NAVAL
POSTGRADUATE
SCHOOL

Identifying Decision Patterns Using Monterey Phoenix

presented to

Office of the Deputy Assistant Secretary of Defense, Systems Engineering

System of Systems Engineering
Collaborators Information Exchange (SoSECIE)

21 August 2018

authored by

John J. Quartuccio

PhD candidate, Systems Engineering Department

jjquartu@nps.edu

Monterey, California

WWW.NPS.EDU

Previous Publications

- This presentation is based on content previously published, as follows:
  - *John Quartuccio, Kristin Giammarco, and Mikhail Auguston, presented by Thomas Moulds. Identifying decision patterns using Monterey Phoenix. In System of Systems Engineering (SoSE), IEEE 12th International Conference, 2017.*
  - *John Quartuccio, Kristin Giammarco, and Mikhail Auguston, presented by Thomas Moulds. Deriving probabilities from behavior models defined in Monterey Phoenix. In System of Systems Engineering (SoSE), IEEE 12th International Conference, 2017.*
  - *John Quartuccio and Kristin Giammarco. A model-based approach to investigate emergent behaviors in systems of systems. In Larry Rainey and Mo Jamshidi, editors, Engineering Emergence: A Modeling and Simulation Approach. CRC Press, 2018. Publication pending.*

Motivation

- System of System Architectures readily capture the *intended* interactions within the context boundary
  - UML/SysML outlines a means to document system behaviors (ref: https://www.omg.org/)
    - Activity diagrams
    - Sequence diagrams
    - State-space diagrams
    - Use-case diagrams
- *What happens when things go wrong?*
  - Identify a way to capture both the desired behaviors and the undesirable behaviors of systems?

Outline

- Identification of patterns
  - Topology
  - Semantics
- Behaviors and a proposed analysis method
- Decision model example
  - Narrative
  - Interactions
  - Constraints
  - Analysis
    - Probability of a trace
    - N-squared diagram of all traces
- Wrap up and discussion

Why Conduct Behavior Analysis?

- Logical analysis at a high level of abstraction
  - Derived from the essence of a behavior – hierarchical and temporal aspects of an interaction
  - Considers the fundamental interactions of the system, both internal and external, but described separately
  - Conducted prior to high cost investment in detailed design
    - Prior to detailed modeling of discrete event, agent-based, physics-based, or hybrid models
    - Prior to physical design and manufacture
  - Enables analysis of both human and machine interactions
- Typical system behavior architectures do not anticipate all possible outcomes, without intentional analysis
  - This problem becomes intractable without tools to help (30 sequential choices of two alternatives results in over 1.07 billion possible outcomes)
  - Derivation of constraints forms a level of requirements to constrain the system behavior to what is expected and desired
- Not intended for detailed considerations such as data through-put, physical performance, geographical or spatial reference

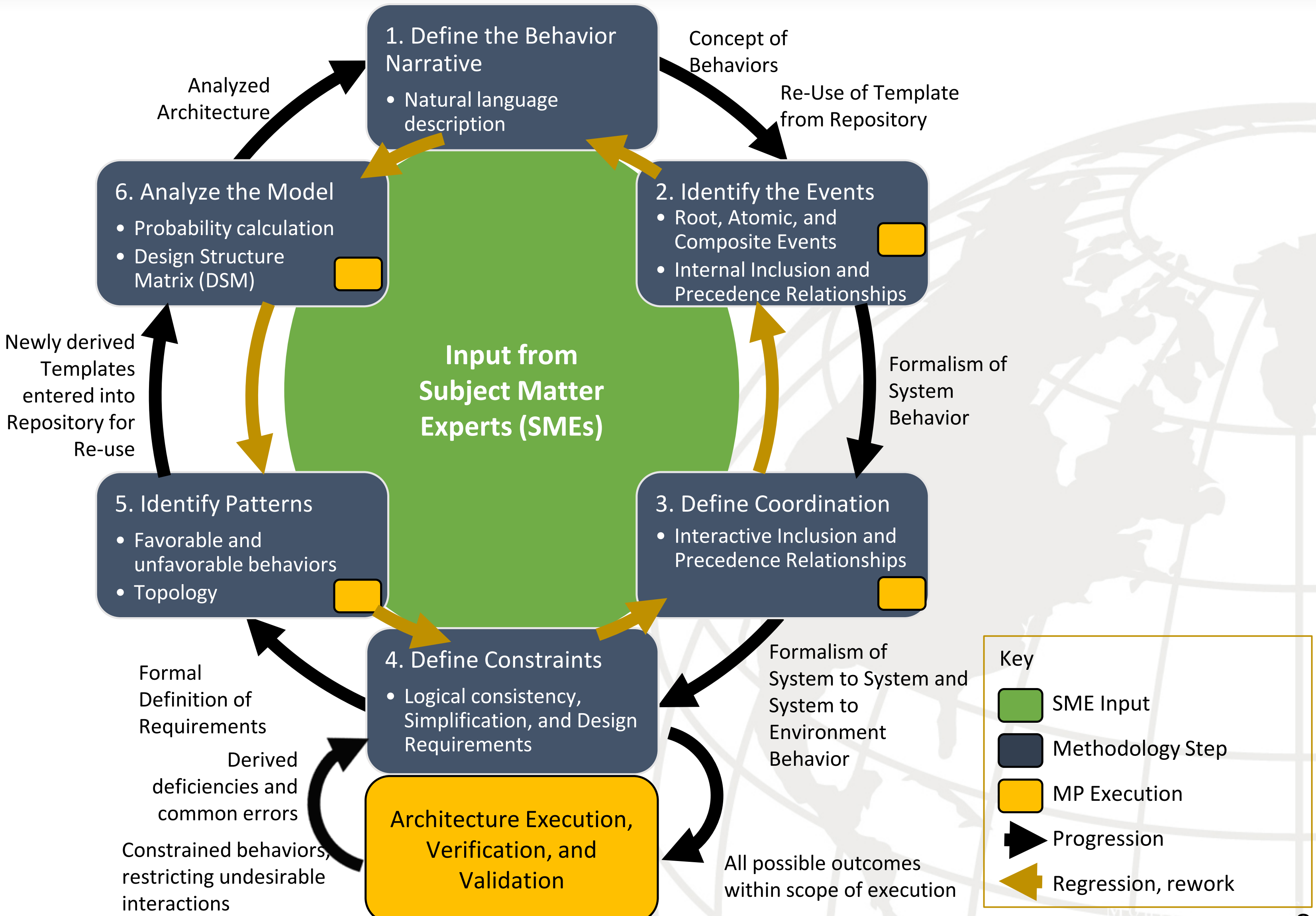
Why consider Patterns?

- Design patterns
  - Re-use of successful patterns
  - Limit or eliminate unwanted patterns
- Model checking
  - Logical consistency
    - Positive-patterns: send then receive, write then read, request then authorize, have fuel then take action, …
    - Anti-patterns: receive before send, read before write, authorize before request, take action without fuel, …
  - Discovery of inherent nature of the architecture
- Design analysis
  - Derive the probability of successful outcomes
  - Derive relative frequency of interactions, e.g. N-squared diagram
    - Well-traveled pathways
    - Rare occurrences
    - Modularity of closely related interactions
- Design of experiments
  - Interactions enable an opportunity for verification in test

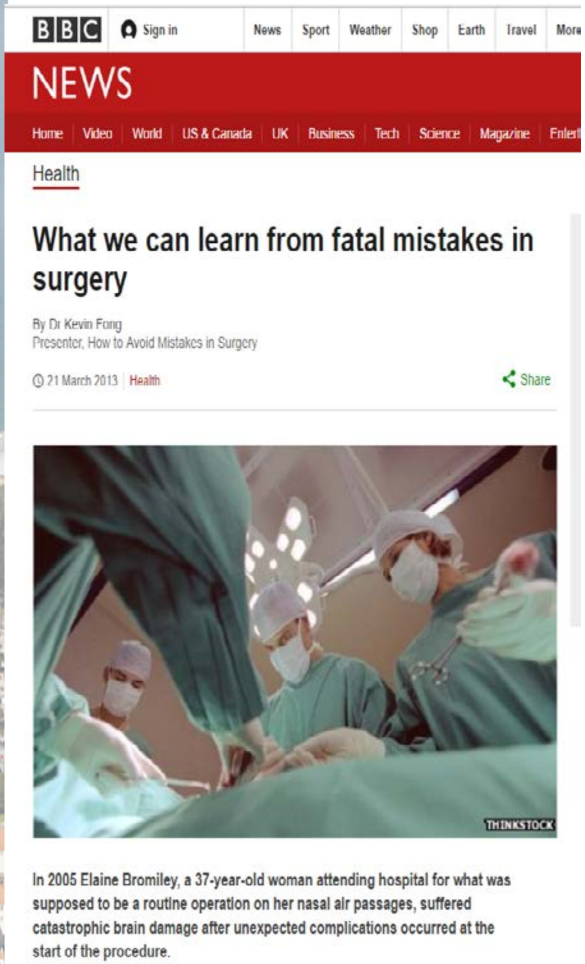
Monterey Phoenix (MP) Basics

- Based upon Small Scope Hypothesis (Jackson, 2012), such that most problems can be found with just a few iterations
- Behavior modeling platform that derives all possible combinations of behaviors, within the scope of execution
- Incorporates a concise language, employing principles of predicate logic
- Behaviors described as hierarchical *(inclusion)*, temporal *(precedence)*, or user-described
- Interactions *within* a system defined separately from interactions *among* systems
- Constraints limit the outcomes of unwanted behaviors and thereby establish a set of requirements for the system
- Attributes easily indicated in the model
  - favorable and unfavorable outcomes used in the example model
- Assertion checking provides a means to query the model, finding any occurrence of a pattern
- Available for anyone to use with the MP-Firebird Analyzer, at https://firebird.nps.edu

Methodology



Decision Pattern Example



BBC Sign in News Sport Weather Shop Earth Travel More

NEWS

Home Video World US & Canada UK Business Tech Science Magazine Entert

Health

What we can learn from fatal mistakes in surgery

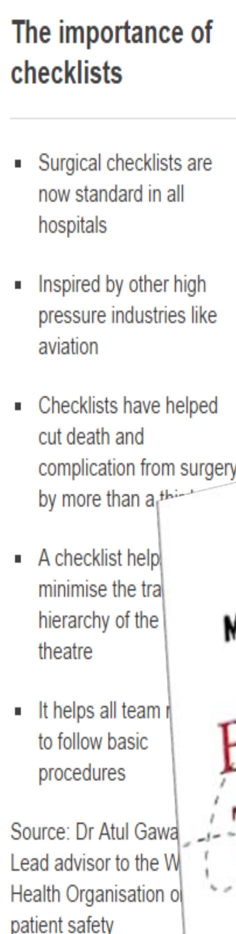
By Dr Kevin Fong
Presenter, How to Avoid Mistakes in Surgery

21 March 2013 Health

Share

THINKSTOCK

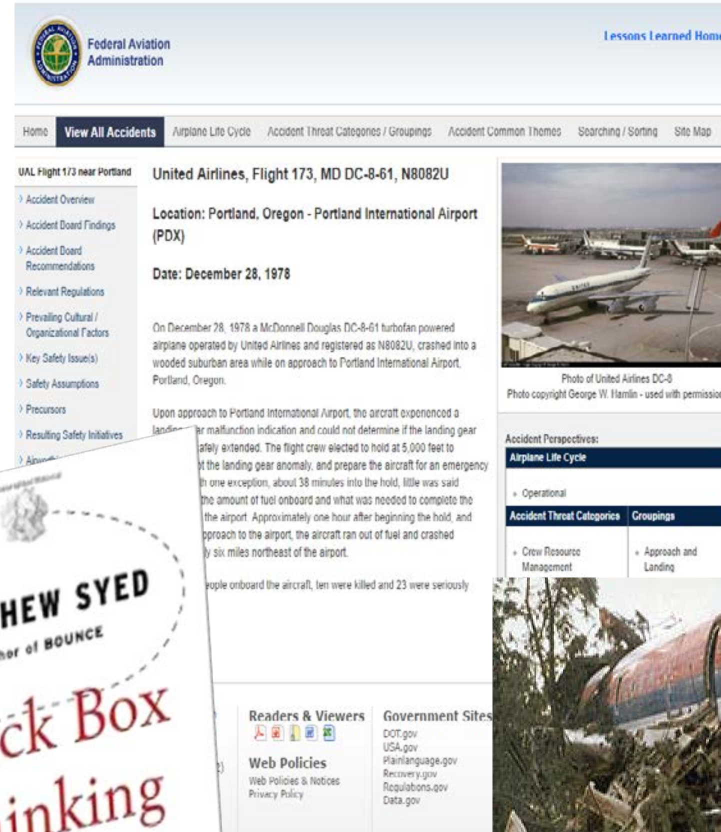
In 2005 Elaine Bromiley, a 37-year-old woman attending hospital for what was supposed to be a routine operation on her nasal air passages, suffered catastrophic brain damage after unexpected complications occurred at the start of the procedure.

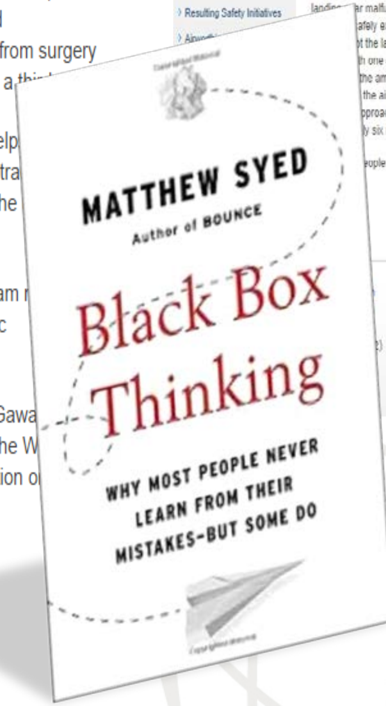


The importance of checklists

- Surgical checklists are now standard in all hospitals
- Inspired by other high pressure industries like aviation
- Checklists have helped cut death and complication from surgery by more than a thi
- A checklist help minimise the tra hierarchy of the theatre
- It helps all team to follow basic procedures

Source: Dr Atul Gawa
Lead advisor to the W
Health Organisation o
patient safety

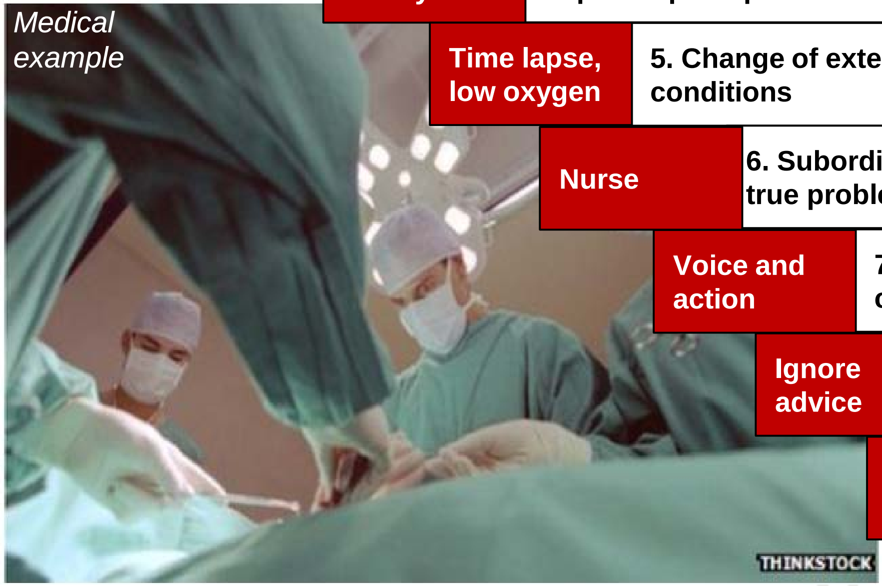




References:
Syed, M. (2015). *Black Box Thinking: the surprising truth about success*. John Murray.
Flight 173: http://lessonslearned.faa.gov/ll_main.cfm?TabID=1&LLID=42
BBC Article: http://www.bbc.com/news/health-21829540

The Behavior Narrative

Medical example	Pattern	Aviation example
Physician	1. Formal Hierarchy	Pilot
Surgery	2. Routine Procedure	Landing
Adverse reaction	3. Unexpected and unplanned occurrence	No gear indication
Clearing airway	4. Intense focus and impaired perception	Hold pattern
Time lapse, low oxygen	5. Change of external conditions	Time lapse, low fuel
Nurse	6. Subordinate recognizes true problem	Flight Eng
Voice and action	7. Attempt to communicate solution	Voice
Ignore advice	8. Leadership failure to recognize the problem	Ignore advice
Death of patient	9. Failure	Crash
Limited	10. Response	Full investigation



Pattern
Medical example
Aviation example

References:
Syed, M. (2015). *Black Box Thinking: the surprising truth about success*. John Murray.
Flight 173: http://lessonslearned.faa.gov/ll_main.cfm?TabID=1&LLID=42
BBC Article: http://www.bbc.com/news/health-21829540

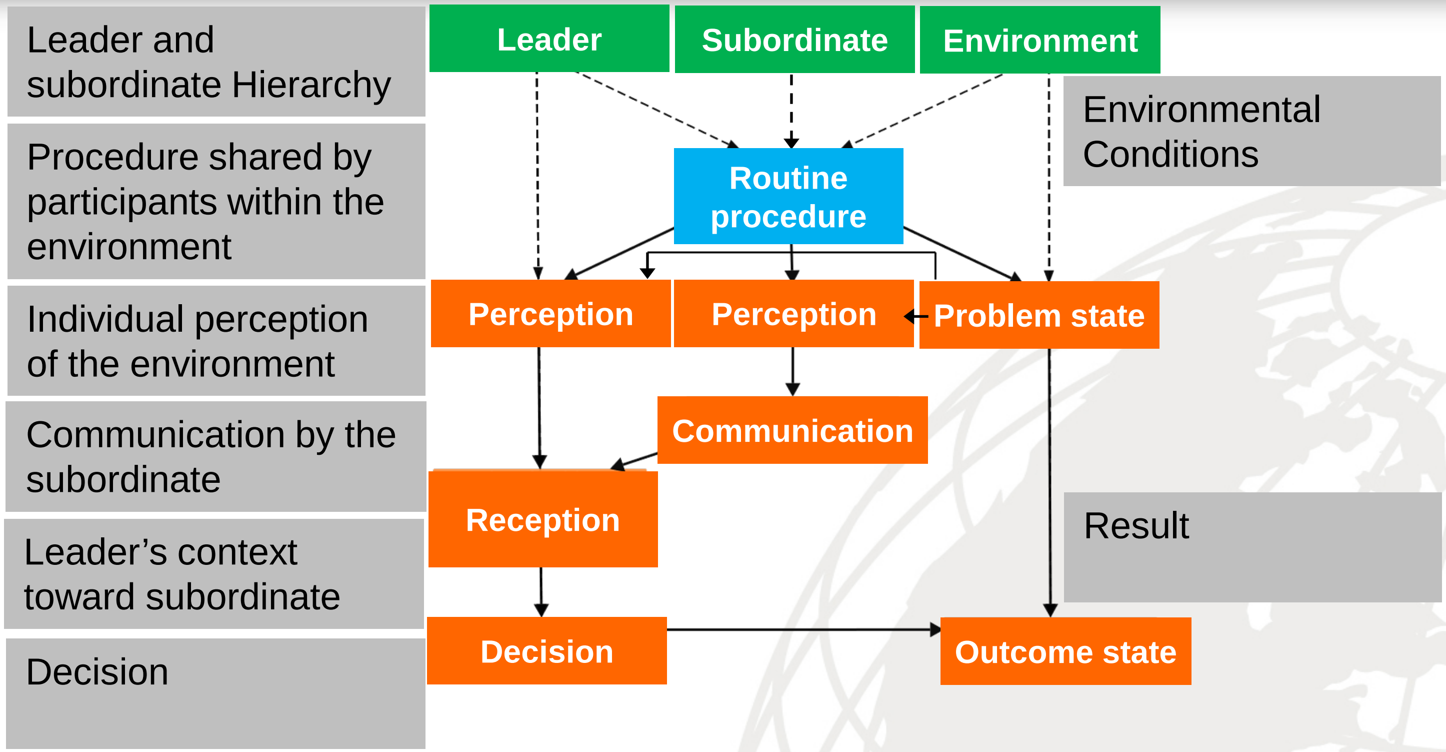
NAVAL POSTGRADUATE SCHOOL

The Behavior Events for Each Scenario

Surgery Scenario	Model Abstraction	Aircraft Mishap Scenario
Condition of the patient	**Environment**	Condition of the aircraft
Surgeon	**Leader**	Pilot
Nurse	**Subordinate**	Flight Engineer
Surgery	**Routine procedure**	Landing
Patient airway	**Problem state**	Fuel state
Surgeon Perception	**Perception**	Pilot Perception
Nurse Perception	**Perception**	Flight Engineer Perception
Nurse Speaking	**Communication**	Flight Engineer Speaking
Surgeon Processes Nurse's Voice	**Reception**	Pilot Processes Flight Engineer's Voice
Next Action	**Decision**	Next Action
Death or Survival	**Outcome state**	Mishap or Landing

Key
- Root event (green) - establishes a hierarchy
- Composite event (orange) - contains sub-events
- Atomic event (blue) - contains no sub-events

The Behavior Model in Monterey Phoenix



Key

- Root event (establishes hierarchy)
- Composite event (contains sub-events)
- Atomic event (contains no sub-events)
- Inclusion relationship
- Precedence relationship
- Note (not part of MP)

Precedence and Inclusion Relationships are shown as solid and dotted arrows, respectively.

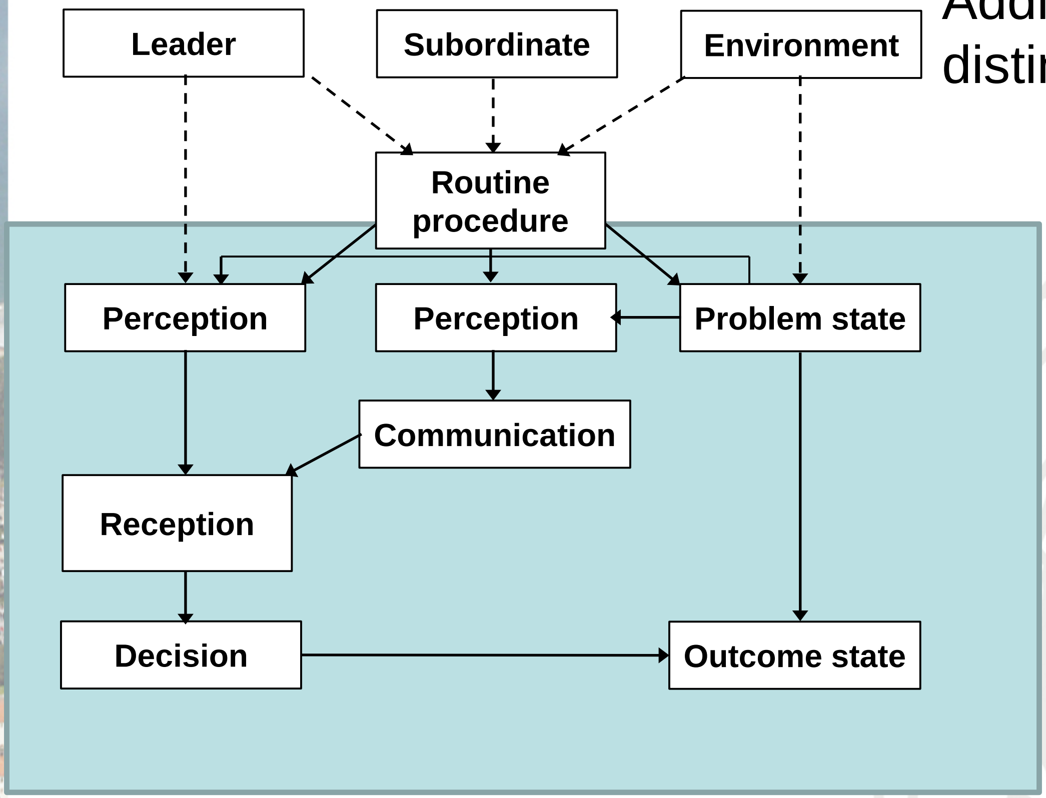
The composite events consist of alternatives between two events, one favorable and one not favorable (e.g. either a problem exists or does not exist)

Execution of the model results in 128 possible event traces or use cases.

Topology of the Decision Pattern

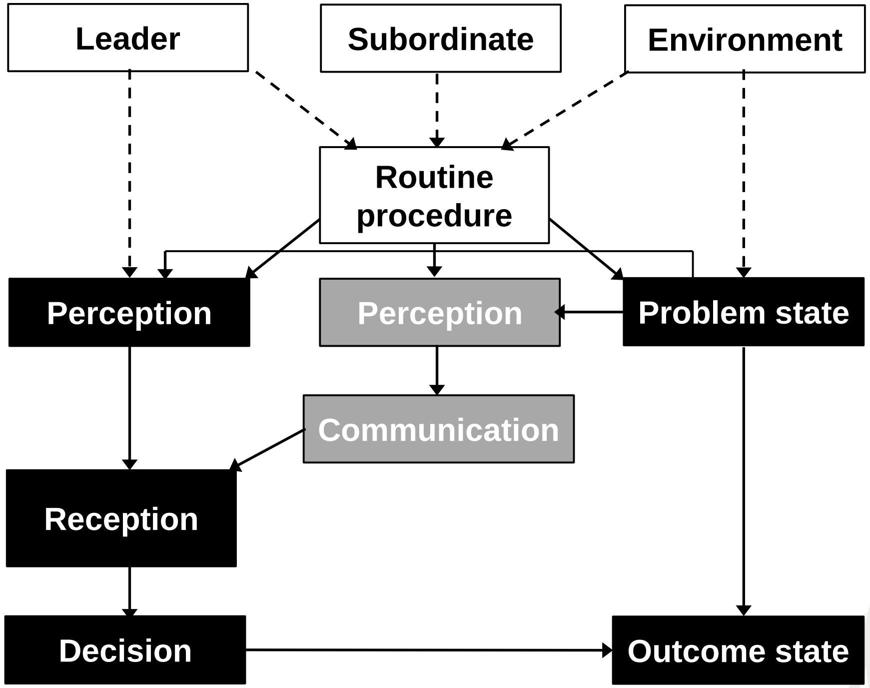
The topology is constant for all traces

Additional semantics are needed to distinguish each of the use cases.



Recognize_environment:	favorable;
Not_recognize_environment:	unfavorable;
Receive_input:	favorable;
Not_receive_input:	unfavorable;
Correct_decision:	favorable;
Not_correct_decision:	unfavorable;
Communicate_observation:	favorable;
Not_communicate_observation:	unfavorable;
Problem:	unfavorable;
No_problem:	favorable;
Successful_outcome:	favorable;
Failed_outcome:	unfavorable;

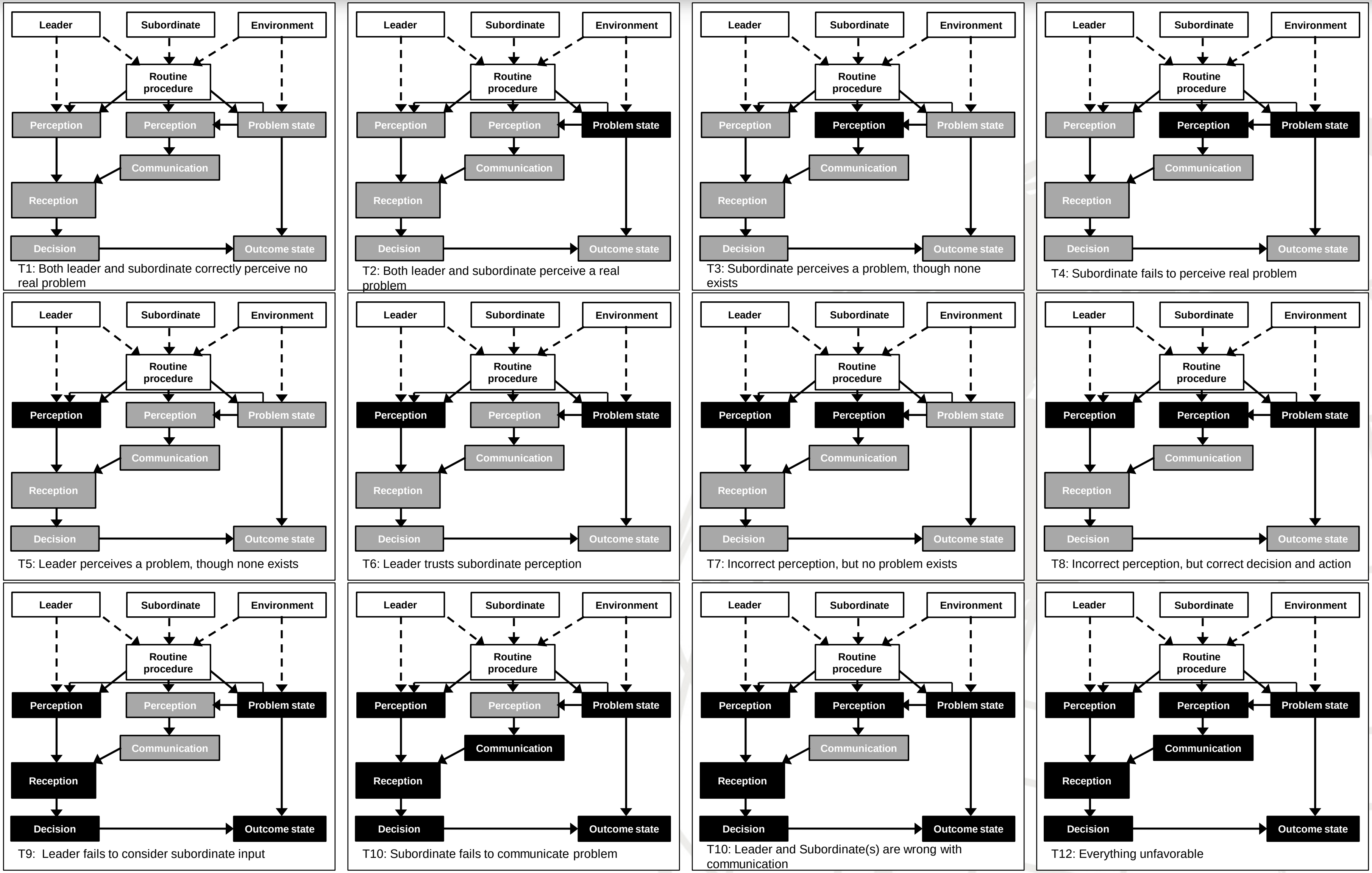
An Instance of Behavior



Template 9 (T9): Leader fails to consider subordinate input

- Execution of the model produced all possible traces or use cases
- The scenario outlined at the beginning of the presentation is identified as Template 9: Leader fails to consider the subordinate input
  - Black textboxes are unfavorable
  - Gray textboxes are favorable

All Possible Behaviors of the Model



T1: Both leader and subordinate correctly perceive no real problem

T2: Both leader and subordinate perceive a real problem

T3: Subordinate perceives a problem, though none exists

T4: Subordinate fails to perceive real problem

T5: Leader perceives a problem, though none exists

T6: Leader trusts subordinate perception

T7: Incorrect perception, but no problem exists

T8: Incorrect perception, but correct decision and action

T9: Leader fails to consider subordinate input

T10: Subordinate fails to communicate problem

T10: Leader and Subordinate(s) are wrong with communication

T12: Everything unfavorable

N-Squared Diagram

Event type	Event name		Leader	Subordinates	Environment	Perception	Recognize_environment	Reception	Receive_input	Decision	Correct_decision	Subordinate	Communication	Communicate_observation	Problem_state	No_problem	Outcome_state	Successful_outcome	Problem	Not_recognize_environment	Not_receive_input	Incorrect_decision	Failed_outcome	Not_communicate_observation	Routine_procedure	favorable	unfavorable
	FROM \ TO (row\column):		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
ROOT	Leader	1				12		12		12															12		
ROOT	Subordinates	2										12															
ROOT	Environment	3													12		12								12		
COMPOSITE	Perception	4					10	12					12							14							
COMPOSITE	Recognize_environment	5																								10	
COMPOSITE	Reception	6							8	12											4						
COMPOSITE	Receive_input	7																								8	
COMPOSITE	Decision	8									8						12					4					
COMPOSITE	Correct_decision	9																								8	
COMPOSITE	Subordinate	10				12							12												12		
COMPOSITE	Communication	11												10										2			
COMPOSITE	Communicate_observation	12						10																		10	
COMPOSITE	Problem_state	13				24										4	12		8								
COMPOSITE	No_problem	14																								4	
COMPOSITE	Outcome_state	15																8					4				
COMPOSITE	Successful_outcome	16																								8	
COMPOSITE	Problem	17																									8
COMPOSITE	Not_recognize_environment	18																									14
COMPOSITE	Not_receive_input	19																									4
COMPOSITE	Incorrect_decision	20																									4
COMPOSITE	Failed_outcome	21																									4
COMPOSITE	Not_communicate_observation	22						2																			2
ATOM	Routine_procedure	23				24									12												
ATOM	favorable	24																									
ATOM	unfavorable	25																									

Key:	
2	Fewest Interactions
24	Most Interactions

Probability derived from the Behavior Model

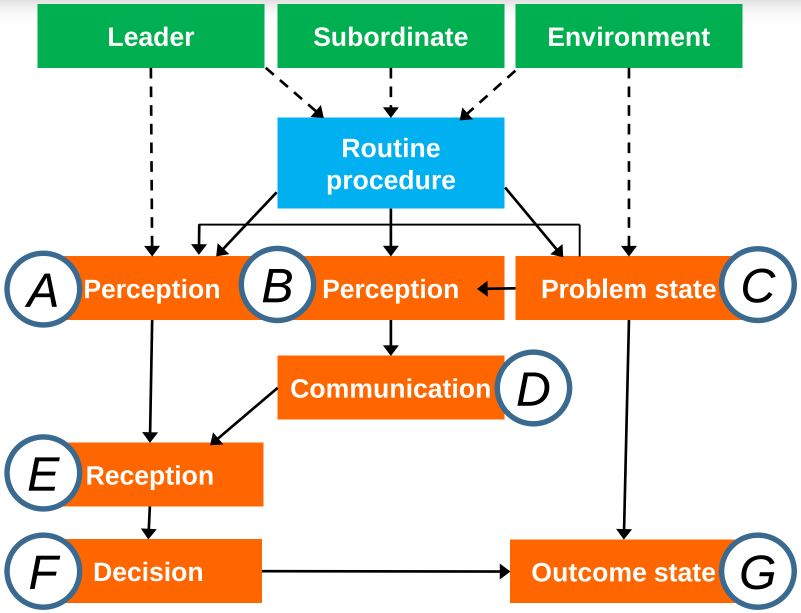
- A model developer has interest in controlling the behaviors of the system of interest
  - Desired behaviors need to be prominent
  - Undesired behaviors need to be identified, then constrained or eliminated

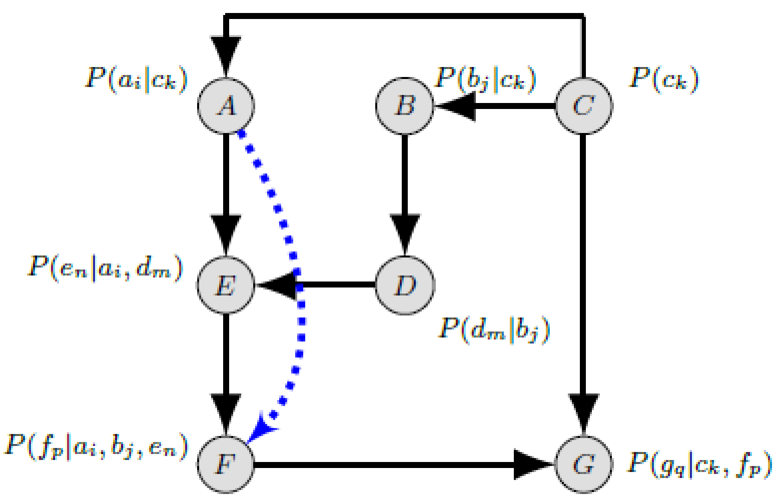
- Constraints form conditional probabilities and can be described within a Bayesian belief network

- Determining the probability of a particular sequence of events (use case) of a Behavior model can help the developer to gauge the effectiveness of the system.

NAVAL POSTGRADUATE SCHOOL

Applying the Approach to the Cross Domain Problem





- The Monterey Phoenix model topology creates the structure for the Bayesian belief network
- Additional relationship is shown for one of the constraints of the model.

The constraints establish explicit cases for conditional probability.

Constraints

logical

Constraint 1: If no problem exists, (k=1), then have a successful outcome (q=1).
P(q = 1 | k = 1) = 1; P(q = 2 | k = 1) = 0

logical

Constraint 2: If the subordinate makes no communication (m = 2), then leader does not receive communication (n=2).
P(n = 2 | m = 2) = 1; P(n = 1 | m = 2) = 0

simplification

Constraint 3: If leader correctly perceives the environment,(i=1), and receives no input from the subordinate,(n=2), then the leader makes a correct decision (p=1).
P(p = 1 | i = 1, n = 2) = 1; P(p = 2 | i = 1, n = 2) = 0

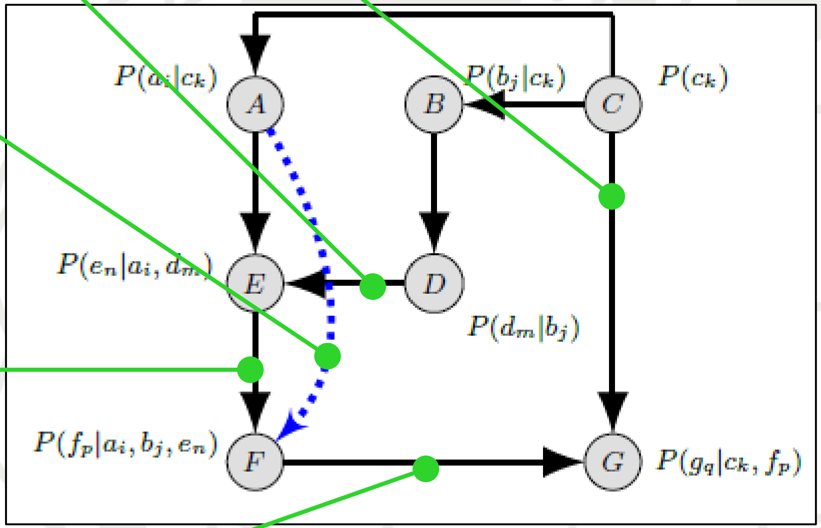
simplification

Constraint 4: If the leader receives communication (n=1), the leader makes a correct decision (p=1), and its corollary..
P(p = 1 | n = 1) = 1; P(p = 2 | n = 1) = 0;
P(p = 1 | n = 2) = 0; P(p = 2 | n = 2) = 1

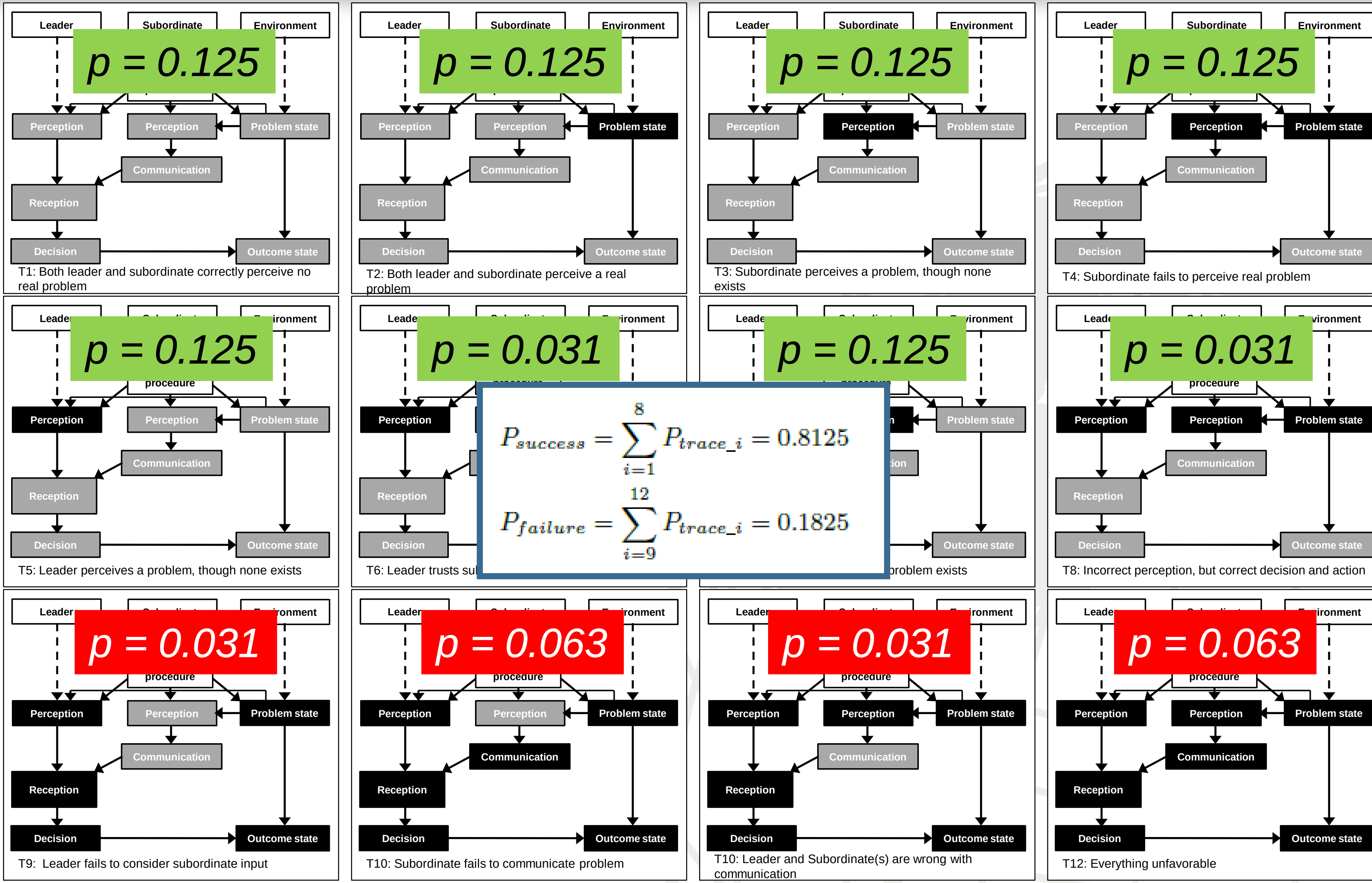
definition

Constraint 5: A correct decision, (p=1), leads to a successful outcome (q=1), and its corollary.
P(q = 1 | p = 1) = 1; P(q = 2 | p = 1) = 0;
P(q = 1 | p = 2) = 0; P(q = 2 | p = 2) = 1

Conditional probability listed for each constraint



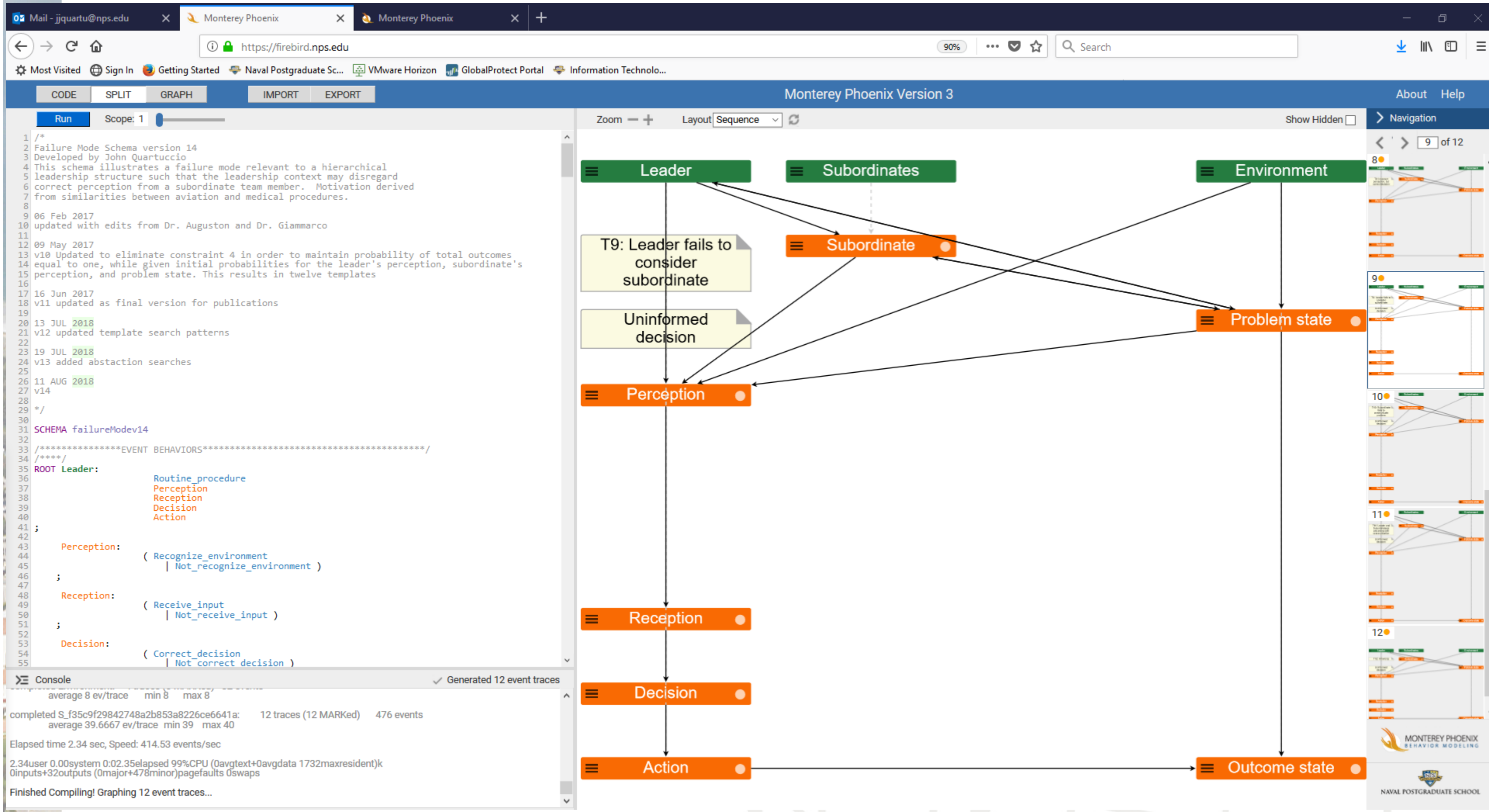
All Possible Behaviors of the Model



$$P_{success} = \sum_{i=1}^{8} P_{trace\_i} = 0.8125$$

$$P_{failure} = \sum_{i=9}^{12} P_{trace\_i} = 0.1825$$

Demonstration of Model Execution



Findings

- Behavior modeling of a cross-domain problem provides insight to decision events
- Patterns of behavior identified as templates
- Assertion checking finds all matches to the template, and marks the trace or use case for identification
- Stochastic properties applied to the MP model
- Monterey Phoenix is available for anyone to use at https://firebird.nps.edu

Conclusions

- Behavior analysis helps the developer to derive alternative paths of execution
  - Exposes the logic behind inherent within the model
  - Enables insight to the fundamental nature of the system

- Once the logical level is established, more detailed levels of performance can be investigated

Questions and Discussion

- Questions?
- Discussion?
- Contact information:

John Quartuccio
jjquartu@nps.edu

NAVAL POSTGRADUATE SCHOOL

References

[1] Mikhail Auguston. Monterey Phoenix, or how to make software architecture executable. In Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming systems languages and applications, pages 1031–1040. ACM, 2009.

[2] Mikhail Auguston. Monterey Phoenix, System and Software Architecture and Workflow Modeling Language Manual (version 2). Naval Postgraduate School, Monterey California, 2 edition, April 2016. https://wiki.nps.edu/display/MP/MP+Crash+Course.

[3] Richard O Duda, Peter E Hart, and David G Stork. Pattern classification. John Wiley & Sons, 2001.

[4] FAA. Lessons learned, United Airlines Flight 173, December 1978. published on-line, downloaded November 12, 2016. http://lessonslearned.faa.gov/ll main.cfm?TabID=1&LLID=42.

[5] Kevin Fong. What we can learn from fatal mistakes in surgery. on-line, downloaded November 12, 2016, March 2013. http://www.bbc.com/news/health-21829540.

[6] Kristin Giammarco. A formal method for assessing architecture model and design maturity using domain-independent patterns. Procedia Computer Science, 28:555–564, 2014.

[7] Daniel Jackson. Alloy: A lightweight object modelling notation. ACM Trans. Softw. Eng. Methodol., 11(2):256–290, April 2002.

[8] Daniel Jackson. Software Abstractions: logic, language, and analysis. MIT press, 2012.

[9] Daniel Jackson and Jeannette Wing. Lightweight formal methods. ACM Comput. Surv., 28(4es):121, 1996.

[10] Cliff B. Jones. A rigorous approach to formal methods. ACM Comput. Surv., 28(4es):121, 1996.

[11] Rudolf K Keller, Reinhard Schauer, S´ebastien Robitaille, and Patrick Pag´e. Pattern-based reverse-engineering of design components. In Proceedings of the 21st international conference on Software engineering, pages 226–235. ACM, 1999.

[12] John Quartuccio, Kristin Giammarco, and Mikhail Auguston. Deriving probabilities from behavior models defined in Monterey Phoenix. In System of Systems Engineering (SoSE), 2017 12th International Conference on, 2017.

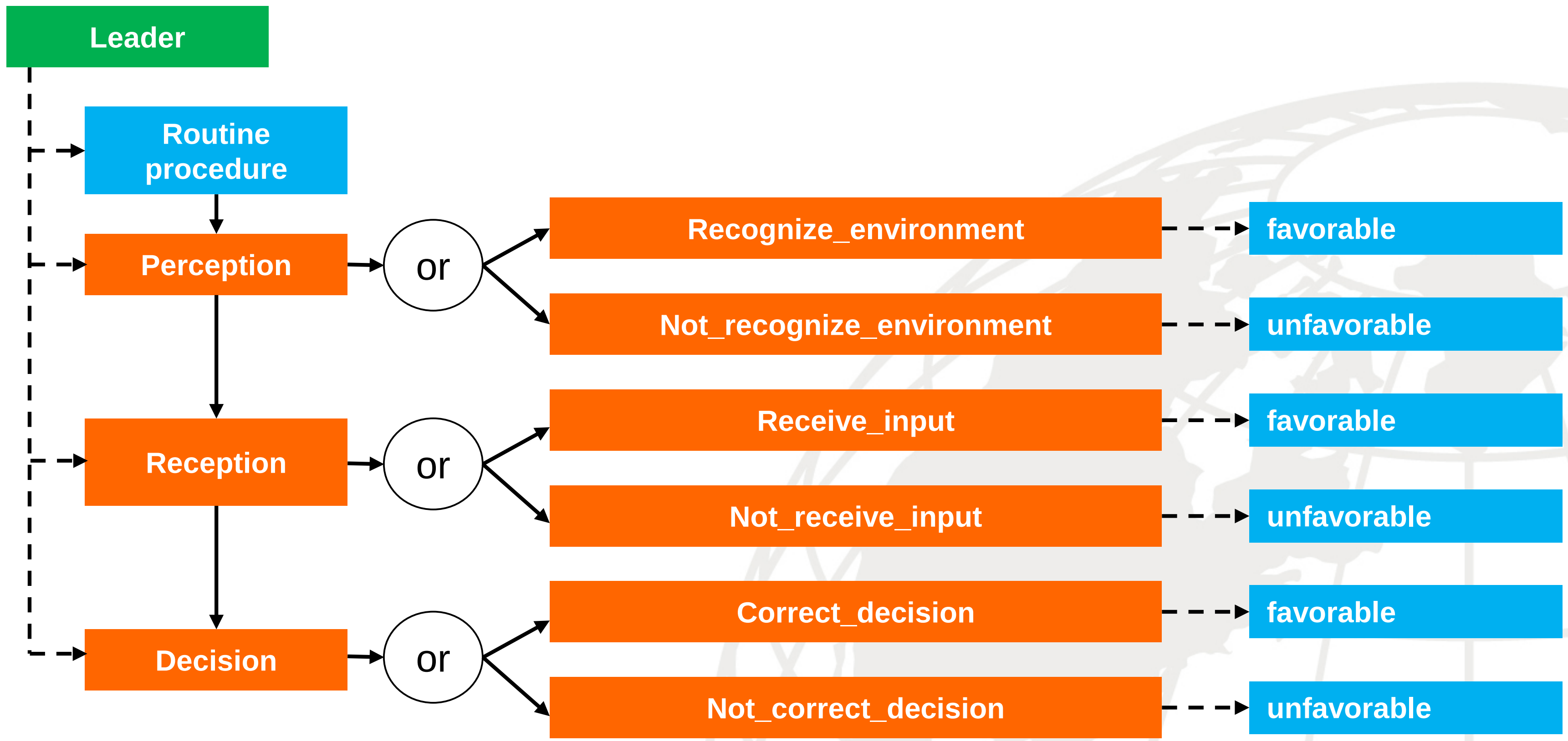
[13] Songzheng Song, Jiexin Zhang, Yang Liu, Mikhail Auguston, Jun Sun, Jin Song Dong, and Tieming Chen. Formalizing and verifying stochastic system architectures using Monterey Phoenix. Software & Systems Modeling, 15(2):453–471, 2016.

[14] Matthew Syed. Black Box Thinking: the surprising truth about success. John Murray, 2015.

[15] Jian Yu, Tan Phan Manh, Jun Han, Yan Jin, Yanbo Han, and Jianwu Wang. Pattern based property specification and verification for service composition. In International Conference on Web Information Systems Engineering, pages 156–168. Springer, 2006.

- Back-up notes on MP syntax

Behaviors of the Leader



Key
- Root event (establishes hierarchy)
- Composite event (contains sub-events)
- Atomic event (contains no sub-events)
- Inclusion relationship
- Precedence relationship
- Note (not part of MP)

Event Behaviors of the Leader

Root event
- established hierarchy

Atomic event
- no subordinate events

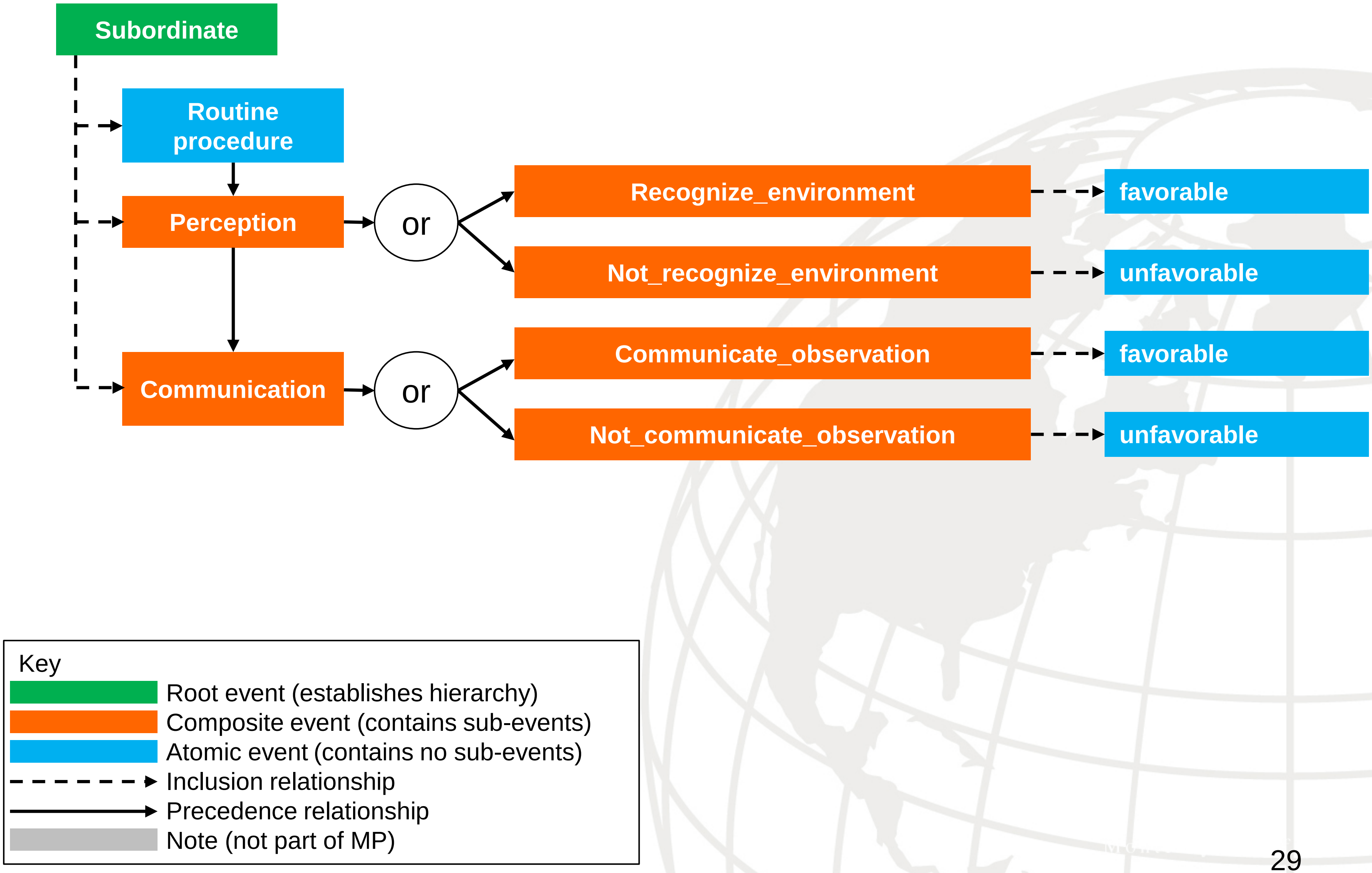
Composite events
- have subordinate events

Order
- events listed in sequence order

```
ROOT Leader:
                    Routine_procedure
                    Perception
                    Reception
                    Decision
;
     Perception:
                    ( Recognize_environment
                        | Not_recognize_environment )
    ;
     Reception:
                    ( Receive_input
                        | Not_receive_input )
    ;
     Decision:
                    ( Correct_decision
                        | Incorrect_decision )
    ;
```

Alternatives
- separated by the “pipe” character, meaning “or”

Event Behaviors of the Subordinate



Event Behaviors of the Subordinate

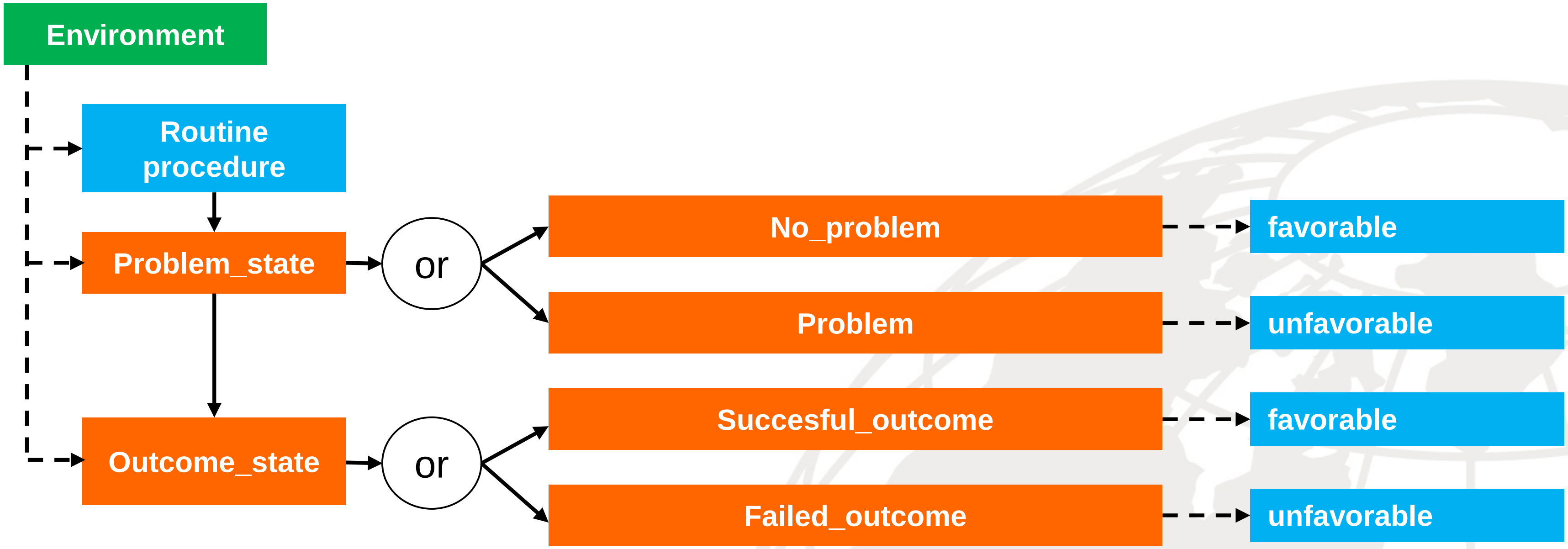
One or many Subordinate events

- indicated by the “plus” character
- determined by scope of execution

- Perception defined previously still holds

```
ROOT Subordinates:  {+ Subordinate +}
;

    Subordinate:
                      Routine_procedure
                      Perception
                      Communication
    ;

    Communication:
                    ( Communicate_observation
                        | Not_communicate_observation )
    ;
......
```

Event Behaviors of the Environment



Key

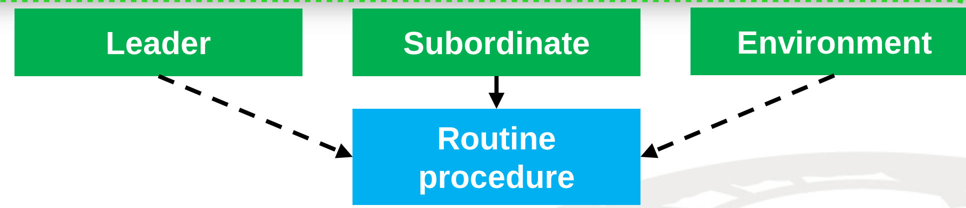
- Root event (establishes hierarchy)
- Composite event (contains sub-events)
- Atomic event (contains no sub-events)
- Inclusion relationship
- Precedence relationship
- Note (not part of MP)

Event Behaviors of the Environment

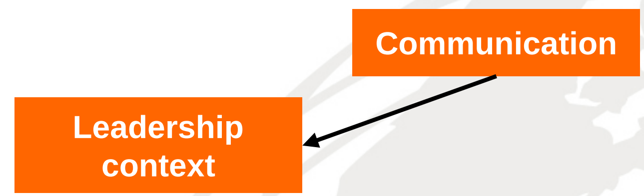
```
ROOT Environment:
                        Routine_procedure
                        Problem_state
                        Outcome_state
;
    Problem_state:
                        ( No_problem
                            | Problem )
    ;

    Outcome_state:
                        ( Successful_outcome
                            | Failed_outcome )
    ;
```

Coordination – Interaction Across Events

Interaction 1: Shared *inclusion* relationship of the Routine_procedure among the Leader, Subordinate, and Environment

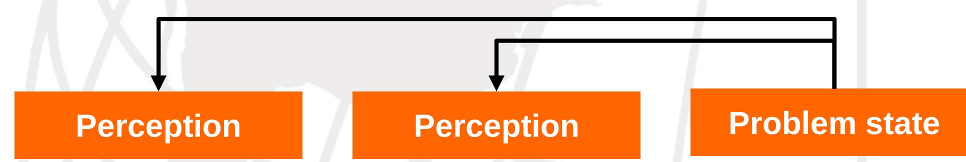


Interaction 2: Subordinate communication *precedes* Leader receipt of communication



Interaction 3: A Decision by the Leader *precedes* an Outcome in the Environment

Interaction 4: The Problem State *precedes* the Perception of both the Leader and Subordinate



Interactions across events are defined separately from the event behaviors, listed on the previous slides. Separating these descriptions affords great flexibility to the model developer.

Coordination – Interaction Across Events

```
/***************COORDINATION************************************/

/****Interaction 1: Sharing the routine_procedure among all roots *\
/**/
Leader, Environment, Subordinates SHARE ALL Routine_procedure;
/**/
/****Interaction 2: Communication by the subordinate precedes the
leadership interpretation of that communication */
/**/
COORDINATE
        $a: Reception                     FROM Leader
    DO COORDINATE
        $b: ( Communicate_observation
            | Not_communicate_observation ) FROM Subordinates
        DO ADD $b PRECEDES $a; OD;
    OD;
/**/
/****Interaction 3: A decision leads an outcome */
/**/
COORDINATE $a: Decision FROM Leader,
           $b: Outcome_state FROM Environment
    DO ADD $a PRECEDES $b; OD;
/**/
/*   Interaction 4: The problem state precedes the perception*/
/**/
COORDINATE $x: Problem_state FROM Environment,
           $y: Perception FROM Leader
    DO ADD $x PRECEDES $y; OD;
/**/
COORDINATE $x: Problem_state FROM Environment
    DO COORDINATE
           $y: Perception FROM Subordinates
      DO ADD $x PRECEDES $y; OD;
    OD;
```

Interaction 1: Shared procedure

Interaction 2: Leader receipt of input depends on communication by the Subordinate, *as in speaking precedes hearing*

Interaction 3: A Decision Leads to an Outcome

Interaction 4: The Problem State precedes the Perception of both the Leader and Subordinate

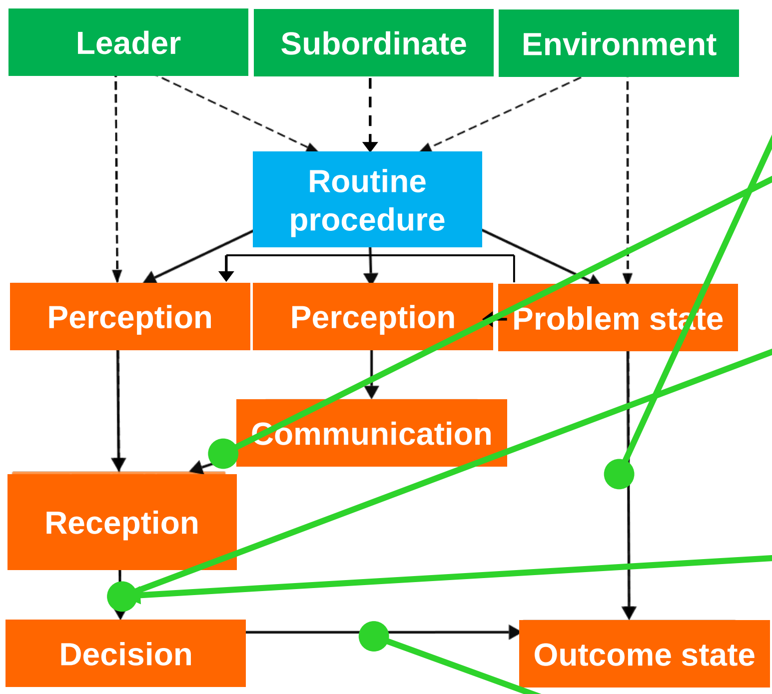
Interactions across events are defined separately from the event behaviors, listed on the previous slides.

Separating these descriptions affords great flexibility to the model developer.

Types of Constraints

- ***Logical consistency***
  - A correct model needs to restrict illogical behavior
  - As an example, ***If*** a message is not sent, ***then*** it cannot be received, or ***If*** a car dos not exist, ***then*** I cannot drive it.
- ***Simplification***
  - Simplification may be applied to improve clarity and encourage the developer's focus on key events
  - As an example, ***If*** a leader receives input, ***then*** always have a correct decision
  - This results in fewer use cases to analyze
- ***Design***
  - Design requirements may be built to eliminate unwanted behaviors
  - As an example, ***If*** an aircraft is out of fuel, ***then*** make the nearest safe landing, ignoring less critical tasks.
  - This example may use automation to achieve the desired result.
- ***Definition***
  - Definition of a particular series of events
  - As an example, ***If*** a leader makes a correct decision, ***then*** always have a successful outcome

Constraints



logical

Constraint 1: If there is no problem in the Environment, then have a successful outcome

logical

Constraint 2: If all Subordinates do not communicate, then the Leader has no input

simplification

Constraint 3: If the Leader recognizes the Environment and does not receive input, then the Leader makes a correct decision

simplification

Constraint 4: If the Leader receives input, then the Leader makes a correct decision, *and* If the Leader does not receive input, then the Leader makes an incorrect decision

definition

Constraint 5: If the Leader makes a correct decision, then have a successful outcome *and* If the Leader makes an incorrect decision, then have a failed outcome

These constraints reduce the number of possible traces or use cases from 128 to 12.

NAVAL
POSTGRADUATE
SCHOOL

Constraints

```
/****Constraint 1: If there is no problem in the environment,
              then always have a success*/
/**/
ENSURE (#No_problem FROM Environment == 1 ->
        #Successful_outcome FROM Environment == 1);
/**/
/****Constraint 2: If all subordinates do not communicate,
              then the leader receives no input*/
/**/
ENSURE (#Not_communicate_observation FROM Subordinates - #Subordinate == 0 ->
          #Not_receive_input FROM Leader == 1);
/**/
/****Constraint 3: If the leader recognizes the environment and does not
            receive input,
              then the leader makes a correct decision*/
/**/
ENSURE (#Recognize_environment FROM Leader - #Not_receive_input == 0 ->
        #Correct_decision == 1);
/**/

/****Constraint 4: Not receiving an input leads to an incorrect decision,
            and its corollary */
/**/

ENSURE (#Receive_input FROM Leader == 1 ->
        #Correct_decision FROM Leader == 1);

ENSURE (#Not_receive_input FROM Leader == 1 ->
        #Incorrect_decision FROM Leader == 1);

/**/
/****Constraint 5: A correct decision leads to a successful outcome,
            and its corollary */
/**/
ENSURE (#Correct_decision FROM Leader == 1 ->
        #Successful_outcome FROM Environment ==1);
ENSURE (#Incorrect_decision FROM Leader == 1 ->
        #Failed_outcome FROM Environment ==1);
```

logical

Constraint 1: If there is no problem in the Environment, then have a successful outcome

logical

Constraint 2: If all Subordinates do not communicate, then the Leader has no input

simplification

Constraint 3: If the Leader recognizes the Environment and does not receive input, then the Leader makes a correct decision

simplification

Constraint 4: If the Leader receives input, then the Leader makes a correct decision, *and* If the Leader does not receive input, then the Leader makes an incorrect decision

definition

Constraint 5: If the Leader makes a correct decision, then have a successful outcome *and* If the Leader makes an incorrect decision, then have a failed outcome

These constraints reduce the number of possible traces or use cases from 128 to 12.

Template Search

```
/*************************************************************/
/****Check for Template 1 (T1):  Both leader and subordinate see
                                 no real problem ************/
/**/
IF EXISTS DISJ

/* from leader */
        $L1: favorable FROM Leader,
        $L2: favorable FROM Leader,
        $L3: favorable FROM Leader,

/* from subordinates */
        $S1: favorable FROM Subordinates,
        $S2: favorable FROM Subordinates,

/* from environment */
        $E1: favorable FROM Environment,
        $E2: favorable FROM Environment

        (
            Leader CONTAINS $L1
            AND
            $L1 BEFORE $L2
            AND
            $L2 BEFORE $L3
            AND
            $S1 BEFORE $S2
            AND
            $S2 BEFORE $L2
            AND
            $E1 BEFORE $E2
            AND
            $L3 BEFORE $E2
        )

THEN MARK; SAY("T1:  Both leader and subordinate see no problem"); FI;
```

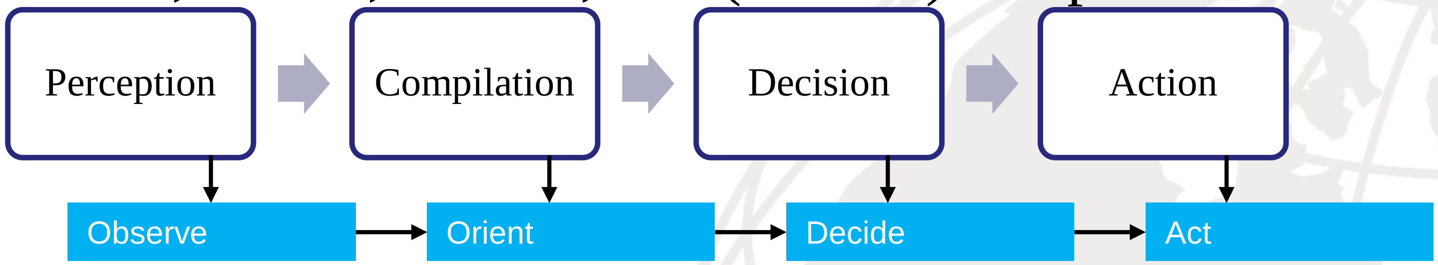
Assertion Checking:
An automated search for each of twelve templates is conducted during execution.

Template 1 is shown, where all alternatives are favorable.

Mark/Say command provides a text statement.

Other Patterns

- Prior patterns were demonstrated for the entire system function
- Segments of the system function also show patterns:
  - Observe, Orient, Decide, Act (OODA) Loop



  - Cooperative OODA Loop