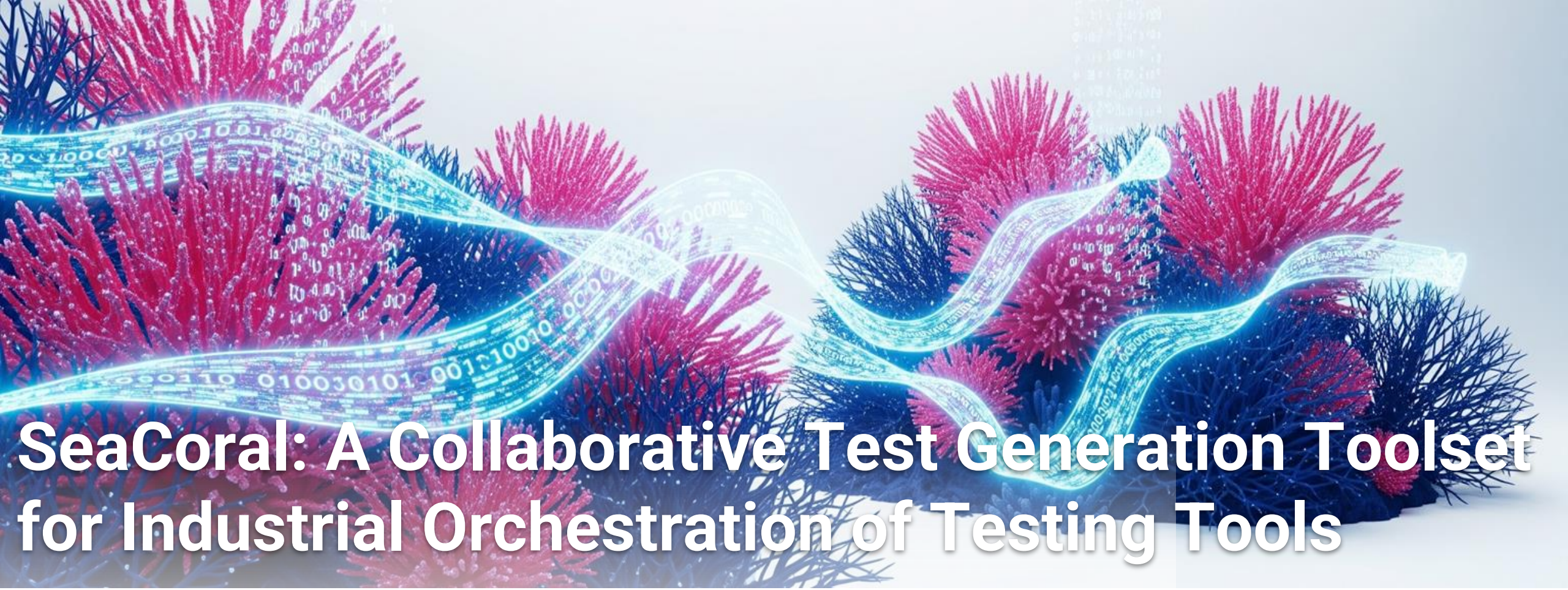


SeaCoral: A Collaborative Test Generation Toolset for Industrial Orchestration of Testing Tools

FM 2026 i-Day, Tokyo, May 20, 2026

Nicolas Berthier
Steven de Oliveira

Nikolai Kosmatov
Delphine Longuet

OCaml PRO

THALES

Work partially funded by ANR and the French Defence Innovation Agency

Context and Motivation

Needs of Thales operational entities

- Provide high **quality assurance** to stakeholders
- **Discover bugs** as early as possible
- Provide **test quality metrics**
- **Reduce costs** (development, testing, maintenance)

Systematic testing is known to be a good practice

- Yet expensive and tedious
- So often postponed (unless imposed by a standard or a client)

Automated test generation can be useful

Context and Motivation

A **wide range of testing and test generation tools** exist for C programs

- Commercial, open-source, academic
- Based on heuristics, fuzzing, model-checking, symbolic execution, mixed techniques
- Fast and efficient but incomplete, rigorous but slow

Industrial adoption **impaired** by several factors

- Each tool is **hard to master on its own**
- **Manual writing** of harnesses and drivers, especially for complex and dynamic data structures
- Need to **master the underlying techniques** to achieve efficient configuration
- Results in **tool-dependent formats** are not immediately usable in testing campaigns

Moreover

- Knowledge of a tool **does not transfer easily** to another tool
- **Different assumptions and coverage criteria definitions** hinder collaboration between tools

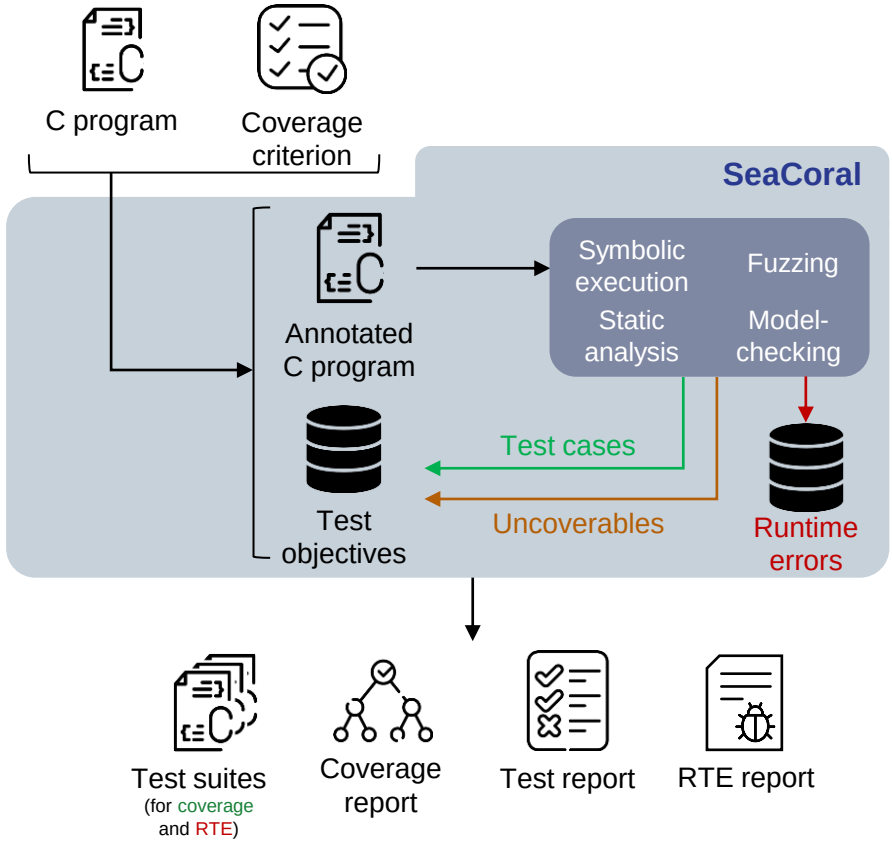
SeaCoral: Collaboration of Testing and Analysis Tools

An all-in-one platform...

- **Simplifying the access** to various test generation and static analysis tools
- **Enabling collaboration between the different tools** through shared test objectives and tests

... to automate test case generation for C programs

- **Code coverage**
  - Generation of **high-coverage test suites** along a chosen coverage criterion (statements, decisions, conditions, limits, mutations...)
  - Identification of **uncoverable code** (dead code, unused condition...)
- **Precise defect detection**
  - Identification of **runtime errors** (crashes, undefined behaviors)
  - Test reproducing the error -> **no false positive**



SeaCoral: Collaboration of Testing and Analysis Tools

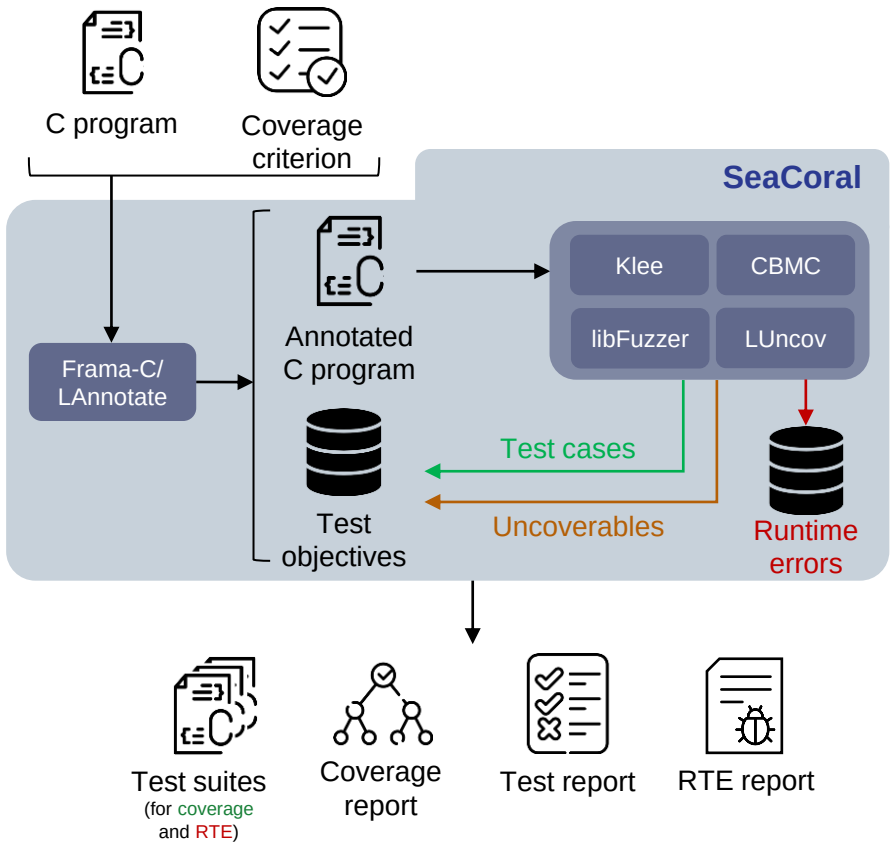
Pre-processing

- **Frama-C/LTest**: annotation by test objectives for a given coverage criterion

Analyzers for test case generation, uncoverability checking and bug finding

- **CBMC**: bounded model-checking
- **KLEE**: symbolic execution
- **libFuzzer**: gray-box fuzzing
- **LUncov**: abstract interpretation and deductive verification

Transparent use of all analyzers through a common interface



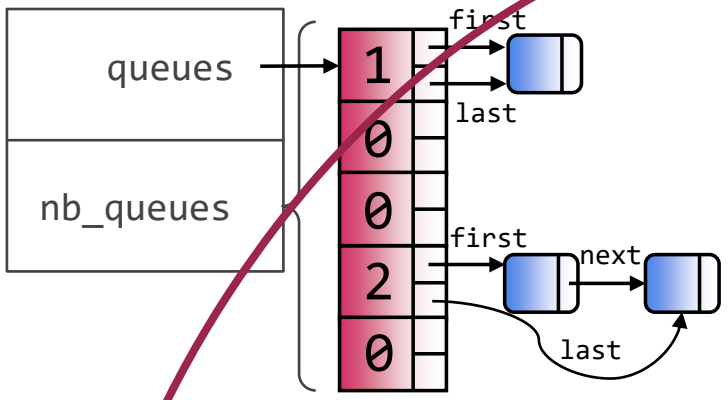
SeaCoral: Example Usage

```
typedef struct node_t {
  int data;
  struct node_t *next;
} node;

typedef struct queue_t {
  node *first;
  node *last;
  int length;
} queue;

typedef struct queues_t {
  int nb_queues;
  queue *queues;
} queues;

queues state;
```




```
int pop(int ind, int *val) {
  queue *q;
  node *n;

  sc_assume (state.nb_queues >= 1);
  if(!(0 <= ind && ind < state.nb_queues))
    return 0;

  q = &state.queues[ind];
  if(q->first == NULL)
    return 0;

  *val = q->first->data;
  n = q->first;

  if(q->first->next != NULL) {
    q->first = q->first->next;
  } else {
    q->first = NULL;
    q->last = NULL;
  }
  free(n);

  q->length--;
  return 1;
}
```

Pre-condition on the number of queues

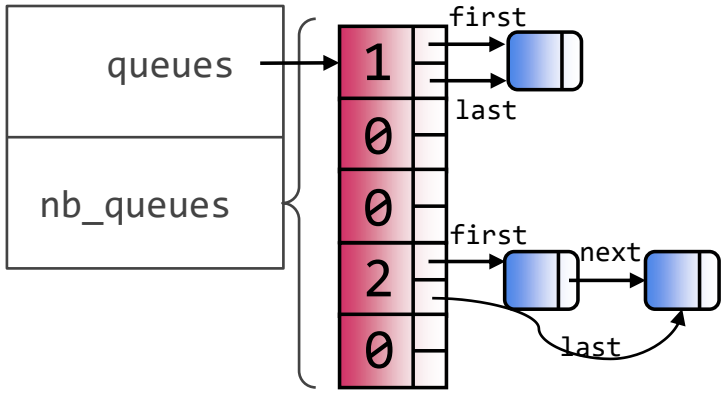
SeaCoral: Example Usage

```
typedef struct node_t {
  int data;
  struct node_t *next;
} node;

typedef struct queue_t {
  node *first;
  node *last;
  int length;
} queue;

typedef struct queues_t {
  int nb_queues;
  queue *queues;
} queues;

queues state;
```



Constraint on the size of the array pointed by the “queues” field of any structure of type `struct queues_t` is equal to field `nb_queues`

seacoral.toml

```
[project]
files = ["queues.c"]
criterion = "CC"
entrypoint = "pop"

[pointer-handling]
array-size-mapping = ["{struct queues_t}:queues:nb_queues"]
```

OCaml PRO THALES

SeaCoral: Example Usage

One `pc_label` annotation denotes one test objective

```
int pop(int ind, int *val)
{
  int __retres;
  queue *q;
  node *n;
  sc_assume(state.nb_queues >= 1);
  pc_label(0 <= ind,1,"CC");
  pc_label(! (0 <= ind),2,"CC");
  pc_label(ind < state.nb_queues,3,"CC");
  pc_label(! (ind < state.nb_queues),4,"CC");
  if (! (0 <= ind && ind < state.nb_queues)) {
    __retres = 0;
    goto return_label;
  }
  q = state.queues + ind;
  pc_label(q->first == (node *)0,5,"CC");
  pc_label(! (q->first == (node *)0),6,"CC");
  if (q->first == (node *)0) {
    __retres = 0;
    goto return_label;
  }
  *val = (q->first)->data;
  n = q->first;
  pc_label((q->first)->next != (struct node_t *)0,7,"CC");
  pc_label(! ((q->first)->next != (struct node_t *)0),8,"CC");
  if ((q->first)->next != (struct node_t *)0) q->first = (q->first)->next;
  else {
    q->first = (node *)0;
    q->last = (node *)0;
  }
  free((void *)n);
  (q->length) --;
  __retres = 1;
  return_label: return __retres;
}
```

SeaCoral: Example Usage

- First run with `libFuzzer`

```
nico@zinn:~/work/ocp/seacoral/etaps/queues_struct$ seacoral --config seacoral.toml --tools libfuzzer --libfuzzer-runs 10000
[0.01][A]{Sc} Starting to log into `_sc/queues.c-CC-@1/logs/1.log'
[0.01][A]{Sc} Initializing working environment...
[1.90][A]{Sc} Doing the hard work...
[2.46][A]{Sc} Launching libfuzzer on `pop'
[3.06][A]{Sc} Extracting new testcases from corpus...
[3.06][A]{Sc} Hard work done
[3.06][A]{Sc} Test 1: covering labels {2, 3}
[3.06][A]{Sc} Test 2: covering labels {1, 4}
[3.06][A]{Sc} Test 3: triggering heap-buffer-overflow at 0x4cd176
[3.06][A]{Sc} Coverage statistics for `pop': cov: 4 (50.0%) uncov: 0 (0.0%) unkwn: 4 (50.0%) with 2 tests
[3.06][A]{Sc} Covered labels: {1, 2, 3, 4}
[3.06][A]{Sc} Uncoverable labels: {}
[3.06][A]{Sc} Crash statistics: rte: 1 test
```

- SeaCoral's execution summary shows
  - The effect of each generated test (covered labels, RTE)
  - The number of covered test objectives (`cov`)
  - The number of test objectives that have been shown uncoverable (`uncov`)
  - The number of test objectives without a definitive verdict (`unkwn`)
  - The number of detected RTEs (`rte`)

SeaCoral: Example Usage

- Second run, with KLEE

```
nico@zinn:~/work/ocp/seacoral/etaps/queues_struct$ seacoral --config seacoral.toml --tools klee
[0.02][A]{Sc} Starting to log into `_sc/queues.c-CC-@2/logs/2.log'
[0.02][A]{Sc} Initializing working environment...
[0.72][A]{Sc} Current coverage statistics for `pop': cov: 4 (50.0%) uncov: 0 (0.0%) unkwn: 4 (50.0%) with 2 tests
                                                     rte: 1 test
[0.72][A]{Sc} Doing the hard work...
[0.85][A]{Sc} Launching klee on `pop'
[19.93][A]{Sc} Extracting new testcases from corpus...
[19.94][A]{Sc} Hard work done
[19.94][A]{Sc} Test 1: covering labels {2, 3}
[19.94][A]{Sc} Test 2: covering labels {1, 4}
[19.94][A]{Sc} Test 3: triggering heap-buffer-overflow at 0x4cd176
[19.94][A]{Sc} Test 4: covering labels {1, 3, 5}
[19.94][A]{Sc} Test 5: covering labels {1, 3, 6, 7}
[19.94][A]{Sc} Test 6: covering labels {1, 3, 6, 8}
[19.94][A]{Sc} Test 7: triggering invalid-memory-address at 0x4cd1f7
[19.94][A]{Sc} Test 8: triggering heap-buffer-overflow at 0x4cd1ef
[19.94][A]{Sc} Coverage statistics for `pop' cov: 8 (100.0%) uncov: 0 (0.0%) unkwn: 0 (0.0%) with 5 tests
[19.94][A]{Sc} Covered labels: {1, 2, 3, 4, 5, 6, 7, 8}
[19.94][A]{Sc} Uncoverable labels: {}
[19.94][A]{Sc} Crash statistics: rte: 3 tests
```

KLEE achieves complete coverage with 3 additional tests, and finds 2 more RTEs

SeaCoral: Example Usage

```
void test_1 () {
  /* Found by libfuzzer after 2s492 in run 1 */
  /* Outcome: covering labels {2, 3} */
  /* Globals, if any */
  state.queues = malloc (sizeof (struct queue_t[1]));
  state.queues[0].first = malloc (sizeof (struct node_t[0]));
  state.queues[0].last = malloc (sizeof (struct node_t[0]));
  state.queues[0].length = 1393819549;
  state.nb_queues = 1;
  /* Effective argument(s), if any */
  int (* val);
  int ind;
  ind = -1389542982;
  val = malloc (sizeof (int[0]));
  (void) pop (ind, val);
}
```

SeaCoral: Example Usage

```
void test_7 () {
  /* Found by klee after 19s828 in run 2 */
  /* Outcome: triggering invalid-memory-address at 0x4cd1f7 */
  /* Globals, if any */
  state.queues = malloc (sizeof (struct queue_t[1]));
  state.queues[0].first = malloc (sizeof (struct node_t[1]));
  state.queues[0].first[0].data = -1;
  state.queues[0].first[0].next = malloc (sizeof (struct node_t[1]));
  state.queues[0].first[0].next[0].data = 0;
  state.queues[0].first[0].next[0].next = NULL;
  state.queues[0].last = malloc (sizeof (struct node_t[0]));
  state.queues[0].length = -1;
  state.nb_queues = 1;
  /* Effective argument(s), if any */
  int (* val);
  int ind;
  ind = 0;
  val = NULL;
  (void) pop (ind, val);
}
```

Invariant violation

(`last` does not point to the last element & `length` is negative)

SeaCoral: Example Usage

```
void initialize (int nb_queues) {
  queue *pt;
  int cpt;

  state.queues = malloc (sizeof(struct queue_t) * nb_queues);
  state.nb_queues = nb_queues;

  for(cpt = 0; cpt < state.nb_queues; cpt++) {
    pt = &state.queues[cpt];
    pt->first = NULL;
    pt->last = NULL;
    pt->length = 0;
  }
}

void init_valid_state () {
  initialize (5);
  add (8,1);
  add (5,1);
  add (6,3);
}
```

An initialization function enables users to define complex initial states

seacoral.toml

```
[project]
files = ["queues.c"]
criterion = "CC"
entrypoint = "pop"

[pointer-handling]
array-size-mapping = ["{struct queues_t}:queues:nb_queues"]

[project]
ignored-globals = ["state"]

[fixtures]
init = "init_valid_state"
```

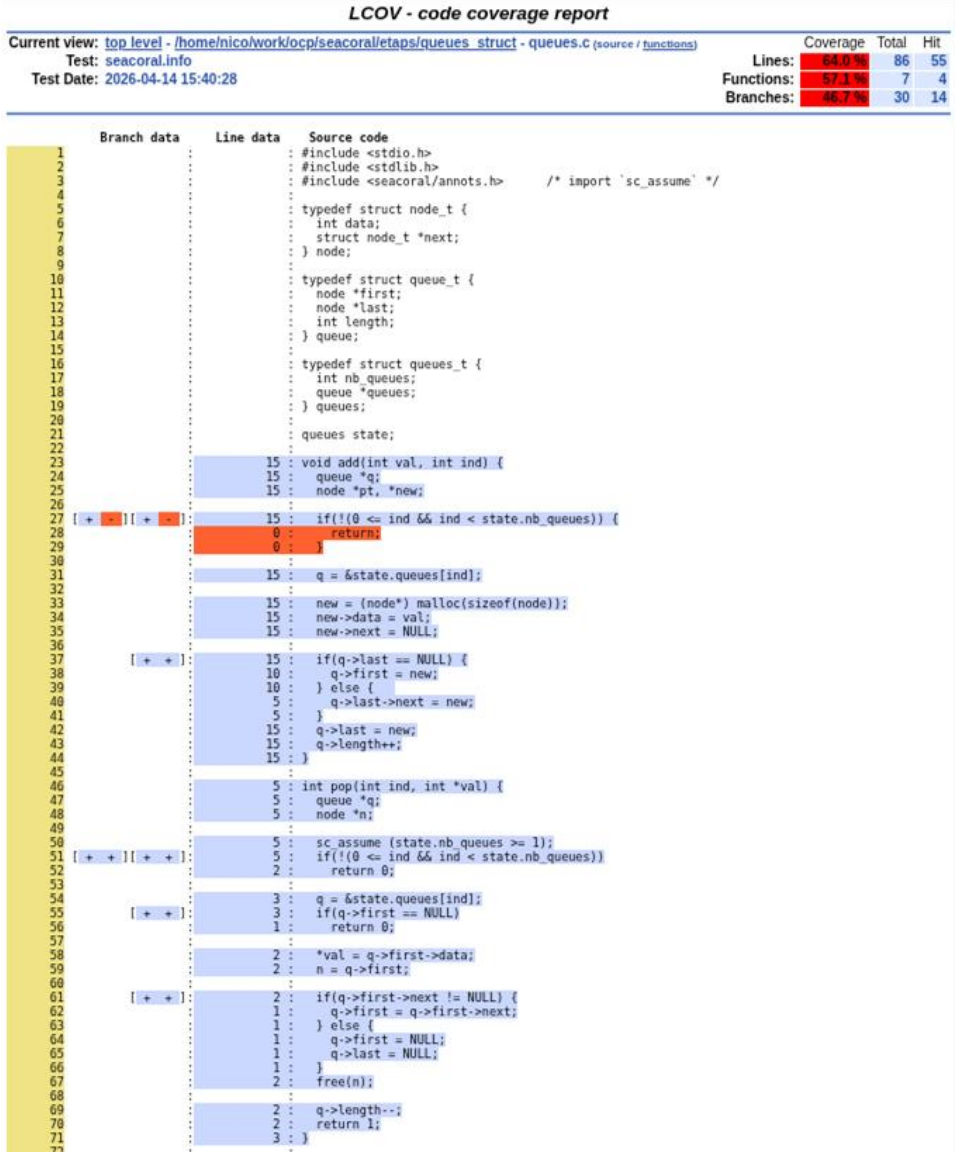
SeaCoral: Example Usage

- Starting again, with `KLEE` and `libFuzzer` launched in parallel

```
nico@zinn:~/work/ocp/seacoral/etaps/queues_struct$ seacoral --config seacoral.toml --tools klee,libfuzzer
[0.01][A]{Sc} Starting to log into `_sc/queues.c-CC-@3/logs/1.log'
[0.01][A]{Sc} Initializing working environment...
[1.98][A]{Sc} Doing the hard work...
[2.53][A]{Sc} Launching klee and libfuzzer in parallel on `pop'
[4.60][A]{Sc} Extracting new testcases from corpus...
[4.60][A]{Sc} Hard work done
[4.60][A]{Sc} Test 1: covering labels {1, 4}
[4.60][A]{Sc} Test 2: covering labels {2, 3}
[4.60][A]{Sc} Test 3: covering labels {1, 3, 5}
[4.60][A]{Sc} Test 4: triggering heap-buffer-overflow at 0x4cce2f
[4.60][A]{Sc} Test 5: covering labels {1, 3, 6, 7}
[4.60][A]{Sc} Test 6: covering labels {1, 3, 6, 8}
[4.60][A]{Sc} Test 7: triggering invalid-memory-address at 0x4cce37
[4.60][A]{Sc} Coverage statistics for `pop': cov: 8 (100.0%) uncov: 0 (0.0%) unkwn: 0 (0.0%) with 5 tests
[4.60][A]{Sc} Covered labels: {1, 2, 3, 4, 5, 6, 7, 8}
[4.60][A]{Sc} Uncoverable labels: {}
[4.60][A]{Sc} Crash statistics: rte: 2 tests
```

Other SeaCoral Features

- **Coverage of user-provided tests** taken into account
- **Test oracles** as C functions or E-ACSL post-conditions
- **LCOV coverage report**



LCOV - code coverage report

Current view: top level - /home/nico/work/ocp/seacoral/etaps/queues_struct - queues.c (source / functions)
Test: seacoral.info
Test Date: 2026-04-14 15:40:28

	Coverage	Total	Hit
Lines:	64.0 %	86	55
Functions:	57.1 %	7	4
Branches:	46.7 %	30	14

```
                Branch data     Line data    Source code
   1                    :                :  #include <stdio.h>
   2                    :                :  #include <stdlib.h>
   3                    :                :  #include <seacoral/annots.h>      /* import `sc_assume` */
   4                    :                :
   5                    :                :  typedef struct node_t {
   6                    :                :    int data;
   7                    :                :    struct node_t *next;
   8                    :                :  } node;
   9                    :                :
  10                    :                :  typedef struct queue_t {
  11                    :                :    node *first;
  12                    :                :    node *last;
  13                    :                :    int length;
  14                    :                :  } queue;
  15                    :                :
  16                    :                :  typedef struct queues_t {
  17                    :                :    int nb_queues;
  18                    :                :    queue *queues;
  19                    :                :  } queues;
  20                    :                :
  21                    :                :  queues state;
  22                    :                :
  23                    :               15 :  void add(int val, int ind) {
  24                    :               15 :    queue *q;
  25                    :               15 :    node *pt, *new;
  26                    :                :
  27 [ + - ][ + - ]     :               15 :    if(!(0 <= ind && ind < state.nb_queues)) {
  28                    :                0 :      return;
  29                    :                0 :    }
  30                    :                :
  31                    :               15 :    q = &state.queues[ind];
  32                    :                :
  33                    :               15 :    new = (node*) malloc(sizeof(node));
  34                    :               15 :    new->data = val;
  35                    :               15 :    new->next = NULL;
  36                    :                :
  37         [ + + ]    :               15 :    if(q->last == NULL) {
  38                    :               10 :      q->first = new;
  39                    :               10 :    } else {
  40                    :                5 :      q->last->next = new;
  41                    :                5 :    }
  42                    :               15 :    q->last = new;
  43                    :               15 :    q->length++;
  44                    :               15 :  }
  45                    :                :
  46                    :                5 :  int pop(int ind, int *val) {
  47                    :                5 :    queue *q;
  48                    :                5 :    node *n;
  49                    :                :
  50                    :                5 :    sc_assume (state.nb_queues >= 1);
  51 [ + + ][ + + ]     :                5 :    if(!(0 <= ind && ind < state.nb_queues))
  52                    :                2 :      return 0;
  53                    :                :
  54                    :                3 :    q = &state.queues[ind];
  55         [ + + ]    :                3 :    if(q->first == NULL)
  56                    :                1 :      return 0;
  57                    :                :
  58                    :                2 :    *val = q->first->data;
  59                    :                2 :    n = q->first;
  60                    :                :
  61         [ + + ]    :                2 :    if(q->first->next != NULL) {
  62                    :                1 :      q->first = q->first->next;
  63                    :                1 :    } else {
  64                    :                1 :      q->first = NULL;
  65                    :                1 :      q->last = NULL;
  66                    :                1 :    }
  67                    :                2 :    free(n);
  68                    :                :
  69                    :                2 :    q->length--;
  70                    :                2 :    return 1;
  71                    :                3 :  }
```

SeaCoral Strengths

Industrial adoption of test generation tools is **impaired** by several factors

- Each tool is **hard to master on its own** ✔
- **Manual writing** of harnesses and drivers, especially for complex and dynamic data structures ✔
- Need to **master the underlying techniques** to achieve efficient configuration ✔
- Results in **tool-dependent formats** are not immediately usable in testing campaigns ✔

Even more

- Knowledge of a tool **does not transfer easily** to another tool ✔
- **Different assumptions and coverage criteria definitions** ✔ hinder **collaboration** between tools ✔

- Unified Platform
- Automatically generated harnesses
- Few configuration levers
- Produces testsuites in C
- Shared test objectives and tests
- Sequential collaboration (reuse of generated tests)
- Parallel collaboration (dynamic sharing of tests)

SeaCoral Strengths (Continued)

An all-in-one platform... *(Recall)*

- **Simplifying the access** to various test generation and static analysis tools
- **Enabling collaboration between the different tools** through shared test objectives and tests

... to automate test case generation for C programs

- **Code coverage**
  - Generation of **high-coverage test suites** along a chosen coverage criterion (statements, decisions, conditions, limits, mutations...) ✓
  - Identification of **uncoverable code** (dead code, unused condition...) ✓
- **Precise defect detection**
  - Identification of **runtime errors** (crashes, undefined behaviors)
  - Test reproducing the error => **no false positive** ✓

- Analyzers can report uncoverable test objectives
  - CBMC in assert mode
  - LUncov
- RTEs reported by analyzers are validated and identified
  - A replay checks the occurrence of the RTE
  - Sanitizer reports are used for identification

Future Work

Add support for more C constructs

- Unions with pointers
- Finer-grained pointer constraints

Integration into testing processes

- Import of user-given tests as seeds for analyzers
- Export tests to exchange formats

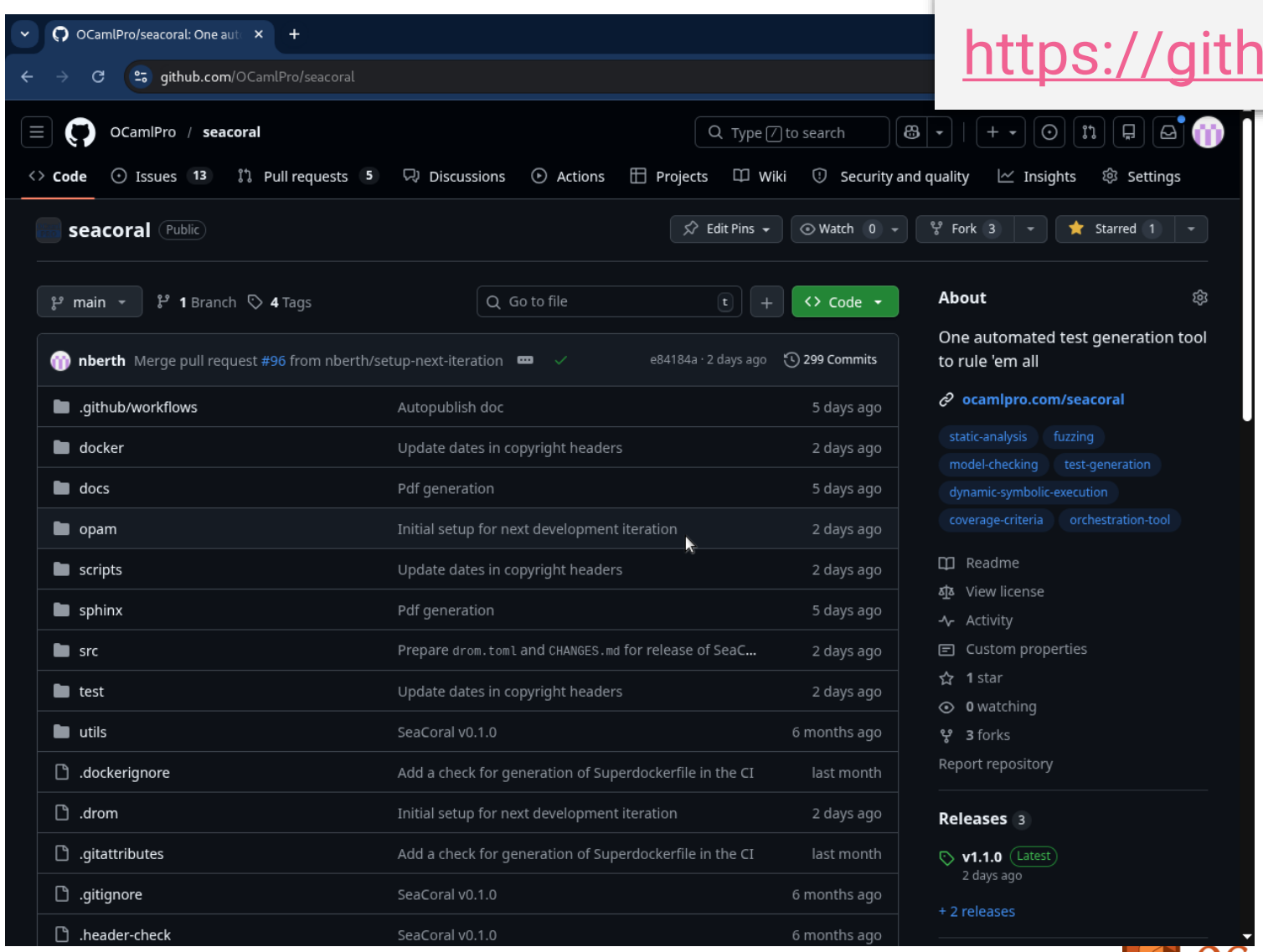
Scalability: **Improve parallel execution capabilities** thanks to incremental use of analyzers

Target the discovery of RTEs (e.g. via suitable annotations)

Integrate more analyzers (e.g. LLM-based, yours…)

SeaCoral is readily available

https://github.com/ocamlpro/seacoral



- Docker images
- Evaluation image with examples and detailed usage instructions