

Ariadne

Laboratorio di Sistemi in Tempo Reale

Corso di Laurea in Informatica Multimediale

11 Dicembre 2008

1 Una breve introduzione ad Ariadne

2 Esercizio

Un breve introduzione ad Ariadne

- sviluppato da UniVr, UniUd, Parades, CWI
- può modellare automi anche con dinamica non lineare
- open source
- nucleo di calcolo scritto in C++
- struttura modulare e facilmente estendibile ed adattabile
- interfaccia in Python (solo dinamiche lineari)
- verifica di proprietà di sicurezza e raggiungibilità

Ariadne: l'interfaccia Python

- consente di accedere al motore di calcolo senza scrivere codice C++
- non è necessario compilare: Python è un linguaggio interpretato
- è flessibile: si può mescolare codice Python con le routine di calcolo di Ariadne
- efficiente: la computazione “pesante” è effettuata dal motore C++
- interfaccia di basso livello: si manipolano direttamente le strutture dati interne di Ariadne

Matrici e Vettori

Matrici e vettori sono costruiti a partire da **liste** contenenti i valori:

```
m = Matrix( [ [1, 2], [3, 4] ] )  
v = Vector( [12, a + 2, 5*b, 3.14] )
```

Nell'esempio a e b possono essere **costanti** o **variabili** Python.

Le dinamiche lineari sono modellate in Ariadne mediante **campi vettoriali affini** che rappresentano equazioni del tipo:

$$\dot{x} = Ax + b$$

Sono definiti a partire da una matrice A e da un vettore b :

```
dyn = AffineVectorField( Matrix, Vector )
```

Esempio: la dinamica della palla che rimbalza

$$\begin{aligned}\dot{x} &= v \\ \dot{v} &= -g\end{aligned}$$

```
dyn = AffineVectorField( Matrix([[0, 1], [0, 0]]),  
                        Vector([0, -g]) )
```

I reset sono rappresentati mediante **trasformazioni lineari**:

$$f(x) = Ax + b$$

definiti a partire da una matrice A e da un vettore b :

```
res = AffineMap( Matrix, Vector )
```

Esempio: il reset della palla che rimbalza

$$x' = x$$

$$v' = -0.5v$$

```
res = AffineMap( Matrix([[1, 0], [0, -0.5]]),  
                Vector([0, 0]) )
```

Regioni di spazio: Rettangoli

Sono regioni rettangolari a n dimensioni, definite a partire da una lista di intervalli:

```
rect = Rectangle([ [b0,e0], [b1,e1] ... [bn,en] ])
```

Esempio: un parallelepipedo

$$p = \{(x, y, z) \in \mathbb{R}^3 : 0 \leq x \leq 1 \wedge -1 \leq y \leq 1 \wedge 0 \leq z \leq \sqrt{2}\}$$

```
import math # funzioni matematiche di Python
```

```
rect = Rectangle([ [0, 1], [-1, 1],  
                  [0, math.sqrt(2)] ])
```

Regioni di spazio: Poliedri

Sono regioni definite a partire da una lista di **disequazioni**:

$$\begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n & \leq & b_1 \\ & \vdots & \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n & \leq & b_m \end{array}$$

rappresentate da una matrice e da un vettore:

```
pol = PolyhedralSet( Matrix, Vector )
```

Esempio: l'invariante della palla che rimbalza: $x \geq 0$

```
inv = PolyhedralSet( Matrix([ [-1, 0] ]),  
                    Vector([0]) )
```

Griglie e Griglie finite

- Le **griglie** sono suddivisioni dello spazio \mathbb{R}^n delle variabili in celle di dimensione fissata:

```
g = Grid( Vector( [0.1, 0.1] )
```

(divisione del piano in celle di lato 0.1)

- Le **griglie finite** sono griglie ristrette ad una regione limitata di spazio:

```
fg = FiniteGrid(g, Rectangle([[0,10],[0,10]]))
```

(divisione del rettangolo $[0, 10], [0, 10]$ in celle di lato 0.1)

HybridGridMaskSet

Rappresentano **regioni ibride** di spazio:

- possiedono un certo numero di **locazioni discrete**
- ad ogni locazione è associata una **griglia finita**
- le regioni sono rappresentate **marcando** le celle delle griglie.

Sono utilizzate per rappresentare le **regioni iniziali**, i **limiti** della regione di calcolo e i **risultati** delle procedure di calcolo della raggiungibilità.

Hybrid Evolver: definizione

`HybridEvolver` è la classe responsabile dell'analisi e della simulazione di automi ibridi.

```
hyb = HybridEvolver( Applicator, Integrator )
```

- `Applicator` calcola l'**evoluzione discreta** del sistema;
- `Integrator` calcola l'**evoluzione continua** del sistema;
- a seconda del sistema studiato, si possono usare diversi tipi di `Applicator` e `Integrator`.

`HybridEvolver` mette a disposizione tre metodi principali:

- `continuous_chainreach(automa, init_set, bounding_set)`
calcola l'evoluzione continua di `automa` a partire da `init_set` e rimanendo all'interno di `bounding_set`;
- `discrete_step(automa, init_set)`
calcola un passo di evoluzione discreta di `automa` a partire da `init_set`;
- `chainreach(automa, init_set, bounding_set)`
alterna il calcolo dell'evoluzione continua e discreta di `automa` a partire da `init_set` e rimanendo all'interno di `bounding_set`, finché la regione raggiunta non si stabilizza.

`init_set`, `bounding_set` ed il risultato dei metodi sono di tipo `HybridGridMaskSet`.

Riassumendo...

- La definizione degli automi utilizza **poliedri** e **rettangoli** per rappresentare:
 - ▶ invarianti,
 - ▶ guardie
- Le routine di calcolo usano **HybridGridMaskSet** per rappresentare:
 - ▶ regione iniziale;
 - ▶ regione di calcolo;
 - ▶ risultato del calcolo;
 - ▶ risultati intermedi.

Modellare la palla che rimbalza in Ariadne (bouncing.py)

```
#!/bin/python
from ariadne import *

# Costanti
g = 9.8

# Definizione dell'automa
automaton=HybridAutomaton("Bouncing ball")

# locazione fly:
dyn=AffineVectorField(Matrix([[0,1],[0,0]]),Vector([0,-g]))
inv=PolyhedralSet(Matrix([[-1,0]]), Vector([0]))
l1=automaton.new_mode(0, dyn, inv)

# Transizione fly -> fly
act=PolyhedralSet(Matrix([[1,0]]), Vector([0]))
res=AffineMap(Matrix([[1,0],[0,-0.5]]), Vector([0,0]))
el2=automaton.new_transition(0,l1.id(),l1.id(),res,act)
```

Modellare la palla che rimbalza in Ariadne (2)

```
# Definizione della griglia
grid=Grid(Vector([0.05,0.05]))
bounding_box=Rectangle([[ -0.1,2.5],[ -10,10]])
fgrid=FiniteGrid(grid,bounding_box)

# Insieme iniziale
init=Rectangle([[1.999,2],[0,0.0001]])
init_set=HybridGridMaskSet()
init_set.new_location(l1.id(),fgrid)
init_set[l1.id()].adjoin_over_approximation(init)

# Bounding set
b_set=HybridGridMaskSet()
b_set.new_location(l1.id(),fgrid)
b_set[l1.id()].adjoin_over_approximation(bounding_box)
```

Modellare la palla che rimbalza in Ariadne (3)

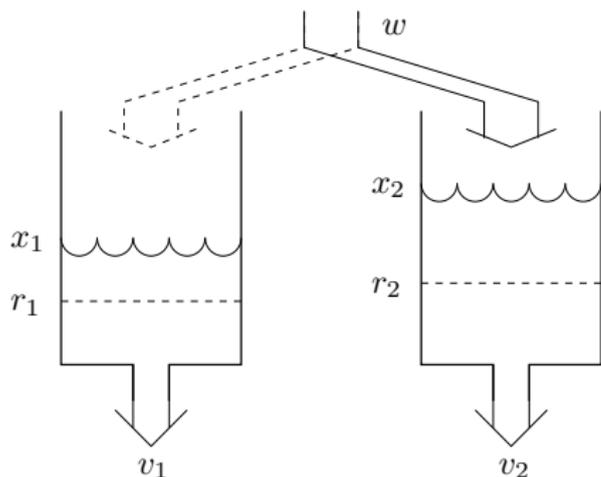
```
# Definizione dell'Hybrid Evolver
apply=Applicator()
integrator=AffineIntegrator(0.1,2,0.1);
hybrid_ev=HybridEvolver(apply,integrator);
set_hybrid_evolver_verbosity(4)

print "Computing chainreachable set..."
reach_set=hybrid_ev.chainreach(automaton,init_set,b_set)
print "Done."

# Txt output
txt=TextFile()
txt.open("bouncing-ball.txt")
txt.write(reach_set[l1.id()])
txt.close()
```

- 1 Una breve introduzione ad Ariadne
- 2 **Esercizio**

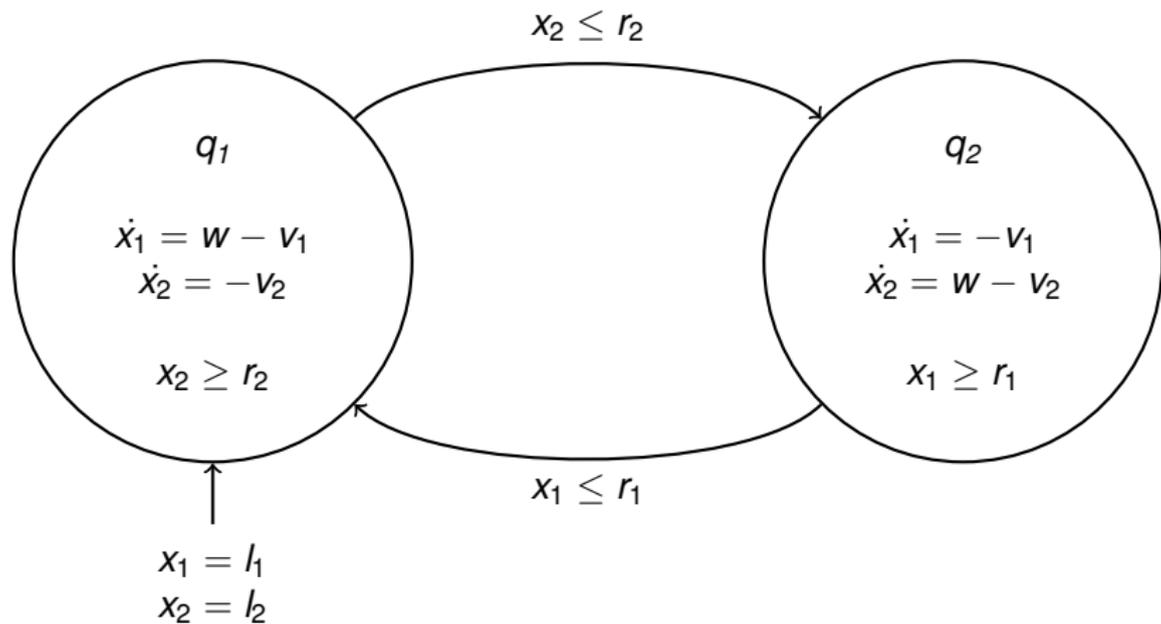
Un sistema con due cisterne



- Il sistema immette acqua con velocità w in una sola cisterna alla volta;
- l'acqua esce dalle cisterne con velocità v_1 e v_2 ;
- x_1 e x_2 rappresentano i livelli dell'acqua

Obiettivo: mantenere i livelli dell'acqua maggiori di r_1 e r_2 .

L'automa del sistema con due cisterne



- Implementare il sistema con due cisterne in Ariadne, usando i seguenti valori dei parametri:
 - ▶ $v_1 = 1.0, v_2 = 1.0;$
 - ▶ $w = 1.50;$
 - ▶ $r_1 = 5.0, r_2 = 4.0;$
 - ▶ $l_1 = 7.0, l_2 = 5.0;$
- Calcolare la regione raggiunta dal sistema;
- Calcolare la regione raggiunta anche quando $w = 2.50;$
- Confrontare il comportamento del sistema nei due casi.

Un piccolo pezzo di codice dell'esercizio

```
# Initial set
init=Rectangle([[7.0,7.001],[5.0,5.001]])
initial_set=HybridGridMaskSet()
initial_set.new_location(q1.id(),fgrid)
initial_set[q1.id()].adjoin_over_approximation(init)
initial_set.new_location(q2.id(),fgrid)

# Bounding set
b_set=HybridGridMaskSet()
b_set.new_location(q1.id(),fgrid)
b_set[q1.id()].adjoin_over_approximation(bounding_box)
b_set.new_location(q2.id(),fgrid)
b_set[q2.id()].adjoin_over_approximation(bounding_box)
```

<http://profs.sci.univr.it/~bresolin/lab04.pdf>