Seminar Software Engineering

T9: Monitoring Spatially-Distributed Systems with Spatio-Temporal Logics

Albin Aliu, 3. Juni 2022


FOR CHRISTMAS I WANT A DRAGON!
BE REALISTIC
Okay I want a consistent notation for ~~probability and statistics~~ logics
WHAT COLOR DO YOU WANT YOUR DRAGON
RED

Outline

i. Runtime Verification in a nutshell

ii. Classification

    a. Signal Temporal Logic (STL)

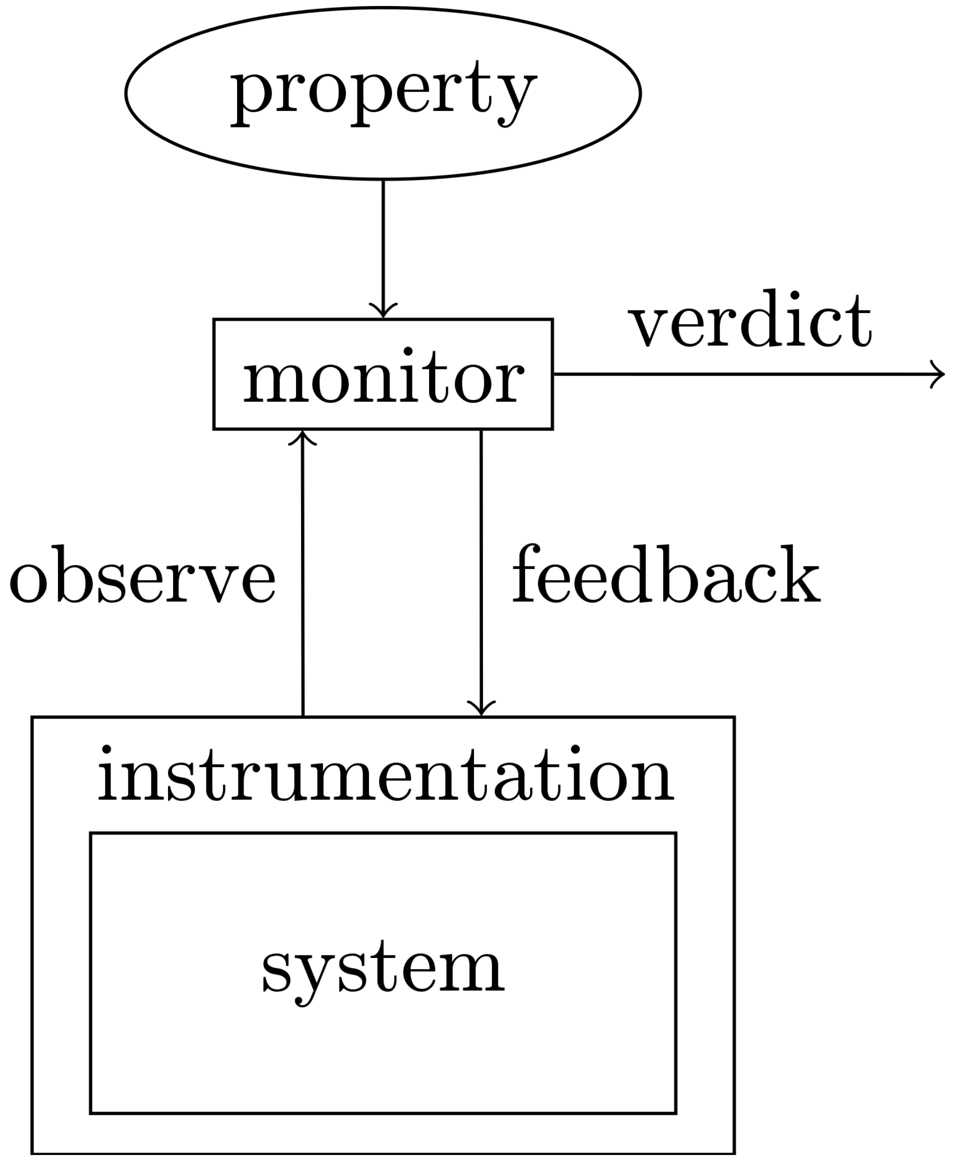
    b. Spatio-Signal Temporal Logic (SSTL)

    c. Spatio-Temporal Reach and Escape Logic (STREL)

iii. Hands-On Lab: RTLola Specification Language

Runtime Verification in a nutshell[1]

- instead of *proving* that our system is correct, we're going to *monitor* it and check whether it *violates* our *specifications*
- from the *specification* we synthesize *monitors*, which *observe* data, that is *extracted* from the system by means of *instrumentation*
- *monitors* can either be *online* or *offline*, meaning they can analyze and monitor data *while* the system is running or they analyze the data *after* the system's execution
- **Advantages:** very precise information on the runtime behaviour of the monitored system, lightweight
  Disadvantages: limited execution coverage



src: https://en.wikipedia.org/wiki/File:Runtime_Verification_Monitor.svg, 07.05.2022

[1] Bartocci, E., Falcone, Y., Francalanza, A., Reger, G. (2018). Introduction to Runtime Verification. In: Bartocci, E., Falcone, Y. (eds) Lectures on Runtime Verification. Lecture Notes in Computer Science, vol 10457. Springer, Cham. https://doi.org/10.1007/978-3-319-75632-5_1

Classification: Preliminaries[1]

- Temporal Logic emerged from the need to specify propositions that depend on some *timing assumptions*, hence the name
- Linear Temporal Logic introduces
  - the *next* operator $\circ\varphi$, meaning φ is true at the next point of the trace (other notation: $\mathbf{X}\varphi$)
  - the *until* operator $\varphi_1 \mathcal{U} \varphi_2$, meaning φ_1 is true from the current point of the trace until φ_2 is true.
- From these two operators, one can derive two more commonly used operators*
  - the *always* operator defined as $\square\varphi \equiv \varphi \mathcal{U}$ false (other notation: $\mathbf{G}\varphi$ for globally)
  - the *eventually* operator defined as $\diamond\varphi \equiv \neg\square\neg\varphi$ (other notation: $\mathbf{F}\varphi$ for finally)

* Can you make the link to *safety* and *liveness* properties?

Signal Temporal Logic (STL)[1]

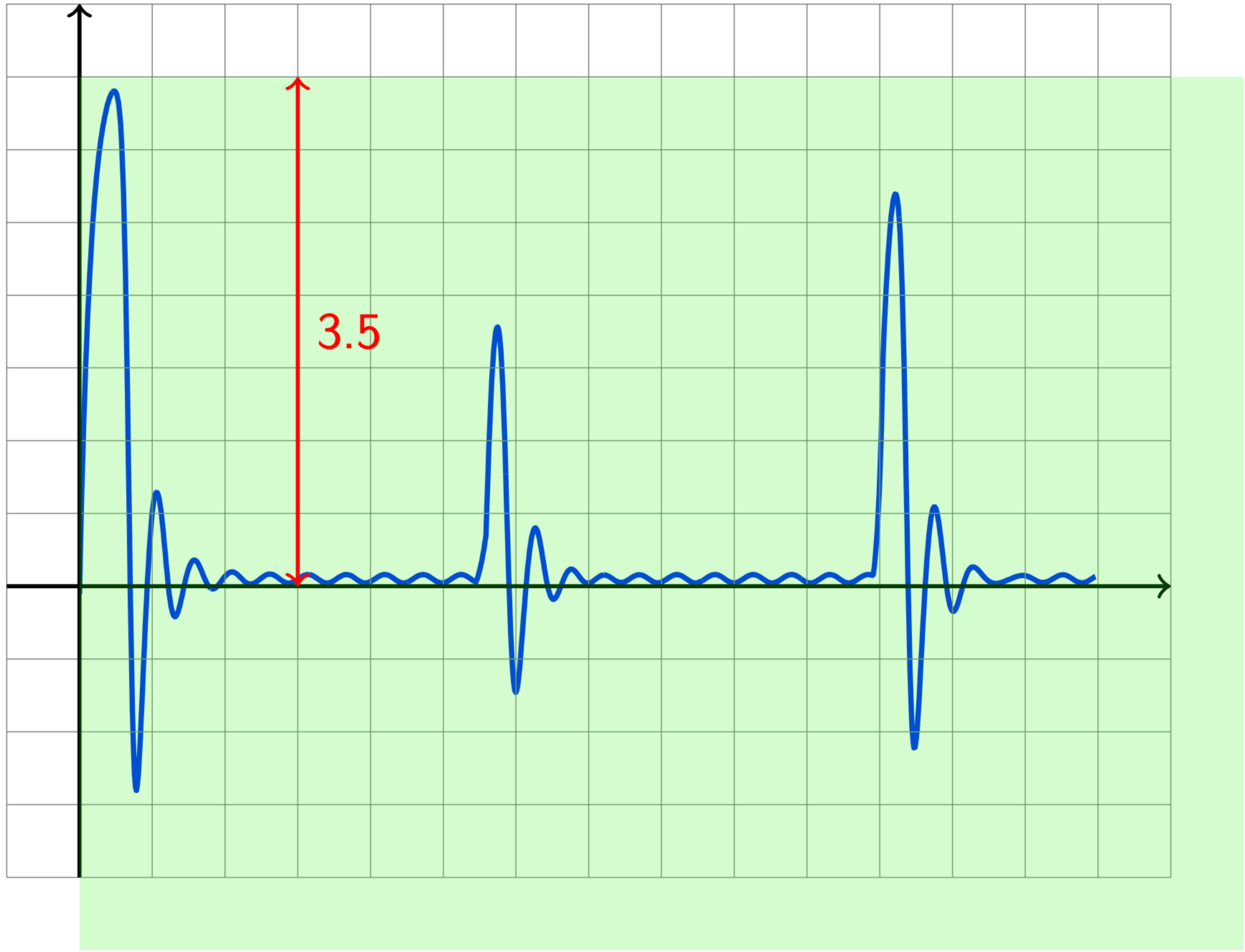
Introduction

- Usually, the *data* you pass to the monitor (figure slide 4) is an *execution trace* of a system, thus it's a *discrete sequence of events*
- Signal Temporal Logic introduces *signals*, where "a *signal* is a function from a set of real time points to a value domain" [1], p. 9
- To work with signals, we add a new predicate
  - $\mu = f(x_1[t], \ldots, x_m[t]) > 0$
    - for some function $f : \mathbb{R}^m \to \mathbb{R}$
    - and $x_i : \mathbb{R}_{\geq 0} \to \mathbb{R}$, $1 \leq i \leq m$ is a signal and $x_i[t]$ is the value of the signal x_i at time t.

Signal Temporal Logic (STL)[1]

The signal is never above 3.5

$$\varphi := \mathsf{G}\ (x[t] < 3.5)$$

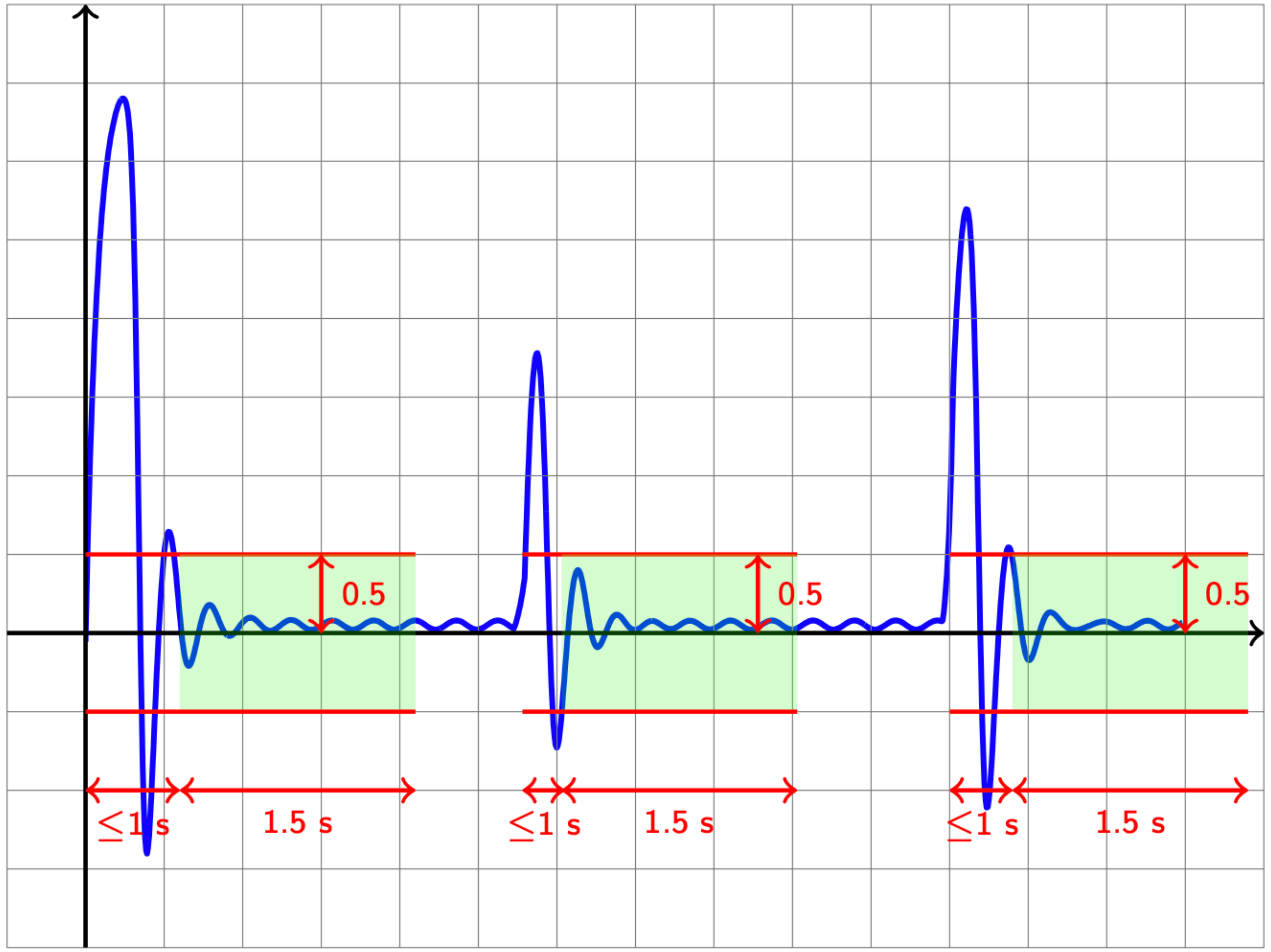


Lecture Slides: On Signal Temporal Logic by Alexandre Donzé
University of California, Berkeley
February 3, 2014

Signal Temporal Logic (STL)[1]

Always $|x| > 0.5 \Rightarrow$ *after 1 s,* $|x|$ *settles under 0.5 for 1.5 s*

$$\varphi := \mathsf{G}(x[t] > .5 \rightarrow \mathsf{F}_{[0,.6]} \; (\mathsf{G}_{[0,1.5]} \; x[t] < 0.5))$$



Lecture Slides: On Signal Temporal Logic by Alexandre Donzé
University of California, Berkeley
February 3, 2014

Spatio-Signal Temporal Logic (SSTL)

Introduction

- Extends STL with notions of *somewhere* and *surround* to express *spatial properties*
  - interpreted over a *discrete model* of the space, represented as a *finite undirected graph*
  - each node represents a *location in the space*, characterized by a set of signals that can be observed in time
  - each edge is weighted and represents the *distance between two nodes*

Spatio-Signal Temporal Logic (SSTL)

Syntax

$$\phi := \text{true} \,|\, \mu \,|\, \neg\psi \,|\, \psi_1 \wedge \psi_2 \,|\, \psi_1 \,\mathcal{U}_J\, \psi_2 \,|\, \circledcirc_{[w_1,w_2]}\, \psi \,|\, \psi_1 \,\mathcal{S}_{[w_1,w_2]}\, \psi_2$$

- Where the STL operators are the atomic proposition μ,
  the standard boolean connectives $\wedge$ (as conjunction) and $\neg$ (as negation)
  the *bounded until* operator $\mathcal{U}_J$, for $J \subset \mathbb{R}$

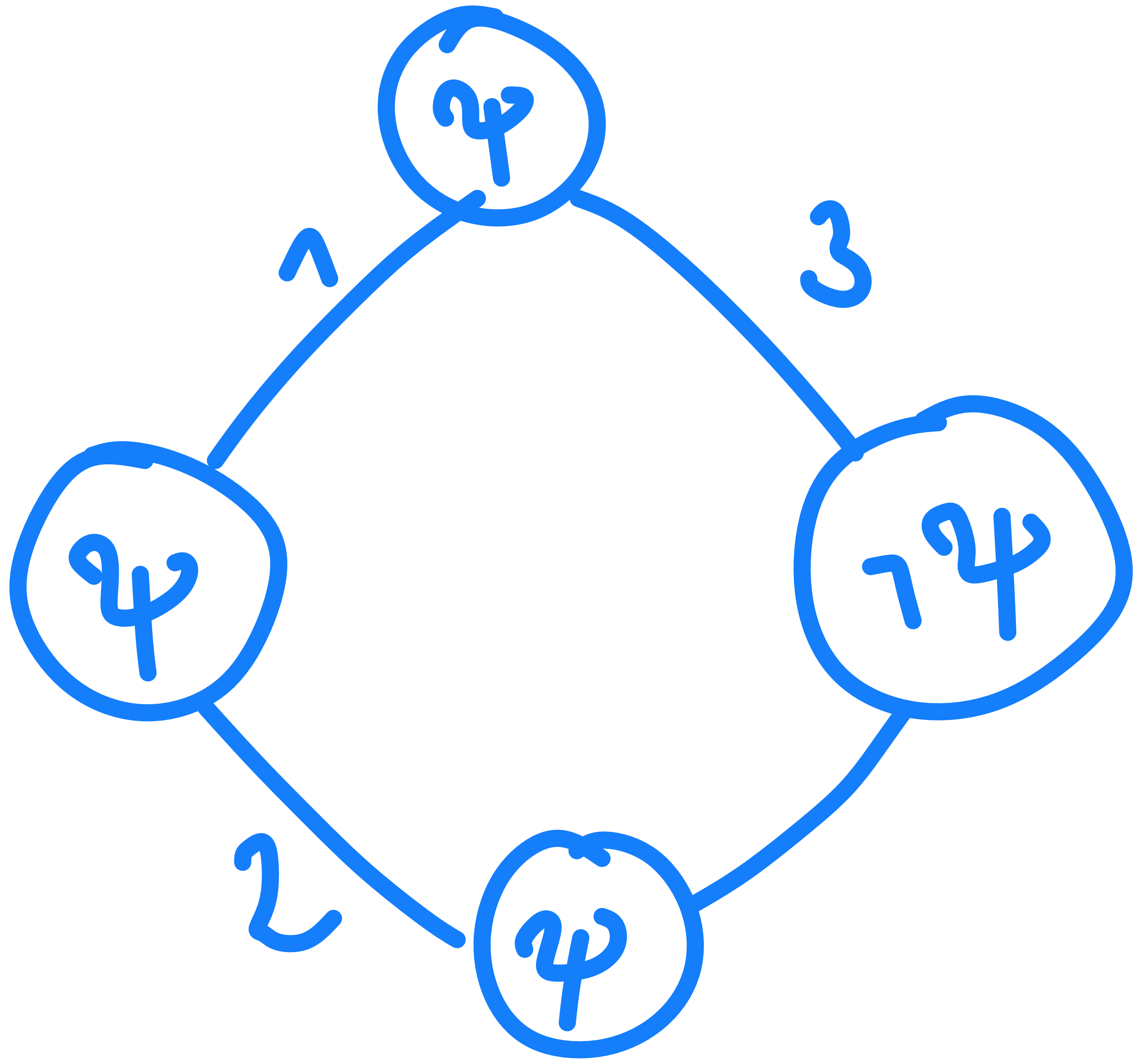
  Reminder: $\psi_1 \,\mathcal{U}_J\, \psi_2$ *means '*ψ_1 *must hold until* ψ_2 *holds and this should happen within* $t \in J$ *time'*

  Remark: All other common connectives and operators are derived by de Morgan's duality

Spatio-Signal Temporal Logic (SSTL)

Somewhere

- $\circledcirc_{[w_1,w_2]}\,\psi$ is the *bounded somewhere* operator
    - 'ψ must hold in a location reachable from the current one with a total cost greater than or equal to w_1 and less than or equal to w_2'
- In which locations does $\circledcirc_{[2,5]}\,\psi$ hold?



Spatio-Signal Temporal Logic (SSTL)

Surround

- $\psi_1 \ \mathcal{S}_{[w_1,w_2]} \ \psi_2$ is the *bounded surround* operator
  - ‘the above formula is true in a location l when l belongs to a subset of locations A, a region, satisfying ψ_1, such that its external boundary $B^+(A)$ (i.e., all the nearest neighbours (not in A) of locations in A) contains only locations satisfying ψ_2 and these locations in $B^+(A)$ must be reached from l by a shortest path of cost between w_1 and w_2’

- Let’s draw a graph in which $\psi_1 \ \mathcal{S}_{[3,6]} \ \psi_2$ holds

Spatio-Temporal Reach and Escape Logic (STREL)

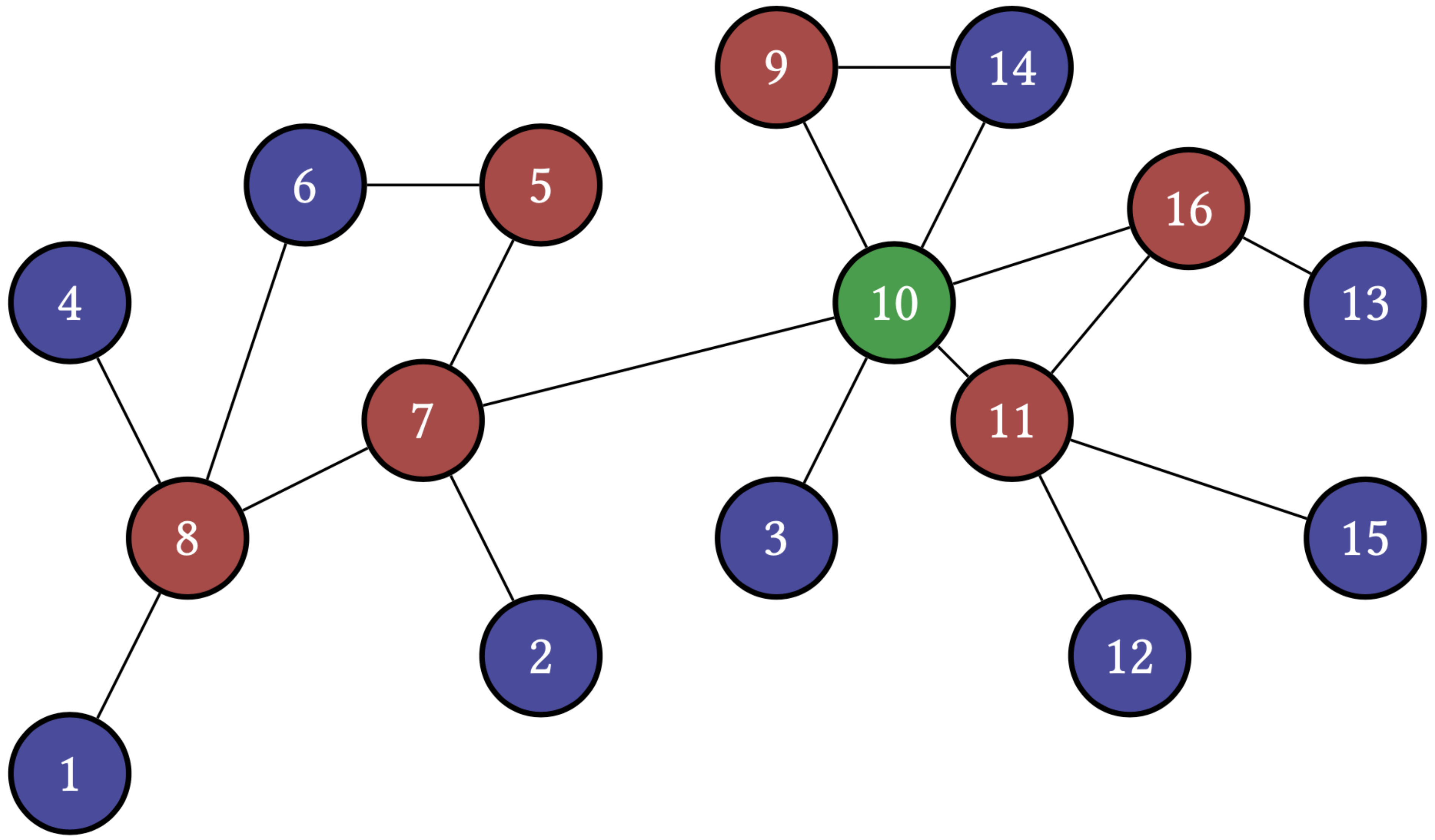
$$\phi := \text{true} \,|\, \mu \,|\, \neg\psi \,|\, \psi_1 \wedge \psi_2 \,|\, \psi_1 \,\mathcal{U}_{[w_1,w_2]}\, \psi_2 \,|\, \ldots \,|\, \psi_1 \,\mathcal{R}^f_d\, \psi_2 \,|\, \mathcal{E}^f_d\, \psi_2$$

- f is a distance function
  - e.g. in a graph this could be 'hops', i.e. going from one node to one of its neighbours is 1 hop

Remark: All other common connectives and operators are derived by de Morgan's duality

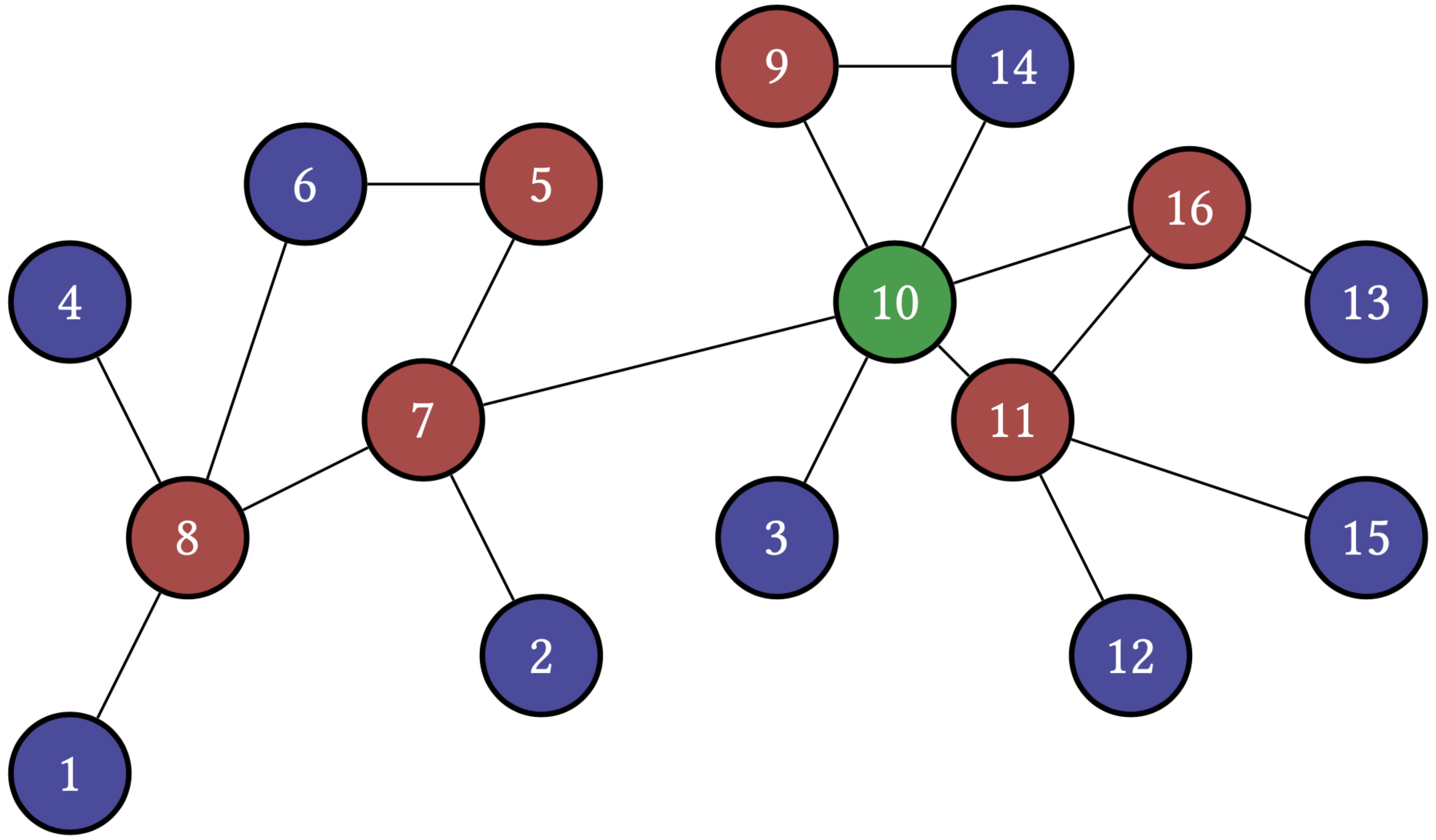
- $\psi_1 \ \mathcal{R}^f_d \ \psi_2$ is the *reachability* operator
  - ‘reaching a location satisfying property ψ_2 passing *only* through locations that satisfy ψ_1, through nodes whose distance form the initial location satisfy the predicate *d*’

$$end\_dev \ \mathcal{R}^{hops}_{m \leq 1} \ router$$



- $\mathscr{E}^{f}_{d}\,\psi$ is the *escape* operator
  - ‘the possibility of escaping from a certain region passing only through locations that satisfy ψ, via a route with distance satisfying the predicate d’

$$\mathcal{E}^{hops}_{m\geq 2}\ \neg end\_dev$$



STREL Examples

- $\Diamond\!\!\!\cdot^{f}_{d}\, \phi := true\ \mathcal{R}^{f}_{d}\ \phi$
- $\boxed{\square}^{f}_{d}\, \phi := \neg \Diamond\!\!\!\cdot^{f}_{d} \neg \phi$

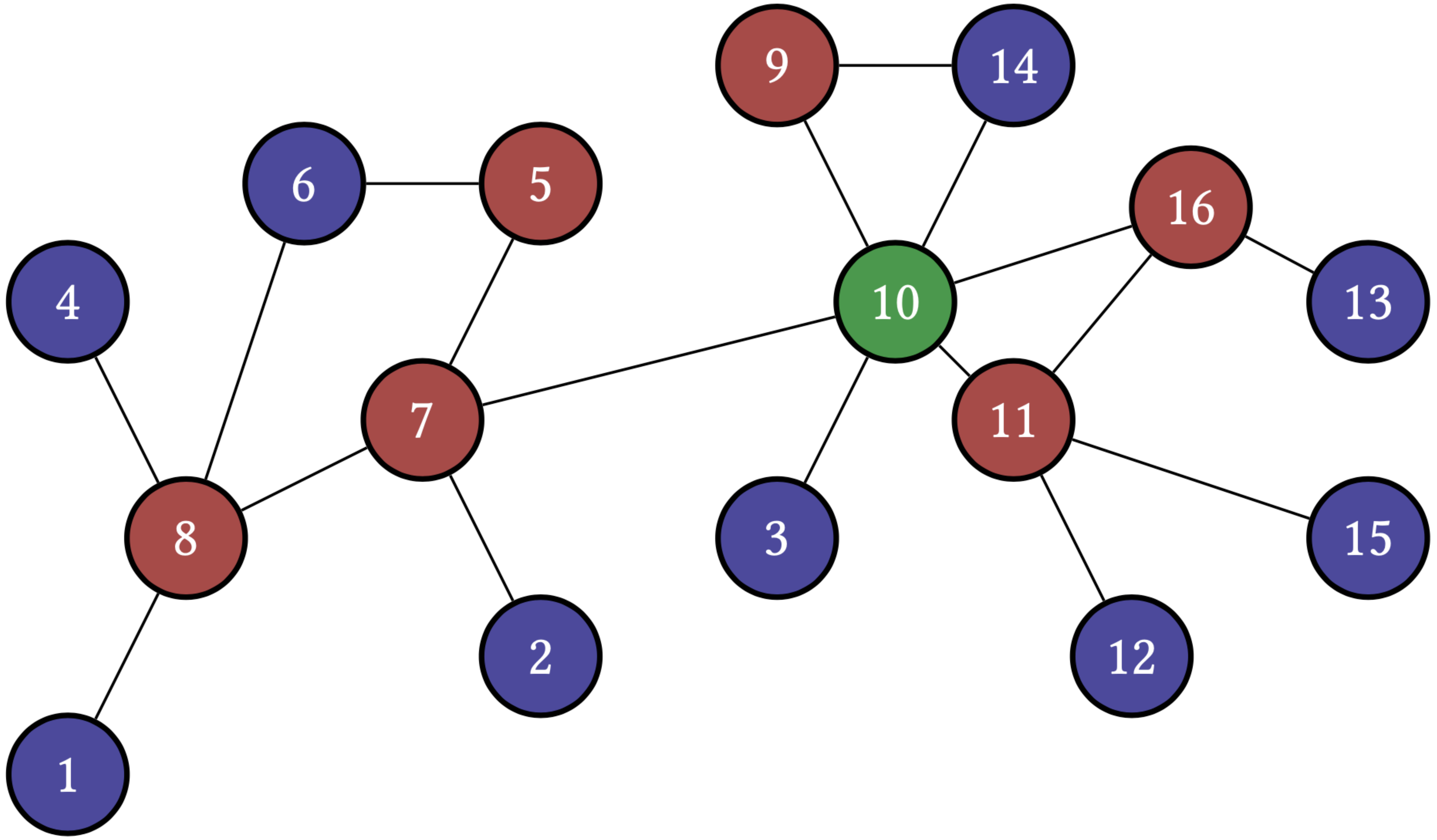
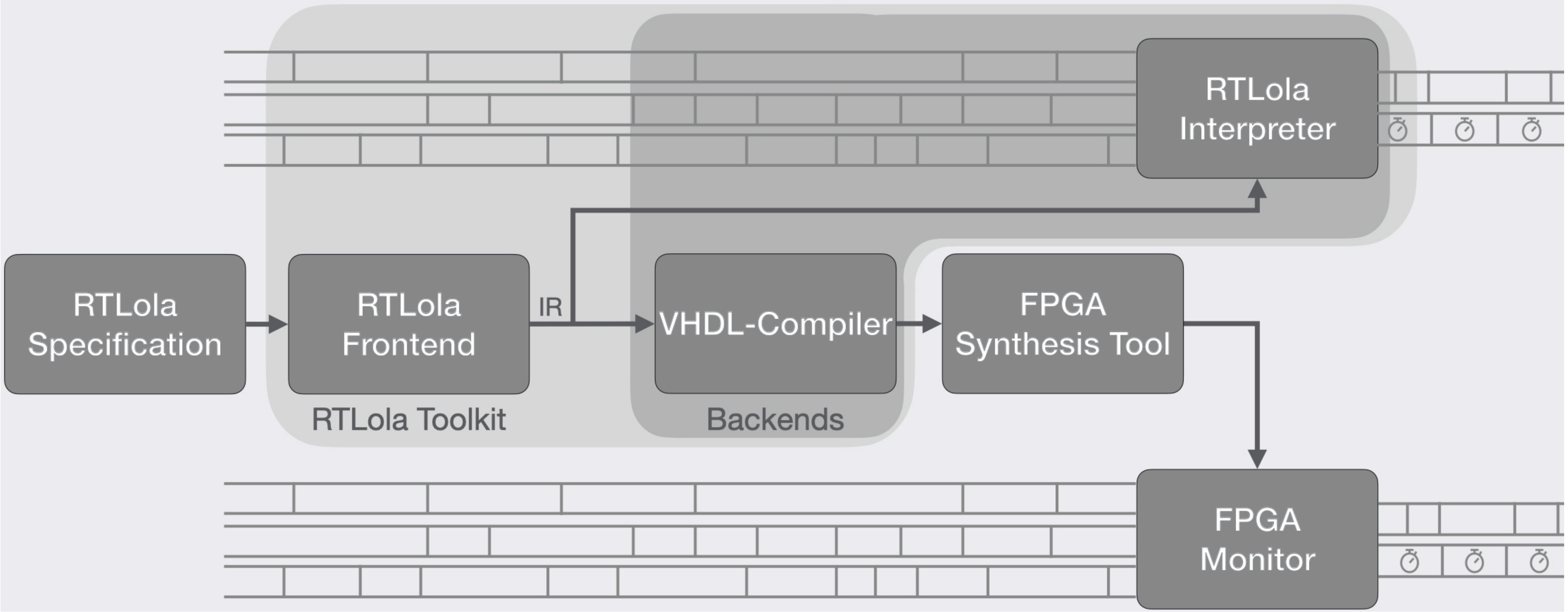


Figure 3: Example of spatial properties. Reachability: $end\_dev\ \mathcal{R}^{hops}_{m\leq 1}\ router$. **Escape:** $\mathcal{E}^{hops}_{m\geq 2}\ \neg end\_dev$. **Somewhere:** $\Diamond\!\!\!\cdot^{hops}_{m\leq 4}\, coord$. **Everywhere:** $\boxed{\square}^{hops}_{m\leq 2}\, router$. **Surround:** $(coord \vee router)\ \circledcirc^{hops}_{m\leq 3}\ end\_dev$.

Temporal Logics vs Programming Languages

Faymonville, P. *et al.* (2019). StreamLAB: Stream-based Monitoring of Cyber-Physical Systems. In: Dillig, I., Tasiran, S. (eds) Computer Aided Verification. CAV 2019. Lecture Notes in Computer Science(), vol 11561. Springer, Cham. https://doi.org/10.1007/978-3-030-25540-4_24

Meet RTLola



https://www.react.uni-saarland.de/tools/rtlola/, June 3, 2022

Why RTLola?

- Very powerful *programming* possibilities, allow for *rule* and *state* based monitors
- As seen, RTLola provides an online monitor
- We can easily emulate STL
- Also, we're in 2022, i.e. IoT, 5G, GPS, everything is super equipped and super fast..
  - Thus, just use the GPS sensor as a "stream" and act accordingly, implementation is easy because we *can program*, instead of writing complicated formulae.
- RTLola monitors are guaranteed to never run out of memory, because the memory consumption is determined statically
- Idea: With a fast enough pipeline, it could be even used for distributed algorithms!

RTLola

Nice!

https://www.react.uni-saarland.de/tools/rtlola/

Until Operator in RTLola

```
output unitlphi1phi2(t: Time) : Bool @ (t+b)| any
close: time == b | !untilphi1phi2(t)
:=
if time <= t+a
 then
   phi1<time>.hold() & unitlphi1phi2[a,b](t).offset(1)
 else
  if time < t+b
   then
    phi1(time).hold() &
    (phi2(time).hold() |
      unitlphi1phi2[a,b](t).offset(1))
   else
    phi1(time).hold() & phi2(time).hold()

trigger unitlphi1phi2[a,b](0)
```

Conclusions

- There are many different temporal logics. However, to specify *correct* formulae is a difficult task
    - “Reading and writing property specifications is not easy for non-experts. Even experts often stare for minutes at relatively small temporal logic formulae (particularly when they have nested "until" operators).”

      — Wikipedia on Runtime Verification
- Runtime verification and specification languages like RTLola make this a lot easier, as they allow for *programming*

References

- Given by the teacher
  - [P1] E. Bartocci, L. Bortolussi, M. Loreti, and L. Nenzi, “Monitoring mobile and spatially distributed cyber-physical systems,” in Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design. ACM, 2017, pp. 146–155.
  - [P2] H. Torfah, “Stream-based monitors for real-time properties,” in Intl. Conf. on Runtime Verification. Springer, 2019, pp. 91–110.
  - [P3] Ezio Bartocci, Luca Bortolussi, Laura Nenzi, Simone Silvetti: MoonLight: A Lightweight Tool for Monitoring Spatio-Temporal Properties.

Demo

- Head over to https://www.react.uni-saarland.de/tools/rtlola/
- Download the binaries for your OS
- `cd` into the directory
- write a specification file, e.g. (as seen in my snake demo):

```
input xcord: Float64
output hitting_left_wall := xcord < 100.0
trigger hitting_left_wall
     "NEAR LEFT WALL"
```

- Modify your program to write into `stdout` in a CSV format
  - don't forget to also print the header, e.g. "xcord, ycord, time" at the beginning of your stream and don't forget the new line \n after every row
- pipe the output into the RTLola interpreter as follows, e.g. with the snake example:

```
python snake.py | ./streamlab monitor snake.lola --online --stdin --stdout
```

  - Here, `snake.lola` is the specification file
- You can find more examples and details here: https://www.react.uni-saarland.de/tools/rtlola/tutorial.html
  The example (drone) data can be downloaded here: https://www.react.uni-saarland.de/tools/rtlola/examples/tutorial.zip
- Enjoy!