ITMO UNIVERSITY

LULEÅ UNIVERSITY OF TECHNOLOGY

Aalto University

Reconstruction of Function Block Logic using Metaheuristic Algorithm: Initial Explorations

Daniil Chivilikhin
PhD student
ITMO University

Anatoly Shalyto
Dr. Sci., professor
ITMO University

Sandeep Patil
PhD student
Luleå University of Technology

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

INDIN ’15, Cambridge, UK, July 23, 2015

Presentation given by

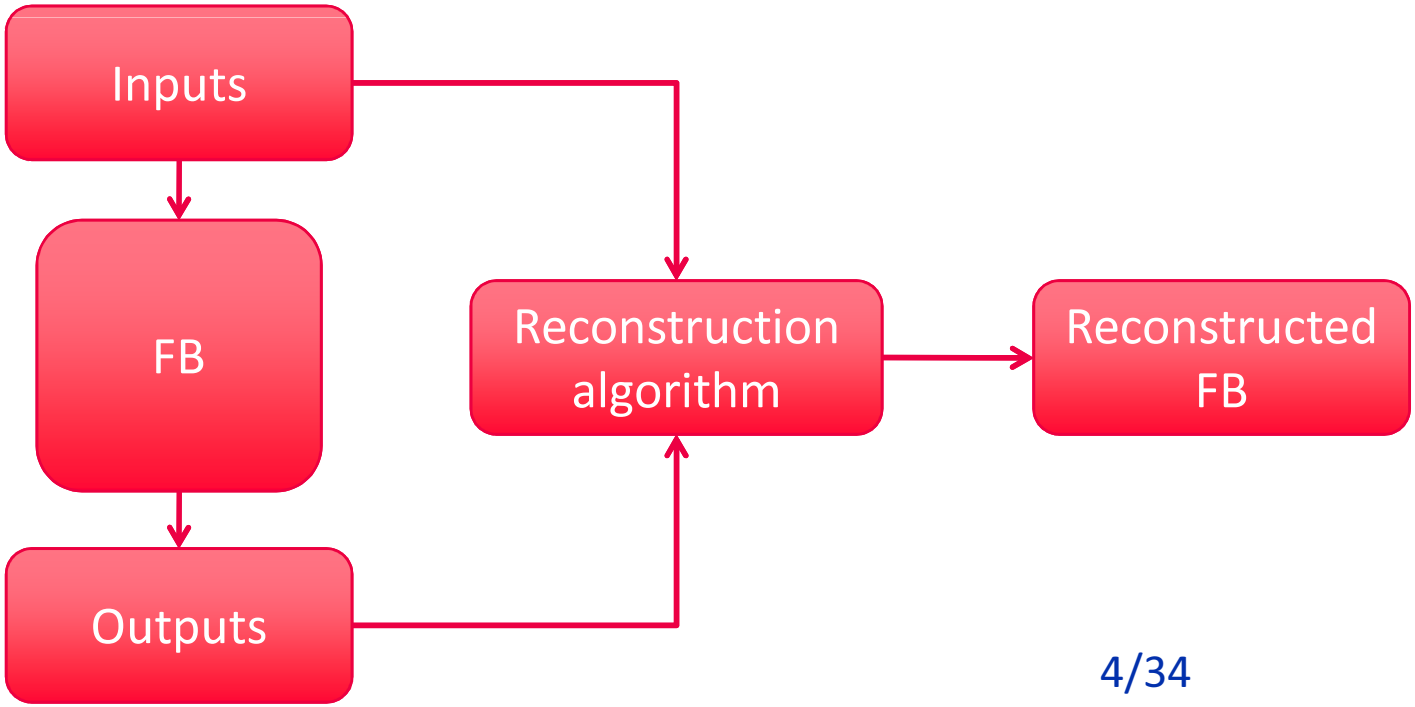
Igor Buzhinsky
PhD student
ITMO University

Motivation

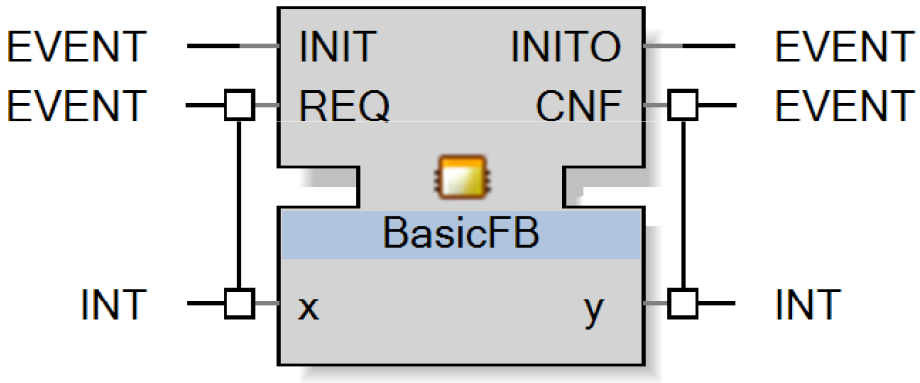
- Migration from legacy code to IEC 61499
- Existing approaches assume that source code is available
- What if
  - source code is lost?
  - there are no engineers that could quickly understand the code?

Problem statement

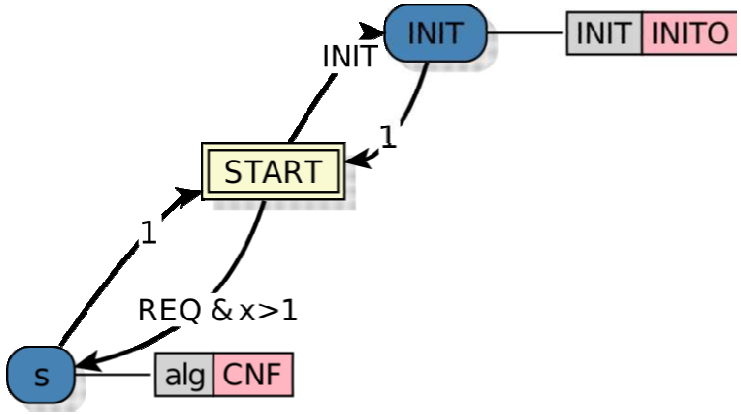
Reconstruct Function Block Logic without using code



IEC 61499 function blocks

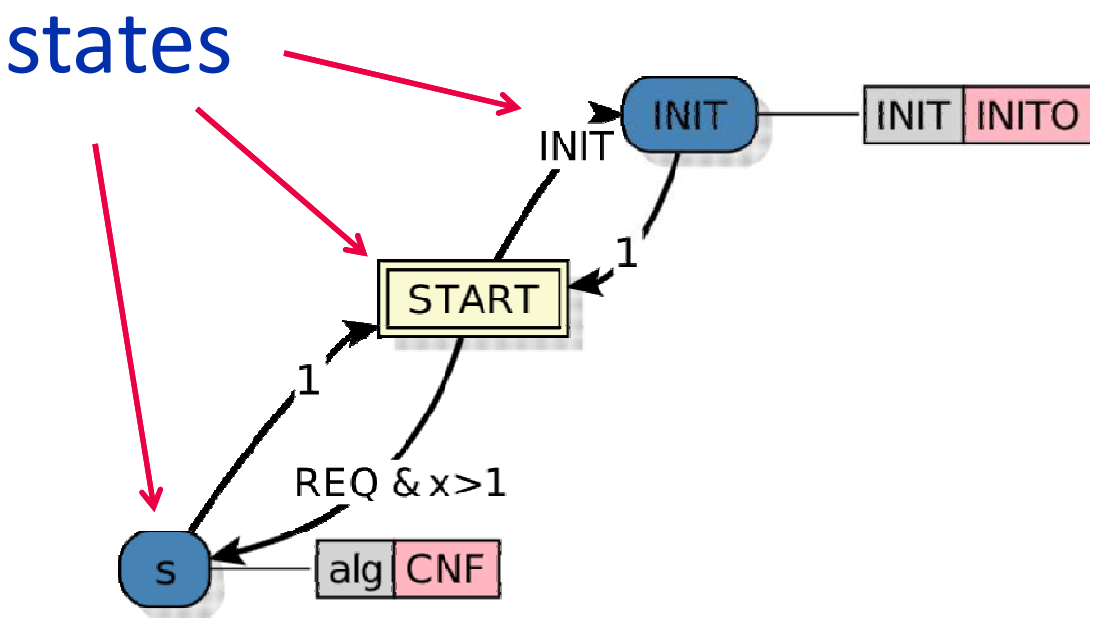


Function block interface



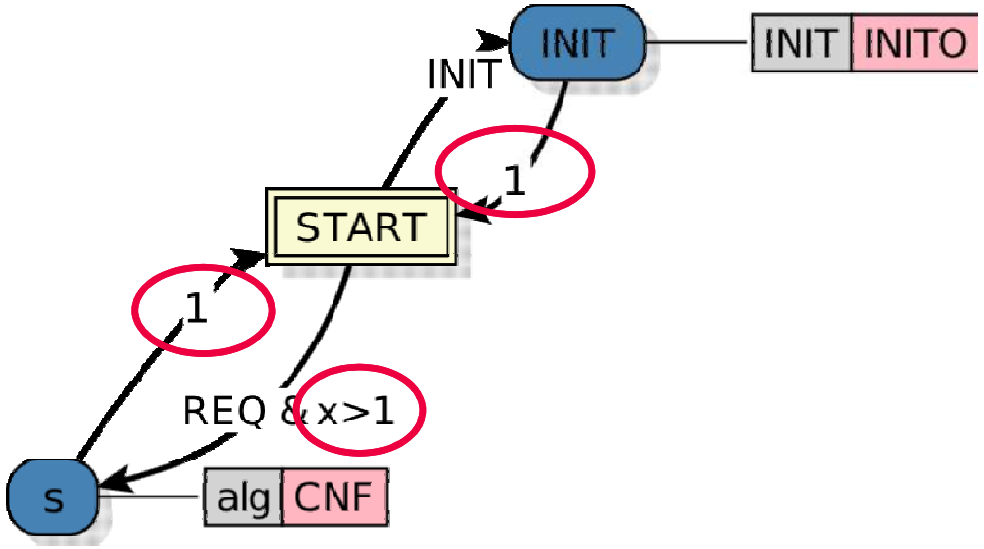
Execution Control Chart (ECC)

IEC 61499 Execution Control Chart



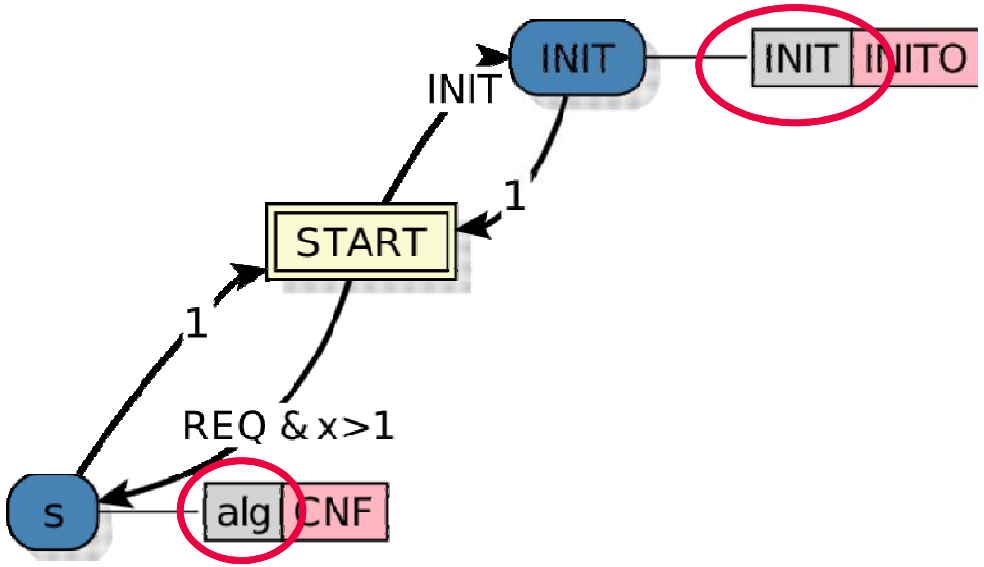
IEC 61499 Execution Control Chart

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



IEC 61499 Execution Control Chart

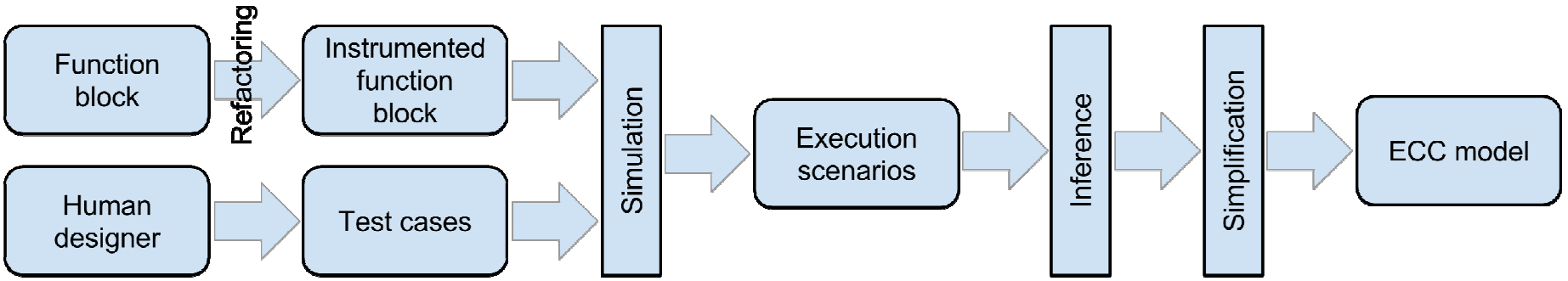
- Algorithms
- Change output variables



Simplifications

- No input/output events (only REQ and CNF)
- Only Boolean input/output variables
- Guard conditions
  - only input variables

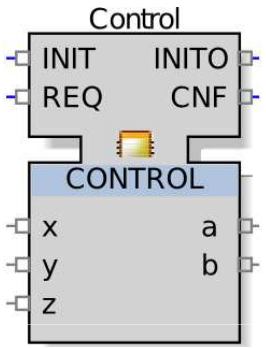
Proposed approach



Execution scenario

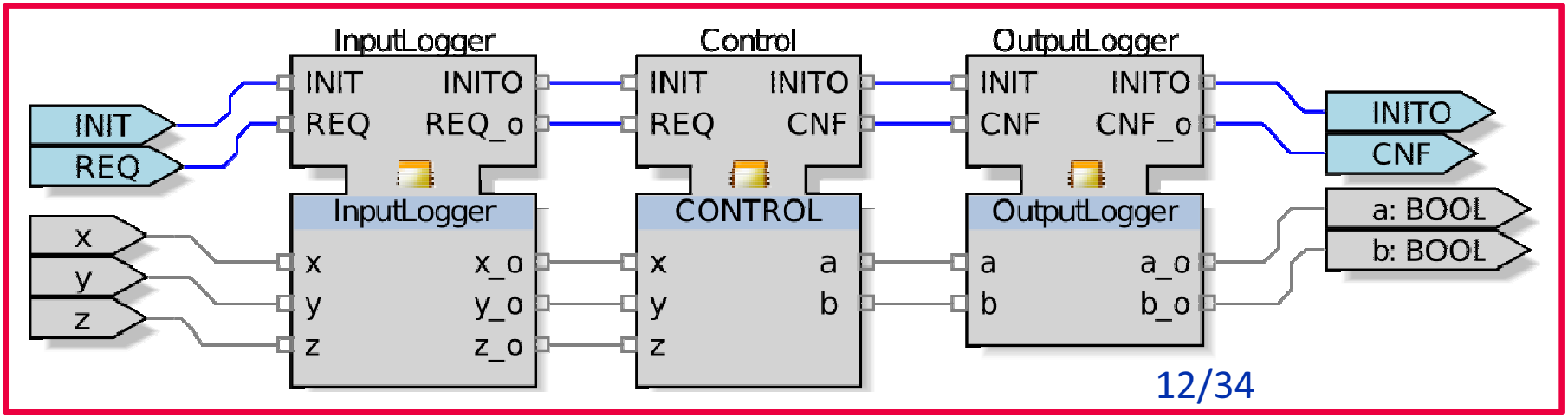
- List of scenario elements
- Scenario element = <(input variable values), (output variable values)>
- Scenario example
  - <000, 00>; <001, 01>; <101, 11>

Recording execution scenarios



Target function block

Automated refactoring



ECC construction algorithm (1)

- Parallel MuACO algorithm [Chivilikhin et al, 2014]
- Metaheuristic
  - Search-based optimization
  - Explore search space in a randomized way

ECC construction algorithm (2)

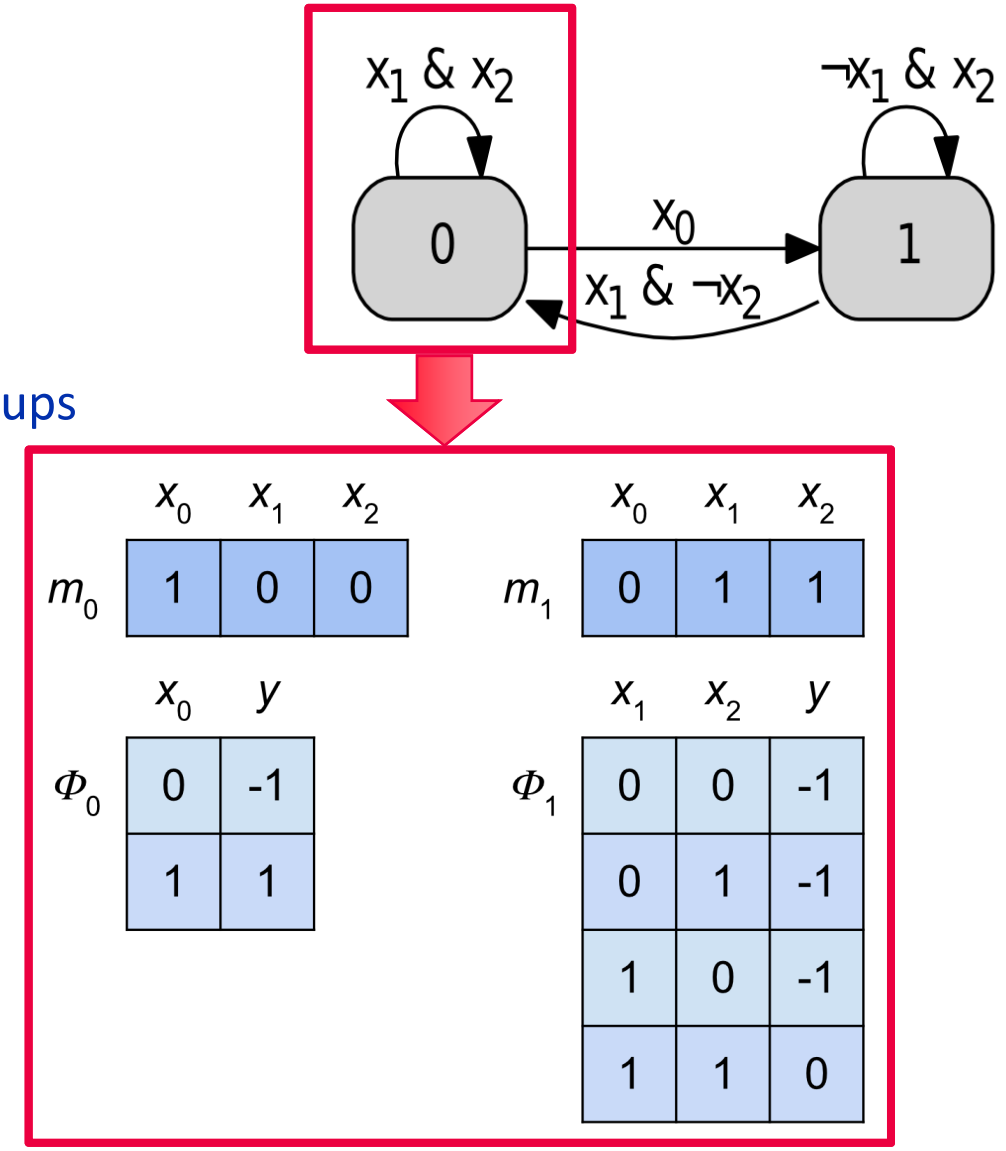
1. Start with **random** solution
2. **Build** new solutions with **mutation operators**
3. **Evaluate** new solutions with **fitness function**

ECC construction algorithm (3)

- Parameterized by
  - Solution representation (model)
  - Mutation operators
  - Fitness function

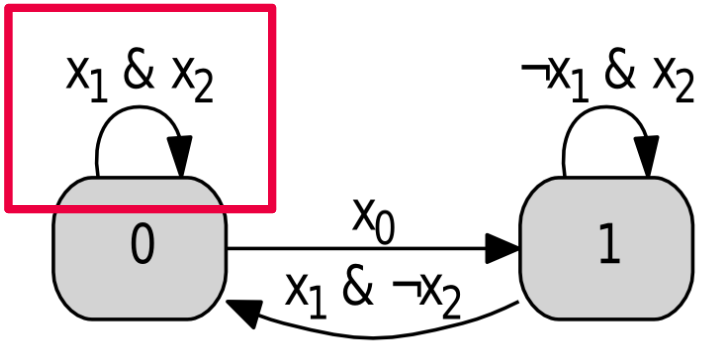
ECC model

- Set of states
- Each state – set of transition groups
- Each group
  - Variable significance mask
  - Reduced transition table
- Does not include algoritms



ECC model

- Set of states
- Each state – set of transition groups
- Each group
  - Variable significance mask
  - Reduced transition table
- Algorithms are not included



m_0

x_0	x_1	x_2
1	0	0

Φ_0

x_0	y
0	-1
1	1

m_1

x_0	x_1	x_2
0	1	1

Φ_1

x_1	x_2	y
0	0	-1
0	1	-1
1	0	-1
1	1	0

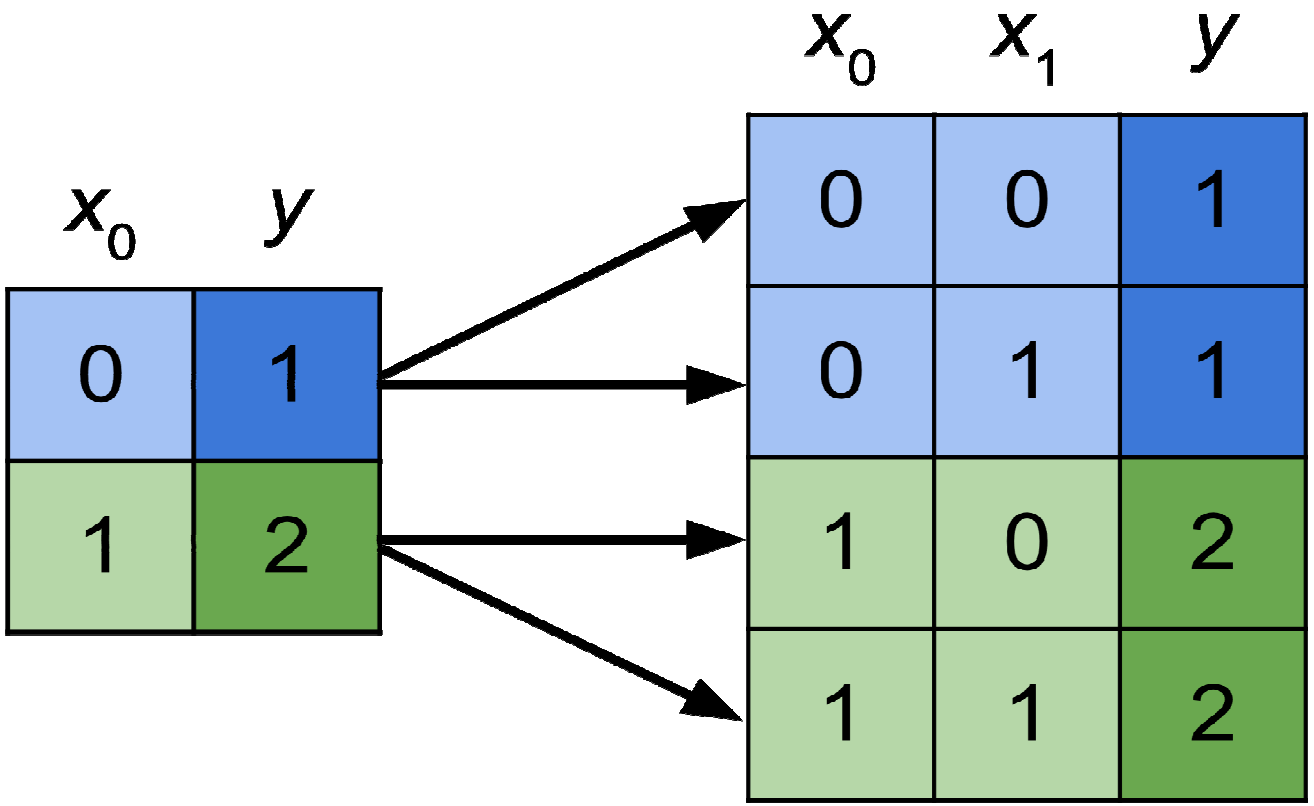
Algorithm representation

- 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

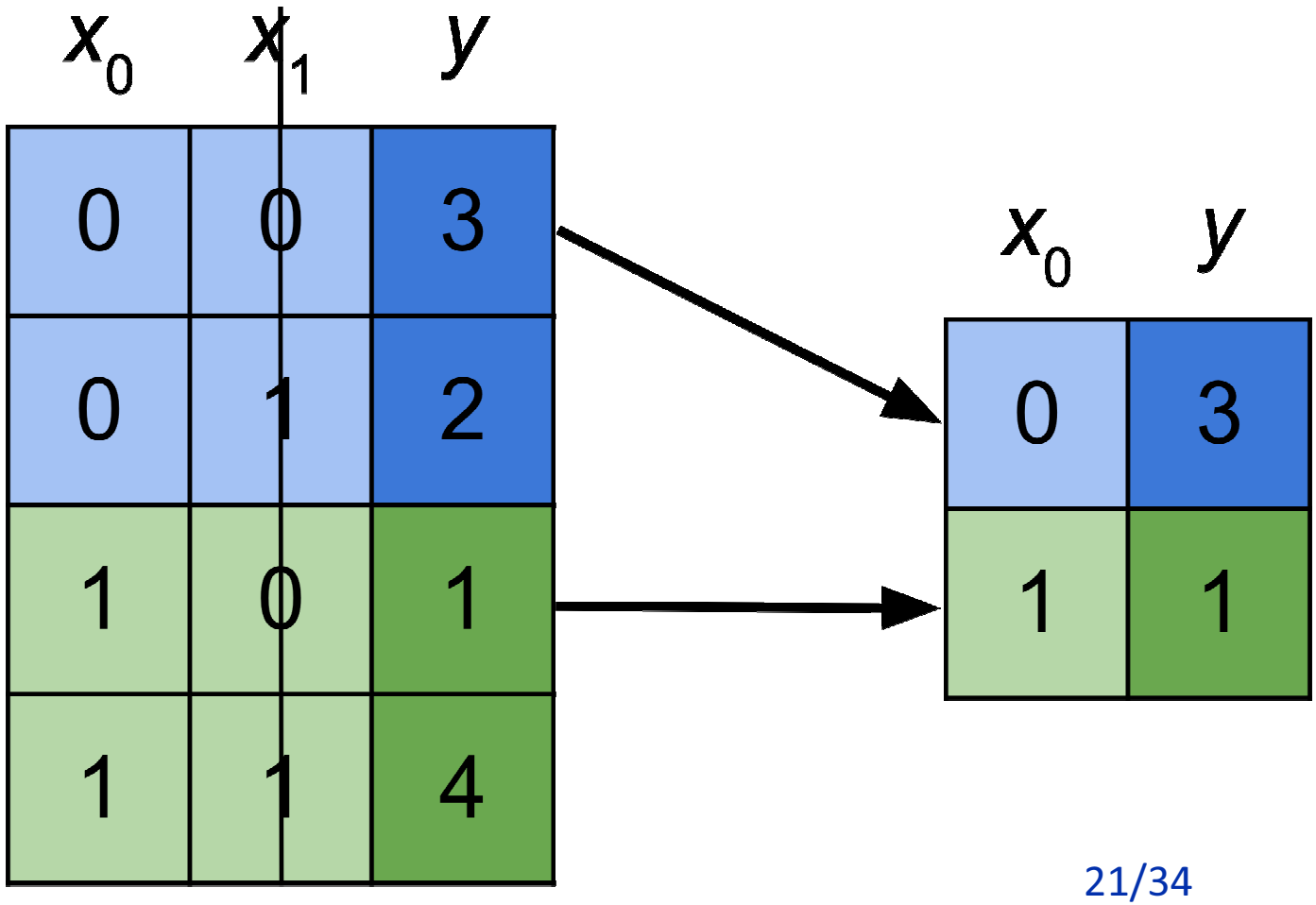
Mutation operators

- **Operator #1: Change transition end state**
  - Pick a random transition
  - Change the state it points to
- **Operator #2: Add/delete transitions**
  - Add random transition to a state
  - Delete random transition

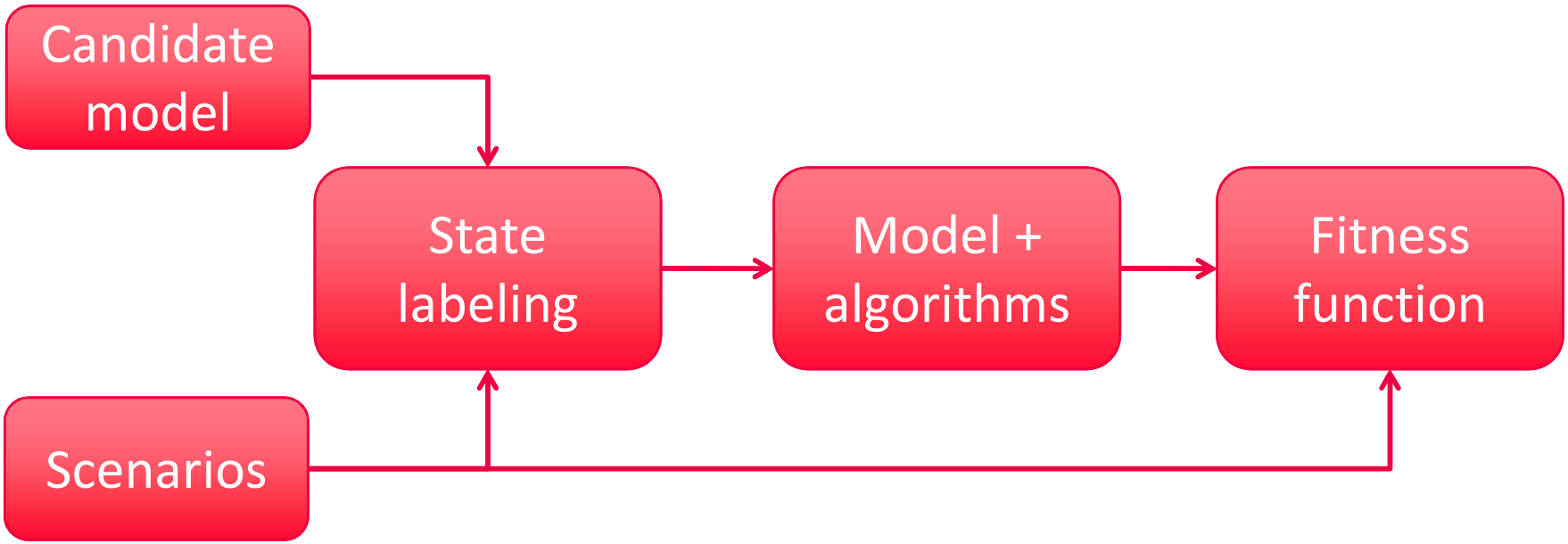
Mutation operator #3: Add significant variable



Mutation operator #4: Delete significant variable



Candidate model evaluation



State labeling: determine appropriate algorithms

- Run scenarios through ECC
- For each state and each output variable

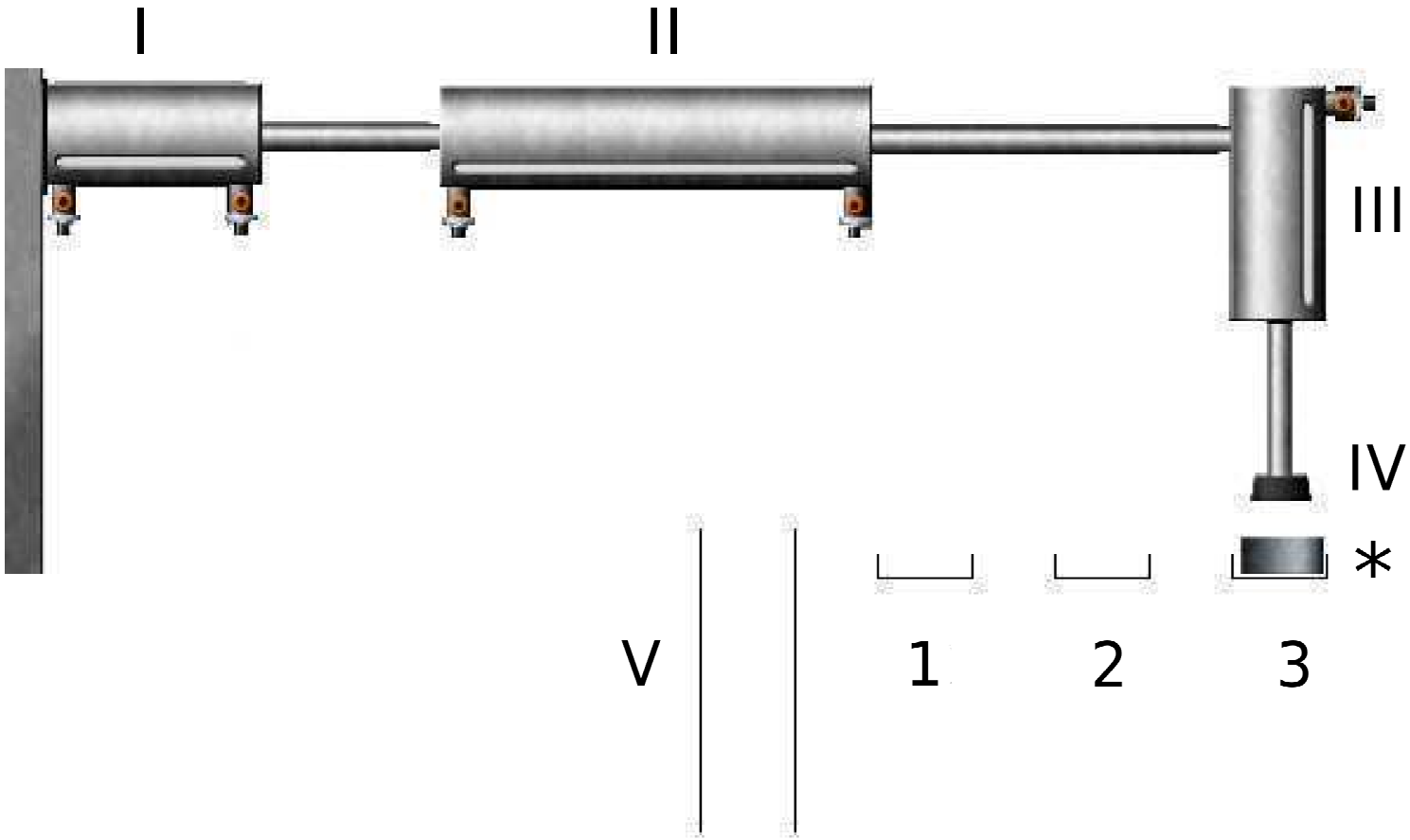
Change	Algorithm	Frequency
0 → 1	‘1’	37
1 → 0	‘0’	58
0 → 0	‘0’	0
1 → 1	‘1’	0

- a_i=‘0’

Fitness function

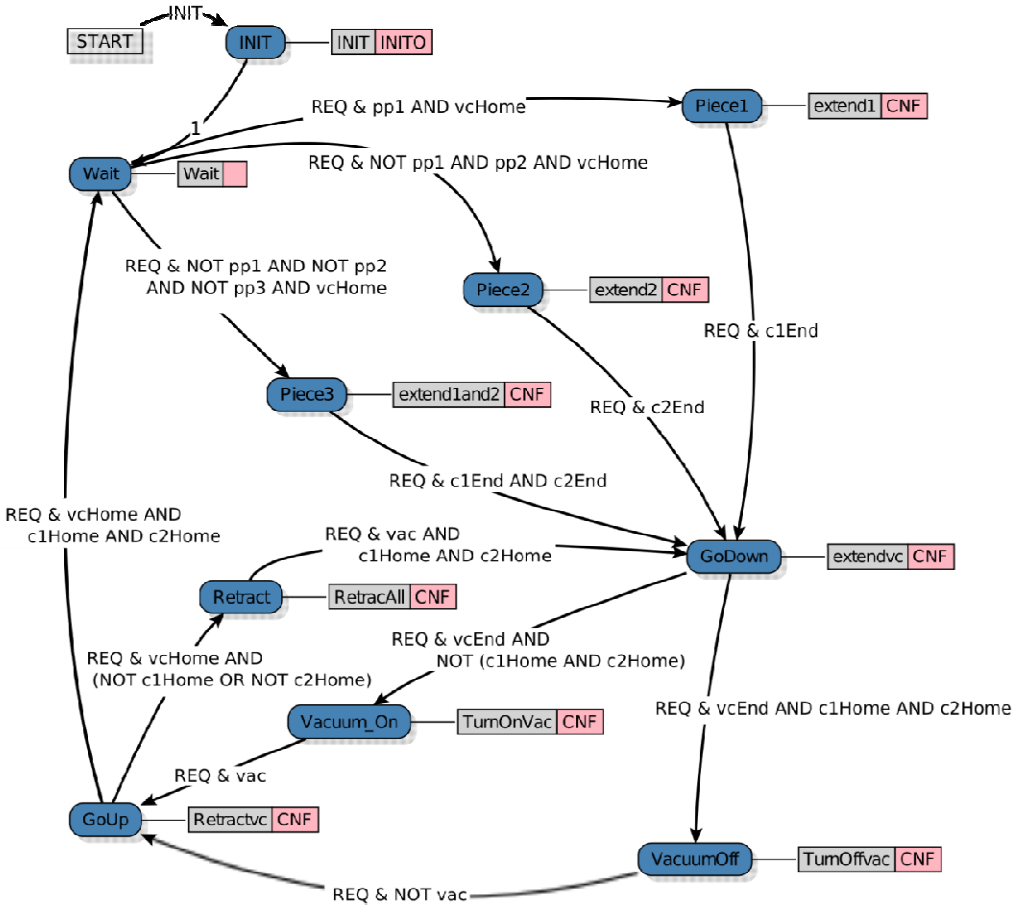
- Run scenarios through ECC
- $F = 0.9\ F_{ed} + 0.1\ F_{fe} + 0.0001\ F_{sc}$
- F_{ed} – edit distance between scenario outputs and candidate solution outputs
- F_{fe} – position of first error in outputs
- F_{sc} – number of times the ECC changed to a new state

Experiments: Pick-n-Place manipulator



Target ECC: CentralizedControl

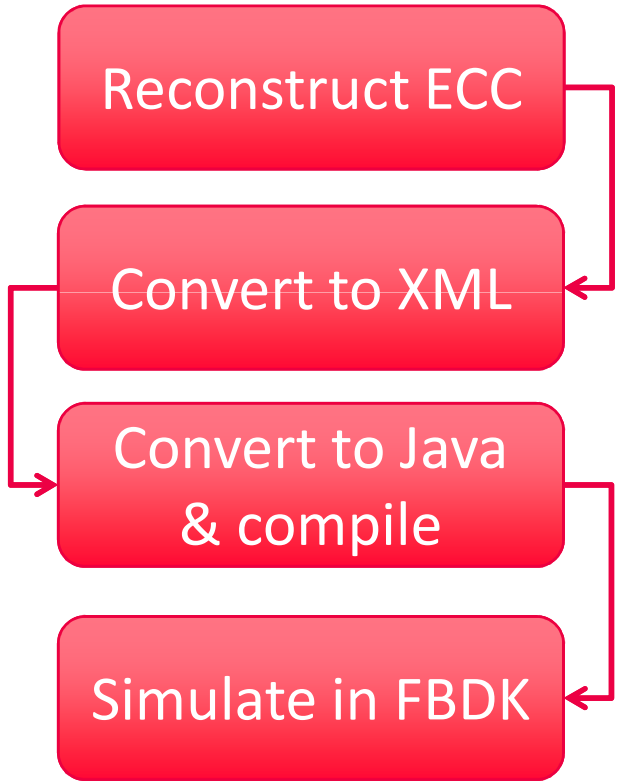
- 9 states
- 14 transitions



Experiment setup

- 10 tests: order of work piece deployment
  - 1, 1-2, 2-3, 3-2-1, …
- Models allowed to have
  - 10 states
  - 4 transition groups in each state

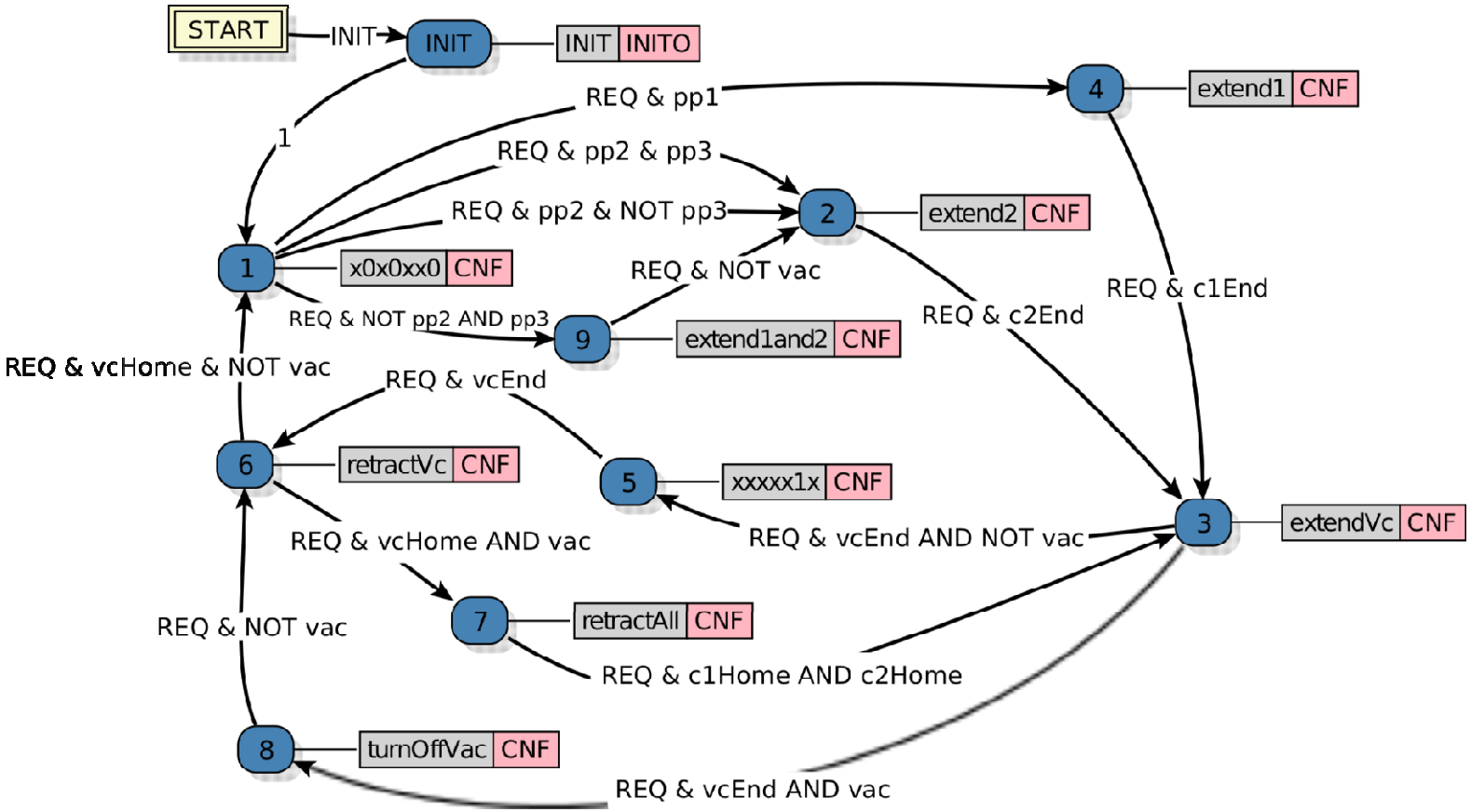
Experiment protocol



Results

- Used 16 cores of 64-core AMD Opteron™ 6378 @ 2.4 Ghz
- Experiment was repeated 20 times
- Average of 4.5 hours to infer perfect ECC
  - from 30 minutes to 10 hours
- All ECCs work correctly in simulation
- On longer test cases: OK

Inferred ECC example



Conclusion

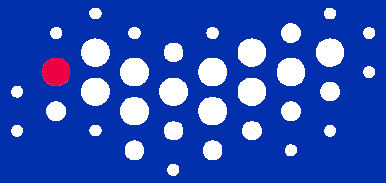
- Proposed an approach for reconstructing FB logic that does not require source code
- Performed sanity-check experiments of the proposed approach – it works

Future work

- Augment model with input/output events
- Handle other types of variables (int, real, string, …)
- Switch to inferring ECCs from expert data
  - Preliminary results @ ISPA'15 in August

Acknowledgements

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



ITMO UNIVERSITY

Thank you for your attention!

Daniil Chivilikhin
Sandeep Patil
Anatoly Shalyto
Valeriy Vyatkin

chivdan@rain.ifmo.ru

ECC model: representing guard conditions

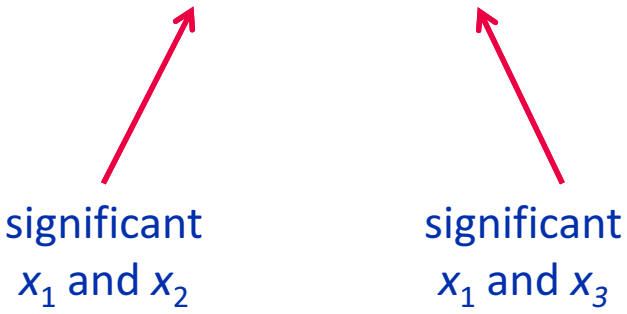
- Issue: large number of input variables
- Solution: reduced tables approach [Polikarpova et al, 2010]
  - variable significance mask
- Only supports simple formulas
  - $x_1 \wedge \neg x_2 \wedge x_3$

	x_0	x_1	x_2
m_1	0	1	1

	x_1	x_2	y
Φ_1	0	0	-1
	0	1	-1
	1	0	-1
	1	1	0

ECC model: representing guard conditions

- Need to support general-form formulas
  - For example: $x_1 \wedge (\neg x_2 \vee \neg x_3)$
- Represent in DNF
  - $(x_1 \wedge \neg x_2) \vee (x_1 \wedge \neg x_3)$



- Use several reduced tables per state

Model simplification

- Data should be explained by the simplest possible model
- Remove unused transitions
- Simplify guard conditions