ROBOSTAR

Fang Yan, Simon Foster, Ibrahim Habli

This project has received funding from the European Union's EU Framework Programme for Research and Innovation Horizon 2020 under Grant Agreement No. 812.788

Assurance Case Process of RoboChart Supported by Formal Verification

What is an Assurance Case ?

A reasoned and compelling argument, supported by a body of evidence, that a system, service or organisation will operate as intended for a defined application in a defined environment.

Can we produce ACs for RoboChart designs automatically?

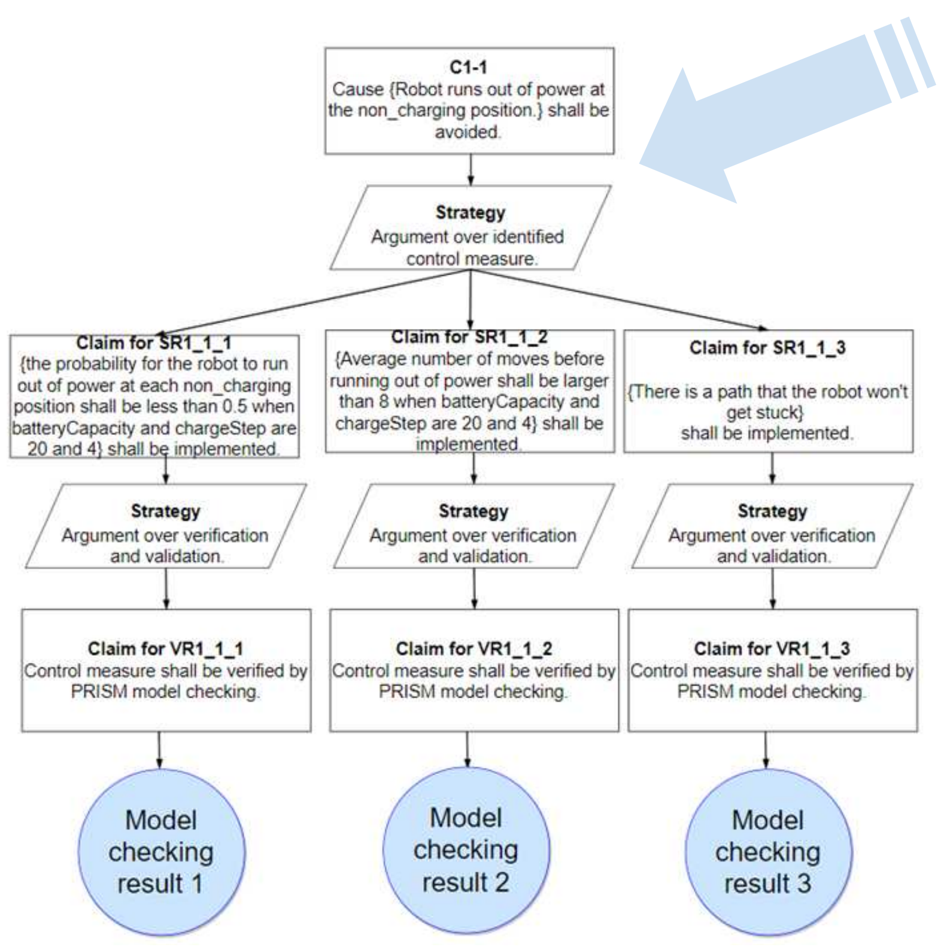
YES. Partially. We provide a model-based solution to (i) generate ACs from RoboChart models and (ii) to generate AC evidence with the formal verification capabilities supported by RoboTool.

What's the advantages of the method?

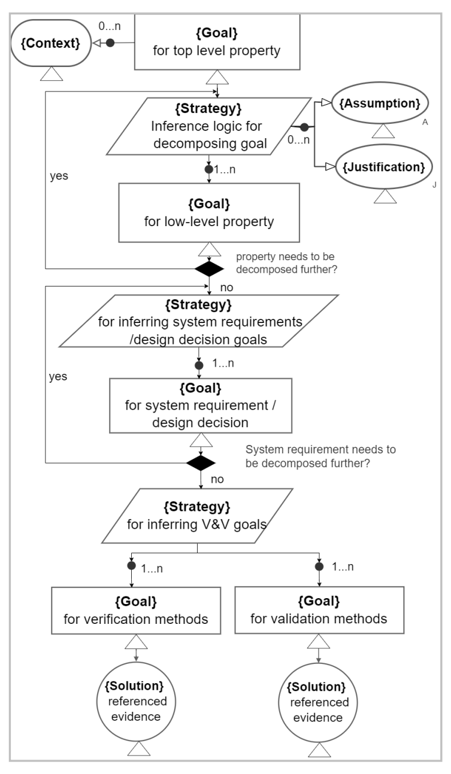
Formal verification has been applied for producing AC evidence, but formal expertise and manual processes are usually involved. We fill the gap between AC claims and the corresponding AC evidence with an **AUTOMATIC** process for RoboChart designs.

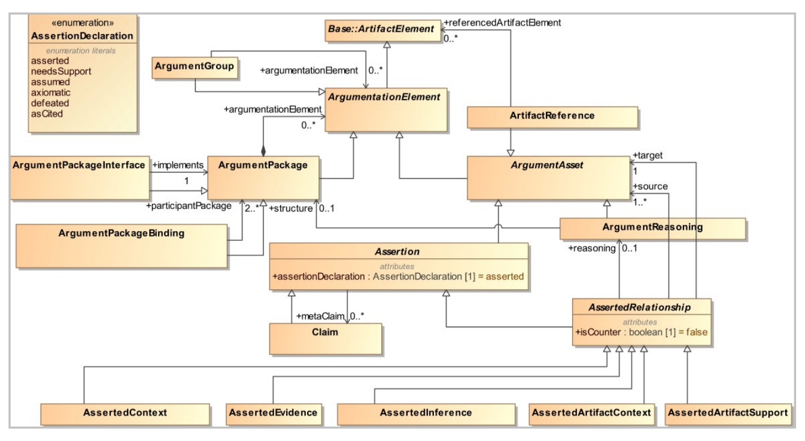
AC structure and claim generation

- A SACM compliant framework for model-based AC construction from RoboChart models combining the **pattern instantiation** and **model query** methods.



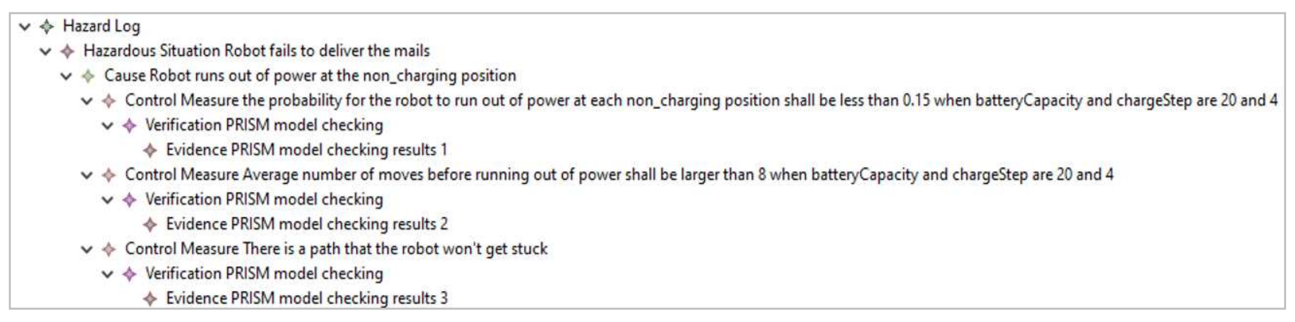
C1-1
Cause {Robot runs out of power at the non_charging position.} shall be avoided.
Strategy
Argument over identified control measure.
Claim for SR1_1_1
{the probability for the robot to run out of power at each non_charging position shall be less than 0.5 when batteryCapacity and chargeStep are 20 and 4} shall be implemented.
Claim for SR1_1_2
{Average number of moves before running out of power shall be larger than 8 when batteryCapacity and chargeStep are 20 and 4} shall be implemented
Claim for SR1_1_3
{There is a path that the robot won't get stuck}
shall be implemented.
Strategy
Argument over verification and validation.
Claim for VR1_1_1
Control measure shall be verified by PRISM model checking.
Claim for VR1_1_2
Control measure shall be verified by PRISM model checking.
Claim for VR1_1_3
Control measure shall be verified by PRISM model checking.
Model checking result 1
Model checking result 2
Model checking result 3





Algorithm 3: Design model query-st/tr

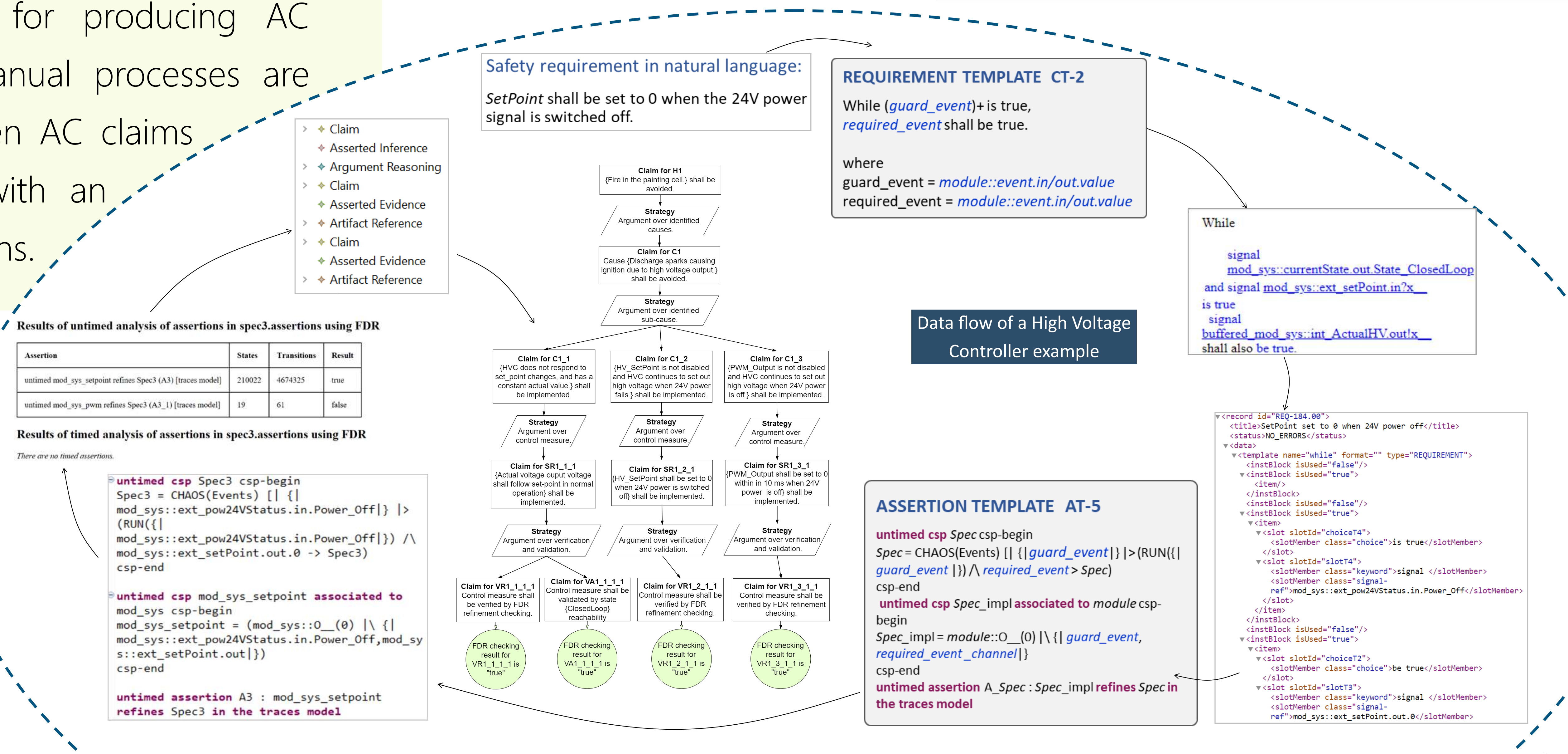
```
forall $AC.inf \in \{i \mid i.source = \text{"Query"}\}$ do
    inf.source.clear
    forall $System.state \in \{s \mid s \models Cond1\}$ do
        state.createClaim&AE
        $inf.Source \leftarrow inf.Source \cup \{stateClaim\}$
        forall
          $System.transition \in \{tr \mid tr \models Cond2\}$
          do
            tr.createEvidence
            $state.AE.Target \leftarrow stateClaim$
            $State.AE.Source \leftarrow$
              $State.AE.Source \cup \{trEvidence\}$
```



- Hazard Log
  - Hazardous Situation Robot fails to deliver the mails
    - Cause Robot runs out of power at the non_charging position
      - Control Measure the probability for the robot to run out of power at each non_charging position shall be less than 0.15 when batteryCapacity and chargeStep are 20 and 4
        - Verification PRISM model checking
          - Evidence PRISM model checking results 1
      - Control Measure Average number of moves before running out of power shall be larger than 8 when batteryCapacity and chargeStep are 20 and 4
        - Verification PRISM model checking
          - Evidence PRISM model checking results 2
      - Control Measure There is a path that the robot won't get stuck
        - Verification PRISM model checking
          - Evidence PRISM model checking results 3

AC Evidence generation by model checking

- FDR and PRISM model checking
- Automated RoboChart DSL assertion generation
- Manual input of requirement in structure natural language/template



Claim
Asserted Inference
Argument Reasoning
Claim
Asserted Evidence
Artifact Reference
Claim
Asserted Evidence
Artifact Reference
Claim for H1
{Fire in the painting cell.} shall be avoided.
Strategy
Argument over identified causes.
Claim for C1
Cause {Discharge sparks causing ignition due to high voltage output.} shall be avoided.
Strategy
Argument over identified sub-cause.
Claim for C1_1
{HVC does not respond to set_point changes, and has a constant actual value.} shall be implemented.
Claim for C1_2
{HV_SetPoint is not disabled and HVC continues to set out high voltage when 24V power fails.} shall be implemented.
Claim for C1_3
{PWM_Output is not disabled and HVC continues to set out high voltage when 24V power is off.} shall be implemented.
Strategy
Argument over control measure.
Strategy
Argument over control measure.
Strategy
Argument over control measure.
Claim for SR1_1_1
{Actual voltage ouput voltage shall follow set-point in normal operation} shall be implemented.
Claim for SR1_2_1
{HV_SetPoint shall be set to 0 when 24V power is switched off} shall be implemented.
Claim for SR1_3_1
{PWM_Output shall be set to 0 within in 10 ms when 24V power is off} shall be implemented.
Strategy
Argument over verification and validation.
Strategy
Argument over verification and validation.
Strategy
Argument over verification and validation.
Claim for VR1_1_1_1
Control measure shall be verified by FDR refinement checking.
Claim for VA1_1_1_1
Control measure shall be validated by state {ClosedLoop} reachability
Claim for VR1_2_1_1
Control measure shall be verified by FDR refinement checking.
Claim for VR1_3_1_1
Control measure shall be verified by FDR refinement checking.
FDR checking result for VR1_1_1_1 is "true"
FDR checking result for VA1_1_1_1 is "true"
FDR checking result for VR1_2_1_1 is "true"
FDR checking result for VR1_3_1_1 is "true"
Data flow of a High Voltage Controller example

Safety requirement in natural language:

SetPoint shall be set to 0 when the 24V power signal is switched off.

REQUIREMENT TEMPLATE CT-2

While (*guard_event*)+ is true,
required_event shall be true.

where
guard_event = *module::event.in/out.value*
required_event = *module::event.in/out.value*

While

signal
mod_sys::currentState.out.State_ClosedLoop
and signal mod_sys::ext_setPoint.in?x__
is true
signal
buffered_mod_sys::int_ActualHV.out!x__
shall also be true.

```
▼<record id="REQ-184.00">
   <title>SetPoint set to 0 when 24V power off</title>
   <status>NO_ERRORS</status>
 ▼<data>
   ▼<template name="while" format="" type="REQUIREMENT">
       <instBlock isUsed="false"/>
     ▼<instBlock isUsed="true">
         <item/>
       </instBlock>
       <instBlock isUsed="false"/>
     ▼<instBlock isUsed="true">
       ▼<item>
         ▼<slot slotId="choiceT4">
             <slotMember class="choice">is true</slotMember>
           </slot>
         ▼<slot slotId="slotT4">
             <slotMember class="keyword">signal </slotMember>
             <slotMember class="signal-
             ref">mod_sys::ext_pow24VStatus.in.Power_Off</slotMember>
           </slot>
         </item>
       </instBlock>
       <instBlock isUsed="false"/>
     ▼<instBlock isUsed="true">
       ▼<item>
         ▼<slot slotId="choiceT2">
             <slotMember class="choice">be true</slotMember>
           </slot>
         ▼<slot slotId="slotT3">
             <slotMember class="keyword">signal </slotMember>
             <slotMember class="signal-
             ref">mod_sys::ext_setPoint.out.0</slotMember>
```

ASSERTION TEMPLATE AT-5

untimed csp *Spec* csp-begin
Spec = CHAOS(Events) [| {|*guard_event*|} |>(RUN({| *guard_event* |}) /\ *required_event* > *Spec*)
csp-end
untimed csp *Spec*_impl **associated to** *module* csp-begin
*Spec*_impl = *module*::O__(0) |\ {| *guard_event*, *required_event_channel* |}
csp-end
untimed assertion A_*Spec* : *Spec*_impl **refines** *Spec* **in the traces model**

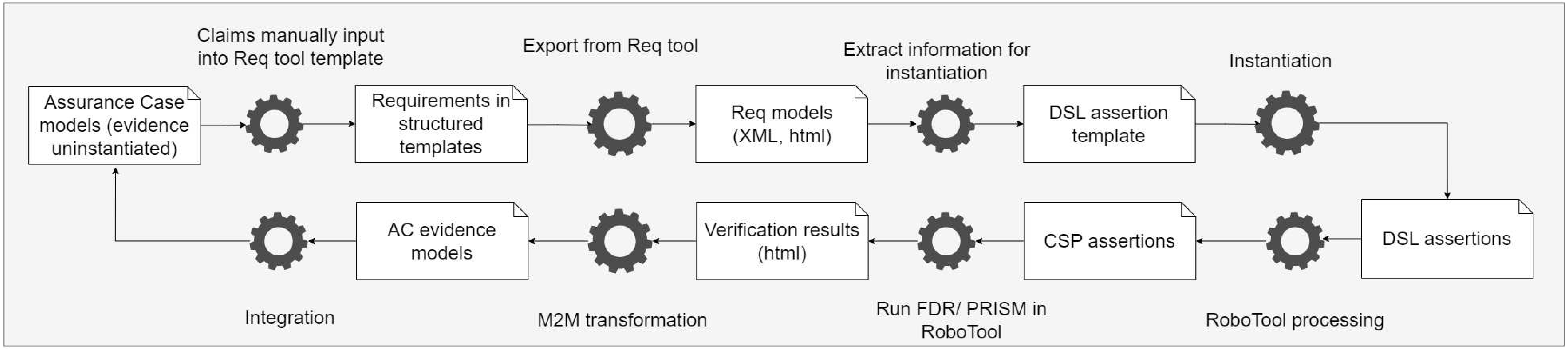
```
untimed csp Spec3 csp-begin
Spec3 = CHAOS(Events) [| {|
mod_sys::ext_pow24VStatus.in.Power_Off|} |>
(RUN({|
mod_sys::ext_pow24VStatus.in.Power_Off|}) /\
mod_sys::ext_setPoint.out.0 -> Spec3)
csp-end

untimed csp mod_sys_setpoint associated to
mod_sys csp-begin
mod_sys_setpoint = (mod_sys::O__(0) |\ {|
mod_sys::ext_pow24VStatus.in.Power_Off,mod_sy
s::ext_setPoint.out|})
csp-end

untimed assertion A3 : mod_sys_setpoint
refines Spec3 in the traces model
```

Results of untimed analysis of assertions in spec3.assertions using FDR

Assertion	States	Transitions	Result
untimed mod_sys_setpoint refines Spec3 (A3) [traces model]	210022	4674325	true
untimed mod_sys_pwm refines Spec3 (A3_1) [traces model]	19	61	false

Results of timed analysis of assertions in spec3.assertions using FDR

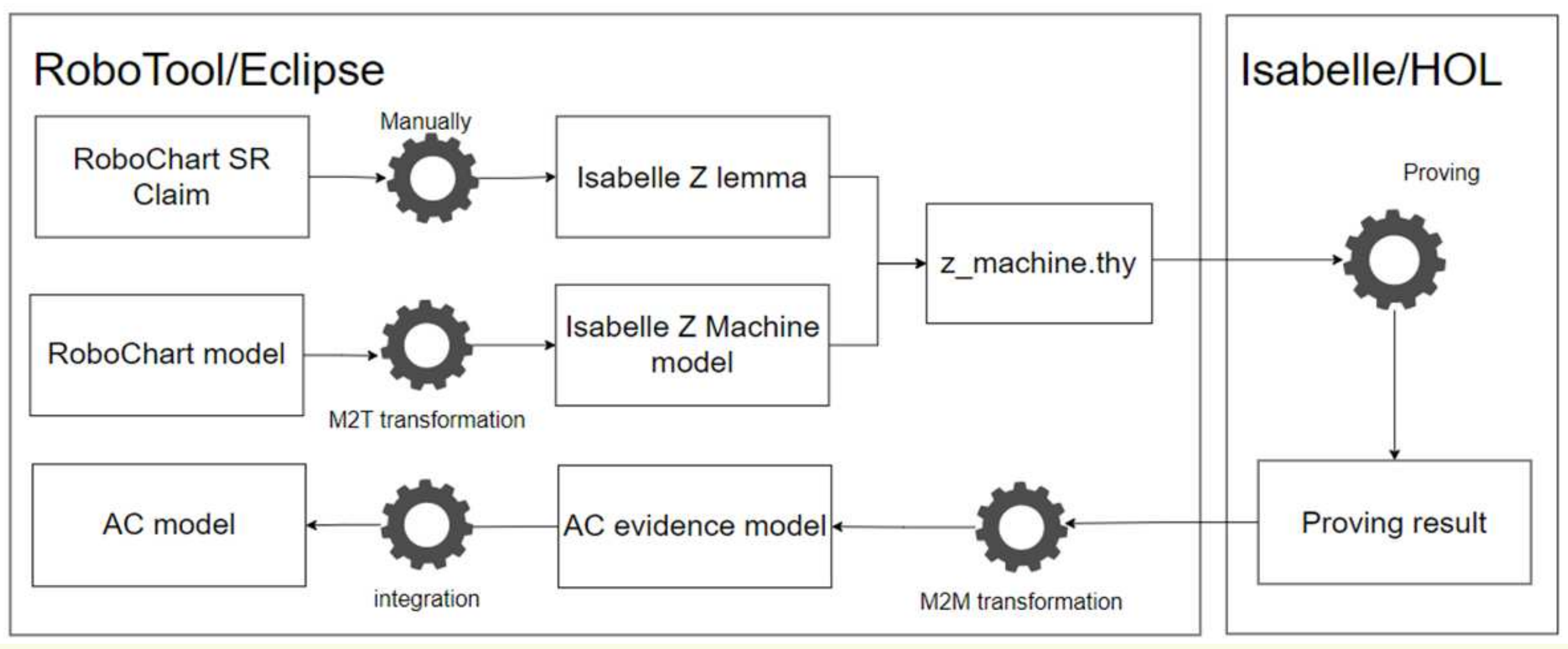
There are no timed assertions.

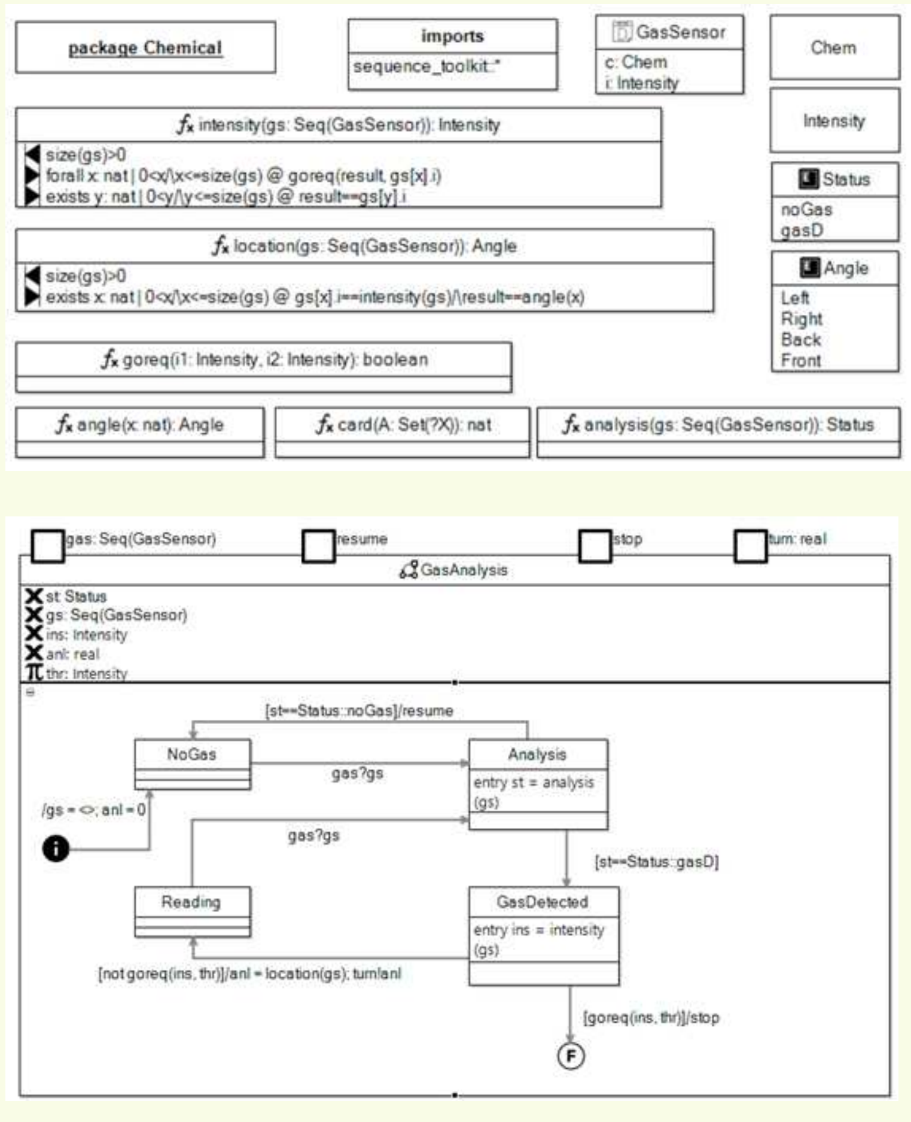


Claims manually input into Req tool template
Export from Req tool
Extract information for instantiation
Instantiation
Assurance Case models (evidence uninstantiated)
Requirements in structured templates
Req models (XML, html)
DSL assertion template
DSL assertions
CSP assertions
Verification results (html)
AC evidence models
Integration
M2M transformation
Run FDR/ PRISM in RoboTool
RoboTool processing

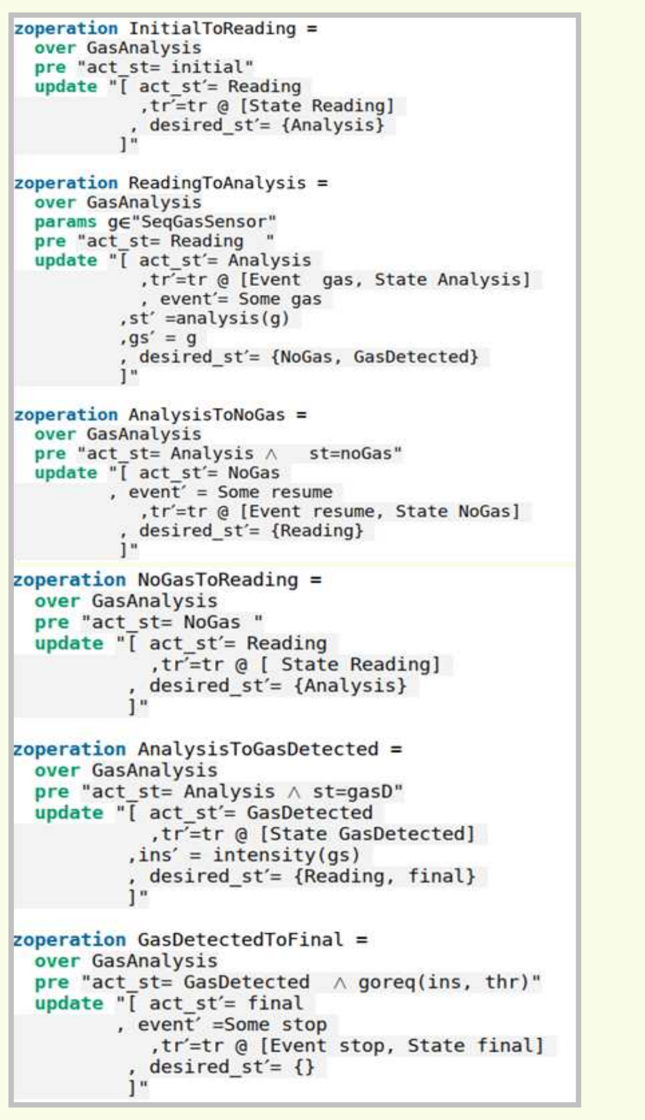
AC Evidence generation by theorem proving

- Automated transformation from RoboChart to formal notation in Isabelle/HOL
- Automated theorem proving (ATP) in Isabelle/HOL
- Manual formalization of AC claims



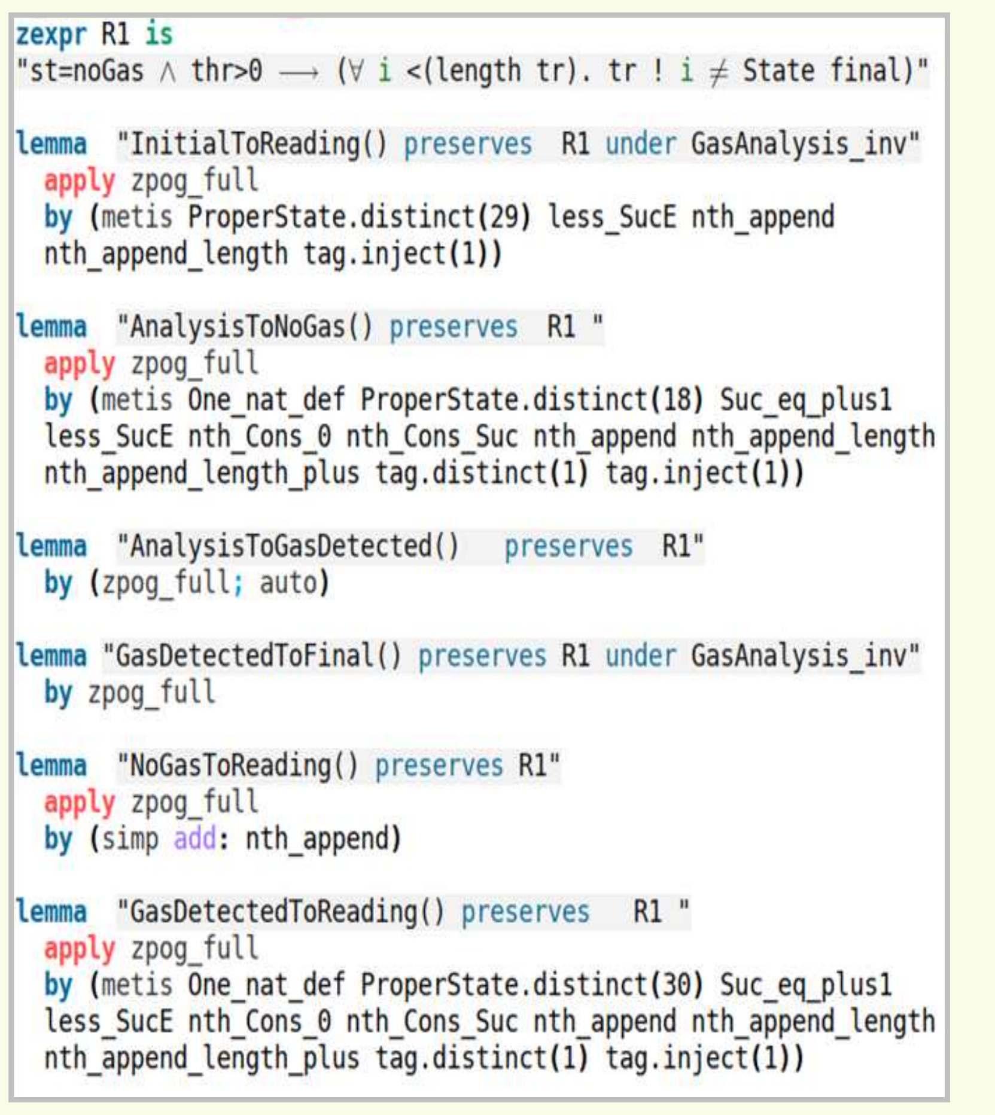


RoboChart model of Chemical Detector



```
zoperation InitialToReading =
  over GasAnalysis
  pre "act_st= initial"
  update "[ act_st′= Reading
           ,tr′=tr @ [State Reading]
          , desired_st′= {Analysis}
         ]"

zoperation ReadingToAnalysis =
  over GasAnalysis
  params g∈"SeqGasSensor"
  pre "act_st= Reading  "
  update "[ act_st′= Analysis
           ,tr′=tr @ [Event  gas, State Analysis]
           , event′= Some gas
         ,st′ =analysis(g)
         ,gs′ = g
         , desired_st′= {NoGas, GasDetected}
         ]"

zoperation AnalysisToNoGas =
  over GasAnalysis
  pre "act_st= Analysis ∧   st=noGas"
  update "[ act_st′= NoGas
          , event′ = Some resume
           ,tr′=tr @ [Event resume, State NoGas]
          , desired_st′= {Reading}
         ]"
zoperation NoGasToReading =
  over GasAnalysis
  pre "act_st= NoGas "
  update "[ act_st′= Reading
           ,tr′=tr @ [ State Reading]
          , desired_st′= {Analysis}
          ]"

zoperation AnalysisToGasDetected =
  over GasAnalysis
  pre "act_st= Analysis ∧ st=gasD"
  update "[ act_st′= GasDetected
           ,tr′=tr @ [State GasDetected]
          ,ins′ = intensity(gs)
          , desired_st′= {Reading, final}
          ]"

zoperation GasDetectedToFinal =
  over GasAnalysis
  pre "act_st= GasDetected  ∧ goreq(ins, thr)"
  update "[ act_st′= final
         , event′ =Some stop
           ,tr′=tr @ [Event stop, State final]
          , desired_st′= {}
          ]"
```

Operations in Isabelle/HOL representing RoboChart transitions



```
zexpr R1 is
"st=noGas ∧ thr>0 ⟶ (∀ i <(length tr). tr ! i ≠ State final)"

lemma  "InitialToReading() preserves  R1 under GasAnalysis_inv"
  apply zpog_full
  by (metis ProperState.distinct(29) less_SucE nth_append
  nth_append_length tag.inject(1))

lemma  "AnalysisToNoGas() preserves  R1 "
  apply zpog_full
  by (metis One_nat_def ProperState.distinct(18) Suc_eq_plus1
  less_SucE nth_Cons_0 nth_Cons_Suc nth_append nth_append_length
  nth_append_length_plus tag.distinct(1) tag.inject(1))

lemma  "AnalysisToGasDetected()   preserves  R1"
  by (zpog_full; auto)

lemma "GasDetectedToFinal() preserves R1 under GasAnalysis_inv"
  by zpog_full

lemma  "NoGasToReading() preserves R1"
  apply zpog_full
  by (simp add: nth_append)

lemma  "GasDetectedToReading() preserves   R1 "
  apply zpog_full
  by (metis One_nat_def ProperState.distinct(30) Suc_eq_plus1
  less_SucE nth_Cons_0 nth_Cons_Suc nth_append nth_append_length
  nth_append_length_plus tag.distinct(1) tag.inject(1))
```

Claim formalization and verification through ATP in Isabelle/HOL for each operation

robostar.cs.york.ac.uk

UNIVERSITY of York

Royal Academy of Engineering

UKRI

Engineering and Physical Sciences Research Council