15th Annual NDIA Systems Engineering Conference 2012

Defining Requirements for Error Handling with Usage Models

Dr. William Bail

The MITRE Corporation

24 Oct 2012

MITRE

Introduction

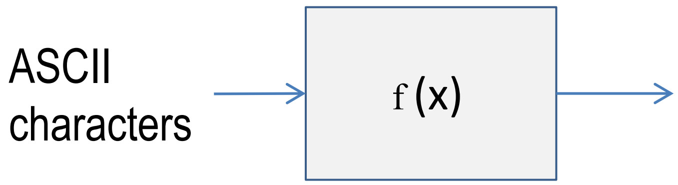
- **Development of software systems that need high levels of dependability generally require some form of error handling**
  - To allow detection of anomalous conditions and recovery from these conditions.
  - Sometimes referred to as robustness characteristics.
- **Defining requirements for such error handling is difficult**
- **Behavioral requirements generally assert positive attributes**
  - Input x produces output y
- **But sometimes, input x produces "error"**
  - Either by plan or by accident
- **This presentation examines one possible approach to defining such requirements**

Requirements

- **System requirements describe what is expected of a system.**
  - Where "requirements" express the desired externally-visible behaviors of a system, as observable by users and other systems
- **Sometimes, requirements address error conditions**
  - Such as out-of-range inputs
  - E.g., where $0 \leq x \leq 100$, compute f(x)
    - Where $x < 0$ or $x > 100$, return 0
- **But what is the requirement if an input of 5 results in an internal error state?**
  - Despite what the requirement says ☹
  - Return 0
  - Return "error"
  - ???

Example

- Consider a system that signals when the exact sequence of "abc" is seen in an on-going input sequence of ASCII characters
- That is:



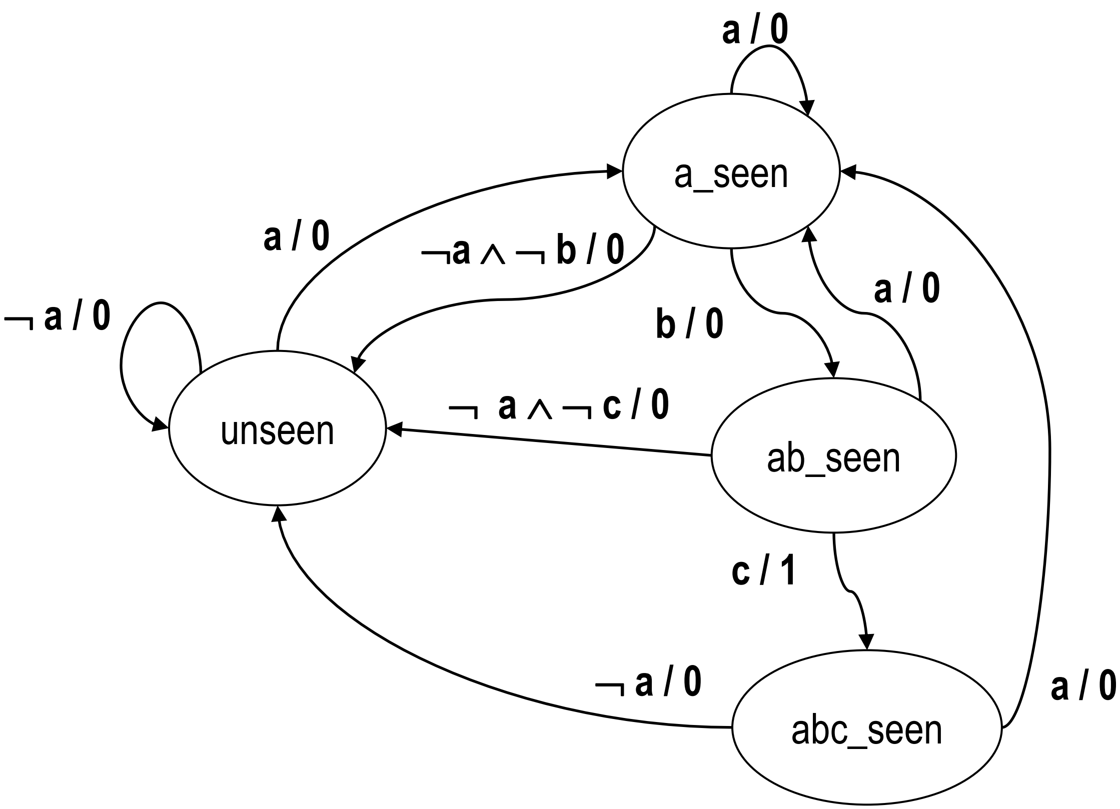
$$f(x) = \begin{cases} 1 \text{ when } x = {*}abc \\ 0 \quad \text{OW} \end{cases}$$

  – Where $x \in \{\text{ASCII characters}\}$

State table for f(x)

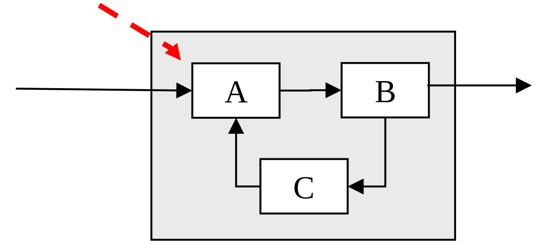
State	Input	Next State	Output
unseen	a	a_seen	0
	$\neg$ a	unseen	0
a_seen	a	a_seen	0
	b	ab_seen	0
	$\neg$ a $\wedge$ $\neg$ b	unseen	0
ab_seen	a	a_seen	0
	c	abc_seen	1
	$\neg$ a $\wedge$ $\neg$ c	unseen	0
abc_seen	a	a_seen	0
	$\neg$ a	unseen	0

State diagram for f(x)



Completeness

- The state machine shown above concisely and unambiguously defines the expected behavior for the component – sort of
  - It is not complete.
- Suppose an input of "a" results in an error state
  - Such as resulting from an exception raised by an internal component
  - E.g. memory leak
  - How do you define the expected response?
- Inherent to specifying "reliability"
  - Likelihood of failure
  - E.g., failure rate to equal 1% of all attempts
- Would like to formally specify this behavior



One approach

- **Use words in a natural language**
- **English**
  - System S shall return a value of
    - 1 when it detects the sequence “abc” in the input stream of ASCII characters
    - 0 for every other situation except for error conditions
    - -1 when it encounters an error condition
- **French**
  - Système S doit renvoyer une valeur de
    - 1 Lorsqu'il détecte la séquence "abc" dans le flux d'entrée de caractères ASCII
    - 0 Pour tous les autres situation, à l'exception des conditions d'erreur
    - -1 Lorsqu'il rencontre une condition d'erreur
- **Lacks the precision of a formal notation such as a state chart**

Another approach

- **Start with a *usage model,* augmented with error likelihoods**
- **A usage model is irreducible discrete event finite state Markov chain, with unique initial and final states**
  - Represented by a directed graph
- **Usage models can be derived from state transition models used to describe the behavior of a software component or system**
- **Usage model U = (S, T, P)**
  - S = set of program states $s_1, \ldots, s_n$
  - T = set of state transitions $t_1,\ldots,t_m$ where each transition is a pair (s_i,s_j)
  - Probability function $P: T \rightarrow (0,1)$
    - Probability of state transition for each transition
  - The sum of all transition probabilities emanating from a state s with transitions T_s equals 1

MITRE

Usage model example

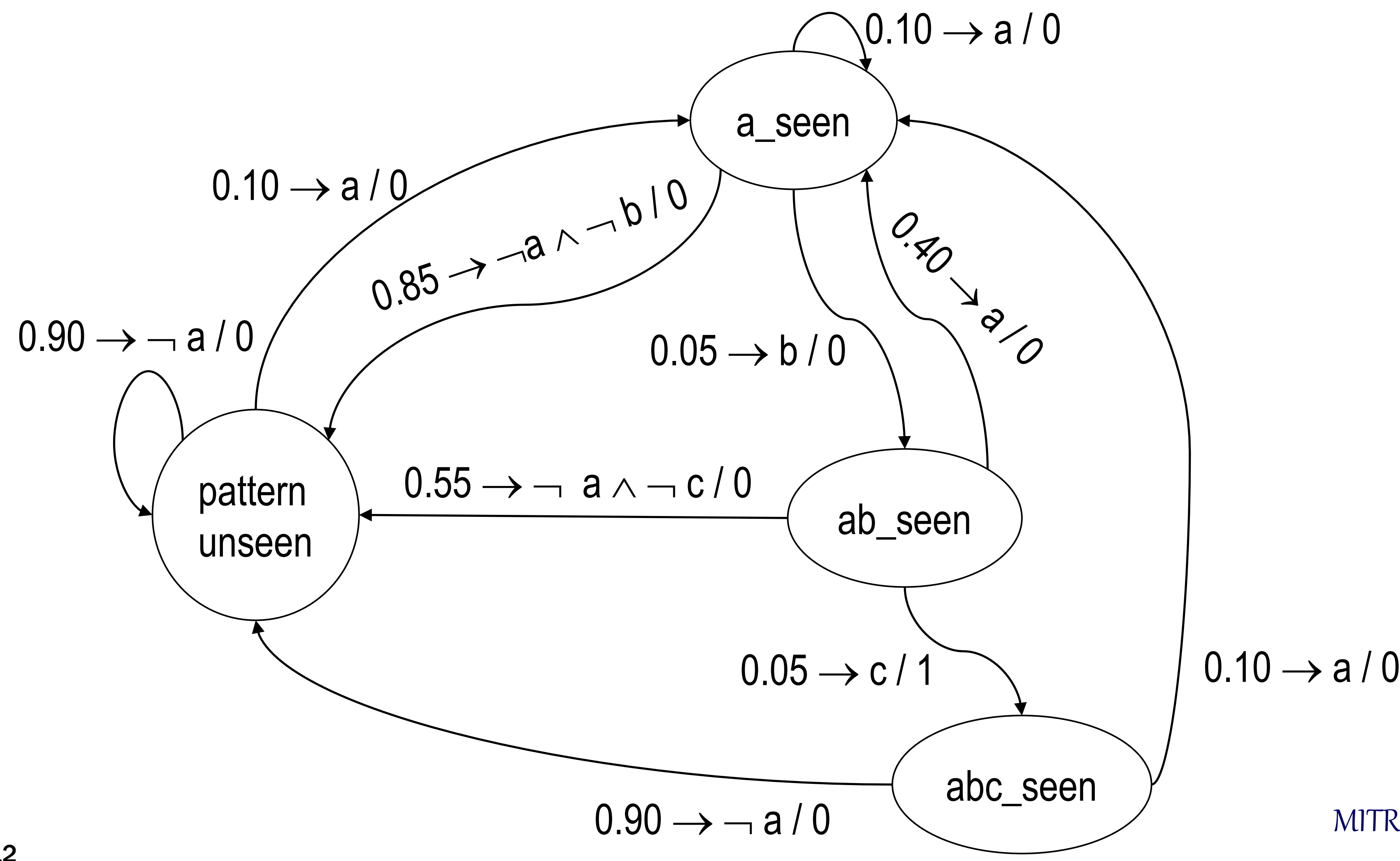
- Basically, a usage model is a state transition model with the likelihood of state transition (input stimulus) as an added factor
- Consider previous sample function and its state chart
- Augment each state transition with a value representing the probability that the specific stimulus will be provided
- The probability may be estimated based on
  - Expected usage profile
  - Historical measurement and experience
  - Prototypes

Usage model for f(x)

- One possible usage model

State	Input	Prob	Next State	Output
unseen	a	0.10	a_seen	0
	¬ a	0.90	unseen	0
a_seen	a	0.10	a_seen	0
	b	0.05	ab_seen	0
	¬ a ∧ ¬ b	0.85	unseen	0
ab_seen	a	0.40	a_seen	0
	c	0.05	abc_seen	1
	¬ a ∧ ¬ c	0.55	unseen	0
abc_seen	a	0.10	a_seen	0
	¬ a	0.90	unseen	0

Usage model diagram – graphical version



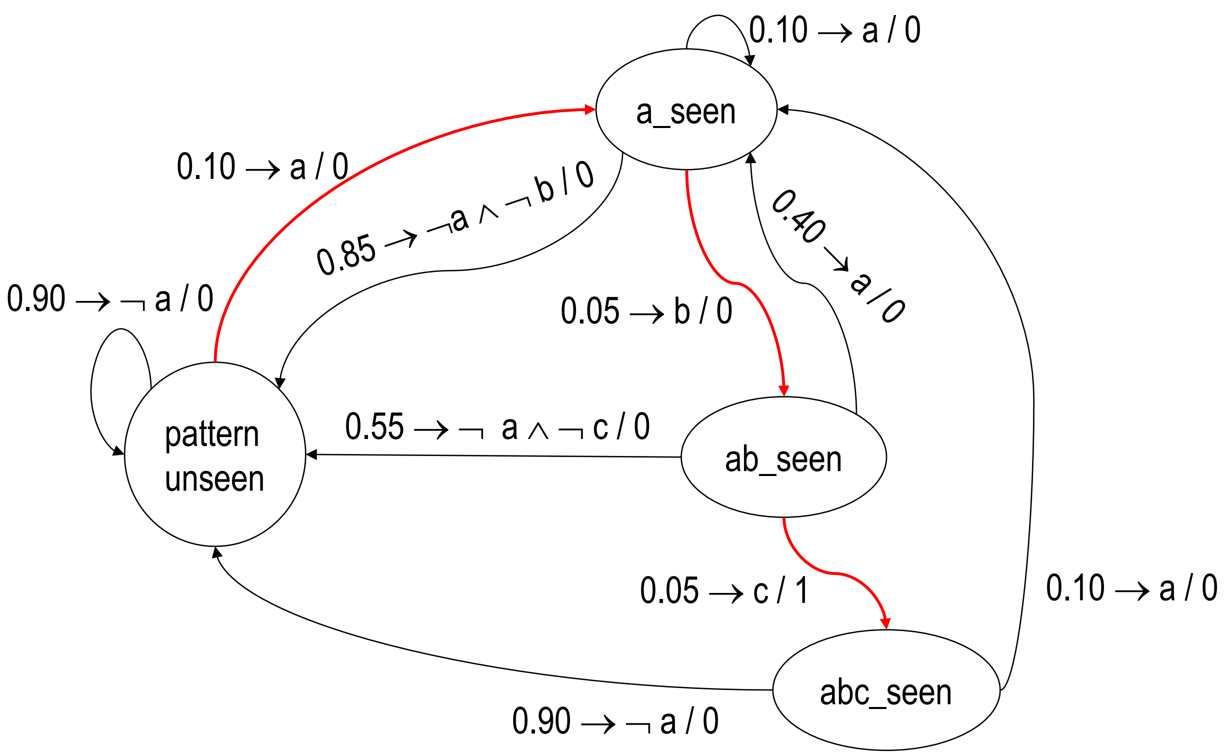
MITRE

Example

∃ 17,576 strings of length 3

Probability of any specific string if all symbols are equally likely = 1/17,576 = 0.000057

Probability of string abc given probs defined in usage model = 0.1*0.05*0.05 = 0.000250



Applications of usage models

- **Usage models can be part of an automatic testing strategy where the test cases are chosen according to the operational profile**
  - Hence can be used to estimate the reliability of the software
  - Accuracy depends on fidelity of assumed operational profile
- **But as defined so far, does not directly support need to be able to specify error handling requirements**
- **Needed:**
  - A way of specifying the desired likelihood of **responses** from the systems
  - That is, with an input of “a”, 99% of the time, an output of 0 is required.
  - The system is required to fail no more than 1% of the time
- **Solution – add the required behavior to the usage model**

Augmented usage model

State	Input	Prob Input	Prob Next State	Next State	Output
unseen	a	0.10	0.99	a_seen	0
	a	0.10	0.01	error	9
	¬ a	0.90	0.99	unseen	0
	¬ a	0.90	0.01	error	9
a_seen	a	0.10	0.99	a_seen	0
	a	0.10	0.01	error	9
	b	0.05	0.99	ab_seen	0
	b	0.05	0.01	error	9
	¬ a ∧ ¬ b	0.85	0.99	unseen	0
	¬ a ∧ ¬ b	0.05	0.01	error	9
ab_seen	a	0.40	0.99	a_seen	0
	a	0.05	0.01	error	9
	c	0.05	0.99	abc_seen	1
	c	0.05	0.01	error	9
	¬ a ∧ ¬ c	0.55	0.99	unseen	0
	¬ a ∧ ¬ c	0.55	0.01	error	9
abc_seen	a	0.10	0.99	a_seen	0
	a	0.10	0.01	error	9
	¬ a	0.90	0.99	unseen	0
	¬ a	0.90	0.01	error	9
error	#	0.90	0.99	unseen	0
	#	0.10	0.01	error	9

Augmented usage model description

Stimulus (input)

Probability of the input occurring
(a characterization of the operational environment)

Probability of the system transitioning to the specific next state

Can be viewed as a required behavior

Next state

Current state

State	Input	Prob Input	Prob Next State	Next State	Output
unseen	a	0.10	0.99	a_seen	0
	a	0.10	0.01	error	9
	¬ a	0.90	0.99	unseen	0
	¬ a	0.90	0.01	error	9
a_seen	a	0.10	0.99	a_seen	0
	a	0.10	0.01	error	9
	b	0.05	0.99	ab_seen	0
	b	0.05	0.01	error	9
	¬ a ∧ ¬ b	0.85	0.99	unseen	0
	¬ a ∧ ¬ b	0.05	0.01	error	9
ab_seen	a	0.40	0.99	a_seen	0
	a	0.05	0.01	error	9
	c	0.05	0.99	abc_seen	1
	c	0.05	0.01	error	9
	¬ a ∧ ¬ c	0.55	0.99	unseen	0
	¬ a ∧ ¬ c	0.55	0.01	error	9
abc_seen	a	0.10	0.99	a_seen	0
	a	0.10	0.01	error	9
	¬ a	0.90	0.99	unseen	0
	¬ a	0.90	0.01	error	9
error	#	0.90	0.99	unseen	0
	#	0.10	0.01	error	9

Response (output)

Required behavior based on stimuli

MITRE

Advantages

- **An integrated model provides a definition of behavior that can be analyzed and modeled as a unit**
  - Interactions of normal and exceptional processing can be more clearly observed
    - as opposed to providing separate definitions.
  - The behavior can be simulated since the model is in the form of a state machine
- **The requirements development process is forced to directly examine both normal and abnormal behaviors up front**
  - As a part of the requirements elicitation and analysis phase
  - Rather than deferring the analysis of abnormal behaviors until later in the development cycle
  - This reduces the risk of inefficient and inappropriate error processing.

Advantages

- **Expectations of reliability can be defined up front**
  - Supports explicit assignment of failure rates as a part of the requirements process
  - Allows developers and users to perform trade-off analyses and make decisions based on an objective consideration of the alternatives
- **Direct support to the verification and test process**
  - Model provides a way of selecting test cases such that the test cases conform to the expected operational profile of how the product will be used
  - Test cases are selected by traversing the state machine, with the selection of inputs based on the likelihoods defined for each of the state transitions
  - Approach already used for normal usage models
    - Incorporating exception handling state transitions can increase the realism of the test process
    - Can also provide ability to assess overall error handling behaviors via simulation

Shortcomings

- **How to determine the appropriate exception rates is not clearly defined**
- **Correlating overall failure rates to the individual likelihoods assigned to each of the exception occurrences is complex**
- **General lack of familiarity of developers in applying this technique**
  - Most developers analyze what the system is to do
  - Considering what might go wrong is not a common practice
  - Introducing this thought process into the requirements elicitation activity might confuse practitioners
    - Until they learn how to apply it
    - Until then, they may fail to adequately capture the true needs
  - Once past learning curve, they will learn how to use it effectively

Conclusions

- Including error handling behaviors in specifying requirements is not commonly practiced
- Current specification models and techniques do not directly support this approach in an integrated way
  - Although support exists for defining exceptional conditions (*e.g.*, natural languages)
- Slight enhancements to the usage modeling technique can support an integrated approach for defining normal behaviors as well as abnormal and exceptional processing
- Approach provides a natural mechanism for defining the desired error rate for the system
  - And continues to support automatic generation of test cases to verify attainment of this error rate
- More research needs to be performed to characterize the most effective ways that this approach can be applied