Formal Methods

JD Kilgallin

CPSC:480

10/03/22

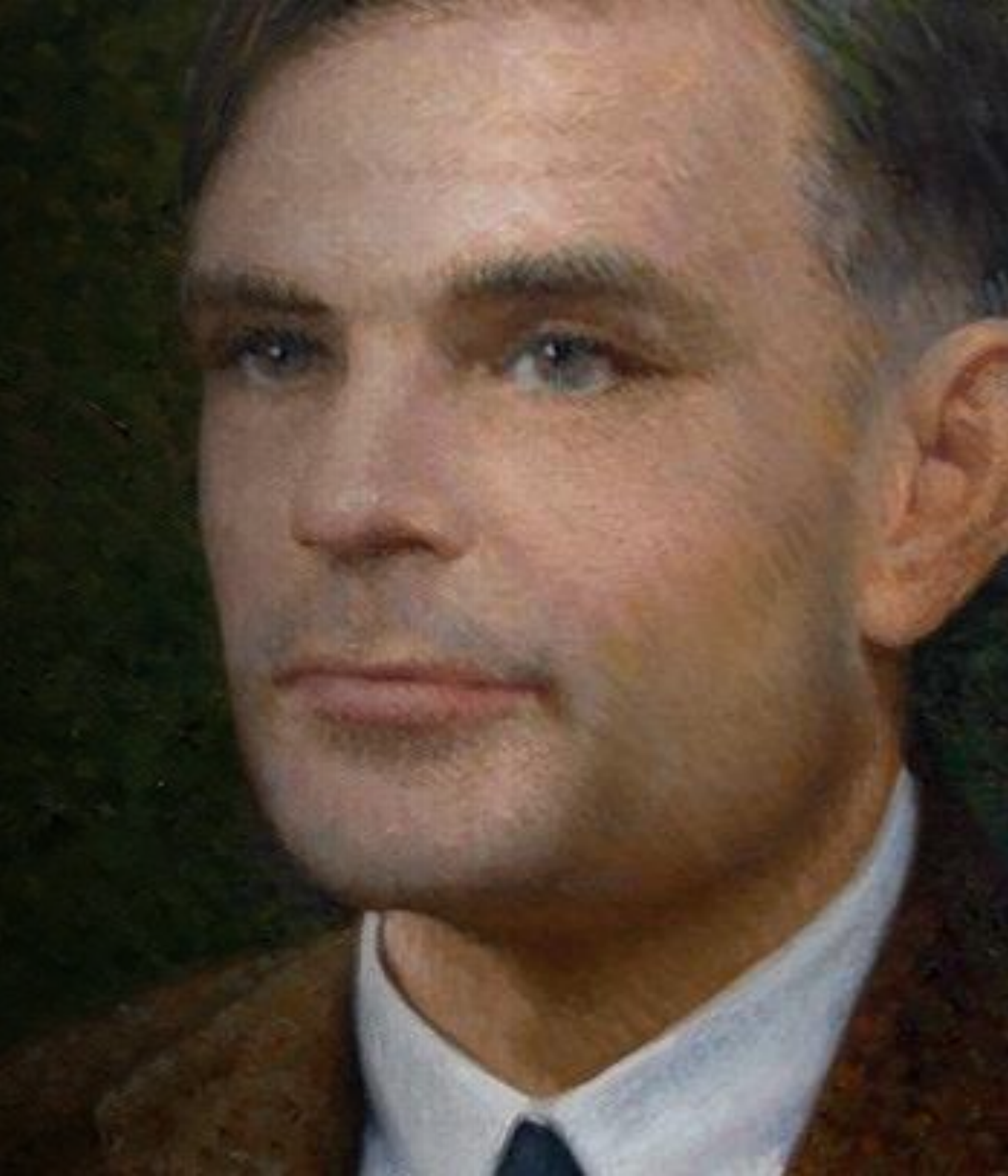

Photo: Alan Turing, Fellowship of the Royal Society

Known for: Turing Machines, Halting Problem, Cryptanalysis of Axis Enigma Machines, Turing Test thought experiment, theory of computing, computational biology, conviction and post-humous pardon under British indecency law.

Formal Methods

JD Kilgallin

CPSC:480

10/03/22

Notes

- Average on quiz 4 was a bit lower than others.
  - Remember, a functional requirement is anything the software *does*, and a non-functional requirement is a constraint on *how*. Any requirement needs to be phrased in a way that someone can determine if it's been met.
  - Review definitions of terms like "traceability", "cohesion", "coupling"
- Project 2 checkpoint feedback is up. Most missed points were:
  - Failure to follow instructions; read them over before submission.
  - Failure to capture all requirements in class and behavioral models.
  - 14 grades 95-100, 9 90-95, 5 85-90, 2 < 85. Good work!
- Midterm course review is open on learn.uakron.edu/evaluation; link on Brightspace? Please submit any feedback you have.
- Office hours/extended review after class today; exam Wednesday.
- Mid-term grades: 35% exam, 5% ea Ex, 20% ea Pr, 10% Qz, no drops

Learning objectives

- Formal methods concepts
- Purpose of formal specification and verification
- Methods of formal modeling
- Temporal Logic of Actions system

- *Only next three slides are examinable; you do not need to learn TLA.*

Formal methods

- *Formal Methods* is a class of techniques for assessing correctness of a system by modelling that system in a mathematical language.
- A software system is modeled in a formal logic with a finite state machine (or similar), and the specification is a set of logical formulas describing requirements of the system.
- *Formal verification* or *model-checking:* Given state machine M along with specification h, algorithmically determine whether or not the behavior of M meets the terms of h.
- Determining whether the formal specification captures the intended requirements is called "pleasantness" and is outside the scope of formal methods.
- Pre-dates software, back to Turing Machines modeling computation.

Benefits of formal verification

- Formal specifications can be model-checked programmatically to verify that they meet certain properties. Enables automation.
- Increases confidence in software reliability.
- Rigor of formal spec necessitates precise design, which helps catch design flaws and ambiguities.
- Gives a machine-readable reference for the system's behavior.
- Test cases and some documentation can be automatically extracted from a model - and can be updated automatically on program change.
- Some form of formal verification required for some certifications.

Drawbacks of formal verification

- Cumbersome due to advanced math, and harder to write than English + code, and is inaccessible to much of the team.
- Limited compatibility matrix of programming languages, formal specification languages, and model-checking tools compounds complexity and learning requirements.
- Some E-type systems cannot be effectively modeled.
- Specifications of systems of any complexity quickly become impossible to model or comprehend.
- Errors (or just excessive abstraction) in the model may lead to false expectations about the product.
- No guarantee of providing actionable proof or finding errors that wouldn't have been found otherwise.

Temporal Logic of Actions (TLA)

- One popular model-checking language and developer suite built for concurrent, hybrid (discrete + realtime), reactive, distributed systems.
- Specs can be written in PlusCal, an algorithm-definition language very similar to program code
- Rich IDE with TLA/PlusCal editor, PlusCal->TLA translator, LaTeX pretty-printer for TLA, TLA syntax checker, TLC model-checker
- Developed and maintained by Leslie Lamport from Microsoft Research, and major reason for his 2013 Turing Award.

TLA definitions

- A "state" is an assignment of values to all variables defined.
- A transition between states is a change in value of some variables (possibly none - called a stuttering step).
- State predicates describe individual states (e.g. "f == x < 0 /\ y > 0")
- Actions define transitions between states, using the "in the next time step" operator, " ` ". (e.g. " g == x' = x+1 /\ y'=y " describes the transition where x is incremented by one and y is unchanged).
- [] operator "always" – "[]P" = "P is true in every time step".
- <> operator "eventually" – "<>P" = "P is true now or at some future point"
- "<>P" is equivalent to "~[]~P" (~ = "not")
- []<>P "At every point, P will eventually be true again" "P is infinitely often true"
- <>[]P "At some point, P will become true and then remain true forever"

Formal specification of "compose email" form

- The recipient line, subject line, and body are initially blank.
- The user may set the cursor into any of the three control fields.
- Typing a character will increment the length of the contents of the selected field.
- Pressing backspace will decrement the length until empty.
- The subject line cannot be more than 255 characters long.
- A message cannot be sent unless it has a recipient, subject, and body.
- Can model cursor position, field length, and constraints.

MODULE *demo*

EXTENDS *Naturals*

"to", etc, represent the current length of the field. "focus" indicates which control the cursor is set to

VARIABLE *to*, *subject*, *body*, *focus*

$$Type\_invariant \triangleq to \in Nat \land subject \in (0 \,..\, 255) \land body \in Nat \land focus \in \{\text{"to"}, \text{"subject"}, \text{"body"}\}$$

$$Init \triangleq Type\_invariant \land to = 0 \land subject = 0 \land body = 0 \land focus = \text{"to"}$$

$$Select(S) \triangleq focus' = S \land to' = to \land subject' = subject \land body' = body$$

$$Type(S) \triangleq S' = S + 1$$
$$Backspace(S) \triangleq \text{IF } S > 0 \text{ THEN } S' = S - 1 \text{ ELSE } S' = S$$
$$Unchanged(S) \triangleq S' = S$$

$$\begin{aligned}
Actions \triangleq\ & \\
& Select(\text{"to"}) \lor Select(\text{"subject"}) \lor Select(\text{"body"}) \\
& \lor (Unchanged(focus) \land \\
& \quad ((focus = \text{"to"} \land Type(to) \land Unchanged(subject) \land Unchanged(body)) \\
& \quad \lor (focus = \text{"to"} \land Backspace(to) \land Unchanged(subject) \land Unchanged(body)) \\
& \quad \lor (focus = \text{"subject"} \land subject < 255 \land Type(subject) \land Unchanged(to) \land Unchanged(body)) \\
& \quad \lor (focus = \text{"subject"} \land Backspace(subject) \land Unchanged(to) \land Unchanged(body)) \\
& \quad \lor (focus = \text{"body"} \land Type(body) \land Unchanged(to) \land Unchanged(subject)) \\
& \quad \lor (focus = \text{"body"} \land Backspace(body) \land Unchanged(to) \land Unchanged(subject)) \\
& \quad \lor (to > 0 \land subject > 0 \land body > 0 \land Unchanged(focus) \\
& \quad\quad \land Unchanged(to) \land Unchanged(subject) \land Unchanged(body)) \quad \text{Send action} \\
& \quad) \\
&)
\end{aligned}$$

$$Spec \triangleq Init \land \Box[Actions]_{\{to,\, subject,\, body,\, focus\}}$$

References

- Alan Turing: the enigma. Andrew Hodges. 2012. Princeton
- Specifying Systems. Leslie Lamport. 2002. Pearson.
- The Beginning of Model-Checking: A Personal Perspective. Allen Emerson. 2008. Springer.
- Formal Verification by Model-Checking. Natasha Sharygina. 2009. Carnegie Mellon University.
- Formal Methods + TLA. Jonathan Kilgallin. 2010. Microsoft.