Inferring Automation Logic from Manual Control Scenarios: Implementation in Function Blocks



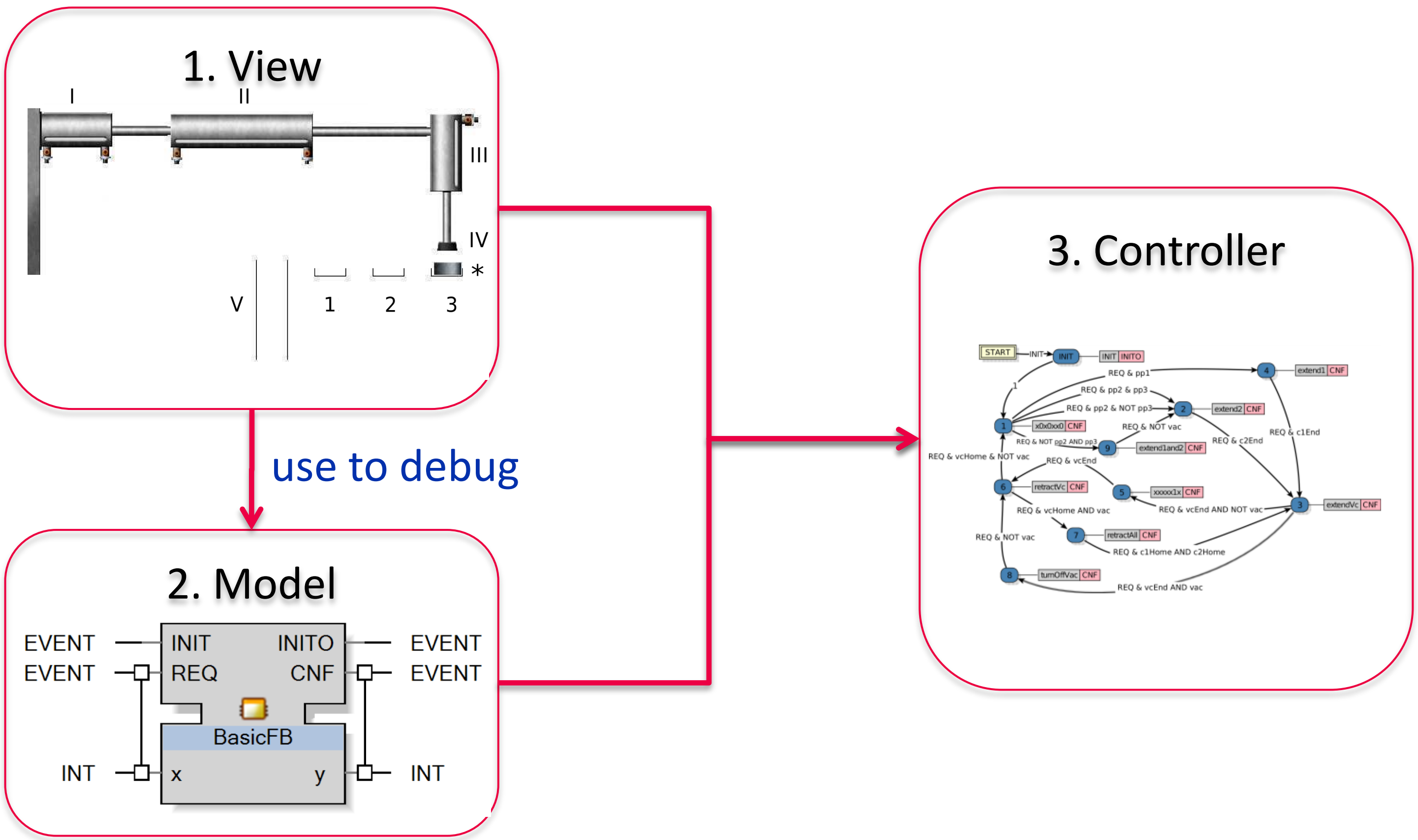
Daniil Chivilikhin
PhD student
ITMO University

Anatoly Shalyto
Dr. Sci., professor
ITMO University

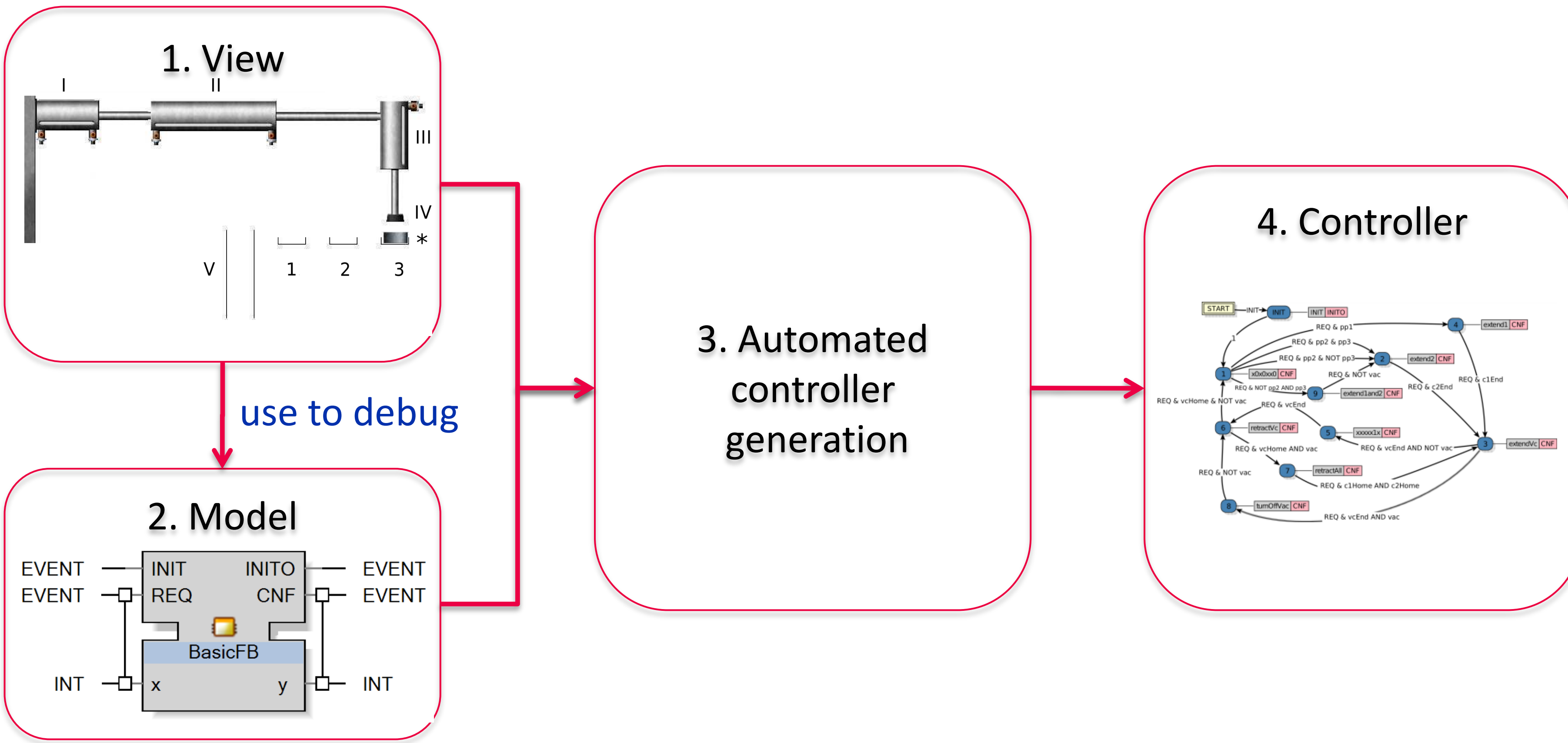
Valeriy Vyatkin
Dr. Eng., professor,
Aalto University,
Luleå University of Technology

DIAS@ISPA ’15, Helsinki, Finland, August 21, 2015

MVC application engineering



MVC application engineering



Problem statement

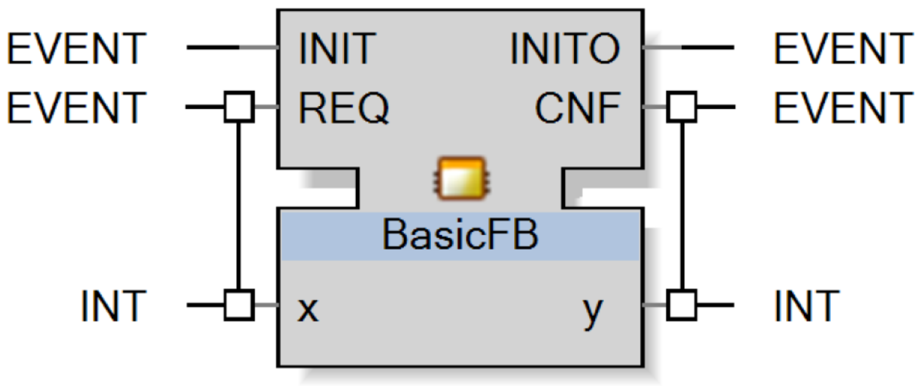
Develop a method for

automated controller generation

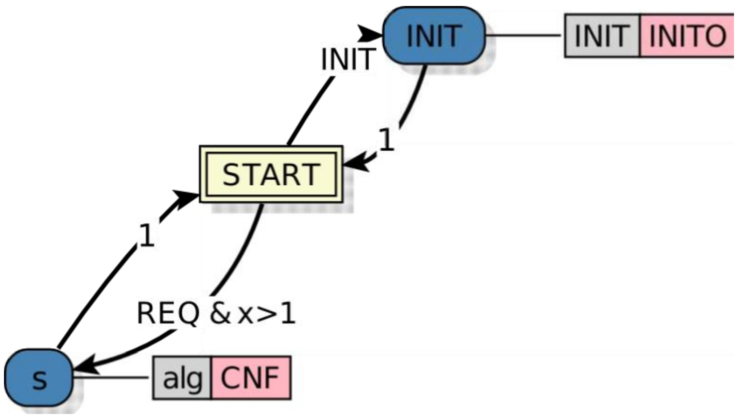
for

MVC applications

Implementation: IEC 61499 function blocks

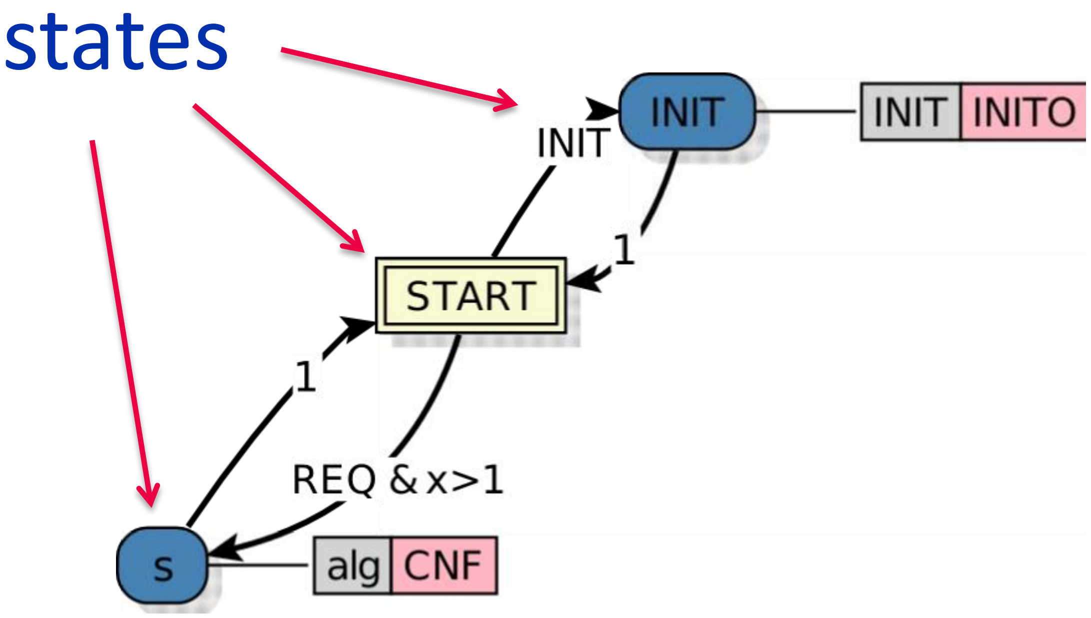


Function block interface



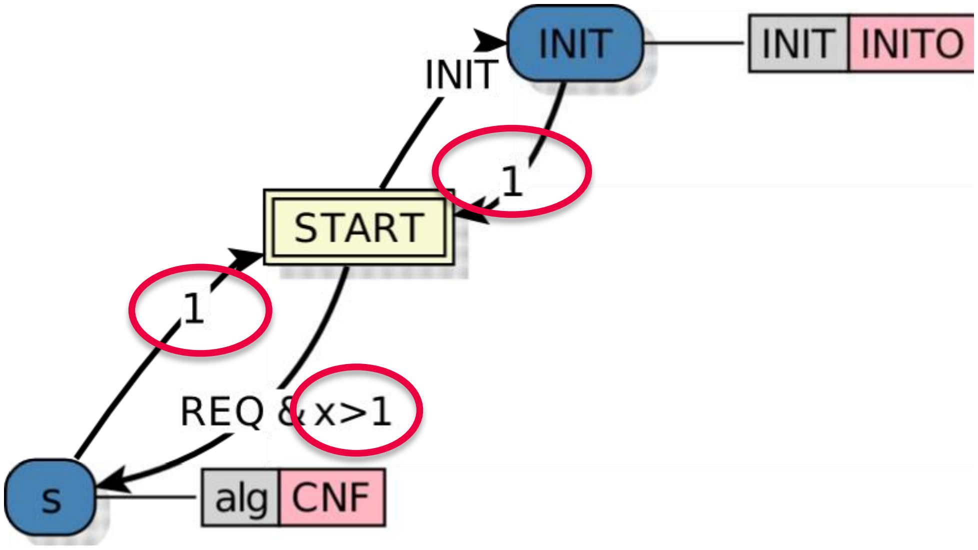
Execution Control Chart (ECC)

IEC 61499 Execution Control Chart



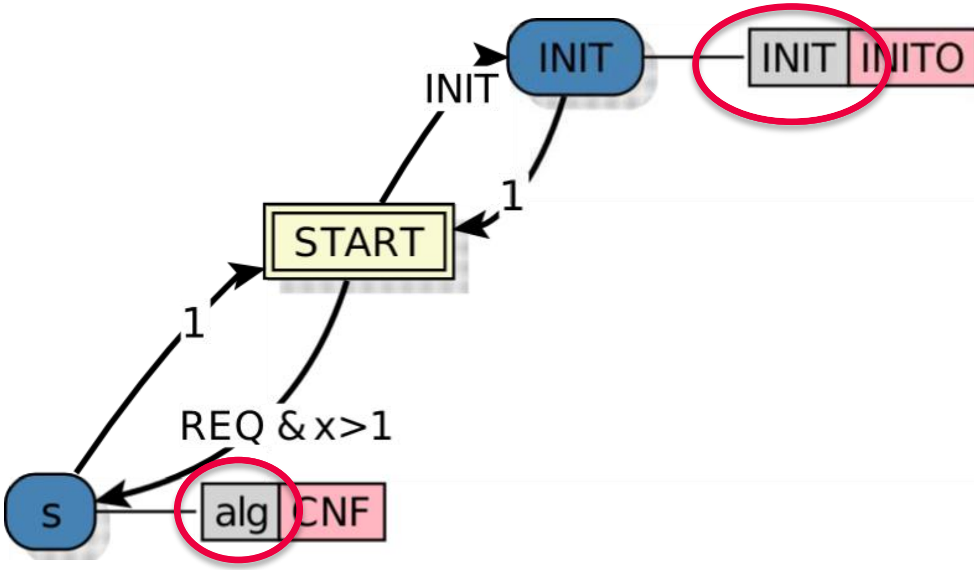
IEC 61499 Execution Control Chart

- Guard conditions
- Boolean formulas
- input/output variables
- internal variables
- constants



IEC 61499 Execution Control Chart

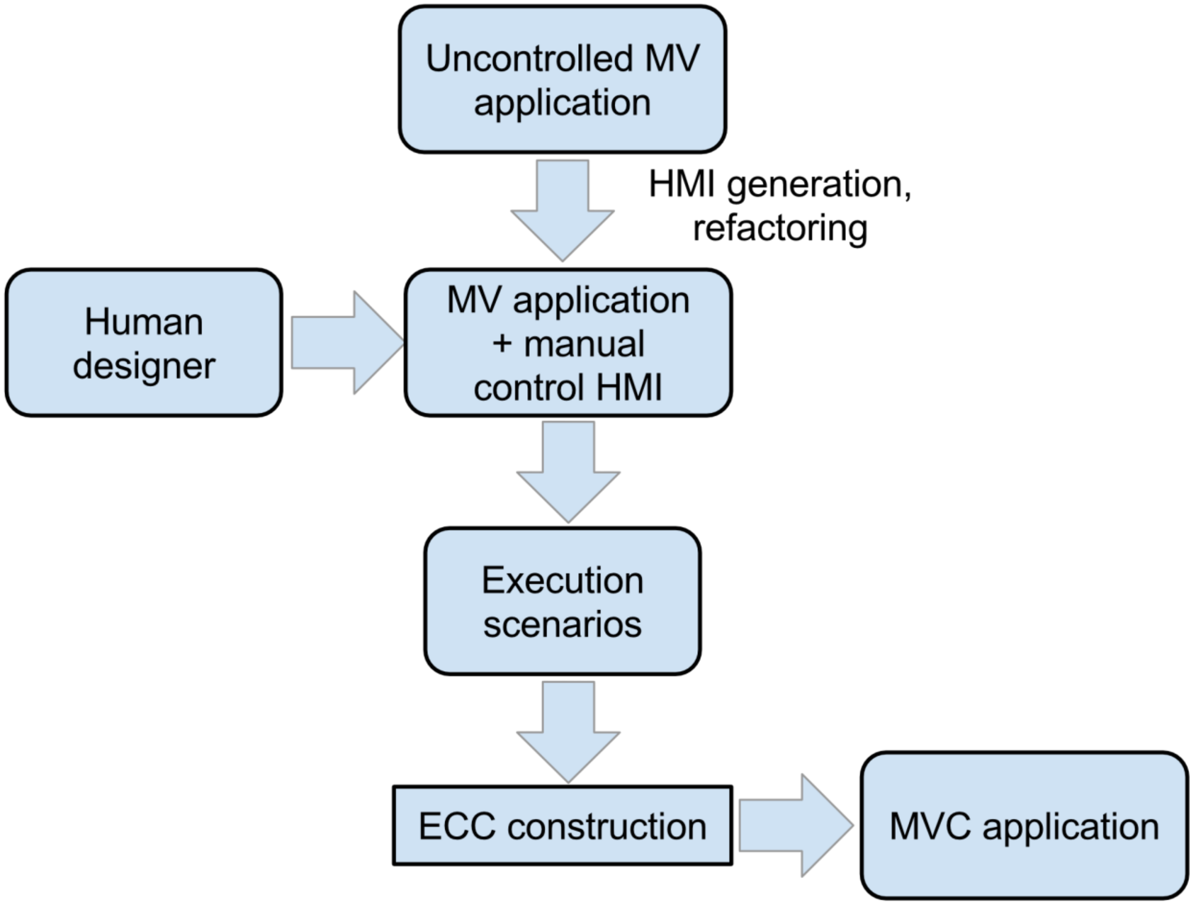
- Algorithms
- Change output variables



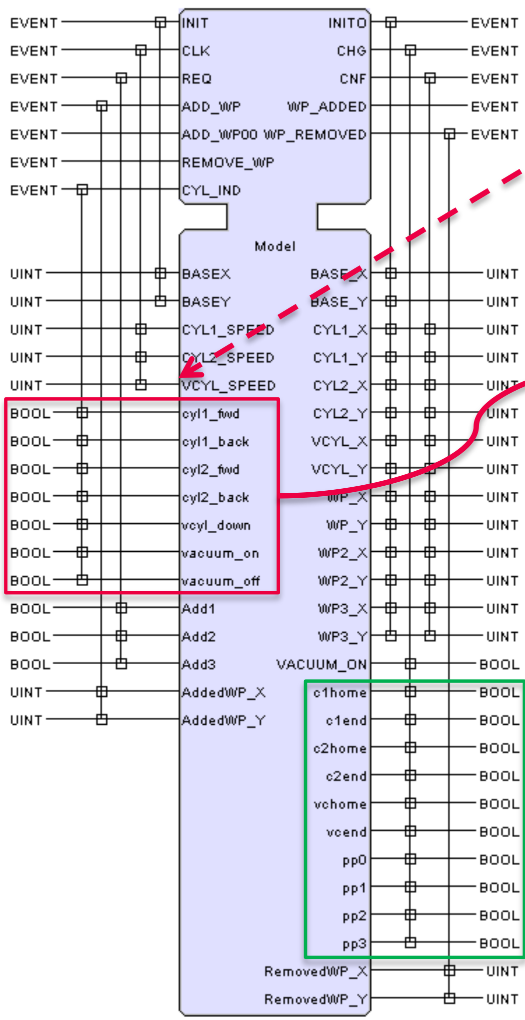
Assumptions and simplifications

- **Model and View** are implemented
- Only **Boolean** input/output variables
- Guard conditions – only input variables

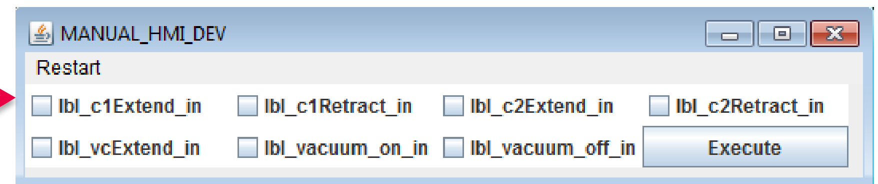
Proposed approach



HMI generation



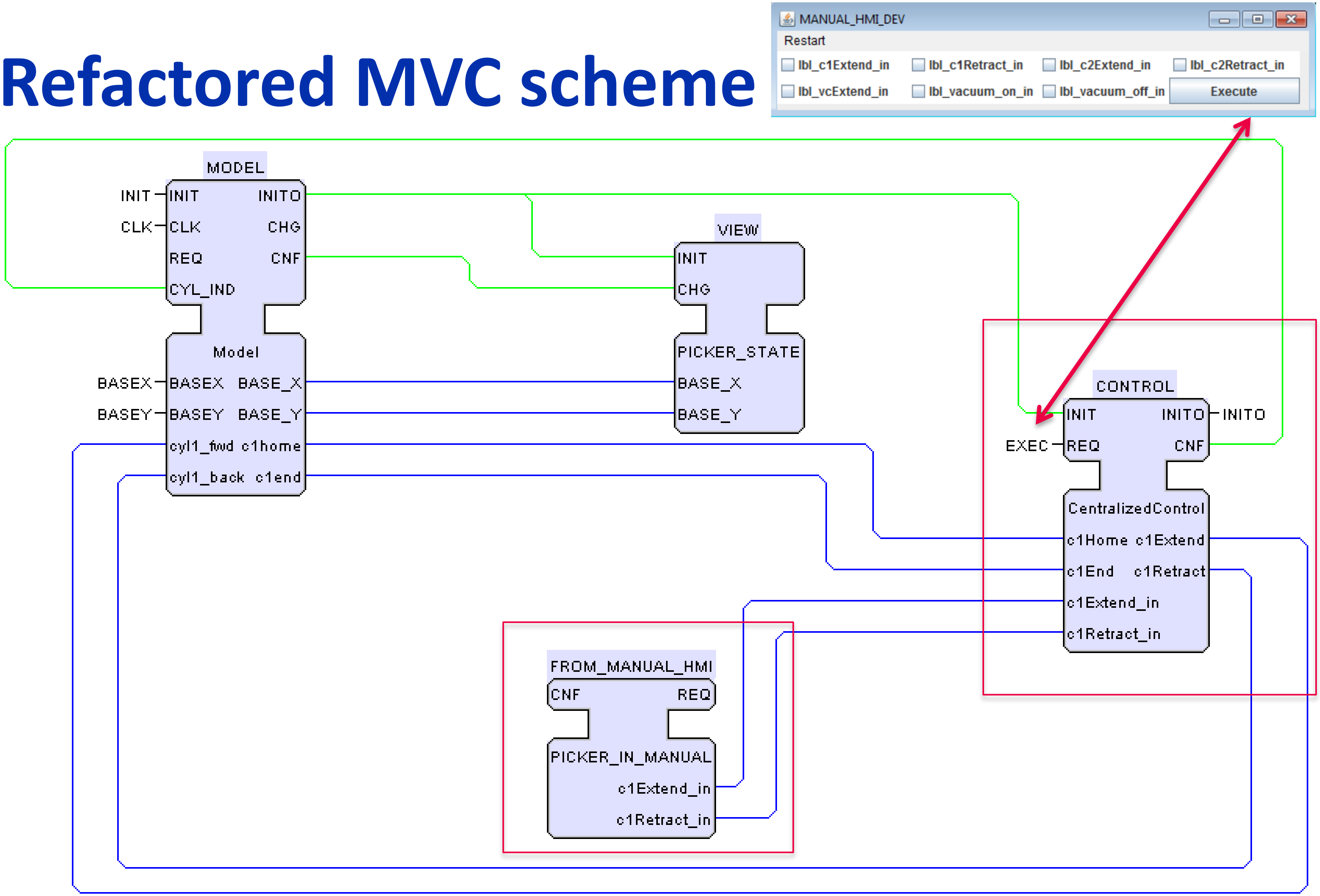
- M.I: Model's inputs that should be set by Controller



- M.O: Model's outputs to be used in controller

ITMO UNIVERSITY

Refactored MVC scheme

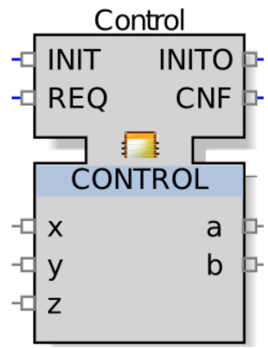


Execution scenario

s_0	s_1	s_2	s_3	s_4	s_5

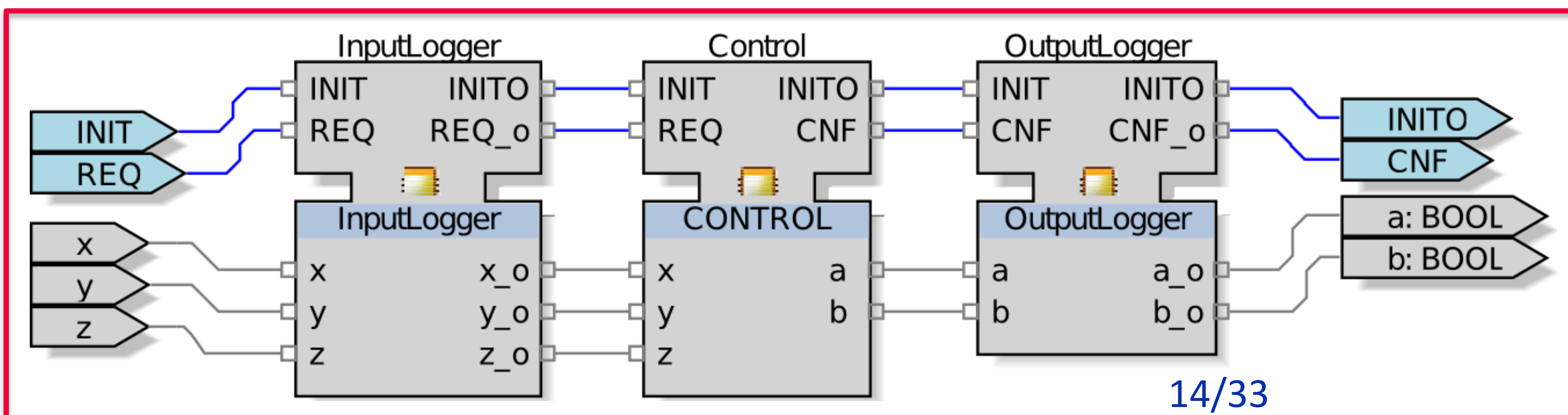
Input event	Input variables	Output variables	Output event
REQ	01011010	101010	CNF

Recording execution scenarios



Dummy function block

Automated refactoring

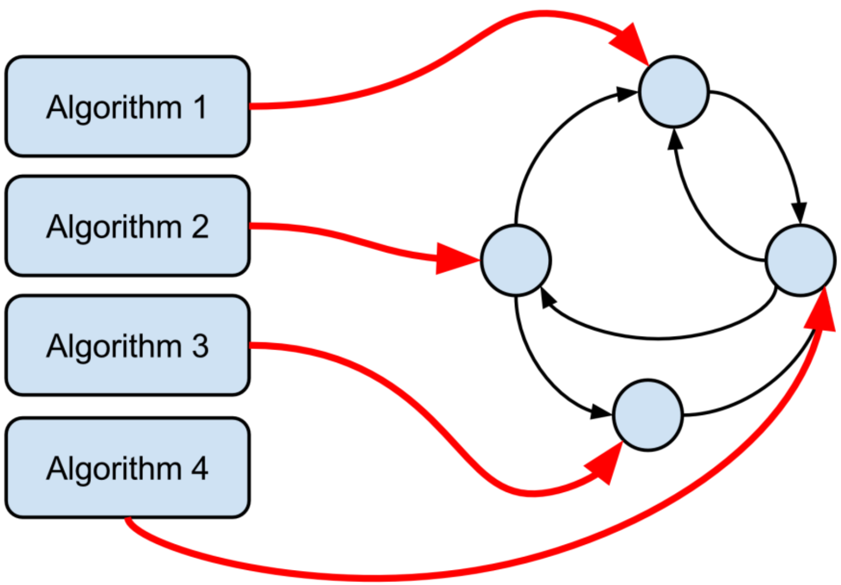


ECC construction algorithm: previous work

- Metaheuristic algorithm
  - Chivilikhin et al. Reconstruction of Function Block Logic using Metaheuristic Algorithm: Initial Explorations / In Proceedings of the 13th IEEE International Conference on Industrial Informatics (INDIN'15)
- No theoretical bounds on running time
- **In one special case** we can do better!

Exact ECC construction

- If each algorithm is used in exactly one state
- We can determine algorithms automatically
- And then construct the ECC
- **+ only for Boolean inputs/outputs!**



Proposed ECC construction algorithm

1. Determine **minimal set** of state algorithms
2. Construct ECC from scenarios labeled by found algorithms
3. Simplify ECC

Algorithm representation

- Algorithms are strings over {‘0’, ‘1’, ‘x’}
- a_i=‘0’: set z_i=0
- a_i=‘1’: set z_i=1
- a_i=‘x’: preserve value of z_i

- Example

z	0	1	1	0
a	x	1	0	1
a(z)	0	1	0	1

Determine initial set of simple algorithms

- For each scenario **s** and each pair of elements s_i *and* s_{i+1}
- Calculate algorithm **a** for transforming $s_i \rightarrow s_{i+1}$
- Function calcAlg(s_i,s_{i+1})

$$a_i = \begin{cases} x, \text{ if } s_i^j = s_{i+1}^j; \\ 0, \text{ if } s_i^j = 1 \wedge s_{i+1}^j = 0; \\ 1, \text{ if } s_i^j = 0 \wedge s_{i+1}^j = 1. \end{cases}$$

- Example

s_i	0	1	0	1
s_{i+1}	0	1	1	0
a	x	x	1	0

Determine initial set of simple algorithms

A = new Set()
for all scenarios $s \in S$ **do**
    for $i = 0$ to $|s| - 1$ **do**
        $A \leftarrow A \cup \{\text{calcAlg}(s_i.\text{out}, s_{i+1}.\text{out})\}$
    end for
end for

Merge algorithms

- Function **merge(a, b)**

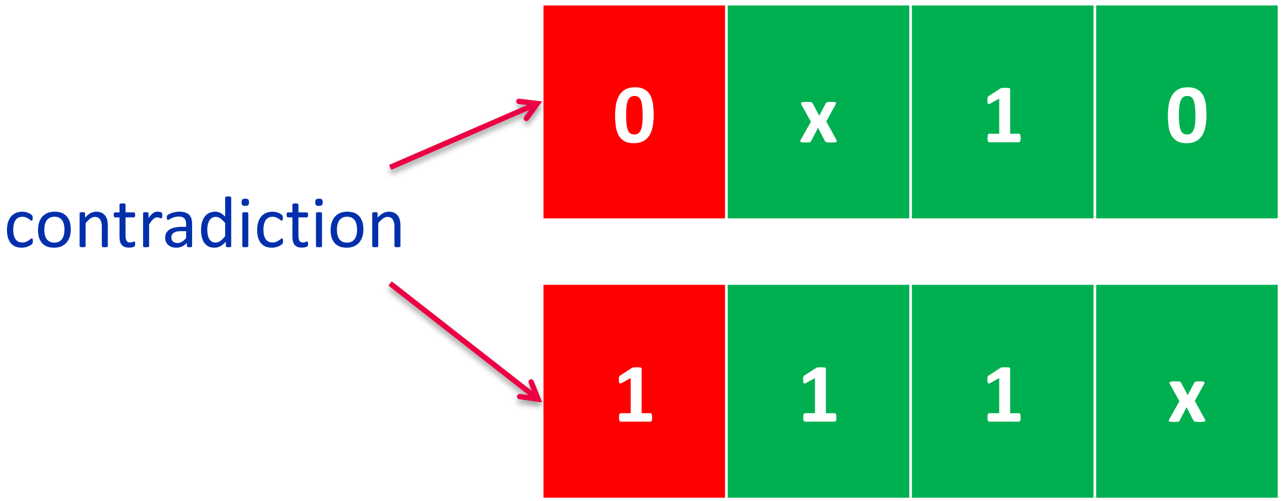
$$m_j^{ab} = \begin{cases} a_j, \text{ if } a_j = b_j; \\ x, \text{ if } a_j = x \vee b_j = x. \end{cases}$$

- Example

a	0	x	1	0
b	0	1	1	x
m^{ab}	0	x	1	x

Only consistent algorithms are merged

- Algorithms are **consistent** if they don't have **contradicting elements**



Checking if merge is valid

- **Invariant**: algorithms are sufficient to represent all scenarios
- For each scenario ***s***
- For each s_i and s_{i+1}

- A’ ← A \ {a, b}
- A’ ← A’ U {m^{ab}}
- **if** A’ satisfies invariant
  then A ← A’

$$\exists a \in A : \text{applyAlg}(a, s_i.\text{out}) = s_{i+1}.\text{out}$$

Merging algorithms: pseudocode

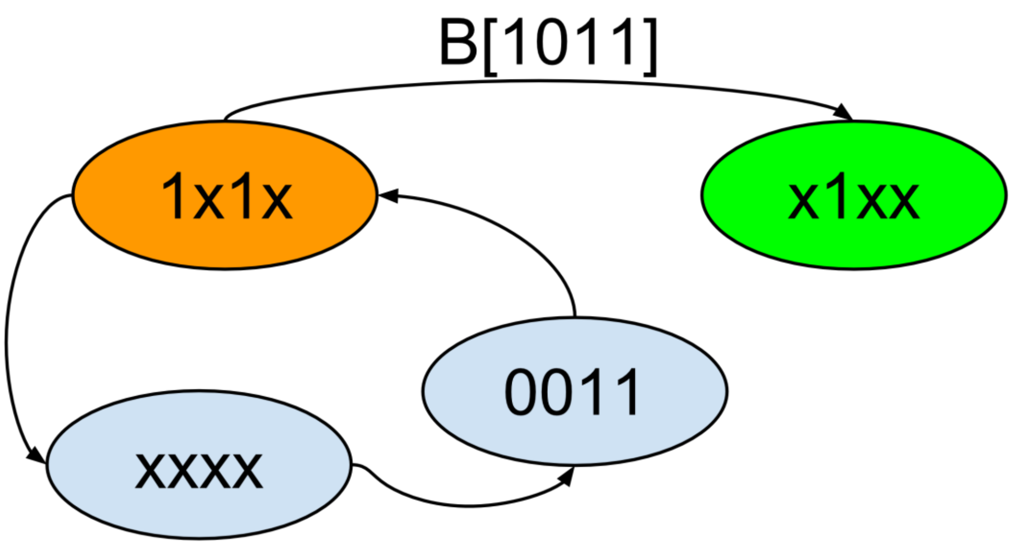
```
for all a ∈ A do
    for all b ∈ A, b ≠ a do
        m^{ab} ← merge(a, b)
        if merge is valid then
            A ← A \ {a, b}
            A ← A ∪ {m^{ab}}
            changed ← true
            goto line 21
        end if
    end for
end for
```

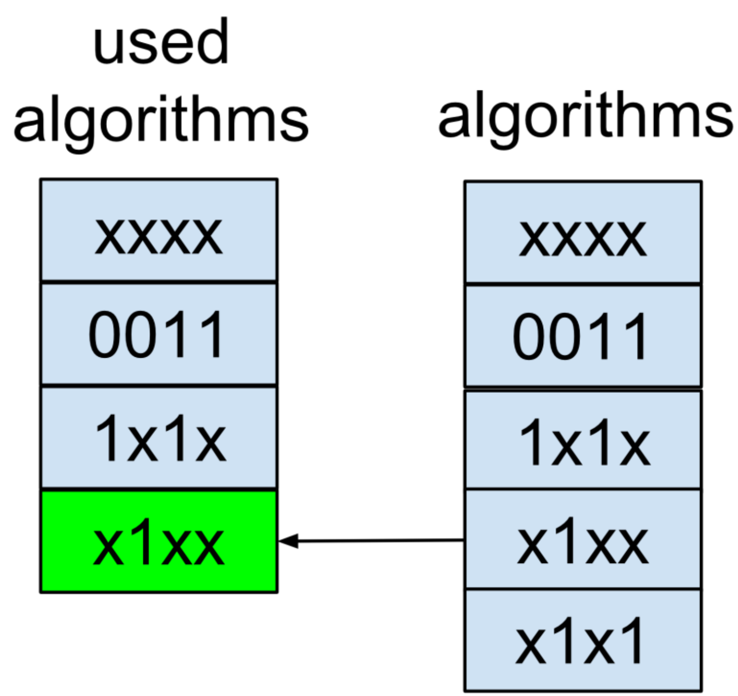
- Try to merge each pair of algorithms
- Until no more merges can be made

Constructing ECC using found algorithms

	S_i	S_{i+1}
in	A[0111]	B[1011]
out	1010	1110

bestMatch(1010,1110) = x1xx





Constructing ECC using found algorithms

$a \leftarrow \text{getBestMatch}(s_i.\text{out}, s_{i+1}.\text{out}, A)$
$y_{\text{new}} \leftarrow -1$
if $a \in A_{\text{used}}$ **then**
    $y_{\text{new}} \leftarrow A.\text{indexof}(a)$
else
    $A_{\text{used}} \leftarrow A_{\text{used}} \cup \{a\}$
    $y_{\text{new}} \leftarrow |A_{\text{used}}| - 1$
end if
$\text{t} = \text{new Transition}(s_{i+1}.e^{\text{in}}, s_{i+1}.\text{in}, y_{\text{new}})$
if $t \notin \tau_{y_{\text{current}}}$ **then**
    $\tau_{y_{\text{current}}} \leftarrow \tau_{y_{\text{current}}} \cup \{t\}$
end if

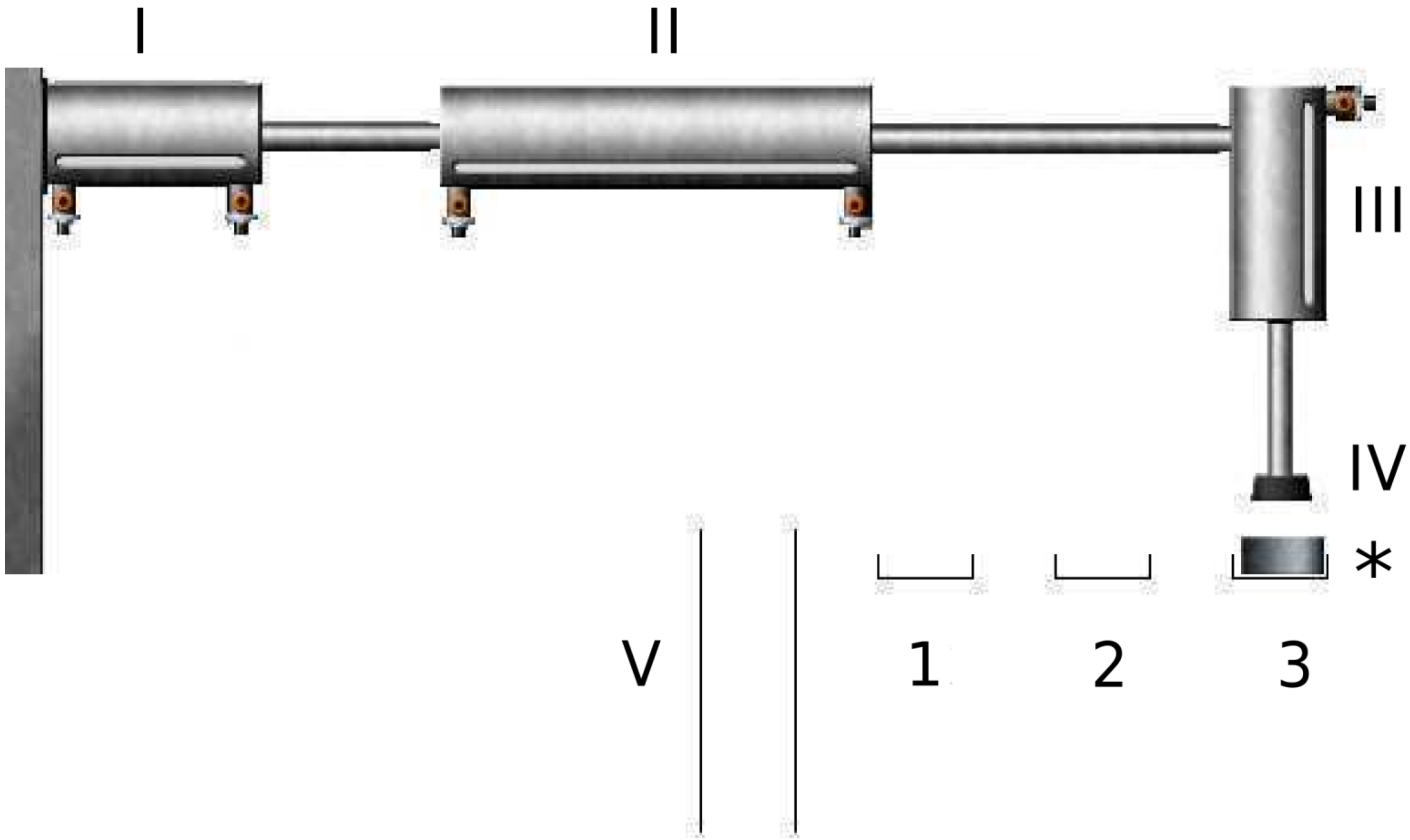
Simplifying ECC

- Constructed ECCs are **redundant**
- Each guard depends on **all** input variables

6 -> 5 ["REQ [c1Home & !c1End & vcEnd & !pp2]"];
~~6 -> 5 ["REQ [c1Home & !c1End & vcEnd & pp2] "];~~
7 -> 0 ["REQ [vcHome & !vac]"];
7 -> 3 ["REQ [vcHome & vac]"];
~~8 -> 7 ["REQ [!c1End & c2End]"];~~
8 -> 7 ["REQ [c1End & !c2End]"];
8 -> 7 ["REQ [c1End & c2End]"];
9 -> 1 ["REQ [c2End & vcHome & !vac]"];

6 -> 5 ["REQ [vcEnd]"];
7 -> 0 ["REQ [vcHome & !vac]"];
7 -> 3 ["REQ [vcHome & vac]"];
8 -> 7 ["REQ [!c2End]"];
8 -> 7 ["REQ [c2End]"];
9 -> 1 ["REQ [c2End]"];

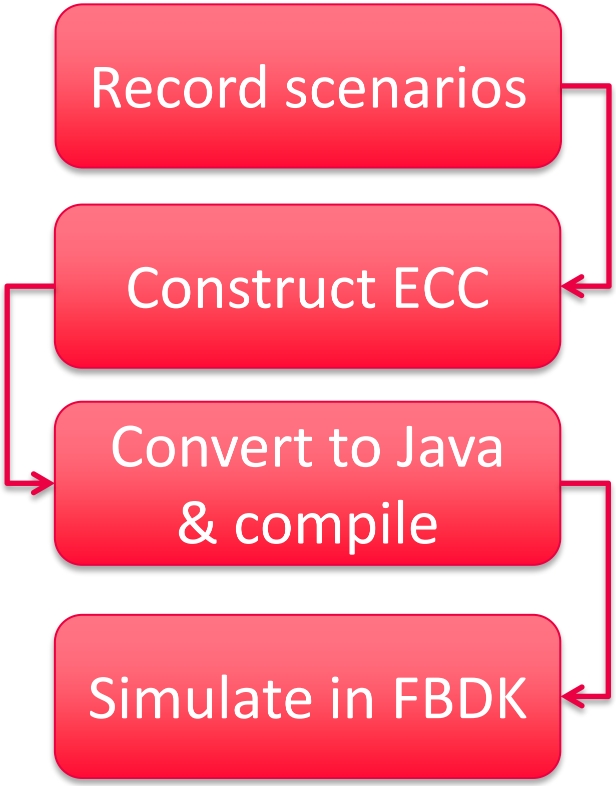
Experiments: Pick-n-Place manipulator



Experiment setup

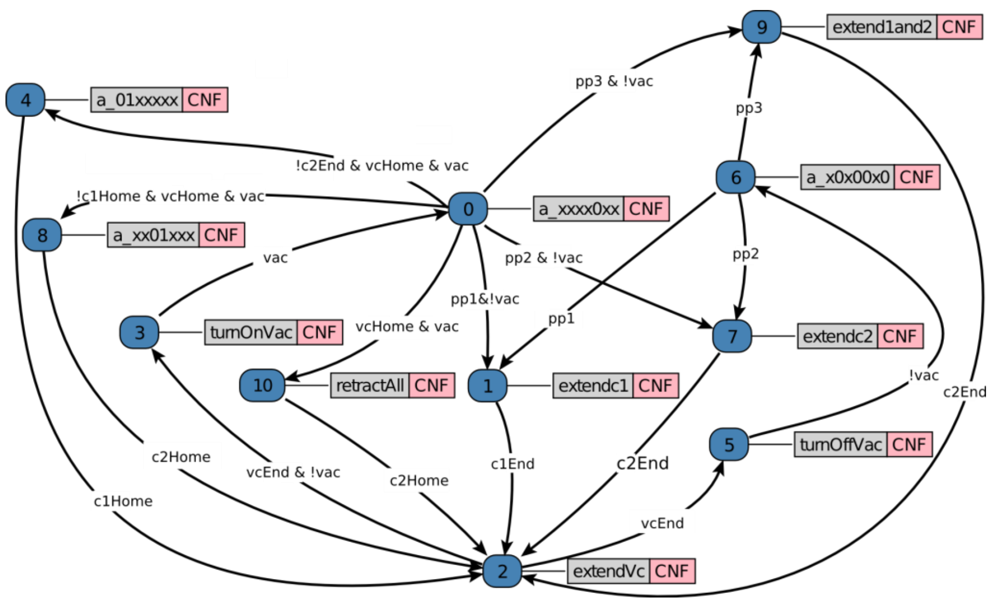
- 10 tests: order of work piece deployment
  - 1, 1-2, 2-3, 1-2-3, 2, 2-1, 2-3, 3-2, 3-2-1

Experiment protocol



Results

- Proposed method constructs the ECC in **less than a minute**
- Previous method required ~ 4.5 hours on 16-core machine
- Simulation showed that the ECC works correctly



Limitations & Future work

- Approach is only useful if **manual control** is **easier** than **designing** the ECC
- User bears all responsibility for scenario **correctness and completeness**
- What about **generalizing**?
  - Consider temporal properties

Acknowledgements

- This work was financially supported by the Government of Russian Federation, Grant 074-U01.

ITMO UNIVERSITY

Thank you for your attention!

rain.ifmo.ru/~chivdan