

# Symbolic Assurance Refinement

Dionisio de Niz, Lutz Wrage  
Assuring Cyber-Physical Systems

**Copyright 2023 Carnegie Mellon University.**

**This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.**

**The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.**

**NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.**

**[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.**

**This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at [permission@sei.cmu.edu](mailto:permission@sei.cmu.edu).**

**Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.**

**DM23-0570**

# DoD Digital Engineering Challenges

## Model-Analyze-Build

**Late Discovery** of Design Errors in DoD Systems is very costly.

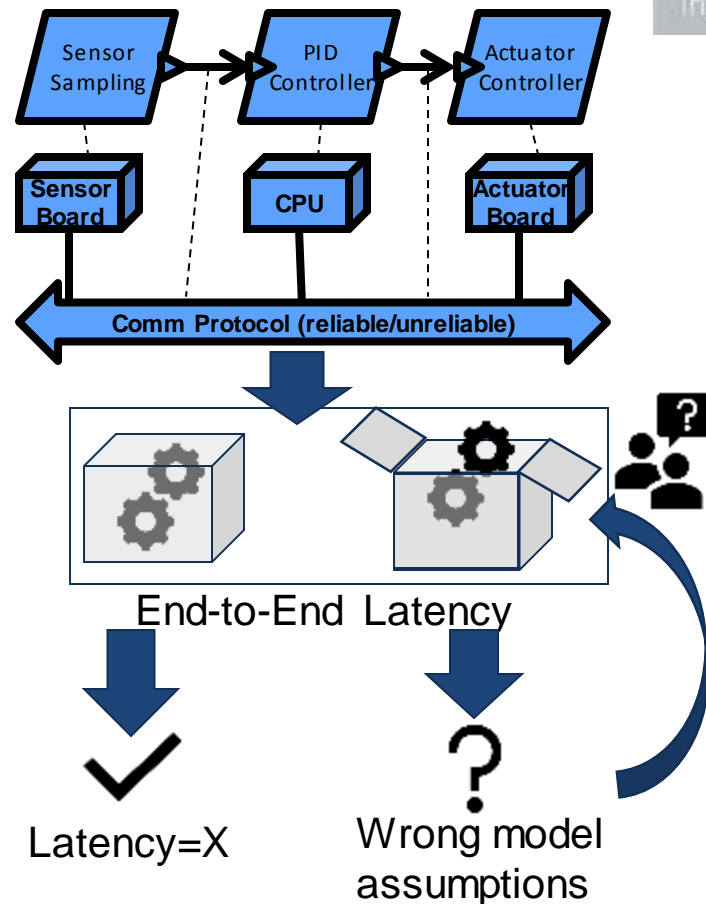
Architecture modeling and analysis can **detect** design **error early**

**BUT:**

Analysis assumptions are often implicit  
if analysis **assumptions not met**: analyses break down for reasons not clear to users of analysis tools.

E.g., e2e Latency Assumption: periods multiple of each other (harmonic)

**DoD barrier for adoption**

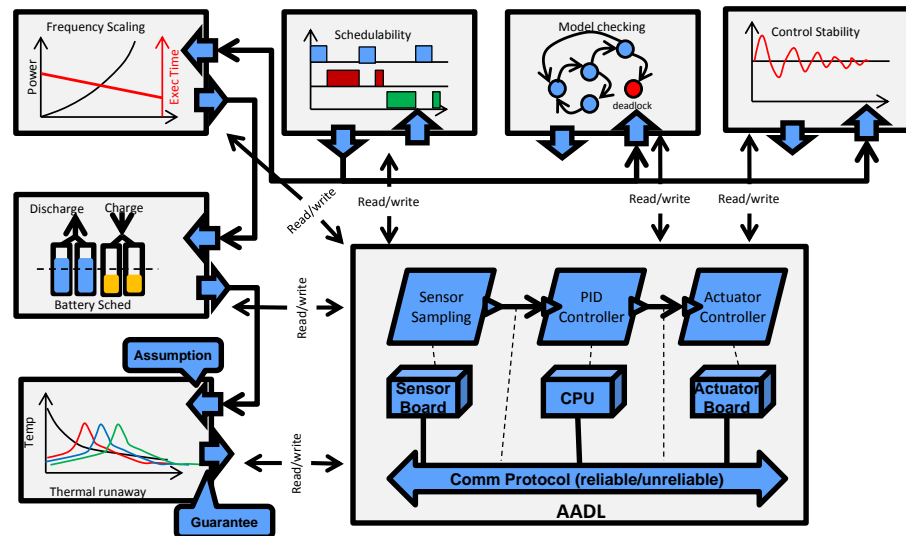


# Digital Engineering: Multiple Claims - Multiple Analyses

## Different Assurance Claims

- Combine multiple analysis
- Validate assumptions
- Resolve assumption conflicts

**Integrate into arguments to satisfy claims**



# Analysis Contract: Tracking Assumptions and Guarantees

**contract** {

inputs:

E2ELatencies

**assumptions:**

areConnectionsDelayed()

areDeadlinesConstrained()

areTasksSchedulable()

areAllThreadsPeriodic()

**analysis:**

meetEndToEndLatencies()

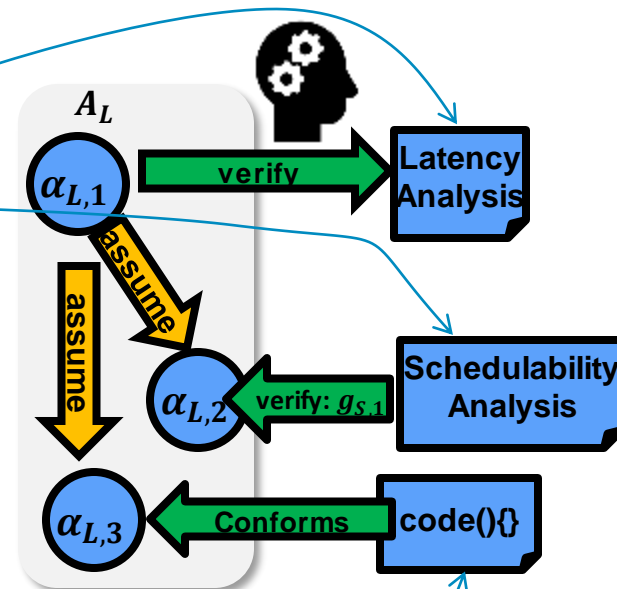
**guarantee:**

[E2EResponses[i] <= E2ELatencies[i]

for i in range(len(Responses))]

}

$$C_L = (A_L, G_L) \text{ with } A_L = \{\alpha_{L,1}, \alpha_{L,2}, \alpha_{L,3}\}$$



# Shift Left And Down to the Metal

## Early Analysis

- Evaluate design decisions with partial information
- E.g., latency analysis before worst-case execution time (WCET)
  - periods of tasks must be multiples of each other

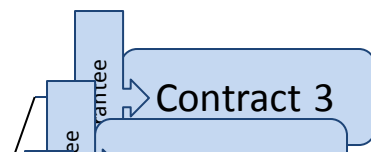
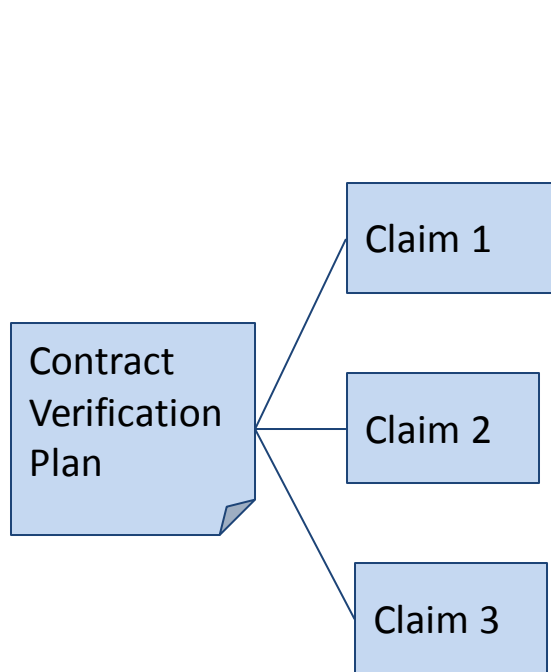
## Refinement

- Track pending information
  - WCET
- Track and execute pending verification
  - Schedulability

## Conformance

- Track implementation assumption
- Verify implementation conformance
  - Task executed strictly periodic  $\alpha_{L,3}$

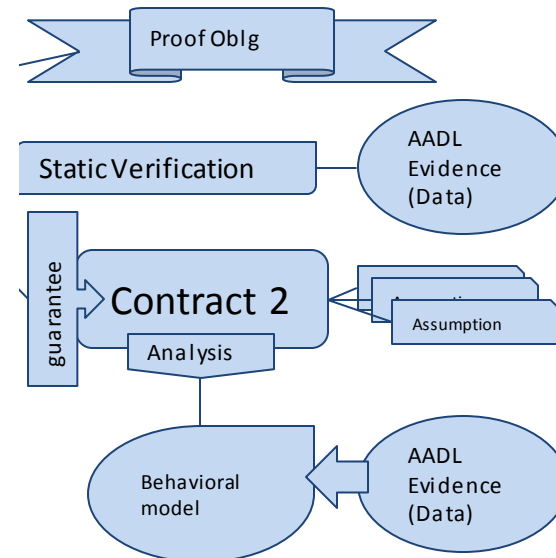
# Assurance Contract Argumentation



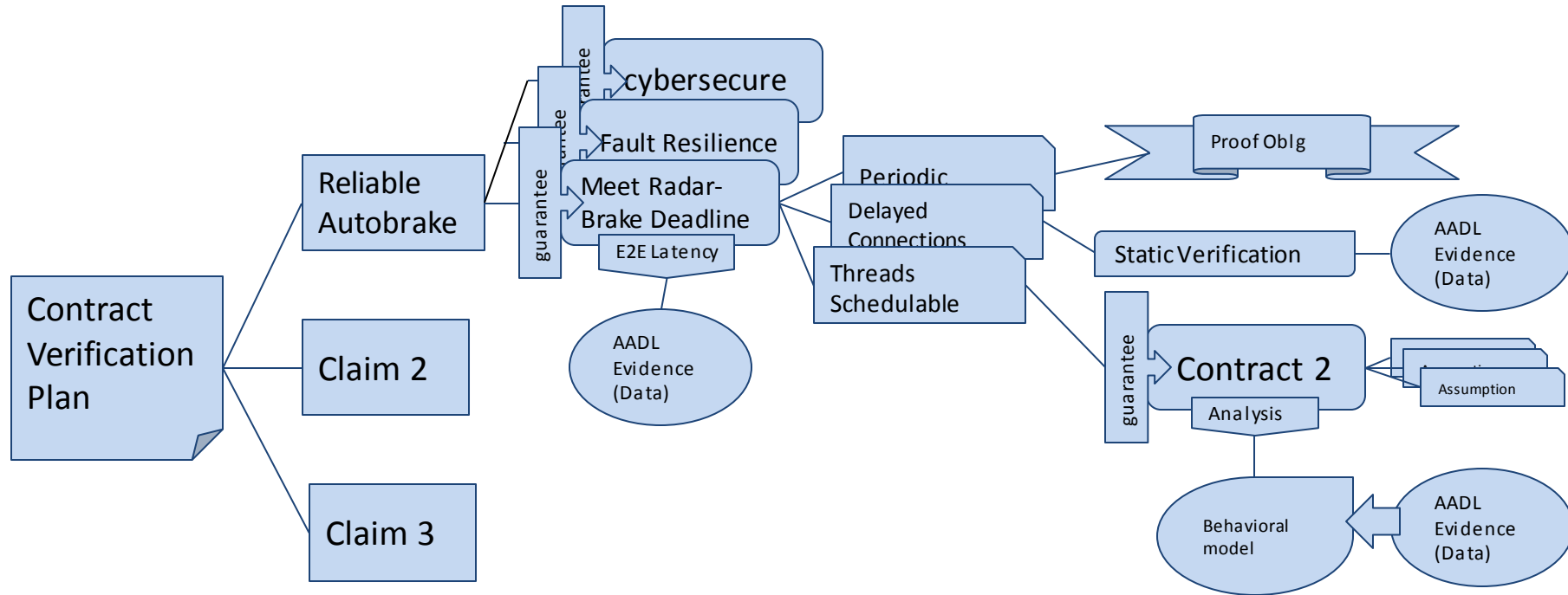
```

1 annex contract {**
2   contract <name> {
3     queries
4       <model var> =
5         <query to obtain model data>
6     domains
7       <domain reference>
8     input assumptions
9       <Bool func to check data consistency>(
10        <model vars>)
11    assumptions
12      <Bool func>( <model vars>)
13      -> <symbolic assertion>
14    analysis
15      <Bool func>( <model vars>)
16      -> <symbolic guarantee>
17  }
18 **};
  
```

Listing 1: Contract Template



# Assurance Contract Argumentation



# Symbolic Contract Argumentation

## Assumptions

- Constraints that must be satisfied for a valid analysis

## Analysis

- Evaluate whether the guarantee can be discharged

## Guarantee

- Assertion presented as a true fact on model

## Implementation

- Satisfiability Modulo Theories (Z3)
- Implements contract argumentation
  - Evaluate whether constraints can be satisfied with facts from analysis guarantees
- Validate assumptions
  - Proof obligations: lack of constraints allow any value that satisfy assumption (e.g., RM priorities)

# Encoding In SMT

## Verification Plan

- $P = (K, D)$ 
  - $K := \{(k_i, C_i)\}$
  - $k_i$  predicates over symbolic variables

## Contract

- $C_i = (V_d, Q, I, A, G, N)$ 
  - $V_d$  symbolic variables from a domain
  - $Q$  model variables (e.g., from AADL models)
  - $I$  input assumptions (if enough data for analysis)
  - $A = (p, a)$  set of assumptions
  - $G$  guarantee
  - $N$  analysis predicate (imperative Boolean function)

---

### Algorithm 1 getSMTEncoding( $Plan$ )

---

```

1:  $F \leftarrow \{k_i \mid (k_i, C_i) \in Plan.K\}$ 
2:  $T \leftarrow \{C_i \mid (k_i, C_i) \in Plan.K\}$ 
3: while  $T \neq \emptyset$  do
4:   select  $t$  from  $T$  and remove it from  $T$ 
5:   if  $t$  is argument then
6:      $T \leftarrow T \cup t.C_\alpha$ 
7:      $F \leftarrow F \cup (\text{replG4C}(t.F_\alpha) \implies t.G_\alpha)$ 
8:   else if  $\bigwedge_{i \in t.I} i$  then
9:     for  $p \in \{p \mid (p, a) \in t.A\}$  do
10:      if  $p$  is contract then
11:         $T \leftarrow T \cup p$ 
12:         $F \leftarrow F \cup (\bigwedge_{\{a \mid (p, a) \in t.A\}} a) \wedge N \implies t.G$ 
13:      else
14:         $F \leftarrow F \cup (p \implies a)$ 
15:      end if
16:    end for
17:   end if
18: end while
19: return  $F$ 

```

---

# Contract Argumentation Scalability

## Exploit Knowledge from Scientific Domain

- Efficient algorithms from specialized domains
  - E.g., greedy worst-case response time in real-time theory
  - Implemented in imperative languages

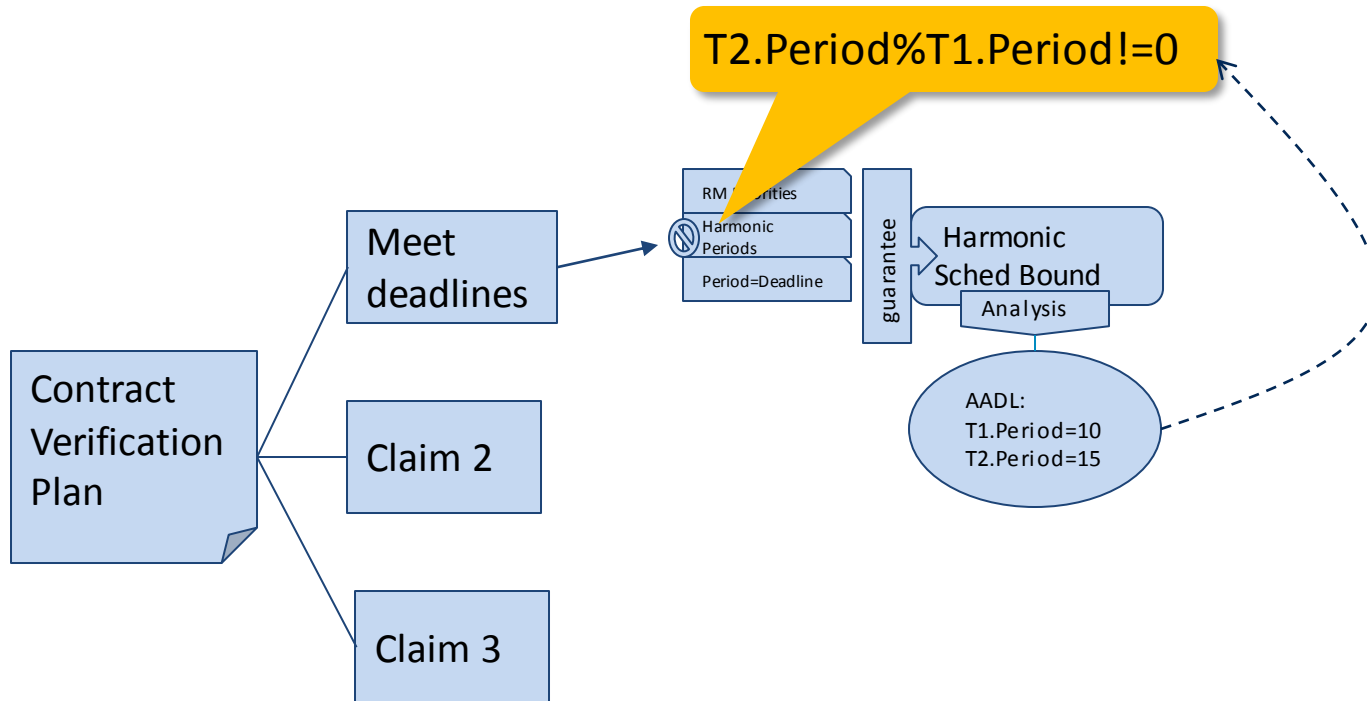
## Assume correctness of analysis

- When validating the contract argumentation
- To be connected with other lower-level verification results
  - E.g., PROSA: coq (theorem prover) verification of real-time theory

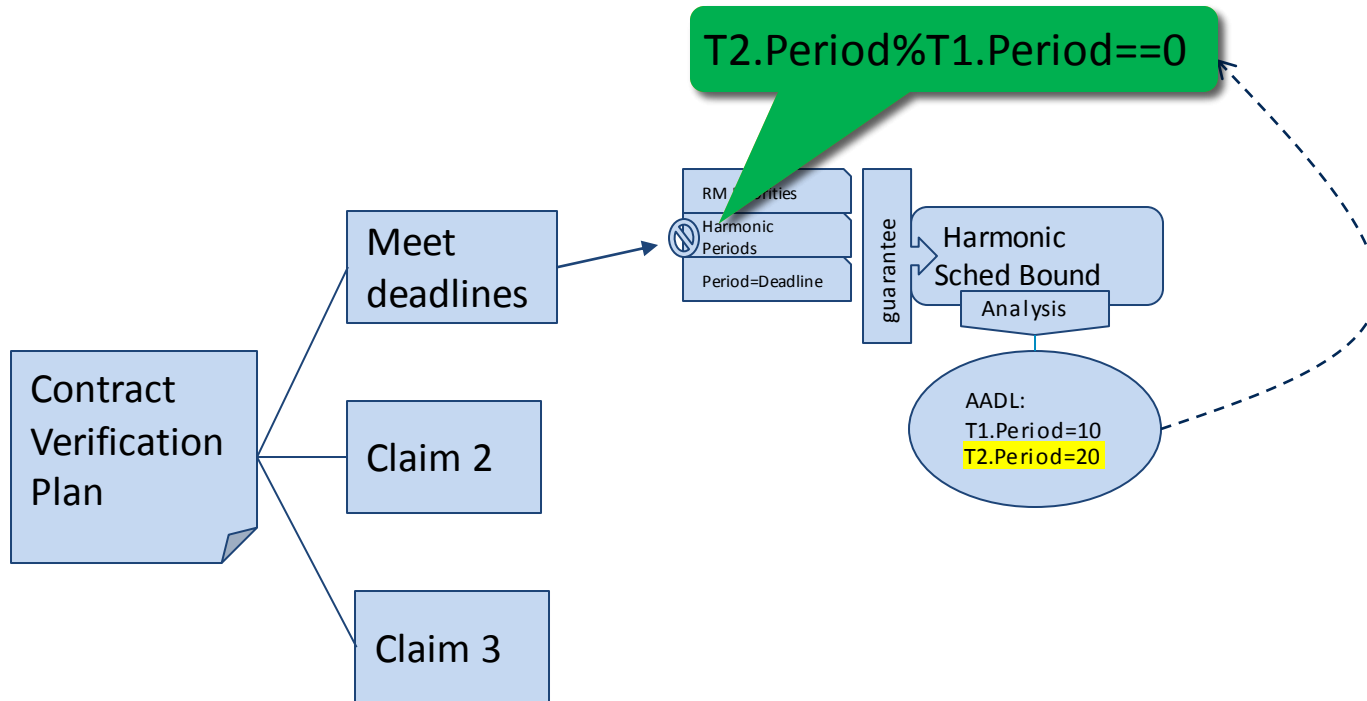
## Correctness of implementation

- Exploit proven properties of runtime mechanisms: e.g., schedulers, hypervisors
- Exploit code generation
- Deferred code verification to conform to assumptions

# Repairing Assumptions



# Repairing Assumptions



# Repairing Assumptions

$T2 \text{ Period} \% T1 \text{ Period} \neq 0$

Contract  
Verification  
Plan

```

1 annex contract {**
2   argument schedulability
3   argument
4     Or(RMBound, RTA)
5   ...
6 }
7
8 contract RMBound {
9   assumptions
10    RMPriorities(periods, priorities )
11   analysis
12    RMBoundTest(...)
13 }
14
15 contract RTA {
16   assumptions
17   ...
18   analysis
19    RTATest (...)
20 }
21 **}

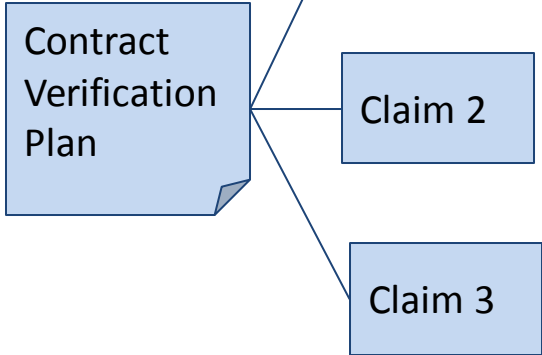
```

Listing 3: Path Selection Based on Assumptions

Suggest  
Non-  
doe  
periods

# Argument Modularity

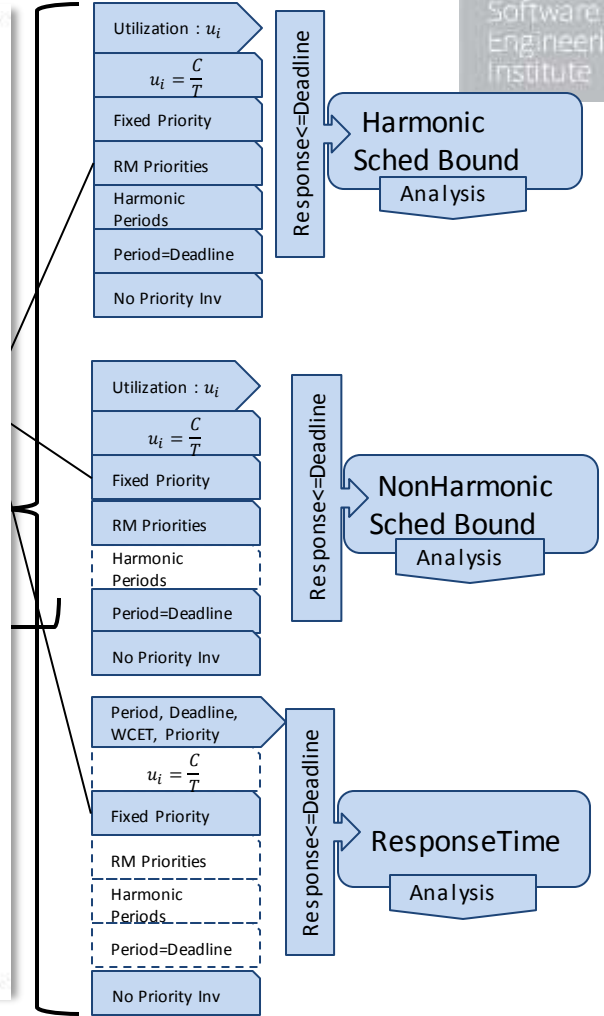
Decomposed into subclaims



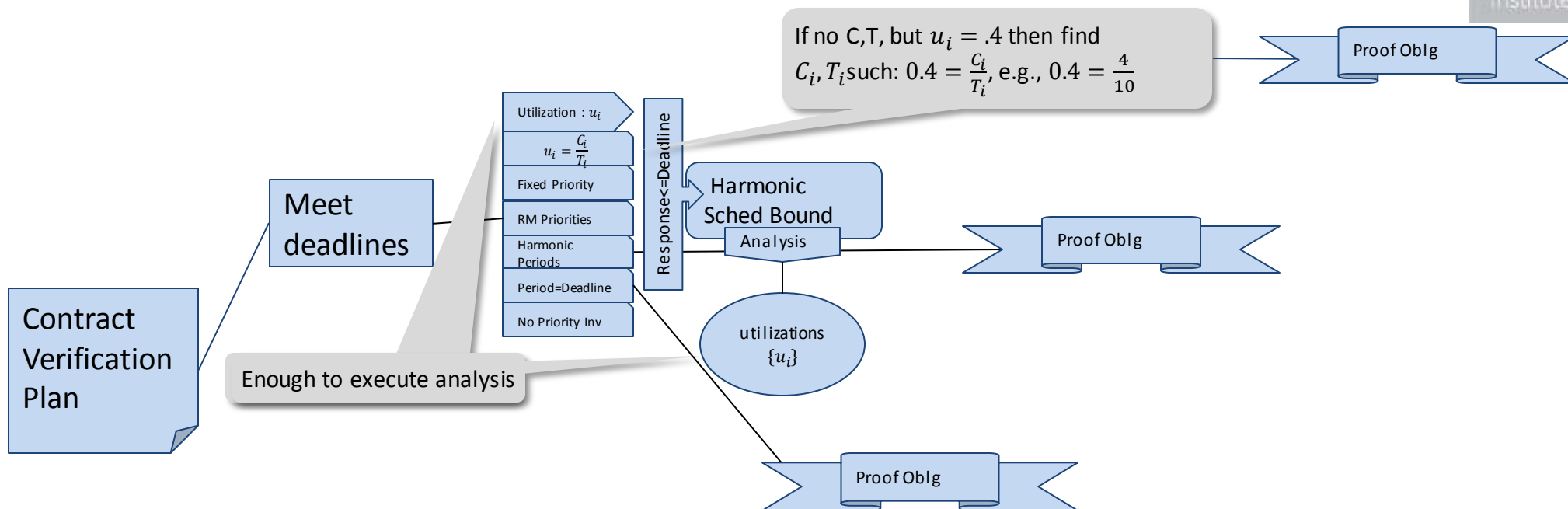
```

1 annex contract {**
2 verification plan myPlan {
3   claims
4     EndToEndDelayArgument->
5     And([E2EResp[i] <= E2ELatency[i]
6         for i in range(len(E2EResp))]
7   }
8
9   argument EndToEndDelayArgument {
10    argument
11    Or(E2ESched, E2ESFlowSpec)->
12    And([E2EResp[i] <= E2ELatency[i]
13        for i in range(len(E2EResp))]
14  }
15
16  contract E2ESched {
17    input assumptions
18    allSchedDataPresent()
19    ...
20  }
21
22  contract E2EFlowSpec {
23    input assumptions
24    notAllSchedDataPresent()
25    ...
26  }
27 **}
  
```

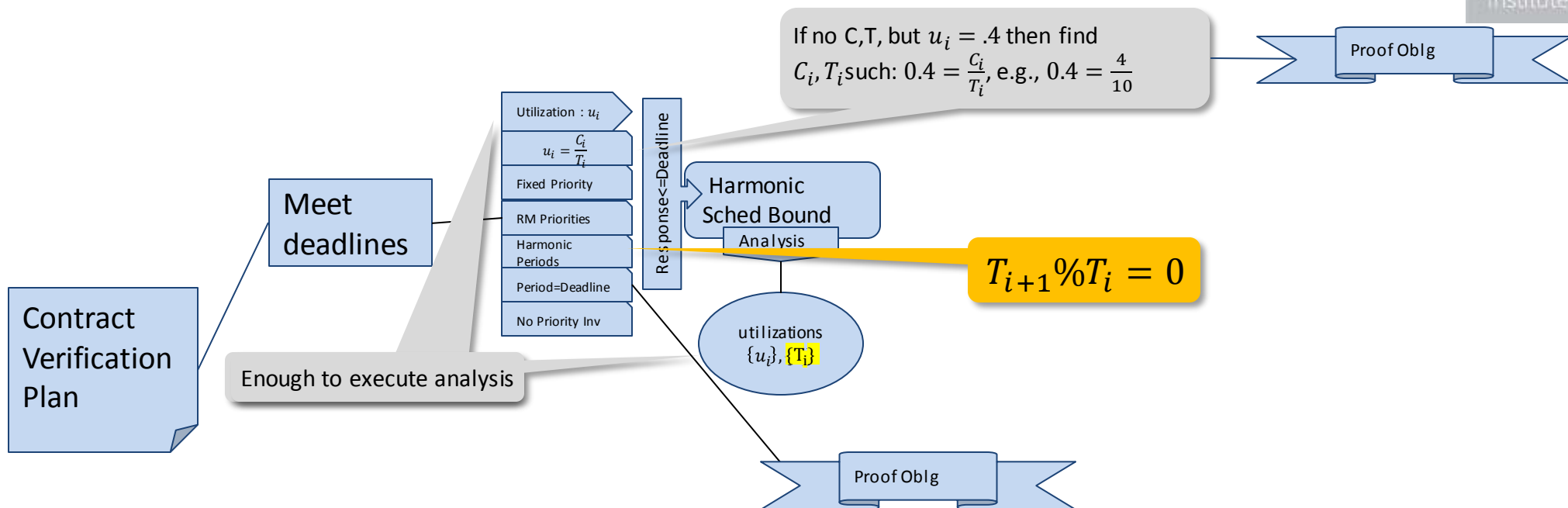
Listing 2: Path Selection Based on Refinement



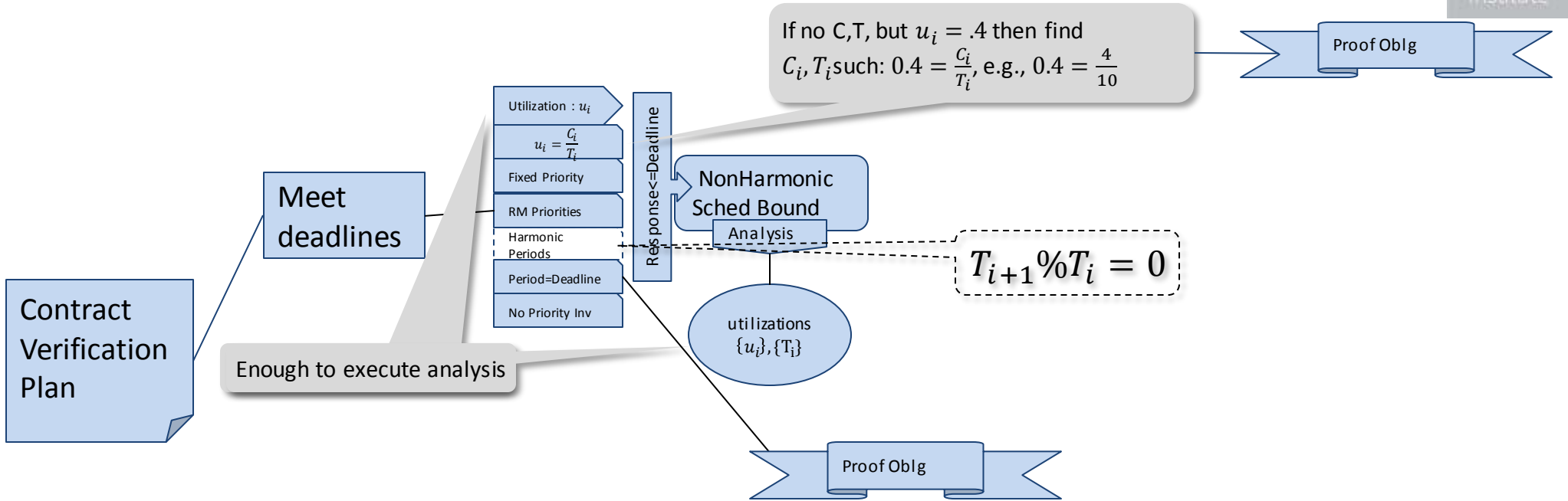
# Refinement throughout development (1)



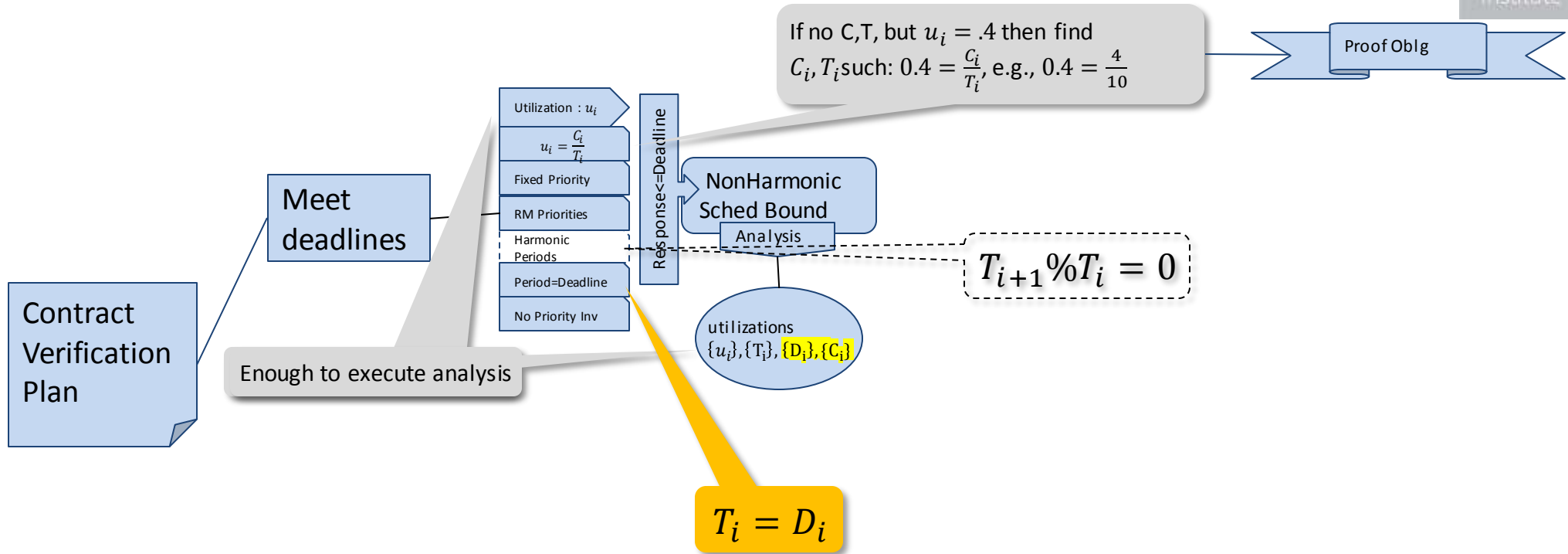
# Refinement throughout development (2)



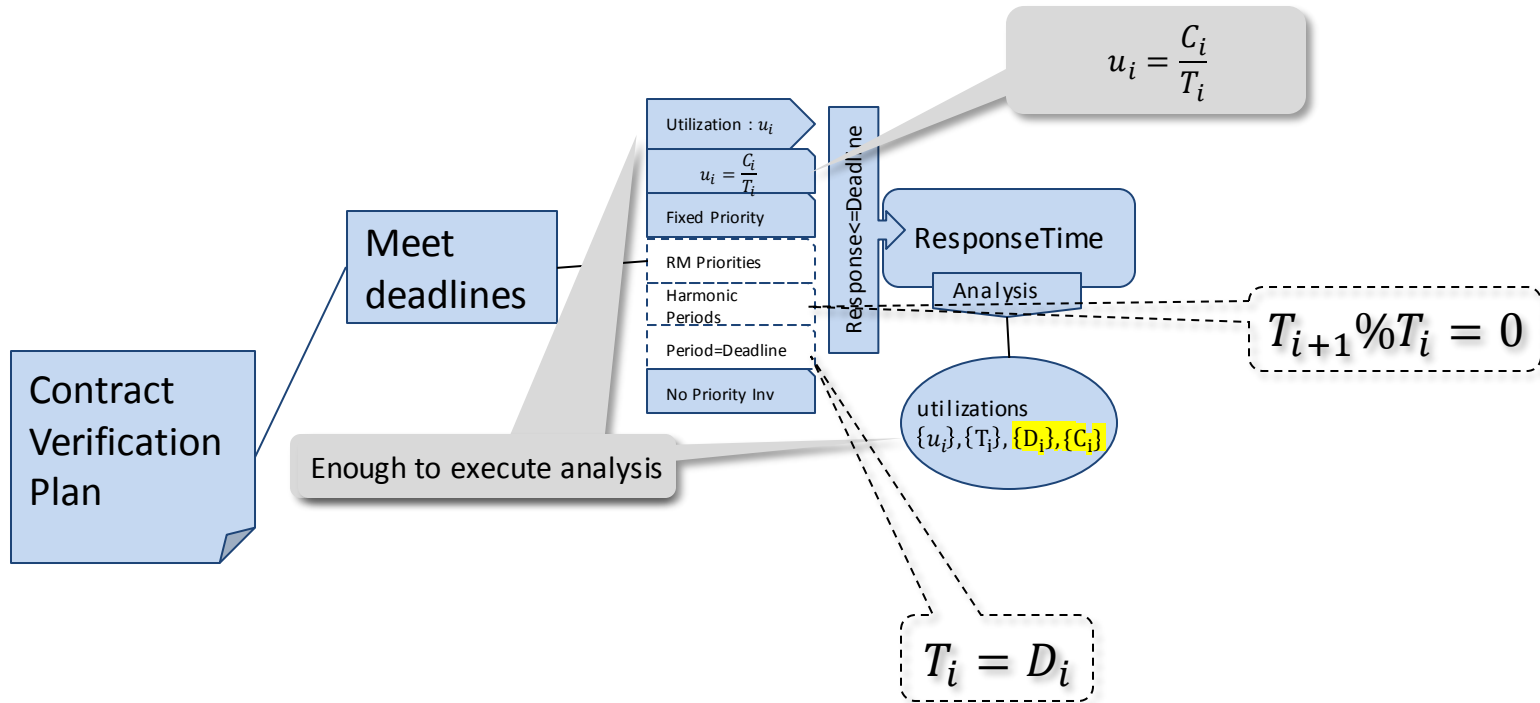
# Refinement throughout development (3)



# Refinement throughout development (4)



# Refinement throughout development (5)



# Concluding Remarks

## Assurance Argumentation

- For Execution Architectural Models (e.g., AADL)
- Encoded as Constraint-Satisfaction Problem (SMT) for automatic evaluation
- Enables Incremental Refinement
  - unassigned variables take satisfying values (proof obligations)
- Encodes alternative contracts based on refinement/design

## Takes Advantage of Previous Analyses Proofs / Efficient Implementations

- Encoded as imperative implementation of predicates over model variables

## Currently Experimenting with Avionics Models and Argumentation