CS 498: Shield Synthesis of Cyber Physical Systems

Shield Synthesis for Real: Enforcing Safety in Cyber-Physical Systems

Presented by:
Samay Varshney (180101097)
Siddhartha Jain (180101078)

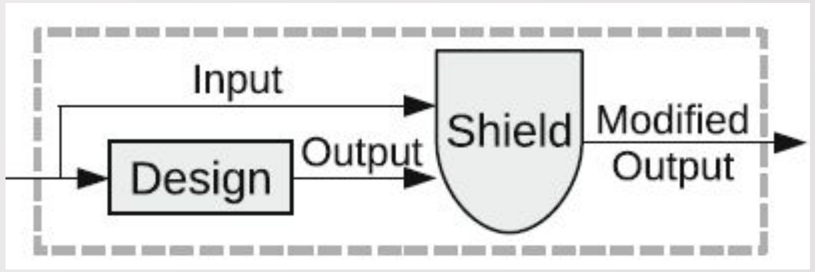
Introduction: Boolean Shield

Bloem et al. introduced the concept of shield, denoted S, to enforce a specification ϕ of a system D with certainty. The goal is to ensure that the combined D◦S never violates ϕ.

For example, D malfunctions and produces an erroneous output O for input I, S will correct O into O’ instantaneously to ensure ϕ(I,O’) holds even when ϕ(I,O) fails.

Here, instantaneously means correction is made in the same clock cycle.

But this method worked only for boolean inputs/outputs.



Introduction: Boolean Shield (contd.)

Boolean shields do not work for systems where signals have real values and need to satisfy constraints such as $x + y \leq 1.53$.

Naively treating the real-valued constraint as a predicate, or a Boolean variable P, may lead to loss of information at the synthesis time and unrealizability at run time.

For example, while the Boolean combination $P \wedge \neg Q \wedge \neg R$ may be allowed, the corresponding real-valued constraint may not have solution,

e.g. with $P : x + y \leq 1.53$, $Q : x < 1.0$ and $R : y < 1.0$.

Preliminaries

Assume that the system, D, is a blackbox with input I and output O.
After the boolean shield computed the corrected output O', such that D satisfies ϕ, we treat the correction computation as a two-player safety game.

Safety Game: The antagonist controls the alphabet Σ_{IO} and the protagonist controls the alphabet Σ_O.

In each state $g \in G$, the antagonist chooses a letter $\sigma_{IO} \in \Sigma_{IO}$ and then the protagonist chooses a letter $\sigma_O \in \Sigma_O$, thus leading to state $g = \delta_G(g, \sigma_{IO}, \sigma_O)$. The resulting state sequence $g_0 g_1 \dots$ is called a play.

A winning region W is a subset of $(G \setminus F)$ states.
A play is winning for the protagonist when it visits only the safe states.
A winning strategy is a function $\omega : G \times \Sigma_{IO} \rightarrow \Sigma_O$ that ensures the protagonist always wins.
The shield S is an implementation of the winning strategy.

Example

Consider the following two formulas in LTL:

$G(A \Rightarrow B_1)$

$G(A \land X(\neg A) \Rightarrow B_2\, U\, A)$ $G(\neg A \Rightarrow X(B_2\, W\, A))$

where G means Globally, X means Next, U means Until, Boolean variable A is an input signal, while B_1 and B_2 are output signals of D. B_1' , B_2' are outputs of boolean shield.

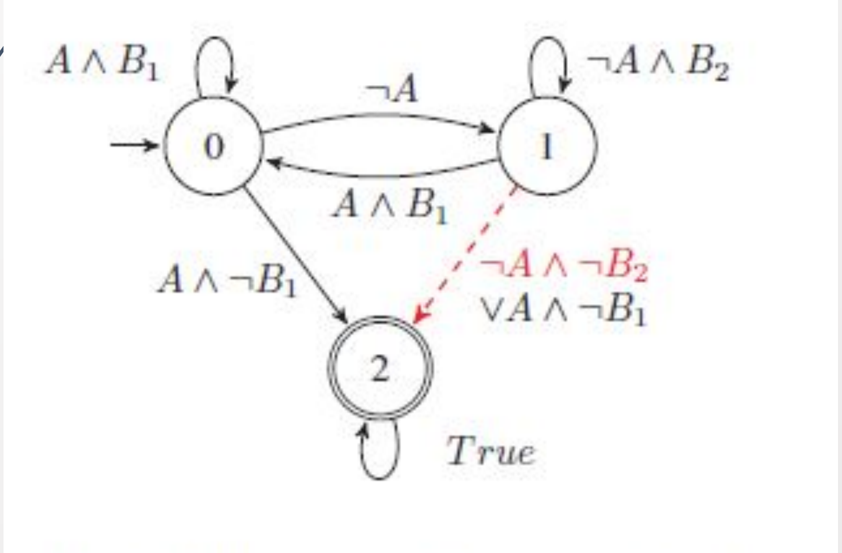
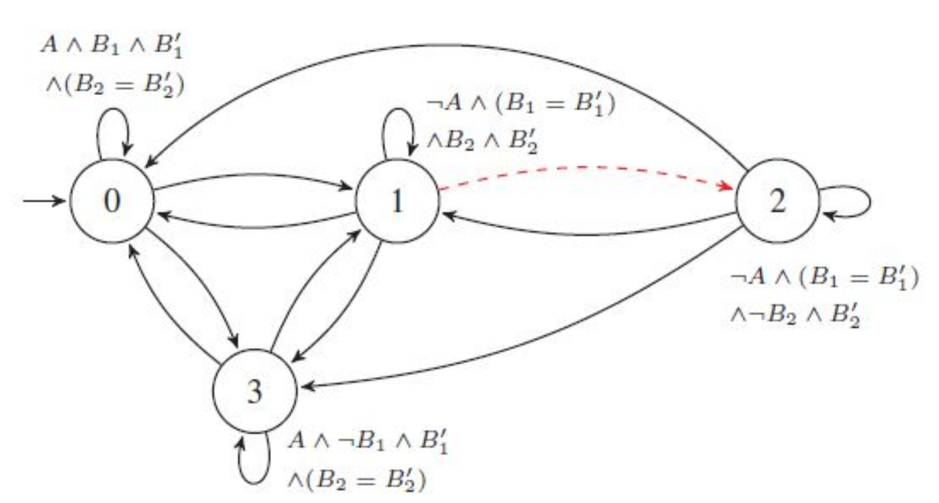


Fig. 2. Boolean safety specification.



$A \land B_1 \land B_1'$
$\land(B_2 = B_2')$
$\neg A \land (B_1 = B_1')$
$\land B_2 \land B_2'$
0
1
2
3
$\neg A \land (B_1 = B_1')$
$\land \neg B_2 \land B_2'$
$A \land \neg B_1 \land B_1'$
$\land(B_2 = B_2')$

Fig. 3. Boolean shield for Properties in Fig. 2.

Realizability of the Boolean Predicates

Using a Boolean shield to generate real-valued correction signals has two problems: realizability of the Boolean predicates and quality of the real-valued signals.

The Boolean specification discussed previously are abstractions of the real-valued LTL properties below:

$G(l=\text{power} \Rightarrow |\mu| < 0.2)$
$G(l=\text{power} \wedge X(l=\text{normal}) \Rightarrow (|\mu| < 0.02)\ U\ (l=\text{power}))$

l denotes the system mode, μ is the normalized error.

In the Boolean versions, A denotes whether the system is in the power mode, while B_1 and B_2 denote $|\mu| < 0.2$ and $|\mu| < 0.02$, respectively.

The combination $\neg B_1 \wedge B_2$ is unrealizable, because $|\mu|$ cannot be both greater than 0.2 and less than 0.02.

Realizability of the Boolean Predicates

However, the shield synthesized by existing methods is not aware of this problem, and thus may produce combinations of Boolean values that are not realizable in the real domain.

As shown by the red edge in figure, if the shield's input is $\neg A \land \neg B_1 \land \neg B_2$, the shield's output will be $\neg B_1 \land B_2$, despite that $|\mu| \geq 0.2 \land |\mu| < 0.02$ is unsatisfiable.

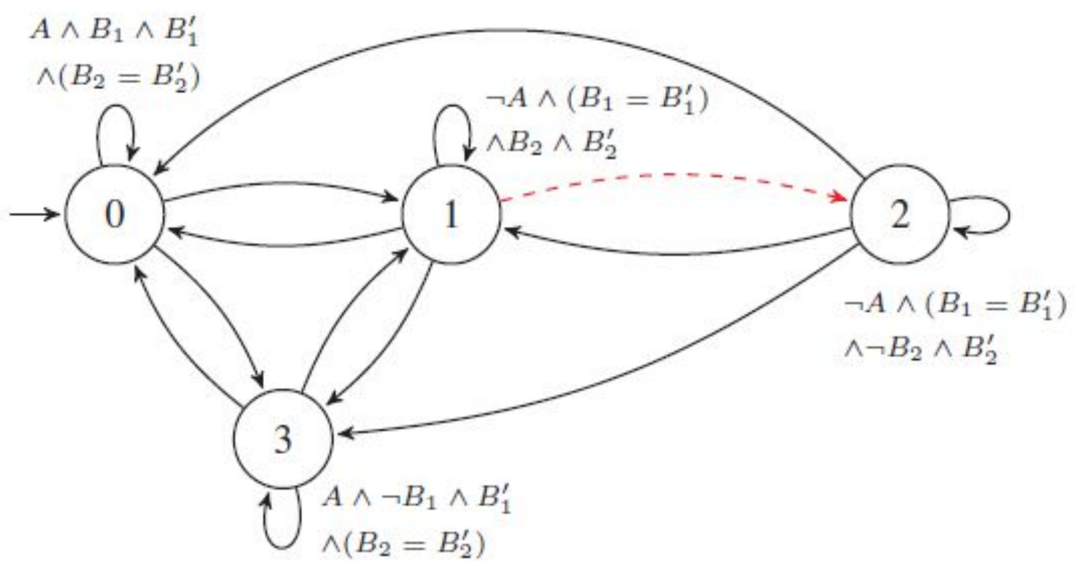

$A \land B_1 \land B_1'$
$\land (B_2 = B_2')$
$\neg A \land (B_1 = B_1')$
$\land B_2 \land B_2'$
0
1
2
3
$\neg A \land (B_1 = B_1')$
$\land \neg B_2 \land B_2'$
$A \land \neg B_1 \land B_1'$
$\land (B_2 = B_2')$

Fig. 3. Boolean shield for Properties in Fig. 2.

Quality of the Real-valued Output

Consider G(A ⇒ B), which abstracts G(I = power ⇒ $|\mu| < 0.2$).

Suppose the original system's output violates the property $|\mu| < 0.2$ as shown by the blue line, where the two erroneous values are in the middle.

The correction computed by an LP solver may be any of the infinitely many values in the interval (-0.2, +0.2). A real system expects the signal to be stable, not arbitrary.

Ideally, we want to generate real-valued signals that are smooth, and consistent with physical laws of the environment, e.g., the green line.

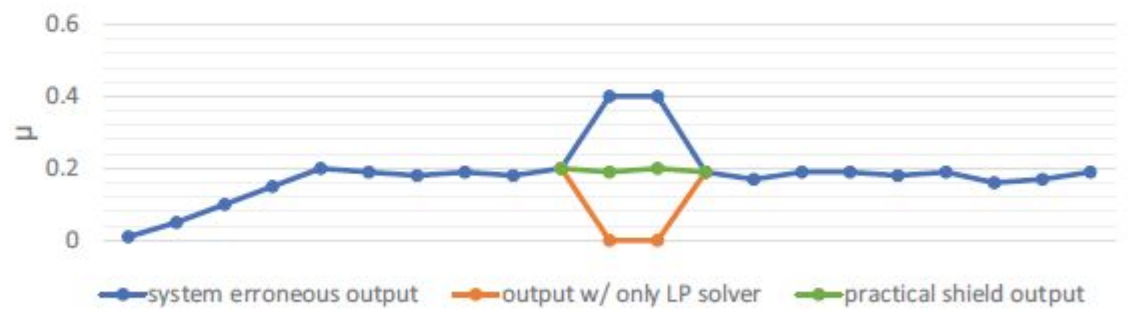


Fig. 4. Importance of the smoothness in real-valued correction signals.

Computing the Predicates

Consider the STL formulas below, which come from the powertrain control system without modification.

$G_{[Ts,\ T]}\,(l = \text{power} \Rightarrow |\mu| < 0.2)$
$G_{[Ts,\ T]}\,(l = \text{power} \wedge X(l = \text{normal})) \Rightarrow G_{[\eta,\ \varsigma / 2]}\,(|\mu| < 0.02)$

To compute P, first, we convert each time interval to a conjunction of linear constraints.

T1: $(t \geq \tau s)$, T2: $(t \leq T)$, T3: $(t \geq \eta)$, T4: $(t \leq \varsigma/2)$

Next, we convert the constraints over real-valued variables to predicates.

L1: $(l = \text{power})$, L2: $(l = \text{normal})$, M1: $(|\mu| < 0.2)$, M2: $(|\mu| < 0.02)$

Computing the Boolean Abstractions

After the set P of predicates is computed, we use it to compute the Boolean abstractions of ϕ_r, I_r, O_r and O_r'.

To compute ϕ from ϕ_r, we replace a real-valued predicate $p \in P$ with a new Boolean variable v_p.

To compute I from I_r, we traverse the predicates in P and, for each predicate $Q \in P$ defined over some real-valued signals in I_r, we add a new Boolean variable v_Q to I. Similarly, O and O' are also computed from O_r and O_r' by creating new Boolean variables.

For example:
to compute I from I_r where I_r is given as "power", there are 2 predicates defined over I_r so 2 boolean variables are added to I. $I(L_1 L_2) = 10$.

To compute O from O_r ($\mu = 0.1$), there are 2 predicates defined over O_r, so $O(M_1 M_2) = 11$.

Computing the Relaxation Automaton R(I,O)

The relaxation automaton R aims to identify impossible combinations of I and O values, and since they will never occur in the shield's input, there is no need to make corrections in the shield's output.

There may be two reasons why a value combination is impossible:

- The values of real-valued predicates are incompatible, e.g., as in $|\mu| < 0.02$ and $|\mu| > 0.2$.
- The values are not consistent with physical laws of the environment, e.g., time never travels backward. For example, with respect to the time interval $[\tau s, T]$, the transition from $T1 \land T2$ to $\neg T1 \land T2$ is impossible.

States in the relaxation automaton R are divided into two types: normal states and impossible states. Here, normal means the I/O behavior of the system D may occur, whereas impossible means it will never occur.

Computing the Relaxation Automaton R(I,O)

The dashed edges come from the physical laws (time never travels backward), while the solid edges comes from the compatibility of real-valued predicates defined over I and μ.

In particular, the combination $\neg M_1 \wedge M_2$ is identified as impossible, because |μ| cannot be greater than 0.2 and less than 0.02 at the same time.

The compatibility checking is performed due to the use of variable partitioning and unsatisfiable (UNSAT) cores.

- P may be divided into subgroups, such that predicates from different subgroups do not interfere with each other.
- When a value combination is proved to be unsatisfiable, we compute its UNSAT core, i.e., a minimal subset that itself is UNSAT.

By leveraging these UNSAT cores, we can significantly speed up the checking of value combinations.

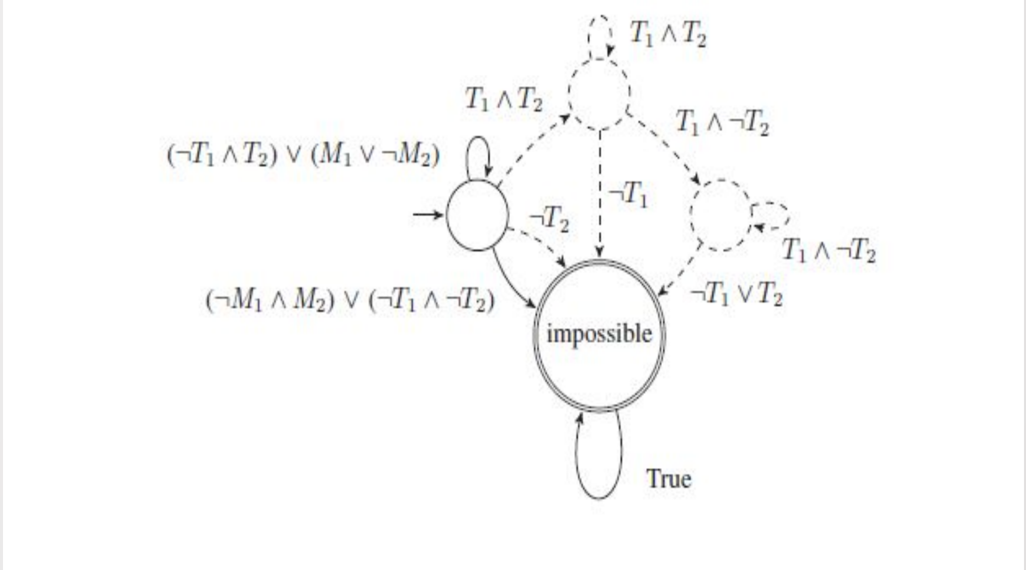


Fig. 5. Relaxation automaton $\mathcal{R}(I,O)$: *impossible* means the system $\mathcal{D}$ will not allow the state to be reached, and the shield $\mathcal{S}$ can treat it as *don't care*.

Computing the Feasibility Automaton F(O’)

The feasibility automaton F aims to capture the combinations of O’ values that are unrealizable in the real domain.
States in F are divided into two types: normal and infeasible. Here, normal means the value combinations are realizable in the real domain, whereas infeasible means the value combinations may be unrealizable.

Upon $\neg M_1' \wedge M_2'$, the automaton goes into the infeasible state, because $\neg(|\mu| < 0.2) \wedge (|\mu| < 0.02)$ has no real-valued solution.

During our computation of the winning strategy ω_r, we need to avoid such unrealizable combinations.

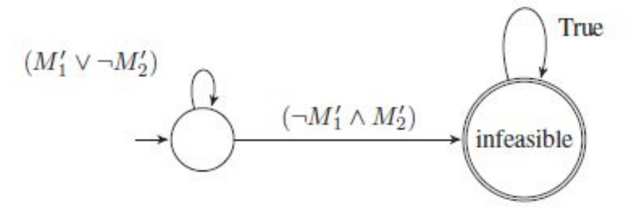


Fig. 6. Feasibility automaton $\mathcal{F}(O')$: *infeasible* means the state is unrealizable, and the shield $\mathcal{S}$ must avoid the related edges while generating solutions.

Solving The Constraint Game

The new safety game G_r is the composition of W, the winning region of the Boolean game G, the relaxation automaton R, and the feasibility automaton F.

Safe states of G_r are either
- states that are both safe in W and feasible in F
- states that are impossible in R.

Finally, we solve G_r using standard algorithms for safety games.

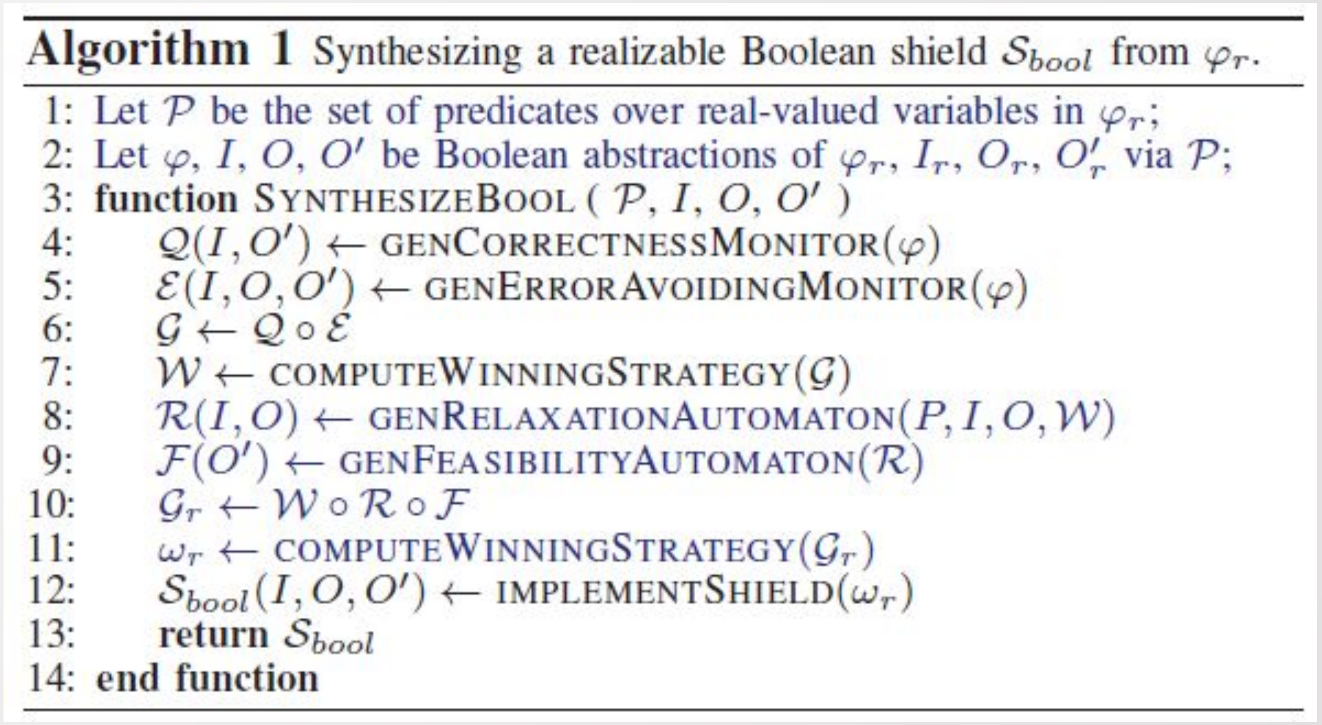
The result is a winning strategy ω_r, which in turn may be implemented as a reactive component S_{bool}. S_{bool} is a Mealy machine that takes I and O signals as input and returns the modified O' signals as output.

Due to the use of R and F, the output of S_{bool} is guaranteed to be realizable at run time.

Synthesizing The Boolean Shield

The idea is to check the compatibility of predicates inside the game-based algorithm for synthesizing the Boolean shield.

Algorithm 1 shows the procedure, where blue highlighted lines address the realizability issue while the remainder follows the classic algorithm.



Algorithm 1 Synthesizing a realizable Boolean shield $\mathcal{S}_{bool}$ from φ_r.

```
Let $\mathcal{P}$ be the set of predicates over real-valued variables in $\varphi_r$;
Let $\varphi$, $I$, $O$, $O'$ be Boolean abstractions of $\varphi_r$, $I_r$, $O_r$, $O'_r$ via $\mathcal{P}$;
function SynthesizeBool ( $\mathcal{P}, I, O, O'$ )
    $\mathcal{Q}(I, O') \leftarrow$ genCorrectnessMonitor($\varphi$)
    $\mathcal{E}(I, O, O') \leftarrow$ genErrorAvoidingMonitor($\varphi$)
    $\mathcal{G} \leftarrow \mathcal{Q} \circ \mathcal{E}$
    $\mathcal{W} \leftarrow$ computeWinningStrategy($\mathcal{G}$)
    $\mathcal{R}(I, O) \leftarrow$ genRelaxationAutomaton($P, I, O, \mathcal{W}$)
    $\mathcal{F}(O') \leftarrow$ genFeasibilityAutomaton($\mathcal{R}$)
    $\mathcal{G}_r \leftarrow \mathcal{W} \circ \mathcal{R} \circ \mathcal{F}$
    $\omega_r \leftarrow$ computeWinningStrategy($\mathcal{G}_r$)
    $\mathcal{S}_{bool}(I, O, O') \leftarrow$ implementShield($\omega_r$)
    return $\mathcal{S}_{bool}$
end function
```

- Error-avoiding monitor outlines all possible ways in which the antagonist may introduce errors in O and the protagonist may introduce corrections in O.
- Q, correctness monitor ensures that ϕ(I,O') always holds.

Generating the Real-Valued Signals

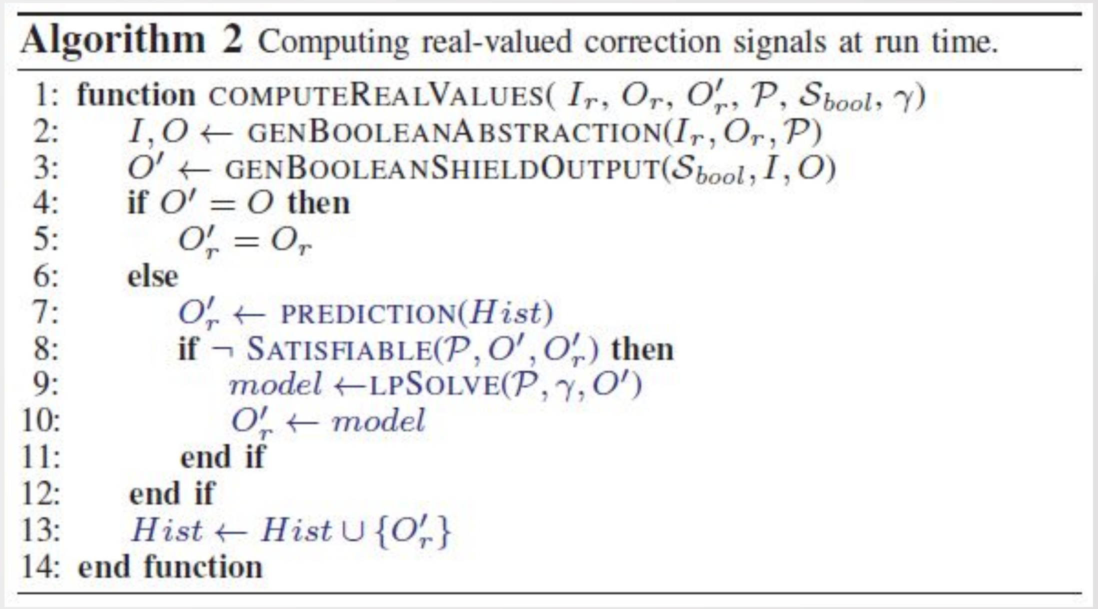
A utility function γ is used to evaluate the quality of the real-valued solution.

1) Robustness Optimization:
To ensure the signal is smooth, we restrict the LP problem using the objective function

$$\min\left(\left|val^i - \frac{\sum_{k=1}^{N} val^{i-k}}{N}\right|\right)$$

where val^i denotes the current value (at the i-th time step) and val^{i-k}, where k = 1, 2, . . . , denotes the value in the recent past.

2) Value Prediction and Validation:
To reduce the computational cost, we develop a two-phase optimization for computing the solution.



Algorithm 2 Computing real-valued correction signals at run time.

```
function ComputeRealValues( $I_r$, $O_r$, $O'_r$, $\mathcal{P}$, $\mathcal{S}_{bool}$, $\gamma$)
    $I, O \leftarrow$ GenBooleanAbstraction($I_r, O_r, \mathcal{P}$)
    $O' \leftarrow$ GenBooleanShieldOutput($\mathcal{S}_{bool}, I, O$)
    if $O' = O$ then
        $O'_r = O_r$
    else
        $O'_r \leftarrow$ Prediction($Hist$)
        if $\neg$ Satisfiable($\mathcal{P}, O', O'_r$) then
            $model \leftarrow$LPSolve($\mathcal{P}, \gamma, O'$)
            $O'_r \leftarrow model$
        end if
    end if
    $Hist \leftarrow Hist \cup \{O'_r\}$
end function
```

Safety Shield for Real

We have implemented our method as a tool that takes the automaton representation of a safety specification as input and returns a real-valued shield as output.

Each execution has two phases:

- generating Boolean values for signals in O', and
- generating real values for signals in O_r'.

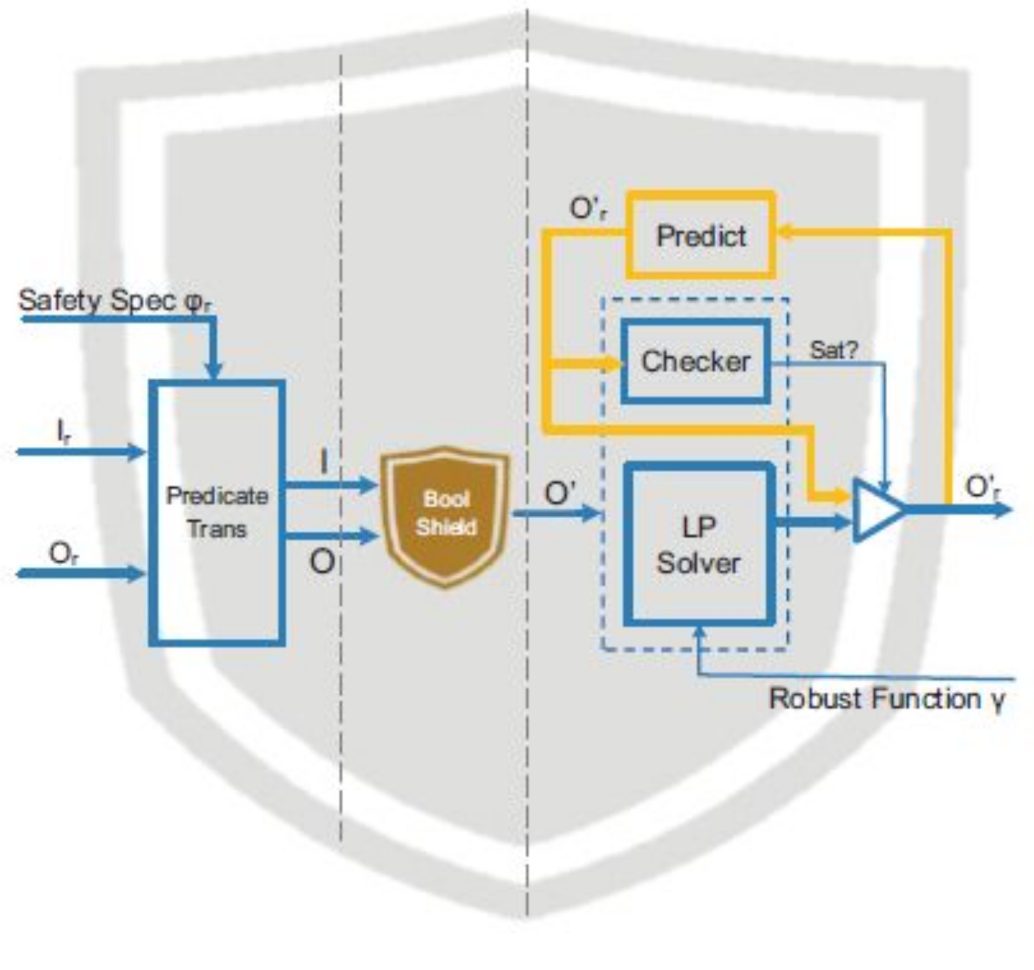


Fig. 1. Overview of the safety shield for real.

Case Study: Powertrain Control System

The green dashed line indicates the safe region, which varies as the system switches between different modes (transition events are highlighted with black dotted line).

The red dashed line represents violations of the specification by the O_r signals.
The solid red line represents corrections made in O_r.

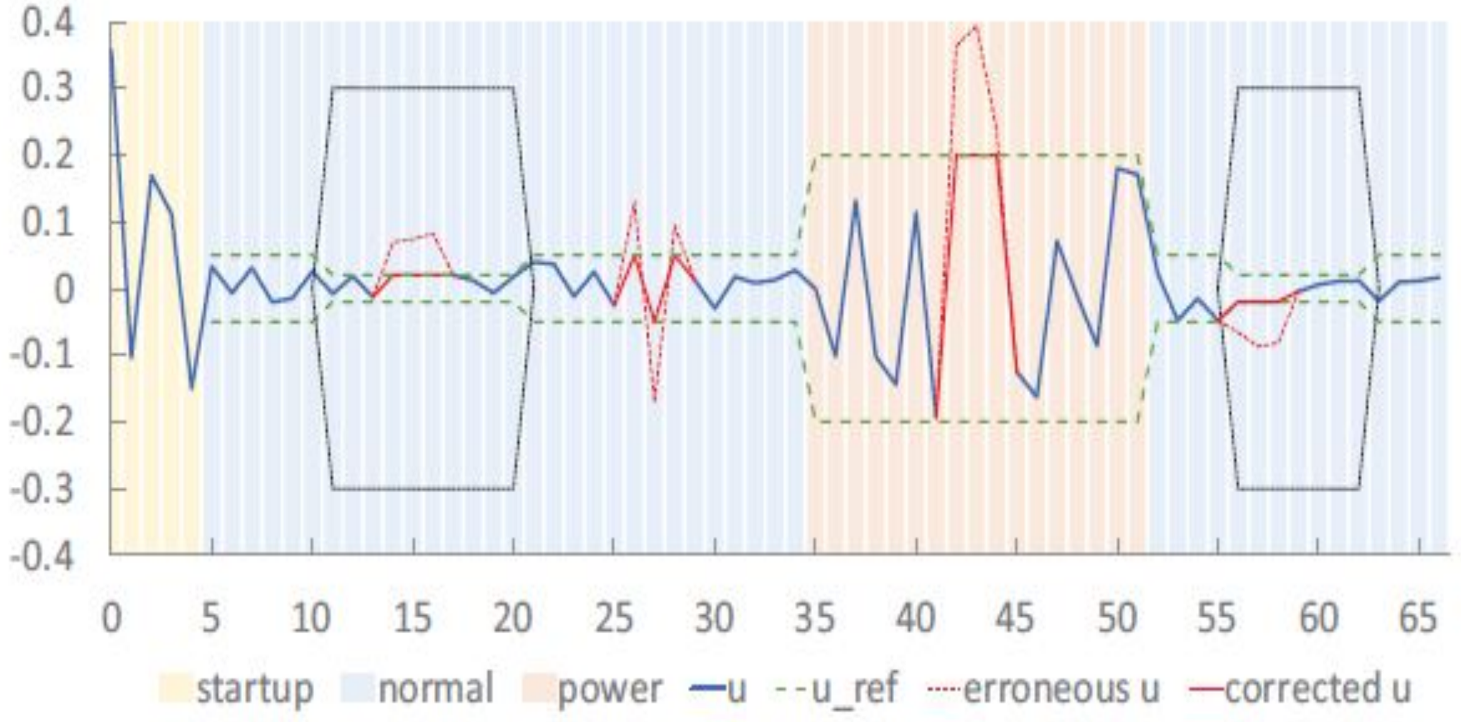


Fig. 7. Automotive powertrain system simulation (w/ and w/o the shield).

Case Study

At t = 5s, based on the safety specification, it is supposed to come to a stop.

However, since we injected an error at t = 6s (in red dashed line), there is an unexpected acceleration and, without the shield, there would have been a collision.

The blue lines show the behavior of the ego vehicle after corrections are made by the shield.

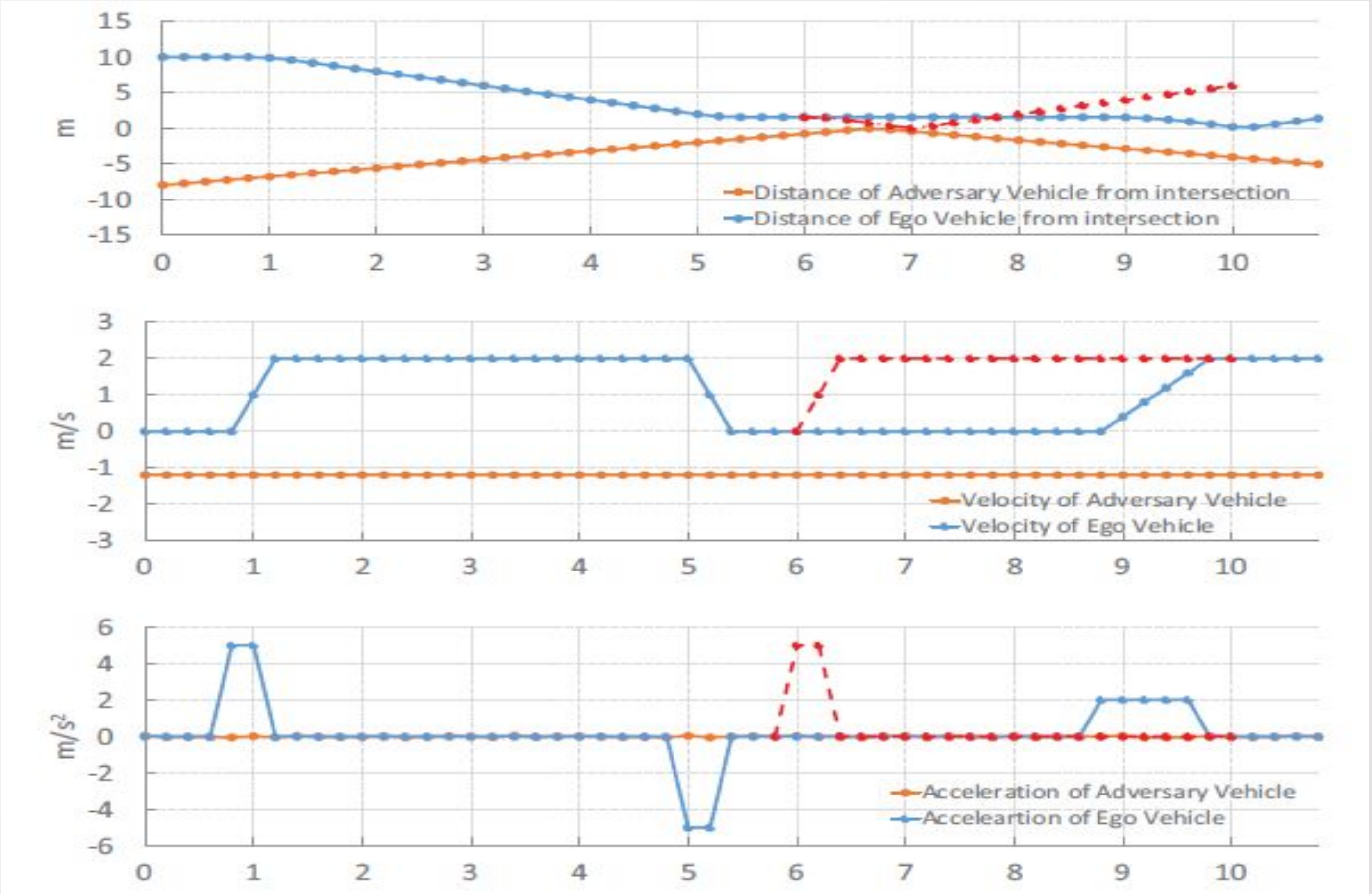
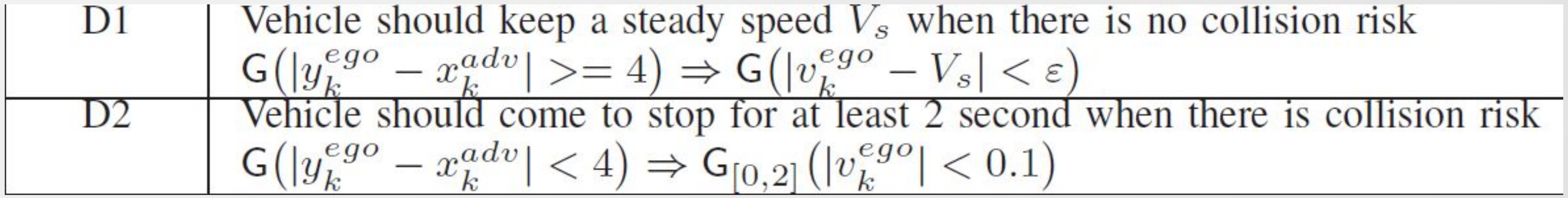


Fig. 8. Position, velocity and acceleration in autonomous driving simulation.



D1	Vehicle should keep a steady speed V_s when there is no collision risk $\mathsf{G}(\lvert y_k^{ego} - x_k^{adv}\rvert >= 4) \Rightarrow \mathsf{G}(\lvert v_k^{ego} - V_s\rvert < \varepsilon)$
D2	Vehicle should come to stop for at least 2 second when there is collision risk $\mathsf{G}(\lvert y_k^{ego} - x_k^{adv}\rvert < 4) \Rightarrow \mathsf{G}_{[0,2]}(\lvert v_k^{ego}\rvert < 0.1)$

Questions?