RTOS-Independent Interaction Analysis in ARA

Gerion Entrup, Jan Neugebauer, Daniel Lohmann

Leibniz Universität Hannover

July 05, 2022
Building an RTOS-Independent RTOS-Aware Analyzer

Where do we stand? We have a Whole-System-Compiler:

- Make the system startup faster: Make dynamic OS-object instantiations static.
- Eliminate RTOS-time: Drop unnecessary syscalls or calculations.
Building an RTOS-Independent RTOS-Aware Analyzer

Where do we stand? We have a Whole-System-Compiler:

- Make the system startup faster: Make dynamic OS-object instantiations static.
- Eliminate RTOS-time: Drop unnecessary syscalls or calculations.

**LUH RTOS-Independent Interaction Analysis in ARA – Motivation**
Building an RTOS-Independent RTOS-Aware Analyzer

Application

```
TaskHandle_t h = NULL;
int main() {
    xTaskCreate(vTask1, "Task1", NULL);
    xTaskCreate(vTask2, "Task2", &h);
    vTaskStartScheduler();
    // should never reach this
    while (1);
    return 0;
}
void vTask1(void* param) {
    while(1) {
        do_stuff();
        vTaskDelay(100);
    }
}
void vTask2(void* param) {
    do_long_operation();
    xTaskDelete(h)
}
```

ARA

Analysis

FreeRTOS
AUTOSAR
Zephyr
POSIX
RTOS API
uses
SSE
SIA
Building an RTOS-Independent RTOS-Aware Analyzer

```c
Application

TaskHandle_t h = NULL;
int main()
{
xTaskCreate(vTask1, "Task1",

  vTaskCreate(vTask2, "Task2", &h), ...

  vTaskDelay(100);

  xTaskDelete(h);

  ...
}

ARA

Analysis

RTOS API

FreeRTOS

AUTOSAR

Zephyr

POSIX

RTOS uses

SSE

SIA

OS Model

LUH RTOS-Independent Interaction Analysis in ARA – Motivation

3–13
Building an RTOS-Independent RTOS-Aware Analyzer

RTOS API uses

TaskHandle_t h = NULL;

int main()
{
xTaskCreate()

vTaskDelay()

xTaskDelete()

...

xTaskDelete(h)
}

Analysis

ARA

FreeRTOS

AUTOSAR

Zephyr

POSIX

RTOS API uses

SSE

SIA

OS Model
Building an RTOS-Independent RTOS-Aware Analyzer

RTOS API
- FreeRTOS
- AUTOSAR
- Zephyr
- POSIX

ARA

Analysis

FreeRTOS
AUTOSAR
Zephyr
POSIX
Building an RTOS-Independent RTOS-Aware Analyzer

```c
int main() {
    xTaskCreate(vTask1, "Task1", NULL);
    xTaskCreate(vTask2, "Task2", &h,...
    artScheduler();
    // should never reach this
    return 0;
}

void vTask1(void) {
    while(1) {
        do_stuff();
        vTaskDelay(100);
    }
}

void vTask2(void) {
    do_long_operation();
    xTaskDelete(h);
}
```

ABA

FreeRTOS
AUTOSAR
Zephyr
POSIX

RTOS API

SSE
SIA

OS Model

LUH RTOS-Independent Interaction Analysis in ARA – Motivation
Building an RTOS-Independent RTOS-Aware Analyzer

RTOS API

FreeRTOS

AUTOSAR

Zephyr

POSIX

ARA

SIA

SSE

...
Building an RTOS-Independent RTOS-Aware Analyzer

Application

TaskHandle_t h = NULL; int main() { xTaskCreate(vTask1, "Task1", NULL); xTaskCreate(vTask2, "Task2", &h); // should never reach this while(1); return 0; }

void vTask1(void *param) {
    while(1) {
        do_stuff();
        vTaskDelay(100);
    }
}

void vTask2(void *param) {
    do_long_operation();
    xTaskDelete(h);
}

Analysis

ARA

xTaskCreate();
xTaskDelay();
xTaskDelete();

FreeRTOS

AUTOSAR

Zephyr

POSIX

RTOS API

OS Model

SSE

SIA

. . .
Building an RTOS-Independent RTOS-Aware Analyzer

RTOS API
- FreeRTOS
- AUTOSAR
- Zephyr
- POSIX

OS Model

ARA
- SIA
- SSE

- RTOS API
- Analyses Requirements
- Our Interface
- Case Studies
- Conclusion
Know your RTOS: How dynamic is the System?

<table>
<thead>
<tr>
<th>instantiation</th>
<th>static</th>
<th>dyn</th>
<th>config</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOSAR</td>
<td>✓</td>
<td></td>
<td>DSL</td>
</tr>
<tr>
<td>FreeRTOS</td>
<td>✓</td>
<td>✓</td>
<td>CPP-Macros</td>
</tr>
<tr>
<td>Zephyr</td>
<td>✓</td>
<td>✓</td>
<td>KConfig</td>
</tr>
<tr>
<td>POSIX</td>
<td>✓</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

```cpp
TaskHandle_t t1, t2;
QueueHandle_t q1;
struct Message {...};

void create(Function f, int prio) {
    xTaskCreate("T " f.name(), f, prio);
}

int main() {
    t1 = create(task_1, 1);
    t2 = create(task_2, 2);
    q1 = xQueueCreate(5, sizeof(Message));
    vTaskStartScheduler();
}

task_1 {
    while(true) {
        Message m = produce();
        xQueueSend(q1, m);
    }
}

task_2 {
    Message m;
    while(true) {
        xQueueReceive(q1, &m);
        consume(m);
    }
}
```
Know your RTOS: How dynamic is the System?

<table>
<thead>
<tr>
<th></th>
<th>instantiation</th>
<th>static</th>
<th>dyn</th>
<th>config</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AUTOSAR</strong></td>
<td>✓</td>
<td></td>
<td></td>
<td>DSL</td>
</tr>
<tr>
<td>FreeRTOS</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>CPP-Macros</td>
</tr>
<tr>
<td>Zephyr</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>KConfig</td>
</tr>
<tr>
<td>POSIX</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

```cpp
TaskHandle_t t1, t2;
QueueHandle_t q1;
struct Message {...};

void create (Function f, int prio) {
    xTaskCreate("T " f.name(), f, prio);
}

int main() {
    t1 = create(task_1, 1);
    t2 = create(task_2, 2);
    q1 = xQueueCreate(5, sizeof(Message));
    vTaskStartScheduler();
}

task_1 {
    while (true) {
        Message m = produce();
        xQueueSend(q1, m);
    }
}

task_2 {
    Message m;
    while (true) {
        xQueueReceive(q1, &m);
        consume(m);
    }
}
```

```
/*oil

EVENT e1:
    TASK = T2;

/*cpp

TASK(T1) {
    m = produce();
    SetEvent(T2);
}

TASK(T2) {
    ActivateTask(T1);
    WaitEvent();
    consume(m);
}
```
A Generic Interface

Observations

- Common ground of all RTOSs: Syscalls.
- OS interaction happens *only* with syscalls.

Main Idea

- Build an abstract interpreter for syscalls.
- Calculate effects on an abstract state.

What are the exact requirements? Let’s look at the algorithms.
Algorithm I – Static Instance Analysis (SIA)

Application Source

```c
TaskHandle_t t1, t2;
QueueHandle_t q1;
struct Message {...};

void create(Function f, int prio) {
    xTaskCreate("T " f.name(), f, prio);
}

int main() {
    t1 = create(task_1, 1);
    t2 = create(task_2, 2);
    q1 = xQueueCreate(5, sizeof(Message));
    vTaskStartScheduler();
}

task_1 {
    while(true) {
        Message m = produce();
        xQueueSend(q1, m); } }

task_2 {
    Message m;
    while(true) {
        xQueueReceive(q1, &m);
        consume(m); } }
```

Static Instance Analysis (SIA)

Instance Graph
Algorithm I – Static Instance Analysis (SIA)

Application Source

```c
TaskHandle_t t1, t2;
QueueHandle_t q1;
struct Message {...};

void create(Function f, int prio) {
    xTaskCreate("T " f.name(), f, prio);
}

int main() {
    t1 = create(task_1, 1);
    t2 = create(task_2, 2);
    q1 = xQueueCreate(5, sizeof(Message));
    vTaskStartScheduler();
}

task_1 {
    while(true) {
        Message m = produce();
        xQueueSend(q1, m);
    }
}
task_2 {
    Message m;
    while(true) {
        xQueueReceive(q1, &m);
        consume(m);
    }
}
```

Instance Graph

1. Iterate all syscalls.
Algorithm I – Static Instance Analysis (SIA)

Application Source

TaskHandle_t t1, t2;
QueueHandle_t q1;
struct Message {...};

void create(Function f, int prio) {
xTaskCreate("T " f.name(), f, prio);
}

int main() {
t1 = create(task_1, 1);
t2 = create(task_2, 2);
q1 = xQueueCreate(5, sizeof(Message));
vTaskStartScheduler();
}

task_1 {
while(true) {
Message m = produce();
xQueueSend(q1, m); } }

task_2 {
Message m;
while(true) {
xQueueReceive(q1, &m);
consume(m); } }

Context
Call Path: main() → create()

Instance Graph

Static Instance Analysis (SIA)
1. Iterate all syscalls.
2. Find their context.
Algorithm I – Static Instance Analysis (SIA)

Application Source

```c
TaskHandle_t t1, t2;
QueueHandle_t q1;
struct Message {...};

void create(Function f, int prio) {
    xTaskCreate("T " f.name(), f, prio);
}

int main() {
    t1 = create(task_1, 1);
    t2 = create(task_2, 2);
    q1 = xQueueCreate(5, sizeof(Message));
    vTaskStartScheduler();
}

task_1 {
    while(true) {
        Message m = produce();
    xQueueSend(q1, m); } }

task_2 {
    Message m;
    while(true) {
        xQueueReceive(q1, &m);
    consume(m); } }
```

Context

Call Path: main() → create()

Instance Graph

Static Instance Analysis (SIA)

1. Iterate all syscalls.
2. Find their context.
3. Calculate the syscall’s effect.
Algorithm I – Static Instance Analysis (SIA)

Application Source

void create(Function f, int prio) {
    xTaskCreate("T " f.name(), f, prio);
}

int main() {
    t1 = create(task_1, 1);
    t2 = create(task_2, 2);
    q1 = xQueueCreate(5, sizeof(Message));
    vTaskStartScheduler();
}

task_1 {
    while(true) {
        Message m = produce();
        xQueueSend(q1, m);  // Task 1 sends a message
    }
}

task_2 {
    Message m;
    while(true) {
        xQueueReceive(q1, &m);
        consume(m);  // Task 2 receives and consumes a message
    }
}

Context

Call Path: main() → create()

Instance Graph

Static Instance Analysis (SIA)

1. Iterate all syscalls.
2. Find their context.
3. Calculate the syscall’s effect.
Algorithm I – Static Instance Analysis (SIA)

Application Source

```c
TaskHandle_t t1, t2;
QueueHandle_t q1;
struct Message {...};

void create(Function f, int prio) {
  xTaskCreate("T " f.name(), f, prio);
}

int main() {
  t1 = create(task_1, 1);
  t2 = create(task_2, 2);
  q1 = xQueueCreate(5, sizeof(Message));
  vTaskStartScheduler();
}

task_1 {
  while(true) {
    Message m = produce();
    xQueueSend(q1, m);  
  }
}

task_2 {
  Message m;
  while(true) {
    xQueueReceive(q1, &m);
    consume(m);  
  }
}
```

Context

Call Path: main() → create()

Instance Graph

Requirements:
1. Identify the syscalls.
2. Identify the syscalls' effect on the Instance Graph.

Static Instance Analysis (SIA)

1. Iterate all syscalls.
2. Find their context.
3. Calculate the syscall’s effect.
Algorithm I – Static Instance Analysis (SIA)

Application Source

```c
TaskHandle_t t1, t2;
QueueHandle_t q1;
struct Message {...};

void create(Function f, int prio) {
    xTaskCreate("T " f.name(), f, prio);
}

int main() {
    t1 = create(task_1, 1);
    t2 = create(task_2, 2);
    q1 = xQueueCreate(5, sizeof(Message));
    vTaskStartScheduler();
}

task_1 {
    while(true) {
        Message m = produce();
        xQueueSend(q1, m); } }

task_2 {
    Message m;
    while(true) {
        xQueueReceive(q1, &m);
        consume(m); } }
```

Context

Call Path: main() → create()

Instance Graph

Static Instance Analysis (SIA)

1. Iterate all syscalls.
2. Find their context.
3. Calculate the syscall’s effect.
Algorithm I – Static Instance Analysis (SIA)

Application Source

```c
TaskHandle_t t1, t2;
QueueHandle_t q1;
struct Message {...};

void create(Function f, int prio) {
    xTaskCreate("T " f.name(), f, prio);
}

int main() {
    t1 = create(task_1, 1);
    t2 = create(task_2, 2);
    q1 = xQueueCreate(5, sizeof(Message));
    vTaskStartScheduler();
}

task_1 {
    while(true) {
        Message m = produce();
        xQueueSend(q1, m); } }

task_2 {
    Message m;
    while(true) {
        xQueueReceive(q1, &m);
        consume(m); } }
```

Context

Call Path: main() → create()

Instance Graph

Static Instance Analysis (SIA)

1. Iterate all syscalls.
2. Find their context.
3. Calculate the syscall’s effect.

Requirements:

1. Identify the syscalls.
2. Identify the syscalls’ effect on the Instance Graph.
Algorithm II – System State Enumeration (SSE)

Application Source

```cpp
TASK T1:
    PRIORITY = 1;
    SCHEDULE = FULL;

TASK T2:
    PRIORITY = 2;
    SCHEDULE = FULL;
    AUTOSTART = TRUE;

EVENT e1:
    TASK = T2;

Message m;
    TASK T1 {
        m = produce();
        SetEvent(T2);
    }

    TASK T2 {
        ActivateTask(T1);
        WaitEvent();
        consume(m);
    }
```

System State Enumeration (SSE)

1. Follow the control flow (across RTOS boundaries).
2. Maintain a system state.
3. Capture states in a graph.
Algorithm II – System State Enumeration (SSE)

Application Source

```c
TASK T1:
    PRIORITY = 1;
    SCHEDULE = FULL;

TASK T2:
    PRIORITY = 2;
    SCHEDULE = FULL;
    AUTOSTART = TRUE;

EVENT e1:
    TASK = T2;

Message m;
    TASK(T1) {
        m = produce();
        SetEvent(T2);
    }

    TASK(T2) {
        ActivateTask(T1);
        WaitEvent();
        consume(m);
    }
```

SSTG

System State Enumeration (SSE)

1. Follow the control flow (across RTOS boundaries).
2. Maintain a system state.

Requirements:
1. Identify syscalls.
2. Identify the syscalls' effect on the system state.
Algorithm II – System State Enumeration (SSE)

**Application Source**

```
TASK T1:
  PRIORITY = 1;
  SCHEDULE = FULL;

TASK T2:
  PRIORITY = 2;
  SCHEDULE = FULL;
  AUTOSTART = TRUE;

EVENT e1:
  TASK = T2;
```

```
Message m;

TASK(T1) {
  m = produce();
  SetEvent(T2);
}

TASK(T2) {
  ActivateTask(T1);
  WaitEvent();
  consume(m);
}
```

**System State Enumeration (SSE)**

1. Follow the control flow (across RTOS boundaries).
2. Maintain a system state.

**Requirements:**

1. Identify syscalls.
2. Identify the syscalls’ effect on the system state.
Algorithm II – System State Enumeration (SSE)

Application Source

```cpp
TASK T1:
  PRIORITY = 1;
  SCHEDULE = FULL;

TASK T2:
  PRIORITY = 2;
  SCHEDULE = FULL;
  AUTOSTART = TRUE;

EVENT e1:
  TASK = T2;
```

```oil
Message m;

TASK(T1) {
  m = produce();
  SetEvent(T2);
}

TASK(T2) {
  ActivateTask(T1);
  WaitEvent();
  consume(m);
}
```

System State Enumeration (SSE)

1. Follow the control flow (across RTOS boundaries).
2. Maintain a system state.

State $n$

- $T1$: ready
- $T2$: active

State $n + 1$

- $T1$: active
- $T2$: blocked
Algorithm II – System State Enumeration (SSE)

Application Source

<table>
<thead>
<tr>
<th>TASK T1:</th>
<th>TASK T2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIORITY = 1; SCHEDULE = FULL;</td>
<td>PRIORITY = 2; SCHEDULE = FULL; AUTOSTART = TRUE;</td>
</tr>
</tbody>
</table>

EVENT e1: TASK = T2;

Message m;

```cpp
TASK(T1) {
  m = produce();
  SetEvent(T2);
}
```

```cpp
TASK(T2) {
  ActivateTask(T1);
  WaitEvent();
  consume(m);
}
```

System State Enumeration (SSE)

1. Follow the control flow (across RTOS boundaries).
2. Maintain a system state.
3. Capture states in a graph.

SSTG

State 1

State n

State n + 1

T1 ready T2 active

T1 active T2 blocked

...
Algorithm II – System State Enumeration (SSE)

1. Follow the control flow (across RTOS boundaries).
2. Maintain a system state.
3. Capture states in a graph.

Requirements:
1. Identify syscalls.
2. Identify the syscalls’ effect on the system state.
Our OS-Model

Algorithmic Requirements:

1. Identify syscalls.
2. Express the syscall’s effect
   - on the Instance Graph (SIA).
   - on the System State (SSE).

Additional requirements:

4. Be greedy (as detailed as possible).

Our Approach

Define an OS interpreter on a combined state. The model needs:

- A list of syscalls
- Arguments
- An interpret() function
- A system state
  - Contains Instance Graph (SIA)
  - Contains OS-object contexts
  - Contains multiple CPU states
Our OS-Model

Algorithmic Requirements:
1. Identify syscalls.
2. Express the syscall’s effect
   - on the Instance Graph (SIA).
   - on the System State (SSE).

Additional requirements:
4. Be greedy (as detailed as possible).
Our OS-Model

Algorithmic Requirements:
1. Identify syscalls.
2. Express the syscall’s effect
   - on the Instance Graph (SIA).
   - on the System State (SSE).

Additional requirements:
4. Be greedy (as detailed as possible).

Our Approach

Define an OS interpreter on a combined state.

The model needs:
- A list of syscalls
  - Name
  - Arguments
  - An interpret() function
- A system state
  - Contains Instance Graph
  - Contains OS-object contexts
  - Contains multiple CPU states
The Abstract System State

AbSS 15
The Abstract System State

InstanceGraph

- t1
  - k_fifo_put
  - q1
  - k_fifo_get
  - t2

CPU 0
IRQ: on
Instance: t1
IP: ABB 5 (line 5)
Call Path: t1_action
Status: syscall

CPU 1
IRQ: off
Instance: t2
IP: ABB 10 (line 10)
Call Path: t2_action
Status: computation

q1 context
Elements: 4

LUH RTOS-Independent Interaction Analysis in ARA – The Model 9 – 13
The Abstract System State

<table>
<thead>
<tr>
<th>InstanceGraph</th>
<th>CPU 0</th>
<th>CPU 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>IRQ: on</td>
<td>IRQ: off</td>
</tr>
<tr>
<td>q1</td>
<td>Instance: t1</td>
<td>Instance: t2</td>
</tr>
<tr>
<td>k_fifo_put</td>
<td>IP: ABB 5 (line 5)</td>
<td>IP: ABB 10 (line 10)</td>
</tr>
<tr>
<td>t2</td>
<td>Call Path: t1_action</td>
<td>Call Path: t2_action</td>
</tr>
<tr>
<td>q1 context</td>
<td>Status: syscall</td>
<td>Status: computation</td>
</tr>
</tbody>
</table>

LUH RTOS-Independent Interaction Analysis in ARA – The Model
The Abstract System State

<table>
<thead>
<tr>
<th>InstanceGraph</th>
<th>CPU 0</th>
<th>CPU 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IRQ: on</td>
<td>IRQ: off</td>
</tr>
<tr>
<td></td>
<td>Instance: t1</td>
<td>Instance: t2</td>
</tr>
<tr>
<td></td>
<td>IP: ABB 5 (line 5)</td>
<td>IP: ABB 10 (line 10)</td>
</tr>
<tr>
<td></td>
<td>Call Path: \textit{t1_action}</td>
<td>Call Path: \textit{t2_action}</td>
</tr>
<tr>
<td></td>
<td>Status: syscall</td>
<td>Status: computation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>t1 context</th>
<th>t2 context</th>
<th>q1 context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status: running</td>
<td>Status: active</td>
<td>Elements: 4</td>
</tr>
<tr>
<td>IP: ABB 5 (line 5)</td>
<td>IP: ABB 10 (line 10)</td>
<td></td>
</tr>
<tr>
<td>Call Path: \textit{t1_action}</td>
<td>Call Path: \textit{t2_action}</td>
<td></td>
</tr>
</tbody>
</table>
@syscall(categories={SyscallCategory.comm},
    signature=(Arg("task", ty=Task, hint=SigType.instance),
        Arg("event_mask")))
def SetEvent(cfg, state, cpu_id, args, va):
    state = state.copy()
    # - store event in event mask
    # - set other tast ready (wake up), if necessary
    # - add interaction into the instance graph
    return state
Validation – Applications

**FreeRTOS**
- **GPSLogger**, geolocation logging
- **LibrePilot**, quadcopter firmware

**AUTOSAR**
- **i4copter**, quadcopter firmware

**POSIX**
- **libmicrohttpd**, HTTP server

**Zephyr**
- **app_kernel**, benchmark application
- **sys_kernel**, benchmark application
## Validation – Results

<table>
<thead>
<tr>
<th>RTOS</th>
<th>SIA</th>
<th>SSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Obj</td>
<td>Int</td>
</tr>
<tr>
<td>FreeRTOS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LibrePilot</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zephyr</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUTOSAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POSIX</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Analyses are incomplete, not the OS model.
## Validation – Results

<table>
<thead>
<tr>
<th>RTOS</th>
<th>SIA</th>
<th>SSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Obj</td>
<td>Int</td>
</tr>
<tr>
<td>FreeRTOS</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LibrePilot</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Zephyr</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>app_kernel</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>sys_kernel</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>AUTOSAR</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>i4copter</td>
<td>✓*</td>
<td>✓</td>
</tr>
<tr>
<td>POSIX</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>libmicrohttpd</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Found interactions: 12 of 15

Problem value analyzer:
C++ wrapper class prevents instance retrieval
# Validation – Results

<table>
<thead>
<tr>
<th>RTOS</th>
<th>SIA</th>
<th>SSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Obj</td>
<td>Int</td>
</tr>
<tr>
<td>FreeRTOS</td>
<td>GPSLogger</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>LibrePilot</td>
<td>✓</td>
</tr>
<tr>
<td>Zephyr</td>
<td>app_kernel</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>sys_kernel</td>
<td>✓</td>
</tr>
<tr>
<td>AUTOSAR</td>
<td>i4copter</td>
<td>✓</td>
</tr>
<tr>
<td>POSIX</td>
<td>libmicrohttpd</td>
<td>✓*</td>
</tr>
</tbody>
</table>

Found interactions: 60 of 62

Problem value analyzer:
Dynamic assignment to an array
## Validation – Results

<table>
<thead>
<tr>
<th>RTOS</th>
<th>SIA</th>
<th>SSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Obj</td>
<td>Int</td>
</tr>
<tr>
<td>FreeRTOS</td>
<td>GPSLogger</td>
<td>✓</td>
</tr>
<tr>
<td>LibrePilot</td>
<td>✓</td>
<td>✓*</td>
</tr>
<tr>
<td>Zephyr</td>
<td>app_kernel</td>
<td>✓</td>
</tr>
<tr>
<td>sys_kernel</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>AUTOSAR</td>
<td>i4copter</td>
<td>✓</td>
</tr>
<tr>
<td>POSIX</td>
<td>libmicrohttpd</td>
<td>✓*</td>
</tr>
</tbody>
</table>

Found interactions: 60 of 62

Problem value analyzer:
Dynamic assignment to an array

### Conclusion

- Results not always complete but sound
- Analyses are incomplete not the OS model
Conclusion

- Problem: Make RTOS-aware analyses RTOS-independent
- Solution: Collect the RTOS specific parts within a model
- Central Design Decision: RTOS interpreter on an abstract state
- Validated for FreeRTOS, AUTOSAR, Zephyr and POSIX with 6 applications

Source: https://github.com/luhsra/ara

Thank you! Do you have questions?