Contract-Based Integration of Cyber-Physical Analyses

Ivan Ruchkin, Dionisio De Niz, Sagar Chaki, David Garlan. Carnegie Mellon University, Pittsburgh, PA, USA.

Problem

CPS engineering combines diverse model-based *analyses* from various engineering domains. Differences in domain abstractions lead to integration issues:

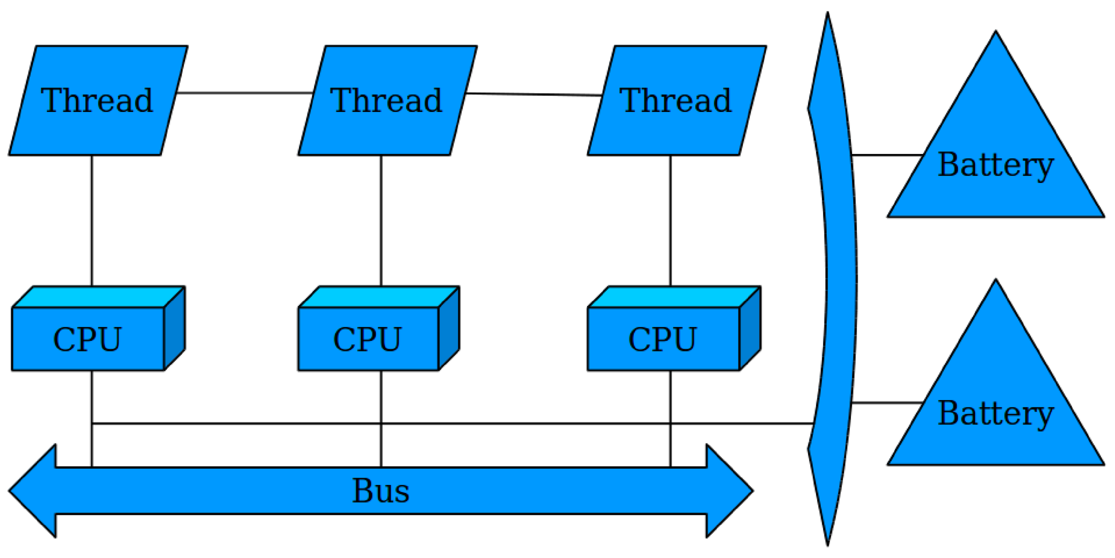
- If an assumption of an analysis are violated by another, the outputs of the former may be invalid.
- Specification of such implicit assumptions and detection of their violation is left to human designers, who are often unable to cope with complexity.
- Analysis integration problems discovered late in development lead to expensive changes to the system.

Hence the research question:

- How to specify analysis compositions and verify their correctness?

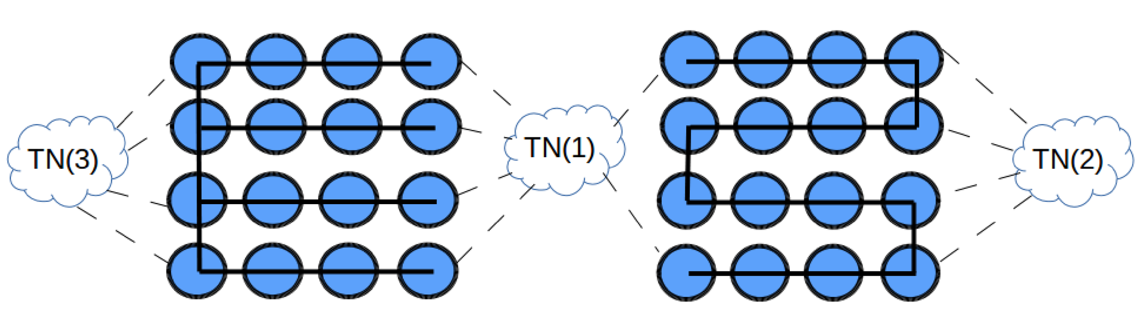
Example System

Consider an autonomous aircraft as an example CPS. It operates data with different classes of security, from normal to top secret (ThSecCl). Periodic threads (T) execute on several processors (C). The aircraft is powered with multi-cell reconfigurable batteries (B). The system's architecture shown below is specified in AADL.



A battery has a matrix of cells, and each cell has a current level of charge. A battery scheduler determines parallel and sequential connections between groups of cells in order to satisfy voltage and current output requirements.

Thermally, each cell exchanges heat with its neighboring cells (*thermal neighbors,* TN) through an electrical connector, affecting the risk of a *thermal runaway*.



Verification Domains

A verification domain $\sigma = (\mathcal{A}, \mathcal{S}, \mathcal{R}, \mathcal{T}, [\![\,]\!]_\sigma)$ formalizes domain-specific constructs for several related analyses.

- $\mathcal{A}$ — a set of sorts, comprised of system elements and standard sorts. E.g., integers $\mathbb{Z}$, threads T, or scheduling policies SchedPol.
- $\mathcal{S}$ — a set of static functions that encode design-time properties. E.g., thread period Per, thread-to-CPU binding CPUBind, and system-wide Voltage.
- $\mathcal{R}$ — a set of runtime functions that encode dynamic properties. E.g., preemption relation canPrmpt(t_1, t_2) and number of cells in a battery b with i thermal neighbors TN(b, i).
- $\mathcal{T}$ — execution semantics of σ — a set of sequences of assignments to $\mathcal{R}$. We use Promela programs to implement the semantics.
- $[\![\,]\!]_\sigma$ — a domain interpretation of $\mathcal{A}$, $\mathcal{S}$, and $\mathcal{T}$. E.g., $[\![\text{SchedPol}]\!]_\sigma$ = {RMS, DMS, EDF}.

Formally, an AADL architectural model **m** is an interpretation $[\![\,]\!]_m$ of $\mathcal{A}$, $\mathcal{S}$, and $\mathcal{T}$. E.g., $[\![T]\!]_m$ = { SensorSample, Ctrl_1, Ctrl_2 }, $[\![\text{CPUBind}]\!]_m$ = { (Ctrl_1, CPU_1), (Ctrl_2, CPU_2), …) }.

$[\![\,]\!]_\sigma \cup [\![\,]\!]_m$ form a full interpretation of $\mathcal{A}$, $\mathcal{S}$, $\mathcal{R}$, and $\mathcal{T}$.

Analysis Contracts

Each analysis is assigned a *contract* — a tuple (I, O, A, G).

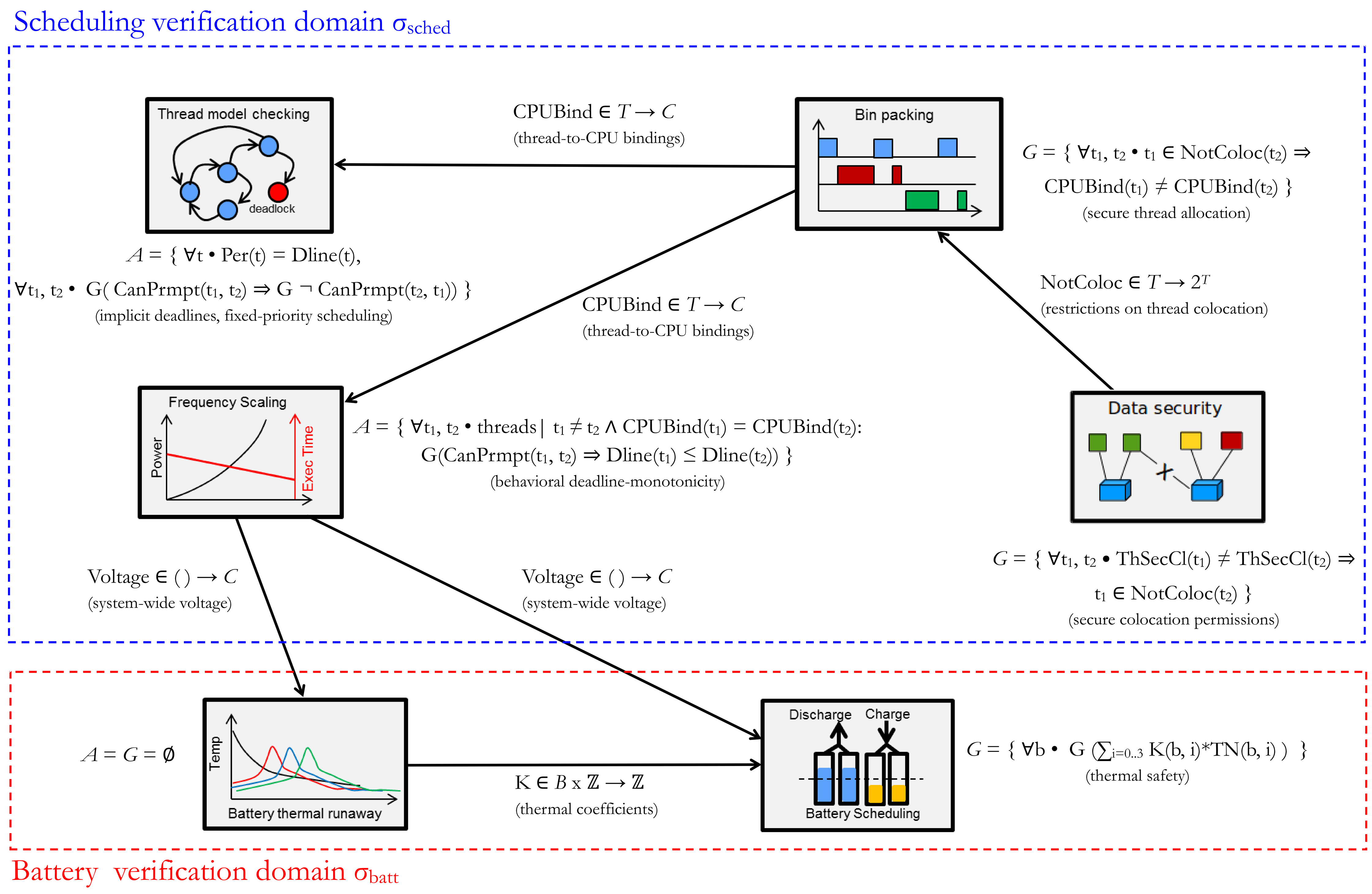
- Inputs $I \subseteq \mathcal{A} \cup \mathcal{S}$ declare elements that the analysis reads.
- Outputs $O \subseteq \mathcal{A} \cup \mathcal{S}$ declare elements that the analysis writes.
- Assumptions $A \subseteq \mathcal{F}_\sigma$ are logical statements that must be satisfied by every input model to the analysis: $\mathbf{m} \vDash A$.
- Guarantees $G \subseteq \mathcal{F}_\sigma$ are logical statements that must be satisfied by every output model of the analysis: $\mathbf{m} \vDash G$.

Assumption and guarantee formulas have the following syntax:

$$\mathcal{F}_\sigma ::= \forall\, v_1 \ldots v_j \bullet \varphi \mid \exists\, v_1 \ldots v_j \bullet \varphi \mid \forall\, v_1 \ldots v_j \bullet \varphi : \psi \mid \exists\, v_1 \ldots v_j \bullet \varphi : \psi,$$

where φ is a predicate logic formula over $\mathcal{A} \cup \mathcal{S}$, ψ is an LTL formula over $\mathcal{A} \cup \mathcal{S} \cup \mathcal{R}$.

Example Analyses



Analysis Ordering

Correct execution of analyses requires satisfaction of all input-output dependencies for each analysis. Formally, contract C_i depends on contract C_j if $C_i.I \cap C_j.O \neq \emptyset$.

Am ordering <$C_1 \ldots C_n$> of contracts is sound if and only if predecessors are not dependent on successors:

$$\forall i \in [1, n] \bullet \forall j \in [1, i) \bullet C_j.I \cap C_i.O = \emptyset.$$

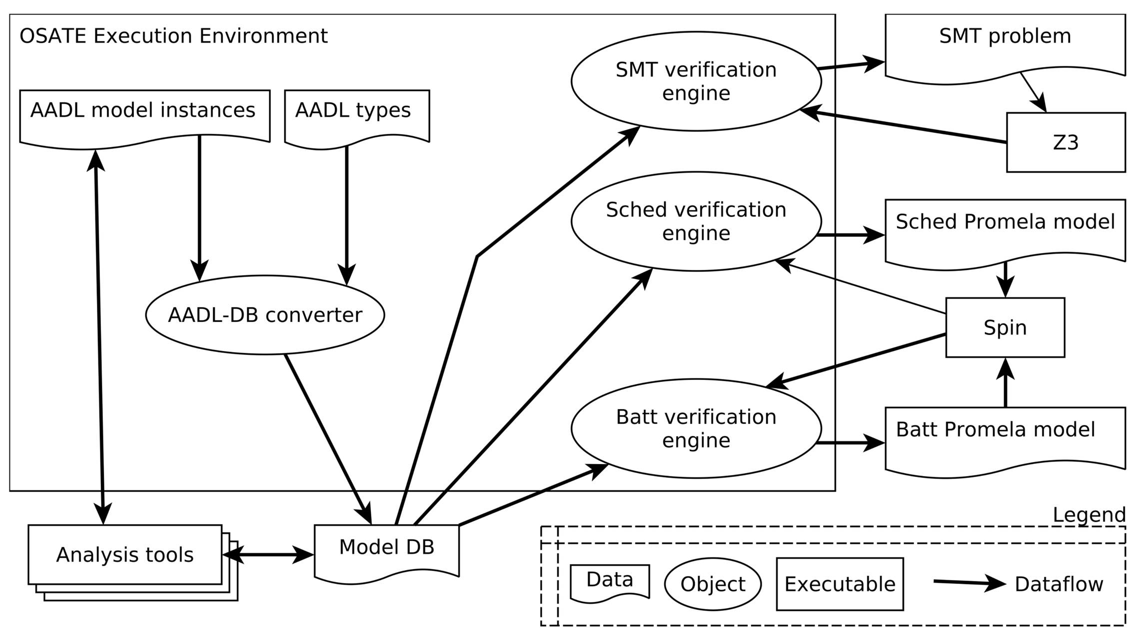
Consider a graph with vertices being contracts and edges being contract dependencies. There exists a sound ordering of contracts if and only if the graph is not cyclic. If it is not cyclic, any topological ordering is sound.

Contract Verification

The goal of contract verification is to decide $\mathbf{m} \vDash \mathcal{F}_\sigma$.

For purely first-order formulas that contain only φ, we decide satisfiability via SMT solving. An SMT program is generated based on $\mathcal{A}$ and $\mathcal{S}$ mentioned in φ, and an SMT solver is invoked on $\neg\varphi$ (or φ for existential quantification). A universally (existentially) quantified contract is satisfied if and only if UNSAT (SAT) is returned.

For formulas combining predicate formula φ and LTL formula ψ, we first generate an SMT program for φ and find all valuations of $v_1 \ldots v_j$ that satisfy φ. For each such valuation we call Spin on a Promela program that implements $\mathcal{T}$ for $\mathbf{m}$ in the domain of ψ. Formula ψ is transformed into an LTL property specification in Promela. A universally (existentially) quantified contract is satisfied if and only if the LTL property holds for all (at least one) valuations. The architecture of our verification tool is shown below:



Experimental Results

Effectiveness: we have been able to detect analysis integration errors and verify their absence for each analysis in the example.

Scalability: the results of scalability experiments with our implementations of $\mathcal{T}$ are shown in the tables below.

$\mathcal{T}_{\text{sched}}$:

Threads	(R/D)MS time*	EDF time*
3	0.01	0.01
4	0.01	0.52
5	0.07	33.4
6	0.37	2290.0
7	2.18	memlim
8	12.4	memlim
9	71.2	memlim
10	421	memlim
11	memlim	memlim

$\mathcal{T}_{\text{batt}}$:

Cells	FGURR time*	FGWRR time*	GPWRR time*
9	0.13	0.15	0.15
12	0.61	2.34	3.94
16	44	31.4	127
20	1060	619	memlim
25	memlim	memlim	memlim

* All times are in seconds.

 This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Carnegie Mellon University
institute for SOFTWARE RESEARCH
Software Engineering Institute