End-to-End Verification for Cyber-Physical Systems

Brandon Bohrer

Thesis Oral

Thesis Committee:
André Platzer (Chair)
Stefan Mitsch
Frank Pfenning
Bradley Schmerl
Tobias Nipkow (TUM)

April 16, 2021
Outline

1. Introduction
2. Related Work (Selected)
3. Modeling
4. Logic User
   - Proof Outline for Example Model
   - Kaisar Language
5. Engineer: Synthesis + Experiments
7. Conclusion
Cyber-Physical Systems (CPS) Need Correctness

Driving
(Studied in this thesis)

Flying

Grids
Constructive Differential Game Logic (CdGL) enables practical, end-to-end verification of cyber-physical systems.
Constructive Differential Game Logic (CdGL) enables practical, end-to-end verification of cyber-physical systems.

- Logician wants: Maximum logical formality
- Engineer wants: Useful, correct implementation artifacts
- Logic-User wants: Easy system modeling, system proof, maintenance
Constructive Differential Game Logic (CdGL) enables practical, end-to-end verification of cyber-physical systems.

- Logician wants: Maximum logical formality
- Engineer wants: Useful, correct implementation artifacts
- Logic-User wants: Easy system modeling, system proof, maintenance

Approach: VeriPhy method + tool (2 implementations)
Classical VeriPhy: Formal, Low-Level, Real-World Proof

Hybrid Systems Theorem Proving Model

\{\text{control};\text{physics}\}^* 

(KeYmaera X / Bellerophon)

dL Proof Script

Cyber-Physical System

Existing Control Code
Classical VeriPhy: Formal, Low-Level, Real-World Proof

Hybrid Systems Theorem Proving Model

{control;physics}*

(KeYmaera X/Bellerophon)
dL Proof Script

Sandbox Model

Standard ML Sandbox

Synthesis

Machine Code

Existing Control Code

Cyber-Physical System
Classical VeriPhy: Formal, Low-Level, Real-World Proof
Constructive VeriPhy: CdGL Supports Controllers

Curry Howard For Games:
- Proofs Are Code
- Systems Refine Games

Constructive Hybrid Games With Refinement (CdGL)

- Kaisar Theorem Proving
- Kaisar (Model+ Proof) Script

Monitor

Synthesis

Control

Optional Sandboxing

High-Level Strategy

Existing Code (Controller Optional)

Cyber-Physical System
Outline

1. Introduction
2. Related Work (Selected)
3. Modeling
4. Logic User
   - Proof Outline for Example Model
   - Kaisar Language
5. Engineer: Synthesis + Experiments
7. Conclusion
Related Work: End-to-End Verification (Selected)

Least | Most

End-to-Endness
- Other
- VD
- RC
- HA
- dL-Based Approaches
  - V1 Classical VeriPhy
  - V2 Constructive VeriPhy
  - HA High-Assurance SPIRAL

Modeling Practicality
- Other
- L
- T
- VD
- RC
- HA
- Coq?
- Coq-Based Approaches
  - VD VeriDrone
  - RC ROSCoq

Proof Practicality
- Coq
- HA
- Others (Non-Deductive)
  - T TuLiP
  - L LTLMoP
  - RF ReachFlow
  - D Drona
  - A Althoff, et al.
Related Work: End-to-End Verification (Selected)

**Least** | ........................................................................ | **Most**
--- | --- | ---
End-to-Endness
- Other
- VD
- RC
- HA
- V2
- V1

**Modeling Practicality**
- Other
- L
- T
- VD
- V1
- RC
- HA
- V2
- Coq?

**Proof Practicality**
- Coq
- HA
- V1
- V2

**dL-Based Approaches**
- V1 Classical VeriPhy
- V2 Constructive VeriPhy
- HA High-Assurance SPIRAL

**Coq-Based Approaches**
- VD VeriDrone
- RC ROSCoq

**Others (Non-Deductive)**
- T TuLiP
- L LTLMoP
- RF ReachFlow
- D Drona
- A Althoff, et al.
Kaisar Combines Existing Ideas

**Underlying Technologies:**
- Static Single Assignment
- Hereditary Harrop Formulas
- Refinement
- Structured Contexts
1 Introduction

2 Related Work (Selected)

3 Modeling

4 Logic User
   Proof Outline for Example Model
   Kaisar Language

5 Engineer: Synthesis + Experiments

6 Logician: Foundations + Formal Guarantees

7 Conclusion
1D Robot is Running Example
Driving is Game Between Angel and Demon

\(? (d \geq 0)\);
\{ v := * \}^d; !(0 \leq v \land v \leq L);
\ t := 0;
\{ d' = -v, t' = 1 \& t \leq \epsilon \};
\(? (t > \frac{1}{2} \cdot \epsilon)\);
Driving is Game Between Angel and Demon

Distance Assumption

\(? (d \geq 0); \{
\{ v := * \}^d; !(0 \leq v \land v \leq L); \\
\ \ \ t := 0; \\
\{ d' = -v, t' = 1 & t \leq \epsilon \}; \\
\ ?(t > \frac{1}{2} \cdot \epsilon); \\
\}\times
\)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(?\phi)</td>
<td>Assume (\phi)</td>
</tr>
<tr>
<td>!\phi</td>
<td>Prove (\phi)</td>
</tr>
<tr>
<td>(x := f)</td>
<td>Assign (x := f)</td>
</tr>
<tr>
<td>({x := *}^d)</td>
<td>Some (x)</td>
</tr>
<tr>
<td>(x := *)</td>
<td>Any (x)</td>
</tr>
<tr>
<td>({x' = f &amp; \psi})</td>
<td>Evolve (x' = f)</td>
</tr>
<tr>
<td>(\alpha \cup \beta)</td>
<td>during (\psi)</td>
</tr>
<tr>
<td>(\alpha; \beta)</td>
<td>Play (\alpha) or (\beta)</td>
</tr>
<tr>
<td>(\alpha \times \alpha^*)</td>
<td>Play (\alpha) then (\beta)</td>
</tr>
<tr>
<td></td>
<td>Repeat (\alpha)</td>
</tr>
</tbody>
</table>
Driving is Game Between Angel and Demon

Distance Assumption

\(? (d \geq 0); \{
\{ v := \ast \}^d; !(0 \leq v \land v \leq L); \\
\}
\}
\{ d' = -v, t' = 1 \& t \leq \epsilon \};
\{ t > \frac{1}{2} \cdot \epsilon \};
\}
\times

Choose speed (= velocity)

\|\|}^d; !(0 \leq v \land v \leq L);
\}
\{ d' = -v, t' = 1 \& t \leq \epsilon \};
\{ t > \frac{1}{2} \cdot \epsilon \};
\}
\times

\begin{align*}
\text{Statement} & \quad \text{Meaning} \\
?\phi & \quad \text{Assume } \phi \\
!\phi & \quad \text{Prove } \phi \\
x := f & \quad \text{Assign } x := f \\
\{ x := \ast \}^d & \quad \text{Some } x \\
x := \ast & \quad \text{Any } x \\
\{ x' = f \& \psi \} & \quad \text{Evolve } x' = f \\
& \text{ during } \psi \\
\alpha \cup \beta & \quad \text{Play } \alpha \text{ or } \beta \\
\alpha; \beta & \quad \text{Play } \alpha \text{ then } \beta \\
\alpha \times \alpha^* & \quad \text{Repeat } \alpha
\end{align*}
Driving is Game Between Angel and Demon

Distance Assumption

\(? (d \geq 0)\);
\{\}
\{v := * \}^d; \!(0 \leq v \land v \leq L); 
t := 0;
\{d' = -v, t' = 1 \land t \leq \epsilon\}; 
? (t > \frac{1}{2} \cdot \epsilon); 
\}
\}

Choose speed (= velocity)

\(\) 

Speed range

\(\) 

Statement | Meaning
--- | ---
\(?\phi\) | Assume \(\phi\)
\!\phi | Prove \(\phi\)
x := f | Assign \(x := f\)
\{x := * \}^d | Some \(x\)
\x := * | Any \(x\)
\{x' = f \& \psi\} | Evolve \(x' = f\) during \(\psi\)
\(\alpha \cup \beta\) | Play \(\alpha\) or \(\beta\)
\(\alpha; \beta\) | Play \(\alpha\) then \(\beta\)
\(\alpha \times \alpha^*\) | Repeat \(\alpha\)
Driving is Game Between Angel and Demon

Distance Assumption

Choose speed (= velocity)

Speed range

Ordinary Diff. Eq. (ODE)

\[ ?(d \geq 0); \]
\[ \{ \{ v := \ast \}^d; !(0 \leq v \land v \leq L); \]
\[ t := 0; \]
\[ \{ d' = -v, t' = 1 \& t \leq \epsilon \}; \]
\[ ?(t > \frac{1}{2} \cdot \epsilon); \]
\[ \times \]

\begin{tabular}{|c|c|}
\hline
Statement & Meaning \\
\hline
?\( \phi \) & Assume \( \phi \) \\
\hline
!\( \phi \) & Prove \( \phi \) \\
\hline
\( x := f \) & Assign \( x := f \) \\
\hline
\{ \( x := \ast \}^d \) & Some \( x \) \\
\hline
\( x := \ast \) & Any \( x \) \\
\hline
\{ \( x' = f \& \psi \} \) & Evolve \( x' = f \) during \( \psi \) \\
\hline
\( \alpha \cup \beta \) & Play \( \alpha \) or \( \beta \) \\
\hline
\( \alpha; \beta \) & Play \( \alpha \) then \( \beta \) \\
\hline
\( \alpha \times \alpha^* \) & Repeat \( \alpha \) \\
\hline
\end{tabular}

\[ d \]
\[ \hat{d} \]

\[ \text{Angel} \]

\[ \text{Demon} \]

\[ \text{STOP} \]
Driving is Game Between Angel and Demon

Distance Assumption

\(?(d \geq 0)\);
\{v := *\}^d; \!(0 \leq v \land v \leq L)\);
\(t := 0\);
\(d' = -v, t' = 1 \land t \leq \epsilon\};
?(t > \frac{1}{2} \cdot \epsilon);\}
\times

Choose speed (\(= \) velocity)

Speed range

Ordinary Diff. Eq. (ODE)

Statement

<table>
<thead>
<tr>
<th>(\phi)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assume (\phi)</td>
<td></td>
</tr>
<tr>
<td>Prove (\phi)</td>
<td></td>
</tr>
<tr>
<td>Assign (x := f)</td>
<td></td>
</tr>
<tr>
<td>Some (x)</td>
<td></td>
</tr>
<tr>
<td>Any (x)</td>
<td></td>
</tr>
<tr>
<td>Evolve (x' = f) during (\psi)</td>
<td></td>
</tr>
<tr>
<td>Play (\alpha \cup \beta)</td>
<td></td>
</tr>
<tr>
<td>Play (\alpha) or (\beta)</td>
<td></td>
</tr>
<tr>
<td>Play (\alpha) then (\beta)</td>
<td></td>
</tr>
<tr>
<td>Repeat (\alpha)</td>
<td></td>
</tr>
</tbody>
</table>

Upper Time Bound

Lower Time Bound
Driving is Game Between Angel and Demon

Distance Assumption

\(? (d \geq 0);\)

\{\{v := *\}^d; !(0 \leq v \land v \leq L); t := 0; \{d' = -v, t' = 1 \land t \leq \epsilon\}; ?(t > \frac{1}{2} \cdot \epsilon); \}\times

Choose speed (= velocity)

Speed range

Ordinary Diff. Eq. (ODE)

\{v := *\}^d, !(0 \leq v \land v \leq L);

Upper Time Bound

We Control Repetition

Lower Time Bound

Lower Time Bound

Distance Assumption

Choose speed (= velocity)

Speed range

Ordinary Diff. Eq. (ODE)

Upper Time Bound

We Control Repetition

Statement

Meaning

?\phi
Assume \phi

!\phi
Prove \phi

x := f
Assign x := f

{\{x := *\}^d
Some x

x := *
Any x

{x' = f \& \psi}
Evolve x' = f
during \psi

\alpha \cup \beta
Play \alpha or \beta

\alpha; \beta
Play \alpha then \beta

\alpha \times \alpha^*
Repeat \alpha
Want to Prove Safety and Liveness

\(? (d \geq 0);\)
\{
\{ v := \ast \}^d; \, !(0 \leq v \land v \leq L); \}
\{ d' = -v, \, t' = 1 \land t \leq \epsilon \};
!(d \geq 0); \quad \text{Safety}
? (t > \frac{1}{2} \cdot \epsilon); \quad \text{Safety}
\} \times
!(d \geq 0 \land d \leq \delta); \quad \text{Liveness}
Abstract: Don't Limit Controller

1D Motion, Static Obstacle

\(? (d \geq 0);\)
\{\{v := \ast\}^d; \neg (0 \leq v \land v \leq L); \}
\{t := 0;\)
\{d' = -v, t' = 1 & t \leq \epsilon;\)
\neg (d \geq 0);\)
\? (t > \frac{1}{2} \cdot \epsilon);\)
\}\times \neg (d \geq 0 \land d \leq \delta);
Abstract: Don’t Limit Controller

1D Motion, Static Obstacle

Instant control

\[ ?(d \geq 0); \]
\[ \{ v := \ast \}^d; \quad !(0 \leq v \land v \leq L); \]
\[ t := 0; \]
\[ \{ d' = -v, t' = 1 \& t \leq \epsilon \}; \]
\[ !(d \geq 0); \]
\[ ?(t > \frac{1}{2} \cdot \epsilon); \]
\[ \times \]
\[ !(d \geq 0 \land d \leq \delta); \]
We Discuss Sandbox in Passing

Classical VeriPhy: Always, automatically use sandbox
Constructive VeriPhy: Optionally, automatically generate sandbox (more control)
Outline

1. Introduction
2. Related Work (Selected)
3. Modeling
4. Logic User
   - Proof Outline for Example Model
     - Kaisar Language
5. Engineer: Synthesis + Experiments
7. Conclusion
Lightning Overview of CdGL Proofs

CdGL formula $[\alpha]\phi$ (resp. $\langle\alpha\rangle\phi$): Angel wins $\alpha$ with goal $\phi$ when Demon (resp. Angel) moves first.
Lightning Overview of CdGL Proofs

\[ \forall t : \mathbb{R}_{\geq 0} \forall r : [0, t] \psi(sol(r)) \rightarrow \psi(sol(t)) \]

\[ [x' = f \& \psi(x)] \phi(x) \]

**DI**

\[ \phi \quad \forall x (\psi \rightarrow [x' := f(\phi)']) \]

\[ [x' = f \& \psi] \phi \]

**DC**

\[ [x' = f \& \psi] \rho \quad [x' = f \& \psi \land \rho] \phi \]

\[ [x' = f \& \psi] \phi \]
Outline

1. Introduction
2. Related Work (Selected)
3. Modeling
4. Logic User
   - Proof Outline for Example Model
   - Kaisar Language
5. Engineer: Synthesis + Experiments
7. Conclusion
\(? (d \geq 0); \)
\{
\{v := \ast \}^d; ~(0 \leq v \land v \leq L); \\
t := 0; \\
\{ d' = -v, t' = 1 \land t \leq \epsilon \}; \\
? (t > \frac{1}{2} \cdot \epsilon); \\
\}\}
\( ?(d \geq 0); \)
\[
\begin{aligned}
\{ & v := \ast \}^d; \quad !(0 \leq v \land v \leq L); \\
& t := 0; \\
& \{ d' = -v, t' = 1 \land t \leq \epsilon \}; \\
& ?(t > \frac{1}{2} \cdot \epsilon);
\end{aligned}
\]
\)

\( ?(d \geq 0); \)
\[
\begin{aligned}
while(d \geq L \cdot \epsilon) \{ \\
& v := L; \quad !(0 \leq v \land v \leq L); \\
& t := 0; \\
& \{ d' = -v, t' = 1 \land t \leq \epsilon \}; \\
& ?(t > \frac{1}{2} \cdot \epsilon);
\}
\]
Proof Outline Approach, Outlined

?(d ≥ 0);
\{ v := \ast \}^d; !(0 ≤ v \land v ≤ L);
\begin{aligned}
    & t := 0; \\
    & \{ d' = -v, t' = 1 \land t ≤ \epsilon \}; \\
    & !(t > \frac{1}{2} \cdot \epsilon);
\end{aligned}
\times

?(d ≥ 0);
while(d ≥ L \cdot \epsilon)\
\begin{aligned}
    & v := L; !(0 ≤ v \land v ≤ L); \\
    & t := 0; \\
    & \{ d' = -v, t' = 1 \land t ≤ \epsilon \}\
    & !(d - (\epsilon - t) \cdot v ≥ 0) \\
    & !(d \leq \text{old}(d) - t \cdot v) \\
\end{aligned};
!(d ≥ 0);
?(t > \frac{1}{2} \cdot \epsilon);
!(d ≥ 0 \land d ≤ \text{old}(d) - L \cdot \frac{\epsilon}{2});

!(d ≥ 0 \land d ≤ L \cdot \epsilon);
Proof Outline Approach, Outlined

\( \exists (d \geq 0); \)
\[ \begin{align*}
\{ & v := * \}^d; \quad ! (0 \leq v \land v \leq L); \\
& t := 0; \\
& \{ d' = -v, t' = 1 \land t \leq \epsilon \}; \\
& ?(t > \frac{1}{2} \cdot \epsilon);
\end{align*} \]

Cross

\( \exists (d \geq 0); \)
\[ \begin{align*}
while (d \geq L \cdot \epsilon) & \{ \\
& v := L; \quad ! (0 \leq v \land v \leq L); \quad ..proof.. \\
& t := 0; \\
& \{ d' = -v, t' = 1 \land t \leq \epsilon \} \\
& \quad ! (d - (\epsilon - t) \cdot v \geq 0) \quad ..proof.. \\
& \quad ! (d \leq \text{old}(d) - t \cdot v) \quad ..proof.. \}; \\
& ! (d \geq 0); \quad ..proof.. \\
& ?(t > \frac{1}{2} \cdot \epsilon); \\
& ! (d \geq 0 \land d \leq \text{old}(d) - L \cdot \frac{\epsilon}{2}); \quad ..proof.. \\
\} \\
& ! (d \geq 0 \land d \leq L \cdot \epsilon); \quad ..proof.. \]
What to Prove?

\(? (d \geq 0); \{
\{ v := \ast \}^d; \! (0 \leq v \land v \leq L); 
\{ d' = -v, t' = 1 \land t \leq \epsilon \}; 
!(d \geq 0); \}
\times
!(d \geq 0 \land d \leq \delta);\}
\)

Holds for some \( v \)

Each time safe

End live + safe

\[
\begin{align*}
[!][\alpha; \! \psi] \phi & \\
[\alpha] (\psi \land \phi)
\end{align*}
\]
Heart of Proof = Strategy = Control + Monitors

\(?d \geq 0\);
\{v := \ast\}^d; !(0 \leq v \land v \leq L);
\ t := 0;
\{d' = -v, t' = 1 \land t \leq \epsilon\};
\ !(d \geq 0);
\ !(t > \frac{1}{2} \cdot \epsilon);\}
\times
\ !(d \geq 0 \land d \leq \delta);\}

Choose \(v\)

Choose guard

\[\exists \phi \quad \frac{\exists x \phi}{[x := \ast]^d \phi}\]
Heart of Proof = Strategy = Control + Monitors

\[(d \geq 0);\]
\[
\{ v := \ast \}^d; \neg (0 \leq v \land v \leq L); \]
\[
t := 0;\]
\[
\{ d' = -v, t' = 1 \land t \leq \epsilon \}; \]
\[
\neg (d \geq 0);\]
\[
\{ t > \frac{1}{2} \cdot \epsilon \}; \]
\[
\} \times\]
\[
\neg (d \geq 0 \land d \leq \delta); \]

Choose \( v \)

Choose invariant

Choose guard

\[\exists x \phi\]
\[\left[ \{ x := \ast \}^d \right] \phi\]

DC

\[ [x' = f \& \psi] \rho \]
\[ [x' = f \& \psi \land \rho] \phi \]

\[ [x' = f \& \psi] \phi \]
Controls include Assignments and Guards

\(? (d \geq 0);\)
\(\text{while}(d \geq L \cdot \epsilon)\{\)

\(v := L; \quad !(0 \leq v \land v \leq L);\)
\(t := 0;\)
\(\{d' = -v, t' = 1 \& t \leq \epsilon\};\)
\(! (d \geq 0);\)
\(? (t > \frac{1}{2} \cdot \epsilon);\)
\(! (d \geq 0 \land d \leq \text{old}(d) - L \cdot \frac{\epsilon}{2});\)
\}\)
\(! (d \geq 0 \land d \leq L \cdot \epsilon);\)

\[\begin{align*}
\text{Safety + Progress} & \quad [:\ast^d]l\quad [x := f]\phi \\
& \quad [\{x := \ast\}^d]\phi \\
& \quad [\ast^d]\text{wh} \quad [\text{while}(\cdots)\{\cdots\}]\phi \\
& \quad [\{\alpha^*\}^d]\phi
\end{align*}\]
Monitors Include Invariants

\(? (d \geq 0)\);
while (d \geq L \cdot \varepsilon) \{
    v := L; \! (0 \leq v \land v \leq L);
    t := 0;
    \{ d' = -v, t' = 1 \& t \leq \varepsilon \\
        \! (d - (\varepsilon - t) \cdot v \geq 0) \\
        \! (d \leq old(d) - t \cdot v) \};
    \! (d \geq 0);
    \? (t > \frac{1}{2} \cdot \varepsilon);
    \! (d \geq 0 \land d \leq old(d) - L \cdot \frac{\varepsilon}{2});
}\}
\! (d \geq 0 \land d \leq L \cdot \varepsilon);

Safety Invariant

Progress Invariant

\[ [x' = f \& \psi]_\rho \quad [x' = f \& \psi \land \rho]_\phi \]

DC

[\[ x' = f \& \psi ]_\phi \]

25 / 64
Outline

1. Introduction
2. Related Work (Selected)
3. Modeling
4. Logic User
   - Proof Outline for Example Model
     - Kaisar Language
5. Engineer: Synthesis + Experiments
7. Conclusion
What are Kaisar’s Goals?

- Reduce learning curve
- Put proofs inline with model for readability
- Minimize coupling for maintainability
let \( \text{inv}(\text{pos}) \leftrightarrow (d \geq 0 \& (d@\text{init} - d) \geq \text{pos}) \);

let \( \text{liveFactor}() = 4 \);

let \( \text{liveIncr}() = V*\text{eps}/(\text{liveFactor}() * \text{liveFactor}()) \);

?\( (d \geq 0 \& V > 0 \& \text{eps} > 0 \& v=0 \& t=0) \);

\[
\text{init}:
\text{for (pos := 0; } !\text{conv.}(d \geq 0 \& (d@\text{init} - d) \geq \text{pos});
?\text{grd.}(\text{pos} \leftarrow d@\text{init} \& d \geq V*\text{eps}); \text{pos} := \text{pos} + \text{liveIncr}())\{ \\
\text{body: } v:=V;
\{ t:=0; \{ d'=-v, t'=1 \& ?(t <\text{eps}) \& ! (d \geq v*(\text{eps}-t)) \& ! (d \leq d@\text{body} - v*t/\text{liveFactor}()) \}; \}
? (t \geq \text{eps}/\text{liveFactor}());
!(\text{inv(pos + liveIncr())}) \text{ using <named_facts> by auto;}\}
!(\text{pos} >= d@\text{init} - \text{eps} \mid d \leq V*\text{eps} + \text{eps}) \text{ by guard(eps)};
!(d \geq 0 \& d \leq (V+1)*\text{eps});\]
let inv(pos) <- (d >= 0 & (d@init - d) >= pos);
let liveFactor() = 4;
let liveIncr() = V*eps/(liveFactor() * liveFactor());
?(d >= 0 & V > 0 & eps > 0 & v=0 & t=0);

init:
for (pos := 0; !conv:(d >= 0 & (d@init - d) >= pos);
 ?grd:(pos <= d@init & d >= V*eps); pos := pos + liveIncr()){
 body: v:=V;
 {t:=0; { d'=-v, t'=1 & ?(t <= eps) & !(d >= v*(eps-t))
 & !(d <= d@body - v*t/liveFactor());
 ?(t >= eps/liveFactor());
 !(inv(pos + liveIncr())) using <named_facts> by auto;}
 !(pos >= d@init - eps | d <= V*eps + eps) by guard(eps);
 !(d >= 0 & d <= (V+1)*eps);
let inv(pos) <-> (d >= 0 & (d@init - d) >= pos);
let liveFactor() = 4;
let liveIncr() = V*eps/(liveFactor() * liveFactor());
?(d >= 0 & V > 0 & eps > 0 & v=0 & t=0);

init:
for (pos := 0; !conv:(d >= 0 & (d@init - d) >= pos);
  ?grd:(pos <= d@init & d >= V*eps); pos := pos + liveIncr()){
body: v:=V;
  {t:=0; { d'=-v, t'=1 & ?(t <= eps) & !(d >= v*(eps-t))
            & !(d <= d@body - v*t/liveFactor());
  }?(t >= eps/liveFactor());
  !(inv(pos + liveIncr())) using <named_facts> by auto;}
!(pos >= d@init - eps | d <= V*eps + eps) by guard(eps);
!(d >= 0 & d <= (V+1)*eps);
# Kaisar Often Reduces Length + Maintenance

<table>
<thead>
<tr>
<th>Language</th>
<th>Name</th>
<th>Lines</th>
<th>Same</th>
<th>Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bellerophon</td>
<td>PLDI-DC</td>
<td>15</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td></td>
<td>PLDI-AS</td>
<td>42</td>
<td>9</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>PLDI-TAC</td>
<td>39</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>PLDI-RA</td>
<td>28</td>
<td>10</td>
<td>18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Language</th>
<th>Name</th>
<th>Lines</th>
<th>Same</th>
<th>Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kaisar</td>
<td>PLDI-DC</td>
<td>7</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td></td>
<td>PLDI-AS</td>
<td>10</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>PLDI-TAC</td>
<td>9</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>PLDI-RA</td>
<td>15</td>
<td>2</td>
<td>13</td>
</tr>
</tbody>
</table>
# Kaisar Often Reduces Length + Maintenance

<table>
<thead>
<tr>
<th>Language</th>
<th>Name</th>
<th>Lines</th>
<th>Same</th>
<th>Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bellerophon</td>
<td>PLDI-DC</td>
<td>15</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td></td>
<td>PLDI-AS</td>
<td>42</td>
<td>9</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>PLDI-TAC</td>
<td>39</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>PLDI-RA</td>
<td>28</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>IJRR</td>
<td>88</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Language</th>
<th>Name</th>
<th>Lines</th>
<th>Same</th>
<th>Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kaisar</td>
<td>PLDI-DC</td>
<td>7</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td></td>
<td>PLDI-AS</td>
<td>10</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>PLDI-TAC</td>
<td>9</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>PLDI-RA</td>
<td>15</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>IJRR</td>
<td>62</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>
Kaisar Often Reduces Length + Maintenance

<table>
<thead>
<tr>
<th>Language</th>
<th>Name</th>
<th>Lines</th>
<th>Same</th>
<th>Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bellerophon</td>
<td>PLDI-DC</td>
<td>15</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td></td>
<td>PLDI-AS</td>
<td>42</td>
<td>9</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>PLDI-TAC</td>
<td>39</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>PLDI-RA</td>
<td>28</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>IJRR</td>
<td>88</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td></td>
<td>RA-L</td>
<td>294</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Language</th>
<th>Name</th>
<th>Lines</th>
<th>Same</th>
<th>Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kaisar</td>
<td>PLDI-DC</td>
<td>7</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td></td>
<td>PLDI-AS</td>
<td>10</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>PLDI-TAC</td>
<td>9</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>PLDI-RA</td>
<td>15</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>IJRR</td>
<td>62</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td></td>
<td>RA-L</td>
<td>491</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>
Outline

1 Introduction
2 Related Work (Selected)
3 Modeling
4 Logic User
   Proof Outline for Example Model
   Kaisar Language
5 Engineer: Synthesis + Experiments
6 Logician: Foundations + Formal Guarantees
7 Conclusion
Angel Tests Do Not Need Code

\(? (d \geq 0) ; \)
while \( (d \geq L \cdot \epsilon) \) {
\( v := L ; \) !\((0 \leq v \land v \leq L) ; \)
\( t := 0 ; \)
\( \{ d' = -v , t' = 1 \land t \leq \epsilon \) 
\!\((d - (\epsilon - t) \cdot v \geq 0) \)
\!\((d \leq old(d) - t \cdot v) \}; \)
\!\((d \geq 0) ; \)
? \((t > \frac{1}{2} \cdot \epsilon) ; \)
\!\((d \geq 0 \land d \leq old(d) - L \cdot \frac{\epsilon}{2}) ; \)
\}
\!\((d \geq 0 \land d \leq \delta) ; \)
Assumptions and ODEs are Monitored

\[
\begin{align*}
?(d \geq 0); \\
\text{while}(d \geq L \cdot \epsilon) \{ \\
\quad v := L; \\
\quad t := 0; \\
\quad \{ d' = -v, t' = 1 \& t \leq \epsilon \\
\quad \quad !(d - (\epsilon - t) \cdot v \geq 0) \\
\quad \quad !(d \leq \text{old}(d) - t \cdot v)\}; \\
\quad ?(t > \frac{1}{2} \cdot \epsilon); \\
\} \\
\end{align*}
\]

\[
\begin{align*}
\text{ref} & \geq * \\
?\phi & \geq ?\phi
\end{align*}
\]
if(\neg(d > 0))
    throw MonitorFailure();
while(d ≥ L \cdot \epsilon) {
    v := L;
    t := 0;
    \{d' = -v, t' = 1 & t ≤ \epsilon
      !(d - (\epsilon - t) \cdot v ≥ 0)
      !(d ≤ old(d) - t \cdot v)\};
    if(\neg(t > \frac{1}{2} \cdot \epsilon))
        throw MonitorFailure();
}

\[
\begin{align*}
\text{ref} & \triangleright \ast \\
\text{throw} & \triangleright \ast \\
\text{if} & \triangleright \phi \geq \phi \\
\text{throw} & \triangleright \text{Err}() \geq \alpha \\
\text{if} & \triangleright \phi \rightarrow \alpha \geq \gamma \quad \beta \geq \gamma \\
\text{if} & \triangleright \{\text{if}(\phi)\alpha \text{ else } \beta\} \geq \gamma
\end{align*}
\]
ODEs are Monitored as Physical State Changes

\[
\text{if}(\neg(d > 0)) \\
\quad \text{throw MonitorFailure()}; \\
\text{while}(d \geq L \cdot \epsilon)\
\quad \text{v} := L; \\
\quad \text{tOld} := 0; \\
\quad \text{dOld} := d; d := *; \ t := *; \\
\quad \text{if}(\neg(t \leq \epsilon \land d - (\epsilon - t) \cdot v \geq 0 \\
\quad \land d \leq dOld - t \cdot v)) \\
\quad \text{throw MonitorFailure()}; \\
\quad \text{if}(\neg(t > \frac{1}{2} \cdot \epsilon)) \\
\quad \text{throw MonitorFailure()}; \\
\]
Need to Implement Demon

\[
\text{if}(\neg(d > 0))
\]

\[
\text{throw MonitorFailure();}
\]

\[
\text{while}(d \geq L \cdot \epsilon)\{
\]

\[
v := L;
\]

\[
t_{\text{Old}} := 0;
\]

\[
d_{\text{Old}} := d; d := *; t := *
\]

\[
\text{if}(\neg(t \leq \epsilon \land d - (\epsilon - t) \cdot v \geq 0
\land d \leq d_{\text{Old}} - t \cdot v))
\]

\[
\text{throw MonitorFailure();}
\]

\[
\text{if}(\neg(t > \frac{1}{2} \cdot \epsilon))
\]

\[
\text{throw MonitorFailure();}
\]

\[
}\}
\]
Demon is Reusable Across Strategies

```plaintext
if(\neg(d > 0))
    throw MonitorFailure();
while(d \geq L \cdot \epsilon)\
    \{ 
        v := L;
        tOld := 0;
        dOld := d; d := \ast; t := \ast;
        if(\neg(t \leq \epsilon \land d - (\epsilon - t) \cdot v \geq 0 
            \land d \leq dOld - t \cdot v))
            throw MonitorFailure();
        if(\neg(t > \frac{1}{2} \cdot \epsilon))
            throw MonitorFailure();
    \}

?(d \geq 0);
while(Guard)\
    \{ 
        v := Speed(v, d); !(0 \leq v \land v \leq L);
        t := 0;
        \{d' = -v, t' = 1 \land t \leq \epsilon
            !(Invariant)\};
        !(d \geq 0);
        ?(t > \frac{1}{2} \cdot \epsilon);
        !(d \geq 0 \land d \leq \text{old}(d) - \text{Metric});
    \}

!(d \geq 0 \land d \leq L \cdot \epsilon);
```
void sense(num_t* out){
    out[0] = senseDist();
    out[1] = senseTime();
    out[2] = ....
}

void actuate(num_t* in){
    wheelL.setVel(in[0]);
    wheelR.setVel(in[0]);
}

void external_control(num_t* out){
    out[0] = complexCode(senseDist(),senseTime());
}
Robot Reaches Goal

- **Obstacle Static**

![Graph showing distance (cm) vs. time (s)]
Receding Obstacle Safe, Not Live

![Graph showing distance vs. time for obstacle static and obstacle receding.](image)

- **Obstacle Static**
- **Obstacle Receding**

Distance [cm] vs. Time [s]
Robot Catches Liveness Failures

![Graph showing time vs. distance for static and receding obstacles](image.png)

- **Obstacle Static**
- **Obstacle Receding**

*Dist.* [cm] vs. *Time* [s] graph with data points marked at specific times.
Approach Scales to Complex 2D Driving (RA-L Model)

\[ \alpha \equiv \{ \text{input}; \text{ctrl}; \text{physics} \} \]

\[ \text{input} \equiv (x, y) = (v_l, v_h); k = ?(\text{Admiss}) \]

\[ \text{ctrl} \equiv \{ a = \cdot \} \]

\[ \text{physics} \equiv t = 0; \{ t' = 1; v' = a; x' = v k (y - 1 k); y' = v k (-x) \} \]

\[ \text{Admiss} \equiv \text{OnCircle} \land x > 0 \land 0 \leq v_l < v_h \land \max(A, B) T \leq v_h - v_l \]

\[ \text{Feas} \equiv -B \leq a \leq A \]

Diagram:
- LIMIT [v_l, v_h]
- k < 0
- k = 0
- k > 0
- (x, y)
- +y
- +x
Approach Scales to Complex 2D Driving (RA-L Model)

\[ \alpha^* \equiv \{ \text{input; ctrl; physics} \}^* \]

\[ \text{input} \equiv (x, y) := *; [vl, vh] := *; k := *; ?(\text{Admiss}); \]

\[ \text{ctrl} \equiv \{ \{ a := * \}^d; !(\text{Feas}); \} \]

\[ \text{physics} \equiv t := 0; \{ t' = 1, v' = a, x' = vk \left( y - \frac{1}{k} \right), y' = vk (-x), \}
\]

\[ \text{Admiss} \equiv \text{OnCircle} \land x > 0 \land 0 \leq vl < vh \land \max(A, B) T \leq vh - vl \]

\[ \text{Feas} \equiv -B \leq a \leq A \]
Model Safely Integrated with Existing Simulation

Summary of empirical results:

- Courses completed safety at $\approx 75\%$ of human speed
- Control and plant each noncompliant $\approx 1\%$ of time
- Caveat: “Co-operative” Demon adjusts path to help Angel

**Takeaway:** VeriPhy can integrate with existing simulations.
Outline

1. Introduction
2. Related Work (Selected)
3. Modeling
4. Logic User
   - Proof Outline for Example Model
   - Kaisar Language
5. Engineer: Synthesis + Experiments
7. Conclusion
Summary of End-to-End Proofs

Theorem (Soundness)

*If a formula has a proof, it is true.*

(\(\mathcal{DL}: \text{machine-checked}; \text{CdGL}: \text{on paper}\))

Theorem (Sandboxing)

*High-level sandbox soundly implements model.* (\(\mathcal{DL}: \text{only}\))

Theorem (Compilation)

*Compiled code soundly implements discrete program.* (\(\mathcal{DL}: \text{only}\))

Theorem (End-to-End)

*Compiled program is safe when model assumptions hold, and raises an alarm when they do not.* (\(\mathcal{DL}: \text{only}\))
Constructive Refinement Reduces Games to Systems

• Constructive refinement ($\alpha \leq_{[]} \beta$): each $[\alpha] \phi$ constructively implies $[\beta] \phi$

Let $A$ be a proof of $([\alpha] \phi)$ and let $\gamma$ be the reification of $A$, i.e., ($A \leadsto \gamma$).
Constructive Refinement Reduces Games to Systems

- Constructive refinement \((\alpha \leq [\beta])\): each \([\alpha] \phi\) constructively implies \([\beta] \phi\)

Let \(A\) be a proof of \(([\alpha] \phi)\) and let \(\gamma\) be the reification of \(A\), i.e., \((A \leadsto \gamma)\).

Theorem (Systemhood)

\(\gamma\) is a hybrid system.
Constructive Refinement Reduces Games to Systems

- Constructive refinement ($\alpha \leq \beta$): each $[\alpha]\phi$ constructively implies $[\beta]\phi$

Let $A$ be a proof of $([\alpha]\phi)$ and let $\gamma$ be the reification of $A$, i.e., ($A \rightsquigarrow \gamma$).

**Theorem (Systemhood)**

$\gamma$ is a hybrid system.

**Theorem (System satisfies postcondition)**

$[\gamma]\phi$ is provable.

VeriPhy Extracts Safe Game
Constructive Refinement Reduces Games to Systems

- Constructive refinement ($\alpha \leq_{[]} \beta$): each $[\alpha]\phi$ constructively implies $[\beta]\phi$

Let $A$ be a proof of ($[\alpha]\phi$) and let $\gamma$ be the reification of $A$, i.e., $(A \leadsto \gamma)$.

Theorem (Systemhood)

$\gamma$ is a hybrid system.

Theorem (System satisfies postcondition)

$[\gamma]\phi$ is provable.  

VeriPhy Extracts Safe Game

Theorem (System refines game)

$\gamma \leq_{[]} \alpha$ is provable.  

Kaisar Proves it All
Outline

1. Introduction
2. Related Work (Selected)
3. Modeling
4. Logic User
   - Proof Outline for Example Model
   - Kaisar Language
5. Engineer: Synthesis + Experiments
7. Conclusion
End-to-End Guarantees
Formal Methods Comparison

<table>
<thead>
<tr>
<th></th>
<th>Logician</th>
<th>Engineer</th>
<th>Logic-User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical VeriPhy</td>
<td>formal</td>
<td>sound monitor program</td>
<td>less prone, less labor</td>
</tr>
<tr>
<td>Constructive VeriPhy</td>
<td>paper</td>
<td>monitors, controls</td>
<td>least prone, less labor</td>
</tr>
</tbody>
</table>
Formal Methods Comparison

<table>
<thead>
<tr>
<th></th>
<th>Logician</th>
<th>Engineer</th>
<th>Logic-User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical VeriPhy</td>
<td>formal</td>
<td>sound monitor program</td>
<td>less prone, less labor</td>
</tr>
<tr>
<td>Constructive VeriPhy</td>
<td>paper</td>
<td>monitors, controls</td>
<td>least prone, less labor</td>
</tr>
</tbody>
</table>

Desired Verified Step

- Formalize CdGL Soundness
- Formalize Kaisar Soundness
- (Verified) Compilation

Difficulty and Reason

- Semantics too rich
- No formal Kaisar rules
- Generalizations needed
Formal Methods Comparison

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical VeriPhy</td>
<td>formal</td>
<td>sound monitor program</td>
<td>less prone, less labor</td>
</tr>
<tr>
<td>Constructive VeriPhy</td>
<td>paper</td>
<td>monitors, controls</td>
<td>least prone, less labor</td>
</tr>
</tbody>
</table>

Future work:

1. Formalize CdGL strategies
2. Export Kaisar to CdGL
3. Merge classical and constructive VeriPhy
Formal Methods Comparison

<table>
<thead>
<tr>
<th></th>
<th>Logician</th>
<th>Engineer</th>
<th>Logic-User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical VeriPhy</td>
<td>formal</td>
<td>sound monitor program</td>
<td>less prone, less labor</td>
</tr>
<tr>
<td>Constructive VeriPhy</td>
<td>paper</td>
<td>monitors, controls</td>
<td>least prone, less labor</td>
</tr>
</tbody>
</table>

Future work:

1. Formalize CdGL strategies
2. Export Kaisar to CdGL
3. Merge classical and constructive VeriPhy
4. Apply to new models (e.g., 2D bicycle model)
Constructive Differential Game Logic (CdGL) enables practical, end-to-end verification of cyber-physical systems.
Background: Formal Methods

Modeling

Model-Checking

Simulation

Theorem-Proving
**Related Work: Proof Languages**

**Concepts in Prior Work**
- **Structured Proof**
  - See: Mizar, Isar, etc.
  - Block-structured, declarative style. "Do for proofs what structured programming did"

- **Input-Output Specs**
  - See: VDM, Event-B, TLA, KeYmaera X, to name a few
  - Specification relates variable's initial + final values
  - Limited references to intermediate values
  - Example:
    ```
    @ensure(n == 2*old(n))
    void db(int& n) { n = 2*n; }
    ```

- **Lexical Scope**
  - See: Most programming languages, many stateless proof languages.

- **Unstructured Proof**
  - See: Bellerophon, Coq-script, apply-script
  - Program that freely combines proof tactics

- **Proof-By-Annotation**
  - See: ESC, Outlines, etc.
  - Focus on writing a program. Put the proof in annotations

- **Definitions**
  - See: Many proof languages
  - User-defined constructs ease reading + maintenance

- **Refinement**
  - See: Event-B, KAT, dRL
  - Transfer results across abstract, concrete models + code

**Generalizing + Combining Concepts**
- **Labeled Proofs**: Explore past, future, hypothetical states
  - `theLabel: e@label e@label(args)`
    - Assigns label to location
    - Evaluates `e` at label
    - Evaluate in hypothetical state

- **Structure + Annotations Mix**
  - `!<Formula>` by proof
  - `<steps>` end
  - Analogy: Structured proof inside unstructured

- **Uniform Ghost Reasoning**
  - `/++...+/-` Use ... in proof, not model
  - `/-- ... --/` Use ... in model, not proof
  - Add and remove proofs, code uniformly. Important for refinement automation

**Novel Technical Challenges**
- **Lexical Scope Despite State**
  - Program statements falsify previously known facts
  - **Solution**: Static Single Assignment

- **Hybrid Game Refinement**
  - Relate proofs of games to proofs of winning strategies
  - **Solution**: CdGl with refinement (§6)

- **Rich Reference across States**
  - What does expression mean in a different state?
  - **Solution**: Static Single Assignment

- **Constructive Arithmetic**
  - When are first-order formulas over constructive reals decidable?
  - **Solution**: Hereditary Harrop Formulas
Kaisar Generalizes Existing Ideas

In Related Work:

Historical Reference

\[ x \geq \text{old}(x) \]

Enables: Invariants

In Kaisar:

Labeled Reference

\[ x \leq \text{x@future}(\text{args}) \]

Enables: Predictive Proofs
Related Work: End-to-End Verification (Selected)

End-to-Endness

- Other
- VD
- RC
- V1
- V2

Modeling Ease

- Coq
- V1
- HA
- V2
- L
- T

Proof Ease

- Coq
- V1
- HA
- V2

Expressiveness

- Other
- L
- T
- RC
- V1
- V2
- Coq

Least
- VD
- HA
- RC

Most
- V1
- V2
- Coq

- dL-Based Approaches
  - HA: High-Assurance SPIRAL
  - V1: Classical VeriPhy
  - V2: Constructive VeriPhy

- Coq-Based Approaches
  - VD: VeriDrone
  - RC: ROSCoq

- Others (Non-Deductive)
  - T: TuLiP
  - L: LTLMoP
  - RF: ReachFlow
  - D: Drona
  - A: Althoff, et. al.
## More End-to-End Comparison

<table>
<thead>
<tr>
<th>Approach</th>
<th>Output</th>
<th>Connection</th>
<th>Input</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachflow</td>
<td>Commands</td>
<td>Monitoring</td>
<td>Hybrid</td>
<td>Safety Monitor</td>
</tr>
<tr>
<td>Drona</td>
<td>Motion</td>
<td>Synthesis</td>
<td>STL</td>
<td>Reach-Avoid</td>
</tr>
<tr>
<td>Althoff, et. al</td>
<td>Commands</td>
<td>Monitoring*</td>
<td>Hybrid System</td>
<td>Safety Monitor*</td>
</tr>
<tr>
<td>LTLMoP+ TuLiP</td>
<td>Commands</td>
<td>Synthesis</td>
<td>Discrete</td>
<td>LTL</td>
</tr>
<tr>
<td>VeriDrone</td>
<td>C</td>
<td>Handwritten</td>
<td>Coq</td>
<td>Safety</td>
</tr>
<tr>
<td>ROSCoq</td>
<td>EXE</td>
<td>Extraction</td>
<td>Coq</td>
<td>Robust Safety?</td>
</tr>
<tr>
<td>HA-SPIRAL</td>
<td>Code</td>
<td>Syntactic</td>
<td>dL</td>
<td>Safety</td>
</tr>
<tr>
<td>VeriPhy (dL)</td>
<td>EXE</td>
<td>Semantic</td>
<td>dL</td>
<td>Safety</td>
</tr>
<tr>
<td>VeriPhy (CdGL)</td>
<td>IR</td>
<td>Refinement</td>
<td>CdGL</td>
<td>Reach-Avoid</td>
</tr>
<tr>
<td>Approach</td>
<td>Output</td>
<td>Modeling Ease</td>
<td>Proof Ease</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------</td>
<td>-----------------</td>
<td>---------------</td>
<td></td>
</tr>
<tr>
<td>Reachflow</td>
<td>Commands</td>
<td>Hardcoded</td>
<td>Automated</td>
<td></td>
</tr>
<tr>
<td>Drona</td>
<td>Motion Plan</td>
<td>Hardcoded</td>
<td>Automated</td>
<td></td>
</tr>
<tr>
<td>Althoff, et. al</td>
<td>Commands</td>
<td>Hardcoded</td>
<td>Automated</td>
<td></td>
</tr>
<tr>
<td>LTLMoP+ TuLiP</td>
<td>Commands</td>
<td>Abstracted</td>
<td>Automated</td>
<td></td>
</tr>
<tr>
<td>VeriDrone</td>
<td>Handwritten</td>
<td>Manual</td>
<td>Coq Tactic</td>
<td></td>
</tr>
<tr>
<td>ROSCoq</td>
<td>Reals</td>
<td>Manual</td>
<td>Coq Tactic</td>
<td></td>
</tr>
<tr>
<td>HA-SPIRAL</td>
<td>Source Code</td>
<td>Hybrid, Subtle</td>
<td>Tactic</td>
<td></td>
</tr>
<tr>
<td>VeriPhy (dL)</td>
<td>No controls.</td>
<td>Hybrid, Subtle</td>
<td>Tactic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Compiled, but poor arithmetic</td>
<td>Hybrid, Abstract</td>
<td>Structured</td>
<td></td>
</tr>
<tr>
<td>VeriPhy (CdGL)</td>
<td>Controls! good arithmetic, but interpreted</td>
<td>Hybrid, Abstract</td>
<td>Structured</td>
<td></td>
</tr>
</tbody>
</table>
Issues of Positional Proof are Self-Evident

```plaintext
implyR(1); andL(-1); andL(-2); loop({'INVARIAINT`}, 1) <(
/* Base case: */ QE,
/* Postcondition: */ QE,
/* Inductive step: */
    auto; <(/* Accelerate: */
        dC({'INVARIAINT`}, 1) <(dI(1), nil);
        dW(1);
        hide(-12=={'FORMULA`}); ...; hide(-17=={'FORMULA`});
        andR(1) <(auto, hide(-11); cut({'FORMULA`}) <(QE
        , orL(-5) <(allL2R(-17); hide(-2); QE, QE)))>
    , /* Brake: */ ... 
)
) ... ~150 lines
```
Design Principles for Kaisar

1. Proof-by-annotation as default
2. Persistent, named contexts with lexical scope
3. Fact selection by positive mention as default
4. Mobile, named references to expressions across states
## More Kaisar Results

<table>
<thead>
<tr>
<th>Model Name (Bellerophon)</th>
<th>Lines</th>
<th>Model</th>
<th>Proof</th>
<th>Assump</th>
<th>Same + Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLDI-DC</td>
<td>15</td>
<td>13</td>
<td>3</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>PLDI-AS</td>
<td>42</td>
<td>15</td>
<td>27</td>
<td>9</td>
<td>9 + 33</td>
</tr>
<tr>
<td>PLDI-TAC</td>
<td>39</td>
<td>15</td>
<td>24</td>
<td>5</td>
<td>19 + 20</td>
</tr>
<tr>
<td>PLDI-RA</td>
<td>28</td>
<td>19</td>
<td>9</td>
<td>0</td>
<td>10 + 18</td>
</tr>
<tr>
<td>PLDI-RAD</td>
<td>29</td>
<td>20</td>
<td>9</td>
<td>0</td>
<td>27 + 2</td>
</tr>
<tr>
<td>IJRR</td>
<td>88</td>
<td>36</td>
<td>52</td>
<td>3</td>
<td>N/A</td>
</tr>
<tr>
<td>RA-L</td>
<td>294</td>
<td>67</td>
<td>227</td>
<td>97</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model Name (Kaisar)</th>
<th>Lines</th>
<th>Model</th>
<th>Proof</th>
<th>Assump</th>
<th>Same + Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLDI-DC</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>PLDI-AS</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>0</td>
<td>6 + 4</td>
</tr>
<tr>
<td>PLDI-TAC</td>
<td>9</td>
<td>7</td>
<td>4</td>
<td>0</td>
<td>7 + 2</td>
</tr>
<tr>
<td>PLDI-RA</td>
<td>15</td>
<td>11</td>
<td>6</td>
<td>0</td>
<td>2 + 13</td>
</tr>
<tr>
<td>PLDI-RAD</td>
<td>16</td>
<td>12</td>
<td>6</td>
<td>0</td>
<td>11 + 5</td>
</tr>
<tr>
<td>IJRR</td>
<td>62</td>
<td>31</td>
<td>31</td>
<td>12</td>
<td>N/A</td>
</tr>
<tr>
<td>RA-L</td>
<td>491</td>
<td>133</td>
<td>372</td>
<td>138</td>
<td>N/A</td>
</tr>
</tbody>
</table>
## Simulation Results (Constructive)

<table>
<thead>
<tr>
<th></th>
<th>Avg. Speed (m/s)</th>
<th></th>
<th>Ctrl Fail.</th>
<th>Plant Fail.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>World</td>
<td>BB</td>
<td>PD1</td>
<td>PD2</td>
</tr>
<tr>
<td>Rect</td>
<td></td>
<td>7.29</td>
<td>6.41</td>
<td>7.19</td>
</tr>
<tr>
<td>Turns</td>
<td></td>
<td>7.24</td>
<td>6.75</td>
<td>7.09</td>
</tr>
<tr>
<td>Clover</td>
<td></td>
<td>15.4</td>
<td>25.3</td>
<td>25.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>58 / 64</td>
<td>58 / 64</td>
<td>58 / 64</td>
</tr>
</tbody>
</table>
Classical VeriPhy

KeYmaera X (Modified)

Isabelle/HOL:
  Arithmetic Translation
  Soundness theorem
  Proof term checker (Generated)

HOL4:
  CakeML compiler
  Specifications of Environment

Constructive VeriPhy

Kaisar (New)
Classical Hybrid Systems Cross-Checked in Isabelle

- Soundness theorem of $d\mathcal{L}$ uniform substitution calculus is formalized in $d\mathcal{L}$
- Axioms, sequent-calculus rules of KeYmaera X formalized, packaged as proof term checker
- Soundness theorem applies to proof checker
- Proof checker was extracted, tested against $\approx 100,000$-step proof terms
- Formalization of integer interval execution included
- **Formalization limitations:** Arithmetic assumed, no ghost in of ODE systems, limited division
- **Implementation limitation:** Proof export is unmaintainable, thus not deployed

```lemmas
lemma proof_sound::"pt_result pt=Some rule ==> QEs_hold pt ==> sound rule"
```
Approach: Model CPS as 2-Player Hybrid Game
Approach: Victory Means Safety and Liveness

Liveness *(Eventually)*: Approach Stop Sign
Safety *(Always)*: To Left of Stop Sign
Refinement Connects Games to Systems

proof carProof = <previous slide> end
let carGame ::= { <game model> };

proves carProof "[carGame](formula)"

- Answers: What game have I proved?
- Impact: Reduce games to systems
\[\text{if}(\neg(d > 0)) \]
\[\text{throw MonitorFailure}();\]
\[\text{while}(d \geq L \cdot \epsilon)\{\]
\[\text{v} := L; \; \neg(0 \leq v \wedge v \leq L);\]
\[t := 0;\]
\[\{ d' = -v, t' = 1 \& t \leq \epsilon \]
\[\neg((d - (\epsilon - t) \cdot v \geq 0)\]
\[\neg((d \leq \text{old}(d) - t \cdot v));\]
\[\neg(d \geq 0);\]
\[\text{if}(\neg(t > \frac{1}{2} \cdot \epsilon));\]
\[\neg(d \geq 0 \wedge d \leq \text{old}(d) - L \cdot \epsilon);\]
\} \]
\[\neg(d \geq 0 \wedge d \leq L \cdot \epsilon);\]