NUST
Defining futures

Formal Verification
Introduction and Applications

Osman Hasan
System Analysis and Verification (SAVE) Lab
National University of Sciences and Technology (NUST)
Islamabad, Pakistan

FEUP, Uporto
July 23, 3015

Objectives

- Formal Verification
  - Why do we need it?
  - What is it?
  - How can we apply it for the analysis of real-world systems?
- Applications
  - Analyzing Thermal Management Algorithms for Multi-core Architectures
  - Reliability Analysis

System Analysis

- Does the system exhibit the desired behavior?
  - Mathematically using paper-and-pencil proof methods
  - Computer Simulations (Testing)

Paper-and-Pencil Proof Methods

- Construct a mathematical model of the system
- Mathematically verify that the system exhibits the desired characteristics
- Accurate
- Scalability
- Error-Prone

Simulation or Testing

- Construct a computer based model of the system
- Analyze the behavior of the system model under a number of test cases to deduce properties of interest

Simulation or Testing –
Example: Remainder and Quotient Program

```
var x, y, q, r: integer;
r := x; q := 0;
while y < r do {
        r := r - y;
        q := q + 1;}
return r, return q;
```

- Property: r and q represent the remainder and quotient for all values of variables x and y
- Test Cases

Input (x,y)	Output (r,q)	Property
(23,4)	(3,5)	True
(2,5)	(2,0)	True
(17,3)	(2,5)	True

- Infer: The property is true since it is found to be true for all the test vectors used

Simulation or Testing – Example: Remainder and Quotient Program

```
var x, y, q, r: integer;
r := x; q := 0;
while y < r do
    r := r - y;
    q := q + 1;
```

- Testing for (24, 4)
  - After the fifth iteration, r = 4, q = 5, y = 4, the loop ends and we get (5, 4), which is wrong
  - Fix: Replace the condition y < r with y ≤ r
- Testing for (23, -2)
  - r is always greater than x and thus we have an infinite loop
  - Fix: Constrain y to have positive values only

Simulation or Testing

- Easy to use
- May lead to wrong conclusions

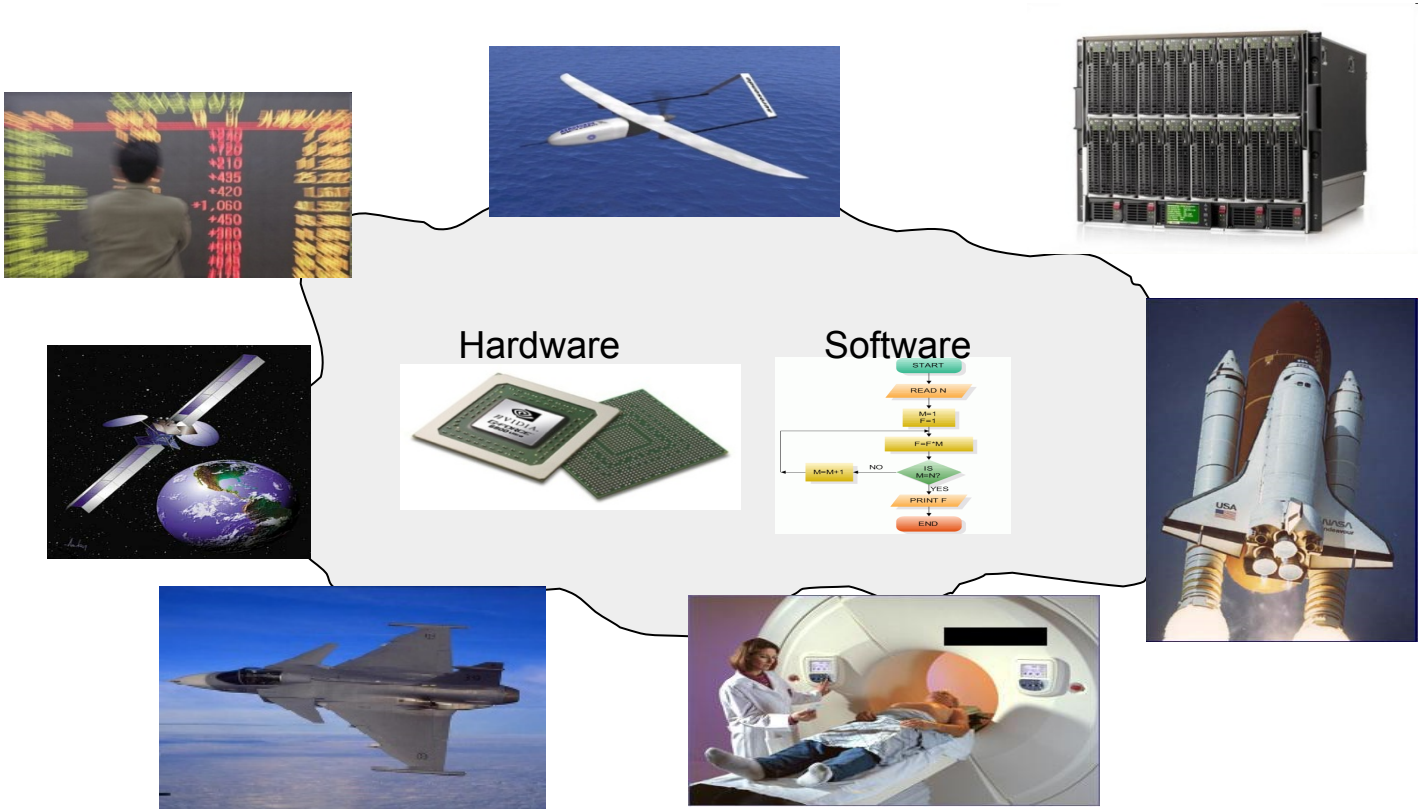
- Practically impossible to test for all possible cases when dealing with large systems
  - Suppose we want to test a 64-bit floating-point division routine. There are 2^{128} combinations.
  - At 1 test/micro second, it will take 10^{25} years

Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence.

Edsgar W. Dijkstra

System Analysis Accuracy

- Extremely Important



System Analysis Accuracy

- Faulty systems can be disastrous
- FDIV bug in Intel Pentium (60 Mhz, 90Mhz)
  - Hardware error in the floating point division unit
  - Expected precision up to 18 positions
  - in practice, only 4 positions
  - Example:
    - 5505001 / 294911
    - Answer from Intel Pentium: 18.666**00093**
    - Expected answer: 18.666**5197**

  - Resulted in net loss of approximately US $500M to the company in 1994

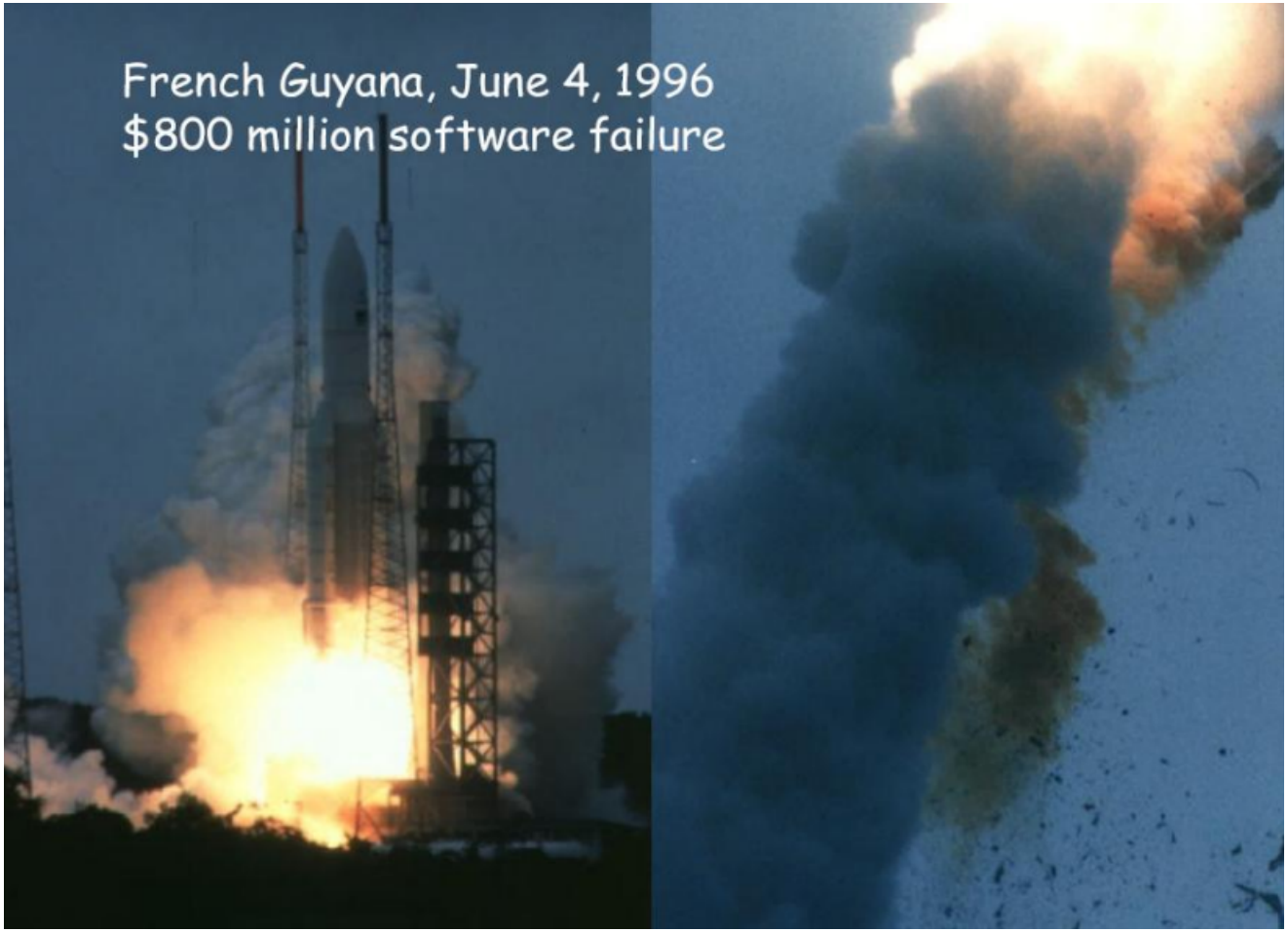
System Analysis Accuracy

- Faulty systems can be disastrous

- Therac-25
  - Software Bug in a Cancer Therapy Machine
  - 3 Deaths and 3 severe injuries between 1985-87

System Analysis Accuracy

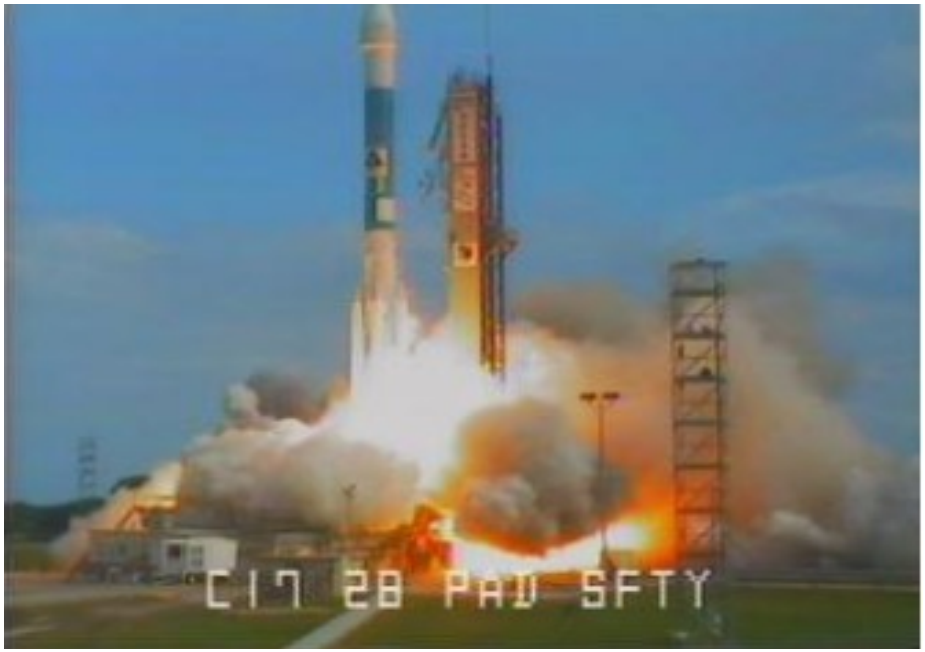
- Faulty systems can be disastrous



System Analysis Accuracy

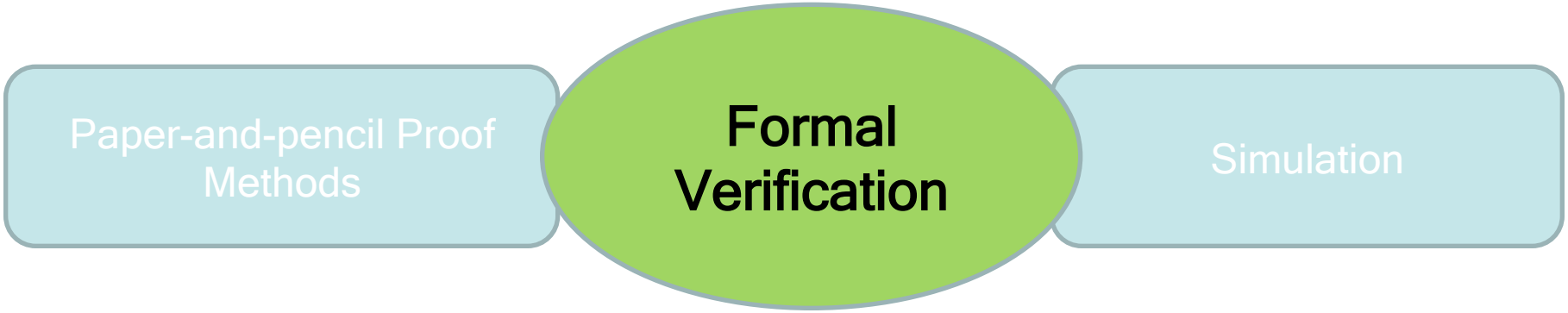
- Faulty systems can be disastrous

- Mars Climate Orbiter
  - Conversion error from English units to metric units
  - Resulted in a loss of US $125M in 1999



Formal Verification

- Bridges the gap between Paper-and-pencil proof methods and simulation



- Shares their advantages
  - As precise as a mathematical proof can be
  - Computers are used for book-keeping
- Not as straightforward to use as testing

Formal Verification

- Construct a computer-based mathematical model of the system (*implementation*)
- Use mathematical methods to check if the implementation satisfies the properties of interest (*specifications*) in a computerized environment

Formal Verification Methods

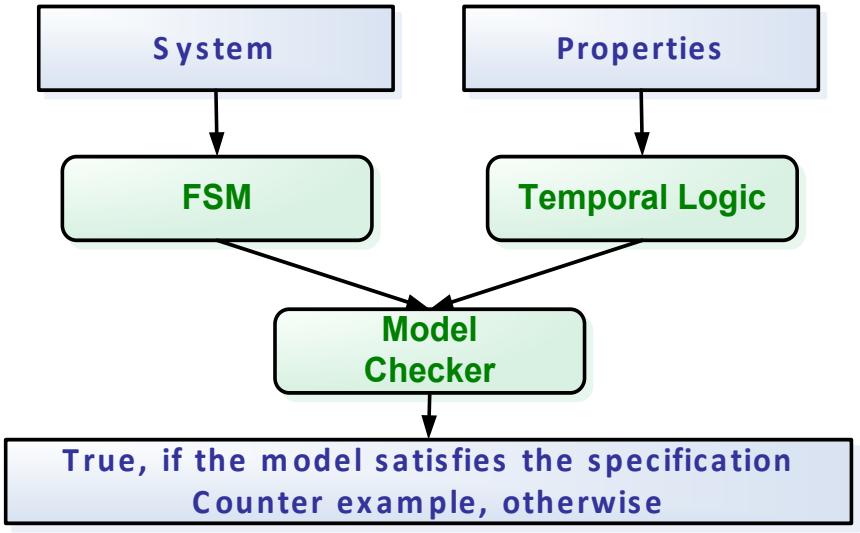
- Most widely used techniques
  - Model Checking
  - Theorem proving

Model Checking

- Typically used for verifying Concurrent Finite-State Systems
  - Rigorous testing is infeasible due to large number of possible scenarios

- Implementation
  - State-transition Graph

- Specifications
  - Temporal Logic



Temporal Logic

- Time is modeled as a sequence of states
- Atomic formulas associated with each state
- Temporal connectives allow us to refer to the future

- F: some Future state (Fp)

- X: neXt state (Xp)

- G: Globally (Gp)

- U: Until (pUq)

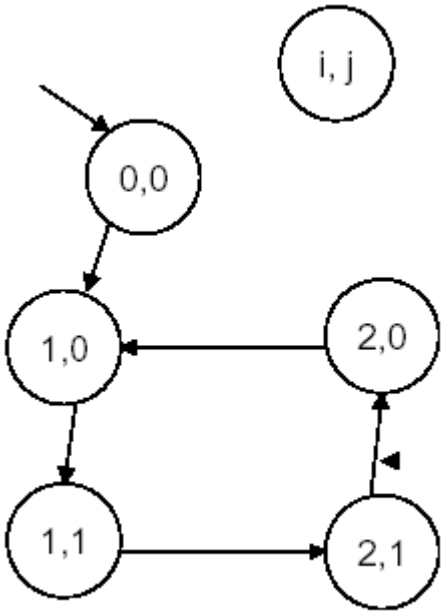
Model Checking

- The verification can be done automatically using rigorous state-exploration methods

- Counter Examples
  - In case of failing properties, counter examples are provided for debugging purposes

Model Checking -
Example – Simple Program

```
bar () {
  int i=0, j=0;
  while (i<2) {
        i++;
        j=i%2;
  }
  i=1;
}
```



- (i,j) = (0,0), (1,0), (1,1), (2,1), (2,0), (1,0)
- Property: j will eventually become 1: F(j = 1)
- Property: j is always less than or equal to 1: G(j ≤ 1)

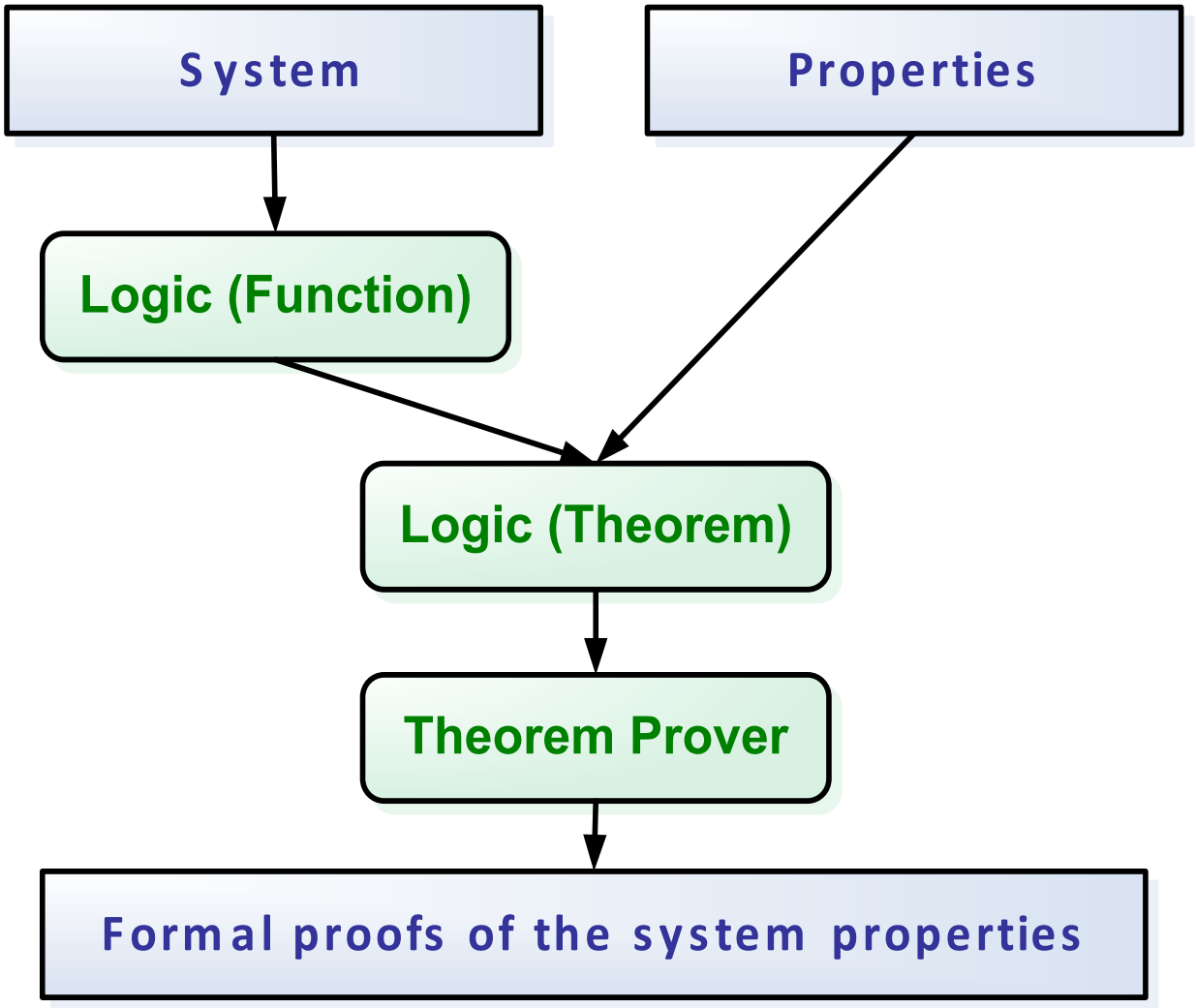
Model Checking

- Extensively used in verifying
  - Software
  - Security Protocols
  - Telecommunication Protocols
  - Digital hardware
  - Analog and mixed signal circuits
  - Microprocessors
  - Biological Systems

Model Checking

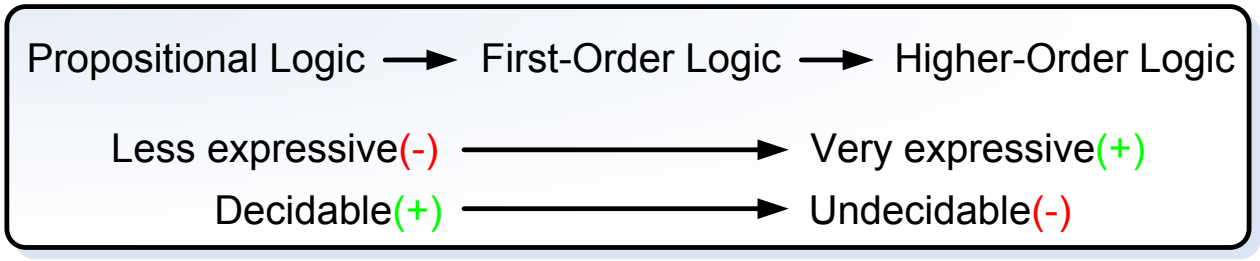
- Advantages
  - Automatic (Push button type analysis tools)
  - No proofs involved
  - Diagnostic counter examples
- Disadvantages
  - Limited expressiveness
  - State-space explosion problem
- Some commonly used Model Checking Tools
  - SPIN
  - NuSMV
  - PRISM (A probabilistic model checker)

Theorem Proving



Logic

- Study of drawing conclusions (reasoning)
  - Propositional logic
    - Supports statements that can be true or false
  - First-order logic (Predicate logic)
    - Quantification over variables (∀: For all, ∃: there exists)
  - Higher-order logic
    - Quantification over sets and functions



Theorem Prover

- A theorem prover consists of
  - A notation (Syntax)
  - A small set of fundamental axioms (facts)
    - Example: $(\neg\neg A) \Leftrightarrow A$
  - A small set of deduction rules
    - Example: Given $(A \rightarrow B)$ and A, we can deduce B
- Soundness is assured as every new theorem must be created from
  - The basic axioms and primitive inference rules
  - Any other already proved theorems (Theory Files)

Example 1: Equivalence Between Programs

```
If (!(x=y) && !(q<s)) then
    a = a + 5;
else {
    If (!(x=y)) then
        a = a - 5;
    else
        a = a +10;}
```

```
if (!a && !b) h();
else {
  if(!a) g();
  else f();
}
```

$(\texttt{if } x \texttt{ then } y \texttt{ else } z) \equiv (x \wedge y) \vee (\neg x \wedge z)$

$$(\neg a \wedge \neg b) \wedge h \vee \neg(\neg a \wedge \neg b) \wedge (\neg a \wedge g \vee a \wedge f)$$

$$\Leftrightarrow$$

$$a \wedge f \vee \neg a \wedge (b \wedge g \vee \neg b \wedge h)$$

```
If (x=y) then
    a = a + 10;
else {
    If (q<s) then
        a = a - 5;
    else
        a = a +5;}
```

```
if(a) f();
else {
  if(b) g();
  else h();
}
```

Theorem Proving

- Extensively used in verifying
  - Compilers
  - Floating-point algorithms
  - Digital Signal Processing systems
  - Optical systems
  - Wireless sensor networks
  - Kinematic Analysis of Robots

Theorem Proving

- Advantages
  - High expressiveness
  - No risk of mistakes (human errors)
  - Some parts of the proofs can be automated
- Disadvantages
  - Detailed and explicit human guidance required for verifying real-world systems
  - The state-of-the-art is limited
- Theorem Proving Tools
  - ACL2 (First-order Logic)
  - Coq (Higher-order Logic)
  - HOL (Higher-order-logic)

Applications

- Analyzing Thermal Management Algorithms for Multi-core Architectures
  - Model Checking
- Reliability Analysis
  - Theorem Proving

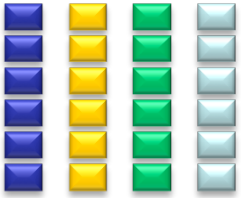
Analyzing Thermal Management Algorithms for Multi-core Architectures using nuXmv

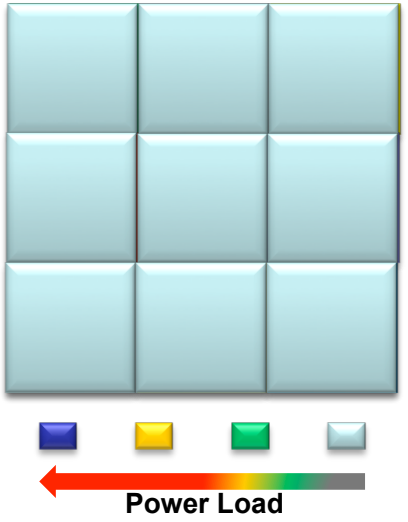
Syed Ali Asadullah Bukhari[1], Faiq Khalid Lodhi[1], **Osman Hasan**[1], Muhammad Shafique[2] and Jorg Henkel[2]

[1]National University of Sciences & Technology, Islamabad, Pakistan
[2]Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

Thermal Hotspots

- Multiple-cores per chip
  - High Performance
  - Verification Re-usabilty
  - Imbalanced Task Mapping
    - Thermal Hot Spots

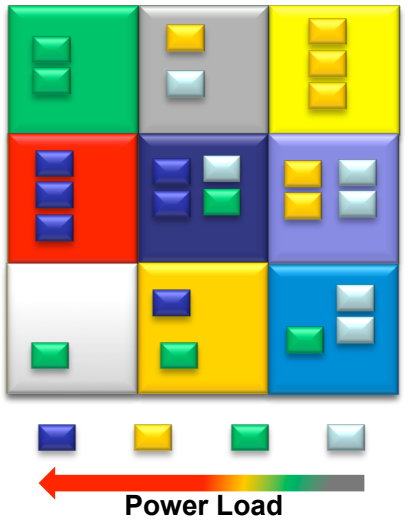




Dynamic Thermal Management

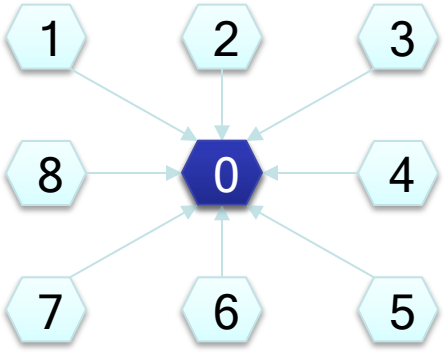
- Dynamic Thermal Management (DTM): An online scheme to manage the heating issues

- Distributes power and the resulting heat evenly by performing *task migration* from hotter regions to cooler ones



Centralized vs Decentralized DTM

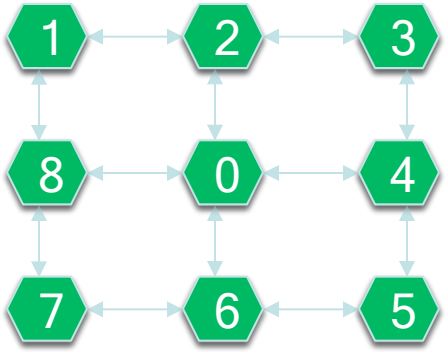
Central



Central Agent

- A central agent responsible for thermal management
- Leads to significant communication overhead

Decentralized



Distributed Agents

- Every node acts as an agent managing immediate neighbors only

Analysis of Decentralized DTM Techniques

- To ensure that the given DTM technique would reach the no hot-spot (stability) state
- To judge the effect of various parameters on time to stability of the DTM technique
- The decentralized nature makes the analysis quite complex

Decentralized DTM Analysis Techniques

Analytical

- Completeness

- Human-Error Prone
- Practically impossible to analyze large and complex systems

Simulation

- Quick insights about the working of the given dDTM scheme

- In-exhaustive
- Impossible to predict all corner cases

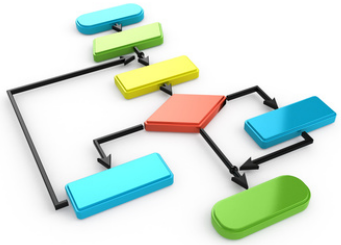
Emulation

- Real time Hardware/Software system interaction

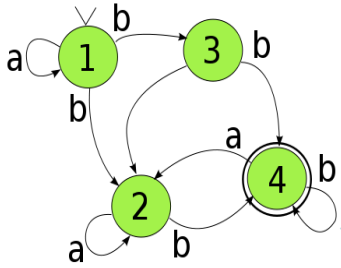
- High Cost
- In-exhaustive
- Impossible to predict all corner cases

- None of these techniques offers a complete and accurate analysis

Proposed Verification Methodology



DTM Algorithm



State Machine

Working Description

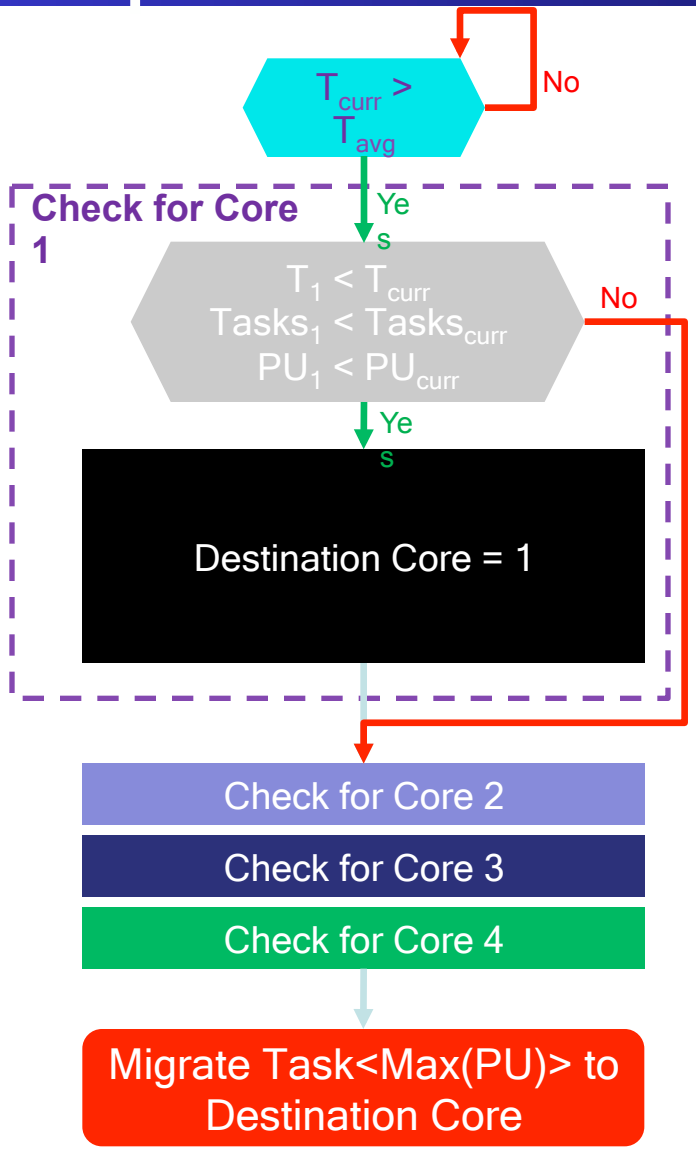
Properties

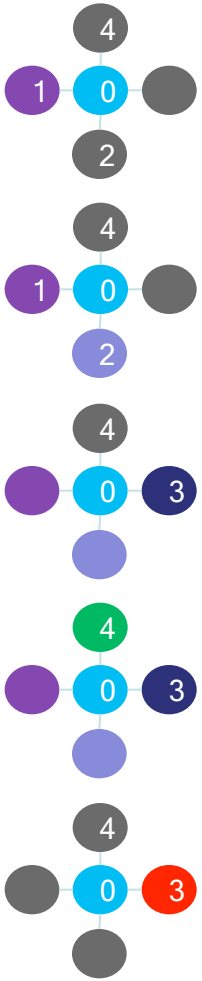
Property True / False
with Counter Example

nuXmv

- New set of types, namely 'Integer' and 'Reals'
- New model checking algorithms
  - SMT-based techniques
  - Abstraction based techniques

Task Migration for Thermal Hot Spot Reduction Algorithm



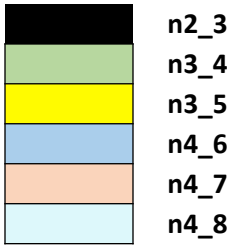


- Two Issues identified during the formal modeling phase
  - Ambiguity when multiple cores are trying to migrate the task on the same core
  - A core already with a hot spot can be assigned new tasks

Formal Modeling in nuXmv

- 6 Core Types were modeled

0	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	26
27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52	53
54	55	56	57	58	59	60	61	62
63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80



0	1	2	3	4
9	10	11	12	13
18	19	20	21	22
27	28	29	30	31
36	37	38	39	40

np_q: p: number of direct neighbors and q: number of second-degree neighbors

- Non-Deterministic Assignments
  - Number of Tasks in a core (0-14)
  - Number of Power units/Task (1-4)
- Temperature, modeled as a real number, is directly proportional to the number of power units

Stability

- Eventually Stability is achieved
  - We always eventually reach a condition when the temperature of all the cores will eventually be less than or equal to the estimated average temperature of the chip.

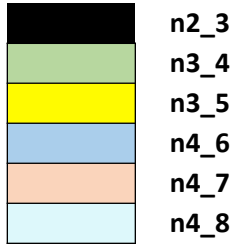
$$\mathbf{GF}(core_0.T_0 <= core_0.T_{avg}\ \&\ core_1.T_0 <= core_1.T_{avg}\ \&\ \ldots\ldots\ \&\ core_{80}.T_0 <= core_{80}.T_{avg})$$

Time to Stability

Experimental Setup					n2_3	n3_4	n3_5	n4_6	n4_7	n4_8
T0	T1	T2	T3	T4	0	1	2	10	11	20
56					49	52	51	63	67	62
56	56				53	53	54	61	63	61
56		56			61	57	55	71	73	69
56			56		-	55	57	72	79	80
56				56	-	-	-	89	87	91
56	56	56			63	59	53	94	92	85
56	56		56		-	61	54	91	98	96
56	56			56	-	57	51	107	114	113
56		56	56		-	62	59	117	112	105
56		56		56	-	58	57	103	109	112
56			56	56	-	60	65	100	105	102
56	56	56	56		-	-	-	115	120	117
56	56	56		56	-	-	-	119	124	118
56	56		56	56	-	-	-	121	125	131
56		56	56	56	-	-	-	132	129	141
56	56	56	56	56	-	-	-	129	131	137

Core number

0	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	26
27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52	53
54	55	56	57	58	59	60	61	62
63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80



Time To Stability

Experimental Setup					n2_3	n3_4	n3_5	n4_6	n4_7	n4_8
T0	T1	T2	T3	T4	0	1	2	10	11	20
56					49	52	51	63	67	62
56	56				53	53	54	61	63	61
56		56			61	57	55	71	73	69
56			56		-	55	57	72	79	80
56				56	-	-	-	89	87	91
56	56	56			63	59	53	94	92	85
56	56		56		-	61	54	91	98	96
56	56			56	-	57	51	107	114	113
56		56	56		-	62	59	117	112	105
56		56		56	-	58	57	103	109	112
56			56	56	-	60	65	100	105	102
56	56	56	56		-	-	-	115	120	117
56	56	56		56	-	-	-	119	124	118
56	56		56	56	-	-	-	121	125	131
56		56	56	56	-	-	-	132	129	141
56	56	56	56	56	-	-	-	129	131	137

0	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	26
27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52	53
54	55	56	57	58	59	60	61	62
63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80

Temperature of core is 56

Temperature of core < 56

0	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	26
27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52	53
54	55	56	57	58	59	60	61	62
63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80

T0, T1 T2 T3 and **T4** represents the temperature of core and its respective neighbors

Formal Reliability Analysis

NUST
Defining Futures

O. Hasan[1], W. Ahmed[1] S. Tahar[2] and M.S. Hamdi[3]

[1]National University of Sciences and Technology, Islamabad Pakistan

[2]Concordia University, Montreal, Canada

[3]Ahmed Bin Mohammed Military College, Doha, Qatar

Concordia University
Hardware Verification Group

Reliability

- A measure of the continuity of service
- Probability that a system performs its intended function until some time t without failing

$$R(t) = Pr(X > t) = 1 - Pr(X \leq t) = 1 - F_X(t)$$

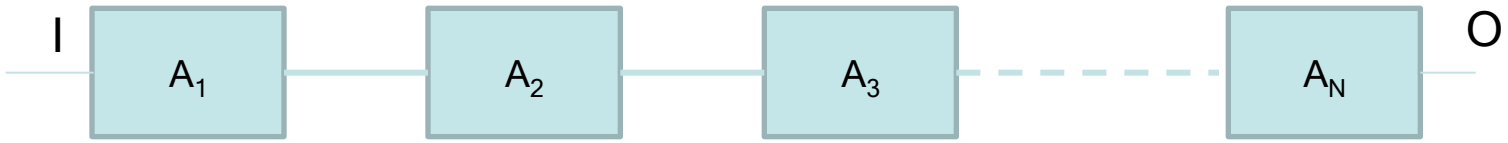
- X: random variable that models the time to failure of the system
  - Commonly used Distributions
    - Exponential
    - Weibul

Reliability Block Diagrams

- Used to asses the reliability of a complex system
  - Partition the system into sub-blocks and connectors (RBD)
  - Find the failure rates of sub-blocks
  - Judge the failure characteristics of the overall system
    - failure rates of individual components
    - RBD configuration

- The overall system failure happens if all the paths for successful execution fail
  - Add more parallelism to meet the reliability goals

Series Reliability Block Diagram

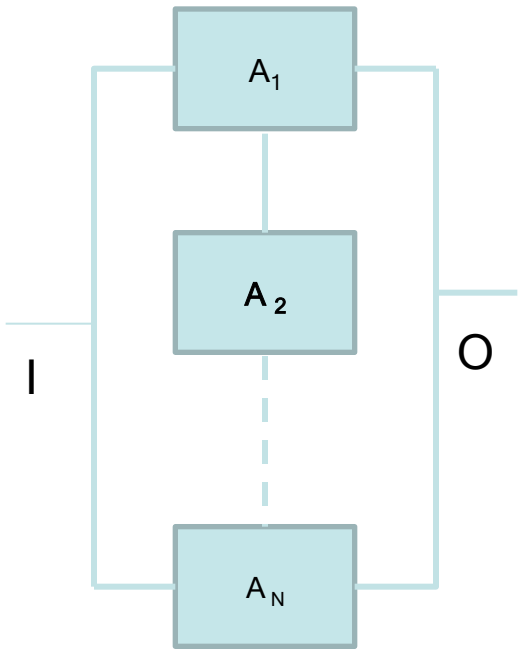


- The overall system is reliable only if all of its components are functioning reliably

$$R_{series}(t) = Pr(A_1(t) \cap A_2(t) \cap A_3(t) \cdots \cap A_N(t)) = \prod_{i=1}^{N} R_i(t)$$

Where $A_i(t)$ are the mutually independent events corresponding to *i serially-connected* components

Parallel Reliability Block Diagrams

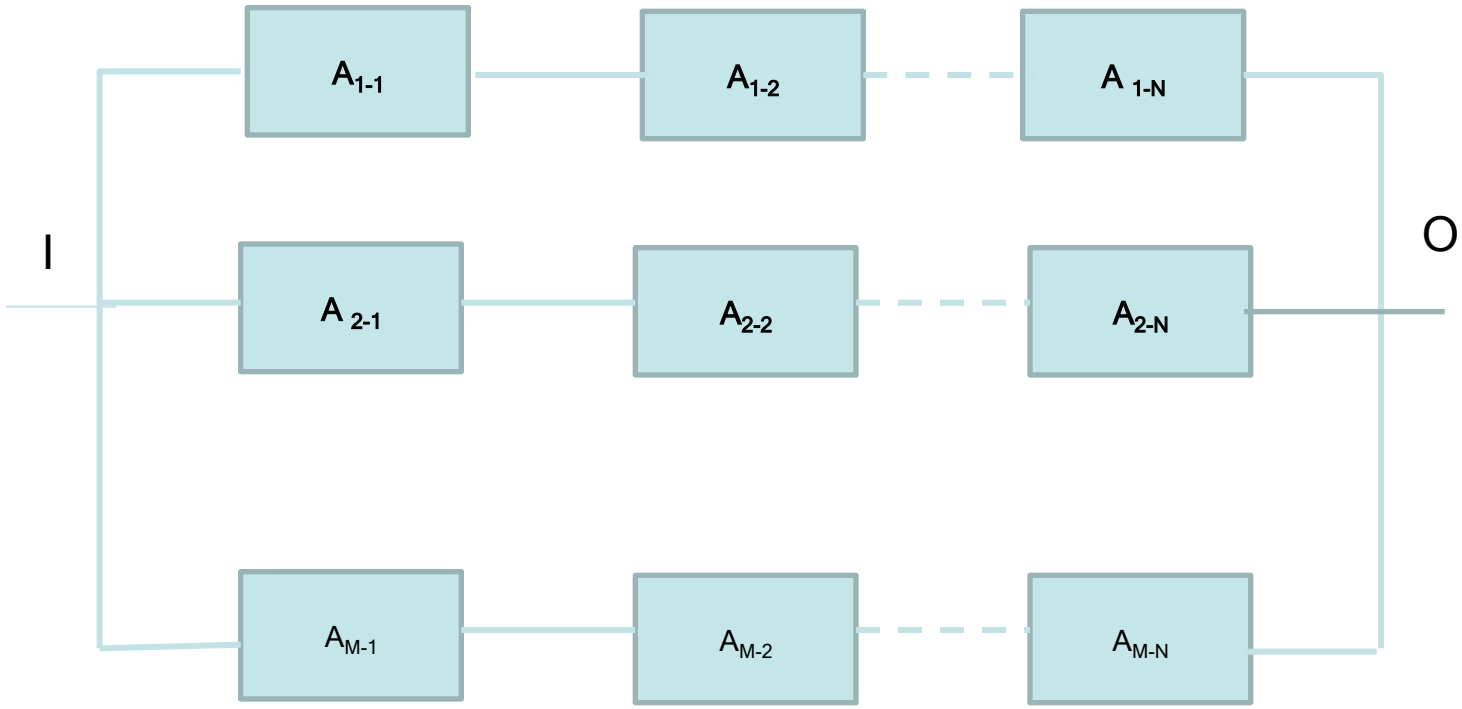


- The overall system reliability mainly depends on the component with the maximum reliability

$$R_{parallel}(t) = Pr(A_1 \cup A_2 \cup A_3 \cdots \cup A_N) = 1 - \prod_{i=1}^{N}(1 - R_i(t))$$

Where $A_i(t)$ are the mutually independent events corresponding to i *parallel-connected* components

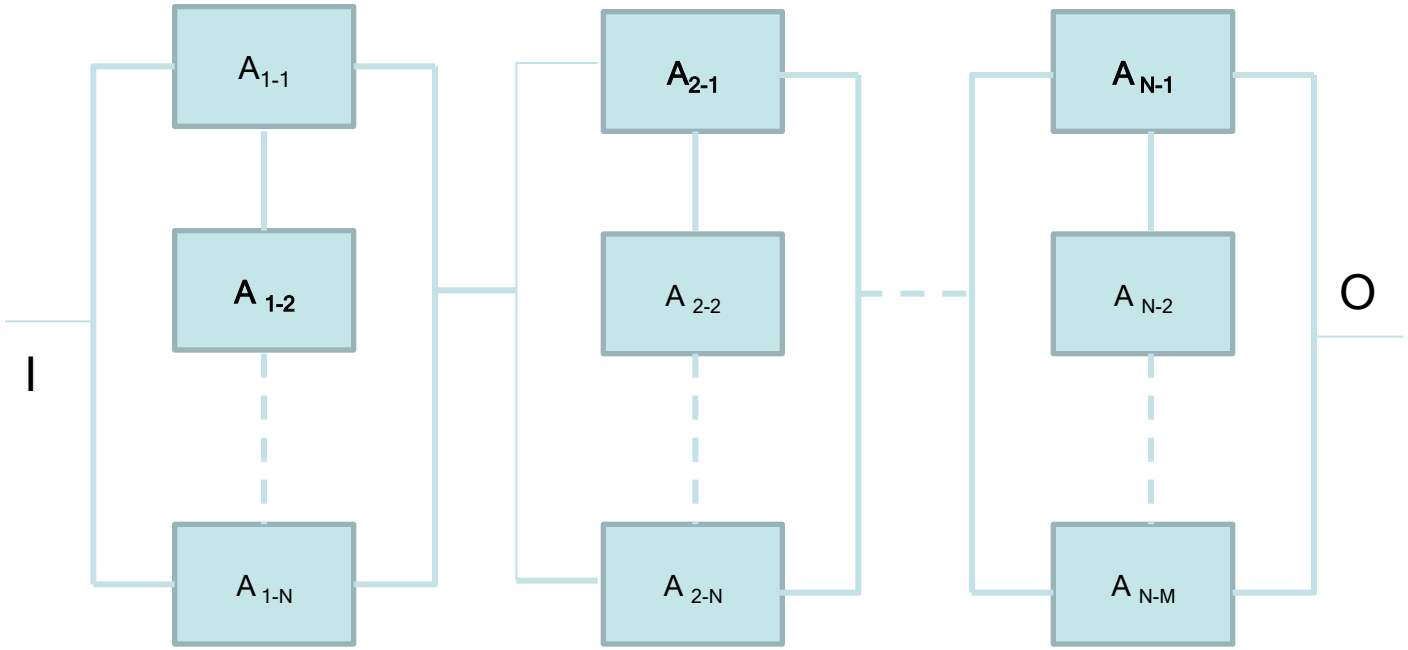
Series-Parallel Reliability Block Diagrams



- A combination of both series and parallel RBD

$$R_{Series-Parallel} = Pr(\bigcap_{i=1}^{N} \bigcup_{j=1}^{M} A_{ij}) = \prod_{i=1}^{N}(1 - \prod_{j=1}^{M}(1 - R_{ij}(t)))$$

Parallel- Series Reliability Block Diagrams



$$R_{Parallel-Series} = Pr(\bigcup_{i=1}^{M} \bigcap_{j=1}^{N} A_{ij}) = 1 - \prod_{i=1}^{M}(1 - \prod_{j=1}^{N}(R_{ij}(t)))$$

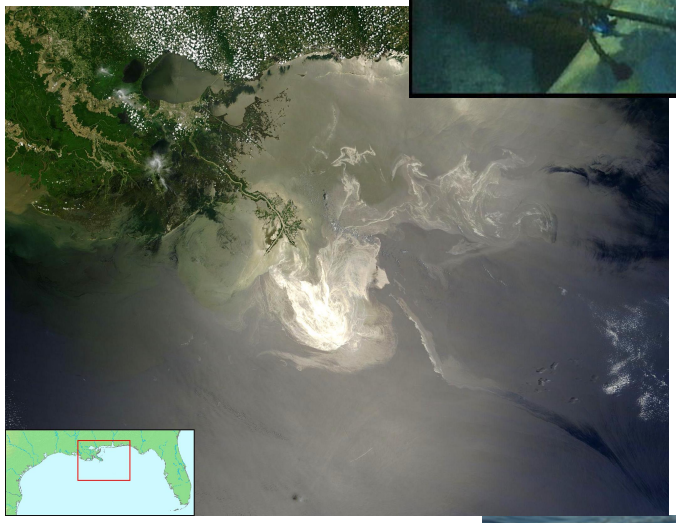
Example: Reliability Analysis of Oil and Gas Pipelines

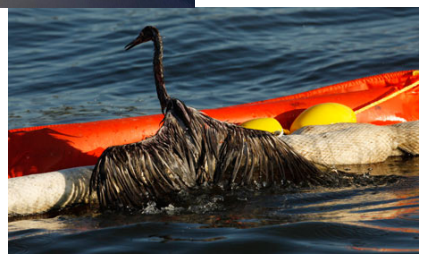
- There are tens of thousands of miles long oil and gas pipelines around the world
  - Some of them aging and are becoming more and more susceptible to failures

- Very important to rigorously analyze their reliability and thus plan timely replacements and maintenance

Methane Gas Leakage on the Deepwater Horizon oil rig – April 2010

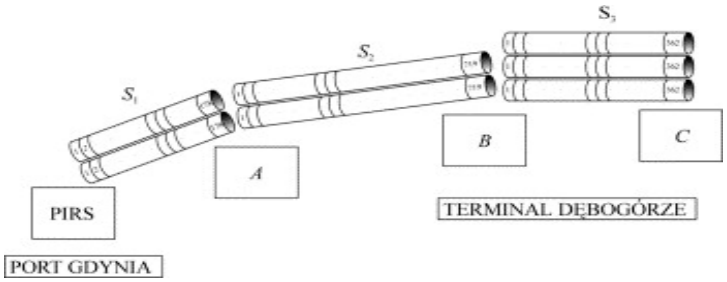
- Killed 11 workers
- Destroyed and sank the rig
- Caused millions of gallons of oil to pour into the Gulf of Mexico
- Took three months to bring the situation under control
- Damage to marine and wildlife habitats

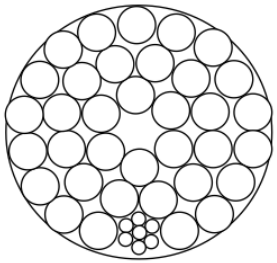


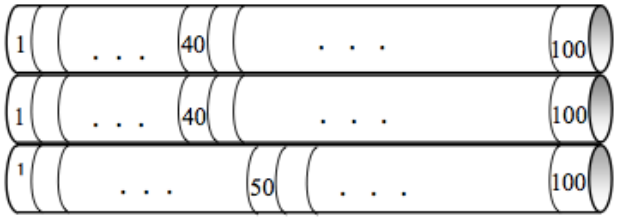


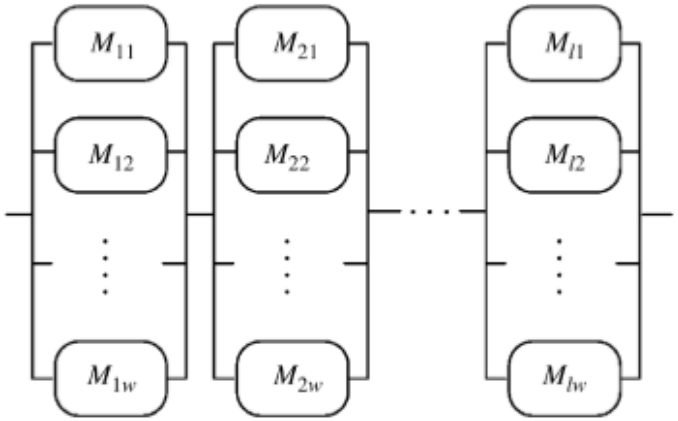
Reliability Analysis of Pipelines

- Partitioning the given pipeline into segments and constructing its equivalent reliability block diagram (RBD)





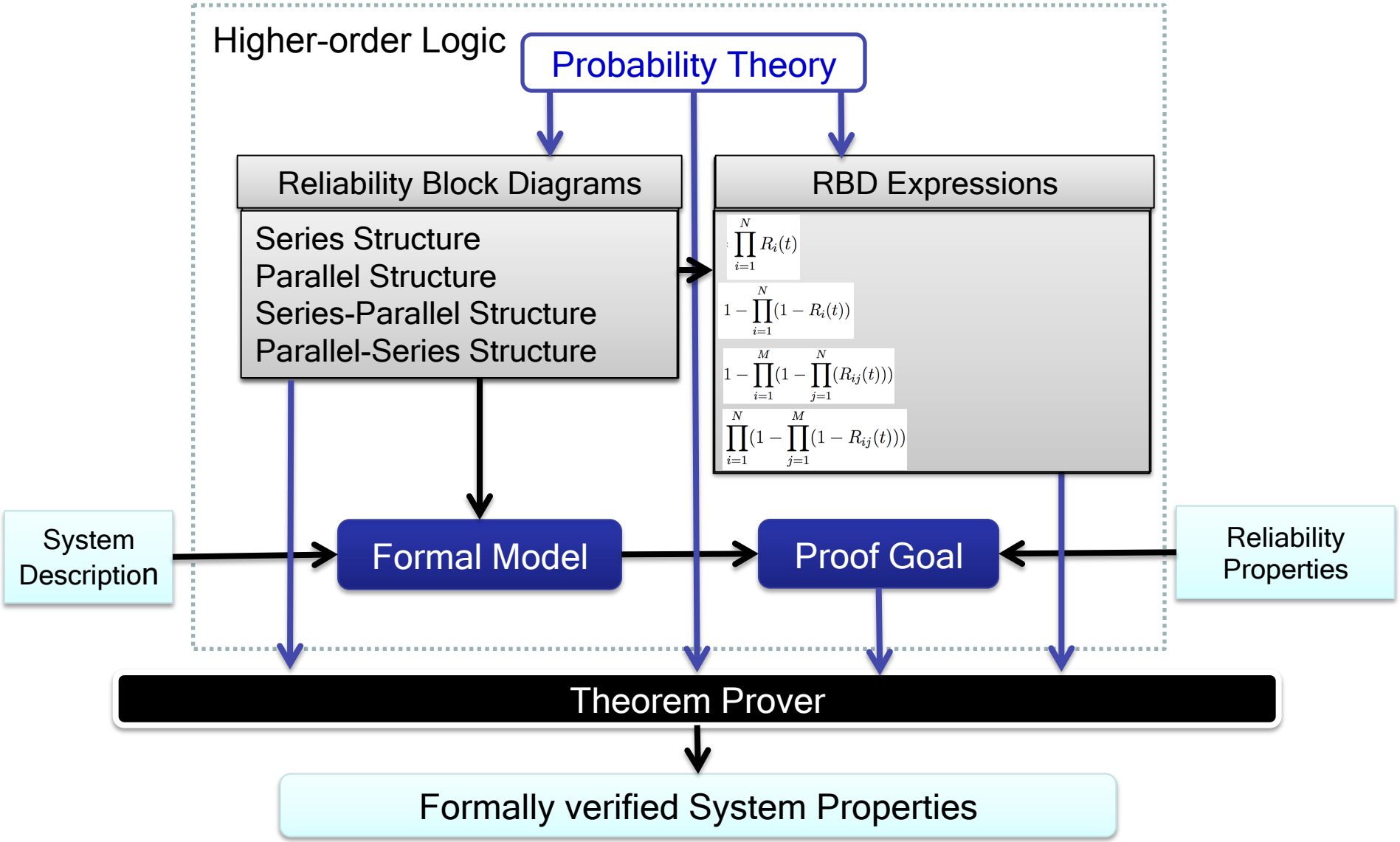




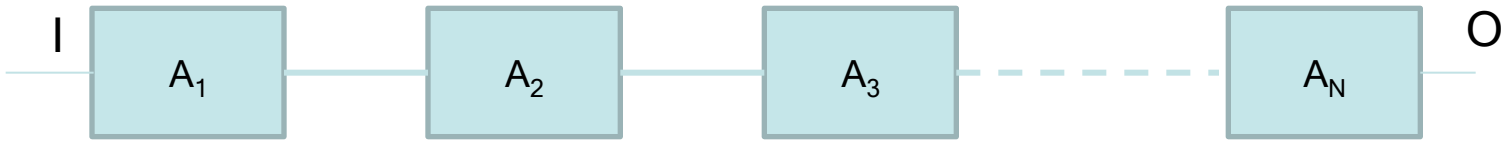
Analysis Techniques

Criteria	Paper-and-Pencil Proof	Simulation	Model Checking	Higher-order-logic Proof Assistants
Expressiveness	☑	☑	☒	☑
Accuracy	☑?	☒	☑	☑
Automation	☒	☑	☑	☑

Formal Reliability Analysis Methodology



Formalization of Series RBD



Definition 1: `⊢ ∀ p L. series_struct p L = inter_list p L`

$$R_{series}(t) = Pr(A_1(t) \cap A_2(t) \cap A_3(t) \cdots \cap A_N(t)) = \prod_{i=1}^{N} R_i(t)$$

Theorem 1:
```
⊢ ∀ p L. prob_space p ∧ (events p = POW (p_space p)) ∧
    1 ≤ LENGTH L ∧ mutual_indep p L ⇒
      (prob p (series_struct p L) = list_prod (list_prob p L))
```

Other RBDs

Definition 2: $\vdash \forall$ L . parallel_struct L = union_list L

Theorem 2: $\vdash \forall$ p L. (prob_space p) $\wedge$
(events p = POW (p_space p)) $\wedge$
(1 $\leq$ LENGTH L) $\wedge$ (mutual_indep p L) $\wedge$
($\forall$ x'. MEM x' L $\Rightarrow$ x' $\in$ events p) $\Rightarrow$
(prob p (parallel_struct L) =
1 - list_prod (one_minus_list (list_prob p L)))

Definition 3: $\vdash \forall$ p L. parallel_series_struct p L =
parallel_struct (list_inter_list p L)

Theorem 3: $\vdash \forall$p L. (prob_space p) $\wedge$
(events p = POW (p_space p)) $\wedge$
($\forall$z. MEM z L $\Rightarrow$ ~NULL z) $\wedge$ (mutual_indep p (FLAT L)) $\wedge$
($\forall$x'. MEM x'(FLAT L) $\Rightarrow$ x' $\in$ events p) $\Rightarrow$
(prob p (parallel_series_struct p L) =
1 − list_prod (one_minus_list(list_rel_list_prod p L)))

Other RBDs

Definition 4: $\vdash \forall$ p L. series_parallel_struct p L = series_struct p (list_union_list L)

Theorem 4: $\vdash \forall$ p L. (prob_space p) $\wedge$
(events p = POW (p_space p)) $\wedge$
($\forall$z. MEM z L $\Rightarrow$ $\sim$NULL z) $\wedge$ (mutual_indep p (FLAT L)) $\wedge$
($\forall$x'. MEM x'(FLAT L) $\Rightarrow$ x' $\in$ events p) $\Rightarrow$
(prob p (series_parallel_struct p L) =
list_prod (one_minus_list (list_compl_rel_list_prod p L)))

- Formalization took about 300 man-hours and more then 8000 lines of HOL4 proof script

Reliability Analysis Case Studies

- Virtual Data Centers
- Oil and Gas Pipelines
- Telecommunication Networks
- Smart Grids

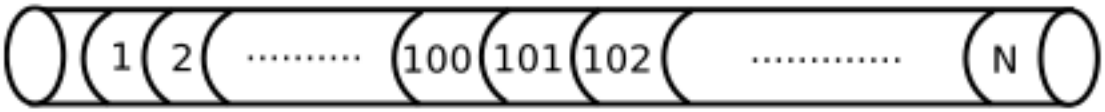
Case Study: A Simple Pipeline Structure

- A 60 segment pipeline with exponentially distributed failure rates, can be sub-divided into 3 categories according to their failure rates
  - 30 segments with $\lambda = 0.0025$
  - 20 segments with $\lambda = 0.0023$
  - 10 segments with $\lambda = 0.015$

Zhang, Z., Shao, B.: Reliability Evaluation of Different Pipe Section in Different Period. In: Service Operations and Logistics, and Informatics, IEEE (2008) 1779- 1782

Case Study: Formal Analysis

- The proposed approach for reliability analysis of pipelines allows us to formally verify generic expressions involving
  - any number of segments
  - arbitrary failure rates



Case Study: Formal Verification

Theorem 6: *Series Pipeline System*

```
⊢ ∀ p L x C. prob_space p ∧ (events p = POW (p_space p)) ∧
     0 ≤ x ∧ 2 ≤ LENGTH (rel_event_list p L x) ∧
     mutual_indep p (rel_event_list p L x) ∧
      list_exp p C L ∧ (LENGTH C = LENGTH L) ⇒
      (pipeline p (rel_event_list p L x) = exp (-list_sum C * x))
```

- The reasoning was very straightforward as we built upon our foundational results
  - About 100 lines of HOL code
- All the variables are universally quantified
  - Specialized to obtain the reliability of any given pipeline with series components
- Guaranteed correctness due to the involvement of a sound theorem prover
  - All the required assumptions for the validity of the result are accompanying the theorem

Objectives

- Formal Verification
  - Why do we need it?
    - Exact Answers (Useful for the analysis of Safety-critical systems)
  - What is it?
    - Mathematically reason about properties of a system using a computer-based tool
  - How can we apply it for the analysis of real-world systems?
    - Mathematically model the system (Implementation)
    - Mathematically model the desired properties (Specification)
    - Use tool support to mathematically prove that the specification holds for your implementation

Conclusions

- Formal Verification is not an alternative to simulation
  - Both techniques have to play together for a successful analysis framework
- Less critical sections of the system
  - Simulation/Testing
- Critical sections of the system that can be expressed as a FSM
  - Model Checking
- Critical sections of the system that cannot be handled by Model Checking
  - Theorem Proving

Thanks!

- For More Information
  - Visit our website
    - http://save.seecs.nust.edu.pk
  - Contact
    - osman.hasan@seecs.nust.edu.pk