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

1. Differential Equations (plant)
2. Continuous controller
3. Discrete version
4. Safety architecture
   - redundancy, validators,
   - COM/MON...
5. Test
   - Simulation
6. Code
   - Binary
7. Unit Test
8. Integration Test
9. Validation Test

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 controllers 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 controllers 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 controllers proposed for these
* lots of arguments/evidences on those simple cases
* are these good controllers individually? 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 controllers proposed for these
* lots of arguments/evidences on those simple cases
* are these good controllers individually? 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 controllers proposed for these
* lots of arguments/evidences on those simple cases
* are these good controllers individually? 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
      - Safety architecture
        - redundancy, validators, COM/MON...

Control theorists
Computer scientists
**Typical development cycle of a controller**

1. **Differential Equations (plant)**
   - Continuous controller
   - Discrete version

2. **Safety architecture**
   - Redundancy, validators, COM/MON...

3. **Test**
   - Simulation

**Control theorists**

**Computer scientists**

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

Diagram: elevation, pitch, travel → controller → fan1, fan2
**Typical Development Cycle of a Controller**

- **Differential Equations (plant)**
- **Continuous controller**
- **Discrete version**
- **Safety architecture**
  - redundancy, validators, COM/MON...
- **Test**
  - Simulation

Control theorists → Computer scientists

![Diagram of controller development cycle]
Typical development cycle of a controller

Differential Equations (plant) → Continuous controller → Discrete version

Safety architecture → Test
    redundancy, validators, COM/MON...

Control theorists
Computer scientists

Test
Simulation

Code → Binary

Unit Test → Integration Test → Validation Test
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

Differential Equations (plant) → Continuous controller → Discrete version

Safety architecture: redundancy, validators, COM/MON...

Control theorists → Code → Binary
Computer scientists

Test → Simulation

Integration Test
Unit Test
Validation Test
Veriﬁcation methods used in the industry

Dynamic analysis
- test, simulation (test on simulated environment)

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, . . .)

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 environment)

Static
- model-checking: logical reasoning about abstraction (models) of the system
**Verification methods used in the industry**

Dynamic analysis
- test, simulation (test on simulated environment)

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 environment)

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:

\[
X_{k+1} = AX_k + B \begin{pmatrix} \text{in}_0 \\ \text{in}_1 \end{pmatrix}
\]

\[
u_k = CX_k + D \begin{pmatrix} \text{in}_0 \\ \text{in}_1 \end{pmatrix}
\]
Fault tolerance architecture:

```
in_0a -> Sat -> Triplex \(_{in_0}\) -> Controller
in_0b -> Sat
in_0c -> Sat

in_1a -> Sat
in_1b -> Sat
in_1c -> Sat

Triplex \(_{in_1}\) -> Controller

\(in_{0\_d}\), \(in_{1\_d}\) -> Controller
\(in_0\) -> Controller
\(in_1\) -> Controller

\(u\) -> System
```
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.

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.
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)
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: \( M(x,y) \)
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 $P(x) \land M(x, y) \models P(y)$
Encode the model semantics as a predicate in SMT logics: \( M(x,y) \)

Perform inductive reasoning for a given property:
- eg: \( \text{true} \models P(\text{init}) \) and \( P(x) \land M(x,y) \models P(y) \)

Compute backward analysis using quantifier elimination: identify over-approximation of states violating the property
Encode the model semantics as a predicate in SMT logics: \( M(x,y) \)

Perform inductive reasoning for a given property:
  * eg: \( \text{true} \models P(\text{init}) \) and \( P(x) \land M(x,y) \models P(y) \)

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 $P(x) \land M(x,y) \models P(y)$
- 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 $P(x) \land M(x,y) \models P(y)$
- 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 practice 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.
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 $P(x) \land M(x, y) \models P(y)$

- 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 practice - 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 $P(x) \land M(x,y) \models P(y)$
- 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 practice - 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

Controller

\[\text{in}_0 \rightarrow \text{Sat} \rightarrow \text{Triplex } in_0 \rightarrow \text{Controller} \rightarrow u\]

\[\text{in}_1 \rightarrow \text{Sat} \rightarrow \text{Triplex } in_1 \rightarrow \text{Controller} \rightarrow u\]

\[\text{in}_0a, \text{in}_0b, \text{in}_0c, \text{in}_1a, \text{in}_1b, \text{in}_1c\]
Abstract Interpretation computes a sound bound (1.2) on each output whatever the value of $in_{x y}$ is.
Backward analysis applied on each triplex proves the specification BIBO.
Backward analysis applied on each triplex proves the specification BIBO.

∀k ∈ ℕ, \(|InA_k| \leq a \land |InB_k| \leq a \land |InC_k| \leq a \implies |Output_k| \leq 3a \land |EqualizationA_k| \leq 2a \land |EqualizationB_k| \leq 2a \land |lyauEqualizationC_k| \leq 2a|
Analysis of the triplex voter

Backward analysis applied on each triplex proves the specification BIBO.

\[ \forall k \in \mathbb{N}, \ |InA_k| \leq a \land |InB_k| \leq a \land |InC_k| \leq a \implies |Output_k| \leq 3a \land |EqualizationA_k| \leq 2a \land |EqualizationB_k| \leq 2a \land |EqualizationC_k| \leq 2a \]

Assuming input is bounded by 1.2, we have output bounded by 3.6.
Providing a bound on the inputs (3.6) an over-approximation of the output is computed:
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

\[ \text{System is bounded!} \]
Rebuilding the analysis

\[ ] - \infty, +\infty[\quad 1.2\]
Rebuilding the analysis

System is bounded!
Rebuilding the analysis

System is bounded!
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).