Authors:
ELSON KURIAN
GIOVANNI DENARO
DANIELA BRIOLA
PIETRO BRAIONE

Presentation:

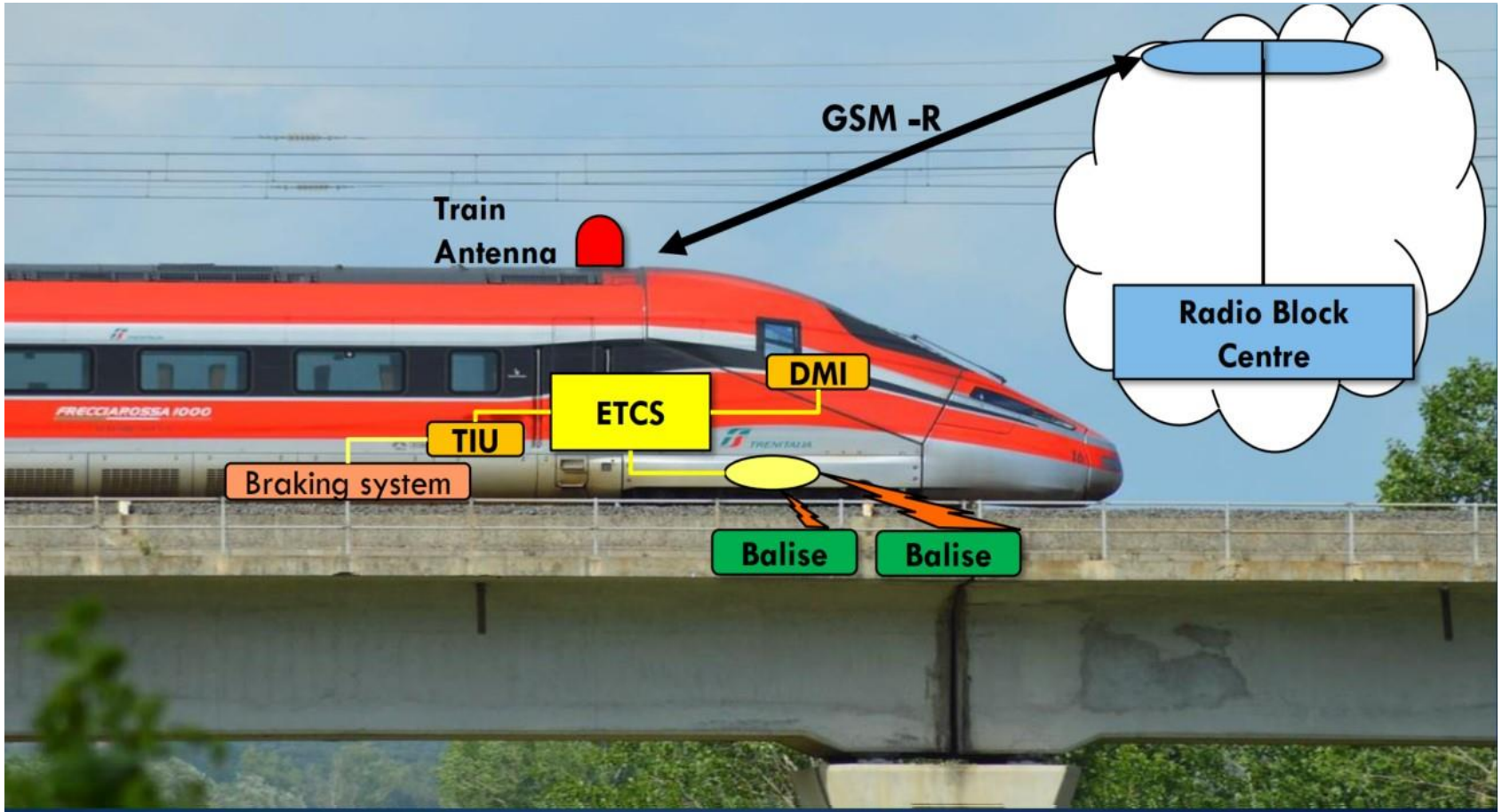
Effective test generation for Safety-Critical Software's with KLEE

AGENDA

- Goal
  - Exploit symbolic execution for test generation for safety critical software
- TECS: Test Engine for Critical Software
  - Tool to automatically generate test cases for SCADE programs
- Case study
  - On 15 SCADE programs
- Conclusions

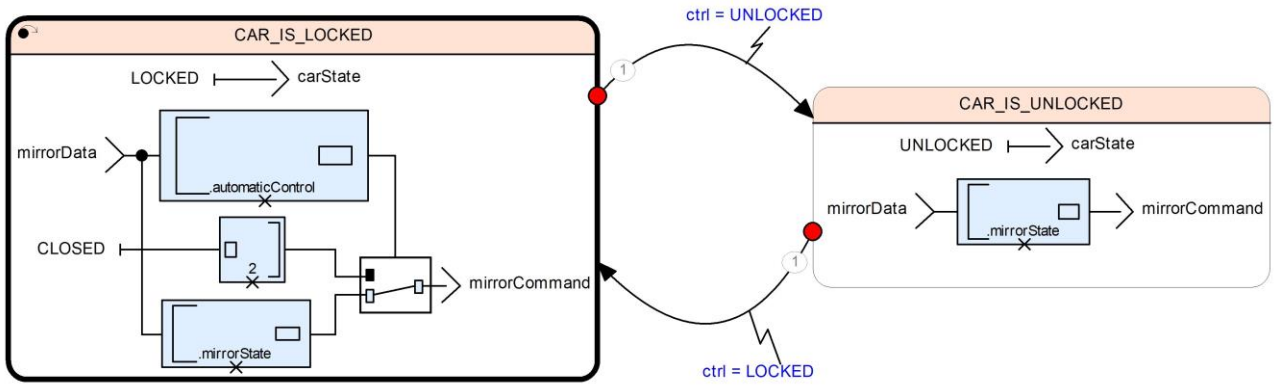
SAFETY- CRITICAL SOFTWARE

- Developing and Testing a Real-Time Embedded Critical Software for our industrial partner in the Railway Domain (RFI)



ETCS IS PROGRAMMED WITH SCADE

- Based on state machines
- Compiles to C (by KCG compiler)
- Guarantees code that meets CEI EN 50128 certification



(a) Working Example (wingMirrorControl) for car

```
##################################################
## WingMirrorControl_WingMirrorFSM, Test case: 00002
##################################################

#Test step 1
SSM::set ctrl UNLOCKED
SSM::set wingMirrorData.automaticControl false
SSM::set wingMirrorData.mirrorState {(OPEN,OPEN)}
SSM::check carState UNLOCKED
SSM::check mirrorCommand {(OPEN, OPEN)}
SSM::cycle

#Test step 2
SSM::set ctrl LOCKED
SSM::set wingMirrorData.automaticControl true
SSM::set wingMirrorData.mirrorState {(OPEN, OPEN)}
SSM::check carState LOCKED
SSM::check mirrorCommand {(CLOSED, CLOSED)}
SSM::cycle
```

(b) The SCADE test case synthesized out of the test inputs from KLEE

Our goal: Exploit symbolic execution for testing SCADE programs using KLEE

- **State of the art:** Open problems of symbolic execution[1], e.g., due to
  - Coping with the *path explosion problem*
  - Suitably handling non-numeric inputs
    - pointers
    - references to dynamically allocated, possibly recursive data structures
  - Tolerating the limitations of constraint solvers with *complex non-linear constraints*

Our intuition: These problems are hard to solve in general, but the restrictions imposed by the development process of safety-critical softwares can foster effective solutions

[1] Baldoni et al. A Survey of Symbolic Execution Techniques. ACM CSUR 2018
[2] Cadar et al. KLEE. USENIX 2008

For example, with SCADE

- SCADE generates KCG C code with safety-oriented restrictions
- **KCG restrictions**
  - No dynamic memory allocation (e.g., no malloc)
  - No pointer arithmetic, no pointer alias
  - No recursive (unbounded) data structures
  - No recursion
  - No loops with unbounded conditions (e.g., no while (TRUE) loops)

https://www.ansys.com/it-it/products/embedded-software/ansys-scade-suite

Our Research Hypothesis

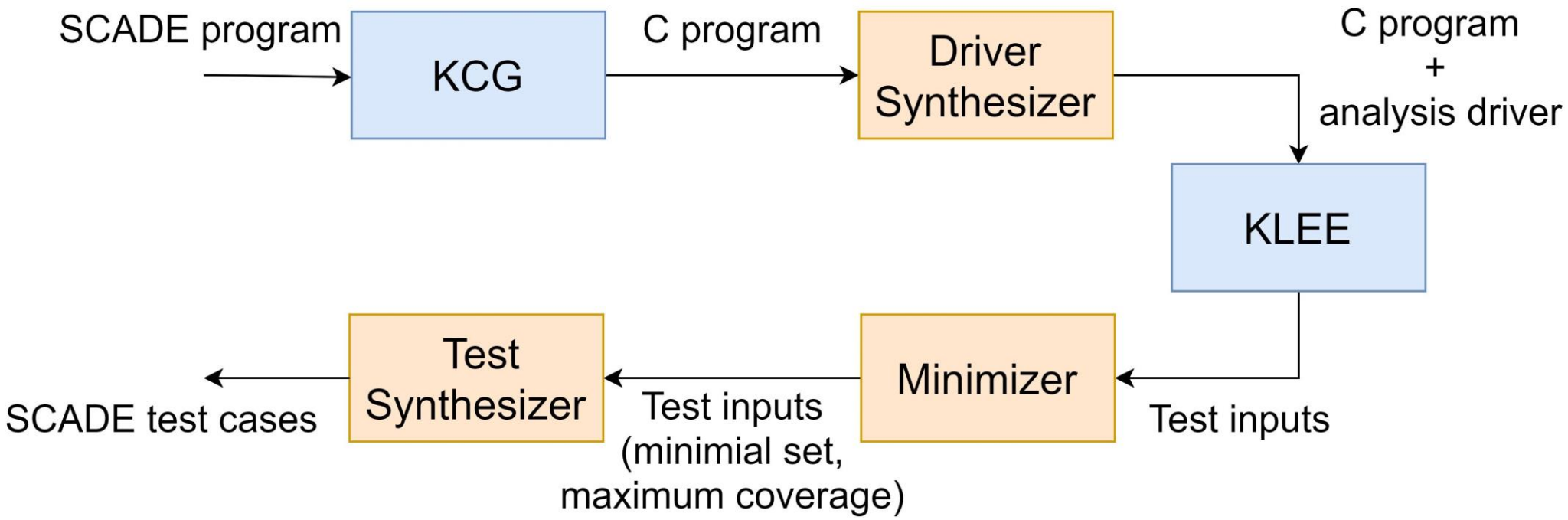
- *Automated test generation* based on *symbolic execution* can be beneficial for systematically testing *safety-critical software*
- The restrictions imposed by the safety-critical softwares can foster effective solutions.
- **Our context:** SCADE and KCG Code

AGENDA

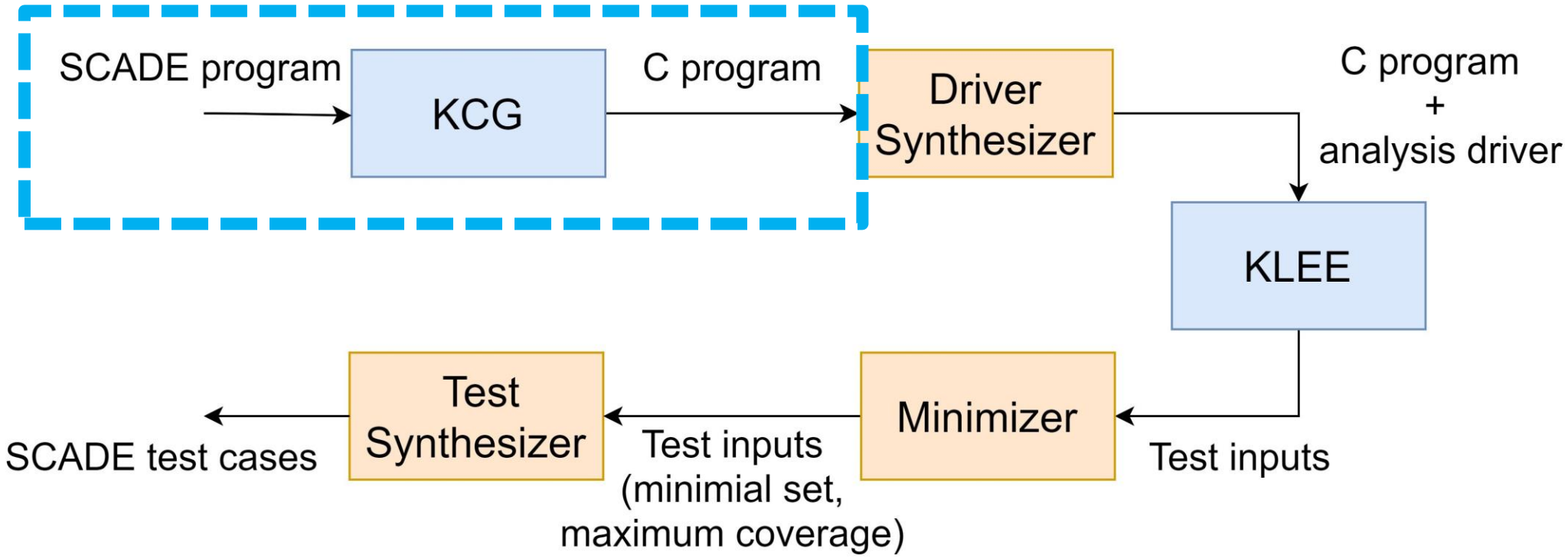
- Goal
  - Exploit symbolic execution for test generation for safety critical software
- TECS: Test Engine for Critical Software
  - Tool to automatically generate test cases for SCADE programs
- Case study
  - On 15 SCADE programs
- Conclusions

TECS: Test Engine for Critical Software

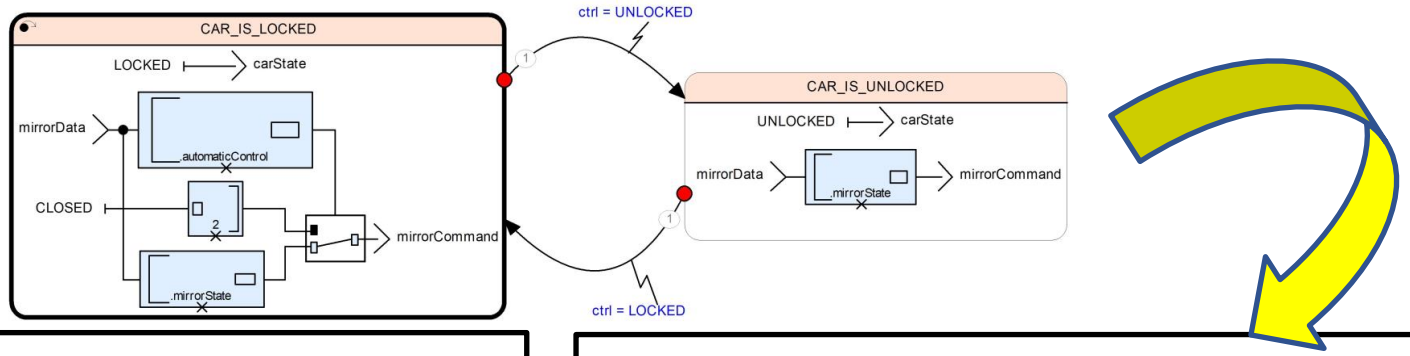
- Automatically generate unit-level test cases for SCADE



TECS: Test Engine for Critical Software



SCADE & KCG



Kcg_types:

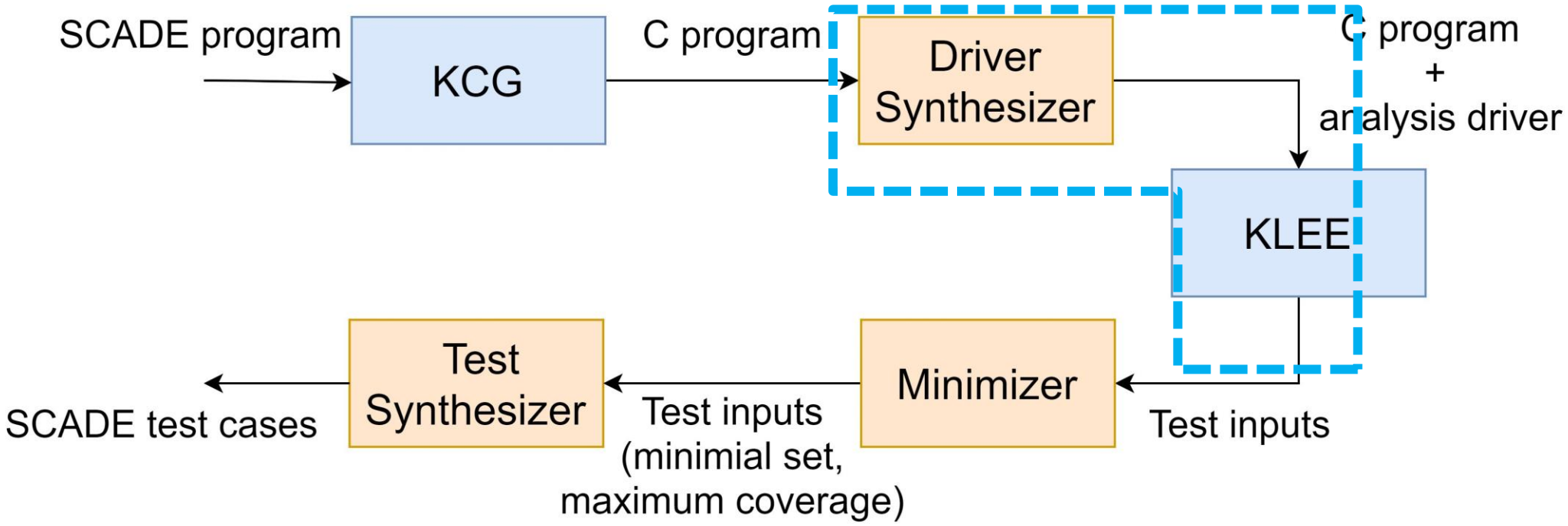
```
…
/* WingMirrorState/ */
typedef enum kcg_tag_WingMirrorState { OPEN, CLOSED }
WingMirrorState;
/* Lock/ */
typedef enum kcg_tag_Lock { UNLOCKED, LOCKED } Lock;
…
/* WingMirrorArray/ */
typedef WingMirrorState WingMirrorArray[2];

/* WingMirrorData/ */
typedef struct kcg_tag_WingMirrorData {
  kcg_bool automaticControl;
  WingMirrorArray mirrorState;
} WingMirrorData;
…
```

WingMirrorControl_CarControl.c

```
…
void WingMirrorControl_CarControl(
  inC_WingMirrorControl_CarControl *inC,
  outC_WingMirrorControl_CarControl *outC)
{
SSM_ST_WingMirrorFSM WingMirrorFSM_state_act;
  kcg_size idx;
switch (outC->WingMirrorFSM_state_nxt) {
    case SSM_st_CAR_IS_UNLOCKED_WingMirrorFSM :
      if (inC->ctrl == LOCKED) {
        WingMirrorFSM_state_act = SSM_st_CAR_IS_LOCKED_WingMirrorFSM;
      }
      else {
        WingMirrorFSM_state_act = SSM_st_CAR_IS_UNLOCKED_WingMirrorFSM;
      }
      break;
…
```

TECS: Test Engine for Critical Software



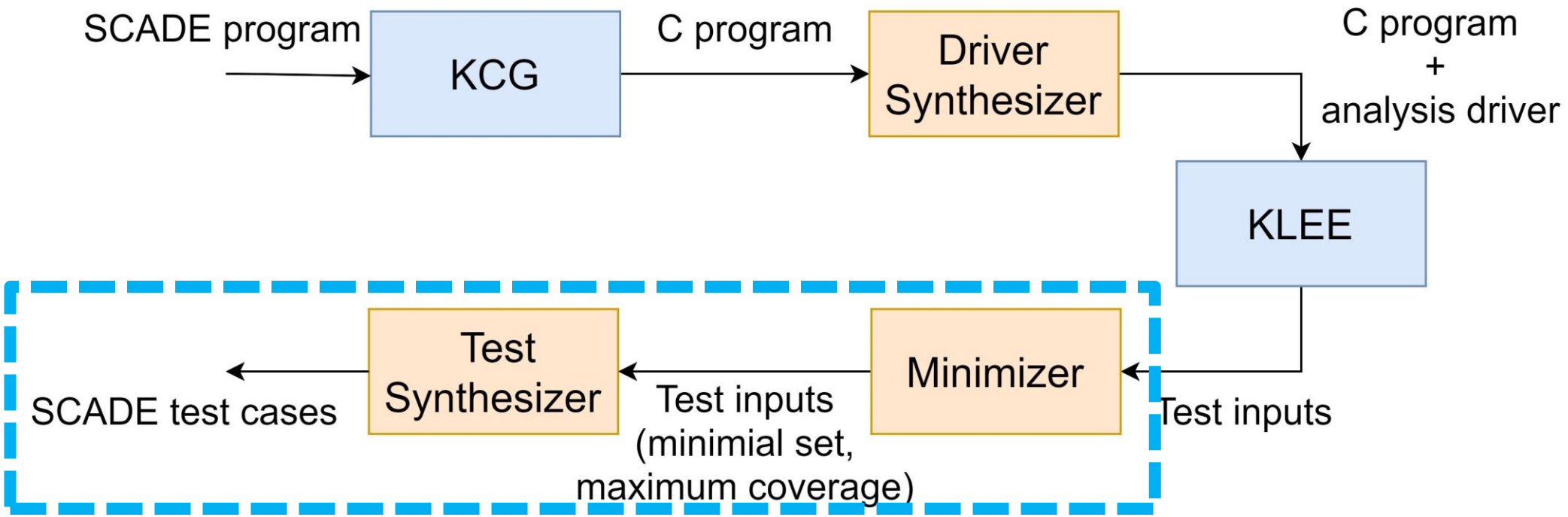
The Driver Synthesizer

- Unfolds all fields of data structures
- Call multiple times the target function to satisfy the “single-state-path-coverage” criterion
- Symbolically executes all paths of target function
- Provides the entry-function to KLEE

Name of the fresh symbol (field, type, sequence) field	type	seq	Test input	enum value
inC.ctrl	enum Lock	1	0	UNLOCKED
inC.wingMirrorData.automaticControl	boolean	1	false	-
inC.wingMirrorData.mirrorState[0]	enum MirrorState	1	0	OPEN
inC.wingMirrorData.mirrorState[1]	enum MirrorState	1	0	OPEN
outC.carState	enum Lock	1	0	UNLOCKED
outC.mirrorCommand[0]	enum MirrorState	1	0	OPEN
outC.mirrorCommand[1]	enum MirrorState	1	0	OPEN
inC.ctrl	enum Lock	2	1	LOCKED
inC.wingMirrorData.automaticControl	boolean	2	true	-
inC.wingMirrorData.mirrorState[0]	enum MirrorState	2	0	OPEN
inC.wingMirrorData.mirrorState[1]	enum MirrorState	2	0	OPEN
outC.carState	enum Lock	2	1	LOCKED
outC.mirrorCommand[0]	enum MirrorState	2	1	CLOSED
outC.mirrorCommand[1]	enum MirrorState	2	1	CLOSED

(a) The test inputs that KLEE generated for an execution path (through the analysis driver) for the sample SCADE program of Figure 1

TECS: Test Engine for Critical Software



Minimizer and Test Synthesizer

- Minimal subset of paths with maximum statement coverage
- Path selection task solved with a linear programming library [GLPK[1]]
- Test inputs from Minimizer -> SCADE (.sss)format

```
##########################################################
## WingMirrorControl_WingMirrorFSM, Test case: 00002
##########################################################

#Test step 1
SSM::set ctrl UNLOCKED
SSM::set wingMirrorData.automaticControl false
SSM::set wingMirrorData.mirrorState {(OPEN,OPEN)}
SSM::check carState UNLOCKED
SSM::check mirrorCommand {(OPEN, OPEN)}
SSM::cycle

#Test step 2
SSM::set ctrl LOCKED
SSM::set wingMirrorData.automaticControl true
SSM::set wingMirrorData.mirrorState {(OPEN, OPEN)}
SSM::check carState LOCKED
SSM::check mirrorCommand {(CLOSED, CLOSED)}
SSM::cycle
```

(b) The SCADE test case synthesized out of the test inputs from KLEE

[1] https://www.gnu.org/software/glpk/

AGENDA

- Goal
  - Exploit symbolic execution for test generation for safety critical software
- TECS: Test Engine for Critical Software
  - Tool to automatically generate test cases for SCADE programs
- Case study
  - On 15 SCADE programs
- Conclusions

Research Questions

RQ1: Does TECS accomplish test generation within acceptable test budgets?

- How many execution paths TECS analyses?
- How long does it take overall to complete the test generation process?

RQ2: Which is the quality of the test suites?

- How many test cases does it generate?
- How thorough are the test suites?
- How do test suites compare with manually derived test suites?

Subject Programs

- 15 programs out of the on-board signalling unit for high-speed rail that our industrial partner is currently developing

sorting railway vehicles into complete trains

To check the consistency of the data that the on-board unit receives from the ground components

Subject	SCADE Model				C code LOC
	#States	#Transistions	#Input	#Output	
shunting	5	10	12	14	646
dc_1	1	1	13	7	175
dc_2	1	1	1	2	43
dc_3	1	1	5	3	95
dc_4	1	1	3	4	62
dc_5	1	1	3	1	32
dc_6	1	1	3	4	67
dc_7	1	1	3	1	32
dc_8	1	1	2	1	30
dc_9	1	1	5	15	464
dc_10	1	1	3	9	239
dc_11	1	1	1	3	69
dc_12	1	1	14	17	96
dc_13	1	1	3	7	67
dc_14	1	1	1	1	35

Table: Statistics of the subject programs

Results

Does TECS accomplish test generation within acceptable test budgets?

Which is the quality of the test suites?

program	TECS			
	time(s)	#paths	#tests	cov_tests
shunting	321	3,367	20	82%
dc_1	0.263	120	8	91.86%
dc_2	0.023	4	2	80%
dc_3	0.086	16	4	100%
dc_4	0.08	5	2	95.45%
dc_5	0.042	4	2	89.29%
dc_6	0.064	3	2	92.65%
dc_7	0.058	4	2	100%
dc_8	0.039	4	2	100%
dc_9	0.082	9	8	89.29%
dc_10	0.461	87	6	100%
dc_11	0.047	3	1	100%
dc_12	0.061	4	2	70.59%
dc_13	0.076	20	4	100%
dc_14	0.053	4	2	86.36%

Table: Results of TECS for the subject programs considered in our case study

Results

Does TECS accomplish test generation within acceptable test budgets?

Which is the quality of the test suites?

program	TECS				manual test		
	time(s)	#paths	#tests	cov_tests	time(hr)	#tests	cov-tests
shunting	321	3,367	20	82%	80	15	94.55%
dc_1	0.263	120	8	91.86%	-	-	-
dc_2	0.023	4	2	80%	-	-	-
dc_3	0.086	16	4	100%	-	-	-
dc_4	0.08	5	2	95.45%	-	-	-
dc_5	0.042	4	2	89.29%	-	-	-
dc_6	0.064	3	2	92.65%	-	-	-
dc_7	0.058	4	2	100%	-	-	-
dc_8	0.039	4	2	100%	-	-	-
dc_9	0.082	9	8	89.29%	-	-	-
dc_10	0.461	87	6	100%	-	-	-
dc_11	0.047	3	1	100%	-	-	-
dc_12	0.061	4	2	70.59%	-	-	-
dc_13	0.076	20	4	100%	-	-	-
dc_14	0.053	4	2	86.36%	-	-	-

Table: Results of TECS for the subject programs considered in our case study

AGENDA

- Goal
  - Exploit symbolic execution for test generation for safety critical software
- TECS: Test Engine for Critical Software
  - Tool to automatically generate test cases for SCADE programs
- Case study
  - On 15 SCADE programs
- Conclusions

Conclusions

- We are studying the feasibility of an automated test generation approach
  - Based on symbolic execution
  - Specifically tailored on the characteristics of programming languages for safety-critical software systems
- We instantiated the proposed approach with the tool TECS
  - Test generator for programs written the SCADE language
- The initial case study suggest that the approach has good potential
- We achieved high model coverage for 15 safety-critical programs in SCADE
  - The test suites are of manageable size
  - The test suites compare well and are complementary with respect to manual test suites
- Work in progress
  - Extensive experiments in the testing phase of the project

Any Question ?

Elson Kurian: e.kurian@campus.unimib.it Giovanni Denaro: giovanni.denaro@unimib.it
Daniela Briola: daniela.briola@unimib.it Pietro Braione: pietro.braione@unimib.it

Thank You !!!