Utilisation de fonctions de Lyapunov pour l’analyse de logiciels de contrôle-commande

Pierre-Loïc Garoche
with the strong support of Eric Féron, Pierre Roux, Tim Wang, Romain Jobredeaux, Assalé Adjé, and Victor Magron

April 1st, 2015
CONTENTS

Context

Background
  Formal analysis of software model/code
  Lyapunov functions and SDP solvers

Application to controller formal analysis
  Hoare style annotations
  Lyapunov function synthesis and Policy Iteration

Soundness of the analysis

Current activities: more complex templates/properties

Conclusion
Contents

Context

Background
Formal analysis of software model/code
Lyapunov functions and SDP solvers

Application to controller formal analysis
Hoare style annotations
Lyapunov function synthesis and Policy Iteration

Soundness of the analysis

Current activities: more complex templates/properties

Conclusion
Some context: Objective

- Analysis of control command software
  - what are they?
  - where do they come from?
  - challenges from a Computer Science perspective
- what is this talk about? what it is not?
CONTROL PEOPLE vs CS PEOPLE
DIFFERENT BACKGROUND, DIFFERENT METHODS

A caricatured view of each community to the other:

Control over CS:
- these guys just implement our controllers. They don’t need to understand them: we do.
- Implementation is a straightforward process.

CS over Control:
- they provide us with numerical specification to implement (eg. PID gains) without any clue on what to validate.
- their proofs are lousy since they do not consider precision of computation.

There is an urgent need to gather the communities and do cross-fertilization:
- what are the interesting properties?
- how to validate them along the development process?
- considering all later stages of development of the system.
CONTROL PEOPLE VS CS PEOPLE
DIFFERENT BACKGROUND, DIFFERENT METHODS

A caricatured view of each community to the other:

Control over CS:
▶ these guys just implement our controllers. They don’t need to understand them: we do.
▶ Implementation is a straightforward process.

CS over Control:
▶ they provide us with numerical specification to implement (eg. PID gains) without any clue on what to validate.
▶ their proofs are lousy since they do not consider precision of computation
CONTROL PEOPLE VS CS PEOPLE
DIFFERENT BACKGROUND, DIFFERENT METHODS

A caricatured view of each community to the other:

Control over CS:
- these guys just implement our controllers. They don’t need to understand them: we do.
- Implementation is a straightforward process.

CS over Control:
- they provide us with numerical specification to implement (eg. PID gains) without any clue on what to validate.
- their proofs are lousy since they do not consider precision of computation

There is an urgent need to gather the communities and do cross-fertilization:
- what are the interesting properties?
- how to validate them along the development process?
- considering all later stages of development of the system.
DESIGNING CONTROLLERS

TYPICAL DEVELOPMENT CYCLE OF A CONTROLLER FOR AN AIRCRAFT

Differential Equations (plant)

Control theorists
DESIGNING CONTROLLERS

Typical development cycle of a controller for an aircraft

Differential Equations (plant) → Continuous controller

Control theorists
DESIGNING CONTROLLERS

TYPICAL DEVELOPMENT CYCLE OF A CONTROLLER FOR AN AIRCRAFT

Differential Equations (plant)

→ Continuous controller

→ Discrete version

Control theorists
DESIGNING CONTROLLERS

TYPICAL DEVELOPMENT CYCLE OF A CONTROLLER FOR AN AIRCRAFT

Differential Equations (plant) → Continuous controller → Discrete version

Control theorists

► Control laws design: typically w. synchronous models
  ► usually simplification of the plant around specific points and controllers proposed for these
DESIGNING CONTROLLERS

TYPICAL DEVELOPMENT CYCLE OF A CONTROLLER FOR AN AIRCRAFT

- Differential Equations (plant)
- Continuous controller
- Discrete version

Control theorists

- Control laws design: typically w. synchronous models
  - usually simplification of the plant around specific points and controllers proposed for these
  - lots of arguments/evidences on those simple cases
**DESIGNING CONTROLLERS**

**TYPICAL DEVELOPMENT CYCLE OF A CONTROLLER FOR AN AIRCRAFT**

![Diagram of control system]

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

- **Control theorists**

- **Control laws design**: typically w. synchronous models
  - usually simplification of the plant around specific points and controllers proposed for these
  - lots of arguments/evidences on those simple cases
  - which property? stability, robustness, performances (need the plant!)
DESIGNING CONTROLLERS

Typical development cycle of a controller for an aircraft

Differential Equations (plant)

→ Continuous controller

→ Discrete version

Control theorists

Control laws design: typically w. synchronous models

► usually simplification of the plant around specific points and controllers proposed for these
► lots of arguments/evidences on those simple cases
► which property? stability, robustness, performances (need the plant!)
► frequency domain proof argument vs state space domain (ie. Lyapunov functions)
DESIGNING CONTROLLERS

TYPICAL DEVELOPMENT CYCLE OF A CONTROLLER FOR AN AIRCRAFT

Differential Equations (plant) → Continuous controller → Discrete version

Control theorists
DESIGNING CONTROLLERS

TYPICAL DEVELOPMENT CYCLE OF A CONTROLLER FOR AN AIRCRAFT

Differential Equations (plant)

→ Continuous controller

→ Discrete version

Control theorists

Computer scientists

Safety architecture

redundancy, validators, COM/MON...
DESIGNING CONTROLLERS

Typical development cycle of a controller for an aircraft

- 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 simultaneous error, the output is still valid or there will still be a working controller)
- protection against software error (bug, Run Time Error)
- protection against hardware error (SEU, crashed computer, deadlock, deadline misses)
DESIGNING CONTROLLERS

TYPICAL DEVELOPMENT CYCLE OF A CONTROLLER FOR AN AIRCRAFT

Differential Equations (plant)

→ Continuous controller

→ Discrete version

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

Control theorists

Computer scientists

Test

Simulation
DESIGNING CONTROLLERS

TYPICAL DEVELOPMENT CYCLE OF A CONTROLLER FOR AN AIRCRAFT

Differential Equations (plant)

→ Continuous controller

→ Discrete version

Control theorists
Computer scientists

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

Test
Simulation

Code
Binary

Unit Test
Integration Test
Validation Test

Control theorists
Computer scientists
DESIGNING CONTROLLERS

TYPICAL DEVELOPMENT CYCLE OF A CONTROLLER FOR AN AIRCRAFT

Differential Equations (plant)

→ Continuous controller

→ Discrete version

Safety architecture

redundancy, validators,
COM/MON...

Test
Simulation

Control theorists

Computer scientists

Binary

Unit Test
Integration Test
Validation Test

DESIGNING CONTROLLERS

TYPICAL DEVELOPMENT CYCLE OF A CONTROLLER FOR AN AIRCRAFT

Differential Equations (plant) → Continuous controller

Control theorists

Discrete version

Computer scientists

Safety architecture

redundancy, validators, COM/MON...

Test

Simulation

Code
DESIGNING CONTROLLERS

Typical development cycle of a controller for an aircraft

- Differential Equations (plant)
- Continuous controller
- Discrete version

Control theorists → Safety architecture (redundancy, validators, COM/MON...)
Computer scientists → Test → Simulation

Code

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

TYPICAL DEVELOPMENT CYCLE OF A CONTROLLER FOR AN AIRCRAFT

Differential Equations (plant)

→ Continuous controller
→ Discrete version

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

Control theorists
Computer scientists

Code
Binary

Unit Test
Integration Test
Simulation
Validation Test

Test
**System Example: Basic Triplication Pattern**

- **Sat in 0**: inputs \( in_0a, in_0b, in_0c \)
- **Sat in 1**: inputs \( in_1a, in_1b, in_1c \)
- **Triplex in 0**: output \( in_0 \)
- **Triplex in 1**: output \( in_1 \)
- **Controller**: inputs \( in_0, in_1 \), output \( u \)
- **System**: inputs \( in_0_d, in_1_d \)
EXAMPLE OF A SAFETY COMPONENT: A TRIPLEX VOTER

- Mean computation: $\frac{1}{3} (a + b + c)$
EXAMPLE OF A SAFETY COMPONENT: A TRIPLEX VOTER

- Mean computation: $1/3 (a + b + c)$
- Middle value: $\max(\max(\min(a, b), \min(a, c)), \min(b, c))$
EXAMPLE OF A SAFETY COMPONENT: A TRIPLEX VOTER

- **Mean computation:** $1/3 (a + b + c)$
- **Middle value:** $\max(\max(\min(a,b), \min(a,c)), \min(b,c))$
- **Or a realistic one:**

```plaintext
equalized1 = input1 - equalization1;
df1 = equalized1 - output;
st1 = if (df1 > 0.5) then 0.5 else (if (df1 < -0.5) then -0.5 else df1);
equalization1 = 0.0 -> pre (equalization1) + (pre (st1) - pre (satCentering)) * 0.05;

equalized2 = input2 - equalization2;
df2 = equalized2 - output;
st2 = if (df2 > 0.5) then 0.5 else (if (df2 < -0.5) then -0.5 else df2);
equalization2 = 0.0 -> pre (equalization2) + (pre (st2) - pre (satCentering)) * 0.05;

equalized3 = input3 - equalization3;
df3 = equalized3 - output;
st3 = if (df3 > 0.5) then 0.5 else (if (df3 < -0.5) then -0.5 else df3);
equalization3 = 0.0 -> pre (equalization3) + (pre (st3) - pre (satCentering)) * 0.05;

c1 = equalized1 > equalized2;
c2 = equalized2 > equalized3;
c3 = equalized3 > equalized1;

output = if (c1 = c2) then equalized2 else (if (c2 = c3) then equalized3 else equalized1);

d1 = equalization1 > equalization2;
d2 = equalization2 > equalization3;
d3 = equalization3 > equalization1;

centering = if (d1 = d2) then equalization2 else (if (d2 = d3) then equalization3 else equalization1);
satCentering = if (centering > 0.25) then 0.25 else (if (centering < -0.25) then -0.25 else centering);
```

this is the simple version without alarms
Current setting – Dos/Don’ts

Our objective:

- consider software to be validated, ie. certified under DO-178C.
- we have Code + Specification
- need to validate the code wrt specification
- using formal methods
**CURRENT setting – DOS/DON’ts**

Our objective:
- consider software to be validated, ie. certified under DO-178C.
- we have Code + Specification
- need to validate the code wrt specification
- using formal methods

Weakness of the current industrial process:
- models are not considered (yet) as code
  - even if code is fully generated from models
- most model level properties are not under DO-178C scope: they do not concern software only
Current Setting – Dos/Don’ts

Our objective:

- consider software to be validated, i.e. certified under DO-178C.
- we have Code + Specification
- need to validate the code wrt specification
- using formal methods

Weakness of the current industrial process:

- models are not considered (yet) as code
  - even if code is fully generated from models
- most model level properties are not under DO-178C scope: they do not concern software only

In the following:

- System and controller properties
- Discrete models/code: No continuous nor hybrid models
- Formal verification methods and tools
Contents

Context

Background
  Formal analysis of software model/code
  Lyapunov functions and SDP solvers

Application to controller formal analysis
  Hoare style annotations
  Lyapunov function synthesis and Policy Iteration

Soundness of the analysis

Current activities: more complex templates/properties

Conclusion
Contents

Context

Background
  Formal analysis of software model/code
    Lyapunov functions and SDP solvers

Application to controller formal analysis
  Hoare style annotations
    Lyapunov function synthesis and Policy Iteration

Soundness of the analysis

Current activities: more complex templates/properties

Conclusion
Object under analysis – The Input

System:

- code
  - set of functions, sequence of instructions, mix of boolean conditions, integer counters, floating point computations, pointers
  - no dynamic allocation (malloc), no nested loops
- models
  - similar notions but simpler: no pointers, more types,
  - we can assume knowledge is provided on model components: this is a linear controller, an anti-windup, a saturation, etc

Property:

axiomatic semantics, aka predicate over values
  - all reachable values are bounded
  - a given bad region is unreachable
  - high level properties: overshoot bounded
  - …
**How to verify safety properties:**

Let such a discrete system be defined as

- set of states $\Sigma$
- initial states $Init \subseteq \wp(\Sigma)$
- dynamics: $Step \subseteq \wp(\Sigma \times \Sigma)$

Almost all analyses are based on “induction”

- SMT-based model checking
  - encode the system semantics $S$ and property $P$ as logical predicates
  - check the inductiveness $P$ wrt $S$ through calls to SMT solvers
  - if $\neg expr$ is unsat then $expr$ is always true

- Deductive methods
  - Express the intended axiomatics semantics (Pre/Post) over imperative code
  - Mecanization of predicate transformation: weakest precondition computation
  - Prove that each construct soundly transforms predicates
  - For loops: amounts to compute (inductive) loop invariants

- Static analysis
  - compute an inductive over-approximation of reachable states
  - in a specific abstraction
Weakest Precondition: \( WP(code, \text{Post}) \)

weakest precondition that guarantees to have Post after running code

Proving contract: 
\[ Pre \implies WP(code, \text{Post}) \]

Use SMT-solvers, or proof assistant

**Deductive methods main idea – by Floyd**

Figure 5. Algorithm to compute quotient \( Q \) and remainder \( R \) of \( X + Y \), for integers \( X \geq 0, Y > 0 \)
Classical abstract fixpoint computation: Kleene algorithm

Reachable states $\mathcal{R}$ are defined as the least fixpoint of the function

$$\mathcal{R} = \text{lfp}_\perp f$$

where $f : X \mapsto \{ s' \in \wp(\Sigma) | s' \in \text{Init} \lor \exists s \in X, (s, s') \in \text{Step} \}$

Instead of computing $\mathcal{R}$ we compute $\mathcal{R}^\#$ such that $\gamma(\mathcal{R}^\#) \supseteq \mathcal{R}$ and

$$\mathcal{R}^\# = \text{lfp}_\perp f^\#$$

where $f^\# : X \mapsto \alpha(\text{Init}) \sqcup^\#
\exists s' \in \Sigma, \exists s \in \gamma(\mathcal{R}^\#), \text{Step}(s, s')$

where an abstract domain is defined by

- $\langle D, \sqsubseteq^\# \rangle$ a partially ordered set of abstract elements
- $\sqcup^#$ a join operator
- $\alpha : \wp(\Sigma) \to D$ an abstraction function
- $\gamma : D \to \wp(\Sigma)$ a concretization function

Eg. interval abstraction, convex polyhedra, etc
**Example**

\[x = \text{rand}(0, 12); \quad y = 42;\]

**while** \(x > 0\) {
\[x = x - 2;\]
\[y = y + 4;\]
}

\[x = \text{rand}(0, 12)\] \[x \leq 0\] \[x > 0\] \[y = 42\]
**Example**

\[ x = \text{rand}(0, 12); \]
\[ y = 42; \]

\[ \text{while} \ (x > 0) \{ \]
\[ x = x - 2; \]
\[ y = y + 4; \]
\[ \} \]

\[ x = \text{rand}(0, 12) \]
\[ y = 42 \]

\[ x \leq 0 \]

\[ x > 0 \]

\[ x = x - 2 \]
\[ y = y + 4 \]
**Example**

```plaintext
0x = rand(0, 12);  
y = 42;
while (x > 0) {
    x = x - 2;
    y = y + 4;
}
```
Example

\[
\begin{align*}
0x & = \text{rand}(0, 12); \\
y & = 42; \\
\textbf{while} & \ (x > 0) \ {\{} \\
\ & \ x = x - 2; \\
\ & \ y = y + 4; \\
\} \\
\end{align*}
\]
**Example**

\[0 x = \text{rand}(0, 12); 1 y = 42;\]

\[\textbf{while } x > 0 \textbf{ } \{\]
\[3 x = x - 2;\]
\[4 y = y + 4;\]
\[\} 5\]

The diagram illustrates the flow of the algorithm with the following steps:

1. Initialize \(x = \text{rand}(0, 12)\) and \(y = 42\).
2. While \(x > 0\): subtract 2 from \(x\) and add 4 to \(y\).
3. If \(x \leq 0\), return.
4. Update \(x = x - 2\) and \(y = y + 4\).
5. Repeat from step 2.

The diagram also shows the range of \(x\) and \(y\) values with axis labels and a blue shaded area representing the range of \(x\) values from 0 to 12.
EXAMPLE

0 \( x = \text{rand}(0, 12); \) 1 \( y = 42; \)

\textbf{while} 2 \((x > 0)\) {
    
    3 \( x = x - 2; \)
    
    4 \( y = y + 4; \)

\}

\( x = \text{rand}(0, 12) \)
\( y = 42 \)
\( x \leq 0 \)
\( x > 0 \)
Example cont’d

\(x = \text{rand}(0, 12); y = 42;\)

\[\text{while } (x > 0) \{
\begin{align*}
3x &= x - 2; \\
4y &= y + 4;
\end{align*}\]

\}\]
**Example cont’d**

\[\begin{align*}
0 & : x = \text{rand}(0, 12); \\
1 & : y = 42; \\
\textbf{while} & : (x > 0) \\
2 & : \quad \left\{ \begin{array}{l}
3 & : x = x - 2; \\
4 & : y = y + 4;
\end{array} \right\
\end{align*}\]
EXAMPLE CONT’D

\[ x = \text{rand}(0, 12); \]
\[ y = 42; \]

\begin{align*}
\text{while } & (x > 0) \{ \\
& x = x - 2; \\
& y = y + 4;
\}
\end{align*}
EXAMPLE CONT’D

\[ x = \text{rand}(0, 12); \quad y = 42; \]

while \((x > 0)\) {
    \[ x = x - 2; \]
    \[ y = y + 4; \]
}

\[ x = \text{rand}(0, 12) \]
\[ y = 42 \]
\[ x \leq 0 \]
\[ x > 0 \]
Example cont’d

\[ x = \text{rand}(0, 12); \]
\[ y = 42; \]

while \( x > 0 \) {
\[ x = x - 2; \]
\[ y = y + 4; \]
}

\[ x = \text{rand}(0, 12) \]
\[ y = 42 \]
\[ x \leq 0 \]
Example cont’d

\[ 0x = \text{rand}(0, 12); 1y = 42; \]
\[
\text{while } 2(x > 0) \{
3x = x - 2;
4y = y + 4;
\}
\]

Diagram:

\[ \text{while } x > 0 \{
\text{if } x > 0 \rightarrow \text{next state}
\text{if } x \leq 0 \rightarrow \text{next state}
\]
Example cont’d

\[ x = \text{rand}(0, 12); \]
\[ y = 42; \]

\[ \textbf{while } x > 0 \{ \]
\[ x = x - 2; \]
\[ y = y + 4; \]
\[ \} \]

\[ x = \text{rand}(0, 12); \]
\[ y = 42; \]

\[ x > 0 \]
\[ x \leq 0 \]
**Example cont’d**

\[
0x = \text{rand}(0, 12); 1y = 42;
\]

while \(x > 0\) {
\[
3x = x - 2; 4y = y + 4;
\]
}

\[
x = \text{rand}(0, 12) \quad y = 42 \quad x > 0 \quad y = y + 4 \quad x \leq 0
\]
Example cont’d

\[ 0x = \text{rand}(0, 12); \] \[ 1y = 42; \]

\textbf{while} \( x > 0 \) {
\[ 2x = x - 2; \]
\[ 3y = y + 4; \]
\}
Example cont’d

\[ \begin{align*}
0x &= \text{rand}(0, 12); \\
1y &= 42; \\
\text{while} \ 2(x > 0) \ { \begin{align*}
3x &= x - 2; \\
4y &= y + 4;
\end{align*} } \\
\} \\
\end{align*} \]

\[
\begin{array}{cccc}
0 & \xrightarrow{x = \text{rand}(0, 12)} & 1 & \xrightarrow{y = 42} \\
& & 2 & \xrightarrow{x \leq 0} \\
& & 3 & \xrightarrow{x = x - 2} \\
& & 4 & \xrightarrow{y = y + 4}
\end{array}
\]
**Example cont’d**

\[ x = \text{rand}(0, 12); \]
\[ y = 42; \]

while \( x > 0 \) {
  \[ x = x - 2; \]
  \[ y = y + 4; \]
}\n
\*fixpoint reached*
**Example cont’d**

\[
x_0 = \text{rand}(0, 12);  
y_1 = 42;
\]

**while** \( x > 0 \) \{ 
\[
x_3 = x - 2;
\]
\[
y_4 = y + 4;
\]
\}

\[
\text{fixpoint reached}
\]
Example cont’d

0 \( x = \text{rand}(0, 12); \)

1 \( y = 42; \)

\[ \text{while } x > 0 \{ \]

2 \( x = x - 2; \)

3 \( y = y + 4; \)

\} \]

4 \( x = x - 2 \)

5 \( x \leq 0 \)
EXAMPLE CONT’D

\[ x = \text{rand}(0, 12); \]
\[ y = 42; \]
\[ \text{while } (x > 0) \{ \]
\[ x = x - 2; \]
\[ y = y + 4; \]
\[ \} \]

Diagram:

- Node 0: \( x = \text{rand}(0, 12) \)
- Node 1: \( y = 42 \)
- Node 2: \( x > 0 \)
- Node 3: \( x = x - 2 \)
- Node 4: \( y = y + 4 \)
- Node 5: \( x \leq 0 \)
Example cont’d

\[ x = \text{rand}(0, 12); \]
\[ y = 42; \]

\[
\text{while } x > 0 \{ \\
\quad x = x - 2; \\
\quad y = y + 4; \\
\}
\]
EXEMPLARY CONT’D

\[ x = \text{rand}(0, 12); \]
\[ y = 42; \]

\textbf{while} \( x > 0 \) \{ \\
\[ x = x - 2; \]
\[ y = y + 4; \]
\}

\begin{align*}
0 \quad & \quad 1 \quad & \quad 2 \quad & \quad 3 \quad & \quad 4 \quad & \quad 5 \\
x = \text{rand}(0, 12) \quad & \quad x = x - 2 \quad & \quad x > 0 \quad & \quad \cdots \quad & \quad y = y + 4 \quad & \quad x \leq 0
\end{align*}
EXAMPLE CONT’D

\[ x = \text{rand}(0, 12); \quad y = 42; \]

while \((x > 0)\) {
\[ x = x - 2; \]
\[ y = y + 4; \]
}\n
0 \[ x = \text{rand}(0, 12) \]
1 \[ y = 42 \]
2 \[ x \leq 0 \]
3 \[ x > 0 \]
4 \[ x = x - 2 \]
5 \[ x \leq 0 \]
CONTENTS

Context

Background
  Formal analysis of software model/code
  Lyapunov functions and SDP solvers

Application to controller formal analysis
  Hoare style annotations
  Lyapunov function synthesis and Policy Iteration

Soundness of the analysis

Current activities: more complex templates/properties

Conclusion
Quadratic Lyapunov function for linear systems

Let $A$ be a square matrix. Define the linear system:

$$x^{k+1} = Ax^k, \ k \geq 0, \ \text{a given } x^0$$

A matrix $P$ satisfies Lyapunov conditions for the system iff:

$$P - \text{Id} \succeq 0, \quad P - A^T PA \succ 0, \quad \text{(L)}$$

- Id is the identity matrix;
- $M \succ 0$ means $M = M^T$ and $\forall x \neq 0, x^T M x > 0$;
- $M \succeq 0$ means $M = M^T$ and $\forall x, x^T M x \geq 0$.

$P - \text{Id} \succeq 0$ implies the boundedness:

$$\|x\|_2^2 \leq \beta$$

$x^T P x \leq \alpha$

$P - A^T PA \succ 0$ guarantees the strict decrease:

$$x^T P x \leq \alpha$$

$$x^T A^T P A x \leq \alpha$$
**QUAD. LYAPUNOV FCN FOR LINEAR SYSTEMS**

Let $A$ be a square matrix. Define the linear system:

$$x^{k+1} = Ax^k, k \geq 0, \text{ a given } x^0$$

A matrix $P$ satisfies Lyapunov conditions for the system iff:

$$P - \text{Id} \succeq 0, \quad P - A^T PA \succ 0 \quad \text{(L)}$$

- $\text{Id}$ is the identity matrix;
- $M \succ 0$ means $M = M^T$ and $\forall x \neq 0, x^T M x > 0$;
- $M \succeq 0$ means $M = M^T$ and $\forall x, x^T M x \geq 0$.

$P - \text{Id} \succeq 0$ is equivalent to:

$$\forall \alpha \geq 0, \exists \beta \geq 0 \text{ s.t. } \left( \begin{array}{cc} -\alpha & 0 \\ 0 & P \end{array} \right) - \left( \begin{array}{cc} -\beta & 0 \\ 0 & \text{Id} \end{array} \right) \succeq 0$$

$P - A^T PA \succ 0$ is equivalent to:

$$\forall \alpha \geq 0$$

$$\left( \begin{array}{cc} -\alpha & 0 \\ 0 & P \end{array} \right) - \left( \begin{array}{cc} -\alpha & 0 \\ 0 & A^T PA \end{array} \right) \succ 0$$
Convex optimization

Solver for LMI: Linear Matrix Inequalities.

Typically an implementation of a Primal/Dual algorithm.

Rely on notion of (topological) duality:

The SDP cone is self-dual:

Dual problem (LMI):
\[
\inf_{p,X} \langle b, p \rangle \\
\text{subject to} \quad F_0 + \sum_{i=1}^{m} p_i F_i + X = 0 \\
X \succeq 0
\]

Primal problem:
\[
\sup_{Z} \langle F_0, Z \rangle \\
\text{subject to} \quad \langle F_i, Z \rangle + b_i = 0, \ i = 1, \ldots, m \\
Z \succeq 0
\]

Available tools: CSDP, SDPA, VSDP, Sedumi (old), Mosek, SDPT3, mainly within Matlab
GENERAL PRINCIPLES OF THE ALGORITHM

Main principles:
▶ start from a (feasible) solution
▶ while not reached a duality gap of width $\epsilon$ (precision)
  ▶ try to reach the minimally of a cost function combining the duality gap and a barrier function
  ▶ compute the zeros of the cost function (KKT conditions)
  ▶ linearize (compute a first order Taylor approximation)
  ▶ compute the next iterate in the (linear) direction obtained

The algorithm relies on basic linear algebra:
▶ computation of trace, product, transpose
▶ resolution of a linear system: eg. using Choleski / Singular Value Decomposition / QR factorization
CONTENTS

Context

Background
  Formal analysis of software model/code
  Lyapunov functions and SDP solvers

Application to controller formal analysis
  Hoare style annotations
  Lyapunov function synthesis and Policy Iteration

Soundness of the analysis

Current activities: more complex templates/properties

Conclusion
CONTENTS

Context

Background
   Formal analysis of software model/code
   Lyapunov functions and SDP solvers

Application to controller formal analysis
   Hoare style annotations
   Lyapunov function synthesis and Policy Iteration

Soundness of the analysis

Current activities: more complex templates/properties

Conclusion
AUTOMATIC ANNOTATION OF AN IMPERATIVE CODE

- Input: Simulink diagram + Lyapunov function
- Consider the Lyapunov function as a loop invariant
- Perform strongest-postcondition: carry LF over the code

In practice:
- Since LF $P$ is invertible, manipulate $Q = P^{-1}$
- a sublevel set is described by a specific predicate: $q(Q, x, 1) : x^tQ^{-1}x \leq 1$
- Image of $q(Q, x, 1)$ by $T$ is $q(TQT^\top, x, 1)$
- Implication btw ellipsoids is checked using Choleski decomposition
Exemple d’annotations

```c
while (1) {
    /*@
    requires q(Q,x,1);
    ensures q(T1*Q*T1',x1,1);
    */
    { y1=0.4990*x1+0.1*x2; }
    /*@
    requires q(Q,x,1);
    ensures q(T2*T1*Q*T1'*T2',x2,1);
    */
    { y2=0.01*x1+1.0*x2; }
    /*@
    requires q(Q,x,1);
    ensures q(T3*T2*T1*Q*T1'*T2'*T3',x3,1);
    */
    { x1=y1; }
    /*@
    requires q(Q,x,1);
    ensures q(T4*T3*T2*T1*Q*T1'*T2'*T3'*T4',x,1);
    */
    { x2=y2; }
}
```

Proof wise: use of Frama-C/PVS
- each Hoare triple is discharged by Why3/PVS
- implication between ellipsoids is managed outside PVS (Choleski)
CONTENTS

Context

Background
   Formal analysis of software model/code
   Lyapunov functions and SDP solvers

Application to controller formal analysis
   Hoare style annotations
       Lyapunov function synthesis and Policy Iteration

Soundness of the analysis

Current activities: more complex templates/properties

Conclusion
**Static analysis of discrete controllers**

**Linear controllers and their properties**

- **Linear invariants** commonly used in static analysis are not well suited:
  - at best costly;
  - at worst no result.

Control theorists have known for a long time that quadratic invariants are a good fit for linear systems.

Characterizing a small stable ellipsoid for a linear system:
- using SDP to optimize a Lyapunov function (shape)
- minimize a scalar to fit the input (ratio)
- different LMI heuristics
  - minimize condition number
  - preserve shape
  - consider inputs

Not really suited for Kleene iterations.
Linear invariants commonly used in static analysis are not well suited:
  ▶ at best costly;
  ▶ at worst no result.

Control theorists have known for a long time that quadratic invariants are a good fit for linear systems.

Characterizing a small stable ellipsoid for a linear system:
  ▶ using SDP to optimize a Lyapunov function (shape)
  ▶ minimize a scalar to fit the input (ratio)
  ▶ different LMI heuristics
  ▶ minimize condition number
  ▶ preserve shape
  ▶ consider inputs

Not really suited for Kleene iterations.
Static analysis of discrete controllers

Linear controllers and their properties

- Linear invariants commonly used in static analysis are not well suited:
  - at best costly;
  - at worst no result.

- Control theorists have known for a long time that quadratic invariants are a good fit for linear systems.
Linear invariants commonly used in static analysis are not well suited:
- at best costly;
- at worst no result.

Control theorists have known for a long time that quadratic invariants are a good fit for linear systems.
Static analysis of discrete controllers

Linear controllers and their properties

- **Linear invariants** commonly used in static analysis are not well suited:
  - at best costly;
  - at worst no result.

- Control theorists have known for a long time that **quadratic invariants** are a good fit for linear systems.

Characterizing a small stable ellipsoid for a linear system:

- using SDP to optimize a Lyapunov function (shape)
- minimize a scalar to fit the input (ratio)
- different LMI heuristics
  - minimize condition number
  - preserve shape
  - consider inputs
- Not really suited for Kleene iterations
A first step preprocesses the system to support the analysis:

- extraction of the control flow graph;
- reduction of the linear system: avoid redundancy and therefore singular matrices
- example of computation of a close loop representation;

\[
\begin{align*}
\text{true }, \quad x_c & := 0 \\
x_c & := 0 \\
x_p & := 0 \\
y_c & := 0 \\
u & := 0 \\
\end{align*}
\]
Bounding templates: Policy Iterations (Min-Policies)

Once a global description is extracted, bound the templates on all updates.

- Iterates *downward* from a postfixpoint (like a narrowing).
- Comparable to Newton-Raphson method:

\[
\begin{align*}
    b_{1,1} \\
    b_{1,1} \\
    b_{1,1}
\end{align*}
\]

In practice, does not reduce the Lyapunov template bound but enables the computation of bounds on other templates, e.g. \( x_i^2 \)
Bounding templates: Policy Iterations (Min-Policies)

Once a global description is extracted, bound the templates on all updates.

- Iterates downward from a post-fixpoint (like a narrowing).
- Comparable to Newton-Raphson method:

\[ b_{1,1}, \sigma_{1} \]

\[ b_{1,1}, \sigma_{2} \]

Policy computation (computing \( \sigma_{i} \)): many local SDP problems.

Policy solving (computing \( b_{i} \)): one global LP problem.

In practice, does not reduce the Lyapunov template bound but enable the computation of bounds on other templates, eg. \( x_{i}^{2} \)
Bounding templates: Policy Iterations (Min-Policies)

Once a global description is extracted, bound the templates on all updates.

- Iterates **downward** from a postfixpoint (like a narrowing).
- Comparable to Newton-Raphson method:

\[ b_{1,1}, b_{1,1} \]

\[ \sigma_1, b_0 \]

In practice, does not reduce the Lyapunov template bound but enable the computation of bounds on other templates, eg. \( x_i^2 \)
Bounding templates: Policy Iterations (Min-Policies)

Once a global description is extracted, bound the templates on all updates.

- Iterates downward from a postfixpoint (like a narrowing).
- Comparable to Newton-Raphson method:

\[ b_1, 1 \\ b_0 \]

Policy computation (computing \( \sigma_i \)): many local SDP problems.

Policy solving (computing \( b_i \)): one global LP problem.

In practice, does not reduce the Lyapunov template bound but enable the computation of bounds on other templates, eg. \( x_i^2 \).
Bounding templates: Policy Iterations (Min-Policies)

Once a global description is extracted, bound the templates on all updates.

- Iterates downward from a postfixpoint (like a narrowing).
- Comparable to Newton-Raphson method:

In practice, does not reduce the Lyapunov template bound but enable the computation of bounds on other templates, eg. $x_i^2$. 
Bounding templates: Policy Iterations (Min-Policies)

Once a global description is extracted, bound the templates on all updates.

- Iterates \underline{downward} from a postfixpoint (like a narrowing).
- Comparable to Newton-Raphson method:

\[ b_1,1, \quad \sigma_2, \quad \sigma_1 \]

\[ b_0 \quad b_1,1 \]

\[ b_2 \quad b_1 \]

In practice, does not reduce the Lyapunov template bound but enable the computation of bounds on other templates, eg. \( x_i^2 \)
Bounding templates: Policy Iterations (Min-Policies)

Once a global description is extracted, bound the templates on all updates.

- Iterates downward from a postfixpoint (like a narrowing).
- Comparable to Newton-Raphson method:

\[
\begin{align*}
\sigma_1 &> b_{1,1} \\
\sigma_2 &> b_{2,1} \\
\sigma_3 &> b_{3,1} \\
\vdots &> b_{n,1}
\end{align*}
\]

- Policy computation (computing \(\sigma_i\)): many local SDP problems.

In practice, does not reduce the Lyapunov template bound but enable the computation of bounds on other templates, eg. \(x_i^2\).
Bounding templates: Policy Iterations (Min-Policies)

Once a global description is extracted, bound the templates on all updates.

- Iterates downward from a postfixpoint (like a narrowing).
- Comparable to Newton-Raphson method:

\[
\begin{align*}
& b_0 \\
& \sigma_1 \\
& \sigma_2 \\
& b_{1,1} \\
& b_1 \\
& b_2
\end{align*}
\]

- Policy computation (computing \(\sigma_i\)): many local SDP problems.
- Policy solving (computing \(b_i\)): one global LP problem.

In practice, does not reduce the Lyapunov template bound but enable the computation of bounds on other templates, eg. \(x_i^2\).
Automatic analysis

Input: synchronous model (in Lustre)

```plaintext
node top(in0 : real) returns (x0, x1, x2 : real);
let
    assert (in0 >= -1. and in0 <= 1.);
    x0 = 0. -> 0.9379 * pre x0 - 0.0381 * pre x1 - 0.0414 * pre x2 + 0.0237 * in0;
    x1 = 0. -> -0.0404 * pre x0 + 0.968 * pre x1 - 0.0179 * pre x2 + 0.0143 * in0;
    x2 = 0. -> 0.0142 * pre x0 - 0.0197 * pre x1 + 0.9823 * pre x2 + 0.0077 * in0;
```

tel

Output: invariant on reachable states

\[6.2547x_0^2 + 12.1868x_1^2 + 3.8775x_2^2 - 10.61x_0x_1 - 2.4306x_1x_2 + 2.4182x_1x_2 \leq 1.0029\]

\[|x_0| \leq 0.4236 \land |x_1| \leq 0.3371 \land |x_2| \leq 0.5251.\]
CONTENTS

Context

Background
   Formal analysis of software model/code
   Lyapunov functions and SDP solvers

Application to controller formal analysis
   Hoare style annotations
   Lyapunov function synthesis and Policy Iteration

Soundness of the analysis

Current activities: more complex templates/properties

Conclusion
Is the bounded Lyapunov template a sound postfixpoint?

The computation
- of the Lyapunov template with its initial bound
- of the precise bound using PI
relied on calls to convex optimization algorithms
Is the bounded Lyapunov template a sound postfixpoint?

The computation
- of the Lyapunov template with its initial bound
- of the precise bound using PI
relied on calls to convex optimization algorithms

Questions:
- Is the Lyapunov function obtained a good template when considering floating point computation in the solvers and in the program?
- Is the bound computed safe?
Can we trust SDP solvers?

Computation of $\sup_{Z} \langle F, Z \rangle$ with $\bigwedge_{i \in [0,m]} \langle F_i, Z \rangle + b_i$ and $Z \succeq 0$

Typically solved with an implementation of an interior point algorithm:

- start from a (feasible) solution
- while not reached a duality gap of width $\epsilon$ (precision)
  - try to reach the minimally of a cost function combining the duality gap and a barrier function
  - compute the zeros of the cost function (KKT conditions)
  - linearize (compute a first order Taylor approximation)
  - compute the next iterate in the (linear) direction obtained

However this process showed to be error-prone:

- solutions provided by the solvers may not be feasible solutions
- tentative of explanation:
  - as the duality gap gets smaller, the numerical issues gets larger
  - and the dynamics of the algorithm may diverge from its real definition
**Can we trust SDP solvers?**

Computation of $\sup_Z \langle F, Z \rangle$ with $\bigwedge_{i \in [0,m]} \langle F_i, Z \rangle + b_i$ and $Z \succeq 0$

Typically solved with an implementation of an interior point algorithm:

- start from a (feasible) solution
- while not reached a duality gap of width $\epsilon$ (precision)
  - try to reach the minimally of a cost function combining the duality gap and a barrier function
  - compute the zeros of the cost function (KKT conditions)
  - linearize (compute a first order Taylor approximation)
  - compute the next iterate in the (linear) direction obtained

However this process showed to be error-prone:

- solutions provided by the solvers may not be feasible solutions
- tentative of explanation:
  - as the duality gap gets smaller, the numerical issues gets larger
  - and the dynamics of the algorithm may diverge from its real definition
We want to check the feasibility of a potential solution provided by the SDP solver:

\[ P \succeq 0 \text{ and } P - A^\top PA \succeq 0 \]
We want to check the feasibility of a potential solution provided by the SDP solver:

\[ P \succeq 0 \text{ and } P - A^TPA \succeq 0 \]

Checking the positive semi-definiteness can be done with Choleski decomposition: finding a \( Z \) such that \( Z'Z = P \).

Unfortunately, due to floating point errors, it will compute \( Z \) such that \( Z'Z = P + eId \succeq 0 \) where \( e \) can be bounded.

\[ \Rightarrow \quad M - eId \succeq 0 \] computed with a floating point implementation is a sufficient condition to obtain \( M \succeq 0 \)
SYSTEMS COMPUTATION ARE NOT PERFORMED WITH REALS

In practice the system or code dynamics is computed with floats:

\[ x_{k+1} = Ax_k + Bu_k + \text{errors} \]

We compute necessary conditions on guards and assignments to ensure safe floating point soundness:

- **in guards:** \( \text{fl}(e) \leq \text{fl}(c) \) may hold while \( e \leq c \) does not
  \[ \implies \text{compute a safer bound } c' \text{ such that } \text{fl}(e) \leq \text{fl}(c) \implies e \leq c' \]

- **in assignments** (evaluated through quadratic templates):
  \[ r(t) \leq b' \implies \text{fl}(r)(t) \leq b \]

Proofs are complex and performed in Coq, thanks to Pierre Roux, but specific to case of guards/assigns/order of evaluation, etc...

For the Hoare style annotations, template bounds are made bigger to account for floating point errors.
CONTENTS

Context

Background
  Formal analysis of software model/code
  Lyapunov functions and SDP solvers

Application to controller formal analysis
  Hoare style annotations
  Lyapunov function synthesis and Policy Iteration

Soundness of the analysis

Current activities: more complex templates/properties

Conclusion
SATURATION

Heuristic to constraints the Lyapunov function synthesis: use sector-bound.

Idea:
- the saturated part may be not strictly contracting
- but it is assumed bounded by a reasonable order of magnitude

Expressed as a LMI constraint using S-procedure
**Piecewise Quadratic Lyapunov Functions**

For stable switched linear systems, a common Lyapunov function may not exist.

Method by Morari et al, Rantzer and Johansson to compute piecewise quadratic LF.

- System defined as partition of zones: $X^i = \{c^i, T^i\}$.
- Build a set of local Lyapunov function $P^i$ such that
  - $x \in X^i, T^i(x) \in X^j, x^\top P^i x \leq 0 \implies (T^i(x))^\top P^j T^i(x) \leq 0$
  - bound variable values in each zone
  - quadratic number of constraints in the LMI wrt number of zones.
- Reducing the set of possible zone transitions is performed using Motzkin transposition theorem
Try to generalize the computation of Lyapunov function to fixed-degree polynomial:

\[
\inf_{p \in \mathbb{R}[x], w \in \mathbb{R}} \begin{cases} 
  w, \\
  p(x) \leq 0, \\
  \forall i \in \mathcal{I}, p(T^i(x)) \leq p(x), \\
  \kappa(x) \leq w + p(x), \\
\end{cases} \quad \forall x \in \text{Init}, \forall x \in X^i, \forall x \in \mathbb{R}^d. 
\]  

(1)

where \(\kappa(x)\) is a user-specified sublevel property (boundness, safe srt, avoiding bad regions, ...)

[Diagram showing polynomial and Lyapunov function]
EXTENDING POLICY ITERATION TO POLYNOMIAL SYSTEMS

Static analysis use of PI was limited to
- linear updates,
- up to quadratic guards and
- quadratic templates:

Propose an extension of policy iteration that
- considers piecewise polynomial systems
- with polynomial templates
Context

Background
  Formal analysis of software model/code
  Lyapunov functions and SDP solvers

Application to controller formal analysis
  Hoare style annotations
  Lyapunov function synthesis and Policy Iteration

Soundness of the analysis

Current activities: more complex templates/properties

Conclusion
WHERE’RE WE HEADED?

- from code to dynamical systems … and back
- more systems/properties covered
- more powerful tools/analyses to address them
- handle hybrid systems? with property over infinite horizon?
Ease the integration of policy iteration in static analysis tools
  ▶ extract the control flow graph from a code
  ▶ combine results of PI with classical Kleene computations
    ▶ domain reductions
Handle combination of code with plant
  ▶ local handling of floats, ie. in controller not in plant
  ▶ put the plant equations in the code
Systems/Properties covered

- Address the analysis of close-loop system level properties
  - robustness (margins)
  - performances
- Controllers with
  - saturations
  - anti-windup
  - linear interpolation of gain
  - MPC with embedded optimization (LP, QP or SOCP)
  - …
MORE ANALYSES

- Extend Policy Iteration to handle piecewise templates
- Piecewise polynomial templates using SOS
- Probabilistic properties
- other approaches to reachable states approximation

… applied on code analysis
Hybrid systems

No clue on how to start.
Thank you