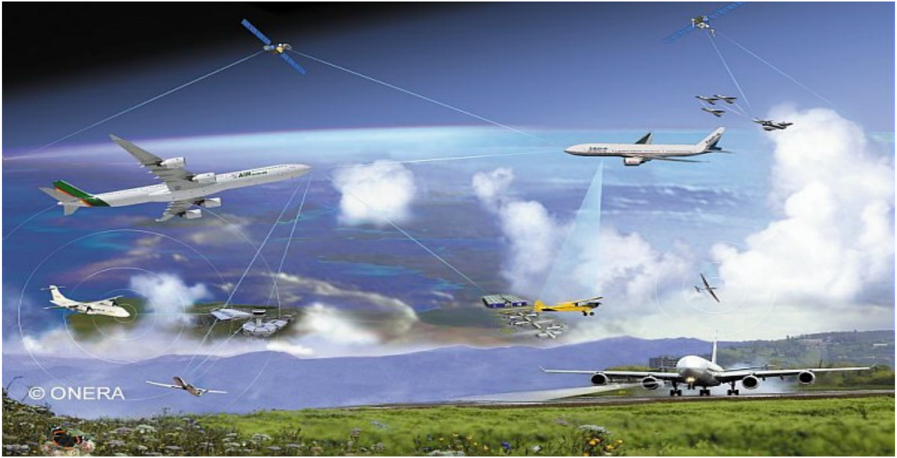


Analysis of critical control systems: combining formal analyses

Michael Dierkes joint work with Rémi Delmas, Pierre Roux, Romain Jobredeaux, Adrien Champion, and Pierre-Loïc Garoche

September 23rd 2013 – FMICS

Typical development cycle of a controller

Differential Equations (plant)

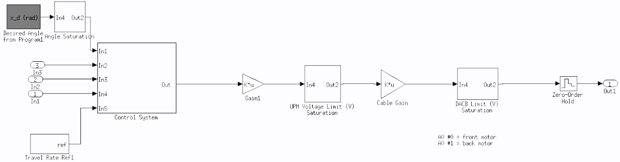
Control theorists

Typical development cycle of a controller

Differential Equations (plant)

↪ Continuous controller

Control theorists



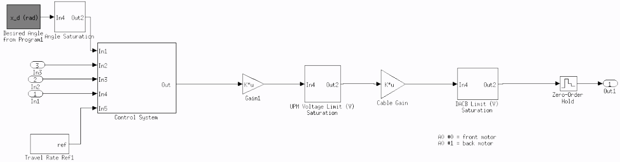
Typical development cycle of a controller

Differential Equations (plant)

Continuous controller

Discrete version

Control theorists



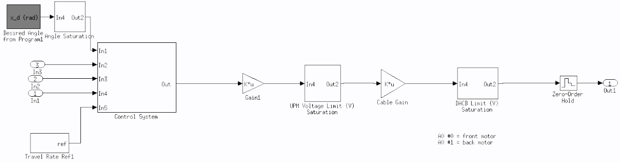
Typical development cycle of a controller

Differential Equations (plant)

→ Continuous controller

→ Discrete version

Control theorists



- Control laws design:
  - usually simplification of the plant around specific points and controlers proposed for these

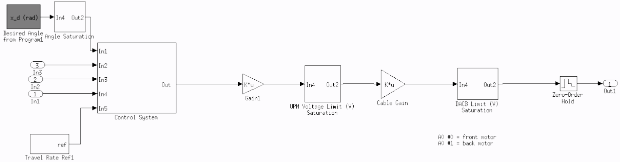
Typical development cycle of a controller

Differential Equations (plant)

→ Continuous controller

→ Discrete version

Control theorists



- Control laws design:
  - usually simplification of the plant around specific points and controlers proposed for these
  - lots of arguments/evidences on those simple cases

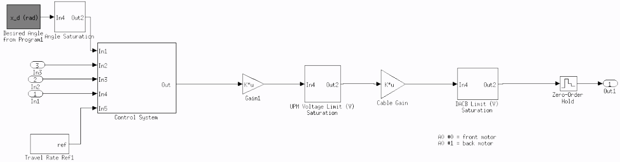
Typical development cycle of a controller

Differential Equations (plant)

→ Continuous controller

→ Discrete version

Control theorists



- Control laws design:
  - usually simplification of the plant around specific points and controlers proposed for these
  - lots of arguments/evidences on those simple cases
  - are these good controlers individualy? when composed?

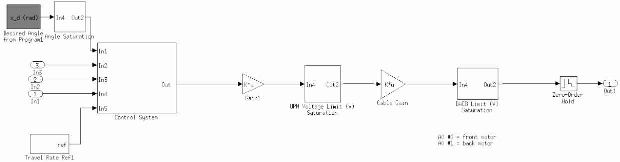
TYPICAL DEVELOPMENT CYCLE OF A CONTROLLER

Differential Equations (plant)

→ Continuous controller

→ Discrete version

Control theorists



- Control laws design:
  - usually simplification of the plant around specific points and controlers proposed for these
  - lots of arguments/evidences on those simple cases
  - are these good controlers individualy? when composed?
  - which property? stability, robustness, performances (need the plant!)

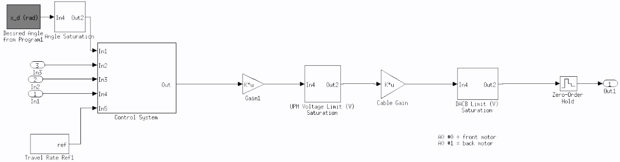
TYPICAL DEVELOPMENT CYCLE OF A CONTROLLER

Differential Equations (plant)

→ Continuous controller

→ Discrete version

Control theorists



- Control laws design:
  - usually simplification of the plant around specific points and controlers proposed for these
  - lots of arguments/evidences on those simple cases
  - are these good controlers individualy? when composed?
  - which property? stability, robustness, performances (need the plant!)
  - frequency domain proof argument vs state space domain (ie. Lyapunov functions)

Typical development cycle of a controller

Differential Equations (plant)

Continuous controller

Discrete version

Control theorists

Computer scientists

Safety architecture

redundancy, validators,
COM/MON...

Typical development cycle of a controller

Differential Equations (plant)

Continuous controller

Discrete version

Control theorists

Computer scientists

Safety architecture

redundancy, validators, COM/MON...

Test

Simulation

- Fault tolerance: set of constructs to recover from system/hardware failures
  * is this architecture sound (ie. when there is less than n simultaneaous error, the output is still valid or there will still be a working controler)

Typical development cycle of a controller

Differential Equations (plant)

→ Continuous controller

→ Discrete version

Control theorists

Computer scientists

Safety architecture

redundancy, validators, COM/MON...

Test

Simulation

Typical development cycle of a controller

Differential Equations (plant)

Continuous controller

Discrete version

Control theorists

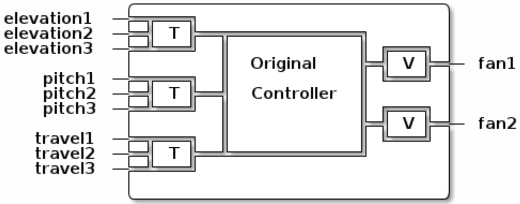
Computer scientists

Safety architecture

redundancy, validators, COM/MON...

Test

Simulation



TYPICAL DEVELOPMENT CYCLE OF A CONTROLLER

Differential Equations (plant)

→ Continuous controller

→ Discrete version

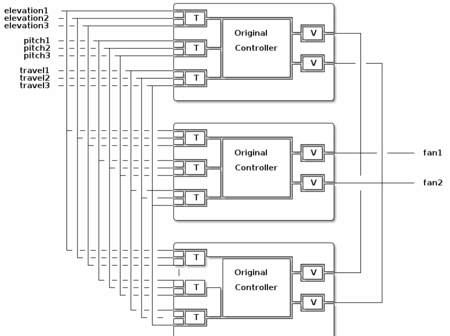
Control theorists

Computer scientists

↓

Safety architecture ⇝ Test Simulation

redundancy, validators, COM/MON...



Typical development cycle of a controller

Differential Equations (plant)

Continuous controller

Discrete version

Control theorists

Computer scientists

Safety architecture

redundancy, validators,
COM/MON...

Test
Simulation

Code

Typical development cycle of a controller

Differential Equations (plant)

→ Continuous controller

→ Discrete version

Control theorists

Computer scientists

↓

Safety architecture

redundancy, validators, COM/MON...

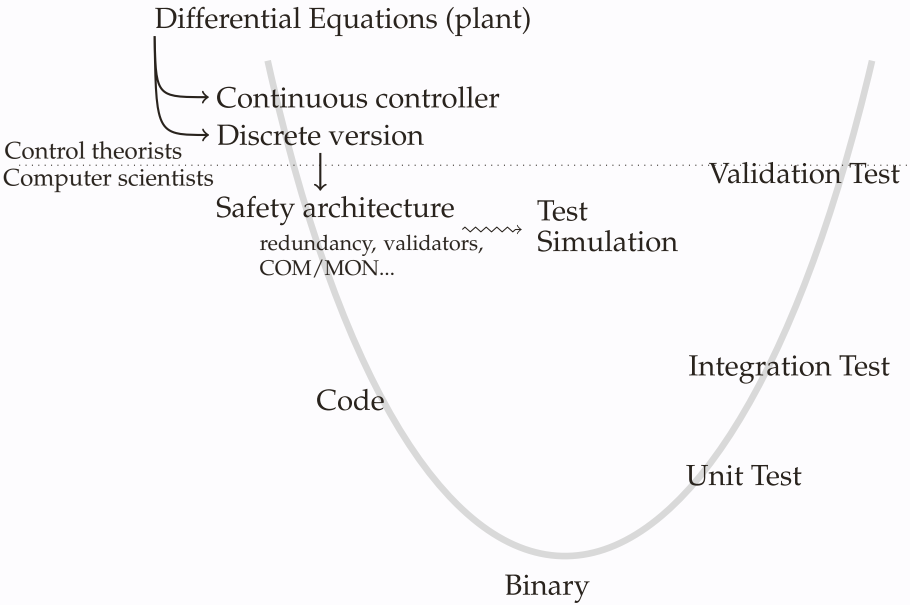
Test

Simulation

Code

- Actual implementation:
  - floats not reals
  - pointers, arrays, memory access → potential failure
  - real world: overflows

Typical development cycle of a controller



Verification methods used in the industry

Dynamic analysis

- test, simulation (test on simulated environement)

Verification methods used in the industry

Dynamic analysis

- test, simulation (test on simulated environement)

Static

- model-checking: logical reasoning about abstraction (models) of the system

Verification methods used in the industry

Dynamic analysis

- test, simulation (test on simulated environement)

Static

- model-checking: logical reasoning about abstraction (models) of the system
  - SAT/SMT based model-checking: encode model-checking problem as SMT satisfiability check. Eg. (k-)inductiveness of a property on the model semantics.

Verification methods used in the industry

Dynamic analysis

- test, simulation (test on simulated environement)

Static

- model-checking: logical reasoning about abstraction (models) of the system
  - SAT/SMT based model-checking: encode model-checking problem as SMT satisfiability check. Eg. (k-)inductiveness of a property on the model semantics.
- static analysis of the code/model: compute an abstract representation of reachable state, mainly focuses on numerical accuracy, or data structure topology and manipulation (null pointers access, arrays, …)

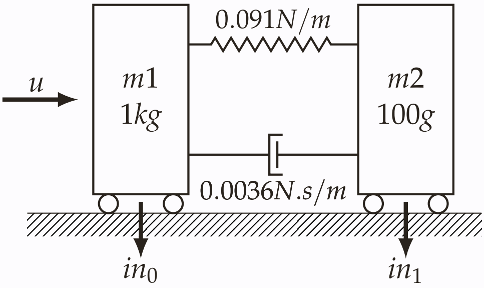
Running example

Simple yet hard to analyze controller for a mass-spring damper

Running example

Simple yet hard to analyze controller for a mass-spring damper

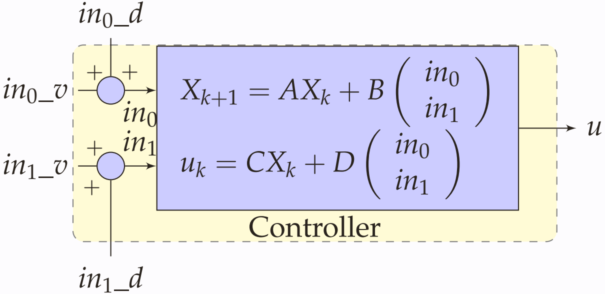
System to be controlled:



Running example

Simple yet hard to analyze controller for a mass-spring damper

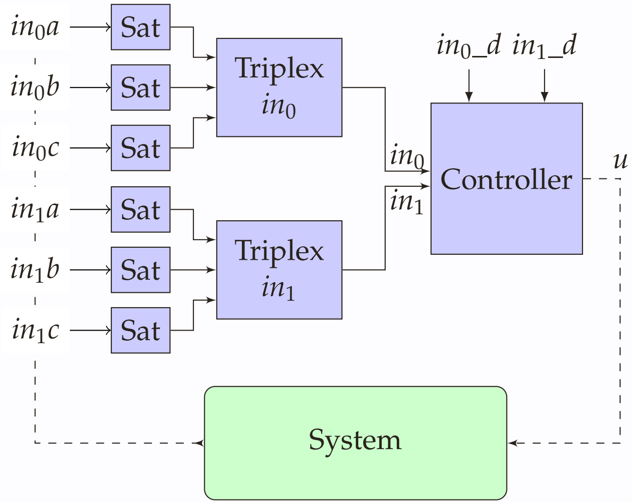
Controller itself:



Running example

Simple yet hard to analyze controller for a mass-spring damper

Fault tolerance architecture:



Non linear analyses based on Abstract interpretation

- Choose an appropriate abstraction depending on the property to be proved (boundedness, relationship between variables, memory issues, etc)

Non linear analyses based on Abstract interpretation

- Choose an appropriate abstraction depending on the property to be proved (boundedness, relationship between variables, memory issues, etc)
- Express the model semantics in the abstract domain

Non linear analyses based on Abstract interpretation

- Choose an appropriate abstraction depending on the property to be proved (boundedness, relationship between variables, memory issues, etc)
- Express the model semantics in the abstract domain
- Compute an over approximation of reachable states in the abstract domain.

Non linear analyses based on Abstract interpretation

- Choose an appropriate abstraction depending on the property to be proved (boundedness, relationship between variables, memory issues, etc)
- Express the model semantics in the abstract domain
- Compute an over approximation of reachable states in the abstract domain.

Non linear analyses based on Abstract interpretation

- Choose an appropriate abstraction depending on the property to be proved (boundedness, relationship between variables, memory issues, etc)
- Express the model semantics in the abstract domain
- Compute an over approximation of reachable states in the abstract domain.

Stable linear controllers with or without saturations are analyzed using a specific abstract domain:

1. The control flow graph of the controller is identified

Non linear analyses based on Abstract interpretation

- Choose an appropriate abstraction depending on the property to be proved (boundedness, relationship between variables, memory issues, etc)
- Express the model semantics in the abstract domain
- Compute an over approximation of reachable states in the abstract domain.

Stable linear controllers with or without saturations are analyzed using a specific abstract domain:

1. The control flow graph of the controller is identified
2. The stability of each linear subsystem is analyzed and provides a quadratic Lyapunov function (ellipsoid)

Non linear analyses based on Abstract interpretation

- Choose an appropriate abstraction depending on the property to be proved (boundedness, relationship between variables, memory issues, etc)
- Express the model semantics in the abstract domain
- Compute an over approximation of reachable states in the abstract domain.

Stable linear controllers with or without saturations are analyzed using a specific abstract domain:

1. The control flow graph of the controller is identified
2. The stability of each linear subsystem is analyzed and provides a quadratic Lyapunov function (ellipsoid)
3. The set of reachable states is bounded using the generated ellipsoids.

Model-checking based on SMT solvers

- Encode the model semantics as a predicate in SMT logics: M(x,y)

Model-checking based on SMT solvers

- Encode the model semantics as a predicate in SMT logics: M(x,y)
- Perform inductive reasoning for a given property:

Model-checking based on SMT solvers

- Encode the model semantics as a predicate in SMT logics: M(x,y)
- Perform inductive reasoning for a given property:
  - eg: $true \models P(init)$ and $P(x) \wedge M(x, y) \models P(y)$

Model-checking based on SMT solvers

- Encode the model semantics as a predicate in SMT logics: M(x,y)
- Perform inductive reasoning for a given property:
  - eg: $true \models P(init)$ and $P(x) \wedge M(x, y) \models P(y)$
- Compute backward analysis using quantifier elimination: identify over-approximation of states violating the property

Model-checking based on SMT solvers

- Encode the model semantics as a predicate in SMT logics: M(x,y)
- Perform inductive reasoning for a given property:
  - eg: $true \models P(init)$ and $P(x) \wedge M(x,y) \models P(y)$
- Compute backward analysis using quantifier elimination: identify over-approximation of states violating the property
  - characterize a disjunction of polyhedra over-approximating bad states

Model-checking based on SMT solvers

- Encode the model semantics as a predicate in SMT logics: M(x,y)
- Perform inductive reasoning for a given property:
  - eg: $true \models P(init)$ and $P(x) \wedge M(x,y) \models P(y)$
- Compute backward analysis using quantifier elimination: identify over-approximation of states violating the property
  - characterize a disjunction of polyhedra over-approximating bad states
  - proving the non reachability of this set from the initial state proves the property

Model-checking based on SMT solvers

- Encode the model semantics as a predicate in SMT logics: M(x,y)
- Perform inductive reasoning for a given property:
  - eg: $true \models P(init)$ and $P(x) \wedge M(x,y) \models P(y)$
- Compute backward analysis using quantifier elimination: identify over-approximation of states violating the property
  - characterize a disjunction of polyhedra over-approximating bad states
  - proving the non reachability of this set from the initial state proves the property

Model-checking based on SMT solvers

- Encode the model semantics as a predicate in SMT logics: M(x,y)
- Perform inductive reasoning for a given property:
  * eg: $true \models P(init)$ and $P(x) \wedge M(x, y) \models P(y)$
- Compute backward analysis using quantifier elimination: identify over-approximation of states violating the property
  * characterize a disjunction of polyhedra over-approximating bad states
  * proving the non reachability of this set from the initial state proves the property

Both techniques perform well in practive - and are used industrially - but

- are restricted to linear inductive or k-inductive properties;

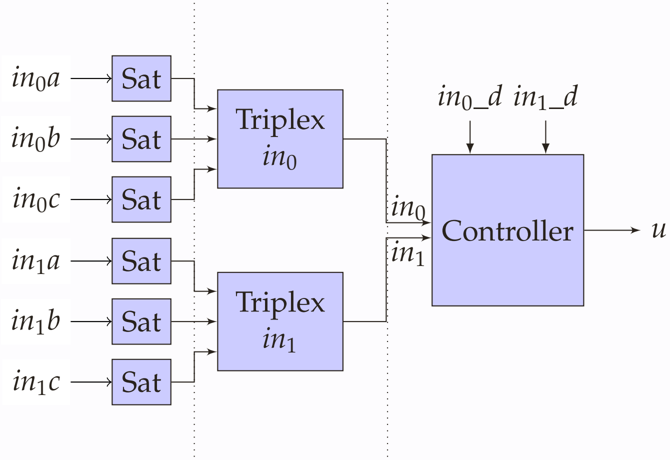
Model-checking based on SMT solvers

- Encode the model semantics as a predicate in SMT logics: M(x,y)
- Perform inductive reasoning for a given property:
  * eg: $true \models P(init)$ and $P(x) \wedge M(x,y) \models P(y)$
- Compute backward analysis using quantifier elimination: identify over-approximation of states violating the property
  * characterize a disjunction of polyhedra over-approximating bad states
  * proving the non reachability of this set from the initial state proves the property

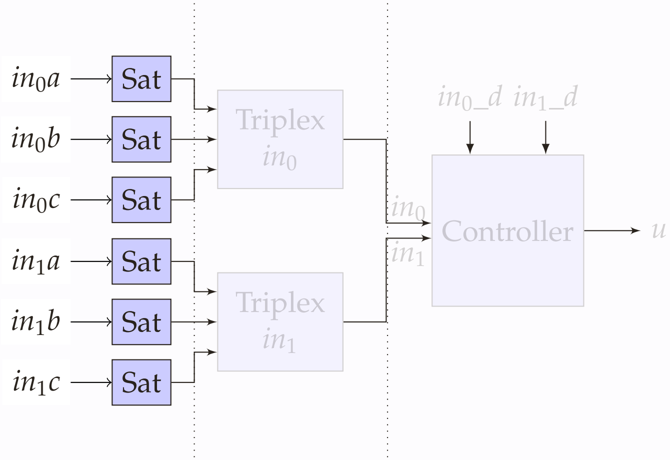
Both techniques perform well in practive - and are used industrially - but

- are restricted to linear inductive or k-inductive properties;
- do not give good results in presence of complex numerical computations

Combining Analyses

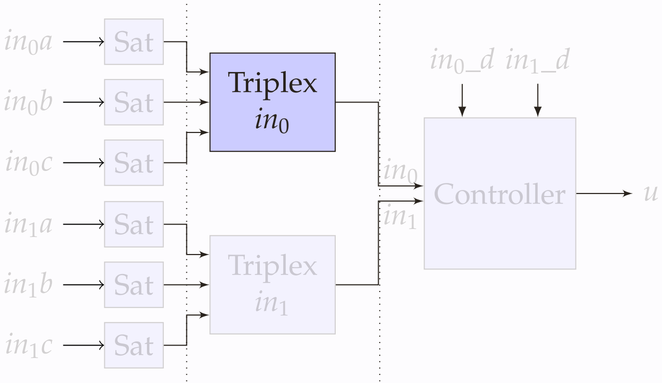


BASIC SATURATIONS



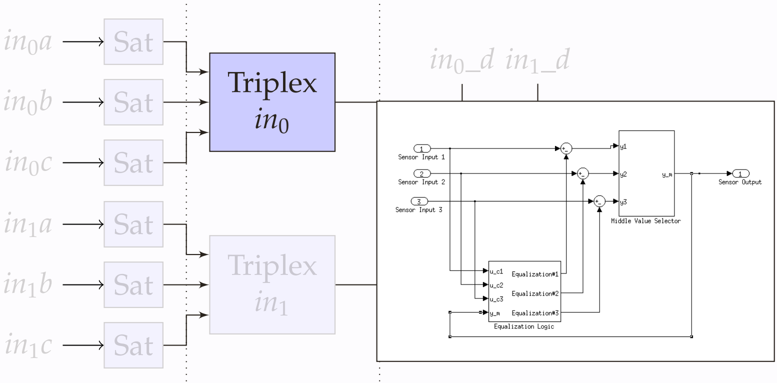
Abstract Interpretation computes a sound bound (1.2) on each ouptut whatever the value of in_xy is.

ANALYSIS OF THE TRIPLEX VOTER



Backward analysis applied on each triplex proves the specification BIBO.

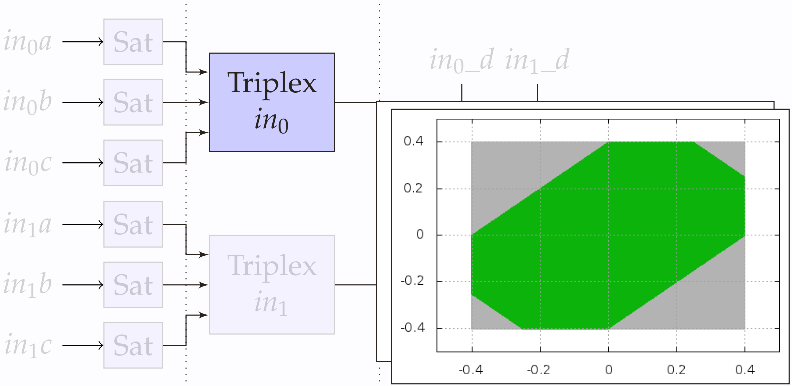
Analysis of the triplex voter



Backward analysis applied on each triplex proves the specification BIBO.

$$\forall k \in \mathbb{N}, \quad |InA_k| \leq a \wedge |InB_k| \leq a \wedge |InC_k| \leq a \implies |Output_k| \leq 3a \;\wedge$$
$$|EqualizationA_k| \leq 2a \wedge |EqualizationB_k| \leq 2a \wedge |lyauEqualizationC_k| \leq 2a$$

Analysis of the triplex voter

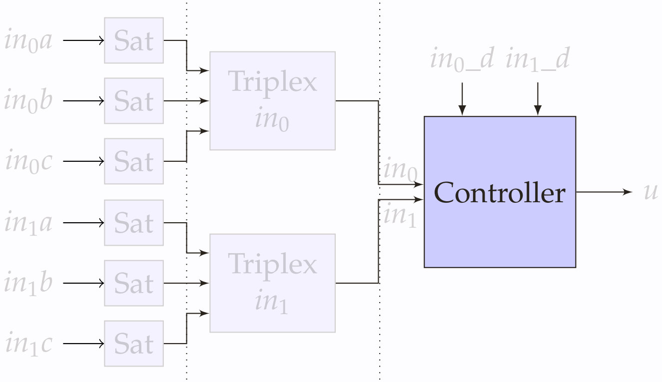


Backward analysis applied on each triplex proves the specification BIBO.

$$\forall k \in \mathbb{N},\ \ |InA_k| \leq a \wedge |InB_k| \leq a \wedge |InC_k| \leq a \implies |Output_k| \leq 3a \ \wedge$$
$$|EqualizationA_k| \leq 2a \wedge |EqualizationB_k| \leq 2a \wedge |lyauEqualizationC_k| \leq 2a$$

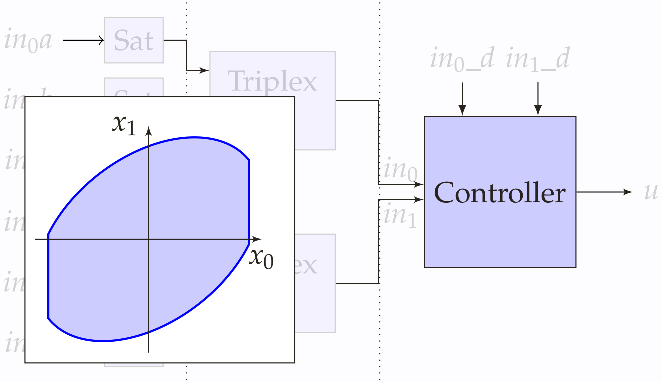
Assuming input is bounded by 1.2, we have output bounded by 3.6.

ANALYSIS OF THE CONTROLLER



Providing a bound on the inputs (3.6) an over-approximation of the output is computed:

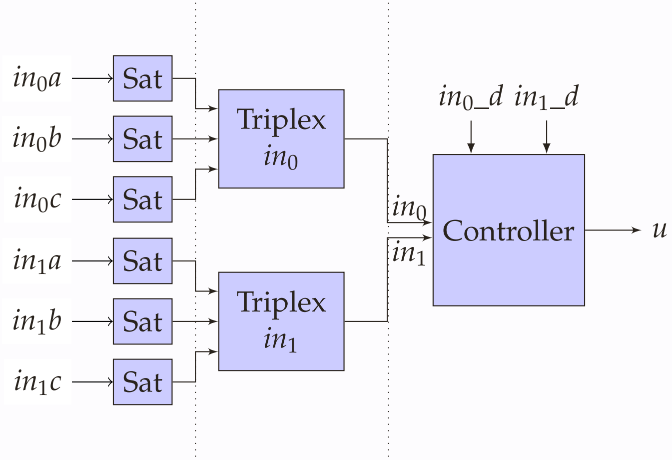
Analysis of the controller



Providing a bound on the inputs (3.6) an over-approximation of the output is computed: $|u| \leq 194.499$.

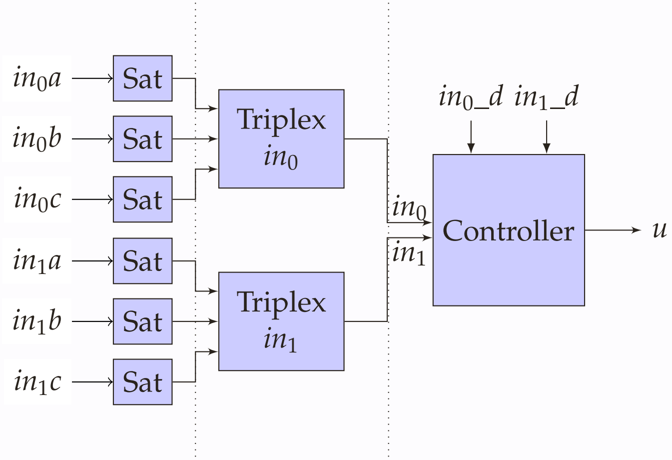
$$0.098\,x_3^2 - 0.224\,x_3\,x_2 + 0.040\,x_3\,x_1 - 0.026\,x_3\,x_0 + 0.141\,x_2^2 - 0.053\,x_2\,x_1 \\ +0.030\,x_2\,x_0 + 0.024\,x_1^2 - 0.017\,x_1\,x_0 + 0.019\,x_0^2 \leq 14.259$$

REBUILDING THE ANALYSIS



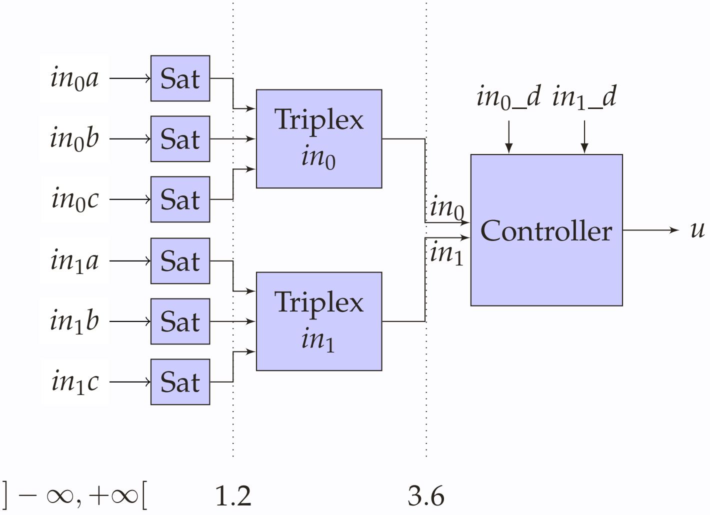
$]-\infty, +\infty[$

REBUILDING THE ANALYSIS

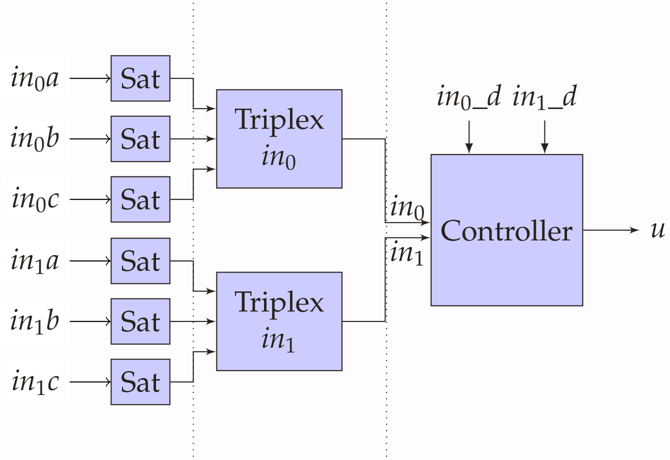


$]-\infty, +\infty[$ 1.2

Rebuilding the analysis

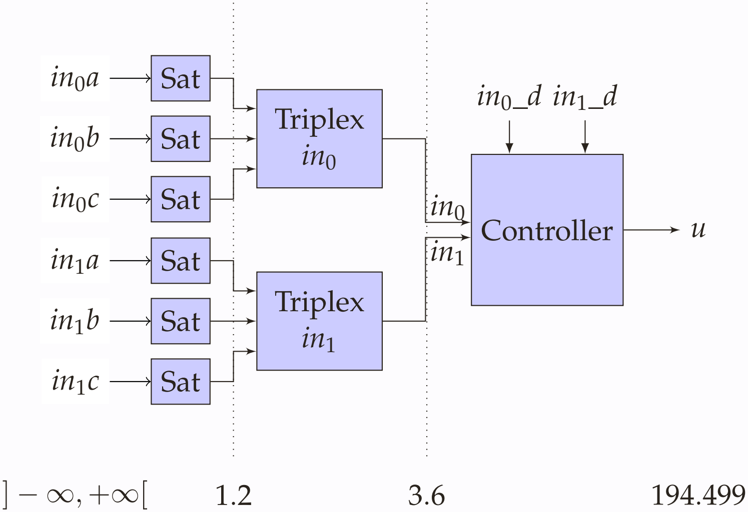


Rebuilding the analysis



$]-\infty, +\infty[$ 1.2 3.6 194.499

REBUILDING THE ANALYSIS



System is bounded!

Conclusion

Successful approach to analyze representative example un-analyzable with a single method.

Conclusion

Successful approach to analyze representative example un-analyzable with a single method.

We are advocating for

- formal specification
- traceability of component origin to help select the best method to analyze them
- combination of formal methods to achieve the complete verification of the software

Conclusion

Successful approach to analyze representative example un-analyzable with a single method.

We are advocating for

- formal specification
- traceability of component origin to help select the best method to analyze them
- combination of formal methods to achieve the complete verification of the software

Good results on simple usecase. Currently addressing the analysis of industry-level FADEC (collab. with industrial partners) and academic yet representative examples of aircraft controllers (collab. with Polytech Montréal, Georgia Tech and NASA).