



Verification of Statechart Models

Type: Research



What is Formal Verification?

Formal verification is the process of checking whether a design satisfies some requirements using formal mathematical methods.

In some software and hardware systems (like Aviation systems, Medical systems, etc.), ensuring that the system is “CORRECT” is of supreme importance.

Errors are difficult to detect using standard testing techniques and are very expensive.

Formal Verification techniques come to our aid here. The correctness of the (software or hardware) system is mathematically proven. Hence it can be said to be always reliable!

Formal verification requires Formal Specification

Verification checks if the design possesses certain properties. The properties are mostly obtained from the system's specification.

System specification and requirements in natural language => large amount of ambiguity

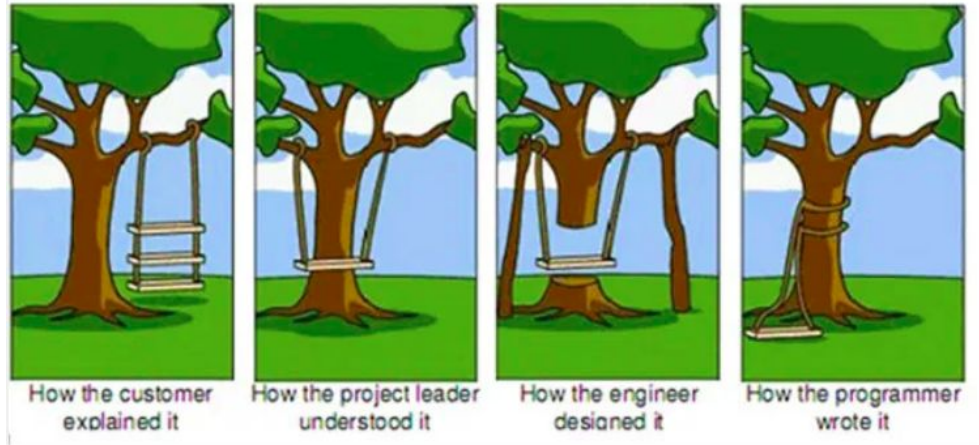
2 initial steps are essential:

1. Formal description of the system
2. Formal specification of the required properties

Defect : system does not fulfill one of the properties.

Correct : satisfies all properties obtained from its specification.

So correctness is always relative to a specification, and is not an absolute property of a system



Need for Statecharts

A **statechart** is a visual formalism for the specification of complex systems. Statechart diagrams are one of the five UML diagrams used to model the dynamic nature of a system. They define different states of an object during its lifetime and these states are changed by events.

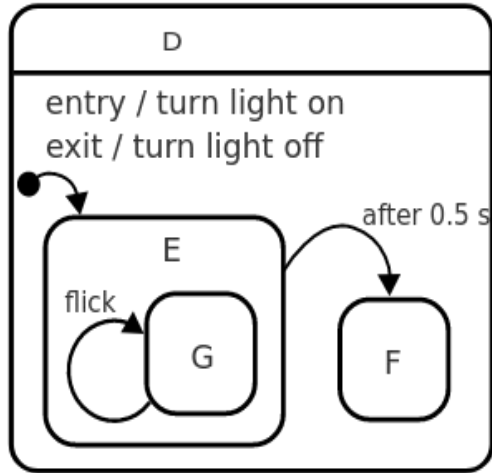
Primary motivation was to overcome the limitations of conventional state machines in describing complex systems.

Proliferation of states with increasing system size and complexity.

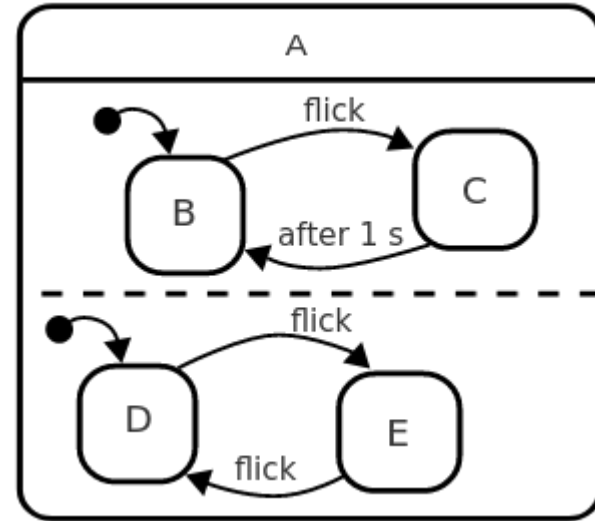
Lack of means of structuring complex systems.

Extends state machines along three orthogonal dimensions – **hierarchy**, **concurrency** and **communication** - creating a compact visual notation which allows designers to structure and modularise system specifications.

Statechart: Examples



Hierarchy



Concurrency

Statechart Model Elements

Every statechart model contains 4 elements:-

- S: Set of one or more states
- sc: Single unique state which will contain all elements such as states, transitions etc for the statechart.
- A: Transitions to execute i.e. Entry, Exit, make a transition.
- E: To trigger the transitions by an external interaction by the model.

Every state is represented by 9 elements:-

- n: Name of the state
- p: unique parent
- V: set of variables - static, local, parameter depending on their scope.
- Entry action
- Exit action
- T: Set of transitions
- C: Child state set
- h: History marker
- i: Initial sub-state

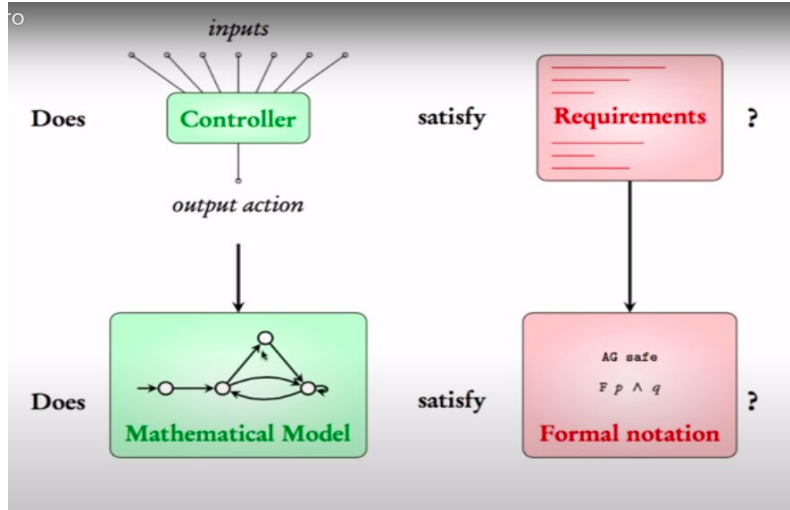
Every transition is represented by 7 elements:-

- n: Name of the transition
- p: unique parent state p for the transition
- src: source state
- e: event e which triggers the transition
- g: guard statement for the transition
- a: entry/exit
- dest: Destination state

Statechart Transitions

- ❑ Transitions can be guarded
- ❑ States can have multiple transitions for the same event
- ❑ Transitions can happen automatically
- ❑ Transitions can be delayed
- ❑ History

Model Checking



Checking if the mathematical model satisfies the properties **is equivalent to** verifying if the system satisfies the requirements.

Given an

1. input model
2. property in temporal logic(logic specifying properties over time)

A model checker determines whether the property holds, by **exploring all possible system states** in a brute-force manner and examining all possible system scenarios in a structured manner.

It returns a **counterexample trace** in case the property fails.

In this way, it can be shown that a given system model truly satisfies a certain property.

Model Checking - Phases

Modelling Phase:

1. Model the system model description language

System represented as transition diagrams in framework.

Statechart → **transition system**
Flatten the hierarchy.

2. Formalize property to be checked using **property specification language**. (LTL, CTL, OBDD)

Running Phase:

1. Solely algorithmic approach.
2. Validity of the property under consideration is checked in all states of the system model
3. Algorithm varies based on how the properties are defined

Analysis Phase:

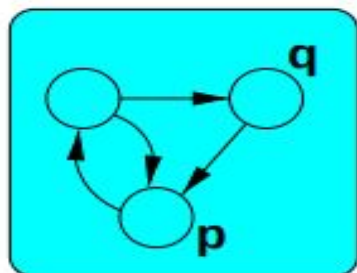
1. Property satisfied? → check next property (if any);
2. Property violated? →
 - a. analyze generated counterexample by simulation;
 - b. refine the model, design, or property;
 - c. repeat the entire procedure.
 - d.
3. Out of memory? → try to reduce the model and try again.

temporal formula

$G(p \rightarrow Fq)$

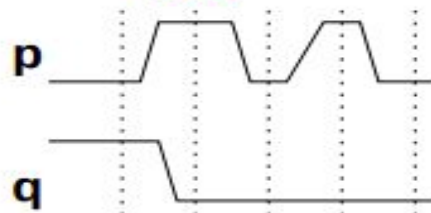
Model
Checker

yes!



finite-state model

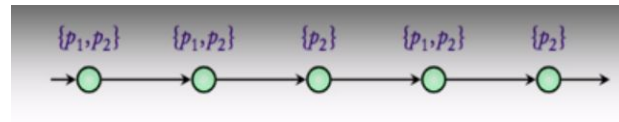
no!



counterexample

Model Checking - Property Satisfaction

1. A standalone property is called an atomic property (p).
2. The set $AP = p_1, p_2 \dots p_k$ represents the set of all atomic properties.
3. Every state in the transition diagram satisfies a subset of properties in AP. Thus for each state $S \subseteq 2^{AP}$ represents the set of properties it satisfies.
4. If a trace of the transition system is mapped to the properties the states in the trace satisfy $t(S)$, it will result in an infinite word over the powerset. ($\Sigma = 2^{AP}$).
5. A property $\phi = f(p_1, p_2 \dots p_k)$ itself is a subset of the set of infinite words over the power set.
6. A transition system is said to satisfy a property, if the set of all possible traces (from all initial states) in the transition system is a subset of AP-INF.
 $T(S) \subseteq AP - INF$



AP-INF = set of **infinite words** over $PowerSet(AP)$

Property 1: p_1 is always true

$\{ A_0 A_1 A_2 \dots \in AP-INF \mid \text{each } A_i \text{ contains } p_1 \}$

$\{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \dots$
 $\{ p_1 \} \{ p_1, p_2 \} \{ p_1 \} \{ p_1, p_2 \} \{ p_1 \} \{ p_1, p_2 \} \dots$

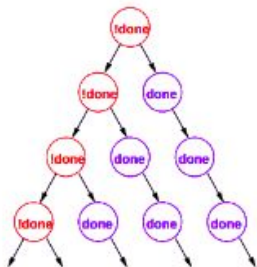
Model Checking - Temporal logic

Express properties of “Reactive Systems”

- Non terminating behaviours
- without explicit reference to time.

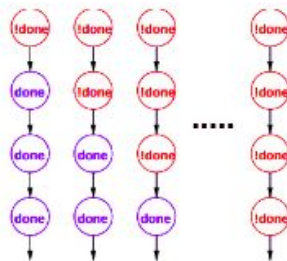
The execution of any model can be viewed in 2 broad ways:

An infinite computation tree:



CTL used

A infinite set of paths



LTL used

LTL:

Linear Time Temporal Logic

1. Interpreted over each path
2. linear model of time
3. temporal operators

CTL:

Computation Tree Logic

1. Interpreted over computation tree
2. branching model of time
3. temporal operators plus path quantifiers

Model Checking - Bottle neck

Once the property is defined, an exhaustive check of the property is done over all states of the flattened statechart.

If **property holds for all** -> **Model satisfies the property**. However,

1. Exhaustive analysis may require to store all the states
2. The state space may be exponential in the number of components
3. **State Space Explosion**: too much memory required

The solution: Symbolic Model Checking:

1. Symbolic representation
2. Different search algorithms (like Fixed Point Model checking, Bounded Model checking)

Example1: LTL + Bounded Model Checking

LTL

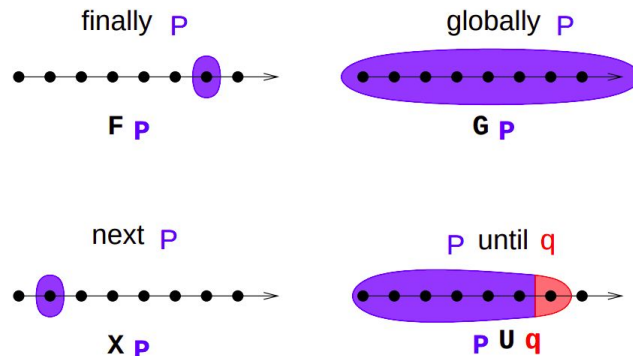
$\phi := \text{true} \mid p_i \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid X\phi \mid \phi_1 U \phi_2$

$p_i \in AP$ $\phi_1, \phi_2 : \text{LTL formulas}$

► $\phi_1 \rightarrow \phi_2: \neg\phi_1 \vee \phi_2$ **(Implies)**

► $F\phi: \text{true} U \phi$ **(Eventually)**

► $G\phi: \neg F\neg\phi$ **(Always)**



Some Properties as LTL Formulae:

- **Safety:** “it never happens that a train arrives and the bar is up”
 $G\neg(\text{train-arrives} \wedge \text{bar-up})$
- **Liveness:** “if input, then eventually output”
 $G(\text{input} \rightarrow F \text{output})$

Example1: LTL + Bounded Model Checking

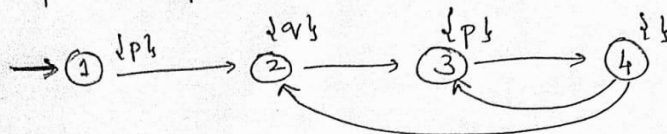
Bounded Model Checking:

1. Looks for counter-example paths of increasing length k (oriented to finding bugs)
2. For each k ,
 - a. Build boolean formula (eg: negation) that is satisfiable iff there is a counter-example of length k
 - b. Satisfiability of the boolean formulas is checked using a SAT procedure
3. If satisfied \rightarrow return satisfying assignment (i.e., a counter-example)

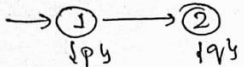
Example1: LTL + Bounded Model Checking

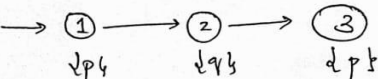
FORMULA: $G(p \rightarrow Fq)$

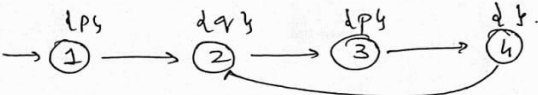
NEGATED FORMULA: $F(p \wedge G \neg q)$

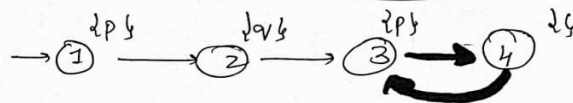


$K=0$:  No counter example

$K=1$:  No counter example

$K=2$:  No counter example.

$K=3$: 



Counter example
in 2nd Trace!

There exists path with $K=3$ which starts with p and all further paths do not contain q ! Negation satisfied.

Example2: CTL + Fixed Point Model Checking

CTL

State formulae

$\phi := \text{true} \mid p_i \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid E\alpha \mid A\alpha$

$p_i \in AP$ ϕ_1, ϕ_2 : State formulae α : Path formula

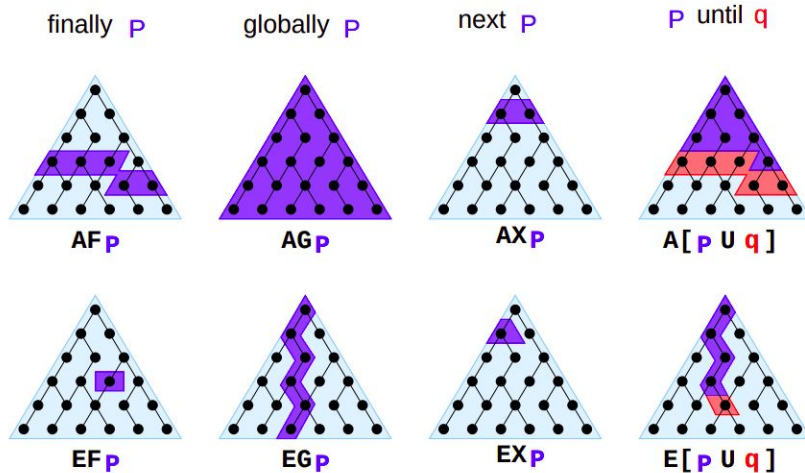
Path formulae

$\alpha := X\phi_1 \mid \phi_1 U \phi_2 \mid F\phi_1 \mid G\alpha_1$

ϕ : State formula α_1, α_2 : Path formulae

Every temporal operator (F, G, X, U) preceded by a path quantifier (A or E).

- Universal modalities (AF, AG, AX, AU): true in all paths starting in the current state.
- Existential modalities (EF, EG, EX, EU): true in some paths starting in the current state.



Some Properties as CTL Formulae:

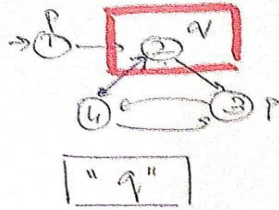
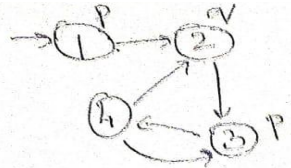
- **Safety:** $AG \neg (C1 \wedge C2)$
- **Liveness:** $AG \neg (C1 \wedge C2)$

Example2: CTL + Fixed Point Model Checking

Fixed Point Model Checking:

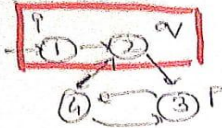
1. Traverse formula structure “Bottom up
2. For each subformula build set of satisfying states;
 - a. compare result with initial set of states.
 - b. If initial states is not present in satisfying states : **Return False**
3. If initial states is a subset of states satisfying the entire formula: **Return True**

Example 2: CTL + Fixed Point Model Checking

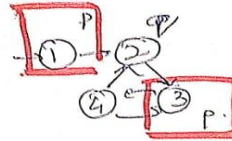


CTL:
 $AG(p \rightarrow AFq)$

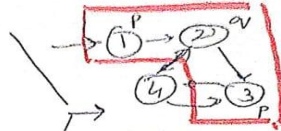
Now go bottom up and find set of states where CTL is held.



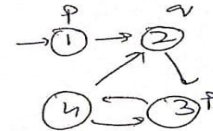
$AF Q$



P



$P \rightarrow AF Q$



$AG(P \rightarrow AF Q)$
 EMPTY

The initial state / Set of states satisfying CTL

\therefore The system/model does not satisfy CTL.

Paper: Exploiting Hierarchy in the Abstraction-Based Verification of Statecharts Using SMT Solvers

As the set of possible configurations for a system becomes unmanageably large or even infinite, the symbolic model checking is impractical.

A possible solution : Abstraction

generic technique for reducing complexity by hiding details that are not relevant for the property to be verified.

Difficulty: finding the proper precision of abstraction - coarse enough to avoid complexity issues but fine enough to prove the desired property.

Counterexample-Guided Abstraction Refinement (CEGAR) is an automatic technique that initially starts with a coarse abstraction and refines it iteratively based on the counterexamples until the proper precision is obtained.

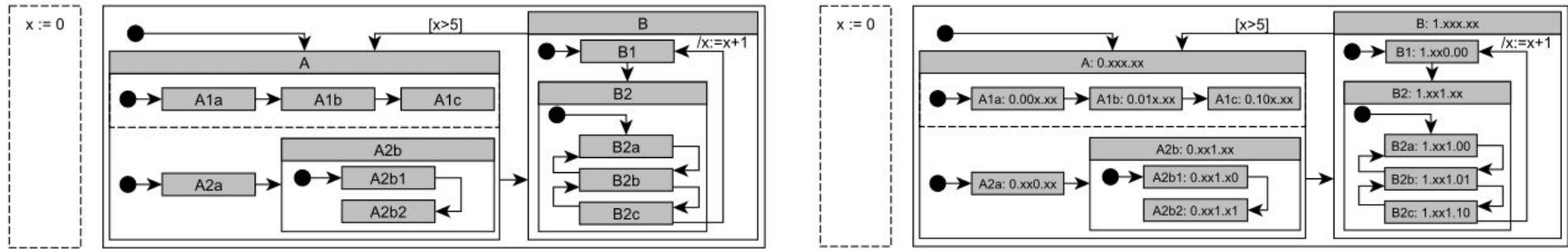
A challenge in applying verification algorithms on statechart models is the presence of

1. Parallelism
2. Hierarchy

Solution: unique encoding function.

The encoding algorithm used ensures that parallelism and hierarchy of statecharts is maintained when encoding states to bit vectors.

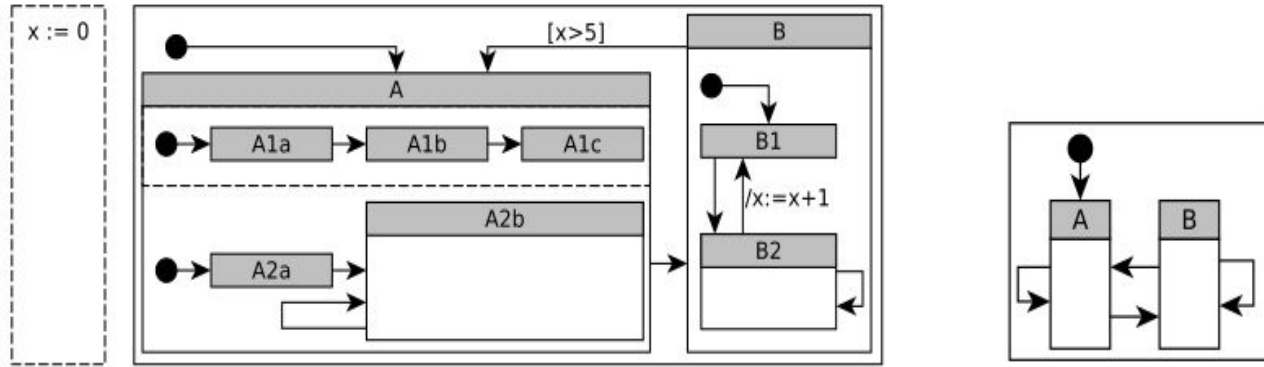
The transitions are then modeled as formulae on the bit strings.



Encoding takes care of both regions and depth of a state.

Also inter- level transition information is not lost

Abstraction + Model Checking:



Property is first checked over the abstraction state space instead of concrete statechart space.

Existential property of the abstraction, : each concrete path in the statechart has its corresponding abstract path

1. If no counterexample found in abstract statechart -> Concrete statechart satisfies property
2. Else if error is encountered :
 - a. Corresponding path in Statechart is a counter example **OR**
 - b. Further refinement of abstract diagram required. Go a few levels down.

Paper: Verification of UML Statechart Models of Embedded Systems

- Automated verification of behavioural requirements through model transformation and application of an off-the-shelf model checker.
- Transform UML models of concurrent object-oriented systems to the input format of the model checker SPIN.

Model transformation:

Statechart diagram(UML) -> Extended Hierarchical Automata (EHA) -> Process Meta Language (Promela) -> SPIN(model checker)

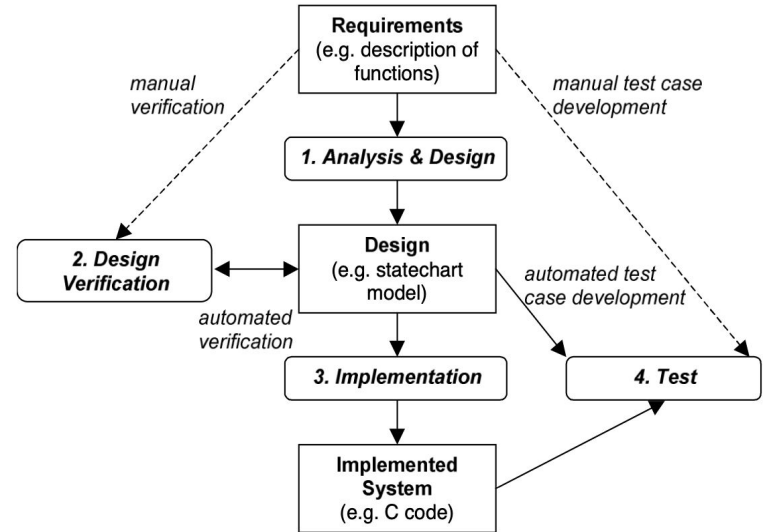


Figure 1: The relationship of the different phases of system development

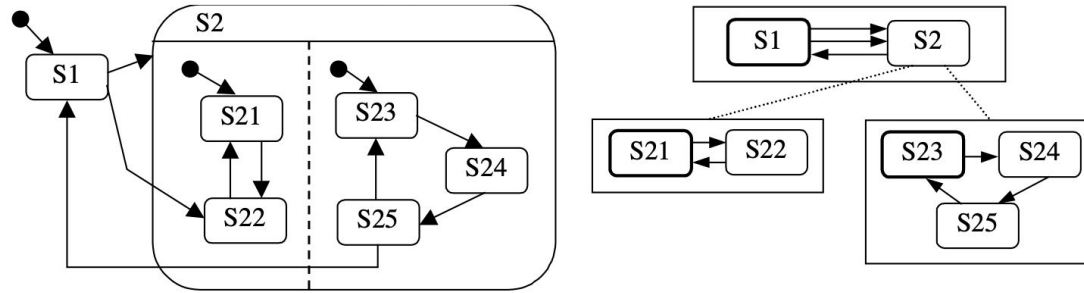


Figure 2: An UML statechart and the structure of the corresponding EHA (the concurrent regions in the refinement of state S2 are represented by two separate sequential automata in the EHA)

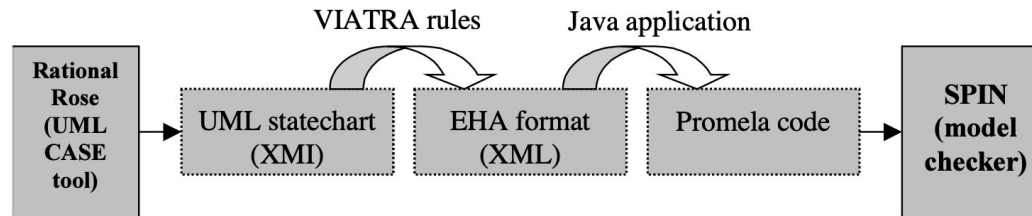


Figure 3: Implementation of the model transformation

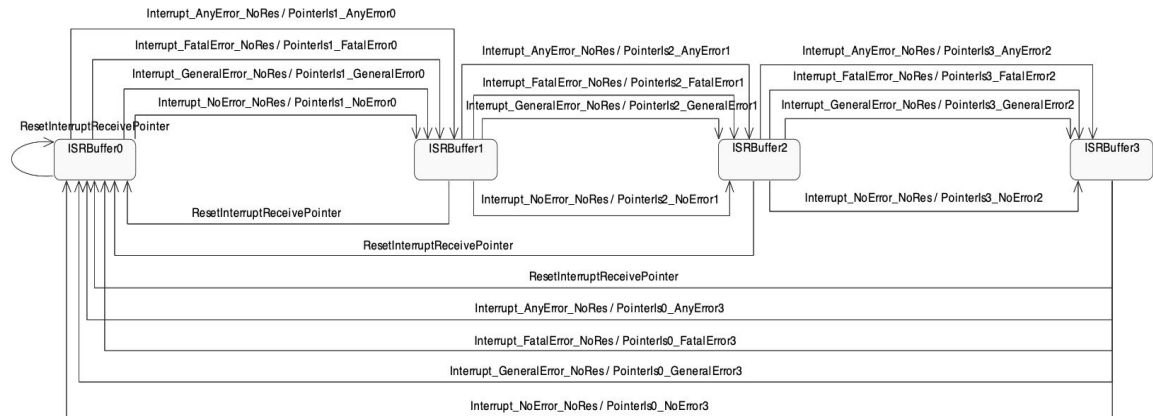


Figure 4: Statechart of the interrupt controller of the ASI Master

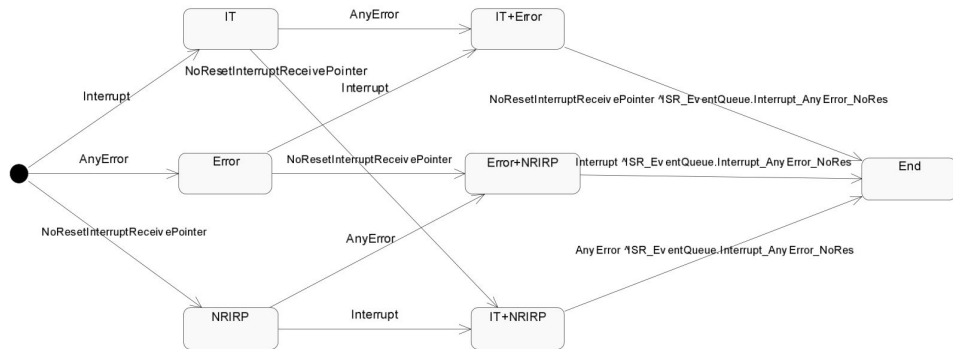


Figure 5: Statechart diagram of the event generation

Figure 5 shows the condition of the event generation for event *Interrupt_AnyError_NoRes*

Paper: Verification of UML Statechart Models of Embedded Systems

Design Verification:

- SPIN automatically checks for deadlocks and unexecutable code during verification.
- States are marked with labels to help analyse the correctness of the system.
- Designers need not have knowledge about these labels since SPIN provides automatic translation from LTL formulae into the *never* process.
- Verification by exhaustive state space search.
- Cons: Application is constrained due exhaustive nature of verification -> **state space explosion.**

Thank you

By-

Arpitha Malavalli (IMT2018504)

Neetha Reddy (IMT2018050)

Tanishq Jaswani (IMT2018077)

References

- [1] Czipó, Bence & Hajdu, Ákos & Tóth, Tamás & Majzik, István. (2017). Exploiting Hierarchy in the Abstraction-Based Verification of Statecharts Using SMT Solvers. Electronic Proceedings in Theoretical Computer Science. 245. 31-45. 10.4204/EPTCS.245.3.
- [2] Darvas, Adam & Majzik, István & Benyó, Balázs. (2002). Verification of UML Statechart Models of Embedded Systems.
- [3] Baier, Christel & Katoen, Joost-Pieter. (2008). Principles of Model Checking.
- [4] Bhaduri, Purandar & Ramesh, S.. (2004). Model Checking of Statechart Models: Survey and Research Directions.