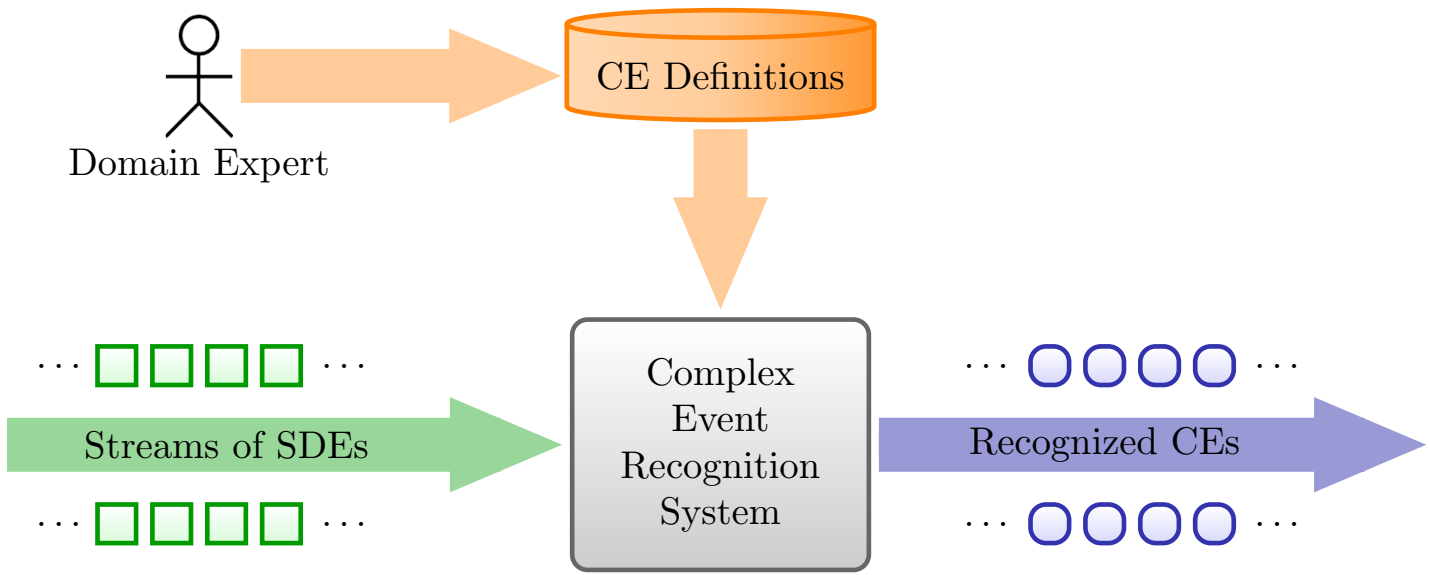
Towards A Simple Event Calculus for Run-Time Reasoning

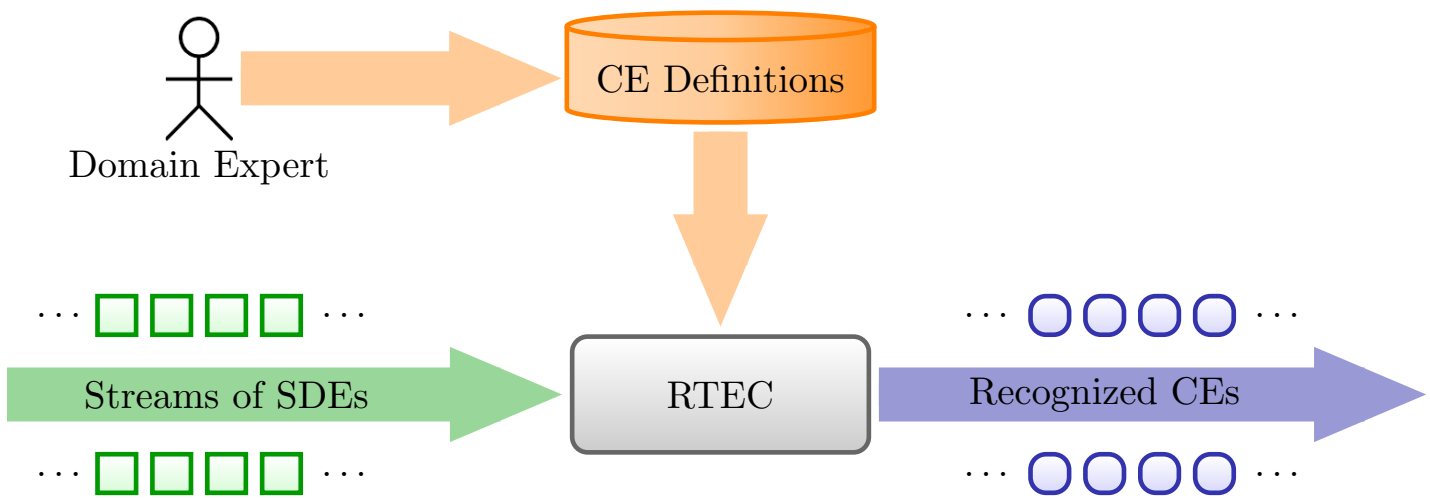
Christos Vlassopoulos[1,2] Alexander Artikis[3,1]
(http://cer.iit.demokritos.gr)

[1]Institute of Informatics and Telecommunications, NCSR "Demokritos", Athens, Greece
[2]Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Greece
[3]Department of Maritime Studies, University of Piraeus, Greece

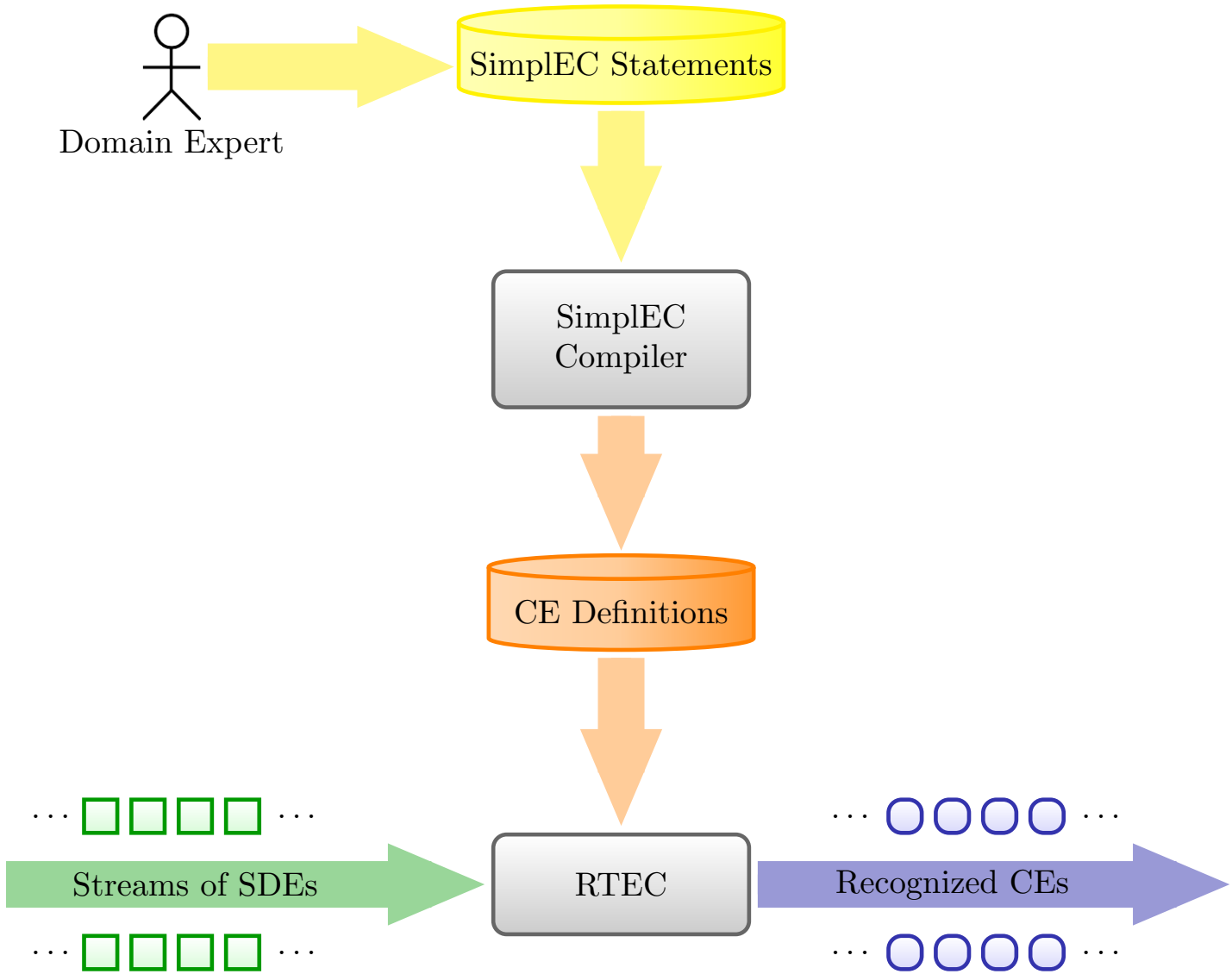
Complex Event Recognition



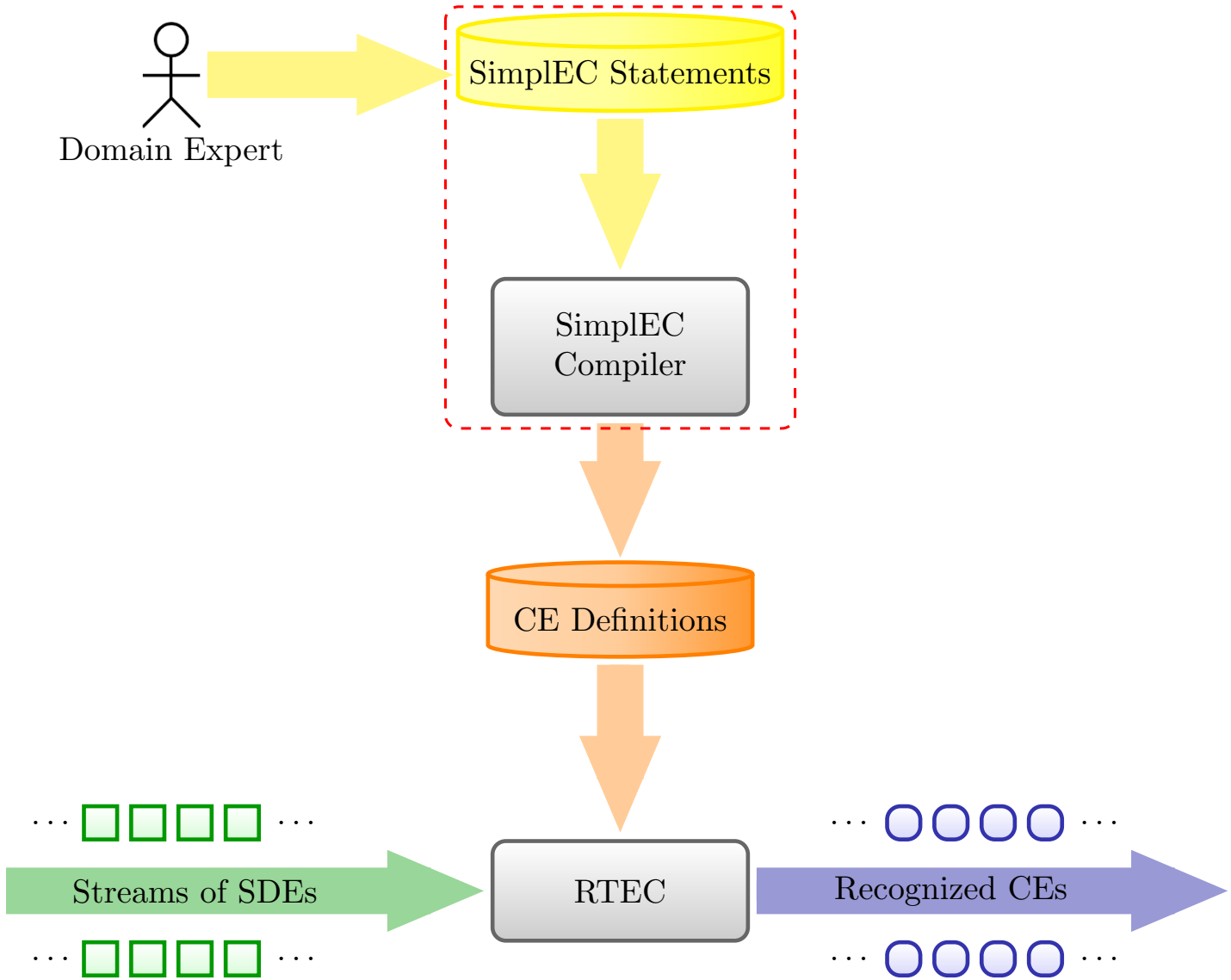
Complex Event Recognition



Complex Event Recognition



Complex Event Recognition



Event Calculus

- A logic programming language for representing and reasoning about events and their effects.
- Key components:
  - event (typically instantaneous).
  - fluent: a property that may have different values at different points in time.
- Built-in representation of inertia:
  - $F = V$ holds at a particular time-point if $F = V$ has been *initiated* by an event at some earlier time-point, and not *terminated* by another event in the meantime.

Run-Time Event Calculus (RTEC)

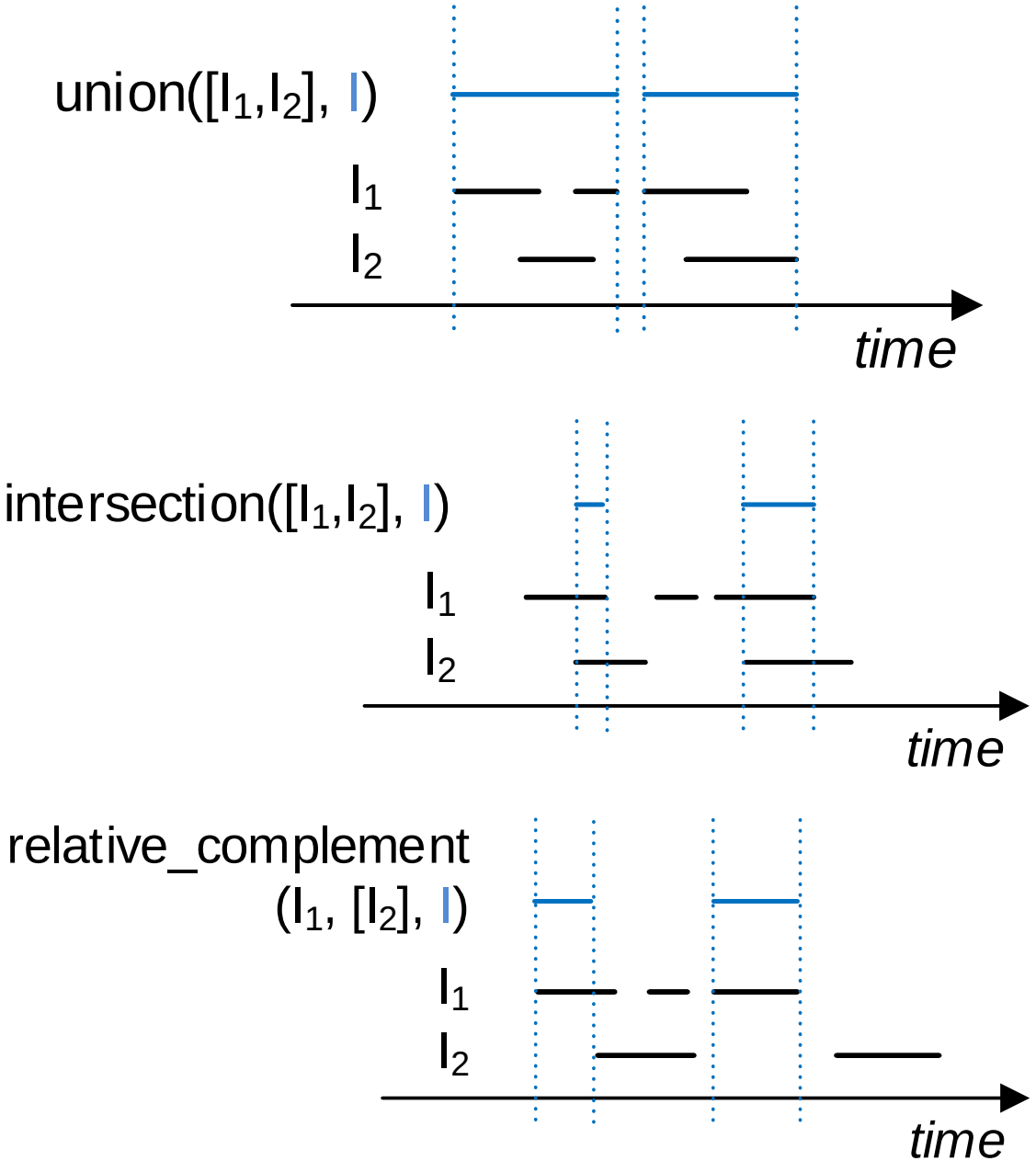
A logic programming implementation of the Event Calculus.

- Continuous narrative assimilation queries on data streams.
- Linear time model, support for multi-valued fluents.
- Windowing and Forgetting mechanisms.
- Hierarchies and Caching.

Run-Time Event Calculus (RTEC)

Predicate	Meaning
happensAt(E, T)	Event E occurs at time T
initiatedAt$(F = V, T)$	At time T a period of time for which $F = V$ is initiated
terminatedAt$(F = V, T)$	At time T a period of time for which $F = V$ is terminated
holdsFor$(F = V, I)$	I is the list of the maximal intervals for which $F = V$ holds continuously
holdsAt$(F = V, T)$	The value of fluent F is V at time T
union_all$([J_1, \ldots, J_n],\ I)$	$I = (J_1 \cup \ldots \cup J_n)$
intersect_all$([J_1, \ldots, J_n],\ I)$	$I = (J_1 \cap \ldots \cap J_n)$
relative_complement_all $(I',\ [J_1, \ldots, J_n],\ I)$	$I = I' \setminus (J_1 \cup \ldots \cup J_n)$

Interval Manipulation



Complex Event Definitions in RTEC: Simple Fluents

initiatedAt$(CE,\ T) \leftarrow$
happensAt$(E_{In_1},\ T)$,
[conditions]

...

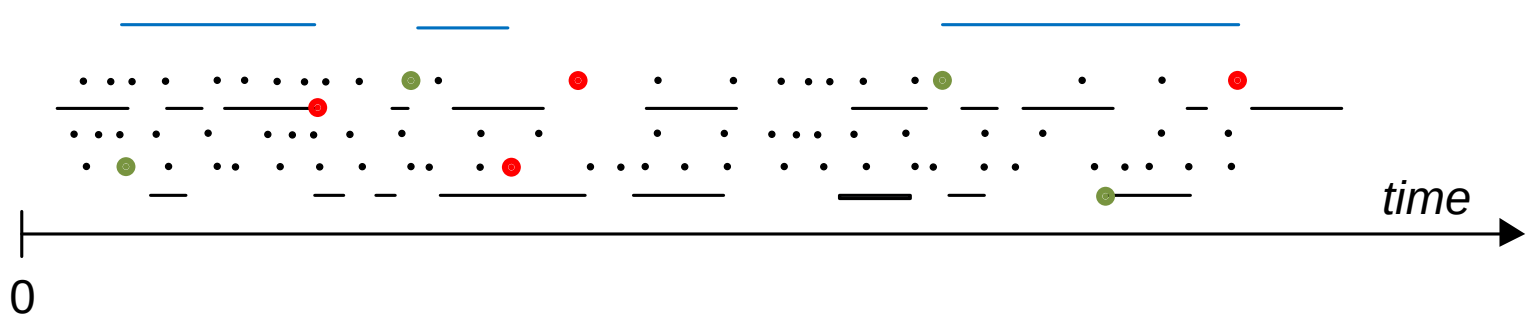
initiatedAt$(CE,\ T) \leftarrow$
happensAt$(E_{In_i},\ T)$,
[conditions]

terminatedAt$(CE,\ T) \leftarrow$
happensAt$(E_{T_1},\ T)$,
[conditions]

...

terminatedAt$(CE,\ T) \leftarrow$
happensAt$(E_{T_j},\ T)$,
[conditions]

Complex Event recognition: **holdsFor**$(CE,\ I)$



Complex Event Definitions in RTEC: Simple Fluents

Complex Event definition:

$$\begin{aligned}
&\textbf{initiatedAt}(\mathit{leaving_object}(P, Obj) = \text{true},\ T) \leftarrow \\
&\quad \textbf{happensAt}(\mathit{appear}(Obj),\ T), \\
&\quad \textbf{holdsAt}(\mathit{inactive}(Obj) = \text{true},\ T), \\
&\quad \textbf{holdsAt}(\mathit{close}(P, Obj) = \text{true},\ T), \\
&\quad \textbf{holdsAt}(\mathit{person}(P) = \text{true},\ T) \\
&\textbf{terminatedAt}(\mathit{leaving_object}(P, Obj) = \text{true},\ T) \leftarrow \\
&\quad \textbf{happensAt}(\mathit{disappear}(Obj),\ T)
\end{aligned}$$

Complex Event Definitions in RTEC: Statically Determined Fluents

$$\begin{aligned}
&\textbf{holdsFor}(CE,\ I)\ \leftarrow \\
&\quad \textbf{holdsFor}(F_1,\ I_{F_1}), \\
&\quad \ldots, \\
&\quad \textbf{holdsFor}(F_f,\ I_{F_f}), \\
&\quad \mathit{interval_manipulation}_1(I_\alpha, \ldots, I_\omega), \\
&\quad \ldots, \\
&\quad \mathit{interval_manipulation}_k(I_A, \ldots, I_\Omega)
\end{aligned}$$

where

$$\begin{aligned}
&\mathit{interval_manipulation}(I_1, \ldots, I_n,\ I) : \\
&\quad \mathsf{union}([I_1, \ldots, I_n],\ I) \\
&\quad \mathsf{intersection}([I_1, \ldots, I_n],\ I) \\
&\quad \mathsf{relative_complement}(I_1,\ [I_2, \ldots, I_n],\ I)
\end{aligned}$$

Complex Event Definitions in RTEC: Statically Determined Fluents

Complex Event definition:

$$\begin{aligned}
&\textbf{holdsFor}(\mathit{fighting}(P_1, P_2) = \text{true},\ I) :- \\
&\quad \textbf{holdsFor}(\mathit{abrupt}(P_1) = \text{true},\ I_1), \\
&\quad \textbf{holdsFor}(\mathit{abrupt}(P_2) = \text{true},\ I_2), \\
&\quad \textbf{union_all}([I_1, I_2],\ I_3), \\
&\quad \textbf{holdsFor}(\mathit{close}(P_1, P_2) = \text{true},\ I_4), \\
&\quad \textbf{intersect_all}([I_3, I_4],\ I_5), \\
&\quad \textbf{holdsFor}(\mathit{inactive}(P_1) = \text{true},\ I_6), \\
&\quad \textbf{holdsFor}(\mathit{inactive}(P_2) = \text{true},\ I_7), \\
&\quad \textbf{union_all}([I_6, I_7],\ I_8), \\
&\quad \textbf{relative_complement_all}(I_5, [I_8], I).
\end{aligned}$$

Simplified Event Calculus (SimplEC)

- Writing Complex Event definitions in RTEC involves:
  - Event Calculus
  - Prolog
  - Thorough knowledge of RTEC's mechanisms
  - Collaboration between domain experts and programmers
- Alas, not everyone is keen on (logic) programming!
- SimplEC: Simplified Event Calculus
  - High level notation
  - A level of abstraction towards the technical details of RTEC
  - Addresses domain experts with little (or no) programming skills

SimplEC

SimplEC Statements:

initiate $passenger\_density(ID, VT) = Val$ **iff**
$passenger\_density\_change(ID, VT, Val)$.

Equivalent RTEC code:

initiatedAt$(passenger\_density(ID, VT) = Val,\ T)$:–
happensAt$(passenger\_density\_change(ID, VT, Val),\ T)$.
simpleFluent$(passenger\_density(\_, \_) = high)$.
simpleFluent$(passenger\_density(\_, \_) = medium)$.
simpleFluent$(passenger\_density(\_, \_) = low)$.
event$(passenger\_density\_change(\_, \_, \_))$.
outputEntity$(passenger\_density(\_, \_) = high)$.
outputEntity$(passenger\_density(\_, \_) = medium)$.
outputEntity$(passenger\_density(\_, \_) = low)$.
inputEntity$(passenger\_density\_change(\_, \_, \_))$.

SimplEC

SimplEC Statements:

initiate *passenger_density*(*ID*, *VT*) = *Val* **iff**
passenger_density_change(*ID*, *VT*, *Val*).

Equivalent RTEC code:

initiatedAt(*passenger_density*(*ID*, *VT*) = *Val*, *T*) :-
happensAt(*passenger_density_change*(*ID*, *VT*, *Val*), *T*).
simpleFluent(*passenger_density*(_, _) = *high*).
simpleFluent(*passenger_density*(_, _) = *medium*).
simpleFluent(*passenger_density*(_, _) = *low*).
event(*passenger_density_change*(_, _, _)).
outputEntity(*passenger_density*(_, _) = *high*).
outputEntity(*passenger_density*(_, _) = *medium*).
outputEntity(*passenger_density*(_, _) = *low*).
inputEntity(*passenger_density_change*(_, _, _)).

SimplEC

SimplEC Statements:

initiate *passenger_density*(*ID*, *VT*) = *Val* **iff**
passenger_density_change(*ID*, *VT*, *Val*).

Equivalent RTEC code:

initiatedAt(*passenger_density*(*ID*, *VT*) = *Val*, *T*) :-
happensAt(*passenger_density_change*(*ID*, *VT*, *Val*), *T*).
simpleFluent(*passenger_density*(_, _) = *high*).
simpleFluent(*passenger_density*(_, _) = *medium*).
simpleFluent(*passenger_density*(_, _) = *low*).
event(*passenger_density_change*(_, _, _)).
outputEntity(*passenger_density*(_, _) = *high*).
outputEntity(*passenger_density*(_, _) = *medium*).
outputEntity(*passenger_density*(_, _) = *low*).
inputEntity(*passenger_density_change*(_, _, _)).

SimplEC

SimplEC Statements:

initiate *person*(*P*) **iff**
(**start** *walking*(*P*) **or**
start *active*(*P*) **or**
start *running*(*P*) **or**
start *abrupt*(*P*)),
not *disappear*(*P*).

terminate *person*(*P*) **iff**
disappear(*P*).

Equivalent RTEC code:

initiatedAt(*person*(*P*) = true, *T*) :−
happensAt(**start**(*walking*(*P*) = true), *T*),
not **happensAt**(*disappear*(*P*), *T*).

initiatedAt(*person*(*P*) = true, *T*) :−
happensAt(**start**(*active*(*P*) = true), *T*),
not **happensAt**(*disappear*(*P*), *T*).

initiatedAt(*person*(*P*) = true, *T*) :−
happensAt(**start**(*running*(*P*) = true), *T*),
not **happensAt**(*disappear*(*P*), *T*).

initiatedAt(*person*(*P*) = true, *T*) :−
happensAt(**start**(*abrupt*(*P*) = true), *T*),
not **happensAt**(*disappear*(*P*), *T*).

terminatedAt(*person*(*P*) = true, *T*) :−
happensAt(*disappear*(*P*), *T*).

simpleFluent(*person*(_) = *true*).
sDFluent(*walking*(_) = *true*).
sDFluent(*active*(_) = *true*).
sDFluent(*running*(_) = *true*).
sDFluent(*abrupt*(_) = *true*).
event(*disappear*(_)).
outputEntity(*person*(_) = *true*).
inputEntity(*walking*(_) = *true*).
inputEntity(*active*(_) = *true*).
inputEntity(*running*(_) = *true*).
inputEntity(*abrupt*(_) = *true*).
inputEntity(*disappear*(_)).

SimplEC

SimplEC Statements:

$moving(P_1, P_2)$ **iff**
$walking(P_1)$,
$walking(P_2)$,
$close(P_1, P_2)$.

Equivalent RTEC code:

holdsFor$(moving(P_1, P_2) = \text{true}, I)$:−
holdsFor$(walking(P_1) = \text{true}, I_1)$,
holdsFor$(walking(P_2) = \text{true}, I_2)$,
holdsFor$(close(P_1, P_2) = \text{true}, I_3)$,
intersect_all$([I_1, I_2, I_3], I)$.
sDFluent$(moving(\_, \_) = \text{true})$.
sDFluent$(walking(\_) = \text{true})$.
sDFluent$(close(\_, \_) = \text{true})$.
outputEntity$(moving(\_, \_) = \text{true})$.
inputEntity$(walking(\_) = \text{true})$.
inputEntity$(close(\_, \_) = \text{true})$.

SimplEC

SimplEC Statements:

$moving(P_1, P_2)$ **iff**
$walking(P_1)$,
$walking(P_2)$,
$close(P_1, P_2)$.

Equivalent RTEC code:

holdsFor$(moving(P_1, P_2) = \text{true}, I)$:−
holdsFor$(walking(P_1) = \text{true}, I_1)$,
holdsFor$(walking(P_2) = \text{true}, I_2)$,
holdsFor$(close(P_1, P_2) = \text{true}, I_3)$,
intersect_all$([I_1, I_2, I_3], I)$.
sDFluent$(moving(\_, \_) = \text{true})$.
sDFluent$(walking(\_) = \text{true})$.
sDFluent$(close(\_, \_) = \text{true})$.
outputEntity$(moving(\_, \_) = \text{true})$.
inputEntity$(walking(\_) = \text{true})$.
inputEntity$(close(\_, \_) = \text{true})$.

SimplEC

SimplEC Statements:

$moving(P_1, P_2)$ **iff**
$walking(P_1)$,
$walking(P_2)$,
$close(P_1, P_2)$.

Equivalent RTEC code:

holdsFor$(moving(P_1, P_2) = \text{true}, I)$:−
holdsFor$(walking(P_1) = \text{true}, I_1)$,
holdsFor$(walking(P_2) = \text{true}, I_2)$,
holdsFor$(close(P_1, P_2) = \text{true}, I_3)$,
intersect_all$([I_1, I_2, I_3], I)$.
sDFluent$(moving(\_, \_) = \text{true})$.
sDFluent$(walking(\_) = \text{true})$.
sDFluent$(close(\_, \_) = \text{true})$.
outputEntity$(moving(\_, \_) = \text{true})$.
inputEntity$(walking(\_) = \text{true})$.
inputEntity$(close(\_, \_) = \text{true})$.

SimplEC

SimplEC Statements:

$moving(P_1, P_2)$ **iff**
$walking(P_1)$,
$walking(P_2)$,
$close(P_1, P_2)$.

Equivalent RTEC code:

holdsFor$(moving(P_1, P_2) = \text{true}, I)$:−
holdsFor$(walking(P_1) = \text{true}, I_1)$,
holdsFor$(walking(P_2) = \text{true}, I_2)$,
holdsFor$(close(P_1, P_2) = \text{true}, I_3)$,
intersect_all$([I_1, I_2, I_3], I)$.
sDFluent$(moving(\_, \_) = \text{true})$.
sDFluent$(walking(\_) = \text{true})$.
sDFluent$(close(\_, \_) = \text{true})$.
outputEntity$(moving(\_, \_) = \text{true})$.
inputEntity$(walking(\_) = \text{true})$.
inputEntity$(close(\_, \_) = \text{true})$.

SimplEC

SimplEC Statements:

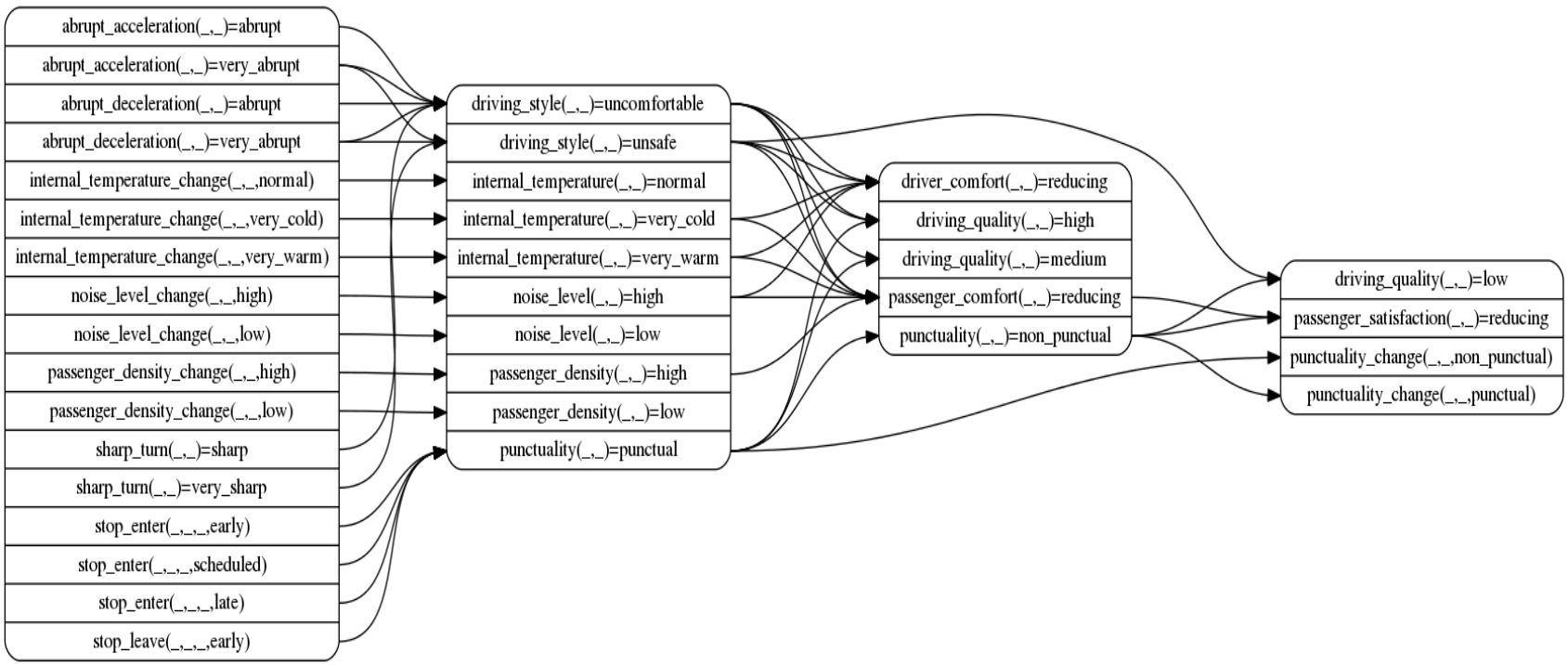
$fighting(P_1, P_2)$ **iff**
$(abrupt(P_1)$ **or** $abrupt(P_2))$,
$close(P_1, P_2)$,
not $(inactive(P_1)$ **or** $inactive(P_2))$.

Equivalent RTEC code:

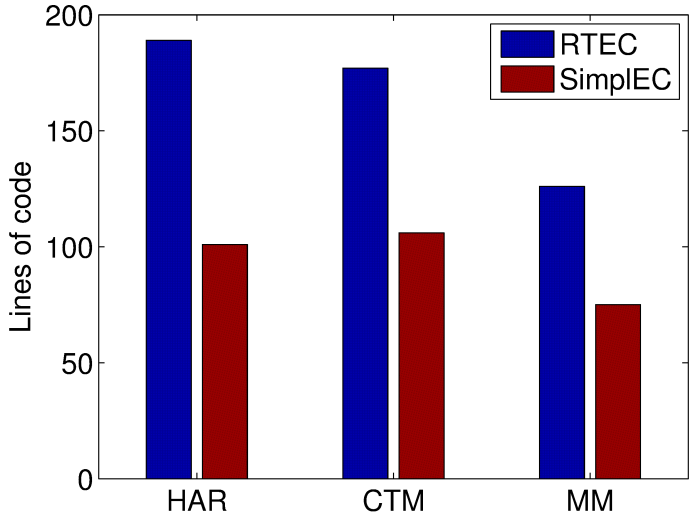
holdsFor$(fighting(P_1, P_2) = \text{true}, I)$:−
holdsFor$(abrupt(P_1) = \text{true}, I_1)$,
holdsFor$(abrupt(P_2) = \text{true}, I_2)$,
union_all$([I_1, I_2], I_3)$,
holdsFor$(close(P_1, P_2) = \text{true}, I_4)$,
intersect_all$([I_3, I_4], I_5)$,
holdsFor$(inactive(P_1) = \text{true}, I_6)$,
holdsFor$(inactive(P_2) = \text{true}, I_7)$,
union_all$([I_6, I_7], I_8)$,
relative_complement_all$(I_5, [I_8], I)$.
sDFluent$(fighting(\_, \_) = \text{true})$.
sDFluent$(abrupt(\_) = \text{true})$.
sDFluent$(close(\_, \_) = \text{true})$.
sDFluent$(inactive(\_) = \text{true})$.
outputEntity$(fighting(\_, \_) = \text{true})$.
inputEntity$(abrupt(\_) = \text{true})$.
inputEntity$(close(\_, \_) = \text{true})$.
inputEntity$(inactive(\_) = \text{true})$.

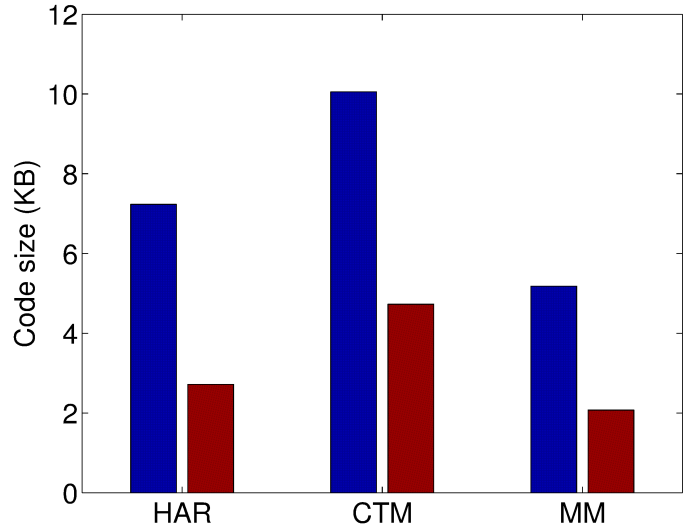
SimplEC: Dependency Graph Generation

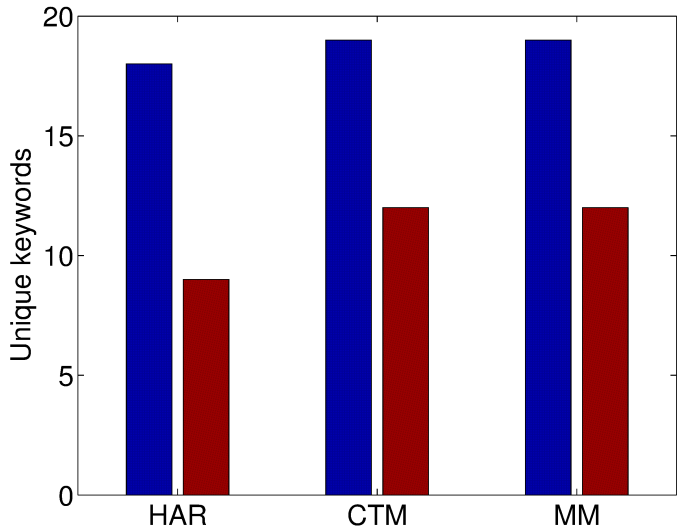
City Transport Monitoring:



Evaluation







HAR = Human Activity Recognition
CTM = City Transport Monitoring
MM = Maritime Monitoring

More...

- RTEC and SimplEC code available on GitHub
  - `https://github.com/aartikis/RTEC`
- Future work:
  - Have SimplEC tested by the domain experts of the datAcron (Big Data Analytics for Time Critical Mobility Forecasting) project – see `http://datacron-project.eu`
  - Construct simpler and more orderly dependency graphs

Thank you!

Any questions?