European Commission | Horizon 2020 European Union funding for Research & Innovation

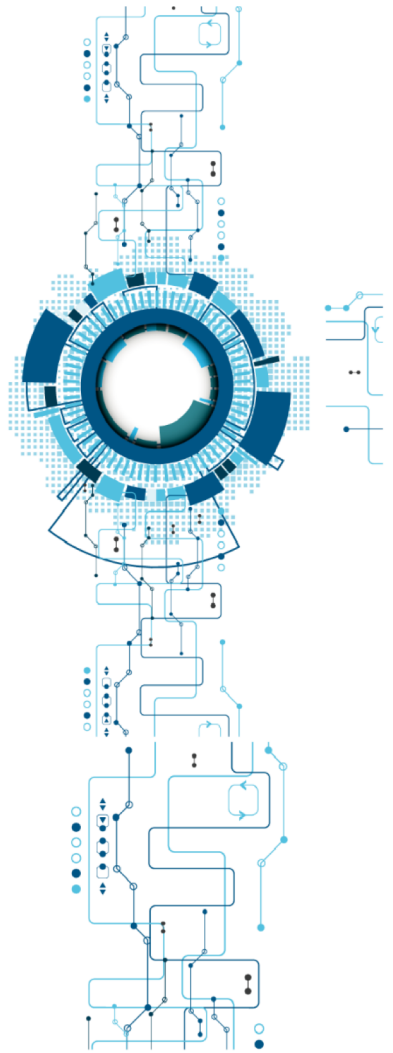
Hardware-assisted runtime verification in COEMS

EU COST IC1402 Meeting, 22./23. Mar. 2018

Svetlana Jakšić, Dan Li, **Volker Stolz**
HVL

Høgskulen på Vestlandet

ICT-10-2016 COEMS
Continuous Observation of Embedded Multicore Systems



ACCEMIC AIRBUS THALES Høgskulen på Vestlandet

Project Overview

Hardware-assisted tracing/monitoring:

- **Increase test efficiency**
  Automated online observation of embedded processors without time limitation and without intrusiveness simplifying the overall test process
- **Increase debug efficiency**
  New verification methodology automatically controls a multitude of constraints – a plurality of elusive, non-deterministically occurring failures can be detected.
- **Increase test effectivity**
  Efficient combination of functional tests, structural tests (both data and control flow) and runtime verification
- **Improve embedded systems performance**
  Detailed performance data and execution times, supporting resource optimization (power consumption, runtime, processing performance, deployment)

Applications

Finding Data Races

Finding Timing Bugs

Finding Functional Bugs

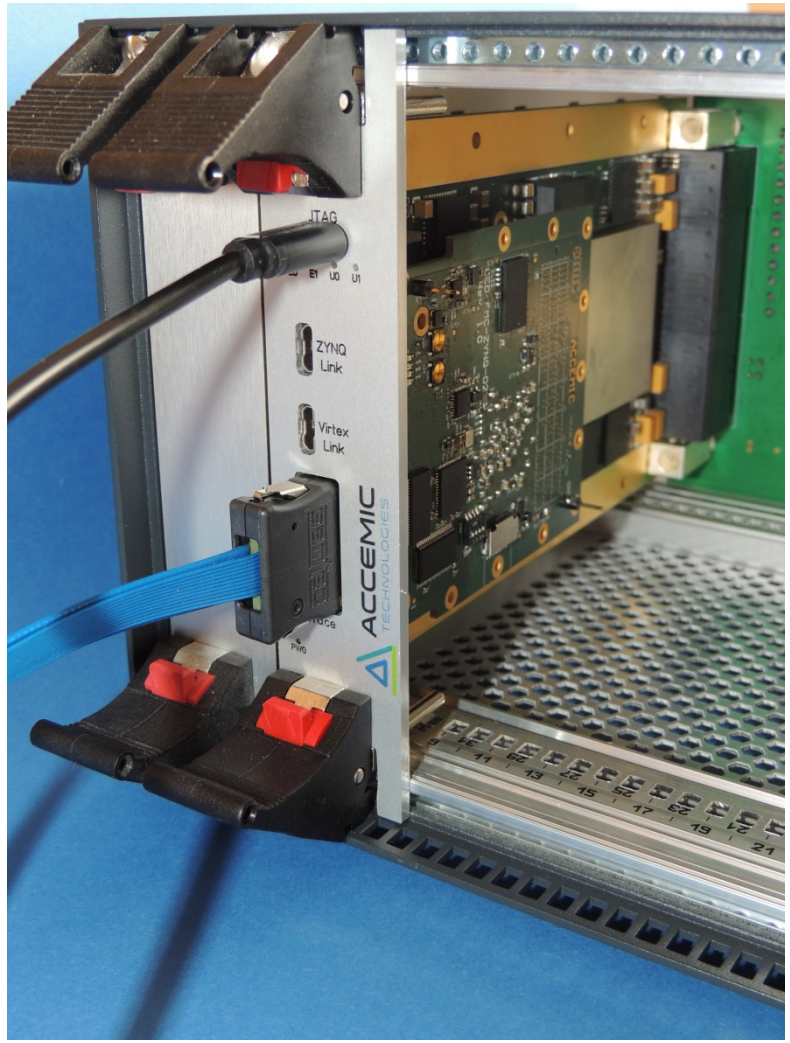
Measuring Coverage

- Execution Coverage
- Branch Coverage
- MC/DC Coverage

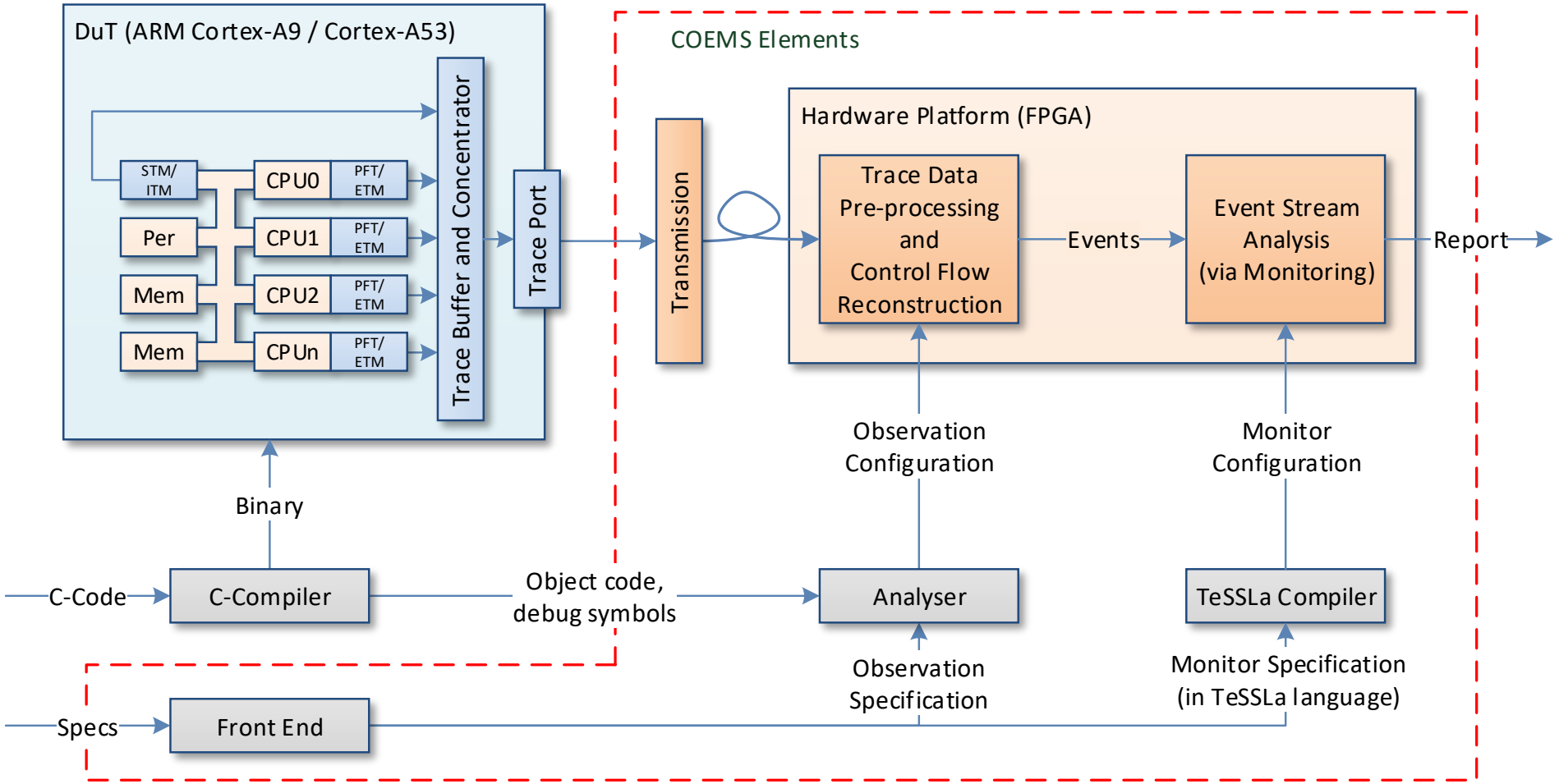
Measurement of Worst-Case Execution Time and Worst-Case Response Time

Hardware Platform

- Hardware
  - Virtex-7 series FPGA (available)
  - Zynq Ultrascale+ SOC (under development)
  - Interface to Aurora (Nexus, HSSTP)
  - VPX / FMC form factor
- Functionality
  - Online trace data processing
    (Coresight trace data -> event stream)
  - Supported architectures:
    ARM Cortex, some others
  - Online processing of event stream



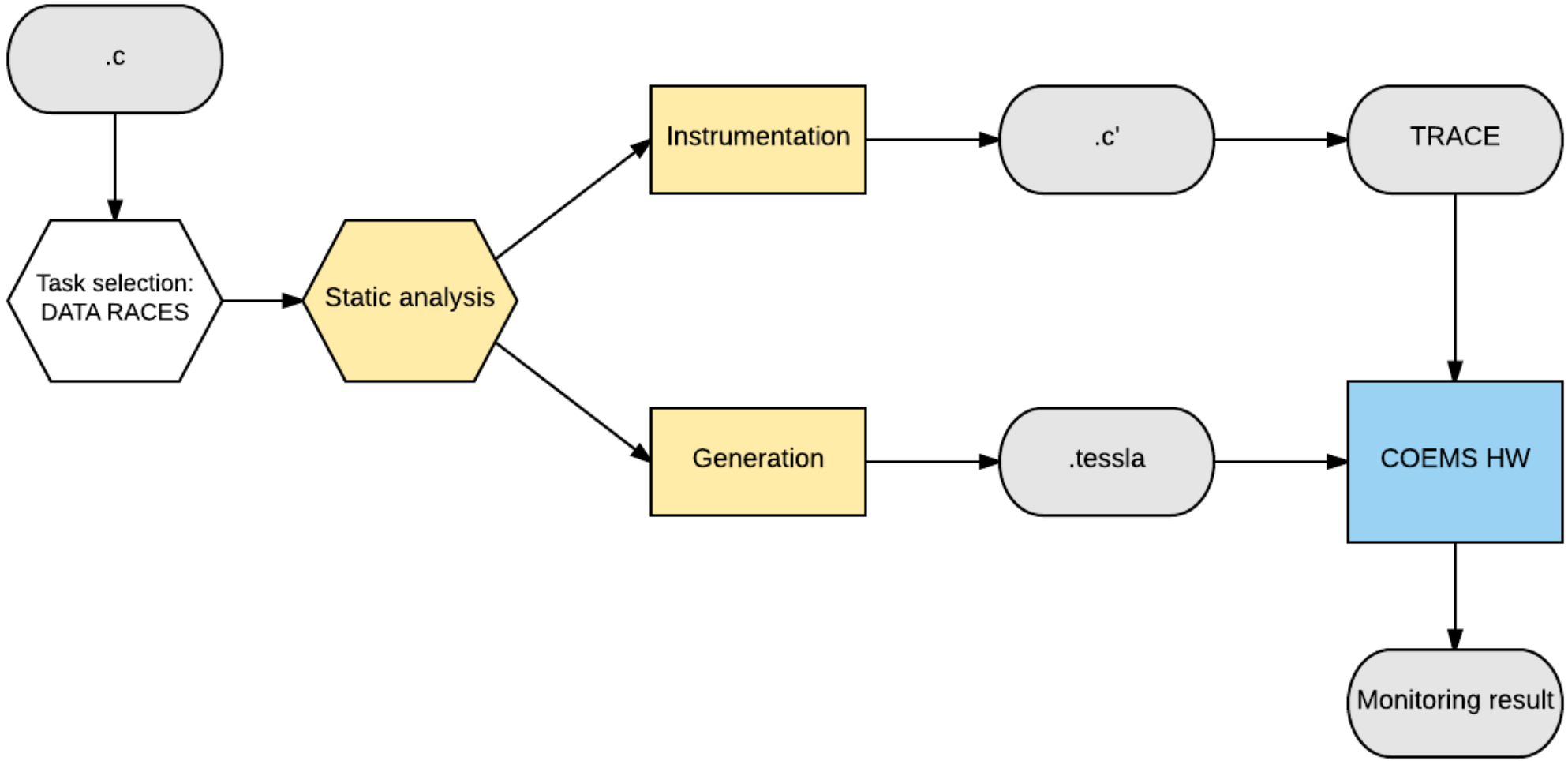
System Overview



Requirements from the industry partners

- Detection and optimization of shared resources
- Waiting time for a lock
- Performance analysis on functions – hot spots
- Bus and memory usage – bottlenecks
- Inter-thread communication
- Detection and optimization of thread migration and data exchange
- Lower cost of logging/instrumentation than `printf`
- Logging write accesses to selected variables
- Execution, branch and MC/DC coverage

Workflow


.c
Task selection:
DATA RACES
Static analysis
Instrumentation
.c'
TRACE
Generation
.tessla
COEMS HW
Monitoring result

Example – data race, no locks

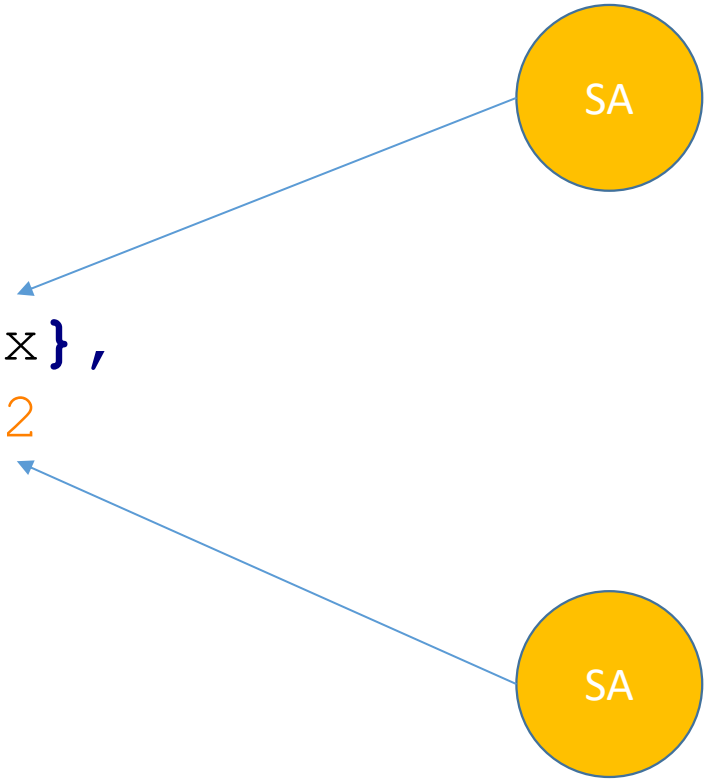
```c
#include <stdio.h>
#include <pthread.h>

int x = 0;

void* count(void *arg) {
    for ( int i = 0; i < 100; i++ ) {
        x++; // unprotected!
    }
    return NULL;
}

int main() {
    pthread_t p1, p2;
    pthread_create( &p1, NULL, count, NULL);
    pthread_create( &p2, NULL, count, NULL);
    // fight!
    pthread_join(p1, NULL);
    pthread_join(p2, NULL);
    printf( "Counted %d\n", x );
    return 0;
}
```

Data races in multi-threaded programs:

- two or more threads access the same memory location concurrently,
- at least one of the accesses is a write,
- the threads are not using any exclusive locks to control their accesses to that memory.

Example – data race, no locks

```
dataRace(
shared variables = {x},
number of threads = 2
)
```



TeSSLa high-level specification

Example – data race, no locks

```
def two_threads_read := T1_reads_x == () && T2_reads_x == ()
def a_thread_writes := merge(T1_writes_x,T2_writes_x) == ()
def dataRace := if two_threads_read && a_thread_writes then ()
out dataRace
def dataRace_in_line := if (time(dataRace)==time(line)) then line
out dataRace_in_line
```

```
def time_read_x := time(merge(T1_reads_x,T2_reads_x))
def time_write_x := time(merge(T1_writes_x,T2_writes_x))
def badInterleave := if last(time_read_x,time_read_x) > default(time_write_x,0) then ()
out badInterleave
```

Example – data race, no locks


```
1674914046544: instruction = "load"
1674914046544: function = "count"
1674914046544: line = 8
1674914046544: column = 10
1674914046544: threadid = 140592004323072
1674914046544: readvar = "x"
1674914046544: readvaraddr = 6299740
1674914048890: instruction = "add"
1674914048890: function = "count"
1674914048890: line = 8
1674914048890: column = 10
1674914048890: threadid = 140592004323072
1674914050773: instruction = "store"
1674914050773: function = "count"
1674914050773: line = 8
1674914050773: column = 10
1674914050773: threadid = 140592004323072
1674914050773: writevar = "x"
1674914050773: writevaraddr = 6299740
```

Example – data race, no locks

.tessla

```
def two_threads_read := T1_reads_x == () && T2_reads_x == ()
def a_thread_writes := merge(T1_writes_x,T2_writes_x) == ()
def dataRace := if two_threads_read && a_thread_writes then ()
out dataRace
def dataRace_in_line := if (time(dataRace)==time(line)) then line
out dataRace_in_line
```

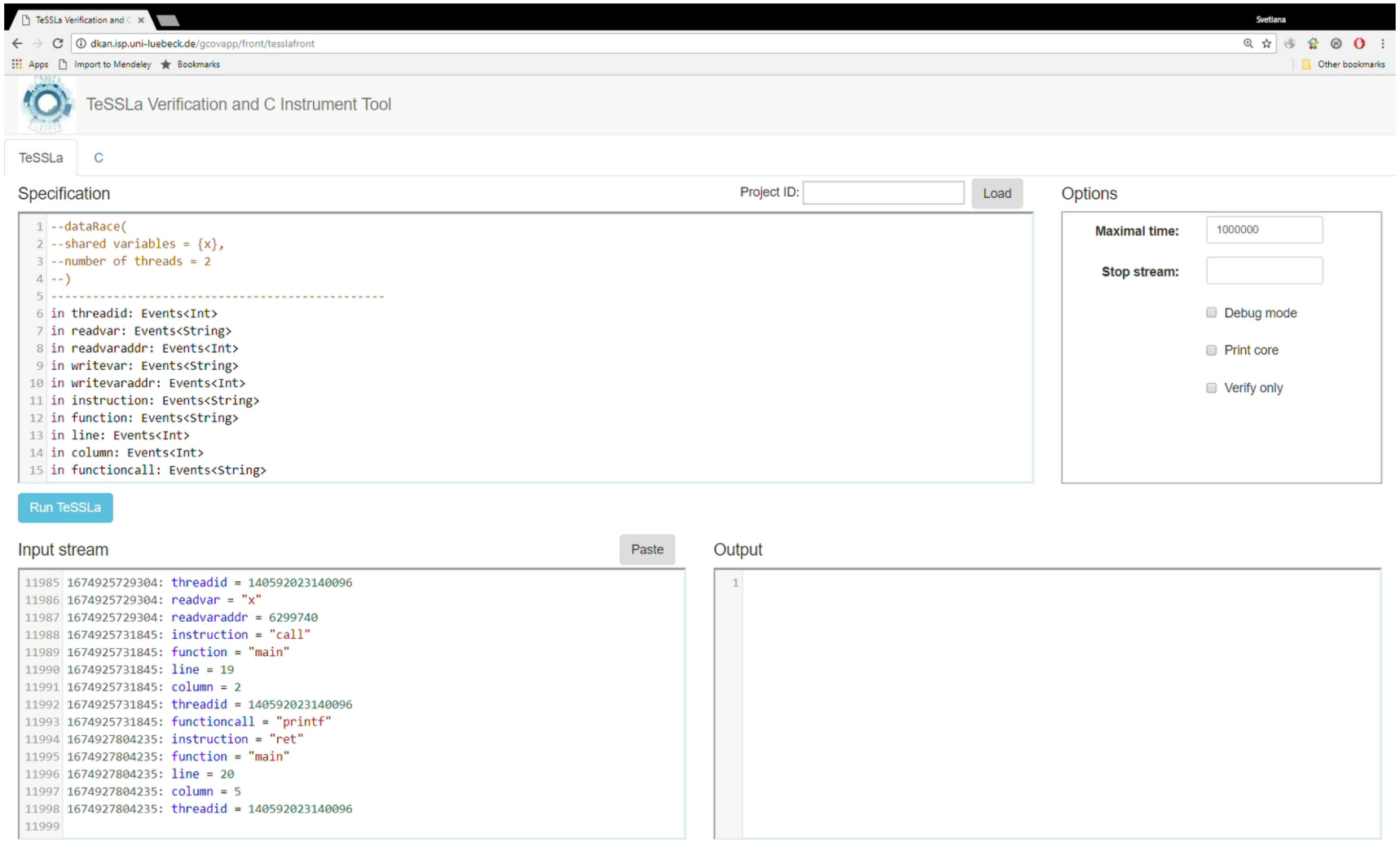
TRACE

```
1674914046544: instruction = "load"
1674914046544: function = "count"
1674914046544: line = 8
1674914046544: column = 10
1674914046544: threadid = 140592004323072
1674914046544: readvar = "x"
1674914046544: readvaraddr = 6299740
1674914048890: instruction = "add"
1674914048890: function = "count"
1674914048890: line = 8
1674914048890: column = 10
1674914048890: threadid = 140592004323072
```

COEMS HW

```
1749594547010: T1_reads_x = ()
1749594550976: T1_writes_x = ()
1749594571346: T1_reads_x = ()
1749594575683: T1_writes_x = ()
1749595268897: T2_reads_x = ()
1749595268897: dataRace_in_line = 8
1749595268897: dataRace = ()
1749595273268: T2_writes_x = ()
```

Demo


TeSSLa Verification and C
Svetlana
dkan.isp.uni-luebeck.de/gcovapp/front/tesslafront
Apps
Import to Mendeley
Bookmarks
Other bookmarks
TeSSLa Verification and C Instrument Tool
TeSSLa
C
Specification
Project ID:
Load
Options
--dataRace(
--shared variables = {x},
--number of threads = 2
--)
in threadid: Events<Int>
in readvar: Events<String>
in readvaraddr: Events<Int>
in writevar: Events<String>
in writevaraddr: Events<Int>
in instruction: Events<String>
in function: Events<String>
in line: Events<Int>
in column: Events<Int>
in functioncall: Events<String>
Maximal time:
1000000
Stop stream:
Debug mode
Print core
Verify only
Run TeSSLa
Input stream
Paste
Output
1674925729304: threadid = 140592023140096
1674925729304: readvar = "x"
1674925729304: readvaraddr = 6299740
1674925731845: instruction = "call"
1674925731845: function = "main"
1674925731845: line = 19
1674925731845: column = 2
1674925731845: threadid = 140592023140096
1674925731845: functioncall = "printf"
1674927804235: instruction = "ret"
1674927804235: function = "main"
1674927804235: line = 20
1674927804235: column = 5
1674927804235: threadid = 140592023140096

Example – data race, with locks

```
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t m;
int x=0;
pthread_mutex_t l;
int y=0;

void* f(void *arg) {
   for ( int i = 0; i < 10; i++ ) {
   pthread_mutex_lock(&m);
      x++;
   pthread_mutex_unlock(&m);
   }
   return NULL;
}
void* g(void *arg) {
   for ( int i = 0; i < 10; i++ ) {
      pthread_mutex_lock(&l);
      y++;
      x++;
      pthread_mutex_unlock(&l);
   }
   return NULL;
}
```

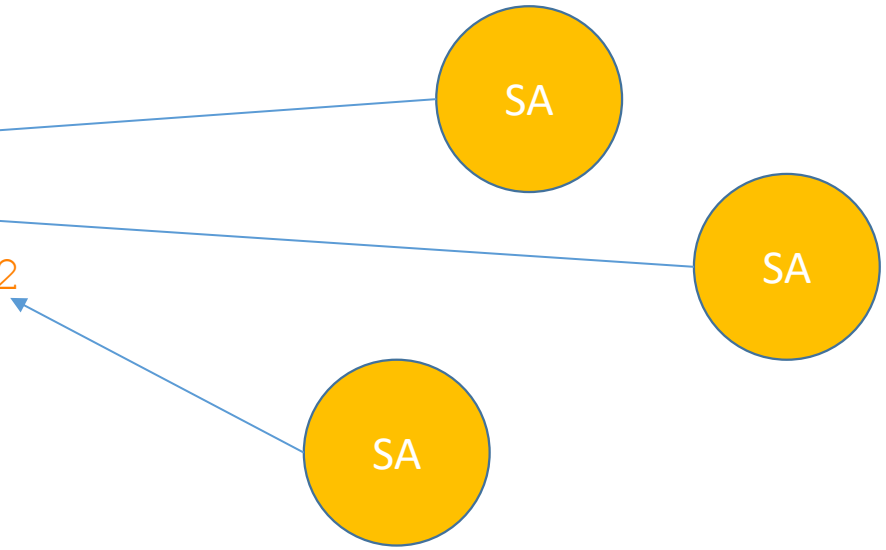
```
int main() {
    pthread_t p1;
    pthread_t p2;
    pthread_mutex_init( &m, NULL );
    pthread_mutex_init( &l, NULL );
    pthread_create( &p1, NULL, f, NULL );
    pthread_create( &p2, NULL, g, NULL );
    pthread_join(p1, NULL);
    pthread_join(p2, NULL);
    printf( "x= %d\n", x );
    printf( "y= %d\n", y );
    return 1;
}
```

Lockset algorithm (Eraser) enforces

- that every shared variable is always protected by some set of locks
- this set of locks must be held by any thread accessing the variable

Example – data race, with locks

```
def correctLocking(
shared variables = {x,y}
total number of locks = 2
total number of threads = 2
)
```




```
def lock_1_always_protects_1 := T1_protecting_1_with_1 && T2_protecting_1_with_1
out lock_1_always_protects_1

def lock_2_always_protects_1 := T1_protecting_1_with_2 && T2_protecting_1_with_2
out lock_2_always_protects_1

def error_1 := !(lock_1_always_protects_1 || lock_1_always_protects_1)
```

Example – data race, with locks


```
1980123194759: instruction = "call"
1980123194759: function = "g"
1980123194759: line = 21
1980123194759: column = 9
1980123194759: threadid = 140338129528576
1980123194759: functioncall = "pthread_mutex_lock"
1980123194759: mutexlock = "l"
1980123194759: mutexlockaddr = 6303872
1980123201358: instruction = "load"
1980123201358: function = "g"
1980123201358: line = 22
1980123201358: column = 10
1980123201358: threadid = 140338129528576
1980123201358: readvar = "y"
1980123201358: readvaraddr = 6303864
1980123204027: instruction = "add"
1980123204027: function = "g"
1980123204027: line = 22
1980123204027: column = 10
1980123204027: threadid = 140338129528576
```

Example – data race, with locks

.tessla

```
def lock_1_always_protects_1 := T1_protecting_1_with_1 &&
T2_protecting_1_with_1
out lock_1_always_protects_1

def lock_2_always_protects_1 := T1_protecting_1_with_2 &&
T2_protecting_1_with_2
out lock_2_always_protects_1

def error_1 := !(lock_1_always_protects_1 || lock_1_always_protects_1)
```

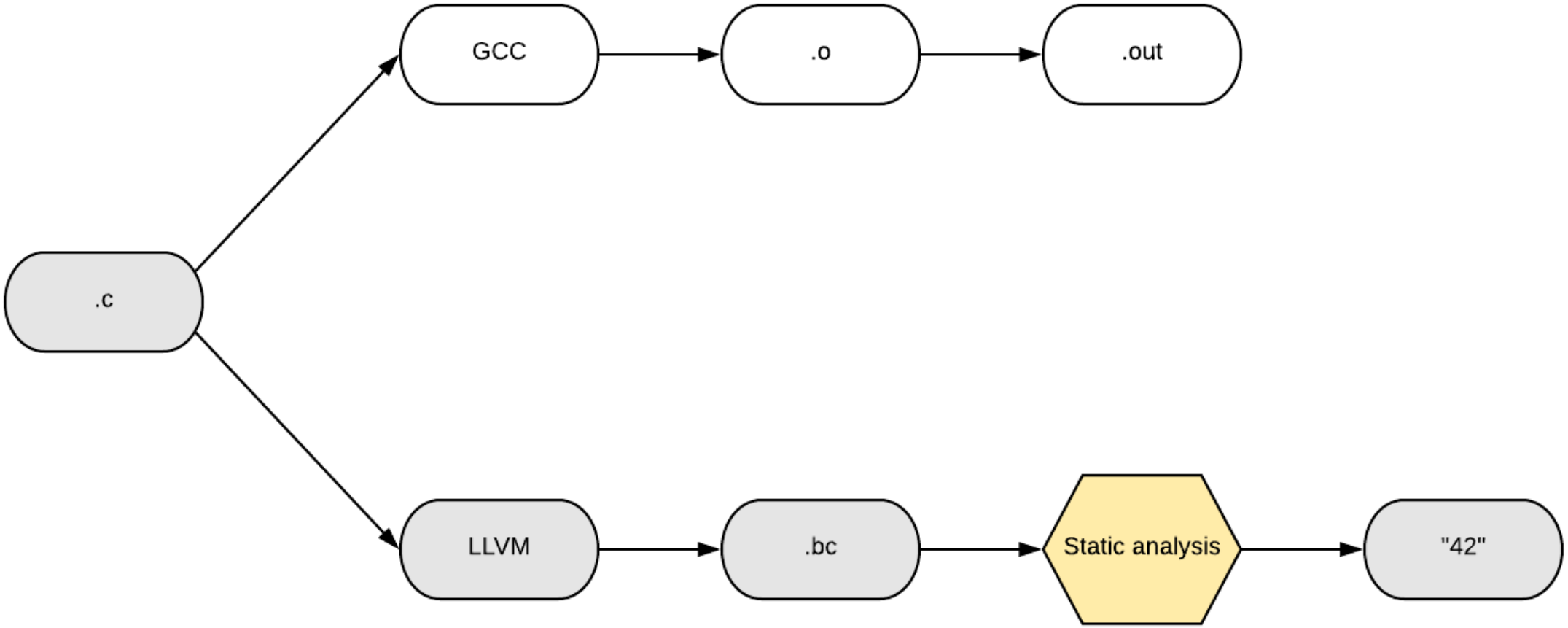
TRACE

```
1980123194759: instruction = "call"
1980123194759: function = "g"
1980123194759: line = 21
1980123194759: column = 9
1980123194759: threadid = 140338129528576
1980123194759: functioncall = "pthread_mutex_lock"
1980123194759: mutexlock = "l"
1980123194759: mutexlockaddr = 6303872
1980123201358: instruction = "load"
1980123201358: function = "g"
1980123201358: line = 22
```

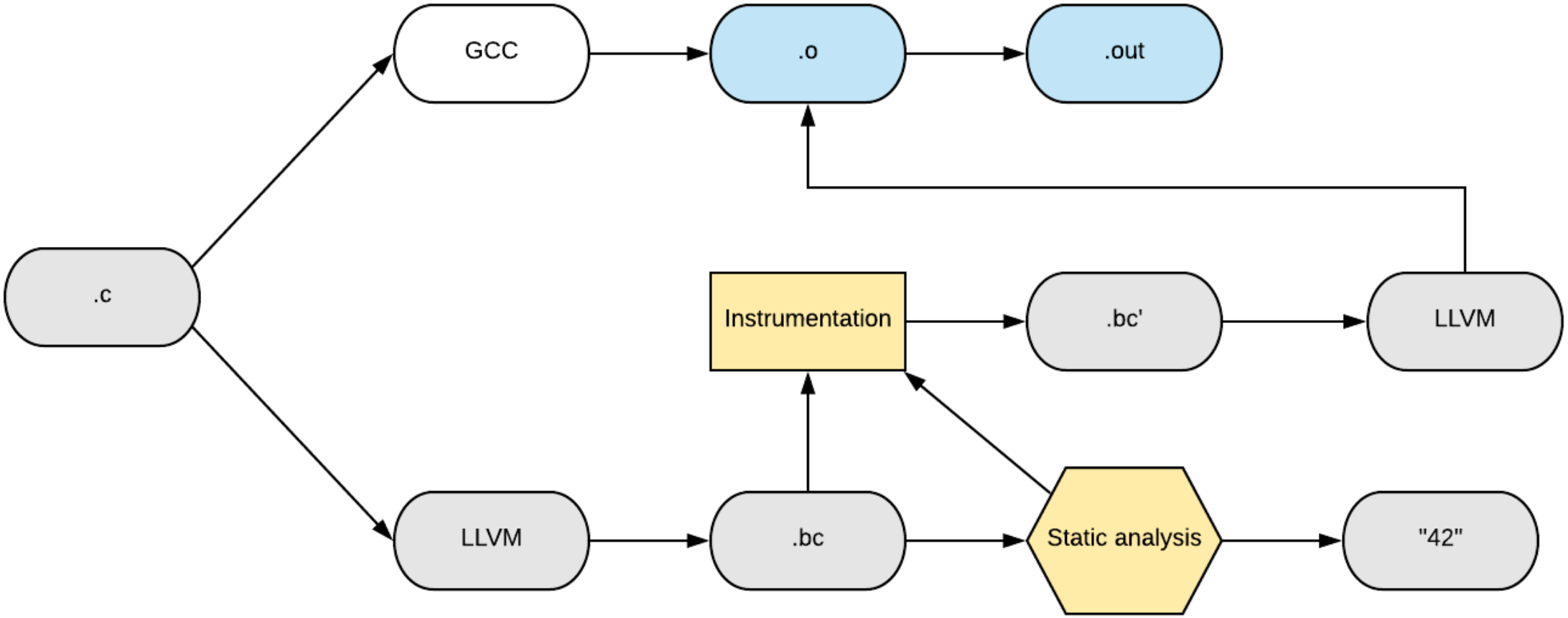
COEMS HW

```
1980123526343: T1_access_2 = ()
1980123526343: lock_2_always_protects_2 = false
1980123526343: lock_1_always_protects_2 = true
1980123625398: T2_access_2 = ()
1980123625398: lock_2_always_protects_2 = false
1980123625398: lock_1_always_protects_2 = false
1980123625398: error_in_line = 13
1980123632803: T2_access_2 = ()
1980123632803: lock_2_always_protects_2 = false
1980123632803: lock_1_always_protects_2 = false
1980123632803: error_in_line = 13
1980123654293: T2_access_2 = ()
1980123654293: lock_2_always_protects_2 = false
1980123654293: lock_1_always_protects_2 = false
```

Workflow



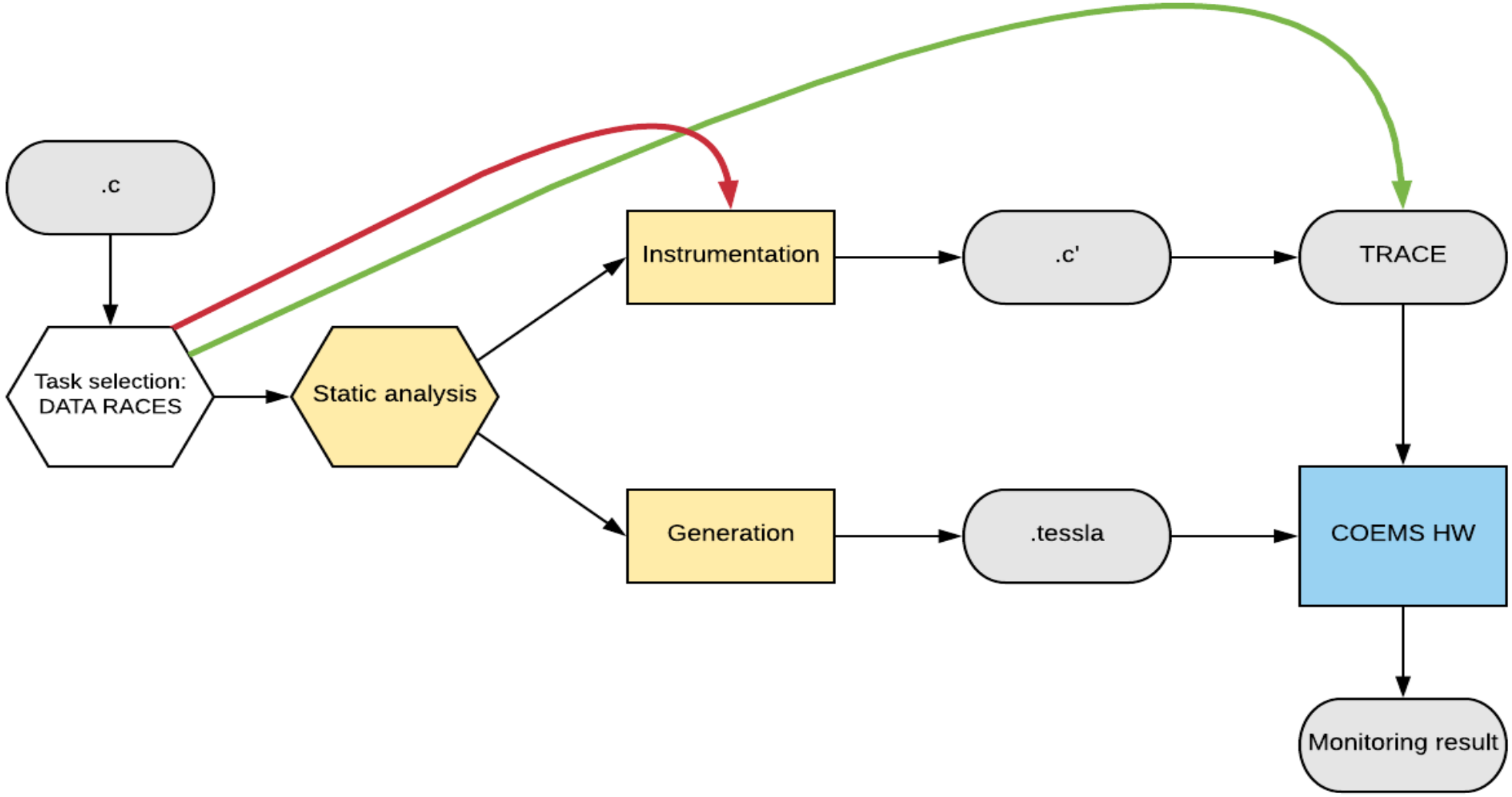
Workflow


GCC
.o
.out
.c
Instrumentation
.bc'
LLVM
LLVM
.bc
Static analysis
"42"

Counting threads and locks

- Counting
  - Number of threads that can be created
  - Number of locks that can be initialized
- Standard static analysis:
  Flemming Nielson, Hanne Riis Nielson, and Chris Hankin:
  Principles of program analysis. Springer, 1999.
- Control flow graph, basic blocks, lattice, transition function
- Never underestimate!
- Examples
- LLVM IR, C++ implementation

Workflow


.c
Task selection:
DATA RACES
Static analysis
Instrumentation
Generation
.c'
.tessla
TRACE
COEMS HW
Monitoring result

Limitations and optimizations

- Trace is limited with respect to data
  example: *which* lock is taken?
- Need ITM-instructions to log additional data
  (“cheap" logging into trace buffer)
- Avoid instrumentation when not necessary
  - On constant variables
  - On variables that are never written to
  - Collapsing multiple accesses into one
  - Using aliasing/escape analysis
  - Variable accesses that are clear from the assembly code?

Without Instrumentation

- We can use static analysis for lock operations
- Over-approximation of locks in *Pun/Steffen/Stolz: Effect-polymorphic behaviour inference for deadlock checking, 2016*
- Should work for both source- and object code.
- Warnings with low number of false positives

Conclusion

- Related work: static and dynamic race detectors; ThreadSanitizer, ERASER
- Static analysis-component focused on *races*
- Can be reused for other purposes (coverage, …)
- Beyond COEMS
  - What can be done in no-hardware/no-instrumentation setting?
  - Some expertise on IntelPT now
  - Static analysis results on binary(!) carry over to both IntelPT and CoreSight
- Publications:
  - T.Scheffel *et al.*: “Rapidly Adjustable Non-Intrusive Online Monitoring for Multi-core Systems”, SBFM 2017
  - S. Jakšić *et al.:* “COEMS — open traces from the industry”, RV-CUBES 2017