Examining the CARA Specification

Elsa L Gunter,

Yi Meng

NJIT

Capturing Tagged Req As LTL Spec

- Goal: Express tagged requirements as LTL formulae to enable model checking
- LTL not expressive enough, so we must approximate
- Choice A: Give weaker statements and prove that those weaker statements hold
  - » Know that is doesn't guarantee that the original requirements hold

Capturing Tagged Req As LTL Spec

- Choice B: Give stronger statements
  - » Not always possible (may force the new requirements to always be false)
  - » May rule out desired implementations
- Choice C: Give statements that may be weaker, but which when combined with domain knowledge implies the original requirements

Limitations of LTL for Composite Systems

- Can only specify that something happens now, next, eventually or always (and until)
- Next usually too strong – irrelevant actions in independent parts of system can intervene
- Generally forced to use Eventually when something stronger is wanted

- Can't adequately relate values from one point in time to another

Example

8: The CARA will monitor the Air OK line whenever the pump is plugged in

8.1 If the Air OK signal remains low for 10 seconds

8.1.2 A level 1 alarm is issued

Example

- This roughly becomes
- Always ((PlugIN = true /\ AirOK >= limit and Next (timer = 0 and (AirOK < limit Until timer >= 10))) implies Eventually (timer >= 10 and AirOK_Alarm = 1))
- Relies heavily on domain knowledge
  - » Values for timer increments, reflecting "true" passage of time in seconds

Example

- Correctness depends on visibility and control of variables
- System controlled: AirOK_Alarm
- Environment controlled, system visible: Plugin, AirOK
- Envoronment controlled, system hidden: timer
- Constant: limit

Modeling Environment

- Correctness depends on valid modeling of required domain knowledge
  - » Verification?
- Typically need new model for each new modeling language for the system
- Each model should be motivated by the requirement, not the implementation
  - » Only expressed using environment controlled variables and system variables exported

Specification Analysis:

- LTL Spec can be checked with model checker/thoemr prover such as Spin, Pet, Dove
- Must input functional model based on tagged requirements (EFSMs)
- Must also input approximate functional model of system environment
  - » Thereby capturing needed domain knowledge
- Prove every behavior of combined functional satisfies the formal logical statements

Upshot

- Captured most of tagged requirements by approximate LTL formulae (using combination of Choice A and Choice C)
- Found many places where more than one interpretation was possible
- Ongoing work on checking EFSM specification against LTL formulae in Pet and Dove
- Need modular approach to avoid state space explosion

Formalizing Original Spec

- Reasons:
  - To facilitate checking the specification for self-consistency
  - To allow checking other forms of specification against the tagged requirements

DOVE

DOVE is a tool built on theorem prover Isabelle to construct, simulate and prove LTL properties of Finite State Machines

Graphical interface for building FSMs

Isabelle used for proving properties

Developed by DSTO in Australia

CARA in DOVE

Work with Yi Meng

Translate CARA EFSM Spec into DOVE

Translate LTL formulae into DOVE

Prove they hold (or rather find that they don’t

Difficulty: Needed to add support for composing FSMs in DOVE

PET

- Path Evaluation Tester – Elsa Gunter and Doron Peled
- Based on translating code to control flow graphs
- Combines automatic theorem proving in HOL with model checking
- Programs input in simple concurrent language, compiled to visual flow graphs
- Model checking based LTL fomulae

Findings

- Been able to prove some requirements hold of Penn EFSM spec
- Found collections of places where it does not
  - » Cause is typically raise conditions
- Discrepancies tend to highlight questionable aspects of requirements