SpeAR: A Framework for Specifying and Analyzing Requirements

What is SpeAR?

SpeAR is a functional requirements development framework with the following features:

- A set of frequently used specification patterns that can be translated into formal notations.
- A domain specific language for specifications that addresses system (and subsystem) interfaces, assumptions, and behaviors.
- Constraint-based analysis of these formal representations against a set of properties that validate the requirements set.

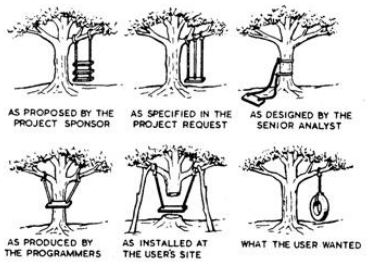
What isn't SpeAR?

Using SpeAR successfully will require the user to develop their specifications at a high level of abstraction.

- Characterizing a system in a high level of detail (such as precise timing mechanisms) is not supported by the supported formalisms.
- Analysis depends on model checking so there is an inherent scalability limitation.

Why Formalize and Analyze Requirements?

Requirements are the main vehicle through which we describe the desired behaviors of complex, critical systems that we build.



Modern systems are increasingly complex, generating thousands of requirements. They are difficult to interpret, and the ambiguous nature of natural language leads to errors.

Background

SCR Tool

Expresses requirements as relations between monitored (inputs) and controlled (locals, outputs) variables. System assumptions or environmental constraints are also captured.

Features include:

- Expressing requirements in a tabular format that can be used to check for completeness.
- A simulator to test and exercise the specification.
- Translating the specification to the SPIN model checker for formal analysis of invariant properties.
- Translation to PVS theorem prover for analysis using Timed Automata theories.
- Source code generation
- Test case generation

RAT

The Requirements Analysis Tool (RAT) accepts a set of requirements ***R*** expressed in PSL and performs the following analyses:

- Checks if ***R*** is strict (unexpected behaviors are ruled out) by proving that a set of assertions ***A*** (properties that are expected to hold on the behaviors described by ***R***) is logically entailed by ***R***.
- Checks if ***R*** is permissive (interesting or intended behaviors are allowed to happen) by checking if a set of possibilities ***P*** are compatible with ***R.***
- Checks if ***R*** is realizable.

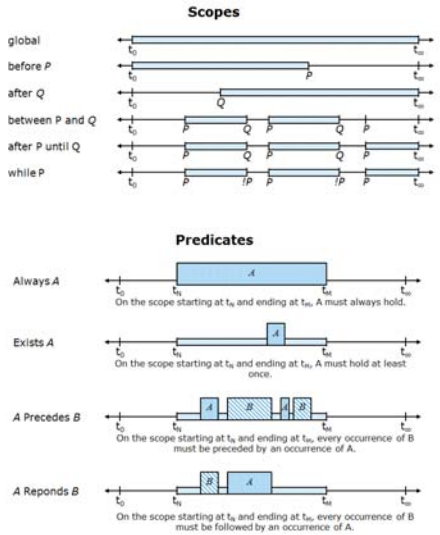
Features:

- Uses BDD-based model checking (NuSMV) to perform analyses.
- Requirements are specified using PSL.

Approved for public release; distribution is unlimited.
Case Number 88ABW-2014-1944, 24 April 2014.

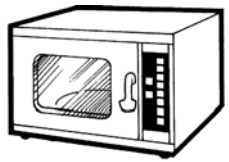
Kansas State Specification Patterns

Kansas State University developed formalisms for commonly used design patterns. Each pattern is the result of pairing a scope and predicate to express a temporal relationship between conditions/events in a system.

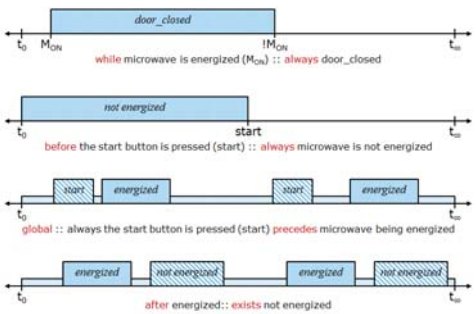


Microwave Oven

To make the concepts concrete, consider the example of a microwave oven. It has a start and clear button, number keys, and can be energized (cooking) or de-energized.

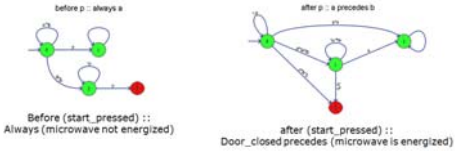


Example Microwave Oven Patterns



Representing Requirements for Analysis

Each requirement is provided as a pattern. We translate these patterns into Lustre synchronous observers. These observers were proven equivalent to the published LTL formulas using model checking. Shown below are two synchronous observers that capture the before/always pattern and the after/precedes pattern. The green states denote accepting states, the red states failed states, and the grey states are non-accepting.



Before (start_pressed) ::
Always (microwave not energized)

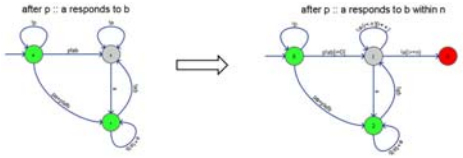
after (start_pressed) ::
Door_closed precedes (microwave is energized)

Handling Liveness

Our chosen analysis tool, JKind, cannot check liveness properties. Some scope/predicate pairings represent a liveness condition. These pairings are:

- global/exists
- global/responds
- after(until)/exists
- after(until)/responds

In these cases, JKind is not able to reason about the pattern. We handle this restriction by bounding the infinite nature of a liveness pattern, turning it into a safety property that is checkable.



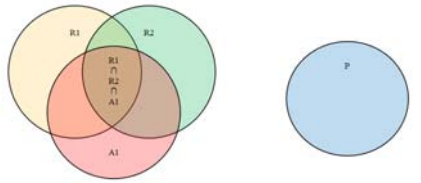
Domain Specific Language

SpeAR specifications are captured in a Domain Specific Language (DSL) that requires an explicit interface, assumptions, and requirements definitions.

The structure of the DSL can be used to check common mistakes, such as references to undefined signals, units disagreement, and improper usage of variables. It also provides support for compositional specifications.

How the analysis in SpeAR works

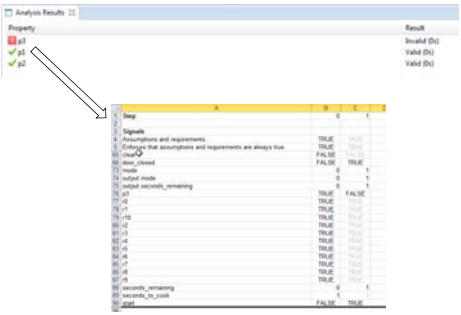
The intersection of the set of traces that each of our requirements accept defines the set of traces our system accepts.



If a property P accepts the set of traces that our system accepts, then our system satisfies P.

Analysis Results

Analysis results are provided to the user in a graphical interface. Each property is shown to either accept all traces that the requirements and assumptions accept, or a trace that it doesn't accept is given. In this case the user can use this information to understand the discrepancy.



Download SpeAR, it's open source!

http://www.github.com/afifarek/spear

Check out the releases tab, or build from source using Eclipse.

Contact

Lucas Wagner
lgwagner@rockwellcollins.com

Aaron Fifarek
aaron.fifarek@linquest.com

Dan DaCosta
dacosta@cs.umn.edu

Kerianne Gross
kerianne.gross@us.af.mil

Rockwell Collins