A Synchronisation Facility for a Stream Processing Coordination Language

Anna Tikhonova

28 January 2015

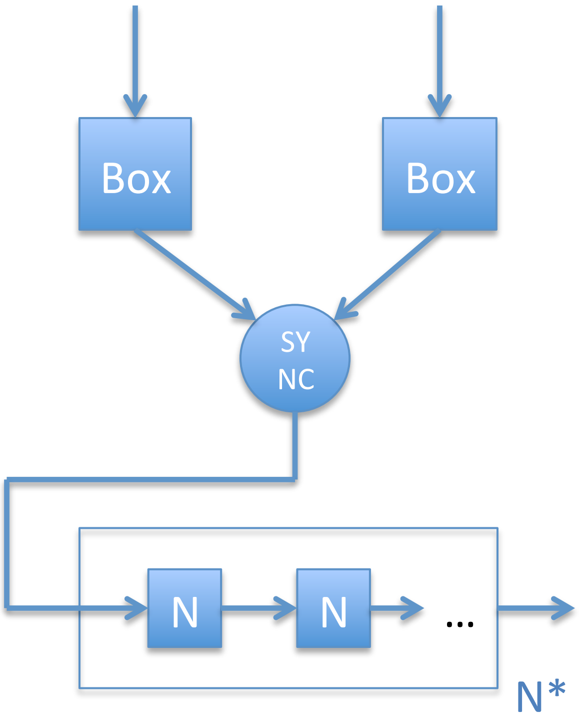
AstraKahn

A program is a network of

- Boxes – **not** analysable
- Synchronisers

connected by

- Channels
- Wiring patterns
  - Static
  - Dynamic (serial replication)



Objectives

A dedicated language of synchronisers:

- Implement a compiler
  - Transition analysis
  - Data transformation analysis

Output from the serial replication pipeline:

- Explore the role of synchronisers in the serial replication

Synchroniser

- State machine
  - Several input and output channels
  - Non-deterministic
  - Dedicated language (analysable)
- Store variables
  - Save messages
- State variables
  - Fold regular transition matrices

The Language (examples)

```
synch zip2 (a, b | c) {
  store ma, mb;
  start {
    on:
      a {
        ma = this;
        goto s1;
      }
      b {
        mb = this;
        goto s2;
      }
  }
  s1 {
    on:
      b {
        send ma || this => c;
      }
  }
  s2 {
    on:
      a {
        send this || mb => c;
      }
  }
}
```

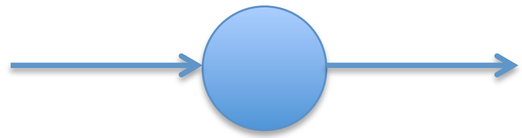
```
synch counter (a | b) {
  state int(2) c = 0;
  start {
    on:
      a & [c < 3] {
        set c = [c + 1];
      }
     a & [c == 3] {
       set c = [0];
       send this => b;
     }
  }
}
```

Term system

- Message Definition Language (MDL)
  - Basic terms
  - Constructors

- Examples:
  - Variable `$t`
  - Record `{ x:$x, y:$y, z:$z }`
  - Choice `(: v:$v, w:$v :)`

Passports



p_i => p_o

```
a.(x, y || t)
  send this => c
```

```
a ~ (: uniq:{x:$x, y:$y | $t} :)
```

`c` is the same as `a`

```
b.?w(r, k)
  send ?z(x:[r]) => d
```

```
d ~ (: z:{x:Int} :)
```

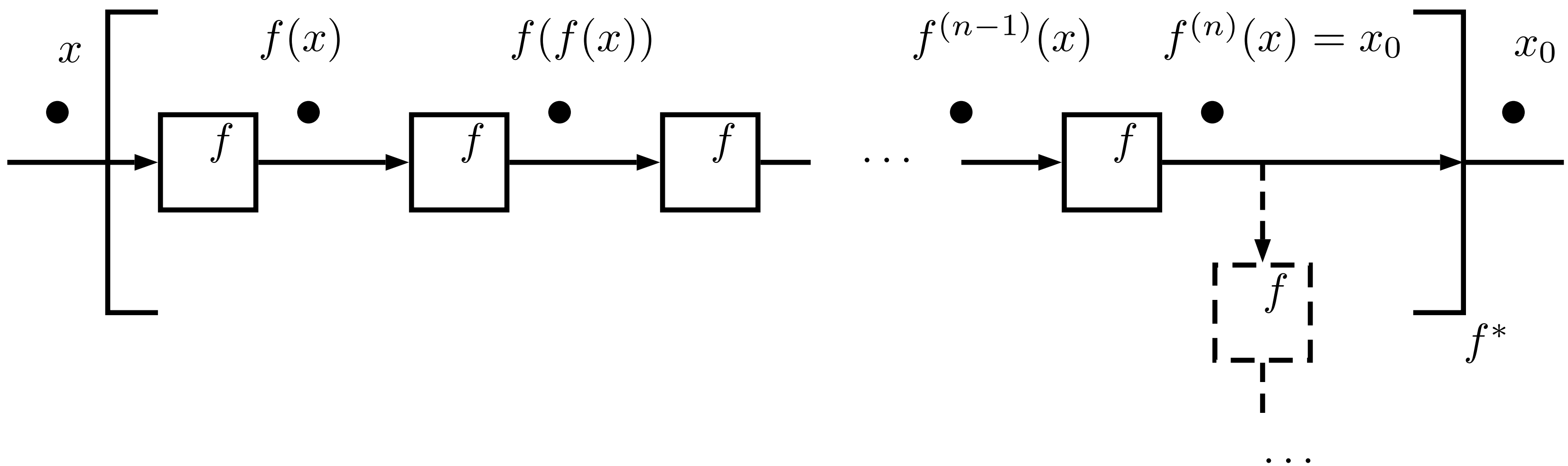
```
b.?v
```

```
b ~ (: v:$v, w:{r:Int, k:$k} :)
```

The compiler

- Lexical and syntax analysis
- Static analysis
  - Semantic checking
  - Type checking
- Runtime data structure
- Passport

Serial Replication

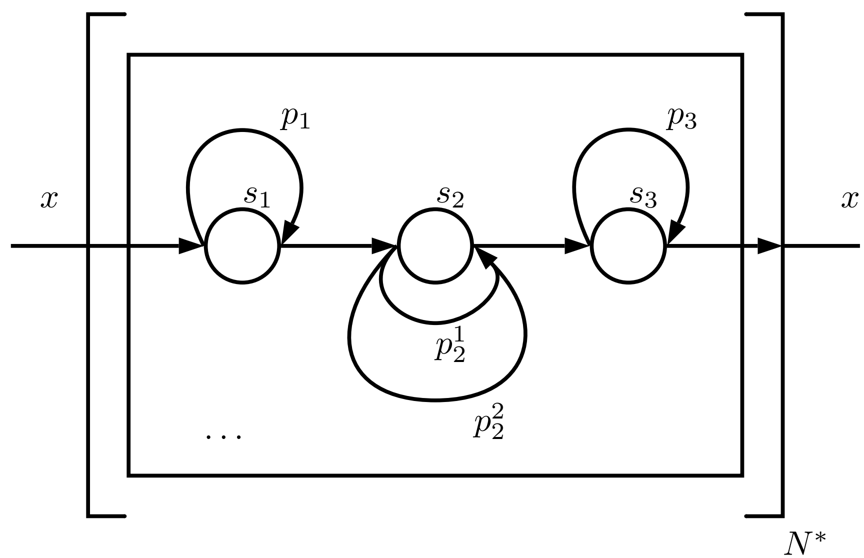


$\boldsymbol{x_0}$ is the fixed point message

An inactive replica transmits the fixed point message without change

Forward Fixed Point

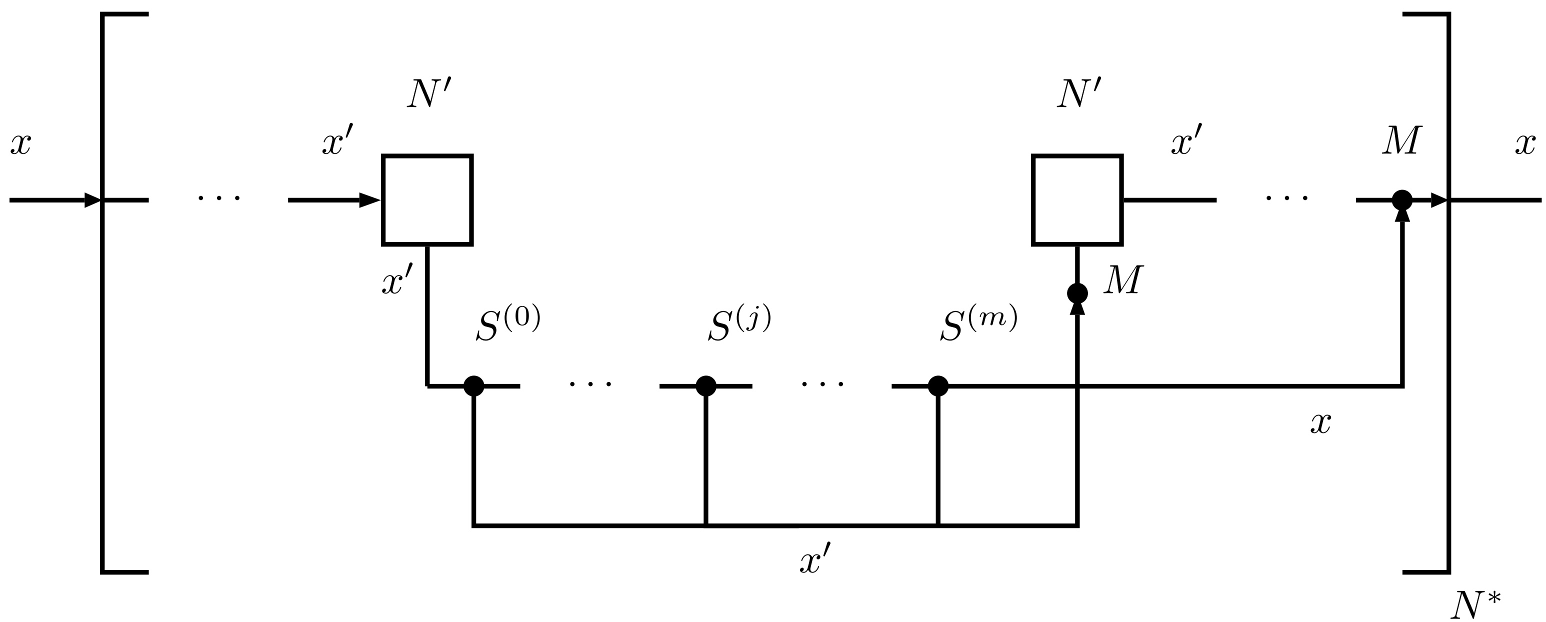
Every synchroniser on the fixed-point path transmits the fixed-point message without change



The fixed-point condition of **N** is

$p_1 \wedge (p_2{}^1 \vee p_2{}^1) \wedge p_3$

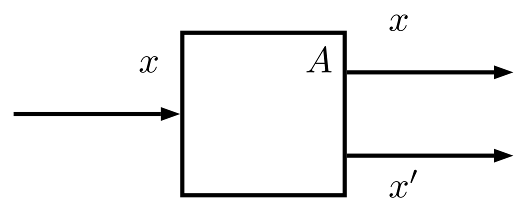
Output from the Serial Replication

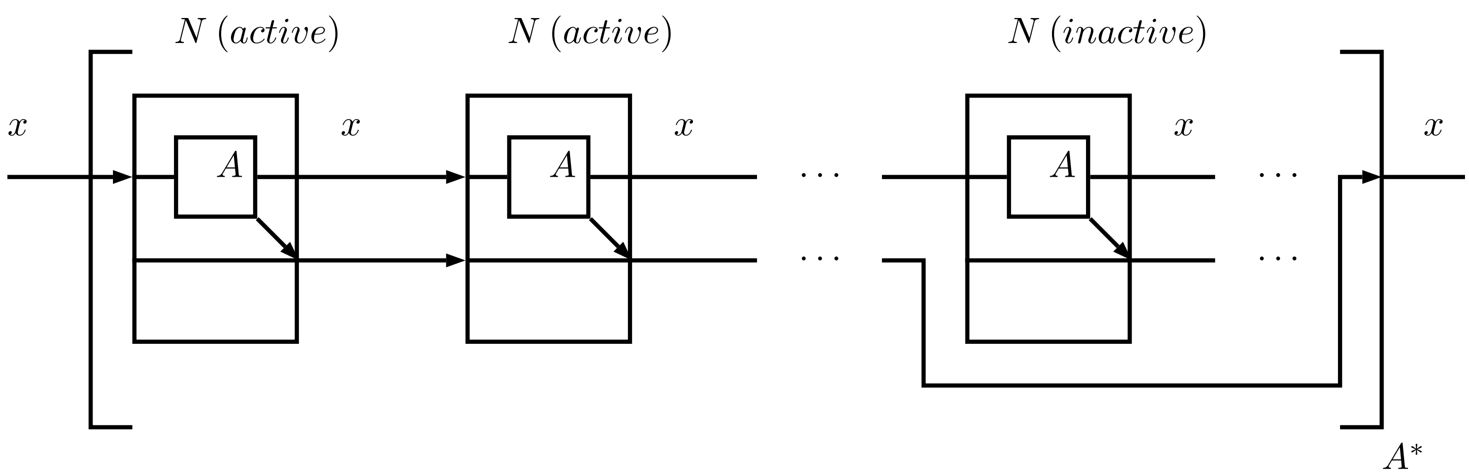


Synchroniser **S**$^{(j)}$, j=0,m checks if the fixed point condition extracted from the **j**-th synchroniser on the fixed point path is satisfied:

- **Yes** – the message is sent to **S**$^{(j+1)}$
- **No** – the message is sent to the next replica of **N**

Another Approach





Channel x' is auxiliary

Reverse Fixed Point

- prevents an unnecessary cascade
- is a state of a network in which it transmits any message without change

- A synchroniser can have a reverse fixed point
- A replica has a reverse fixed point if:
  - The replica has the fixed-point path
  - Every synchroniser on the fixed-point path has a reverse fixed point (possibly a closed subset of states)

Fixed Point Detection

- The fixed-point path search
  - DFS-based
  - Shared for both types of fixed point
- Fixed point detection in a synchroniser
  - Forward fixed point
  - Reverse fixed point

Conclusion

- The compiler
  - Transition analysis
  - Data transformation analysis
- Analysis of the fixed point approach
  - Another approach suggested
- Future directions
  - Flow inheritance
  - Statistical properties