The Experiences of Developing the Industrial-strength tools for Modeling, Testing and Verification: A Formal-Methods Perspective

Geguang Pu

2024

Outline

- Where are we?
- Tools
- Lessons we got
- The Future

Where are we?

Where are we?

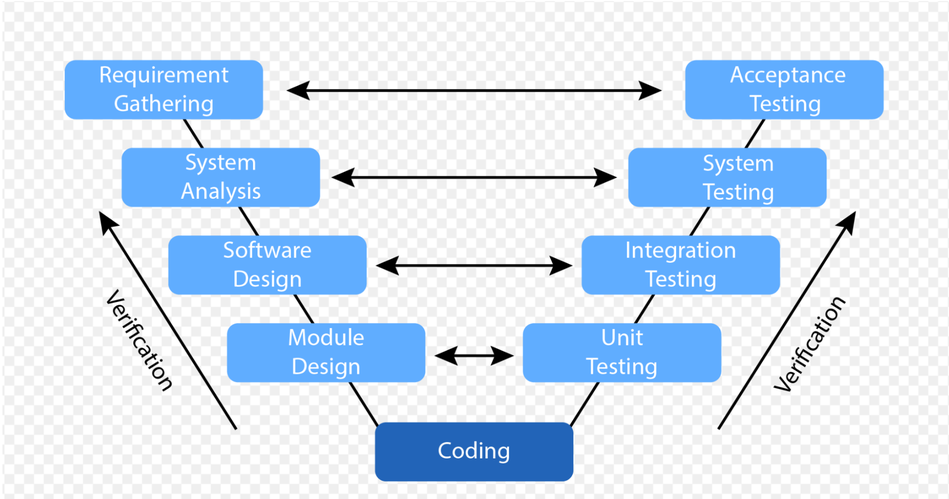
AI is controlling the World

AI is developed and programmed by smart people

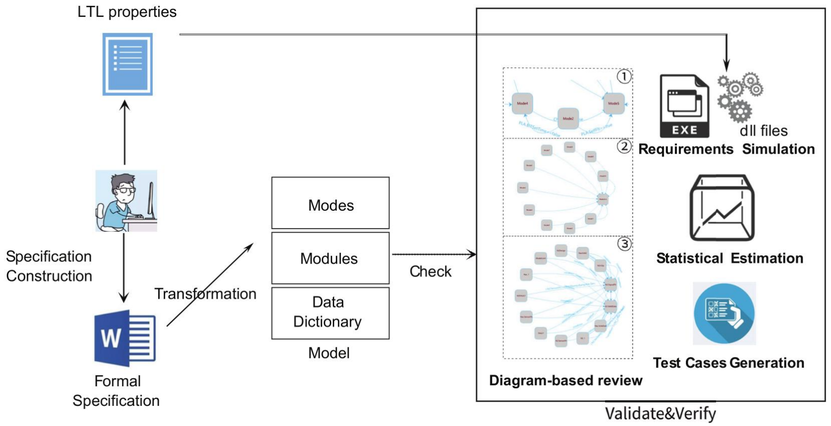
syllogism

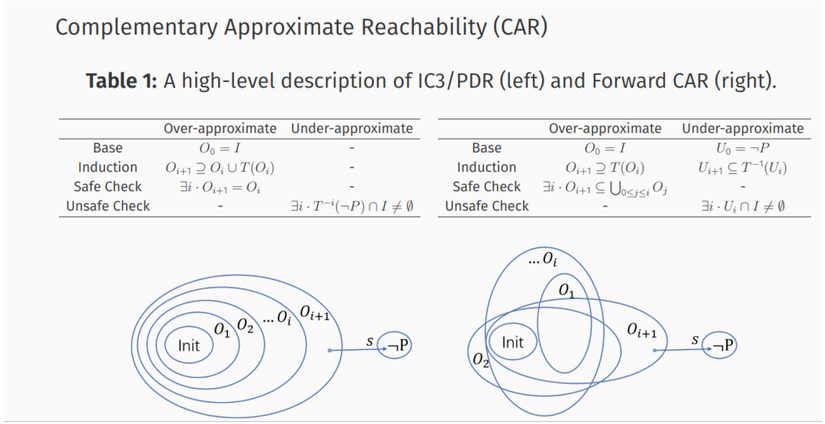
Where are we?

Constructing *reliable* software *efficiently* is our permanent goal in software community.



Where are we?

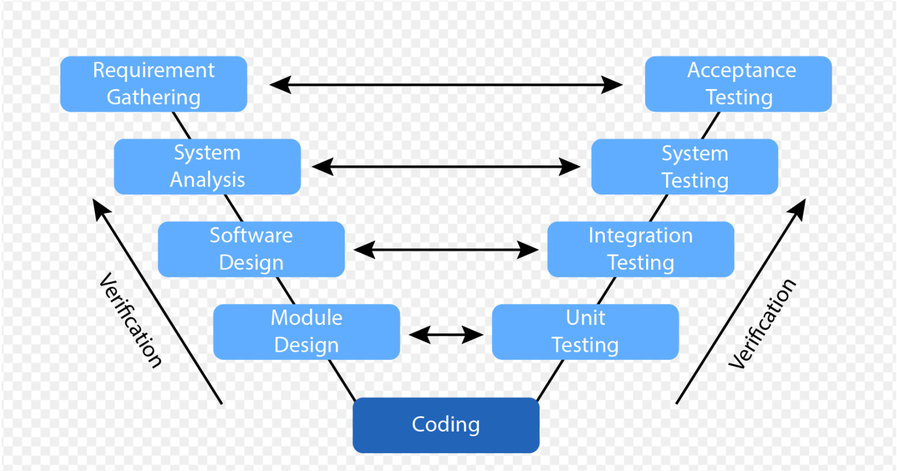




1 Software Requirement Modeling

2 System Behavior Modeling

3 Formal Verification Engine



4 Software Testing

Formal Methods play an important role:
formal semantics, dataflow analysis, symbolic execution, model checking…

Tools

Software Requirement Modeling

Software Modeling is to deal with the complexity of software

The key idea: Not for general modeling, only for domain modeling

Tools

Software Requirement Modeling

The Approach: an automated and formal approach to requirement modeling and analysis

- In safety-critical domain, look forward to using formal methods to improve quality of the product or follow the standard (e.g. Do-178B/DO-333)
- How to use formal methods in requirement is still a problem
- Formal method is too hard to learn for software engineers. They struggle with math notations.

FREPA: an automated and formal approach to requirement modeling and analysis in aircraft control domain. Jincao Feng, Weikai Miao, Hanyue Zheng, Yihao Huang, Jianwen Li, Zheng Wang, Ting Su, Bin Gu, Geguang Pu, Mengfei Yang, Jifeng He， FSE2020

Tools

Software Requirement Modeling

- The Current State in Industry

  Informal document

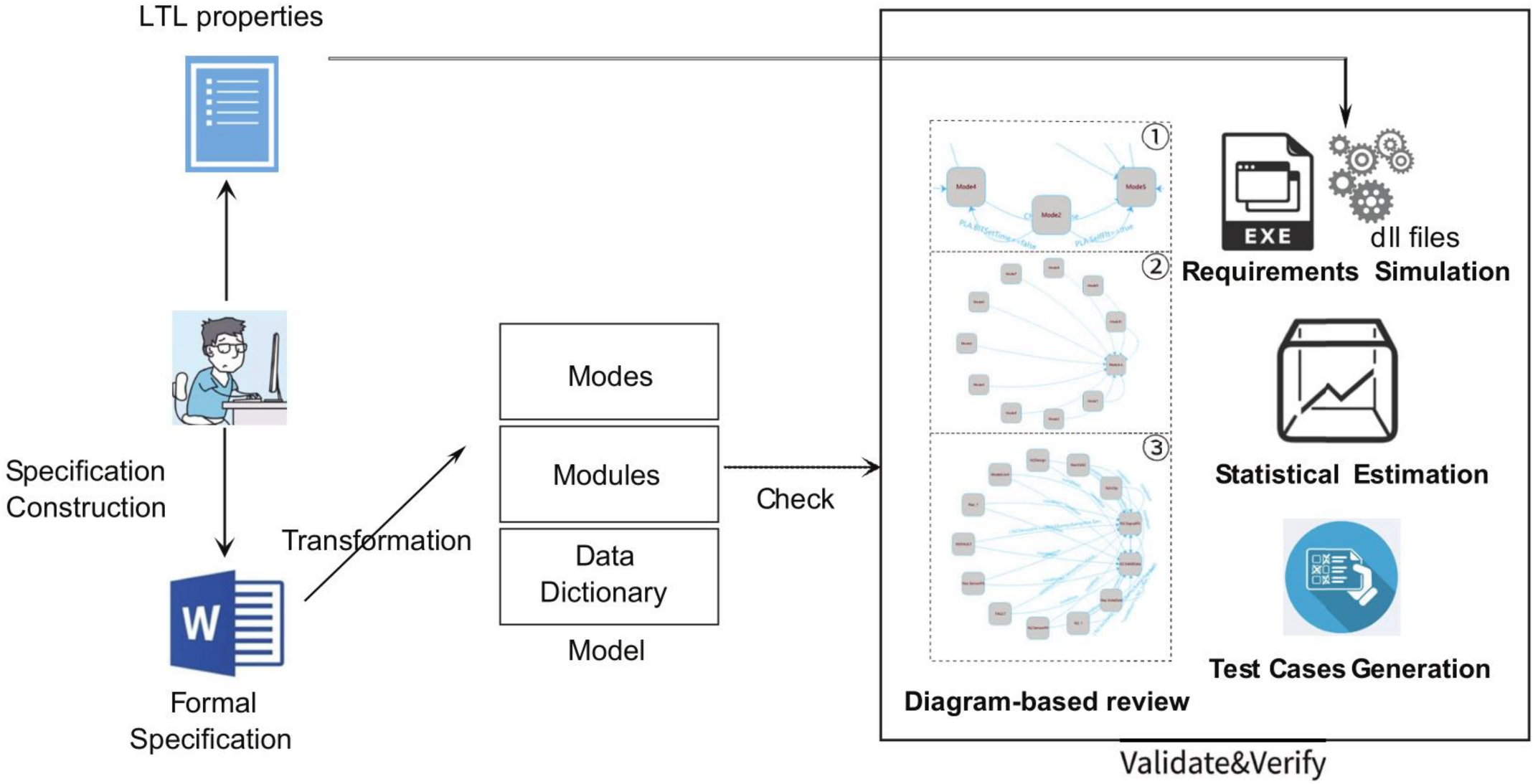
  Manul manipulation

  Human Inspection

1、Sun direction
Entry condition: *modeflag*=10
Initialization:
a. set voltage *q* as zero
b. set angular velocity ω,φ,ϕ as zero
Procedure workflow:
period is 16ms
a. call 1.1.1
(ProcDataGyro(), the progress of gyro data)
b. If(q>0)
{call 1.1.2 (the diagnosis of STS)}
else
{Call 1.1.3 (the modification of STS)}
period is 32ms
a. call 1.1.4 (the start and maintenance of Jet)
Mode transition conditions:
a. if(ω>0){set *modeflag* as 1}
b. if(φ>0){set *modeflag* as 7}
1、ProcDataGyro
ID: 1.1.1
Input: *q*
Output: *q*
…

Tools

Software Requirement Modeling



Approach Outline

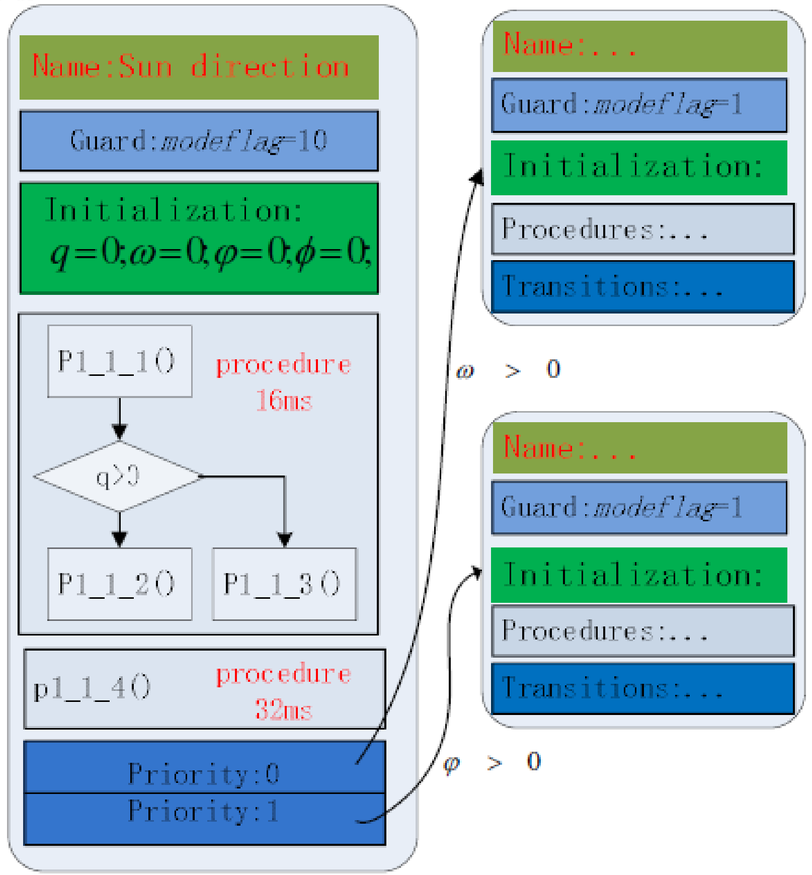
Tools

Software Requirement Modeling

Template: Bold denotes keyword

1、Sun direction
Entry condition: *modeflag*=10
Initialization:
a. set voltage *q* as zero
b. set angular velocity ω,φ,ϕ as zero
Procedure workflow:
period is 16ms
a. call 1.1.1
(ProcDataGyro(), the progress of gyro data)
b. If(q>0)
{call 1.1.2 (the diagnosis of STS)}
else
{Call 1.1.3 (the modification of STS)}
period is 32ms
a. call 1.1.4 (the start and maintenance of Jet)
Mode transition conditions:
a. if(ω>0){set *modeflag* as 1}
b. if(φ>0){set *modeflag* as 7}
1、ProcDataGyro
ID: 1.1.1
Input: *q*
Output: *q*
…

mode 1:
mode name: Sun direction
guard: *modeflag*==10
initialization:
q=0;ω=0; φ=0;ϕ=0;
procedures:
period:16ms
call 1.1.1;
if(*q*>0){call 1.1.2;}
else{call 1.1.3;}
period:32ms
call 1.1.4;
transitions:
{**priority**: 0
condition:ω>0
target mode:
modeflag=1
}
{ **priority**: 1
condition:φ>0
target mode:
modeflag=7
}
module 1:
name: ProcDataGyro
ID: 1.1.1
Input: *q*
Output: *q*
…



From informal Doc to Formal Spec

Tools

Software Requirement Modeling

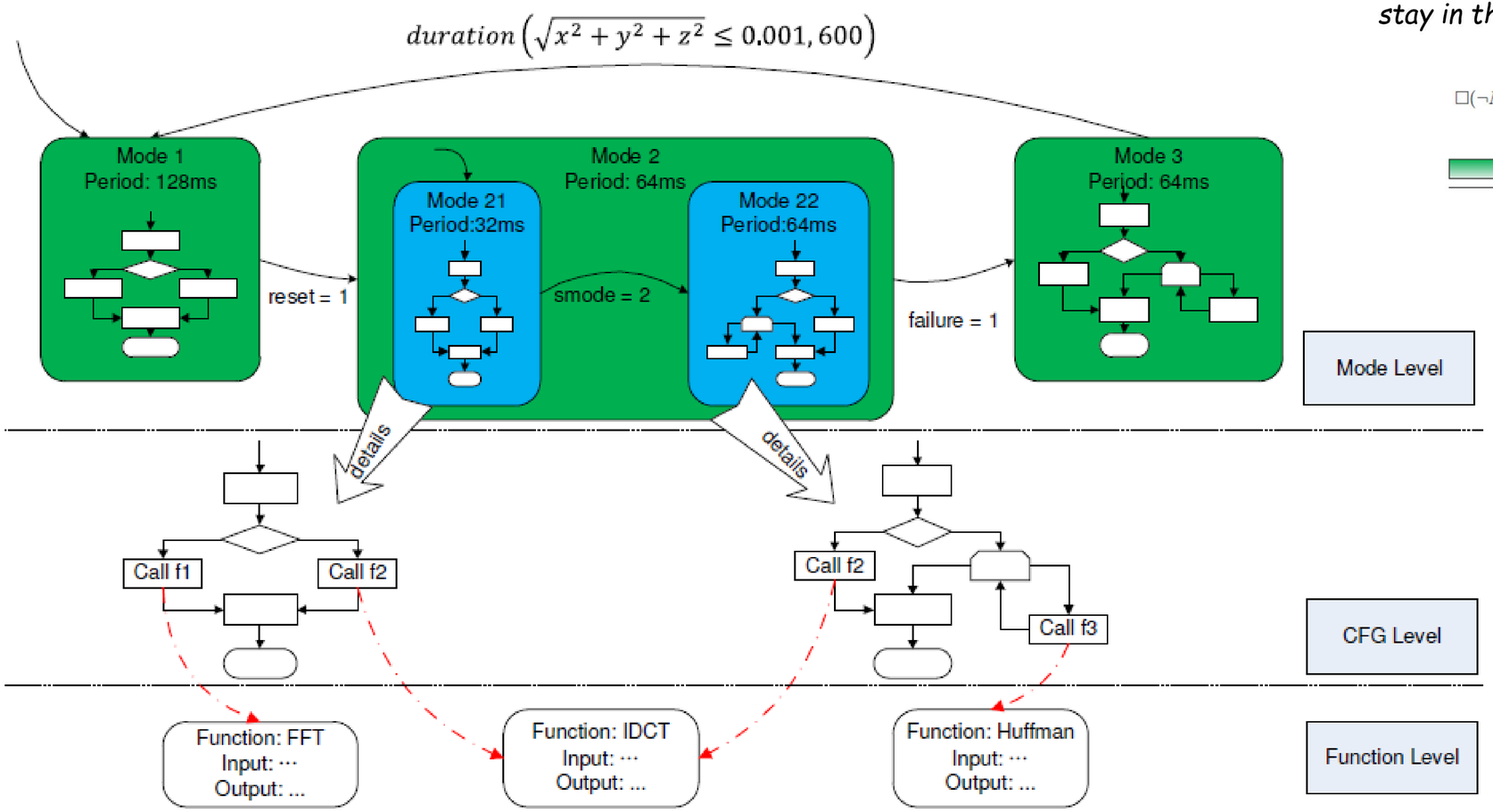
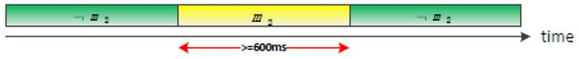


Figure 1. ModeChart Overall

"When the system switchs to mode M2, it should stay in this mode at least 600ms"

$\square(\neg M_2 \frown M_2 \frown \neg M_2 \Rightarrow l \geq 600ms)$

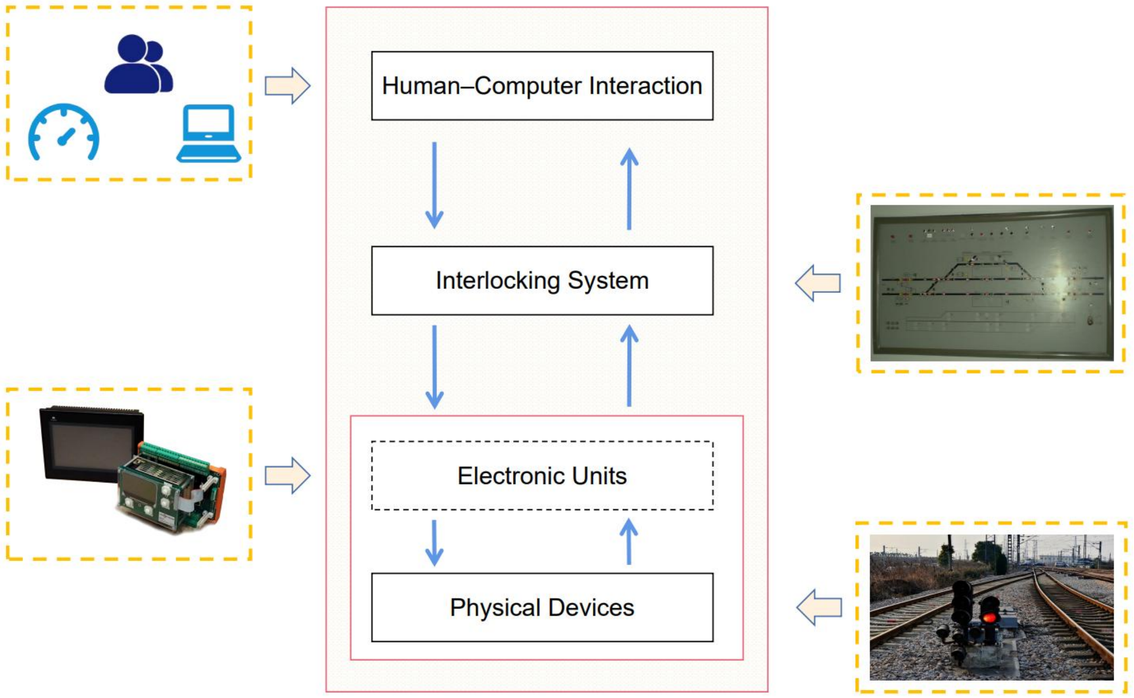


Formal Notations

Tools

System Behavior Modeling

Domain modeling: Interlocking System in Railway



System Behavior Modeling

Domain modeling: Interlocking System in Railway

- Maintain the states of devices.
- Decide whether it is safe to perform operations accordingly.
- “When there are human beings on the track, trains are not allowed to pass.”

System Behavior Modeling

Domain modeling: Interlocking System in Railway

The “Safety Goal”

- Train movements must not collide.
- Train movements must not derail.
- Tain movements must not collide with authorized vehicles or human beings crossing the railway.
- Protect railway employees from trains.

Tools

System Behavior Modeling

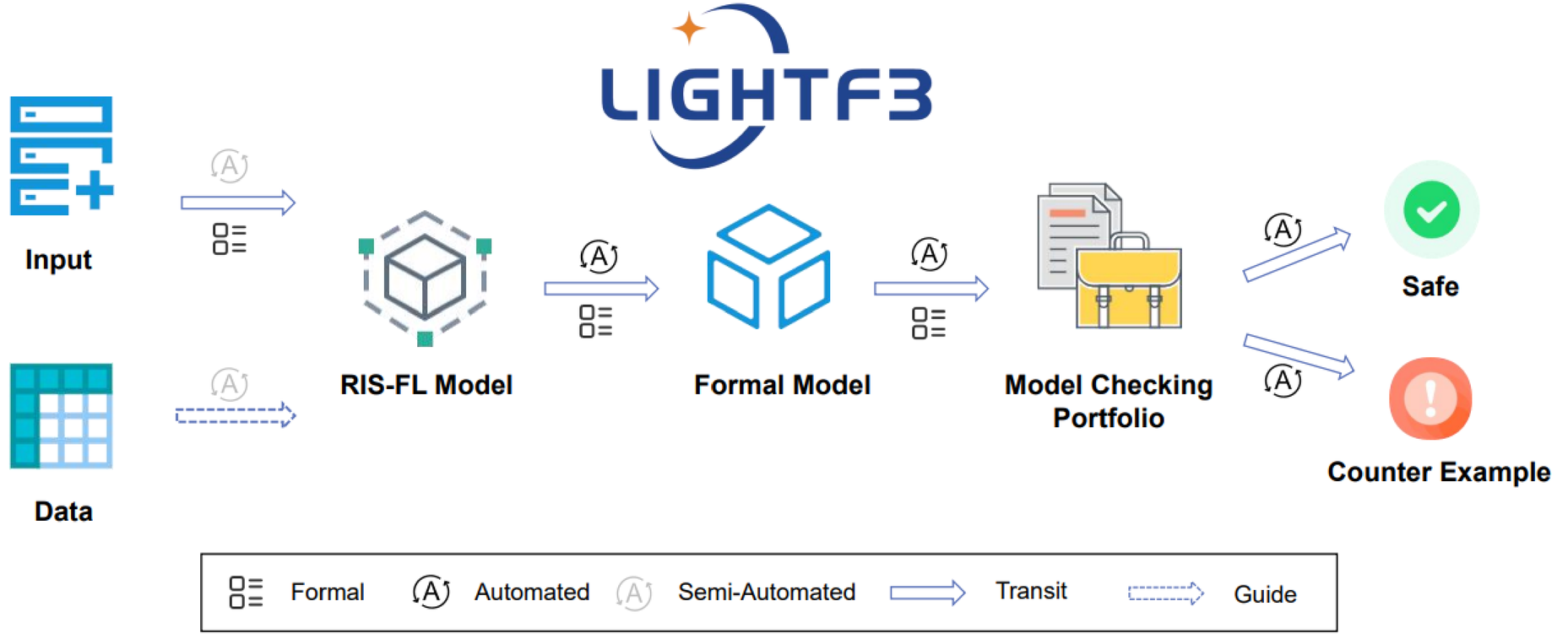
The Key Idea

- Propose a new formal language RIS-FL to allow writing formal descriptions at a low cost.
- Can effectively transit to model checking problems and carry any latest Aiger-based verifier.
- Perform well on real large-scale stations

LightF3: A Lightweight Fully-Process Formal Framework for Automated Verifying Railway Interlocking Systems, Yibo Dong, Xiaoyu Zhang, Yicong Xu, Chang Cai, Yu Chen,Weikai Miao, Jianwen Li, Geguang Pu, FSE 2023

Tools

System Behavior Modeling



Tools

System Behavior Modeling

Underlying logic: FQLTL

- LTL_p with finite quantifiers
- Syntax:

$$\phi ::= t \mid (\phi) \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \rightarrow \phi \mid P(t_1, \ldots, t_k) \mid ALL\ d\ \cdot \phi \mid SOME\ d\ \cdot \phi \mid PRE\ \phi \mid X\ \phi \mid \phi\ U_{[m_1,m_2]}\ \phi \mid \phi\ S_{[m_1,m_2]}\ \phi$$

Where:

t: atomic terms

P: predicate

X: next

$U_{[m_1,m_2]}$: Until satisfied in the future m_1 to m_2 cycles.

Example

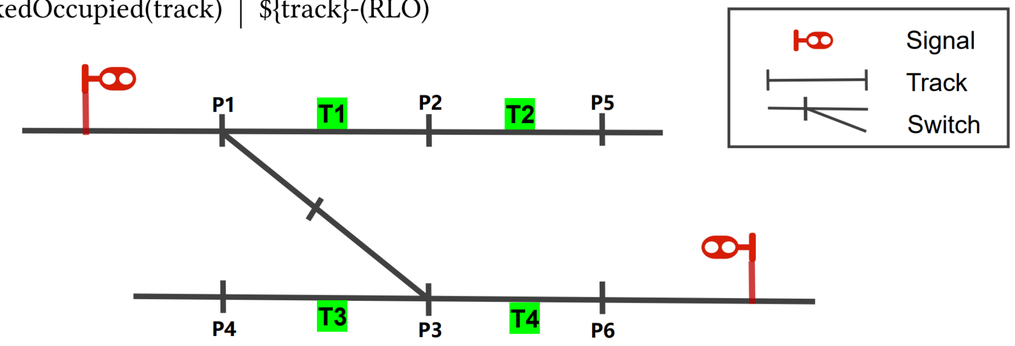
- To express the safety goal:
  If a track that contains switches is released, then following properties should hold:
  • The track is logically clear for at least 3 seconds.
  • The track is in route released state.
  • The track is not route locked or occupied.

- One shall write generic properties like:
  - SubRequirement-1 := ALL track (SOME switch (BelongToTrack(switch,track)) & Released(track) ⟶ LogicallyClearElapsed(track));
  - SubRequirement-2 := ALL track (SOME switch (BelongToTrack(switch,track)) & Released(track) ⟶ RouteReleased(track));
  - SubRequirement-3 := ALL track (SOME switch (BelongToTrack(switch,track)) & Released(track) ⟶ ¬ RouteLockedOccupied(track));

RIS-FL model

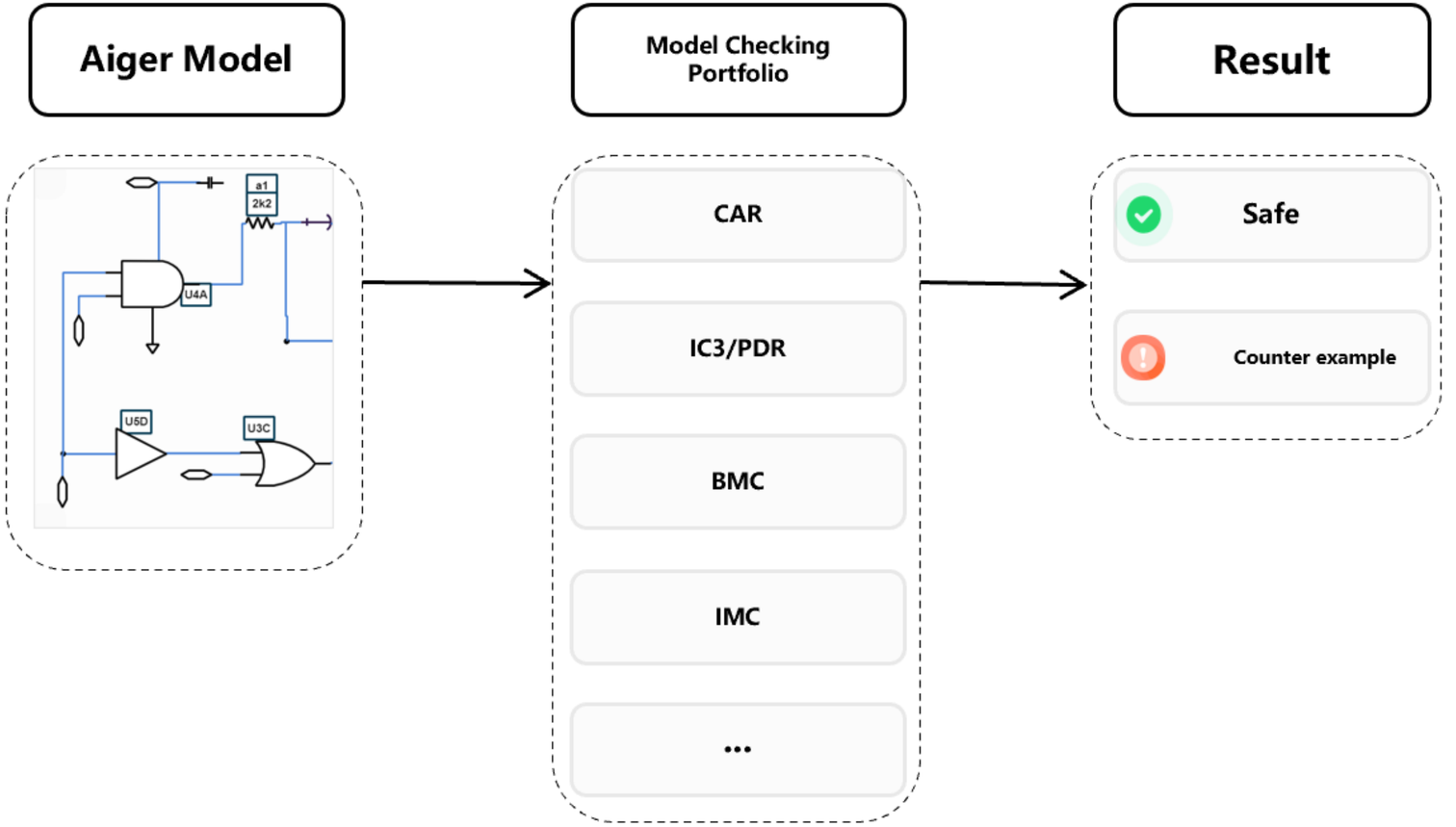
- $(Model, Props, Constraints)$,
  where:
  - $Props$: $FQLTL$ formulas, to be checked
  - $Constraints$: $FQLTL$ formulas, serve as assumptions.
  - $Model$: $\{\{Device\}, Topology, Rule\}$,
    where:
    - $Device$: $(Type, ID, \{Attr\})$
    - $Topology$: $(Vertex = \{Device\},\ Edge = \{(Device, Device)\})$
    - $Rule$: LTL_p formulas

Function name	Literal name
Released(track)	${track}-(R)
LogicallyClearElapsed(track)	${track}-(LCE)
RouteReleased(track)	${track}-(RR)
RouteLockedOccupied(track)	${track}-(RLO)



Tools

System Behavior Modeling



Formal Verification

Our Observation

The model checking technique can really help in industry for two areas: EDA verification engine and Railway Signal Verification

SYNOPSYS®

VC Formal

Leading formal innovations

Formal Applications

Automatic Extracted Properties (AEP)	Formal Coverage Analyzer (FCA)	X-Propagation Verification (FXP)	Connectivity Checking (CC)	Formal Register Verification (FRV)	Sequential Equivalence (SEQ)

PROVER®

Prover iLock®

Version 5.32

The fact

SAT-based modeling checking techniques dominate in formal verification area

IC3/PDR has been the prominent approach to safety Model Checking IC3/PDR maintains an over-approximate state sequence for proving the correctness. The sequence refinement methodology is known to be crucial for performance

Formal Verification

New model checking engine CAR

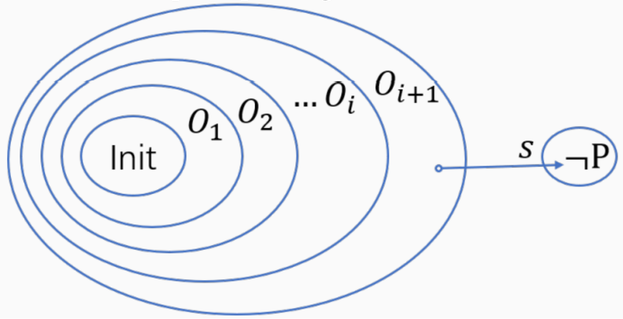
Complementary Approximate Reachability (CAR)

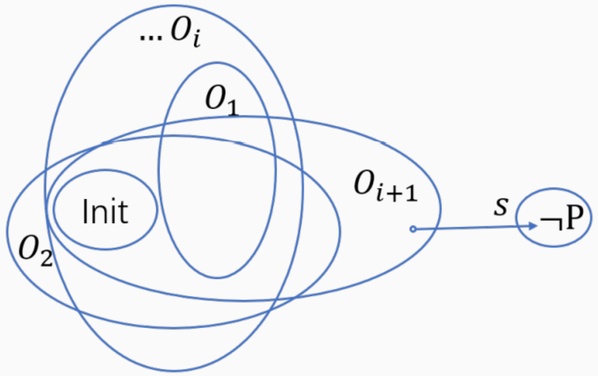
Table 1: A high-level description of IC3/PDR (left) and Forward CAR (right).

	Over-approximate	Under-approximate
Base	$O_0 = I$	-
Induction	$O_{i+1} \supseteq O_i \cup T(O_i)$	-
Safe Check	$\exists i \cdot O_{i+1} = O_i$	-
Unsafe Check	-	$\exists i \cdot T^{-i}(\neg P) \cap I \neq \emptyset$

	Over-approximate	Under-approximate
Base	$O_0 = I$	$U_0 = \neg P$
Induction	$O_{i+1} \supseteq T(O_i)$	$U_{i+1} \subseteq T^{-1}(U_i)$
Safe Check	$\exists i \cdot O_{i+1} \subseteq \bigcup_{0 \le j \le i} O_j$	-
Unsafe Check	-	$\exists i \cdot U_i \cap I \neq \emptyset$

Inclusion property





- **SimpleCAR: An Efficient Bug-Finding Tool Based on Approximate Reachability， Jianwen Li, Rohit Dureja, Geguang Pu, Kristin Yvonne Rozier, Moshe Y. Vardi， CAV2018**
- **Searching for i-Good Lemmas to Accelerate Safety Model Checking，Yechuan Xia, Anna Becchi, Alessandro Cimatti, Alberto Griggio, Jianwen Li, Geguang Pu， CAV2023**

The current state for optimization in model checkers mostly depends on luck

The motivation: we should know why our optimization algorithms behave well or bad?

The current state for optimization in model checkers

mostly depends on luck

The motivation: we should know why our optimization algorithms behave well or bad?

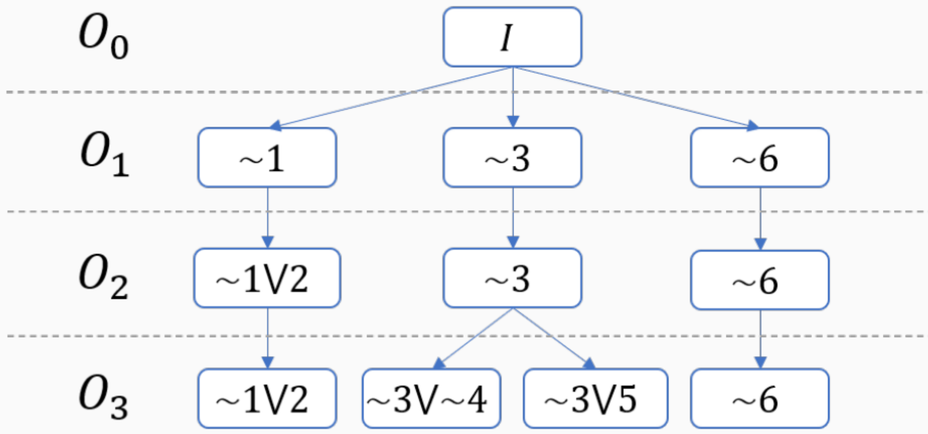
Two ideas:

1、Visualize the search track

2、Hack the model checker

Formal Verification

Visualization of Over(-approximation) Sequence



$$O_0 = I$$

$$O_1 = \neg 1 \wedge \neg 3 \wedge \neg 6$$

$$O_2 = (\neg 1 \vee 2) \wedge \neg 3 \wedge \neg 6$$

$$O_3 = (\neg 1 \vee 2) \wedge (\neg 3 \vee \neg 4) \wedge (\neg 3 \vee 5) \wedge \neg 6$$

Formal Verification

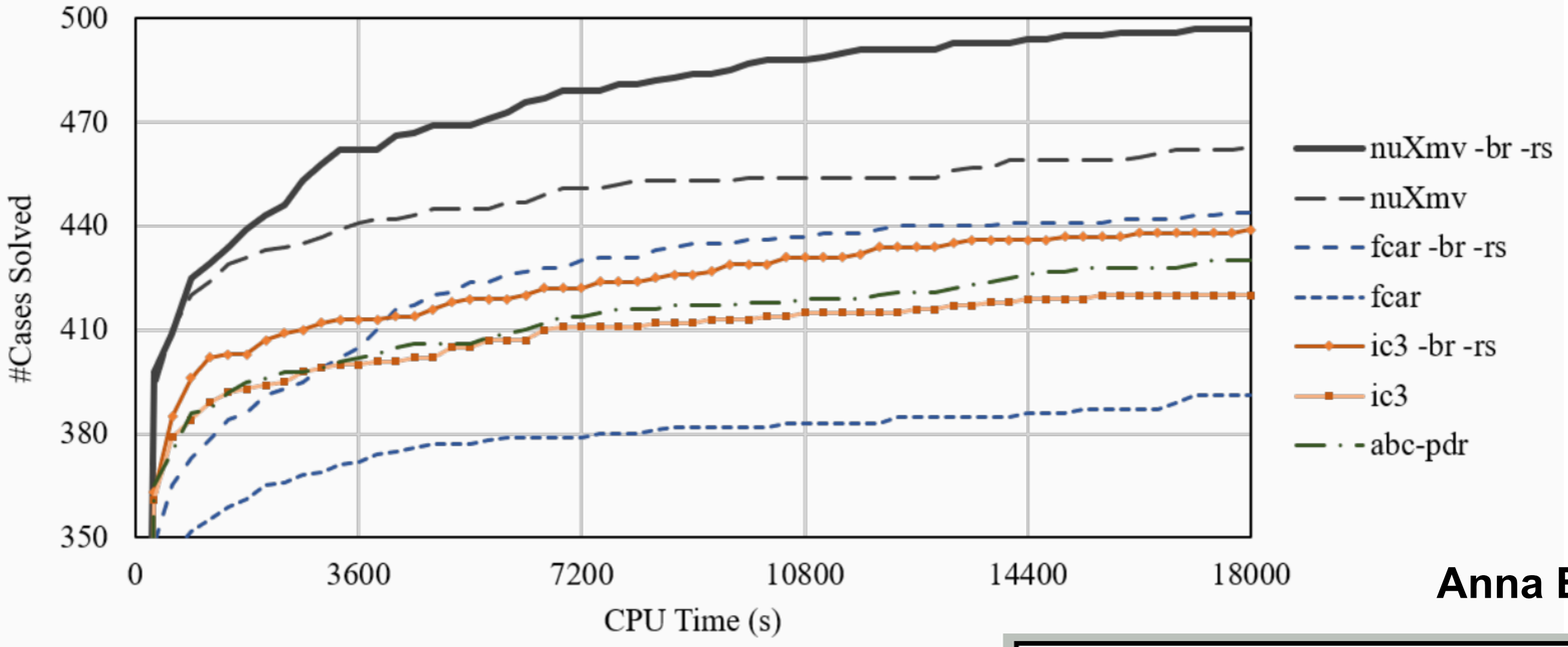
Definition (i-Good Lemma). Let c be a lemma that was added at frame i by IC3/CAR (at some previous step in the execution of the algorithm), i.e. $F_i \models c$. We say that c is i-good if c now holds also at frame $i+1$, i.e. $F_{i+1} \models c$.

Theorem 1. IC3 terminates with safe at frame i ($i > 0$), if and only if every lemma at frame i is i-good.

Theorem 2. CAR terminates with safe at frame i ($i > 0$), if every lemma at frame i is i-good.

Tools

Formal Verification



Heuristics for searching i-good lemmas

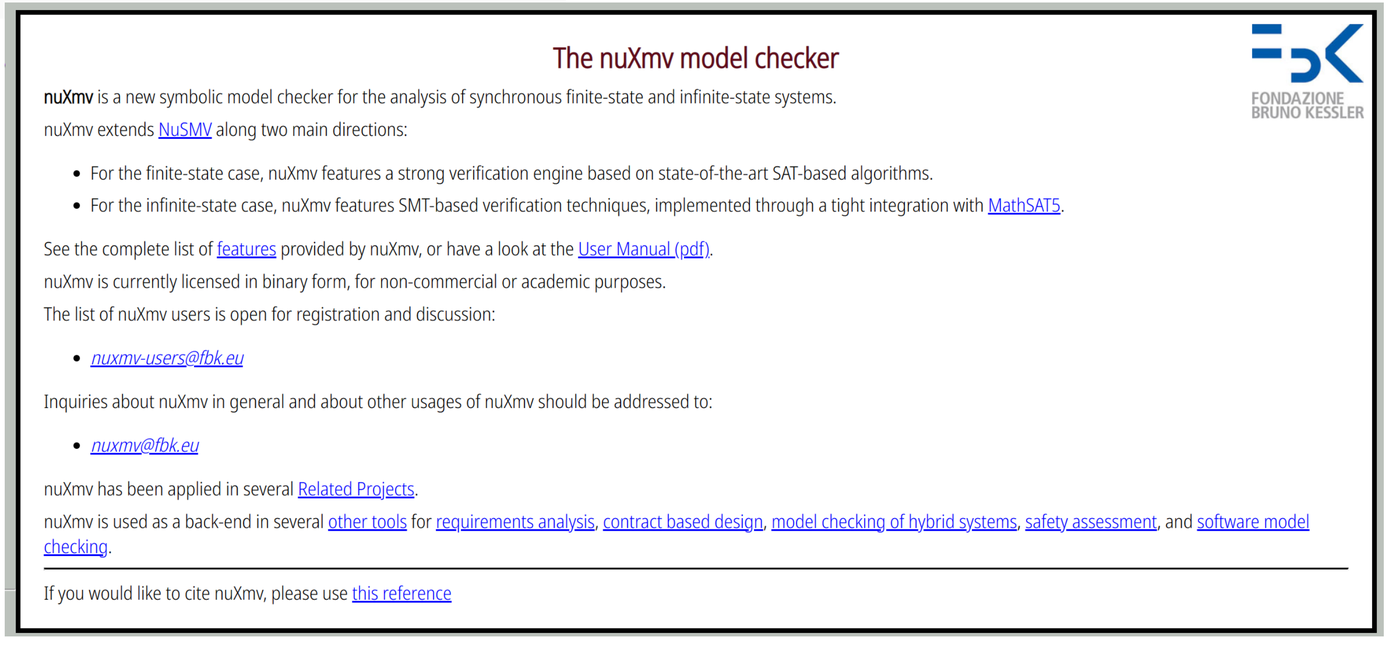
- Unsatisfiable cores are the source for lemmas (and states) in both IC3 and CAR.
- Using SAT assumptions with different orders will return different unsat cores.
  is_SAT ($O_i \wedge T$, $(1 \wedge 2 \wedge 3 \wedge \neg 4 \wedge 5 \wedge 6)'$) unsat core: $3' \wedge \neg 4'$
  is_SAT ($O_i \wedge T$, $(1 \wedge 2 \wedge 6 \wedge 5 \wedge 3 \wedge \neg 4)'$) unsat core: $6'$

Branching

- Maintain a priority for every state variable
- Reward variables in lemma c, if c is identified as an i-good lemma
- Order the SAT assumptions before SAT calls

Anna Becchi, Alessandro Cimatti, Alberto Griggio

Tools	Algorithms	Available Flags
IC3Ref	IC3 (ic3)	-br | -rs
SimpleCAR	Forward CAR (fcar)	-br | -rs
nuXmv	IC3 (nuXmv)	-br | -rs
IIMC	QUIP (iimc-quip)	-
IIMC	IC3 (iimc-ic3)	-
ABC	PDR (abc-pdr)	-



The nuXmv model checker

FONDAZIONE BRUNO KESSLER

nuXmv is a new symbolic model checker for the analysis of synchronous finite-state and infinite-state systems.

nuXmv extends NuSMV along two main directions:

- For the finite-state case, nuXmv features a strong verification engine based on state-of-the-art SAT-based algorithms.
- For the infinite-state case, nuXmv features SMT-based verification techniques, implemented through a tight integration with MathSAT5.

See the complete list of features provided by nuXmv, or have a look at the User Manual (pdf).

nuXmv is currently licensed in binary form, for non-commercial or academic purposes.

The list of nuXmv users is open for registration and discussion:

- *nuxmv-users@fbk.eu*

Inquiries about nuXmv in general and about other usages of nuXmv should be addressed to:

- *nuxmv@fbk.eu*

nuXmv has been applied in several Related Projects.

nuXmv is used as a back-end in several other tools for requirements analysis, contract based design, model checking of hybrid systems, safety assessment, and software model checking.

If you would like to cite nuXmv, please use this reference

Automated Testing

Testing is the most mainstream approach to ensuring the quality of software

Dozens of testing methods

Thousands of testing researchers

Too many commercial testing tools

Automated Testing

Testing is the most mainstream approach to ensuring the quality of software

Dozens of testing methods

Thousands of testing researchers

Too many commercial testing tools

Tools

Automated Testing

Testing is the most mainstream approach to ensuring the quality of software

Dozens of testing methods

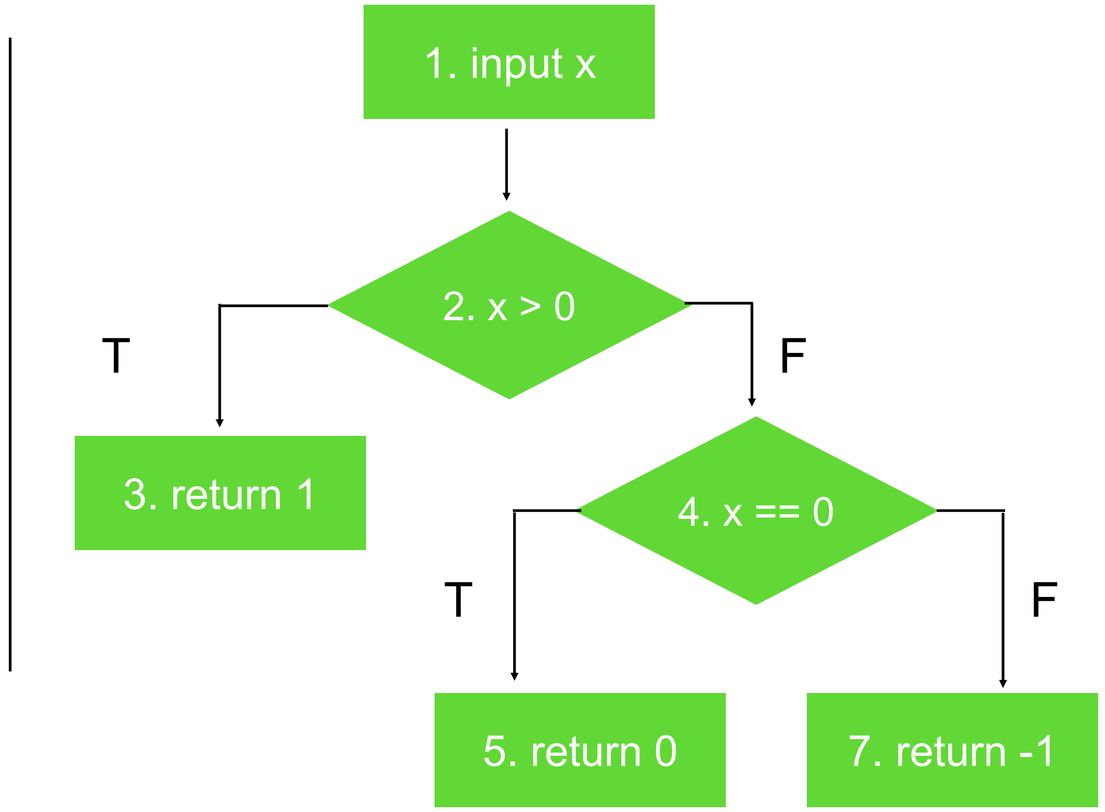
Thousands of testing researchers

Too many commercial testing tools

Key Point in Testing : How to generate test cases automatically

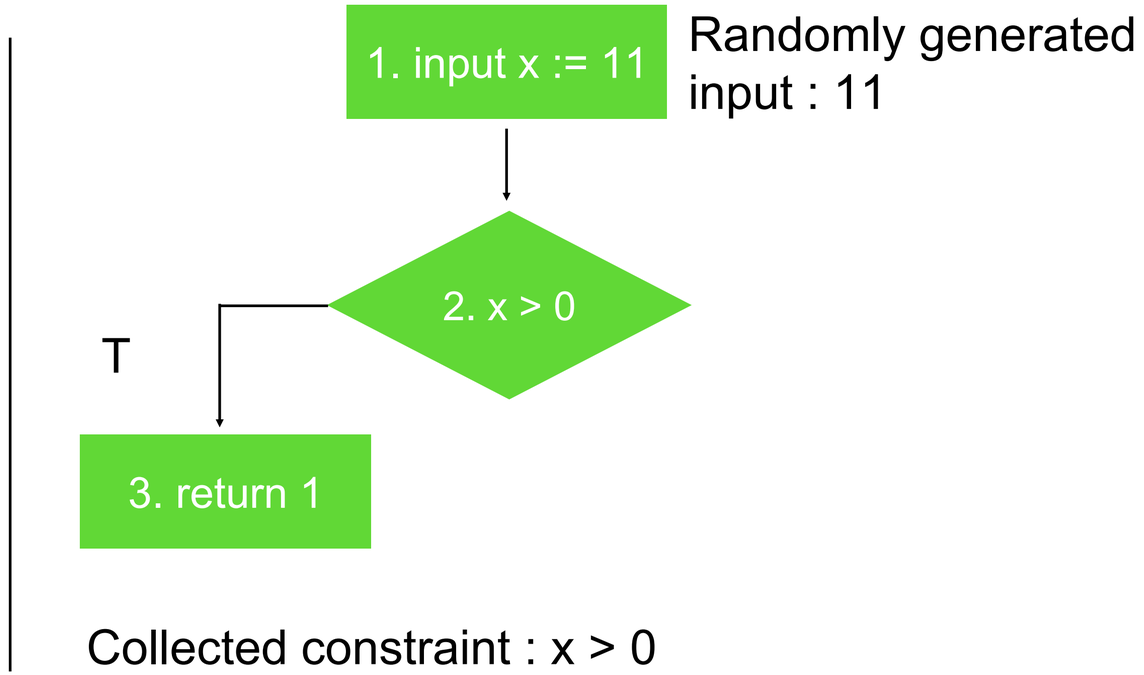
Dynamic Symbolic Execution

```
1  int checkSign (int x){
2    if (x > 0)
3      return 1;
4    else if (x == 0)
5      return 0;
6    else
7      return -1;
8  }
```



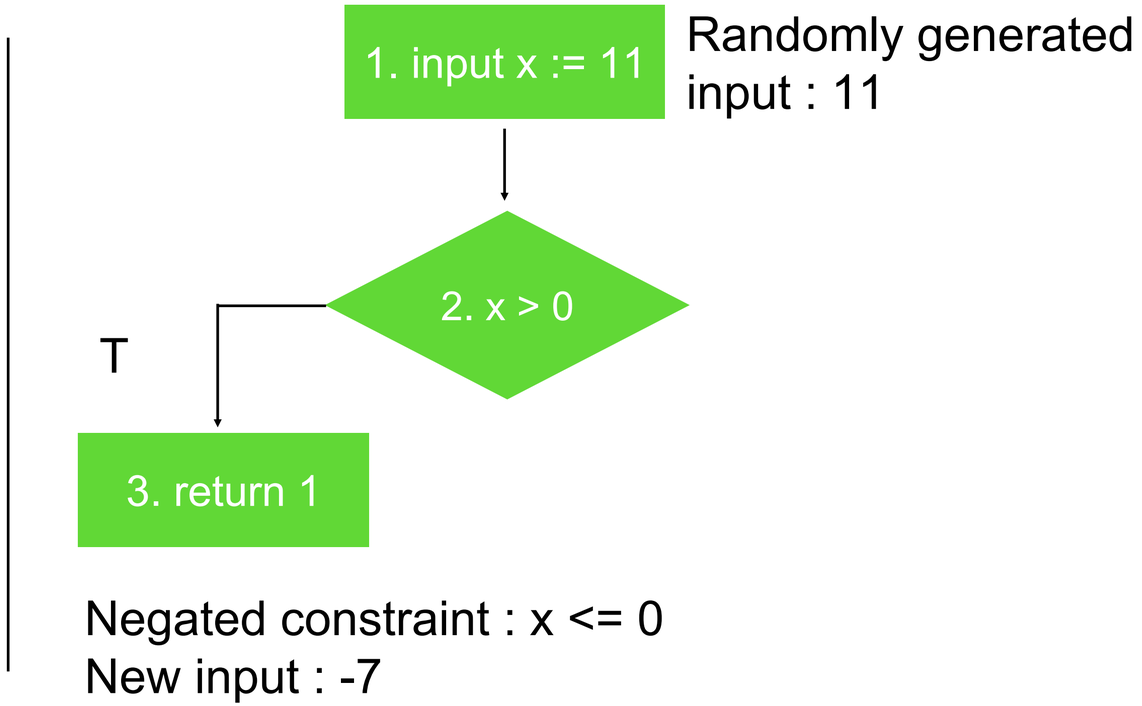
Dynamic Symbolic Execution

```
1  int checkSign (int x){
2    if (x > 0)
3      return 1;
4    else if (x == 0)
5      return 0;
6    else
7      return -1;
8  }
```



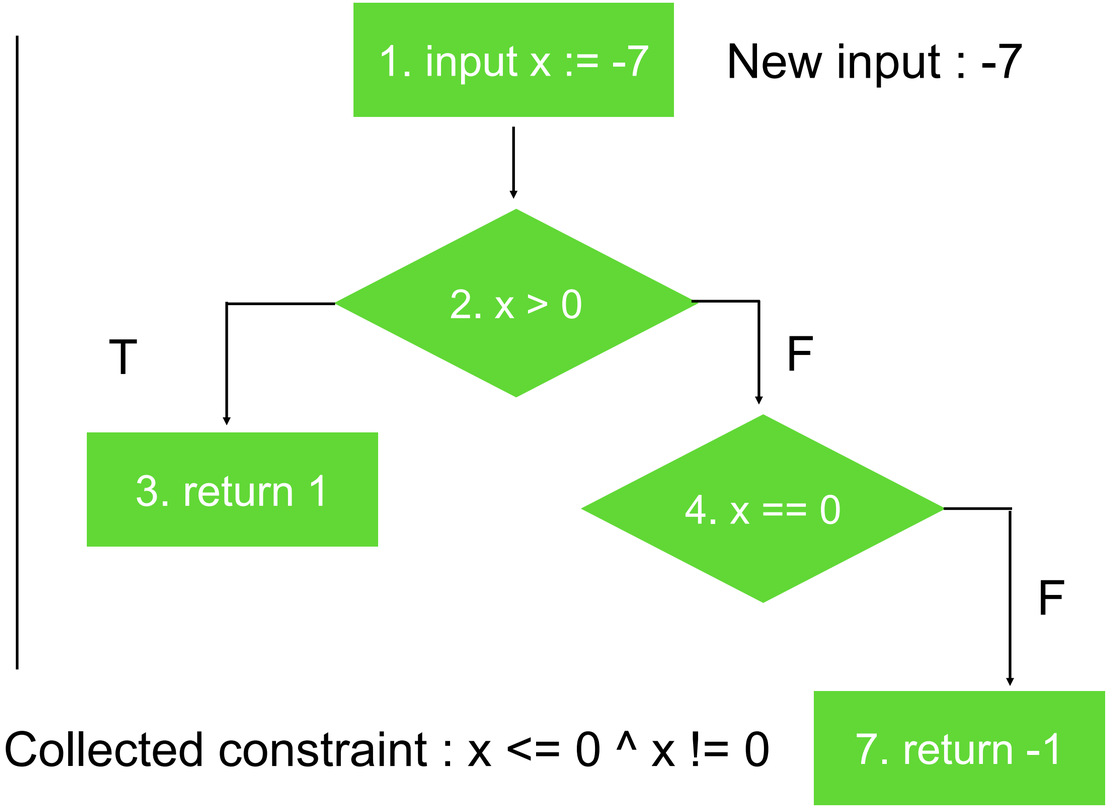
Dynamic Symbolic Execution

```
1  int checkSign (int x){
2    if (x > 0)
3      return 1;
4    else if (x == 0)
5      return 0;
6    else
7      return -1;
8  }
```



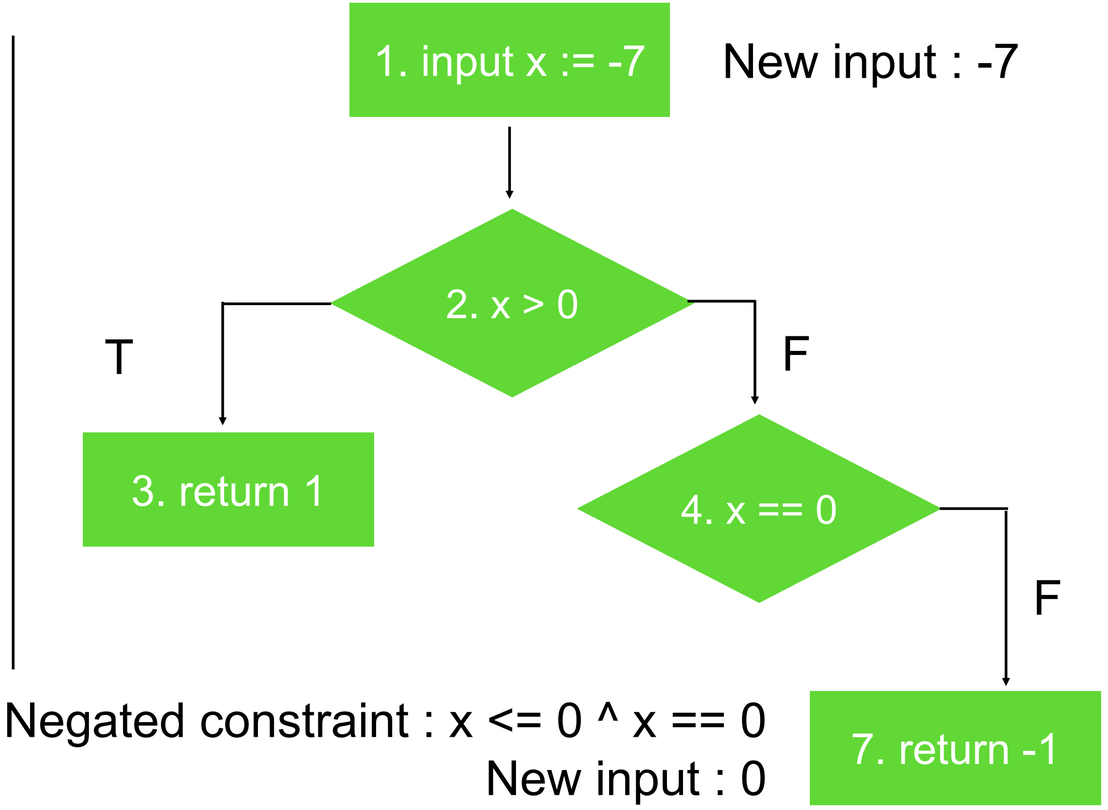
Dynamic Symbolic Execution

```
1  int checkSign (int x){
2    if (x > 0)
3      return 1;
4    else if (x == 0)
5      return 0;
6    else
7      return -1;
8  }
```



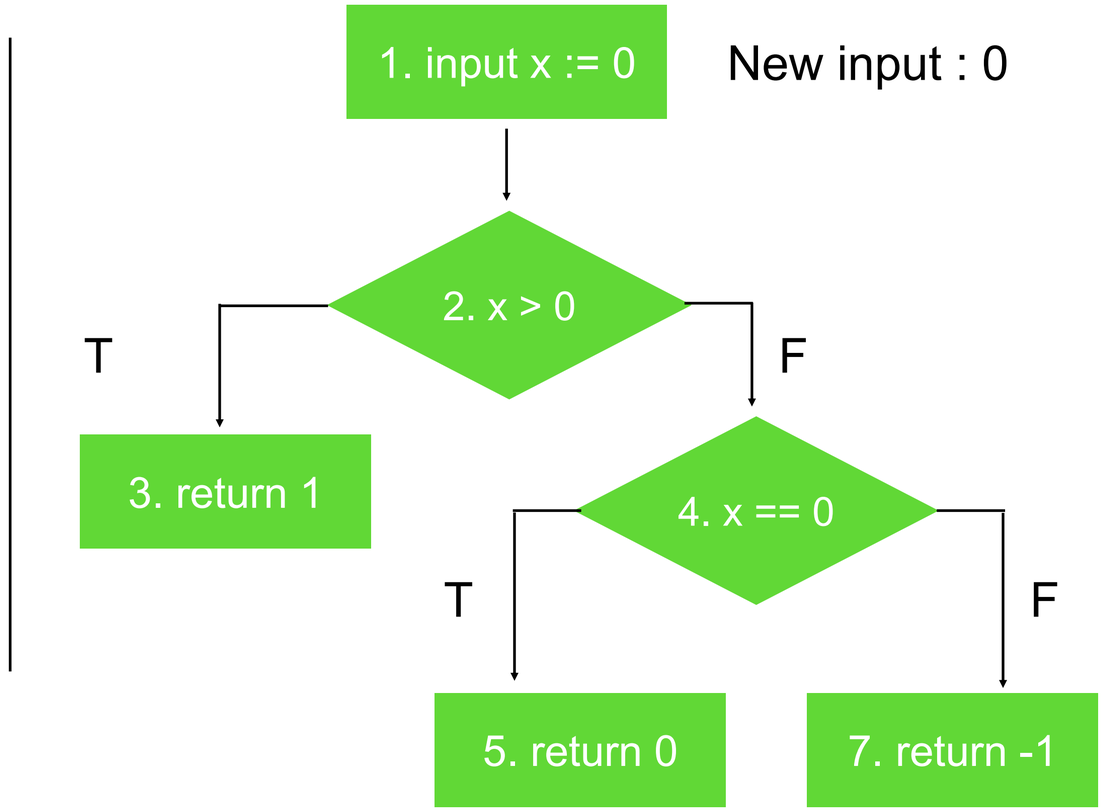
Dynamic Symbolic Execution

```
1  int checkSign (int x){
2    if (x > 0)
3      return 1;
4    else if (x == 0)
5      return 0;
6    else
7      return -1;
8  }
```



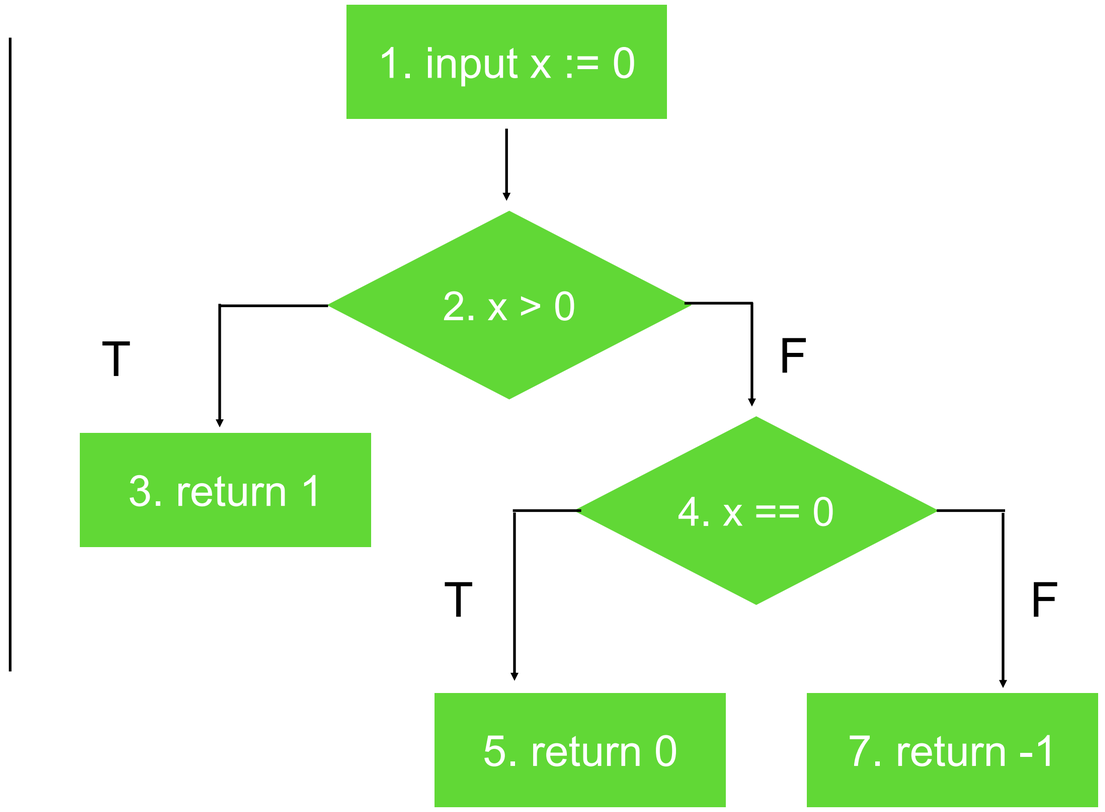
Dynamic Symbolic Execution

```
1  int checkSign (int x){
2    if (x > 0)
3      return 1;
4    else if (x == 0)
5      return 0;
6    else
7      return -1;
8  }
```



Dynamic Symbolic Execution

```
1  int checkSign (int x){
2    if (x > 0)
3      return 1;
4    else if (x == 0)
5      return 0;
6    else
7      return -1;
8  }
```



Test suite :
{ 11, -7, 0 }

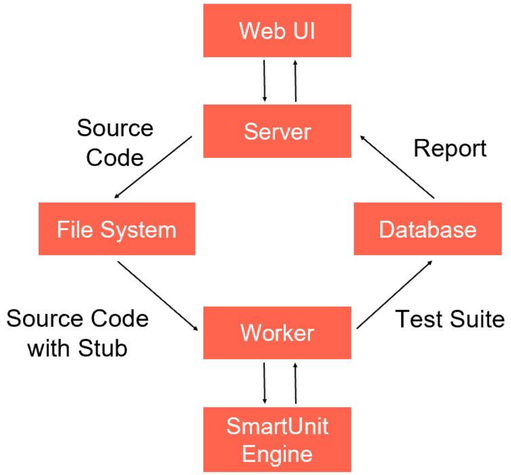
Tools

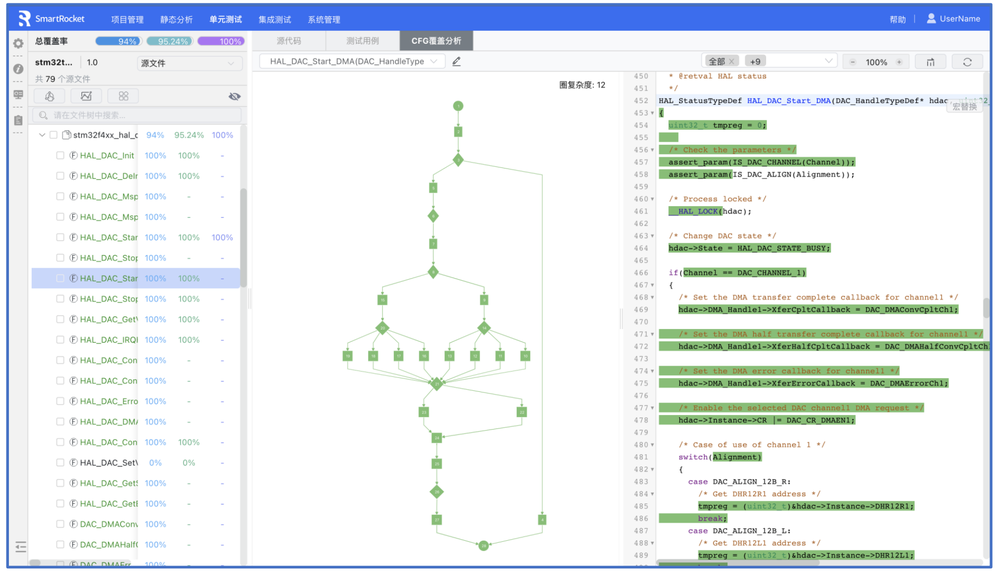
Automated Testing

The Simple Idea leads to better commercial tool

Automatically {
- inserts stubs for function calls & variables (Global variables, function inputs, etc.)
- generates test suite. (For LDRA Testbed, Tessy, etc.)
- generates test report. (Statement, branch, MC/DC coverage)
}

Cloud-based Platform Workflow





Chengyu Zhang, Yichen Yan, Hanru Zhou, Yinbo Yao, Ke Wu, Ting Su, Weikai Miao, Geguang Pu: Smartunit: empirical evaluations for automated unit testing of embedded software in industry. ICSE (SEIP) 2018: 296-305

The Simple Idea leads to better commercial tool

How about the performance of SmartUnit ?

Subjects	# Files	# Functions	# LOC
Aerospace Software	8	54	3,769
Automotive Software	4	330	31,760
Subway Signal Software	108	874	37,506
SQLite	2	2,046	126,691
PostgreSQL	906	6,105	279,809
Total	1,028	9,409	479,535

How about the performance of SmartUnit ?

Subjects	Statement Coverage*				Branch Coverage*				MC/DC Coverage*			
	N/A	0-50%	50-99%	100%	N/A	0-50%	50-99%	100%	N/A	0-50%	50-99%	100%
Aerospace Software	1	3	10	41	1	5	8	41	45	2	-	8
Automotive Software	1	3	11	315	1	6	8	315	274	5	1	50
Subway Signal Software	6	1	50	817	6	2	55	811	558	11	11	294
SQLite	86	86	206	1668	86	119	205	1636	1426	118	149	351
PostgreSQL	687	732	1044	3642	687	1102	804	3512	4083	1308	249	465

Most of the functions can achieve 100% coverage

Can SmartUnit find the potential runtime exceptions in real-world software?

- Array index out of bounds
- Fixed memory address
- Divided by zero

- Divided by zero

```
1  static void getLocalPayload(int nUsable, u8 flags, int nTotal, int *pnI
2    int nLocal, nMinLocal, nMaxLocal;
3    if( flags==0x0D ){
4      nMinLocal = (nUsable - 12) * 32 / 255 - 23;
       nMaxLocal = nUsable - 35;
     }else{
5      nMinLocal = (nUsable - 12) * 32 / 255 - 23;
6      nMaxLocal = (nUsable - 12) * 64 / 255 - 23;
7    }
8    nLocal = nMinLocal + (nTotal - nMinLocal) % (nUsable - 4);
9  }
```

- Array index out of bounds

```
1  static char *cmdline_option_value(int argc, char **argv, int i) {
2    if (i == argc) {
3      utf8_printf(stderr, "Error: missing argument to %s\n",
4                  argv[0], argv[argc - 1]);
       exit(1);
5    }
6    return argv[i];
7  }
```

Lessons we got

Lessons we got

Listen to customers

Lessons we got

No silver bullet

Nearly 60 engineers

Each team is about 15 engineers for one tool

Meeting everyday

Review everyday

Find problems as early as possible

Management is more important than techniques

Hire, train, fire

Lessons we got

Believe success belongs to the persevering!

Believe the future

Future?

Future

AI is controlling the World ?

ChatGPT

Get instant answers, find creative inspiration, learn something new.

Future

What is the future of the way to produce software?

Future

What is the future of the way to produce software?

Input : Software Requirement

Output : Software Product

What is the future of the way to produce software?

Input : Software Requirement **Output : Software Product**

New Powerful Compiler

Formal method is the future

Thank You!