ChorEr: un analizzatore statico per generare Automi Coreografici da codice sorgente Erlang

Presentata da: Gabriele Genovese
Relatore: Prof. Ivan Lanese

Alma Mater Studiorum - Università di Bologna
Corso di Laurea Triennale in Informatica - Scuola di Scienze
A.A. 2022/2023 Sessione I

Cosa sono le coreografie?

- Modello matematico
- Formalizza le comunicazioni tra *N* partecipanti
- Studiate nella teoria dei tipi e nei linguaggi di programmazione
- Verifica semantica di proprietà

Due tipi di vista:

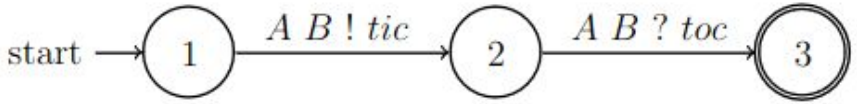
- Locale: singolo partecipante
- Globale: tutti i partecipanti

Automi Coreografici

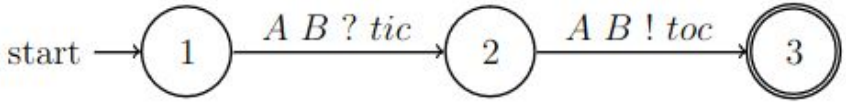
Caratteristiche:

- Rappresentazione tramite FSA
- Algoritmi e teoria già studiati

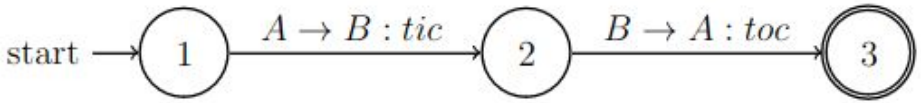
Esempio vista locale partecipante A:



Esempio vista locale partecipante B:



Esempio vista globale



Perché Erlang?

- *Modello ad attori*
- *Message passing*
- Sistema di comunicazione semplice ed efficace:
  - spawn: crea processi
  - ! : invia messaggi
  - receive: riceve messaggi

```
4
5   spawn(test, prova, []).
6
```

```
4
5   Processo1 ! messaggio.
6
```

```
3
4   receive
5       patternMatching1 -> do_something();
6       patternMatchingN -> do_something();
7       _ -> do_something()
8   end.
```

Modalità d’uso

Crea gli automi coreografici di un programma in Erlang

Input:

- File Erlang
- Entry point del programma
- Opzioni

Output:

- File DOT delle viste locali
- File DOT della vista globale

```
digraph graph1 {
    # grafo orientato da destra a sinistra
  rankdir="LR";

    # definizione dei nodi
  n_0 [label="start", shape="plaintext"];
  n_1 [id="1", shape=circle, label="1"];
  n_2 [id="2", shape=circle, label="2"];
  n_3 [id="4", shape=doublecircle, label="3"];
    # definizione degli archi
  n_0 -> n_1;
  n_1 -> n_2 [label="A -> B : tic"];
  n_2 -> n_3 [label="B -> A : toc"];
}
```

Funzionamento

Tre fasi:

- Estrazione metadati
- Creazione viste locali
- Creazione vista globale

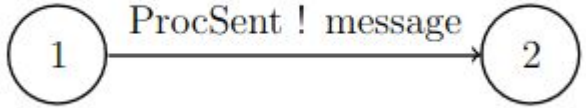
Parola chiave	Supporto
atom	sì
integer	sì
float	sì
boolean	no
tuple	sì
list	sì
record	no
map	no
binary	no
if	sì
case	sì
receive	sì

Parola chiave	Supporto
!	sì
match	sì
function	in parte
when	no
self	sì
spawn	sì
rand:uniform	sì
try catch	no
after	no
math operation	no
fun	no
attribute	in parte

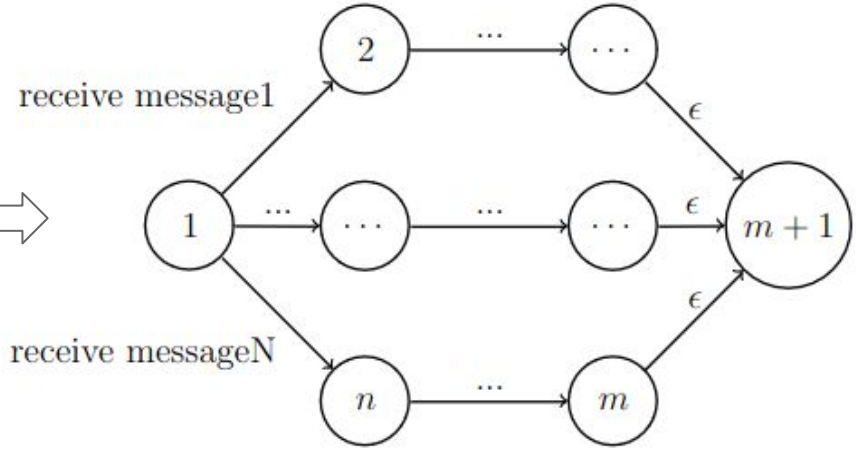
Costrutti supportati

```
4
5   spawn(test, prova, []).
6
```


```
4
5   Processo1 ! messaggio.
6
```




```
3
4   receive
5       patternMatching1 -> do_something();
6       patternMatchingN -> do_something();
7       _ -> do_something()
8   end.
```



Corrispondenza tra codice e vista locale

```erlang
-module(simple).
-export([main/0, dummy1/0, dummy2/0]).

dummy1() ->
    receive
        ciao -> done
    end,
    d2 ! bello.

dummy2() ->
    d1 ! ciao,
    receive
        bello -> done
    end.

main() ->
    A = spawn(?MODULE, dummy1, []),
    register(d1, A),
    B = spawn(?MODULE, dummy2, []),
    register(d2, B).
```

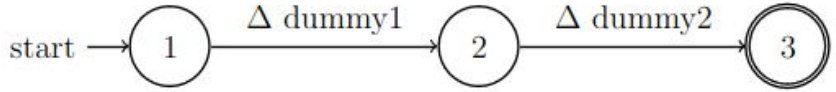


Figura 4.8: Vista locale di `main`

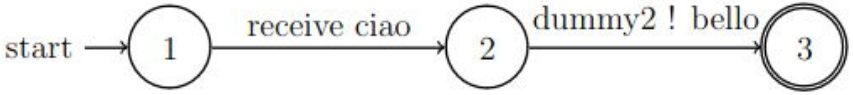


Figura 4.9: Vista locale di `dummy1`

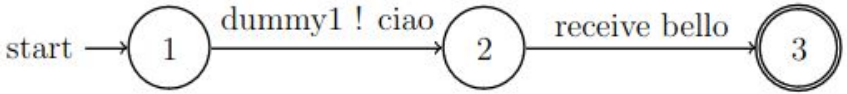


Figura 4.10: Vista locale di `dummy2`

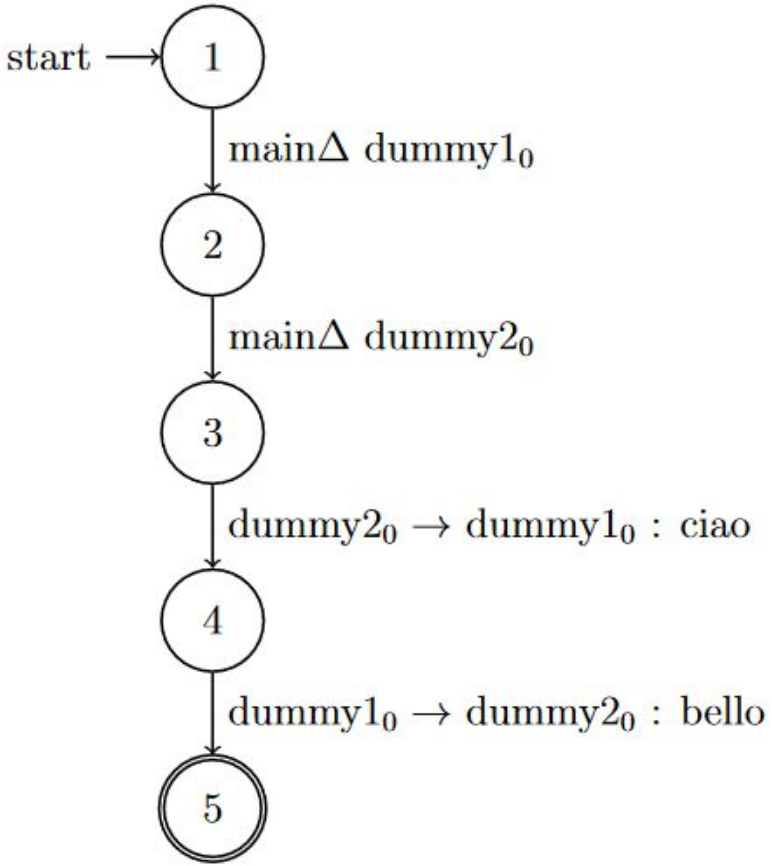
Esempio semplice - viste locali

```erlang
-module(simple).
-export([main/0, dummy1/0, dummy2/0]).

dummy1() ->
    receive
        ciao -> done
    end,
    d2 ! bello.

dummy2() ->
    d1 ! ciao,
    receive
        bello -> done
    end.

main() ->
    A = spawn(?MODULE, dummy1, []),
    register(d1, A),
    B = spawn(?MODULE, dummy2, []),
    register(d2, B).
```



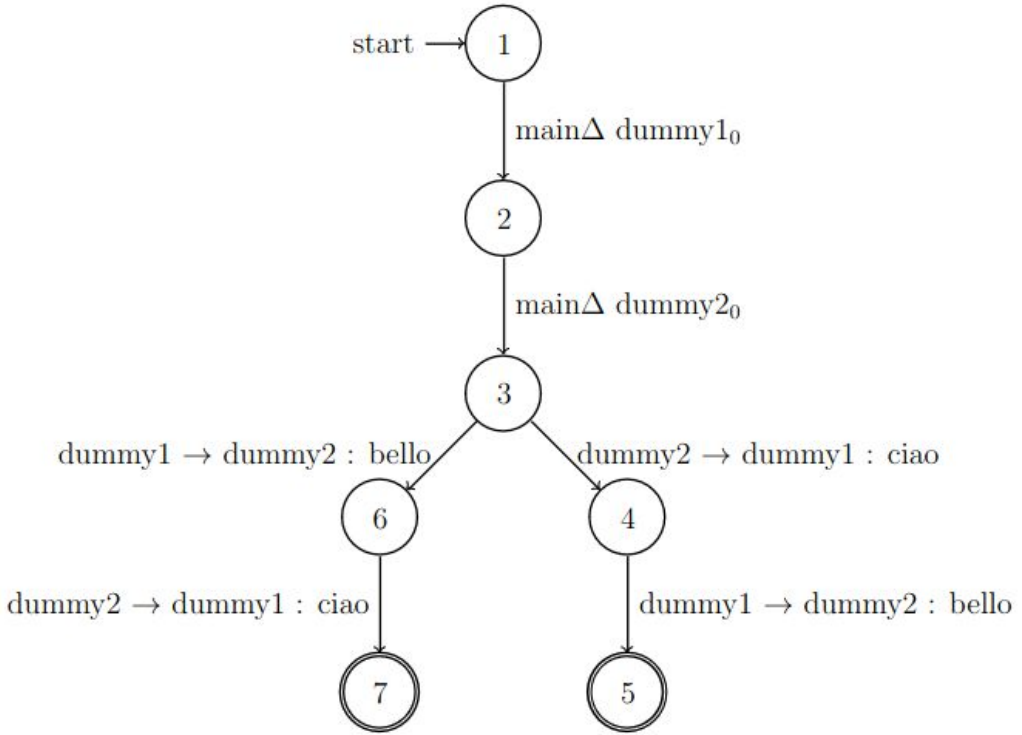
Esempio semplice - vista globale

```erlang
-module(simple).
-export([main/0, dummy1/0, dummy2/0]).

dummy1() ->
    d2 ! bello,
    receive
        ciao -> done
    end.

dummy2() ->
    d1 ! ciao,
    receive
        bello -> done
    end.

main() ->
    A = spawn(?MODULE, dummy1, []),
    register(d1, A),
    B = spawn(?MODULE, dummy2, []),
    register(d2, B).
```



Esempio con due possibili esecuzioni - vista globale

Conclusioni – Risultati e Lavori futuri

- Proof of concept funzionante per piccoli programmi
- Supporto per comunicazioni ***asincrone***
- Risultato approssima il comportamento del sistema

- Refactor codice e documentazione
- Migliore **approssimazione**
- Supporto di altri costrutti