Translation of Lyapunov Stability Proofs to Machine Arithmetic

Vivien Maisonneuve

Eighth meeting of the French community of compilation

Nice, July 2014
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

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
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?
Levels of Description

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

**MATLAB, Simulink Realization**: very low-level C
- 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 $\implies x_c$ bounded (hence $y_c$ bounded)

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

Lyapunov theory provides a framework to compute inductive invariants.

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

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

Lyapunov theory provides a framework to compute inductive invariants.

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

Static analysis to show that the invariant holds from source code.
Lyapunov theory applies on a system with real arithmetic.

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

\[ u \]

\[ y \quad y_d \]

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

\[
Bc = [1; 0];
\]

\[
Cc = [564.48, 0];
\]

\[
Dc = -1280;
\]

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

\[
\text{receive(y, 2); receive(yd, 3); while (1)}
\]

\[
yc = \max(\min(y - yd, 1), -1);
u = Cc*xc + Dc*yc;
xc = Ac*xc + Bc*yc;
\]

\[
\text{send(u, 1); receive(y, 2); receive(yd, 3); end}
\]
Example System

[Feron ICSM’10]:
mass-spring system.

Open-loop stability:
$x_c$ bounded.

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

Lyapunov theory $\implies x_c = \begin{pmatrix} x_{c1} \\ x_{c2} \end{pmatrix}$ 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

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

\[ B_c = [1; 0]; \]

\[ C_c = [564.48, 0]; \]

\[ D_c = -1280; \]

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

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

\text{while} \ (1)

\% \ x_c \in E_P

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

\[ u = Cc*x_c + Dc*y_c; \]

\[ x_c = Ac*x_c + Bc*y_c; \]

\text{send}(u, 1);

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

\% \ x_c \in E_R \subset E_P

\text{end}
Example System: Invariants

\[ xc = \text{zeros}(2, 1); \]
\% \( x_c \in \mathcal{E}_P \)

receive(\textit{y}, 2); receive(\textit{yd}, 3);
\% \( x_c \in \mathcal{E}_P \)

while (1)
\% \( x_c \in \mathcal{E}_P \)

\[ yc = \max(\min(y - yd, 1), -1); \]
\% \( x_c \in \mathcal{E}_P, \ y_c^2 \leq 1 \)
\% \((x_c^T y_c) \in \mathcal{E}_{Q_{\mu}}, \ Q_{\mu} = \begin{pmatrix} \mu P & 0 \\ 0 & 1 - \mu \end{pmatrix}, \ \mu = 0.9991 \)

\[ u = Cc*xc + Dc*yc; \]
\% \((x_c^T y_c) \in \mathcal{E}_{Q_{\mu}} \)

\[ xc = Ac*xc + Bc*yc; \]
\% \( x_c \in \mathcal{E}_R, \ R = [(A_c \ B_c)Q_{\mu}^{-1}(A_c \ B_c)^T]^{-1} \)

send(\textit{u}, 1);
\% \( x_c \in \mathcal{E}_R \)

receive(\textit{y}, 2); receive(\textit{yd}, 3);
\% \( x_c \in \mathcal{E}_R \)
\% \( x_c \in \mathcal{E}_P \)

end
Example System: Invariants

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

\% \quad (x_c \ y_c) \in \mathcal{E}_{Q\mu}
\quad x_c = A_c x_c + B_c y_c;
\% \quad x_c \in \mathcal{E}_R, \quad R = [(A_c \ B_c)Q_\mu^{-1}(A_c \ B_c)^T]^{-1}
Example System

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

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

while (1)
  \% \( x_c \in E_P \)
  y_c = \max(\min(y - yd, 1), -1); \\
  u = C_c x_c + D_c y_c; \\
  x_c = A_c x_c + B_c y_c; \\
  send(u, 1); \\
  receive(y, 2); receive(yd, 3); \\
  \% \( x_c \in E_P \)
end

Using limited-precision arithmetic:

1. Constant values are altered \( \Rightarrow E_P \) no longer valid;
2. Rounding errors during computations. Adapt invariants.
Example System

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

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

Using limited-precision arithmetic:

1. Constant values are altered
Example System

\[
Ac = \begin{bmatrix} 0.4990 & -0.0500 \\ 0.0100 & 1.0000 \end{bmatrix}; \\
Bc = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \\
Cc = \begin{bmatrix} 564.48 & 0 \end{bmatrix}; \\
Dc = -1280;
\]

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

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

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

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

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

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

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

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

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

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

\[
\text{end}
\]

Using limited-precision arithmetic:

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

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

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

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

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

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

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

\[
xc = Ac*xc + Bc*y_c;
\]

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

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

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

\[
\text{end}
\]

Using limited-precision arithmetic:

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

1. Constant values are altered \( \Rightarrow \mathcal{E}_P \) no longer valid;

2. Rounding errors during computations.

\textbf{Adapt invariants.}
Theoretical Framework

Transpose code + invariants in two steps:

Real

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

Transpose code + invariants in two steps:

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

**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>% (\bar{d})</td>
</tr>
<tr>
<td>i</td>
<td>(\tilde{i})</td>
<td>(\bar{i})</td>
</tr>
<tr>
<td>% d' = (\theta(d, i))</td>
<td>% (\tilde{d}' = \theta(\tilde{d}, \tilde{i}))</td>
<td>% (\bar{d}' \supset \theta(\bar{d}, \bar{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
Example System, 32-bit Floating-Point Numbers

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

1 Convert constants:

\[
A_{cf} = \begin{bmatrix} 0.498999999999999911821580299874766109466552734375, \\
-0.050000000000000000000000277555756156289135105907917022705078125; \\
0.01000000000000000000000020816681711721685132943093776702880859375, \\
1.00000 \end{bmatrix} \\
B_{cf} = [1; 0]; \\
C_{cf} = [564.48000000000001818989403545856475830078125, 0] \\
D_{cf} = -1280
Example System, 32-bit Floating-Point Numbers

\[
x_c = \text{zeros}(2, 1);
% \ x_c \in \mathcal{E}_P
\]
\[
\text{receive}(y, 2); \text{receive}(yd, 3);
% \ x_c \in \mathcal{E}_P
\]
while (1)
% \ x_c \in \mathcal{E}_P
\[
y_c = \max(\min(y - yd, 1), -1);
% \ x_c \in \mathcal{E}_P, \ y_c^2 \leq 1
% \ (x_c, y_c) \in \mathcal{E}_{Q_\mu}, \ Q_\mu = \begin{pmatrix} \mu P & 0 \\ 0 & 1 - \mu \end{pmatrix}
\]
\[
u = Cc \ast x_c + Dc \ast y_c;
% \ (x_c, y_c) \in \mathcal{E}_{Q_\mu}
\]
\[
x_c = Ac \ast x_c + Bc \ast y_c;
% \ x_c \in \mathcal{E}_R, \ R = [(A_c \ B_c)Q_\mu^{-1}(A_c \ B_c)^T]^{-1}
\]
send(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
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 \(E_s \subset \mathcal{E}_P\).
Example System, 32-bit Floating-Point Numbers

\[ x_c = \text{zeros}(2, 1); \]
\[ \% \ x_c \in \mathcal{E}_P \]
receive\((y, 2)\); receive\((yd, 3)\);
\[ \% \ x_c \in \mathcal{E}_P \]
\textbf{while} (1)
\[ \% \ x_c \in \mathcal{E}_P \]
\[ y_c = \max(\min(y - yd, 1), -1); \]
\[ \% \ x_c \in \mathcal{E}_P, \ y_c^2 \leq 1 \]
\[ \% \ (x_c, y_c) \in \mathcal{E}_{Q_\mu}, \ Q_\mu = \begin{pmatrix} \mu & 0 \\ 0 & 1 - \mu \end{pmatrix} \]
\[ u = Cc*x_c + Dc*y_c; \]
\[ \% \ (x_c, y_c) \in \mathcal{E}_{Q_\mu} \]
\[ x_c = Acf*x_c + Bcf*y_c; \]
\[ \% \ x_c \in \mathcal{E}_R, \ R = [(A_c \ B_c)Q_\mu^{-1}(A_c \ B_c)^T]^{-1} \]
\[ \text{send}(u, 1); \]
\[ \% \ x_c \in \mathcal{E}_R \]
receive\((y, 2)\); receive\((yd, 3)\);
\[ \% \ x_c \in \mathcal{E}_R \]
\[ \% \ x_c \in \mathcal{E}_P \]
\textbf{end}

In the rest of the code:
- \( A_c, B_c \) replaced by \( A_{cf}, B_{cf} \);
Example System, 32-bit Floating-Point Numbers

\[
\begin{align*}
\mathbf{x}_c &= \text{zeros}(2, 1); \\
\% \quad \mathbf{x}_c \in \mathcal{E}_P \\
\text{receive}(\mathbf{y}, 2); \text{receive}(\mathbf{yd}, 3); \\
\% \quad \mathbf{x}_c \in \mathcal{E}_P \\
\text{while} \ (1) \\
\% \quad \mathbf{x}_c \in \mathcal{E}_P \\
\mathbf{y}_c &= \max(\min(\mathbf{y} - \mathbf{yd}, 1), -1); \\
\% \quad \mathbf{x}_c \in \mathcal{E}_P, \quad y^2_c \leq 1 \\
\% \quad (\mathbf{x}_c, \mathbf{y}_c) \in \mathcal{E}_{\mathcal{Q}_P}, \quad \mathcal{Q}_P = \begin{pmatrix} \mu & 0 \\ 0 & 1 - \mu \end{pmatrix} \\
\mathbf{u} &= \mathbf{Cc}^*\mathbf{x}_c + \mathbf{Dc}^*\mathbf{y}_c; \\
\% \quad (\mathbf{x}_c, \mathbf{y}_c) \in \mathcal{E}_{\mathcal{Q}_P} \\
\mathbf{x}_c &= \mathbf{Acf}^*\mathbf{x}_c + \mathbf{Bcf}^*\mathbf{y}_c; \\
\% \quad \mathbf{x}_c \in \mathcal{E}_S, \quad S = [(\mathbf{Acf} \quad \mathbf{Bcf})\mathcal{Q}_P^{-1}(\mathbf{Acf} \quad \mathbf{Bcf})^T]^{-1} \\
\text{send}(\mathbf{u}, 1); \\
\% \quad \mathbf{x}_c \in \mathcal{E}_S \\
\text{receive}(\mathbf{y}, 2); \text{receive}(\mathbf{yd}, 3); \\
\% \quad \mathbf{x}_c \in \mathcal{E}_S \\
\% \quad \mathbf{x}_c \in \mathcal{E}_P \\
\text{end}
\end{align*}
\]

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 \( S \);
Example System, 32-bit Floating-Point Numbers

\[ \text{xc} = \text{zeros}(2, 1); \]
% \( \text{xc} \in \mathcal{E}_P \)
receive(y, 2); receive(yd, 3);
% \( \text{xc} \in \mathcal{E}_P \)
while (1)
% \( \text{xc} \in \mathcal{E}_P \)
yc = max(min(y - yd, 1), -1);
% \( \text{xc} \in \mathcal{E}_P \), \( y_c^2 \leq 1 \)
% \( (\text{xc} \ y_c) \in \mathcal{E}_{Q_{\mu}} \), \( Q_{\mu} = \begin{pmatrix} \mu^P & 0 \\ 0 & 1-\mu \end{pmatrix} \)
u = Cc*xc + Dc*yc;
% \( (\text{xc} \ y_c) \in \mathcal{E}_{Q_{\mu}} \)
xc = Acf*xc + Bcf*yc;
% \( \text{xc} \in \mathcal{E}_S \), \( S = [(A_{cf} \ B_{cf})Q_{\mu}^{-1}(A_{cf} \ B_{cf})^T]^{-1} \)
send(u, 1);
% \( \text{xc} \in \mathcal{E}_S \)
receive(y, 2); receive(yd, 3);
% \( \text{xc} \in \mathcal{E}_S \)
% \( \text{xc} \in \mathcal{E}_P \)
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 \( \mathcal{E}_S \subset \mathcal{E}_P \).
Example System, 32-bit Floating-Point Numbers

2 Replace functions:

\[
\begin{array}{l}
\% (x_c, y_c) \in \mathcal{E}_{Q_\mu} \\
x_c = A_c f x_c + B_c f y_c; \\
% x_c \in \mathcal{E}_S, \quad S = [(A_{cf} \ B_{cf}) Q_\mu^{-1} (A_{cf} \ B_{cf})^T]^{-1}
\end{array}
\]

\[
\ldots
\]

- Replace $+$ and $\times$ by their FP counterparts;
- Increase $\mathcal{E}_S$ 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.:

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

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

\( (x_c, y_c) \in \mathcal{E}_{Q_\mu} \)

\( x_c = A_{cf}x_c + B_{cf}y_c; \)

\( x_c \in \mathcal{E}_S, \quad S = [(A_{cf} B_{cf})Q_\mu^{-1}(A_{cf} B_{cf})^T]^{-1} \)

send(u, 1);

\( x_c \in \mathcal{E}_S \)

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

\( x_c \in \mathcal{E}_S \)

\( x_c \in \mathcal{E}_P \)

end

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

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

In the rest of the code:

- Replace $\mathcal{E}_S$ by $\mathcal{E}_T$;
Example System, 32-bit Floating-Point Numbers

\[ \{ x_c \} \in \mathcal{E}_{Q_{\mu}} \]
\[ x_c = A_c x_c + B_c y_c; \]
\[ \% x_c \in \mathcal{E}_T \]
\[ \text{send}(u, 1); \]
\[ \% x_c \in \mathcal{E}_T \]
\[ \text{receive}(y, 2); \text{receive}(yd, 3); \]
\[ \% x_c \in \mathcal{E}_T \]
\[ \% x_c \in \mathcal{E}_P \]
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, can’t 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.
Suitable method if bounded error.

1. **Arithmetic paradigms:**
   - OK with floating point: rounding error is bounded for +, −, * if far enough from extremal values;
   - Same for fixed point;
   - Not sure what happens with two integers;
Extensions of LyaFloat

Suitable method if bounded error.

1. **Arithmetic paradigms:**
   - OK with floating point: rounding error is bounded for +, −, * if far enough from extremal values;
   - Same for fixed point;
   - Not sure what happens with two integers;

2. **Other functions** (non-linear systems):
   - Differentiable, periodic functions (cos) (can be computed with an abacus/polynomial interpolation);
   - Differentiable functions restricted to a finite range (assuming values in the range).
Related Work

Compute bounds from source code:
- Astrée;
- PhD P. Roux.

From pseudocode to C:
- Feron ICSM’10.

Floating-point arithmetic:
- PhD P. Roux.

Proof translation, code-level invariants.
Closed loop.
Conclusion

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

LyaFloat: implementation for Lyapunov-theoretic proofs on floating-point arithmetic.

Future work:

- Support for other arithmetic paradigms, more functions, more invariant propagators;
- Coq rather than Python \(\implies\) formalization (or proof?) of propagators;
- ...or generate Coq scripts?
Translation of Lyapunov Stability Proofs to Machine Arithmetic

Vivien Maisonneuve

Eighth meeting of the French community of compilation

Nice, July 2014