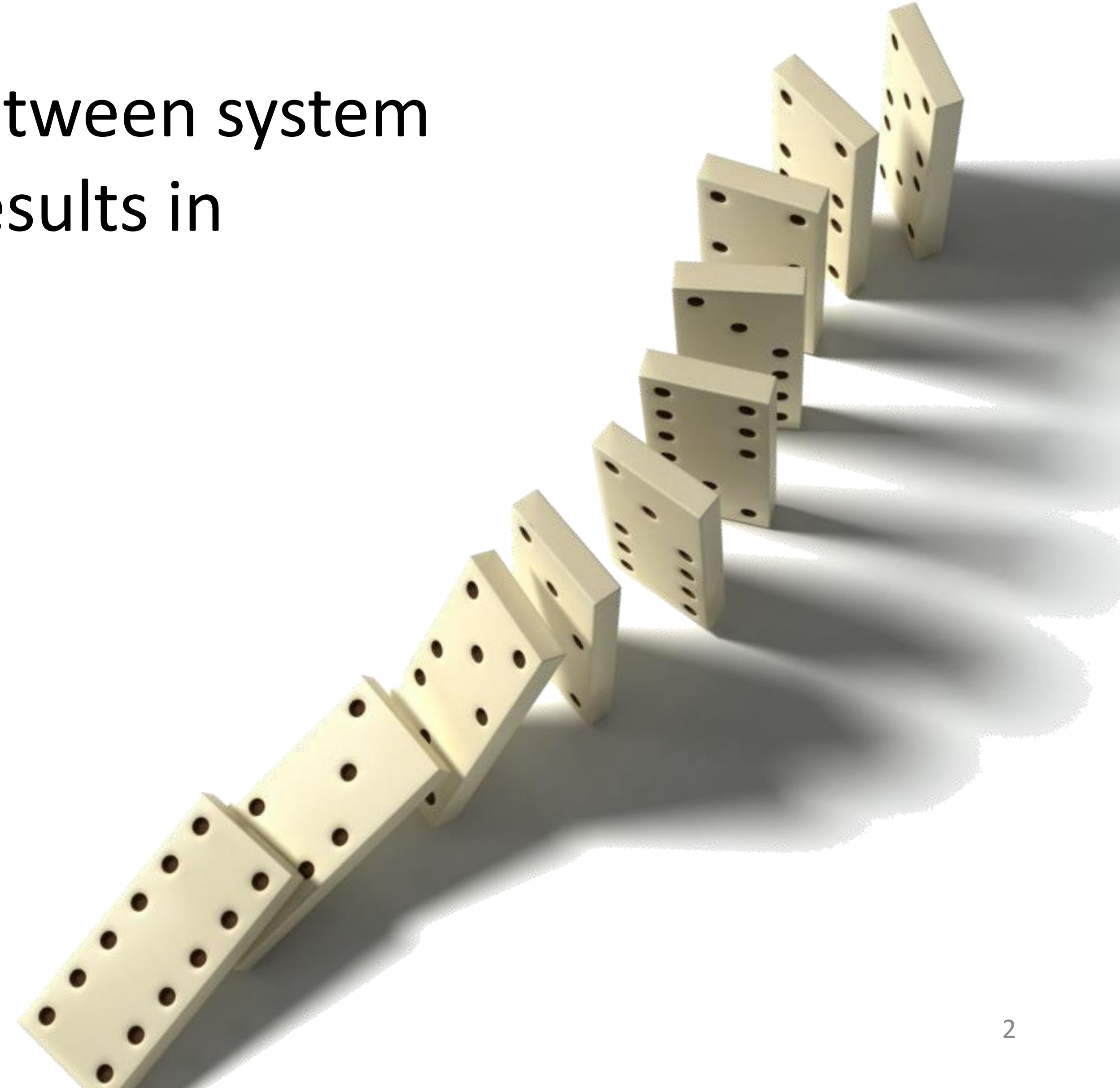
Formal Methods for Human-automation Interaction

Matthew Bolton

System Failure is Complex

Interactions between system components results in breakdowns



Human-automation Interaction:

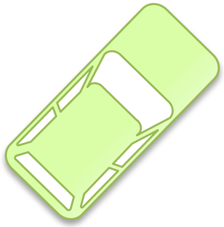
A major contributor to failures in safety critical systems

Medicine

44,000 and 98,000 deaths and 1,000,000 injuries a year

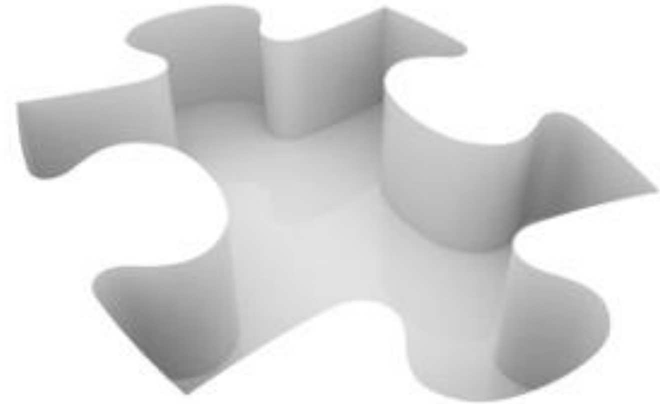
Aviation

75.5% accidents in general aviation and ~ 50% in commercial aviation



Highway Safety

75% of all roadway crashes

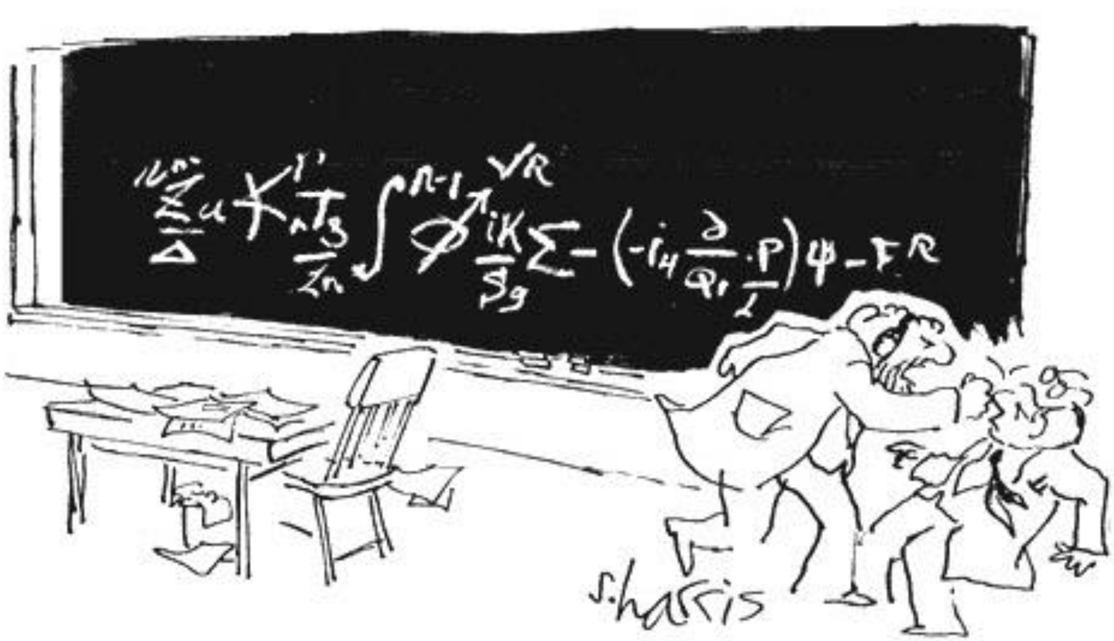


Traditional analysis and evaluation techniques can miss human interactions that could lead to system failure

Computer hardware and software engineers have similar problems

Formal Methods:

Tools and techniques for **proving** that a system will always perform as intended



"You want proof? I'll give you proof!"

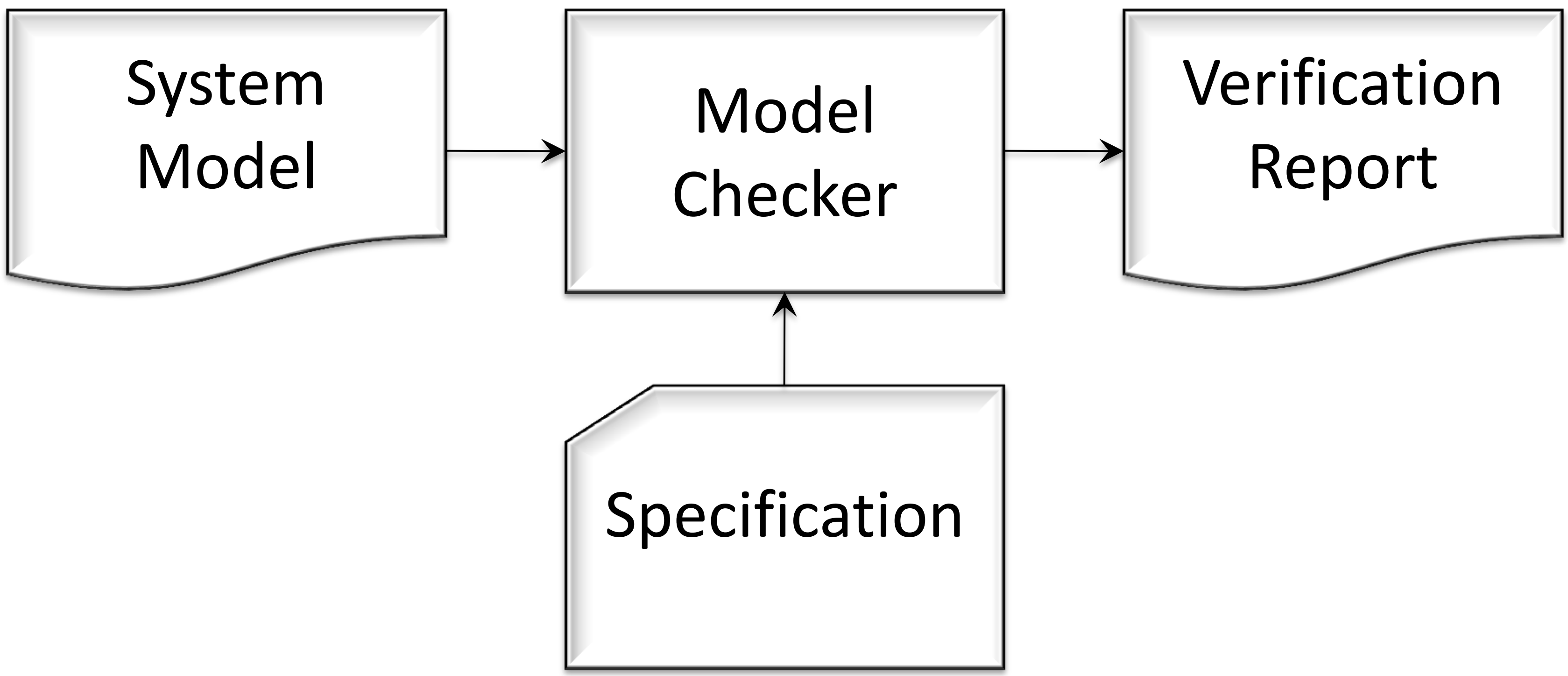
Formal Methods:

Tools and techniques for **proving** that a system will always perform as intended

- **Modeling** – Representing a system's behavior in a mathematical formalism
- **Specification** – Formally expressing a desirable property about the system
- **Verification** – Proving that the model adheres to the specification

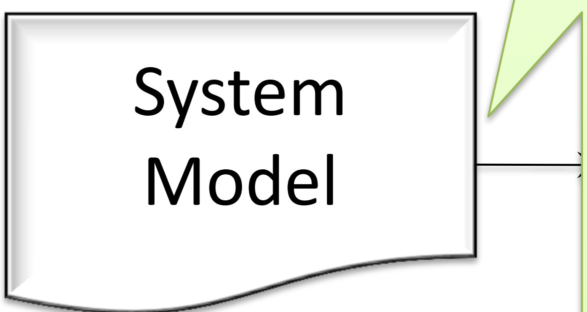
Model checking:

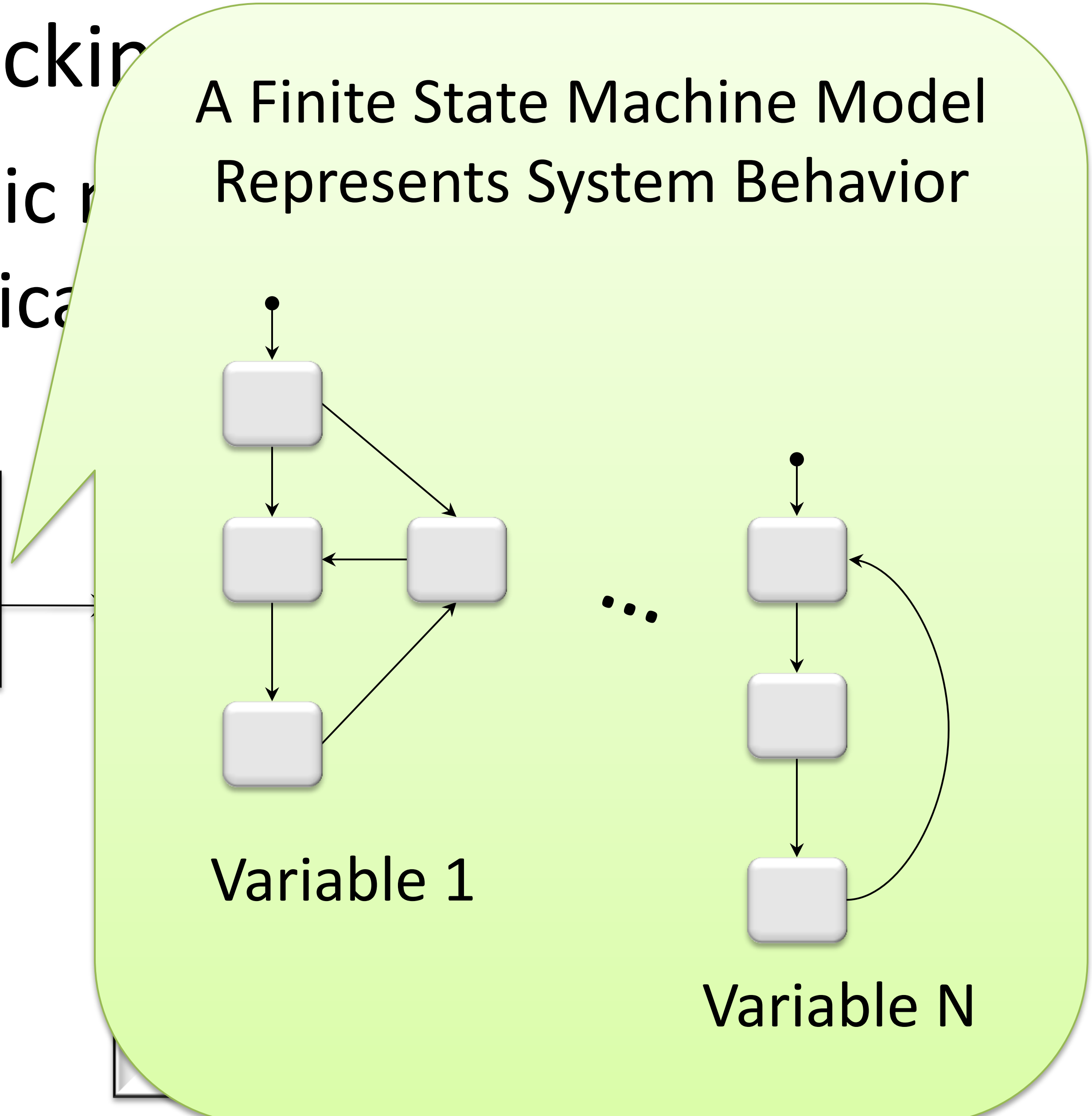
An automatic means of performing formal verification



Model checkin

An automatic r
formal verifica





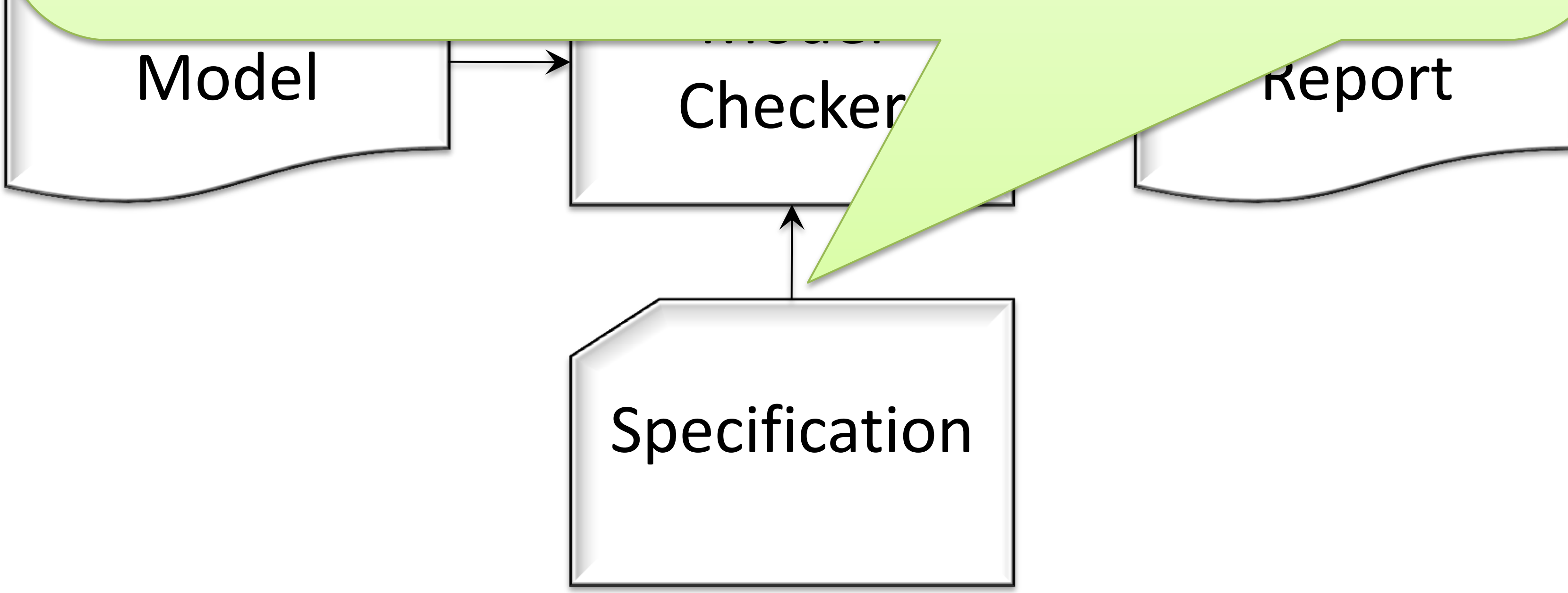
A Temporal Logic Specification Property Asserts Desirable Qualities About the System

For example: “The system should never reach unsafe state X”

$$G \neg (X)$$

Or, “The system should always eventually reach state Y”

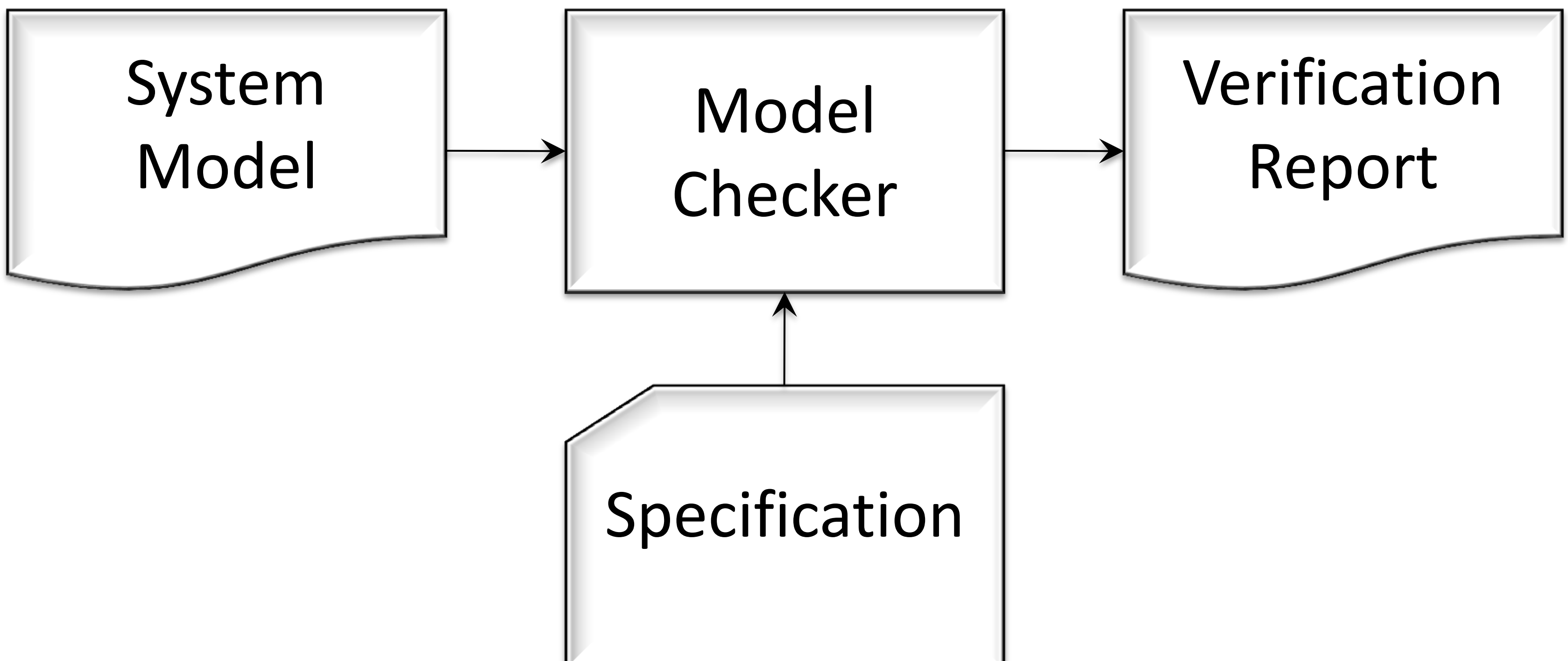
$$F\, Y$$



Model checking

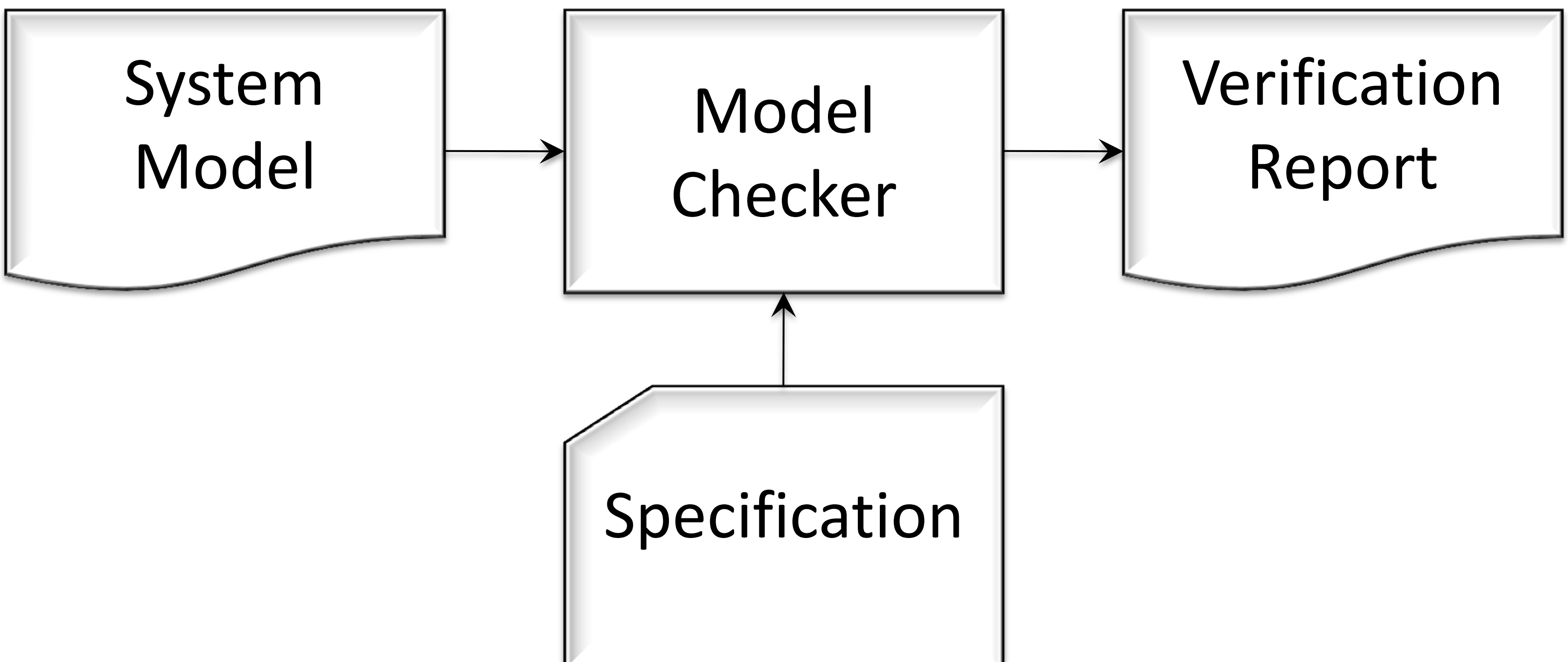
An automatic me
formal verificatio

A model checker "searches" through the model's statespace looking for violations



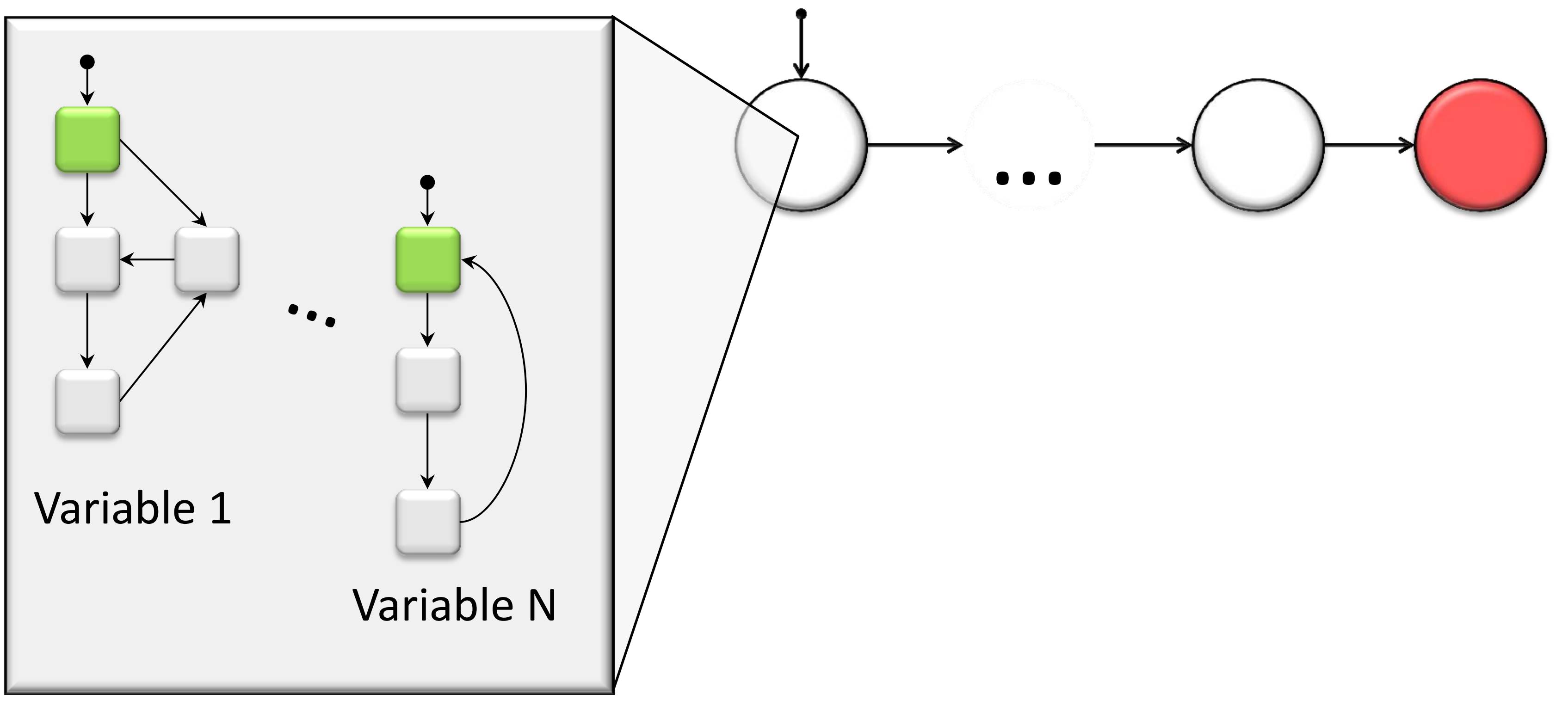
A confirmation or counterexample is returned

# of performing



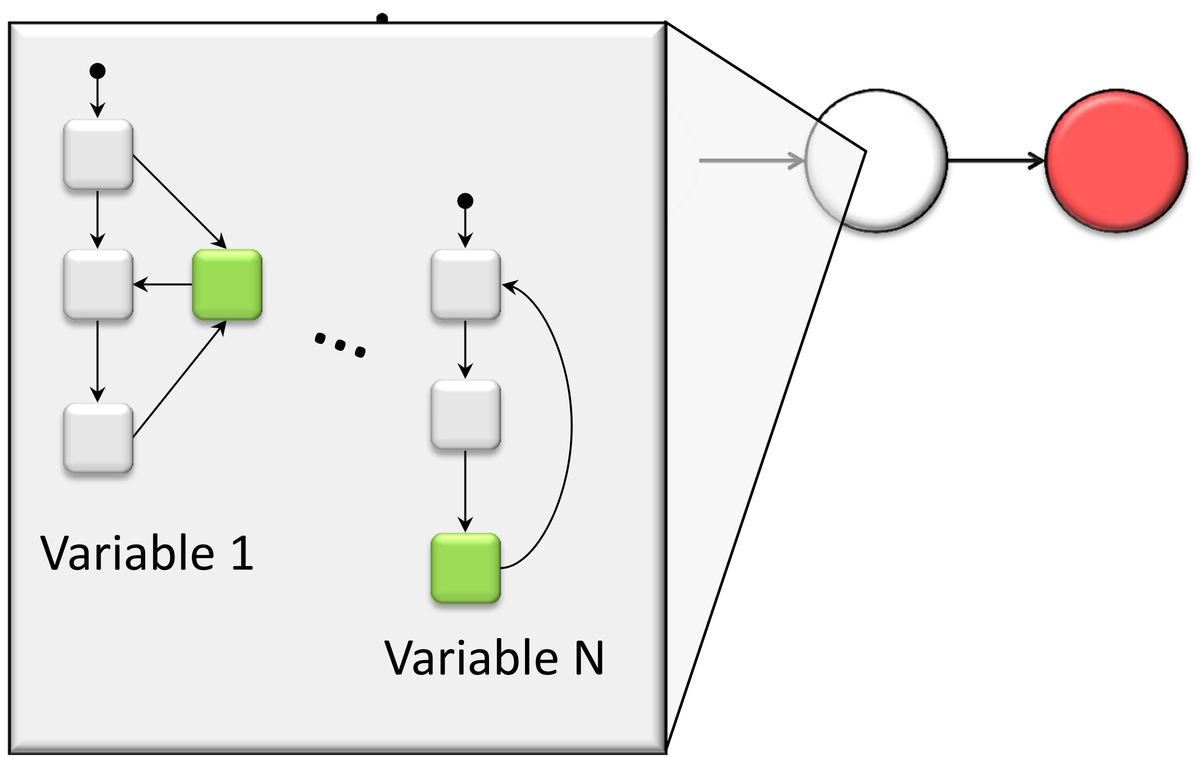
Counterexample

A sequence of states that lead up to a violation



Counterexample

A sequence of states that lead up to a violation



Model Checking Really Works!!

Used to prove that a floating point division bug was removed from the design of the Intel Pentium processor

Used to stabilize Windows by allowing hardware creators to model check that their drivers adhered to the required protocol

Let’s dig into this a little more …

Modeling

You want to model system behavior with robust mathematics

- This can be many things
- Usually, this means using a finite state transition system:
  - System has a finite number of states
  - There are a set of initial states
  - There are inputs
  - States transition between each other based on the inputs or other indicators of state
  - States and/or transitions can map to outputs

Modeling

Automata theory offers many finite state machine constructs:

- Deterministic finite state machines
- Nondeterministic finite state machines
- Mealy machines
- Moore machines
- Etc.

Modeling

However, most analyst use more expressive notations (expressively identical, but often easier to work with):

- State Charts
- Petri Nets
- Special formal modeling languages (promella, SMV, SAL, etc.)

Modeling

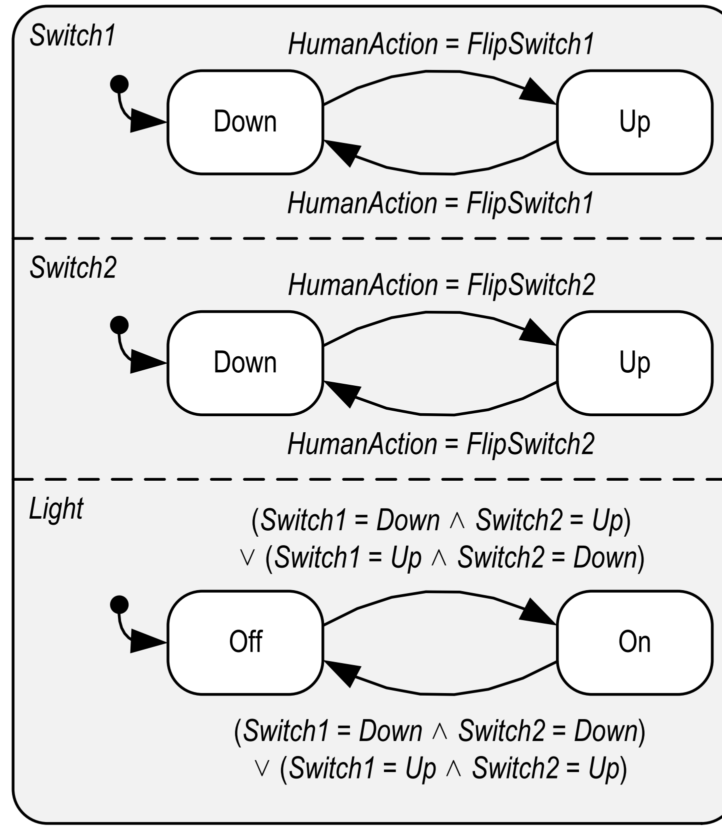
However, most analyst use more expressive notations (expressively identical, but often easier to work with):

- **State Charts**
- Petri Nets
- Special formal mode… lan…
  (promella, SMV, SAL, et…

We will focus on these for the remainder of this presentation because they represent many of the formal modeling concepts in a visual notation

State Charts

- A more expressive formalism for modeling complex system behavior
- A visual formalism
- Hierarchical
- Has memory / history
- Can have concurrency

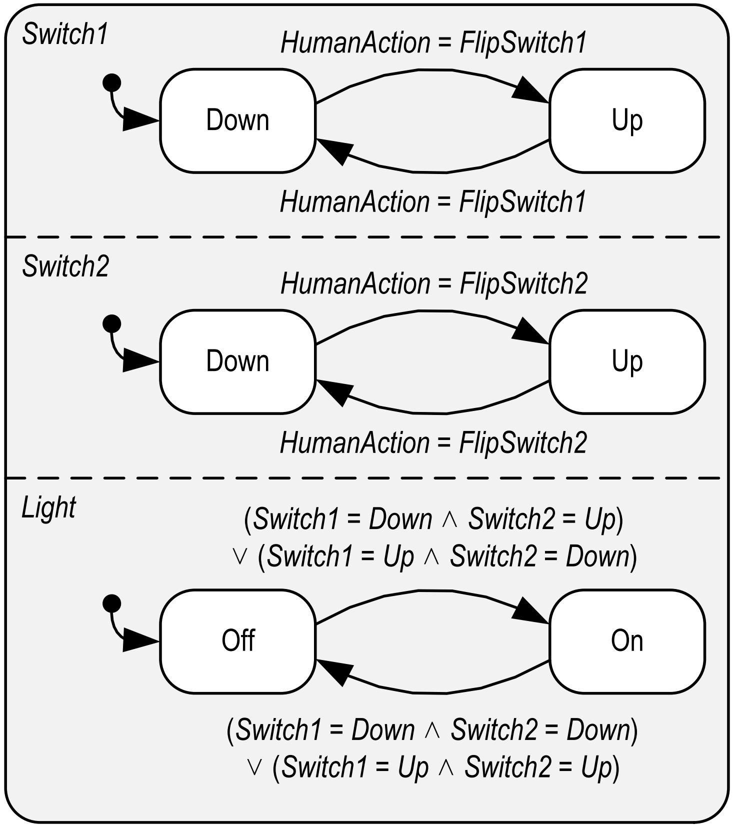


State Charts

Example: State Chart for representing a light switch system

- Concurrent machines represent the state of each switch and the light itself
- States are rounded rectangles
- Each component has an initial state (pointed to by a dotted arrow)
- Boolean logic indicates when a transition (arrow) occurs
- The state of the light will change in response the switches

- More info on state charts http://www.wisdom.weizmann.ac.il/~harel/SCANNED.PAPERS/Statecharts.pdf



For Safety Critical System…

- Model the behavior of the target system
- Encompass the interactions between system components in the model
- Prove that the system adheres to the specification

Specification

- A specification asserts properties you want to be true in the system
- Usually reasons about the relationship of different states in ordinally over time
- Usually expressed as a temporal logic

Specification with Temporal Logic

- Temporal logic allows us to reason about states and/or variable values over ordinal time
- We can assert things like:
  - This should never be true
  - This should always be true
  - This should always happen next
  - X should always happen before Y
  - Etc.

Specification with Temporal Logic

Two dominant types:

- Linear Temporal Logic (LTL)
  - Reasons about all paths through the model
- Computation Tree Logic (CTL)
  - Reasons about path through a computation tree (there can be branching points)

- Both use basic, binary logic operators but add some additional operators

LTL

Temporal operators:

Name	Operator	Interpretation
Global	G ϕ $\Box\ \phi$	ϕ will always be true
NeXt	X ϕ $\circ\ \phi$	ϕ will be true in all next states
Future	F ϕ $\Diamond\ \phi$	ϕ will eventually be true
Until	ϕ U ψ	ϕ will be true until ψ is true

LTL Examples

Jon is always late: G (Jon is late)

I will have a job in the future: F (I have a job)

If I flip a switch, the light will be on in the next state:
(Switch1 = Flipped → X (Light = On))

The light will be on until I unflip a switch:
(Light = On U Switch1 = UnFlipped)

What about this?
G (Switch1 = UnFlipped → X ((Switch1 = Flipped ∧ Light = On) U (Switch1 = UnFlipped)))

CTL

CTL operators are a combination of a path qualifier and a temporal operator:

Path Qualifier:

A – Through all paths

E – Through one or more paths

CTL

CTL operators are a combination of a path qualifier and a temporal operator:

Path Qualifier:

A – Through all paths

E – Through one or more paths

LTL Operators are the equivalent of CTL operators that start with A

CTL

CTL operators are a combination of a path qualifier and a temporal operator:

Path Qualifier:

A – Through all paths

E – Through one or more paths

We can reason about existence using E:
You can see if something is possible

What to check for…

- Safety properties:
  - Properties starting with AG (CTL) or G (LTL)
    Something good should always be true
    or something bad should never happen
    "The machine should never irradiate the patient"
- Liveness:
  - Assertions that use AF (CTL) or F (LTL)
    Something good eventually happens
    Response: something happens in response to something earlier
    "the system always eventually stops running"
- Existence:
  - Assertions that use EF
    The system can do something
    "The system can allow the person to turn the system off"

What to check for…

- Safety properties:
  - Properties starting with AG (CTL) or G (LTL)
    Something good should always be true
    or something bad should never happen
    “The machine should never irradiate the patient”
- Liveness:
  - Assertions that use AF (CTL) or F (LTL)
    Something good eventually happens
    Response: something happens in response to
    “the system always eventually stops running
- Existence:
  - Assertions that use EF
    The system can do something
    “The system can allow the person to turn the system off”

Only CTL can positively assert existence

Clearly this can be used for evaluating system safety…

Using Formal Methods for Human-automation Interaction

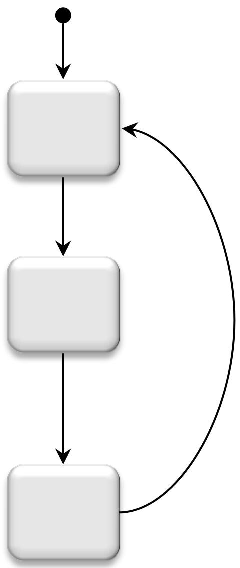
- Proving properties about interfaces to encourage safety
  - Usability analyses
  - Mode confusion analyses
- Proving properties about system safety with models of human behavior
  - Cognitive models
  - Task models

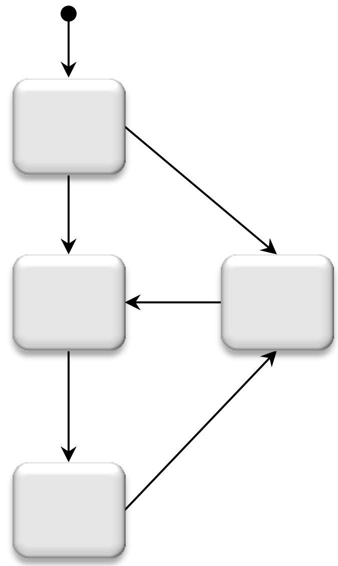
Usability Analyses

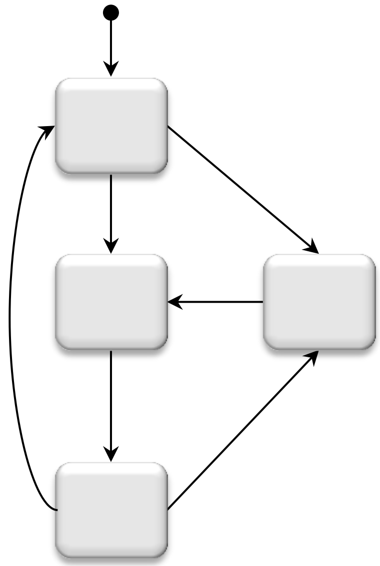
- Model interfaces as finite state machines
- Prove properties indicative of good usability about the interfaces
  - Reachability (interface states can be reached)
  - Visibility (the interface should give feedback)
  - Task-related (things can be accomplished)
  - Reliability (things that make the system reliable):
    - Undoability (things can be done)
    - Consistent behavior
      (the interface always responds the same way)
    - Deadlock freedom

Mode Confusion

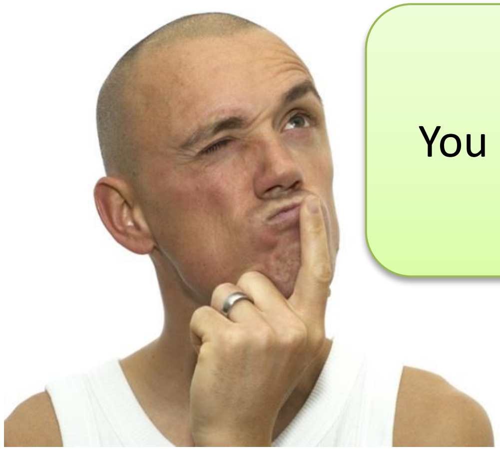
Mode Confusion



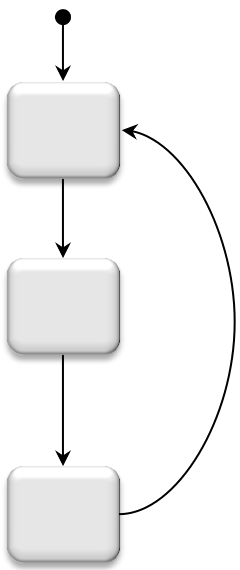


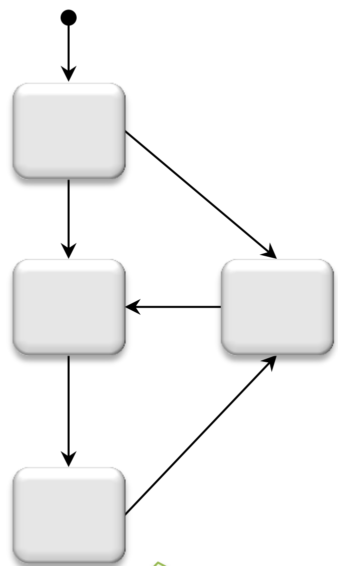


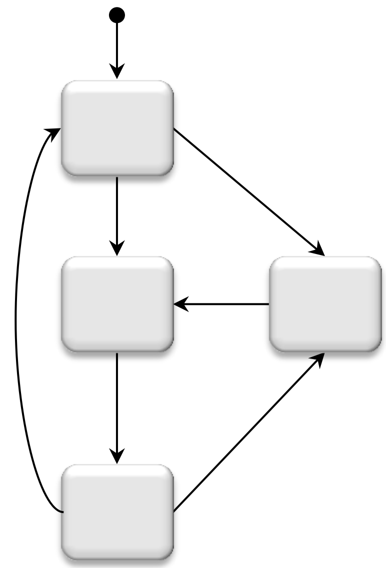
You model the state of the automation

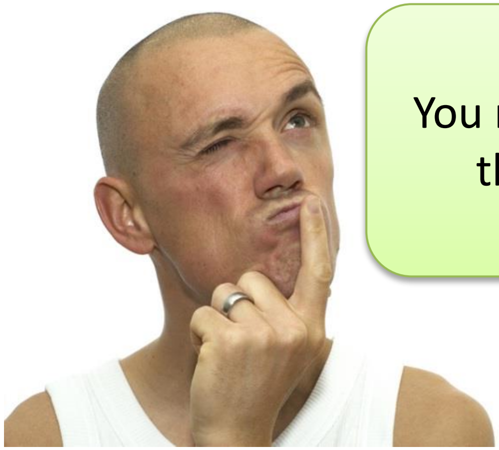


Mode Confusion



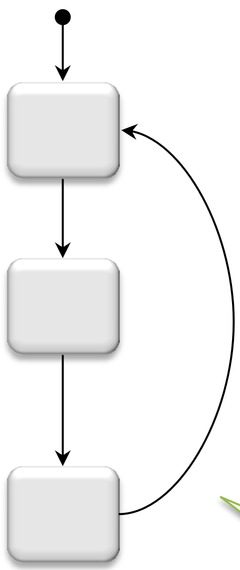


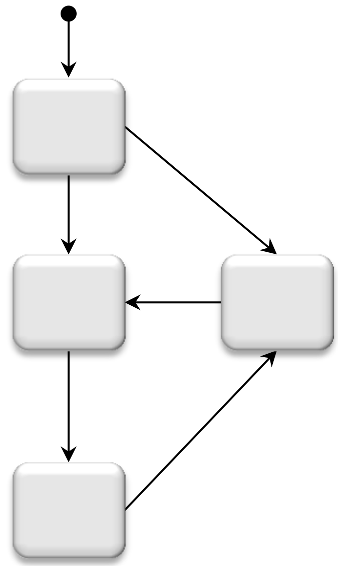


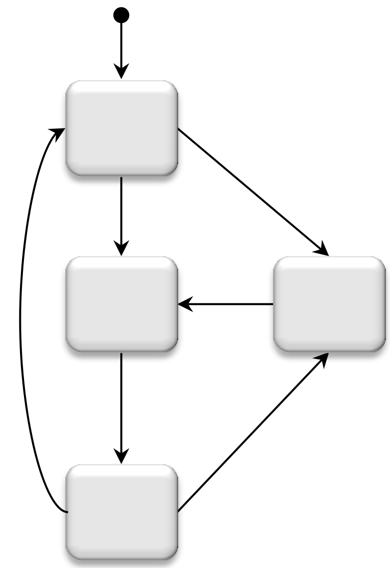


You model the state of the automation, the human-automation interface

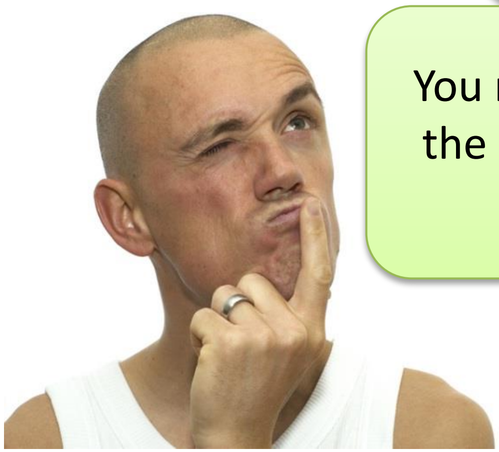
Mode Confusion





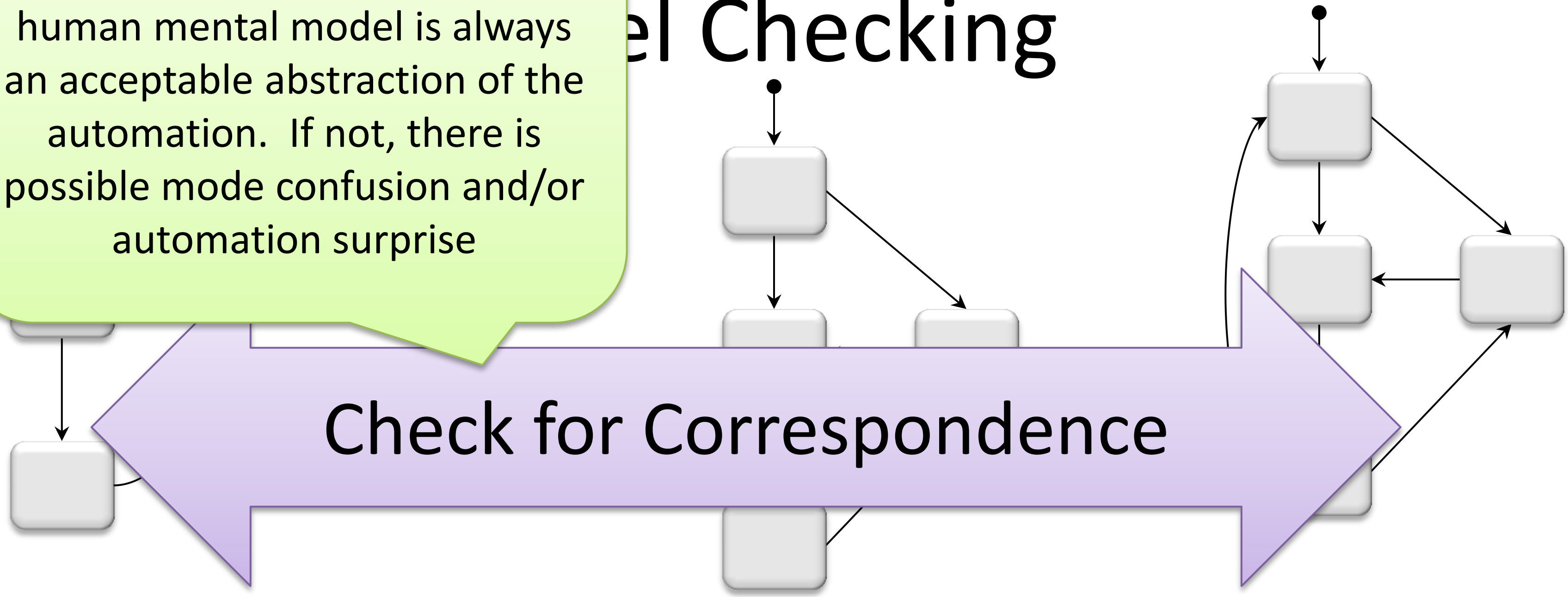


You model the state of the automation, the human-automation interface , and the human mental model



Mode Confusion with el Checking

You model check that the human mental model is always an acceptable abstraction of the automation. If not, there is possible mode confusion and/or automation surprise

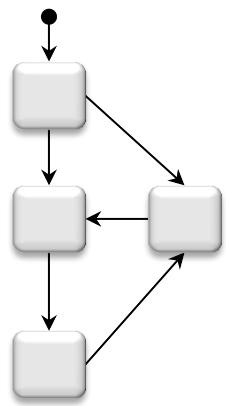


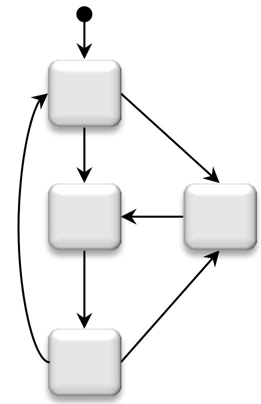
Checking System Safety with Human Behavior

Modeling cognitive behavior …

Checking System Safety with Human Behavior

Modeling cognitive behavior …

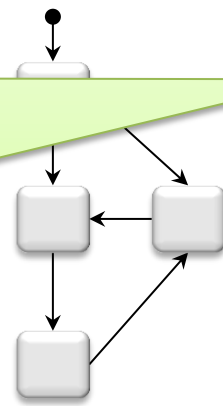


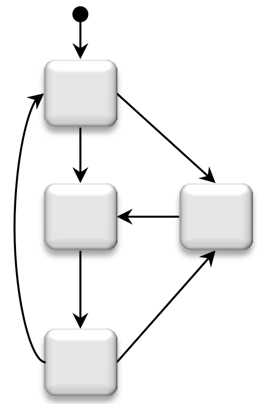


Checking System Safety with Human Behavior

Other system elements are modeled as finite state machines or similar formalisms
(This may include a model of the environment)

behavior …

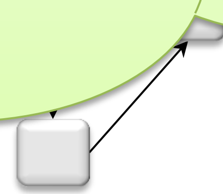


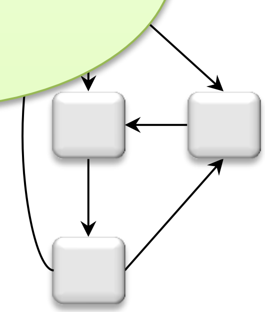


Checking System Safety

The human cognition is modeled as a collection of production rules:

- Attending to different information
- Processing / categorizing that information
- Selecting a response
- Performing the selected response

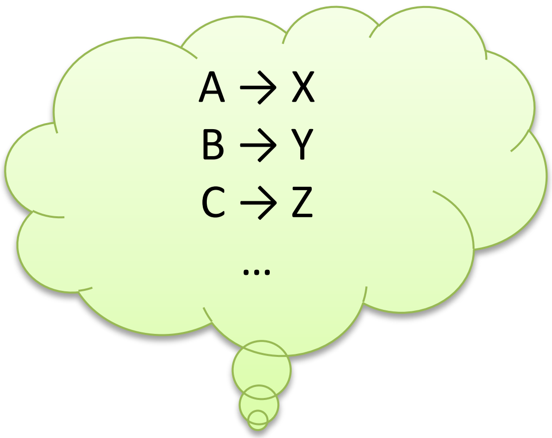


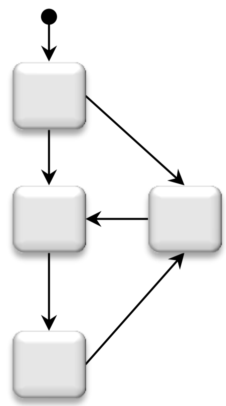


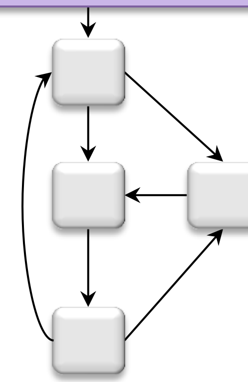
You can check for a number of things:

- That the system is safe for the modeled human behavior or meets other performance requirements
- That the human operator will always achieve their desired goals

  Note: errors can be organically produced by the production rules







Cognitive Models are Great But…

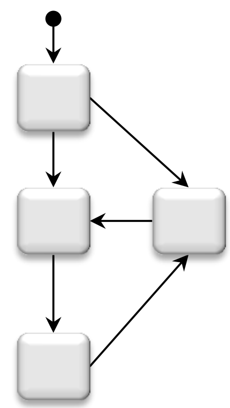
- The cognitive architectures are not widely used
- The use of cognitive models can lead to complex models which can limit analyses

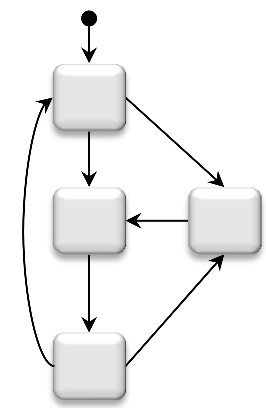
Checking System Safety with Human Behavior

Task analytic behavior models…

Checking System Safety with Human Behavior

Task analytic behavior models…

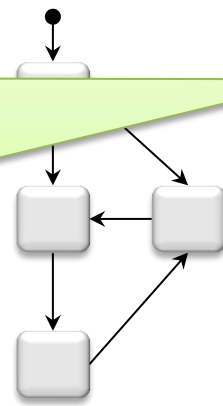


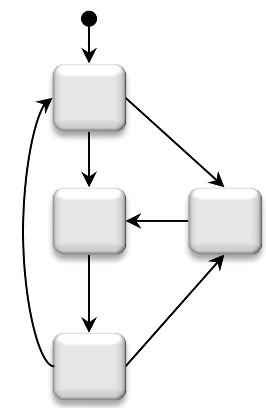


Checking System Safety with Human Behavior

Other system elements are modeled as finite state machines or similar formalisms
(This may include a model of the environment)

vior models…



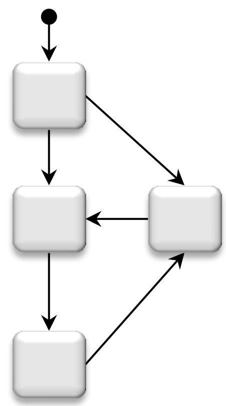


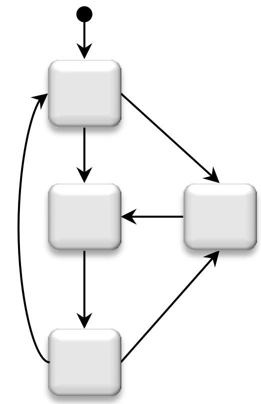
Checking System Safety
nan Behavior

models…

Human Behavior is Modeled Using Task Analytic Behavior Models

- Product of a cognitive task analysis
- Hierarchy (network) of goal directed activities and actions
- Strategic knowledge controls when activities execute and complete
- Modifiers control relationships between activities and actions



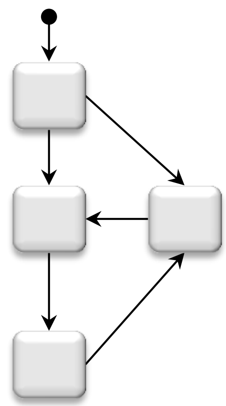


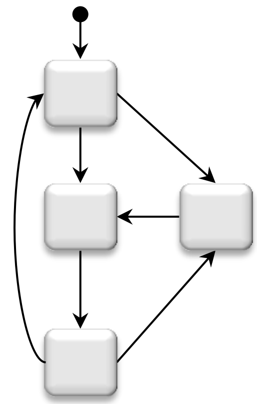
Checking System Safety
nan Behavior

models…

Human Behavior is Modeled Using
Task Analytic Behavior Models

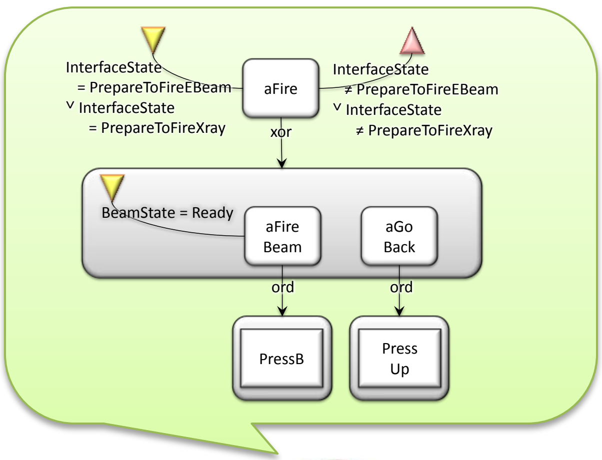
Task model are given formal
semantics that treat them as a finite
state machine

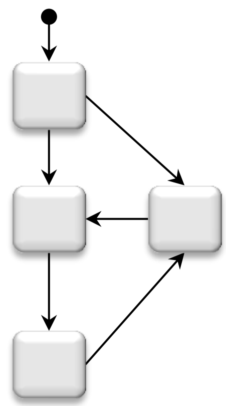


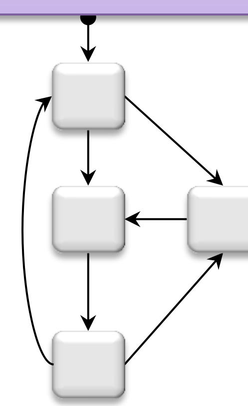


You can check for a number of things:

- That the system is safe for the modeled human behavior or meets other performance requirements
- That the human operator will always achieve their desired goals



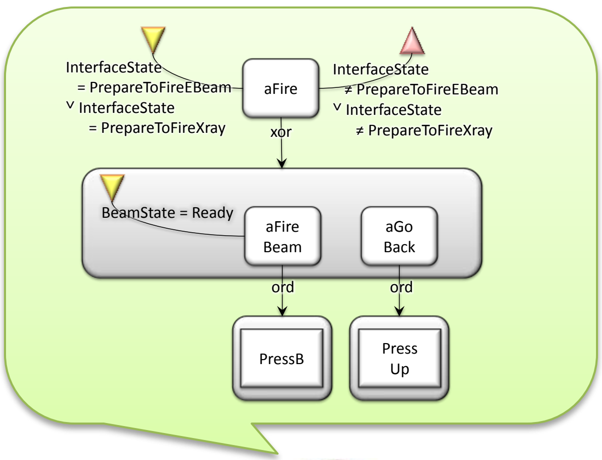


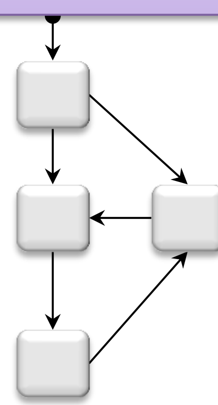


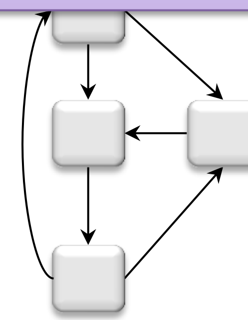
Checking
with Hu

Task analytic behavio

- Human error must be manually included and/or generated in the task structure
- This allows the verification to evaluate the robustness of the system to human error







Task Models

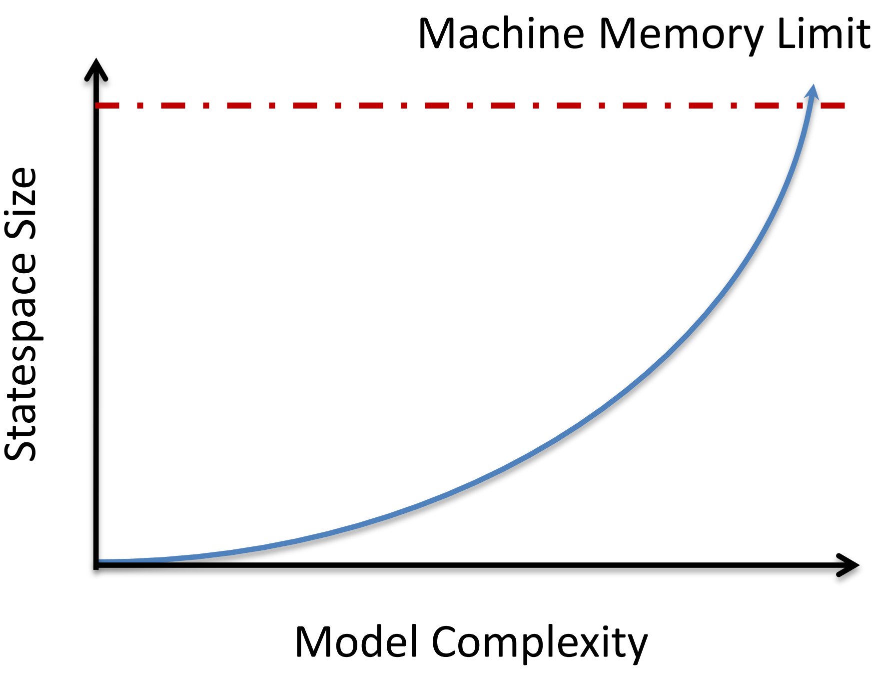
- More widely used than cognitive models
- Potentially more computationally efficient than cognitive models
- Provide less cognitive explanation
- Cannot organically produce erroneous behaviors

If this stuff is so great, why isn't everybody using it?

Limitations

Scalability:

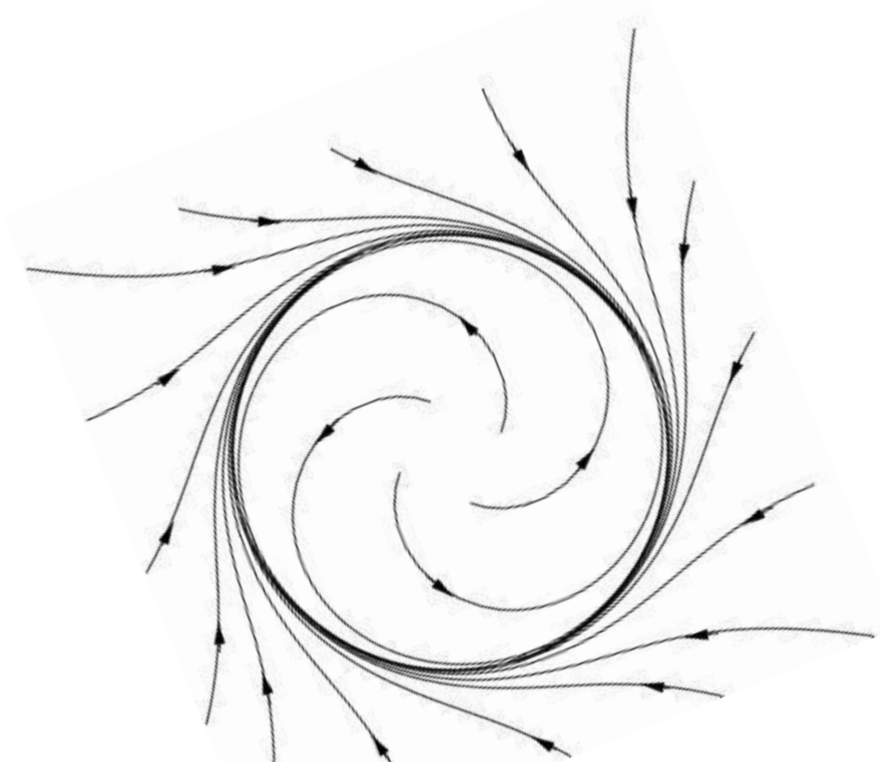
Combinatorial explosion ("the state explosion problem") limits the size of models that can be checked and the verification time



Limitations

Notation expressiveness:

It can be difficult to model concepts using formal modeling notations. Concepts such as non-linear dynamics and time can be very tricky. Clever abstraction and slicing techniques must be used.



Limitations

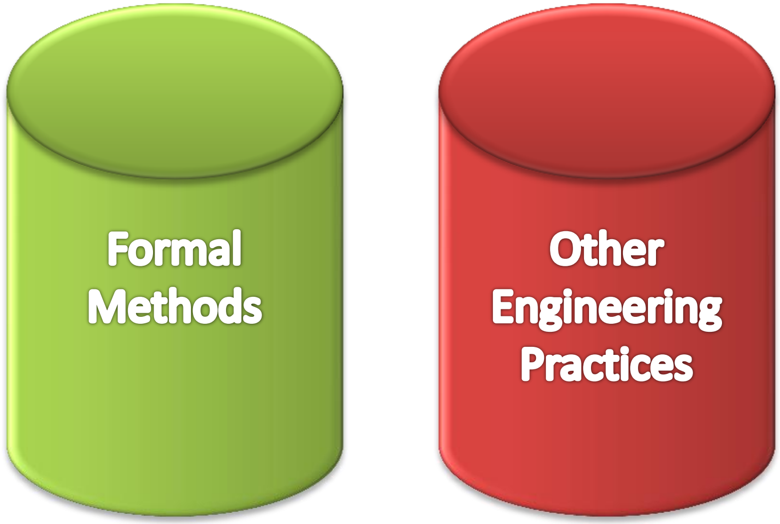
Learnability:

Formal methods can be difficult to learn and teach

Limitations

Lack of Integration:

Formal methods are not well integrated into systems engineering and industrial engineering environments



Researchers are Actively Trying to Address These Limitations

Conclusions

- Formal methods are very powerful and represent another tool in the human factors toolbox
- Formal methods can be used to evaluate human-automation interaction in a number of ways:
  - Find usability problems
  - Detect mode confusion
  - Evaluate system safety and performance
  - Evaluate the robustness of a system to human error
- Formal methods are limited and should thus be used synergistically with other techniques
- Research is actively improving form human-automation interaction analyses and integrating analysis and design techniques

For more information…

Bolton, M. L., Bass, E. J., & Siminiceanu, R. I. (2013). Using formal verification to evaluate human-automation interaction in safety critical systems, a review. *IEEE Transactions on Systems, Man and Cybernetics: Systems, 43*(3), 488-503. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6472094