Online Appendix

For paper "IPL: An Integration Property Language for Multi-Model Cyber-Physical Systems"

Ivan Ruchkin, Joshua Sunshine, Grant Iraci,
Bradley Schmerl, and David Garlan

Institute for Software Research, Carnegie Mellon University

A Appendix

A.1 Extended Acknowledgements

The authors would like to thank Stefano Tonetta, André Platzer, Sagar Chaki, Dionisio de Niz, Bruce Krogh, Ashutosh Pandey, Nathan Fulton, Ada Zhang, and anonymous reviewers for their helpful feedback and pointers to important related work.

This material is based on research sponsored by AFRL and DARPA under agreement number FA8750-16-2-0042. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the AFRL, DARPA or the U.S. Government.

A.2 Syntax

Definition 8 (Rigid term) Rigid terms *of the language are defined as follows:*

$$\text{RTERM} ::= \text{VAR} \mid \text{CONST} \mid \text{ELEM} \mid \text{RTERM.PROP} \mid \text{BFUNC}(\text{RTERM}, \dots \text{RTERM}).$$

Definition 9 (Rigid atom) Rigid atoms *are logical formulas over rigid terms:*

$$\text{RATOM} ::= \text{RATOM} \wedge \text{RATOM} \mid \neg\text{RATOM} \mid \text{RTERM}.$$

Highest-level syntax elements are model instances and formulas:

$$\begin{aligned}\text{MDLINST} &::= \text{MATOM}\{|\text{RTERM}_1 \dots \text{RTERM}_n|\},\\ \text{FORMULA} &::= \forall \text{VAR} : \text{RTERM} \cdot \text{FORMULA} \mid \text{MDLINST} \mid \text{RATOM} \mid\\ &\quad \text{FORMULA} \wedge \text{FORMULA} \mid \neg\text{FORMULA}.\end{aligned}$$

LTL Plugin Syntax A common model for LTL is a labeled transition system M_{ts} (LTS). State variables are interpreted modally, and more complex elements of state (i.e., modally evaluated functions and relations) are exposed as MFUNC. To express temporal formulas, the LTL plugin introduces five syntactic elements as behavioral atoms:

- State variable: STVAR ::= $\underline{v}$.
- State function: MFUNC ::= $\underline{g}(t_1 \dots t_n)$, where $t_1 \dots t_n \in$ MATOM.
- Background function: BFUNC ::= $g(t_1 \dots t_n)$, where $t_1 \dots t_n \in$ MATOM.
- Until: TATOM_u ::= TATOM U TATOM.
- Next: TATOM_x ::= X TATOM.
- Conjunction: TATOM_a := TATOM $\wedge$ TATOM.
- Negation: TATOM_n := $\neg$TATOM.
- A wrapper replaces the MATOM plugin point:

MATOM ::= TATOM ::= RATOM | TERM | TATOM_u | TATOM_x | TATOM_a | TATOM_n.

Other LTL modalities are expressed using the until operator and Boolean constants: $\mathsf{G}\ p \equiv p\ \mathsf{U}\ \bot$; $\mathsf{F}\ p \equiv \top\ \mathsf{U}\ p$.

PCTL Plugin Syntax We plug extended PCTL (its variant used in PRISM) into IPL as the second property language. Common models for PCTL are MDPs and DTMCs. Flexible terms (state variables/functions) are used the same way as for LTL. MATOM is defined differently, using several layered behavioral atoms.

- PATHPROP is a logical expression on a model path using temporal modalities, similar to TATOM in LTL but using bounded until.
- RWDPATHPROP is a logical expression combining co-safe LTL and certain operators to predicate paths on which rewards are calculated.
- PPROP is a boolean check of a probability of a path given by PATHPROP.
- PQUERY is a value query of a probability of a path given by PATHPROP.
- RWDPROP is a boolean check of a reward of a path given by RWDPATHPROP.
- RWDQUERY is a value query of a reward of a path given by RWDPATHPROP.

$$
\begin{aligned}
\text{PATHPROP} ::= {} & \text{RATOM} \mid \text{TERM} \mid \text{PATHPROP} \wedge \text{PATHPROP} \mid \neg\text{PATHPROP} \mid \\
& \text{PATHPROP}\ \mathsf{U}^{\leq \mathsf{k}}\ \text{PATHPROP} \mid \mathsf{X}\ \text{PATHPROP}, \\
\text{RWDPATHPROP} ::= {} & \text{RATOM} \mid \text{TERM} \mid \text{RWDPATHPROP} \wedge \text{RWDPATHPROP} \mid \\
& \text{RWDPATHPROP} \vee \text{RWDPATHPROP} \mid \mathsf{X}\ \text{RWDPATHPROP} \mid \\
& \text{RWDPATHPROP}\ \mathsf{U}^{\leq \mathsf{k}}\ \text{RWDPATHPROP} \mid \mathsf{C}^{\leq \mathsf{k}} \mid \mathsf{I}^{=t} \mid \mathsf{S}, \\
\text{PPROP} ::= {} & P_{o \sim p}[\text{PATHPROP}],\ \text{PQUERY} ::= P_{o=?}[\text{PATHPROP}] \\
\text{RWDPROP} ::= {} & R^{r}_{o \sim v}[\text{RWDPATHPROP}],\ \text{RWDQUERY} ::= R^{r}_{o=?}[\text{RWDPATHPROP}], \\
\text{MATOM} ::= {} & \text{PPROP} \mid \text{PQUERY} \mid \text{RWDPROP} \mid \text{RWDQUERY},
\end{aligned}
$$

where $p \in [0,1], \sim \in \{<, \leq, >, \geq\}, o \in \{max, min, \emptyset\}, t \in \mathbb{N}, \mathrm{k} \in \mathbb{N} \cup \{\inf\}, v \in \mathbb{R}$, and r is a character string (reward structure name).

A.3 Syntactic Examples

To give the intuition about the IPL syntax, we provide examples of acceptable and unacceptable formulas in Tab. 1 with the LTL plugin.

Examples 1–3 illustrate base cases of formulas to be supported: unquantified, quantified, and modal. These formulas trivially preserve modularity, although do not improve expressiveness over existing tools. Examples 4 and 5 show the main use case of IPL — quantification over model instances with modalities.

Example 6 is unacceptable because it violates modularity. The existential quantifier cannot be interpreted by the behavioral model that is necessary to interpret the modality. On the other hand, the modality cannot be interpreted by a view. To check such formulas, views and models would need to be merged, which goes against our design for modularity and tractability. Example 7 also violates modularity: no single model can interpret flexible variables from two different models. To check such formulas, we would need to compose models against the modularity and tractability principles.

#	Example formula	Description	In IPL
(1)	$\overline{f}(1+2) = 3$	View function over a rigid term	Y
(2)	$\forall x : \overline{X} \cdot P(x)$	Quantification over a view-defined set	Y
(3)	$(\mathsf{G}\ \underline{y} = 10)\{\|\|\}$	Model instance over a modality w/ a state var	Y
(4)	$\forall x : \overline{X} \cdot (\mathsf{G}\ P(x, \underline{y}))\{\|\|\}$	Quantification over a model instance	Y
(5)	$\exists x : \overline{X} \cdot (P(x) \rightarrow (\mathsf{F}\ Q(x, \underline{y}))\{\|\|\})$	Quantification over an atom w/ a model instance	Y
(6)	$(\mathsf{G}\ (\exists x : \overline{X} \cdot P(x, \underline{y})))\{\|\|\}$	Model instance over a quantified formula	N
(7)	$(\mathsf{F}\ (\underline{y} = \underline{z}))\{\|\|\}$	Mixed models inside the same instance: M_1 owns $\underline{y}$ and M_2 owns $\underline{z}$	N

Table 1. Examples of acceptable/unacceptable IPL syntax.

A.4 Semantics

Native semantics We interpret IPL formulas in the following context of Γ ($\mathbb{V}$, M_i with I_q^{M}, I^{V}, and I^B), q, $\omega \equiv \langle q_1, q_2, \ldots \rangle$ (provided by $\mathsf{M}.traces$), and μ. We do not allow free variables, so all variables in IPL syntax have to be eventually bound to values. We define satisfaction of native operations in IPL formulas (FORMULA) that also cover all operations in RATOM. Satisfaction of formula f is denoted as $C \models f$, where C is the context in which f is evaluated.

$[\![\text{CONST}]\!]_\Gamma = I^B(\text{CONST}); [\![\text{VAR}]\!]_\mu = \mu(\text{VAR});$

$[\![\text{VFUNC}(r_1, \ldots r_n)]\!]_{\Gamma,\mu} = I^{\mathsf{V}}(\text{VFUNC})([\![r_1]\!]_{\mathbb{V},\mu} \ldots [\![r_n]\!]_{\mathbb{V},\mu}),$
where $r_1 \ldots r_n \in$ RTERM;

$[\![\text{ELEM}]\!]_{\Gamma,q,\mu} = I^{\mathsf{V}}(\text{ELEM}) = \{e\} \subseteq \mathbb{E};$

$[\![\text{RTERM.PROP}]\!]_{\Gamma,q,\mu} = I^{\mathsf{V}}(\text{PROP})([\![\text{RTERM}]\!]_{\mathbb{V},\mu});$

$\Gamma, \omega, \mu \models \neg f$ *iff* $\Gamma, \omega, \mu \not\models f$, where $f \in$ FORMULA or $f \in$ RATOM;

$[\![\text{STVAR}]\!]_{\Gamma,q} = I^{\mathsf{M}}_q(\text{STVAR});$

$[\![\text{MFUNC}(t_1, \ldots t_n)]\!]_{\Gamma,q,\mu} = I^{\mathsf{M}}_q(\text{MFUNC})([\![t_1]\!]_{\Gamma,q,\mu}, \ldots [\![t_n]\!]_{\Gamma,q,\mu}),$
where $t_1 \ldots t_n \in$ TERM;

$\Gamma, \omega, \mu \models f_1 \wedge f_2$ *iff* $\Gamma, \omega, \mu \models f_1$ and $\Gamma, \omega, \mu \models f_2$,
where f_1 and f_2 are either FORMULA or RATOM;

$\Gamma, \omega, \mu \models \forall x : r \cdot f$ *iff* $\Gamma, \omega, \mu' \models f$, where $r \in$ RTERM,
f is either FORMULA or RATOM, $\mu' = \mu \cup \{x \mapsto v\}$ for all v in $[\![r]\!]_{\Gamma,\mu}$;

$\Gamma, \mu \models (a)[|p_1 \ldots p_n|]$ *iff* $\mathbb{V}, \mathsf{M}([\![p_1]\!]_{\mathbb{V},\mu} \ldots [\![p_n]\!]_{\mathbb{V},\mu}), \mu \models a$, where $a \in$ MATOM.

LTL plugin semantics The model is a state transition system M_{ts} (a state set, an action set, a transition function, an initial state, and a state interpretation I^{M}_q to determine valid propositions in state q). It defines traces $\mathsf{M}_{ts}.trcs$. $\omega^{i,j}$ means a substring of ω from element i to element j inclusively.

$$\Gamma, \omega, \mu \models f \quad \textit{iff} \quad \Gamma, q, \mu \models f, \text{ where } q \in \omega^{1,1} \text{ and } f \in \text{TERM};$$
$$\Gamma, \omega, \mu \models \mathsf{X}\ \text{TATOM} \quad \textit{iff} \quad \Gamma, \omega^{2,\infty} \text{ and } \mu \models \text{TATOM};$$
$$\Gamma, \omega, \mu \models \text{TATOM}_1\ \mathsf{U}\ \text{TATOM}_2 \quad \textit{iff}$$
$$\exists i : \mathbb{N} \cdot (\Gamma, \omega^{1,i}, \mu \models \text{TATOM}_1 \wedge \Gamma, \omega^{i+1,\infty}, \mu \models \text{TATOM}_2).$$

We describe the meaning of FORMULA in terms of only Γ by iterating over all traces in the model without any up-front variable mappings:

$$\Gamma \models \text{FORMULA} \quad \textit{iff} \quad \forall \omega : \mathsf{M}_{ts}.trcs \cdot \Gamma, \omega, \emptyset \models \text{FORMULA}.$$

PCTL plugin semantics To evaluate a PCTL formula, we use an MDP M_{mdp} as a behavioral model in Γ (or a DTMC M_{dtmc} if there is no non-determinism). It is characterized by finite state set S, finite action set A, an initial state, probability transition function $P : S \times S \to [0, 1]$, a discount factor $\gamma \in [0, 1]$, reward structures $r_i : S \times S \to \mathbb{R}_{\geq 0}$, and a state interpretation I^{M}_q to determine valid propositions in state q.

Satisfaction of temporal operators in PATHPROP and RWDPATHPROP is characterized for model state q and path ω in the same way as in LTL, with the only difference being the bounded until:

$$\Gamma, \omega, \mu \models f_1 \ \mathsf{U}^{\leq \mathsf{k}} \ f_2 \quad \textit{iff}$$
$$\exists i : \mathbb{N} \cdot (i \leq k \wedge \Gamma, \omega^{i,\infty}, \mu \models f_2) \wedge (\forall j : \mathbb{N} \cdot j < i \rightarrow \Gamma, \omega^{1,j}, \mu \models f_1).$$

Given policy $\pi : S \rightarrow A$, P (one of policies Π) induces a probability measure Pr_q^π over paths $Paths(q)$ starting in state q. In turn Pr_q^π induces a probability function over formulas that determines the probability of taking a path that satisfies formula f from state q: $Prob^\pi(q, f) = Pr_q^\pi\{\omega \in Paths(q) \mid \omega \models f\}$.

We can now define formula satisfaction for PPROP and valuation for PQUERY:

$$\Gamma, q, \mu \models P_{o \sim p}[f] \quad \textit{iff} \quad \mathrm{opt}_{\pi \in \Pi} \ Prob^\pi(q, [\![f]\!]_{\Gamma,\mu}) \sim p,$$
$$[\![P_{o=?}[f]]\!]_{\Gamma,q,\mu} = \mathrm{opt}_{\pi \in \Pi} \ Prob^\pi(q, [\![f]\!]_{\Gamma,\mu})),$$

where $f \in$ PATHPROP, $\sim \in \{<, \leq, >, \geq\}$, $\mathrm{opt}_{\pi \in \Pi}$ stands for $\sup_{\pi \in \Pi}$ if $o \equiv \max$, and $\inf_{\pi \in \Pi}$ if $o \equiv \min$, and no operator if $o \equiv \emptyset$.

Rewards formulas are evaluated analogously:

$$\Gamma, q, \mu \models R_{o \sim p}[f] \text{ iff } \mathrm{opt}_{\pi \in \Pi} \ \mathrm{Exp}^\pi(q, X_{[\![f]\!]_{\Gamma,\mu}}) \sim p,$$
$$[\![R_{o=?p}[f]]\!]_{\Gamma,q,\mu} = \mathrm{opt}_{\pi \in \Pi} \ \mathrm{Exp}^\pi(q, X_{[\![f]\!]_{\Gamma,\mu}}) \sim p,$$

where $X_f : Paths^\pi(q) \rightarrow \mathbb{R}_{\geq 0}$ is a random reward variable for paths that satisfy f (defined canonically for co-safe LTL and special formulas [1]), and Exp^π is its expectation with respect to Pr_q^π; other variables mean the same as above.

A.5 Encoding of the Motivating Integration Case

For the motivating example of the mobile robot, our approach is to connect the two models (M_{po} and M_{pl}) through a power view (V_{po}) and a PCTL property, as shown in Fig. 1.

We exemplify the IPL syntax and semantics by deconstructing the motivating integration property in terms of the IPL syntax below. The formal meaning of this decomposition corresponds to the intuition stated in the integration case.

$$\text{FORMULA} = \forall t_1, t_2, t_3 : \overline{\mathit{Tasks}} \cdot \text{FORMULA}_1, \tag{1}$$
$$\text{FORMULA}_1 = \text{RTERM} \rightarrow \text{MDLINST} = 1,$$
$$\text{RTERM} = t_1.\mathit{type} = t_3.\mathit{type} = \mathrm{STR} \wedge t_2.\mathit{type} = \mathrm{ROT} \wedge$$
$$t_1.\mathit{end} = t_2.\mathit{start} = t_3.\mathit{start} \ \wedge t_1.\mathit{start} \neq t_3.\mathit{end} \wedge \Sigma_{i=1}^3 t_i.\mathit{energy} \leq \overline{\mathit{MaxBat}},$$
$$\text{MDLINST} = \text{PPROP}\{|\underline{\mathit{initloc}} = t_1.\mathit{start}, \underline{\mathit{goal}} = t_3.\mathit{end},$$
$$\underline{\mathit{initbat}} = \Sigma_{i=1}^3 t_i.\mathit{energy} + \overline{\mathit{err\_cons}}|\},$$
$$\text{PPROP} = P_{max=?}[\text{PATHPROP}],$$
$$\text{PATHPROP} = P_{max=?}[(\underline{\mathit{loc}} = t_1.\mathit{start} \ \mathsf{U} \ (\underline{\mathit{loc}} = t_2.\mathit{start} \ \mathsf{U} \ \underline{\mathit{loc}} = t_3.\mathit{end})) \wedge$$
$$(\mathsf{F} \ \underline{\mathit{loc}} = t_2.\mathit{start})].$$

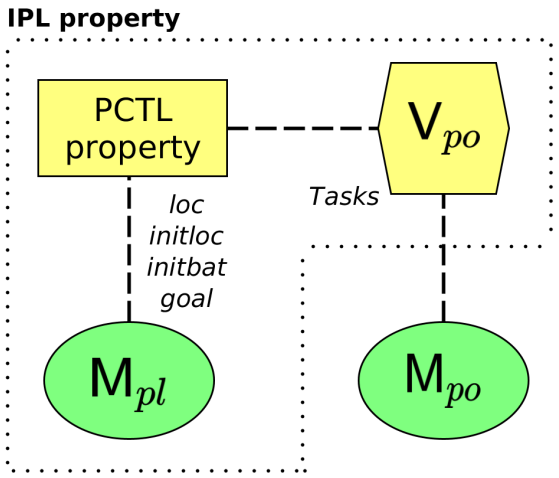

Fig. 1. Model integration for the motivating case. The dots show the scope of the integration property.

Now, let us illustrate the application of the IPL verification algorithm to the above encoding. The formula is already in its PNF. The next step is to remove quantifiers and replace t_1, t_2, and t_3 with their free counterparts t_1^f, t_2^f, t_3^f:

$$t_1.type = t_3.type = \text{STR} \wedge t_2.type = \text{ROT} \wedge \tag{2}$$
$$t_1.end = t_2.start = t_3.start \ \wedge t_1.start \neq t_3.end \wedge \Sigma_{i=1}^{3} t_i.energy \leq \overline{MaxBat} \rightarrow$$
$$P_{max=?}[(\underline{loc} = t_1.start \ \mathsf{U} \ (\underline{loc} = t_2.start \ \mathsf{U} \ \underline{loc} = t_3.end)) \wedge (\mathsf{F} \ \underline{loc} = t_2.start)]$$
$$\{|\underline{initloc} = t_1.start, \underline{goal} = t_3.end, \underline{initbat} = \Sigma_{i=1}^{3} t_i.energy + \overline{err\_cons}|\} = 1.$$

Then MDLINST is abstracted away twice, once with a real-valued function $f(t_1^f, t_2^f, t_3^f)$ and once with a real constant CA:

$$t_1.type = t_3.type = \text{STR} \wedge t_2.type = \text{ROT} \wedge \tag{3}$$
$$t_1.end = t_2.start = t_3.start \ \wedge t_1.start \neq t_3.end \wedge \Sigma_{i=1}^{3} t_i.energy \leq \overline{MaxBat} \rightarrow$$
$$\text{FA}(t_1, t_2, t_3) = 1;$$

$$t_1.type = t_3.type = \text{STR} \wedge t_2.type = \text{ROT} \wedge \tag{4}$$
$$t_1.end = t_2.start = t_3.start \ \wedge t_1.start \neq t_3.end \wedge \Sigma_{i=1}^{3} t_i.energy \leq \overline{MaxBat} \rightarrow$$
$$\text{CA} = 1.$$

Following the abstraction, the saturation process would determine all tuples of free variable values that satisfy the following search formula:

$$t_1.type = t_3.type = \text{STR} \wedge t_2.type = \text{ROT} \wedge \tag{5}$$
$$t_1.end = t_2.start = t_3.start \ \wedge t_1.start \neq t_3.end \wedge \Sigma_{i=1}^{3} t_i.energy \leq \overline{MaxBat} \rightarrow$$
$$\text{FA}(t_1, t_2, t_3) = 1$$
$$\not\Rightarrow$$
$$t_1.type = t_3.type = \text{STR} \wedge t_2.type = \text{ROT} \wedge$$
$$t_1.end = t_2.start = t_3.start \ \wedge t_1.start \neq t_3.end \wedge \Sigma_{i=1}^{3} t_i.energy \leq \overline{MaxBat} \rightarrow$$
$$\text{CA} = 1.$$

For each tuple (t_1, t_2, t_3) satisfying Eq. 5, M_{pl} (initialized according to MDLINST, with the values derived from the tuple) is queried for the interpretation of PATHPROP (as defined in Eq. 1):

$$P_{max=?}[(\underline{loc} = t_1.start \ \mathsf{U} \ (\underline{loc} = t_2.start \ \mathsf{U} \ \underline{loc} = t_3.end)) \wedge (\mathsf{F} \ \underline{loc} = t_2.start)].$$

Finally, the obtained interpretations will be conjoined with the negation of Eq. 3 for the ultimate satisfiability check:

$$\neg \ (\forall t_1, t_2, t_3 : \overline{Tasks} \cdot t_1.type = t_3.type = \text{STR} \wedge t_2.type = \text{ROT} \wedge \tag{6}$$
$$t_1.end = t_2.start = t_3.start \ \wedge t_1.start \neq t_3.end \wedge \Sigma_{i=1}^{3} t_i.energy \leq \overline{MaxBat} \rightarrow$$
$$\text{FA}(t_1, t_2, t_3) = 1).$$

Receiving UNSAT from Eq. 6 would mean that the original formula is valid, while receiving SAT would mean that it is invalid.

A.6 Formula Transformations

Below is a formalization of the *RemQuant* transformation. It is executed until it is not applicable to the input formula.

$$RemQuant \equiv Qx \cdot f \rightarrow f\{x/\hat{x}\}.$$

Transformation $ToPNF$ is formalized as the following rewrite system:

$$ToPNF \equiv \ (Qx \cdot f_1) \wedge f_2 \rightarrow Qx \cdot (f_1 \wedge f_2),$$
$$\neg(Qx \cdot f) \rightarrow \overline{Q}x \cdot \neg f,$$

assuming f_2 does not contain free occurrences of x (otherwise they will be uniquely renamed).

Transformation $ConstAbst$ can be described in the following manner:

$$ConstAbst \equiv \ f\{|p_1 \ldots p_n|\} \rightarrow C,$$

where C is an uninterpreted constant of the same type as f.

Transformation *FuncAbst* can be described in the following manner.

$$FuncAbst \equiv \ f\{|p_1 \dots p_n|\} \rightarrow F(x_1 \dots x_n),$$

where f is a formula and F is an uninterpreted function of the same type as f. Its parameters $x_1 \dots x_n$ are all free variables that occur in formula f and terms $p_1 \dots p_n$.

A.7 Soundness Proof

In this part of the appendix, we provide formal proofs for the theorems that help demonstrate soundness of the IPL verification algorithm.

Theorem 1. *Absence of flexible interpretations that agree with I^F_{sv} and satisfy $\neg f^{FA}$ is necessary and sufficient for validity of f^{FA} on I^F:*

$$\nexists I \cdot I^F_{sv} \subseteq I \wedge I \models \neg f^{FA} \textit{ iff } I^F \models f^{FA}.$$

Proof. Soundness: equivalent transformation of the left side yields $\forall I \cdot I^F_{sv} \subseteq I \rightarrow I \models f^{FA}$. Instantiating I with I^F, we get $I^F_{sv} \subseteq I^F \rightarrow I^F \models f^{FA}$. Since the premise of this implication holds by construction, modus ponens yields $I^F \models f^{FA}$.

Completeness: we need to show that if $\exists I \cdot I^F_{sv} \subseteq I \wedge I \models \neg f^{FA}$, then $I^F \not\models f^{FA}$. Assume, for contradiction, that $I^F \models f^{FA}$.

Instantiating $\exists$, we have interpretation I' that satisfies $\neg f^{FA}$ and agrees with I^F_{sv} (and, therefore, with I^F on sv).

We will construct a variable assignment μ, starting from $\emptyset$, by unwrapping quantifiers in f^{FA}. By f^{FA}_i we mean a formula resulting from removing the first i quantifiers from f^{FA}. That is,

$$f^{FA}_i \equiv Q_{i+1} x_{i+1} : \mathrm{D}_{i+1} \cdot \dots Q_n x_n : \mathrm{D}_n \cdot f^{FA}.$$

Consider the two cases of the outermost quantifier Q_1 in f^{FA}. f^{FA}_1 is the result of removing Q_1 from f^{FA}, with x_1 being a free variable in f^{FA}_1.

If $Q_1 \equiv \forall$, then pick v_1 from $I' \models \neg f^{FA}$ by pushing negation over the universal quantifier and instantiating the resulting existential quantifier with v_1. Then, $I', v_1 \models \neg f^{FA}_1$. We also get $I^F, v_1 \models f^{FA}_1$ by instantiating the universal quantifier in f^{FA} with v_1.

If $Q_1 \equiv \exists$, then pick v_1 from $I^F \models f^{FA}$ by instantiating the existential quantifier, leading to $I^F, v_1 \models f^{FA}_1$. Like the above, by pushing the negation for I' we get $I' \models \forall x_1 : \mathrm{D}_1 \cdot \neg f^{FA}_1$. Instantiating the universal quantifier with v_1, we get $I' \models \neg f^{FA}_1$.

Either way, we add $x_1 \mapsto v_1$ to μ and repeat the above process for the remaining $n-1$ quantifiers, arriving at $f^{FA}_n \equiv \hat{f}^{FA}$, an assignment μ for all variables $x_1 \dots x_n$, and two assertions:

$$I', \mu \models \neg \hat{f}^{FA}, \tag{7}$$

$$I^F, \mu \models \hat{f}^{FA}. \tag{8}$$

Now we will show that $\mu \in sv$. Consider $\hat{f}^{CA}$ that corresponds to $\hat{f}^{FA}$, $\hat{f}^{CA} = ConstAbst(f)$. Let I^{CA} be some interpretation of the constant abstractions in $\hat{f}^{CA}$.

We consider two cases of $\hat{f}^{CA}$ by the principle of excluded middle. If $I^{CA}, \mu \models \hat{f}^{CA}$, then by combining it with Eq. 7 we get $I', I^{CA}, \mu \models \hat{f}^{FA} \not\Leftrightarrow \hat{f}^{CA}$. If $I^{CA}, \mu \models \hat{f}^{CA}$, then by combining it with Eq. 8 we get $I^F, I^{CA}, \mu \models \hat{f}^{FA} \not\Leftrightarrow \hat{f}^{CA}$. Thus, $\mu \in sv$.

Since $\mu \in sv$, I' and I^F agree on valuations of F_i for μ because these are determined by I^F_{sv}. Since interpretation of $\hat{f}^{FA}$ only depends on $F(x_1, \ldots x_n)$ and free variables $x_1 \ldots x_n$ (determined by μ), both interpretations (Eqs. 7 and 8) should agree on the validity of $\hat{f}^{FA}$. Since the semantics of IPL is unambiguous, this leads us to a contradiction.

We conclude that $I^F \models f^{FA}$.

Corollary 1. *Validity of formula f is equivalent to unsatisfiability $I^F_{sv} \models \neg f^{FA}$.*

$$\mathsf{M} \models f \text{ iff } \nexists I \cdot I^F_{sv} \subseteq I \wedge I \models \neg f^{FA}.$$

Proof. By construction of f^{FA} in the algorithm, $\mathsf{M} \models f$ has is semantically equivalent to $I^F \models f^{FA}$. By Thm. 1, the latter is equivalent to $\nexists I \cdot I^F_{sv} \subseteq I \wedge I \models \neg f^{FA}$.

Corollary 2. *The IPL verification algorithm is sound for solving the IPL formula validity problem. The algorithm terminates if (i) satisfiability checking is decidable, (ii) behavioral checking with M is decidable, and (iii) search formula $\hat{f}^{FA} \not\Leftrightarrow \hat{f}^{CA}$ has a finite number of satisfying values for free variables (e.g., when quantification domains D_i are finite).*

Proof. The algorithm equivalently transforms f to its PNF and performs a functional abstraction, which is an equivalent transformation under full interpretation I^F. Soundness follows from the Cor. 1 that shows that the last step of the algorithm is equivalent to the semantic validity of IPL formulas.

Termination of the verification algorithm follows from termination of the search of sv and construction of I^F_{sv}. The search terminates due to decidability of satisfiability checking (premise (i) above) and finiteness of the free variable values to satisfy the formula under check (premise (iii) above). For each μ in sv, construction of I^F_{sv} terminates because behavioral checking with M is decidable (premise (ii) above).

A.8 IPL Implementation

The IPL implementation is based on the Xtext language framework (`https://www.eclipse.org/Xtext`) in the Eclipse IDE, with its sources available online (`https://github.com/bisc/IPL`). Xtext automatically generates an IPL parser, an object model of the constructs, and other supporting software infrastructure from the IPL grammar file. This infrastructure supports view-SMT

translators and verifiers for IPL statements. Development of IPL specifications is done in a modified version of Eclipse bundled with the IPL implementation.

To make the grammar described in Sec. A.2 unambiguous for parsing, we "flattened out" rules regarding logical and background operators. As a result, the implemented syntax is more permissive that the abstract one. However, to preserve the restrictions of the abstract grammar, we implemented typechecking to detect violations. For instance, even though the implemented grammar allows stacking multiple behavioral models, typechecking flags such cases as errors.

As a basis for architectural views, we use the Architecture Analysis and Description Language (AADL, version 2.1) [2]. AADL is an increasingly popular modeling tool for embedded system, featuring an SAE standard designation and multiple extensions. It also fits our assumptions on architectural elements (described in the prerequisites in the main paper): they are statically defined and can have custom properties with fixed values. IPL's implementation relies on OSATE2 (version 2.3.0) [3] — an open-source IDE for AADL. Based on Xtext as well, OSATE2 provides a capability of parsing and instantiating AADL models, which serve as views.

We implemented an AADL-to-SMT translator to convert from architectural to formal views, as per their respective definitions in the main paper. To interface with SMT solvers, we used the SMT-LIB v2.6 [4] syntax to abstract away from specific implementations. We used Z3 (version 4.5.0) [5] and CVC4 (version 4.1.5) [6] as backend solvers.

A.9 Mobile Robot Case Study: Integration Properties

We capture the robot's meaningful behaviors with the following definition:

Definition 10 (Mission) *A* mission *is a finite sequence of commands with contiguous and non-self-intersecting locations. A* power-successful mission *is one that can be completed with a given initial power budget (from 0 to* $\overline{MaxBat}$*).*

Missions differ by the types of available atomic tasks of the robot:

- Forward tasks
- Empty tasks (for missions of variable length with a fixed number of vars)
- Rotation tasks
- Charging tasks
- Other actuation tasks (e.g., turning off a sensor)

To be correctly integrated, M_{po} and M_{pl} should satisfy three properties:

- M_{pl} and M_{po} agree on the map of the location.
- If M_{po} considers a mission power-successful, then M_{pl} should do so.
- If M_{pl} considers a mission power-successful, then M_{po} should do so.

In the rest of this section, we express these properties using one or several IPL formulas. If all the formulas hold, then the models are considered consistent.

Map Consistency. First, let us handle the property of consistency with respect to maps, which means that different models agree on locations present in the map. We start with integration properties of V_{po} and V_{map}.

Property 2 *Any location is reachable.*

"For any location, there are an incoming and outgoing tasks."

$$\forall l : \overline{Locs} \cdot (\exists t_{in}, t_{out} : \overline{Tasks} \cdot l.id = t_{in}.endloc = t_{out}.startloc).$$

Property 3 *Every straight motion task corresponds to an edge in the map.*

"For any straight motion task, there is a pair of locations"

$$\forall t : \overline{Tasks} \cdot t.type = \mathrm{STR} \rightarrow \exists l_1, l_2 : \overline{Locs} \cdot$$

"where the task begins and ends connected by an edge."

$$l_1.id = t.startloc \wedge l_2.id = t.endloc \wedge l_1 \in l_2.edges \wedge l_2 \in l_1.edges.$$

In a similar way, one can assure that V_{map} is fully consistent with V_{po}. Given that the above properties are satisfied, we can turn to consistency between $\mathsf{V}_{po}/\mathsf{V}_{map}$ and M_{pl}.

Property 4 *Every location in the map exists in the MDP.*

"Any location from V_{map} exists in M_{pl}"

$$\forall l : \overline{Locs} \cdot P_{max=?}[\underline{loc} = l.id]\{|\underline{initloc} = l.id, \underline{goal} = l.id, \underline{initbat} = 1|\} = 1.$$

If this trivial property does not pass, it indicates that the set of locations is not consistent. Further, assuming continuous location IDs, one can ensure that M_{pl} does not have more locations than V_{map} by attempting to get to a location with the ID smaller than the minimum (similarly for larger than the maximum):

Property 5 *No reachable locations with IDs smaller than the minimal ID.*

"For any two distinct locations, one starting and one with the smallest ID,"

$$\forall l_{init}, l_{min} : \overline{Locs} \cdot l_{init} \neq l_{min} \wedge (\forall l_o : \overline{Locs} \cdot l_{min}.id \leq l_o.id) \rightarrow$$

"any path in M_{pl} attempting to get the ID of the smallest minus 1,"

$$P_{max=?}[\mathsf{F}\ \underline{loc} = l_{min}.id - 1]$$

"initialized correctly, would fail."

$$\{|\underline{initloc} = l_{init}.id, \underline{goal} = l_{min}.id - 1, \underline{initbat} = \overline{MaxBat}|\} = 0.$$

This property uses a slightly weaker notion of existence (reachability), but it is sufficient for our purposes: if a location is not reachable in the MDP, it would not affect other verification. Finally, we can demonstrate that the edges are consistent by showing two properties below.

Property 6 *Any pair of locations with an edge can be traversed directly.*

"For any task, the behaviors in M_{pl} *going from its start to its end"*

$$\forall t : \overline{Tasks} \cdot t.type = \mathrm{STR} \rightarrow P_{max=?}[\underline{loc} = t.startloc \ \mathsf{U} \ \underline{loc} = t.endloc]$$

"initialized correctly, should succeed if given enough battery."

$$\{|\underline{initloc} = t.startloc, \underline{goal} = t.endloc, \underline{initbat} = t.energy + 1|\} = 1.$$

Property 7 *Any pair of locations without an edge cannot be traversed directly.*

"For any pair of distinct locations without an edge between them,"

$$\forall l_1, l_2 : \overline{Locs} \cdot l_1 \neq l_2 \wedge l_1 \notin l_2.edges \rightarrow$$

"any behavior in M_{pl} *aiming to go between them,"*

$$P_{max=?}[\underline{loc} = l_1.id \ \mathsf{U} \ \underline{loc} = l_2.id]$$

"when initialized correctly, would fail even with a maximum charge."

$$\{|\underline{initloc} = l_1.id, \underline{goal} = l_2.id, \underline{initbat} = \overline{MaxBat}|\} = 0.$$

If the above properties are valid, we can conclude that M_{po} and M_{pl} are consistent with respect to a certain map, thus satisfying Property # 1.

Power success: from M_{po} to M_{pl} Here we highlight a property that checks that if M_{po} considers a mission power-successful, then M_{pl} will as well. Informally, we check that *"any power-successful mission in* M_{po} *with up to four sequential straight motion (*STR*) tasks (the last three possibly empty —* EMP*) will correspond to such executions in* M_{pl} *that visit all sequence points in the correct order will be power successful,"* encoded in IPL below.

Property 8 *If* M_{po} *considers a mission power-successful,* M_{pl} *should do so.*

"Any mission with up to four straight motion tasks"

$$\forall t_1, t_2, t_3, t_4 : \overline{Tasks} \cdot$$

"connected to each other in a sequence"

$$t_1.endloc = t_2.startloc \wedge t_2.endloc = t_3.startloc \wedge t_3.endloc = t_4.startloc \wedge$$

"that is non-empty, can have empty tasks only in the end,"

$$t_1.type \neq \mathrm{EMP} \wedge (t_2.type = \mathrm{EMP} \rightarrow t_3.type = \mathrm{EMP}) \wedge$$
$$(t_3.type = \mathrm{EMP} \rightarrow t_4.type = \mathrm{EMP}) \wedge$$

"contains no self-intersecting tasks"

$$(\nexists i : \overline{Tasks} \cdot (i = t_1 \vee i = t_2 \vee i = t_3 \vee i = t_4) \wedge i.type = \mathrm{STR} \wedge$$
$$((i \neq t_1 \wedge t_1.endloc = i.endloc \wedge t_1.type = \mathrm{STR}) \vee$$
$$(i \neq t_2 \wedge t_2.endloc = i.endloc \wedge t_2.type = \mathrm{STR}) \vee$$
$$(i \neq t_3 \wedge t_3.endloc = i.endloc \wedge t_3.type = \mathrm{STR}) \vee$$
$$(i \neq t_4 \wedge t_4.endloc = i.endloc \wedge t_4.type = \mathrm{STR}))) \wedge$$

"and that is a power-successful mission in M_{po}*"*
$\Sigma_{i=1}^{4} t_i.energy \leq \overline{MaxBat} \rightarrow$
"will correspond to such executions in M_{pl} *that visit all sequence points"*
$P_{max=?}[(\mathsf{F}\ \underline{loc} = t_2.startloc) \wedge (\mathsf{F}\ \underline{loc} = t_3.startloc) \wedge (\mathsf{F}\ \underline{loc} = t_4.startloc) \wedge$
"in the correct order"
$((\underline{loc} = t_1.start)\ \mathsf{U}\ (\underline{loc} = t_2.start\ \mathsf{U}\ (\underline{loc} = t_3.start\ \mathsf{U}\ \underline{loc} = t_4.end)))]$
"and, when initialized correctly, will be power-successful."
$\{|\underline{initloc} = t_1.start, \underline{goal} = t_4.end, \underline{initbat} = \Sigma_{i=1}^{4} t_i.energy + \overline{err\_cons}|\} = 1.$

One quantified variable per task models sequences only up to a certain length. This limitation is acceptable because after a certain length (7-10 steps depending on the map), any mission is self-intersecting. Thus, Prop. 8 encodes a practically relevant property for a specific map. Its *sv* contains 142 vectors, meaning that model checking is called 142 times. The full verification completes in 93 seconds.

Power success: from M_{pl} to M_{po} Now we highlight a property variant for power success in M_{pl} implying power success in M_{po}. This time, we use an existentially quantified power budget that has a risk of succeeding in M_{pl} but failing in M_{po}. The *sv* for this property contains 100 vectors of free variable values. This property relies on monotonicity of power dynamics: given a mission, its success with a certain initial energy leads to its success with larger initial energies. The converse is true about insufficient energies.

Property 9 *If* M_{pl} *considers a mission power-successful,* M_{po} *should do so.*

"For any mission with exactly four straight motion tasks"
$\forall t_1, t_2, t_3, t_4 : \overline{Tasks} \cdot t_1.type = t_2.type = t_3.type = t_4.type = \mathrm{STR} \wedge$
"connected to each other in a sequence"
$t_1.endloc = t_2.startloc \wedge t_2.endloc = t_3.startloc \wedge t_3.endloc = t_4.startloc \wedge$
"without self-intersections"
$distinct(t_1.startloc, t_2.startloc, t_3.startloc, t_4.startloc, t_4.endloc) \rightarrow$
"there exists such an energy budget greater than the energy expected by M_{po}*"*
$(\exists b : \mathbb{N} \cdot b \geq \Sigma_{i=1}^{4} t_i.energy - \overline{err\_mdp} \wedge$
"that if M_{pl}*, going through all the sequence points"*
$P_{max=?}[(\mathsf{F}\ \underline{loc} = t_2.startloc) \wedge (\mathsf{F}\ \underline{loc} = t_3.startloc) \wedge (\mathsf{F}\ \underline{loc} = t_4.startloc) \wedge$
"in the correct order"
$((\underline{loc} = t_1.start)\ \mathsf{U}\ (\underline{loc} = t_2.start\ \mathsf{U}\ (\underline{loc} = t_3.end\ \mathsf{U}\ \underline{loc} = t_4.end)))]$
"and initialized correctly, is power-successful on that budget,"
$\{|\underline{initloc} = t_1.start, \underline{goal} = t_4.end, \underline{initbat} = b|\} = 1 \rightarrow$
"then M_{po} *should also be power-successful that budget."*
$\Sigma_{i=1}^{4} t_i.energy - \overline{err\_cons} < b).$

A.10 Mobile Robot Case Study: Performance Information

We evaluated the performance of an Eclipse-based IPL implementation using variants of the M_{po}-to-M_{pl} property (e.g., Prop. 8). In particular, we executed 24 verification runs by varying the number of mission tasks and the map, and toggling each of the mission features—variable length missions, charging, and rotations.

We observed the following dependent variables: count of *sv*, total time, saturation time, interpretation time, time in SMT, and time in model checking. We did not find IPL's memory demands limiting since at most one external tool was executing at each point (which, however, indicates potential for parallelizing the model checking process). The performance results for the 24 executions are shown in Tab. 2.

The high-level outcomes are: (a) verification times vary from dozens of seconds to over 6 hours. Counts of *sv* vary from dozens to over a thousand; (b) we found that longer missions lead to increase in both saturation and interpretation times, whereas missions with more features primarily affect the saturation process; (c) the model checking times grew linearly with increases in length across feature groups, with little response to increases in features; (d) the saturation times grow substantially with more features, especially when considering rotations due to additional quantified variables and constraints; (e) IPL's mean overhead was 0.74% (stdev 0.78%); (f) IPL's memory demands were not limiting, since at most one external tool ran at a time.

Map	# steps	Variable length ?	Charging ?	Rotation ?	# satvals	Saturation time (s)	Interp. time (s)	overhead (%)	Total time (s)
map0	4	n	n	n	50	4.1	16.1	1.3	20.4
map0	5	n	n	n	40	5.5	22.7	0.9	28.5
map0	6	n	n	n	34	9.7	55.2	0.5	65.2
map0	7	n	n	n	16	7.8	85.9	0.4	94.0
map0	4	y	n	n	142	55.4	37.1	0.6	93.1
map0	5	y	n	n	182	142.6	95.3	0.3	238.5
map0	6	y	n	n	216	234.1	197.5	0.3	432.8
map0	7	y	n	n	232	336.4	446.1	0.2	783.8
map0	4	n	y	n	86	22.7	43.7	0.5	66.8
map0	5	n	y	n	100	37.7	67.0	0.4	105.1
map0	6	n	y	n	108	47.8	116.7	0.3	165.0
map0	7	n	y	n	99	107.9	191.1	0.3	299.9
map0	4	y	y	n	195	71.4	85.0	0.3	156.9
map0	5	y	y	n	295	243.8	171.2	0.2	416.0
map0	6	y	y	n	403	468.9	373.3	0.2	843.5
map0	7	y	y	n	502	949.9	656.1	0.1	1608.0
map0	8	y	y	n	559	1467.7	1407.2	0.1	2876.6
map3b	4	n	n	y	56	213.1	19.7	1.0	235.1
map3b	5	n	n	y	60	315.6	33.1	0.9	352.0
map3b	6	n	n	y	44	450.8	63.6	0.8	518.5
map3b	4	y	n	y	162	2768.1	42.1	0.2	2815.0
map3b	5	y	n	y	222	5692.3	75.5	0.1	5773.5
map3b	6	y	n	y	266	8618.4	168.1	0.1	8793.4
map3b	7	y	n	y	266	10137.3	256.2	0.1	10403.8
map3b	4	y	y	y	440	5663.5	15410.6	0.0	21078.6

Table 2. Full performance results

They were run sequentially on the following platform: Intel Core i7-7600U, Ubuntu 17.04, Eclipse Oxygen 1a, OSATE 2.3.0 (debug mode) [7], Z3 solver 4.5.0 [5], PRISM model checker 4.4.beta [1] with Rabinizer 3.1 [8]. The dataset and analysis are available online[1].

A.11 Second Integration Case: LTL Property for Real-Time System

We are interested in demonstrating customizability of IPL, not only for various logics but also different CPS domains. IPL generalizes our prior work, and its LTL plugin subsumes the analysis contracts language [9] for assumptions/guarantees. To show that, we apply IPL to a pair of CPS models entirely different from the mobile robot case study: a model of thread scheduling and a model of processors, taken from prior work [9]. We instantiate IPL's syntax for both models, briefly explain their semantics, describe an important integration property, and express it in IPL.

Consider an embedded real-time system with periodic threads ($\overline{Threads}$) that need to execute their tasks before the deadlines (*dline*). Each thread should be allocated on a CPU (member of $\overline{CPUs}$) via a binding relation $\overline{CPUBind}$, containing pairs of threads and processors to which they are bound. The goal of modeling is to create a schedulable system with minimal power consumption, which correlates with CPU frequencies (*freq*).

We represent this system using two models. The scheduling model (M_{sch}) is a discrete software model that captures the threads' and the scheduler behaviors. Implemented in Spin in the related work, M_{sch} encapsulates the logic of of the thread scheduling policy and the rules behind preemption, encoded in a relation $\underline{CanPrmpt}$. The scheduling view (V_{sch}) exposes $\overline{Threads}$, *dline*, their periods, and worst-case execution times. The CPU model (M_{cpu}) is a hardware/physical model describing the electrical dynamics in a processor. M_{cpu} encapsulates the relationship between *freq*, maximum frequency (*maxfreq*), voltage, and current, and the algorithms to reduce these variables. The CPU view (V_{cpu}) exposes $\overline{CPUs}$, *freq*, *maxfreq*, and $\overline{CPUBind}$.

M_{sch} and M_{cpu} serve two competing purposes: M_{sch} schedules threads so that they do not miss deadlines (which may result in failures of the whole system), and M_{cpu} reduces frequency of CPU (to make the system more power-efficient). These two models are also not independent: frequency reduction may lead to deadline misses, since threads take longer to compute on CPUs with smaller frequencies. We aim to integrate M_{sch} and M_{cpu} by specifying and verifying a property that expresses absence of conflict between the two models.

In this case, integration relies on an observation from related work [9]: when CPU frequencies are reduced by a frequency scaling algorithm, deadlines are not missed if the scheduler and threads *behave* as deadline-monotonic (not necessarily that the prescribed policy is deadline-monotonic). We omit the assumptions on the frequency scaling algorithm, and focus on the described deadline-monotonicity property.

[1] `https://github.com/bisc/IPLProjects/tree/master/IPLRobotProp/performance-analysis`

Deadline monotonicity depends on CPU frequencies, bindings, and timing behaviors of the scheduler. We approach this task similarly to the mobile robot example: use behavioral semantics of M_{sch} and abstract away the details of M_{cpu} by using V_{cpu}. Thus, the context of this IPL specification is M_{sch}, V_{sch}, and V_{cpu}.

We iterate over all CPUs with reduced frequency and demand that all threads allocated to such CPUs behave deadline-monotonically; that is, they only preempt threads with larger deadlines. We use two layers of quantification wrapped around two rigid terms and a model instance with a temporal atom inside.

Property 10 *All CPUs with reduced frequency behave deadline-monotonically.*

"All CPUs whose frequency was scaled down"

$$\forall c : \overline{CPUs} \cdot \overbrace{c.freq < c.maxfreq}^{\text{RTERM}} \rightarrow$$

"should only bind pairs of threads that"

$$\forall t_1, t_2 : \overline{Threads} \cdot \overbrace{\overline{CPUBind}(t_1, c) \wedge \overline{CPUBind}(t_2, c)}^{\text{RTERM}} \rightarrow$$

"behave deadline-monotonically with respect to each other."

$$\overbrace{(\mathsf{G}\ \underbrace{\underline{CanPrmpt}(t_1, t_2) \rightarrow t_1.dline < t_2.dline}_{\text{TATOM}})\{|\underline{thrdset} = \{t_1, t_2\}, \underline{cpu} = c|\}}^{\text{MDLINST}}.$$

This property can be checked by the IPL verification algorithm. Using V_{sch} and V_{cpu}, the saturation process will find all values of c, t_1, and t_2 satisfying the two instances of RTERM. For these values MDLINST will be behaviorally evaluated on M_{sch}. After obtaining the necessary interpretations of MDLINST, the final satisaction check will be done to determine the property's validity.

From the system perspective, Prop. 10 should be verified every time after CPU frequencies are scaled down. If verification succeeds, we know that deadlines will not be missed, and the power consumption has been minimized. Thus, M_{sch} and M_{cpu} will remain integrated and non-conflicting with each other.

References

1. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic Model Checking. In Bernardo, M., Hillston, J., eds.: Formal Methods for Performance Evaluation. Number 4486 in Lecture Notes in Computer Science. Springer Berlin Heidelberg (2007) 220–270
2. Feiler, P.H., Gluch, D.P.: Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language. 1 edn. Addison-Wesley, Upper Saddle River, NJ (2012)
3. Feiler, P., Greenhouse, A.: OSATE Plugin Guide. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA (2006)
4. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa (2017)

5. De Moura, L., Bjrner, N.: Z3: An Efficient SMT Solver. In: Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. TACAS'08/ETAPS'08, Berlin, Heidelberg, Springer-Verlag (2008) 337–340
6. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanovi'c, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In Gopalakrishnan, G., Qadeer, S., eds.: Proceedings of the 23rd International Conference on Computer Aided Verification (CAV '11). Volume 6806 of Lecture Notes in Computer Science., Springer (July 2011) 171–177
7. Feiler, P., Wrage, L., Delange, J., Siebel, J.: OSATE2 (2015) github.com/osate.
8. Komarkova, Z., Kretinsky, J.: Rabinizer 3: Safraless Translation of LTL to Small Deterministic Automata. In: Automated Technology for Verification and Analysis. Lecture Notes in Computer Science, Springer, Cham (November 2014) 235–241
9. Ruchkin, I., de Niz, D., Chaki, S., Garlan, D.: Contract-based Integration of Cyber-physical Analyses. In: Proc. of the Intl. Conf. on Embedded Software (EMSOFT), New York, NY, USA, ACM (2014) 23:1–23:10