INFUSE: Towards Efficient Context Consistency by Incremental-Concurrent Check Fusion

Lingyu Zhang, Huiyan Wang, Chang Xu, and Ping Yu
Context-aware Computing

• **Context**: a model of applications’ runtime environments\textsuperscript{[9-12]}
  • E.g., GPS data, speed, temperature, picture, etc.

• **Usage**: applications’ smart adaption based on contexts facilitate people’s lives

SmartHome application
Context Problem

• Quality problems: inaccurate, incomplete, or conflicting with each other due to uncontrollable sensor instability[^9-12]

• Unexpected consequence: leading to applications’ misbehaviors or crashes

[^9-12]: Misbehavior: improper temperature
Common Practice

• Constraint checking: checking contexts against *consistency constraints* to see whether any *violation* (named *context inconsistency*) occurs

• Constraint example: “no robot can be in two rooms at the same time”

\[ S_{loc} : \forall v_x \in R_x \ (\text{not} \ (\exists v_y \in R_y \ (\text{Same}(v_x, v_y)))) \]
Existing Constraint Checking Techniques

• Two typical research lines

Multi-threads

Concurrent checking (Con-C\(^{[11]}\))

Split checking workload into units carrying similar workloads

Full checking (ECC\(^{[5]}\))

Reduce redundant computing by analyzing reusable results

Incremental checking (PCC\(^{[10]}\))

Reusable results (gray parts)
Low-efficiency Problem of Existing Techniques

• Features of context in nowadays dynamic environment: large volume and changing frequently

• Bringing unacceptable overhead to existing techniques

<table>
<thead>
<tr>
<th>Technique</th>
<th>Time cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECC</td>
<td>19.1 ~ 137.7 h</td>
</tr>
<tr>
<td>Con-C</td>
<td>11.2 ~ 68.0 h</td>
</tr>
<tr>
<td>PCC</td>
<td>3.3 ~ 5.9 h</td>
</tr>
</tbody>
</table>

Time cost of existing techniques for handling one-hour context data in SmartCity application (1.7 million data lines and 48 constraints)

Call for more efficient checking techniques

Cannot validate in time
Our Natural Idea: Check Fusion

- Two orthogonal dimensions: PCC (2006) and Con-C (2013)

The more, the better
small workload not suitable to split
Con-C’s underlying assumption

The less, the better
large workload hard to analyze
PCC’s underlying assumption

No substantial work after one decade since their initial proposals
Two Brute-Force Solutions Do not Work

- Respect “the less, the better”: splitting small workload into concurrent units

<table>
<thead>
<tr>
<th>Technique</th>
<th>Checking time</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFUSE$_0$</td>
<td>$\geq 46.7$ min</td>
</tr>
<tr>
<td>PCC</td>
<td>$26.4 \sim 44.2$ min</td>
</tr>
</tbody>
</table>

Checking time comparison from our evaluation

Performance compromise: even less efficient than pure incremental checking
Two Brute-Force Solutions Do not Work

- Respect “the more, the better”: enlarging workload for fusion checking

Interference between workloads leads to wrong checking results

PCC’s semantics was not designed for multiple workloads
Two Faced Problems

- Summary of two brute-force solutions
  - Respect “the less, the better”: correct but inefficient
  - Respect “the more, the better”: efficient but incorrect

- Two problems for achieving both correctness and efficiency
  - What-To-Check: Which workloads can be checked together?
  - How-To-Check: How to correctly conduct fusion checking for multiple workloads?
What-To-Check: Example

- Robot localization application
  - Three robots ($r_1$, $r_2$, and $r_3$) move between two rooms (x and y)

Context: robots in a room at a certain time

Timeline:

1. $r_3$ enters room $y$
2. $r_2$ leaves room $y$
3. $r_2$ enters room $y$
4. $r_3$ leaves room $y$
5. $r_3$ enters room $x$
6. $r_3$ leaves room $x$

Robots’ movements induce context changes to update contexts along the timeline

Missed by sensor
What-To-Check: Example

\[ S_{\text{loc}}: \forall v_x \in R_x \ (\text{not} \ (\exists v_y \in R_y \ (\text{Same}(v_x, v_y)))) \]
What-To-Check: Example

Correct but time consuming

\[ S_{loc}: \forall v_x \in R_x \ (\text{not} \ (\exists \ v_y \in R_y \ (\text{Same}(v_x, v_y)))) \]
What-To-Check: Example

Cannot compromise validity for efficiency

\[ S_{loc}: \forall v_x \in R_x \ (\text{not} \ (\exists v_y \in R_y \ (\text{Same}(v_x, v_y)))) \]

Not valid

Inc={v_x = r_3, v_y = r_3}

No detected incs
Pursuing Efficiency with Validity Guarantee

• Goal: composing a group with context changes *as many as possible* while guaranteeing the correctness of checking results
How to Pursue Efficiency with Validity Guaranteed?

What makes the two groups different in validity?

\[ S_{loc} : \forall v_x \in R_x \text{ (not } (\exists v_y \in R_y \text{ (Same}(v_x, v_y)))) \]

Inc = \{v_x = r_3, v_y = r_3\}  

Not valid  

Valid  

Inc = \{v_x = r_3, v_y = r_3\}  

No detected incs
Interference Between Changes Breaks Validity

How many roles can context changes play concerning inconsistency occurrence?

Timeline

Not valid

Valid

No inc

Expose the inc

Inc={\(v_x = r_3, v_y = r_3\)}

Hide the inc

No inc

Different roles
Validity Criterion

How to know the role of a context change?

• Three roles of context changes concerning inconsistency occurrence

<table>
<thead>
<tr>
<th>E-change</th>
<th>H-change</th>
<th>I-change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possibly expose new</td>
<td>Possibly hide existing</td>
<td>Irrelevant to any</td>
</tr>
<tr>
<td>inconsistencies</td>
<td>inconsistencies</td>
<td>inconsistency</td>
</tr>
</tbody>
</table>

• Interference: E-change followed by H-change (may not be contiguous)

• Validity criterion: avoiding any interference in a group
Knowing the Role by Bottom-up Derivation

\[ S_{loc}: \forall v_x \in R_x \text{ (not } (\exists v_y \in R_y \text{ (Same}(v_x, v_y)))) \]

- **E-change**: \[ \forall v_x \in R_x \]

- **Not**: \[ \text{not} \]

- **Same**: \[ \text{Same}(v_x, v_y) \]

\[ \exists v_y \in R_y \]

\[ \text{chg}_1: <+ , R_y, r_3^> \]

- **Derivation rules**

  - **impact(chg, \forall v \in C(f))** =
    1. base_impact(chg, v), when chg affects C,
    2. impact(chg, f) \cup \{m_{FF}\}, when chg affects f;

  - **impact(chg, \exists v \in C(f))** =
    1. base_impact(chg, \exists), when chg affects C,
    2. impact(chg, f) \cup \{m_{FF}\}, when chg affects f;

  - **impact(chg, not (f)) = flipSet(impact(chg, f));**

  - **impact(chg, (f_1) and (f_2)) =**
    1. impact(chg, f_1) \cup \{m_{FF}\}, when chg affects f_1,
    2. impact(chg, f_2) \cup \{m_{FF}\}, when chg affects f_2;

  - **impact(chg, (f_1) or (f_2)) =**
    1. impact(chg, f_1) \cup \{m_{TT}\}, when chg affects f_1,
    2. impact(chg, f_2) \cup \{m_{TT}\}, when chg affects f_2;

  - **impact(chg, \underline{(f_1)} implies \underline{(f_2))} =**
    1. flipSet(\underline{(impact(chg, f_1))} \cup \{m_{TT}\}, when chg affects f_1,
    2. impact(chg, f_2) \cup \{m_{TT}\}, when chg affects f_2.

**Efficient**: only related to static structure of constraints and previous checking results
Composing Groups by Validity Criterion

Timeline

chg_1

E-change

No interference

chg_2

I-change

No interference

chg_3

E-change

No interference

chg_4

E-change

No interference

chg_5

H-change

Interference
Two Faced Problems

• What-To-Check
  • Which workloads can be checked together for enlarging workload?
  

• How-To-Check
  • How to correctly conduct fusion checking after enlarging workload?

PCC’s semantics was not designed for multiple workloads
PCC Originally Designed for Single Context Change

- Incremental checking in PCC

- Incremental checking we need

Core of incremental checking: difference of contexts before and after changing

How to know the difference of contexts between $P_0$ and $P_4$?

Require new semantics
Cumulative Effects Show the Difference

- Accumulate context changes’ effects on contexts in their temporal orders

<table>
<thead>
<tr>
<th>ASet</th>
<th>DSet</th>
<th>USet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Truly added elements</td>
<td>Truly deleted elements</td>
<td>Updated elements</td>
</tr>
</tbody>
</table>

Cumulative effects
Extending PCC’s Capability to a Group of Changes

- Divide incremental checking into several mutually exclusive cases according to cumulative effects

### ASet

<table>
<thead>
<tr>
<th></th>
<th>(\emptyset)</th>
<th>({a})</th>
<th>(\emptyset)</th>
<th>(\emptyset)</th>
<th>({a})</th>
<th>({a})</th>
<th>({a})</th>
</tr>
</thead>
</table>

When subformula not affected (Affected\((f) = F\))

### Dset

<table>
<thead>
<tr>
<th></th>
<th>(\emptyset)</th>
<th>({d})</th>
<th>(\emptyset)</th>
<th>({d})</th>
<th>(\emptyset)</th>
<th>({d})</th>
<th>({d})</th>
</tr>
</thead>
</table>

### USet

<table>
<thead>
<tr>
<th></th>
<th>(\emptyset)</th>
<th>(\emptyset)</th>
<th>(\emptyset)</th>
<th>({u})</th>
<th>({u})</th>
<th>({u})</th>
<th>(\emptyset)</th>
</tr>
</thead>
</table>

When subformula affected (Affected\((f) = T\))

\[ \text{Incremental semantics for universal formula} \]

\[ \tau_{\text{partial}}[\forall v \in C(f)]_\alpha = \]

1. \(\tau_0[\forall v \in C(f)]_\alpha\), if \(\text{Affected}(f) = F\) and \((\text{ASet} = \emptyset\) and \(\text{DSet} = \emptyset\) and \(\text{USet} = \emptyset\))

2. \(\tau_0[\forall v \in C(f)]_\alpha \wedge t_1 \wedge \cdots \wedge t_a\), where \((t_1, \ldots, t_a) = \text{eval}_{\text{partial}}(\tau[f]\text{bind}((v,y_j),\alpha) | y_j \in \text{ASet}),\)
   if \(\text{Affected}(f) = F\) and \((\text{ASet} \neq \emptyset\) and \(\text{DSet} = \emptyset\) and \(\text{USet} = \emptyset\))

3. \(\tau_0[\forall v \in C(f)]_\alpha \wedge t_1 \wedge \cdots \wedge t_a + u\) | \(x_i \in C - (\text{ASet} \cup \text{USet})\),
   where \((t_1, \ldots, t_a + u) = \text{eval}_{\text{partial}}(\tau[f]\text{bind}((v,y_j),\alpha) | y_j \in \text{ASet} \cup \text{USet}),\)
   if \(\text{Affected}(f) = F\) and \((\text{DSet} \neq \emptyset\) or \(\text{USet} \neq \emptyset\)).

4. \(\tau_0[\forall v \in C(f)]_\alpha \wedge t_1 \wedge \cdots \wedge t_n\), where \((t_1, \ldots, t_n) = \text{eval}_{\text{partial}}(\tau[f]\text{bind}((v,x_i),\alpha) | x_i \in C),\)
   if \(\text{Affected}(f) = T\) and \((\text{ASet} = \emptyset\) and \(\text{DSet} = \emptyset\) and \(\text{USet} = \emptyset\)).

5. \(\tau_0[\forall v \in C(f)]_\alpha \wedge t_1 \wedge \cdots \wedge t_n\), where \((t_1, \ldots, t_a + u) = \text{eval}_{\text{partial}}(\tau[f]\text{bind}((v,y_j),\alpha) | y_j \in \text{ASet} \cup \text{USet})\)
   and \((t_{a+u+1}, \ldots, t_n) = \text{eval}_{\text{partial}}(\tau[f]\text{bind}((v,x_i),\alpha) | x_i \in C - (\text{ASet} \cup \text{USet})),\)
   if \(\text{Affected}(f) = T\) and \((\text{ASet} \neq \emptyset\) or \(\text{DSet} \neq \emptyset\) or \(\text{USet} \neq \emptyset\)).
Concurrent Point Selection

- **Concurrent point**: indicating where concurrent checking starts

- **Selection criterion**: the highest affected universal or existential formula

**Sufficient workload to be split into units**

**Units contain similar workloads due to different variable assignments**

\[
\forall v_x \in R_x
\]

\[
\exists v_y \in R_y
\]

\[
\text{Same}(v_x, v_y)
\]
Adaptive Switching among Different Semantics

**Incremental semantics**

\[
\tau_{\text{partial}}[\forall u \in C(f)] = \\
(1) \tau_0[\forall u \in C(f)]_0, \text{ if Affected}(f) = F \text{ and } (ASet = \emptyset \text{ and } DSet = \emptyset \text{ and } USet = \emptyset). \\
(2) \tau_0[\forall u \in C(f)]_0 \land t_1 \land \cdots \land t_a, \text{ where } (t_1, \cdots, t_a) = \text{eval}_{\text{partial}}(\tau[f]_{\text{bind}}((v, y_j), \alpha) \mid y_j \in ASet), \\
\text{if Affected}(f) = F \text{ and } (ASet \neq \emptyset \text{ and } DSet = \emptyset \text{ and } USet = \emptyset). \\
(3) T \land \tau_0[f]_{\text{bind}}((v, x_z-a-u), \alpha) \land t_1 \land \cdots \land t_{a+u} \mid x_i \in C - (ASet \cup USet), \\
\text{where } (t_1, \cdots, t_{a+u}) = \text{eval}_{\text{partial}}(\tau[f]_{\text{bind}}((v, y_j), \alpha) \mid y_j \in ASet \cup USet), \\
\text{if Affected}(f) = F \text{ and } (ASet \neq \emptyset \text{ or } USet \neq \emptyset). \\
(4) T \land t_1 \land \cdots \land t_n, \text{ where } (t_1, \cdots, t_n) = \text{eval}_{\text{partial}}(\tau[f]_{\text{bind}}((v, x_i), \alpha) \mid x_i \in C), \\
\text{if Affected}(f) = T \text{ and } (ASet = \emptyset \text{ and } DSet = \emptyset \text{ and } USet = \emptyset). \\
(5) T \land t_1 \land \cdots \land t_n, \text{ where } (t_1, \cdots, t_{a+u}) = \text{eval}_{\text{partial}}(\tau[f]_{\text{bind}}((v, y_j), \alpha) \mid y_j \in ASet \cup USet) \\
\text{and } (t_{a+u+1}, \cdots, t_n) = \text{eval}_{\text{partial}}(\tau[f]_{\text{bind}}((v, z_i), \alpha) \mid x_i \in C - (ASet \cup USet), \\
\text{if Affected}(f) = T \text{ and } (ASet \neq \emptyset \text{ or } DSet \neq \emptyset \text{ or } USet \neq \emptyset).
\]

**Concurrent semantics**

\[
\tau_{\text{entire}}[\forall u \in C(f)] = \\
T \land \tau_{\text{entire}}[f]_{\text{bind}}((v, x_z), \alpha) \land \cdots \land \tau_{\text{entire}}[f]_{\text{bind}}((v, x_n), \alpha) \mid x_i \in C
\]

**Full semantics**

\[
\tau_{\text{entire}}[\forall u \in C(f)] = \\
(1) \tau_{\text{entire}}[f]_{\text{bind}}((v, x_z), \alpha) \mid x_i \in C
\]

---

When further checking required

Concurrent or not

How to further check
Two Faced Problems

• **What-To-Check**
  • Which workloads can be checked together for enlarging workload?

• **How-To-Check**
  • How to correctly conduct fusion checking after enlarging workload?
Theoretical Guarantee

• What-To-Check
  • Which workloads can be checked together for enlarging workload?

  WHAT-Correctness Theorem
  *Given any consistency constraint and associated context pool, INFUSE produces the same result for its arranged valid context changes, no matter it checks these changes as a whole or individually.*

• How-To-Check
  • How to correctly conduct fusion checking after enlarging workload?

  HOW-Correctness Theorem
  *Given any consistency constraint and associated context pool, INFUSE produces the same result by its check fusion semantics, as existing constraint checking techniques do.*
Evaluation

• Research Questions
  • RQ1 (Motivation): How do existing constraint checking techniques behave when handling large-volume dynamic contexts? (already shown earlier)
  
  • RQ2 (Effectiveness): How effective is INFUSE in constraint checking for detecting context inconsistencies, as compared with existing techniques?
  
  • RQ3 (Practical Usage): How effective is INFUSE in constraint checking under real-life settings?
Experimental Design and Set Up

• Subjects
  • SmartCity application with 4.3 million vehicle data (e.g., GPS data, speed, direction) and 48 consistency constraints (also used in existing work\textsuperscript{[9-12]} for evaluation)

• Workloads
  • Three distinct hour-based groups of data with light (311,240 changes), median (843,686 changes) and heavy (1,664,900 changes) workloads

• Techniques for comparison
  • Two versions in our work: INFUSE (elite version), INFUSE\textsubscript{0} (brute-force version)
  • Existing techniques and their improved versions: ECC\textsubscript{O}\textsuperscript{[5]}, ECC\textsubscript{G}, Con-C\textsubscript{O}\textsuperscript{[11]}, Con-C\textsubscript{G}, PCC\textsubscript{O}\textsuperscript{[10]}, PCC\textsubscript{G}
RQ2 (Effectiveness)

• Checking time comparison for all techniques on all workloads

- Time comparison on light workload
  - Most efficient with 0.0x-18.6x improvement
  - INFUSE was the most efficient technique on all workloads

- Time comparison on median workload
  - Most efficient with 2.4x-105.4x improvement

- Time comparison on heavy workload
  - Most efficient with 3.1x-171.1x improvement
RQ2 (Effectiveness)

• Checking time comparison for all techniques on all workloads

![Graphs showing time comparison for different workloads for INFUSE and INFUSE₀.](image)

- 0.4x efficiency improvement for INFUSE₀ in light workload.
- 5.1x efficiency improvement for INFUSE₀ in median workload.
- 6.0x efficiency improvement for INFUSE₀ in heavy workload.

Valid Context Change Groups

Fusion Soundness

Difference between INFUSE and INFUSE₀ was large and kept increasing.
RQ3 (Practical usage)

- Simulate real-life settings according to real timestamps

<table>
<thead>
<tr>
<th>Workload</th>
<th>Checking techniques</th>
<th>Oracle incs (#)</th>
<th>Reported incs/* (#)</th>
<th>$T_{\text{cost}}$(s)</th>
<th>$R_{FN}$ (%)</th>
<th>$R_{FP}$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light</td>
<td>ECC&lt;sub&gt;O&lt;/sub&gt;</td>
<td>3,254</td>
<td>3,254</td>
<td>128.6</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td></td>
<td>Con-C&lt;sub&gt;O&lt;/sub&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PCC&lt;sub&gt;O&lt;/sub&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ECC&lt;sub&gt;G&lt;/sub&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Con-C&lt;sub&gt;G&lt;/sub&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PCC&lt;sub&gt;G&lt;/sub&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INFUSE&lt;sub&gt;0&lt;/sub&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INFUSE</td>
<td></td>
<td></td>
<td>3,254</td>
<td>10.8</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

All techniques reported correct checking results, but INFUSE took the least time.

False negative rate
False positive rate
RQ3 (Practical usage)

- False negative/positive rates are more crucial since they reflect correctness.

The less, the better

<table>
<thead>
<tr>
<th>Workload</th>
<th>Checking techniques</th>
<th>Oracle incs (#)</th>
<th>Reported incs/* (#)</th>
<th>$T_{cost}$(s)</th>
<th>$R_{FN}$ (%)</th>
<th>$R_{FP}$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
<td>ECC$_O$</td>
<td>8,647/694*</td>
<td>3,850.9</td>
<td>96.8%</td>
<td>92.0%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Con-C$_O$</td>
<td>14,209/897*</td>
<td>3,593.9</td>
<td>95.8%</td>
<td>93.7%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PCC$_O$</td>
<td>20,942/19,369*</td>
<td>1,513.7</td>
<td>9.6%</td>
<td>7.5%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ECC$_G$</td>
<td>20,412/1,415*</td>
<td>3,588.4</td>
<td>93.4%</td>
<td>93.1%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Con-C$_G$</td>
<td>20,779/19,293*</td>
<td>1,950.8</td>
<td>10.0%</td>
<td>7.2%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PCC$_G$</td>
<td>21,377/19,414*</td>
<td>1,099.7</td>
<td>9.4%</td>
<td>9.2%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INFUSE$_0$</td>
<td>20,922/19,371*</td>
<td>1,588.5</td>
<td>9.6%</td>
<td>7.4%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INFUSE</td>
<td>21,436</td>
<td>456.6</td>
<td>0.0%</td>
<td>0.0%</td>
<td></td>
</tr>
</tbody>
</table>

| Heavy    | ECC$_O$             | 4,934/392*     | 4,032.1             | 98.7%       | 92.1%        |
|          | Con-C$_O$           | 6,611/463*     | 3,748.2             | 98.4%       | 93.0%        |
|          | PCC$_O$             | 22,574/1,028*  | 3,410.8             | 96.5%       | 95.5%        |
|          | ECC$_G$             | 14,617/801*    | 3,574.8             | 97.3%       | 94.5%        |
|          | Con-C$_G$           | 20,824/957*    | 3,375.5             | 96.8%       | 95.4%        |
|          | PCC$_G$             | 29,115/1,178*  | 3,594.4             | 96.0%       | 96.0%        |
|          | INFUSE$_0$          | 22,302/1,013*  | 3,463.2             | 96.6%       | 95.5%        |
|          | INFUSE              | 29,642         | 2954.6              | 0.0%        | 0.0%         |

INFUSE still took the least time

INFUSE still reported correct checking results while others suffered from varying degrees of quality problems.

Effective under real-life settings.
Conclusion and Future Work

• Work summary
  • Addressed what-to-check and how-to-check problems of fusion checking with theoretical guarantee
  • 18.6x–171.1x speed up to existing techniques with quality guarantees

• Future work
  • Less conservative grouping strategy
  • Adaptive concurrency control
Thank you!

Comments are welcome!

Email: zly@smail.nju.edu.cn
Experimental Results of all 24 hours data (1)

Most efficient with 3.0x-120.3x efficiency improvement
Experimental Results of all 24 hours data (2)

INFUSE’s efficiency improvement over existing checking techniques on 24 hour-based groups (sorted by increasing workloads)

With the growth of workload, INFUSE’s efficiency improvement generally hold a stably increasing trend.
The less red color the better

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ECC</td>
<td>T_{train}(min)</td>
<td>14.8</td>
<td>18.4</td>
<td>59.1</td>
<td>61.7</td>
<td>64.2</td>
<td>62.3</td>
<td>64.8</td>
<td>62.6</td>
<td>62.6</td>
<td>62.6</td>
<td>62.6</td>
<td>62.6</td>
<td>62.6</td>
<td>62.6</td>
<td>62.6</td>
<td>62.6</td>
<td>62.6</td>
<td>62.6</td>
<td>62.6</td>
<td>62.6</td>
<td>62.6</td>
<td>62.6</td>
<td>62.6</td>
</tr>
<tr>
<td>ECC</td>
<td>R_{Abs} (%)</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
</tr>
<tr>
<td>ECC</td>
<td>R_{FPR} (%)</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
</tr>
</tbody>
</table>

INFUSE achieved zero false negative and positive for almost all groups.