FORMAL VERIFICATION OF UI USING COGNITIVE MODELS: A PRELIMINARY INVESTIGATION

Hadjar Homaei
“Program testing can be used to show the presence of bugs, but never to show their absence!“

Edsger Dijkstra
China Air A300B4-622R Nagoya, Japan

- 26 APR 1994
- Fatalities: 264

![Descent path of China Airlines Flight 140](image.png)

- Nose-down at the Outer Marker, Went Nose-up by the Autopilots.
- Activated the GO lever, GO-AROUND thrust.

Descent path of China Airlines Flight 140 (Created referring to [4])
Human Factor Causes

The co pilot had inadvertently triggered the GO lever.

The placement and design of the GO lever on the thrust lever may have allowed the copilot to inadvertently trigger the GO lever when he tried to move the thrust.

The captain might have been unaware that the aircraft was under autopilot control, or he believed that manual controls input would override or disengage the autopilot.
Ultimate Goal!

- Not to wait for a plane crash to figure out system design problems!
"Quality is never an accident; it is always the result of intelligent effort."

John Ruskin
Usability Testing and Inspection

- A set of methods where an evaluator inspects a user interface
Usability Testing and Inspection

- **Cheap**
  - Can be done *before* the system is even implemented

- **Extremely informal!**
  - Expressed in human language: Ambiguities and misunderstandings of specifications
  - Depend on developers and users assumptions rather than facts
  - **Can not be applied to safety critical interfaces**
Verification of Systems

- Ensuring the “correctness” of the system, Software or Hardware or a combination

- Safety requirements such as the absence of deadlocks and similar critical states that can cause the system to crash.

- Techniques
  - Simulation
  - Testing
  - Deductive Verification
  - Model Checking
Approaches to Formal Verification

- **Model Checking**
  - Consists of a systematically exhaustive exploration of the mathematical model.

- **Logical Inference**
  - Consists of using a formal version of mathematical reasoning about the system, usually using theorem proving software such as a HOL theorem prover, the ACL2, Isabelle, or Coq theorem provers.
Model Checking

- Advantages of Model Checking
  - It is fully automatic
  - It provides a counter example whenever the system fails to satisfy a given property.
Model Checking

- 3 Steps of Model Checking
  - Modeling
  - Specification
  - Verification
Cognitive Architectures

- A Blueprint for intelligent systems.
- Architecture: models both behavior and structural properties
Cognitive Architectures

- **ACT-R**, developed at Carnegie Mellon University under John R. Anderson.
- **Apex** developed under Michael Freed at NASA Ames Research Center.
- **CHREST**, developed under Fernand Gobet at Brunel University and Peter C. Lane at the University of Hertfordshire.
- **CLARION** the cognitive architecture, developed under Ron Sun at Rensselaer Polytechnic Institute and University of Missouri.
- **Copycat**, by Douglas Hofstadter and Melanie Mitchell at the Indiana University.
- **DUAL**, developed at the New Bulgarian University under Boicho Kokinov.
- **EPIC**, developed under David E. Kieras and David E. Meyer at the University of Michigan.
- The **H-Cogaff** architecture, which is a special case of the CogAff schema. (See Taylor & Sayda, and Sloman refs below).
- **IDA and LIDA**, developed under Stan Franklin at the University of Memphis.
- **PRODIGY**, by Veloso et al.
- **PRS 'Procedural Reasoning System'**, developed by Michael Georgeff and Amy Lansky at SRI International.
- **Psi-Theory** developed under Dietrich Dörner at the Otto-Friedrich University in Bamberg, Germany.
- **R-CAST**, developed at the Pennsylvania State University.
- **Soar**, developed under Allen Newell and John Laird at Carnegie Mellon University and the University of Michigan.
- Society of mind and its successor the **Emotion machine** proposed by Marvin Minsky.
- **Subsumption** architectures, developed e.g. by Rodney Brooks (though it could be argued whether they are cognitive).
Cognitive Architectures

- Symbolic (SOAR, ACT-R)
- Connectionist
- Hybrid (CLARION)
- Centralized (SOAR, ACT-R, EPIC)
- Decentralized (Distributed) (ICS)
Cognitive Architectures

- Characteristics

  - Implementation of cognition as a whole (Holism, e.g. Unified theory of cognition).

  - The architecture often tries to reproduce the behavior of the modeled system (human), in a way that timely behavior (reaction times) of the architecture and modeled cognitive systems can be compared in detail.

  - Learning (not for all cognitive architectures)
Cognitive Architectures

Characteristics

- **Parameter-free**: The system does not depend on parameter tuning (not for all)

- Some early theories such as SOAR and ACT-R originally focused only on the 'internal' information processing of an intelligent agent,

- On some theories the architecture may be composed of different kinds of sub-architectures (e.g., CLARION).
ACT-R (Adaptive Control of Thought—Rational)

- ACT-R aims to define the basic and irreducible cognitive and perceptual operations that enable the human mind.

- In theory, each task that humans can perform should consist of a series of these discrete operations.
ACT-R (Adaptive Control of Thought—Rational)

- The ACT-R theory has a computational implementation as an interpreter of a special coding language (written in Lisp)
- The language primitives and data-types are designed to reflect the theoretical assumptions about human cognition
- "models" can be created (i.e., programs) using ACT-R
**ACT-R** (Adaptive Control of Thought—Rational)

- Running a model automatically produces a step-by-step simulation of human behavior which specifies each individual cognitive operation:
  - Memory encoding and retrieval
  - Visual and auditory encoding
  - Motor programming and execution
  - Mental imagery manipulation

- Each step is associated with quantitative predictions of latencies and accuracies. The model can be tested by comparing its results with the data collected in behavioral experiments.
Attentional Blink

- Observers often miss a second target (T2) if it follows an identified first target item (T1) within half a second in rapid serial visual presentation (RSVP).

- If two targets are presented in immediate succession, however, accuracy is excellent (Lag 1 sparing).
Starting Point

- Task: Recognize a specific type of stimulus among the fast presentation of stimuli.
- Target Specification: Avoid Blink Condition
- Model it within ACT-R Using JACT-R
- Model check it Using JPF
Java PathFinder

- Fully Automatic Translation
- Based on custom-made Java Virtual Machine
  - Handle all of Java, since it works with bytecodes
  - Written in Java
- Efficient encoding of states
- Modular design for easy extensions
- Supports LTL checking with properties expressed in Bandera’s BSL notation
- Incorporates a number of search strategies
  - DFS, BFS, A*, Best-first, etc.
- Supports source-2-source abstractions
Java PathFinder

Java Code

```java
void add(Object o) {
    buffer[head] = o;
    head = (head+1)%size;
}

Object take() {
    ...tail=(tail+1)%size;
    return buffer[tail];
}
```

Bytecode

```
0:  iconst_0
1:  istore_2
2:  goto #39
5:  getstatic
8:  aload_0
9:  fload_2
10: aaload
```

Model Checker

Special JVM

JAVAC  →  JVM
Java PathFinder

- Handle full Java language
  - but only for closed systems
  - Cannot handle native code
    - no Input/output through GUls, files, Networks, …
    - Must be modeled by java code instead

- Allows Nondeterministic Environments
  - JPF traps special nondeterministic methods

- Checks for User-defined assertions, deadlock and LTL properties
Demo
Questions?
Thank you!