint: 0,
minGapIn: 1,
minSeparation: 0,
minSeparationOutsideSafetyZone: 0,
safetyZoneLength: 0
```
The *dReach* Approach

- C/C++ Code
- Safety Property Invariants
- Logic Formulas + Fast Solvers

Safe!

Bug!

The Implementation

The overall loop

The Implementation
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!