Infering information in case of lost events in a trace

Marie Martin, Michel Dagenais

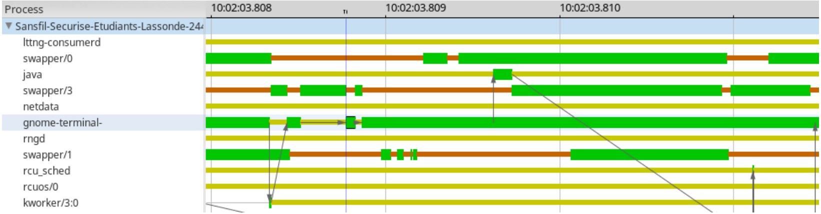
École Polytechnique de Montréal
DORSAL

Agenda

Introduction

Context

Lost events during tracing
→ Discard & overwrite modes



A trace with missing events

Parallel analysis of traces
→ "missing" past events

Objectives

1. Find incoherent events ;
2. Show inconsistency and uncertainty ;
3. Infer information about missing events.

Objectives

1. Find incoherent events ;
2. Show inconsistency and uncertainty ;
3. Infer information about missing events.

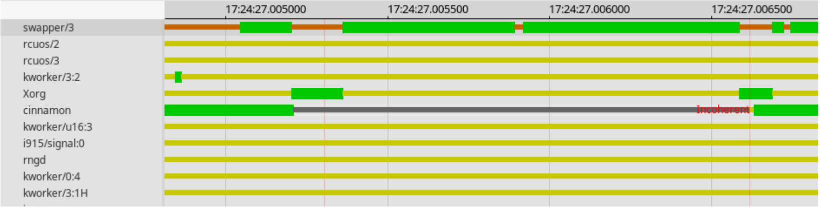
Objectives

1. Find incoherent events ;
2. Show inconsistency and uncertainty ;
3. Infer information about missing events.

Summary

Using Finite-State Machines (FSM): consistency & state certainty check

→ when reading the trace, update the state of the FSM with each event



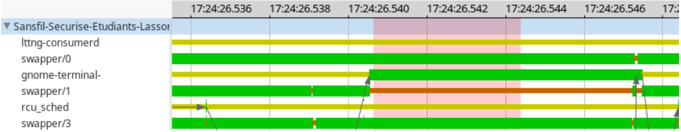
Detection of an inconsistent state

Objectives

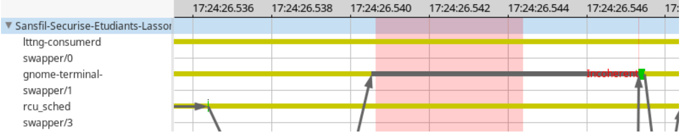
1. Find incoherent events ;
2. Show inconsistency and uncertainty ;
3. Infer information about missing events.

Framework

Example – intuitive reasoning



A trace with an inconsistent state – as displayed in the Control Flow View



A trace with an inconsistent state – as displayed in the Coherence View

Event inference

Objective

Find states between the **first certain state after the incoherence** and the **last known consistent state**

Basis

FSM $\leftrightarrow$ Graph

State $\leftrightarrow$ Node
Transition $\leftrightarrow$ Arc

$\Rightarrow$ Find a path between a **starting state** and a **target state**

Event inference

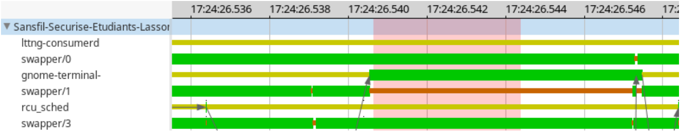
Algorithm

Dijkstra's shortest path: computes the shortest path in a directed graph

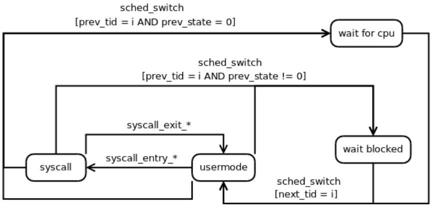
Iterates over states, updating their distance from the start using weights on arcs (see following example)

Weights: statistics on transitions
$\rightarrow$ for a given scenario, the most likely transition is the one that has been taken most frequently

Example

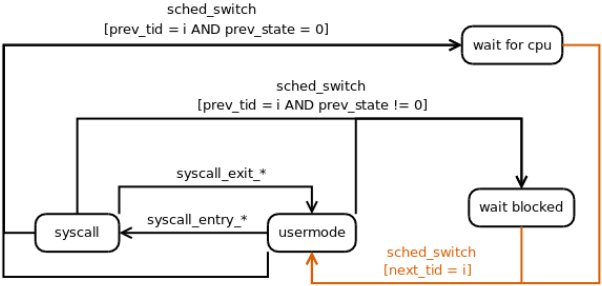


A trace with an inconsistent state – as displayed in the Control Flow View



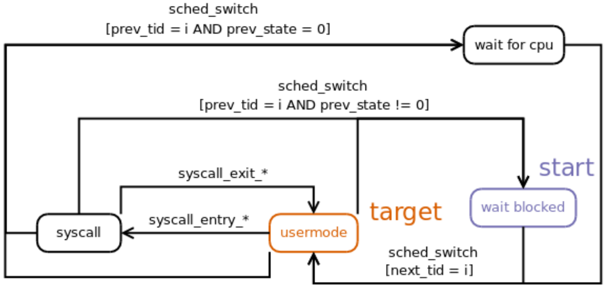
The simplified FSM modeling a process

Example



Set of possible transitions, following from the consistency check
→ Select the "best" (most likely) transition to get the starting point of the algorithm

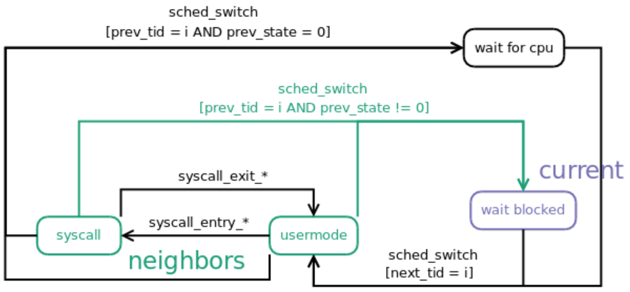
Example



	usermode	syscall	wait for cpu	wait blocked
dist	∞	∞	∞	0
prev	UNDEFINED	UNDEFINED	UNDEFINED	UNDEFINED

unvisited = { usermode, syscall, wait for cpu, wait blocked }

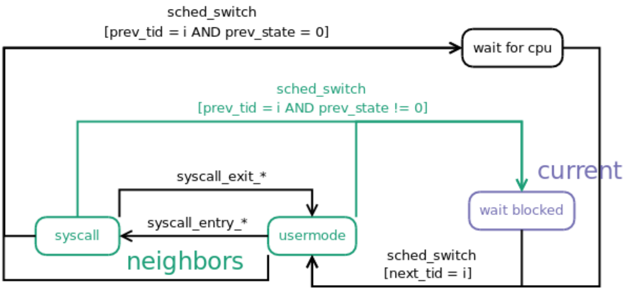
Example



	usermode	syscall	wait for cpu	wait blocked
dist	∞	∞	∞	0
prev	UNDEFINED	UNDEFINED	UNDEFINED	UNDEFINED

unvisited = { usermode, syscall, wait for cpu, wait blocked }

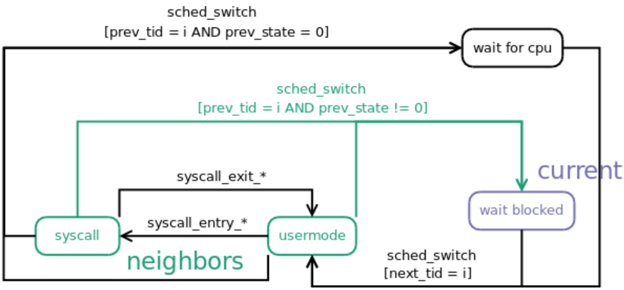
Example



	usermode	syscall	wait for cpu	wait blocked
dist	0.0135	∞	∞	0
prev	wait blocked	UNDEFINED	UNDEFINED	UNDEFINED

unvisited = { usermode, syscall, wait for cpu, wait blocked }

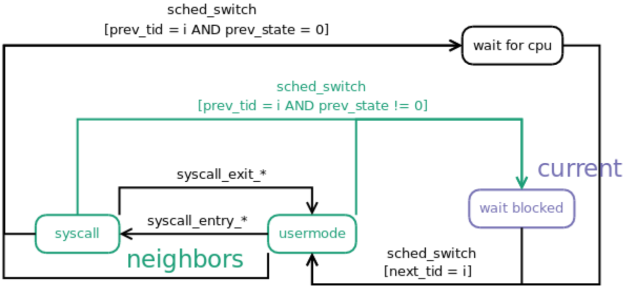
Example



	usermode	syscall	wait for cpu	wait blocked
dist	0.0135	1	∞	0
prev	wait blocked	wait blocked	UNDEFINED	UNDEFINED

unvisited = { usermode, syscall, wait for cpu, wait blocked }

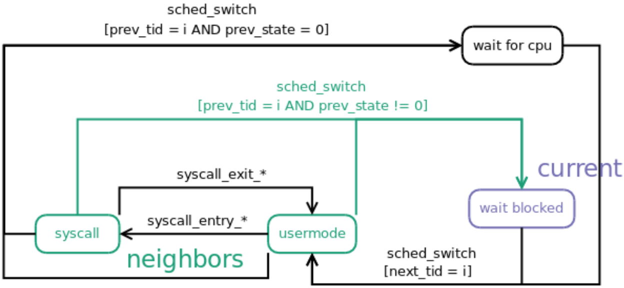
Example



	usermode	syscall	wait for cpu	wait blocked
dist	0.0135	1	∞	0
prev	wait blocked	wait blocked	UNDEFINED	UNDEFINED

unvisited = { usermode, syscall, wait for cpu }

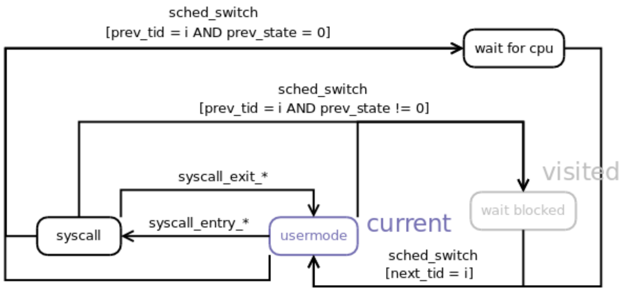
Example



	usermode	syscall	wait for cpu	wait blocked
dist	0.0135	1	∞	0
prev	wait blocked	wait blocked	UNDEFINED	UNDEFINED

unvisited = { usermode, syscall, wait for cpu }
current = min_dist(unvisited) = usermode

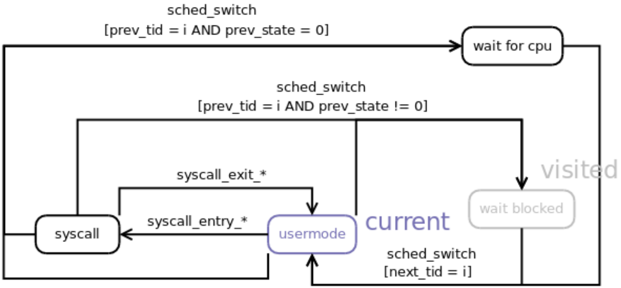
Example



	usermode	syscall	wait for cpu	wait blocked
dist	0.0135	1	1	0
prev	wait blocked	wait blocked	UNDEFINED	UNDEFINED

unvisited = { usermode, syscall, wait for cpu }

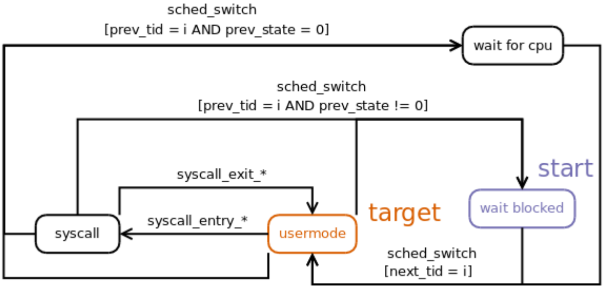
Example



	usermode	syscall	wait for cpu	wait blocked
dist	0.0135	1	1	0
prev	wait blocked	wait blocked	UNDEFINED	UNDEFINED

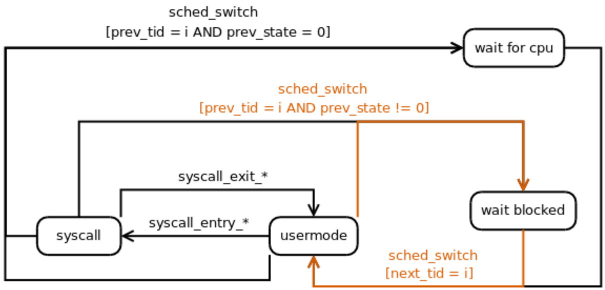
unvisited = { usermode, syscall, wait for cpu }
current = target ⇒ STOP

Example



	usermode	syscall	wait for cpu	wait blocked
prev	wait blocked	wait blocked	UNDEFINED	UNDEFINED

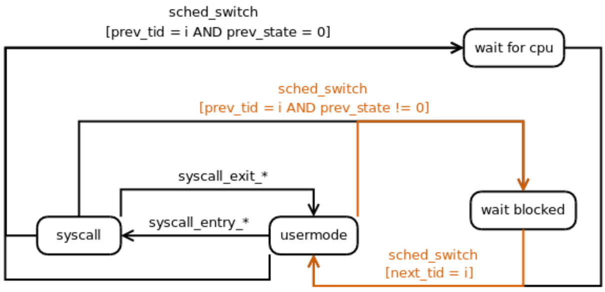
Example



	usermode	syscall	wait for cpu	wait blocked
prev	wait blocked	wait blocked	UNDEFINED	UNDEFINED

path = (usermode → wait blocked → usermode)

Example



	usermode	syscall	wait for cpu	wait blocked
prev	wait blocked	wait blocked	UNDEFINED	UNDEFINED

path = (usermode $\rightarrow$ wait blocked $\rightarrow$ usermode)

inferred events = (sched_switch)

Content inference

Basis

If a transition occurs, then its conditions are verified.

We can make assumptions based on the conditions, in order to infer information about the content of the event.

Example

```
<test id="prev_state_0">
    <if>
        <condition>
            <field name="prev_state" />
            <stateValue type="long" value="0" />
        </condition>
    </if>
</test>
```

XML condition

Multi-valued inferred event

Sometimes, several values are possible for the same event field.

```xml
<condition>
    <stateAttribute type="constant" value="Threads" />
    <stateAttribute type="eventField" value="tid" />
    <stateAttribute type="constant" value="Status" />
    <stateValue type="int" value="$PROCESS_STATUS_RUN_USERMODE" />
</condition>
```

XML condition

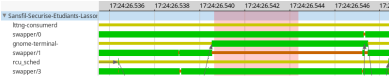
We can try to discriminate between values using information from the state system.
Otherwise, we leave it to the user to select the most appropriate value (see following views).

Results

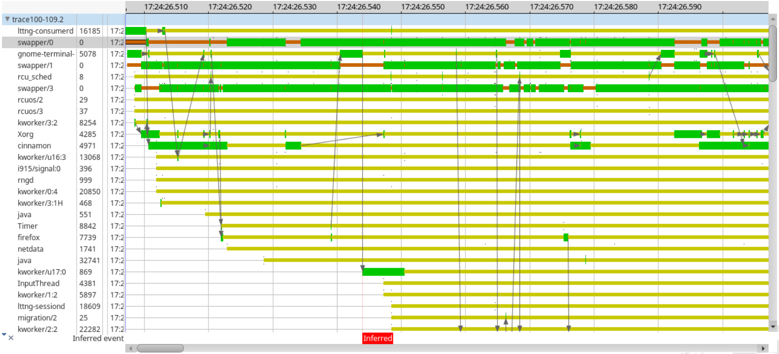
Methodology

1. Definition of the FSM in XML (from the state provider definition)
2. Deletion of chosen events from a 'real-world' trace
3. Execution of the analysis in Trace Compass
4. Selection of the Global Coherence View to show inferred events

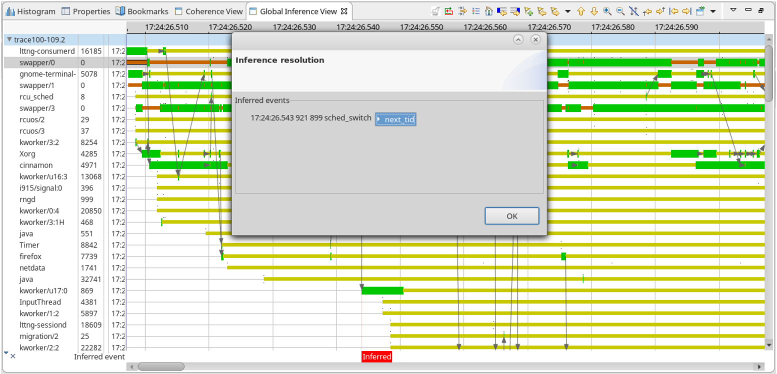
Inference view



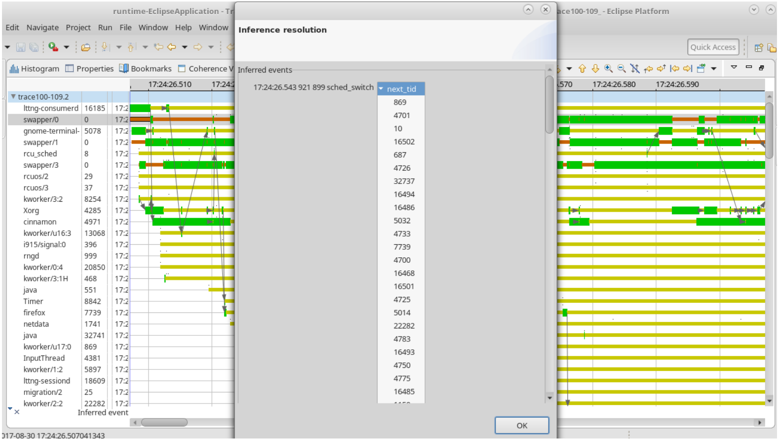
Control Flow View of the original trace



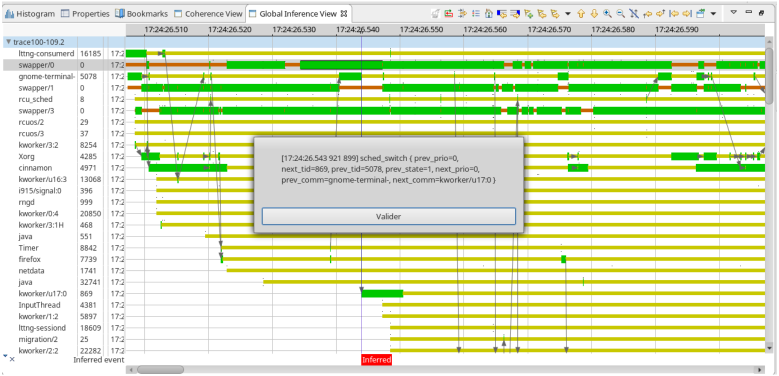
Inference selection



Inference selection



Inferred event details



Benchmark

Average on 15 runs of the analysis module
$\rightarrow$ Force check for every event

	mytrace1	mytrace2	many-threads	trace2	mytrace3
size	164K	26M	8.1M	14M	86M
nb. events	2188	40902	240644	595641	2689393
no check (s)	1.24	3.31	9.56	15.43	41.7
check + infer (s)	1.26	3.53	12.18	19.4	93.6
nb. inferred events	1	3	29433	769	1501
overhead (%)	1.6	6.6	27.4	25.7	124.5

Analysis execution with event inference

(CPU AMD A10-8700P, 4 cores, 16G RAM)

Conclusion

Limits

Current cost of XML analysis is high for big traces

XML FSM is user-defined

Content inference lacks semantics

The most likely path may not be the path that has really been taken, especially as losing events may occur in an unusual case.

Future work

Continuous work on improving the algorithms and the view
$\rightarrow$ scalability

Machine-learning for more accurate probabilities

Automatic computation of the XML FSM, given some traces of the system

Conclusion

Improve Trace Compass by helping the user be aware of trace (un)consistency

Proof-of-concept that we can retrieve lost/inaccessible information in a trace, using state machines

A step towards the parallel analysis of traces

Any questions?

marie.martin@polymtl.ca
github : MMartin5/events-analysis

Architecture

Inference algorithm

→ On-demand, after the analysis

→ Incoherences are collected from the previous analysis

→ Creates inferred event(s) for each incoherence

Inference trace

→ "Artificial" trace with inferred events

→ Only used by the Global Coherence View