# Specification and Analysis of Requirements (SpeAR) Tool

Lucas Wagner
lgwagner@rockwellcollins.com

Aaron Fifarek
Aaron.fifarek@linquest.com

Kerianne Gross
Kerianne.Gross@us.af.mil

Rockwell Collins



Approved for Public Release, Distribution Unlimited Case #88ABW-2014-1944

## What is SpeAR?

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

- Formalisms to help users express requirements unambiguously.
- Domain-specific language to capture the contextual information.
- Constraint based analysis of formalized requirements against a set of validating properties.

## What isn't SpeAR?

- A detailed design tool.



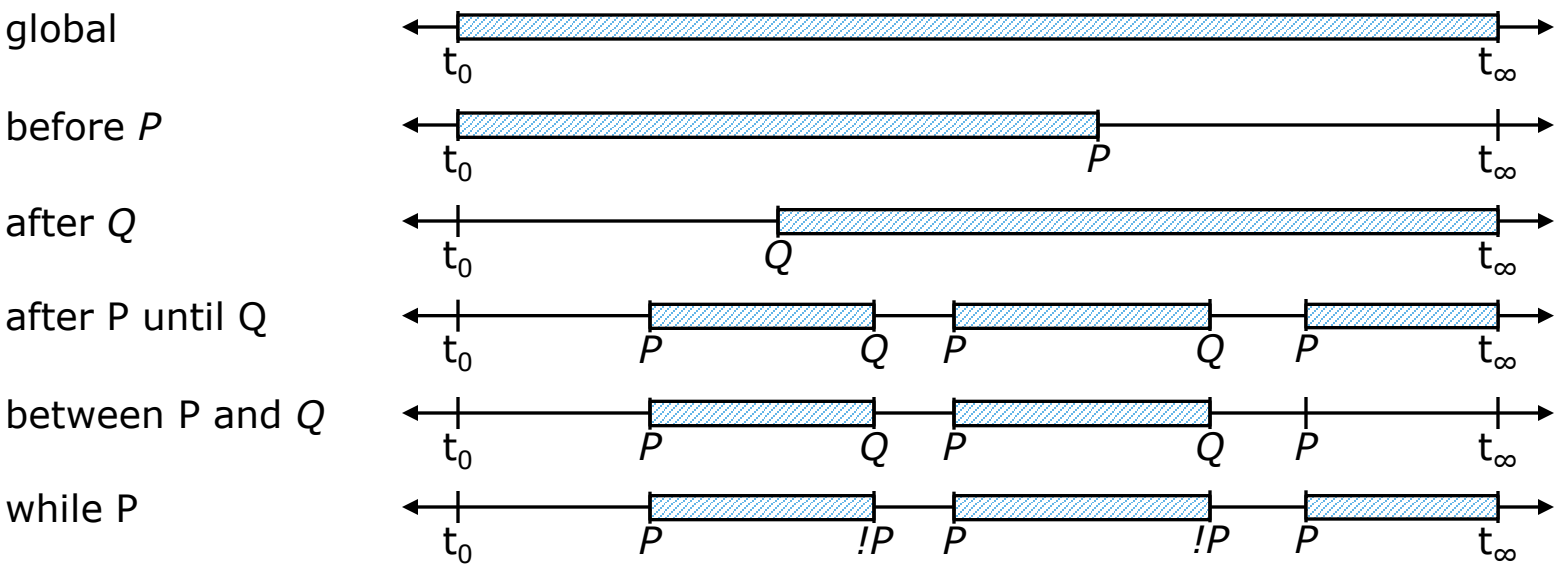
Approved for Public Release, Distribution Unlimited Case #88ABW-2014-1944

# Formalizing Requirements in SpeAR

Spear formalizes requirements using the specification patterns developed at Kansas State University.

http://patterns.projects.cis.ksu.edu/

## Scopes

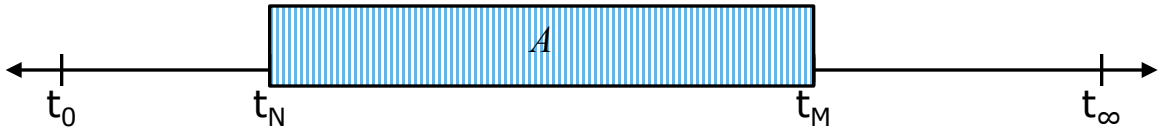





Approved for Public Release, Distribution Unlimited Case #88ABW-2014-1944

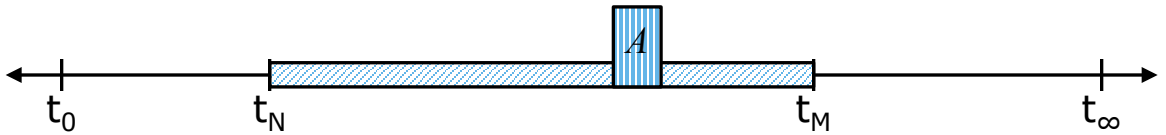
# Predicates

always *A*



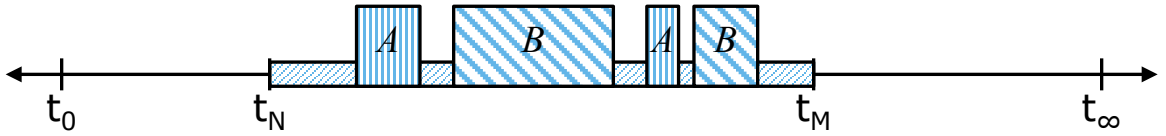

On the scope starting at $[t_N, t_M)$ A must always hold.

exists *A*



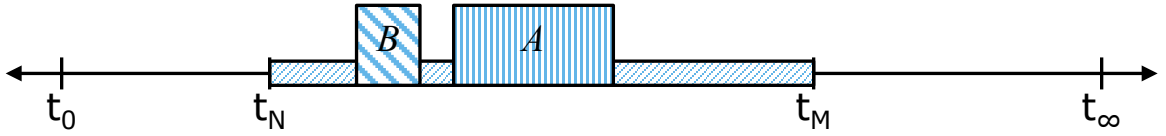

On the scope $[t_N, t_M)$ A must hold at least once.

*A* precedes *B*




On the scope $[t_N, t_M)$ every occurrence of B must be preceded by an occurrence of A.

*A* reponds *B*




On the scope $[t_N, t_M)$ every occurrence of B must be followed by an occurrence of A.



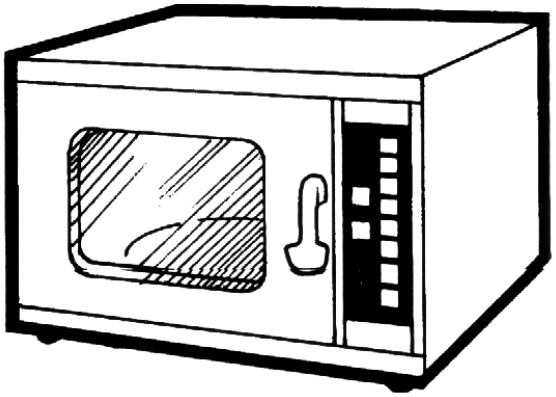
Approved for Public Release, Distribution Unlimited Case #88ABW-2014-1944

# Microwave Example

To illustrate the use of these patterns consider the example of a microwave oven. Describing the requirements of a microwave oven can be done in SpeAR.

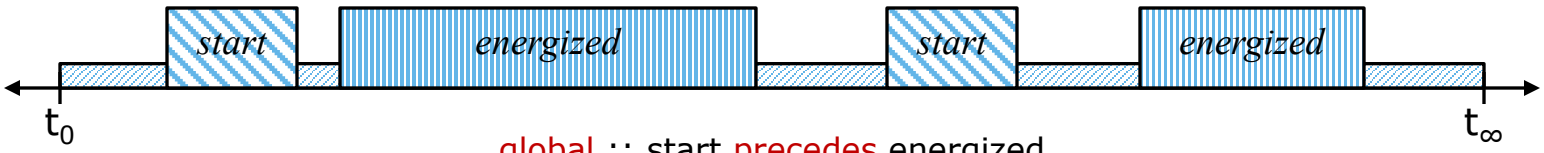
***Inputs***

start
clear
door closed

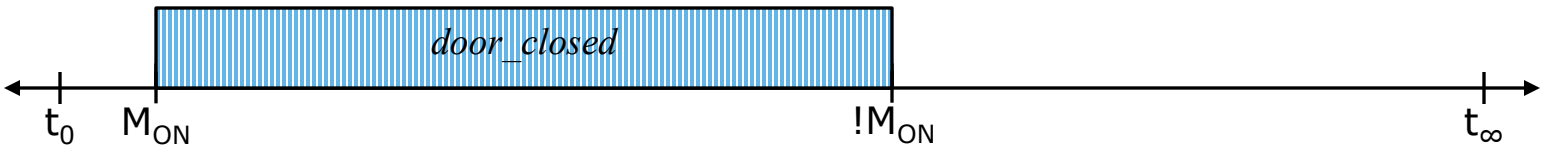


***Outputs***

energized
time_remaining




global :: start precedes energized




while energized ($M_{ON}$) :: always door_closed



Approved for Public Release, Distribution Unlimited Case #88ABW-2014-1944

# Domain Specific Language

SpeAR specifications are captured in a a domain specific language that requires explicit interface, assumptions and requirements definitions.

mode_logic.spear | equals.spear

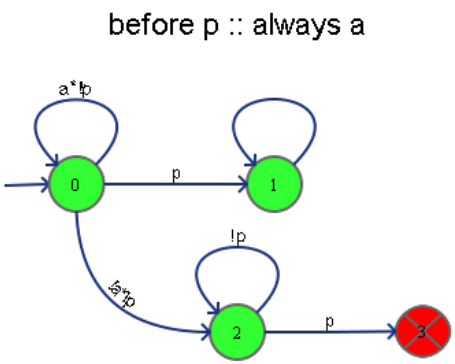
```
package microwave.mode_logic;

procedure mode_logic(clear,start,door_closed,seconds_to_cook) returns (output);

import microwave.definitions.*;
import microwave.definitions.mode_type;
import microwave.definitions.mode_type.*;
import microwave.definitions.modelogic_output_type;

Units:
    seconds;

State:
    clear : bool;
    start : bool;
    door_closed : bool;
    seconds_to_cook : int seconds;

    seconds_remaining : int;
    mode : mode_type;

    output : modelogic_output_type;

Requirements:
    a0 = global :: always (second_to_cook >= 0 seconds);

    r0 = initial :: (mode == SETUP) and (seconds_remaining == 0 seconds);

    r1 = after (pre mode == SETUP) :: (mode == SETUP) responds (not start or clear or seconds_to_cook <= 0 seconds);
    r2 = after (pre mode == SETUP) ::
        ((mode == COOKING) and (seconds_remaining == seconds_to_cook)) responds ( start and not clear and door_closed and seconds_to_cook > 0 seconds);
    r3 = after (pre mode == SETUP) ::
        ((mode == SUSPENDED) and (seconds_remaining == seconds_to_cook)) responds (start and not clear and not door_closed and seconds_to_cook > 0 seconds);

    r4 = after (pre mode == COOKING) :: ((mode == COOKING) and seconds_remaining == pre seconds_remaining - 1 seconds) responds
        ((pre mode == COOKING) and (door_closed and not clear and seconds_remaining > 0 seconds)) ;
    r5 = after (pre mode == COOKING) :: (mode == SUSPENDED) responds (not door_closed or clear and (seconds_remaining > 0 seconds));
    r6 = after (pre mode == COOKING) :: (mode == SETUP) responds (seconds_remaining <= 0 seconds);

    r7 = after (pre mode == SUSPENDED) :: (( mode == SUSPENDED ) and (pre seconds_remaining == seconds_remaining)) responds
        (not start or not door_closed and not clear);
    r8 = after (pre mode == SUSPENDED) :: (mode == COOKING) responds (start and door_closed and not clear);
    r9 = after (pre mode == SUSPENDED) :: (mode == SETUP) responds clear;

    r10 = global :: always (output == modelogic_output_type {mode = mode, seconds_remaining = seconds_remaining});

Properties:
    p1 = while (mode == COOKING) :: always door_closed;
    p2 = before start :: always (mode == SETUP);
    p3 = while ((mode == COOKING) or (mode == SUSPENDED)) :: always (seconds_remaining <= pre seconds_remaining);
    p4 = global :: always (mode <> COOKING);
```

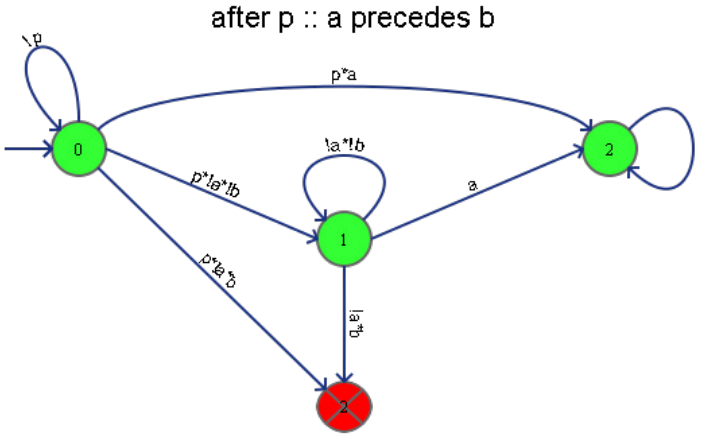


Approved for Public Release, Distribution Unlimited Case #88ABW-2014-1944

# Representing the Patterns in Lustre for Analysis

Each property is translated into a finite state machine. The green states denote accepting states, the red states failed states, and the grey states are non-accepting.



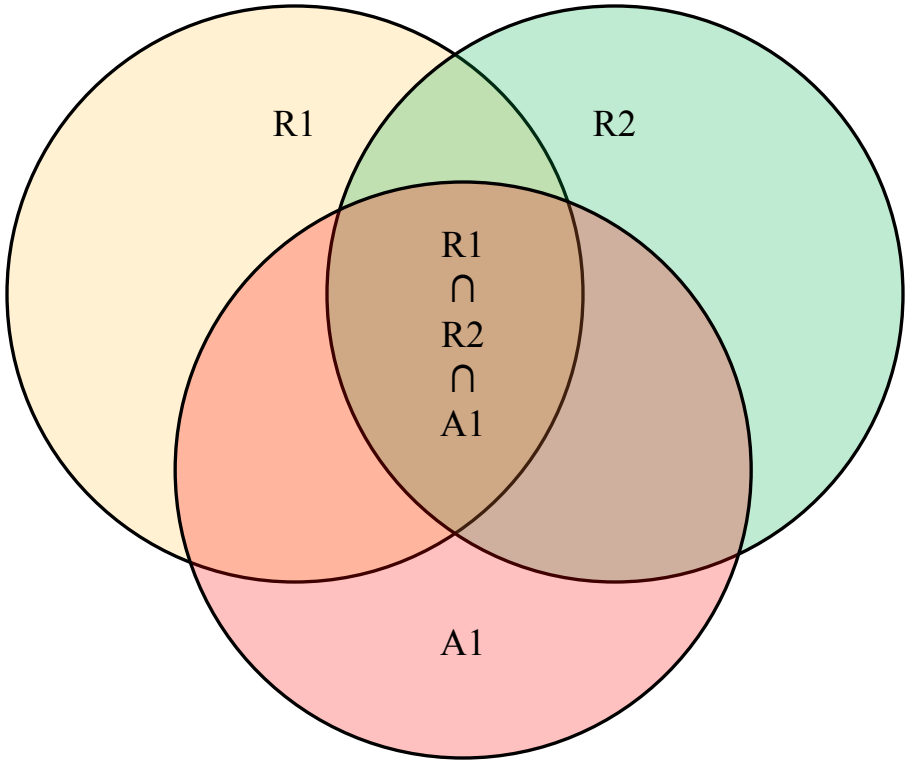






Approved for Public Release, Distribution Unlimited Case #88ABW-2014-1944

# Analysis in SpeAR

The intersection of the set of traces that our requirements accept defines the set of traces our system accepts. If a property P also accepts that set of traces, the system satisfies the property.



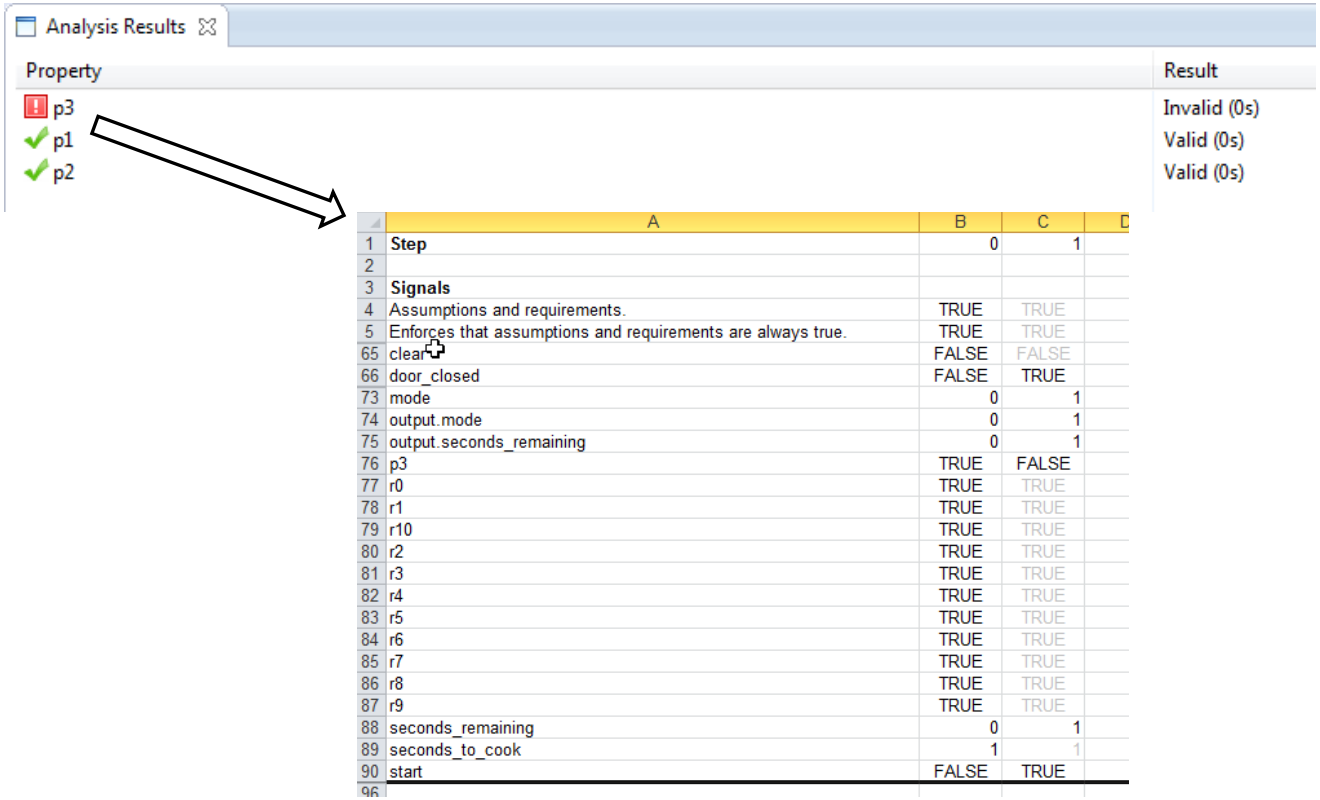

If a property P also accepts that set of traces our system satisfies that property.



Approved for Public Release, Distribution Unlimited Case #88ABW-2014-1944

# Counterexamples

Analysis results are provided to the user in a graphical interface. If a property is violated by the requirements a trace is provided to the user for debugging.



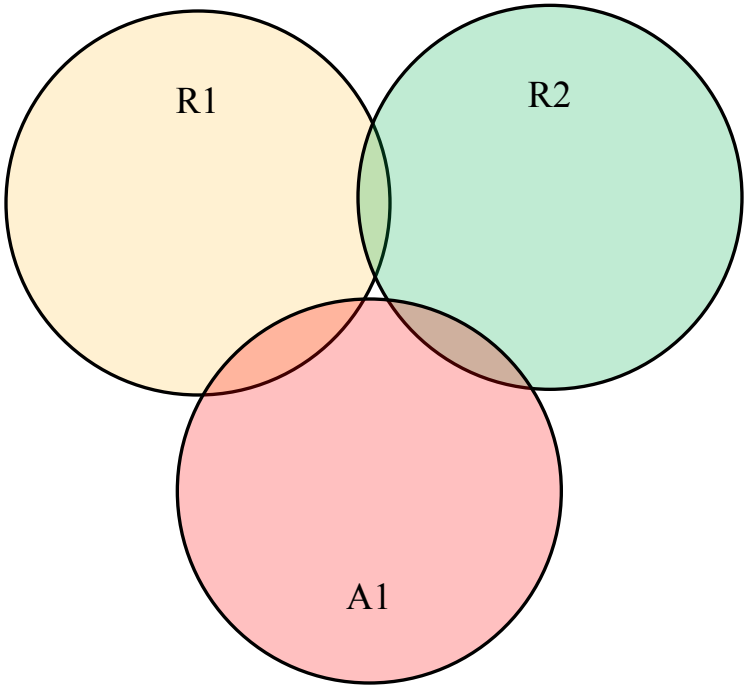

| | A | B | C | D |
|---|---|---|---|---|
| 1 | **Step** | 0 | 1 | |
| 2 | | | | |
| 3 | **Signals** | | | |
| 4 | Assumptions and requirements. | TRUE | TRUE | |
| 5 | Enforces that assumptions and requirements are always true. | TRUE | TRUE | |
| 65 | clear | FALSE | FALSE | |
| 66 | door_closed | FALSE | TRUE | |
| 73 | mode | 0 | 1 | |
| 74 | output.mode | 0 | 1 | |
| 75 | output.seconds_remaining | 0 | 1 | |
| 76 | p3 | TRUE | FALSE | |
| 77 | r0 | TRUE | TRUE | |
| 78 | r1 | TRUE | TRUE | |
| 79 | r10 | TRUE | TRUE | |
| 80 | r2 | TRUE | TRUE | |
| 81 | r3 | TRUE | TRUE | |
| 82 | r4 | TRUE | TRUE | |
| 83 | r5 | TRUE | TRUE | |
| 84 | r6 | TRUE | TRUE | |
| 85 | r7 | TRUE | TRUE | |
| 86 | r8 | TRUE | TRUE | |
| 87 | r9 | TRUE | TRUE | |
| 88 | seconds_remaining | 0 | 1 | |
| 89 | seconds_to_cook | 1 | 1 | |
| 90 | start | FALSE | TRUE | |
| 96 | | | | |



Approved for Public Release, Distribution Unlimited Case #88ABW-2014-1944

# Realizability

If there is no set of traces that satisfy the intersection of the requirements set, the set is not realizable. SpeAR (in-progress) will be able to identify unrealizable specifications.




If there is no set of traces that all the requirements and assumptions accept, the system is not realizable.



Approved for Public Release, Distribution Unlimited Case #88ABW-2014-1944

## Download Spear, it's open source!

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

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

## Questions?