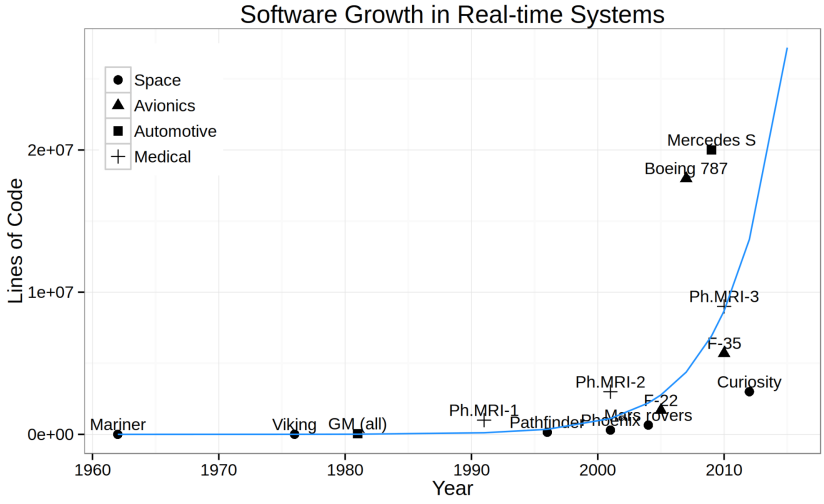
Mining Task Precedence Graphs from Real-Time Embedded System Traces

Oleg Iegorov Sebastian Fischmeister

Department of Electrical and Computer Engineering
University of Waterloo, Canada
{oiegorov,sfischme}@uwaterloo.ca

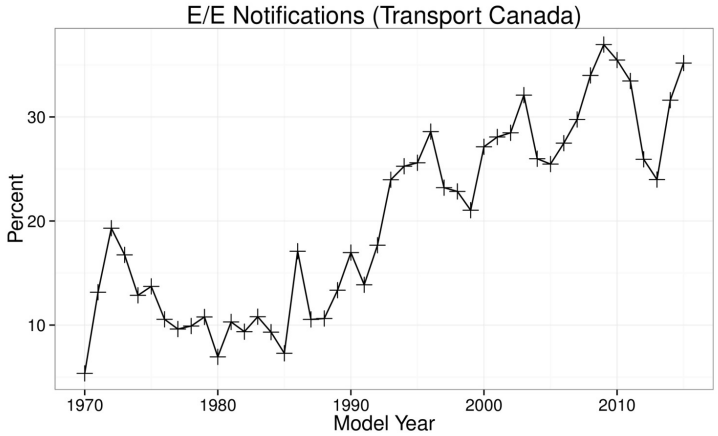
April 13, 2018



Sources:

- Charette RN. "This Car Runs on Code". IEEE Spectrum. 2009 Feb 1;46(3):3.
- Holzmann GJ. "Landing a Spacecraft on Mars". IEEE Software. 2013 Mar;30(2):83-6.

Software-based Recalls in Automotive



Johnson, Taylor T., Raghunath Gannamaraju, and Sebastian Fischmeister. "A Survey of Electrical and Electronic (E/E) Notifications for Motor Vehicles." 24th International Technical Conference on the Enhanced Safety of Vehicles (ESV). 2015.

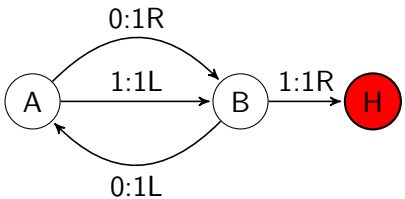
Busy Beaver

$S(n, m)$: the largest number of steps taken by an n-state, m-symbol machine started on an initially blank tape before halting.

Busy Beaver

$S(n, m)$: the largest number of steps taken by an n-state, m-symbol machine started on an initially blank tape before halting.

	A	**B**
0	1RB	1LA
1	1LB	1RH

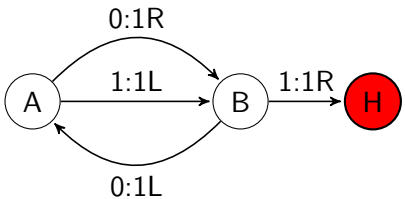


`http://bit.ly/IJF9dO`

Busy Beaver

$S(n, m)$: the largest number of steps taken by an n-state, m-symbol machine started on an initially blank tape before halting.

	A	**B**
0	1RB	1LA
1	1LB	1RH

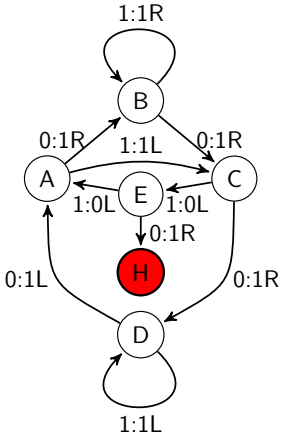


`http://bit.ly/IJF9dO`

0 0 1 1 **1** 1 0 0 (6 steps, four "1"s total)

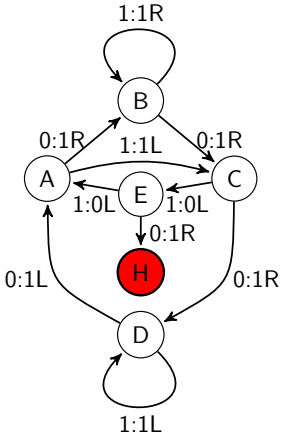
Busy Beaver

$S(5, 2) =$



Busy Beaver

$S(5,2) =$

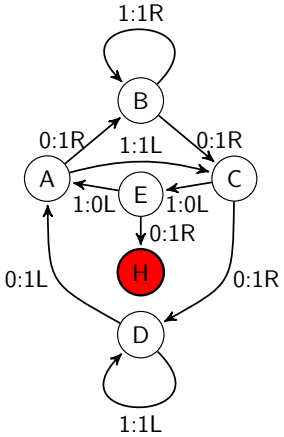


$\geq 47,176,870$

$S(2,5) =$

Busy Beaver

$S(5,2) =$



$\geq 47,176,870$

$S(2,5) = \geq 1.9 \times 10^{704}$

Consequence from Current Trends

Need better theory, methods, and tools to build and understand systems.

Consequence from Current Trends

Need better theory, methods, and tools to build and understand systems.

$\rightarrow$ Reverse engineering

Consequence from Current Trends

Need better theory, methods, and tools to build and understand systems.

$\rightarrow$ Reverse engineering

Good news: more complexity means more data!

Consequence from Current Trends

Need better theory, methods, and tools to build and understand systems.

$\rightarrow$ Reverse engineering

Good news: more complexity means more data!

$\rightarrow$ **Data-driven** reverse engineering

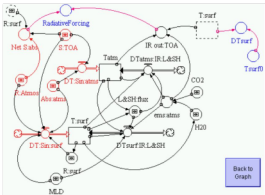
Reverse Engineering

"(Software) reverse engineering is the process of analyzing a subject system to create representations of the system at a higher level of abstraction" [Chikofsky et al, 1990]

System

reverse engineering

System model



Traditionally used in the domain of desktop and enterprise software

Uses of Reverse Engineering in Real-time Systems

Possible applications:

- legacy code maintenance
- debugging
- anomaly detection
- testing
- documentation

Uses of Reverse Engineering in Real-time Systems

Possible applications:

- legacy code maintenance
- debugging
- anomaly detection
- testing
- documentation

Timinig is neglected in standard software reverse engineering tools.

Uses of Reverse Engineering in Real-time Systems

Possible applications:

- legacy code maintenance
- debugging
- anomaly detection
- testing
- documentation

Timinig is neglected in standard software reverse engineering tools.

$\rightarrow$ Tools for reverse engineering for real-time systems are needed

Characteristics of a Reverse Engineering Tool

- Representation of a system
  - Finite state machine
  - Petri net
  - Regular expressions
  - UML model
  - and others...
- Extraction of information from a system
  - Static: from source code
  - Dynamic: from traces
  - Hybrid
- Learning behaviour
  - Active: create new test cases and observe reaction
  - Passive: only observe

Characteristics of a Reverse Engineering Tool

- Representation of a system
  - Finite state machine
  - Petri net
  - Regular expressions
  - UML model
  - Task precedence graph ⟵ TPG_miner
  - and others...
- Extraction of information from a system
  - Static: from source code
  - Dynamic: from traces
  - Hybrid
- Learning behaviour
  - Active: create new test cases and observe reaction
  - Passive: only observe

Characteristics of a Reverse Engineering Tool

- Representation of a system
  - Finite state machine
  - Petri net
  - Regular expressions
  - UML model
  - Task precedence graph ⟵ TPG_miner
  - and others...
- Extraction of information from a system
  - Static: from source code
  - Dynamic: from traces ⟵ TPG_miner
  - Hybrid
- Learning behaviour
  - Active: create new test cases and observe reaction
  - Passive: only observe

Characteristics of a Reverse Engineering Tool

- Representation of a system
  - Finite state machine
  - Petri net
  - Regular expressions
  - UML model
  - Task precedence graph ⟵ TPG_miner
  - and others...
- Extraction of information from a system
  - Static: from source code
  - Dynamic: from traces ⟵ TPG_miner
  - Hybrid
- Learning behaviour
  - Active: create new test cases and observe reaction
  - Passive: only observe ⟵ TPG_miner

TPG miner (Task Precedence Graph miner)

Real-Time
Embedded System

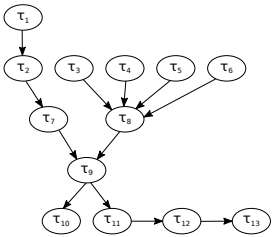
execution

Trace

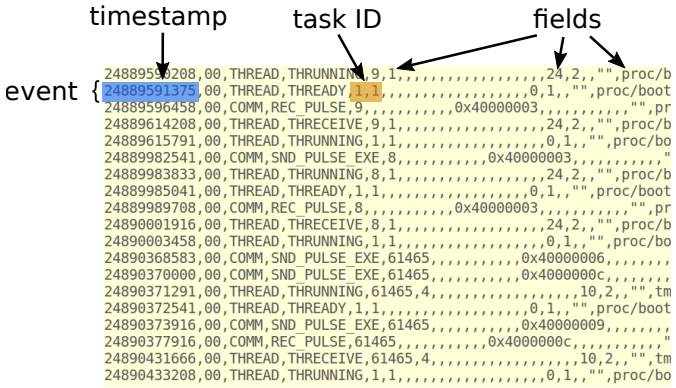
```
time,bus,task_length,bitcount,mode,bytes…,b1,b2,b3,b4,b5,…
0.000387,1,20,148000,78,Rx,3,00,00,07,…,…
0.000633,1,223,236000,122,Rx,8,00,00,00,00,00,00,00,33
0.000883,1,25,224000,116,Rx,8,00,00,09,00,09,78,78,07
0.007112,1,84,240000,124,Rx,8,00,00,00,00,00,00,00,BC
0.007379,1,22,238000,123,Rx,8,01,FA,01,FD,00,00,00,23
0.009284,1,23,216000,112,Rx,7,02,05,02,02,00,00,3C,…
0.010064,1,85,202000,105,Rx,6,00,00,00,00,11,00,11,3D
0.011098,1,80,202000,104,Rx,6,00,00,00,00,11,00,…,…
0.012758,1,224,240000,124,Rx,8,20,00,00,09,00,AF,00,00
0.012924,1,82,202000,104,Rx,6,00,00,00,00,11,00,…
0.013168,1,413,236000,122,Rx,8,03,00,13,03,00,00,00,00
0.014136,1,208,232000,120,Rx,8,07,26,02,00,10,00,00,19
0.016638,1,20,148000,78,Rx,3,00,00,07,…,…
0.018708,1,84,240000,124,Rx,8,00,00,00,00,00,00,00,BC
0.019001,1,22,238000,123,Rx,8,01,FB,01,FC,00,00,00,23
0.020038,1,22,238000,123,Rx,8,01,FD,01,FB,00,00,00,23
0.02026,1,23,216000,112,Rx,7,02,00,02,09,00,00,37,…
0.022233,1,213,234000,121,Rx,8,00,FE,9C,00,FF,84,00,20
0.024536,1,80,202000,105,Rx,6,00,00,00,00,11,00,…
0.024786,1,302,242000,125,Rx,8,00,00,00,00,00,00,00,00
0.024993,1,82,202000,104,Rx,6,00,00,00,00,11,00,…
0.026888,1,208,232000,120,Rx,8,07,26,02,00,15,00,00,00,E8
0.028051,1,20,148000,78,Rx,3,00,00,07,…,…
0.030755,1,223,236000,122,Rx,8,00,00,00,00,00,00,00,33
0.030987,1,25,224000,116,Rx,8,00,00,00,09,78,78,78,07
0.031235,1,84,240000,124,Rx,8,00,00,00,00,00,00,00,BC
0.032101,1,22,238000,123,Rx,8,01,FB,01,FC,00,00,00,23
0.032325,1,23,216000,112,Rx,7,02,02,02,00,00,00,39,…
0.034611,1,213,234000,121,Rx,8,00,FE,9C,00,FF,84,00,11,32
0.036613,1,80,202000,105,Rx,6,00,00,00,00,11,00,…,…
0.036861,1,224,240000,124,Rx,8,20,00,00,00,00,AF,00,00
0.037069,1,82,202000,104,Rx,6,00,00,00,00,11,00,…
0.040395,1,574,122000,65,Rx,2,73,01,…,…,…
0.041609,1,540,144000,76,Rx,3,0A,00,09,…,…
0.042565,1,20,148000,78,Rx,3,00,00,07,…,…
0.042835,1,84,240000,124,Rx,8,00,00,00,00,00,00,00,BC
0.043065,1,25,224000,116,Rx,8,00,09,00,09,78,78,78,07
0.043313,1,22,238000,123,Rx,8,01,FC,01,FB,00,00,00,23
0.044388,1,23,216000,112,Rx,7,02,05,02,00,00,00,3C,…
0.044846,1,340,216000,112,Rx,7,10,00,00,00,C9,00,1E,…
0.045092,1,208,232000,122,Rx,8,07,26,00,00,10,00,00,11
0.048772,1,80,202000,105,Rx,6,00,00,00,00,11,00,…
0.048922,1,320,142000,75,Rx,3,00,3D,4E,…,…
0.04913,1,82,202000,104,Rx,6,00,00,00,00,11,00,…
0.050755,1,208,232000,120,Rx,8,07,26,02,00,14,00,00,5E
0.05473,1,20,148000,78,Rx,3,00,00,07,…,…
0.054974,1,225,236000,122,Rx,8,00,00,00,00,00,00,00,33
0.055208,1,25,224000,116,Rx,8,00,09,00,09,78,78,78,07
0.055454,1,84,240000,124,Rx,8,00,00,00,00,00,00,00,BC
0.056328,1,22,238000,123,Rx,8,01,FB,01,FD,00,00,00,24
0.05655,1,23,216000,111,Rx,7,02,05,02,00,00,00,3A,…
0.058175,1,214,232000,120,Rx,8,03,AF,00,19,00,00,11,2A
```

TPG_miner

Task Precedence Graph

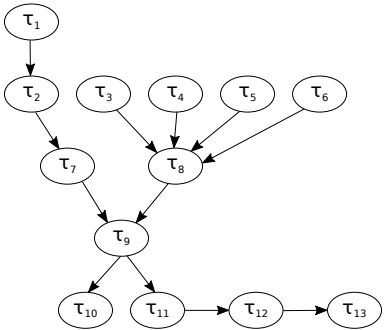


Real-time System Execution Trace



- *Task ID*: conjuction of fields that uniquely identify a task.
- *Event*: tuple <timestamp, task ID>
- *Trace*: chronologically ordered list of events

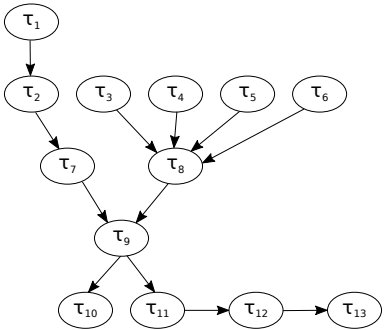
Task Precedence Graph (TPG)



TPG is a DAG:

- Nodes (τ_i): task IDs;
- Edges ($\tau_i \rightarrow \tau_j$): immediate precedence relations (τ_i's output is τ_j's input), $\tau_i \ll \tau_j$.

Task Precedence Graph (TPG)



TPG is a DAG:

- Nodes (τ_i): task IDs;
- Edges ($\tau_i \rightarrow \tau_j$): immediate precedence relations (τ_i's output is τ_j's input), $\tau_i \ll \tau_j$.

Problem: How to mine immediate precedence relations from a trace?

TPG_Miner Approach

1. Transactionalize trace to different events

TPG_Miner Approach

1. Transactionalize trace to different events
2. Identify occurrence of events between events ($\rightarrow$ occurrence pattern)

TPG_Miner Approach

1. Transactionalize trace to different events
2. Identify occurrence of events between events ($\rightarrow$ occurrence pattern)
3. Extract precedence relations

TPG_Miner Approach

1. Transactionalize trace to different events
2. Identify occurrence of events between events ($\rightarrow$ occurrence pattern)
3. Extract precedence relations
4. Create DAG from relations

TPG_Miner Approach

1. Transactionalize trace to different events
2. Identify occurrence of events between events ($\rightarrow$ occurrence pattern)
3. Extract precedence relations
4. Create DAG from relations
5. Clean up the DAG

Occurrence Pattern (op)

Given a string s, its **occurrence pattern**, $op(s)$, is the shortest substring of s that "covers" s:

$$s = \langle 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0 \rangle$$

Occurrence Pattern (op)

Given a string s, its **occurrence pattern**, $op(s)$, is the shortest substring of s that "covers" s:

$$s = \langle \boxed{1,0,0,1,0}, \boxed{1,0,0,1,0}, 1, 0, 0 \rangle$$

- $p_1 = \langle 1, 0, 0, 1, 0 \rangle$ is the occurrence pattern of s, $op(s) = p_1$

Occurrence Pattern (op)

Given a string s, its **occurrence pattern**, $op(s)$, is the shortest substring of s that "covers" s:

$$s = \langle 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0 \rangle$$

- $p_1 = \langle 1, 0, 0, 1, 0 \rangle$ is the occurrence pattern of s, $op(s) = p_1$
- $p_2 = \langle 3, 2 \rangle$ is not a substring of s

Occurrence Pattern (op)

Given a string s, its **occurrence pattern**, $op(s)$, is the shortest substring of s that "covers" s:

$$s = \langle \boxed{1,0}, 0, \boxed{1,0}, \boxed{1,0}, 0, \boxed{1,0}, \boxed{1,0}, 0 \rangle$$

- $p_1 = \langle 1, 0, 0, 1, 0 \rangle$ is the occurrence pattern of s, $op(s) = p_1$
- $p_2 = \langle 3, 2 \rangle$ is not a substring of s
- $p_3 = \langle 1, 0 \rangle$ is a substring of s, doesn't cover s

Occurrence Pattern (op)

Given a string s, its **occurrence pattern**, $op(s)$, is the shortest substring of s that "covers" s:

$$s = \langle \boxed{1,0,0,1,0,1,0,0,1,0}, 1, 0, 0 \rangle$$

- $p_1 = \langle 1, 0, 0, 1, 0 \rangle$ is the occurrence pattern of s, $op(s) = p_1$
- $p_2 = \langle 3, 2 \rangle$ is not a substring of s
- $p_3 = \langle 1, 0 \rangle$ is a substring of s, doesn't cover s
- $p_4 = \langle 1, 0, 0, 1, 0, 1, 0, 0, 1, 0 \rangle$ covers s, but not the shortest (see p_1).

Occurrence Pattern (op)

Given a string s, its **occurrence pattern**, $op(s)$, is the shortest substring of s that "covers" s:

$$s = \langle 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0 \rangle$$

- $p_1 = \langle 1, 0, 0, 1, 0 \rangle$ is the occurrence pattern of s, $op(s) = p_1$
- $p_2 = \langle 3, 2 \rangle$ is not a substring of s
- $p_3 = \langle 1, 0 \rangle$ is a substring of s, doesn't cover s
- $p_4 = \langle 1, 0, 0, 1, 0, 1, 0, 0, 1, 0 \rangle$ covers s, but not the shortest (see p_1).

Given a trace $T = \langle \tau_i, \tau_h, .. \rangle$, $\boldsymbol{op(\tau_i, \tau_j)}$ is the occurrence pattern of a string s whose elements are the numbers of occurrences of τ_i between consecutive occurrences of τ_j.

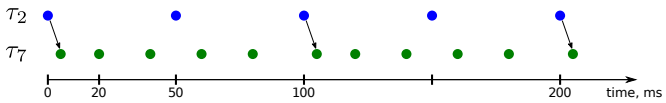
$$\begin{aligned} T \quad = \quad & \langle\ \tau_1,\ \boldsymbol{\tau_2},\ \tau_3,\ \tau_4,\ \tau_5,\ \tau_6,\ \boldsymbol{\tau_7},\ \tau_3,\ \tau_4,\ \boldsymbol{\tau_7},\ \tau_5,\ \tau_4,\ \boldsymbol{\tau_7},\ \boldsymbol{\tau_2},\ \tau_1,\ \tau_5,\ \boldsymbol{\tau_7},\ \tau_6,\ \boldsymbol{\tau_7},\ \tau_3, \\ & \boldsymbol{\tau_2},\ \tau_5,\ \tau_4,\ \boldsymbol{\tau_7},\ \boldsymbol{\tau_7},\ \boldsymbol{\tau_7},\ \tau_5,\ \boldsymbol{\tau_2},\ \boldsymbol{\tau_7},\ \tau_1,\ \boldsymbol{\tau_7},\ \boldsymbol{\tau_2},\ \boldsymbol{\tau_7},\ \tau_5,\ \tau_1,\ \boldsymbol{\tau_7},\ \tau_3 \rangle \end{aligned}$$

$$op(\tau_2, \tau_7) = \langle 1, 0, 0, 1, 0 \rangle$$

Occurrence Patterns and Immediate Precedence Relations

Theorem

If a task τ_i is an immediate predecessor of a task τ_j ($\tau_i \ll \tau_j$) in a real-time system S, then there exists $op(\tau_i,\tau_j)$ in any execution trace of S captured during at least two hyperperiods of S.



Visualization of occurrences of tasks τ_2 and τ_7 during two hyperperiods;
$\tau_2 \ll \tau_7$; $op(\tau_2, \tau_7) = \langle 1, 0, 0, 1, 0 \rangle$.

Mining Occurrence Patterns in a Trace

```
Function mine_ops(T)
    Data: trace T
    Result: op(τ_i, τ_j) ∀τ_i, τ_j in T
    foreach τ_i ∈ T do
        foreach τ_j ∈ T, i ≠ j do
            s ← numbers of occurrences of τ_j between consecutive occurrences of τ_i in T
            foreach prefix q of s, length(q)=1,2,..,length(s)/2 do
                if q covers s then
                    op(τ_i, τ_j) ← q
                    break
                else if length(q)==length(s)/2 then
                    op(τ_i, τ_j) ← NULL
            end
            return op
        end
    end
```

$$s = \langle 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0 \rangle$$

Mining Occurrence Patterns in a Trace

```
Function mine_ops(T)
    Data: trace T
    Result: op(τ_i, τ_j) ∀τ_i, τ_j in T
    foreach τ_i ∈ T do
        foreach τ_j ∈ T, i ≠ j do
            s ← numbers of occurrences of τ_j between consecutive occurrences of τ_i in T
            foreach prefix q of s, length(q)=1,2,..,length(s)/2 do
                if q covers s then
                    op(τ_i, τ_j) ← q
                    break
                else if length(q)==length(s)/2 then
                    op(τ_i, τ_j) ← NULL
            end
            return op
        end
    end
```

$$s = \langle \mathbf{1}, \mathbf{0}, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0 \rangle$$

$$q = \langle 1 \rangle$$

Mining Occurrence Patterns in a Trace

```
Function mine_ops(T)
    Data: trace T
    Result: op(τ_i, τ_j) ∀τ_i, τ_j in T
    foreach τ_i ∈ T do
        foreach τ_j ∈ T, i ≠ j do
            s ← numbers of occurrences of τ_j between consecutive occurrences of τ_i in T
            foreach prefix q of s, length(q)=1,2,..,length(s)/2 do
                if q covers s then
                    op(τ_i, τ_j) ← q
                    break
                else if length(q)==length(s)/2 then
                    op(τ_i, τ_j) ← NULL
            end
            return op
        end
    end
```

$$s = \langle \mathbf{1}, \mathbf{0}, \mathbf{0}, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0 \rangle$$

$$q = \langle 1, 0 \rangle$$

Mining Occurrence Patterns in a Trace

```
Function mine_ops(T)
    Data: trace T
    Result: op(τ_i, τ_j) ∀τ_i, τ_j in T
    foreach τ_i ∈ T do
        foreach τ_j ∈ T, i ≠ j do
            s ← numbers of occurrences of τ_j between consecutive occurrences of τ_i in T
            foreach prefix q of s, length(q)=1,2,..,length(s)/2 do
                if q covers s then
                    op(τ_i, τ_j) ← q
                    break
                else if length(q)==length(s)/2 then
                    op(τ_i, τ_j) ← NULL
            end
            return op
        end
    end
```

$$s = \langle \mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{0}, \mathbf{1}, 0, 0, 1, 0, 1, 0, 0 \rangle$$
$$q = \langle 1, 0, 0 \rangle$$

Mining Occurrence Patterns in a Trace

```
Function mine_ops(T)
    Data: trace T
    Result: op(τ_i, τ_j) ∀τ_i, τ_j in T
    foreach τ_i ∈ T do
        foreach τ_j ∈ T, i ≠ j do
            s ← numbers of occurrences of τ_j between consecutive occurrences of τ_i in T
            foreach prefix q of s, length(q)=1,2,..,length(s)/2 do
                if q covers s then
                    op(τ_i, τ_j) ← q
                    break
                else if length(q)==length(s)/2 then
                    op(τ_i, τ_j) ← NULL
            end
            return op
        end
    end
```

$$s = \langle \mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{0}, 1, 0, 0, 1, 0, 1, 0, 0 \rangle$$
$$q = \langle 1, 0, 0, 1 \rangle$$

Mining Occurrence Patterns in a Trace

```
Function mine_ops(T)
    Data: trace T
    Result: op(τ_i, τ_j) ∀τ_i, τ_j in T
    foreach τ_i ∈ T do
        foreach τ_j ∈ T, i ≠ j do
            s ← numbers of occurrences of τ_j between consecutive occurrences of τ_i in T
            foreach prefix q of s, length(q)=1,2,..,length(s)/2 do
                if q covers s then
                    op(τ_i, τ_j) ← q
                    break
                else if length(q)==length(s)/2 then
                    op(τ_i, τ_j) ← NULL
            end
            return op
        end
    end
```

$$s = \langle \mathbf{1, 0, 0, 1, 0}, \mathbf{1, 0, 0, 1, 0, 1, 0, 0} \rangle$$

$$q = \langle 1, 0, 0, 1, 0 \rangle$$

Mining a TPG from a Trace

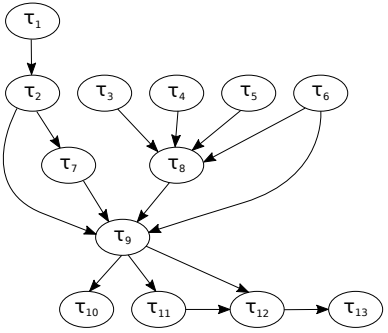
Function `mine_TPG(`T`)`

Data: trace T
Result: TPG of T
nodes $\leftarrow$ unique task IDs in T
edges $\leftarrow$ mine_ops(T)
TPG $\leftarrow$ (nodes, edges)
TPG $\leftarrow$ transitive_closure(TPG)
return TPG

Mining a TPG from a Trace

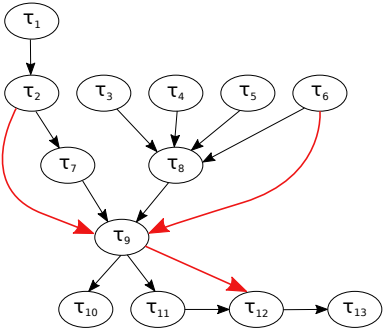
Function `mine_TPG(`T`)`

- **Data:** trace T
- **Result:** TPG of T
- nodes $\leftarrow$ unique task IDs in T
- edges $\leftarrow$ mine_ops(T)
- **TPG $\leftarrow$ (nodes, edges)**
- TPG $\leftarrow$ transitive_closure(TPG)
- **return** TPG



Mining a TPG from a Trace

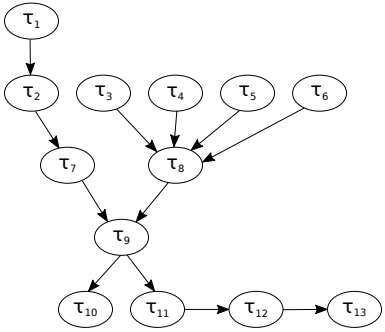
Function `mine_TPG(`T`)`
- **Data:** trace T
- **Result:** TPG of T
- nodes $\leftarrow$ unique task IDs in T
- edges $\leftarrow$ mine_ops(T)
- TPG $\leftarrow$ (nodes, edges)
- **TPG $\leftarrow$ transitive_closure(TPG)**
- **return** TPG



Mining a TPG from a Trace

Function `mine_TPG(`T`)`

- **Data:** trace T
- **Result:** TPG of T
- nodes $\leftarrow$ unique task IDs in T
- edges $\leftarrow$ mine_ops(T)
- TPG $\leftarrow$ (nodes, edges)
- TPG $\leftarrow$ transitive_closure(TPG)
- **return TPG**



Anomaly-based Intrusion Detection Systems

- Intrusion detection system (IDS) monitors a system/network for malicious activity
- Anomaly-based IDS identifies observations (packets, syscalls, etc.) which do not conform to an expected pattern observed during normal operation.
- IDS must be trustworthy, low false positive rate is often more important than low false negative rate
- Desired features of an IDS:
  - online operation
  - explainable results
- TPG represents an expected pattern for an anomaly-based IDS.

TPGs for Anomaly Detection

Train TPG on other traces and perform online anomaly detection:

TPGs for Anomaly Detection

Train TPG on other traces and perform online anomaly detection:

Data: trace stream $\mathbb{W}$, TPG G
Result: trained G or anomalous event ε in $\mathbb{W}$
foreach $\varepsilon \in W$ *in the increasing order of* $\varepsilon.time$ **do**
  $\upsilon \leftarrow \varepsilon$.task_id
  foreach τ *such that* $e(\tau, \upsilon) \in G$ **do**
    $u(\tau, \upsilon) \leftarrow$ append the # of occurrences of τ since previous occurrence of υ
    if $u(\tau, \upsilon)$ *is not a prefix of* $op(\tau, \upsilon)$ **then**
      if *training* **then** remove $e(\tau, \upsilon)$ from G; update G
      else return ε.time, $u(\tau, \upsilon)$, $op(\tau, \upsilon)$
  end
  if *training* **then return** G
  else return False
end

$$
\begin{aligned}
T &= \langle \\
op(\tau_2, \tau_7) &= \langle 1, 0, 0, 1, 0 \rangle \\
u(\tau_2, \tau_7) &= \langle
\end{aligned}
$$

TPGs for Anomaly Detection

Train TPG on other traces and perform online anomaly detection:

Data: trace stream $\mathbb{W}$, TPG G
Result: trained G or anomalous event ε in $\mathbb{W}$
foreach $\varepsilon \in W$ *in the increasing order of* $\varepsilon.time$ **do**
    $\upsilon \leftarrow \varepsilon$.task_id
    foreach τ *such that* $e(\tau, \upsilon) \in G$ **do**
        $u(\tau, \upsilon) \leftarrow$ append the # of occurrences of τ since previous occurrence of υ
        if $u(\tau, \upsilon)$ *is not a prefix of* $op(\tau, \upsilon)$ **then**
            if *training* **then** remove $e(\tau, \upsilon)$ from G; update G
            else return ε.time, $u(\tau, \upsilon)$, $op(\tau, \upsilon)$
    end
    if *training* **then return** G
    else return False
end

$$
\begin{aligned}
T &= \langle \tau_1, \tau_2, \\
op(\tau_2, \tau_7) &= \langle 1, 0, 0, 1, 0 \rangle \\
u(\tau_2, \tau_7) &= \langle 1 \rangle
\end{aligned}
$$

TPGs for Anomaly Detection

Train TPG on other traces and perform online anomaly detection:

Data: trace stream $\mathbb{W}$, TPG G
Result: trained G or anomalous event ε in $\mathbb{W}$
foreach $\varepsilon \in W$ *in the increasing order of* $\varepsilon.time$ **do**
    $\upsilon \leftarrow \varepsilon$.task_id
    foreach τ *such that* $e(\tau, \upsilon) \in G$ **do**
        $u(\tau, \upsilon) \leftarrow$ append the # of occurrences of τ since previous occurrence of υ
        if $u(\tau, \upsilon)$ *is not a prefix of* $op(\tau, \upsilon)$ **then**
            if *training* **then** remove $e(\tau, \upsilon)$ from G; update G
            else return ε.time, $u(\tau, \upsilon)$, $op(\tau, \upsilon)$
    end
    if *training* **then return** G
    else return False
end

$$
\begin{aligned}
T &= \langle \tau_1, \boldsymbol{\tau_2}, \tau_3, \tau_4, \tau_5, \tau_6, \boldsymbol{\tau_7} \\
op(\tau_2, \tau_7) &= \langle 1, 0, 0, 1, 0 \rangle \\
u(\tau_2, \tau_7) &= \langle 1 \rangle \qquad\qquad u(\tau_2, \tau_7) \text{ is a prefix of } op(\tau_2, \tau_7)
\end{aligned}
$$

TPGs for Anomaly Detection

Train TPG on other traces and perform online anomaly detection:

Data: trace stream $\mathbb{W}$, TPG G
Result: trained G or anomalous event ε in $\mathbb{W}$
foreach $\varepsilon \in W$ *in the increasing order of* $\varepsilon.time$ **do**
    $\upsilon \leftarrow \varepsilon$.task_id
    foreach τ *such that* $e(\tau, \upsilon) \in G$ **do**
        $u(\tau, \upsilon) \leftarrow$ append the # of occurrences of τ since previous occurrence of υ
        if $u(\tau, \upsilon)$ *is not a prefix of* $op(\tau, \upsilon)$ **then**
            if *training* **then** remove $e(\tau, \upsilon)$ from G; update G
            else return ε.time, $u(\tau, \upsilon)$, $op(\tau, \upsilon)$
    end
    if *training* **then return** G
    else return False
end

$$
\begin{aligned}
T &= \langle \tau_1, \boldsymbol{\tau_2}, \tau_3, \tau_4, \tau_5, \tau_6, \boldsymbol{\tau_7}, \tau_3, \tau_4, \boldsymbol{\tau_7} \\
op(\tau_2, \tau_7) &= \langle 1, 0, 0, 1, 0 \rangle \\
u(\tau_2, \tau_7) &= \langle 1, 0 \rangle \qquad\qquad u(\tau_2, \tau_7) \text{ is a prefix of } op(\tau_2, \tau_7)
\end{aligned}
$$

TPGs for Anomaly Detection

Train TPG on other traces and perform online anomaly detection:

Data: trace stream $\mathbb{W}$, TPG G
Result: trained G or anomalous event ε in $\mathbb{W}$
foreach $\varepsilon \in W$ *in the increasing order of* $\varepsilon.time$ **do**
    $\upsilon \leftarrow \varepsilon$.task_id
    foreach τ *such that* $e(\tau, \upsilon) \in G$ **do**
        $u(\tau, \upsilon) \leftarrow$ append the # of occurrences of τ since previous occurrence of υ
        if $u(\tau, \upsilon)$ *is not a prefix of* $op(\tau, \upsilon)$ **then**
            if *training* **then** remove $e(\tau, \upsilon)$ from G; update G
            else return ε.time, $u(\tau, \upsilon)$, $op(\tau, \upsilon)$
    end
    if *training* **then return** G
    else return False
end

$$
\begin{aligned}
T &= \langle \tau_1, \boldsymbol{\tau_2}, \tau_3, \tau_4, \tau_5, \tau_6, \boldsymbol{\tau_7}, \tau_3, \tau_4, \boldsymbol{\tau_7}, \tau_5, \boldsymbol{\tau_2} \\
op(\tau_2, \tau_7) &= \langle 1, 0, 0, 1, 0 \rangle \\
u(\tau_2, \tau_7) &= \langle 1, 0, 1 \rangle
\end{aligned}
$$

TPGs for Anomaly Detection

Train TPG on other traces and perform online anomaly detection:

Data: trace stream $\mathbb{W}$, TPG G
Result: trained G or anomalous event ε in $\mathbb{W}$
foreach $\varepsilon \in W$ *in the increasing order of* $\varepsilon.time$ **do**
    $v \leftarrow \varepsilon$.task_id
    foreach τ *such that* $e(\tau, v) \in G$ **do**
        $u(\tau, v) \leftarrow$ append the $\#$ of occurrences of τ since previous occurrence of v
        if $u(\tau, v)$ *is not a prefix of* $op(\tau, v)$ **then**
            if *training* **then** remove $e(\tau, v)$ from G; update G
            else return ε.time, $u(\tau, v)$, $op(\tau, v)$
    end
    if *training* **then return** G
    else return False
end

$$
\begin{aligned}
T &= \langle \tau_1, \boldsymbol{\tau_2}, \tau_3, \tau_4, \tau_5, \tau_6, \boldsymbol{\tau_7}, \tau_3, \tau_4, \boldsymbol{\tau_7}, \tau_5, \boldsymbol{\tau_2}, \tau_4, \boldsymbol{\tau_2} \\
op(\tau_2, \tau_7) &= \langle 1, 0, 0, 1, 0 \rangle \\
u(\tau_2, \tau_7) &= \langle 1, 0, 2 \rangle
\end{aligned}
$$

TPGs for Anomaly Detection

Train TPG on other traces and perform online anomaly detection:

Data: trace stream $\mathbb{W}$, TPG G
Result: trained G or anomalous event ε in $\mathbb{W}$
foreach $\varepsilon \in W$ *in the increasing order of* $\varepsilon.time$ **do**
    $v \leftarrow \varepsilon$.task_id
    foreach τ *such that* $e(\tau, v) \in G$ **do**
        $u(\tau, v) \leftarrow$ append the # of occurrences of τ since previous occurrence of v
        if $u(\tau, v)$ *is not a prefix of* $op(\tau, v)$ **then**
            if *training* **then** remove $e(\tau, v)$ from G; update G
            else return ε.time, $u(\tau, v)$, $op(\tau, v)$
    end
    if *training* **then return** G
    else return False
end

$$
\begin{aligned}
T &= \langle \tau_1, \boldsymbol{\tau_2}, \tau_3, \tau_4, \tau_5, \tau_6, \boldsymbol{\tau_7}, \tau_3, \tau_4, \boldsymbol{\tau_7}, \tau_5, \boldsymbol{\tau_2}, \tau_4, \boldsymbol{\tau_2}, \boldsymbol{\tau_7} \\
op(\tau_2, \tau_7) &= \langle 1, 0, 0, 1, 0 \rangle \\
u(\tau_2, \tau_7) &= \langle 1, 0, 2 \rangle \qquad\qquad u(\tau_2, \tau_7) \text{ is } \textbf{not} \text{ a prefix of } op(\tau_2, \tau_7)
\end{aligned}
$$

Evaluation in Automotive Context

Modern vehicles:

- becoming connected (internet, bluetooth, radars, etc.);
- becoming autonomous (automatic parking, autopilot).

[1] https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/
[2] https://www.pcmag.com/news/355281/tesla-model-s-hackers-return-for-encore-attack

Evaluation in Automotive Context

Modern vehicles:

- becoming connected (internet, bluetooth, radars, etc.);
- becoming autonomous (automatic parking, autopilot).

Security threats:

1. Remotely hack one or several ECUs, run malicious code on them.
2. Spoof the identity of the hacked ECUs.
3. Take over the control of the vehicle.

Examples:

- Miller and Valasek on Jeep Cherokee (2015) [1]
- Tencent Keen Lab on Tesla Model S (2016, 2017) [2]

[1] https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/
[2] https://www.pcmag.com/news/355281/tesla-model-s-hackers-return-for-encore-attack

Evaluation in Automotive Context (cont.)

Controller Area Network (CAN) bus:

- de-facto standard for interconnecting vehicle's ECUs;
- ECUs broadcast messages;
- no authentification;

CAN bus trace:

time	message ID	bytes	b1	b2	b3	b4	b5	b6	b7	b8
....	....	....	....	....	....	....	....	....	....	....
1478193190.235495	0545	8	d8	5f	00	8b	00	00	00	00
1478193190.235633	05f0	2	01	00						
1478193190.236088	0130	8	0b	80	00	ff	0f	80	0c	ea
1478193190.236324	0131	8	f2	7f	00	00	15	7f	0c	35
1478193190.236565	0140	8	00	00	00	00	1e	0d	2c	3b
1478193190.236815	02c0	8	15	00	00	00	00	00	00	00
....	....	....	....	....	....	....	....	....	....	....

Case Studies

CAN bus traces:

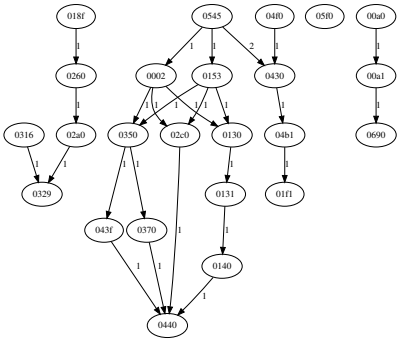
Study 1 Hyundai YF Sonata with injected and annotated spoofed messages (HCRL lab at Korea University [3])

Study 2 Production vehicle exercised on a test track (traces provided by our partner).

[3] http://ocslab.hksecurity.net/Datasets/CAN-intrusion-dataset

Case Study I

- 4 traces with annotated spoofed messages.
- spoofed messages have unique message IDs in two traces (DoS.csv and Fuzzy.csv):
  - use these traces to train a TPG;
  - train on "normal" parts: before the first spoofed message, 1000 events in each part.



The trained TPG

Case Study I (cont.)

Anomaly detection in the remaining 2 traces (*gear.csv* and *RPM.csv*):

- attacks were detected prior to the first spoofed message – very good:
  - 29 events before the first spoofed message in *gear.csv*
  - 7 events before the first spoofed message in *RPM.csv*
  - detected anomalies are probably caused by the activation of an external board used to inject spoofed messages.

Case Study I (cont.)

Anomaly detection in the remaining 2 traces (*gear.csv* and *RPM.csv*):

- attacks were detected prior to the first spoofed message – very good:
  - 29 events before the first spoofed message in *gear.csv*
  - 7 events before the first spoofed message in *RPM.csv*
  - detected anomalies are probably caused by the activation of an external board used to inject spoofed messages.
- false positive rates:
  - *gear.csv*: 0.019 (19 out of 1000 events)
  - *RPM.csv*: 0.007 (7 out of 1000 events)
  - (almost) all false positives are returned in the beginning of traces.

Observation: synchronization between analyzer and runtime needed

Case Study II

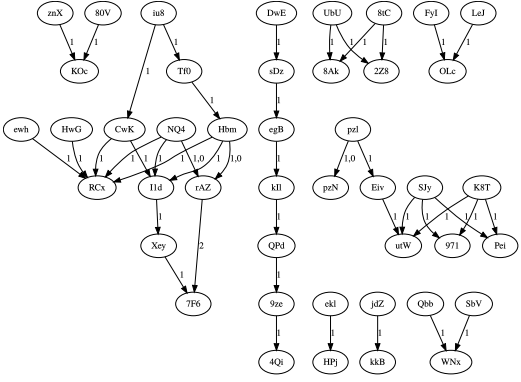
- Traces capture normal behavior of vehicle's ECUs, no anomalies
- The goal: evaluation of false positive rate
- 15 traces in total:

Trace	Duration (s)	# events	# tasks
1	312	793,653	252
2	245.5	653,932	240
3	361.3	907,038	224
4	390	1,044,558	251
5	352.6	891,195	235
6	287	727,508	243
7	288.7	767,148	239
8	307.2	1,260,299	243
9	600	1,425,667	219
10	343.2	866,187	229
11	319	847,882	241
12	249.9	635,463	235
13	342.1	991,105	276
14	314.3	834,942	257
15	283.8	754,786	239

Case Study II (cont.)

Evaluate the number of false positives (FPs):

1) build a TPG G using trace #1;
2) detect anomalies in trace #2 using G; detected anomalies are FPs.
3) refine G with trace #2;
4) detect anomalies in trace #3 using G; detected anomalies are FPs.
5) refine G with trace #3;
6) apply the same procedure on traces ##4-15.



The trained TPG. Only nodes with I/O edges shown

Trace	# FPs
1	—
2	140
3	220
4	11
5	1
6	38
7	2
8	9
9	0
10	0
11	0
12	0
13	0
14	0
15	0

Many Other Applications

- Insight and understanding for developers

Many Other Applications

- Insight and understanding for developers
- Documentation of (legacy) systems

Many Other Applications

- Insight and understanding for developers
- Documentation of (legacy) systems
- Debugging

Many Other Applications

- Insight and understanding for developers
- Documentation of (legacy) systems
- Debugging
- Safety assessment

Many Other Applications

- Insight and understanding for developers
- Documentation of (legacy) systems
- Debugging
- Safety assessment
- Runtime monitoring

Many Other Applications

- Insight and understanding for developers
- Documentation of (legacy) systems
- Debugging
- Safety assessment
- Runtime monitoring
- Model generation for simulation and optimization

Many Other Applications

- Insight and understanding for developers
- Documentation of (legacy) systems
- Debugging
- Safety assessment
- Runtime monitoring
- Model generation for simulation and optimization
- ...

Conclusion

- **TPG_miner**: an approach to reverse-engineer real-time embedded systems via mining task precedence graphs from system traces

Conclusion

- **TPG_miner**: an approach to reverse-engineer real-time embedded systems via mining task precedence graphs from system traces
- Evaluation in the context of automotive security

Conclusion

- **TPG_miner**: an approach to reverse-engineer real-time embedded systems via mining task precedence graphs from system traces

- Evaluation in the context of automotive security
  - can be used as an anomaly-based intrusion detection system

Conclusion

- **TPG_miner**: an approach to reverse-engineer real-time embedded systems via mining task precedence graphs from system traces

- Evaluation in the context of automotive security
  - can be used as an anomaly-based intrusion detection system
  - straightforward application on CAN bus traces

Conclusion

- **TPG_miner**: an approach to reverse-engineer real-time embedded systems via mining task precedence graphs from system traces

- Evaluation in the context of automotive security
  - can be used as an anomaly-based intrusion detection system
  - straightforward application on CAN bus traces
- Detects anomalies in streaming data

Conclusion

- **TPG_miner**: an approach to reverse-engineer real-time embedded systems via mining task precedence graphs from system traces

- Evaluation in the context of automotive security
  - can be used as an anomaly-based intrusion detection system
  - straightforward application on CAN bus traces
- Detects anomalies in streaming data
- Explains which expected system behavior has been violated

Conclusion

- **TPG_miner**: an approach to reverse-engineer real-time embedded systems via mining task precedence graphs from system traces

- Evaluation in the context of automotive security
  - can be used as an anomaly-based intrusion detection system
  - straightforward application on CAN bus traces
- Detects anomalies in streaming data
- Explains which expected system behavior has been violated
- Requires little training to achieve low false positive rate

Conclusion

- **TPG_miner**: an approach to reverse-engineer real-time embedded systems via mining task precedence graphs from system traces

- Evaluation in the context of automotive security
  - can be used as an anomaly-based intrusion detection system
  - straightforward application on CAN bus traces
- Detects anomalies in streaming data
- Explains which expected system behavior has been violated
- Requires little training to achieve low false positive rate

- Implemented in R: `https://bitbucket.org/oiegorov/tpg_miner`

Conclusion

- **TPG_miner**: an approach to reverse-engineer real-time embedded systems via mining task precedence graphs from system traces

- Evaluation in the context of automotive security
  - can be used as an anomaly-based intrusion detection system
  - straightforward application on CAN bus traces
- Detects anomalies in streaming data
- Explains which expected system behavior has been violated
- Requires little training to achieve low false positive rate

- Implemented in R: `https://bitbucket.org/oiegorov/tpg_miner`
  - the README file explains how to reproduce results of Case Study I.