Formal Verification of Intelligent Systems Modeled as Decision Procedures

Siddhartha Bhattacharyya, Thomas C. Eskridge and Marco Carvalho
Motivation

• Autonomous agents are controlling or coordinating autonomous systems to autonomously execute missions in battle space, civil airspace, cyber space

• Autonomous agents can be designed as
  – Cognitive architecture (Soar, ACT-R)
  – Perception, production system, memory

• Autonomous agents need to be rigorously analyzed to guarantee satisfaction of requirements, correctness to build trust on them
Research Challenges

• Cognitive architecture provides a simulation environment but lacks rigorous analytical capability
  – Translation into formal environment enables analytical capability
• Addressing the differences in the cognitive model and formal verification
  – Lack of visibility into algorithmic methods
  – Use of complex constructs
  – Dynamic nature of autonomy (rules modified at runtime)
Cognitive Model with Formal Verification Flow

- Requirements are coded into a Soar cognitive agent
- Agent is transformed into formal environment for verification
  - Generates runtime monitors
  - Corrects the present design
- Soar agent can learn efficient ways
  - Creates or modifies rules which are evaluated and/or verified
Cognitive Architecture

• Agent architecture
  – Integration of several components
• Perception
• Memory
• Production systems (decision procedures)

Ref: http://educatech.sytes.net/wiki/Soar
Soar Processing Cycle

- Proposed Soar processing cycle (refs: 3)

- Proposed soar processing cycle
- Generic representation
  - Any rule that is true can be executed
    - Satisfies diverse range of cognitive models
Uppaal a Real Time Verification Tool

- Modeling, validation and verification of real-time systems modeled as networks of timed automata, extended with data types (bounded integers, arrays, etc.)
  - Editor
  - Simulator
  - Verifier
Translation from Cognitive Model to Uppaal

Challenges in translation:
• Architectural Integrity
• Rule execution formalisms
• Cognitive engine flow
Soar Parsing for Translation

1. Create Antlr grammar for Soar
2. Generate the Soar parser
3. Create the data structure
4. Generate the xml for Uppaal

sp {counter*propose*initialize-counter
   (state <s> ^superstate nil
        ^name)

   -->
   (<s> ^operator <o> +)
   (<o> ^name initialize-counter)

}
Scheduler: Maintaining Generic Processing Cycle

- Notion of implementing a scheduler that executes a more generic representation
- Do not need to differentiate between propose, apply and other phases
- The satisfaction of the precondition selects the rule to be executed

![Diagram showing the processing cycle with a rule to be run and a check phase with a negation of the goal](image)
Mapping Soar to Uppaal with Counter

```plaintext
sp {counter*propose*initialize-counter
  (state <s> ^superstate nil
    ^name)
  -->
  (<s> ^operator <o> +)
  (<o> ^name initialize-counter)
}

sp {counter*apply*initialize-counter
  (state <s> ^operator <op>)
  (<op> ^name initialize-counter)
  -->
  (<s> ^name counter
    ^num 2)
```

```plaintext
Run_Rule?
s_superstate == nil && s_name == nil
  s_operator_o_name = initialize_counter,
  s_superstate = not_nil

Run_Rule?
  s_operator_o_name == initialize_counter
  s_name = counter, s_num = 2
```

```
Run_Rule?

Run
```
Mapping Soar to Uppaal with Counter (contd.)

```c
sp {counter*propose*increment
  (state <s> ^name counter
   ^num <c>)}

  -->
  (<s> ^operator <op> + =)
  (<op> ^name increment
   ^first <c>)}

sp {counter*apply*increment
  (state <s> ^name counter
   ^operator <op>
   ^num <c>)}

  (<op> ^name increment
   ^first <c>)}

  -->
  (<s> ^num (+ <c> 1)
   ^num <c> -)}
```

```plaintext
Run_Rule?

s_name == counter
s_operator_o_name = increment,
  s_operator_o_first = s_num

Run_Rule?

s_name == counter &&
  s_operator_o_name == increment &&
  s_operator_o_first == s_num

s_num = s_num + 1
```
Properties checked:

For all paths eventually it reaches the goal: $A<> s_{num} == 7$

For all paths eventually is the number is larger than the specified: $A<> s_{num}>7$
Mapping Soar to Uppaal: Pilot Agent

Executes sequence of tasks for its mission: preflight checks, flight plan, file the plan, launch, navigate, refuel, land and reach destination
Mapping Soar to Uppaal: Pilot Agent (contd.)

Properties checked:
• All paths eventually lead to reaching destination
  \( A<> \text{Goal.Goal} \)
• Does there exist a point when next waypoint is not reached but the navigator says it has been reached
  \( E<> t< \text{Time_To_Next_Waypoint and Navigate}_0.\text{Run} \)
• Does there exist a condition where next waypoint is not reached but the UAV is trying to refuel
  \( E<> \text{Waypoint.Navigator}_0.\text{Run and Refuel}_0.\text{Run} \)
• Does there exist a path where fuel is not checked
  \( E[] \text{Check_Fuel == false} \)

**Uppaal’s new feature generates test cases to indicate coverage based on states and edges traversed through the properties checked**
Counterexample

Does there exist a point when next destination is not reached but the navigator says it has been reached

**Property spec:** $E<> t< \text{Time\_To\_Next\_Waypoint}$ and $\text{Navigate\_0.Run}$

**Correction:** adding guard $\text{Check\_Fuel} = \text{false}$
Conclusion and Future Work

• Developed automated translator from Soar to Uppaal
• Performed formal verification of cognitive model designed in Soar
• Method can be extended to other similar cognitive models with appropriate modifications
• Extend the translator to handle other relevant constructs in cognitive models
• Evaluate the translation going back from Uppaal to Soar
• Extend the framework to integrate learning and the associated verification
References


2. Enhancing autonomy with Trust, S. Bhattacharyya, J. Davis, M. Matessa et al. AUVSI 2015.

3. Extending the soar cognitive architecture, J. E. Laird, 2008 Proceeding AGI


5. www.uppaal.org