Analysis Contracts for CPS

Presenter: Dionisio de Niz¹

dionisio@sei.cmu.edu

CPS V&V Workshop

Team: Ivan Ruchkin², Sagar Chaki¹, David Garlan²

¹Software Engineering Institute &
²School of Computer Science
Motivation

The development of Cyber-Physical Systems (aircrafts, cars, trains, robots, etc.) increasingly relies on many types of analyses from different disciplines for assurance purposes

- Control stability, scheduling, logic, thermal, power, aerodynamics, etc.

Large CPS are integrated out of components developed by suppliers that use their own analysis methods and make their own assumptions

Analysis assumption mismatches are discovered late in the system integration phase

- Difficult and costly to solve
Boeing 787 Suppliers

Source: Boeing / Reuters
Inter-Domain Analyses Interactions

- Scheduling + Frequency Scaling
- Selected Voltage
- Battery Recharge Scheduling
- Cell Interconnects
- Thermal Runaway Analysis

Source: National Renewable Energy Laboratory
Analysis Contracts

- Frequency Scaling
  - Power
  - Exec Time
- Schedulability
- Model checking
  - deadlock
- Control Stability
- Sensor Board
- CPU
- Actuator Board
- Sensor Sampling
- PID Controller
- Actuator Controller
- Assumption
- Guarantee
- Battery Sched
- Thermal runaway
- Discharge
- Charge
- Comm Protocol (reliable/unreliable)
- Software Engineering Institute | Carnegie Mellon
Analysis Contracts Structure

Domain (e.g. in SPIN)

Contract (Annex)

Analysis Plugin

Domain (e.g. in SMT)

Contract (Annex)

Analysis Plugin

AADL Model

Contract (Annex)

Analysis Plugin

Contract (Annex)

Analysis Plugin
Contract Semantic Basis

Domain \((\mathcal{A}, \mathcal{S}, \mathcal{R}, \mathcal{T}, \llbracket \cdot \rrbracket_\sigma)\)

- **Sorts** \(\mathcal{A} = \{A_1, A_2, \ldots\}\), e.g. Booleans, Integers, Threads, Priorities
- **Static Properties** : \(\mathcal{S} = \{S_i\}, S_i : A_1 \times \cdots \times A_j \mapsto A_k\)
  - Design-time invariants, Regular operators: \((\land, \mathcal{B} \times \mathcal{B} \mapsto \mathcal{B})\)
- **Runtime Properties** : \(\mathcal{R} = \{R_i\}, R_i : A_1 \times \cdots \times A_j \mapsto A_k\)
  - Evolving valuation at different states \(q\): \(q(R_i)\)
- **Domain execution semantics** : \(\mathcal{T}\)
  - Infinite sequence of assignments to runtime properties (executions)
- **Domain interpretation of sorts/static properties** : \(\llbracket \cdot \rrbracket_\sigma\)
  - E.g. allowed schedulers, some left *uninterpreted*

**Architectural Model Interpretation** : \(\llbracket \cdot \rrbracket_M\) on \(\mathcal{A}, \mathcal{S}, \mathcal{T}\)

- E.g. threads and periods: \(\llbracket T \rrbracket_M = \{t_1, t_2\}; \llbracket \text{Per} \rrbracket_M = \{t_1 \mapsto 10, t_2 \mapsto 20\}\)

**Executions of system defined by M** : \(\llbracket \mathcal{T} \rrbracket_M\)

- Combining \(\llbracket \cdot \rrbracket_\sigma, \llbracket \cdot \rrbracket_M\) into \(\llbracket \cdot \rrbracket\)
- Each state \(q\) in possible states \(Q\) maps \(R_i\) to function \(q(R_i)\): \(\llbracket A_1 \rrbracket \times \cdots \times \llbracket A_j \rrbracket \mapsto \llbracket A_k \rrbracket\)
- With all infinite sequence of states \(Q^\omega\)
- \(\llbracket \mathcal{T} \rrbracket_M \subseteq Q^\omega\)
Contract Language

Contract formulas

- Given domain \( \sigma = (\mathcal{A}, \mathcal{S}, \mathcal{R}, \mathcal{I}, \mathcal{O}) \),
- \( \mathcal{F}_\sigma ::= \forall v_1, ..., v_j \cdot \phi \mid \exists v_1, ..., v_j \cdot \phi \mid \forall v_1, ..., v_j \cdot \phi : \psi \mid \exists v_1, ..., v_j \cdot \phi : \psi \)
  - \( v_i : A_i, \phi : \) static (first order) formula
  - \( \psi : \) LTL formula

Contract \( C = (I, O, A, G) \)

- \( I \subseteq (\mathcal{A} \cup \mathcal{S}) : \) Sorts and properties read by the analysis
- \( O \subseteq (\mathcal{A} \cup \mathcal{S}) : \) Sorts and properties written by the analysis
- \( A \subseteq \mathcal{F}_\sigma : \) assumptions: must be true in input
- \( G \subseteq \mathcal{F}_\sigma : \) guarantees: must be true in output
Contract Verification

Given model M and set of analyses $\mathcal{AN} = \{An_i\}$:

- For $An_i$, $C = (I, O, A, G)$: M application to $An_i$ iff
  
  $- \forall a \in A \cdot M \models a, \forall g \in G \cdot An_i(M) \models g$

Valid analysis ordering: no dependencies from later analysis

- Contract (& analysis) dependency: $d(C_i, C_j): C_i. I \cap C_j. O \neq \emptyset$

Contract Formulas

- First order: in SMT (Z3)
- LTL : Model checker
- FOL + LTL: Generate all solutions for FOL, check LTL in each
Example: Surveillance Aircraft

Software

```
main()
{
  for(i=0;;)
    ...
}
```

Security: Top Secret

```
main()
{
  for(i=0;;)
    ...
}
```

Security: Secret

Analysis

**Security**: tasks of different level to different processor

**Scheduling**: meet all deadlines

**Freq. Scaling**: minimize power

**Logic**: no deadlocks or race conditions

**Battery scheduling**: meet battery lifetime

**Battery thermal**: no runaways
Surveillance Aircraft Contracts

Security Analysis

- $A_{sec}.C: I = \{T, ThSecCl\}, O = \{NotColoc\}, A = \emptyset, G = \{g\}$
  - $g: \forall t_1, t_2 \cdot ThSecCl(t_1) \neq ThSecCl(t_2) \Rightarrow t_1 \in NotColoc(t_2)$

Multiprocessor scheduling: (Binpacking + scheduling)

- $A_{sched}.C: I = \{T, C, NotColoc, Per, WCET, Dline\}, O = \{CPUBind\}, A = \emptyset, G = \{g\}$
  - $g: \forall t_1, t_2 \cdot NotColoc(t_2) \Rightarrow CPUBind(t_1) \neq CPUBind(t_2)$

Frequency Scaling

- $A_{freqsc}.C: I = \{T, C, CPUBind, Dline\}, O = \{CPUFreq\}, G = \emptyset, A = \{a\}$
  - $a: \forall t_1, t_2 \cdot CPUBind(t_1) = CPUBind(t_2) \Rightarrow Dline(t_1) < Dline(t_2)$

Model checking periodic program (REK):

- $A_{rek}.C: I = \{T, C, Per, Dline, WCET, CPUBind\}, O = \{ThSafe\}, G = \emptyset, A = \{a_1, a_2\}$
  - $a_1: \forall t_1 . Per(t) = Dline(t)$, $a_2: \forall t_1, t_2 \cdot G(CanPrmpt(t_1, t_2) \Rightarrow Dline(t_1) < Dline(t_2))$

Thermal runaway:

- $A_{therm}.C: I = \{B, BatRows, BatCols, Voltage\}, O = \{K\}, A = \emptyset, G = \emptyset$

Battery Scheduling

- $A_{bsched}.C: I = \{B, BatRows, BatCols\}, O = \{BatConnSchedPol, HasReqLifetime, SeriqlReq, ParalRea\}, A = \emptyset, G = \{g\}$
  - $g: G(K(0) \times TN(0) + K(1) \times TN(1) + K(2) \times TN(2) + K(3) \times TN(3)) \geq 0$
Satisfying Assumptions in Specific System: Frequency Scaling

\[ a: \forall t_1, t_2 \cdot CPUBind(t_1) = CPUBind(t_2) \Rightarrow G(\text{CanPrmp}(t_1, t_2) \Rightarrow \text{Dline}(t_1) < \text{Dline}(t_2)) \]
Satisfying Assumptions in Specific System: Battery Scheduling Assumption

\[ g: G(K(0) \times TN(0) + K(1) \times TN(1) + K(2) \times TN(2) + K(3) \times TN(3) \geq 0) \]

Ratio of cells with 0, 1, 2, 3 neighbors:

\[ 1 \cdot TN(1) - 1 \cdot TN(2) + 10 \cdot TN(3) \geq 0 \]

Cells with 0 neighbors: 1
Cells with 1 neighbor: 14
Cells with 2 neighbors: 1
Cells with 3 neighbors: 12

1 \cdot 14 - 1 \cdot 10 + 10 \cdot 2 = 14 \geq 0
1 \cdot 2 - 1 \cdot 14 + 10 \cdot 0 = -12 < 0
ACTIVE Tool

Plugins for OSATE that consist of

Contract Annex Sublanguage (Parser)
  • Captures analysis contracts within AADL models

ACTIVE Executer
  • Interprets contracts and identifies valid execution schedules

ACTIVE Verifier
  • Identifies and run contract verification engines
package Contracts
public
with Contract_Properties, SEI,
   Deployment_Properties;
annex contract {**
   name SecureAllocationAnalysis
   input thread, thread.SecurityLevel
   output thread.Not_Colocated
   guarantees
      forall x1: thread, x2:thread :
      x1 != x2 ->
         (x1.SecurityLevel = x2.SecurityLevel) ||
         x1 in x2.Not_Colocated
      ...
**};
end Contracts;

package AircraftPackage
public
with Contract;
with Contract_Properties, SEI,
   Deployment_Properties;
annex contract {**
   system implementation AircraftSystem.i
   ...
   annex contract {**
      use contracts::SecureAllocationAnalysis,
         contracts::BinPackingAnalysis,
      ...
      **};
   end AircraftSystem.i;
end AircraftPackage;
Analysis Contract Annex Examples

**name** LlrekAnalysis


**output** system.Is_Thread_Safe

**assumes**

forall t: thread : t.Period = t.Deadline

**assumes**

forall t1:thread, t2: thread | t1 != t2 :

G (CanPreempt(t1, t2) => (GlCanPreempt(t2, t1)))

**name** BatterySchedulingAnalysis


**output** device.Serial_Req, device.Paral_Req, device.Bat_Scheduler

**guarantees**

forall b: device :

G (b.K0 * b.TN(0) + b.K1*b.TN(1) + b.K2*b.TN(2) + b.K3*b.TN(3) >= 0 )
ACTIVE Executer

Responsibilities

- Parse “use” subclauses in scope
- Retrieve referred contracts from libraries
- Create dependency graph
  * no cycle support
- Correct sequence of execution of analysis and verifiers
  - Plugin wrappers to prevent compiling dependencies
    + Monitors plugin completion

Diagram:

- User
- Executer
- Wrapper
- Plugin
- AADL Instance Model
- External analysis tool

Legend:

- Human
- Data
- Object
- Executable
- Call/access

Software Engineering Institute | Carnegie Mellon
ACTIVE Verifier

Decompose contract formulas into segments verifiable by single verifier

1. \text{<Quan><Var><Predicate Expression>}
   - e.g.: \text{forall t:thread | t.Period = t.Deadline}
   - sent to SMT solver. Each solution is later blocked to get all solutions
   - Identifies part of model that should be verified

2. \text{<LTLExpression>}
   - Identify domain atoms and functions: e.g. \text{thread, CanPreempt()}
   - Select verifier that can interpret all atoms, functions, and operators
   - Translate AADL model into verifier data (e.g. threads, periods, scheduler, etc.)
   - Run verifier
   - Interpret and put results back into AADL model
Conclusions

Analyses from multiple domains are key for development of CPS

- But inconsistent assumptions may compromise results

Analysis contracts to automatically verify assumptions

- Analysis contract language & verification framework
- Implementation in AADL sub-annex

ACTIVE (Analysis ContracT Integration VErifier) Tool

- Extensible
- ACTIVE Executer + ACTIVE Verifier

Tested on two domains and five analyses