On the Probabilistic Symbolic Analysis of Software

Corina Pasareanu

CMU-SV

NASA Ames

Probabilistic Symbolic Execution

- Quantifies the likelihood of reaching a target event
  - e.g., goal state or assert violation
  - under uncertainty conditions from the environments
- Example
  - check that the probability of an unmanned aerial vehicle turning too fast is less than 10^-6
  - analyze the vehicle's control software
  - under suitable probabilistic profiles built from the telemetry data of hundreds of hours of operation from previous versions or similar systems
- Simulation
  - traditionally used
  - very expensive
- Probabilistic symbolic execution
  - complements simulation
  - for increased assurance at reduced cost

Probabilistic Software Analysis

- Traditional approaches are based on probabilistic model checking, e.g. PRISM
- Models
  - difficult to maintain
  - abstract away details that impact chances of executing target events
- We aim to perform the analysis at code level

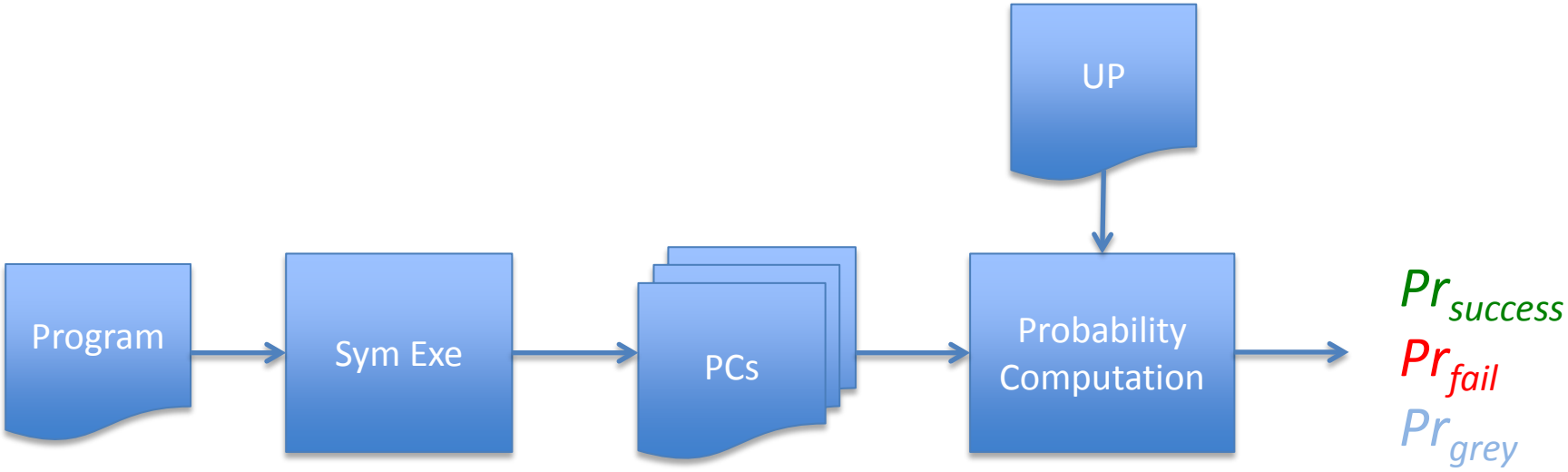
Probabilistic Software Analysis

- Traditional approaches are based on probabilistic model checking, e.g. PRISM
- Models
  - difficult to maintain
  - abstract away details that impact chances of executing target events
- We aim to perform the analysis at code level

Probabilistic Symbolic Execution

- Bounded symbolic execution
  - generates symbolic constraints over program paths
- Quantification procedure, e.g., model counting
  - quantifies the probability of satisfying the constraints
  - when target event (success or fail) reached or bound reached, estimate the number of inputs that satisfy the conditions to reach that event
- Applications
  - Computing reliability: when software is involved in contributing to a system-level event
  - Analysis of cyber-physical systems
  - Quantitative information flow analysis for security
  - Program understanding and debugging
  - Certification …

Probabilistic Symbolic Analysis



Usage Profiles

what are they?

- probabilistic characterizations of software interactions with external environment
  - users, physical world, other components
- assign to each valid combination of inputs the probability to occur during execution

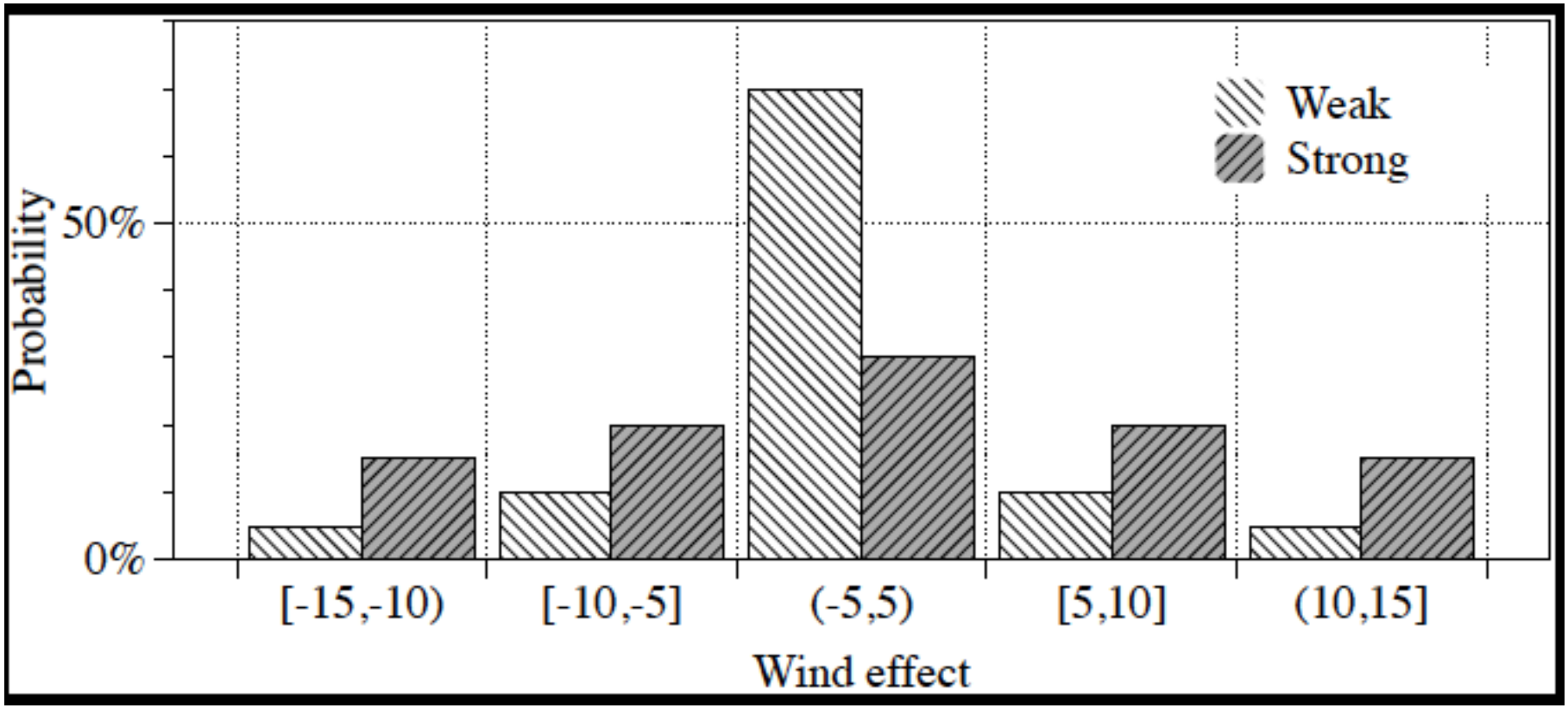
$$<c_1,p_1> <c_2,p_2> \ldots$$

who creates them?

- monitoring usage of previous versions/similar systems
- expert/domain knowledge
- physical phenomena, e.g. wind effect

we assume they are given

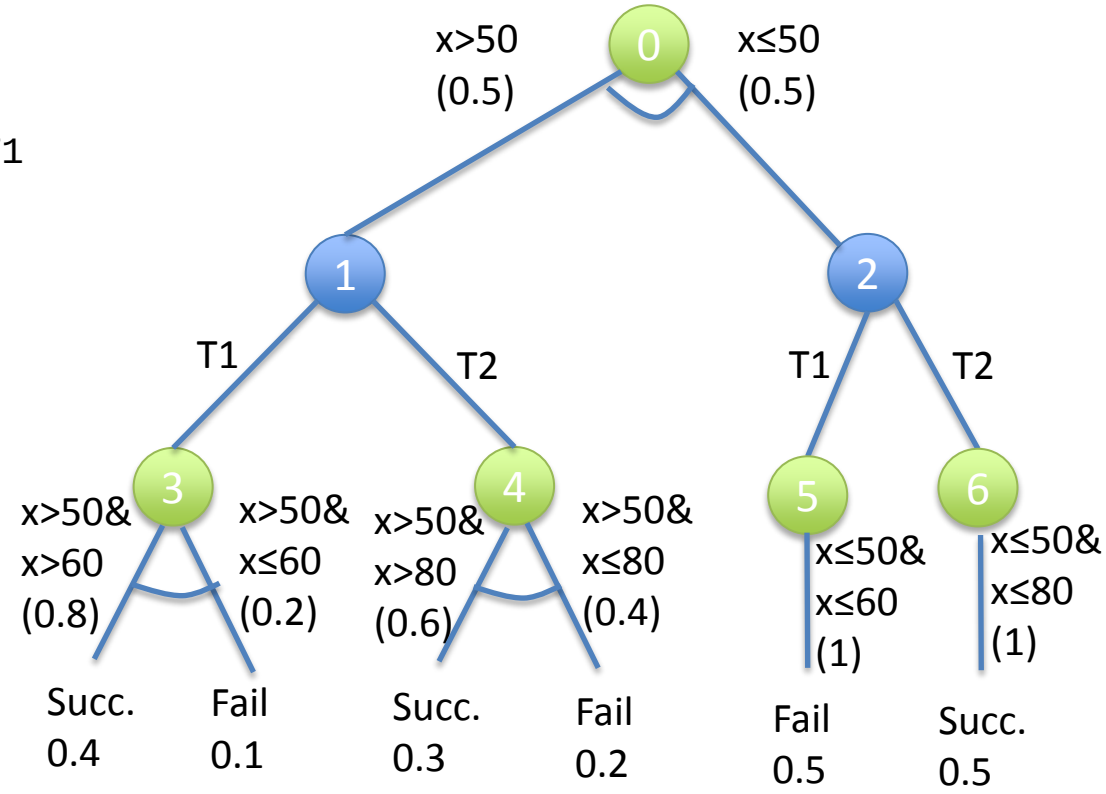
Example Usage Profile



- Arbitrary UPs – handled through discretization
- UPs can be seen as “pre-conditions”

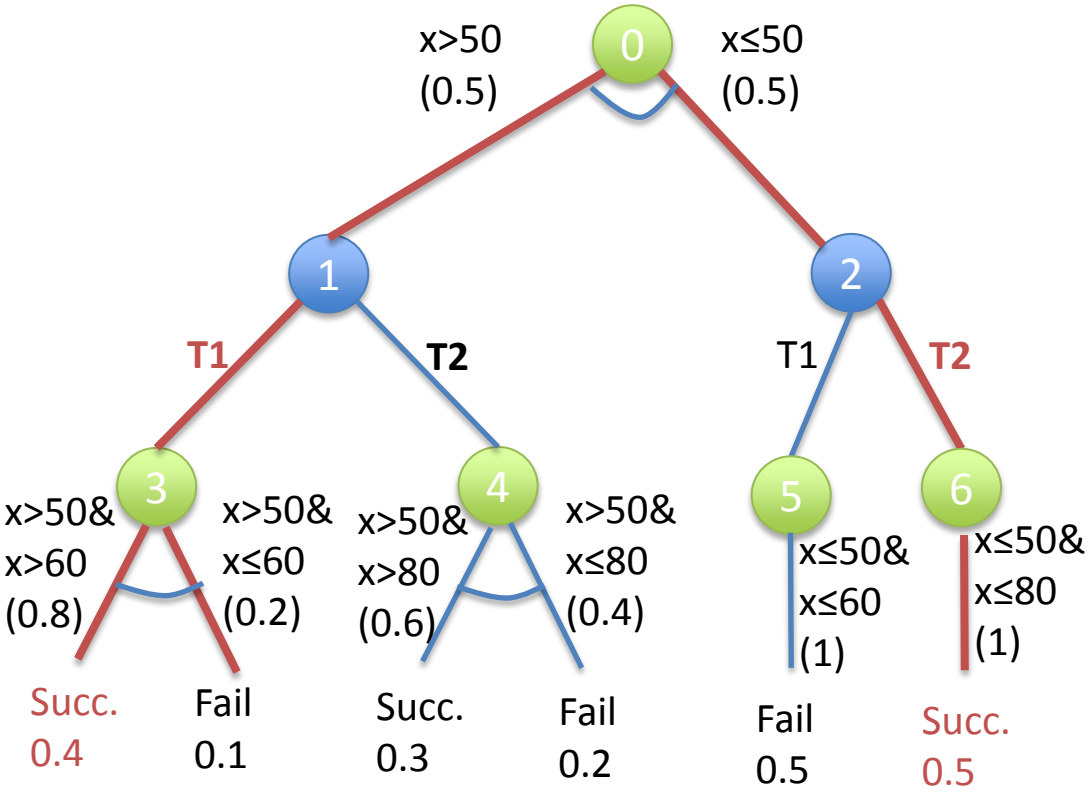
Example

```
// domain of x is [0..100]
public static void test(int x){
  if(x > 50)
    x++;
  if(Verify.getBoolean()) { //T1
    if(x > 61)
      println("success");
    else
      assert false;
  } else { //T2
        if(x <= 81)
          println("success");
        else
          assert false;
     }
  }
```



Scheduler: resolves non-deterministic choices to maximize probability of success

Example



Max: $Pr_{success}$ =0.9

Statistical Symbolic Execution

- Approximate algorithms
  - Monte-Carlo sampling of symbolic paths based on conditional probabilities in the symbolic execution tree
  - Reinforcement learning to iteratively computing schedulers
  - Pruning of already explored paths

Comparison with classical Monte-Carlo simulation

- For each explored path
  - We compute the full count associated with the PC; we need to explore each path only once; our approach enables aggressive pruning
  - Simulation needs to sample many many times along the same paths to achieve the desired confidence
- Usage profiles
  - Summarize hundreds of hours of operation/simulation

Summary

- White-box methodology for finite domains using model counting, with explicit measure of confidence [ICSE 2013]
- Dealing with floating-point numbers and arbitrarily complex constraints [PLDI 2014]
- Statistical techniques for increased scalability [FSE 2014]
- Synthesis of tree-like schedulers for multithreading [ASE 2014]
- Improved support for data structures

Future Work

- Infer usage profiles from telemetry data
- Compute admissible input distributions to guarantee certain probabilistic safety properties
  - ACASX: airborne collision avoidance system
- So far we have studied *memory-less* schedulers
  - May not be enough for computing maximal properties for bounded properties
  - History dependent schedulers are more powerful
- Parallel sampling
  - Preliminary implementation
  - Some overhead due to thread contention
  - Reduce analysis time by 30%

Collaborators

- Antonio Filieri (University of Stuttgart, Germany)
- Kasper Luckow (Aalborg University, Denmark)
- Willem Visser and Jaco Geldenhuys (University of Stellenbosch, South Africa)
- Marcelo d'Amorim and Mateus Borges (Federal University of Pernambuco, Brazil)
- Matt Dwyer (University of Nebraska, Lincoln, USA)

http://ti.arc.nasa.gov/profile/pcorina/probabilistic/

“Rare event”

```
// domain of x is [0..100]
void testMethod (int x) {
  if (Verify.getBoolean()) {
    if (x < 2) {
     ... println(" success ");
     return ;
    } else {
     if (Verify.getBoolean())
     if (Verify.getBoolean())
      ... // repeat 500 times
     if (x > 5) {
      ... println(" success ");
      return ;
     }
    }
  }
  assert false ;
}
```

$Pr_{success}$ =0.96

Hard to compute with approximate techniques

Pruning helps