Using interval methods and guaranteed integration of ODEs for the verification of embedded software.

Olivier BOUISSOU and Matthieu MARTEL

CEA LIST
Laboratoire Modélisation et Analyse de Systèmes en Intéraction

Université de Perpignan Via Domitia
Laboratoire ELIAUS

Small Workshop on Interval Methods 2008

Motivation

Context of this work : static analysis of safety critical softwares.

Behaviour of the program:

- collects inputs from a sensor;
- computes the response to that input;
- sends orders to actuators.

Abstract interpretation based static analysis:

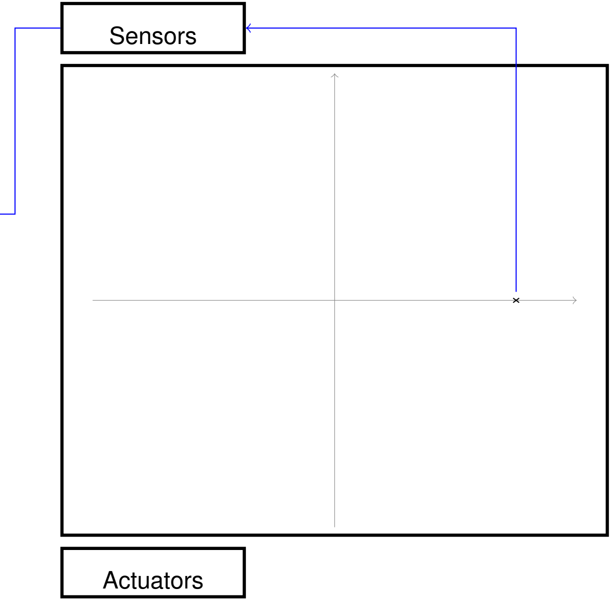
- computes invariants, i.e. properties that are true for all possible executions.
- usualy overapproximates the inputs by a constant interval.

Starting point of this work

Embedded programs are a part of a hybrid system and verification techniques don't take that information into account.

Example of an execution.

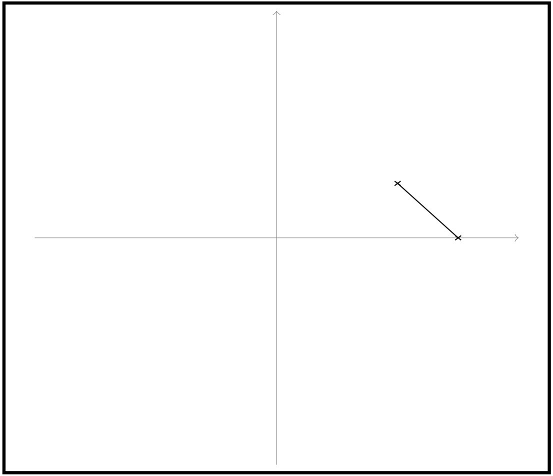
```
int main() {
 while(true) {
   sens.y?X
   X'=update(X);
   if (X'<0)
    act.cmd!1
   if (X'>15)
    act.cmd!0
   wait 0.01
 }
}
```



Example of an execution.

```
int main() {
 while(true) {
  sens.y?X
  X'=update(X);
  if (X'<0)
   act.cmd!1
  if (X'>15)
   act.cmd!0
  wait 0.01
 }
}
```

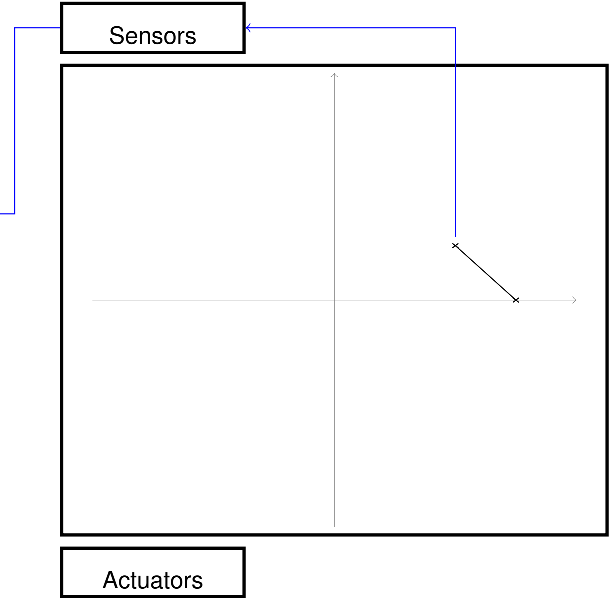
Sensors



Actuators

Example of an execution.

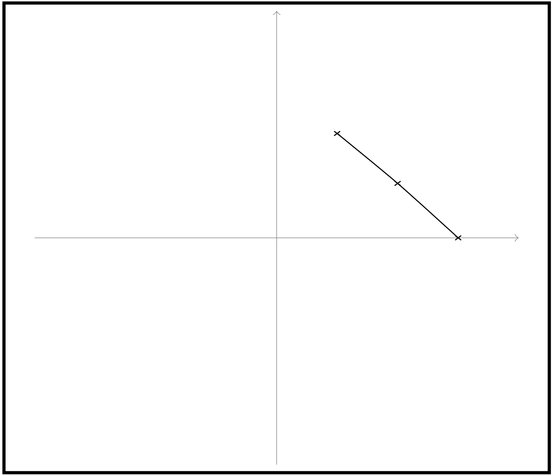
```
int main() {
 while(true) {
   sens.y?X
   X'=update(X);
   if (X'<0)
    act.cmd!1
   if (X'>15)
    act.cmd!0
   wait 0.01
 }
}
```



Example of an execution.

```
int main() {
 while(true) {
  sens.y?X
  X'=update(X);
  if (X'<0)
   act.cmd!1
  if (X'>15)
   act.cmd!0
  wait 0.01
 }
}
```

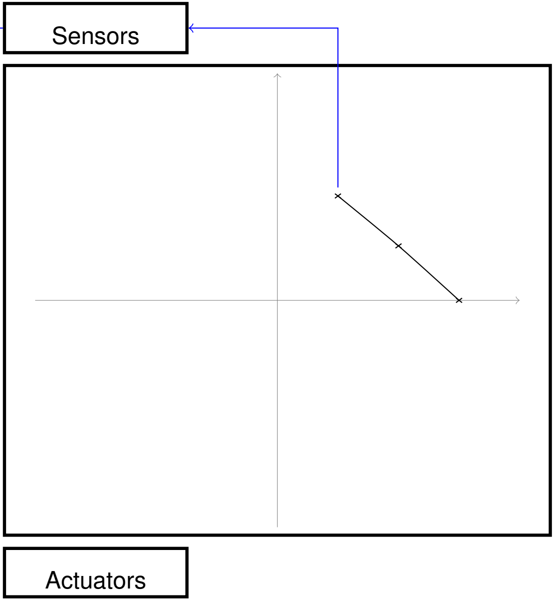
Sensors



Actuators

Example of an execution.

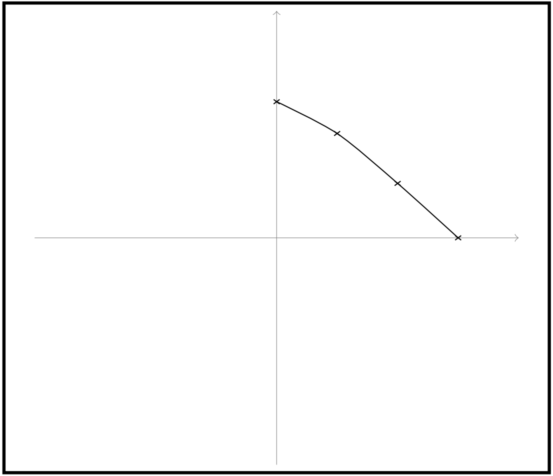
```
int main() {
 while(true) {
   sens.y?X
   X'=update(X);
   if (X'<0)
    act.cmd!1
   if (X'>15)
    act.cmd!0
   wait 0.01
 }
}
```



Example of an execution.

```
int main() {
 while(true) {
  sens.y?X
  X'=update(X);
  if (X'<0)
   act.cmd!1
  if (X'>15)
   act.cmd!0
  wait 0.01
 }
}
```

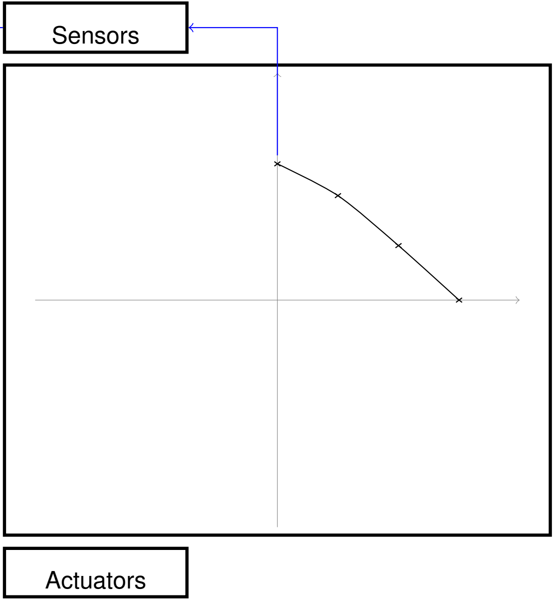
Sensors



Actuators

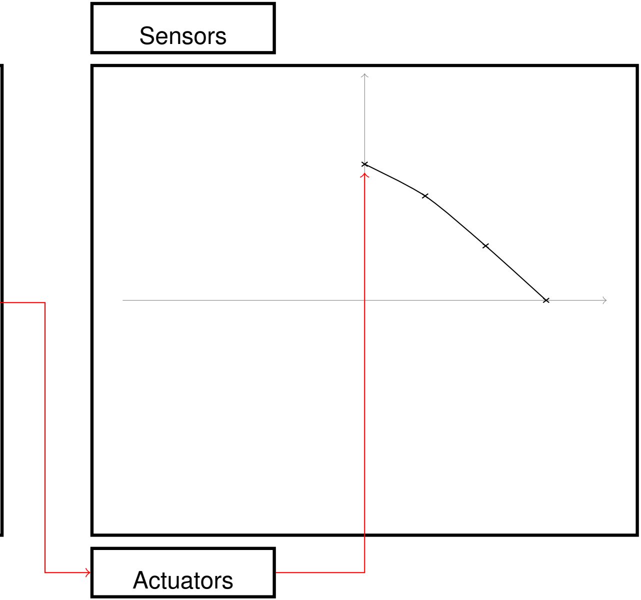
Example of an execution.

```
int main() {
 while(true) {
  sens.y?X
  X'=update(X);
  if (X'<0)
   act.cmd!1
  if (X'>15)
   act.cmd!0
  wait 0.01
 }
}
```



Example of an execution.

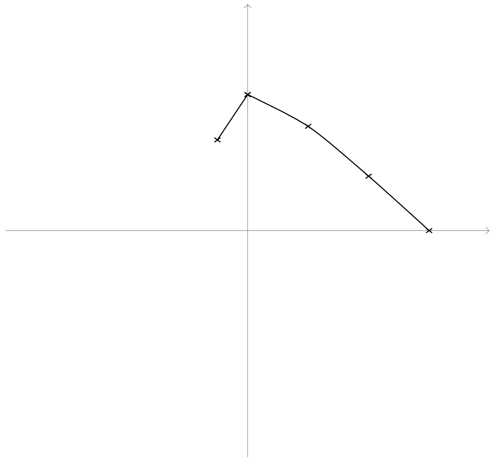
```
int main() {
 while(true) {
  sens.y?X
  X'=update(X);
  if (X'<0)
   act.cmd!1
  if (X'>15)
   act.cmd!0
  wait 0.01
 }
}
```



Example of an execution.

```
int main() {
 while(true) {
  sens.y?X
  X'=update(X);
  if (X'<0)
   act.cmd!1
  if (X'>15)
   act.cmd!0
  wait 0.01
 }
}
```

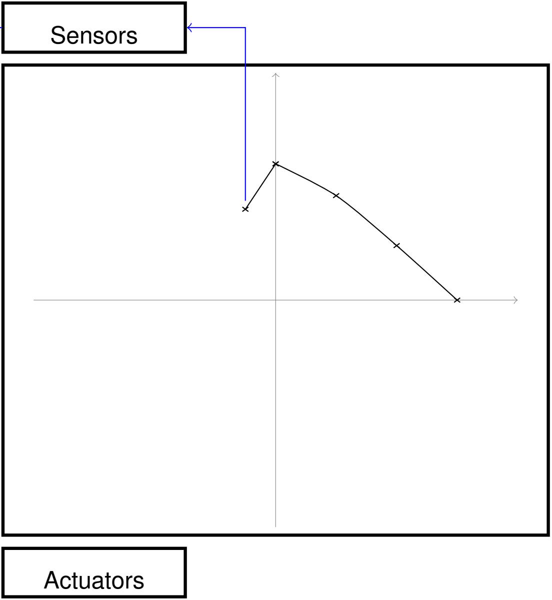
Sensors



Actuators

Example of an execution.

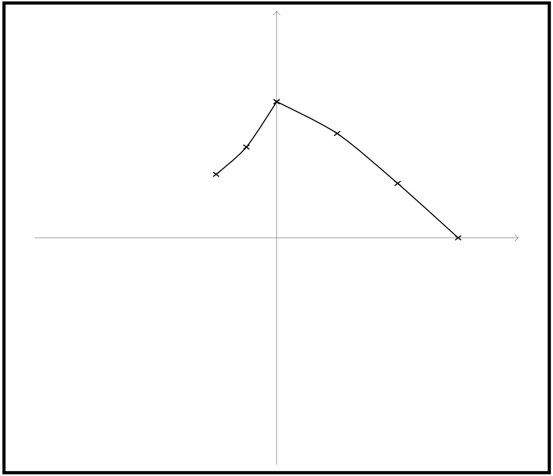
```
int main() {
 while(true) {
  sens.y?X
  X'=update(X);
  if (X'<0)
   act.cmd!1
  if (X'>15)
   act.cmd!0
  wait 0.01
 }
}
```



Example of an execution.

```
int main() {
 while(true) {
  sens.y?X
  X'=update(X);
  if (X'<0)
   act.cmd!1
  if (X'>15)
   act.cmd!0
  wait 0.01
 }
}
```

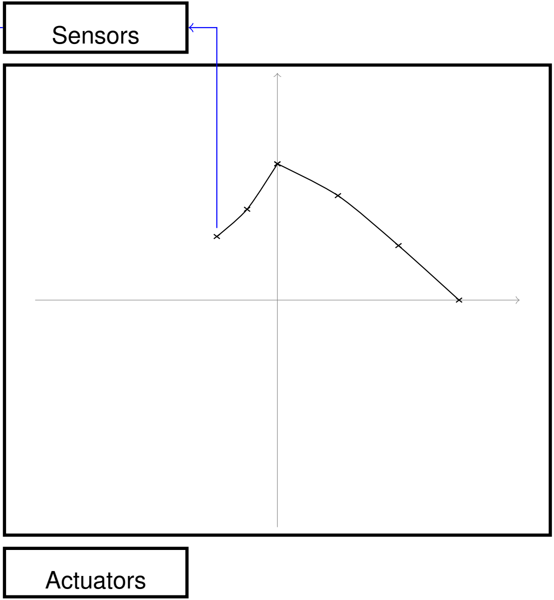
Sensors



Actuators

Example of an execution.

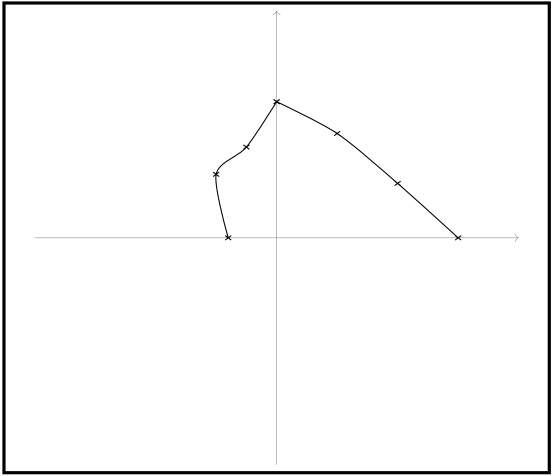
```
int main() {
 while(true) {
  sens.y?X
  X'=update(X);
  if (X'<0)
   act.cmd!1
  if (X'>15)
   act.cmd!0
  wait 0.01
 }
}
```



Example of an execution.

```
int main() {
 while(true) {
  sens.y?X
  X'=update(X);
  if (X'<0)
   act.cmd!1
  if (X'>15)
   act.cmd!0
  wait 0.01
 }
}
```

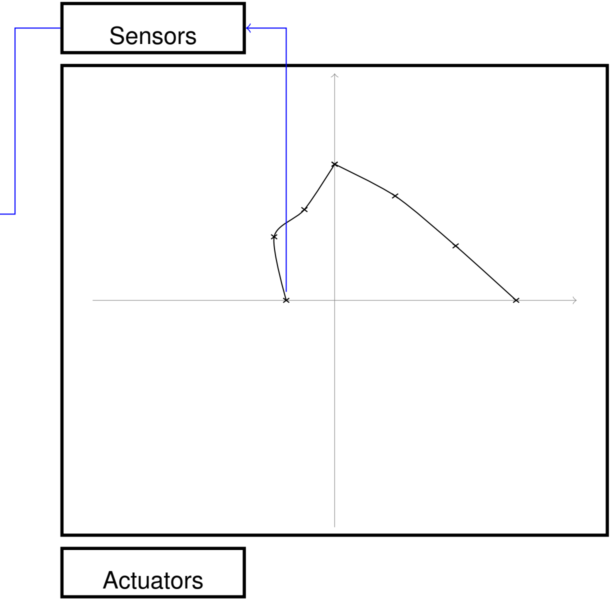
Sensors



Actuators

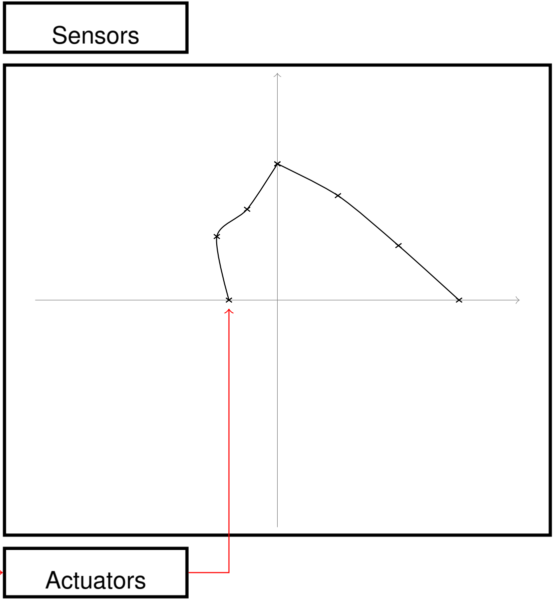
Example of an execution.

```
int main() {
 while(true) {
  sens.y?X
  X'=update(X);
  if (X'<0)
   act.cmd!1
  if (X'>15)
   act.cmd!0
  wait 0.01
 }
}
```



Example of an execution.

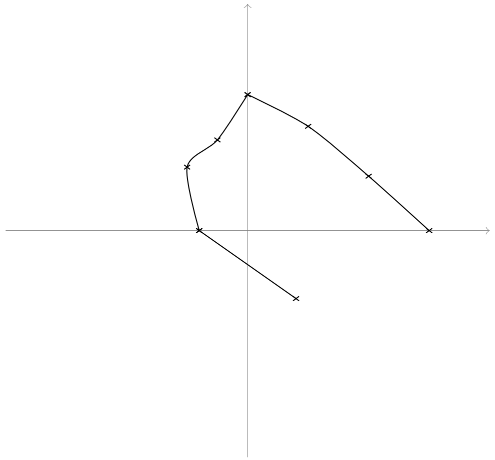
```
int main() {
 while(true) {
  sens.y?X
  X'=update(X);
  if (X'<0)
   act.cmd!1
  if (X'>15)
   act.cmd!0
  wait 0.01
 }
}
```



Example of an execution.

```
int main() {
 while(true) {
  sens.y?X
  X'=update(X);
  if (X'<0)
   act.cmd!1
  if (X'>15)
   act.cmd!0
  wait 0.01
 }
}
```

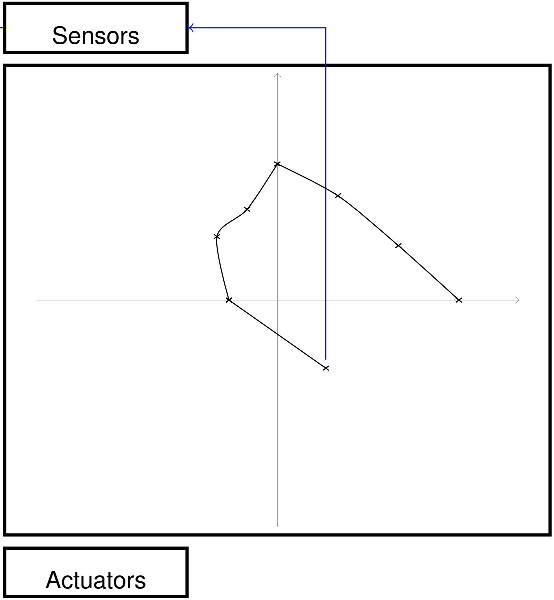
Sensors



Actuators

Example of an execution.

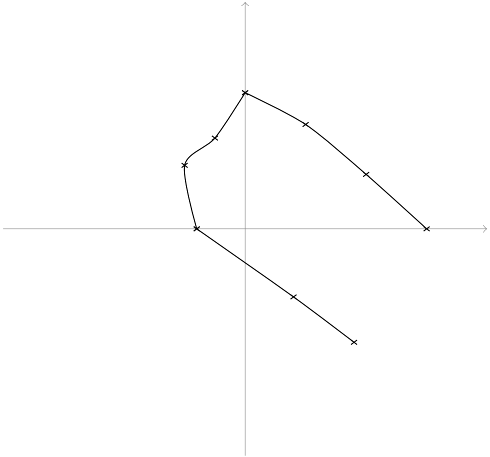
```
int main() {
 while(true) {
   sens.y?X
   X'=update(X);
   if (X'<0)
    act.cmd!1
   if (X'>15)
    act.cmd!0
   wait 0.01
  }
}
```



Example of an execution.

Sensors

```
int main() {
 while(true) {
  sens.y?X
  X'=update(X);
  if (X'<0)
   act.cmd!1
  if (X'>15)
   act.cmd!0
  wait 0.01
 }
}
```



Actuators

Example of an analysis.

[-10,10]

```
int main() {
 while(true) {
  sens.y?X
  X'=update(X);
  if (X'<0)
   act.cmd!1
  if (X'>15)
   act.cmd!0
  wait 0.01
 }
}
```

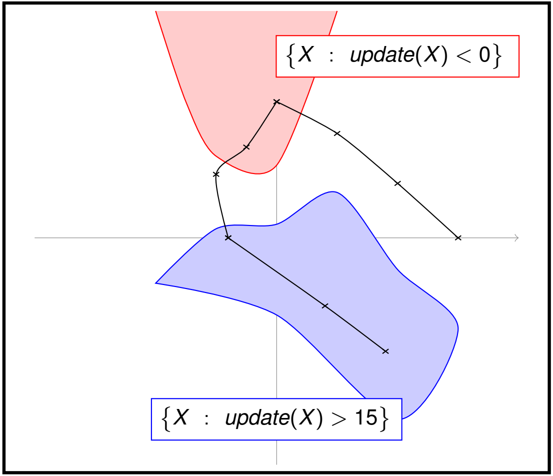
- Overapproximation of the continuous inputs.
- Forget about the actuators.
- Standard abstract interpretation analysis.
- Problem : overapproximation of the results because we lose the relation between two consecutives `sens`.

Actuators

Our method for the analysis.

1. Analyse the program to detect the "switching regions"

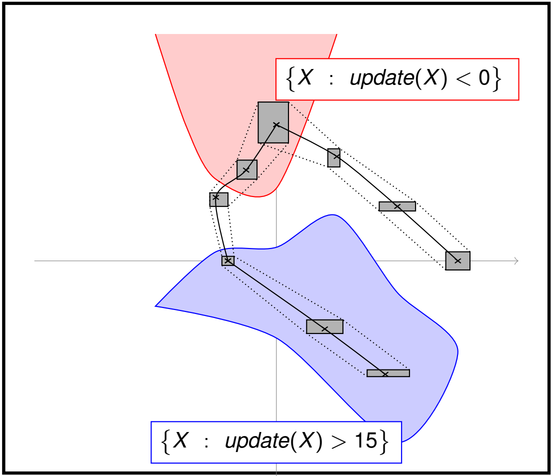
```
int main() {
 while(true) {
  sens.y?X
  X'=update(X);
  if (X'<0)
   act.cmd!1
  if (X'>15)
   act.cmd!0
  wait 0.01
 }
}
```



Our method for the analysis.

1. Analyse the program to detect the “switching regions”

2. Use guaranteed integration to overapproximate the trajectories.



Detecting the switching regions: abstracting a boolean function.

```
int main() {
 while(true) {
  sens.y?X
  X'=update(X);
  if (X'<0)
   act.cmd!1
  if (X'>15)
   act.cmd!0
  wait 0.01
 }
}
```

- A switch occur when an `act` statement is executed.
- We need to test if, given a value `X` for the sensor inputs, the statement on line n will be executed.
- Boolean function φ:

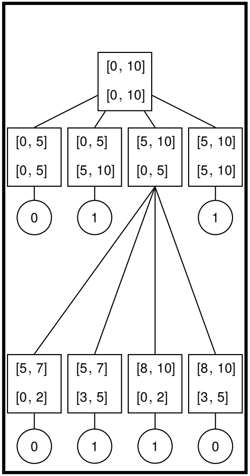
  given an input X, is the statement on line n executed?

- We need to construct the zones $\{X : \varphi(X) = 0\}$ and $\{X : \varphi(X) = 1\}$, i.e. build a representation of φ.

Problems:

- floating point intervals and not integer functions;
- representation of boolean functions;
- computing this representation.

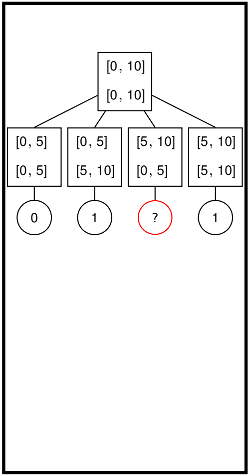
Representation of a boolean function : quad trees



Quad tree:

+ representation of a function $f : \mathbb{R}^n \rightarrow \mathbb{B}$;
+ two kinds of nodes :
  - non-terminal nodes with a box as attribute and 2^n children;
  - terminal node with a value $v \in \{0, 1\}$ as attribute.

Representation of a boolean function : quad trees



Quad tree:

+ representation of a function $f : \mathbb{R}^n \to \mathbb{B}$;
+ two kinds of nodes :
  - non-terminal nodes with a box as attribute and 2^n children;
  - terminal node with a value $v \in \{0, 1\}$ as attribute.
+ abstract quad trees: terminal value $v \in \{0, 1, ?\}$.
+ the precision depends on the depth.

A simple branch and bound algorithm.

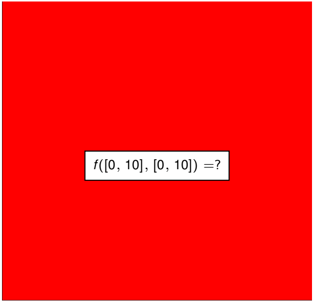
Problem

Given a function $f : \mathbb{R}^n \to \mathbb{B}$ and a precision N, construct the corresponding abstract quad tree.

Example :

$f(x, y) = d((x, y), (3.5, 6)) \leq 1 \vee d((x, y), (7, 1.2)) \leq 1$

- $x = [0, 10]$ and $y = [0, 10]$
- $d((x, y), (3.5, 6)) = [0, 78.25]$
- $d\big((x, y), (3.5, 6)\big) \leq 1 =?$
- $d\big((x, y), (7, 1.2)\big) \leq 1 =?$
- $f(x, y) =?$



Iterate 1

A simple branch and bound algorithm.

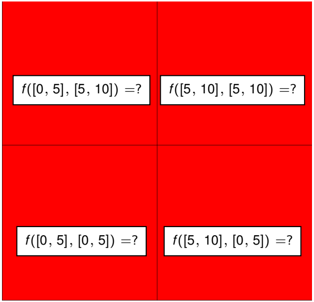
Problem

Given a function $f : \mathbb{R}^n \rightarrow \mathbb{B}$ and a precision N, construct the corresponding abstract quad tree.

Example :

$f(x, y) = d((x, y), (3.5, 6)) \leq 1 \vee d((x, y), (7, 1.2)) \leq 1$

- $x = [0, 5]$ and $y = [0, 5]$
- $d((x, y), (7, 1.2)) \subseteq [4, \infty[$
- $d\big((x, y), (7, 1.2)\big) \leq 1 = 0$
- $d((x, y), (3.5, 6)) = [1, 48.25]$
- $d\big((x, y), (3.5, 6)\big) \leq 1 = ?$
- $f(x, y) = ?$



Iterate 2

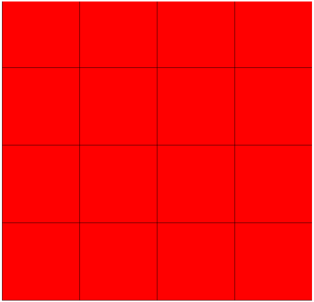
A simple branch and bound algorithm.

Problem

Given a function $f : \mathbb{R}^n \rightarrow \mathbb{B}$ and a precision N, construct the corresponding abstract quad tree.

Example :

$f(x, y) = d((x, y), (3.5, 6)) \leq 1 \vee d((x, y), (7, 1.2)) \leq 1$



Iterate 4

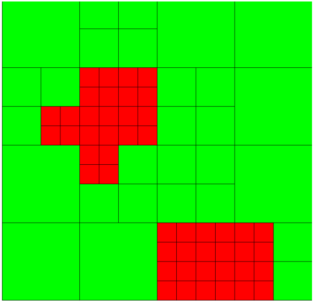
A simple branch and bound algorithm.

Problem

Given a function $f : \mathbb{R}^n \to \mathbb{B}$ and a precision N, construct the corresponding abstract quad tree.

Example :

$f(x,y) = d((x,y),(3.5,6)) \leq 1 \vee d((x,y),(7,1.2)) \leq 1$



Iterate 8

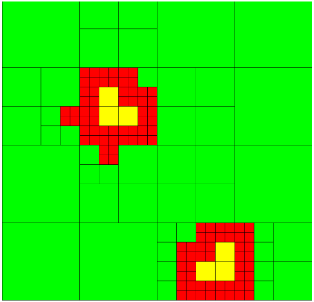
A simple branch and bound algorithm.

Problem

Given a function $f : \mathbb{R}^n \to \mathbb{B}$ and a precision N, construct the corresponding abstract quad tree.

Example :

$f(x, y) = d((x, y), (3.5, 6)) \leq 1 \vee d((x, y), (7, 1.2)) \leq 1$



Iterate 10

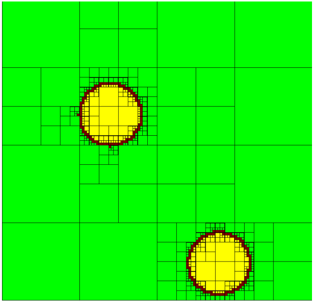
A simple branch and bound algorithm.

Problem

Given a function $f : \mathbb{R}^n \rightarrow \mathbb{B}$ and a precision N, construct the corresponding abstract quad tree.

Example :

$$f(x, y) = d((x, y), (3.5, 6)) \leq 1 \vee d((x, y), (7, 1.2)) \leq 1$$



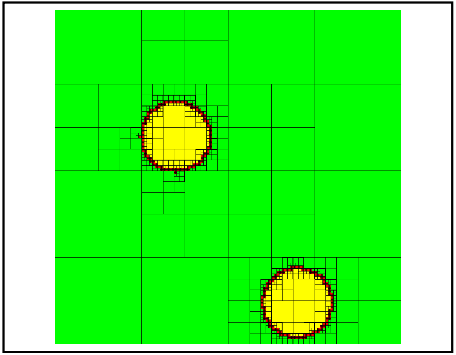
Iterate 16

Experimental results.

For the reachability problem.

The boolean function is a simple interval based interpreter that tests if a given line of code is executed.

```
double x,y;
/*!npk x = [0,10] */
/*!npk y = [0,10] */


double distance (double x,double y,
                 double a,double b) {
  double res;
  res = (x-a)*(x-a) + (y-b)*(y-b);
  return res;
}


void main() {
  double res,res2;
  int v;

  res = distance (x,y,3.5,6);
  res2 = distance (x,y,7,1.2);
  if ((res<=1)||(res2<=1)) {
    v=0;
  }

}
```

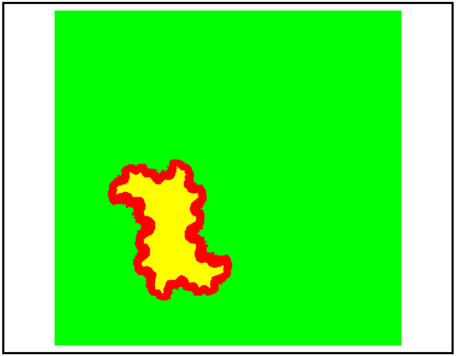


Experimental results.

For the reachability problem.

The boolean function is a simple interval based interpreter that tests if a given line of code is executed.

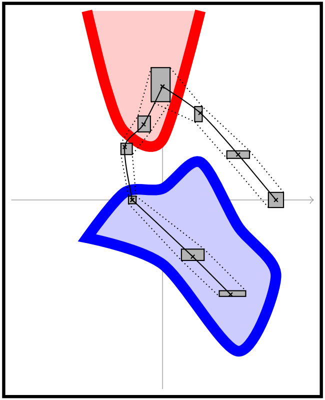
```
/* !npk x = [-2,4] */
/* !npk y = [-2,4] */
double x,y;

int update (double x,double y) {
  double a,b,xt,yt,d;
  int i = 0;
  a = 0.32; b = 0.43; d = 0;
  while (d<10) {
    xt = x*x-y*y+a;
    yt = 2*x*y + b;
    x = xt;    y = yt;
    d=x*x+y*y;
    if (i>=10)
      d = 30;
    i++;
  }
  return i;
}

void main() {
  int v = 0 , i = 0;
  i = update (x,y);
  if (i>=11)
    v = 1;
}
```



Guaranteed integration of ODEs.

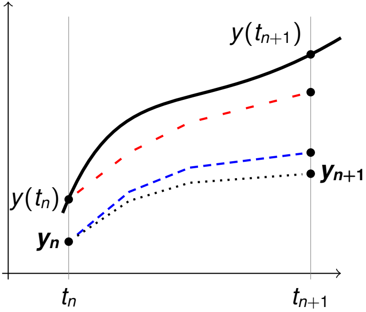
Result of the first analysis:

- we have a *spatial* criteria to detect the mode switchings;
- we know the sampling *times*, i.e. instants when the sensors return a value.



- The evolution of the continuous variables is given by ODEs.
- Given an ODE $\dot{y} = f(y)$ and a time t_n, we want to compute a box $\boldsymbol{y_n}$ such that $y(t_n) \in \boldsymbol{y_n}$.
- Existing validated methods: based on Taylor series decomposition.
- Existing non validated numerical methods: Euler, Runge-Kutta, Heun, ...

A new approach based on non validated methods.



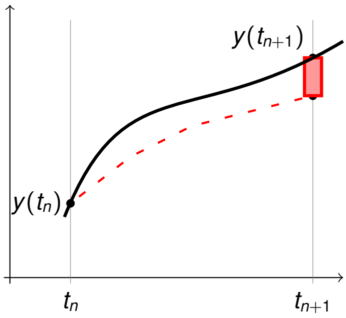
How to turn a numerical method into a validated one :

- A numerical method is $y_{n+1} = \Phi(y_n, h_n)$.
- Three sources of errors :
  - if $y_n = y(t_n)$ is exact, y_{n+1} differs from $y(t_{n+1})$;
  - if y_n has an error ϵ_n, how is it propagated into the next step;
  - floating point computations.

First implementation: with RK4

$$k_1 = f(y_n) \qquad k_2 = f\left(y_n + \frac{h}{2}k_1\right)$$
$$k_3 = f\left(y_n + \frac{h}{2}k_2\right) \qquad k_4 = f(y_n + hk_3)$$
$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

GRKLib : three sources of errors.



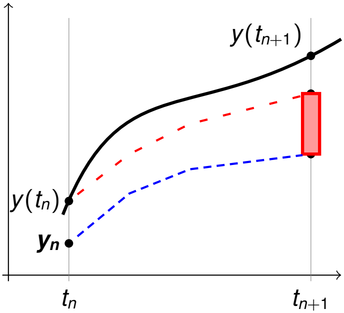
1. One step error:

- the function that gives y_{n+1} differs from $y(t)$.
- $y_{n+1} = \varphi(t_{n+1})$
- $\forall i \in [0,4],\ \frac{d^i y}{dt^i}(t_n) = \frac{d^i \varphi_n}{dt^i}(t_n)$
- $\exists t' \in [t_n, t_{n+1}]$ such that $y(t_{n+1}) - \varphi_n(t_{n+1}) = \frac{d^5(y-\varphi_n)}{dt^5}(t')$

$$y(t_{n+1}) - \varphi_n(t_{n+1}) \in \frac{d^4 f}{dt^4}(\tilde{y}) - \frac{d^5 \varphi_n}{dt^5}([t_n, t_{n+1}])$$

$$\epsilon_{n+1} = \frac{d^4 f}{dt^4}(\tilde{y}) - \frac{d^5 \varphi_n}{dt^5}([t_n, t_{n+1}])$$

GRKLib : three sources of errors.



2. Error propagation:

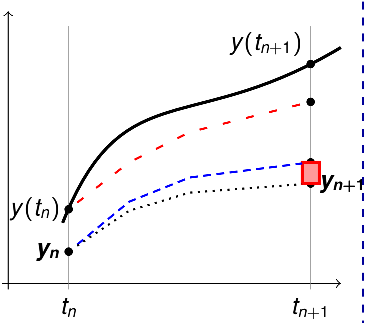
- starting point is $\boldsymbol{y_n}$ instead of $y(t_n)$,
- the propagated error is the difference between $\psi(y_n)$ and $\psi(y(t_n))$
- $\exists y \in [y_n, y(t_n)]$ such that $\chi_{n+1} = J(\psi, y).(y_n - y(t_n))$

$$\chi_n \in J(\psi, [y_n, y_n + \epsilon_n]).\epsilon_n$$

- We use QR-factorization technique to reduce the wrapping effect.

$$\epsilon_{n+1} = \frac{d^4 f}{dt^4}(\tilde{y}) - \frac{d^5 \varphi_n}{dt^5}([t_n, t_{n+1}]) + J(\psi, [y_n, y_n + \epsilon_n]).\epsilon_n$$

GRKLib : three sources of errors.



3. Computation error:

- we use floating point numbers and not real numbers to compute y_{n+1};
- we must overapproximate the computation erros;
- Solution: *global error arithmetic*

$$[\![a]\!]_{\mathbb{E}} = f_a + e_a\overrightarrow{\varepsilon_e} \;\; and \;\; [\![b]\!]_{\mathbb{E}} = f_b + e_b\overrightarrow{\varepsilon_e}$$

$$[\![a+b]\!]_{\mathbb{E}} = \uparrow_{\sim} (f_a + f_b) + (e_a + e_b + \downarrow_{\sim} (f_a + f_b))\overrightarrow{\varepsilon_e}$$

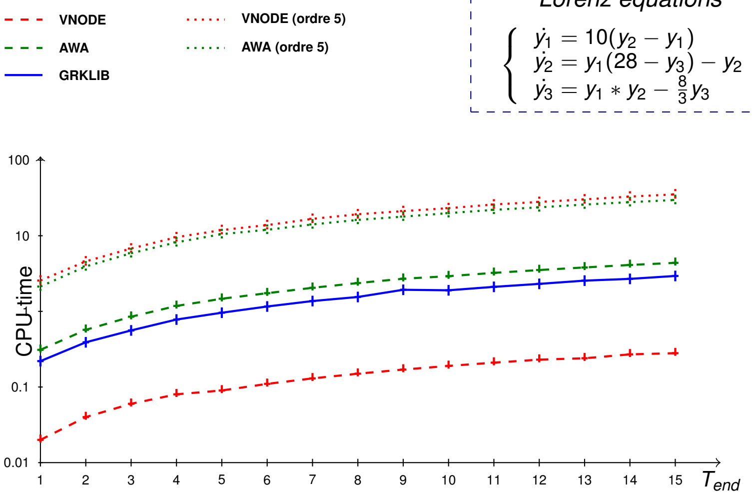
- We use this arithmetic to compute y_{n+1}, and thus get an interval E_{n+1} that contains all the computation erros.

$$\epsilon_{n+1} = \frac{d^4 f}{dt^4}(\tilde{y}) - \frac{d^5 \varphi_n}{dt^5}([t_n, t_{n+1}]) + J(\psi, [y_n, y_n + \epsilon_n]).\epsilon_n + E_{n+1}$$

Benchmarks: CPU time VS T_{end}

Lorenz equations

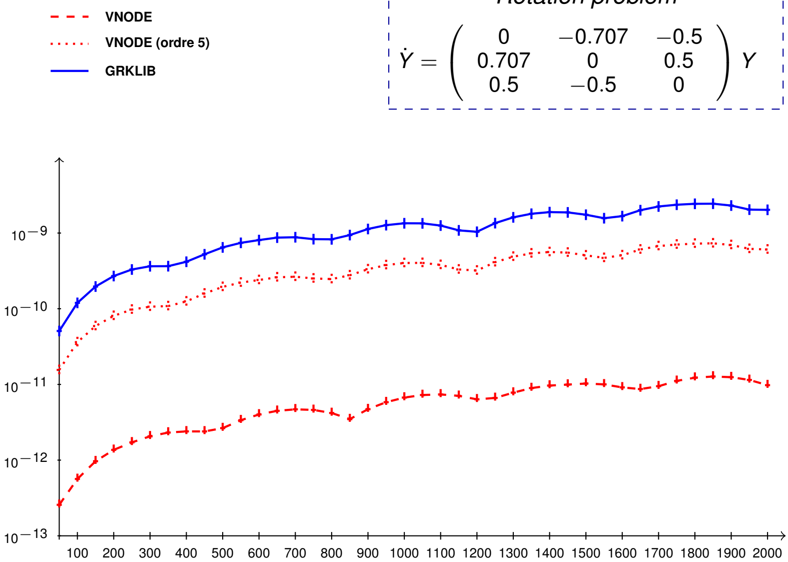
$$\begin{cases} \dot{y}_1 = 10(y_2 - y_1) \\ \dot{y}_2 = y_1(28 - y_3) - y_2 \\ \dot{y}_3 = y_1 * y_2 - \frac{8}{3} y_3 \end{cases}$$



Benchmarks: Enclosure width VS T_{end}

Rotation problem

$$\dot{Y} = \begin{pmatrix} 0 & -0.707 & -0.5 \\ 0.707 & 0 & 0.5 \\ 0.5 & -0.5 & 0 \end{pmatrix} Y$$



Conclusion

Formal verification of embedded softwares

- considering the program alone showed its limits;
- we must take the physical environment into account.

Interval methods:

- allow to abstract the program and thus simplify the analysis;
- allow to compute a safe overapproximation of the continuous dynamics.

Future work:

- unify both analysis into a common tool;
- improve GRKLib by adding higher order methods and by using other domains (zonotopes).