The Four-Color Theorem

- **1852**: Guthrie conjectured Every planar map is four-colorable
- **1976**: Appel & Haken proved the theorem using an assembly program on a IBM 370-168 computer
- **2004**: Gonthier verified the proof of the theorem using the Coq proof checker
- **2005**: Devlin [Math. Assoc. America] announced Last doubts removed about the proof of the Four Color Theorem
- **2006**: Harrison partially verified HOL light, the logical kernel of Coq, using HOL light itself

Even for most non-typically well defined problem - math, formalization and verification are not so easily attainable
A Different Aspect of Uncertainty

1976 layers
• Assembly program
• Assembler
• Operating system (with VM!)
• Mainframe

2004 layers
• Data: proof
• Application: proof-checker
• Compiler(s)
• Operating system + updates
• Dual-core system
• Network connection
A Typical Application

2010 layers

• Data
• Application
• Compiler(s)
• Operating system(s)
• Virtualization layer(s)
• Multi core / multi processor
• Heterogeneous network

Dynamic aspects

• Runtime downloadable data / scripts
• Dynamic libraries
• Dynamic compilation
• Online SW updates
• Anti virus at the background
• Viruses
• OS patches
• Virtualization layer
• Cloud computing
• …

The interfaces *between* abstraction layers as well as *inside* layers get more *complex, dynamic and unstable* – more reasons for doubts!!!
Outline

• Motivation and conception of an “evolutionary” approach for verification
• Supporting examples
• Initial thoughts about potential directions
Motivation

• **Verification task** refers to a single, isolated transition
  – Given model, system, assumptions, specification
  – Apply an algorithmic verification process
  – Desired correctness outcome: once proved - done forever

• Modern systems are of a more progressive nature
  – Systems evolve, assumptions change
  – Underlying models adapt, correctness criteria get refined
  – Verification methods improve, adjust
  – Correctness concerns are never fully satisfied

• **Hypothesis**
  – System’s fast evolution and complexity make it increasingly inefficient / impossible to target system time-snapshots by isolated verification tasks
Proposal: Evolutionary Verification

**Challenge:** Extend the scope of formal-methods research from (isolated) verification tasks to the context of (evolutionary) verification process

This requires the development of a *formal framework* that can *adapt* to and express the *evolution* of

- Specifications
- Computational/programming model
- Verification methods
- Correctness criteria and metrics
- Methods for handling intermediate, incorrect states

... and their *ongoing integration* into the implementation process.
Put into Historical Perspective

Strongly Inspired by some of Amir Pnueli’s Major Contributions

- **Transformational System**
  - Verification Task
  - Input
  - Valid!
  - Adding laziness to the verification process
  - Adding time and state to the system and its spec

- **Compiler**
  - Input
  - Valid for P!

- **Reactive System**
  - Verification Task
  - Input*
  - Valid!

- **Evolving System**
  - Verification Process
  - Adding time and state to the verification process???
Case for Evolution (1) - Racing

• Characteristics
  – Systems are too complex to fully verify in advance
  – System’s (at least initial) reaction/output is required earlier than full verification can complete

• Examples
  – Just in time (JIT) compilation
  – Dynamic binary optimizers (DBO)
  – Virtualization layers
Case for Evolution (2) - Unpredictability

- Characteristics
  - System behavior is changing dynamically
  - Modes of operations / usage environments are amorphous / not known in advance

- Examples
  - WEB applications, e.g. Java scripts
  - Viruses and anti viruses
  - Operating systems
  - Server networks
  - Cloud computing
Case for Evolution (3) - Maliciousness

• Characteristics
  – Optimized systems
  – Explicit interfaces (e.g. ISA, programming model) are preserved, yet implicit assumptions of the applications are broken
  – Knowledge of implementation details enables unexpected attacks
• Examples - RSA encryption
  – *Side channel attack* on the Secure Socket Layer (SSL) protocol (protecting online transactions)
  – Exploits intimate knowledge of HW optimizations such as caches and branch prediction
  – Exploit intimate knowledge of the algorithmic implementation of the protocol
  – Utilize “innocent” OS features such time sharing to “spy” into the protocol
  – Gain *observability* into tiny timing effects uncovering the private key
So How Evolution?

• All three cases (racing, unpredictability, maliciousness) have several characteristics in common
  – Complexity
  – Impossibility to validate in advance
  – A sense of continuous struggle for correctness
  – Need to tolerate intermediate failures

• Can “incessant, lazy-verification” become a more robust evolutionary model?
  – Specification, verification are building blocks of the continuous design process

• While competing for system resources, need to address
  – How to manage the evolving specification, correctness status
  – What to do about incorrect output?
  – How to fix a failing system?
  – How to improve verification over time (learn)?
Why is Evolutionary Verification an “Appropriate” Challenge?

• Interesting? – subjective
• Difficult? – necessary, not sufficient
• Inspired by real world problems
• Has the potential of expanding the scope and outreach of formal methods, by
  – Addressing some fundamental questions about the very nature of formal models
    • What is a (good) specification?
    • What defines the limits and the desired flexibilities of a formal model?
  – Allowing for better design engineering
Partial List of Related Trends and Potential Directions

- Open Verification Methodology (OVM) initiative
- Subject/aspect oriented programming
  - Separation of concerns
- Self verification
  - Assertions
  - Artificial intelligence methods
  - SHADOWS
- Any method of gradual verification
  - Bounded model checking
- Many relevant ideas I heard in the first day of the symposium
Thank You