End-to-End Verification for Cyber-Physical Systems

Rose 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

6 Logician: Foundations + Formal Guarantees

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
Classical VeriPhy: Formal, Low-Level, Real-World Proof
Classical VeriPhy: Formal, Low-Level, Real-World Proof
Constructive VeriPhy: CdGL Supports Controllers
逻辑师
§2 - dŁ 正式化
§4 - 构造性 GL
§5 - 构造性 dGL
§6 - 完善
§3 - 编译证明

逻辑用于混合系统

逻辑师

$dŁ$

桂系

构造师

监视器
- §3

控制
- §8

调试
- §2

逻辑用户（§7）
可读性？ 可维护性？
模型错误
行数

传说
- 讨论
- 讨论简要
- 未讨论
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

Modeling Practicality
- Other
- L
- T
- VD
- RC
- HA
- Coq?

Proof Practicality
- Coq
- HA

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.
Related Work: End-to-End Verification (Selected)

<table>
<thead>
<tr>
<th>Least</th>
<th>End-to-Endness</th>
<th>Most</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Other</td>
<td>VD</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Least</th>
<th>Modeling Practicality</th>
<th>Most</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Other</td>
<td>L</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Least</th>
<th>Proof Practicality</th>
<th>Most</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Coq</td>
<td>HA</td>
</tr>
</tbody>
</table>

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

Distance Assumption
Choose speed (velocity)
Speed range
Ordinary Diff. Eq. (ODE)
Upper Time Bound
Lower Time Bound
We Control Repetition
Driving is Game Between Angel and Demon

Distance Assumption

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

\(\times\)

<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 := \ast}^d)</td>
<td>Some (x)</td>
</tr>
<tr>
<td>(x := \ast)</td>
<td>Any (x)</td>
</tr>
<tr>
<td>({x' = f &amp; \psi})</td>
<td>Evolve (x' = f) during (\psi)</td>
</tr>
<tr>
<td>(\alpha \cup \beta)</td>
<td>Play (\alpha) or (\beta)</td>
</tr>
<tr>
<td>(\alpha; \beta)</td>
<td>Play (\alpha) then (\beta)</td>
</tr>
<tr>
<td>(\alpha \times \alpha^*)</td>
<td>Repeat (\alpha)</td>
</tr>
</tbody>
</table>
Driving is Game Between Angel and Demon

**Distance Assumption**

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

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

\( t := 0; \)

\( \{ d' = -v, t' = 1 \land t \leq \epsilon \}; \)

\( ?(t > \frac{1}{2} \cdot \epsilon); \)

\( \times \)

**Choose speed (\( = \) velocity)\)**

<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 := \ast}^d)</td>
<td>Some (x)</td>
</tr>
<tr>
<td>(x := \ast)</td>
<td>Any (x)</td>
</tr>
<tr>
<td>({x' = f \land \psi})</td>
<td>Evolve (x' = f)</td>
</tr>
<tr>
<td></td>
<td>during (\psi)</td>
</tr>
<tr>
<td>(\alpha \cup \beta)</td>
<td>Play (\alpha) or (\beta)</td>
</tr>
<tr>
<td>(\alpha; \beta)</td>
<td>Play (\alpha) then (\beta)</td>
</tr>
<tr>
<td>(\alpha \times \alpha^*)</td>
<td>Repeat (\alpha)</td>
</tr>
</tbody>
</table>
Driving is Game Between Angel and Demon

Distance Assumption

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

Choose speed (= velocity)

Speed range

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

Distance Assumption

Choose speed (= velocity)

Speed range

Ordinary Diff. Eq. (ODE)

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)

Upper Time Bound

Lower Time Bound

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

Statement

Meaning

\(?\phi\)
Assume \(\phi\)

\(!\phi\)
Prove \(\phi\)

\(x := f\)
Assign \(x := f\)

\(x := *\)
Some \(x\)

\(x' = f \land \psi\)
Any \(x\)

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)

Upper Time Bound

Lower Time Bound

We Control Repetition

<table>
<thead>
<tr>
<th>Statement</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>?ϕ</td>
<td>Assume ϕ</td>
</tr>
<tr>
<td>!ϕ</td>
<td>Prove ϕ</td>
</tr>
<tr>
<td>x := f</td>
<td>Assign x := f</td>
</tr>
<tr>
<td>x := *</td>
<td>Some x</td>
</tr>
<tr>
<td>{x' = f &amp; ψ}</td>
<td>Any x</td>
</tr>
<tr>
<td>x := f</td>
<td>Evolve x' = f</td>
</tr>
<tr>
<td>α ∪ β</td>
<td>during ψ</td>
</tr>
<tr>
<td>α; β</td>
<td>Play α or β</td>
</tr>
<tr>
<td>α× α*</td>
<td>Play α then β</td>
</tr>
<tr>
<td></td>
<td>Repeat α</td>
</tr>
</tbody>
</table>

? (d ≥ 0);

{v := *} \[ d; !(0 ≤ v ∧ v ≤ L); \]
t := 0;
\{d' = -v, t' = 1 & t ≤ ε\};
?(t > 1/2 · ε);

× Distance Assumption

Choose speed (= velocity)

Speed range

Ordinary Diff. Eq. (ODE)

Upper Time Bound

Lower Time Bound

We Control Repetition
Want to Prove Safety and Liveness

\(? (d \geq 0); \\
\{ \\
\{ v := \ast \}^d; \; !(0 \leq v \land v \leq L); \\
t := 0; \\
\{ d' = -v, t' = 1 \& t \leq \epsilon \}; \\
!(d \geq 0);' \\
? (t > \frac{1}{2} \cdot \epsilon);' \\
\}\times \\
!(d \geq 0 \land d \leq \delta); \quad \text{Liveness} \)
This Model is Simplified

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

? (d ≥ 0);
{v := ∗}d; !(0 ≤ v ∧ v ≤ L);
t := 0;
{d′ = −v, t′ = 1 & t ≤ ϵ};
!(d ≥ 0);
?(t > \frac{1}{2} · ϵ);

1D Motion, Static Obstacle

Instant control

This Model is Simplified
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

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

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

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

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

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

\[ [x' = f & \psi] \phi \]
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 ≥ 0);
{
{v := *}^d; !(0 ≤ v ∧ v ≤ L);

  t := 0;

  {d' = -v, t' = 1 & t ≤ \epsilon};

?(t > \frac{1}{2} \cdot \epsilon);

}\times
Proof Outline Approach, Outlined

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

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

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

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

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

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

\(?(d \geq 0)\);
while \((d \geq L \cdot \epsilon)\) {
\{ v := L; !(0 \leq v \land v \leq L) \land t := 0; \}
\{ d' = -v, t' = 1 \land t \leq \epsilon \}
\!(d \geq 0) \land d \leq old(d) - t \cdot v \}\) \proof ..
\} \proof ..
\!(d \geq 0) \land d \leq old(d) - L \cdot \epsilon \}\) \proof ..
} \proof ..
What to Prove?

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

Holds for some \(v\)

Each time safe

End live + safe

\[ \frac{[\alpha ; !\psi] \phi}{[\alpha](\psi \land \phi)} \]
Heart of Proof = Strategy = Control + Monitors

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

\[
\begin{array}{c}
\exists x \phi \\
\\n[: \exists d] \exists x \phi
\end{array}
\]
Heart of Proof = Strategy = Control + Monitors

\(? (d \geq 0) ;
\{
\{v := \ast \}^d \};
!(0 \leq v \wedge 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 \wedge d \leq \delta);

Choose \( v \)

Choose invariant

Choose guard

\([:*^d]\) \\exists \, \phi
\frac{\exists x \, \phi}{\{x := \ast \}^d \phi}

DC

\[ x' = f \& \psi \] \rho
\frac{\[ x' = f \& \psi \& \rho \] \phi}{\[ x' = f \& \psi \] \phi}
Controls include Assignments and Guards

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

\text{Safety} + \text{Progress}

\[ [\star^d]l \quad \frac{[x := f] \phi}{[\{x := \star\}^d] \phi} \]

\[ [\star^d]wh \quad \frac{[\text{while}(\cdots)\{\cdots\}] \phi}{[\{\alpha^*\}^d] \phi} \]
Monitors Include Invariants

?\((d \geq 0)\);
while\((d \geq L \cdot \epsilon)\) {
  \(v := L; !(0 \leq v \land v \leq L)\);
  \(t := 0;\)
  \(\{d' = -v, t' = 1 \land t \leq \epsilon\}
    \land !(d - (\epsilon - t) \cdot v \geq 0)
    \land !(d \leq \text{old}(d) - t \cdot v);\)
  !(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);
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
What are Kaisar’s Goals?

- Reduce learning curve
- Put proofs inline with model for readability
- Minimize coupling for maintainability
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);
Practical Proofs Use Kaisar Features Together

```plaintext
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);
```

Definitions Reduce Coupling

Labels and Names Decouple State Changes

Proofs Are Inline
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>
<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
7. Conclusion
Angel Tests Do Not Need Code

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

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

\(\text{ref} \geq \ \ast\)
\(?\phi \geq ?\phi\)
Monitor Failures Are Reported

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

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

\[
\text{ref} \geq \ast \\
\neg \phi \geq \phi
\]

\[
\text{throw} \geq \ast \\
\text{throw Err}(); \geq \alpha
\]

\[
\text{if} \geq \phi \rightarrow \alpha \geq \gamma \quad \beta \geq \gamma \\
\{\text{if}(\phi)\alpha \text{ else } \beta\} \geq \gamma
\]
ODEs are Monitored as Physical State Changes

\[
\begin{align*}
    &\text{if } (\neg (d > 0)) \\
    &\quad \text{throw MonitorFailure();} \\
    &\text{while}(d \geq L \cdot \epsilon) \{ \\
    &\quad v \leftarrow L; \\
    &\quad tOld \leftarrow 0; \\
    &\quad dOld \leftarrow d; d \leftarrow \ast; t \leftarrow \ast; \\
    &\quad \text{if } (\neg (t \leq \epsilon \land d - (\epsilon - t) \cdot v \geq 0 \\
     \land d \leq dOld - t \cdot v)) \\
    &\quad \quad \text{throw MonitorFailure();} \\
    &\quad \text{if } (\neg (t > \frac{1}{2} \cdot \epsilon)) \\
    &\quad \quad \text{throw MonitorFailure();}
    &\} 
\end{align*}
\]
if (¬(d > 0))
    throw MonitorFailure();
while (d ≥ L · ϵ) {
    v := L;
    tOld := 0;
    dOld := d; d := *; t := *;
    if (¬(t ≤ ϵ ∧ d − (ϵ − t) · v ≥ 0
        ∧ d ≤ dOld − t · v))
        throw MonitorFailure();
    if (¬(t > 1/2 · ϵ))
        throw MonitorFailure();
}
Demon is Reusable Across Strategies

\[ i f (\neg (d > 0)) \]
\[ t h r o w \ MonitorFailure(); \]
\[ w h i l e (d \geq L \cdot \epsilon) \{ \]
\[ v := L; \]
\[ tO ld := 0; \]
\[ dO ld := d; d := \ast; t := \ast; \]
\[ i f (\neg (t \leq \epsilon \wedge d - (\epsilon - t) \cdot v \geq 0 \wedge d \leq dO ld - t \cdot v)) \]
\[ t h r o w \ MonitorFailure(); \]
\[ i f (\neg (t > \frac{1}{2} \cdot \epsilon)) \]
\[ t h r o w \ MonitorFailure(); \]
\[ \} \]

\[ ?(d \geq 0); \]
\[ w h i l e (\text{Guard}) \{ \]
\[ v := \text{Speed}(v, d); !(0 \leq v \wedge v \leq L); \]
\[ t := 0; \]
\[ \{d' = -v, t' = 1 \& t \leq \epsilon \wedge (\text{Invariant})\}; \]
\[ !(d \geq 0); \]
\[ ?(t > \frac{1}{2} \cdot \epsilon); \]
\[ !(d \geq 0 \wedge d \leq old(d) - \text{Metric}); \]
\[ \} \]
\[ !(d \geq 0 \wedge d \leq L \cdot \epsilon); \]
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());
}

“Free” code:
• Control (monitor)
• Plant monitor
• Fallback
• Numerics
Robot Reaches Goal

![Graph showing a line with time on the x-axis and distance on the y-axis, indicating the robot reaching its goal over time. The graph is labeled with 'Obstacle Static'.]
Receding Obstacle Safe, Not Live
Robot Catches Liveness Failures

![Graph showing distance vs. time with points labeled P1, P2, and P3.](image)

- **Obstacle Static**
- **Obstacle Receding**
Approach Scales to Complex 2D Driving (RA-L Model)
Approach Scales to Complex 2D Driving (RA-L Model)

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

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

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

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

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

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

(dŁ: machine-checked; CdGL: on paper)

Theorem (Sandboxing)

*High-level sandbox soundly implements model.* (dŁ only)

Theorem (Compilation)

*Compiled code soundly implements discrete program.* (dŁ only)

Theorem (End-to-End)

*Compiled program is safe when model assumptions hold, and raises an alarm when they do not.* (dŁ 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., $(A \sim \gamma)$. 
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 $\mathcal{A}$ be a proof of ($[\alpha] \phi$) and let $\gamma$ be the reification of $\mathcal{A}$, i.e., ($\mathcal{A} \rightsquigarrow \gamma$).

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

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

Logician | Engineer | Logic-User
---|---|---
Classical VeriPhy | formal | sound monitor program | less prone, less labor
Constructive VeriPhy | paper | monitors, controls | least prone, less labor

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."

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

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

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

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:
```c
@ensure(n == 2*old(n))
void dbl(int& n) { n = 2*n; }
```

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

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

Generalizing + Combining Concepts

Labeled Proofs: Explore past, future, hypothetical states

- `theLabel:` assigns label to location
- `e@label` evaluates e at label
- `e@label(args)` 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

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

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

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

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

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

End-to-Endness

- **Other**
- **VD**
- **RC**
- **V2**
- **V1**

Modeling Ease

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

Proof Ease

- **Coq**
- **V1**
- **HA**
- **V2**

Expressiveness

- **Other**
- **L**
- **T**
- **RC**
- **VD**
- **V1**
- **HA**
- **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>d\mathcal{L}</td>
<td>Safety</td>
</tr>
<tr>
<td>VeriPhy (d\mathcal{L})</td>
<td>EXE</td>
<td>Semantic</td>
<td>d\mathcal{L}</td>
<td>Safety</td>
</tr>
<tr>
<td>VeriPhy (CdGL)</td>
<td>IR</td>
<td>Refinement</td>
<td>CdGL</td>
<td>Reach-Avoid</td>
</tr>
</tbody>
</table>
## Related Work: Practicality

<table>
<thead>
<tr>
<th>Approach</th>
<th>Output</th>
<th>Modeling Ease</th>
<th>Proof Ease</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachflow</td>
<td>Commands</td>
<td>Hardcoded</td>
<td>Automated</td>
</tr>
<tr>
<td>Drona</td>
<td>Motion Plan</td>
<td>Hardcoded</td>
<td>Automated</td>
</tr>
<tr>
<td>Althoff, et. al</td>
<td>Commands</td>
<td>Hardcoded</td>
<td>Automated</td>
</tr>
<tr>
<td>LTLMoP + TuLiP</td>
<td>Commands</td>
<td>Abstracted</td>
<td>Automated</td>
</tr>
<tr>
<td>VeriDrone</td>
<td>Handwritten</td>
<td>Manual</td>
<td>Coq Tactic</td>
</tr>
<tr>
<td>ROSCoq</td>
<td>Reals</td>
<td>Manual</td>
<td>Coq Tactic</td>
</tr>
<tr>
<td>HA-SPIRAL</td>
<td>Source Code</td>
<td>Hybrid, Subtle</td>
<td>Tactic</td>
</tr>
<tr>
<td>VeriPhy (dL)</td>
<td>No controls. Compiled, but poor arithmetic</td>
<td>Hybrid, Subtle</td>
<td>Tactic</td>
</tr>
<tr>
<td>VeriPhy (CdGL)</td>
<td>Controls! good arithmetic, but interpreted</td>
<td>Hybrid, Abstract</td>
<td>Structured</td>
</tr>
</tbody>
</table>
Issues of Positional Proof are Self-Evident

\[
\text{ implyR}(1); \text{ andL}(-1); \text{ andL}(-2); \text{ loop} \left( \{ \text{‘INARIANT’} \}, 1 \right) < ( \\
/* Base case: */ QE, \\
/* Postcondition: */ QE, \\
/* Inductive step: */ \\
\text{ auto}; < (/* Accelerate: */ \\
\text{ dC} \left( \{ \text{‘INARIANT’} \}, 1 \right) < \text{dI}(1), \text{nil}); \\
\text{ dW}(1); \\
\text{ hide}(-12==\{ \text{‘FORMULA’} \}); \ldots; \text{ hide}(-17==\{ \text{‘FORMULA’} \}); \\
\text{ andR}(1) < (\text{auto}, \text{hide}(-11); \text{cut} \left( \{ \text{‘FORMULA’} \} \right) < (QE \\
, \text{orL}(-5) < (\text{allL2R}(-17); \text{hide}(-2); QE, QE))) > \\
, /* Brake: */ \ldots \\
) \\
) \ldots \sim 150 \text{ lines}
\]
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>World</th>
<th>BB</th>
<th>PD1</th>
<th>PD2</th>
<th>PD3</th>
<th>Human</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rect</td>
<td>7.29</td>
<td>6.41</td>
<td>7.19</td>
<td>12.0</td>
<td><strong>16.1</strong></td>
</tr>
<tr>
<td>Turns</td>
<td>7.24</td>
<td>6.75</td>
<td>7.09</td>
<td>9.14</td>
<td><strong>9.58</strong></td>
</tr>
<tr>
<td>Clover</td>
<td>15.4</td>
<td>25.3</td>
<td>25.6</td>
<td>26.2</td>
<td><strong>30.3</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>World</th>
<th>BB</th>
<th>PD1</th>
<th>PD2</th>
<th>PD3</th>
<th>Human</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rect</td>
<td>0%</td>
<td>0.69%</td>
<td><strong>0.79%</strong></td>
<td>6.55%</td>
<td>10.1%</td>
</tr>
<tr>
<td>Turns</td>
<td>2.67%</td>
<td>0.87%</td>
<td><strong>0.07%</strong></td>
<td>9.57%</td>
<td>17.5%</td>
</tr>
<tr>
<td>Clover</td>
<td>7.89%</td>
<td>0.26%</td>
<td><strong>0%</strong></td>
<td>0.21%</td>
<td><strong>0%</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>World</th>
<th>BB</th>
<th>PD1</th>
<th>PD2</th>
<th>PD3</th>
<th>Human</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rect</td>
<td>0%</td>
<td>0.69%</td>
<td><strong>0.79%</strong></td>
<td>6.14%</td>
<td>0.04%</td>
</tr>
<tr>
<td>Turns</td>
<td>2.33%</td>
<td>0.74%</td>
<td><strong>0%</strong></td>
<td>9.46%</td>
<td>13.8%</td>
</tr>
<tr>
<td>Clover</td>
<td>1.17%</td>
<td>0%</td>
<td><strong>0%</strong></td>
<td>0%</td>
<td><strong>0%</strong></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 \(\approx100,000\)-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

lemma proof_sound:"pt_result pt=Some rule ==> QEs_hold pt ==> sound rule"
Approach: Model CPS as 2-Player Hybrid Game

2-Player Games

Angel Player

Demon Opponent
(Optional Untrusted Controller)

Control Software
(Or Sandbox)

Actuators/Hardware

Adversarial Timing

Obstacles + Agents

Physical Environment
Approach: Victory Means Safety and Liveness

- **Liveness** *(Eventually)*: Approach Stop Sign
- **Safety** *(Always)*: To Left of Stop Sign

Angel Player

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

proves carProof "[carGame](formula)"

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