PSL for Assume-Guarantee Contracts in AADL Models

18 October 2011
Darren Cofer
Outline

- Rockwell Collins’ DARPA META project
- Objectives
- Assume-Guarantee Verification
- Contracts on Components and Patterns
- Use of PSL in AADL
Vision

- System design & verification through pattern application and compositional reasoning
Objectives

- Verification reuse: components and design patterns
- Compositional verification for scalability
- Capture behavioral requirements in assume-guarantee contracts
- Embed contracts in AADL models
Compositional behavior verification

- **Given**
  - Assumptions for system
  - Assumptions/Guarantees for components/patterns (contracts)
- **Prove**
  - System guarantees (requirements)
- **Hierarchical**
  - Assumptions treated as axioms at one level but proof obligations at the next
  - “Leaf level” guarantees are proved outside the system design

**Example (to prove)**

\[ A_S \rightarrow A_A \]
\[ A_S \land G_A \rightarrow A_B \]
\[ A_S \land G_A \land G_B \rightarrow A_C \]
\[ A_S \land G_A \land G_B \land G_C \rightarrow G_S \]
Compositional behavior verification

- Derived from Property Specification Language (PSL) formalism
  - IEEE standard
  - In wide use for hardware verification
  - Using small subset

- Assume / Guarantee style specification
  - Assumptions: “Under these conditions”
  - Guarantees: “…the system will do X”

- Local definitions can be created to simplify properties

```plaintext
Contract:

fun abs(x: real) : real = if (x > 0) then x else -x ;

const ADS_MAX_PITCH_DELTA: real = 3.0 ;
const FCS_MAX_PITCH_SIDE_DELTA: real = 2.0 ;
const CSA_MAX_PITCH_DELTA: real = 5.0 ;
const CSA_MAX_PITCH_DELTA_STEP: real = 5.0 ;

property AD_L_Pitch_Step_Delta_Valid =
  true ->
  abs(AD_L.pitch.val - prev(AD_L.pitch.val, 0.0)) < ADS_MAX_PITCH_DELTA ;

property AD_R_Pitch_Step_Delta_Valid =
  true ->
  abs(AD_R.pitch.val - prev(AD_R.pitch.val, 0.0)) < ADS_MAX_PITCH_DELTA ;

property Pitch_lr_ok =
  abs(AD_L.pitch.val - AD_R.pitch.val) < FCS_MAX_PITCH_SIDE_DELTA ;

property some_fgs_active =
  (FD_L.mds.active or FD_R.mds.active) ;

active_assumption: assume some_fgs_active ;

transient_assumption :
  assume AD_L_Pitch_Step_Delta_Valid and
  AD_R_Pitch_Step_Delta_Valid and Pitch_lr_ok ;

transient_response_1 :
  assert true -> abs(CSA.CSA_Pitch_Delta) < CSA_MAX_PITCH_DELTA ;

transient_response_2 :
  assert true ->
  abs(CSA.CSA_Pitch_Delta - prev(CSA.CSA_Pitch_Delta, 0.0)) < CSA_MAX_PITCH_DELTA_STEP ;
```
Example: Flight Control System

- Prove **transient response** property
  - The autopilot will not cause a sharp change in pitch of aircraft.
  - Even when one FGS fails and the other assumes control
- Given assumptions about the **environment**
  - The sensed aircraft pitch from the air data system is within some absolute bound and doesn’t change too quickly
  - The discrepancy in sensed pitch between left and right side sensors is bounded.
- and guarantees provided by **components**
  - When a FGS is active, it will generate an acceptable pitch rate
- As well as **facts** provided by design patterns
  - Leader selection: at least one FGS will always be active, and transition time on failure is bounded

```
transient_response_2 : assert true ->
  abs(CSA.CSA_Pitch_Delta - prev(CSA.CSA_Pitch_Delta, 0.0)) < CSA_MAX_PITCH_DELTA_STEP;
```
Compositional reasoning and patterns

• Guarantees provided by pattern are encoded as facts
• Attached at pattern instantiation
  – Model-independent
  – Assumptions
  – Pre/post-conditions
• Describe relationships between several components
  – In this example, the Leader and Valid fields for the left and right FGSs

```plaintext
pattern_instance Leader_Select_1 :  
  -- sync single-step delay between elements
  assume single_step_delay_comm(FGS_L, FGS_R);
  assume single_step_delay_comm(FGS_R, FGS_L);
  -- All non-failed nodes agree on who is the leader
  leader_agreement:
    assert (FGS_L.LSO.Valid and FGS_R.LSO_Valid) =>
      FGS_L.LSO.Leader = FGS_R.LSO.Leader;
  -- If a node fails, leadership is transferred to a non-failed node
  leader_transfer_1:  
    assert (prev(not(FGS_L.LSO.Valid), false) =>
      (FGS_R.LSO.Valid => FGS_R.LSO.Leader != Get_Property(FGS_L, Leader_Select_ID)));
  leader_transfer_2:
    assert prev(not(FGS_R.LSO.Valid), false) =>
      (FGS_L.LSO.Valid => (FGS_L.LSO.Leader != Get_Property(FGS_R, Leader_Select_ID)));
  -- If any non-failed nodes exist, one of them will be the leader
  leader_existence:
    assert (prev(FGS_L.LSO.Valid or FGS_R.LSO.Valid, false)) =>
      (FGS_L.LSO.Valid => (FGS_L.LSO.Leader >= 1 and FGS_L.LSO.Leader <= 2)) and
      (FGS_R.LSO.Valid => (FGS_R.LSO.Leader >= 1 and FGS_R.LSO.Leader <= 2));
  -- If the leader does not fail, it shall remain the leader.
  leader_persistence_1: assert
    (prev(FGS_L.LSO.Valid and
      FGS_L.LSO.Leader = Get_Property(FGS_L, Leader_Select_ID), false) =>
      (FGS_L.LSO.Valid =>
        FGS_L.LSO.Leader = Get_Property(FGS_L, Leader_Select_ID)));
  leader_persistence_2: assert
    (prev(FGS_R.LSO.Valid and
      FGS_R.LSO.Leader = Get_Property(FGS_R, Leader_Select_ID), false) =>
      (FGS_R.LSO.Valid =>
        FGS_R.LSO.Leader = Get_Property(FGS_R, Leader_Select_ID)));
end pattern_instance Leader_Select_1 ;
```
Current implementation

- **AGREE**
  - Assume-Guarantee Reasoning Environment
  - Eclipse/OSATE plug-in
- PSL contracts are just an AADL string property
  - `PSL_Properties::Contract => "..."`
  - This is where we need an annex
- AGREE generates verification conditions based on contracts and model structure
- Translated to LUSTRE
- Analyzed by KIND model checker
- Counterexamples in table format