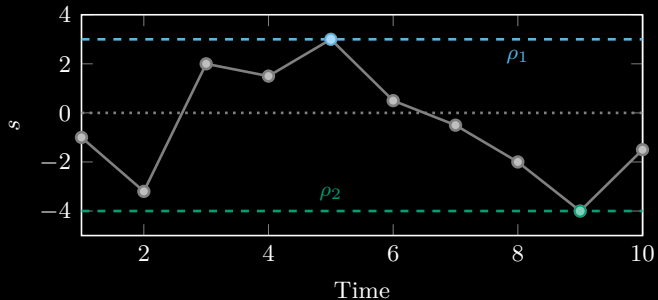# SignalTemporalLogic.jl
INTRODUCTION

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

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:

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

SPECIFICATIONS

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

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

```julia
julia> using SignalTemporalLogic
```
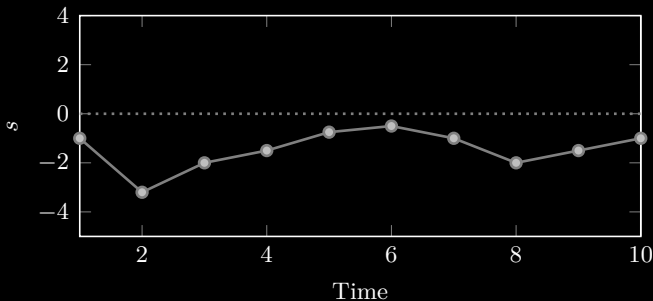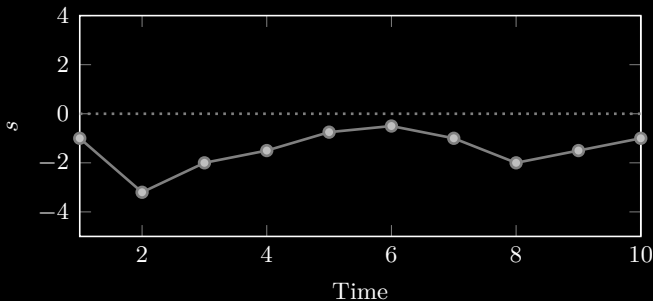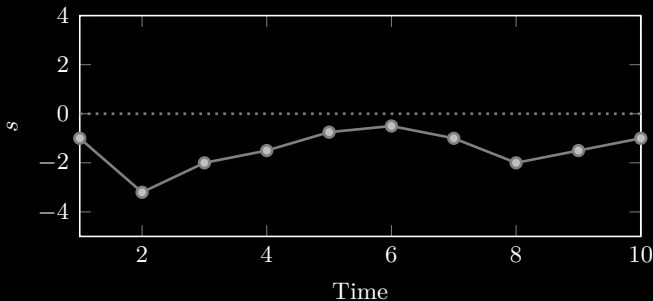
Specifications

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

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

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

SPECIFICATIONS

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

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

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

τ # \tau<TAB>

SPECIFICATIONS

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

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

```julia
julia> using SignalTemporalLogic
julia> τ = [-1.0, -3.2, -2.0, -1.5, -0.75, -0.5, -1.0, -2.0, -1.5, -1.0];
julia> ψ = @formula ◊(sₜ → sₜ > 0);
```
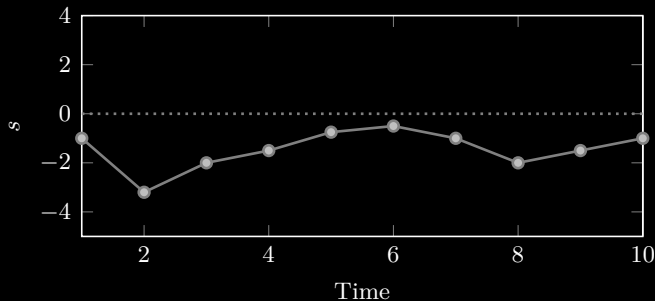
τ # \tau<TAB>

Specifications

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

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

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

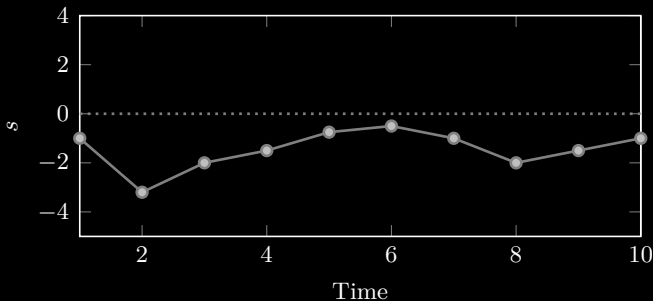τ # \tau<TAB>

ψ # \psi<TAB>

◊ # \lozenge<TAB>

Specifications

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

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

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


τ # \tau<TAB>

ψ # \psi<TAB>

◊ # \lozenge<TAB>
```
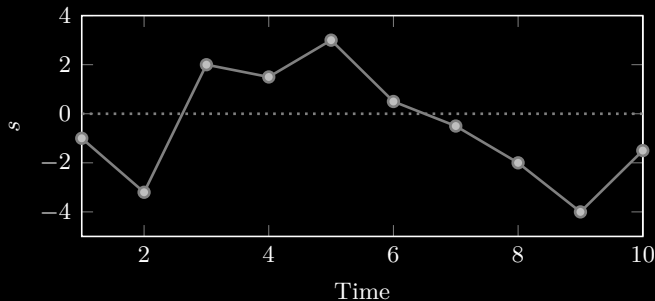
Specifications

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

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

```julia
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 ◊(s_t → s_t > 0);
julia> ψ(τ)
true


τ # \tau<TAB>

ψ # \psi<TAB>

◊ # \lozenge<TAB>
```

Robustness

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

Robustness

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

```julia
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
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
```

ρ # \rho<TAB>

Robustness

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

```julia
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);
```
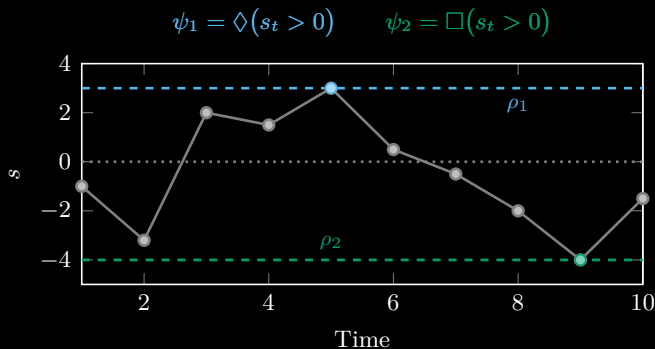
ρ # \rho<TAB>

□ # \square<TAB>

Robustness

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

```julia
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


ρ # \rho<TAB>
□ # \square<TAB>
```

Robustness

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

```julia
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


ρ # \rho<TAB>

□ # \square<TAB>
```

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)

```julia
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)

```julia
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]])
```
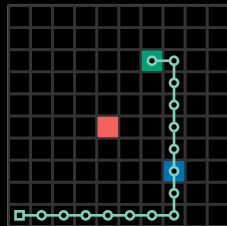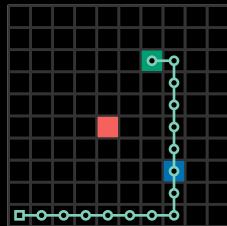
GRID WORLD

Grid World

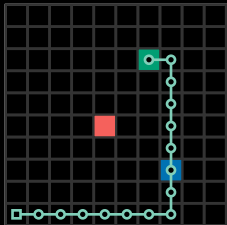$F(s_t)$: the state s at time t contains an obstacle

GRID WORLD

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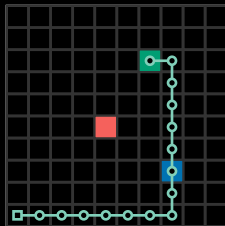
$G(s_t)$: the state s at time t is the goal

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
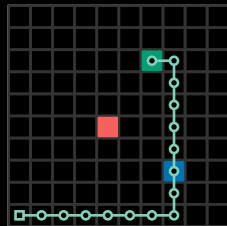
GRID WORLD

$F(s_t)$: 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

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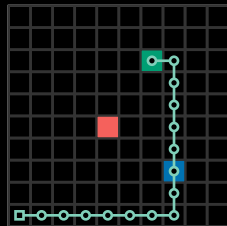
GRID WORLD

$F(s_t)$: 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

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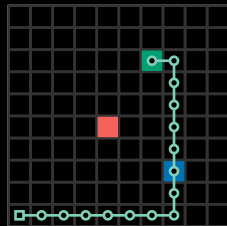
GRID WORLD

$F(s_t)$: 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

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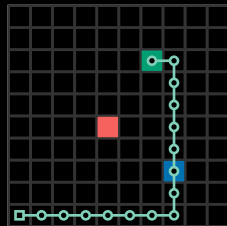
Grid World

$F(s_t)$: 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

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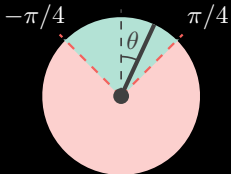


```
F = @formula s_t → s_t == [5, 5]
G = @formula s_t → s_t == [7, 8]
C = @formula s_t → s_t == [8, 3]
ψ = LTLSpecification(@formula ◊(G) ∧ 𝒰(¬G, C) ∧ □(¬F))
```
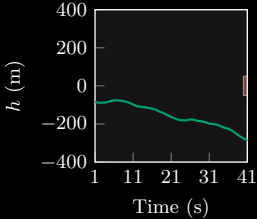
Continuum World

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

$\psi = \Diamond G(s_t) \land \Box \neg F(s_t)$

```
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"
$B(s_t)$: $|\theta_t| \leq \pi/4$
$\psi = \Box B(s_t)$

```
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"

$S(s_t): |h_t| \geq 50$

$\psi = \Box_{[40,41]} S(s_t)$

```
S = @formula s→abs(s[1])≥50
ψ = @formula □(40:41, S)
```

Resources

Resources

github.com/sisl/SignalTemporalLogic.jl

Resources

github.com/sisl/SignalTemporalLogic.jl

github.com/mossr/STL-mini-lecture