Verified Runtime Validation of Verified Cyber-Physical System Models

Stefan Mitsch  André Platzer

Computer Science Department, Carnegie Mellon University

CPS V&V I&F Workshop, Dec. 12, 2014

For Details, see ModelPlex paper at RV’14
Formal Verification in CPS Development

Real CPS

Proof

Reachability Analysis

Verification Results

safe

Verification results about models only apply if CPS fits to the model

\Rightarrow

Verifiably correct runtime model validation
Formal Verification in CPS Development

Real CPS

Model $\alpha^*$

Control $\alpha_{ctrl}$

$$v := v + 1$$

Plant $\alpha_{plant}$

$$x' = v$$

Proof

Reachability Analysis

Verification Results

safe $\rightarrow$ safe

Verification results about models only apply if CPS fits to the model $\Rightarrow$ verifiably correct runtime model validation.
Formal Verification in CPS Development

Verification results about models only apply if CPS fits to the model

Verifiably correct runtime model validation
Ensures that verification results about models apply to CPS implementations.
Runtime Model Validation

Ensures that verification results about models apply to CPS implementations

Insights

- Verification results transfer to CPS when validating model compliance
- Compliance with model is characterizable in logic
- Compliance formula transformed by proof to executable monitor

model adequate?  control safe?  until next cycle?
Model Validation at Runtime

“Simplex for Models”
Model Validation at Runtime

**Compliance Monitor**  Checks CPS for compliance with model at runtime

**Fallback**  Safe action, executed when monitor is not satisfied

**Challenge**  What conditions do the monitors need to check to be safe?
Challenge: Monitorability

- Our current monitors compare two consecutive states (but: which conditions can we actually observe?)
- Monitoring a history of states: becomes necessary when using temporal operators in safety condition
Challenge: Monitor assumptions if not modeled otherwise

- Intercepts all communication: sensors - controller - actuators
- Untampered values, time-consistent and unit-consistent values
- No execution overhead, no clock drift
- No communication delays (sensor - controller - monitor - actuator)
Challenge: Fallback and Enforceability

- Cannot just disallow unsafe actions, need fallback (redundant)
- Which properties are enforceable with a specific fallback action?
- What is an appropriate fallback to enforce a specific property?
- Enforceability of temporal properties is tricky
Model Validation at Runtime

“Simplex for Models”

Challenge: Fallback assumptions if not modeled otherwise

- Executable unconditionally
- Immediate reaction
Model Validation at Runtime

“Simplex for Models”

Challenge: Platform assumptions

- Reals vs. floats (currently: interval arithmetic)
- Correct compiler and processor
When are two states linked through a run of model $\alpha$?

$\alpha \subseteq \text{Offline}(x^{-}, x^{+}) \in \rho(\alpha)$

Semantical: reachability relation of $\alpha$

$\Rightarrow \langle \alpha(x) \rangle (x^{+} = x)$

Logic ($d_{L}$): starting at $x^{-}$ there exists a run of $\alpha$ to a state where $x^{+} = x$

Real arithmetic: check at runtime (efficient)
When are two states linked through a run of model $\alpha$?

Semantical: $(x^-, x^+) \in \rho(\alpha)$

reachability relation of $\alpha$
Monitor Characterization

When are two states linked through a run of model $\alpha$?

A prior state characterized by $x^-$

Model $\alpha$

A posterior state characterized by $x^+$

Semantical:

$\{x^-, x^+\} \in \rho(\alpha)$

$\Leftrightarrow$ Theorem

Logic ($dL$):

$(x = x^-) \rightarrow \langle \alpha(x) \rangle (x = x^+)$

Starting at $x = x^-$ exists a run of $\alpha$ to a state where $x = x^+$

Offline
When are two states linked through a run of model $\alpha$?

- Offline
  - Semantical: $(x^-, x^+) \in \rho(\alpha)$
  - Logic ($d\mathcal{L}$): $(x = x^-) \rightarrow \langle \alpha(x) \rangle (x = x^+)$
  - Real arithmetic: $F(x^-, x^+)$

- Theorem
- $d\mathcal{L}$ Proof
- Check at runtime (efficient)
When are two states linked through a run of model $\alpha$?

- A prior state characterized by $x^-$
- A posterior state characterized by $x^+$

Offline

Semantical: $(x^-, x^+) \in \rho(\alpha)$

Logic ($d\mathcal{L}$): $(x = x^-) \implies \langle \alpha(x) \rangle (x = x^+)$

Real arithmetic:

$F(x^-, x^+) \uparrow$ d$\mathcal{L}$ proof

Theorem

Starting at $x = x^-$ exists a run of $\alpha$ to a state where $x = x^+$

Check at runtime (efficient)
Challenges

What is missing to ensure that proofs apply to real CPS?

- Monitorability, fallback and enforceability, implementation
- Synthesis
- Model quality, model adaptation

\[ i - 1 \xrightarrow{\text{Model } \alpha} i \xrightarrow{\text{ctrl}} \text{plant} \xrightarrow{\text{Prediction Monitor}} i + 1 \]

Model Monitor
- model adequate?

Controller Monitor
- control safe?

Prediction Monitor
- until next cycle?
Synthesis Challenges

- Proof calculus of $\mathcal{dL}$ executes models symbolically

Proof attempt:

$$(x = x^-) \rightarrow \langle \alpha(x) \rangle (x = x^+)$$
Synthesis Challenges

- Proof calculus of $d\mathcal{L}$ executes models symbolically

Proof attempt:

$\alpha \vdash (x = x^-) \rightarrow \langle \text{climb} \cup \text{descend} \rangle (x = x^+)$

$\langle U \rangle \frac{\langle \text{climb} \rangle \phi \lor \langle \text{descend} \rangle \phi}{\langle \text{climb} \cup \text{descend} \rangle \phi}$

Stefan Mitsch, André Platzer—Verified Runtime Validation of Verified Cyber-Physical System Models
Proof calculus of dŁ executes models symbolically

**Synthesis Challenges**

- Proof calculus of dŁ executes models symbolically

![Diagram](attachment:image.png)

- Model $\alpha$
  - $i - 1 \rightarrow$ climb
  - $i \rightarrow$ descend

- Proof attempt:
  - $(x = x^-) \rightarrow \langle\text{climb} \cup \text{descend}\rangle (x = x^+)$
  - $\langle\text{climb}\rangle (x = x^+) \lor \langle\text{descend}\rangle (x = x^+)$

- Monitor: The subgoals that cannot be proved express all the conditions on the relations of variables imposed by the model.
Synthesis Challenges

- Proof calculus of \( d\mathcal{L} \) executes models symbolically

**Diagram:**
- Prior state: \( x^- \)
- Posterior state: \( x^+ \)
- 

**Formula:**
- \( F_1(x^-, x^+) \)
- \( F_2(x^-, x^+) \)
- \( (x = x^-) \rightarrow \langle \text{climb} \cup \text{descend} \rangle (x = x^+) \)

**Monitor:**
- The subgoals that cannot be proved express all the conditions on the relations of variables imposed by the model → execute at runtime

**Challenges:**
- Nested loops
- Differential equations without polynomial solutions requires cutting in differential (in)variants: need to make sure what is cut in is related to the model
- Proof tactics for full automation
Synthesis Challenges

- Proof calculus of $d\mathcal{L}$ executes models symbolically

\[ \text{Model } \alpha \]

prior state $x^{-}$ \quad $i-1$ \quad \text{climb} \quad \text{descend} \quad i \quad \text{posterior state } x^{+}

\[ (x = x^{-}) \rightarrow \langle \text{climb} \cup \text{descend} \rangle (x = x^{+}) \]

\[ \langle \text{climb} \rangle (x = x^{+}) \quad \lor \quad \langle \text{descend} \rangle (x = x^{+}) \]

\[ F_1(x^{-}, x^{+}) \quad \lor \quad F_2(x^{-}, x^{+}) \]

Monitor: \[ F_1(x^{-}, x^{+}) \lor F_2(x^{-}, x^{+}) \]

- The subgoals that cannot be proved express all the conditions on the relations of variables imposed by the model \( \rightsquigarrow \) execute at runtime
Synthesis Challenges

- Proof calculus of $dL$ executes models symbolically

\[
\text{Model } \alpha \\
\text{prior state } x^\rightarrow_i \rightarrow \text{climb} \leftarrow \text{descend} \rightarrow \text{posterior state } x^\rightarrow_{i+1}
\]

Challenges

- Nested loops
- Differential equations without polynomial solutions require cutting in differential (in)variants: need to make sure what is cut in is related to the model
- Proof tactics for full automation

Monitor: $F_1(x^\rightarrow_i, x^\rightarrow_{i+1}) \lor F_2(x^\rightarrow_i, x^\rightarrow_{i+1})$

- The subgoals that cannot be proved express all the conditions on the relations of variables imposed by the model $\leadsto$ execute at runtime
Useful features

- Analyze and improve model quality
  - Unsatisfiable monitor $\Rightarrow$ model has no runs
  - Find largest satisfiable condition (but: what are core features?)
- Analyze (near) monitor violations
  - In system conditions: bug reports to fix incorrect implementation
  - In environment conditions: counterexamples to adapt inadequate model

For this, we need to

- Collect violated subconditions
- Trace (violated) monitor conditions to model statements
- Distinguish between system and environment in the model
- Measure “degree” of monitor satisfaction/violation
Stefan Mitsch
smitisch@cs.cmu.edu
www.cs.cmu.edu/~smitisch