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

Olivier Hermant, Vivien Maisonneuve

MINES ParisTech

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.

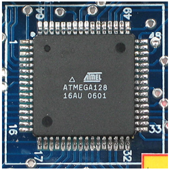
Uses a low-power processor or a microcontroller.

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

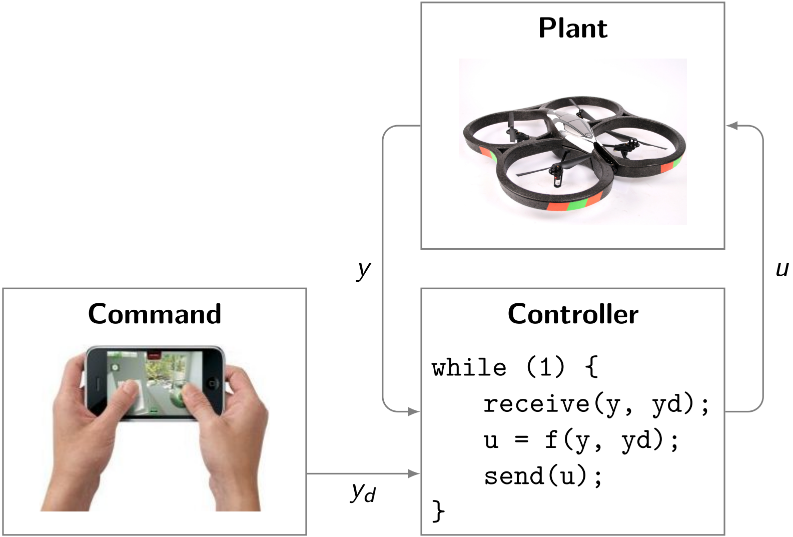
Example

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

ATMEGA128: 16 MHz, 4 KB RAM, 128 KB ROM

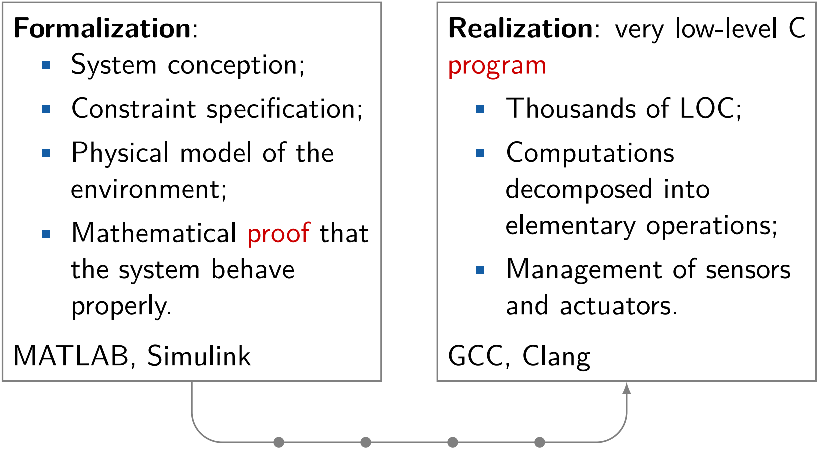


Control-Command System



```
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

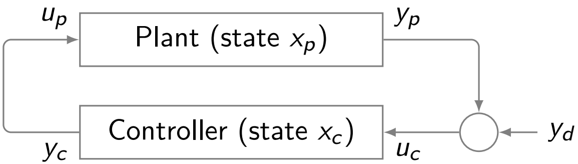
Gradual **transformations**

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.

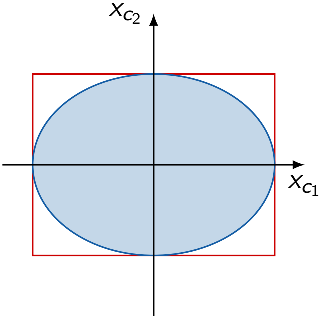


- Open loop stability: u_c bounded $\Longrightarrow$ x_c bounded (hence y_c bounded)
- Closed loop stability: y_d bounded $\Longrightarrow$ x_c, x_p bounded (hence y_c, y_p bounded)

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.

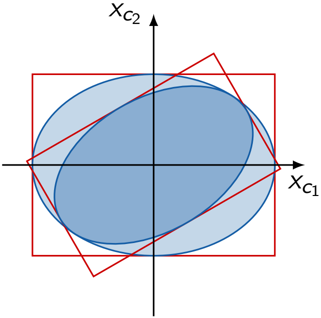


Static analysis to show that the invariant holds from source code.

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.



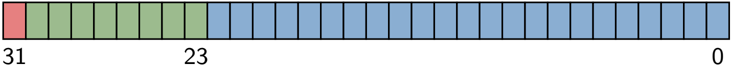
Static analysis to show that the invariant holds from source code.

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.

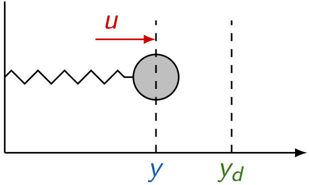
1. Constant values are altered;
2. Rounding errors during computations.

$\Longrightarrow$ 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.

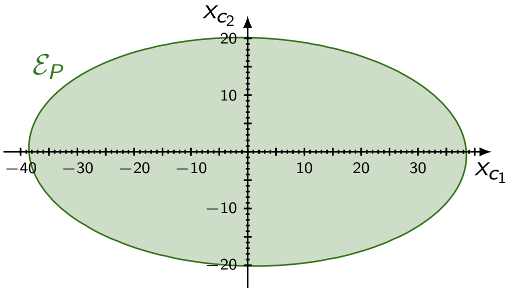
```
Ac = [0.4990, -0.0500;
       0.0100, 1.0000];
Bc = [1; 0];
Cc = [564.48, 0];
Dc = -1280;
xc = zeros(2, 1);
receive(y, 2); receive(yd, 3);
while (1)
  yc = max(min(y - yd, 1), -1);
  u = Cc*xc + Dc*yc;
  xc = Ac*xc + Bc*yc;
  send(u, 1);
  receive(y, 2); receive(yd, 3);
end
```

Example System: Stability Ellipse

Lyapunov theory $\Longrightarrow x_c = \begin{pmatrix} x_{c_1} \\ x_{c_2} \end{pmatrix}$ belongs to the ellipse:

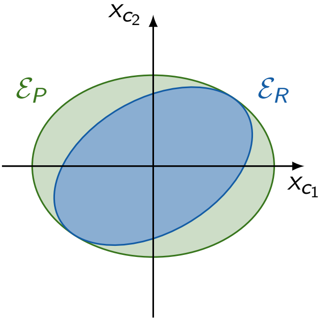
$$\mathcal{E}_P = \{x \in \mathbb{R}^2 \,|\, x^T \cdot P \cdot x \leq 1\} \qquad P = 10^{-3} \begin{pmatrix} 0.6742 & 0.0428 \\ 0.0428 & 2.4651 \end{pmatrix}$$

$$x_c \in \mathcal{E}_P \Longleftrightarrow 0.6742 x_{c_1}^2 + 0.0856 x_{c_1} x_{c_2} + 2.4651 x_{c_2}^2 \leq 1000$$



Example System

```
Ac = [0.4990, -0.0500;
       0.0100, 1.0000];
Bc = [1; 0];
Cc = [564.48, 0];
Dc = -1280;
xc = zeros(2, 1);
receive(y, 2); receive(yd, 3);
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;
  send(u, 1);
  receive(y, 2); receive(yd, 3);
  % $x_c \in \mathcal{E}_R \subset \mathcal{E}_P$
end
```



Example System

```
Ac = [0.4990, -0.0500;
       0.0100, 1.0000];
Bc = [1; 0];
Cc = [564.48, 0];
Dc = -1280;
xc = zeros(2, 1);
receive(y, 2); receive(yd, 3);
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;
  send(u, 1);
  receive(y, 2); receive(yd, 3);
  % $x_c \in \mathcal{E}_P$
end
```

Using limited-precision arithmetic:

Example System

```
Ac = [0.4990, -0.0500;
      0.0100, 1.0000];
Bc = [1; 0];
Cc = [564.48, 0];
Dc = -1280;
xc = zeros(2, 1);
receive(y, 2); receive(yd, 3);
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;
  send(u, 1);
  receive(y, 2); receive(yd, 3);
  % $x_c \in \mathcal{E}_P$
end
```

Using limited-precision arithmetic:

1. Constant values are altered

Example System

```
Ac = [0.4990, -0.0500;
      0.0100, 1.0000];
Bc = [1; 0];
Cc = [564.48, 0];
Dc = -1280;
xc = zeros(2, 1);
receive(y, 2); receive(yd, 3);
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;
  send(u, 1);
  receive(y, 2); receive(yd, 3);
  % ~~$x_c \in \mathcal{E}_P$~~
end
```

Using limited-precision arithmetic:

1. Constant values are altered $\Longrightarrow \mathcal{E}_P$ no longer valid;

Example System

```
Ac = [0.4990, -0.0500;
      0.0100, 1.0000];
Bc = [1; 0];
Cc = [564.48, 0];
Dc = -1280;
xc = zeros(2, 1);
receive(y, 2); receive(yd, 3);
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;
  send(u, 1);
  receive(y, 2); receive(yd, 3);
  % ~~$x_c \in \mathcal{E}_P$~~
end
```

Using limited-precision arithmetic:

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

Example System

```
Ac = [0.4990, -0.0500;
      0.0100, 1.0000];
Bc = [1; 0];
Cc = [564.48, 0];
Dc = -1280;
xc = zeros(2, 1);
receive(y, 2); receive(yd, 3);
while (1)
  % ~~x_c ∈ ℰ_P~~
  yc = max(min(y - yd, 1), -1);
  u = Cc*xc + Dc*yc;
  xc = Ac*xc + Bc*yc;
  send(u, 1);
  receive(y, 2); receive(yd, 3);
  % ~~x_c ∈ ℰ_P~~
end
```

Using limited-precision arithmetic:

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

Adapt invariants.

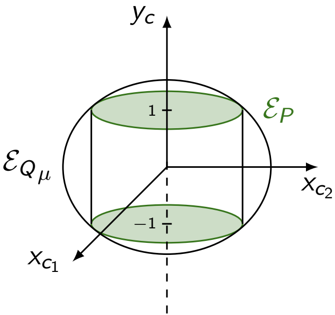
Example System: Invariants

```
xc = zeros(2, 1);
% $x_c \in \mathcal{E}_P$
receive(y, 2); receive(yd, 3);
% $x_c \in \mathcal{E}_P$
while (1)
  % $x_c \in \mathcal{E}_P$
  yc = max(min(y - yd, 1), -1);
  % $x_c \in \mathcal{E}_P, \quad y_c^2 \leq 1$
  % $\left(\begin{smallmatrix} x_c \\ y_c \end{smallmatrix}\right) \in \mathcal{E}_{Q_\mu}, \quad Q_\mu = \left(\begin{smallmatrix} \mu P & 0 \\ 0 & 1-\mu \end{smallmatrix}\right), \quad \mu = 0.9991$
  u = Cc*xc + Dc*yc;
  % $\left(\begin{smallmatrix} x_c \\ y_c \end{smallmatrix}\right) \in \mathcal{E}_{Q_\mu}$
  xc = Ac*xc + Bc*yc;
  % $x_c \in \mathcal{E}_R, \quad R = \left[(A_c \;\; B_c) Q_\mu^{-1} (A_c \;\; B_c)^{\mathtt{T}}\right]^{-1}$
  send(u, 1);
  % $x_c \in \mathcal{E}_R$
  receive(y, 2); receive(yd, 3);
  % $x_c \in \mathcal{E}_R$
  % $x_c \in \mathcal{E}_P$
end
```

Example System: Invariants

% $x_c \in \mathcal{E}_P, \quad y_c^2 \leq 1$

% $\begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu}, \quad Q_\mu = \begin{pmatrix} \mu P & 0 \\ 0 & 1-\mu \end{pmatrix}, \quad \mu = 0.9991$



% $\begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu}$

```
xc = Ac*xc + Bc*yc;
```

% $x_c \in \mathcal{E}_R, \quad R = \left[(A_c \;\; B_c) Q_\mu^{-1} (A_c \;\; B_c)^{\mathtt{T}}\right]^{-1}$

Theoretical Framework

Transpose code + invariants in two steps:

Real

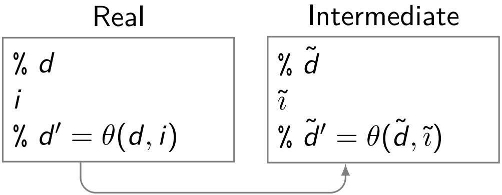
% d

i

% $d' = \theta(d, i)$

Theoretical Framework

Transpose code + invariants in two steps:



Real

% d
i
% $d' = \theta(d, i)$

Intermediate

% $\tilde{d}$
$\tilde{\imath}$
% $\tilde{d}' = \theta(\tilde{d}, \tilde{\imath})$

Code: constants converted into machine numbers

Invariants recomputed using the same propagation theorem θ

Example System, 32-bit Floating-Point Numbers

```
Ac = [0.4990, -0.0500;
       0.0100, 1.0000];
Bc = [1; 0];
Cc = [564.48, 0];
Dc = -1280;
xc = zeros(2, 1);
...
```

1. Convert constants:

```
Acf = [0.49899999999999999911182158029987476766109466552734375,
       -0.05000000000000000277555756156289135105907917022705078125;
       0.01000000000000000020816681711721685132943093776702880859375,
       1.0000]
Bcf = [1; 0];
Ccf = [564.48000000000001818989403545856475830078125, 0]
Dcf = -1280
```

Example System, 32-bit Floating-Point Numbers

```
xc = zeros(2, 1);
% $x_c \in \mathcal{E}_P$
receive(y, 2); receive(yd, 3);
% $x_c \in \mathcal{E}_P$
while (1)
  % $x_c \in \mathcal{E}_P$
  yc = max(min(y - yd, 1), -1);
  % $x_c \in \mathcal{E}_P, \quad y_c^2 \leq 1$
  % $\left(\begin{smallmatrix} x_c \\ y_c \end{smallmatrix}\right) \in \mathcal{E}_{Q_\mu}, \quad Q_\mu = \left(\begin{smallmatrix} \mu P & 0 \\ 0 & 1-\mu \end{smallmatrix}\right)$
  u = Cc*xc + Dc*yc;
  % $\left(\begin{smallmatrix} x_c \\ y_c \end{smallmatrix}\right) \in \mathcal{E}_{Q_\mu}$
  xc = Ac*xc + Bc*yc;
  % $x_c \in \mathcal{E}_R, \quad R = \left[(A_c \;\; B_c) Q_\mu^{-1} (A_c \;\; B_c)^{\mathrm{T}}\right]^{-1}$
  send(u, 1);
  % $x_c \in \mathcal{E}_R$
  receive(y, 2); receive(yd, 3);
  % $x_c \in \mathcal{E}_R$
  % $x_c \in \mathcal{E}_P$
end
```

In the rest of the code:

Example System, 32-bit Floating-Point Numbers

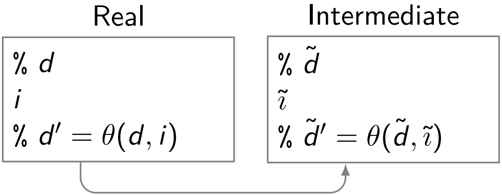
```
xc = zeros(2, 1);
% $x_c \in \mathcal{E}_P$
receive(y, 2); receive(yd, 3);
% $x_c \in \mathcal{E}_P$
while (1)
  % $x_c \in \mathcal{E}_P$
  yc = max(min(y - yd, 1), -1);
  % $x_c \in \mathcal{E}_P, \quad y_c^2 \leq 1$
  % $\left(\begin{smallmatrix} x_c \\ y_c \end{smallmatrix}\right) \in \mathcal{E}_{Q_\mu}, \quad Q_\mu = \left(\begin{smallmatrix} \mu P & 0 \\ 0 & 1-\mu \end{smallmatrix}\right)$
  u = Cc*xc + Dc*yc;
  % $\left(\begin{smallmatrix} x_c \\ y_c \end{smallmatrix}\right) \in \mathcal{E}_{Q_\mu}$
  xc = Acf*xc + Bcf*yc;
  % $x_c \in \mathcal{E}_S, \quad S = \left[(A_{cf} \;\; B_{cf}) Q_\mu^{-1} (A_{cf} \;\; B_{cf})^{\mathrm{T}}\right]^{-1}$
  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:

- 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

% $\tilde{d}$

$\tilde{\imath}$

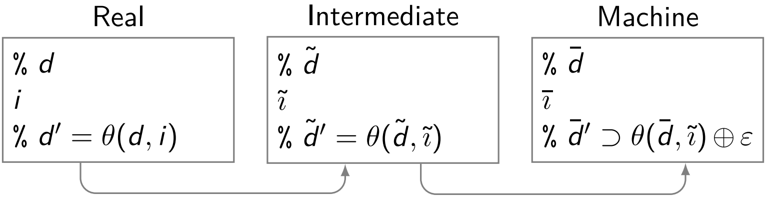
% $\tilde{d}' = \theta(\tilde{d}, \tilde{\imath})$

Code: constants converted into machine numbers

Invariants recomputed using the same propagation theorem θ

Theoretical Framework

Transpose code + invariants in two steps:



Real

% d
i
% $d' = \theta(d, i)$

Intermediate

% $\tilde{d}$
$\tilde{\imath}$
% $\tilde{d}' = \theta(\tilde{d}, \tilde{\imath})$

Machine

% $\bar{d}$
$\bar{\imath}$
% $\bar{d}' \supset \theta(\bar{d}, \tilde{\imath}) \oplus \varepsilon$

Code: constants converted into machine numbers

Invariants recomputed using the same propagation theorem θ

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

2 Replace functions:

```
...
% $\left(\begin{smallmatrix} x_c \\ y_c \end{smallmatrix}\right) \in \mathcal{E}_{Q_\mu}$
xc = Acf*xc + Bcf*yc;
% $x_c \in \mathcal{E}_S, \quad S = \left[(A_{cf} \;\; B_{cf}) Q_\mu^{-1} (A_{cf} \;\; B_{cf})^{\mathtt{T}}\right]^{-1}$
...
```

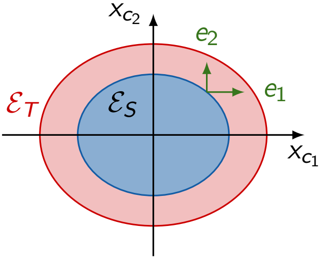
- Replace $+$ and $\times$ by their FP counterparts;
- Increase $\mathcal{E}_S$ to include arithmetic error.

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.:

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

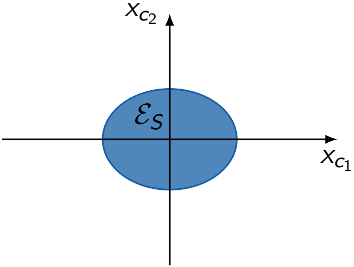


$\mathcal{E}_T$ can be the smallest magnification of $\mathcal{E}_S$ s.t. $(*)$ holds.

Example System, 32-bit Floating-Point Numbers

```
    ...
    % $\binom{x_c}{y_c} \in \mathcal{E}_{Q_\mu}$
    xc = Acf*xc + Bcf*yc;
    % $x_c \in \mathcal{E}_S, \quad S = \left[(A_{cf} \;\; B_{cf}) Q_\mu^{-1} (A_{cf} \;\; B_{cf})^{\mathtt{T}}\right]^{-1}$
    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:

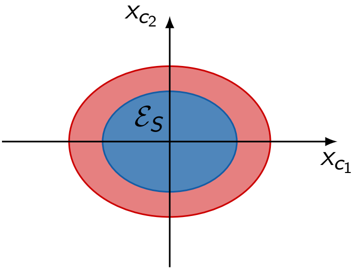


Example System, 32-bit Floating-Point Numbers

```
    ...
    % $\binom{x_c}{y_c} \in \mathcal{E}_{Q_\mu}$
    xc = Acf*xc + Bcf*yc;
    % $x_c \in \mathcal{E}_T$
    send(u, 1);
    % $x_c \in \mathcal{E}_T$
    receive(y, 2); 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 $\mathcal{E}_T$;



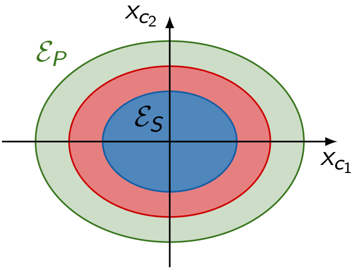
Example System, 32-bit Floating-Point Numbers

```
  ...
  % $\binom{x_c}{y_c} \in \mathcal{E}_{Q_\mu}$
  xc = Acf*xc + Bcf*yc;
  % $x_c \in \mathcal{E}_T$
  send(u, 1);
  % $x_c \in \mathcal{E}_T$
  receive(y, 2); 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! ⇒ Stable in 32 bits.
If not, cannot conclude.



Automation: The LyaFloat Tool

In Python, using SymPy.

```
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, 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

MINES ParisTech

14èmes journées Approches Formelles dans l'Assistance au Développement Logiciel

Bordeaux, 9 Juin 2015