Formal Methods at Work

M. Carro

IMDEA Software Institute
and
Technical University of Madrid

May 11, 2015
Part I

Where Do We Stand?
Software is omnipresent in everyday life

Today’s car: typically 100 processing units, 100 M. lines of code, 600 Quijotes, 7,000 programmer years.
Software is omnipresent in everyday life

Plane: computers manage controls, calculate routes, ...
Software is omnipresent in everyday life

Large interconnected systems: independent, isolated, automatic decision making, which must be globally correct.
Software is omnipresent in everyday life:

- Cell phones (from O.S. to compression algorithms to user interfaces).
- HDTV (routing, encoding and decoding).
- Buy and sell on the Internet (web interfaces, databases, encryption).
- Stock market (algorithmic trading, high frequency trading).
Managed by extremely complex software.
All of them critical to a certain degree.
Some extremely critical
Managed by extremely complex software.
All of them *critical* to a certain degree.
Some *extremely* critical

Challenge:
How to develop complex software, with resources that are always limited, assuring that it will work correctly?
How Far Are We from Giving Reasonable Guarantees?
(Only showing some types of problems)

Skype bug sends messages to unintended recipients.

July 16, 2012: Skype bug sends messages to unintended recipients.
July 13, 2012: Apple's "in-app purchase" service for iOS bypassed by Russian hacker.
July 12, 2012: Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).
July 5, 2012: iOS, Mac app crashes linked to botched FairPlay DRM.
July 7, 2012: Still infected, 300,000 PCs to lose Internet access.
July 12, 2012: Hackers expose 453,000 credentials allegedly taken from Yahoo service.
July 13, 2012: German security experts find major flaw in credit card terminals.
How Far Are We from Giving Reasonable Guarantees?

(Only showing some types of problems)

- Skype bug sends messages to unintended recipients.
- Apple’s “in-app purchase” service for iOS bypassed by Russian hacker.
How Far Are We from Giving Reasonable Guarantees?

(Only showing some types of problems)

Skype bug sends messages to unintended recipients.

Apple’s “in-app purchase” service for iOS bypassed by Russian hacker.

Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).
How Far Are We from Giving Reasonable Guarantees?
(Only showing some types of problems)

- Skype bug sends messages to unintended recipients.
- Apple’s “in-app purchase” service for iOS bypassed by Russian hacker.
- Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).
- iOS, Mac app crashes linked to botched FairPlay DRM.
How Far Are We from Giving Reasonable Guarantees?
(Only showing some types of problems)

- Skype bug sends messages to unintended recipients.
- Apple’s “in-app purchase” service for iOS bypassed by Russian hacker.
- Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).
- iOS, Mac app crashes linked to botched FairPlay DRM.
- Still infected, 300,000 PCs to lose Internet access.
How Far Are We from Giving Reasonable Guarantees?

(Only showing some types of problems)

- Skype bug sends messages to unintended recipients.
- Apple’s “in-app purchase” service for iOS bypassed by Russian hacker.
- Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).
- iOS, Mac app crashes linked to botched FairPlay DRM.
- Still infected, 300,000 PCs to lose Internet access.
- Hackers expose 453,000 credentials allegedly taken from Yahoo service.
How Far Are We from Giving Reasonable Guarantees?

(Only showing some types of problems)

Skype bug sends messages to unintended recipients.

Apple’s “in-app purchase” service for iOS bypassed by Russian hacker.

Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).

iOS, Mac app crashes linked to botched FairPlay DRM.

Still infected, 300,000 PCs to lose Internet access.

Hackers expose 453,000 credentials allegedly taken from Yahoo service.

German security experts find major flaw in credit card terminals.
How Far Are We from Giving Reasonable Guarantees?

(Only showing some types of problems)

Skype bug sends messages to unintended recipients.

Apple’s “in-app purchase” service for iOS bypassed by Russian hacker.

Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).

iOS, Mac app crashes linked to botched FairPlay DRM.

Still infected, 300,000 PCs to lose Internet access.

Hackers expose 453,000 credentials allegedly taken from Yahoo service.

German security experts find major flaw in credit card terminals.

Defects leave critical military, industrial infrastructure open to hacks (Niagara Framework, linking 11+ million devices in 52 countries).
How Far Are We from Giving Reasonable Guarantees?

(Only showing some types of problems)

- **July 16, 2012**: Skype bug sends messages to unintended recipients.
- **July 13, 2012**: Apple’s “in-app purchase” service for iOS bypassed by Russian hacker.
- **July 12, 2012**: Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).
- **July 5, 2012**: iOS, Mac app crashes linked to botched FairPlay DRM.
- **July 7, 2012**: Still infected, 300,000 PCs to lose Internet access.
- **July 12, 2012**: Hackers expose 453,000 credentials allegedly taken from Yahoo service.
- **July 13, 2012**: German security experts find major flaw in credit card terminals.
- **July 13, 2012**: Defects leave critical military, industrial infrastructure open to hacks (Niagara Framework, linking 11+ million devices in 52 countries).
How Far Are We from Giving Reasonable Guarantees?

(Only showing some types of problems)

July 16, 2012: Skype bug sends messages to unintended recipients.

July 13, 2012: Apple’s “in-app purchase” service for iOS bypassed by Russian hacker.

July 12, 2012: Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).

July 5, 2012: iOS, Mac app crashes linked to botched FairPlay DRM.

July 7, 2012: Still infected, 300,000 PCs to lose Internet access.

July 12, 2012: Hackers expose 453,000 credentials allegedly taken from Yahoo service.

July 13, 2012: German security experts find major flaw in credit card terminals.

The Ariane 5 Incident

Example: effect of a single integer overflow.
The Ariane 5 Incident

Example: effect of a single integer overflow.

From the outside...

- June 4, 1996: After launch, the Ariane 5 rocked exploded.
- This was its maiden voyage.
- It flew for about 37 sec. only in Kourou’s sky.
- No injury in the disaster.
## Mechanical details

- Normal behavior of the launcher for 36 sec. after lift-off.
- Failure of both Inertial Reference Systems almost simultaneously.
- Strong pivoting of the nozzles of the boosters and Vulcain engine.
- Self-destruction at an altitude of 4000 m. (1000 m. from the pad).
Mechanical details
- Normal behavior of the launcher for 36 sec. after lift-off.
- Failure of both Inertial Reference Systems almost simultaneously.
- Strong pivoting of the nozzles of the boosters and Vulcain engine.
- Self-destruction at an altitude of 4000 m. (1000 m. from the pad).

Forensic analysis
- Both inertial computers failed because of overflow on one variable.
- This caused a software exception and stops these computers.
- These computers sent post-mortem info through the bus.
- Normally the main computer receives velocity info through the bus.
- The main computer was confused and pivoted the nozzles.
But, why?

- The faulty program was working correctly on Ariane 4.
- The faulty program was not tested for A5 (since it worked for A4).
- But the velocity of Ariane 5 is far greater than that of Ariane 4.
- That caused the overflow in one variable.
- The faulty program happened to be **useless** for Ariane 5.
But, why?

- The faulty program was working correctly on Ariane 4.
- The faulty program was not tested for A5 (since it worked for A4).
- But the velocity of Ariane 5 is far greater than that of Ariane 4.
- That caused the overflow in one variable.
- The faulty program happened to be **useless** for Ariane 5.

Message

- Does the product conform to specifications?
- Maybe it’s important to carry experience in formal methods to industry!
- Is it done?
Formal Methods in Industry

- Actually, yes — at least in some domains, for some types of applications.
- Using many approaches, actually [WLBF09].

We will see some examples
Part II

The General Does Not Need a Driver [BBFM99]
Driving a Subway Train Is Easy… Is It?

- Only two buttons: start and stop.
  - No crossroads, no people crossing, …
- Should be easy to automate!
- But...
  - Wait for people to get on board.

\[1\] Same with trains, only worse.
Driving a Subway Train Is Easy… Is It?

- Only two buttons: start and stop.
  - No crossroads, no people crossing, …
- Should be easy to automate!
- But...
  - Wait for people to get on board.
  - Watch for people doing funny things close to the train.

1 Same with trains, only worse.

madrid institute for advanced studies in software development technologies
Driving a Subway Train Is Easy... Is It?

- Only two buttons: start and stop.
  - No crossroads, no people crossing, ...  
- Should be easy to automate!
- But...
  - Wait for people to get on board.
  - Watch for people doing funny things close to the train.
  - Don’t get too close to the next train.

\(^1\)Same with trains, only worse.
Driving a Subway Train Is Easy... Is It?

- Only two buttons: start and stop.
  - No crossroads, no people crossing, ...
- Should be easy to automate!
- But...
  - Wait for people to get on board.
  - Watch for people doing funny things close to the train.
  - Don’t get too close to the next train.
  - How can you to try to see if it works?¹

¹Same with trains, only worse.
Driving a Subway Train Is Easy... Is It?

- Only two buttons: start and stop.
  - No crossroads, no people crossing, ...
- Should be easy to automate!
- But...
  - Wait for people to get on board.
  - Watch for people doing funny things close to the train.
  - Don’t get too close to the next train.
  - How can you to try to see if it works?¹
  - And, if it doesn’t, how long does it take to reproduce the bug in your computer?

¹Same with trains, only worse.

madrid institute for advanced studies in software development technologies
Driving a Subway Train Is Easy... Is It?

- Only two buttons: start and stop.
  - No crossroads, no people crossing, ...
- Should be easy to automate!
- But...
  - Wait for people to get on board.
  - Watch for people doing funny things close to the train.
  - Don’t get too close to the next train.
  - How can you to try to see if it works?¹
  - And, if it doesn’t, how long does it take to reproduce the bug in your computer?
  - Or, plainly, how can you reproduce it?

¹Same with trains, only worse.
Towards an Automatically Managed Subway Line

- Add extra sensors.
- Add extra mechanisms (doors apart apart from coaches’).
- Model the environment very precisely.
- Correctness by Construction.
Paris Metro Line 14

- October 15th, 1998, Tolbiac to Madeleine. (Now extended)
- 40000 passenger / hour, 85 sec. between trains in peak hour.
- 2009: 60 million passengers / year.
- Decision based on previous experience:
  - Completely automated line.
  - Completely developed using formal methods for control systems.
  - Having manually and automatically driven trains.
Subsystems

- Automatic control and signaling.
- Platform doors.
- Audio and Video.
- Operating control center.
Subsystems

- **Automatic control and signaling.**
  - In the train.
  - In the operating control center.
  - Along the tracks.

- Platform doors.
- Audio and Video.
- Operating control center.
Subsystems

- Automatic control and signaling.
  - In the train.
  - In the operating control center.
  - Along the tracks.
  - Running alone on specific, non-interruptible, microprocessor boards.
- Platform doors.
- Audio and Video.
- Operating control center.
Subsystems

- Automatic control and signaling.
  - In the train.
  - In the operating control center.
  - Along the tracks.
  - Running alone on specific, non-interruptible, microprocessor boards.
- Platform doors.
- Audio and Video.
- Operating control center.
- Developed using the *B Method* [Abr96], now evolved into Event B [Abr10].
Correctness by construction
Make sure that every step is correct

A Taste of B

Model the problem.
Determine the properties to hold.
Refine the model (several times, adding more details).
Prove that the refinement is correct.

Software requirements

Heavy human intervention
Abstract model
Light human intervention
Concrete model
No human intervention
Executable code
Correctness by construction
Make sure that every step is correct

A Taste of B

Model 
Determine 
Refine 
Prove 
the problem.
the properties to hold.
the model (several times, adding more details).
that the refinement is correct.

Software requirements
Heavy human intervention
Abstract model
Light human intervention
Concrete model
No human intervention
Executable code

Abstract model 1
Refinement
Abstract model 2
Refinement
Final abstract model
Correctness by construction
Make sure that every step is correct

A Taste of B

Model the problem.
Determine the properties to hold.
Refine the model (several times, adding more details).
Prove that the refinement is correct.

Software requirements

Abstract model

Concrete model

Executable code

Concrete model 1

Concrete model 2

Final concrete model
Correctness by construction
Make sure that every step is correct

A Taste of B

Model the problem.
Determine the properties to hold.
Refine the model (several times, adding more details).
Prove that the refinement is correct.

Software requirements

Heavy human intervention

Abstract model

Light human intervention

Concrete model

No human intervention

Executable code

Final concrete model
Translation
Program
Compilation
Executable code
INVARIENTS

invBDay : \( \text{birthday} \in PERSON \rightarrow DATE \)

EVENTS

Initialisation

begin
  initBDay : \( \text{birthday} := \emptyset \)
end

Event addBDay \( = \)

any \( p,d \)

where

  inPerson : \( p \in PERSON \)
  inDate : \( d \in DATE \)

then

  newBDay : \( \text{birthday}(p) := d \)

end

END
INVARIANTS

\(\text{invBDay} : birthday \in \text{PERSON} \rightarrow \text{DATE}\)

EVENTS

Initialisation

begin

\(\text{initBDay} : birthday := \emptyset\)

end

Event \(\text{addBDay} \triangleq\)

any

\(p, d\)

where

\(\text{inPerson} : p \in \text{PERSON}\)

\(\text{inDate} : d \in \text{DATE}\)

\(\text{checkBday} : p \notin \text{dom}(\text{birthday})\)

then

\(\text{newBDay} : birthday(p) := d\)

end

END
Proofs: Prove **mathematically** refinements are right.
#### Proofs: Prove mathematically refinements are right.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Premise</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>HYP</td>
<td>( n &gt; 0 )</td>
<td>( n &gt; 0 = 0 )</td>
</tr>
<tr>
<td>HYP</td>
<td>( n &gt; 0 )</td>
<td>( n &gt; 0 )</td>
</tr>
<tr>
<td>AND_R</td>
<td>( n &gt; 0 )</td>
<td>( (n &gt; 0 \land 0 = 0) )</td>
</tr>
<tr>
<td>OR_R</td>
<td>( n &gt; 0 )</td>
<td>( (n &gt; 0 \land 0 = 0) \lor (n &gt; 0 \land 0 = 0) )</td>
</tr>
<tr>
<td>EQL</td>
<td>( n &gt; 0 \lor n &lt; d )</td>
<td>( (n &lt; d \land 0 = 0) \lor (n &gt; 0 \land 0 = 0) )</td>
</tr>
<tr>
<td>EQ_LR</td>
<td>( b = n, n &gt; 0 \lor n &lt; d )</td>
<td>( (b &lt; d \land 0 = 0) \lor (b &gt; 0 \land 0 = 0) )</td>
</tr>
<tr>
<td>ARITH</td>
<td>( a + b + 0 = n )</td>
<td>( n &gt; 0 \lor n &lt; d )</td>
</tr>
<tr>
<td>ARITH</td>
<td>( n &gt; 0 \lor n &lt; d )</td>
<td>( a = 0 )</td>
</tr>
<tr>
<td>NEG</td>
<td>( a \in \mathbb{N}, a + b + 0 = n )</td>
<td>( n &gt; 0 \lor n &lt; d )</td>
</tr>
<tr>
<td>EQ_LR</td>
<td>( a \in \mathbb{N}, a + b + 0 = n )</td>
<td>( n &gt; 0 \lor n &lt; d )</td>
</tr>
<tr>
<td>ARITH</td>
<td>( a \in \mathbb{N}, a + b + 0 = n )</td>
<td>( n &gt; 0 \lor n &lt; d )</td>
</tr>
<tr>
<td>ARITH</td>
<td>( a, c \in \mathbb{N}, a + b + c = n )</td>
<td>( n &gt; 0 \lor n &lt; d )</td>
</tr>
<tr>
<td>NEG</td>
<td>( a, c \in \mathbb{N}, a + b + c = n )</td>
<td>( n &gt; 0 \lor n &lt; d )</td>
</tr>
<tr>
<td>MON</td>
<td>( a, b, c, d, n \in \mathbb{N}, 0 &lt; d, a + b + c = n )</td>
<td>( n &gt; 0 \lor n &lt; d )</td>
</tr>
</tbody>
</table>

*madrid institute for advanced studies in software development technologies*
Proofs: Prove mathematically refinements are right.

Note: invariants are the most important piece of information. Although many proofs are not aimed at establishing invariants, virtually all of them involve invariants.
Tools!

- Automate some (many) proofs.
- Automatically generate code.
- Many advantages when code is generated.
- Example at hand: code duplication.
  - Electronic in tunnels: interference with boards.
  - Possible corruption of data.
  - All data duplicated in different formats.
  - Code works on both copies.
  - Constant comparison for consistency.
Statistics

- **Lines of B**: 115,000.
- **Lines of Ada**: 86,000.
- **Lemmas of B**: 27,800.
- **Automatically proven**: 92%.
- **Time to develop**: 4 years (aprox.)
Statistics

- **Lines of B**: 115,000.
- **Lines of Ada**: 86,000.
- **Lemmas of B**: 27,800.
- **Automatically proven**: 92%.
- **Time to develop**: 4 years (aprox.)
- **Bugs in development computer**: 0
- **Bugs in target computer**: 0
- **Bugs in on-site tests**: 0
- **Bugs since deployment**: 0
Part III

Amazon Had 2.000.000.000.000 Things to Care About [NTZ^{+}14]
Landscape

Some numbers for AWS’s S3

- 2013: 2,000,000,000,000 (2 trillions) objects, 1.1 million requests per second.
- High availability.
- Scalability.
- S3 just one AWS service.
- Essential complexity high → unavoidable human errors.
Previously in...

- Standard “verification” techniques in industry.
  - Deep design reviews.
  - Code reviews (c.f. DB train code reviews).
  - Static code analysis.
  - Stress testing.
  - Fault-injection testing.
  - ...

...human intuition is poor at estimating the true probability of supposedly “extremely rare” combinations of events in systems operating at a scale of millions of requests per second.
The History According to the Starring Roles

- Engineer C.N. dissatisfied with bugs in implementations of distributed algorithms.
- Looked for ways to correct them — not thinking on formal methods.
- But read paper on formal verification of Chord using Alloy.
- Tried to use it, but not rich enough — no good for formal methods!
The History According to the Starring Roles

- Engineer C.N. dissatisfied with bugs in implementations of distributed algorithms.
- Looked for ways to correct them — not thinking on formal methods.
- But read paper on formal verification of Chord using Alloy.
- Tried to use it, but not rich enough — no good for formal methods!
- C.N. read a paper on Leslie Lamport Paxos algorithm — essential in distributed systems.
- At the end of the paper, a TLA+ [Lam02] / TLC formalization.
- TLA+ also devised by Leslie Lamport.
- Maybe TLA+ / TLC was worth something?
- Tried the same example as in Alloy — success!
The Busy Engineers

- But engineers too busy to try new things.
- Unless there is a new need — and a new need appeared.
The Busy Engineers

- But engineers too busy to try new things.
- Unless there is a new need — and a new need appeared.

Enter DynamoDB

- Scalable, high-performance, high-availability storage.
- Testing, stressing, fault injection — but really high confidence was necessary.
  - Otherwise data from companies could be lost.
- T.R. (DynamoDB’s coauthor) went on to prove relevant properties.
- Informal proofs already found bugs.
- Which other subtle problems could be hidden?
TLA+ Time

- T.R. learned TLA+, formalized (some) DynamoDB algorithms.
- Run distributed version of model checker TLC.
  - Cluster of ten EC$^2$ instances.
  - Each 8 cores + hyperthreads.
  - 23 GB ram.
- Small part of algorithm OK.
- But in the full fault-tolerance algorithm a bug was found.
  - Very subtle — many conditions had to be met.
  - But historically possible.
  - Bug had passed all reviews.
  - Other two bugs were found later on.
The History Goes On

• New DynamoDB features first modeled and verified in TLA+ — and bugs were found ahead of time.
• Presentation to teams: *Debugging Designs.*
  • *Exahustively testable pseudo-code.*
• New fault-tolerant distributed algorithm specified and checked.
  • Two bugs found.
• Management started pushing TLA+ usage in teams.
What the Protagonists Say...

[...] help engineers to get the design right. [...] If the design is broken then the code is almost certainly broken. Coding mistakes extremely unlikely to compensate mistakes in design.

Engineers probably deceived into believing that code is ‘correct’ because appears to correctly implement the broken design. Unlikely to realize that design is incorrect while focusing on coding.

[...] gain a better understanding of the design. [...] can only increase chances that they will get code right.

[...] write better assertions [...] , a good way to reduce errors in code.

Formal methods help engineers to find strong invariants, so formal methods can help to improve assertions, which help improve the quality of code.
Part IV

That Is Not What I Meant [PPS⁺03]
A Different Problem

How misleading are (well-written) natural language specifications?

In general, a lot.

Natural language is ambiguous.

Even if it’s not, it relies a lot on “common knowledge” or “situational knowledge”.

Kids give us lots of examples: flawless reasoning without knowledge.

A Real Conversation with a 6-Year Old Girl

- Do not leave the water running. What will happen if we run out of water?
- We buy another faucet.
- (Pointing below the sink). Look: the tap gets water from the pipe, which comes from the wall. Buying another faucet will not help. Now, what would be done?
- We buy another wall.
A Different Problem

How misleading are (well-written) natural language specifications?

- In general, a lot.
- Natural language is ambiguous.
- Even if it’s not, it relies a lot on “common knowledge” or “situational knowledge”.
- Kids give us lots of examples: flawless reasoning without knowledge.
A Different Problem

How misleading are (well-written) natural language specifications?

- In general, a lot.
- Natural language is ambiguous.
- Even if it’s not, it relies a lot on “common knowledge” or “situational knowledge”.
- Kids give us lots of examples: flawless reasoning without knowledge.

A Real Conversation with a 6-Year Old Girl

- Do not leave the water running. What will happen if we run out of water?
  - We buy another faucet.
  - (Pointing below the sink). Look: the tap gets water from the pipe, which comes from the wall. Buying another faucet will not help. Now, what would be do?
  - We buy another wall.
A Different Problem

How misleading are (well-written) natural language specifications?

- In general, a lot.
- Natural language is ambiguous.
- Even if it’s not, it relies a lot on “common knowledge” or “situational knowledge”.
- Kids give us lots of examples: flawless reasoning without knowledge.

A Real Conversation with a 6-Year Old Girl

- Do not leave the water running. What will happen if we run out of water?
A Different Problem

How misleading are (well-written) natural language specifications?

- In general, a lot.
- Natural language is ambiguous.
- Even if it’s not, it relies a lot on “common knowledge” or “situational knowledge”.
- Kids give us lots of examples: flawless reasoning without knowledge.

A Real Conversation with a 6-Year Old Girl

- Do not leave the water running. What will happen if we run out of water?
- We buy another faucet.
A Different Problem

How misleading are (well-written) natural language specifications?

- In general, a lot.
- Natural language is ambiguous.
- Even if it’s not, it relies a lot on “common knowledge” or “situational knowledge”.
- Kids give us lots of examples: flawless reasoning without knowledge.

A Real Conversation with a 6-Year Old Girl

- Do not leave the water running. What will happen if we run out of water?
- We buy another faucet.
- (Pointing below the sink). Look: the tap gets water from the pipe, which comes from the wall. Buying another faucet will not help. Now, what would be do?
A Different Problem

How misleading are (well-written) natural language specifications?

- In general, a lot.
- Natural language is ambiguous.
- Even if it’s not, it relies a lot on “common knowledge” or “situational knowledge”.
- Kids give us lots of examples: flawless reasoning without knowledge.

A Real Conversation with a 6-Year Old Girl

- Do not leave the water running. What will happen if we run out of water?
- We buy another faucet.
- (Pointing below the sink). Look: the tap gets water from the pipe, which comes from the wall. Buying another faucet will not help. Now, what would be do?
- We buy another wall.
A Different Experiment

- Existing, well-proven implementation of a smartcard.
- Natural language specification.
- How well can a team of engineers capture understand these specifications?
Using Models to Generate Tests

System

Abstraction

Model

Proof

Verified model

Code generation

Executable code

Tests
Using Models to Generate Tests

1. System
2. Abstraction
3. Model
4. Proof
5. Verified model
6. Manual generation
7. Executable code
8. Code generation
9. Executable code
10. Test generation
11. Tests
12. Tester
13. Execution
14. Results
A SmartCard is...

- Fully programmable one-chip computer.
- Microprocessor, RAM (currently 256-4096 Bytes), EEPROM (2-16 KBytes), and ROM (8-64 KBytes).
- Hierarchical filesystem.
- Serial interface.
- Sometimes specialized cryptographic microprocessor.
A SmartCard is...

- Fully programmable one-chip computer.
- Microprocessor, RAM (currently 256-4096 Bytes), EEPROM (2-16 KBytes), and ROM (8-64 KBytes).
- Hierarchical filesystem.
- Serial interface.
- Sometimes specialized cryptographic microprocessor.

Command interpreter:
- Read commands from stdin.
- Interpret, execute, write output to stdout.
- Execution usually depends on previous commands.
System Under Study

- WAP module in GSM standard, implemented in a SmartCard.
Subcomponents

- Each subcomponent has an expected behavior (i.e., what it expects to receive and return in every state).
Tester: Behavior

From the behavior of the component one can extract the expected behavior of the tester: what it has to do at every moment to check the component.
The System to Test
The System to Test
The System to Test
The System to Test
The System to Test
The System to Test

madrid institute for advanced studies in software development technologies
The System to Test
The System to Test
The Behavior of the Test Suite
The Behavior of the Test Suite
Test Coverage and Generation

- Testing cannot (in general) ensure correctness.
- Only existence of errors can be proven.
- Exhaustive testing: very expensive
  (in time to generate tests and to execute them)
- Test generation needs to execute the program.
  - *If I call operation A with data D, which output would I obtain?*
Test Generation for the SmartCard

- Using CLP.
- For a component a with a state machine, a collection of clauses like:

  \[
  \text{step\_a(StateIn, Input, StateOut, Output)} : \neg \\
  \text{guard(StateIn, Input),} \\
  \text{assign(StateIn, Input, StateOut, Output).}
  \]

- Different components can be chained together:

  \[
  \text{step\_comp(StateIn, In, StateOut, Out)} : \neg \\
  \text{step\_a(StateOut, In, S0, O0),} \\
  \text{step\_b(S0, O0, S1, O1),} \\
  \ldots \\
  \text{step\_k(Sn, On, StateOut, Out).}
  \]

- Enumerate traces.
- Use constraints to reduce trace sizes.

madrid institute for advanced studies in software development technologies
CLP to Reduce Traces

- Transition has to read value $v$ from input channel $i$.

  $$v = \text{read}(i);$$

  $$\text{if } (v == k) \text{ then } \ldots$$

  $$\text{else } \ldots$$

- In \textbf{else} branch: assuming all possible values for $v$ not feasible.
- Make a trace with $v = k$ and another with $v \neq k$.
- Constraint in the latter $\rightarrow$ generate refined feasible traces later on.
  - Requiring $v = k$ after that $\rightarrow v \neq k \land v = k \rightarrow$ trace not generated.
  - A trace that requires $v \geq k$ is reduced to requiring $v > k$.
- Using sets of values (= constraints) helps reduce the search space.
- Still, some operations are difficult to model.
- \textbf{AskRandom}($n$): only length modeled.
Evaluation

- 60,000 test sequences, varying length.
- Testing with only 2%-3% of sequences.
- Around one hour to execute.
- Summary: out of 1506 test sequences, 84 mismatches.
- All of them due to misinterpretation of documentation or faults in recently optimized versions of software.
Evaluation

- 60,000 test sequences, varying length.
- Testing with only 2%-3% of sequences.
- Around one hour to execute.
- Summary: out of 1506 test sequences, 84 mismatches.
- All of them due to misinterpretation of documentation or faults in recently optimized versions of software.

1. Make no assumptions.
2. Premature optimization is the root of all evil.
Conclusions

- Yes, formal methods are used in industry — more than usually thought.
- **Not** to write payroll software.
  - But payroll software is already written...
- High-availability, dependable software.
  - NASA; Prolog to formalize JVM class system; SLAM at Microsoft; Esterel for hardware, avionics, and cars; Airbus automotive industry, cyber-physical systems.
- Also to discover bugs in existing implementations.
  - E.g., FREAK SSL negotiation bug — team INRIA, Microsoft Research, IMDEA Software Institute.
Recommendations [BH06] and Old Man Sayings

- Use well-tested, well-documented formal method.
- With tool support.
- If possible, have an expert at hand – at least at the beginning.
- Document everything: every assumption, every decision.
- Don’t lower quality standards.
- Test and test again.
Recommendations [BH06] and Old Man Sayings

- Use well-tested, well-documented formal method.
- With tool support.
- If possible, have an expert at hand – at least at the beginning.
- Document everything: every assumption, every decision.
- Don’t lower quality standards.
- Test and test again.
- Trying to formalize will force you to think about a problem. Thinking about a problem will make you understand it.
Jean-Raymond Abrial.  
*The B-Book: Assigning Programs to Meanings.*  

Jean-Raymond Abrial.  
*Modeling in Event-B: System and Software Engineering.*  

Patrick Behm, Paul Benoit, Alain Faivre, and Jean-Marc Meynadier.  
METEOR: A Successful Application of B in a Large Project.  

Jonathan Bowen and Michael Hinchey.  
Ten Commandments of Formal Methods... Ten Years Later.  

Leslie Lamport.  
*Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers.*


Formal Methods at Work

M. Carro

IMDEA Software Institute
and
Technical University of Madrid

May 11, 2015