Des réels aux flottants : préservation automatique de preuves de stabilité de Lyapunov

Olivier Hermant, Vivien Maisonneuve

14èmes journées Approches Formelles dans l’Assistance au Développement Logiciel
Bordeaux, 9 Juin 2015
Embedded Systems

An embedded system is a computer system with a dedicated function, within a larger mechanical or electrical system.

Constraints:

- Power consumption;
- Performance (RT);
- Safety;
- Cost.

Uses a low-power processor or a microcontroller.

Commonly found in consumer, cooking, industrial, automotive, medical, commercial and military applications.
Example

Quadricopter, DRONE Project, MINES ParisTech & ÉCP
⇒ Parrot AR.Drone.

ATMEGA128: 16 MHz, 4 KB RAM, 128 KB ROM
Control-Command System

```c
while (1) {
    receive(y, yd);
    u = f(y, yd);
    send(u);
}
```
Levels of Description

**Formalization:**
- System conception;
- Constraint specification;
- Physical model of the environment;
- Mathematical **proof** that the system behave properly.

MATLAB, Simulink

**Realization:** very low-level C program
- Thousands of LOC;
- Computations decomposed into elementary operations;
- Management of sensors and actuators.

GCC, Clang

Gradual **transformations**

**How to ensure that the executed program is correct?**
Stability Proof

Show that the system parameters are bounded during its execution.

Essential for system safety.

- **Open loop stability**: \( u_c \) bounded \( \Rightarrow \) \( x_c \) bounded  
  (hence \( y_c \) bounded)

- **Closed loop stability**: \( y_d \) bounded \( \Rightarrow \) \( x_c, x_p \) bounded  
  (hence \( y_c, y_p \) bounded)
Stability Invariant

Linear invariants not well suited. Quadratic invariants (ellipsoids) are a good fit for linear systems.

Lyapunov theory provides a framework to compute inductive invariants.

Static analysis to show that the invariant holds from source code.
Stability Invariant

**Linear invariants** not well suited.
**Quadratic invariants** (ellipsoids) are a good fit for linear systems.

**Lyapunov theory** provides a framework to compute inductive invariants.

Static analysis to show that the invariant holds from source code.
Numerical Precision

Lyapunov theory applies on a system with real arithmetic.

In machine implementations, numerical values are approximated by binary, limited-precision values.

- **Floating point** (IEEE 754):
  
  \[ (-1)^s \times 2^{e-127} \times m \]

- **Fixed point**:
  
  \[ (-1)^s \times e + 2^{-24} \times m \]

- **Rationals** using pairs of integers.
Numerical Precision

Lyapunov theory applies on a system with real arithmetic.

In machine implementations, numerical values are approximated by binary, limited-precision values.

1. Constant values are altered;
2. Rounding errors during computations.

⇒ Stability proof does not apply, invariant does not fit.

How to adapt the stability proof?
Example System

[Feron ICSM’10]:
mass-spring system.

Open-loop stability:
\( x_c \) bounded.

Closed-loop stability:
\( x_c, x_p \) bounded.

\[
\begin{bmatrix}
0.4990 & -0.0500 \\
0.0100 & 1.0000
\end{bmatrix}; \quad
\begin{bmatrix}
1 \\
0
\end{bmatrix}; \\
\begin{bmatrix}
564.48 \\
0
\end{bmatrix}; \\
-1280; \\
\begin{bmatrix}
0
\end{bmatrix}; \\
\begin{bmatrix}
\text{receive}(y, 2); \text{receive}(yd, 3); \\
\text{while} (1) \\
\quad yc = \max(\min(y - yd, 1), -1); \\
\quad u = Cc*xc + Dc*yc; \\
\quad xc = Ac*xc + Bc*yc; \\
\quad \text{send}(u, 1); \\
\quad \text{receive}(y, 2); \text{receive}(yd, 3); \\
\text{end}
\end{bmatrix}
\]
Example System: Stability Ellipse

Lyapunov theory \(\Rightarrow x_c = \begin{pmatrix} x_{c1} \\ x_{c2} \end{pmatrix}\) belongs to the ellipse:

\[
\mathcal{E}_P = \{x \in \mathbb{R}^2 \mid x^T \cdot P \cdot x \leq 1\} \quad P = 10^{-3} \begin{pmatrix} 0.6742 & 0.0428 \\ 0.0428 & 2.4651 \end{pmatrix}
\]

\[x_c \in \mathcal{E}_P \iff 0.6742x_{c1}^2 + 0.0856x_{c1}x_{c2} + 2.4651x_{c2}^2 \leq 1000\]
Example System

\[
A_c = \begin{bmatrix} 0.4990 & -0.0500 \\ 0.0100 & 1.0000 \end{bmatrix};
\]

\[
B_c = \begin{bmatrix} 1 \\ 0 \end{bmatrix};
\]

\[
C_c = \begin{bmatrix} 564.48 \\ 0 \end{bmatrix};
\]

\[
D_c = -1280;
\]

\[
xc = \text{zeros}(2, 1);
\]

\[
\text{receive}(y, 2); \text{receive}(yd, 3);
\]

while (1)

\%
\[
x_c \in \mathcal{E}_P
\]

\[
y_c = \max(\min(y - yd, 1), -1);
\]

\[
u = C_c*xc + D_c*y_c;
\]

\[
xc = A_c*xc + B_c*y_c;
\]

\[
\text{send}(u, 1);
\]

\[
\text{receive}(y, 2); \text{receive}(yd, 3);
\]

\%
\[
x_c \in \mathcal{E}_R \subset \mathcal{E}_P
\]

end
Example System

\[ Ac = \begin{bmatrix} 0.4990 & -0.0500 \\ 0.0100 & 1.0000 \end{bmatrix}; \]
\[ Bc = [1; 0]; \]
\[ Cc = [564.48, 0]; \]
\[ Dc = -1280; \]
\[ xc = \text{zeros}(2, 1); \]
\[ \text{receive}(y, 2); \text{receive}(yd, 3); \]
\[ \text{while} \ (1) \]
  \[ \% \ x_c \in \mathcal{E}_P \]
  \[ yc = \max(\min(y - yd, 1), -1); \]
  \[ u = Cc*xc + Dc*yc; \]
  \[ xc = Ac*xc + Bc*yc; \]
  \[ \text{send}(u, 1); \]
  \[ \text{receive}(y, 2); \text{receive}(yd, 3); \]
  \[ \% \ x_c \in \mathcal{E}_P \]
\[ \text{end} \]

Using limited-precision arithmetic:
Example System

\[
\begin{align*}
A_c &= \begin{bmatrix} 0.4990, & -0.0500; \\
          0.0100, & 1.0000 \end{bmatrix}; \\
B_c &= [1; 0]; \\
C_c &= [564.48, 0]; \\
D_c &= -1280; \\
xc &= \text{zeros}(2, 1);
\end{align*}
\]

\text{receive}(y, 2); \text{receive}(yd, 3);

\text{while (1)}

\quad \% \ x_c \in \mathcal{E}_P \\
yc &= \max(\min(y - yd, 1), -1); \\
u &= C_c*xc + D_c*yc; \\
xc &= A_c*xc + B_c*yc; \\
send(u, 1); \\
\text{receive}(y, 2); \text{receive}(yd, 3); \\
\quad \% \ x_c \in \mathcal{E}_P \\
\text{end}

Using limited-precision arithmetic:

1. Constant values are altered.
Example System

\[
\begin{bmatrix}
0.4990 & -0.0500 \\
0.0100 & 1.0000
\end{bmatrix};
\]

\[
\begin{bmatrix}
1 \\
0
\end{bmatrix};
\]

\[
\begin{bmatrix}
564.48 \\
0
\end{bmatrix};
\]

\[-1280;
\]

\[
x_c = \text{zeros}(2, 1);
\]

\[
\text{receive}(y, 2); \text{receive}(yd, 3);
\]

\[
\text{while (1)}
\]

\[
\% \ x_c \in \mathcal{E}_P
\]

\[
y_c = \max(\min(y - yd, 1), -1);
\]

\[
u = Cc\times xc + Dc\times yc;
\]

\[
xc = Ac\times xc + Bc\times yc;
\]

\[
\text{send}(u, 1);
\]

\[
\text{receive}(y, 2); \text{receive}(yd, 3);
\]

\[
\% \ x_c \in \mathcal{E}_P
\]

\[
\text{end}
\]

Using limited-precision arithmetic:

1. Constant values are altered
   \[\Rightarrow \mathcal{E}_P\] no longer valid;
Example System

\[
\begin{align*}
\mathbf{A}_c &= \begin{bmatrix}
0.4990 & -0.0500 \\
0.0100 & 1.0000
\end{bmatrix}; \\
\mathbf{B}_c &= \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \\
\mathbf{C}_c &= \begin{bmatrix} 564.48 \\ 0 \end{bmatrix}; \\
\mathbf{D}_c &= -1280; \\
\mathbf{x}_c &= \text{zeros}(2, 1);
\end{align*}
\]

receive(y, 2); receive(yd, 3);

while (1)
  \% \mathbf{x}_c \in \mathcal{E}_P
  \mathbf{y}_c = \max(\min(\mathbf{y} - \mathbf{y}_d, 1), -1);
  \mathbf{u} = \mathbf{C}_c \mathbf{x}_c + \mathbf{D}_c \mathbf{y}_c;
  \mathbf{x}_c = \mathbf{A}_c \mathbf{x}_c + \mathbf{B}_c \mathbf{y}_c;
  \text{send}(\mathbf{u}, 1);
  \text{receive}(\mathbf{y}, 2); \text{receive}(\mathbf{y}_d, 3);
  \% \mathbf{x}_c \in \mathcal{E}_P
end

Using limited-precision arithmetic:

\begin{enumerate}
\item Constant values are altered \(\Longrightarrow \mathcal{E}_P\) no longer valid;
\item Rounding errors during computations.
\end{enumerate}
Example System

\[
\begin{bmatrix}
0.4990 & -0.0500 \\
0.0100 & 1.0000
\end{bmatrix}
\]

\[
[1; 0]
\]

\[
[564.48, 0]
\]

\[-1280\]

\[
\text{zeros}(2, 1)
\]

\[
\text{receive}(y, 2); \text{receive}(yd, 3);
\]

\[
\text{while (1)}
\]

\%
\[
\text{xe} \in \mathcal{E}_P
\]

\[
yc = \max(\min(y - yd, 1), -1);
\]

\[
u = Cc*xc + Dc*yc;
\]

\[
\text{send}(u, 1);
\]

\[
\text{receive}(y, 2); \text{receive}(yd, 3);
\]

\%
\[
\text{xe} \in \mathcal{E}_P
\]

\[
\text{end}
\]

Using limited-precision arithmetic:

1. Constant values are altered \( \Rightarrow \mathcal{E}_P \) no longer valid;
2. Rounding errors during computations.

Adapt invariants.
Example System: Invariants

\[ \text{xc} = \text{zeros}(2, 1); \]
% \( x_c \in \mathcal{E}_P \)
\text{receive}(y, 2); \text{receive}(yd, 3);
% \( x_c \in \mathcal{E}_P \)
\textbf{while} (1)
% \( x_c \in \mathcal{E}_P \)
\text{yc} = \text{max}(\text{min}(y - yd, 1), -1);
% \( x_c \in \mathcal{E}_P, \ y_c^2 \leq 1 \)
% \( (x_c \ y_c) \in \mathcal{E}_{Q_\mu}, \ Q_\mu = (\mu P \ 0 \ 0 \ 1 - \mu), \ \mu = 0.9991 \)
\text{u} = \text{Cc}*\text{xc} + \text{Dc}*\text{yc};
% \( (x_c \ y_c) \in \mathcal{E}_{Q_\mu} \)
\text{xc} = \text{Ac}*\text{xc} + \text{Bc}*\text{yc};
% \( x_c \in \mathcal{E}_R, \ R = [(A_c \ B_c)Q_\mu^{-1}(A_c \ B_c)^T]^{-1} \)
\text{send}(\text{u}, 1);
% \( x_c \in \mathcal{E}_R \)
\text{receive}(y, 2); \text{receive}(yd, 3);
% \( x_c \in \mathcal{E}_R \)
% \( x_c \in \mathcal{E}_P \)
\textbf{end}
Example System: Invariants

% $x_c \in \mathcal{E}_P$, $y_c^2 \leq 1$
% $(x_c \ y_c) \in \mathcal{E}_{Q\mu}$, $Q_\mu = \begin{pmatrix} \mu P & 0 \\ 0 & 1-\mu \end{pmatrix}$, $\mu = 0.9991$

\[ y_c \]

\[ \mathcal{E}_{Q\mu} \]

\[ \mathcal{E}_P \]

\[ x_{c_2} \]

\[ x_{c_1} \]

% $(x_c \ y_c) \in \mathcal{E}_{Q\mu}$

\[ x_c = A_c x_c + B_c y_c; \]

% $x_c \in \mathcal{E}_R$, $R = [(A_c \ B_c)Q_\mu^{-1}(A_c \ B_c)^T]^{-1}$
Theoretical Framework

Transpose code + invariants in two steps:

Real

\%
\text{d}
\i
\i
\%
\text{d}' = \theta(d, i)
Theoretical Framework

Transpose code + invariants in two steps:

Real

\[
\begin{align*}
\% d \\
i \\
% d' &= \theta(d, i)
\end{align*}
\]

Intermediate

\[
\begin{align*}
\% \tilde{d} \\
\tilde{i} \\
% \tilde{d}' &= \theta(\tilde{d}, \tilde{i})
\end{align*}
\]

**Code**: constants converted into machine numbers

**Invariants** recomputed using the same propagation theorem \(\theta\)
Example System, 32-bit Floating-Point Numbers

\[ Ac = \begin{bmatrix} 0.4990, & -0.0500; \\ 0.0100, & 1.0000 \end{bmatrix}; \]
\[ Bc = [1; 0]; \]
\[ Cc = [564.48, 0]; \]
\[ Dc = -1280; \]
\[ xc = \text{zeros}(2, 1); \]
...

1. Convert constants:

\[ Acf = \begin{bmatrix} 0.498999999999999911182158029987476766109466552734375, \\ -0.050000000000000000000000277555756156289135105907917022705078125, \\ 0.01000000000000000000000020816681711721685132943093776702880859375, \\ 1.0000 \end{bmatrix} \]
\[ Bcf = [1; 0]; \]
\[ Ccf = [564.48000000000000001818989403545856475830078125, 0] \]
\[ Dcf = -1280 \]
Example System, 32-bit Floating-Point Numbers

```
x_c = zeros(2, 1);
% x_c ∈ E_P
receive(y, 2); receive(yd, 3);
% x_c ∈ E_P
while (1)
  % x_c ∈ E_P
  y_c = max(min(y - yd, 1), -1);
  % x_c ∈ E_P, y_c^2 ≤ 1
  % (x_c y_c) ∈ E_Q_µ, Q_µ = (µP 0) 0 1 -µ
  u = Cc*x_c + Dc*y_c;
  % (x_c y_c) ∈ E_Q_µ
  x_c = Ac*x_c + Bc*y_c;
  % x_c ∈ E_R, R = [(A_c B_c)Q_µ^{-1}(A_c B_c)^T]^{-1}
  send(u, 1);
  % x_c ∈ E_R
  receive(y, 2); receive(yd, 3);
  % x_c ∈ E_R
  % x_c ∈ E_P
end
```

In the rest of the code:

• A_c, B_c replaced by A_c f, B_c f;
• R depends on A_c, B_c, replaced by S;
• Check if E_S ⊂ E_P.
Example System, 32-bit Floating-Point Numbers

\[ \mathbf{x}_c = \text{zeros}(2, 1); \]
\[ \% \mathbf{x}_c \in \mathcal{E}_P \]
receive(y, 2); receive(yd, 3);
\[ \% \mathbf{x}_c \in \mathcal{E}_P \]
while (1)
\[ \% \mathbf{x}_c \in \mathcal{E}_P \]
\[ \mathbf{y}_c = \text{max}(\text{min}(\mathbf{y} - \mathbf{yd}, 1), -1); \]
\[ \% \mathbf{x}_c \in \mathcal{E}_P, \quad y^2_c \leq 1 \]
\[ \% \left( \begin{array}{l} \mathbf{x}_c \\ \mathbf{y}_c \end{array} \right) \in \mathcal{E}_{Q_\mu}, \quad Q_\mu = \left( \begin{array}{cc} \mu^P & 0 \\ 0 & 1-\mu \end{array} \right) \]
\[ \mathbf{u} = \mathbf{C}_c*\mathbf{x}_c + \mathbf{D}_c*\mathbf{y}_c; \]
\[ \% \left( \begin{array}{l} \mathbf{x}_c \\ \mathbf{y}_c \end{array} \right) \in \mathcal{E}_{Q_\mu} \]
\[ \mathbf{x}_c = \mathbf{A}_{cf}*\mathbf{x}_c + \mathbf{B}_{cf}*\mathbf{y}_c; \]
\[ \% \mathbf{x}_c \in \mathcal{E}_S, \quad S = \left( \begin{array}{cc} \mathbf{A}_{cf} & \mathbf{B}_{cf} \\ \mathbf{B}_{cf} & \mathbf{A}_{cf} \end{array} \right) Q^{-1}_\mu (\mathbf{A}_{cf}^T \mathbf{B}_{cf})^{-1} \]
send(u, 1);
\[ \% \mathbf{x}_c \in \mathcal{E}_S \]
receive(y, 2); receive(yd, 3);
\[ \% \mathbf{x}_c \in \mathcal{E}_S \]
\[ \% \mathbf{x}_c \in \mathcal{E}_P \]
end

In the rest of the code:

- \( A_c, B_c \) replaced by \( A_{cf}, B_{cf} \);
- \( R \) depends on \( A_c, B_c \), replaced by \( S \);
- Check if \( \mathcal{E}_S \subset \mathcal{E}_P \).
Theoretical Framework

Transpose code + invariants in two steps:

Real

\[
\% d \\
\% i \\
\% d' = \theta(d, i)
\]

Intermediate

\[
\% \tilde{d} \\
\% \tilde{i} \\
\% \tilde{d}' = \theta(\tilde{d}, \tilde{i})
\]

**Code**: constants converted into machine numbers

**Invariants** recomputed using the same propagation theorem \( \theta \)
# Theoretical Framework

Transpose code + invariants in two steps:

<table>
<thead>
<tr>
<th>Real</th>
<th>Intermediate</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>( % d )</td>
<td>( % \tilde{d} )</td>
<td>( % \tilde{d} )</td>
</tr>
<tr>
<td>( i )</td>
<td>( \tilde{i} )</td>
<td>( \tilde{i} )</td>
</tr>
<tr>
<td>( % d' = \theta(d, i) )</td>
<td>( % \tilde{d}' = \theta(\tilde{d}, \tilde{i}) )</td>
<td>( % \tilde{d}' \supset \theta(\tilde{d}, \tilde{i}) \oplus \varepsilon )</td>
</tr>
</tbody>
</table>

**Code:** constants converted into machine numbers

**Invariants** recomputed using the same propagation theorem \( \theta \)

**Code:** real functions +, *... replaced by their machine counterparts

**Invariants** enlarged to include rounding error

Preserve invariant shape for propagation
Replace functions:

\[
\begin{align*}
\% \left( \begin{array}{c} x_c \\ y_c \end{array} \right) & \in E_{Q_\mu} \\
x_c & = A_c f \times x_c + B_c f \times y_c; \\
\% x_c & \in E_S, \quad S = \left[ (A_c f \ B_c f) Q_\mu^{-1} (A_c f \ B_c f)^T \right]^{-1}
\end{align*}
\]

- Replace + and \( \times \) by their FP counterparts;
- Increase \( E_S \) to include arithmetic error.
Example System, 32-bit Floating-Point Numbers

e_1, e_2 is the arithmetic error on x_{c_1}, x_{c_2}.

\mathcal{E}_T \supset \mathcal{E}_S is an ellipse s.t.:

\forall x_c \in \mathcal{E}_S, \forall x'_c \in \mathbb{R}^2, \\
|x'_c - x_{c_1}| \leq e_1 \land |x'_c - x_{c_2}| \leq e_2 \implies x'_c \in \mathcal{E}_T \quad (*)

\mathcal{E}_T can be the smallest magnification of \mathcal{E}_S s.t. (*) holds.
Example System, 32-bit Floating-Point Numbers

\[
\begin{align*}
\% \ (x_c, y_c) & \in E_{Q \mu} \\
x_c & = Acf * x_c + Bcf * y_c; \\
\% \ x_c & \in E_S, \quad S = [(A_{cf} \ B_{cf})Q^{-1}(A_{cf} \ B_{cf})^T]^{-1} \\
\text{send}(u, 1); \\
\% \ x_c & \in E_S \\
\text{receive}(y, 2); \text{receive}(yd, 3); \\
\% \ x_c & \in E_S \\
\% \ x_c & \in E_P \\
\text{end}
\end{align*}
\]

In the rest of the code:
Example System, 32-bit Floating-Point Numbers

\[ \begin{align*}
\% \left( \begin{array}{c} x_c \\ y_c \end{array} \right) & \in \mathcal{E}_{Q \mu} \\
x_c & = \text{Acf} \cdot xc + \text{Bcf} \cdot yc; \\
\% \ x_c & \in \mathcal{E}_T \\
\text{send}(u, 1); \\
\% \ x_c & \in \mathcal{E}_T \\
\text{receive}(y, 2); \text{receive}(yd, 3); \\
\% \ x_c & \in \mathcal{E}_T \\
\% \ x_c & \in \mathcal{E}_P \\
\text{end}
\end{align*} \]

In the rest of the code:
- Replace \( \mathcal{E}_S \) by \( \mathcal{E}_T \);
Example System, 32-bit Floating-Point Numbers

\[ (x_c, y_c) \in \mathcal{E}_{Q, \mu} \]
\[ x_c = Acf*x_c + Bcf*y_c; \]
\[ x_c \in \mathcal{E}_T \]
\[ \text{send}(u, 1); \]
\[ x_c \in \mathcal{E}_T \]
\[ \text{receive}(y, 2); \text{receive}(yd, 3); \]
\[ x_c \in \mathcal{E}_T \]
\[ x_c \in \mathcal{E}_P \]
end

In the rest of the code:

- Replace \( \mathcal{E}_S \) by \( \mathcal{E}_T \);
- Check if \( \mathcal{E}_T \subset \mathcal{E}_P \).

It works! \( \Rightarrow \) Stable in 32 bits. If not, cannot conclude.
Automation: The LyaFloat Tool

In Python, using SymPy.

```python
from lyafloat import *
setfloatify(constants=True, operators=True, precision=53)

P = Rational("1e-3") * Matrix(rationals(
    ["0.6742 0.0428", "0.0428 2.4651"]))
EP = Ellipsoid(P)

xc1, xc2, yc = symbols("xc1 xc2 yc")
Ac = Matrix(constants(["0.4990 -0.0500", "0.0100 1.0000"]))
ES = Ellipsoid(R)
print("ES included in EP :", ES <= EP)

i = Instruction({xc: Ac * xc + Bc * yc},
    pre=[zc in EQmu], post=[xc in ES])
ET = i.post()[xc]
print("ET =", ET)
print("ET included in EP :", ET <= EP)
```
Closed Loop

Closed-loop system:
- Pseudocode for controller and for environment;
- send & receive;
- Only controller code is changed.

Does not work with 32 bits.
OK with 128 bits.
Related Work

Compute bounds from source code, \textit{open-loop} case:

- Astrée;
- PhD P. Roux.

From pseudocode to C:

- Feron ICSM’10.

Floating-point arithmetic:

- PhD P. Roux.
Conclusion

Theoretical framework to translate invariants on code with real arithmetic, while preserving the overall proof structure.

LyaFloat: implementation for Lyapunov-theoretic proofs on floating-point arithmetic. Suitable method if bounded error.

Future work:

1. Other **arithmetic paradigms**:
   - OK with floating point: rounding error bounded for +, −, * if no extremal value;
   - Same for fixed point;
   - Not sure what happens with rationals;

2. **Other functions** (non-linear systems):
   - Differentiable, periodic functions (cos);
   - Differentiable functions restricted to a finite range.

3. More **formal guarantees**: Coq rather than Python
   - formalization (or proof?) of propagators;
   - or generate Coq scripts.
Des réels aux flottants : préservation automatique de preuves de stabilité de Lyapunov

Olivier Hermant, Vivien Maisonneuve

14èmes journées Approches Formelles dans l’Assistance au Développement Logiciel
Bordeaux, 9 Juin 2015