Digital Trust & Security Guest Lecture Series
University of Manchester

Automated Software Verification and Synthesis in Unmanned Aerial Vehicles

Lucas Cordeiro
School of Computer Science
lucas.cordeiro@manchester.ac.uk

Outline

Verification and Synthesis Overview

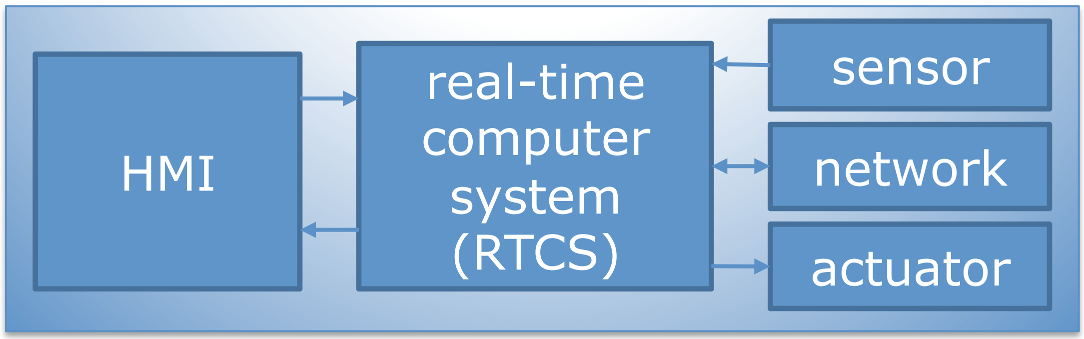
Vision for Future Research

Synergies and Potential Collaboration

Verifying Embedded Software in UAV is Hard

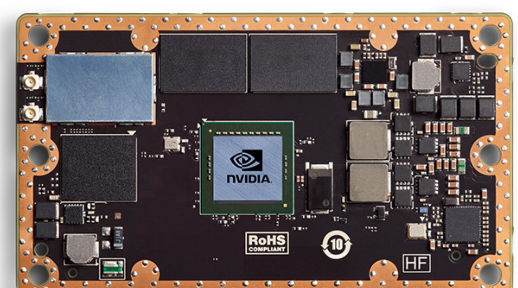
- Unmanned Aerial Vehicles (UAVs) are **systems-of-systems** that couple their **cyber** and **physical** components

Increase in lines of code

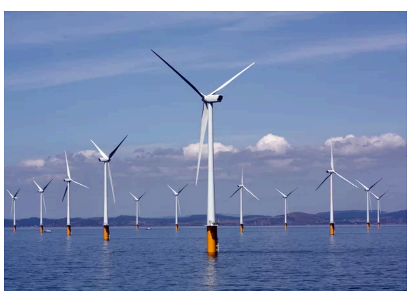


Machine learning

mass production



[multi-core processors
with limited amount of energy]



safety-critical applications

Security Challenges in UAVs

- Security vulnerabilities can lead to **drastic consequences**

Boeing Unmanned Little Bird H-6U

Attacked by **rogue camera software** and by a **virus** delivered through a compromised USB stick

- Security raises **additional challenges**
  - Vulnerability analysis (software connected with hardware)
  - Remote accessibility (device authentication, access control)
  - Patch management (vendors might be long gone)
  - Attacks from physical world (GPS spoofing and replay attack)

Security Vulnerabilities

```
int getPassword() {
  char buf[4];
  gets(buf);
  return strcmp(buf, ”SMT”);
}
```

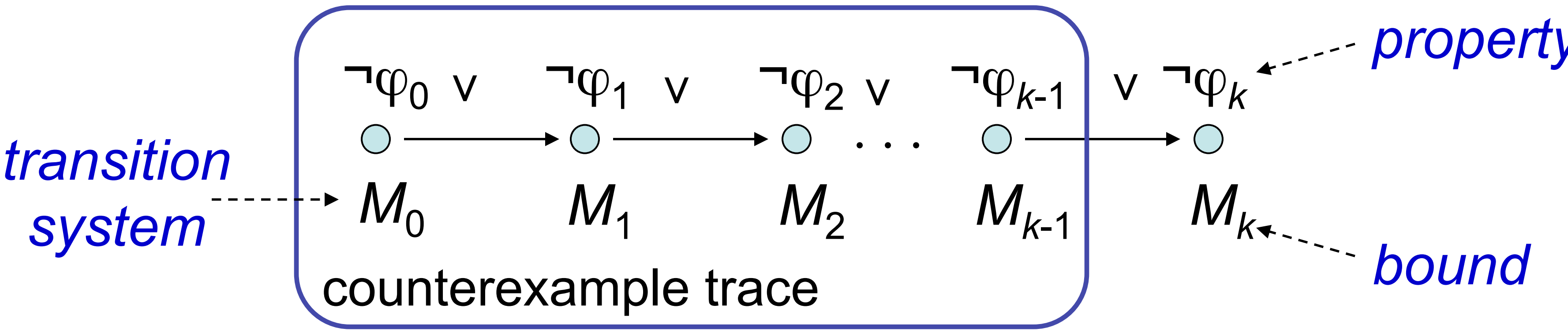
```
void main(){
  int x=getPassword();
  if(x){
    printf(“Access Denied\n”);
    exit(0);
  }
  printf(“Access Granted\n”);
}
```

- What happens if the user enters “SMT”?
- On a Linux x64 platform running GCC 4.8.2, an input consisting of 24 arbitrary characters followed by *]*, <*ctrl-f*>, and @, will bypass the “Access Denied” message
- A longer input will run over into other parts of the **computer memory**

Barrett et al., Problem Solving for the 21st Century, 2014.

Bounded Model Checking (BMC)

Basic idea: check negation of given property up to given depth



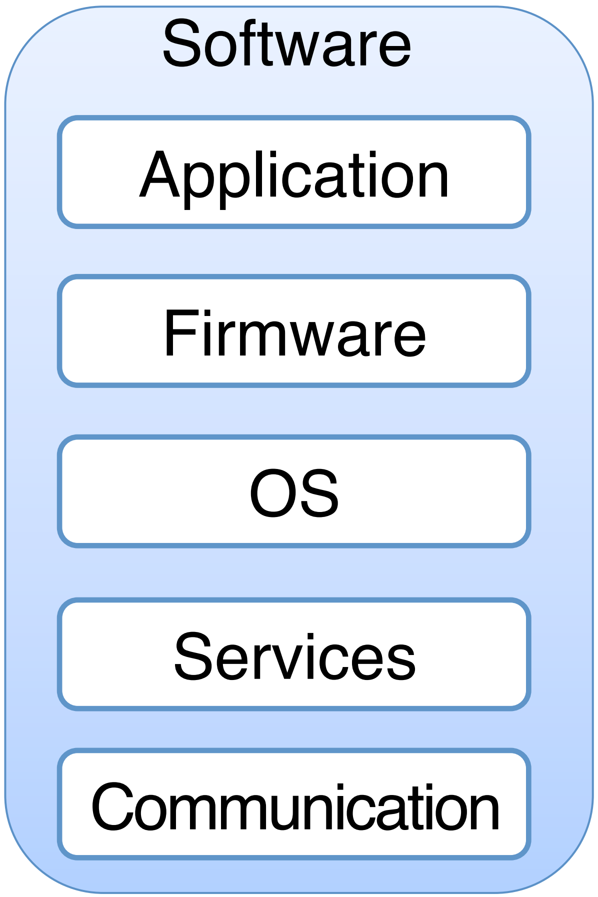
- Transition system *M* unrolled *k* times
  - for programs: loops, recursion, …
- Translated into verification condition ψ such that

ψ **satisfiable iff** φ **has counterexample of max. depth *k***

BMC has been applied successfully to
verify HW and SW

Ensure Software Security in UAVs

- BMC techniques can be used to ensure **software security**



Requirements	Definition
Availability	services are accessible if requested by authorized users
Integrity	data completeness and accuracy are preserved
Confidentiality	only authorized users can get access to the data

Critical Software Vulnerabilities

- Null pointer dereference

```
int main() {
  double *p = NULL;
  int n = 8;
 for(int i = 0; i < n; ++i )
    *(p+i) = i*2;
  return 0;
}
```

A NULL pointer dereference occurs when the application dereferences a pointer that it expects to be valid, but is NULL

Scope	Impact
Availability	Crash, exit and restart
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands

Critical Software Vulnerabilities

- Null pointer dereference
- Double free

```
int main(){
  char* ptr = (char *)malloc(sizeof(char));
  if(ptr==NULL) return -1;
  *ptr = 'a';
  free(ptr);
  free(ptr);
  return 0;
}
```

The product calls *free()* twice on the same memory address, leading to modification of unexpected memory locations

Scope	Impact
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands

Critical Software Vulnerabilities

- Null pointer dereference
- Double free
- Unchecked Return Value to NULL Pointer Dereference

```
String username = getUserName();
if (username.equals(ADMIN_USER)) {
...
}
```

The product does not check for an error after calling a function that can return with a NULL pointer if the function fails

Scope	Impact
Availability	Crash, exit and restart

Critical Software Vulnerabilities

- Null pointer dereference
- Double free
- Unchecked Return Value to NULL Pointer Dereference
- Division by zero
- Missing free
- Use after free
- APIs rule based checking

Satisfiability Modulo Theories

SMT decides the **satisfiability** of first-order logic formulae using the combination of different **background theories**

Theory	**Example**
Equality	$x_1 = x_2 \land \neg (x_1 = x_3) \Rightarrow \neg (x_1 = x_3)$
Bit-vectors	(b >> i) & 1 = 1
Linear arithmetic	$(4y_1 + 3y_2 \geq 4) \lor (y_2 - 3y_3 \leq 3)$
Arrays	$(j = k \land a[k]=2) \Rightarrow a[j]=2$
Combined theories	$(j \leq k \land a[j]=2) \Rightarrow a[i] < 3$

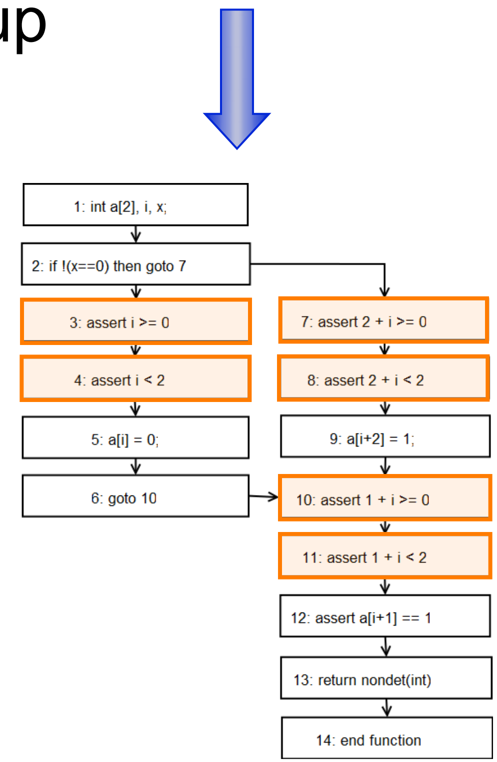
Software BMC

```
int getPassword() {
  char buf[4];
  gets(buf);
  return strcmp(buf, "ML");
}
```

```
void main(){
  int x=getPassword();
  if(x){
    printf("Access Denied\n");
    exit(0);
  }
  printf("Access Granted\n");
}
```

- program modelled as transition system
  - *state*: *pc* and program variables
  - derived from control-flow graph
  - added safety properties as extra nodes
- program unfolded up to given bounds
- unfolded program optimized to reduce blow-up
  - constant propagation
  - forward substitutions

crucial



Software BMC

- program modelled as transition system
  - *state*: *pc* and program variables
  - derived from control-flow graph
  - added safety properties as extra nodes
- program unfolded up to given bounds
- unfolded program optimized to reduce blow-up
  - constant propagation
  - forward substitutions

  crucial
- front-end converts unrolled and optimized program into SSA

```
int getPassword() {
  char buf[4];
  gets(buf);
  return strcmp(buf, "ML");
}
```

```
void main(){
  int x=getPassword();
  if(x){
    printf("Access Denied\n");
    exit(0);
  }
  printf("Access Granted\n");
}
```

$g_1 = x_1 == 0$
$a_1 = a_0 \text{ WITH } [i_0{:=}0]$
$a_2 = a_0$
$a_3 = a_2 \text{ WITH } [2+i_0{:=}1]$
$a_4 = g_1 \; ? \; a_1 : a_3$
$t_1 = a_4[1+i_0] == 1$

Software BMC

```
int getPassword() {
  char buf[4];
  gets(buf);
  return strcmp(buf, "ML");
}
```

```
void main(){
  int x=getPassword();
  if(x){
    printf("Access Denied\n");
    exit(0);
  }
  printf("Access Granted\n");
}
```

- program modelled as transition system
  - *state*: *pc* and program variables
  - derived from control-flow graph
  - added safety properties as extra nodes
- program unfolded up to given bounds
- unfolded program optimized to reduce blow-up
  - constant propagation
  - forward substitutions

  crucial
- front-end converts unrolled and optimized program into SSA
- extraction of *constraints C* and *properties P*
  - specific to selected SMT solver, uses theories
- satisfiability check of $C \wedge \neg P$

$$C := \begin{bmatrix} g_1 := (x_1 = 0) \\ \wedge\ a_1 := store(a_0, i_0, 0) \\ \wedge\ a_2 := a_0 \\ \wedge\ a_3 := store(a_2, 2 + i_0, 1) \\ \wedge\ a_4 := ite(g_1, a_1, a_3) \end{bmatrix}$$

$$P := \begin{bmatrix} i_0 \geq 0 \wedge i_0 < 2 \\ \wedge\ 2 + i_0 \geq 0 \wedge 2 + i_0 < 2 \\ \wedge\ 1 + i_0 \geq 0 \wedge 1 + i_0 < 2 \\ \wedge\ select(a_4, i_0 + 1) = 1 \end{bmatrix}$$

Software BMC Applied to Security

```
int getPassword() {
  char buf[4];
  gets(buf);
  return strcmp(buf, "SMT");
}
```

```
void main(){
  int x=getPassword();
  if(x){
    printf("Access Denied\n");
    exit(0);
  }
  printf("Access Granted\n");
}
```

buffer overflow attack

SSA & loop unrolling

```
sp0,sp1,sp2:BITVECTOR(8);
ip:BITVECTOR(8);
m0,m1,m2,m3,m4,m5 : ARRAY BITVECTOR(8) OF BITVECTOR(8);
in : ARRAY INT OF BITVECTOR(8);
ASSERT sp1 = BVSUB(8,sp0,0bin100);
ASSERT m1 = m0 WITH [sp1] := in[1];
ASSERT m2 = m1 WITH [BVPLUS(8,sp1,0bin1)] := in[2];
ASSERT m3 = m2 WITH [BVPLUS(8,sp1,0bin10)] := in[3];
ASSERT m4 = m3 WITH [BVPLUS(8,sp1,0bin11)] := in[4];
ASSERT m5 = m4 WITH [BVPLUS(8,sp1,0bin100)] := in[5];
ASSERT sp2 = BVPLUS(8,sp1,0bin100);
ASSERT ip = m5[sp2];
ASSERT NOT ip = m0[sp0];
CHECKSAT;
```

4-character array *buf*

reclaim the memory occupied by *buf*

ip is loaded with the location pointed to by sp

We wish to determine whether it is possible to set *ip* to a value that we choose instead of the location of the if statement

Verifying Multi-threaded Programs

Idea: iteratively generate all possible interleavings and call the BMC procedure on each interleaving

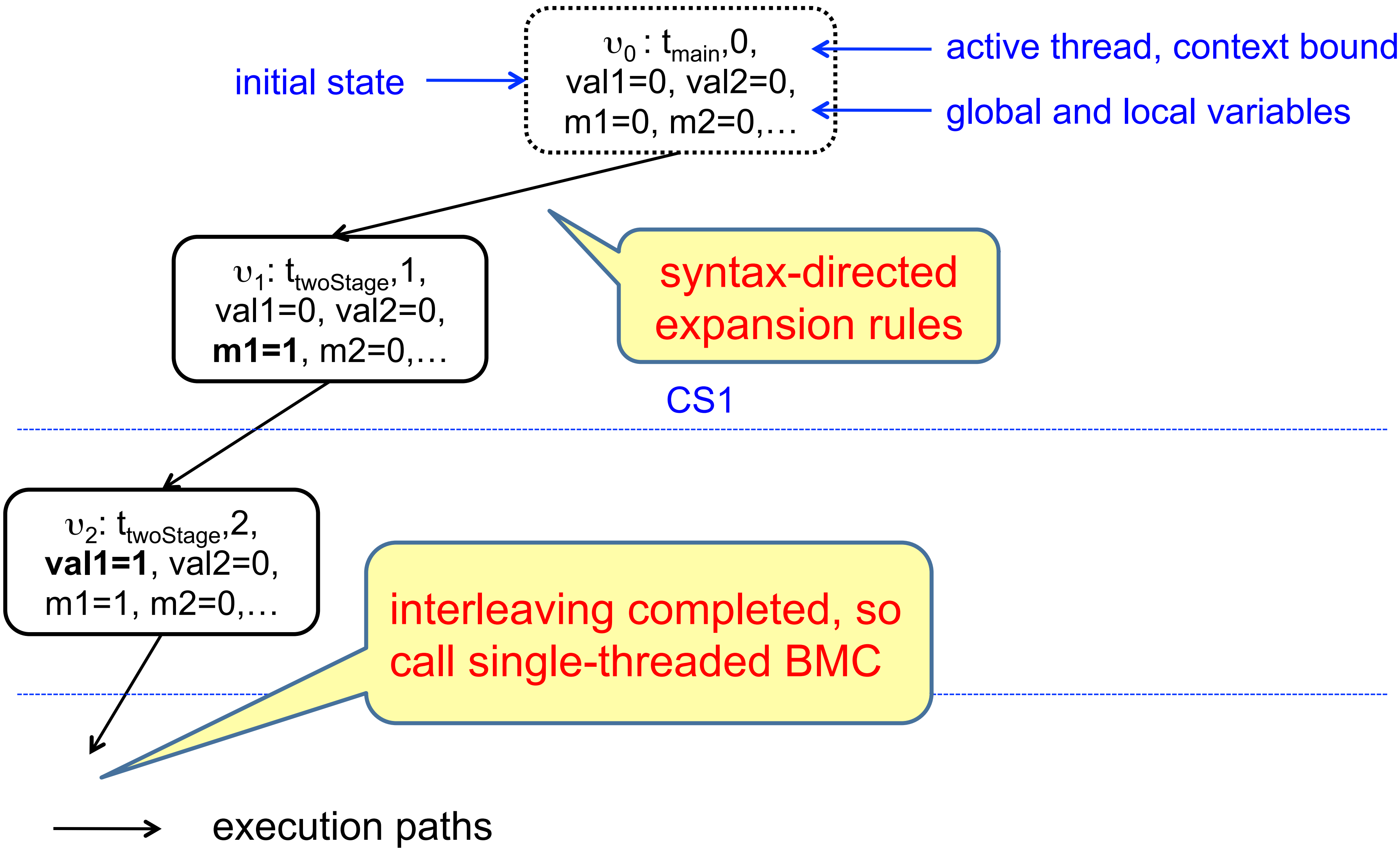
- **symbolic** model checking: on each individual interleaving
- **explicit state** model checking: explore all interleavings

```
void *threadA(void *arg) {
  lock(&mutex);
  x++;
  if (x == 1) lock(&lock);
  unlock(&mutex); (CS1)
  lock(&mutex);   (CS3)
  x--;
  if (x == 0) unlock(&lock);
  unlock(&mutex);
}
```

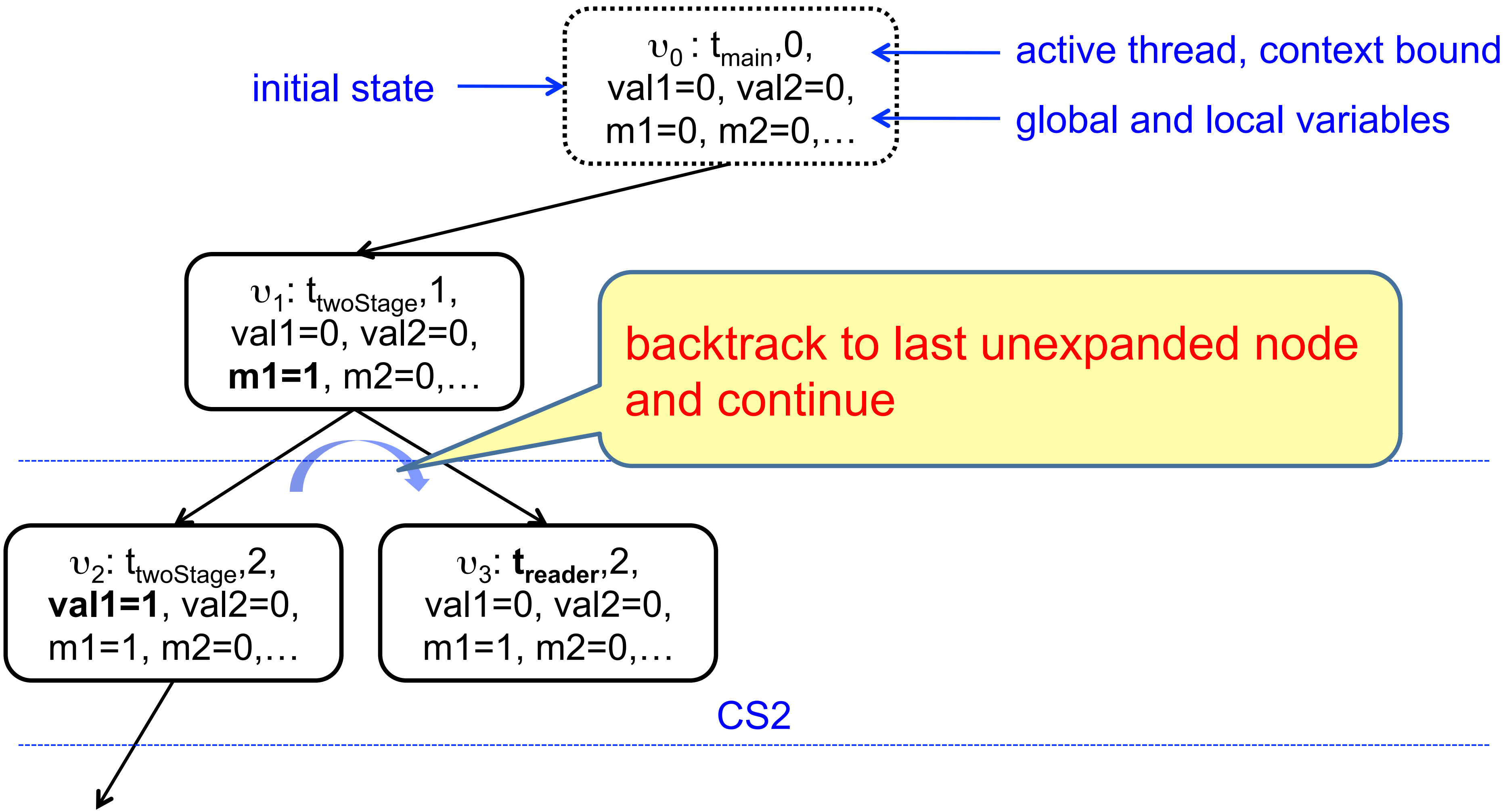
Deadlock

```
void *threadB(void *arg) {
  lock(&mutex);
  y++;
     y == 1) lock(&lock); (CS2)
     ock(&mutex);
     ck(&mutex);
  y- ;
  if (y == 0) unlock(&lock);
  unlock(&mutex);
}
```

Lazy exploration of the Reachability Tree



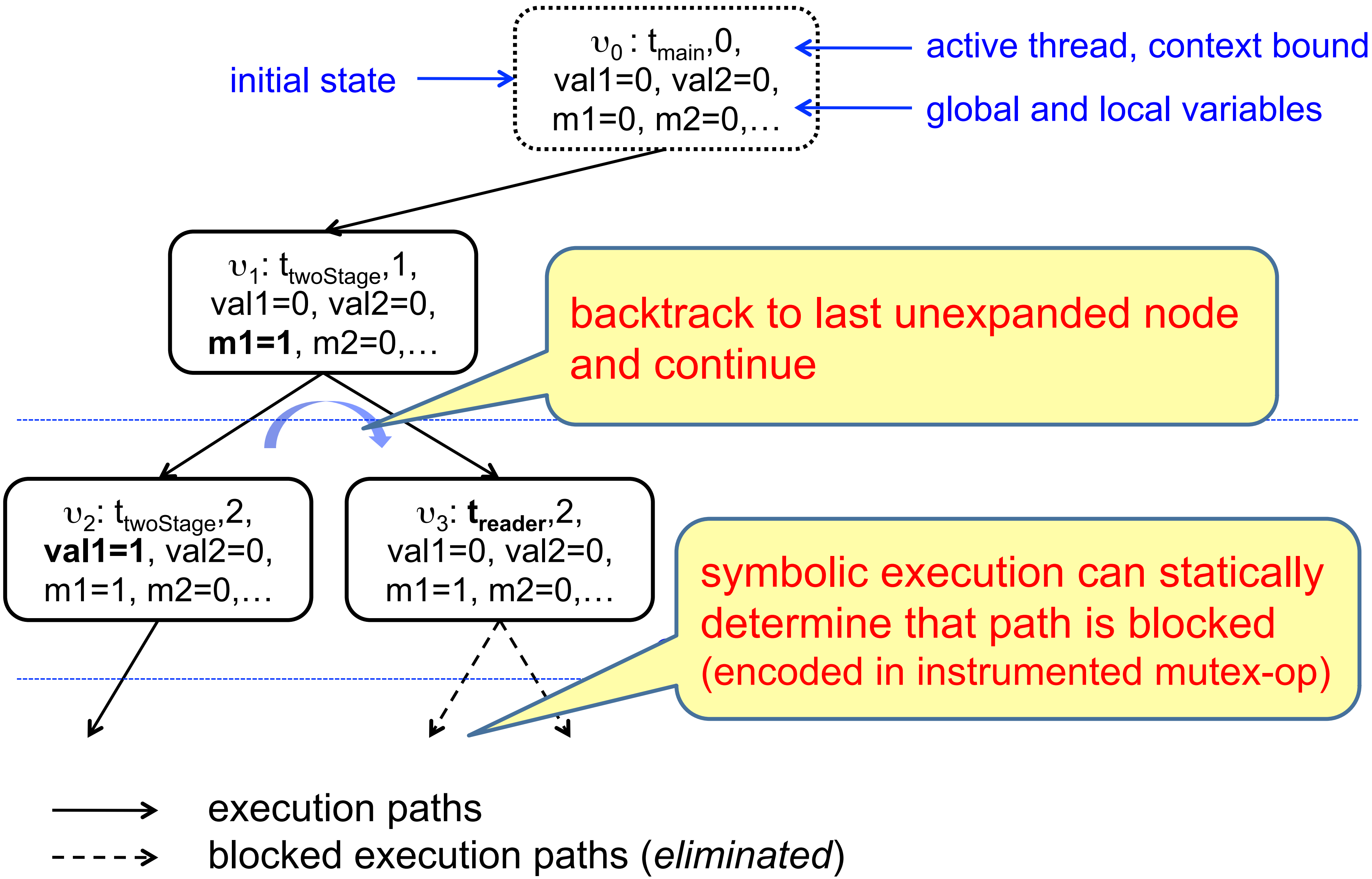
Lazy exploration of the Reachability Tree



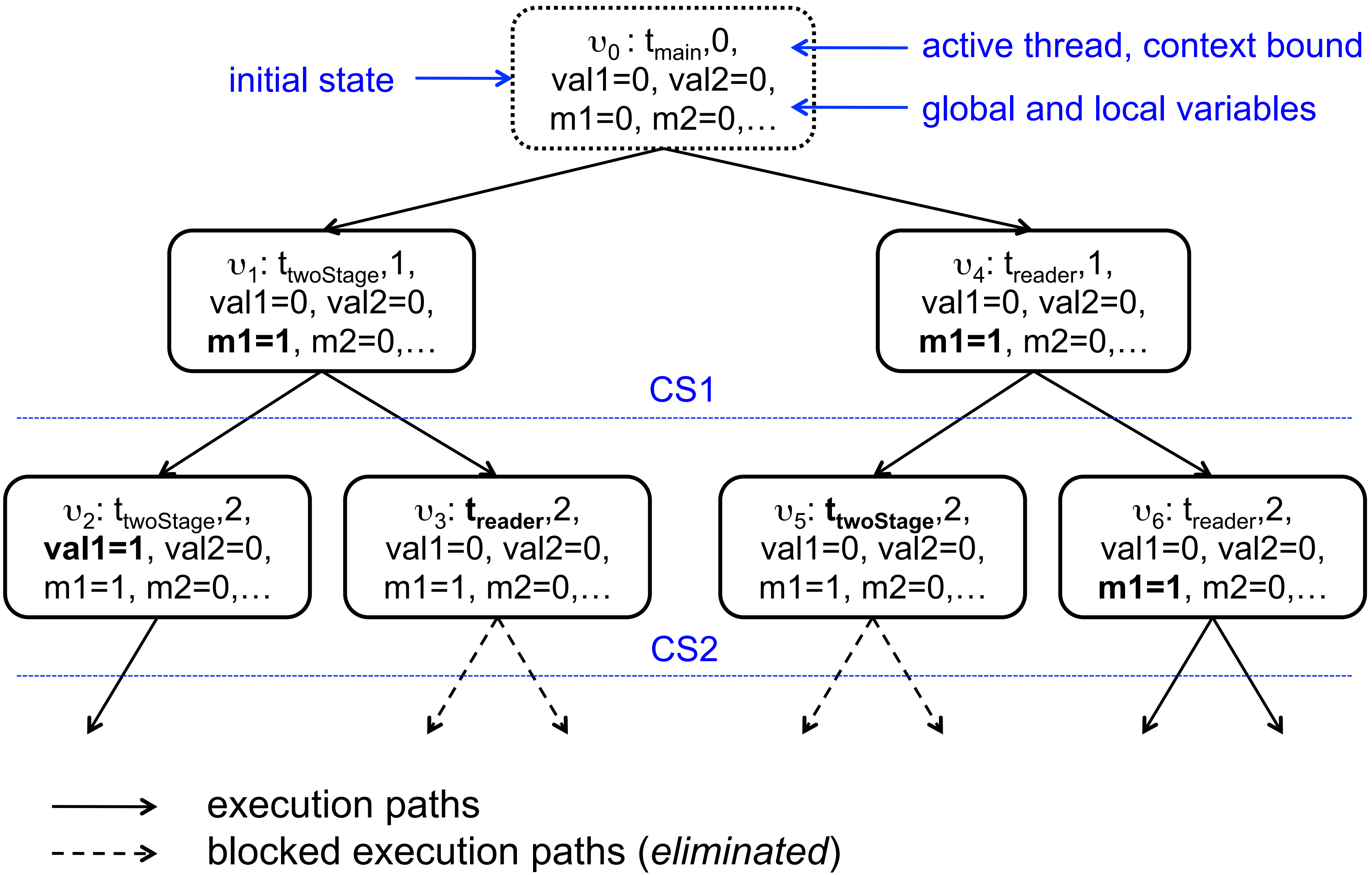
→ execution paths

- - -→ blocked execution paths (*eliminated*)

Lazy exploration of the Reachability Tree

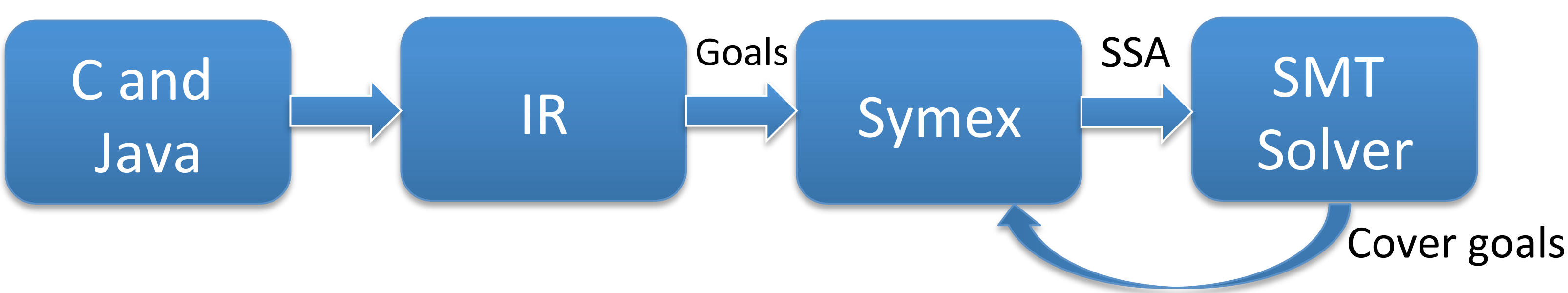


Lazy exploration of the Reachability Tree



BMC / SE for Coverage Test Generation

- Translate the program to an **intermediate representation** (IR)
- Add goals indicating the **coverage**
  - *location, branch, decision, condition and path*
- **Symbolically** execute IR to produce an SSA program
- Translate the resulting SSA program into a **logical formula**
- Solve the formula iteratively to cover different goals
- Interpret the solution to figure out the **input conditions**
- Spit those input conditions out as a test case



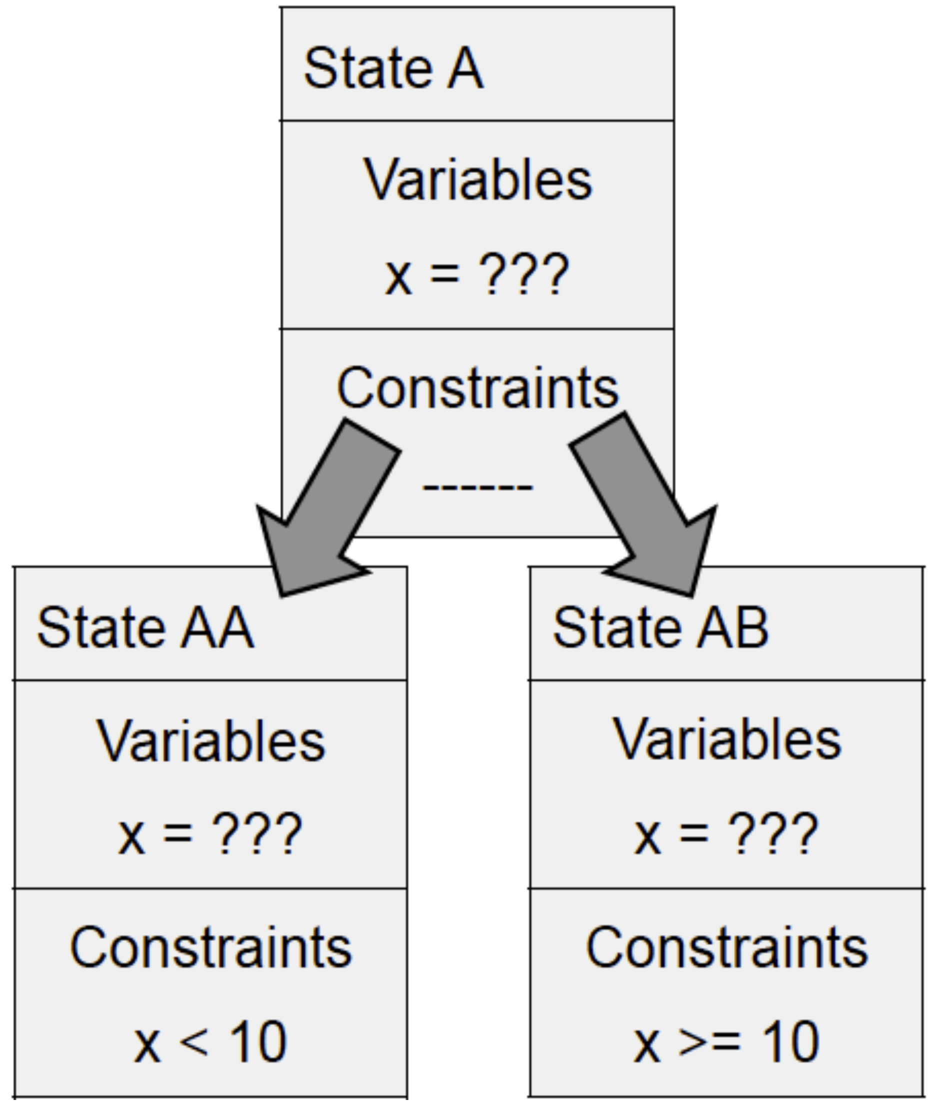
Coverage Test Generation for Security

```
x = input();
if (x >= 10)
{
  if (x < 100)
    vulnerable_code();
  else
    func_a();
}
else
  func_b();
```

Kruegel, C. Finding Vulnerabilities in Embedded Software, ISSTA 2017.

Coverage Test Generation for Security

```
x = input();
if (x >= 10)
{
  if (x < 100)
    vulnerable_code();
  else
    func_a();
}
else
  func_b();
```



Coverage Test Generation for Security

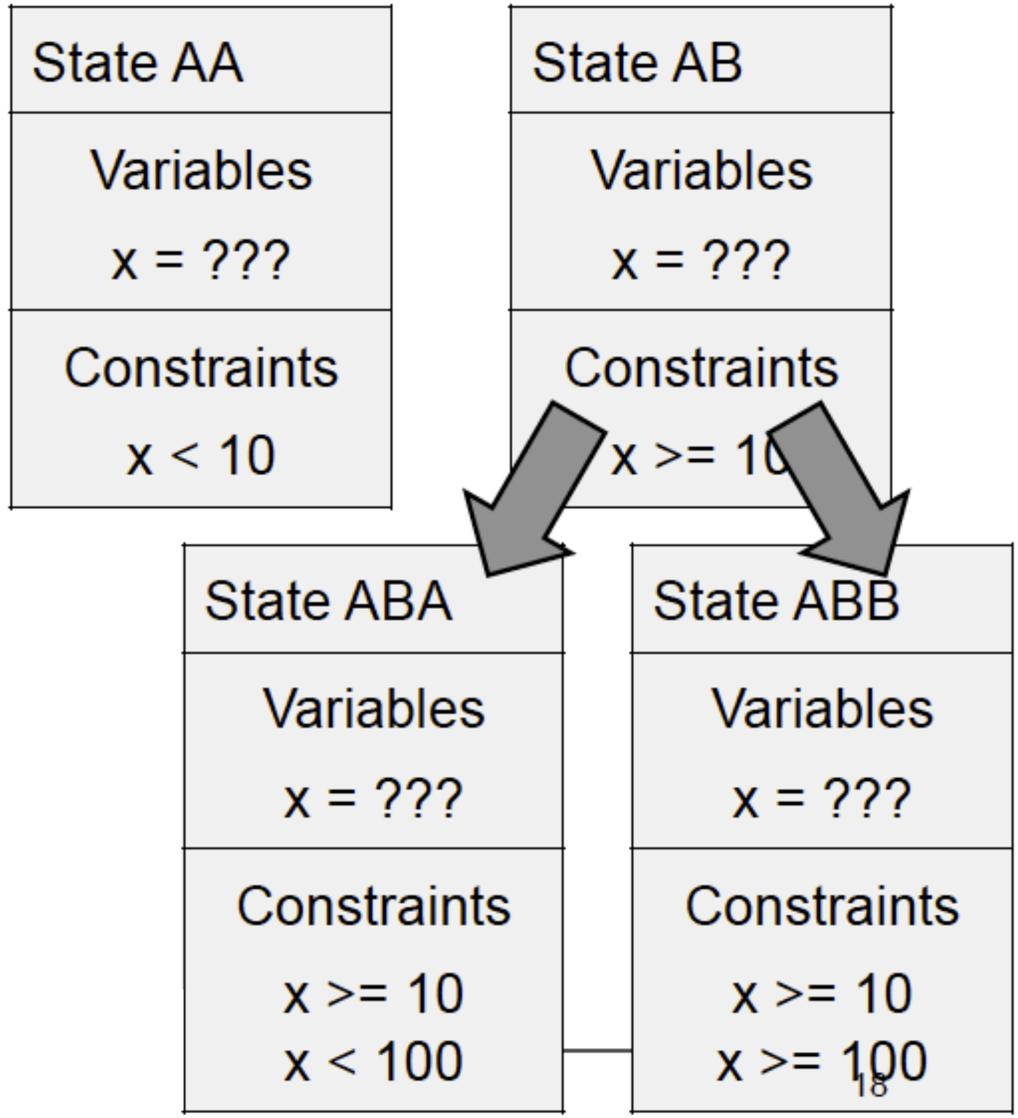
```
x = input();
if (x >= 10)
{
  if (x < 100)
    vulnerable_code();
  else
    func_a();
}
else
  func_b();
```

State AA
Variables x = ???
Constraints x < 10

State AB
Variables x = ???
Constraints x >= 10

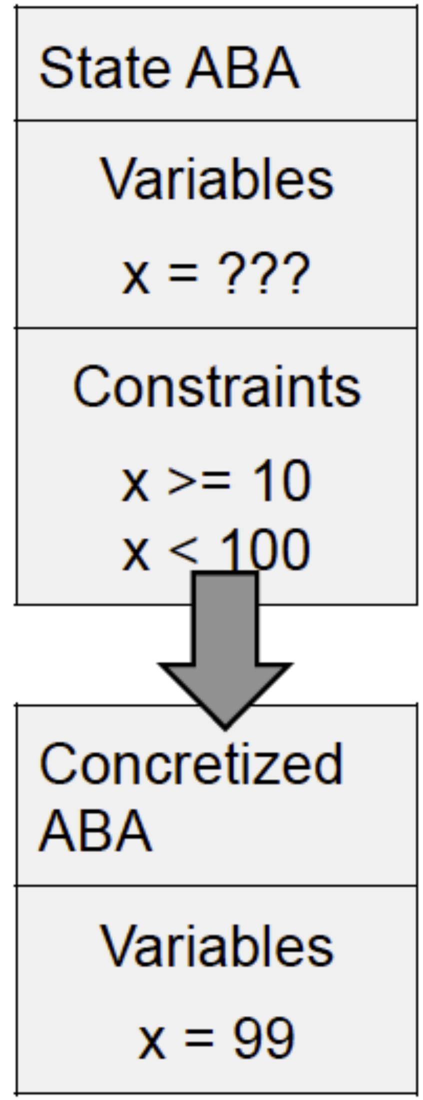
Coverage Test Generation for Security

```
x = input();
if (x >= 10)
{
  if (x < 100)
    vulnerable_code();
  else
    func_a();
}
else
  func_b();
```



Coverage Test Generation for Security

```
x = input();
if (x >= 10)
{
  if (x < 100)
    vulnerable_code();
  else
    func_a();
}
else
  func_b();
```



BMC / SE for Coverage Test Generation

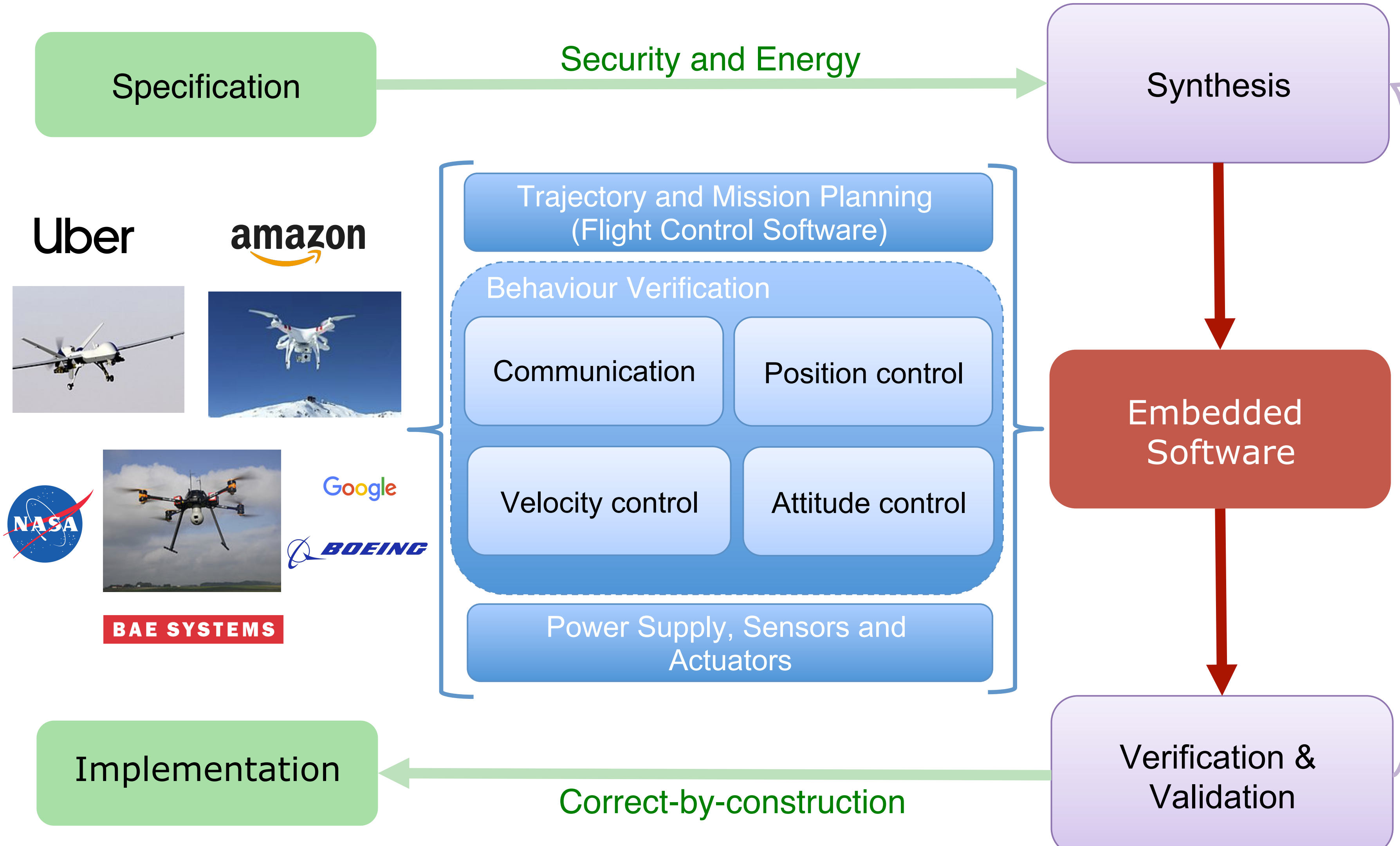
- Pros:
  - Precise
  - no false positive (with correct environment model)
  - produces directly-actionable inputs
- Cons:
  - Not easily scalable
    - ▹ constraint solving is NP-complete
    - ▹ state and path explosion
- Combining Approaches
  - Symbolic Execution, Fuzzing, and Sanitizers

Research Goals in Program Analysis and Cyber-Security

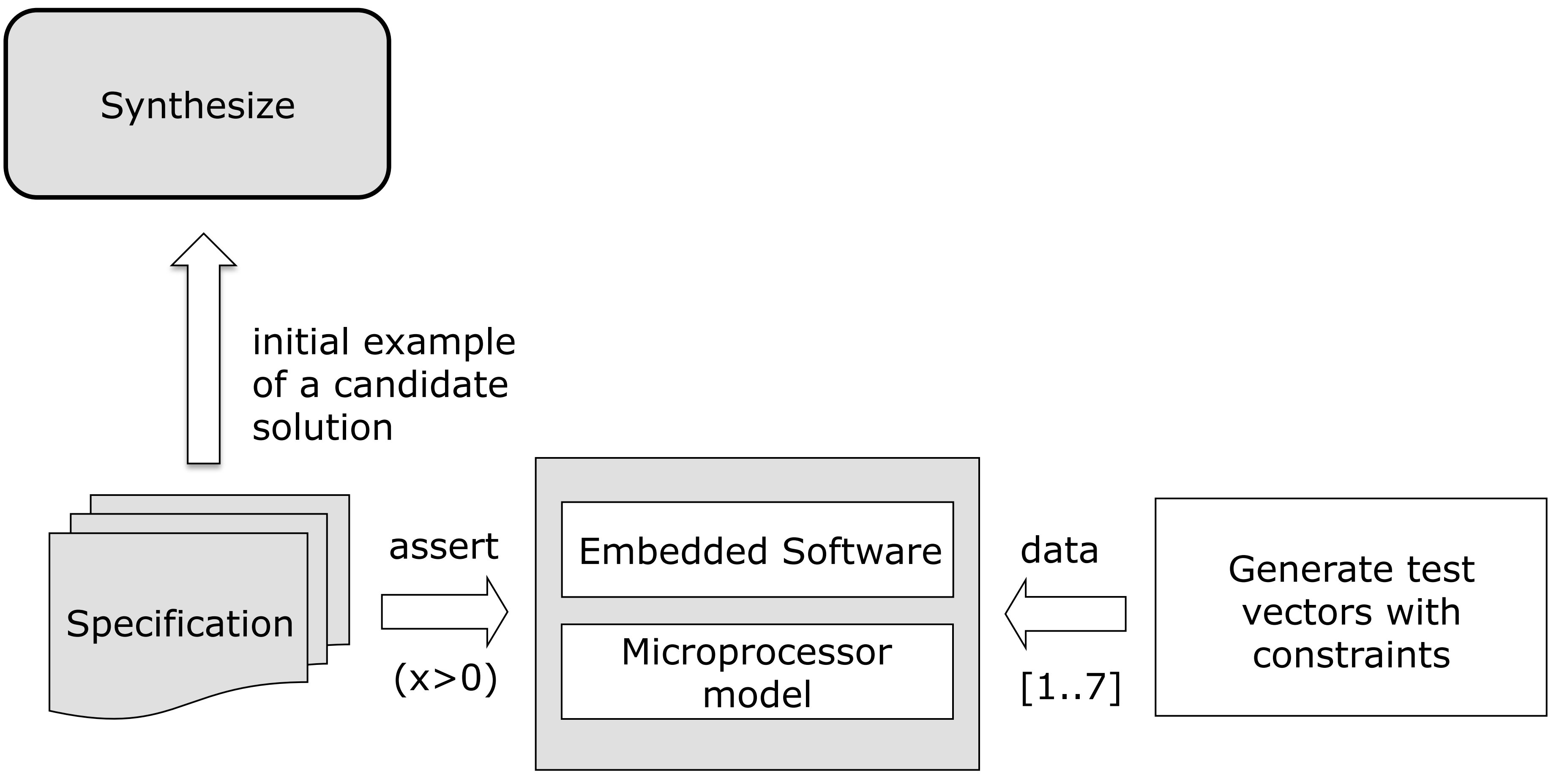
leverage **program analysis/synthesis** to improve **coverage** and reduce **verification time** for finding **vulnerabilities** in software

leverage **program analysis/synthesis** to achieve **correct-by-construction** software systems considering **security** aspects

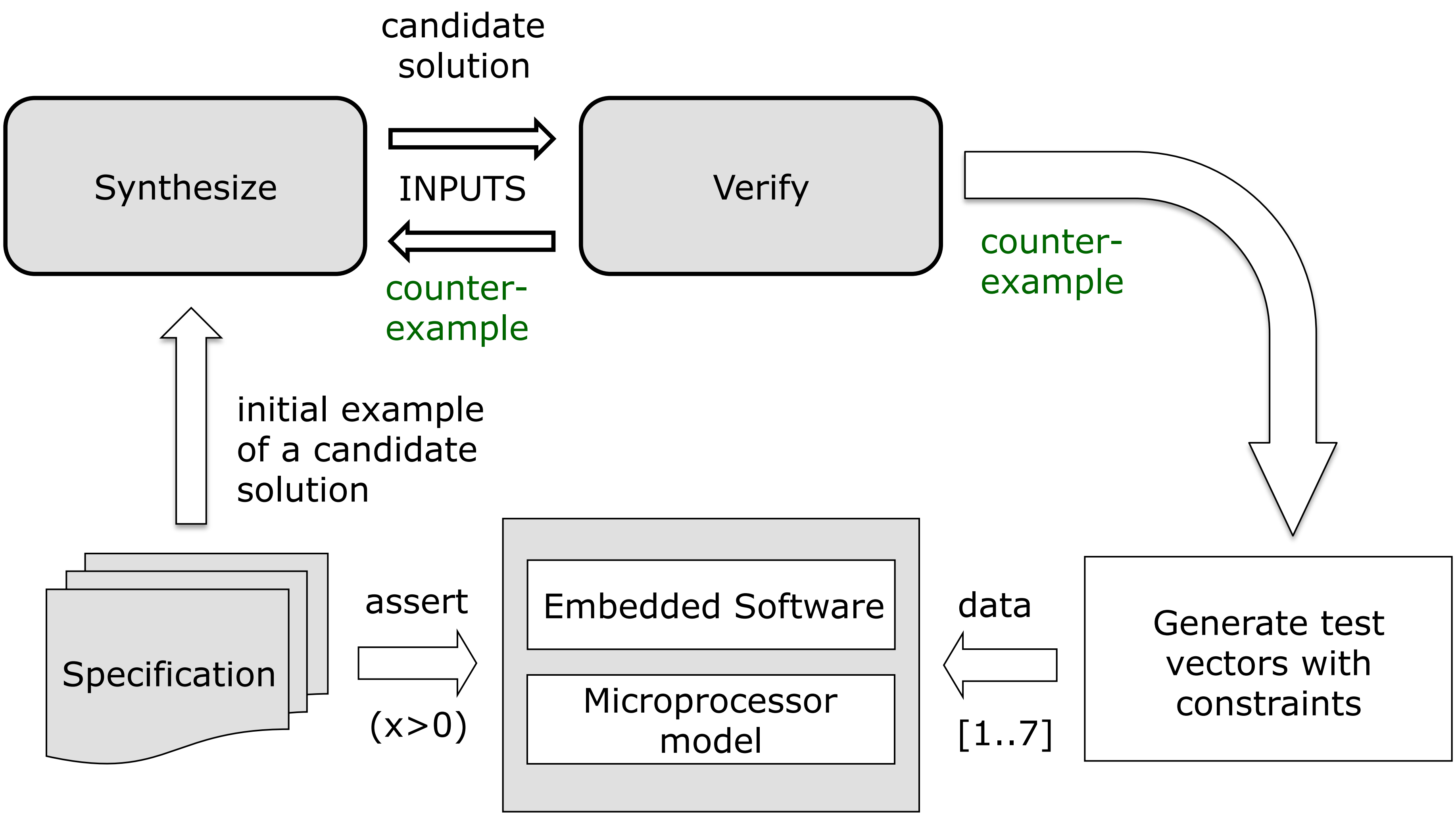
Vision for Future Research



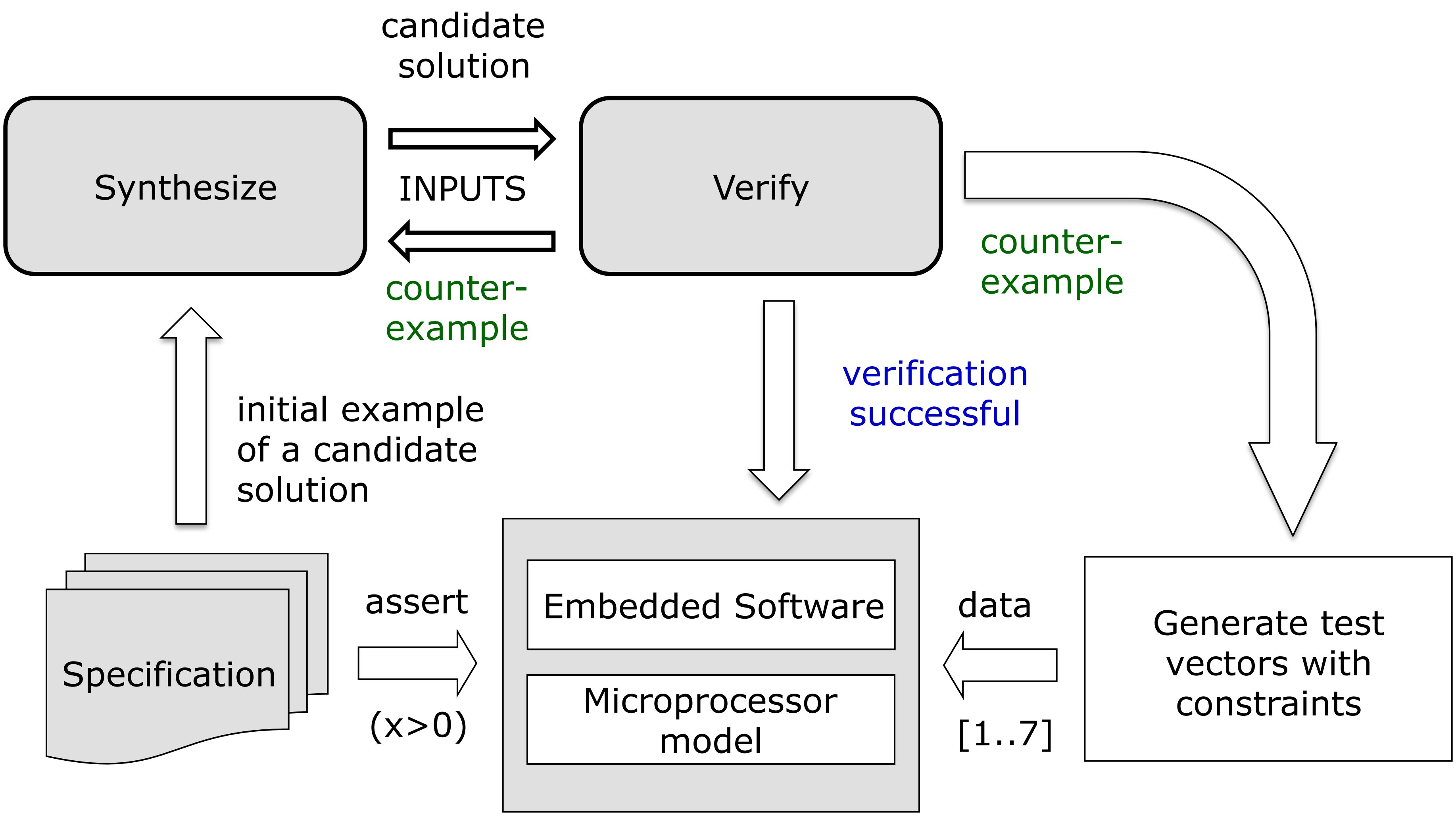
Automated Software Verification and Synthesis for UAVs



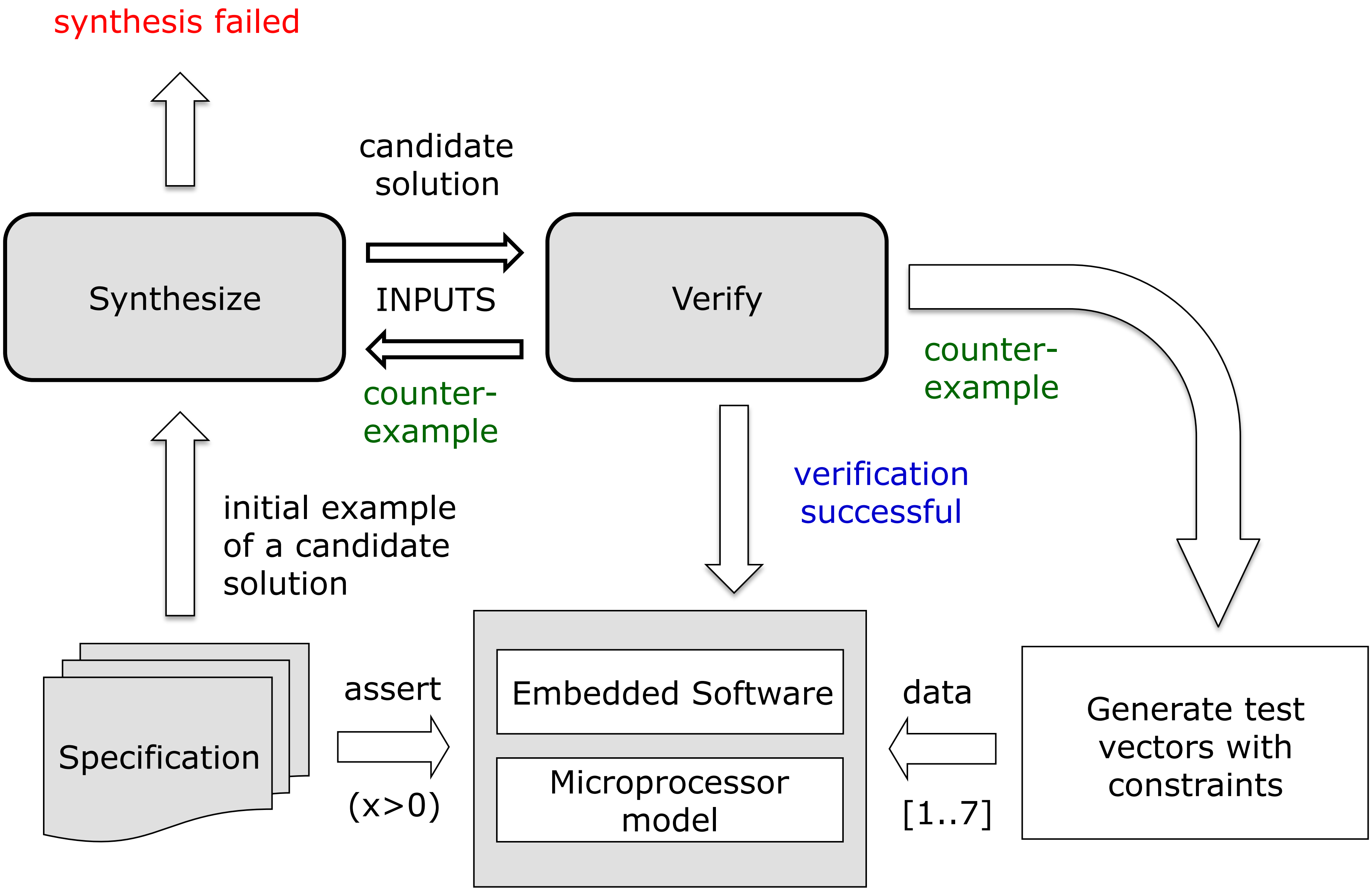
Automated Software Verification and Synthesis for UAVs



Automated Software Verification and Synthesis for UAVs

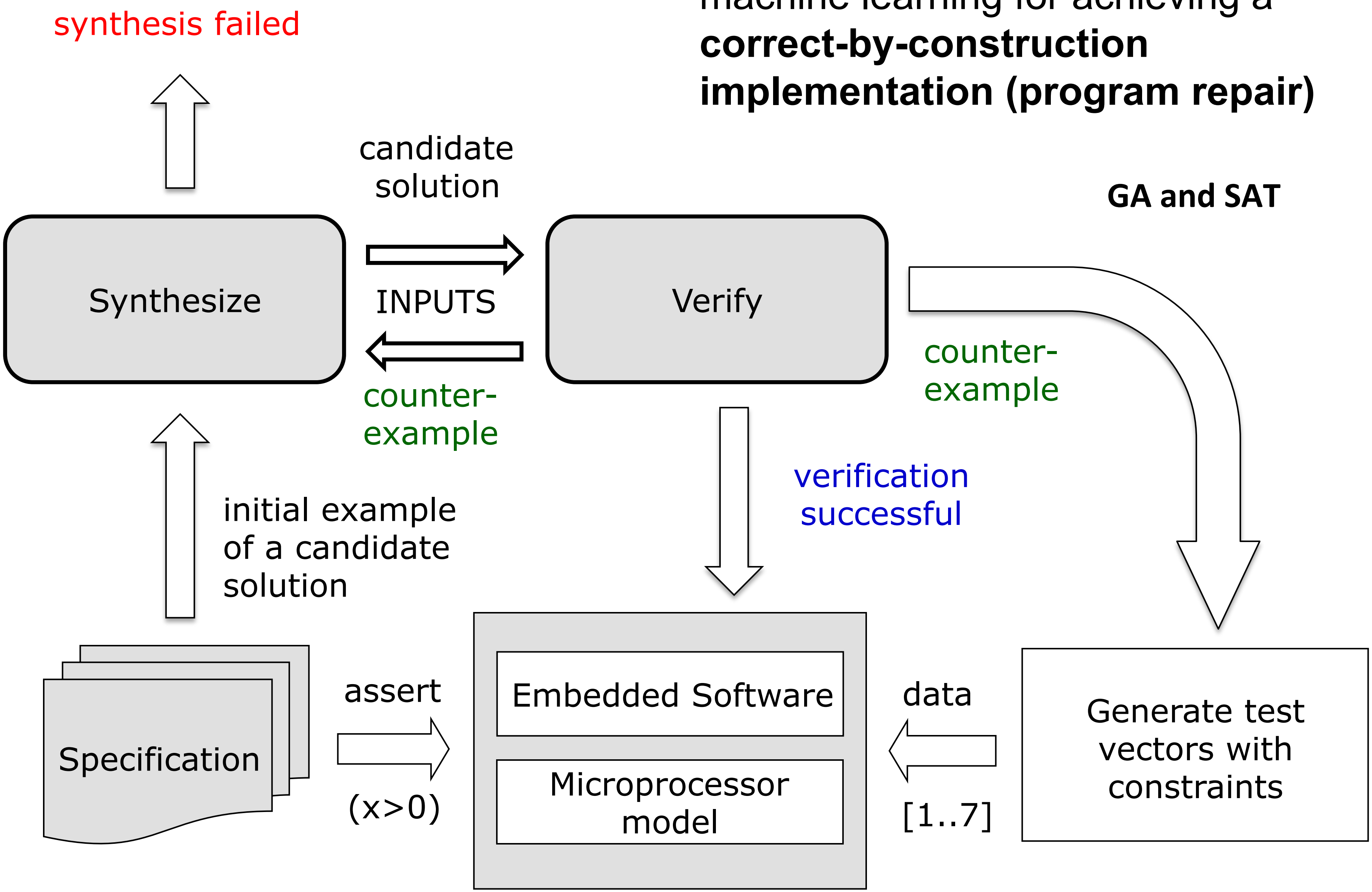


Automated Software Verification and Synthesis for UAVs



Automated Software Verification and Synthesis for UAVs

machine learning for achieving a **correct-by-construction implementation (program repair)**



Synthesizing Control Software in UAVs

- **Counterexample guided induction synthesis** automates the controller design that is **correct-by-construction**

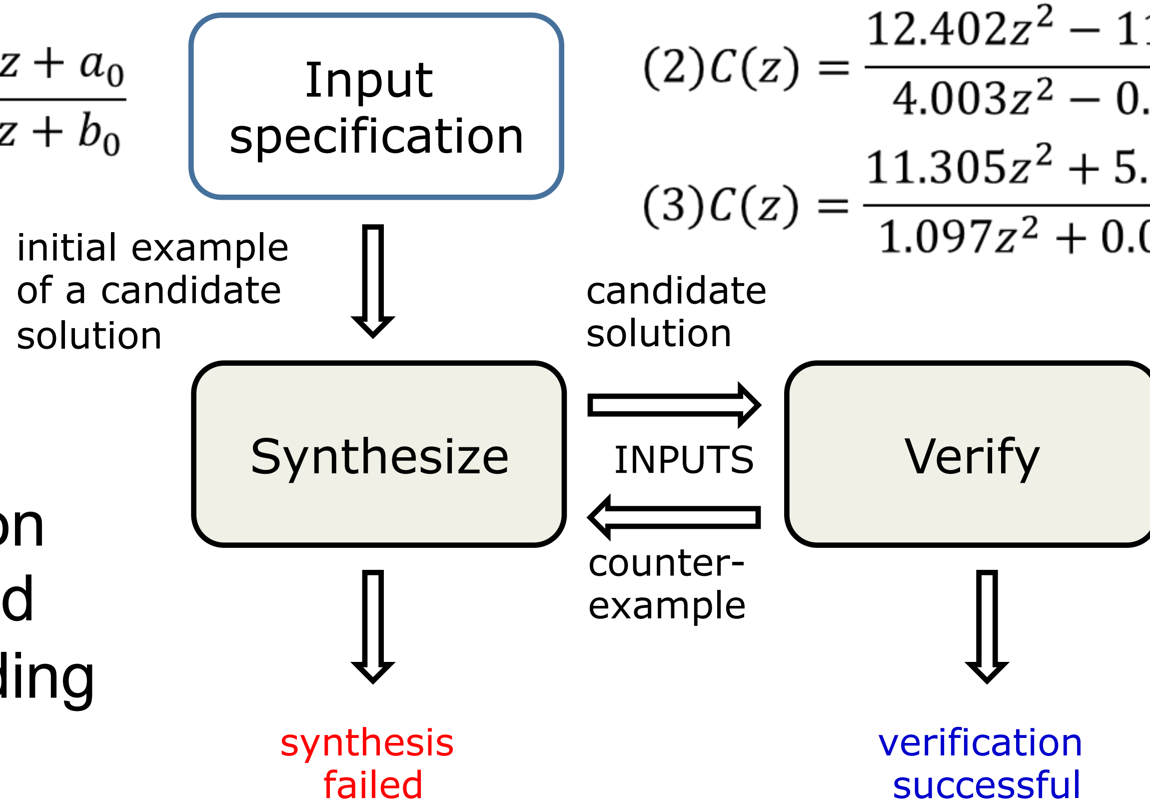
stability, safety, performance specifications

$$C(z) = \frac{a_2 z^2 + a_1 z + a_0}{b_2 z^2 + b_1 z + b_0}$$

$$(1)\ C(z) = \frac{0.026}{z + 1.002}$$

$$(2)\ C(z) = \frac{12.402z^2 - 11.439z + 0.596}{4.003z^2 - 0.287z + 0.015}$$

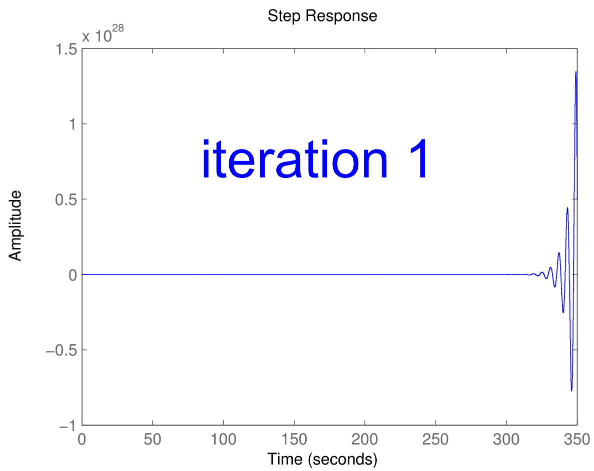
$$(3)\ C(z) = \frac{11.305z^2 + 5.864z + 4.901}{1.097z^2 + 0.063z + 0.128}$$

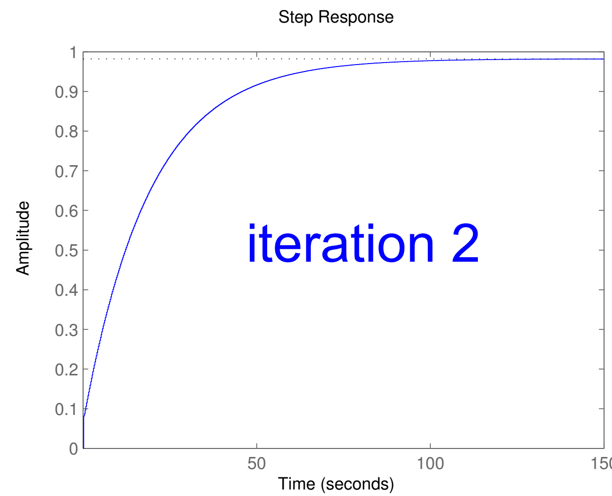


finite-precision arithmetic and related rounding errors

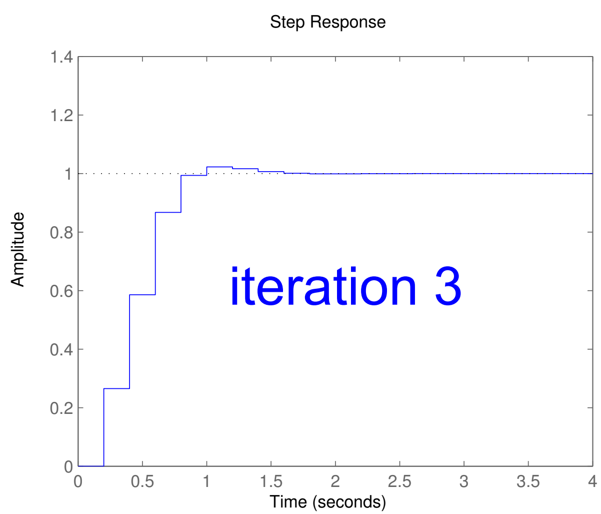
Synthesizing Stable Controllers in UAVs

- Step responses for a **closed-loop control system** with **FWL effects** and for each synthesize iteration



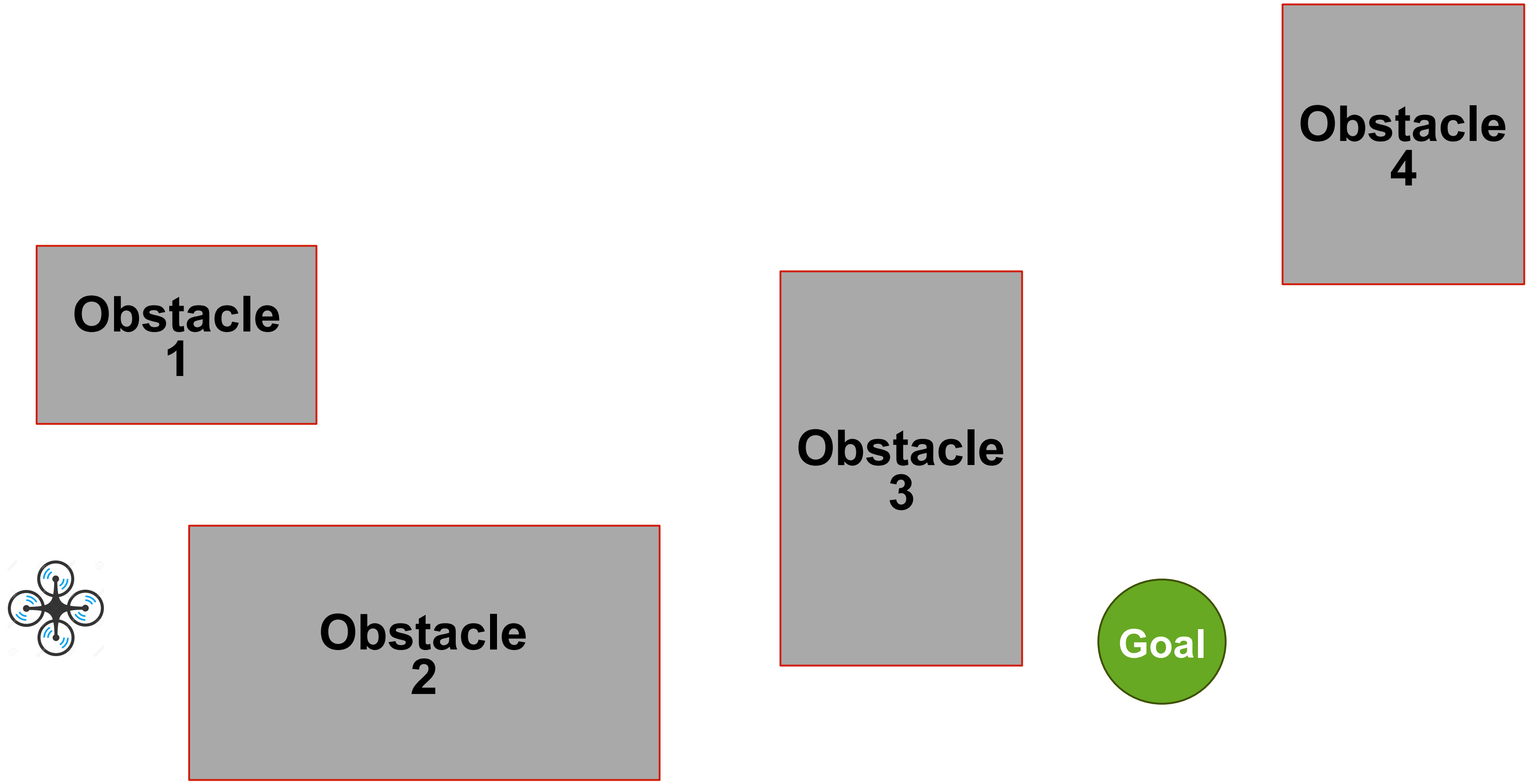


A digital system is **stable** *iff* all of its poles are inside the z-plane unitary circle



Trajectory Planning for UAVs

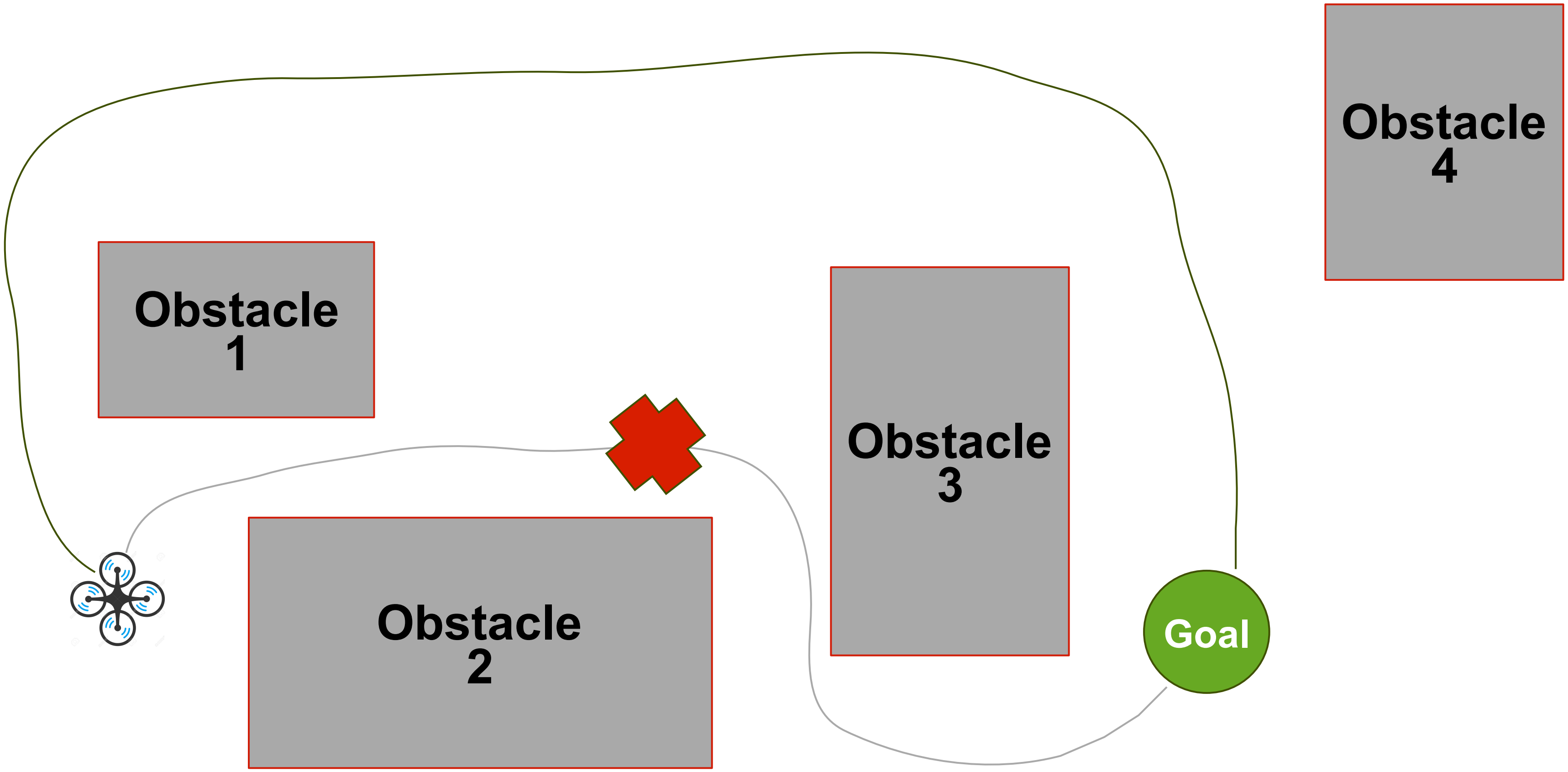
- What is the shortest trajectory for this UAV?



Trajectory Planning for UAVs

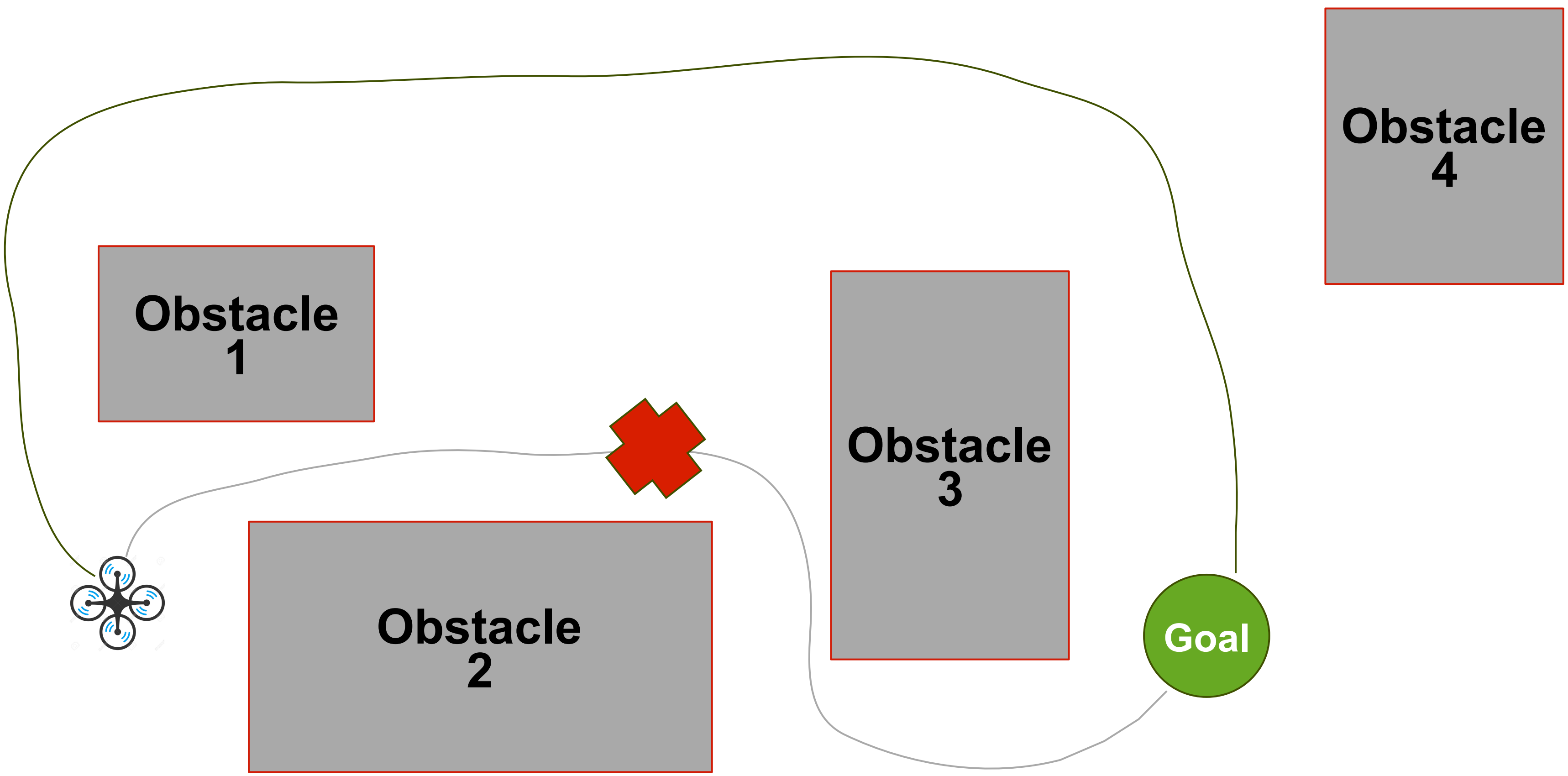
- What is the shortest trajectory for this UAV?

  system's dynamics



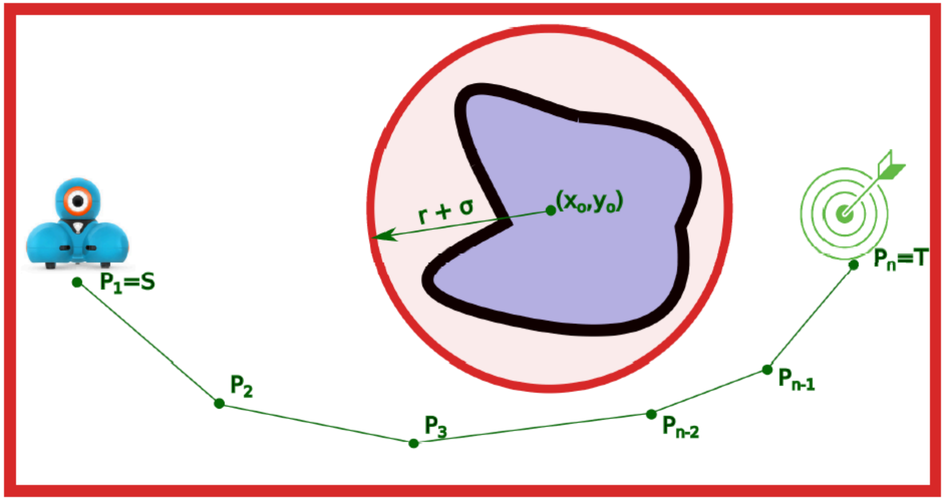
Trajectory Planning for UAVs

- How to find a solution that satisfies the constraints and minimizes the path length?



Path Optimization Problem

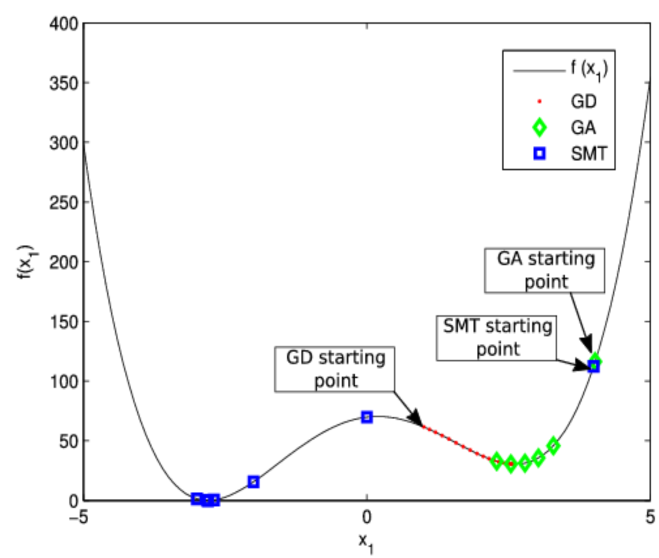
- The search space is delimited by a rectangle
- Obstacles are modeled by circles



$$J(L) = \sum_{i=1}^{n-1} \|P_{i+1} - P_i\|_2$$

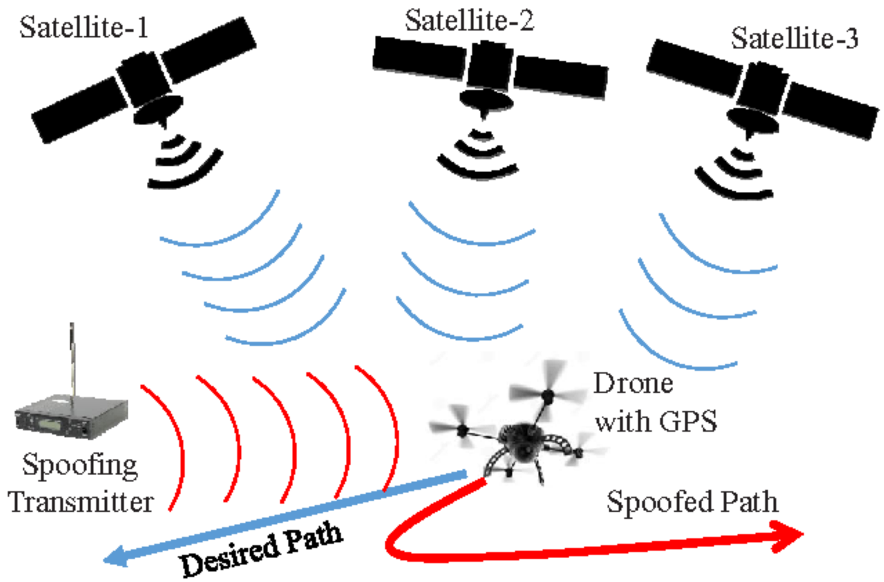
$$\begin{aligned} \min_{L} \quad & J(L), \\ \text{s.t.} \quad & p_{i\lambda}(L) \notin \mathbb{O} \\ & p_{i\lambda}(L) \in \mathbb{E} \\ & i = 1, \ldots, n-1 \end{aligned}$$

no intersection between the path and obstacles



What are the real life attacks to UAVs?

- GPS spoofing



Civilian GPS signals without encrypted signals

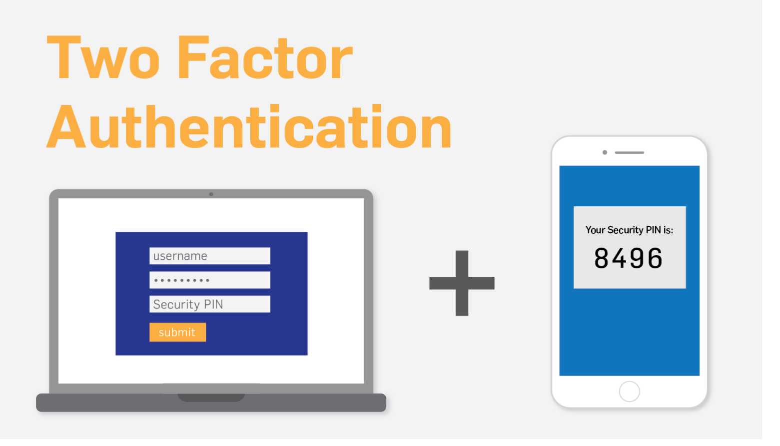
What are the real life attacks to UAVs?

- GPS spoofing
- No encryption

Encryption is extra implementation cost for performance and energy

What are the real life attacks to UAVs?

- GPS spoofing
- No encryption
- No authentication

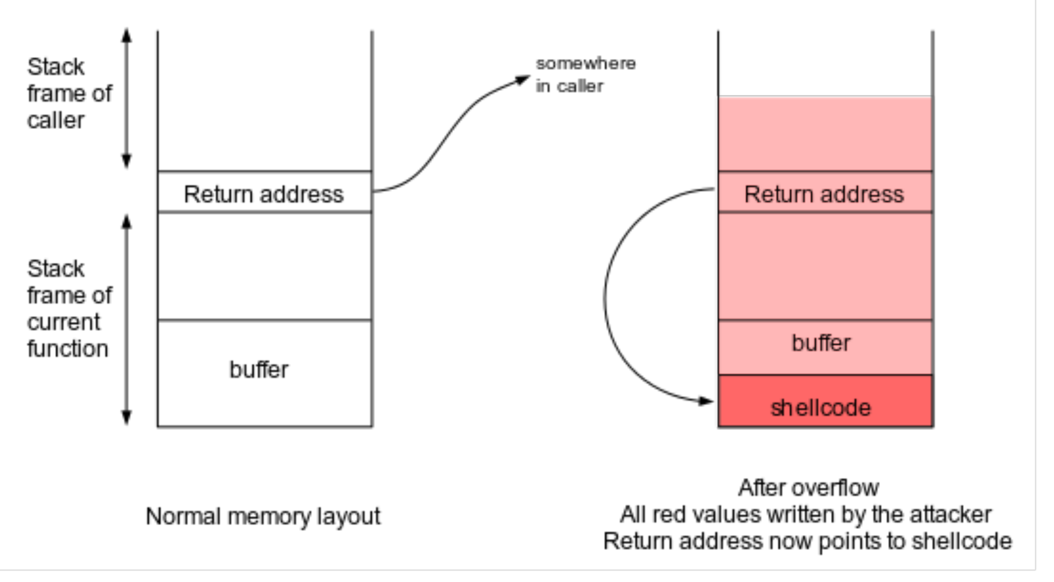


Vulnerability: “Insufficient connection protection”

What are the real life attacks to UAVs?

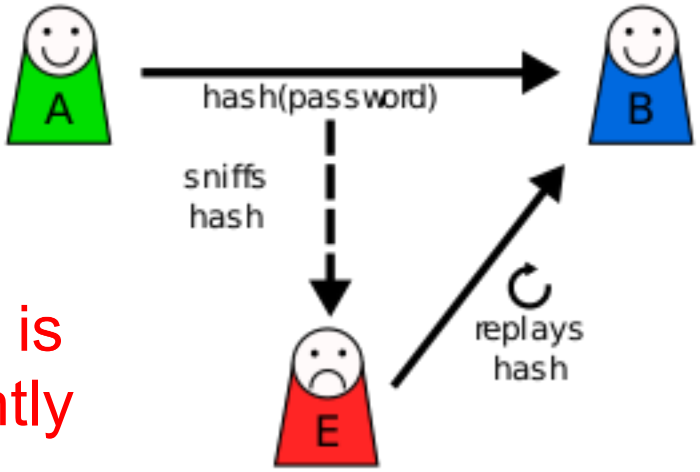
- GPS spoofing
- No encryption
- No authentication
- Large packets causing stack overflow

cause the program to crash or operate incorrectly



What are the real life attacks to UAVs?

- GPS spoofing
- No encryption
- No authentication
- Large packets causing stack overflow
- Replay attack



valid data transmission is maliciously or fraudulently repeated or delayed

What are the real life attacks to UAVs?

- GPS spoofing
- No encryption
- No authentication
- Large packets causing stack overflow
- Replay attack
- Etc

Research Mission

Automated **verification** and **synthesis** to ensure the **software security** in **UAVs**

Methods, algorithms, and tools to write software with respect to security