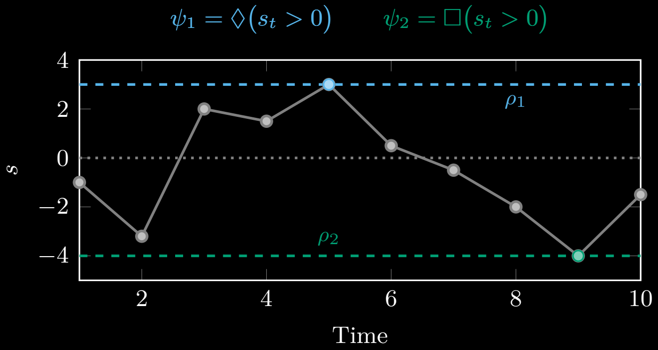
SignalTemporalLogic.jl

Introduction



Robert Moss
Stanford AA228V/CS238V
mossr@cs.stanford.edu

Installation

Installation

You can install the `SignalTemporalLogic.jl` package via:

```julia
using Pkg
Pkg.add("SignalTemporalLogic")
```

Installation

You can install the `SignalTemporalLogic.jl` package via:

```julia
using Pkg
Pkg.add("SignalTemporalLogic")
```

Then you can run this to use the package:

```julia
using SignalTemporalLogic
```

Specifications

Let's define the following *specification* over a trajectory τ:

Specifications

Let's define the following *specification* over a trajectory τ:

$$\psi(\tau) = \Diamond\big(s_t > 0\big) \qquad \text{“\textit{Eventually} } (\Diamond)\text{\textit{, the state will be greater than zero.}”}$$

Specifications

Let's define the following *specification* over a trajectory τ:

$$\psi(\tau) = \Diamond(s_t > 0) \qquad \text{“\textit{Eventually} } (\Diamond)\text{\textit{, the state will be greater than zero.}”}$$

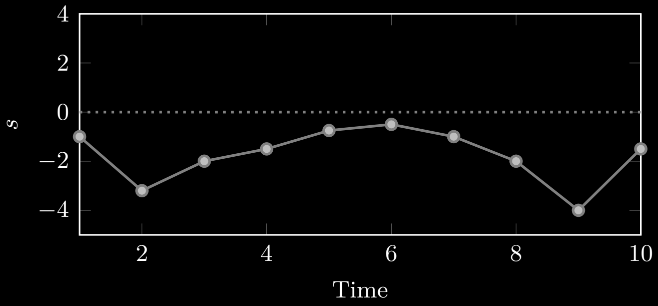
```julia
julia> using SignalTemporalLogic
```

Specifications

Let's define the following *specification* over a trajectory τ:

$$\psi(\tau) = \lozenge\big(s_t > 0\big) \qquad \text{“\textit{Eventually} } (\lozenge)\text{\textit{, the state will be greater than zero.}”}$$

```julia
julia> using SignalTemporalLogic
julia> τ = [-1.0, -3.2, -2.0, -1.5, -0.75, -0.5, -1.0, -2.0, -4.0, -1.5];
```

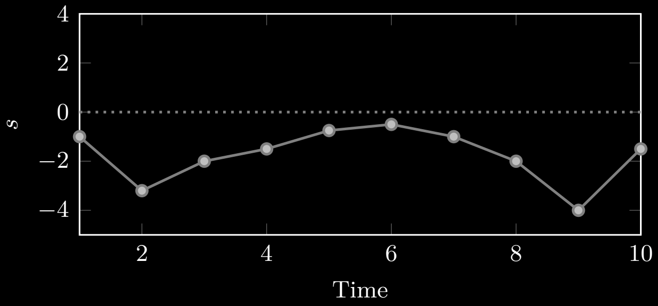


Specifications

Let's define the following *specification* over a trajectory τ:

$$\psi(\tau) = \Diamond\big(s_t > 0\big) \qquad \text{“\textit{Eventually} } (\Diamond)\text{\textit{, the state will be greater than zero.}”}$$

```
julia> using SignalTemporalLogic
julia> τ = [-1.0, -3.2, -2.0, -1.5, -0.75, -0.5, -1.0, -2.0, -4.0, -1.5];
julia> ψ = @formula ◊(sₜ -> sₜ > 0);
```

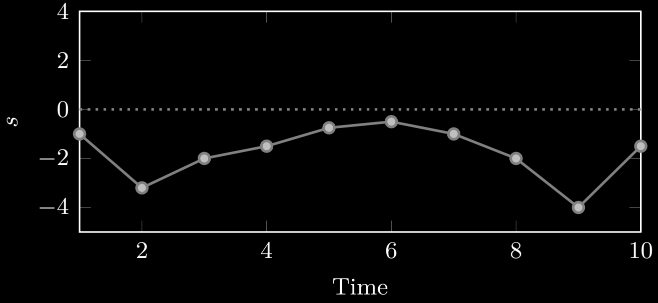


Specifications

Let's define the following *specification* over a trajectory τ:

$$\psi(\tau) = \lozenge(s_t > 0) \qquad \text{“\textit{Eventually} } (\lozenge)\text{\textit{, the state will be greater than zero.}”}$$

```julia
julia> using SignalTemporalLogic
julia> τ = [-1.0, -3.2, -2.0, -1.5, -0.75, -0.5, -1.0, -2.0, -4.0, -1.5];
julia> ψ = @formula ◊(sₜ -> sₜ > 0);
julia> ψ(τ)
false
```

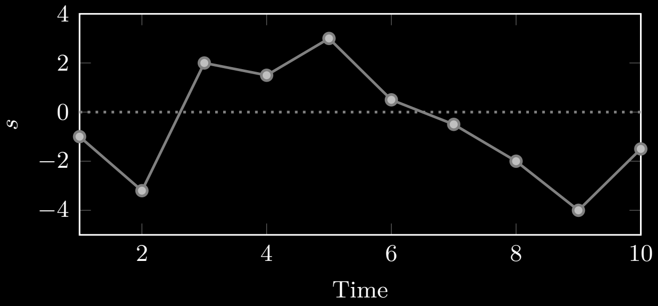


Specifications

Let's define the following *specification* over a trajectory τ:

$$\psi(\tau) = \Diamond\big(s_t > 0\big) \qquad \text{“\textit{Eventually} } (\Diamond)\text{\textit{, the state will be greater than zero.}”}$$

```
julia> using SignalTemporalLogic
julia> τ = [-1.0, -3.2, 2.0, 1.5, 3.0, 0.5, -0.5, -2.0, -4.0, -1.5];
julia> ψ = @formula ◊(sₜ -> sₜ > 0);
julia> ψ(τ)
true
```



Robustness

You can also compute the *robustness* of a trajectory τ.

Robustness

You can also compute the *robustness* of a trajectory τ.

```
julia> using SignalTemporalLogic
julia> τ = [-1.0, -3.2, 2.0, 1.5, 3.0, 0.5, -0.5, -2.0, -4.0, -1.5];
julia> ψ₁ = @formula ◊(sₜ -> sₜ > 0);
```

Robustness

You can also compute the *robustness* of a trajectory τ.

```
julia> using SignalTemporalLogic
julia> τ = [-1.0, -3.2, 2.0, 1.5, 3.0, 0.5, -0.5, -2.0, -4.0, -1.5];
julia> ψ₁ = @formula ◊(sₜ -> sₜ > 0);
julia> ρ₁ = ρ(τ, ψ₁)
3.0
```

Robustness

You can also compute the *robustness* of a trajectory τ.

```
julia> using SignalTemporalLogic
julia> τ = [-1.0, -3.2, 2.0, 1.5, 3.0, 0.5, -0.5, -2.0, -4.0, -1.5];
julia> ψ₁ = @formula ◊(sₜ -> sₜ > 0);
julia> ρ₁ = ρ(τ, ψ₁)
3.0
julia> ψ₂ = @formula □(sₜ -> sₜ > 0);
```

Robustness

You can also compute the *robustness* of a trajectory τ.

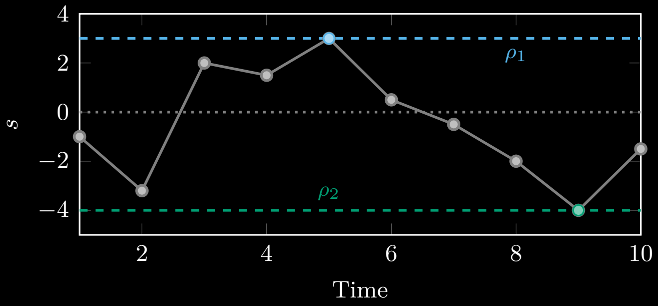
```
julia> using SignalTemporalLogic
julia> τ = [-1.0, -3.2, 2.0, 1.5, 3.0, 0.5, -0.5, -2.0, -4.0, -1.5];
julia> ψ₁ = @formula ◊(sₜ → sₜ > 0);
julia> ρ₁ = ρ(τ, ψ₁)
3.0
julia> ψ₂ = @formula □(sₜ → sₜ > 0);
julia> ρ₂ = ρ(τ, ψ₂)
-4.0
```

Robustness

You can also compute the *robustness* of a trajectory τ.

```
julia> using SignalTemporalLogic
julia> τ = [-1.0, -3.2, 2.0, 1.5, 3.0, 0.5, -0.5, -2.0, -4.0, -1.5];
julia> ψ₁ = @formula ◊(sₜ -> sₜ > 0);
julia> ρ₁ = ρ(τ, ψ₁)
3.0
julia> ψ₂ = @formula □(sₜ -> sₜ > 0);
julia> ρ₂ = ρ(τ, ψ₂)
-4.0
```

$\psi_1 = \Diamond(s_t > 0)$ $\psi_2 = \Box(s_t > 0)$



Use in Projects

Wrappers are provided in the textbook/projects:

Use in Projects

Wrappers are provided in the textbook/projects:

Linear temporal logic (LTL)

```julia
struct LTLSpecification <: Specification
    formula # formula specified using SignalTemporalLogic.jl
end
evaluate(ψ::LTLSpecification, τ) = ψ.formula([step.s for step in τ])
```

Use in Projects

Wrappers are provided in the textbook/projects:

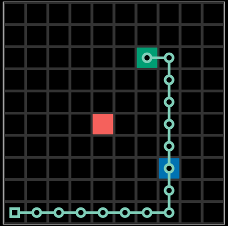
Linear temporal logic (LTL)

```
struct LTLSpecification <: Specification
    formula # formula specified using SignalTemporalLogic.jl
end
evaluate(ψ::LTLSpecification, τ) = ψ.formula([step.s for step in τ])
```

Signal temporal logic (STL, includes time interval)

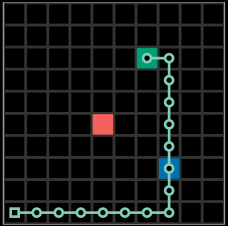
```
struct STLSpecification <: Specification
    formula # formula specified using SignalTemporalLogic.jl
    I       # time interval (e.g. 3:10)
end
evaluate(ψ::STLSpecification, τ) = ψ.formula([step.s for step in τ[ψ.I]])
```

Grid World



Grid World

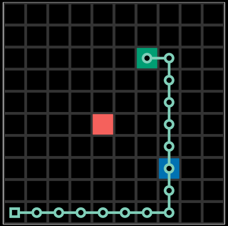
$F(s_t)$: the state s at time t contains an obstacle



Grid World

$F(s_t)$: the state s at time t contains an obstacle

$G(s_t)$: the state s at time t is the goal

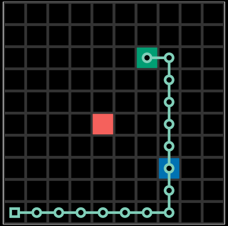


Grid World

$F(s_t)$: the state s at time t contains an obstacle

$G(s_t)$: the state s at time t is the goal

$C(s_t)$: the state s at time t is the checkpoint



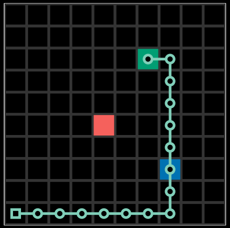
Grid World

$F(s_t)$: the state s at time t contains an obstacle

$G(s_t)$: the state s at time t is the goal

$C(s_t)$: the state s at time t is the checkpoint

$$\psi = \underbrace{\Diamond G(s_t)}_{\text{reaches goal}}$$



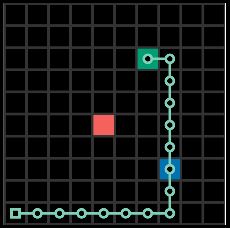
Grid World

$F(s_t)$: the state s at time t contains an obstacle

$G(s_t)$: the state s at time t is the goal

$C(s_t)$: the state s at time t is the checkpoint

$$\psi = \underbrace{\Diamond G(s_t)}_{\text{reaches goal}} \wedge \underbrace{\neg C(s_t)\, \mathcal{U}\, G(s_t)}_{\text{reach checkpoint before goal}}$$



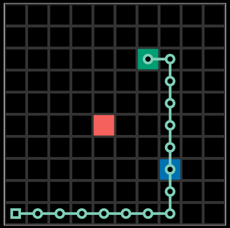
Grid World

$F(s_t)$: the state s at time t contains an obstacle

$G(s_t)$: the state s at time t is the goal

$C(s_t)$: the state s at time t is the checkpoint

$$\psi = \underbrace{\Diamond G(s_t)}_{\text{reaches goal}} \wedge \underbrace{\neg C(s_t)\,\mathcal{U}\,G(s_t)}_{\text{reach checkpoint before goal}} \wedge \underbrace{\Box\neg F(s_t)}_{\text{always avoid obstacles}}$$



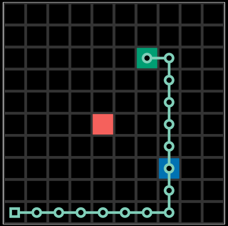
Grid World

$F(s_t)$: the state s at time t contains an obstacle

$G(s_t)$: the state s at time t is the goal

$C(s_t)$: the state s at time t is the checkpoint

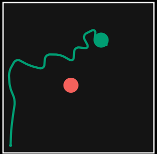
$$\psi = \underbrace{\Diamond G(s_t)}_{\text{reaches goal}} \wedge \underbrace{\neg C(s_t)\,\mathcal{U}\,G(s_t)}_{\text{reach checkpoint before goal}} \wedge \underbrace{\Box\neg F(s_t)}_{\text{always avoid obstacles}}$$



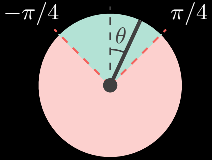
```
F = @formula sₜ → sₜ == [5, 5]
G = @formula sₜ → sₜ == [7, 8]
C = @formula sₜ → sₜ == [8, 3]
ψ = LTLSpecification(@formula ◊(G) ∧ 𝒰(¬G, C) ∧ □(¬F))
```

Continuum World

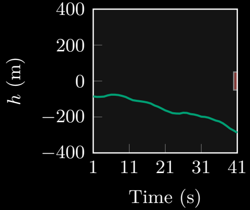
System	Property	Implementation

Continuum World

	"Reach the goal without hitting the obstacle"
	$G(s_t)$: s_t is in the goal region
	$F(s_t)$: s_t is in the obstacle region
	$\psi = \lozenge G(s_t) \wedge \square \neg F(s_t)$
		`G = @formula s→norm(s.-[6.5,7.5])≤0.5`
		`F = @formula s→norm(s.-[4.5,4.5])≤0.5`
		`ψ = @formula ◊(G) ∧ □(¬F)`

Inverted Pendulum

System	Property	Implementation
Inverted Pendulum 	*"Keep the pendulum balanced"* $B(s_t)$: $\lvert\theta_t\rvert \leq \pi/4$ $\psi = \square B(s_t)$	`B = @formula s→abs(s[1])≤π/4` `ψ = @formula □(B)`

Aircraft Collision Avoidance

System	Property	Implementation
Aircraft Collision Avoidance 	*"Ensure at least 50 meters relative altitude between 40 and 41 seconds"* $S(s_t)$: $\lvert h_t\rvert \geq 50$ $\psi = \square_{[40,41]} S(s_t)$	`S = @formula s→abs(s[1])≥50` `ψ = @formula □(40:41, S)`