Crema

devastating capability, revolutionary advantage

Research Briefing

Jacob Torrey & Mark Bridgman

May 21, 2015

In a Nutshell

- **What are we doing?**

Crema was a program to explore the sub-Turing complete (TC) programming languages and execution environments. By restricting the computational expressiveness of programs to the minimum needed to perform the programmer's intent, "Weird machines" can be eliminated and more powerful formal methods explored

OR

Give the developers the programming tools to make development of safer & more secure software easier and automated analysis problems easier

Objective

devastating capability, revolutionary advantage

- **Crema aimed to demonstrate the feasibility and security benefits of general purpose sub-TC programming languages**
  - Create a language and corresponding execution environment that is purposefully sub-TC
  - Explore computing tasks that explicitly need Turing completeness and how to logically isolate them

- **Explore the impact on formal methods when the computational models are restricted**
  - With Crema, software can be analyzed with more *granularity* and/or at *larger scale*
  - Analyses undecidable for TC languages may be possible

Background Information

- **Weird Machines**

- **LLVM**
  - Modular and abstracted open-source compilation tool-chain
  - Compiles to immediate representation (IR) for advanced optimization and static analysis/symbolic execution

- **KLEE**
  - Performs symbolic execution using LLVM IR
  - Executes most/all code paths in program to explore for crash/error states

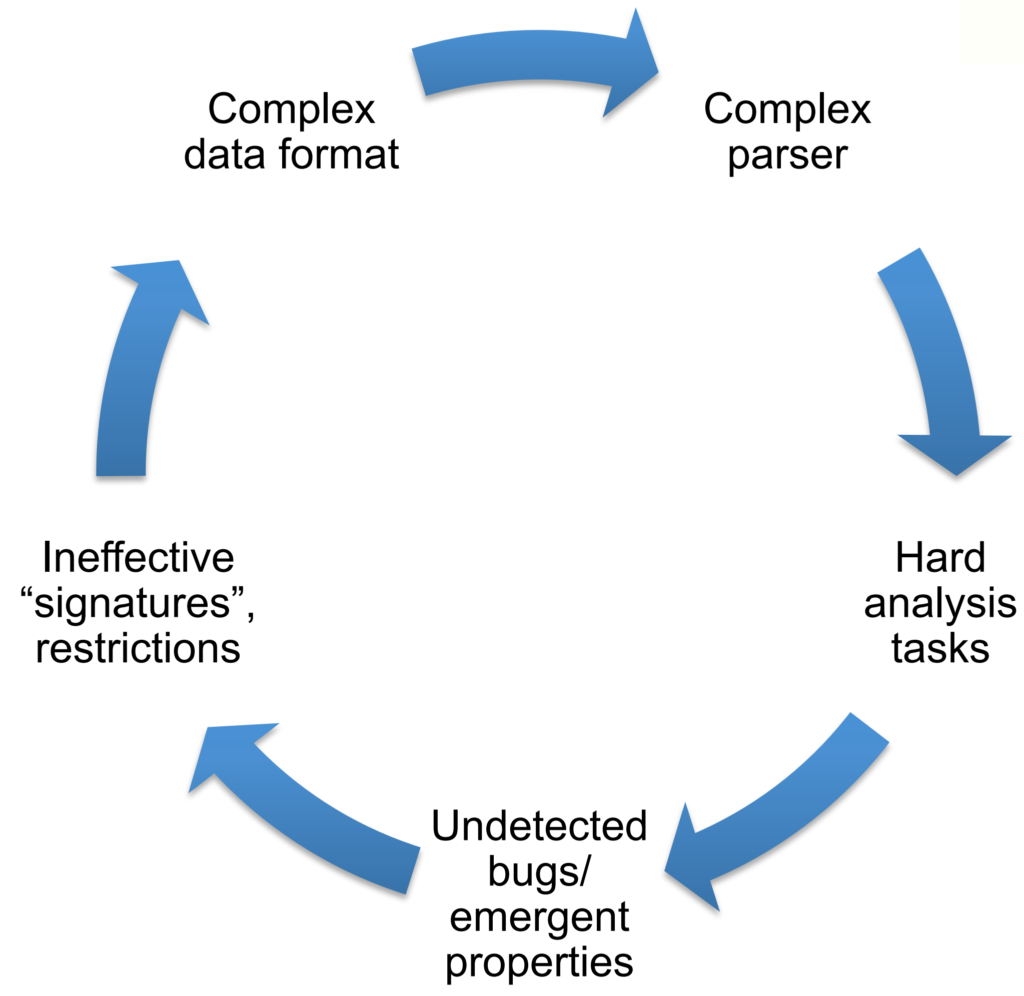
Problem Statement

devastating capability, revolutionary advantage

- **TC languages provide more expressiveness than most programmers need for most tasks**
  - This often leads to unintended emergent behaviors or "weird machines" programmed by attacker
- **The majority of general purpose programming languages are designed to be TC and are susceptible to weird machine behavior**
- **Turing completeness and the Halting problem makes certain forms of formal methods/program analysis undecidable or intractable, decreasing software quality**

“Circle of Bugs”

devastating capability, revolutionary advantage



Complex data format
Complex parser
Hard analysis tasks
Undetected bugs/ emergent properties
Ineffective “signatures”, restrictions

Input Parsing is Safety-Critical

- **Problem: code receives attacker-controlled inputs, inputs drive execution flow, system enters untrustworthy state (often enabling arbitrary computation by attacker)**
  - General computation model is needed for input handling (model must at least match practice!)
- **Predicting behavior of a complex code (e.g., parsers) on inputs is hard to impossible**
- **Software verification: producing proofs that software remains in the safe, intended state no matter what the inputs**
  - Challenge: state explosion, some properties cannot be established or proved algorithmically

“Undecidability Cliff”

- **The more powerful/expressive an execution environment is, the harder it is to analyze**

- **Automated analysis tends to become provably impossible after a complexity threshold**
  - Hierarchies of complexity exist to describe such thresholds

- **Undecidability "cliff":**
  - Automatically recognizing whether a TC program halts or loops forever is impossible
  - Automatically verifying if two parsers are equivalent becomes undecidable at "non-deterministic context free”

Technical Approach

devastating capability, revolutionary advantage

- **Developed proof-of-concept sub-TC language front-end for LLVM**
  - Designed to be general purpose and minimal learning curve
  - Could be used as parser “bridge” into a formal type system
  - Used in *transducers/*parsers that convert input into structured data
- **Explore sub-TC space from LangSec perspective**
  - How the lack of halting problem reduces weird machines
  - Limits power given to attackers in the event of compromise
- **Explore program analysis improvements**
  - More granular checks are now possible to verify correctness
  - State-space growth shown to be slower/SMT problems easier

Results

And Impact

devastating capability, revolutionary advantage

- **Prototype language (Crema)**
  - Sub-TC
  - Easy to learn
  - Capable of performing most* programming tasks
  - Open source (https://github.com/ainfosec/crema)
- **KLEE on C versus Crema highlights benefit for state-space explosion**

Crema Example

devastating capability, revolutionary advantage

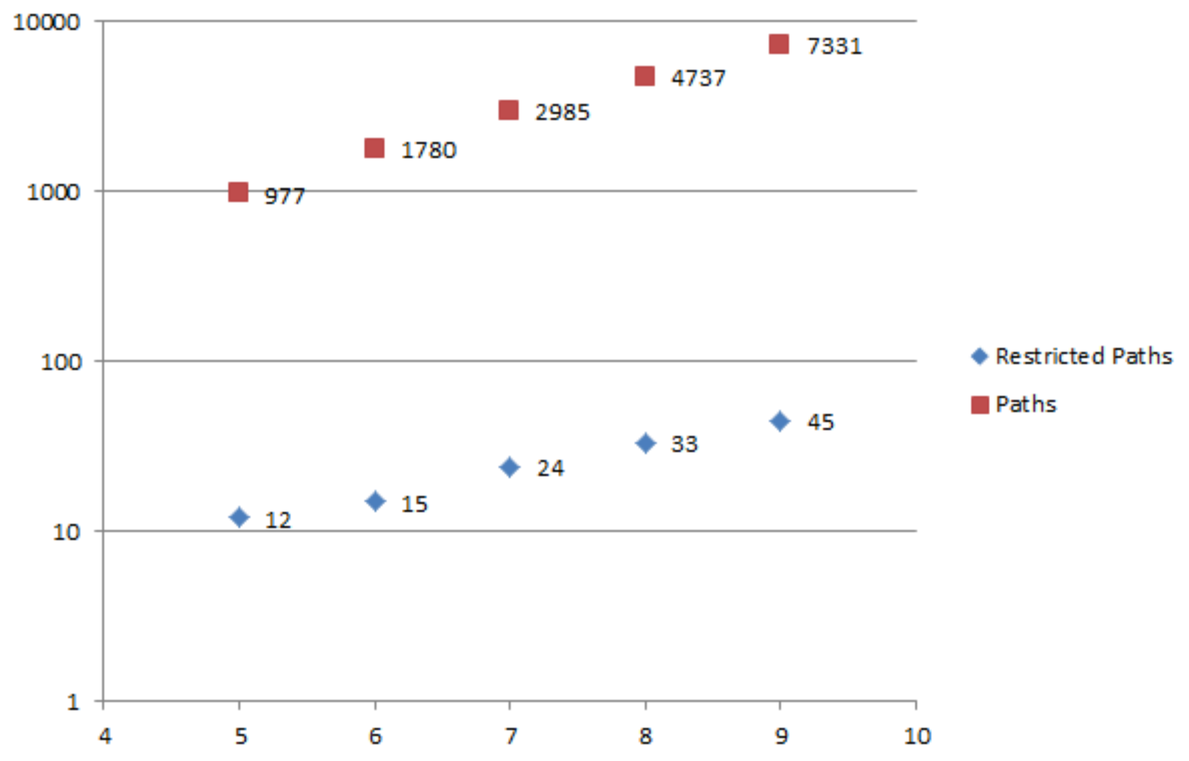
- “FizzBuzz”

```
int hundred[] = crema_seq(1, 100)

foreach(hundred as i) {
  int_print(i)
  str_print(" ")
  if (i % 3 == 0) {
    str_print("Fizz")
  }
  if (i % 5 == 0) {
    str_print("Buzz")
  }

  str_println(" ")
}
```

State-Space Growth

devastating capability, revolutionary advantage

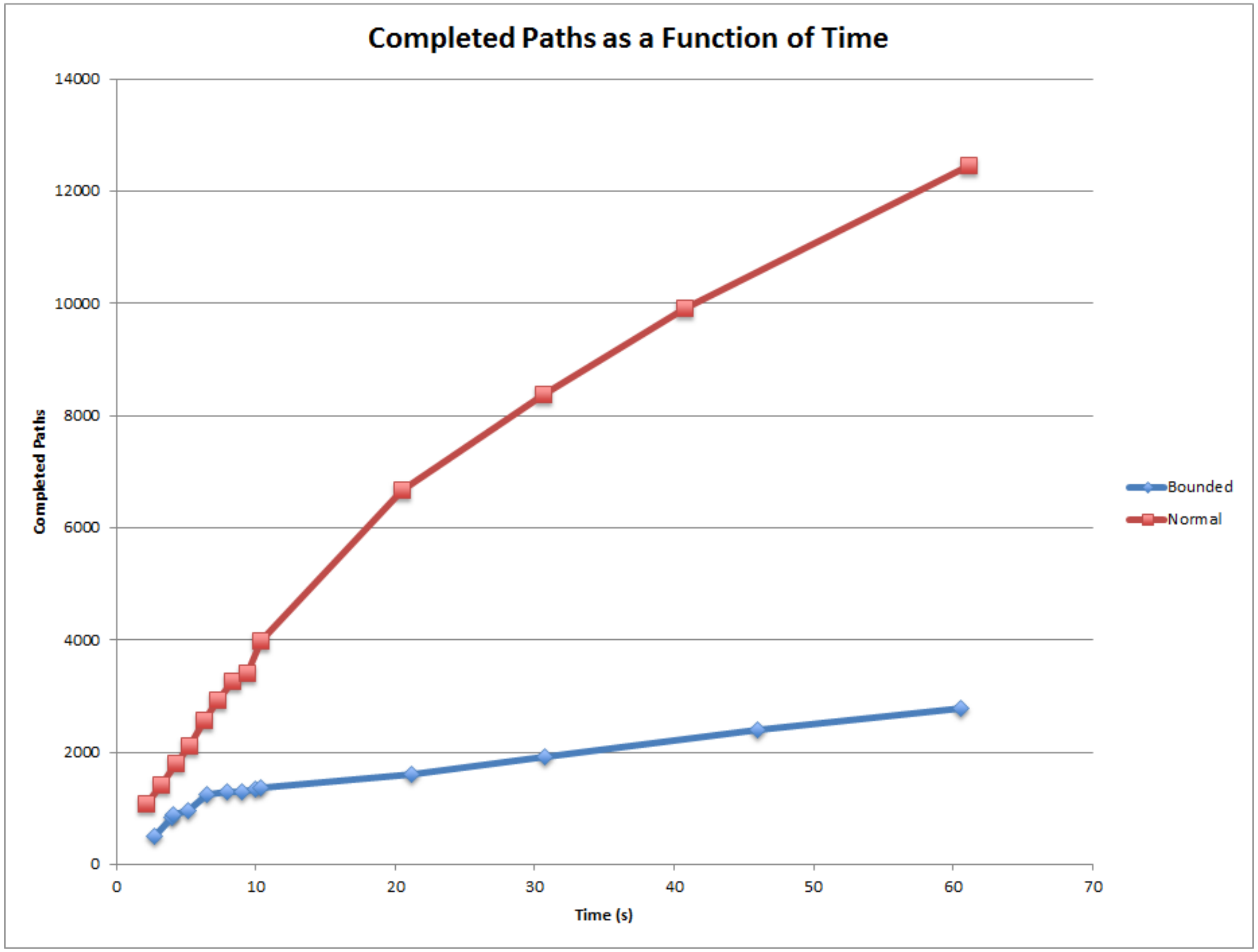
- **Symbolic input length vs. number of paths to search**



State-Space Growth II

devastating capability, revolutionary advantage

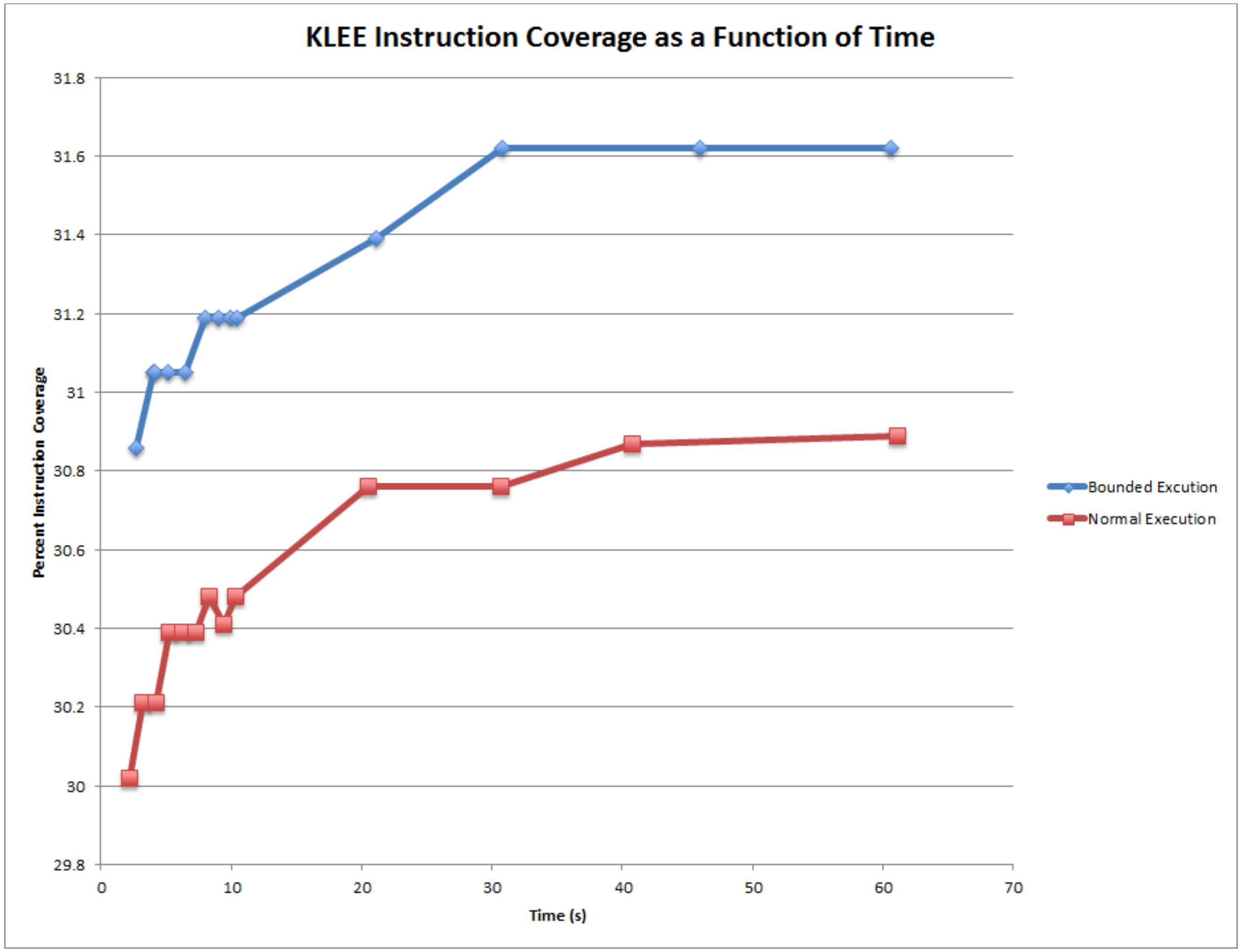
- **Paths to search as function of time:**



State-Space Growth II

devastating capability, revolutionary advantage

- **Instruction Coverage***



State-Space Growth (cont.)

devastating capability, revolutionary advantage

- **Qmail C-language parser versus Crema parser for SMTP:**
  - SMT solving time creates bottle-neck

Parser	Execution Time (s)	Max. States	Instruction Coverage (%)	Branch Coverage (%)
Qmail C	31.50	678	44.47	33.96
Crema	28.67	76	61.97	37.74

Reference Monitor Implications

devastating capability, revolutionary advantage

- Reference monitors are automatons recognizing a *language of events*
- Currently are prefix-based (can only identify bad patterns of events early)
- With a Walther-recursive model, can strengthen the "power" of the reference monitors to a larger language set

Impact

devastating capability, revolutionary advantage

- **Break the cycle of complexity**
- **Provides ability to verify software previously out of range for contemporary methods**
- **Automatically limits risks incurred through poor programming practices**
- **Answers key questions and provides <u>empirical data</u> on restricted computational models**

Future Work

devastating capability, revolutionary advantage

- **Restricting JIT compilation model in LLVM/HW**
  - While the program source may be sub-TC, an attacker may be able to inject TC LLVM IR
  - FPGA or customizable CPU environment
    - Crema LangSec paper modeled a CPU environment with a bit set to enforce “forward-only execution” of loop-unrolled programs
    - Bring Crema benefits to hardware and embedded
- **More powerful formal methods tools**
  - What is possible now that was once infeasible?
    - When there is no concerns over termination and undecidability, what FM techniques can now be implemented/made tractable
  - GCC/LVVM static analysis hinting to programmers to use restricted semantics
    - “Please write this portion in Crema”

Future Work (cont.)

devastating capability, revolutionary advantage

- **Automatic source conversion/TC detection**
  - Translate existing source where possible
  - Identifying TC regions or where human-in-the-loop is needed
- **Identify security-sensitive code regions, specifically for handling input parsing and limiting expressiveness in those regions**
  - Easy for programmers to create code that cannot be analyzed; Crema provides framework to describe semantics to verifier
  - “IR” for representing verification hints and challenges
  - Lessens expertise required for formal verification
- **Hammer port to Crema**
  - Formally verified reference implementations of parsers
- **Solve P=NP to reduce SMT solving times ;)**

Summary

devastating capability, revolutionary advantage

- **Re-envision programming language development**
  - Prevent feature-creep in formal language development as we do in software development
  - Powerful enough for many tasks, easy enough for analysis/ verification – the "sweet spot"
    - The only safe method for input-driven programs
- **Analyze sub-TC languages and environments through lens of LangSec**
- **Explored new capabilities for formal methods**
  - Analysis highlighted benefits of restricted environment vis-à-vis verification state-space growth

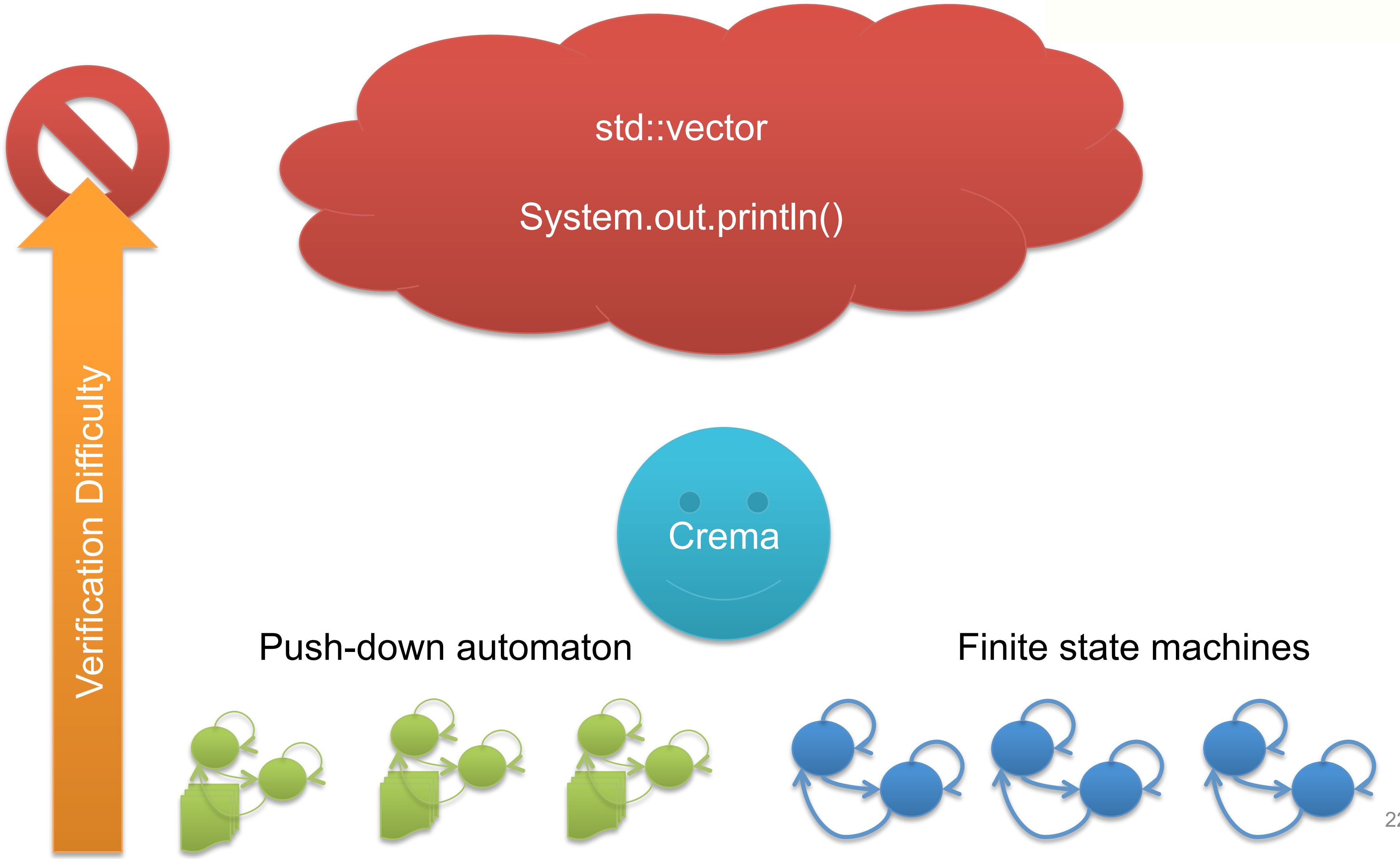
Acknowledgments

devastating capability, revolutionary advantage

- We would like to thank DARPA and Dr. John Everett for sponsoring this work

- Additional thanks to Sergey Bratus, Halvar Flake and Julien Vanegue for their input and support of this work

Questions/Discussion

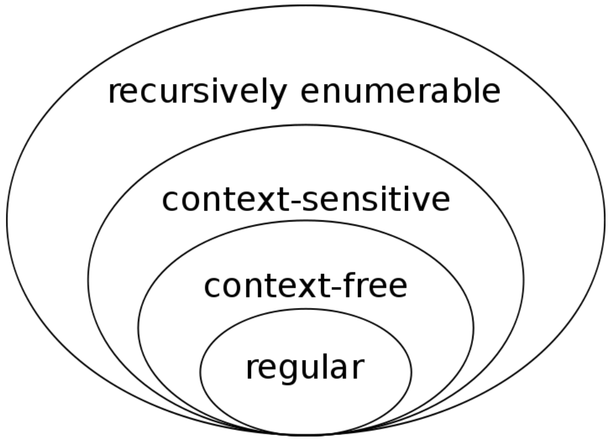


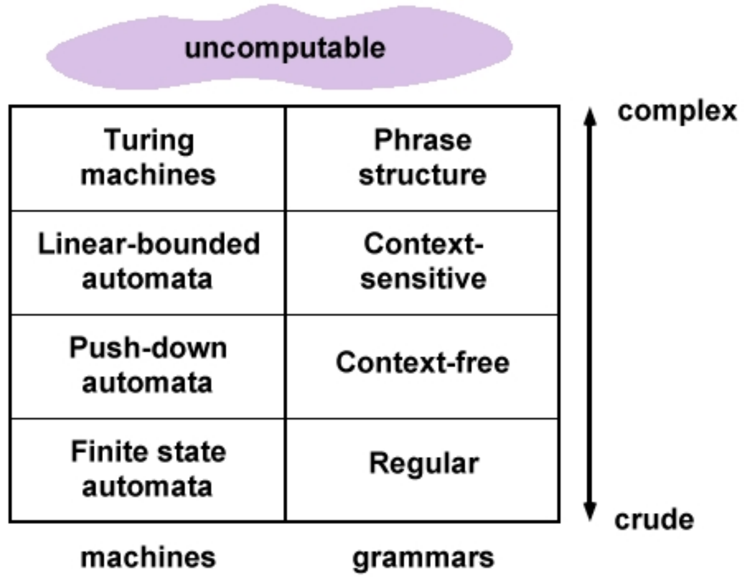
Backup

devastating capability, revolutionary advantage

A Hierarchy of Complexity

- **Chomsky's Hierarchy**
  - Provides a structure to the expressiveness of languages and the power of the recognizers needed to parse them





machines	grammars
Turing machines	Phrase structure
Linear-bounded automata	Context-sensitive
Push-down automata	Context-free
Finite state automata	Regular

Turing Completeness

devastating capability, revolutionary advantage

- **Turing completeness**
  - A Turing machine is theoretical device which consists of an infinite ‘tape’ of cells, each that can contain a symbol from some defined alphabet
  - TC is used to describe a language that is capable of fully simulating a Turing machine