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:

\[ \text{RTERM} ::= \text{VAR} | \text{CONST} | \text{ELEM} | \text{RTERM}.\text{PROP} | \text{BFUNC}(\text{RTERM}, \ldots, \text{RTERM}). \]

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

\[ \text{RATOM} ::= \text{RATOM} \land \text{RATOM} | \neg \text{RATOM} | \text{RTERM}. \]

Highest-level syntax elements are model instances and formulas:

\[ \text{MDLINST} ::= \text{MATOM} \{ | \text{RTERM}_1 \ldots \text{RTERM}_n | \}, \]
\[ \text{FORMULA} ::= \forall \text{VAR} : \text{RTERM} \cdot \text{FORMULA} | \text{MDLINST} | \text{RATOM} | \text{FORMULA} \land \text{FORMULA} | \neg \text{FORMULA}. \]
**LTL Plugin Syntax**  A common model for LTL is a labeled transition system $M_{ls}$ (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: $\text{stvar} ::= v$
- State function: $\text{mfunc} ::= g(t_1 \ldots t_n)$, where $t_1 \ldots t_n \in \text{MATOM}$.
- Background function: $\text{bfunc} ::= g(t_1 \ldots t_n)$, where $t_1 \ldots t_n \in \text{MATOM}$.
- Until: $\text{TATOM}_u ::= \text{TATOM} \cup \text{TATOM}$.
- Next: $\text{TATOM}_x ::= X \text{TATOM}$.
- Conjunction: $\text{TATOM}_a ::= \text{TATOM} \land \text{TATOM}$.
- Negation: $\text{TATOM}_n ::= \neg \text{TATOM}$.
- A wrapper replaces the MATOM plugin point:

$$\text{MATOM} ::= \text{TATOM} ::= \text{RATOM} | \text{TERM} | \text{TATOM}_u | \text{TATOM}_x | \text{TATOM}_a | \text{TATOM}_n.$$  

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

**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.

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

$$\text{PATHPROP} ::= \text{RATOM} | \text{TERM} | \text{PATHPROP} \land \text{PATHPROP} | \neg \text{PATHPROP} | \text{PATHPROP} U^k \text{PATHPROP} | X \text{PATHPROP},$$

$$\text{RWDPATHPROP} ::= \text{RATOM} | \text{TERM} | \text{RWDPATHPROP} \land \text{RWDPATHPROP} | \text{RWDPATHPROP} \lor \text{RWDPATHPROP} | X \text{RWDPATHPROP} | \text{RWDPATHPROP} U^k \text{RWDPATHPROP} | C^{s,t} | t = t | S,$$

$$\text{PPROP} ::= P_{o \sim p} \left[ \text{PATHPROP} \right], \text{PQUERY} ::= P_{o = ?} \left[ \text{PATHPROP} \right],$$

$$\text{RWDPROP} ::= R_{o \sim p}^{s \leq k} \left[ \text{RWDPATHPROP} \right], \text{RWDQUERY} ::= R_{o = ?}^{s \leq k} \left[ \text{RWDPATHPROP} \right],$$

$$\text{MATOM} ::= \text{PPROP} | \text{PQUERY} | \text{RWDPROP} | \text{RWDQUERY},$$

where $p \in [0, 1], \sim \in \{<, \leq, >, \geq\}, o \in \{\text{max}, \text{min}, \emptyset\}, t \in \mathbb{N}, k \in \mathbb{N} \cup \{\text{inf}\}, v \in \mathbb{R}$, 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.

<table>
<thead>
<tr>
<th>#</th>
<th>Example formula</th>
<th>Description</th>
<th>In IPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>( f(1 + 2) = 3 )</td>
<td>View function over a rigid term</td>
<td>Y</td>
</tr>
<tr>
<td>(2)</td>
<td>( \forall x : X \cdot P(x) )</td>
<td>Quantification over a view-defined set</td>
<td>Y</td>
</tr>
<tr>
<td>(3)</td>
<td>( (G y = 10){</td>
<td></td>
<td>} )</td>
</tr>
<tr>
<td>(4)</td>
<td>( \forall x : X \cdot (G P(x, y)){</td>
<td></td>
<td>} )</td>
</tr>
<tr>
<td>(5)</td>
<td>( \exists x : X \cdot (P(x) \rightarrow (F Q(x, y)){</td>
<td></td>
<td>}) )</td>
</tr>
<tr>
<td>(6)</td>
<td>( (G (\exists x : X \cdot P(x, y))){</td>
<td></td>
<td>} )</td>
</tr>
<tr>
<td>(7)</td>
<td>( (F (y = z)){</td>
<td></td>
<td>} )</td>
</tr>
</tbody>
</table>

Table 1. Examples of acceptable/unacceptable IPL syntax.

A.4 Semantics

**Native semantics** We interpret IPL formulas in the following context of \( \Gamma (V, M_i \text{ with } I^M_i, I^V_i, \text{ and } I^B_i), q, \omega \equiv \langle q_1, q_2, \ldots \rangle \) (provided by \( M.traces \)), and \( \mu \). 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 \( \text{(FORMULA)} \) 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.
\[\text{[CONST]}_{\Gamma} = I^B(\text{CONST}); [\text{VAR}]_\mu = \mu(\text{VAR});\]
\[\text{[VFUNC}}(r_1, \ldots, r_n)_{\Gamma, \mu} = I^V(\text{VFUNC})([r_1]_{V, \mu} \ldots [r_n]_{V, \mu});\]
where \(r_1 \ldots r_n \in \text{RTERM};\)
\[\text{[ELEM]}_{\Gamma, q, \mu} = I^V(\text{ELEM}) = \{e\} \subseteq E;\]
\[\text{[RTERM.PROP]}_{\Gamma, q, \mu} = I^V(\text{PROP})([\text{RTERM}]_{V, \mu});\]
\(\Gamma, \omega, \mu \models \neg f\) if and only if \(f \in \text{FORMULA} \lor f \in \text{RATOM};\)
\[\text{[STVAR]}_{\Gamma, q} = I^M_q(\text{STVAR});\]
\[\text{[MFUNC}}(t_1, \ldots, t_n)_{\Gamma, q, \mu} = I^M_q(\text{MFUNC})([t_1]_{\Gamma, q, \mu} \ldots [t_n]_{\Gamma, q, \mu});\]
where \(t_1 \ldots t_n \in \text{TERM};\)
\(\Gamma, \omega, \mu \models f_1 \land f_2\) if \(\Gamma, \omega, \mu \models f_1\) and \(\Gamma, \omega, \mu \models f_2;\)
where \(f_1\) and \(f_2\) are either \text{FORMULA} or \text{RATOM};
\(\Gamma, \omega, \mu \models \forall x : r \cdot f\) if \(\Gamma, \omega, \mu' \models f\) where \(r \in \text{RTERM},\)
f is either \text{FORMULA} or \text{RATOM}, \(\mu' = \mu \cup \{x \mapsto v\}\) for all \(v\) in \([r]_{\Gamma, \mu};\)
\(\Gamma, \mu \models (a)[[p_1 \ldots p_n]]\) if \(\forall \mu, M([p_1]_{V, \mu} \ldots [p_n]_{V, \mu}), \mu \models a,\) where \(a \in \text{TATOM}.\)

**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 \(I^M_\mu\) to determine valid propositions in state \(q\)). It defines traces \(M_{ts}.\text{trcs}\). \(w^{i,j}\) means a substring of \(\omega\) from element \(i\) to element \(j\) inclusively.

\[
\Gamma, \omega, \mu \models f \iff \Gamma, q, \mu \models f, \text{ where } q \in \omega^{1,1} \text{ and } f \in \text{TERM};
\]
\(\Gamma, \omega, \mu \models X \text{TATOM}\) if \(\Gamma, \omega, \mu \models \omega_{2,\infty}\) and \(\mu \models \text{TATOM};\)
\(\Gamma, \omega, \mu \models \text{TATOM}_1 \cup \text{TATOM}_2\) if
\(\exists i : N : (\Gamma, \omega^{1,i}, \mu \models \text{TATOM}_1 \land \Gamma, \omega^{i+1,\infty}, \mu \models \text{TATOM}_2).\)

We describe the meaning of \text{FORMULA} in terms of only \(\Gamma\) by iterating over all traces in the model without any up-front variable mappings:

\(\Gamma \models \text{FORMULA} \iff \forall \omega : M_{ts}.\text{trcs} \cdot \Gamma, \omega, \emptyset \models \text{FORMULA}.\)

**PCTL plugin semantics** To evaluate a PCTL formula, we use an MDP \(M_{mdp}\) as a behavioral model in \(\Gamma\) (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 \(P : S \times S \rightarrow [0,1]\), a discount factor \(\gamma \in [0,1]\), reward structures \(r_1 : S \times S \rightarrow \mathbb{R}_{\geq 0}\), and a state interpretation \(I^M_\mu\) to determine valid propositions in state \(q\).

Satisfaction of temporal operators in \text{PATHPROP} and \text{BWDPATHPROP} is characterized for model state \(q\) and path \(\omega\) in the same way as in LTL, with the only difference being the bounded until:
Given policy $\pi : S \rightarrow A$, $P$ (one of policies $\Pi$) induces a probability measure $Pr^\pi_q$ over paths $Paths(q)$ starting in state $q$. In turn $Pr^\pi_q$ induces a probability function over formulas that determines the probability of taking a path that satisfies formula $f$ from state $q$: $\text{Prob}^\pi(q,f) = Pr^\pi_q(\omega \in Paths(q) \mid \omega \models f)$.

We can now define formula satisfaction for $\text{PPROP}$ and valuation for $\text{PQUERY}$:

$\Gamma, q, \mu \models P_{o \sim p}[f]$ iff $\text{opt}_{\pi \in \Pi} \text{Prob}^\pi(q, [f]_{\Gamma, \mu}) \sim p,$

$[P_{o \sim p}[f]]_{\Gamma, q, \mu} = \text{opt}_{\pi \in \Pi} \text{Prob}^\pi(q, [f]_{\Gamma, \mu}),$

where $f \in \text{PATHPROP}$, $\sim \in \{<, \leq, >, \geq\}$, $\text{opt}_{\pi \in \Pi}$ 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:

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

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

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 $[\Pi]$), and $\text{Exp}^\pi$ is its expectation with respect to $Pr^\pi_q$; other variables mean the same as above.

### A.5 Encoding of the Motivating Integration Case

The models and the view from the motivating example of the mobile robot are shown in Fig. 1. We exemplify the IPL syntax and semantics by deconstructing the motivating integration property in terms of the IPL syntax. The meaning of Equation (1) follows the intuition of the integration case.

**FORMULA** = $\forall t_1, t_2, t_3 : Tasks \cdot \text{FORMULA}_A,$

**FORMULA** = $\text{RTERM} \rightarrow \text{MDLINST} = 1,$

**RTERM** = $t_1.\text{type} = t_3.\text{type} = \text{STR} \land t_2.\text{type} = \text{ROT} \land$

\[
\begin{align*}
&\text{t}_1.\text{end} = t_2.\text{start} = t_3.\text{start} \land t_1.\text{start} \neq t_3.\text{end} \land \Sigma_{i=1}^{3} t_i.\text{energy} \leq \text{MaxBat}, \\
&\text{MDLINST} = \text{PPROP} \{ ||\text{initloc} = t_1.\text{start}, \text{goal} = t_3.\text{end} || \}
\end{align*}
\]

\[
\begin{align*}
&\underbrace{\text{initbat} = \Sigma_{i=1}^{3} t_i.\text{energy} + \text{initcons}}_{\text{PPROP} = P_{\max = ?}[\text{PATHPROP}]}, \\
&\text{PATHPROP} = (F \text{ loc} = t_2.\text{startloc}) \land (F \text{ loc} = t_3.\text{startloc}) \land \\
&( (\text{loc} = t_1.\text{start}) \cup (\text{loc} = t_2.\text{start}) \cup (\text{loc} = t_3.\text{start} \cup \text{loc} = t_3.\text{end})).
\end{align*}
\]

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 would
be to remove quantifiers and replace $t_1, t_2, t_3$ with their free counterparts
$t^f_1, t^f_2, t^f_3$. Then MDLINST would be abstracted away twice, once with a real-valued
function $f(t^f_1, t^f_2, t^f_3)$ and once with a real constant $C$. The saturation process
would determine all tuples of free variable values that satisfy the search formula.
For each such tuple, the interpretation of MDLINST would be collected from
$M_{pl}$ initialized with appropriate values derived from the tuples. Finally, this
interpretation will be added to the final satisfiability check of the formula’s
negation. Receiving UNSAT from that check would mean that the formula is
valid, andd receiving SAT would mean that it is invalid.

\[ \text{RemQuant} \equiv Qx \cdot f \rightarrow f[\hat{x}] \]

Transformation ToPNF is formalized as the following rewrite system:
\[ \text{ToPNF} \equiv (Qx \cdot f_1) \land f_2 \rightarrow Qx \cdot (f_1 \land f_2), \]
\[ \neg (Qx \cdot f) \rightarrow \neg f, \]
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:
\[ \text{ConstAbst} \equiv f[p_1 \ldots p_n] \rightarrow C, \]
where $C$ is an uninterpreted constant of the same type as $f$. 

---

**Fig. 1.** Models and a view for the motivating case. The dots show the property scope.

**A.6 Formula Transformations**

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

\[ \text{RemQuant} \equiv Qx \cdot f \rightarrow f[\hat{x}] \]

Transformation ToPNF is formalized as the following rewrite system:
\[ \text{ToPNF} \equiv (Qx \cdot f_1) \land f_2 \rightarrow Qx \cdot (f_1 \land f_2), \]
\[ \neg (Qx \cdot f) \rightarrow \neg f, \]
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:
\[ \text{ConstAbst} \equiv f[p_1 \ldots p_n] \rightarrow C, \]
where $C$ is an uninterpreted constant of the same type as $f$. 

Transformation $\text{FuncAbst}$ can be described in the following manner.

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

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

### 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 $I_{sv}^F$ and satisfy $\neg f^{FA}$ is necessary and sufficient for validity of $f^{FA}$ on $I^F$:

$$\exists I \cdot I_{sv}^F \subseteq I \land I \models \neg f^{FA} \iff I^F \models f^{FA}.$$  

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

*Completeness:* we need to show that if $\exists I \cdot I_{sv}^F \subseteq I \land I \models \neg f^{FA}$, 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 $I_{sv}^F$ (and, therefore, with $I^F$ on $sv$).

We will construct a variable assignment $\mu$, starting from $\emptyset$, by unwrapping quantifiers in $f^{FA}$.

By $f_i^{FA}$ we mean a formula resulting from removing the first $i$ quantifiers from $f^{FA}$. That is,

$$f_i^{FA} \equiv Q_{i+1} x_{i+1} : D_{i+1} \ldots Q_n x_n : D_n \cdot f^{FA}.$$  

Consider the two cases of the outermost quantifier $Q_1$ in $f^{FA}$. $f_1^{FA}$ is the result of removing $Q_1$ from $f^{FA}$, with $x_1$ being a free variable in $f_1^{FA}$.

If $Q_1 \equiv \forall$, then pick $v_1$ from $I^F \models f^{FA}$ by pushing negation over the universal quantifier and instantiating the resulting existential quantifier with $v_1$. Then, $I', v_1 \models \neg f_1^{FA}$. We also get $I^F, v_1 \models f_1^{FA}$ 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 $I^F, v_1 \models f_1^{FA}$. Like the above, by pushing the negation for $I'$ we get $I' \models \forall x_1 : D_1 \cdot \neg f_1^{FA}$. Instantiating the universal quantifier with $v_1$, we get $I' \models \neg f_1^{FA}$.

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

\begin{align*}
I', \mu & \models \neg f^{FA}, \quad (2) \\
I^F, \mu & \models f^{FA}. \quad (3)
\end{align*}
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. 2 we get \( I', I^{CA}, \mu \models \hat{f}^{FA} \iff \hat{f}^{CA} \). If \( I^{CA}, \mu \not\models \hat{f}^{CA} \), then by combining it with Eq. 3 we get \( I^{FA}, I^{CA}, \mu \models \hat{f}^{FA} \iff \hat{f}^{CA} \).

Thus, \( \mu \in sv \).

Since \( \mu \in sv \), \( I' \) and \( I^{FA} \) agree on valuations of \( F_i \) for \( \mu \) because these are determined by \( I^{FA} \). 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 \( \mu \)), both interpretations (Eqs. 2 and 3) 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 \( I^{FA} \models \neg f^{FA} \).

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

**Proof.** By construction of \( f^{FA} \) in the algorithm, \( M \models f \) has is semantically equivalent to \( I^{FA} \models f^{FA} \). By Thm. 1, the latter is equivalent to \( \exists I \cdot I^{sv} \subseteq I \land I \models \neg f^{FA} \).

**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 \( \hat{f}^{FA} \iff \hat{f}^{CA} \) 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 \( I^{FA} \). 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 \( I^{FA} \). 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 \( \mu \) in \( sv \), construction of \( I^{FA} \) 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 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_po and M_pl 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.”

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

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

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

\[
\forall t : \text{Tasks} \cdot t.type = \text{STR} \rightarrow \exists l_1, l_2 : \text{Locs}.
\]

“where the task begins and ends connected by an edge.”

\[
l_1.id = t.startloc \land l_2.id = t.endloc \land l_1 \in l_2.edges \land l_2 \in l_1.edges.
\]

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 \( V_{po}/V_{map} \) and \( M_{pl} \).

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

“For any location from \( V_{map} \) exists in \( M_{pl} \)”

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

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,”

\[
\forall l_{init}, l_{min} : \text{Locs} \cdot l_{init} \neq l_{min} \land (\forall l_o : \text{Locs} \cdot l_{min}.id \leq l_o.id) \rightarrow
\]

“any path in \( M_{pl} \) attempting to get the ID of the smallest minus 1,”

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

“initialized correctly, would fail.”

\[
\{\text{initloc} = l_{init}.id, \text{goal} = l_{min}.id - 1, \text{initbat} = \text{MaxBat}\} = 0.
\]

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”

\[ \forall t : \text{Tasks} \cdot t.type = \text{STR} \rightarrow P_{max=?}[loc = t.startloc \cup loc = t.endloc] \]

initialized correctly, should succeed if given enough battery.”

\{ |initloc = t.startloc, goal = t.endloc, 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,”

\[ \forall l_1, l_2 : \text{Locs} \cdot l_1 \neq l_2 \land l_1 \notin l_2.\text{edges} \rightarrow \]

“any behavior in $M_{pl}$ aiming to go between them,”

\[ P_{max=?}[loc = l_1.id \cup loc = l_2.id] \]

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

\{ |initloc = l_1.id, goal = l_2.id, initbat = 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”

\[ \forall t_1, t_2, t_3, t_4 : \text{Tasks} \]

“connected to each other in a sequence”

\[ t_1.endloc = t_2.startloc \land t_2.endloc = t_3.startloc \land t_3.endloc = t_4.startloc \land \]

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

\[ t_1.type \neq \text{EMP} \land (t_2.type = \text{EMP} \rightarrow t_3.type = \text{EMP}) \land \]

\[ (t_3.type = \text{EMP} \rightarrow t_4.type = \text{EMP}) \land \]

“contains no self-intersecting tasks”

\[ (\exists i : \text{Tasks} \cdot (i = t_1 \lor i = t_2 \lor i = t_3 \lor i = t_4) \land i.type = \text{STR} \land \]

\[ ((i \neq t_1 \land i.endloc = i.endloc \land i.type = \text{STR}) \lor \]

\[ (i \neq t_2 \land i.endloc = i.endloc \land t_2.type = \text{STR}) \lor \]

\[ (i \neq t_3 \land i.endloc = i.endloc \land t_3.type = \text{STR}) \lor \]

\[ (i \neq t_4 \land i.endloc = i.endloc \land t_4.type = \text{STR})) \land \]

\]
“and that is a power-successful mission in $M_{po}$”

$\Sigma_{i=1}^4 t_i.\text{energy} \leq \text{MaxBat} \rightarrow$

“will correspond to such executions in $M_{pl}$ that visit all sequence points”

$P_{\text{max}=?}[\langle \text{F loc} = t_2.\text{startloc} \rangle \land \langle \text{F loc} = t_3.\text{startloc} \rangle \land \langle \text{F loc} = t_4.\text{startloc} \rangle \land$

“in the correct order”

$\langle \langle \text{loc} = t_1.\text{start} \rangle \cup \langle \text{loc} = t_2.\text{start} \cup \langle \text{loc} = t_3.\text{start} \cup \text{loc} = t_4.\text{end} \rangle \rangle]$

“and, when initialized correctly, will be power-successful.”

\{\langle \text{initloc} = t_1.\text{start}, \text{goal} = t_4.\text{end}, \text{initbat} = \Sigma_{i=1}^4 t_i.\text{energy} + \text{errcons} \rangle \} = 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”

$\forall t_1,t_2,t_3,t_4: \text{Tasks} \cdot t_1.\text{type} = t_2.\text{type} = t_3.\text{type} = t_4.\text{type} = \text{STR} \land$

“connected to each other in a sequence”

$\text{t_1.endloc} = t_2.\text{startloc} \land \text{t_2.endloc} = t_3.\text{startloc} \land \text{t_3.endloc} = t_4.\text{startloc} \land$

“without self-intersections”

$\text{distinct(t_1.startloc,t_2.startloc,t_3.startloc,t_4.startloc,t_4.endloc)} \rightarrow$

“there exists such an energy budget greater than the energy expected by $M_{po}$”

$(\exists b : \mathbb{N} \cdot b \geq \Sigma_{i=1}^4 t_i.\text{energy} - \text{errmdp}) \land$

“that if $M_{pl}$, going through all the sequence points”

$P_{\text{max}=?}[\langle \text{F loc} = t_2.\text{startloc} \rangle \land \langle \text{F loc} = t_3.\text{startloc} \rangle \land \langle \text{F loc} = t_4.\text{startloc} \rangle \land$

“in the correct order”

$\langle \langle \text{loc} = t_1.\text{start} \rangle \cup \langle \text{loc} = t_2.\text{start} \cup \langle \text{loc} = t_3.\text{end} \cup \text{loc} = t_4.\text{end} \rangle \rangle]$

“and initialized correctly, is power-successful on that budget,”

$\{\langle \text{initloc} = t_1.\text{start}, \text{goal} = t_4.\text{end}, \text{initbat} = b \rangle \} = 1 \rightarrow$

“then $M_{po}$ should also be power-successful on that budget.”

$\Sigma_{i=1}^4 t_i.\text{energy} - \text{errcons} < b$.
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. 3). 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.

<table>
<thead>
<tr>
<th>Map</th>
<th>Steps</th>
<th>Variable length</th>
<th>Charging</th>
<th>Rotation</th>
<th># Satvals</th>
<th>Saturation time (s)</th>
<th>Interpretation time (s)</th>
<th>Overhead (%)</th>
<th>Total time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>map0</td>
<td>4</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>50</td>
<td>4.1</td>
<td>16.1</td>
<td>20.4</td>
</tr>
<tr>
<td>map0</td>
<td>5</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>40</td>
<td>5.5</td>
<td>22.7</td>
<td>28.5</td>
</tr>
<tr>
<td>map0</td>
<td>6</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>34</td>
<td>9.7</td>
<td>55.2</td>
<td>65.2</td>
</tr>
<tr>
<td>map0</td>
<td>7</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>16</td>
<td>7.8</td>
<td>85.9</td>
<td>94.0</td>
</tr>
<tr>
<td>map0</td>
<td>4</td>
<td>y</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>142</td>
<td>55.4</td>
<td>37.1</td>
<td>93.1</td>
</tr>
<tr>
<td>map0</td>
<td>5</td>
<td>y</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>182</td>
<td>142.6</td>
<td>95.3</td>
<td>238.5</td>
</tr>
<tr>
<td>map0</td>
<td>6</td>
<td>y</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>216</td>
<td>234.1</td>
<td>197.5</td>
<td>432.8</td>
</tr>
<tr>
<td>map0</td>
<td>7</td>
<td>y</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>232</td>
<td>336.4</td>
<td>446.1</td>
<td>783.8</td>
</tr>
<tr>
<td>map0</td>
<td>4</td>
<td>n</td>
<td>n</td>
<td>y</td>
<td>n</td>
<td>86</td>
<td>22.7</td>
<td>43.7</td>
<td>66.8</td>
</tr>
<tr>
<td>map0</td>
<td>5</td>
<td>n</td>
<td>n</td>
<td>y</td>
<td>n</td>
<td>100</td>
<td>37.7</td>
<td>67.0</td>
<td>105.1</td>
</tr>
<tr>
<td>map0</td>
<td>6</td>
<td>n</td>
<td>n</td>
<td>y</td>
<td>n</td>
<td>108</td>
<td>47.8</td>
<td>116.7</td>
<td>165.0</td>
</tr>
<tr>
<td>map0</td>
<td>7</td>
<td>n</td>
<td>n</td>
<td>y</td>
<td>n</td>
<td>99</td>
<td>107.9</td>
<td>191.1</td>
<td>299.9</td>
</tr>
<tr>
<td>map0</td>
<td>4</td>
<td>y</td>
<td>y</td>
<td>n</td>
<td>n</td>
<td>195</td>
<td>71.4</td>
<td>85.0</td>
<td>156.9</td>
</tr>
<tr>
<td>map0</td>
<td>5</td>
<td>y</td>
<td>y</td>
<td>n</td>
<td>n</td>
<td>295</td>
<td>243.8</td>
<td>171.2</td>
<td>416.0</td>
</tr>
<tr>
<td>map0</td>
<td>6</td>
<td>y</td>
<td>y</td>
<td>n</td>
<td>n</td>
<td>403</td>
<td>468.9</td>
<td>373.3</td>
<td>843.5</td>
</tr>
<tr>
<td>map0</td>
<td>7</td>
<td>y</td>
<td>y</td>
<td>n</td>
<td>n</td>
<td>502</td>
<td>949.9</td>
<td>656.1</td>
<td>1608.0</td>
</tr>
<tr>
<td>map0</td>
<td>8</td>
<td>y</td>
<td>y</td>
<td>n</td>
<td>n</td>
<td>559</td>
<td>1467.7</td>
<td>1407.2</td>
<td>2876.6</td>
</tr>
<tr>
<td>map3b</td>
<td>4</td>
<td>n</td>
<td>n</td>
<td>y</td>
<td>n</td>
<td>56</td>
<td>213.1</td>
<td>19.1</td>
<td>235.1</td>
</tr>
<tr>
<td>map3b</td>
<td>5</td>
<td>n</td>
<td>n</td>
<td>y</td>
<td>n</td>
<td>60</td>
<td>315.6</td>
<td>33.1</td>
<td>352.0</td>
</tr>
<tr>
<td>map3b</td>
<td>6</td>
<td>n</td>
<td>n</td>
<td>y</td>
<td>n</td>
<td>44</td>
<td>450.8</td>
<td>63.6</td>
<td>518.5</td>
</tr>
<tr>
<td>map3b</td>
<td>4</td>
<td>y</td>
<td>n</td>
<td>y</td>
<td>n</td>
<td>162</td>
<td>2768.1</td>
<td>42.1</td>
<td>2815.0</td>
</tr>
<tr>
<td>map3b</td>
<td>5</td>
<td>y</td>
<td>n</td>
<td>y</td>
<td>n</td>
<td>222</td>
<td>5692.3</td>
<td>75.5</td>
<td>5773.8</td>
</tr>
<tr>
<td>map3b</td>
<td>6</td>
<td>y</td>
<td>n</td>
<td>y</td>
<td>n</td>
<td>266</td>
<td>8618.4</td>
<td>168.1</td>
<td>8793.4</td>
</tr>
<tr>
<td>map3b</td>
<td>7</td>
<td>y</td>
<td>n</td>
<td>y</td>
<td>n</td>
<td>266</td>
<td>10137.3</td>
<td>256.2</td>
<td>10403.8</td>
</tr>
<tr>
<td>map3b</td>
<td>4</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>440</td>
<td>5663.5</td>
<td>15410.6</td>
<td>21078.6</td>
</tr>
</tbody>
</table>

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) [1], Z3 solver 4.5.0 [2], PRISM model checker 4.4.beta [3] with Rabinizer 3.1 [8]. The dataset and analysis are available online.

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 \((\text{Threads})\) that need to execute their tasks before the deadlines \((\text{dline})\). Each thread should be allocated on a CPU (member of \(\text{CPUs}\)) via a binding relation \(\text{CPUBind}\), 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 \((\text{freq})\).

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

\(M_{\text{sch}}\) and \(M_{\text{cpu}}\) serve two competing purposes: \(M_{\text{sch}}\) schedules threads so that they do not miss deadlines (which may result in failures of the whole system), and \(M_{\text{cpu}}\) 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 \(M_{\text{sch}}\) and \(M_{\text{cpu}}\) 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 \(\text{behave}\) 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 pre-empt 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”

$$\forall c: \text{CPUs} \cdot c.freq < c.\text{maxfreq} \rightarrow$$

“should only bind pairs of threads that”

$$\forall t_1, t_2: \text{Threads} \cdot \text{CPUBind}(t_1, c) \land \text{CPUBind}(t_2, c) \rightarrow$$

“behave deadline-monotonically with respect to each other.”

$$\text{mdlinst} \left( G \text{CanPrmp}(t_1, t_2) \rightarrow t_1.dline < t_2.dline \right) \text{tatom} \left\{ \text{thrdset} = \{ t_1, t_2 \}, \text{cpu} = c \right\}.$$ 

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 satisfaction 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**