A Hybrid Denotational Semantics for Hybrid Systems

Olivier BOUISSOU

CEA Saclay
Laboratoire Modélisation et Analyse de Systèmes en Intéraction

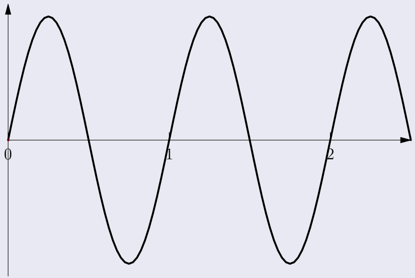
October 1, 2007

Motivating Example.

Program

```
#define SUP 4
#define INF -4
#define h 1/8.0

volatile float x;
static float intgrx=0.0;
void main() {
   float xi;
   while (true) {
     xi = x;
     intgrx += xi*h;
     if (intgrx > SUP)
       intgrx = SUP;
     if (intgrx < INF)
       intgrx = INF;
   }
}
```

Incoming data

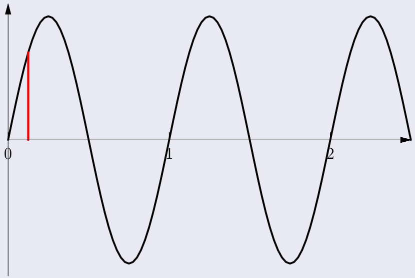


Motivating Example.

Program

```
#define SUP 4
#define INF -4
#define h 1/8.0

volatile float x;
static float intgrx=0.0;
void main() {
   float xi;
   while (true) {
     xi = x;
     intgrx += xi*h;
     if (intgrx > SUP)
       intgrx = SUP;
     if (intgrx < INF)
       intgrx = INF;
   }
}
```

Incoming data



Simulation (dynamical analysis)

Value of `intgrx` after 1 iteration:

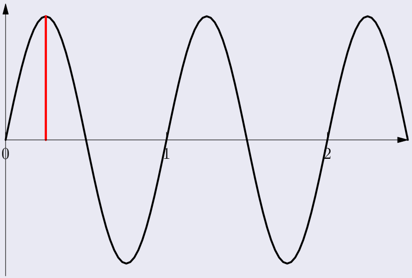
0.08838834762

Motivating Example.

Program

```
#define SUP 4
#define INF -4
#define h 1/8.0

volatile float x;
static float intgrx=0.0;
void main() {
   float xi;
   while (true) {
     xi = x;
     intgrx += xi*h;
     if (intgrx > SUP)
       intgrx = SUP;
     if (intgrx < INF)
       intgrx = INF;
   }
}
```

Incoming data



Simulation (dynamical analysis)

Value of `intgrx` after 2 iterations:

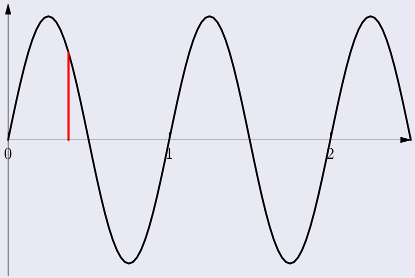
0.2133883476

Motivating Example.

Program

```
#define SUP 4
#define INF −4
#define h 1/8.0

volatile float x;
static float intgrx=0.0;
void main() {
   float xi;
   while (true) {
     xi = x;
     intgrx += xi∗h;
     if (intgrx > SUP)
       intgrx = SUP;
     if (intgrx < INF)
       intgrx = INF;
   }
}
```

Incoming data



Simulation (dynamical analysis)

Value of `intgrx` after 3 iterations:

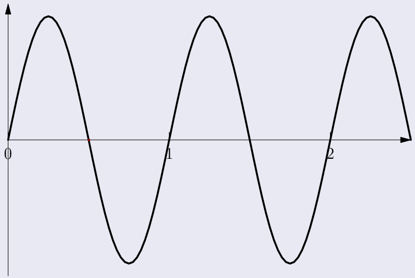
0.3017766952

Motivating Example.

Program

```
#define SUP 4
#define INF -4
#define h 1/8.0

volatile float x;
static float intgrx=0.0;
void main() {
   float xi;
   while (true) {
     xi = x;
     intgrx += xi*h;
     if (intgrx > SUP)
       intgrx = SUP;
     if (intgrx < INF)
       intgrx = INF;
   }
}
```

Incoming data



Simulation (dynamical analysis)

Value of `intgrx` after 4 iterations:

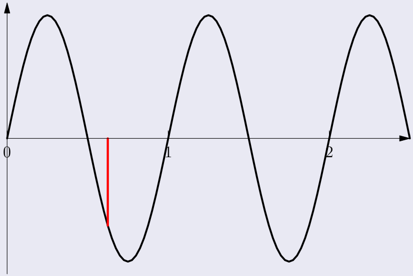
0.3017766952

Motivating Example.

Program

```
#define SUP 4
#define INF -4
#define h 1/8.0

volatile float x;
static float intgrx=0.0;
void main() {
   float xi;
   while (true) {
     xi = x;
     intgrx += xi*h;
     if (intgrx > SUP)
       intgrx = SUP;
     if (intgrx < INF)
       intgrx = INF;
   }
}
```

Incoming data



Simulation (dynamical analysis)

Value of `intgrx` after 5 iterations:

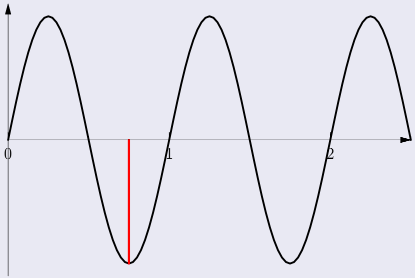
0.2133883476

Motivating Example.

Program

```
#define SUP 4
#define INF -4
#define h 1/8.0

volatile float x;
static float intgrx=0.0;
void main() {
   float xi;
   while (true) {
     xi = x;
     intgrx += xi*h;
     if (intgrx > SUP)
       intgrx = SUP;
     if (intgrx < INF)
       intgrx = INF;
   }
}
```

Incoming data



Simulation (dynamical analysis)

Value of `intgrx` after 6 iterations:

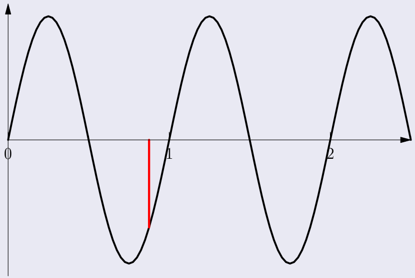
0.08838834762

Motivating Example.

Program

```
#define SUP 4
#define INF −4
#define h 1/8.0

volatile float x;
static float intgrx=0.0;
void main() {
   float xi;
   while (true) {
     xi = x;
     intgrx += xi*h;
     if (intgrx > SUP)
       intgrx = SUP;
     if (intgrx < INF)
       intgrx = INF;
   }
}
```

Incoming data



Simulation (dynamical analysis)

Value of `intgrx` after 7 iterations:

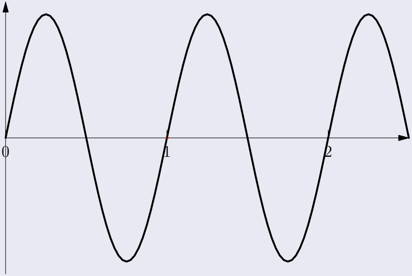
0.

Motivating Example.

Program

```
#define SUP 4
#define INF -4
#define h 1/8.0

volatile float x;
static float intgrx=0.0;
void main() {
   float xi;
   while (true) {
     xi = x;
     intgrx += xi*h;
     if (intgrx > SUP)
       intgrx = SUP;
     if (intgrx < INF)
       intgrx = INF;
   }
}
```

Incoming data



Simulation (dynamical analysis)

Value of `intgrx` after 8 iterations:

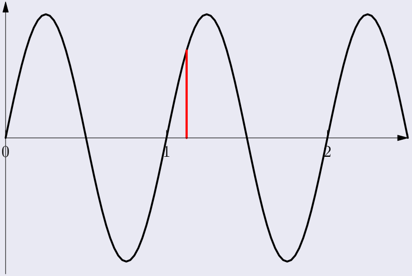
0.

Motivating Example.

Program

```
#define SUP 4
#define INF -4
#define h 1/8.0

volatile float x;
static float intgrx=0.0;
void main() {
   float xi;
   while (true) {
     xi = x;
     intgrx += xi*h;
     if (intgrx > SUP)
       intgrx = SUP;
     if (intgrx < INF)
       intgrx = INF;
   }
}
```

Incoming data



Simulation (dynamical analysis)

Value of `intgrx` after 9 iterations:

0.08838834762

Motivating Example: Analysis

Program

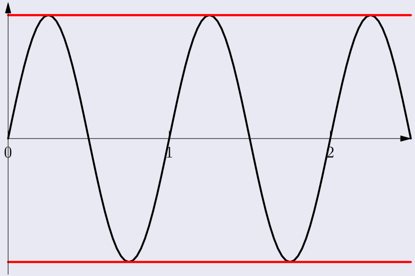
```
#define SUP 4
#define INF -4
#define h 1/8.0

volatile float x;
static float intgrx=0.0;
void main() {
   float xi;
   while (true) {

     xi = x;
     intgrx += xi*h;
     if (intgrx > SUP)
       intgrx = SUP;
     if (intgrx < INF)
       intgrx = INF;
   }
}
```

Simulation (dynamical analysis)

Value of `intgrx` after 9 iterations:

0.08838834762

Incoming data: abstraction



Static analysis

Value of `intgrx` after 1 iteration:

[-h,h]

Motivating Example: Analysis

Program

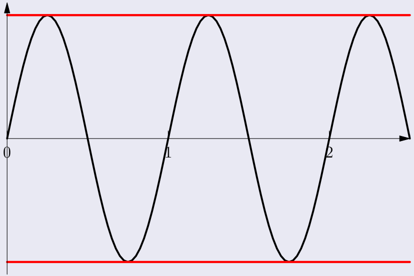
```
#define SUP 4
#define INF -4
#define h 1/8.0

volatile float x;
static float intgrx=0.0;
void main() {
   float xi;
   while (true) {

      xi = x;
      intgrx += xi*h;
      if (intgrx > SUP)
        intgrx = SUP;
      if (intgrx < INF)
        intgrx = INF;
   }
}
```

Simulation (dynamical analysis)

Value of `intgrx` after 9 iterations:

0.08838834762

Incoming data: abstraction



Static analysis

Value of `intgrx` after 2 iterations:

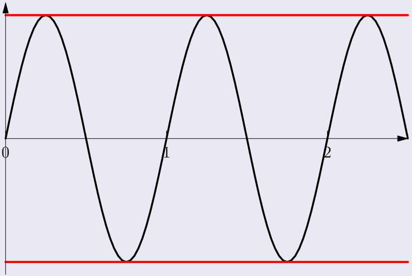
[-2*h,2*h]

Motivating Example: Analysis

Program

```
#define SUP 4
#define INF -4
#define h 1/8.0

volatile float x;
static float intgrx=0.0;
void main() {
   float xi;
   while (true) {

      xi = x;
      intgrx += xi*h;
      if (intgrx > SUP)
         intgrx = SUP;
      if (intgrx < INF)
         intgrx = INF;
   }
}
```

Incoming data: abstraction



Simulation (dynamical analysis)

Value of `intgrx` after 9 iterations:

0.08838834762

Static analysis

Value of `intgrx` after 3 iterations:

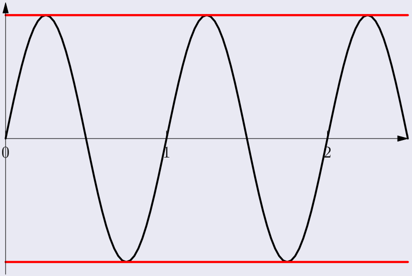
[-3*h,3*h]

Motivating Example: Analysis

Program

```
#define SUP 4
#define INF −4
#define h 1/8.0

volatile float x;
static float intgrx=0.0;
void main() {
   float xi;
   while (true) {

     xi = x;
     intgrx += xi*h;
     if (intgrx > SUP)
       intgrx = SUP;
     if (intgrx < INF)
       intgrx = INF;
   }
}
```

Incoming data: abstraction



Simulation (dynamical analysis)

Value of `intgrx` after 9 iterations:

0.08838834762

Static analysis

Value of `intgrx` after 4 iterations:

[-4*h,4*h]

Motivating Example: Analysis

Program

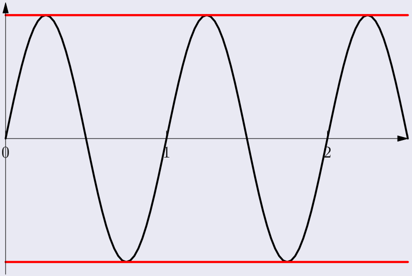
```
#define SUP 4
#define INF −4
#define h 1/8.0

volatile float x;
static float intgrx=0.0;
void main() {
   float xi;
   while (true) {

      xi = x;
      intgrx += xi*h;
      if (intgrx > SUP)
        intgrx = SUP;
      if (intgrx < INF)
        intgrx = INF;
   }
}
```

Simulation (dynamical analysis)

Value of `intgrx` after 9 iterations:

0.08838834762

Incoming data: abstraction



Static analysis

Value of `intgrx` after 4 iterations:

[-4*h,4*h]

Extrapolation (widening):

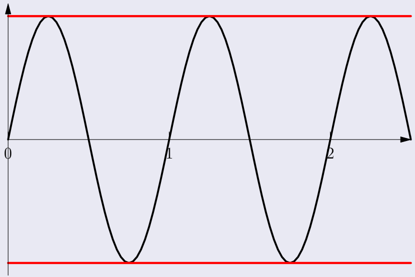
`intgrx`=[-∞,∞]

Motivating Example: Analysis

Program

```
#define SUP 4
#define INF -4
#define h 1/8.0

volatile float x;
static float intgrx=0.0;
void main() {
   float xi;
   while (true) {

     xi = x;
     intgrx += xi*h;
     if (intgrx > SUP)
       intgrx = SUP;
     if (intgrx < INF)
       intgrx = INF;
   }
}
```

Incoming data: abstraction



Simulation (dynamical analysis)

Value of `intgrx` after 9 iterations:

0.08838834762

Static analysis

Value of `intgrx` after 4 iterations:

[-4*h,4*h]

Extrapolation (widening):
`intgrx`=[-∞,∞]
Limitation (narrowing): `intgrx`=[-4,4]

Problem

Loss of precision because:

- we assume that `x` can instantaneously jump from -1 to 1
- we do not consider extra information about `x` (incoming rate, continuous evolution)

Proposed solution:

- analyze the program together with its physical environment
- introduce hybrid statements to the program

Difficulties:

- program and environment have different behaviors:
  - program is discrete
  - environment is continuous
- need to express both in a unified semantics

Hybrid Syntax

Program

```
#define SUP 4
#define INF -4
#define h 1/8.0
sensor x;
actuator k;
static float intgrx=0.0;
void main() {
   float xi;
   while (true) {
     wait(h);
     sens.x?xi;
     intgrx += xi*h;
     if (intgrx > SUP) {
       intgrx = SUP;
       act.k!0;
     }
     if (intgrx < INF)
       intgrx = INF;
   }
}
```

Environment

$$\begin{cases} \dot{x} & = & k * y \\ \dot{y} & = & -k * x \end{cases}$$

We write it:

$$\dot{Y} = F_k(Y)$$

Assumptions for F_k:

- continuous
- piecewise Lipschitz

Goal of the talk

Give a denotational semantics to this system: the semantics is computed using only *one* fix-point.

Discrete Semantics

Assumption

The environment is perfectly known.

- Semantics of discrete statements: textbook

$$
\begin{array}{rcl}
[\![e1+e2]\!] & = & \{(\sigma, n_1+n_2) \mid (\sigma, n_1) \in [\![e1]\!] \text{ and } (\sigma, n_2) \in \sigma\} \\
[\![v<e]\!] & = & \{(\sigma, \mathit{true}) : [\![v]\!]\sigma < [\![e]\!]\sigma\} \\
 & & \cup \{(\sigma, \mathit{false}) : [\![v]\!]\sigma \geq [\![e]\!]\sigma\} \\
[\![v=e]\!] & = & \{(\sigma, \sigma') \mid \sigma' = \sigma\,[v \mapsto n] \text{ and } (\sigma, n) \in [\![e]\!]\} \\
[\![\mathbf{while}(b)\ \mathit{inst}]\!] & = & \mathit{Fix}(\Gamma) \text{ with } \Gamma(\varphi) = \{(\sigma, \sigma') \mid [\![b]\!]\sigma = \mathit{true} \text{ and } (\sigma, \sigma') \in \varphi \circ [\![\mathit{inst}]\!]\} \\
[\![i_1 ; i_2]\!] & = & [\![i_2]\!] \circ [\![i_1]\!]
\end{array}
$$

- Semantics of hybrid statements:

$$
\begin{array}{rcl}
[\![\mathbf{sens}.\mathrm{y?x}]\!] & = & \text{Binds x with the present value of y} \\
[\![\mathbf{wait}\ c]\!] & = & \text{Move time forward of c seconds} \\
[\![\mathbf{act}.\mathrm{k!c}]\!] & = & \text{Chose the function } y_c \text{ for the future}
\end{array}
$$

Discrete Semantics

Assumption

The environment is perfectly known.

- Semantics of discrete statements: textbook

$$
\begin{array}{rcl}
[\![e1 + e2]\!] & = & \{(\sigma, n_1 + n_2) \mid (\sigma, n_1) \in [\![e1]\!] \text{ and } (\sigma, n_2) \in \sigma\} \\
[\![v < e]\!] & = & \{(\sigma, \mathit{true}) : [\![v]\!]\sigma < [\![e]\!]\sigma\} \\
 & & \cup \{(\sigma, \mathit{false}) : [\![v]\!]\sigma \geq [\![e]\!]\sigma\} \\
[\![v = e]\!] & = & \{(\sigma, \sigma') \mid \sigma' = \sigma\,[v \mapsto n] \text{ and } (\sigma, n) \in [\![e]\!]\} \\
[\![\textbf{while}(b)\ \mathit{inst}]\!] & = & \mathit{Fix}(\Gamma) \text{ with } \Gamma(\varphi) = \{(\sigma, \sigma') \mid [\![b]\!]\sigma = \mathit{true} \text{ and } (\sigma, \sigma') \in \varphi \circ [\![\mathit{inst}]\!]\} \\
[\![i_1 ; i_2]\!] & = & [\![i_2]\!] \circ [\![i_1]\!]
\end{array}
$$

- Semantics of hybrid statements:

$$
\begin{array}{rcl}
[\![\textbf{sens}.\mathrm{y?x}]\!] & = & \{(\sigma, \sigma') \mid \sigma' = \sigma[x \mapsto \sigma.y(\sigma.\mathit{time})]\} \\
[\![\textbf{wait}\ c]\!] & = & \text{Move time forward of c seconds} \\
[\![\textbf{act}.\mathrm{k!c}]\!] & = & \text{Chose the function } y_c \text{ for the future}
\end{array}
$$

Discrete Semantics

Assumption

The environment is perfectly known.

- Semantics of discrete statements: textbook

$$
\begin{aligned}
[\![e1 + e2]\!] &= \{(\sigma, n_1 + n_2) \mid (\sigma, n_1) \in [\![e1]\!] \text{ and } (\sigma, n_2) \in \sigma\} \\
[\![v < e]\!] &= \{(\sigma, \mathit{true}) : [\![v]\!]\sigma < [\![e]\!]\sigma\} \\
&\quad \cup \{(\sigma, \mathit{false}) : [\![v]\!]\sigma \geq [\![e]\!]\sigma\} \\
[\![v = e]\!] &= \{(\sigma, \sigma') \mid \sigma' = \sigma\,[v \mapsto n] \text{ and } (\sigma, n) \in [\![e]\!]\} \\
[\![\mathbf{while}(b)\ \mathit{inst}]\!] &= \mathit{Fix}(\Gamma) \text{ with } \Gamma(\varphi) = \{(\sigma, \sigma') \mid [\![b]\!]\sigma = \mathit{true} \text{ and } (\sigma, \sigma') \in \varphi \circ [\![\mathit{inst}]\!]\} \\
[\![i_1 ; i_2]\!] &= [\![i_2]\!] \circ [\![i_1]\!]
\end{aligned}
$$

- Semantics of hybrid statements:

$$
\begin{aligned}
[\![\mathbf{sens}.\mathrm{y?x}]\!] &= \{(\sigma, \sigma') | \sigma' = \sigma[x \mapsto \sigma.y(\sigma.\mathit{time})]\} \\
[\![\mathbf{wait}\ c]\!] &= \{(\sigma, \sigma') \mid \sigma' = \sigma[\mathit{time} \mapsto \sigma.\mathit{time} + c]\} \\
[\![\mathbf{act}.\mathrm{k!c}]\!] &= \text{Chose the function } y_c \text{ for the future}
\end{aligned}
$$

Discrete Semantics

Assumption

The environment is perfectly known.

- Semantics of discrete statements: textbook

$$
\begin{array}{rcl}
[\![e1 + e2]\!] & = & \{(\sigma, n_1 + n_2) \mid (\sigma, n_1) \in [\![e1]\!] \text{ and } (\sigma, n_2) \in \sigma\} \\
[\![v < e]\!] & = & \{(\sigma, \mathit{true}) : [\![v]\!]\sigma < [\![e]\!]\sigma\} \\
 & & \cup \{(\sigma, \mathit{false}) : [\![v]\!]\sigma \geq [\![e]\!]\sigma\} \\
[\![v = e]\!] & = & \{(\sigma, \sigma') \mid \sigma' = \sigma\,[v \mapsto n] \text{ and } (\sigma, n) \in [\![e]\!]\} \\
[\![\textbf{while}(b)\ \mathit{inst}]\!] & = & \mathit{Fix}(\Gamma) \text{ with } \Gamma(\varphi) = \{(\sigma, \sigma') \mid [\![b]\!]\sigma = \mathit{true} \text{ and } (\sigma, \sigma') \in \varphi \circ [\![\mathit{inst}]\!]\} \\
[\![i_1 ; i_2]\!] & = & [\![i_2]\!] \circ [\![i_1]\!]
\end{array}
$$

- Semantics of hybrid statements:

$$
\begin{array}{rcl}
[\![\textbf{sens}.\mathrm{y?x}]\!] & = & \{(\sigma, \sigma') | \sigma' = \sigma[x \mapsto \sigma.y(\sigma.\mathit{time})]\} \\
[\![\textbf{wait}\ c]\!] & = & \{(\sigma, \sigma') \mid \sigma' = \sigma[\mathit{time} \mapsto \sigma.\mathit{time} + c]\} \\
[\![\textbf{act}.\mathrm{k!c}]\!] & = & \left\{(\sigma, \sigma') | \sigma' = \sigma \left[y \mapsto \lambda x. \left\{ \begin{array}{ll} \sigma.y(x) & \mathit{if}\ x \leq \sigma.\mathit{time} \\ y_c(x) & \mathit{else} \end{array} \right. \right] \right\}
\end{array}
$$

Continuous Semantics

Assumption: the discrete evolution is known

- the switching times are known
- the evolution of the environment is governed by *one* IVP

Initial value problem (IVP):

- an ODE $\dot{y} = F(y)$ that governs the evolution
- an initial condition $y(0) = y_0$ that explicits the starting point

Solution an IVP:

- a function y such that $\forall t \in \mathbb{R}_+,\ \dot{y}(t) = F(y(t))$ and $y(0) = y_0$
- y verifies $y = y_0 + \int_0^x F\big(y(s)\big)ds$, i.e. y is a fixpoint of the PICARD operator $P : y \mapsto \lambda x.y_0 + \int_0^x F\big(y(s)\big)ds$

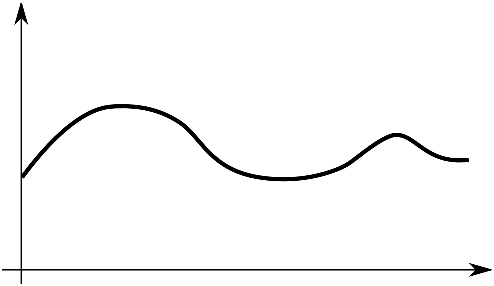
Goal of the continuous semantics

Express y as a the fix-point of a (possibly monotone) function on a lattice and show that it is the limit of Kleene's iterates.

Lattice of Partial Interval Valued Functions

Basic idea

Continuous functions defined on $[0, \infty[$ are elements with perfect information.
Construct a lattice that respect this information order.

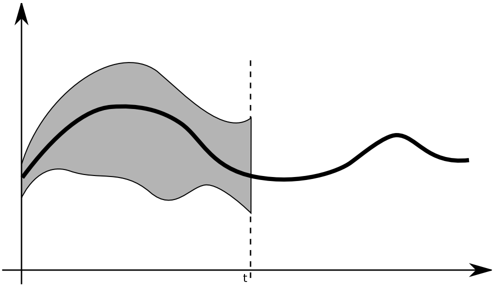


Lattice of Partial Interval Valued Functions

Basic idea

Continuous functions defined on $[0, \infty[$ are elements with perfect information. Construct a lattice that respect this information order.

- Approximation of a continuous function: interval valued function defined on $[0, X]$.

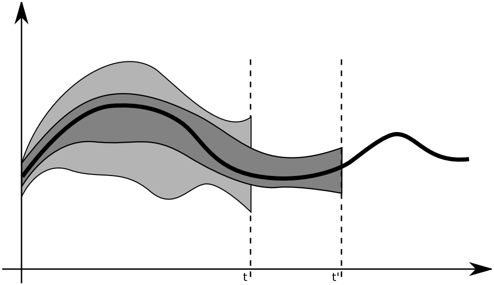


Lattice of Partial Interval Valued Functions

Basic idea

Continuous functions defined on $[0, \infty[$ are elements with perfect information. Construct a lattice that respect this information order.

- Approximation of a continuous function: interval valued function defined on $[0, X]$.
- A function that is defined on $[0, X + 1]$ and that is tighter is a better approximation.

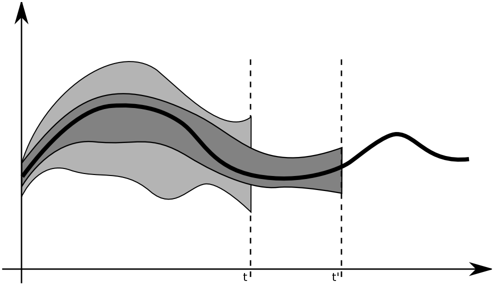


Lattice of Partial Interval Valued Functions

Basic idea

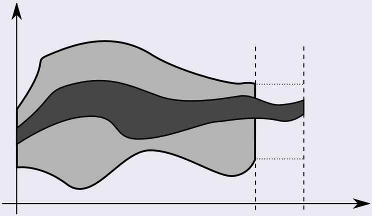
Continuous functions defined on $[0, \infty[$ are elements with perfect information. Construct a lattice that respect this information order.

- Approximation of a continuous function: interval valued function defined on $[0, X]$.
- A function that is defined on $[0, X+1]$ and that is tighter is a better approximation.
- Formally: $\mathcal{IF}_X^0 = \{f : [0, X] \to \mathbb{I}(\mathbb{R})\}$ such that the upper and the lower functions are continuous. $\mathcal{D} = \bigcup_{X \in \mathbb{R}_+} \mathcal{IF}_X^0 \cup \mathcal{IF}_\infty^0$.

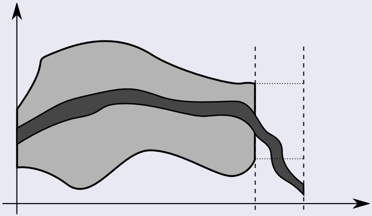


Lattice of Partial Functions: Order and Join

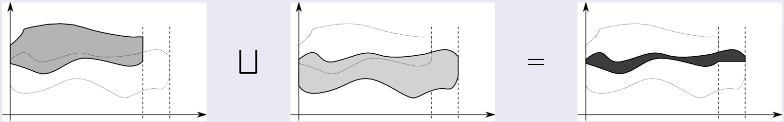
Comparable functions



Incomparable functions

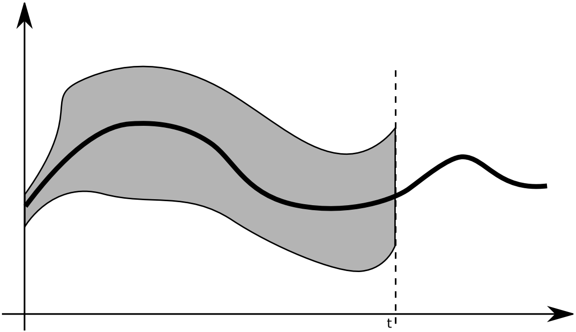


Join



Fixpoint Computation

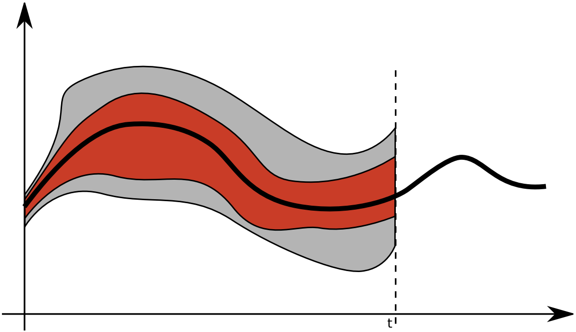
The semantic operator is a modified Picard operator that updates and extends the partial functions.



$$\Gamma_{F,y_0}(f)(x) = \begin{cases} y_0 + \int_0^x F(y(s))ds & \text{if } x \leq X_f \\ J + F(J) * [-e^{\alpha}, e^{\alpha}] * (x - X), & \\ \quad \text{with } J = y_0 + \int_0^X F\big(f(s)\big)ds & \text{otherwise} \end{cases}$$

Fixpoint Computation

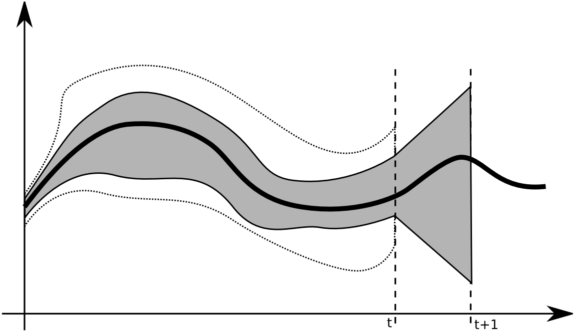
The semantic operator is a modified Picard operator that updates and extends the partial functions.



$$\Gamma_{F,y_0}(f)(x) = \begin{cases} {\color{red} y_0 + \int_0^x F(y(s))ds} & \text{if } x \leq X_f \\ J + F(J) * [-e^\alpha, e^\alpha] * (x - X), & \\ \quad \text{with } J = y_0 + \int_0^X F\big(f(s)\big)ds & \text{otherwise} \end{cases}$$

Fixpoint Computation

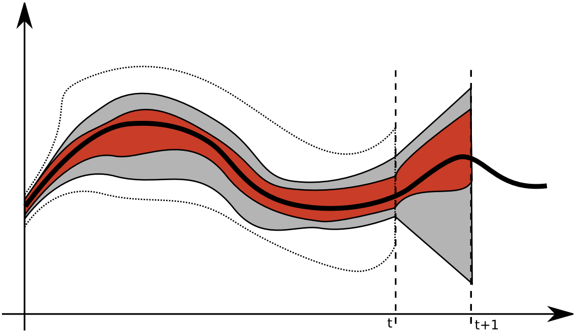
The semantic operator is a modified Picard operator that updates and extends the partial functions.



$$\Gamma_{F,y_0}(f)(x) = \begin{cases} y_0 + \int_0^x F(y(s))ds & \text{if } x \leq X_f \\ J + F(J) * [-e^\alpha, e^\alpha] * (x - X), & \\ \quad \text{with } J = P_{[0,X_f]}(F, y_0)(f)(X) & \text{otherwise} \end{cases}$$

Fixpoint Computation

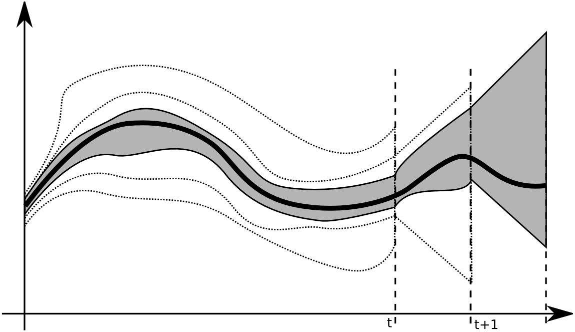
The semantic operator is a modified Picard operator that updates and extends the partial functions.



$$\Gamma_{F,y_0}(f)(x) = \begin{cases} y_0 + \int_0^x F(y(s))ds & \text{if } x \leq X_f \\ J + F(J) * [-e^{\alpha}, e^{\alpha}] * (x - X), & \\ \quad \text{with } J = y_0 + \int_0^X F\big(f(s)\big)ds & \text{otherwise} \end{cases}$$

Fixpoint Computation

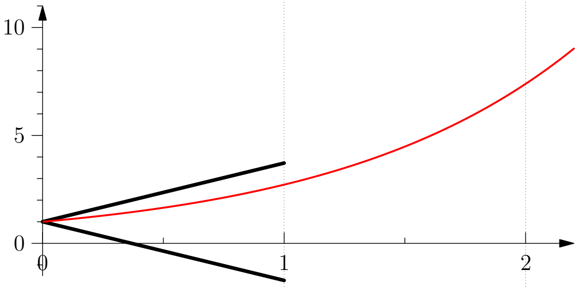
The semantic operator is a modified Picard operator that updates and extends the partial functions.



$$\Gamma_{F,y_0}(f)(x) = \begin{cases} y_0 + \int_0^x F(y(s))ds & \text{if } x \leq X_f \\ J + F(J) * [-e^{\alpha}, e^{\alpha}] * (x - X), & \\ \quad \text{with } J = P_{[0,X_f]}(F, y_0)(f)(X) & \text{otherwise} \end{cases}$$

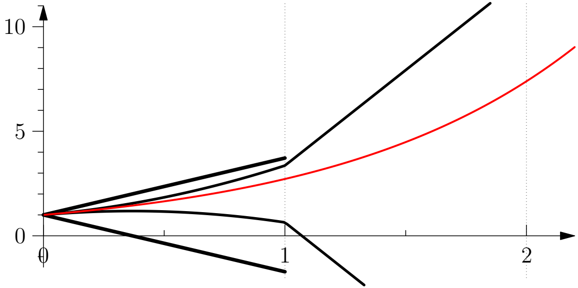
Kleene's Iteration

- We build a sequence of function f_n such that:
  - f_n defined on $[0, n]$
  - $\forall n,\ \forall x \in [0, n],\ y(x) \in f_n(x)$
- We use Keye Martin's measurement theory to prove that the sequence converges.



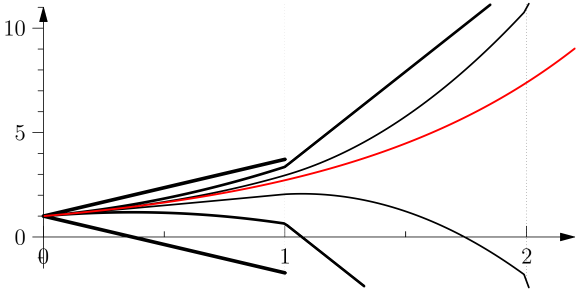
Kleene's Iteration

- We build a sequence of function f_n such that:
  - f_n defined on $[0, n]$
  - $\forall n,\ \forall x \in [0, n],\ y(x) \in f_n(x)$
- We use Keye Martin's measurement theory to prove that the sequence converges.



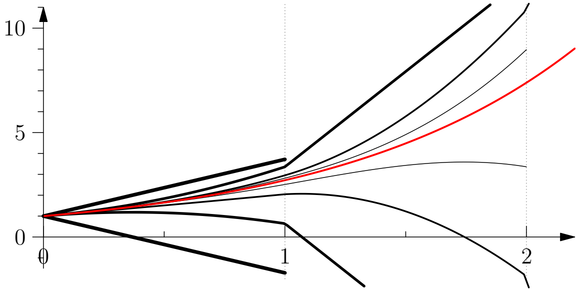
Kleene's Iteration

- We build a sequence of function f_n such that:
  - f_n defined on $[0, n]$
  - $\forall n,\ \forall x \in [0, n],\ y(x) \in f_n(x)$
- We use Keye Martin's measurement theory to prove that the sequence converges.



Kleene's Iteration

- We build a sequence of function f_n such that:
  - f_n defined on $[0, n]$
  - $\forall n,\ \forall x \in [0, n],\ y(x) \in f_n(x)$
- We use Keye Martin's measurement theory to prove that the sequence converges.



Hybrid Semantics

Combining two semantics:

- hybrid environments (σ_d, σ_c)
- semantics of discrete statements remain unchanged
- semantics of a `sens`:

$$[\![\textbf{sens}.\mathrm{y?x}]\!] = \left\{(\sigma, \sigma')|\sigma' = \sigma[x \mapsto \sigma.y(\sigma.time)]\right\}$$

- semantics of a `act`:

$$[\![\textbf{act}.\mathrm{k!c}]\!] = \left\{(\sigma, \sigma')|\sigma' = \sigma \left[y \mapsto \lambda x. \left\{ \begin{array}{ll} \sigma.y(x) & \textit{if } x \leq \sigma.time \\ y_c(x) & \textit{else} \end{array} \right. \right]\right\}$$

Hybrid Semantics

Combining two semantics:

- hybrid environments (σ_d, σ_c)
- semantics of discrete statements remain unchanged
- semantics of a `sens`:

  $[\![\textbf{sens}.\text{y?x}]\!]^{\mathcal{H}} =$ perform one step of the continuous Kleene's iteration

- semantics of a `act`:

$$[\![\textbf{act}.\text{k!c}]\!] = \left\{ (\sigma, \sigma') | \sigma' = \sigma \left[y \mapsto \lambda x. \begin{cases} \sigma.y(x) & \textit{if } x \leq \sigma.\textit{time} \\ y_c(x) & \textit{else} \end{cases} \right] \right\}$$

Hybrid Semantics

Combining two semantics:

- hybrid environments (σ_d, σ_c)
- semantics of discrete statements remain unchanged
- semantics of a `sens`:

$$[\![\mathbf{sens}.\mathrm{y?x}]\!]^{\mathcal{H}}(\sigma_d, \sigma_c) = (\sigma'_d, \sigma'_c) \text{ with } \left\{ \begin{array}{lcl} \sigma'_c & = & \sigma_c[y \mapsto \Gamma^n_{\sigma_d.F, y(0)}(y)] \\ \sigma'_d & = & \sigma_d[x \mapsto mid(\sigma'_c.y(\sigma_d.time))] \end{array} \right.$$

- semantics of a `act`:

$$[\![\mathbf{act}.\mathrm{k!c}]\!] = \left\{(\sigma, \sigma')|\sigma' = \sigma \left[y \mapsto \lambda x. \left\{ \begin{array}{ll} \sigma.y(x) & \text{if } x \leq \sigma.time \\ y_c(x) & \text{else} \end{array} \right. \right]\right\}$$

Hybrid Semantics

Combining two semantics:

- hybrid environments (σ_d, σ_c)
- semantics of discrete statements remain unchanged
- semantics of a `sens`:

$$[\![\textbf{sens}.\mathrm{y?x}]\!]^{\mathcal{H}}(\sigma_d, \sigma_c) = (\sigma'_d, \sigma'_c) \text{ with } \left\{ \begin{array}{lcl} \sigma'_c & = & \sigma_c[y \mapsto \Gamma^{n}_{\sigma_d.F, y(0)}(y)] \\ \sigma'_d & = & \sigma_d[x \mapsto \mathit{mid}(\sigma'_c.y(\sigma_d.\mathit{time}))] \end{array} \right.$$

- semantics of a `act`:

$$[\![\textbf{act}.\mathrm{k!c}]\!]^{\mathcal{H}}(\sigma_d, \sigma_c) = \text{ change the function defining the ODE}$$

Hybrid Semantics

Combining two semantics:

- hybrid environments (σ_d, σ_c)
- semantics of discrete statements remain unchanged
- semantics of a `sens`:

$$[\![\textbf{sens}.\mathrm{y?x}]\!]^{\mathcal{H}}(\sigma_d, \sigma_c) = (\sigma_d', \sigma_c') \text{ with } \begin{cases} \sigma_c' & = & \sigma_c[y \mapsto \Gamma^{n}_{\sigma_d.F, y(0)}(y)] \\ \sigma_d' & = & \sigma_d[x \mapsto \mathit{mid}\big(\sigma_c'.y(\sigma_d.\mathit{time})\big)] \end{cases}$$

- semantics of a `act`:

$$[\![\textbf{act}.\mathrm{k!c}]\!]^{\mathcal{H}}(\sigma_d, \sigma_c) = \left(\sigma_d \left[F \mapsto \lambda t, y. \begin{cases} \sigma_d.F(y,t) & t \leq \sigma_d.\mathit{time} \\ \sigma_c.F_c(y,t) & \text{else} \end{cases} \right], \sigma_c\right)$$

Conclusion

New model for hybrid systems that:

- remains close to existing programs
- is designed to be integrated to existing static analyzers
- does not permit physically impossible phenomena (Zeno effect)

Denotational semantics for this model that:

- unifies the description of the continuous and discrete systems
- uses only one fixpoint to compute the semantics of the whole system

Future work:

- define an abstract semantics and analysis:
  - analysis of the continuous system: validated integration
  - analysis of the discrete system: classic domains (octagons, error series,. . .)
  - analysis of the interactions: needs to be formalized
- define a suitable widening for the continuous functions