



CISPA
HELMHOLTZ CENTER FOR
INFORMATION SECURITY



Funded by
the European Union



erc
European Research Council
Horizon Europe

Two Ways to See the Future:

Combining Prediction and Future-Offset Accesses in RTLola

*Jan Baumeister, Bernd Finkbeiner,
Eduard Müller, **Frederik Scheerer**,
Julia Tillman*

RV 2026 | October 7, 2026





Future Directed Specifications

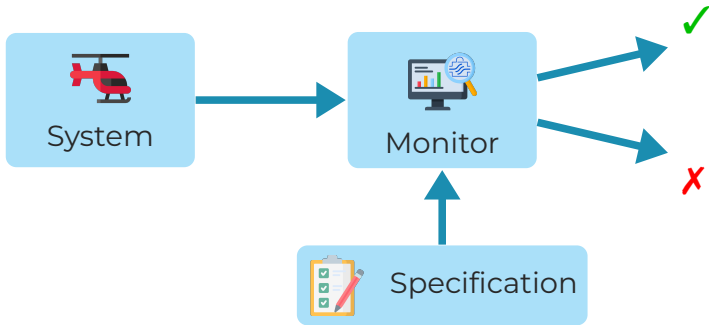
Many specifications are most naturally expressed by reasoning about the future, e.g.:

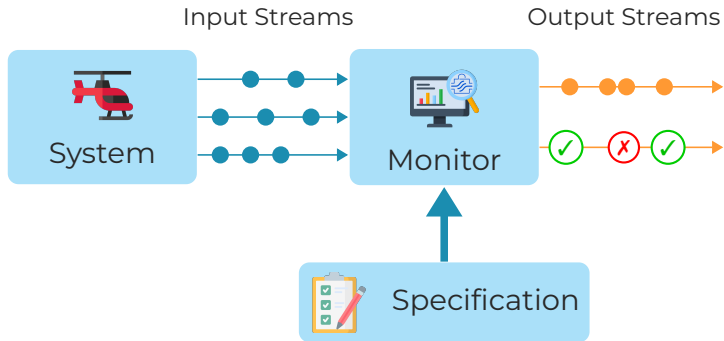
Each takeoff event must be eventually followed by the aircraft reaching a valid flight altitude before the next takeoff

Can be expressed in LTL as

$$\Box(\text{takeoff} \Rightarrow \bigcirc(\neg \text{takeoff} \mathcal{U} \text{alt_reached})).$$

Evaluating this requires **looking ahead**: the verdict at time of *takeoff* depends on values not yet observed.







Two Problems

Expressiveness



Future-directed specs are
often more natural to write

Past-rewriting is expensive
and shifts timestamps

Timeliness

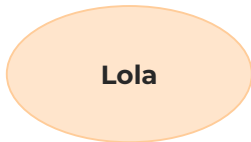


We want a verdict
as early as possible

But future values have
not been observed yet

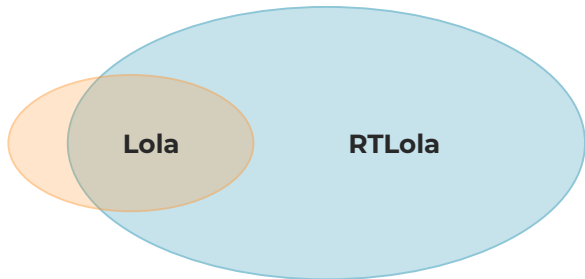


Lola and RTLola



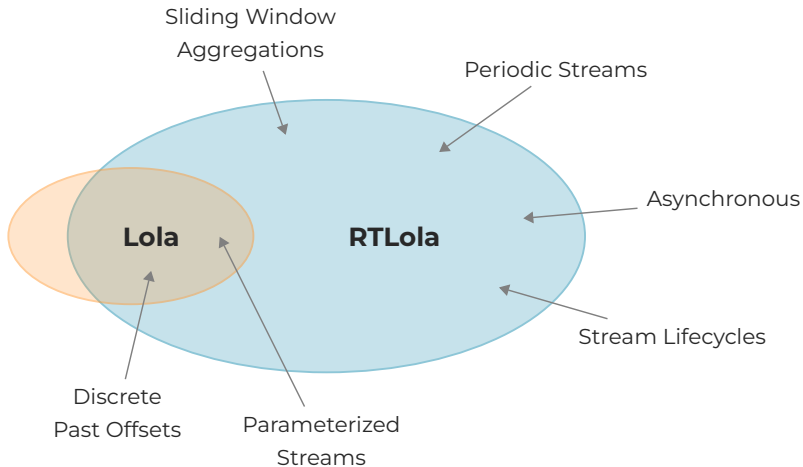


Lola and RTLola



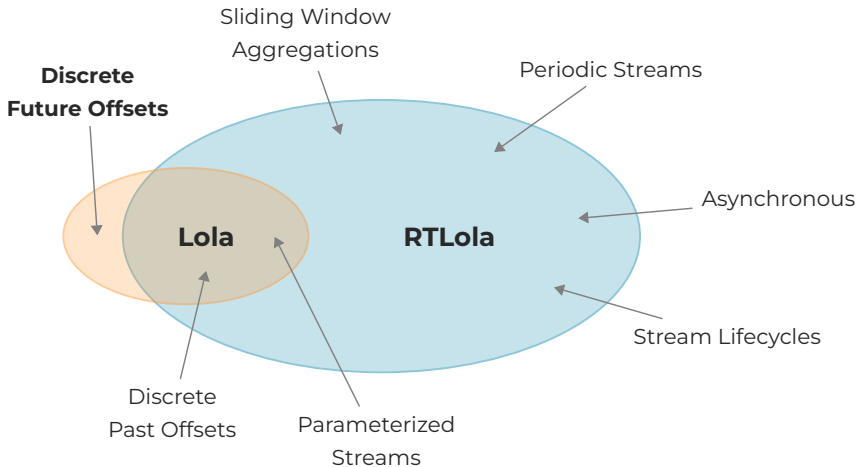


Lola and RTLola



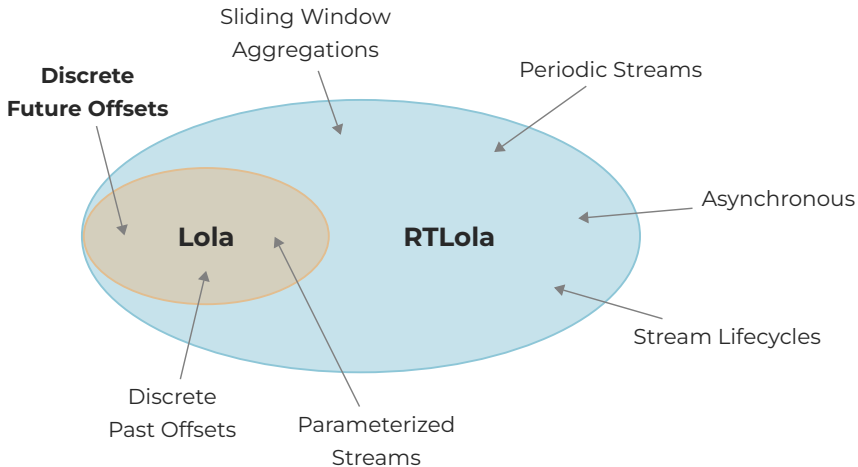


Lola and RTLola



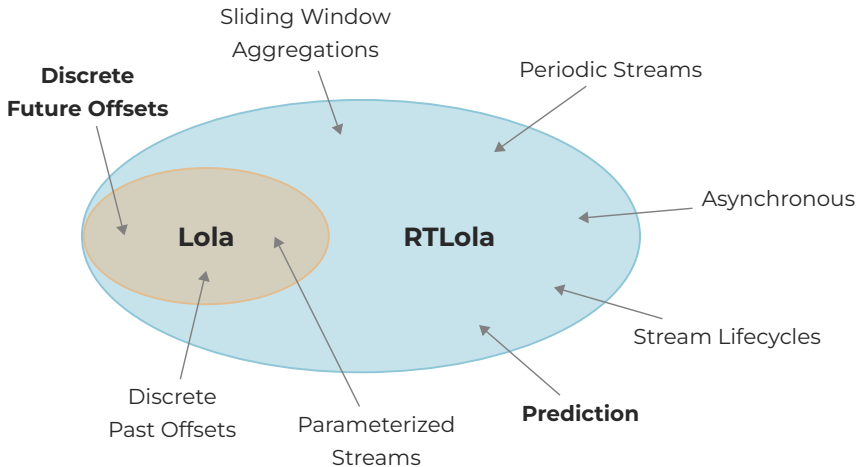


Lola and RTLola





Lola and RTLola





Two Ways to See the Future

Prediction



forecasts future values from
a bounded window of history

immediate,
but potentially imprecise



Two Ways to See the Future

Prediction



forecasts future values from
a bounded window of history

immediate,
but potentially imprecise

Future Accesses



accesses exact
future values directly

precise, but delayed until
the value is observed



Two Ways to See the Future

Prediction



forecasts future values from
a bounded window of history

immediate,
but potentially imprecise

Future Accesses



accesses exact
future values directly

precise, but delayed until
the value is observed

In combination: early estimates and precise confirmations.



Two Ways to See the Future

Prediction



forecasts future values from
a bounded window of history

immediate,
but potentially imprecise

Future Accesses



accesses exact
future values directly

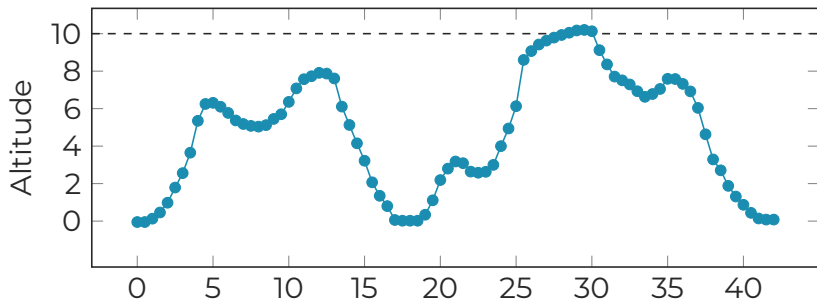
precise, but delayed until
the value is observed

In combination: early estimates and precise confirmations.



Future Accesses

$\Box(\text{takeoff} \Rightarrow \bigcirc(\neg \text{takeoff} \cup \text{alt_reached}))$

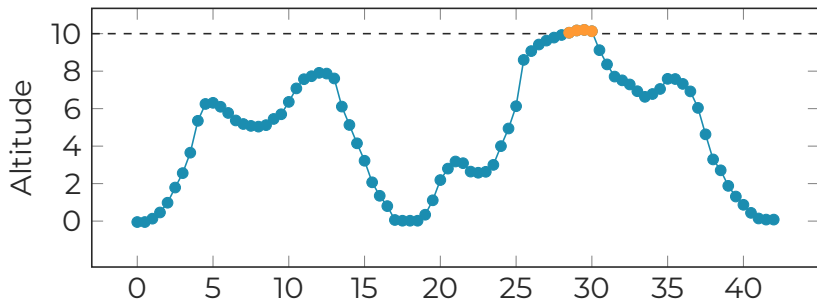


```
input alt : Float64
```



Future Accesses

$\Box(\text{takeoff} \Rightarrow \bigcirc(\neg \text{takeoff} \cup \text{alt_reached}))$



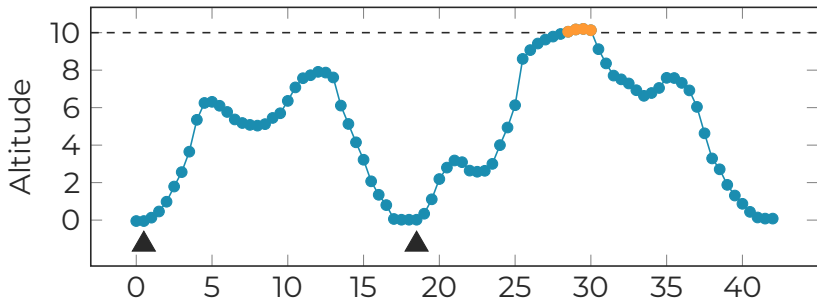
```
input alt : Float64
```

```
output alt_reached := alt > 10.0
```



Future Accesses

$\Box(\text{takeoff} \Rightarrow \bigcirc(\neg \text{takeoff} \cup \text{alt_reached}))$



```
input alt : Float64
```

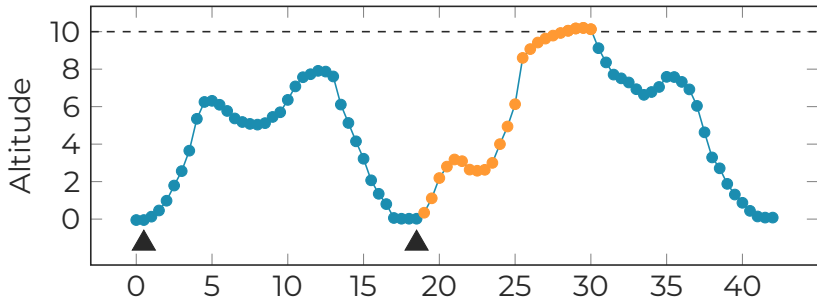
```
output alt_reached := alt > 10.0
```

```
output takeoff := alt ≤ 0.05 ∧ alt.offset(by: 1, or: 0.0) > 0.05
```



Future Accesses

$$\Box(\text{takeoff} \Rightarrow \bigcirc(\neg \text{takeoff} \cup \text{alt_reached}))$$



```
input alt : Float64
```

```
output alt_reached := alt > 10.0
```

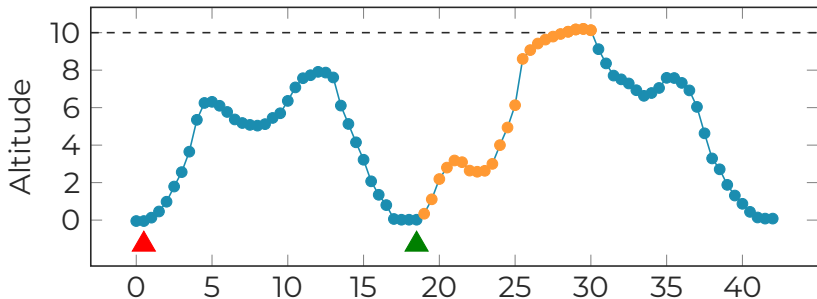
```
output takeoff := alt ≤ 0.05 ∧ alt.offset(by: 1, or: 0.0) > 0.05
```

```
output valid_flight := alt_reached ∨ (¬takeoff ∧ valid_flight.offset(by: 1, or: false))
```



Future Accesses

$$\Box(\text{takeoff} \Rightarrow \bigcirc(\neg \text{takeoff} \cup \text{alt_reached}))$$



```
input alt : Float64
output alt_reached := alt > 10.0
output takeoff := alt ≤ 0.05 ∧ alt.offset(by: 1, or: 0.0) > 0.05
output valid_flight := alt_reached ∨ (¬takeoff ∧ valid_flight.offset(by: 1, or: false))
trigger ¬(takeoff ⇒ valid_flight.offset(by: 1, or: false))
```



Static Analysis



A specification is ...

Well-Defined



if there exists **exactly**
one evaluation
model satisfying all
stream equations.



Static Analysis



A specification is ...

Well-Defined



if there exists **exactly one** evaluation
model satisfying all
stream equations.

Well-Formed



if it satisfies some
static criteria.



Static Analysis



A specification is ...

Well-Defined



if there exists **exactly one** evaluation model satisfying all stream equations.



Well-Formed



if it satisfies some static criteria.



Static Analysis



A specification is ...

Well-Defined



if there exists **exactly one** evaluation model satisfying all stream equations.



Well-Formed



if it satisfies some static criteria.

In Lola:

The specification must not contain a zero-cycle in the dependency graph.



Two Examples



```
input x : ()
```

```
input y : ()
```

```
output a @x^y := b.offset(by: 2, or: false)
```

```
output b @y := a.hold(or: false)
```



Two Examples

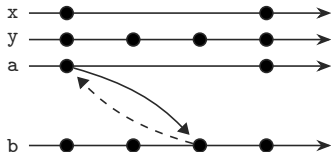


```
input x : ()
```

```
input y : ()
```

```
output a @x^y := b.offset(by: 2, or: false)
```

```
output b @y := a.hold(or: false)
```





Two Examples

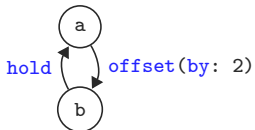
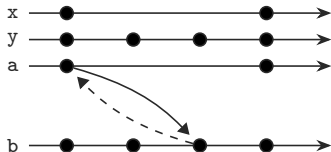


```
input x : ()
```

```
input y : ()
```

```
output a @x^y := b.offset(by: 2, or: false)
```

```
output b @y := a.hold(or: false)
```





Two Examples

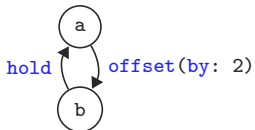
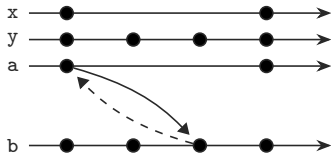


```
input x : ()
```

```
input y : ()
```

```
output a @x^y := b.offset(by: 2, or: false)
```

```
output b @y := a.hold(or: false)
```



Not well-defined,
but no zero cycle.

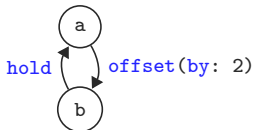
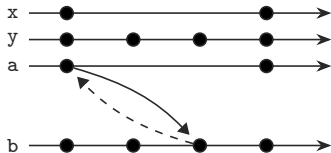


Two Examples



```
input x : ()  
input y : ()  
output a @x^y := b.offset(by: 2, or: false)  
output b @y := a.hold(or: false)
```

```
output a @3s := b.offset(by: 2, or: false)  
output b @3s := c.offset(by: -2, or: false)  
output c @1s := a.hold(or: false)
```

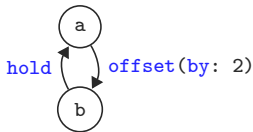
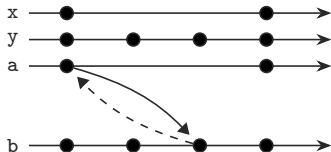




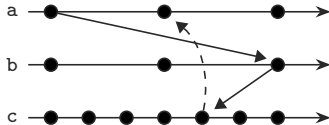
Two Examples



```
input x : ()  
input y : ()  
output a @x^y := b.offset(by: 2, or: false)  
output b @y := a.hold(or: false)
```



```
output a @3s := b.offset(by: 2, or: false)  
output b @3s := c.offset(by: -2, or: false)  
output c @1s := a.hold(or: false)
```

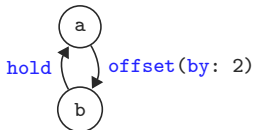
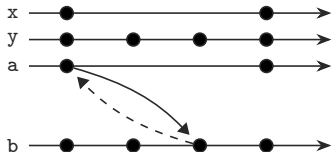




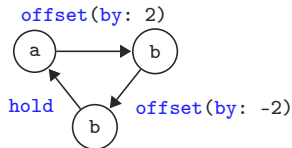
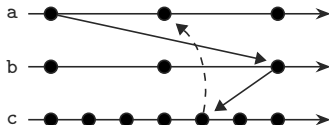
Two Examples



```
input x : ()  
input y : ()  
output a @x^y := b.offset(by: 2, or: false)  
output b @y := a.hold(or: false)
```



```
output a @3s := b.offset(by: 2, or: false)  
output b @3s := c.offset(by: -2, or: false)  
output c @1s := a.hold(or: false)
```

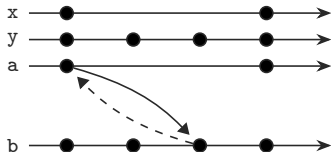




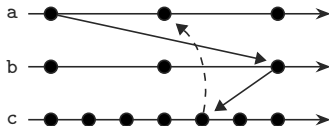
Two Examples



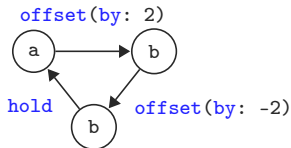
```
input x : ()  
input y : ()  
output a @x^y := b.offset(by: 2, or: false)  
output b @y := a.hold(or: false)
```



```
output a @3s := b.offset(by: 2, or: false)  
output b @3s := c.offset(by: -2, or: false)  
output c @1s := a.hold(or: false)
```



Well-defined,
but has zero cycle.





Well-Formed Specification



A specification is **well-formed** if the dependency graph does not contain a cycle with:



Well-Formed Specification



A specification is **well-formed** if the dependency graph does not contain a cycle with:

Zero-weight cycle

`offset(of: 1)`



`offset(of: -1)`

Contradictory value
(value depends on itself)

From Lola

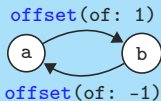


Well-Formed Specification



A specification is **well-formed** if the dependency graph does not contain a cycle with:

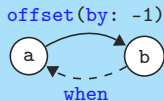
Zero-weight cycle



Contradictory value
(value depends on itself)

From Lola

Filter-edge



Contradictory existence

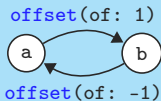
From RTLola



Well-Formed Specification

A specification is **well-formed** if the dependency graph does not contain a cycle with:

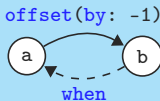
Zero-weight cycle



Contradictory value
(value depends on itself)

From Lola

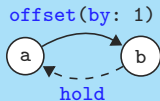
Filter-edge



Contradictory existence

From RTLola

Positive offset + hold

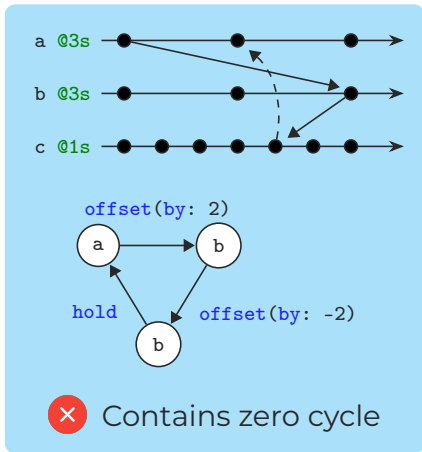


Contradictory value
(hidden zero-weight cycle)

New

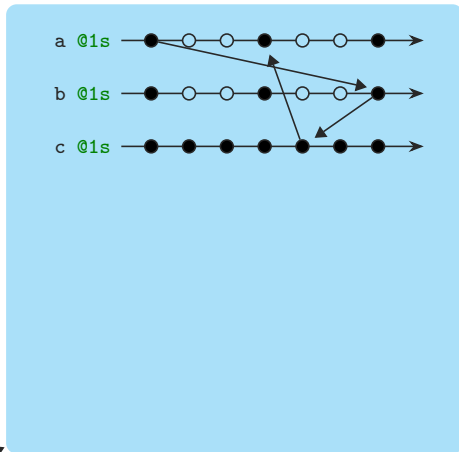
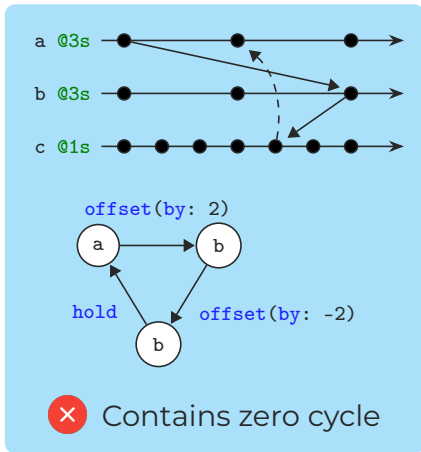


Periodic Dependency Cycles





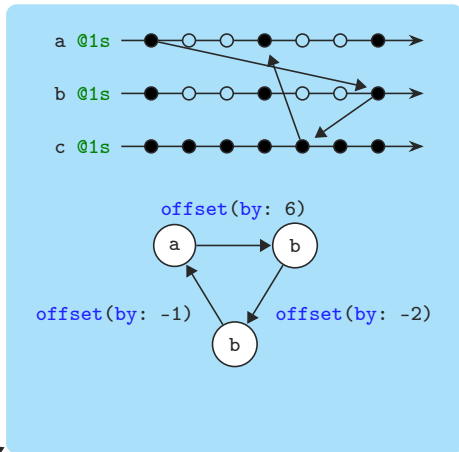
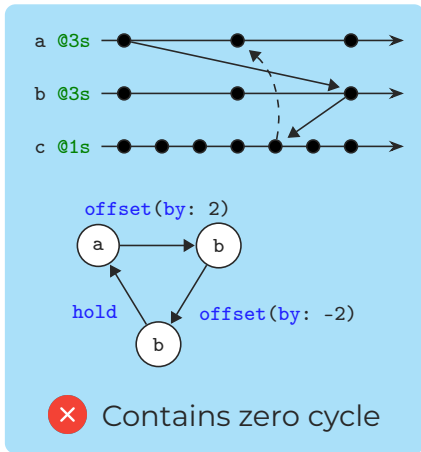
Periodic Dependency Cycles



Translate periodic pacing to common frequency



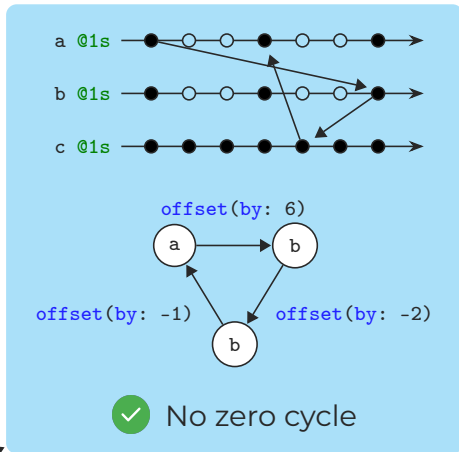
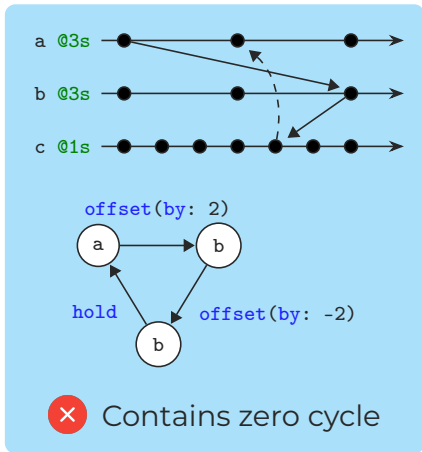
Periodic Dependency Cycles



Translate periodic pacing to common frequency



Periodic Dependency Cycles



Translate periodic pacing to common frequency



Monitoring Algorithm



```
input alt : Float64
output n_alt := alt.offset(by: 1, or: 0.0)
output slope := n_alt.offset(by: -1, or: 0.0) - n_alt.offset(by: 1, or: 0.0)
```



Monitoring Algorithm



```
input alt : Float64
output n_alt := alt.offset(by: 1, or: 0.0)
output slope := n_alt.offset(by: -1, or: 0.0) - n_alt.offset(by: 1, or: 0.0)
```



Resolved State

Stream value was computed



Unresolved State

Waiting for future dependencies

alt →

n_alt →

slope →



Monitoring Algorithm



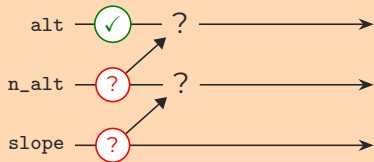
```
input alt : Float64
output n_alt := alt.offset(by: 1, or: 0.0)
output slope := n_alt.offset(by: -1, or: 0.0) - n_alt.offset(by: 1, or: 0.0)
```



Resolved State
Stream value was computed



Unresolved State
Waiting for future dependencies





Monitoring Algorithm



```
input alt : Float64
output n_alt := alt.offset(by: 1, or: 0.0)
output slope := n_alt.offset(by: -1, or: 0.0) - n_alt.offset(by: 1, or: 0.0)
```



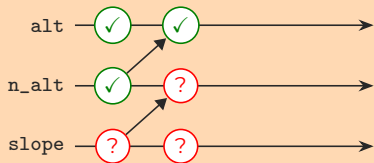
Resolved State

Stream value was computed



Unresolved State

Waiting for future dependencies





Monitoring Algorithm



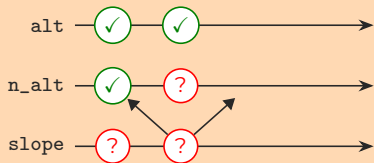
```
input alt : Float64
output n_alt := alt.offset(by: 1, or: 0.0)
output slope := n_alt.offset(by: -1, or: 0.0) - n_alt.offset(by: 1, or: 0.0)
```



Resolved State
Stream value was computed



Unresolved State
Waiting for future dependencies





Monitoring Algorithm



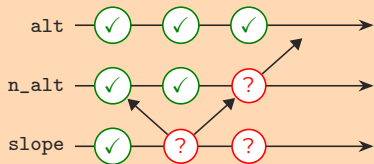
```
input alt : Float64
output n_alt := alt.offset(by: 1, or: 0.0)
output slope := n_alt.offset(by: -1, or: 0.0) - n_alt.offset(by: 1, or: 0.0)
```



Resolved State
Stream value was computed



Unresolved State
Waiting for future dependencies





Monitoring Algorithm



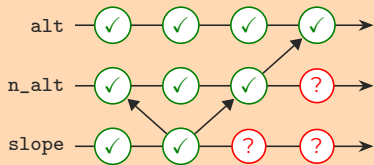
```
input alt : Float64
output n_alt := alt.offset(by: 1, or: 0.0)
output slope := n_alt.offset(by: -1, or: 0.0) - n_alt.offset(by: 1, or: 0.0)
```



Resolved State
Stream value was computed



Unresolved State
Waiting for future dependencies





Monitoring Algorithm 2



Each stream value goes through three stages:

1.



Maybe State

May or may not be computed

2.



Unresolved State

Waiting for future dependencies

3.



Resolved State

Stream value was computed



Monitoring Algorithm 2



Each stream value goes through three stages:

1.  **Maybe State**
May or may not be computed
 2.  **Unresolved State**
Waiting for future dependencies
 3.  **Resolved State**
Stream value was computed
- When pacing depends on unresolved/future values.



Monitoring Algorithm 2

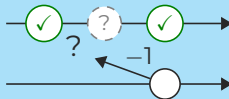


Each stream value goes through three stages:

1.  **Maybe State**
May or may not be computed
2.  **Unresolved State**
Waiting for future dependencies
3.  **Resolved State**
Stream value was computed

When pacing depends on unresolved/future values.

Must be resolved before offsets can be computed:

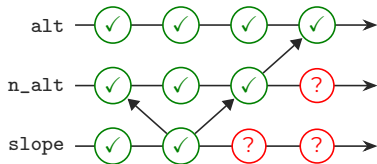




Delay



How much evaluation of a stream must be delayed until all (future) dependencies are resolved.

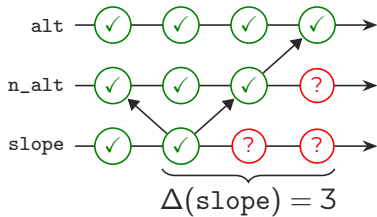




Delay



How much evaluation of a stream must be delayed until all (future) dependencies are resolved.

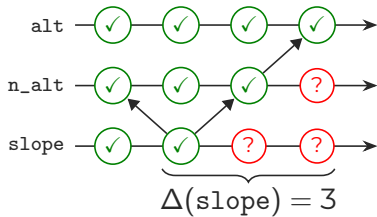




Delay



How much evaluation of a stream must be delayed until all (future) dependencies are resolved.



Required for:

Memory Bound

How many values must be retained in memory

Evaluation Order

Which streams must be evaluated before others



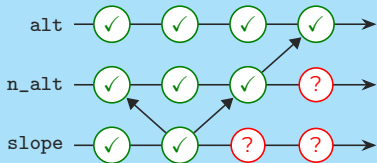
Memory Bound



Bounded



Statically bounded how many values must be retained in memory.





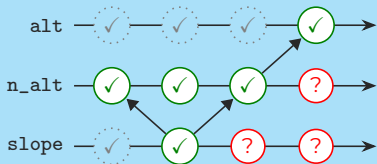
Memory Bound



Bounded



Statically bounded how many values must be retained in memory.





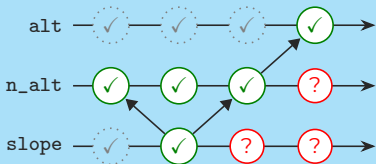
Memory Bound



Bounded



Statically bounded how many values must be retained in memory.

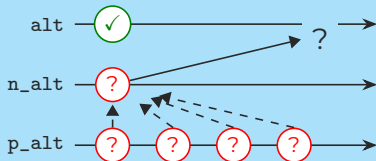


Unbounded



Memory can grow beyond any bound.

(In contrast to Lola also possible without any cycles)





Two Ways to See the Future

Prediction



forecasts future values from
a bounded window of history

immediate,
but potentially imprecise

Future Accesses



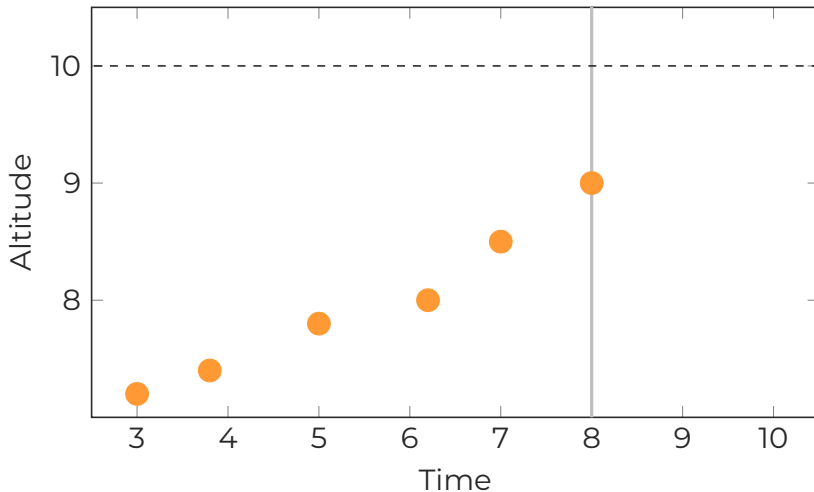
accesses exact
future values directly

precise, but delayed until
the value is observed

In combination: early estimates and precise confirmations.

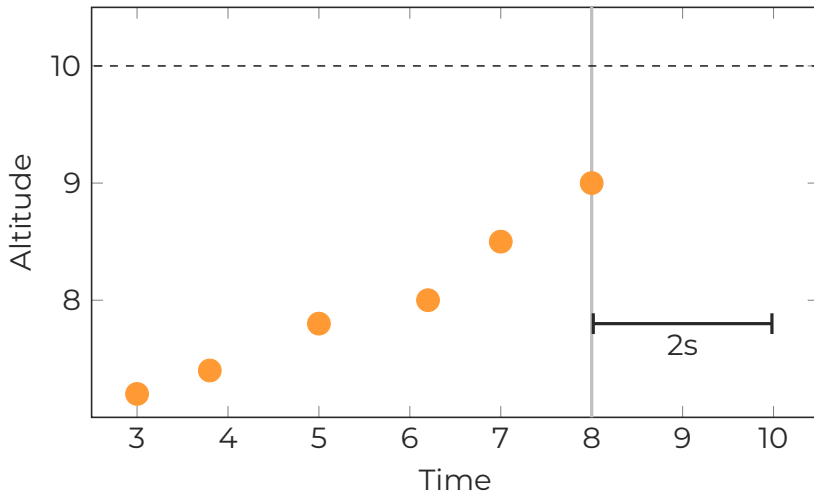


Example: Prediction



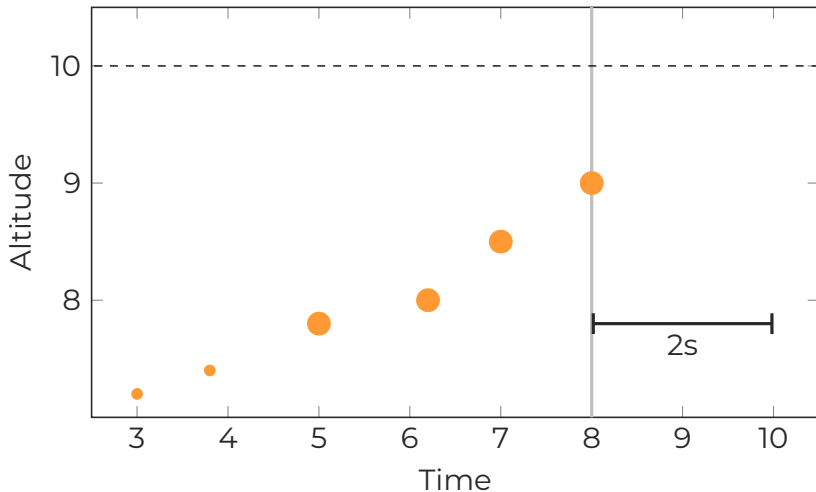


Example: Prediction



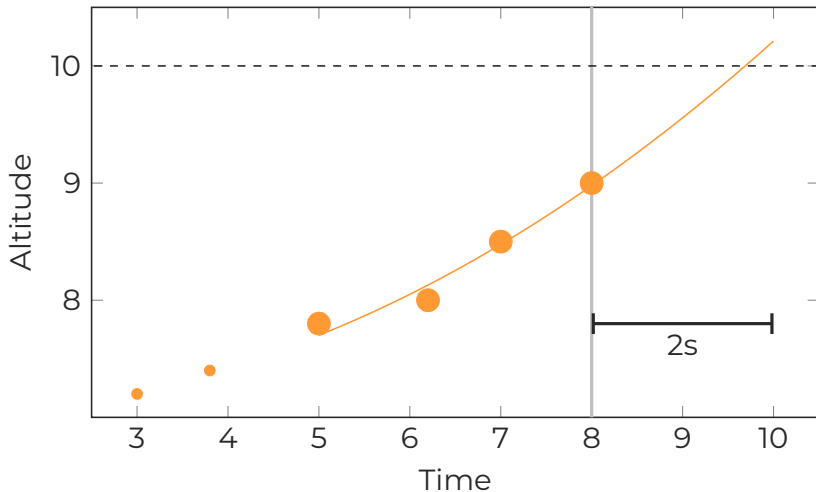


Example: Prediction



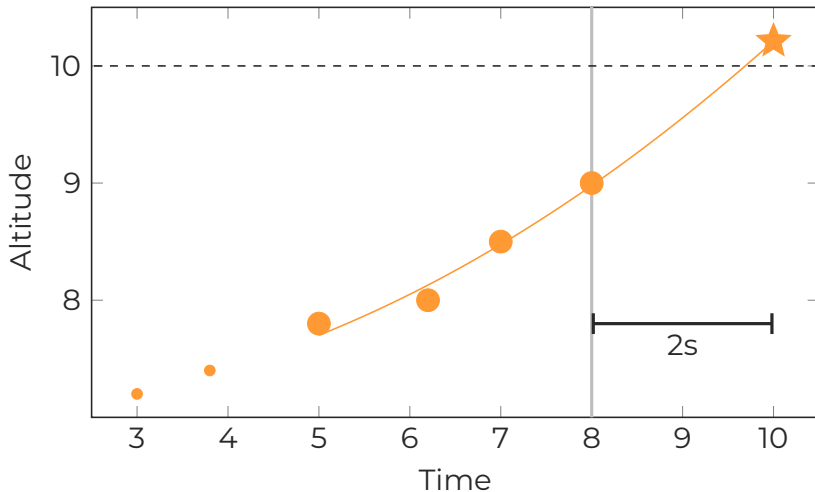


Example: Prediction



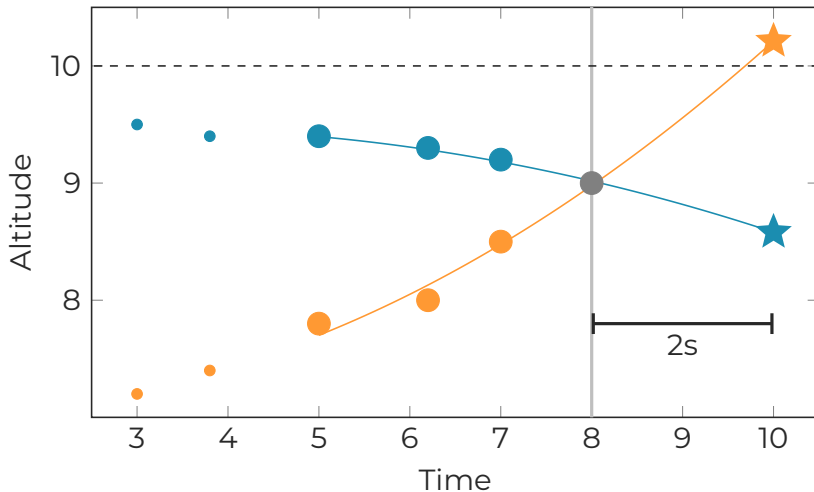


Example: Prediction



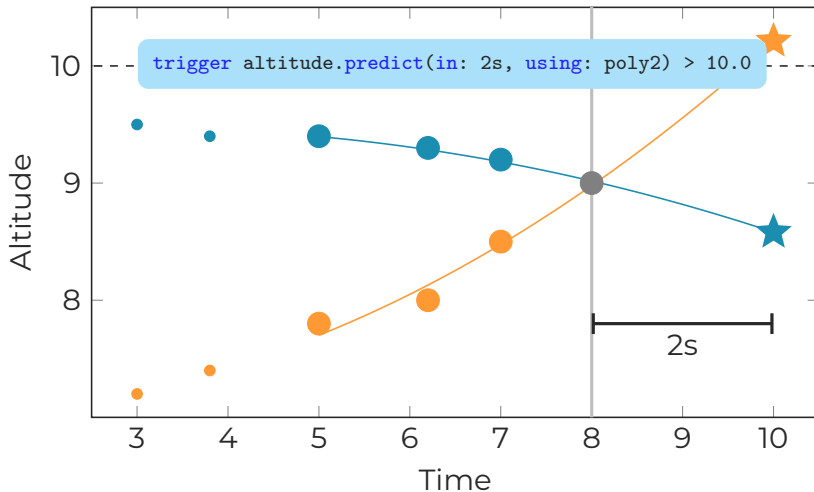


Example: Prediction





Example: Prediction





Syntactic Sugar for Prediction



```
s.predict(using:  $\gamma$ , over_discrete:  $n$ , in:  $\delta$ )
```



Syntactic Sugar for Prediction



```
s.predict(using:  $\gamma$ , over_discrete:  $n$ , in:  $\delta$ )
```

- Stream access $\rightarrow$ influences pacing
- Requires buffering the last n values in memory
- Entangled with RTLola's stream semantics



Syntactic Sugar for Prediction



```
s.predict(using:  $\gamma$ , over_discrete:  $n$ , in:  $\delta$ )
```

- Stream access $\rightarrow$ influences pacing
- Requires buffering the last n values in memory
- Entangled with RTLola's stream semantics

```
predict(( $v_1, t_1$ ), ..., ( $v_n, t_n$ ), using:  $\gamma$ , at:  $t_{now} + \delta$ )
```



Syntactic Sugar for Prediction



```
s.predict(using:  $\gamma$ , over_discrete:  $n$ , in:  $\delta$ )
```

- Stream access $\rightarrow$ influences pacing
- Requires buffering the last n values in memory
- Entangled with RTLola's stream semantics

```
predict(( $v_1, t_1$ ), ..., ( $v_n, t_n$ ), using:  $\gamma$ , at:  $t_{now} + \delta$ )
```

- Pure function: no stream access, no pacing
- Decoupled from stream semantics
- Existing static analyses stay unchanged



Syntactic Sugar for Prediction



```
s.predict(using:  $\gamma$ , over_discrete:  $n$ , in:  $\delta$ )
```

Desugars
to

```
predict(( $v_1, t_1$ ), ..., ( $v_n, t_n$ ), using:  $\gamma$ , at:  $t_{now} + \delta$ )
```

- Stream access $\rightarrow$ influences pacing
- Requires buffering the last n values in memory
- Entangled with RTLola's stream semantics

- Pure function: no stream access, no pacing
- Decoupled from stream semantics
- Existing static analyses stay unchanged



Syntactic Sugar for Prediction



```
... s.predict(over_discrete:  $n$ , using:  $\gamma$ , in:  $\delta$ ) ...
```



Syntactic Sugar for Prediction



```
... s.predict(over_discrete:  $n$ , using:  $\gamma$ , in:  $\delta$ ) ...
```

Attach timestamp to value:

```
output  $s_{ts}$  @s := (s, time)
```



Syntactic Sugar for Prediction



```
... s.predict(over_discrete:  $n$ , using:  $\gamma$ , in:  $\delta$ ) ...
```

Attach timestamp to value:

```
output  $s_{ts}$  @s := (s, time)
```

Access last n values:

```
output  $s_{values}$  @s := ( $s_{ts}$ ,  $s_{ts}.offset(by: -1)$ , ... ,  $s_{ts}.offset(by: -n + 1)$ )
```



Syntactic Sugar for Prediction



```
... s.predict(over_discrete:  $n$ , using:  $\gamma$ , in:  $\delta$ ) ...
```

Attach timestamp to value:

```
output  $s_{ts}$  @s := (s, time)
```

Access last n values:

```
output  $s_{values}$  @s := ( $s_{ts}$ ,  $s_{ts}.\text{offset}(\text{by: } -1)$ , ... ,  $s_{ts}.\text{offset}(\text{by: } -n + 1)$ )
```

Use hold access to decouple pacing:

```
... predict( $s_{values}.\text{hold}()$ , using:  $\gamma$ , time +  $\delta$ ) ...
```



Syntactic Sugar for Prediction



```
... s.predict(over_discrete: n, using:  $\gamma$ , in:  $\delta$ ) ...
```



Attach timestamp to value:

```
output  $S_{ts}$  @s := (s, time)
```

Access last n values:

```
output  $S_{values}$  @s := ( $S_{ts}$ ,  $S_{ts}.offset(by: -1)$ , ... ,  $S_{ts}.offset(by: -n + 1)$ )
```

Use hold access to decouple pacing:

```
... predict( $S_{values}.hold()$ , using:  $\gamma$ , time +  $\delta$ ) ...
```



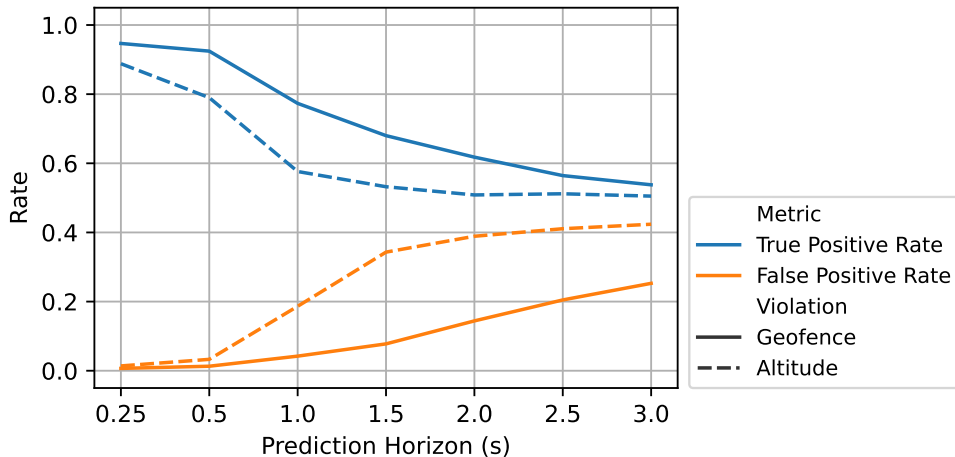
Evaluation Setup

- Data from **Microsoft AirSim** flight simulator
- Previously used to evaluate **Active Monitoring with RTLola** (RV 2025)
- Specifications monitor:
 - Geofence violations
 - Altitude bound violations



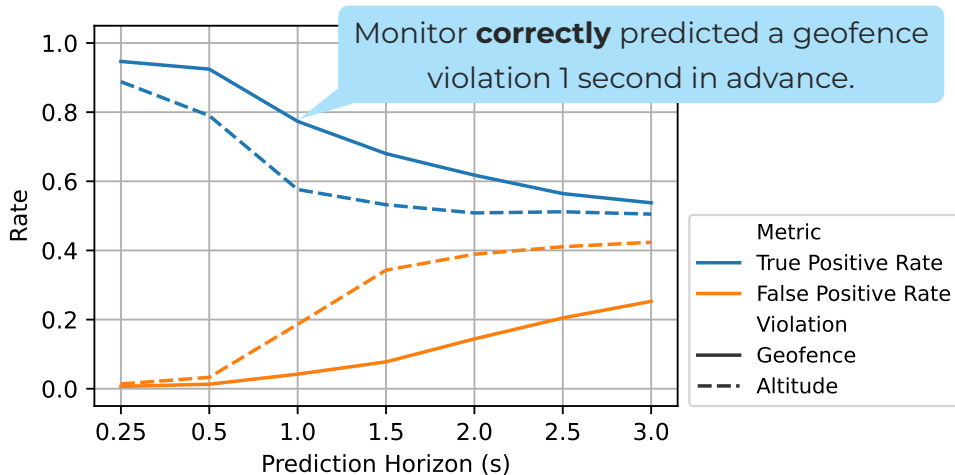


Prediction Evaluation



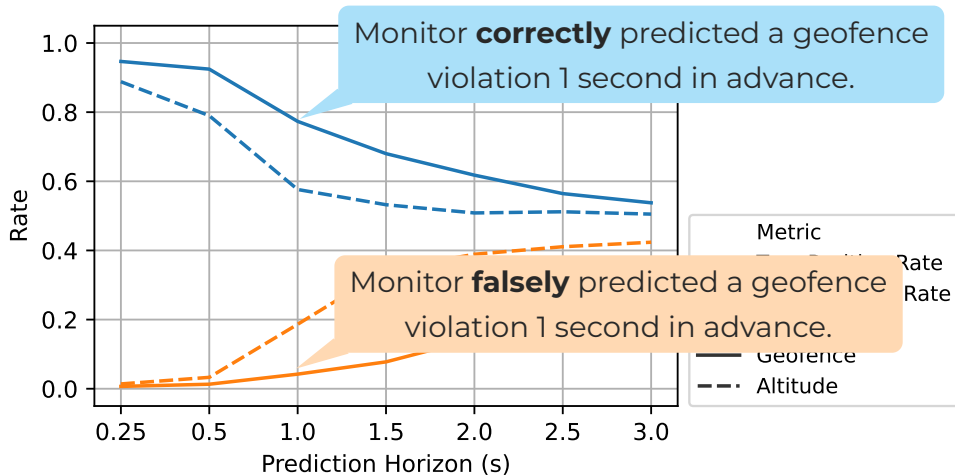


Prediction Evaluation





Prediction Evaluation





Two Ways to See the Future

Prediction



forecasts future values from
a bounded window of history

immediate,
but potentially imprecise

Future Accesses



accesses exact
future values directly

precise, but delayed until
the value is observed

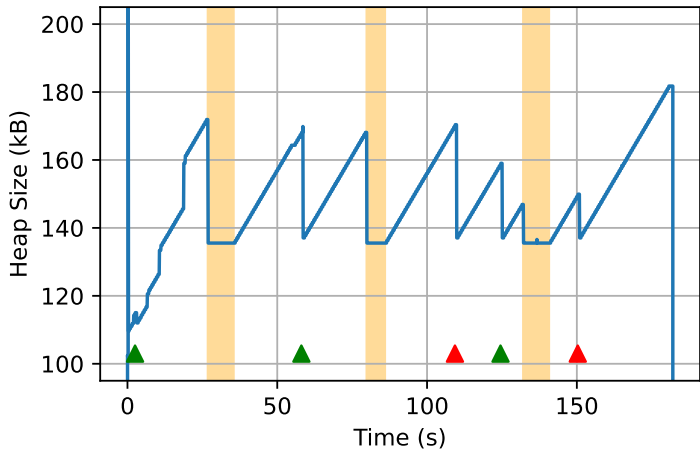
In combination: early estimates and precise confirmations.



Memory Evaluation



$\Box(\text{takeoff} \Rightarrow \bigcirc(\neg \text{takeoff} \cup \text{alt_reached}))$





Runtime Comparison



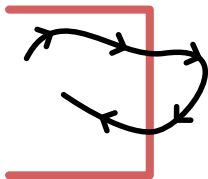
```
output @1Hz outside_fence : Bool := ...
```



Runtime Comparison



```
output @1Hz outside_fence : Bool := ...
```



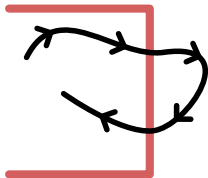


Runtime Comparison



```
output @1Hz outside_fence : Bool := ...
```

```
trigger  $\neg$ (outside_fence  $\Rightarrow$   $\neg$ outside_fence.offset(by: 2s, or: false))
```





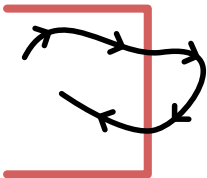
Runtime Comparison



```
output @1Hz outside_fence : Bool := ...
```

```
trigger  $\neg(\text{outside\_fence} \Rightarrow \neg\text{outside\_fence.offset}(\text{by: } 2\text{s}, \text{or: false}))$ 
```

```
trigger  $\neg(\text{outside\_fence.offset}(\text{by: } -2\text{s}, \text{or: false}) \Rightarrow \neg\text{outside\_fence})$ 
```





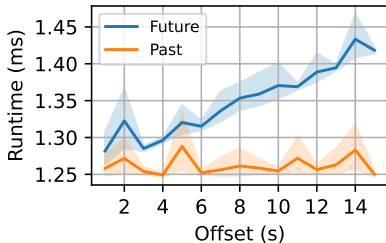
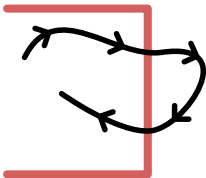
Runtime Comparison



```
output @1Hz outside_fence : Bool := ...
```

```
trigger  $\neg(\text{outside\_fence} \Rightarrow \neg\text{outside\_fence.offset}(\text{by: } 2\text{s}, \text{or: false}))$ 
```

```
trigger  $\neg(\text{outside\_fence.offset}(\text{by: } -2\text{s}, \text{or: false}) \Rightarrow \neg\text{outside\_fence})$ 
```





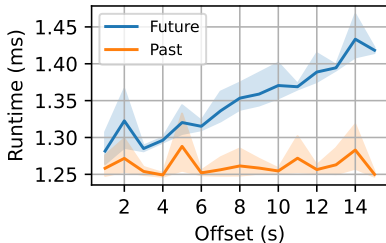
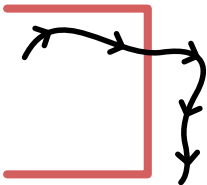
Runtime Comparison



```
output @1Hz outside_fence : Bool := ...
```

```
trigger  $\neg(\text{outside\_fence} \Rightarrow \neg\text{outside\_fence.offset}(\text{by: } 2\text{s}, \text{or: false}))$ 
```

```
trigger  $\neg(\text{outside\_fence.offset}(\text{by: } -2\text{s}, \text{or: false}) \Rightarrow \neg\text{outside\_fence})$ 
```





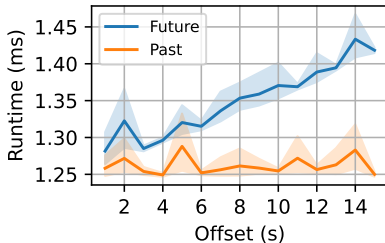
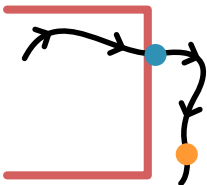
Runtime Comparison



```
output @1Hz outside_fence : Bool := ...
```

```
trigger  $\neg(\text{outside\_fence} \Rightarrow \neg\text{outside\_fence.offset}(\text{by: } 2\text{s}, \text{or: false}))$ 
```

```
trigger  $\neg(\text{outside\_fence.offset}(\text{by: } -2\text{s}, \text{or: false}) \Rightarrow \neg\text{outside\_fence})$ 
```





Summary



Future-Directed Specifications

$\Box(\text{takeoff} \Rightarrow \bigcirc(\neg \text{takeoff} \mathcal{U} \text{alt_reached}))$

Prediction



Immediate, lightweight estimates

Future Accesses



Exact values, delayed

Well-Formedness

Guarantees soundness of
future-offset evaluation

New criterion for RTLola

Algorithm & Memory

Maybe/unresolved/resolved states

Minor computational overhead

Memory often bounded in practice

The icons used in this presentation are designed by Magnific:
www.magnific.com.