Developing a Formal Methods Solution for System Monitoring

Asst. Prof. Ebru Aydın Göl

Department of Computer Engineering

Middle East Technical University

Outline

- System monitoring
- Signal temporal logic
- Requirements mining problem
- Mining specifications for system monitoring

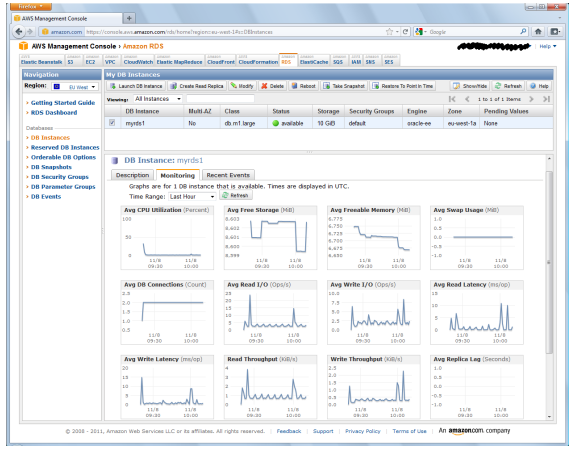
System Monitoring

system monitor : … hardware or software used to monitor resources and performance in a computer system ….

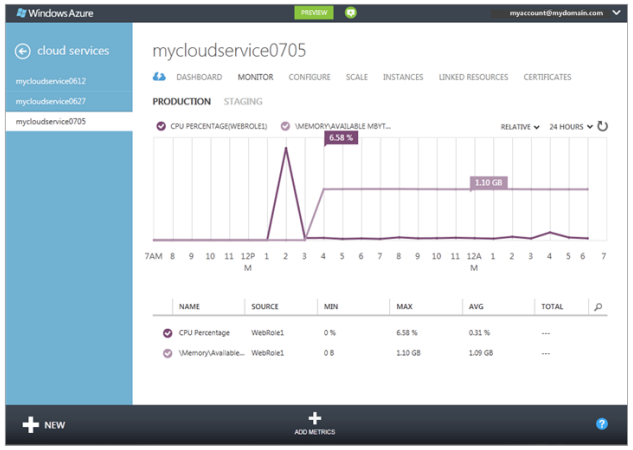
altosoft

idera

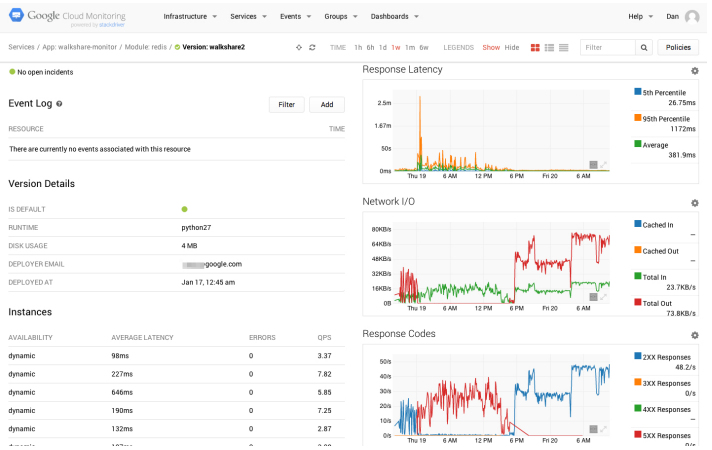
Cloud Monitoring



Amazon

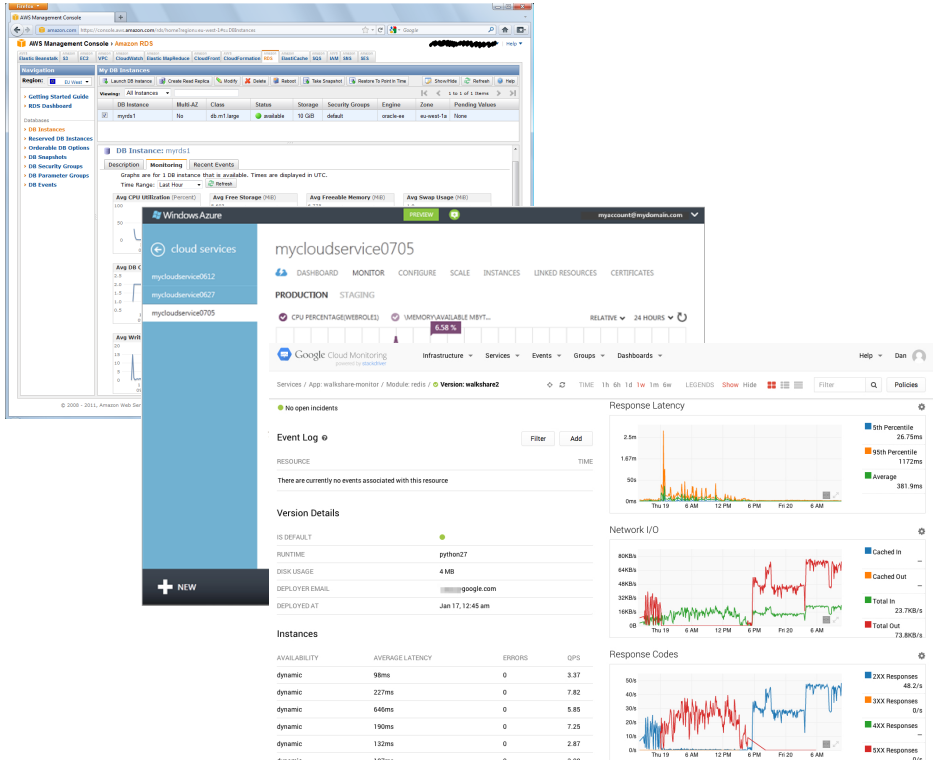


Microsoft



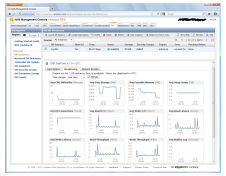
Google

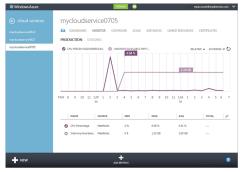
System Monitoring



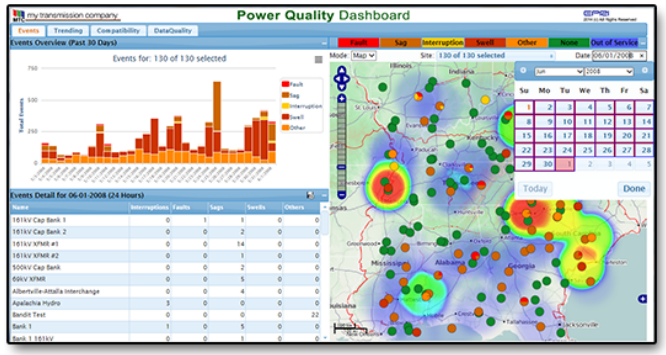
- Visualize the performance and health
- Quickly gain insight in possible problems
- Analyze root-cause

System Monitoring





- Visualize the performance and health
- Quickly gain insight in possible problems
- Analyze root-cause

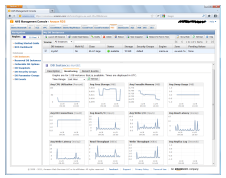
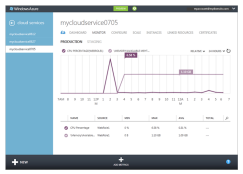
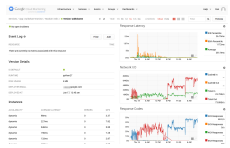
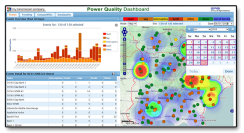


[GPA Grid Protection Alliance]

ICU health monitor

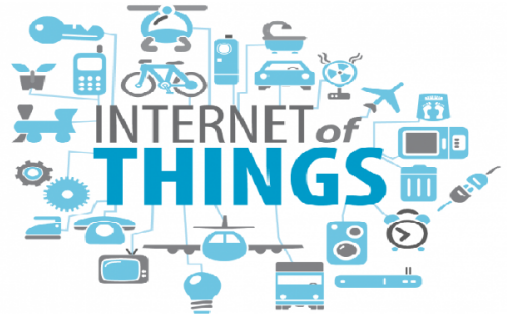
- Here the focus is on software systems
- Various systems are monitored
  - Power systems
  - Human/health, driver etc.

System Monitoring


- Visualize the performance and health
- Quickly gain insight in possible problems
- Analyze root-cause

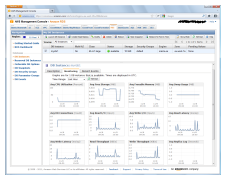
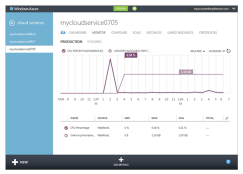
- Here the focus is on software systems
- Various systems are monitored
  - Power systems
  - Human/health, driver etc.

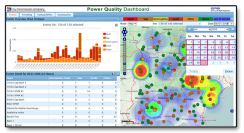


IOT

Industry 4.0
Automated connected systems with monitors
Sensors for health monitoring

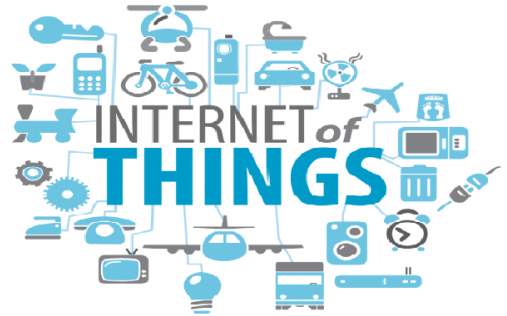
System Monitoring




- Visualize the performance and health
- Quickly gain insight in possible problems
- Analyze root-cause

- Here the focus is on software systems
- Various systems are monitored
  - Power systems
  - Human/health, driver etc.



IOT

Industry 4.0
Automated connected systems with monitors
Sensors for health monitoring

Focus on
- software systems
- alerting mechanism

System Monitoring - Alerting

- Alerting/paging on anomalies
- Optimally before the system breaks, notify on-calls
- Combined with the visuals, the goal is to avoid breaks, locate the root cause quickly

System Monitoring - Alerting

- Alerting/paging on anomalies
- Optimally before the system breaks, notify on-calls
- Combined with the visuals, the goal is to avoid breaks, locate the root cause quickly

- Define rules, when metrics violate your rules, send notification.

System Monitoring - Alerting

- Alerting/paging on anomalies
- Optimally before the system breaks, notify on-calls
- Combined with the visuals, the goal is to avoid breaks, locate the root cause quickly

- Define rules, when metrics violate your rules, send notification.

RULES:

- X% of the servers should be up.
- Average latency should be below T.
- The number of servers restarted in the last 5 minutes should be less than 0.05*total_server_count
- CPU usage is above TC.
- ….

System Monitoring - Alerting

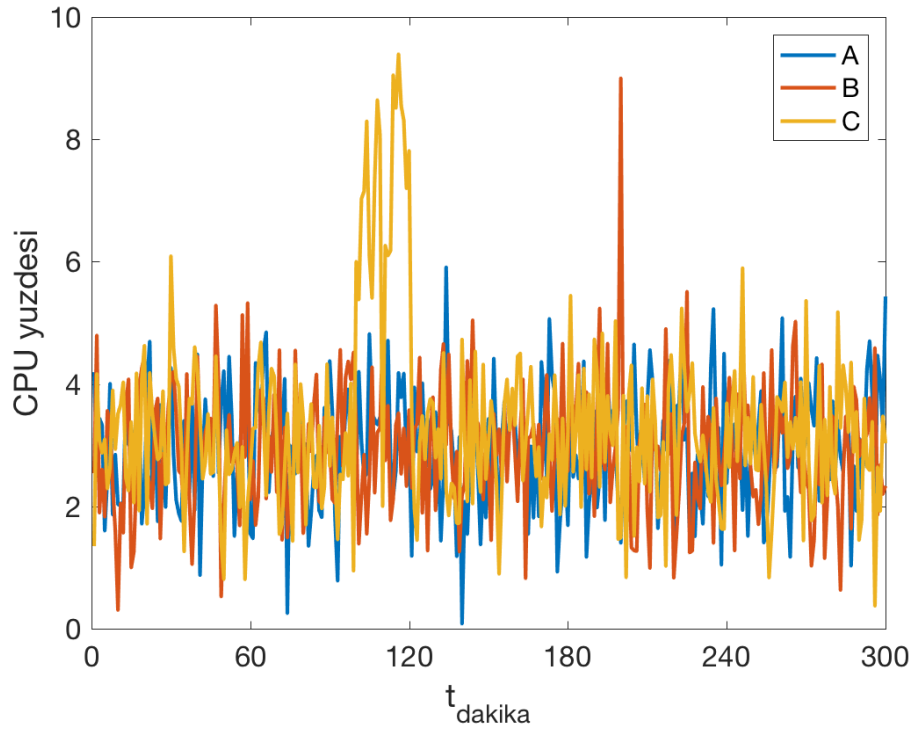
Mistakes in creating alerts, inaccuracies at the setup phase is expensive:

- False negatives: The system fails to alert when necessary.
- E.g Bad weather, connection problem, latency for a cluster is high. If you know, you would act on it (redirect the traffic) before the system breaks. (alerts should be actionable)
- False positives: Annoying, you see the alarm, but can not act on it. The system alerts when there is no problem.

System Monitoring - Alerting

Mistakes in creating alerts, inaccuracies at the setup phase is expensive:

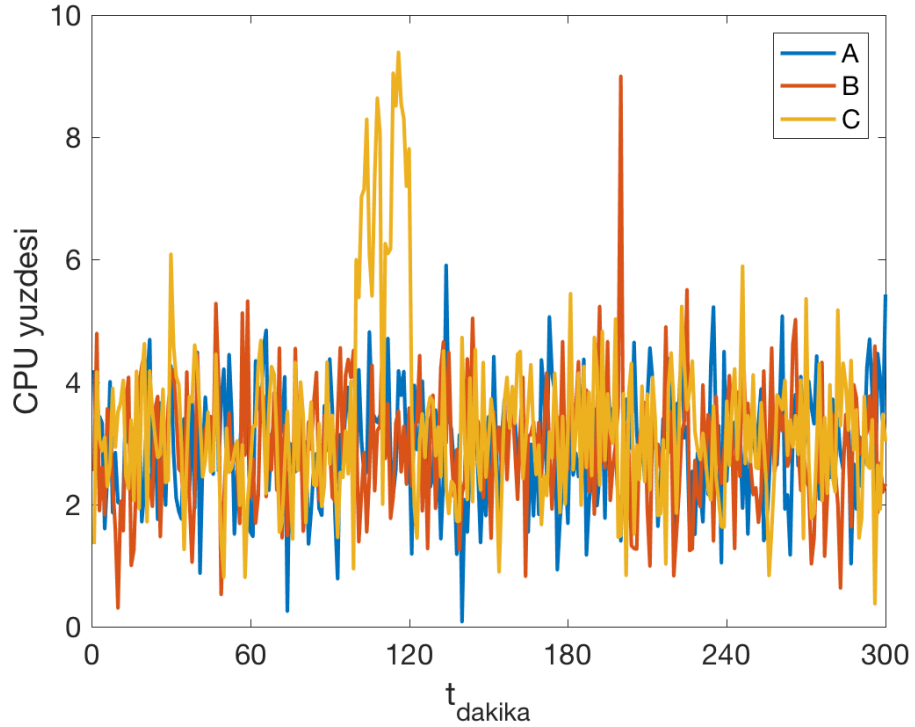
- False negatives: The system fails to alert when necessary.
- E.g Bad weather, connection problem, latency for a cluster is high. If you know, you would act on it (redirect the traffic) before the system breaks.
- False positives: Annoying, you see the alarm, but can not act on it. The system alerts when there is no problem.

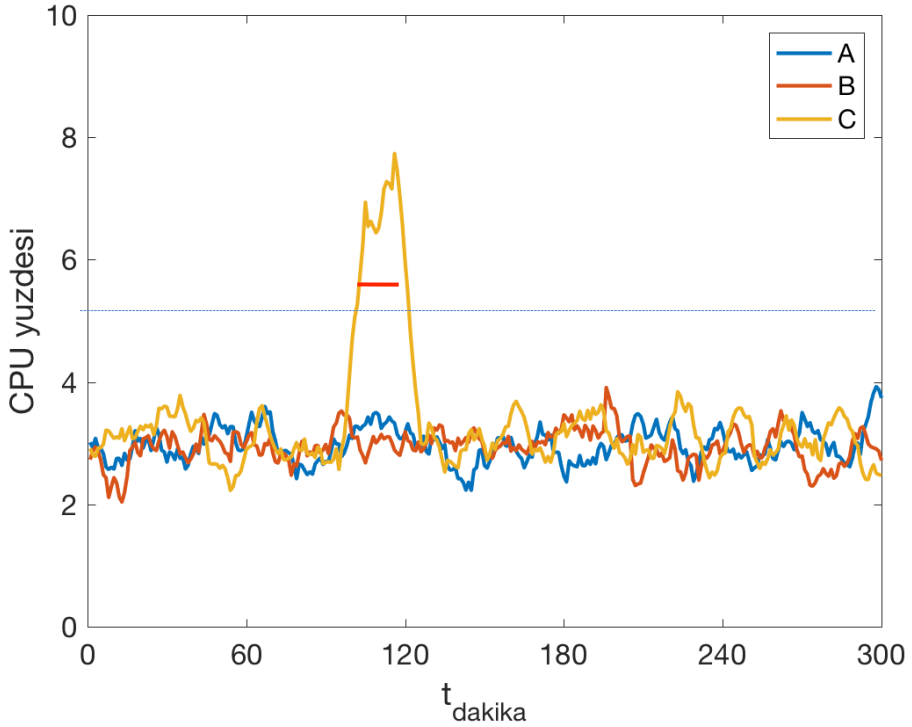


System Monitoring - Alerting

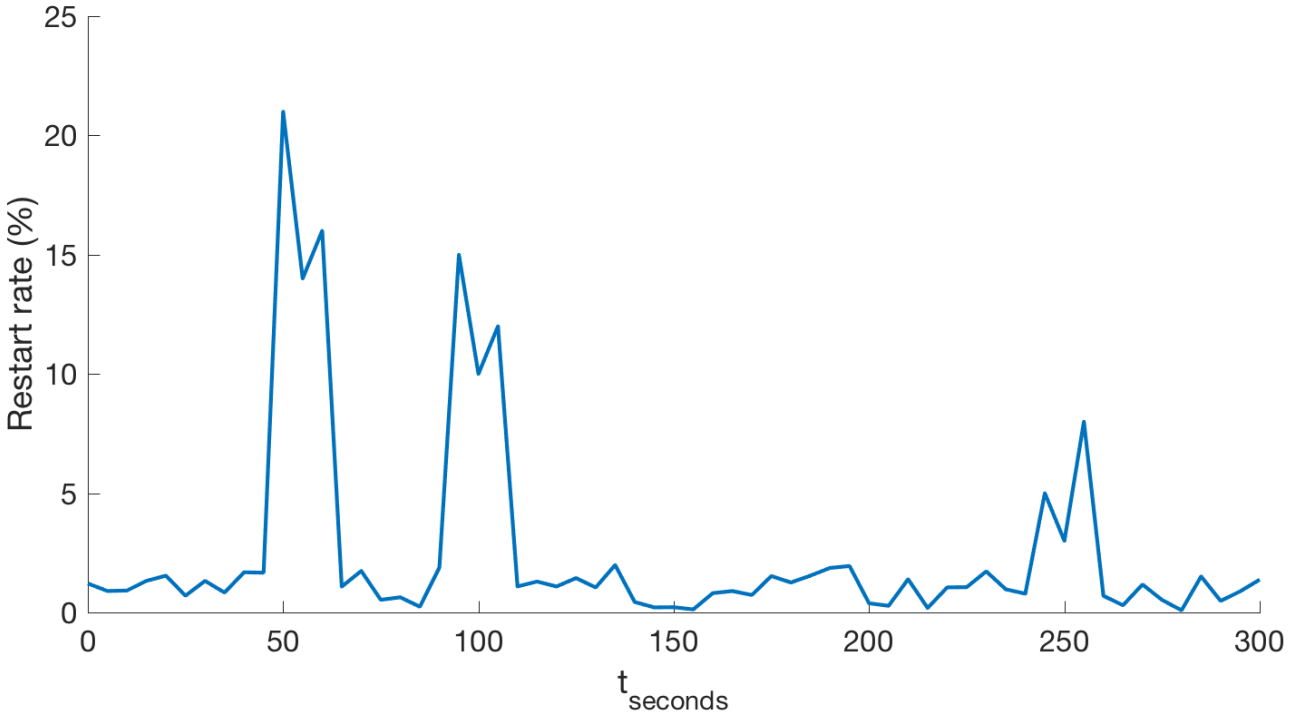
Mistakes in creating alerts, inaccuracies at the setup phase is expensive:

- False negatives: The system fails to alert when necessary.
- E.g Bad weather, connection problem, latency for a cluster is high. If you know, you would act on it (redirect the traffic) before the system breaks.
- False positives: Annoying, you see the alarm, but can not act on it. The system alerts when there is no problem.

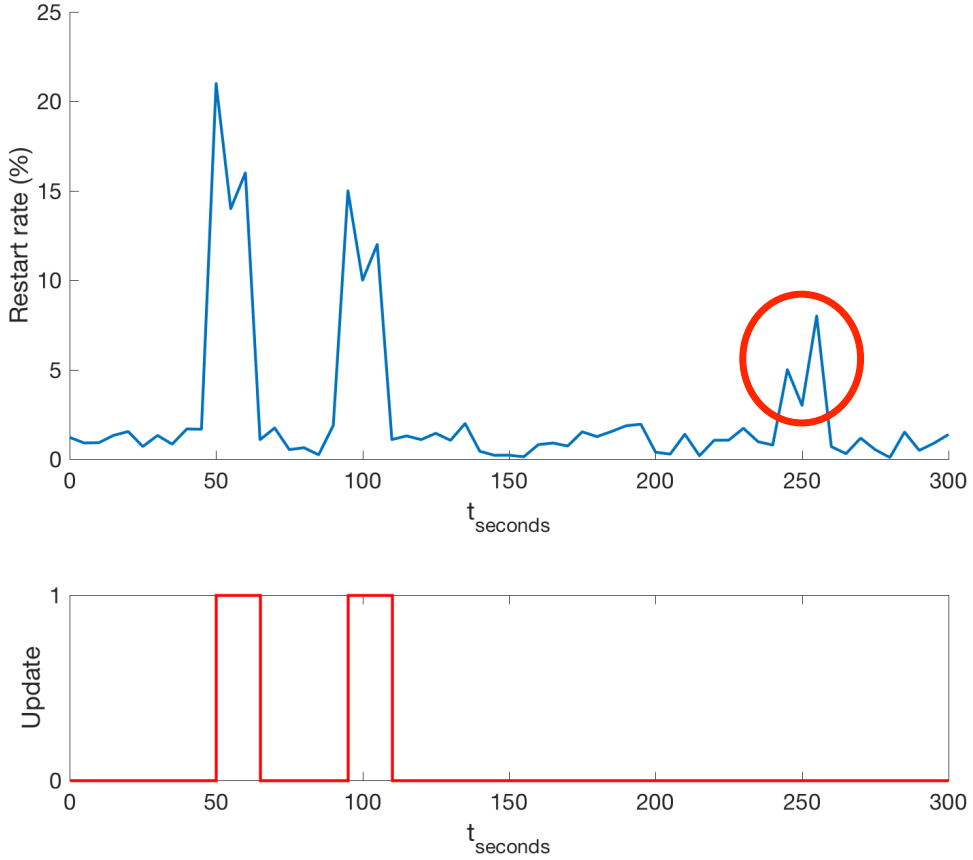




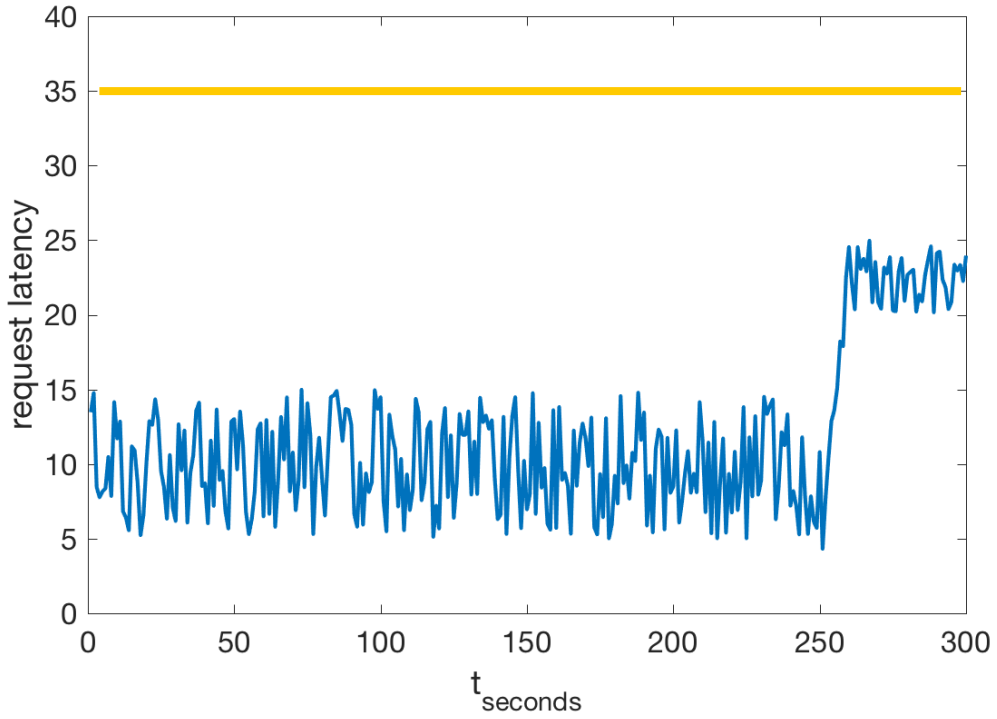
System Monitoring – Alerting - Examples

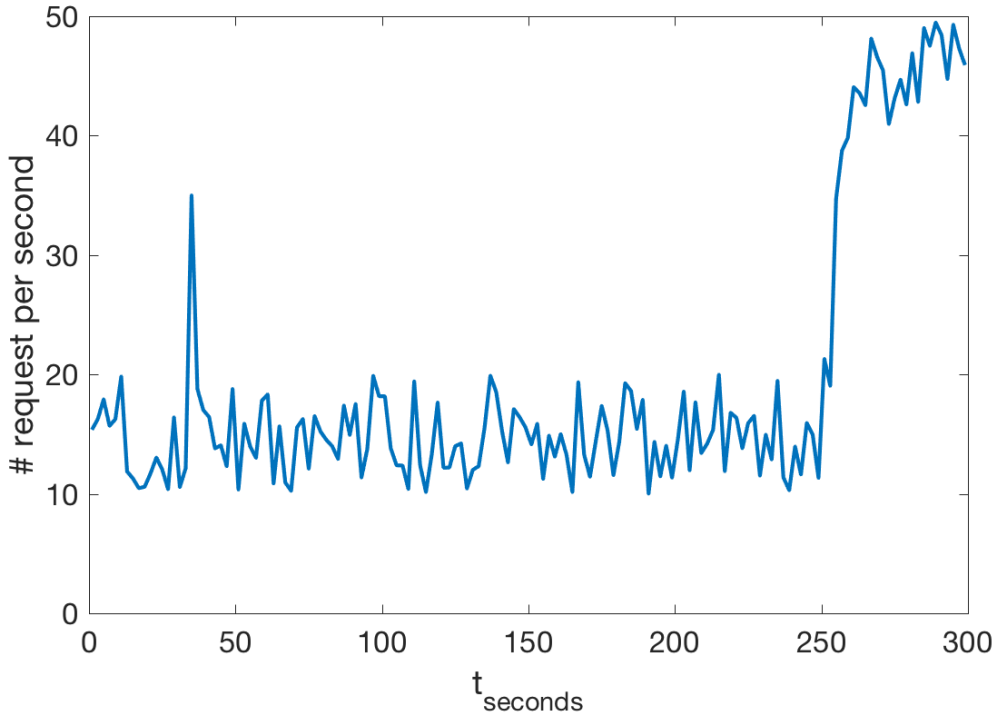


System Monitoring – Alerting - Examples



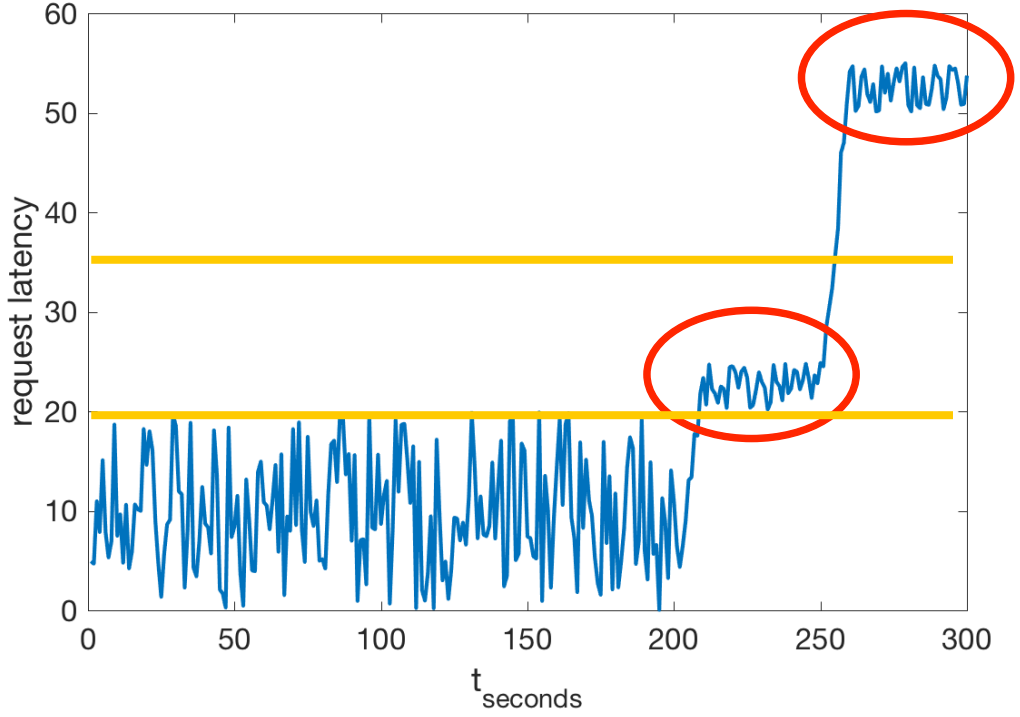
System Monitoring – Alerting - Examples

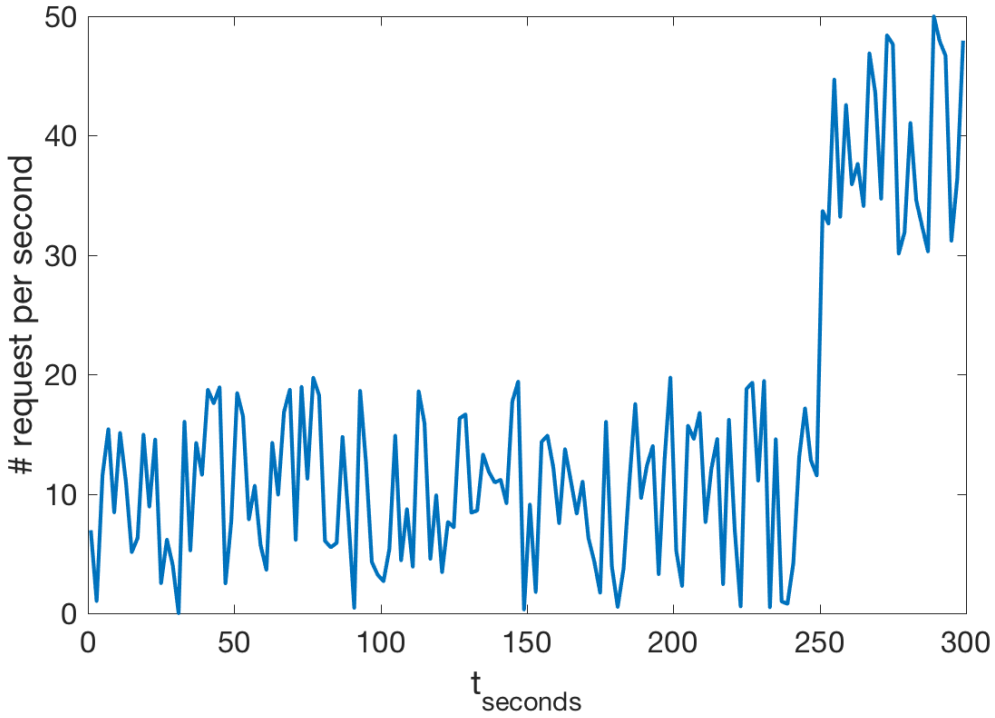




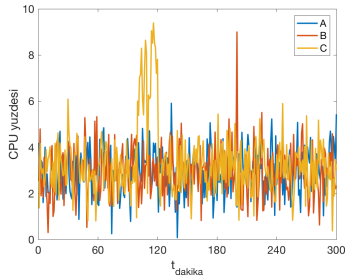
Expected behavior

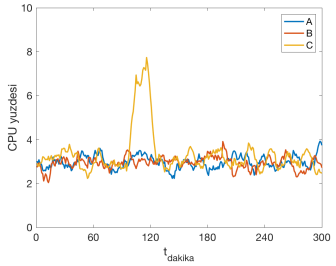
System Monitoring – Alerting - Examples

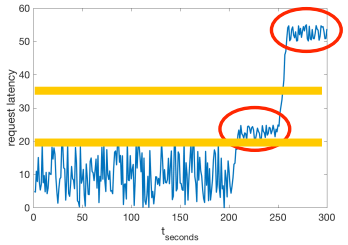


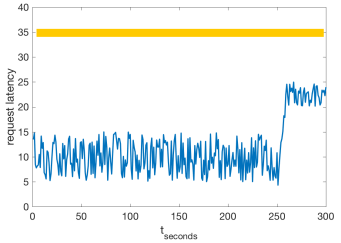


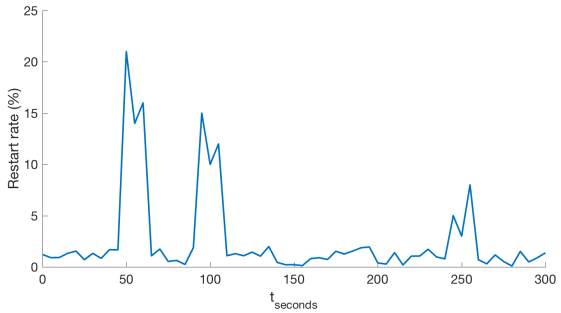
System Monitoring - Alerting

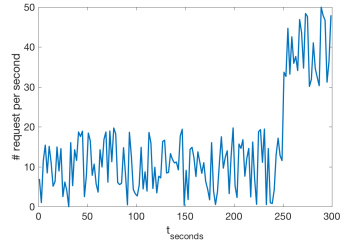


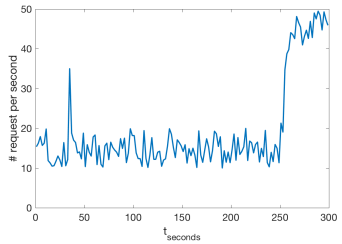






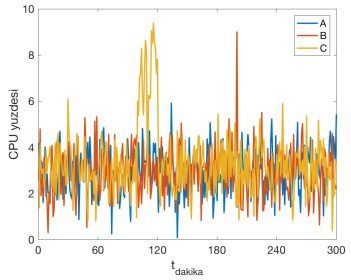


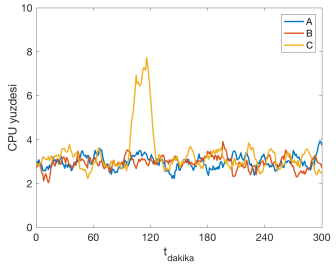


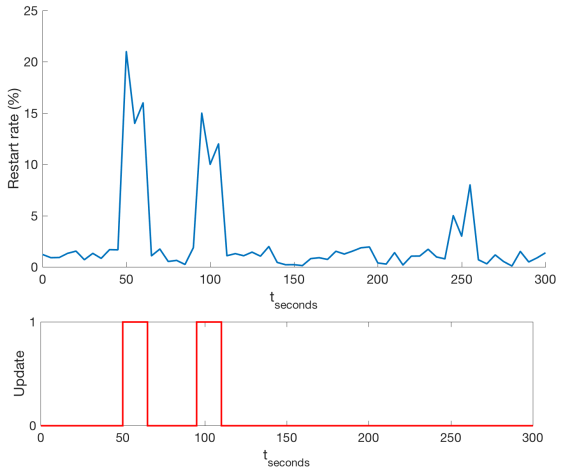


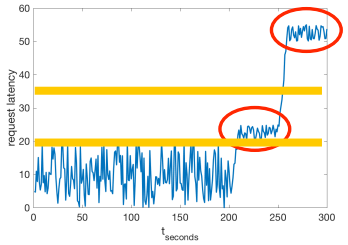
We need to fine tune the alerts for every server-metric,
We need to be more expressive,
We need to automate the process,
Iteratively improve, adapt the alerts in time.

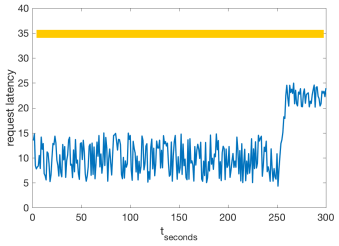
System Monitoring - Alerting

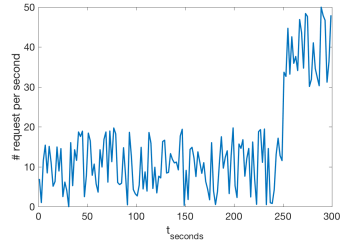


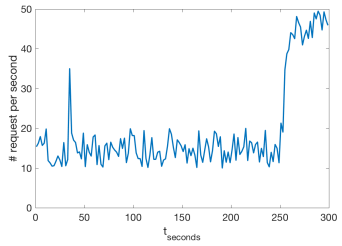










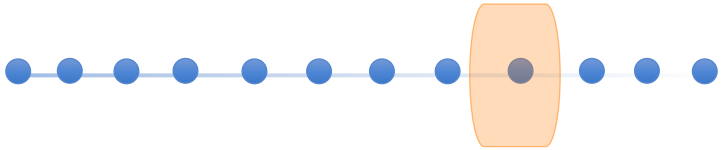


Express the alerts in a formal language
Find formulas (alerts) in an automated way,

We need to fine tune the alerts for every server-metric,
We need to be more expressive,
We need to automate the process,
Iteratively improve, adapt the alerts in time.

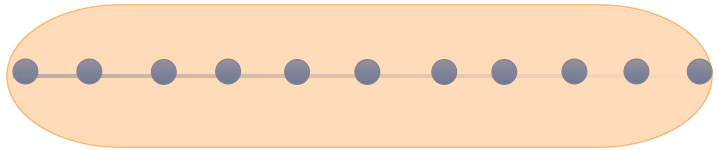
Linear Temporal Logic

Eventually ***a***



F a

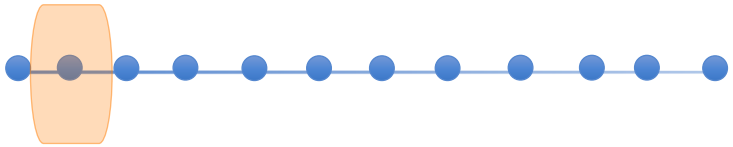
Globally ***a***



G a

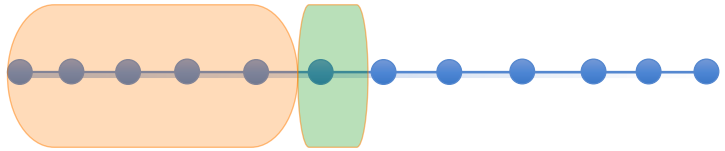
G (r → F g)

Next ***a***



X a

a Until ***b***

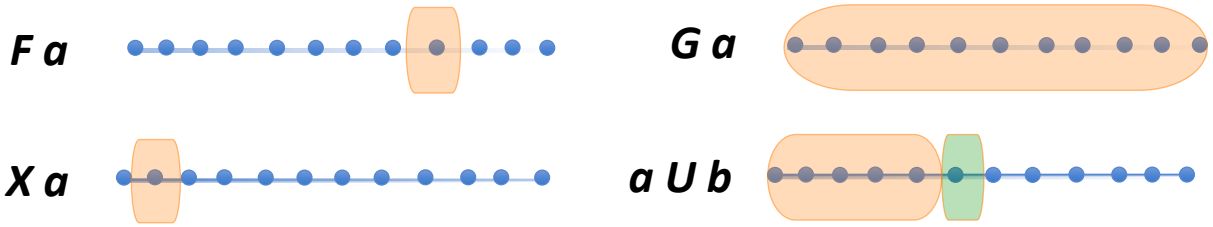


a U b

ϕ=**T**|**F**| ***F***ϕ | **G** ϕ | ϕ **U** ϕ | **X** ϕ | $\phi\wedge\phi$| $\phi\vee\phi$| $\neg\phi$|***p***

Boolean operators: ∧, ∨, ¬
Temporal operators: **G, U, X**

Linear Temporal Logic

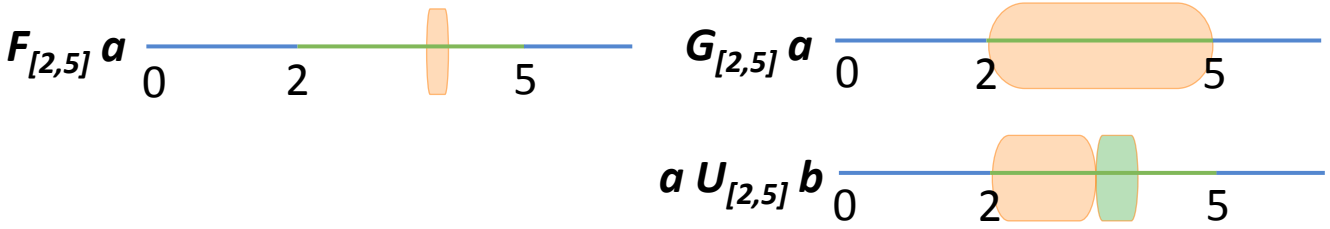


$\phi = \mathbf{T} | \mathbf{F} | F\phi \mid \mathbf{G}\ \phi \mid \phi\ \mathbf{U}\ \phi \mid \mathbf{X}\ \phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid p$

Primarily used in **model checking**
Metric extensions preferred for **monitoring**

G (r → F g)

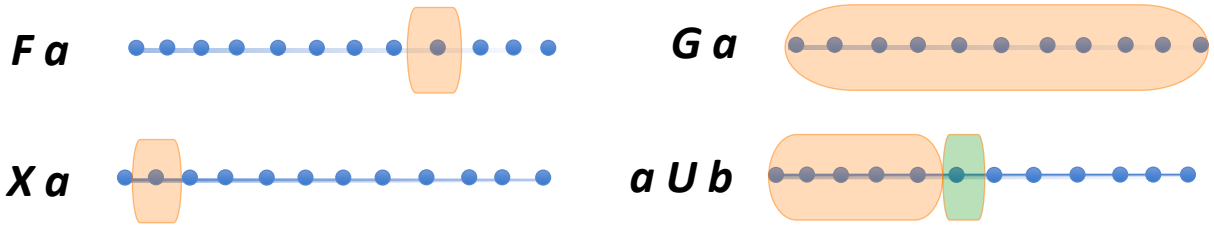
Metric Temporal Logic



Real time, boolean predicates

$\mathbf{G_{[0,t]}\ (r \rightarrow F_{[0,c]}\ g\ )}$

Linear Temporal Logic

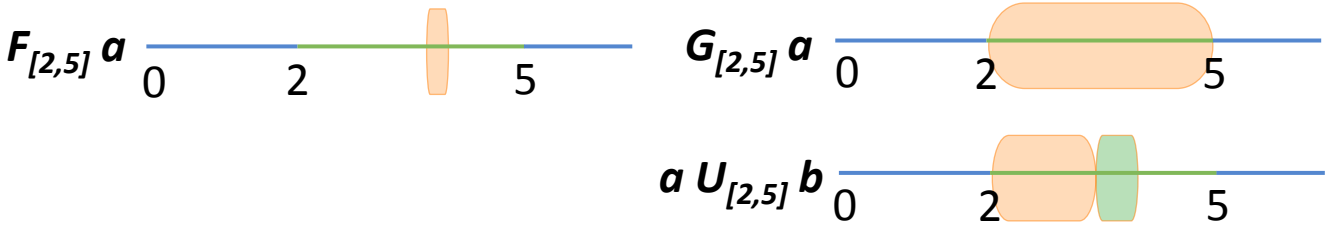


Primarily used in **model checking**
Metric extensions preferred for **monitoring**

$\phi = \mathbf{T} | \mathbf{F} | F\phi \mid \mathbf{G}\ \phi \mid \phi\ \mathbf{U}\ \phi \mid \mathbf{X}\ \phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi | p$

$\mathbf{G}\ (r \rightarrow \mathbf{F}\ g)$

Metric Temporal Logic



Real time, boolean predicates

$\mathbf{G}_{[0,t]}\ (r \rightarrow \mathbf{F}_{[0,c]}\ g)$

Signal Temporal Logic

Real time, real valued predicates, $x > T$

$\mathbf{G}_{[0,T]}\ (r[t] > 0 \rightarrow \mathbf{F}_{[0,D]}\ g[t] > 0)$

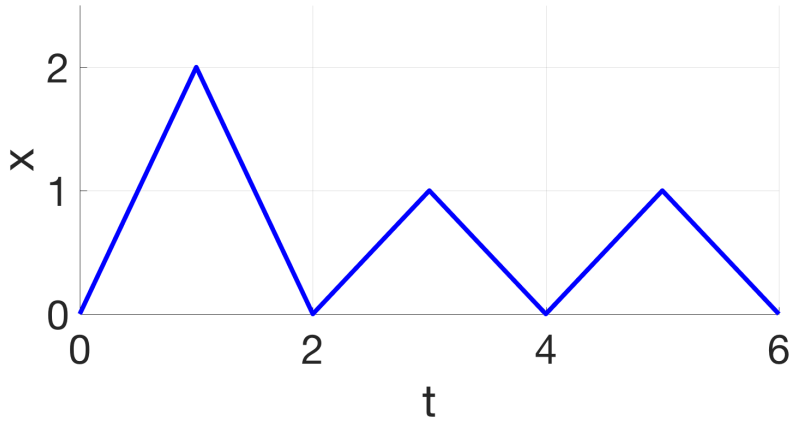
Signal Temporal Logic

- Predicates are over real values (signals), real time
- Allows for quantitative semantics

$$\Phi = T \mid \Phi \mid U_{[l,u]}\Phi \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mu$$

$$F_{[l,u]}\Phi = TU_{[l,u]}\Phi$$

$$G_{[l,u]}\Phi = \neg(F_{[l,u]}\neg\Phi)$$



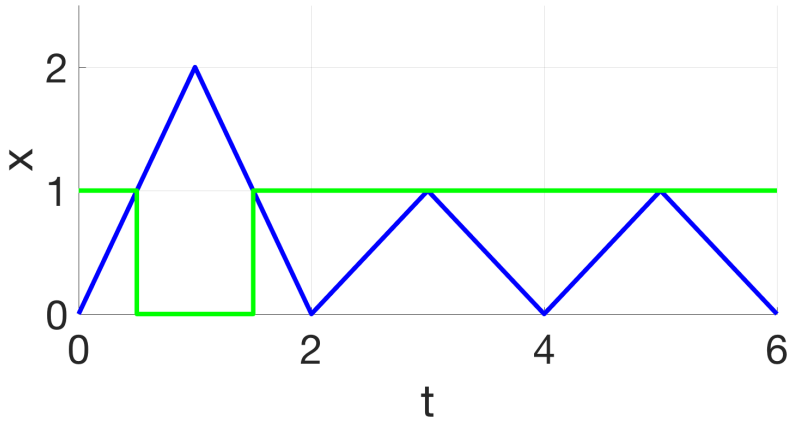
Signal Temporal Logic

$$\Phi = T \mid \Phi \mid U_{[l,u]}\Phi \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mu$$

$$F_{[l,u]}\Phi = TU_{[l,u]}\Phi$$

$$G_{[l,u]}\Phi = \neg(F_{[l,u]}\neg\Phi)$$

- Predicates are over real values (signals), real time
- Allows for quantitative semantics



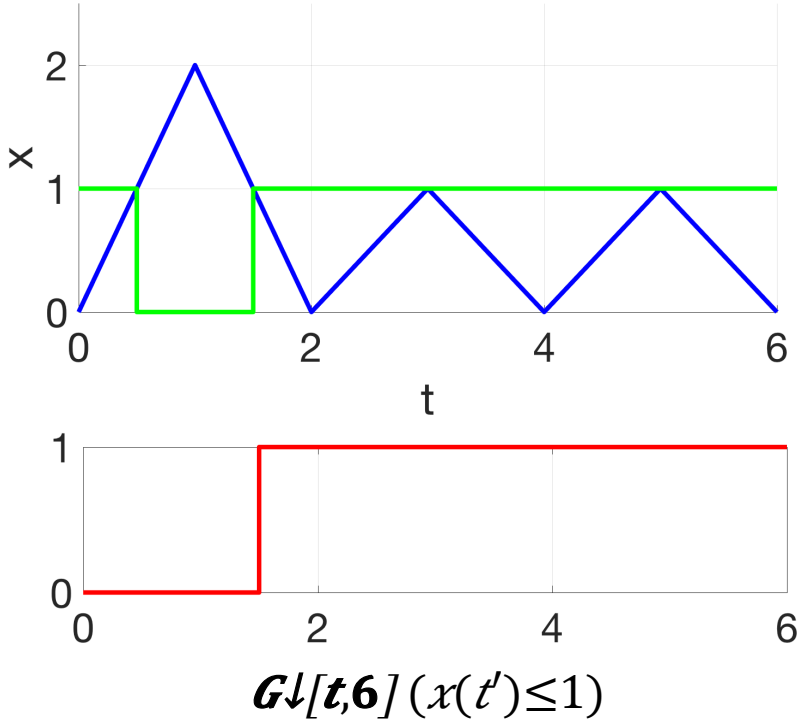
$x(t) \leq 1$

Signal Temporal Logic

$$\Phi = T \mid \Phi \mid U_{[l,u]}\Phi \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mu$$

$$F_{[l,u]}\Phi = TU_{[l,u]}\Phi$$

$$G_{[l,u]}\Phi = \neg(F_{[l,u]}\neg\Phi)$$



$x(t) \leq 1$

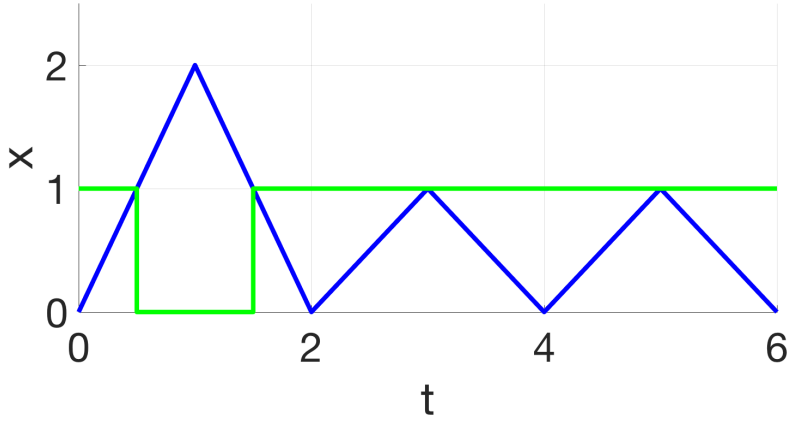
$G_{\downarrow[t,6]}(x(t') \leq 1)$

Signal Temporal Logic

$$\Phi = T \mid \Phi \mid U_{[l,u]}\Phi \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mu$$

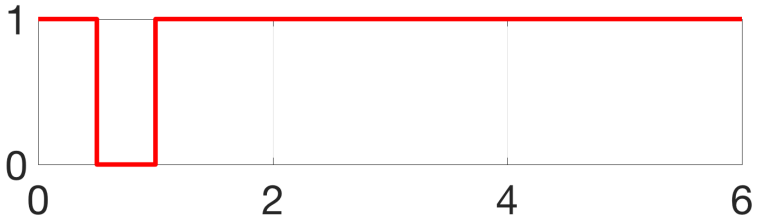
$$F_{[l,u]}\Phi = TU_{[l,u]}\Phi$$

$$G_{[l,u]}\Phi = \neg(F_{[l,u]}\neg\Phi)$$



$x(t) \leq 1$

$G_{[t,6]}(x(t') \leq 1)$



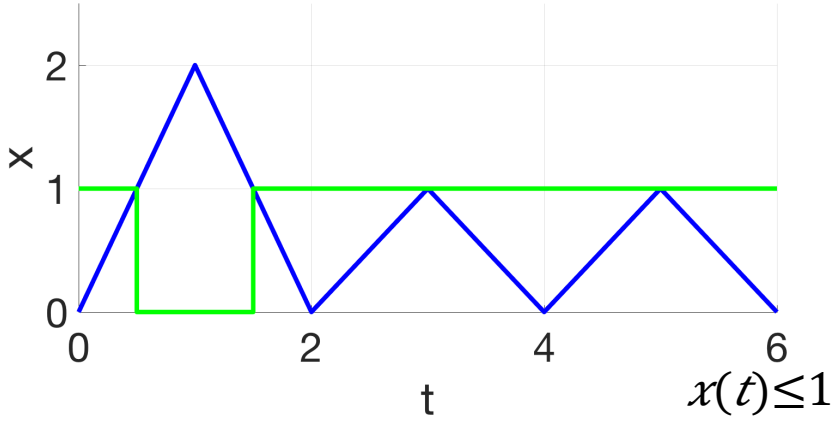
$F_{[t,t+0.5]}(x(t') \leq 1)$

Signal Temporal Logic

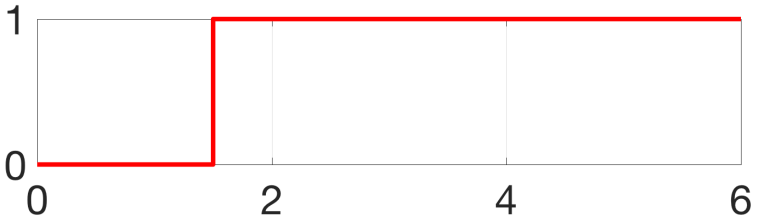
$$\Phi = T \mid \Phi \mid U_{[l,u]}\Phi \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mu$$

$$F_{[l,u]}\Phi = TU_{[l,u]}\Phi$$

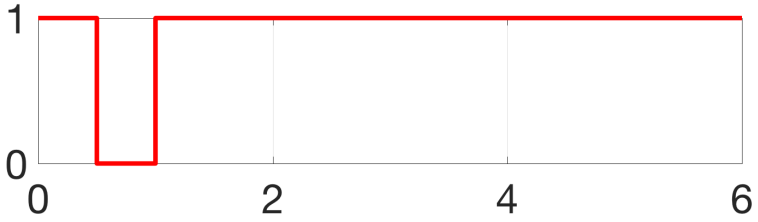
$$G_{[l,u]}\Phi = \neg(F_{[l,u]}\neg\Phi)$$



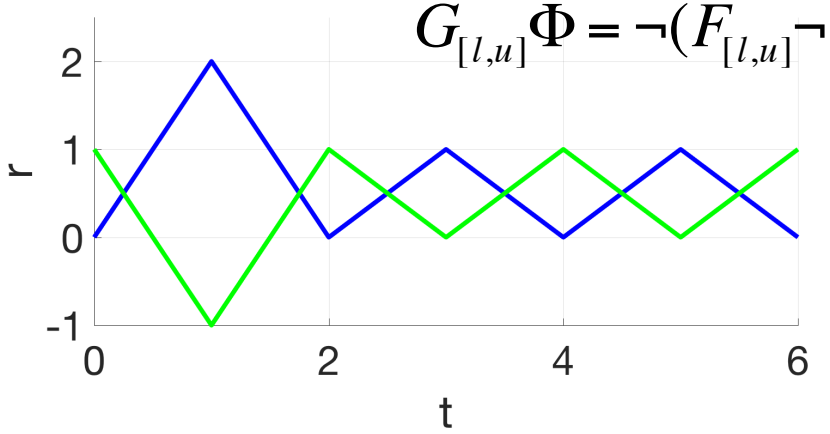
$x(t) \leq 1$



$G_{[t,6]}(x(t') \leq 1)$



$F_{[t,t+0.5]}(x(t') \leq 1)$



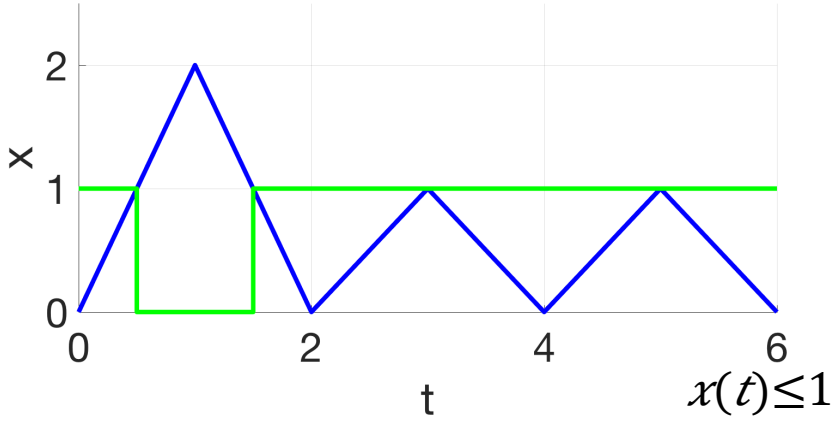
$1-x(t)$

Signal Temporal Logic

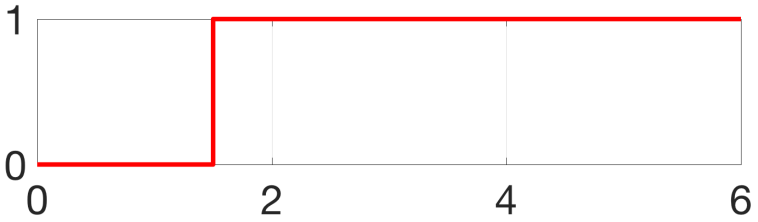
$$\Phi = T \mid \Phi \mid U_{[l,u]}\Phi \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mu$$

$$F_{[l,u]}\Phi = TU_{[l,u]}\Phi$$

$$G_{[l,u]}\Phi = \neg(F_{[l,u]}\neg\Phi)$$

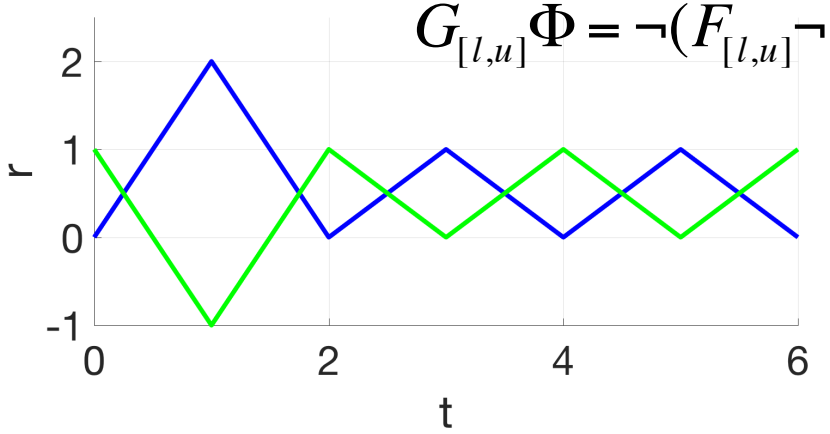


$x(t) \leq 1$

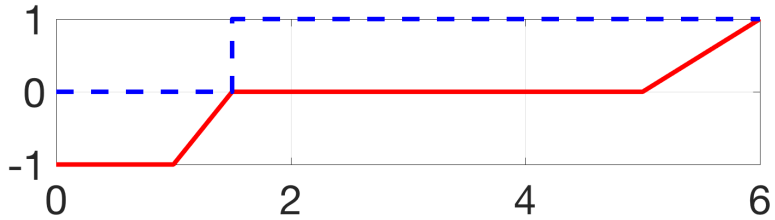


$G_{[t,6]}(x(t') \leq 1)$

$F_{[t,t+0.5]}(x(t') \leq 1)$



$1-x(t)$



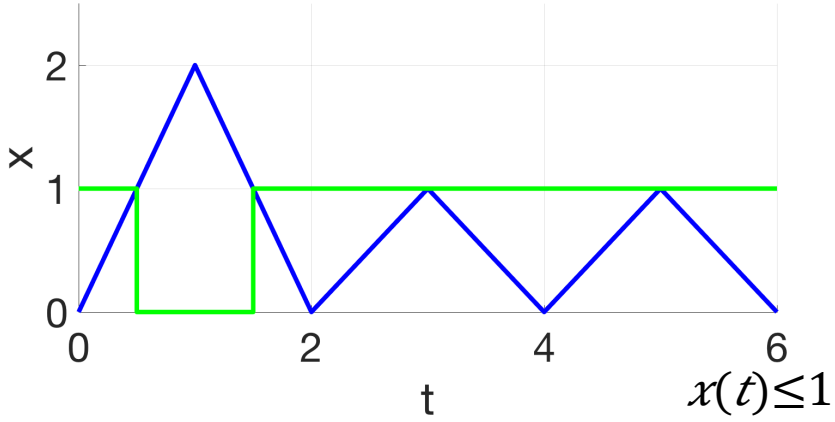
$\min_{t' \in [t,6]}(1-x(t'))$

Signal Temporal Logic

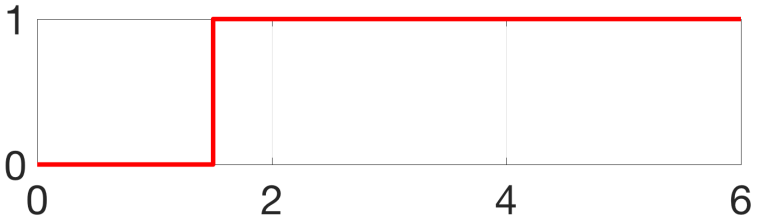
$$\Phi = T \mid \Phi \mid U_{[l,u]}\Phi \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mu$$

$$F_{[l,u]}\Phi = TU_{[l,u]}\Phi$$

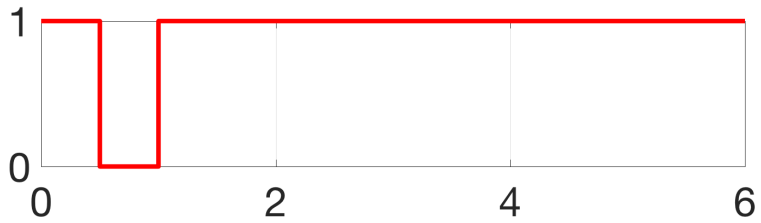
$$G_{[l,u]}\Phi = \neg(F_{[l,u]}\neg\Phi)$$



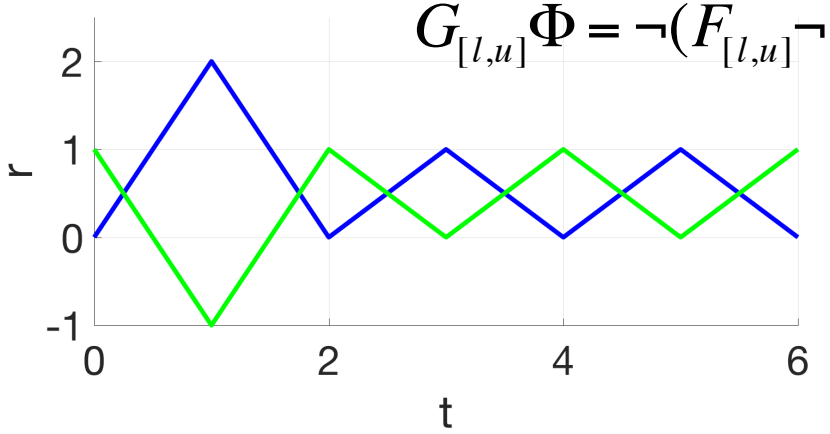
$x(t) \leq 1$



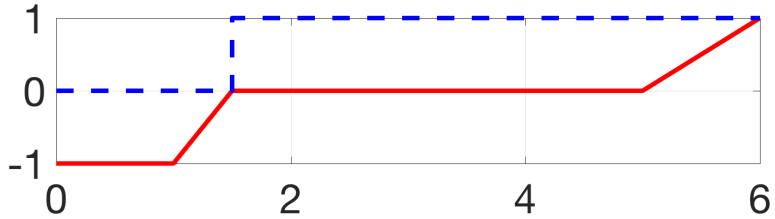
$G_{[t,6]}(x(t') \leq 1)$



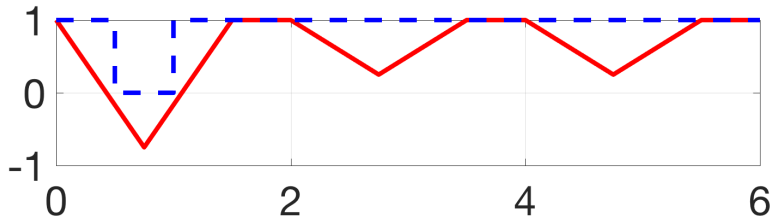
$F_{[t,t+0.5]}(x(t') \leq 1)$



$1-x(t)$



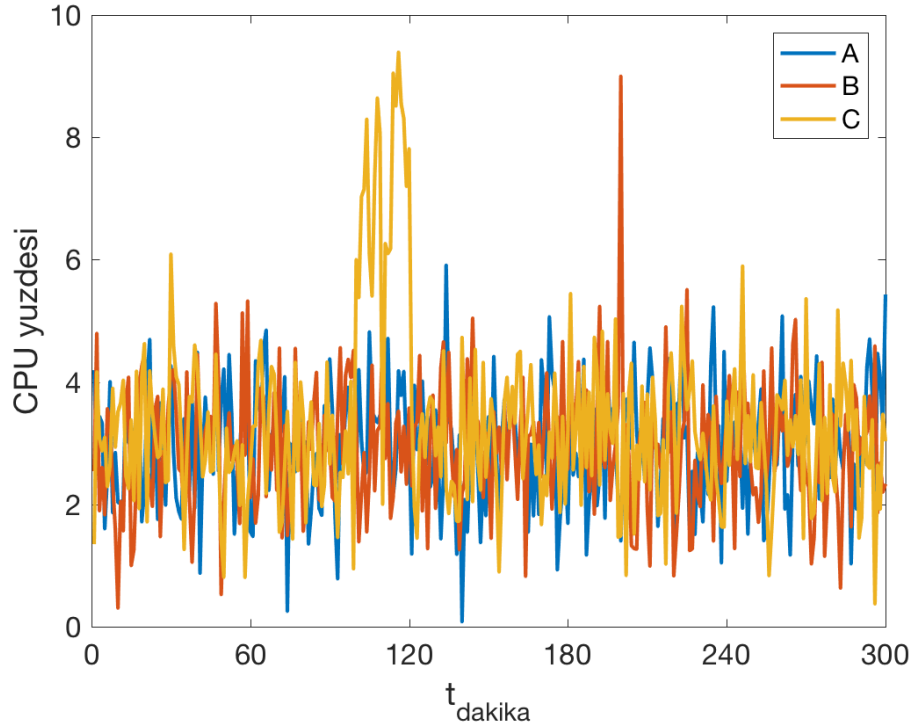
$\min_{t' \in [t,6]}(1-x(t'))$



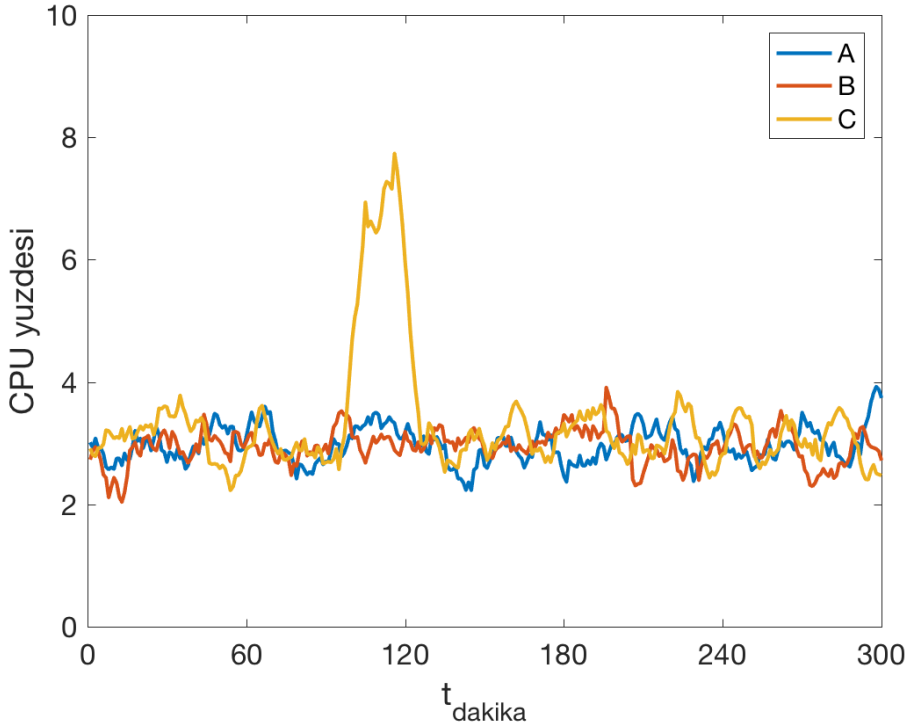
$\max_{t' \in [t,t+0.5]}(1-x(t'))$

Signal Temporal Logic

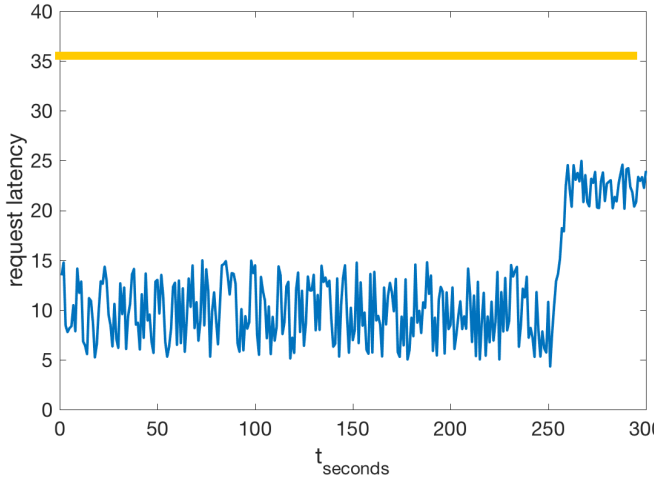
$$G_{[0,SL]}(x(t) > 6 \rightarrow (F_{[0,10]} G_{[0,20]} x(t) < 6)$$

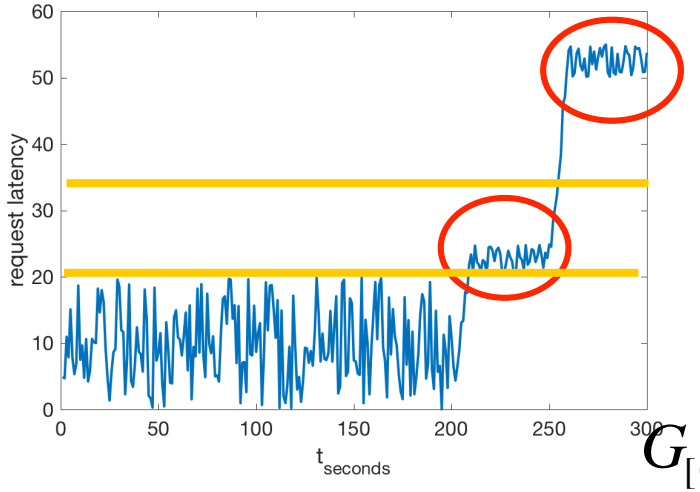


$$G_{[0,SL]}(\neg G_{[0,15]} x(t) > 6)$$



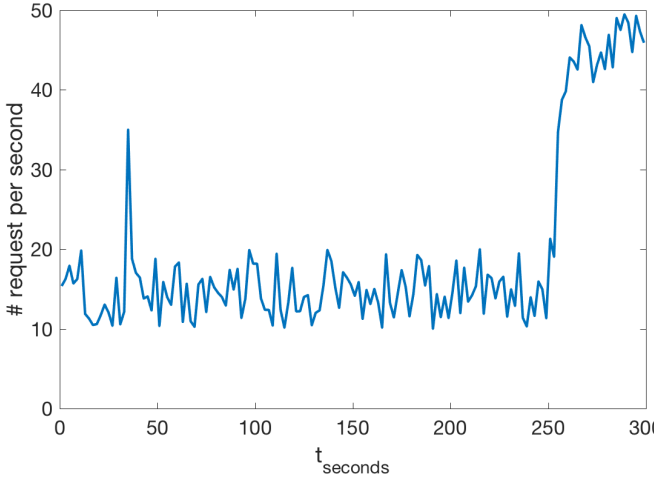
System Monitoring – Alerting - Examples

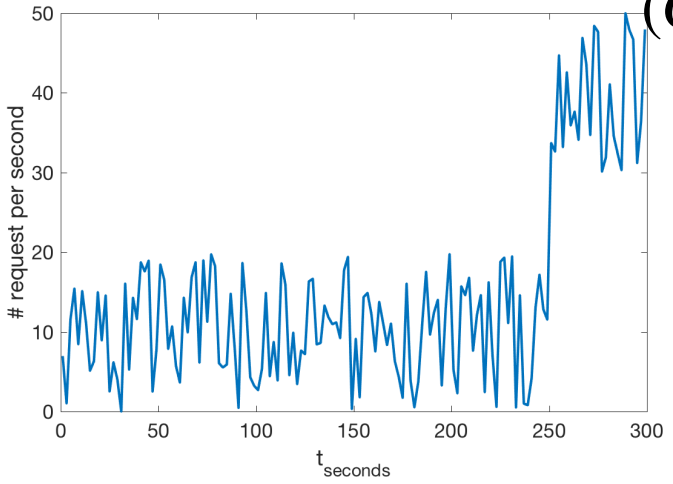




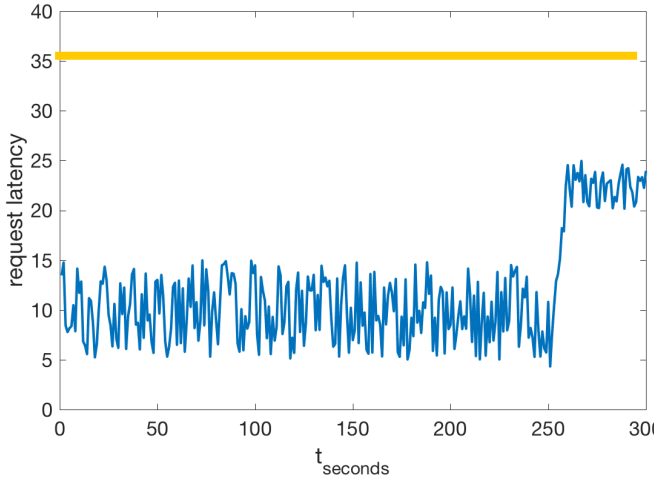
$$G_{[0,SL]}(lat(t) < 35 \wedge$$

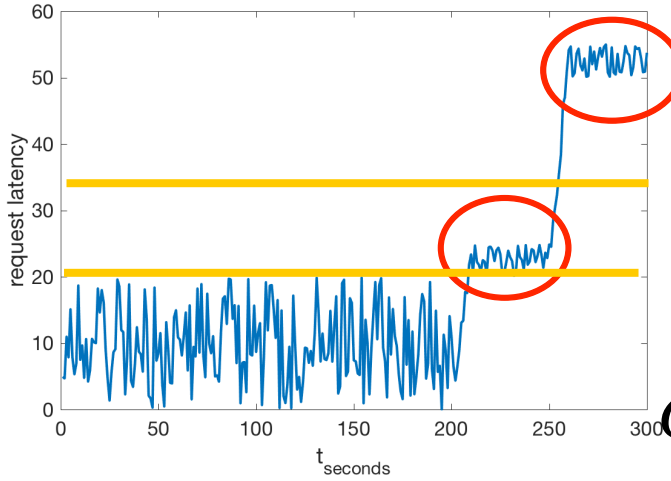
$$(G_{[0,10]}(rate(t) < 30) \rightarrow (F_{[0,10]}G_{[0,20]}lat(t) < 20)))$$

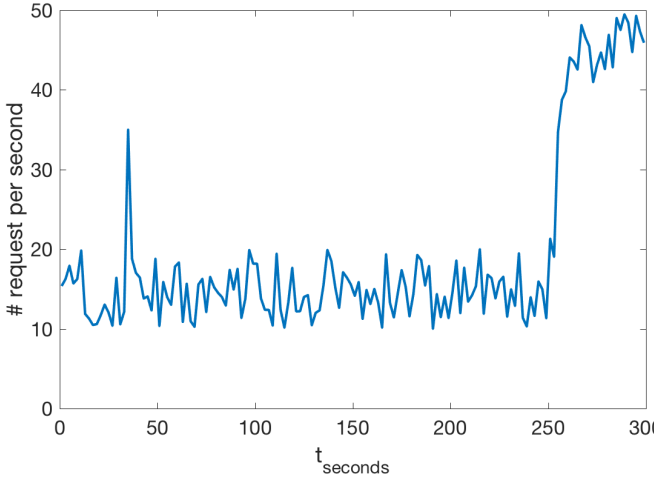


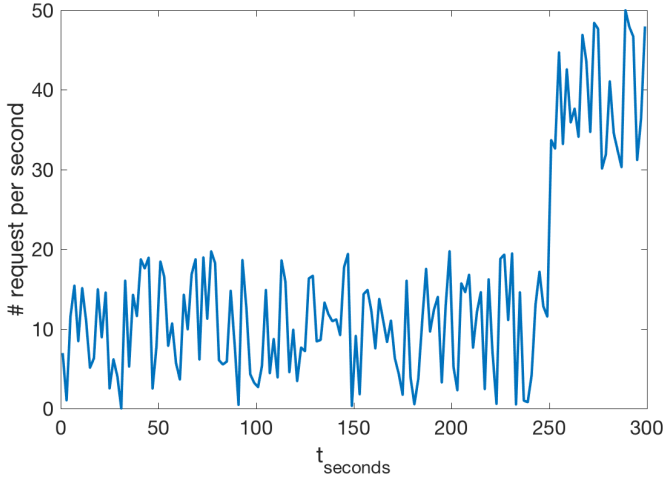


System Monitoring – Alerting - Examples







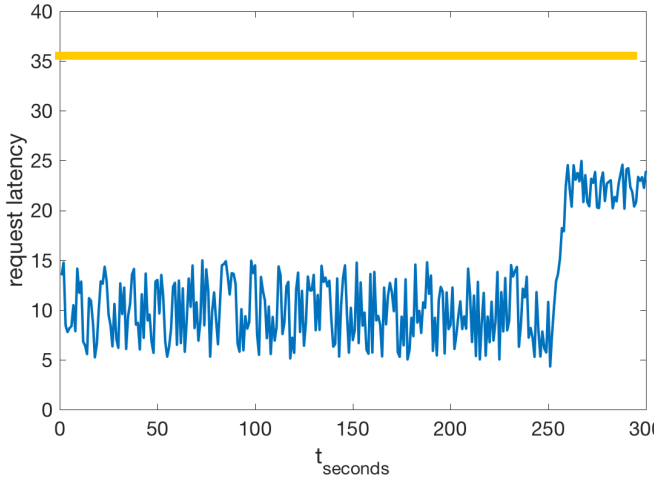


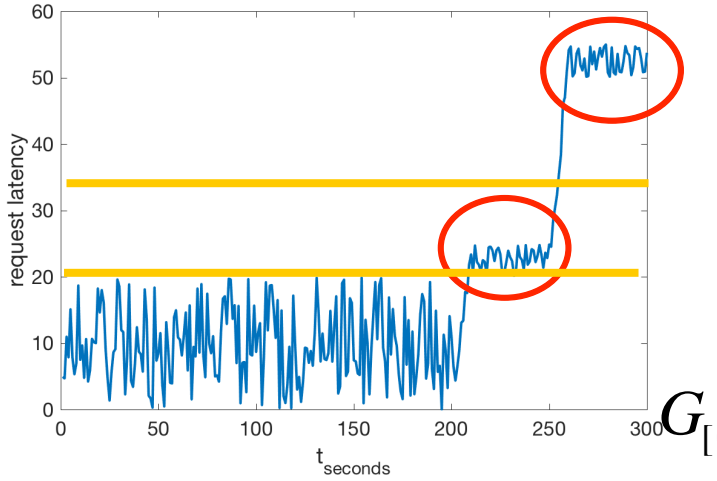
$$G_{[0,SL]}(lat(t) < S_1 \wedge$$

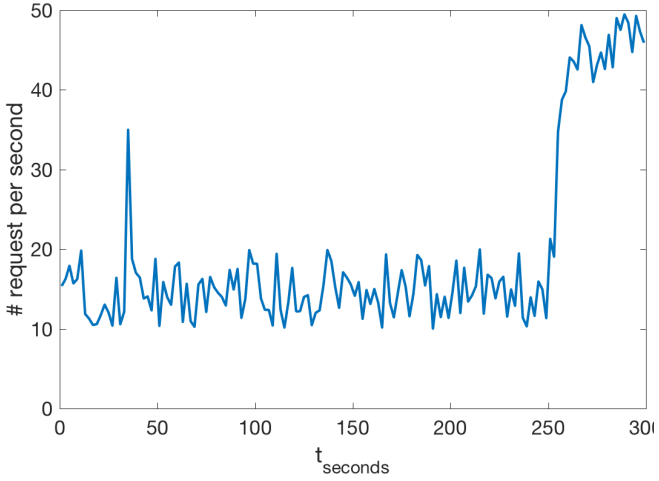
$$(G_{[0,T_1]}(rate(t) < S_2) \rightarrow (F_{[0,T_2]}G_{[0,T_3]}lat(t) < S_3)))$$

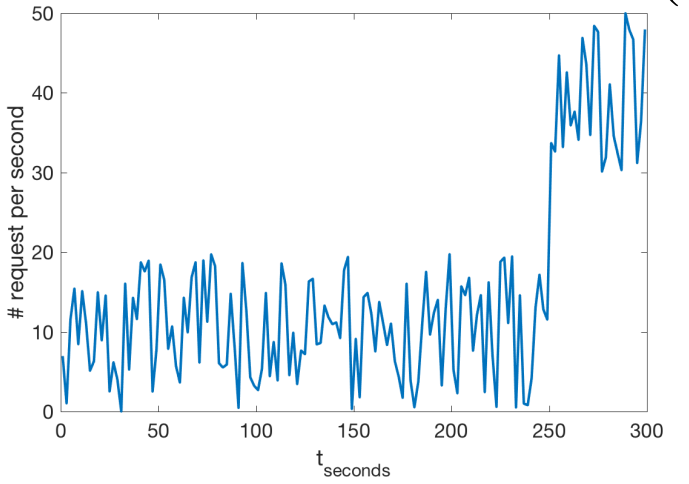
Parameters: S_1, S_2, S_3, T_1, T_2, T_3

System Monitoring – Alerting - Examples









How to find “optimal” formula?
How to perform online monitoring with STL?

$$G_{[0,SL]}(lat(t) < S_1 \wedge (G_{[0,T_1]}(rate(t) < S_2) \rightarrow (F_{[0,T_2]}G_{[0,T_3]}lat(t) < S_3)))$$

Parameters: S_1, S_2, S_3, T_1, T_2, T_3

STL for Monitoring

- How to find the “optimal” formula?
  - Template and parameters
- How to perform online monitoring with STL?
  - Detect the violation as soon as it occurs.
  - Perform the computation with limited resources.

STL for Monitoring

- How to find the “optimal” formula?
  - Template and parameters

Jin, X., Donzé, A., Deshmukh, J. V., Seshia, S.A. 2013. “**Mining requirements from closed-loop control models**”. Proceedings of the International Conference on Hybrid Systems: Computation and Control. 43-52

Bartocci, E., Bortolussi, L., Sanguinetti, G. 2014. “**Data-driven Statistical Learning of Temporal Logic Properties**”. Formal Modeling and Analysis of Timed Systems. Edidörler: Legay A., Bozga M. Springer International Publishing.

Kong Z., Jones A., Ayala, A.A., Aydin Gol, E., Belta, C. 2014. “**Temporal Logic Inference for Classification and Prediction from Data**”. Proceedings of the International Conference on Hybrid Systems: Computation and Control. 273-282**.**

- How to perform online monitoring with STL?
  - Detect the violation as soon as it occurs.
  - Perform the computation with limited resources.

Eisner, C., Fisman, D., Havlicek, D., Lustig Y., McIsaac, A., Van Campenhout, D. 2003. “**Reasoning with Temporal Logic on Truncated Paths**”. Computer Aided Verification. : A. H. Warren, F. Somenzi. Springer International Publishing.

Dokhanchi, A., Hoxha B., Fainekos, G. 2014. “**On-line Monitoring for Temporal Logic Robustness**”. Runtime Verification. Bonakdarpour B., Smolka S.A. Springer, Cham

STL for Monitoring

- How to find the "optimal" formula?
  - Template and parameters

REQUIREMENT MINING

Jin, X., Donzé, A., Deshmukh, J. V., Seshia, S.A. 2013. "**Mining requirements from closed-loop control models**". Proceedings of the International Conference on Hybrid Systems: Computation and Control. 43-52

Bartocci, E., Bortolussi, L., Sanguinetti, G. 2014. "**Data-driven Statistical Learning of Temporal Logic Properties**". Formal Modeling and Analysis of Timed Systems. Edidörler: Legay A., Bozga M. Springer International Publishing.

Kong Z., Jones A., Ayala, A.A., Aydin Gol, E., Belta, C. 2014. "**Temporal Logic Inference for Classification and Prediction from Data**". Proceedings of the International Conference on Hybrid Systems: Computation and Control. 273-282**.**

- How to perform online monitoring with STL?
  - Detect the violation as soon as it occurs.
  - Perform the computation with limited resources.

Eisner, C., Fisman, D., Havlicek, D., Lustig Y., McIsaac, A., Van Campenhout, D. 2003. "**Reasoning with Temporal Logic on Truncated Paths**". Computer Aided Verification. : A. H. Warren, F. Somenzi. Springer International Publishing.

Dokhanchi, A., Hoxha B., Fainekos, G. 2014. "**On-line Monitoring for Temporal Logic Robustness**". Runtime Verification. Bonakdarpour B., Smolka S.A. Springer, Cham

Parametric STL and parameter synthesis problem

- Given a set of signals, and a parametric STL formula, find **parameter valuations** such that all the signals satisfy the formula.

$$G_{[0,SL]}(lat(t) < S_1 \wedge$$
$$(G_{[0,T_1]}(rate(t) < S_2) \rightarrow (F_{[0,T_2]}G_{[0,T_3]}lat(t) < S_3)))$$

Parametric STL and parameter synthesis problem

- Given a set of signals, and a parametric STL formula, find **parameter valuations** such that all the signals satisfy the formula.

- Given a set of signals, and a parametric STL formula, find **tightest parameter valuations** such that all the signals satisfy the formula.

$$G_{[0,SL]}(lat(t) < S_1 \wedge$$
$$(G_{[0,T_1]}(rate(t) < S_2) \rightarrow (F_{[0,T_2]}G_{[0,T_3]}lat(t) < S_3)))$$

Parametric STL and parameter synthesis problem

- Given a set of signals, and a parametric STL formula, find **parameter valuations** such that all the signals satisfy the formula.

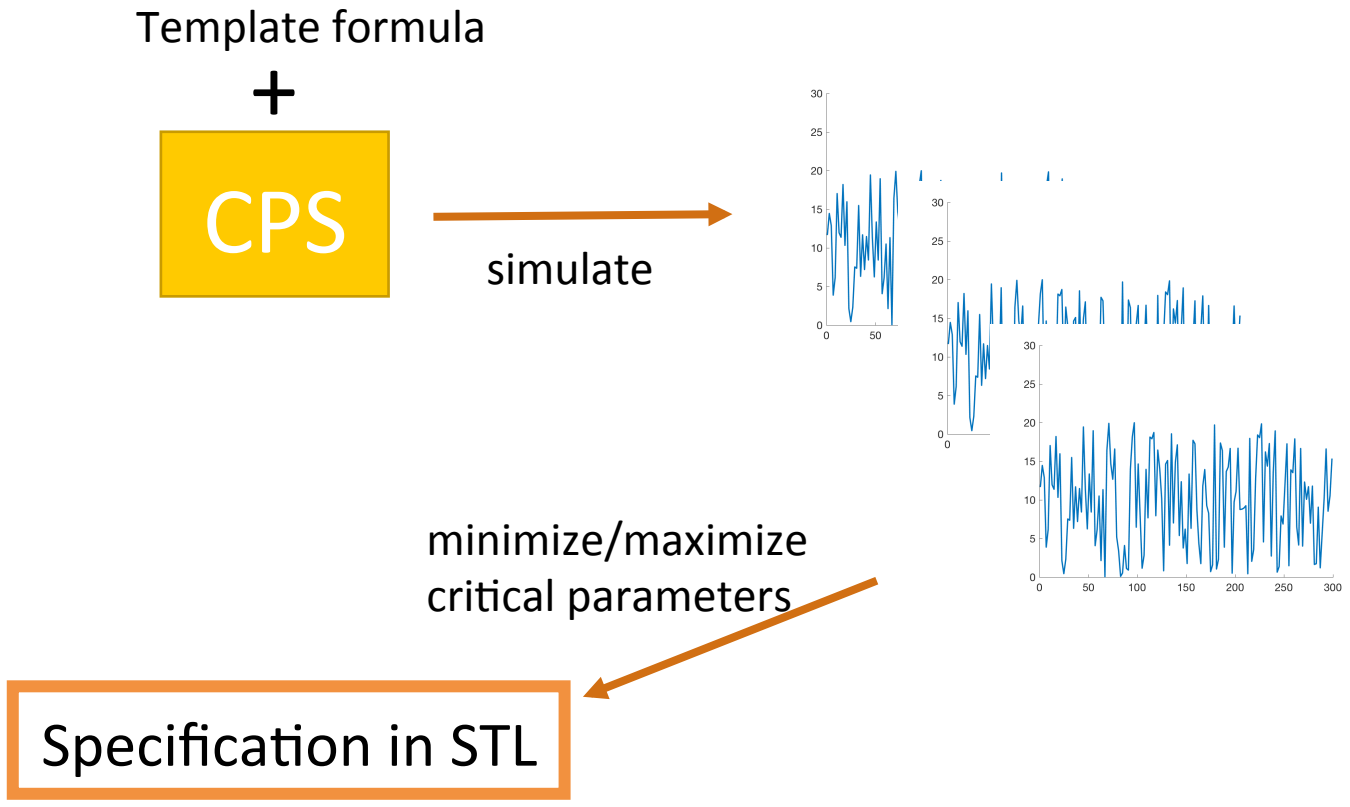
- Given a set of signals, and a parametric STL formula, find **tightest parameter valuations** such that all the signals satisfy the formula.

- Given a set of **positive signals and a set of negative signals**, and a parametric STL formula, find tightest parameter valuations such that all the positive signals satisfy the formula and all the negative signals violate the formula.

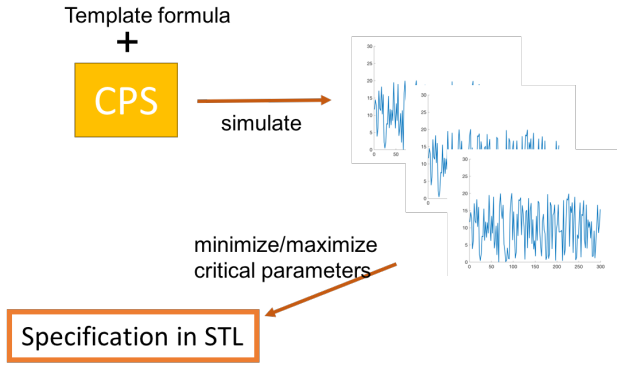
$$G_{[0,SL]}(lat(t) < S_1 \wedge$$
$$(G_{[0,T_1]}(rate(t) < S_2) \rightarrow (F_{[0,T_2]}G_{[0,T_3]}lat(t) < S_3)))$$

Requirements mining

- An application for parameter synthesis problem



Requirements mining



What is the maximum speed that the vehicle can reach?
What is the minimum dwell time in a given var?

Mining Requirements from Closed-Loop Control Models

Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V. Deshmukh, Sanjit A. Seshia

***Abstract*—Formal verification of a control system can be performed by checking if a model of its dynamical behavior conforms to temporal requirements. Unfortunately, adoption of formal verification in an industrial setting is a formidable challenge as design requirements are often vague, non-modular, evolving, or sometimes simply unknown. We propose a framework to mine requirements from a closed-loop model of an industrial-scale control system, such as one specified in Simulink. The input to our algorithm is a *requirement template* expressed in Parametric Signal Temporal Logic: a logical formula in which concrete signal or time values are replaced with parameters. Given a set of *simulation traces* of the model, our method infers values for the template parameters to obtain the *strongest candidate requirement* satisfied by the traces. It then tries to falsify the candidate requirement using a falsification tool. If a counterexample is found, it is added to the existing set of traces and these steps are repeated; otherwise, it terminates with the synthesized requirement. Requirement mining has several usage scenarios: mined requirements can be used to formally validate future modifications of the model, they can be used to gain better understanding of legacy models or code, and can also help enhancing the process of bug-finding through simulations. We demonstrate the scalability and utility of our technique on three complex case studies in the domain of automotive powertrain systems: a simple automatic transmission controller, an air-fuel controller with a mean-value model of the engine dynamics, and an industrial-size prototype airpath controller for a diesel engine. We include results on a bug found in the prototype controller by our method.**

***Index Terms*—Model-based design; Parametric Temporal Logics; Simulink; software engineering and verification**

I. Introduction

Industrial-scale controllers used in automobiles and avionics are now commonly developed using a model-based development (MBD) paradigm [36], [42]. The MBD process consists of a sequence of steps. In the first step, the designer captures the *plant model*, i.e., the dynamical characteristics of the physical parts of the system using differential, logic, and algebraic equations. Examples of plant models include the rotational dynamics model of the camshaft in an automobile engine, the thermodynamic model of an internal combustion engine, and atmospheric turbulence models. The next step is to design a *controller* that employs some specific control law to regulate the behavior of the physical system. The *closed-loop model* consists of the composition of the plant and the controller.

In the next step, the designer may perform extensive *simulations* of the closed-loop model. The objective is to analyze the controller design by observing the time-varying behavior of the signals of interest by exciting the exogenous, time-varying inputs of the closed-loop model. An important aspect of this step is *validation*, i.e. checking if the time-varying behavior of the closed-loop system matches a set of *requirements*. Unfortunately, in practice, these requirements are high-level and often vague. Examples of requirements the authors have encountered in the automotive industry include "better fuel-efficiency", "signal should eventually settle", and "resistance to turbulence". If the simulation behavior is deemed unsatisfactory, then the designer refines or tunes the controller design and repeats the validation step.

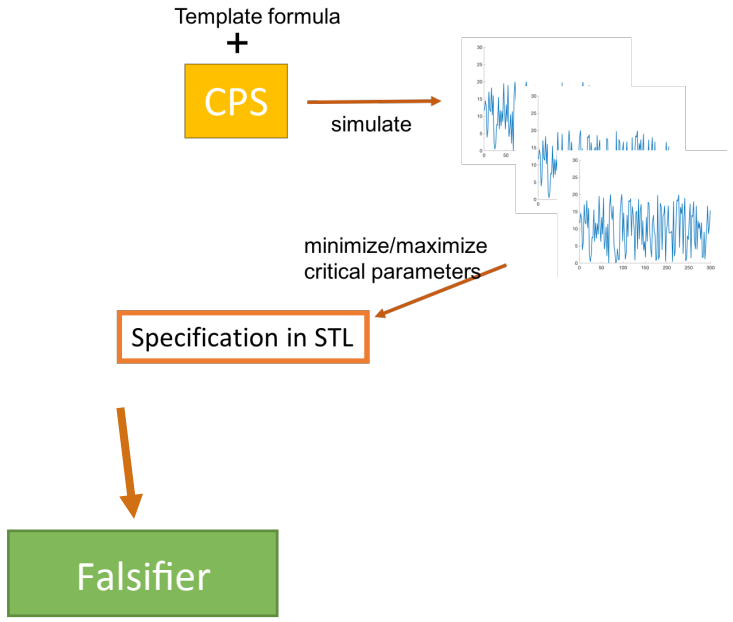
In the formal methods literature, a requirement (also called a *specification*) is a mathematical expression of the design goals or desirable design properties in a suitable logic. In an industrial setting, many companies have made a strenuous effort to document clear and concise requirements. However, for systems built on legacy models or legacy code, requirements are normally not available. Moreover, to date, formal validation tools have been unable to digest the format or scale of industrial-scale requirements and models. As a result, widespread adoption of formal tools has been restricted to testing syntactic coverage of the controller code, which is an open-loop system without the important behavior of the physical system, with the hope that higher coverage implies better chances of finding bugs.

In this paper, we propose a scalable technique to systematically mine requirements from the closed-loop model of an industrial-scale control system from observations of the system behavior. In addition to the closed-loop model, our technique takes as input a *template requirement*. The final output is a synthesized requirement matching the template. We assume that the model is specified in Simulink [41], an industry-wide standard that is able to: (1) express complex dynamics (differential and algebraic equations), (2) capture discrete state-machine behavior by allowing both Boolean and real-valued variables, (3) allow a layered design approach through modularity and hierarchical composition, and (4) perform high-fidelity simulations.

Formalisms such as Metric Temporal Logic (MTL) [2], [31], and later Parametric Signal Temporal Logic (PSTL) [9] have emerged as logics adept at capturing both the real-valued and time-varying behaviors of hybrid control systems. PSTL is particularly well-suited to expressing template requirements

X. Jin and J. Deshmukh are with Toyota Technical Center e-mail: {xiaoqing.jin,jyotirmoy.deshmukh}@tema.toyota.com.
A. Donzé and S. A. Seshia are with the University of California, Berkeley e-mail: {donze,seshia}@eecs.berkeley.edu.

Requirements mining



OK or
Violating trace

What is the maximum speed that the vehicle can reach?
What is the minimum dwell time in a given var?

Mining Requirements from Closed-Loop Control Models

Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V. Deshmukh, Sanjit A. Seshia

***Abstract*—Formal verification of a control system can be performed by checking if a model of its dynamical behavior conforms to temporal requirements. Unfortunately, adoption of formal verification in an industrial setting is a formidable challenge as design requirements are often vague, non-modular, evolving, or sometimes simply unknown. We propose a framework to mine requirements from a closed-loop model of an industrial-scale control system, such as one specified in Simulink. The input to our algorithm is a *requirement template* expressed in Parametric Signal Temporal Logic: a logical formula in which concrete signal or time values are replaced with parameters. Given a set of *simulation traces* of the model, our method infers values for the template parameters to obtain the *strongest candidate requirement* satisfied by the traces. It then tries to falsify the candidate requirement using a falsification tool. If a counterexample is found, it is added to the existing set of traces and these steps are repeated; otherwise, it terminates with the synthesized requirement. Requirement mining has several usage scenarios: mined requirements can be used to formally validate future modifications of the model, they can be used to gain better understanding of legacy models or code, and can also help enhancing the process of bug-finding through simulations. We demonstrate the scalability and utility of our technique on three complex case studies in the domain of automotive powertrain systems: a simple automatic transmission controller, an air-fuel controller with a mean-value model of the engine dynamics, and an industrial-size prototype airpath controller for a diesel engine. We include results on a bug found in the prototype controller by our method.**

***Index Terms*—Model-based design; Parametric Temporal Logics; Simulink; software engineering and verification**

I. Introduction

Industrial-scale controllers used in automobiles and avionics are now commonly developed using a model-based development (MBD) paradigm [36], [42]. The MBD process consists of a sequence of steps. In the first step, the designer captures the *plant model*, i.e., the dynamical characteristics of the physical parts of the system using differential, logic, and algebraic equations. Examples of plant models include the rotational dynamics model of the camshaft in an automobile engine, the thermodynamic model of an internal combustion engine, and atmospheric turbulence models. The next step is to design a *controller* that employs some specific control law to regulate the behavior of the physical system. The *closed-loop model* consists of the composition of the plant and the controller.

In the next step, the designer may perform extensive *simulations* of the closed-loop model. The objective is to analyze the controller design by observing the time-varying behavior of the signals of interest by exciting the exogenous, time-varying inputs of the closed-loop model. An important aspect of this step is *validation*, i.e. checking if the time-varying behavior of the closed-loop system matches a set of *requirements*. Unfortunately, in practice, these requirements are high-level and often vague. Examples of requirements the authors have encountered in the automotive industry include "better fuel-efficiency", "signal should eventually settle", and "resistance to turbulence". If the simulation behavior is deemed unsatisfactory, then the designer refines or tunes the controller design and repeats the validation step.

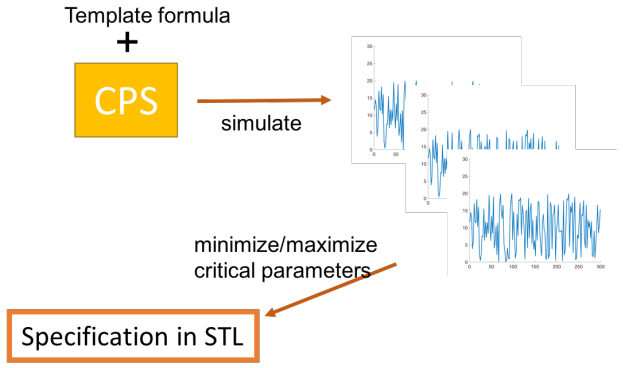
In the formal methods literature, a requirement (also called a *specification*) is a mathematical expression of the design goals or desirable design properties in a suitable logic. In an industrial setting, many companies have made a strenuous effort to document clear and concise requirements. However, for systems built on legacy models or legacy code, requirements are normally not available. Moreover, to date, formal validation tools have been unable to digest the format or scale of industrial-scale requirements and models. As a result, widespread adoption of formal tools has been restricted to testing syntactic coverage of the controller code, which is an open-loop system without the important behavior of the physical system, with the hope that higher coverage implies better chances of finding bugs.

In this paper, we propose a scalable technique to systematically mine requirements from the closed-loop model of an industrial-scale control system from observations of the system behavior. In addition to the closed-loop model, our technique takes as input a *template requirement*. The final output is a synthesized requirement matching the template. We assume that the model is specified in Simulink [41], an industry-wide standard that is able to: (1) express complex dynamics (differential and algebraic equations), (2) capture discrete state-machine behavior by allowing both Boolean and real-valued variables, (3) allow a layered design approach through modularity and hierarchical composition, and (4) perform high-fidelity simulations.

Formalisms such as Metric Temporal Logic (MTL) [2], [31], and later Parametric Signal Temporal Logic (PSTL) [9] have emerged as logics adept at capturing both the real-valued and time-varying behaviors of hybrid control systems. PSTL is particularly well-suited to expressing template requirements

X. Jin and J. Deshmukh are with Toyota Technical Center e-mail: {xiaoqing.jin,jyotirmoy.deshmukh}@tema.toyota.com.
A. Donzé and S. A. Seshia are with the University of California, Berkeley e-mail: {donze,seshia}@eecs.berkeley.edu.

Requirements mining



Define template set
Optimize parameters
Try to combine templates [5]

Data-Driven Statistical Learning of Temporal Logic Properties⋆

Ezio Bartocci[1], Luca Bortolussi[2,3], and Guido Sanguinetti[4,5]

[1] Faculty of Informatics, Vienna University of Technology, Austria
[2] DMG, University of Trieste, Italy
[3] CNR/ISTI, Pisa, Italy
[4] School of Informatics, University of Edinburgh, UK
SynthSys, Centre for Synthetic and Systems Biology, University of Edinburgh, UK

Abstract. We present a novel approach to learn logical formulae characterising the emergent behaviour of a dynamical system from system observations. At a high level, the approach starts by devising a data-driven statistical abstraction of the system. We then propose general optimisation strategies for selecting formulae with high satisfaction probability, either within a discrete set of formulae of bounded complexity, or a parametric family of formulae. We illustrate and apply the methodology on two real world case studies: characterising the dynamics of a biological circadian oscillator, and discriminating different types of cardiac malfunction from electro-cardiogram data. Our results demonstrate that this approach provides a statistically principled and generally usable tool to logically characterise dynamical systems in terms of temporal logic formulae.

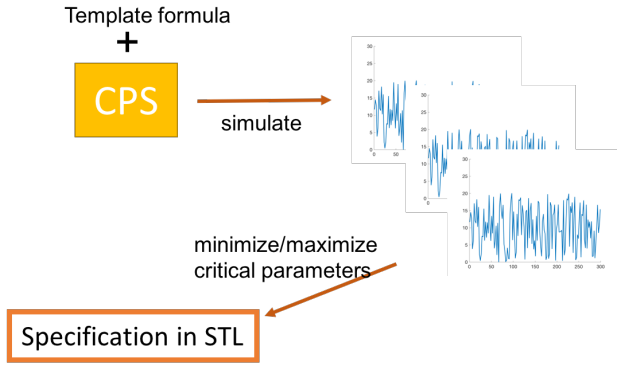
1 Introduction

Dynamical systems are among the most widely used modelling frameworks, with important applications in all domains of science and engineering. Much of the attraction of dynamical systems modelling lies in the availability of effective simulation tools, enabling predictive modelling, and in the possibility of encoding complex behaviours through the interaction of multiple, simple components. This leads naturally to the notion of *emergent properties*, i.e. properties of the system trajectories which are a non-trivial consequence of the local interaction rules of the system components. Emergent properties of deterministic dynamical systems can often be easily verified through simulations. Quantitatively identifying the emergent properties of a stochastic system, instead, is a much harder problem.

In the simplest scenario, one assumes that a mathematical model of the system of interest is already available (e.g. as a continuous time Markov chain, or a stochastic differential equation), generally thanks to the availability of domain expertise. This problem is often termed *mining requirements*: this is an active field of research, with many recent contributions extending its scalability and applicability [18,27]. This approach

⋆ L.B. acknowledges partial support from the EU-FET project QUANTICOL (nr. 600708) and by FRA-UniTS. G.S. acknowledges support from the ERC under grant MLCS306999. E.B. acknowledges the support of the Austrian FFG project HARMONIA (nr. 845631).

Requirements mining



Define template set
Optimize parameters
Try to combine templates

Data-Driven Statistical Learning of Temporal Logic Properties*

Ezio Bartocci[1], Luca Bortolussi[2,3], and Guido Sanguinetti[4,5]

[1] Faculty of Informatics, Vienna University of Technology, Austria
[2] DMG, University of Trieste, Italy
[3] CNR/ISTI, Pisa, Italy
[4] School of Informatics, University of Edinburgh, UK
[5] SynthSys, Centre for Synthetic and Systems Biology, University of Edinburgh, UK

Abstract. We present a novel approach to learn logical formulae characterising the emergent behaviour of a dynamical system from system observations. At a high level, the approach starts by devising a data-driven statistical abstraction of the system. We then propose general optimisation strategies for selecting formulae with high satisfaction probability, either within a discrete set of formulae of bounded complexity, or a parametric family of formulae. We illustrate and apply the methodology on two real world case studies: characterising the dynamics of a biological circadian oscillator, and discriminating different types of cardiac malfunction from electro-cardiogram data. Our results demonstrate that this approach provides a statistically principled and generally usable tool to logically characterise dynamical systems in terms of temporal logic formulae.

1 Introduction

Dynamical systems are among the most widely used modelling frameworks, with important applications in all domains of science and engineering. Much of the attraction of dynamical systems modelling lies in the availability of effective simulation tools, enabling predictive modelling, and in the possibility of encoding complex behaviours through the interaction of multiple, simple components. This leads naturally to the notion of *emergent properties*, i.e. properties of the system trajectories which are a non-trivial consequence of the local interaction rules of the system components. Emergent properties of deterministic dynamical systems can often be easily verified through simulations. Quantitatively identifying the emergent properties of a stochastic system, instead, is a much harder problem.

In the simplest scenario, one assumes that a mathematical model of the system of interest is already available (e.g. as a continuous time Markov chain, or a stochastic differential equation), generally thanks to the availability of domain expertise. This problem is often termed *mining requirements*: this is an active field of research, with many recent contributions extending its scalability and applicability [18,27]. This approach

* L.B. acknowledges partial support from the EU-FET project QUANTICOL (nr. 600708) and by FRA-UniTS. G.S. acknowledges support from the ERC under grant MLCS306999. E.B. acknowledges the support of the Austrian FFG project HARMONIA (nr. 845631).

Introduced a preorder for a subset of STL
Search on both parameter space and template space

Temporal Logic Inference for Classification and Prediction from Data

Zhaodan Kong
Boston University
110 Cummington Street
Boston, MA
zhaodan@bu.edu

Austin Jones
Boston University
15 Saint Mary's Street
Brookline, MA
austinmj@bu.edu

Ana Medina Ayala
Boston University
110 Cummington Street
Boston, MA
duvinci@bu.edu

Ebru Aydin Gol
Boston University
15 Saint Mary's Street
Brookline, MA
ebru@bu.edu

Calin Belta
Boston University
110 Cummington Street
Boston, MA
cbelta@bu.edu

ABSTRACT

This paper presents an inference algorithm that can discover temporal logic properties of a system from data. Our algo-

General Terms

Algorithms, Theory

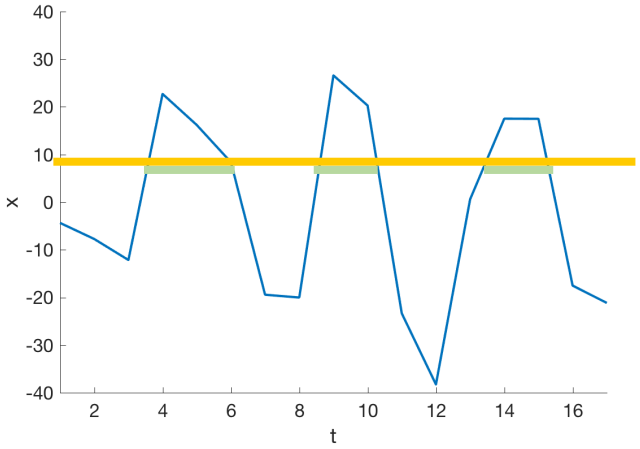
PSTL parameter synthesis

$$\Phi^{T,S} = G_{[0,SL]}(F_{[0,T]}x(t) < S)$$

$Q(\Phi^{T,S}, x)$:quantitative evaluation of **x**

Search over the parameter space.
Quantitative valuation plays a key role.

Find a feasible range for S, T
Define minimum robustness parameter
Use monotonicity properties



Multi-criteria optimization problem

Find T,S such that:

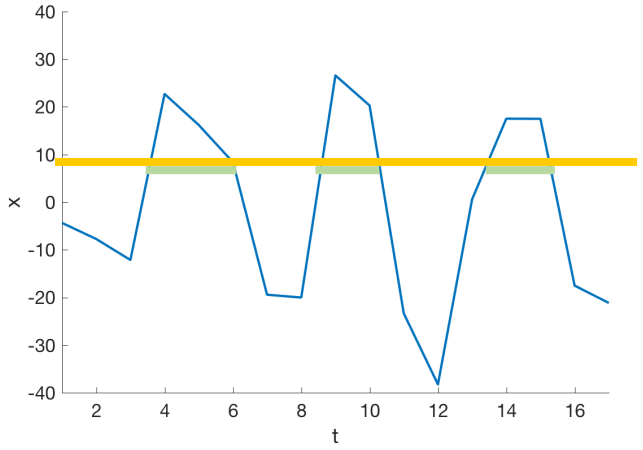
$$Q(\Phi^{T,S}, x) \geq 0$$

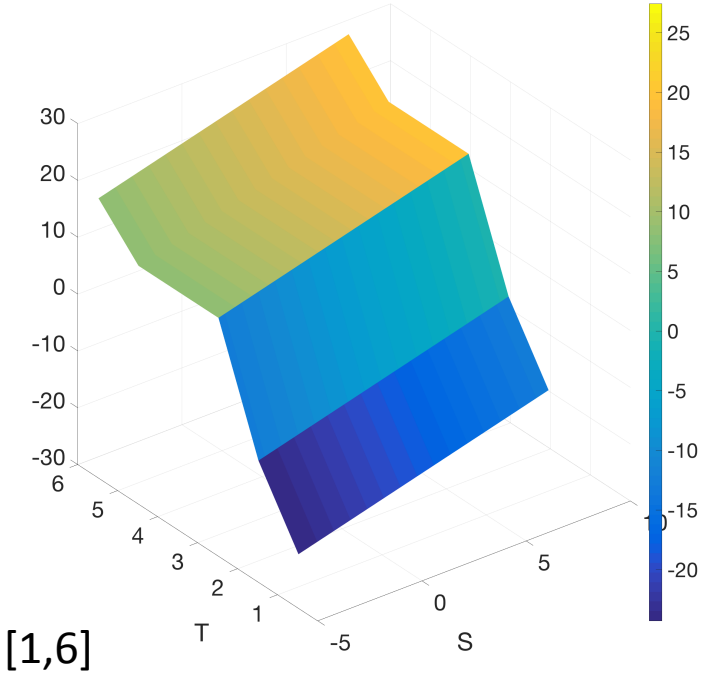
PSTL parameter synthesis

$$\Phi^{T,S} = G_{[0,SL]}(F_{[0,T]}x(t) < S)$$

$Q(\Phi^{T,S}, x)$:quantitative evaluation of **x**

Search over the parameter space.
Quantitative valuation plays a key role.





[1,6]

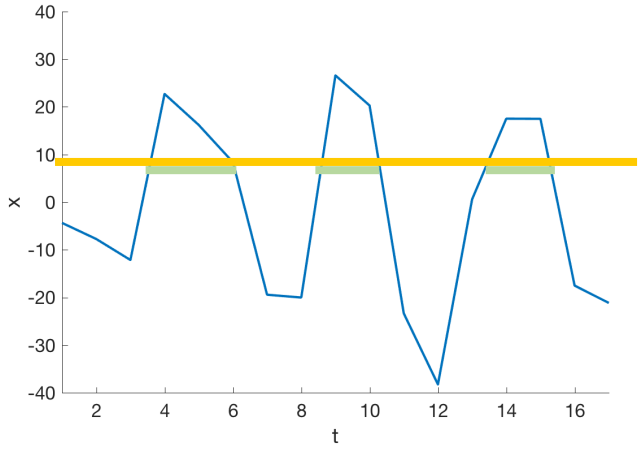
[-4,8]

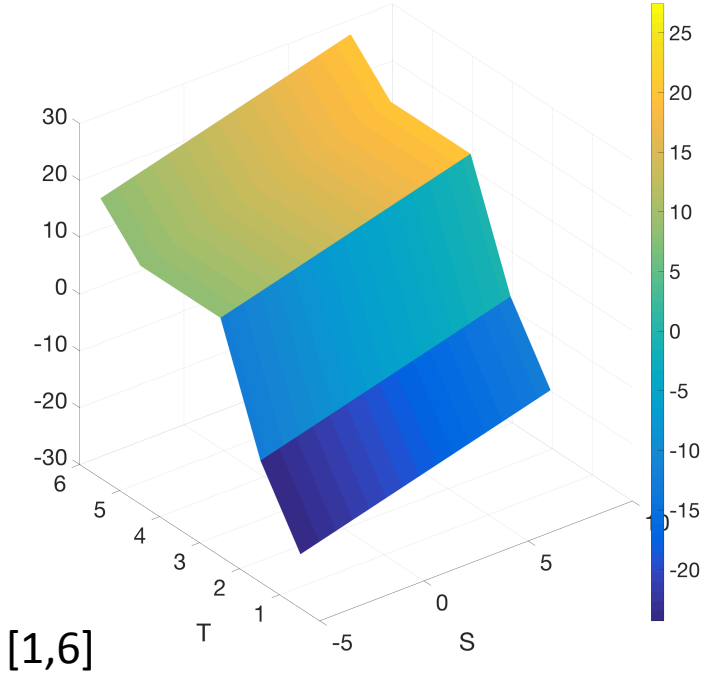
PSTL parameter synthesis

$$\Phi^{T,S} = G_{[0,SL]}(F_{[0,T]}x(t) < S)$$

$Q(\Phi^{T,S}, x)$:quantitative evaluation of **x**

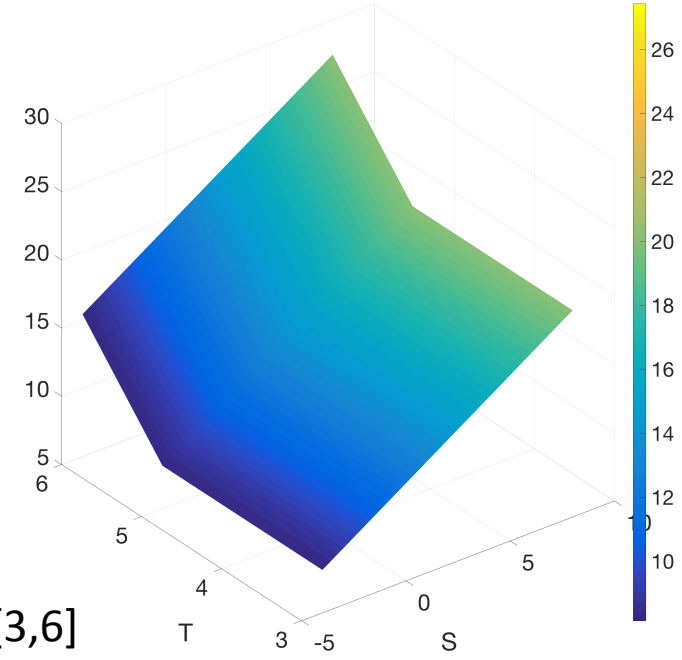
Search over the parameter space.
Quantitative valuation plays a key role.





[1,6]

[-4,8]



[3,6]

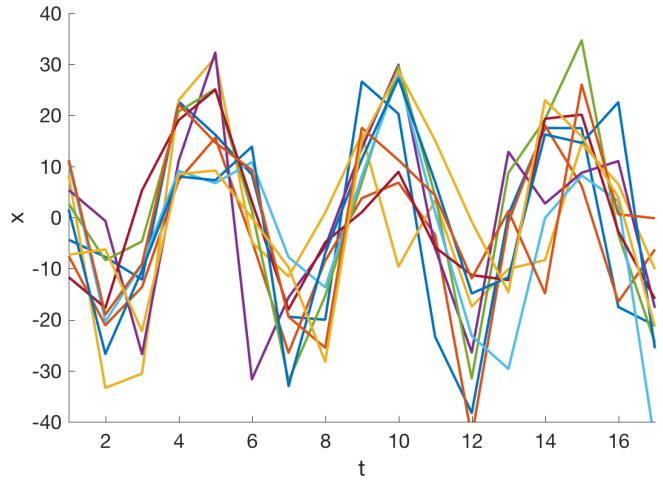
[-4,8]

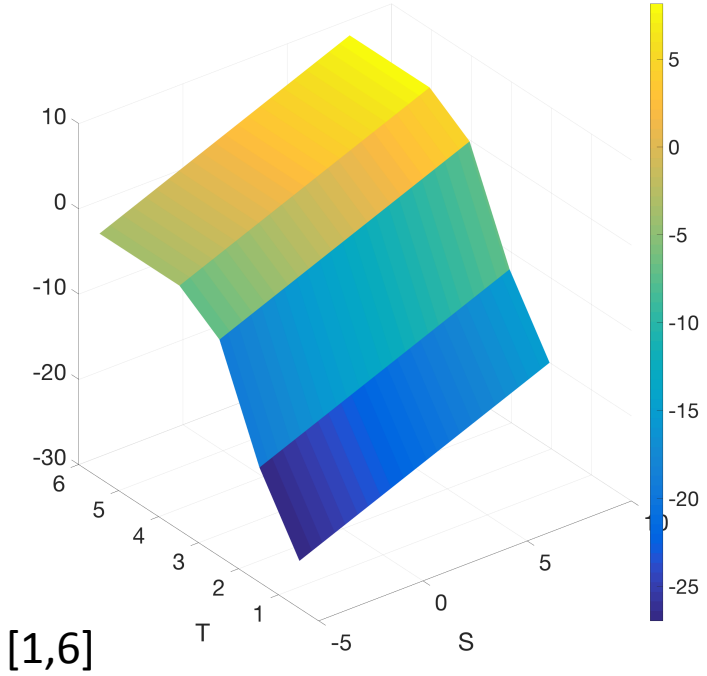
PSTL parameter synthesis

$$\Phi^{T,S} = G_{[0,SL]}(F_{[0,T]}x(t) < S)$$

$Q(\Phi^{T,S}, x)$:quantitative evaluation of **x**

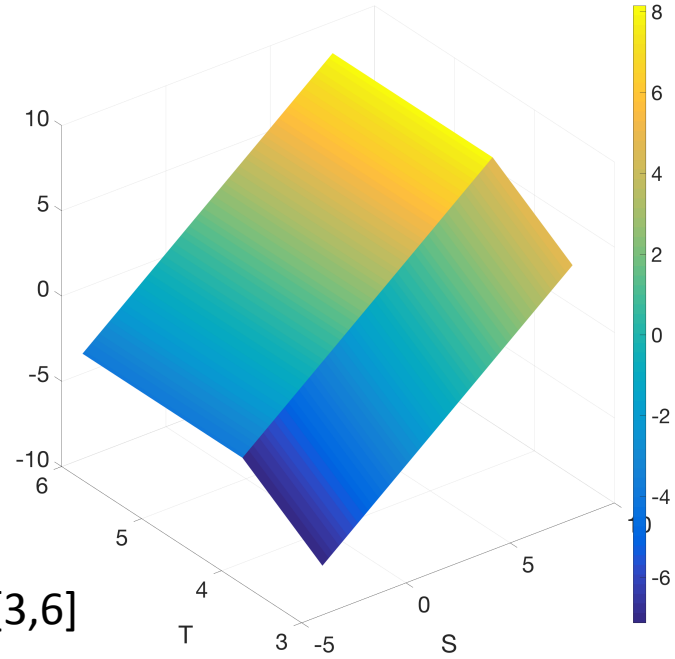
Search over the parameter space.
Quantitative valuation plays a key role.





[1,6]

[-4,8]



[3,6]

[-4,8]

minimize S,T:
subject to

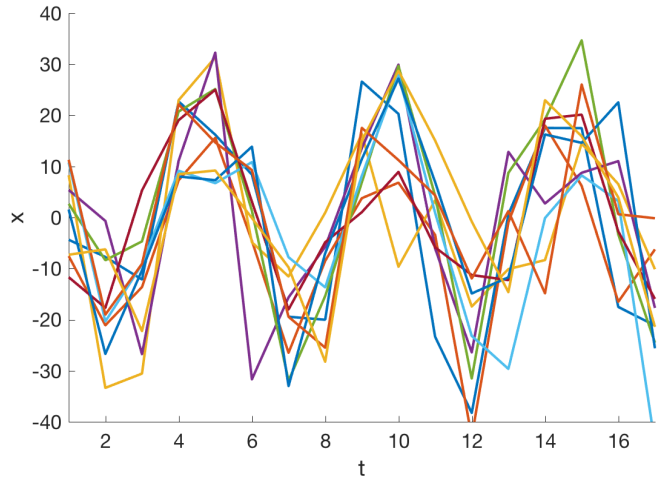
$$\min_x Q(\Phi^{T,S}, x) \geq r, r \geq 0$$

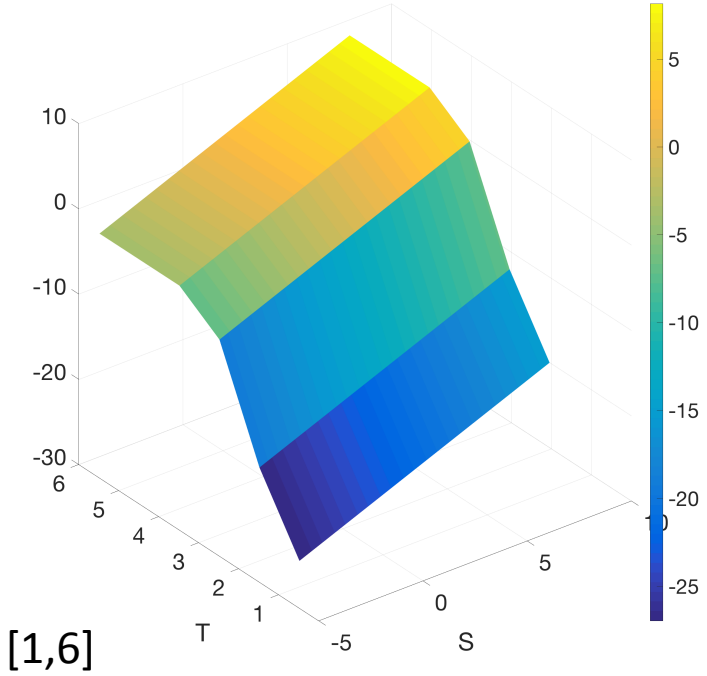
PSTL parameter synthesis

$$\Phi^{T,S} = G_{[0,SL]}(F_{[0,T]}x(t) < S)$$

$Q(\Phi^{T,S}, x)$:quantitative evaluation of **x**

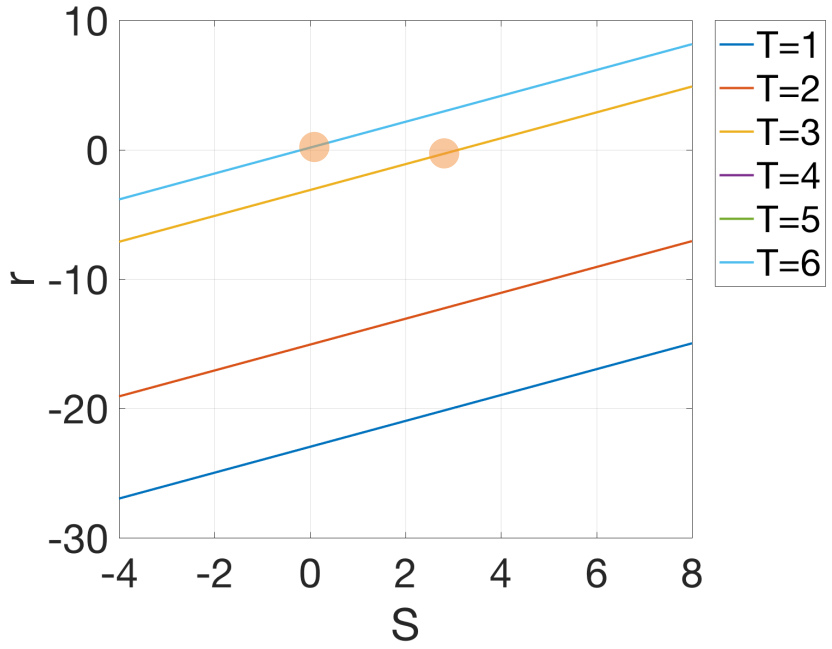
Search over the parameter space.
Quantitative valuation plays a key role.





[1,6]

[-4,8]



minimize S,T:
subject to

$$\min_x Q(\Phi^{T,S}, x) \geq r, r \geq 0$$

PSTL parameter synthesis

- **Monotonicity**
- Easy for simple cases
- Gets harder with:
  - Nested formulas
  - Multi-dimensional optimization
  - Multiple signals to consider

	R , as the parameter increase
$x > S$	Decrease
$x < S$	Increase
$F_{[0.T]} A$	Increase
$G_{[0.T]} A$	Decrease

PSTL parameter synthesis

- **Monotonicity**
- Easy for simple cases
- Gets harder with:
  - Nested formulas
  - Multi-dimensional optimization
  - Multiple signals to consider

	R , as the parameter increase
$x > S$	Decrease
$x < S$	Increase
$F_{[0.T]} A$	Increase
$G_{[0.T]} A$	Decrease

- Use binary search to find the tightest parameter (gamma tight).
- The search order of the parameter is given by the user.

- Unfortunately, not all the formulas are monotonic ($a\ U_{[t,\ t+5]}\ b$).

PSTL parameter synthesis

- **Monotonicity**

  Use binary search to find the tightest parameter (gamma tight).

  The search order of the parameter is given by the user.

  Unfortunately, not all the formulas are monotonic (a $U_{[t, t + 5]}$ b).

- Methods for optimizing the parameter search is needed.
  - Evaluation of a parameter valuation requires computing the quantitative evaluation for every signal in the set

PSTL synthesis for mining monitoring specifications

- Ongoing work:
- Identify a set of basic template formulas: Existing works depend on a given parametric formula

- Develop a search algorithm over the defined set (combine basic formulas and find parameters)

- Optimize the search algorithm: Define a preorder over the set of signals with respect to the basic parametric formulas.

PSTL synthesis for mining monitoring specifications

- Identify a set of basic template formulas

- The main formula is in $G_{[0, SL]}$ (A_1 and A_2 and …. A_n) form

- Basic parametric template formulas for A_i and their analyses
  1. $x < S$
  2. $F_{[0, T]}\, x < S$
  3. $F_{[0, T]}\, G_{[0, T2]}\, x < S$
  4. …

PSTL synthesis for mining monitoring specifications

- Develop a search algorithm over the defined set (combine basic formulas and find parameters)
  1. $A_i \rightarrow A_j$
  2. A_i and A_j
  3. $A_i \, U_{[0,T]} \, A_j$

- Optimize the search algorithm: Define a preorder over the set of signals with respect to the basic parametric formulas.

Conclusion

- Monitoring and smart alerting is essential for hardware and software systems.

- Defining effective alerting rules is an hard problem:
  - Expressivity
  - Automated methods to “pick” best rules.

- It can be cast as a PSTL parameter synthesis problem.

- Parameter synthesis methods relay on monotonicity properties of the parameters.

Conclusion

- Monitoring and smart alerting is essential for hardware and software systems.
- Defining effective alerting rules is an hard problem:
  - Expressivity
  - Automated methods to “pick” best rules.
- It can be cast as a PSTL parameter synthesis problem.
- Parameter synthesis methods relay on monotonicity properties of the parameters.

- STL Parameter synthesis has various applications:
  - Requirement mining for CPS
  - Systems and synthetic biology (automatically characterize and analyze models from experimental data, e.g. blood vessel sprouting, the programmed cell death, ….)
  - Software monitoring, release evaluation (ongoing work)