Northeastern University
Khoury College of
Computer Sciences

System Specification, Verification, and Synthesis CS 4830 / 7485

Stavros Tripakis

1. Introduction, Logistics

This course in one slide

- **Systems**: what is a system? How to accurately describe systems?
- **Specification**: what does it mean for a system to be "correct"? How do we formally describe correctness?
- **Verification**: how do we check correctness?
- **Synthesis**: can we automatically generate systems that are correct by construction, and how?

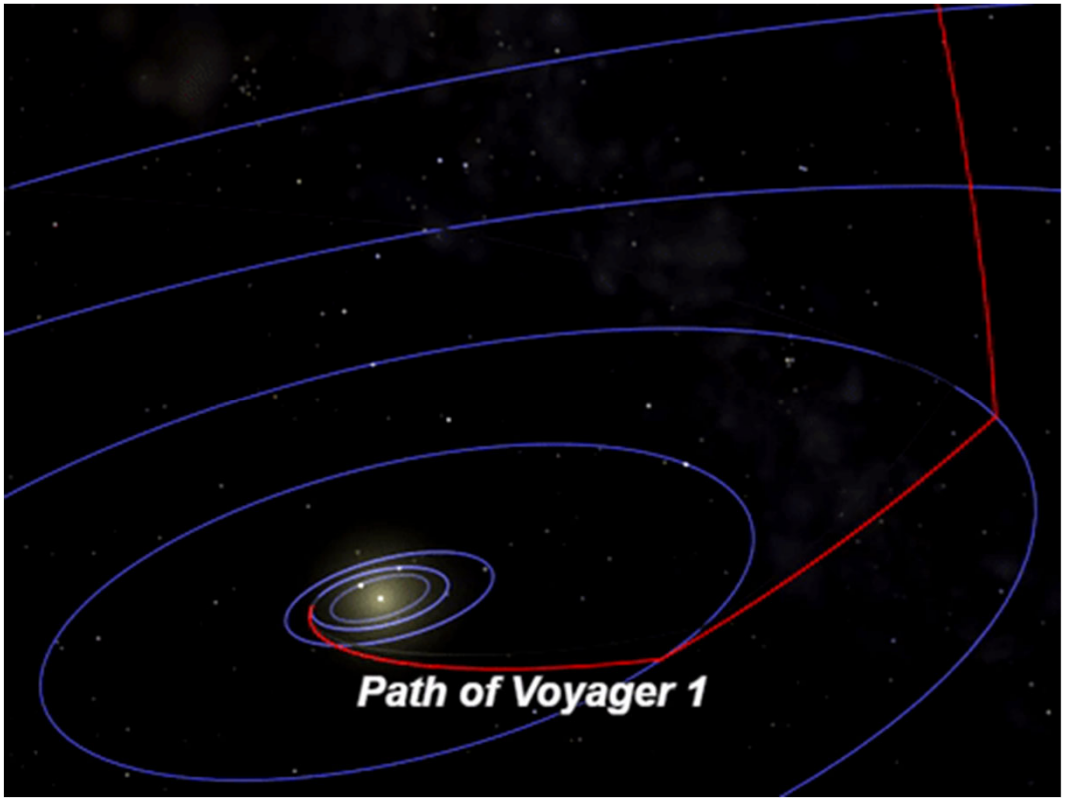
Today:

- **Motivation**: why this course?
- **Introductions**: me and you. Where are you in your studies? Where are you heading towards? What are you expecting to get out of this course?
- **Logistics**
- **Systems**: an introduction

THE SCIENCE OF SOFTWARE AND SYSTEMS

What is science?

science = knowledge that helps us make predictions



Prometheus and Epimetheus

Epimetheus (/ɛpɪˈmiːθiəs/; Greek: Ἐπιμηθεύς, which might mean "hindsight", literally "afterthinker")

Prometheus (/prəˈmiːθiːəs/ ; Greek: Προμηθεύς, pronounced [promɛːtʰéu̯s], possibly meaning "forethought")

Quote: "*It is hard to make predictions, especially about the future*"

Think of some of the sciences you know

What predictions can they make?

- Physics
- Chemistry
- Biology
- Medicine
- Law
- Psychology
- Sociology
- Economics
- Theology
- …
- Mathematics: is it a science?

What is the science of software?

What predictions can we make about the programs we write?

Can I predict that my program will:

- Terminate?
- Never throw an exception?
- Produce the right result?
- Always?
- Sometimes?
- …

Software science = **formal methods**

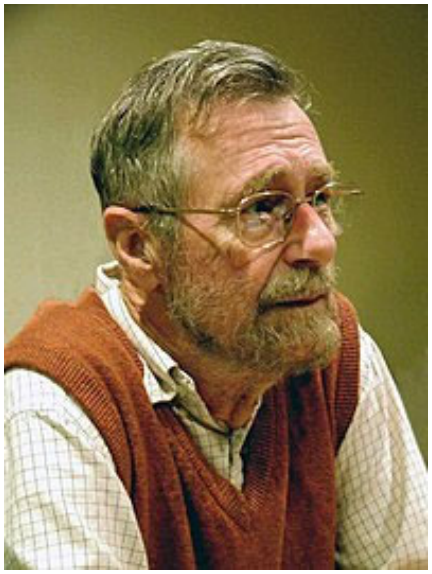
- Formal modeling
- Formal verification
- Model checking
- Theorem proving
- Static analysis
- Abstract interpretation
- Program synthesis
- SAT/SMT solving
- …

Formal methods focus: **proving** program correctness

- Proofs => formal (**mathematical**) definitions
- Proofs => we know what we mean by “correct” (**specification**)
  - E.g., is termination included in correctness?
- Proofs are hard and often tedious => need automation => **computer-aided verification**

But what about testing?

Dijkstra [1930 – 2002]



"Program testing can be used to show the presence of bugs, but never to show their absence!" [Dijkstra, 1970]

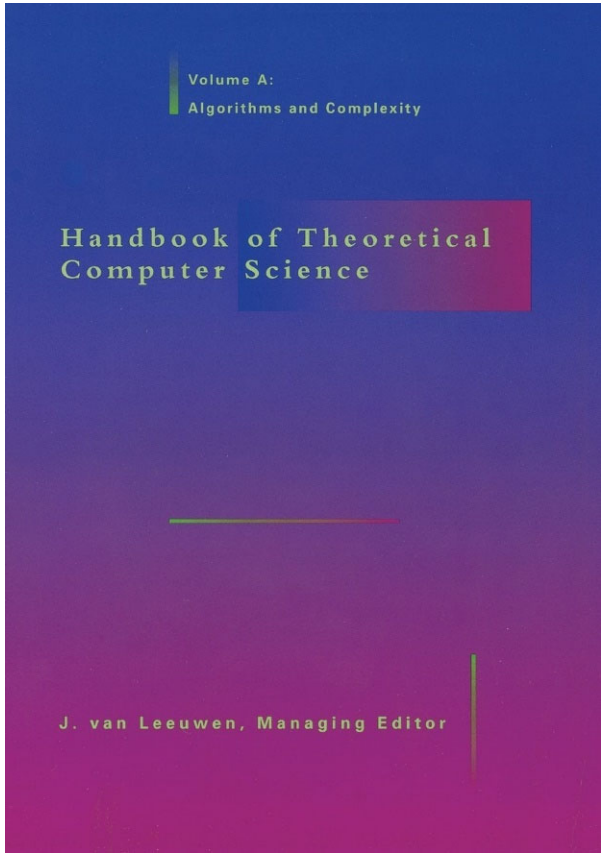
- Nevertheless, testing is fundamental, and a whole discipline by itself. It is also the workhorse of industrial software engineering practice.
- We will explore several techniques in the space between random testing and formal proofs: symbolic execution, concolic testing, … and incomplete methods such as bounded model checking, big-state hashing, …

Is this just theoretical computer science?

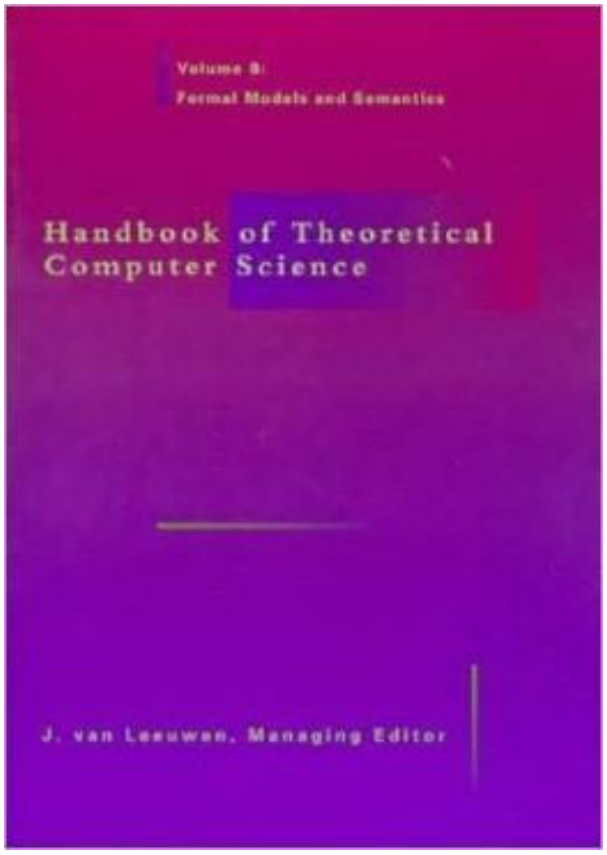
- What about theory of computation, complexity, algorithms, etc?

- Science of software ≠ science of computation
  - Science of computation:
    - What is computation? What is an algorithm?
    - What can be computed? What cannot be computed?
    - How hard/expensive is it to compute something?
    - …
  - Science of software:
    - What can I say about the programs that I write?

Theoretical computer science has (at least) two parts: (1) computability, complexity; (2) software science



Handbook of Theoretical Computer Science – 1990
Volume A: Algorithms and Complexity



Volume B: Formal Models and Semantics
Automata, Languages, Logics, Temporal Logic, Semantics, Concurrency, …

Related topics and courses

- Logic, automated theorem proving, computer-aided deduction, …
  - See Pete Manolios' course *Computer-Aided Reasoning*
- Type theory
- Programming languages
- Software engineering
  - Several courses on the above topics offered by the PL group
- Dependability, reliability, fault-tolerance, …
  - See Thomas Wahl's course on *Reliability*
- Computer-aided system design, system engineering, …

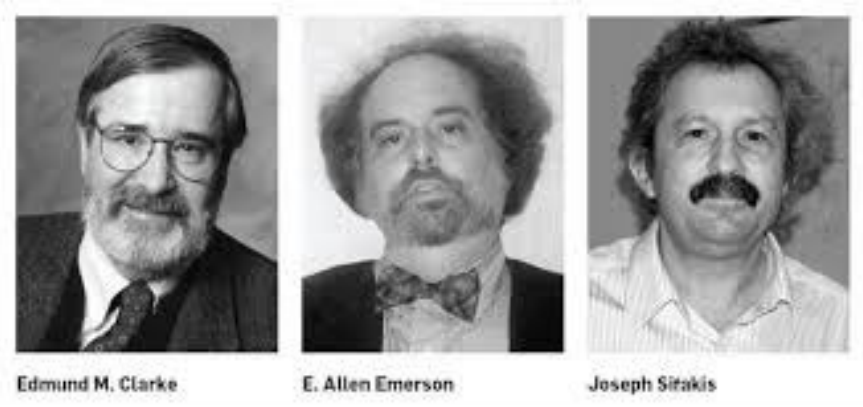
Conferences in the (broad) formal methods area

- CAV: Computer-Aided Verification (1989-)
- TACAS: Tools and Algorithms for the Construction and Analysis of Systems (1995-)
- …
- POPL: Principles of Programming Languages (1973?)
- PLDI: Programming Language Design and Implementation (1979?)
- ASE: Automated Software Engineering (1990-)
- ESEC/FSE: … Foundations of Software Engineering (1993-)
- …
- LICS: Logic in Computer Science (1988-)
- ICALP: International Colloquium on Automata, Languages and Programming (1972-)
- …

Turing awards in the FM area

Robin Milner – 1991
Theorem proving, type theory, concurrency

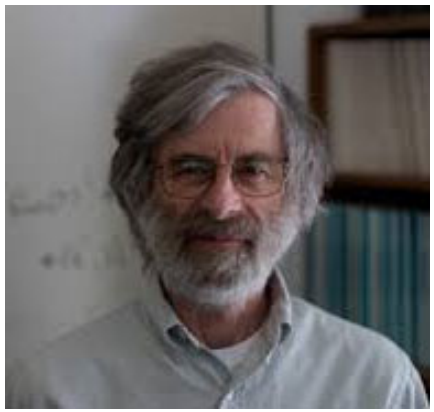
Amir Pnueli – 1996
Temporal logic, verification



Clarke, Emerson, Sifakis – 2007
Model checking

Dijkstra – 1972
Foundations of program design & correctness

Tony Hoare – 1980
Program semantics & correctness



Leslie Lamport – 2013
Distributed systems, Safety and liveness

CONTRIBUTED ARTICLES

How Amazon Web Services Uses Formal Methods

By Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, Michael Deardeuff

Communications of the ACM, Vol. 58 No. 4, Pages 66-73
10.1145/2699417

Comments (1)

VIEW AS: SHARE:

Since 2011, engineers at Am
formal specification and mo
design problems in critical s
motivation and experience,
domain, and what has not. V
we refer to the authors by th

At AWS we strive to build se
to use. External simplicity is
complex distributed systems

Key Insights

- Formal methods find bugs in system designs that cannot be found through any other technique we know of.
- Formal methods are surprisingly feasible for mainstream software development and give good return on investment.
- At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.

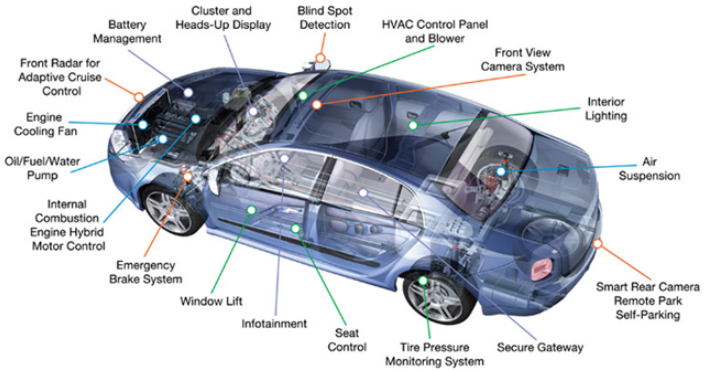
Tripakis

Thesis: Formal methods = modern system theory

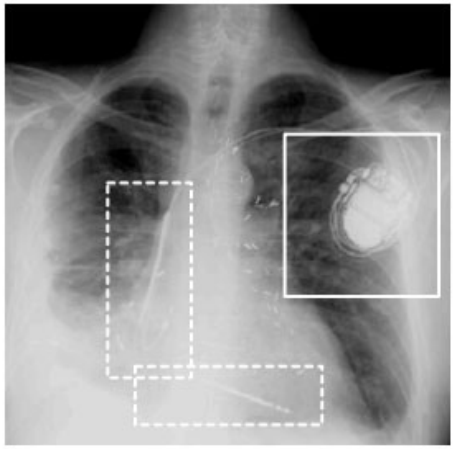
- System modeling: states + transitions
- System composition
- System equivalence
- System abstraction
- System specification
- System semantics (behavior)
- Safety/liveness (qualitative) and quantitative properties
- State space, reachability, (inductive) invariants, deadlocks, …
- …
- Safety-critical systems, distributed systems, real-time systems, hybrid systems, embedded systems, cyber-physical systems, …
- Other theories (e.g., linear differential equations) are (very useful) **special cases**

Some industrial application domains other than “pure” software

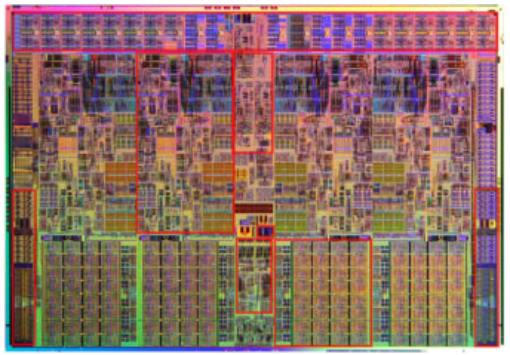
Aerospace/defense



Automotive

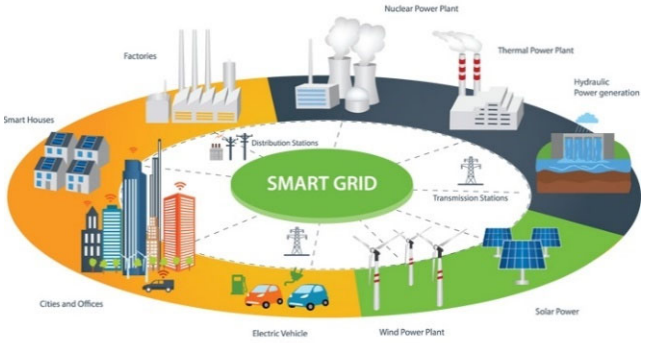


Medical



Electronics Design Automation/EDA (chip design)

Nuclear energy



“Smart” infrastructure

Cyber-physical systems: present

Autonomous car driving through red light

‘Cyber-physical’ systems: future

Courtesy `https://vimeo.com/bsfilms`
Thanks to Christos Cassandras for recommending this video

Some more claims

1. None of these systems could be imagined, let alone built, without software

2. Everything, or almost, is software
   - When you design traditional software you program in C, Java, Python, …
   - When you design hardware you write Verilog code: Verilog programs are software
   - When you design embedded controllers you write Matlab/Simulink code: this too is software
   - When you design a robot?
   - When you design a self-driving car?
   - When you design a market trading algorithm?
   - When you design a drug?
   - …

3. Software is the most complex artifact ever built by humans

A simple program: what does it do?

```
int x := input an integer number > 1;

while x > 1 {
  if x is even
      x := x / 2;
  else
      x := 3*x + 1;
}
```

```
Run starting at 31: 31 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466
233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425
1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077
9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20
10 5 16 8 4 2
```

Collatz conjecture:
The program terminates for every input.
Open problem in mathematics.

What is the mathematics of the science of software?

Language

Truth

…

logic

Logic in this course

- We will use logic as a communication language: definitions, proofs, etc.
- We will also study non-traditional logics developed specifically for model checking (temporal logic)
- Courses focusing on logic and theorem proving:
  - CS 2800 Logic and Computation
  - CS 4820 Computer-Aided Reasoning
  - Software Foundations: online book series/course based on the Coq theorem prover

Quiz (not graded)

Express the following statements in your favorite mathematical formalism:

(a) – You can fool some people sometimes

(b) – You can fool some of the people all of the time

(c) – You can fool some people sometimes but you can't fool all the people all the time [Bob Marley]

(d) – You can fool some of the people all of the time, and all of the people some of the time, but you can not fool all of the people all of the time [Abraham Lincoln]

You can fool some people sometimes

$\exists$ lie, person, time : Believes (lie, person, time)

$\exists$ l, p, t : B(l, p, t)

$\exists$ l $\in$ Lies, p $\in$ People, t $\in$ Time : B(l, p, t)

$\exists$ l, p, t : B(l, p, t) $\wedge$ l $\in$ Lies $\wedge$ p $\in$ People ...

$\exists$ p $\in$ People : F(Fool(p))

$\exists$ p : Reality $\to$ R ... : F (p(☺) $\neq$ ☹)

(a) $\exists p, t : Fool(p, t) = \exists p : \exists t : Fool(p, t)$

(b) $\exists p : \forall t : Fool(p, t)$

(f) $\forall p : \exists t : Fool(p, t)$

$\forall t : \exists p : Fool(p, t)$

(e) $\forall p, \forall t : Fool(p, t)$

(c) (a) $\wedge \neg$ (e)

(g) $\exists t : \forall p : Fool(p, t)$

(f)

Stavros → Mon Sep 11, 2001

Pete → Tue

Jane → ..

In the end, is it worth it?

- Isn’t it too hard to do formal modeling and proofs?
  - It depends on what you compare it to
  - Testing becoming more and more expensive
  - C.f. CACM Amazon Web Services article
- Why bother? Isn’t testing enough?
  - No

Boeing 737 Max 8 accidents

- 2 accidents within 5 months – 346 deaths
- 737 Max 8 planes grounded world-wide since March 2019
- Control system rather than pilot errors
- Dubious business and certification practices

ROUND OF INTRODUCTIONS

Stavros Tripakis

Associate Professor, Northeastern University (since 2018)

- Past:
  - Full Prof., Aalto University, Helsinki, Finland, 2012 - 2018
  - Adjunct Assoc. Prof., UC Berkeley, 2009 - 2018
  - Research Scientist: Cadence Design Systems, Berkeley, 2006 - 2008
  - Postdoc: Berkeley, 1999 – 2001
  - Research Scientist: CNRS, Verimag, France, 2001 – 2006
  - PhD: Verimag Laboratory, Grenoble, France, 1998
  - Undergrad: University of Crete, Greece, 1992

- Research interests
  - Formal methods, model checking, conformance testing, controller/program synthesis
  - Compositionality, contracts, interfaces
  - Embedded and cyber-physical systems
  - Security
  - Formal methods and AI/machine learning

Round of introductions

- Your name
- Your degree and year
- If you are a graduate student, who are you working with / what topic
- Your interests / goals for this course
- Your interests / goals after graduation
- Anything else (e.g., fun things I do outside school / did this summer / …)

LOGISTICS

Syllabus

- Week 1: Introduction, motivation, logistics, systems
- Week 2: Formal system modeling:
  - State machines, automata, transition systems, state spaces
  - Examples: circuits, software, embedded systems, bio systems, AI
  - The model checkers Spin and NuXMV
- Weeks 3-4: Formal specification:
  - Invariants, assertions, regular properties (finite behaviors), infinite behaviors, safety, liveness
  - Temporal logics: LTL, CTL, linear/branching time
  - LTL and CTL in Spin and NuXMV
- Week 5-7: Model checking:
  - State-space exploration, reachability analysis, state explosion
  - Symbolic methods, BDDs
  - CTL model checking, fixpoints
  - LTL model checking, omega automata, Buchi automata, the automata-theoretic method
- Week 8: Incomplete methods:
  - Bounded model checking using SAT/SMT solvers
  - Symbolic execution, concolic testing, …
- Week 9: Compositional verification, assume/guarantee methods, contracts
- Weeks 10-11: Synthesis: controller and program synthesis
- Weeks 12-13: Project presentations / final exam / further topics:
  - Software verification, abstract interpretation, theorem proving, AI verification, …
  - Timed automata, hybrid automata, …

Not covered (partial list)

- Type theory, PL, etc (as mentioned earlier)
- Theorem proving, SAT/SMT, etc (as mentioned earlier)
- Static analysis
- Abstract interpretation (although we will talk about abstraction)
- Software verification in-depth (although we will talk about invariants, inductive invariants, maybe Hoare triples time permitting)
- Concurrency theory, Petri nets, process algebra, …
- Theory of formal languages and automata in-depth

Lectures and questions

I like my lectures to be interactive:

- I depend on you to make this happen

Ask questions if you don't understand something

- OK to interrupt me
- OK to object to something I say

Remember: there are no stupid questions

- Mistakes are the only way to learn
- Learning theory confirms this (learning from positive and negative examples)

Reading

- Textbooks:
  - Principles of Model Checking, by Baier and Katoen
  - Model Checking, by Clarke, Grumberg and Peled
  - Books on Spin by Gerard Holzmann (hands-on)
  - Logic in Computer Science, by Huth and Ryan (has chapter on TL)

- Going deeper:
  - Handbook of Model Checking, by Clarke, Henzinger, Veith, Bloem
  - Papers in conferences like CAV, TACAS, POPL, …
  - Various tool competitions:
    - SAT (http://www.satcompetition.org/) since 2002, SMT (https://smt-comp.github.io/) since 2005, SV-COMP (https://sv-comp.sosy-lab.org/) since 2012, hardware model checking (http://fmv.jku.at/hwmcc17/) since 2007, reactive synthesis (http://www.syntcomp.org/) since 2014, syntax-guided synthesis (https://sygus.org/) since 2014, …

Tools

- The NuXMV model checker:
  - https://nuxmv.fbk.eu/

- The Spin model checker:
  - http://spinroot.com/

- **Install and familiarize yourselves with these tools ASAP**

Web page

- http://www.ccs.neu.edu/~stavros/ssvs19.html

- Go over web page