7th UCAAT User Conference on Advanced Automated Testing

ETSI


# ALGEBRAIC METHODS OF MODEL-BASED TESTING OF DISTRIBUTED SYSTEMS


## Presented by Dr. Oleksandr Letychevskyi

**Glushkov Institute of Cybernetics, Kyiv, Ukraine**

# Our Algebraic Approach

- Glushkov's algebraic school is known from early 70-th: automated theorem proving;
- Algebraic Programing System (1990): symbolic computations, rewriting rules technique;
- Insertion Modeling (2000): generalization of transition systems interaction, theory of agents and environments;
- Behavior Algebra (2010) – symbolic modeling, predicate transformers theory, algebraic matching.

# Our Experience in MBT

- Motorola projects. Creation of test model behavior algebra specifications, generation of traces by symbolic modeling, concretizing and test creations (TAT technology);
- Generation of tests for legacy code testing. Conversion of Cobol programs to behavior algebra specifications and symbolic trace generation;
- Technology of white-box MBT;
- Decentralized (blockchain) systems testing. Extension of behavior algebra by usage of quantifiers, symbolic modeling of multiagent environment.

# Problem Statement

- We have distributed system consisting of the nodes interacting with each other by messages.
- Model-Based Testing is:
  - Creation of test model from requirements or design specifications;
  - Execute the system due to the behavior of the test model;
  - Check for compliance of system with model.
- Network can be heterogenous.

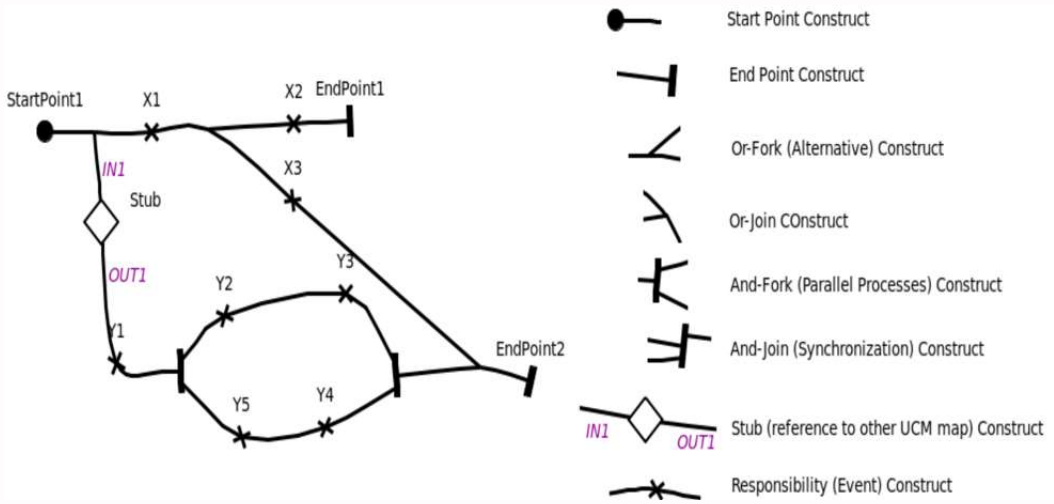
# Problem Statement

- Problems in testing of distributed system
  - Exponential explosion due to high interleaving of concurrent processes;
  - What is coverage? How many entities are needed?
  - Redundant testing;
  - Time-consuming testing, problem of testing harness;
  - Thresholds, overloading;
  - Leveraging of abstract levels;
  - When stop the test generation?

# Behavior Algebra as Specifications for Test Model

- Within the scope of the insertion modeling method we use the *behavior algebra* specifications for the formalization of the project introduced in 1997, by D. Gilbert and A. Letichevsky.
- The operations are the *prefixing a.u* (where *a* is an action and *u* is a behavior) and *non-deterministic choice* of behaviors *u + v*. The terminal constants are successful termination Δ, deadlock 0, and unknown behavior ⊥. The approximation relation ⊑ is a partial order on the set of behaviors.
- *B0 = a1.a2.B1 + a3.B2, B1 = a4, B2 =…*
- The behavior algebra is also enriched by two operations: parallel (||) and sequential (;) compositions of behaviors.

# Behavior Algebra. Use Case Maps.



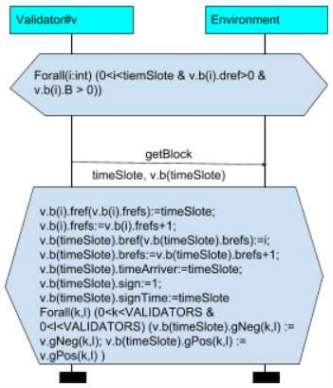

Behavior algebra operations can be mapped to the UCM. It is used for visualizing the graphical scenario that corresponds to behavior algebra expressions.

# Behavior Algebra. Actions.

- Every action is also defined by a couple, namely, the **precondition** and **postcondition** of an action, given as an expression in some formal theory.

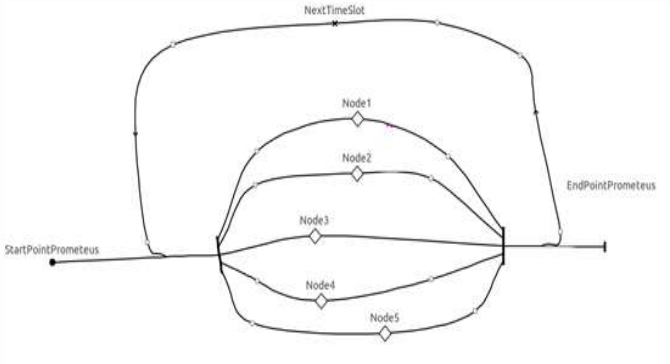
$$Action(A,B) = (A > B) \;\&\&\; !(A == 0) \;\text{->}\; B = (B + 1)/A$$

The semantic of the action presented in C-like syntax means that if precondition $(A > B)$ $\&\&\; !(A == 0)$ is true for concrete values of $A$ and $B$ or is satisfiable for symbolic (arbitrary) values of $A$ and $B$, then we can change attribute B by the assignment $B = (B + 1)/A$. The action can be parametrized by the attributes used in the action's conditions.




Action presented as MSC diagram. The transition between two states can be illustrated by MSC statement (message sending, local actions).

# Example of Consensus Protocol

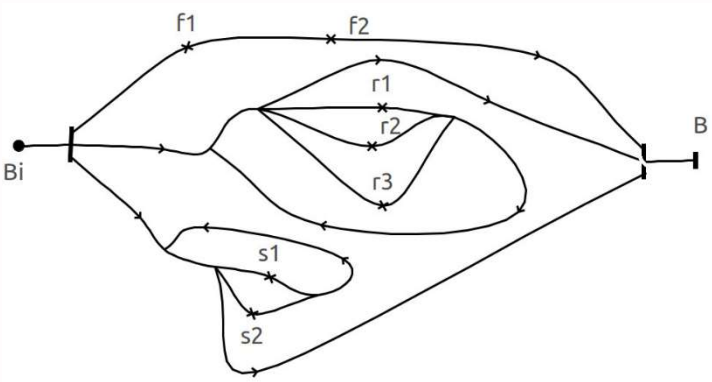



- The nodes work concurrently and in terms of insertion modeling, their high-level behavior *B0* can be presented by the following behavior algebra expression:

$$B0 = B;B0 + \Delta,$$

$$B = NextTimeSlot.(B1 \parallel B2 \parallel \ldots \parallel Bn)$$

or $B = NextTimeSlot.(\parallel Bi,\ 1<=i<=n)$




$$Bi = F \parallel R \parallel S,$$

$$F = f1.f2.B,$$

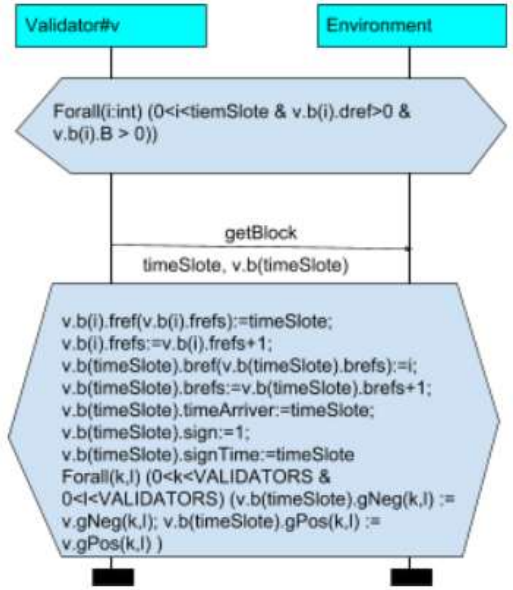
$$R = r1.R + r2.R + r3.R + B,$$

$$S = s1.S + s2.S + B$$

# Example of Consensus Protocol

- During its time slot, each validator can perform the following concurrent actions:
- form a block during predefined time slot *(f1)*, create references for other blocks, and send block *(f2)*;
- receive blocks *(r1)* and send/receive positive gossip messages about the receiving of missed blocks *(r2/r3)*;
- send/receive negative gossip messages about missed blocks in the previous time slot *(s1/s2)*.

*f1 = (timeslot == blockID) -> p(f1) (BlockCreation()),*
*f2 = 1 -> p(f2) (ReferenceCreation()),*
*r1 = ~(timeslot == blockID) -> p(r1) ReceiveBlock(),*
*r2 = 1 -> p(r2) (SendBlock()),*
*r3 = 1 -> p(r3) (ReceiveBlock()),*
*s1 = BlockMissed() -> p(s1) SendGossip(),*
*s2 = 1 -> p(s2) ReceiveGossip*



# Symbolic Modeling of Behavior Algebra Expressions

- The initial state of agents can be presented by an initial formula. Starting from the initial formula, we can apply actions corresponding to the behavior algebra expression.
- An action is **applicable** if its precondition is satisfiable and consistent with the current state. Starting from the formula of the initial state S0 and from the initial behavior B0, we select the action and move to the next behavior.
- In the first step, we check the satisfiability of the conjunction

$$S0 \wedge Pa1$$

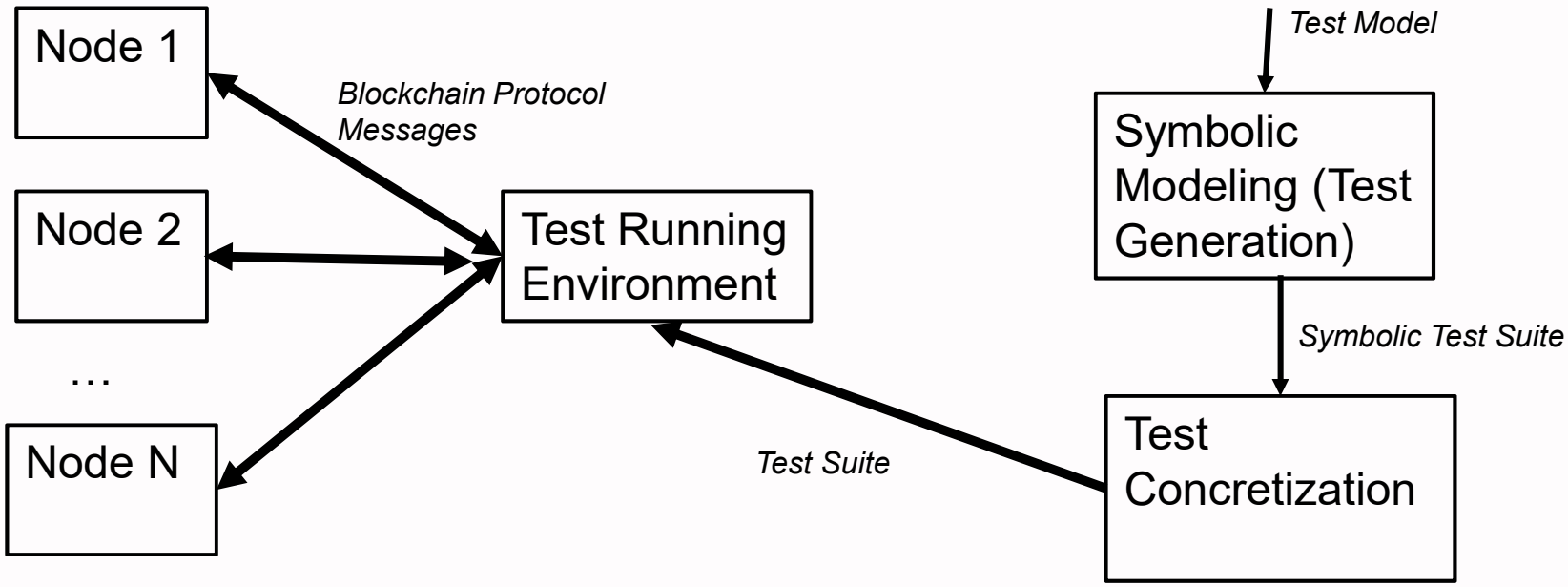
if B0 = a1.B1 and Pa1 is a precondition of a1.

The next state of the environment is obtained using a **predicate transformer**, that is, the function PT over the current agent state, precondition, and postcondition.

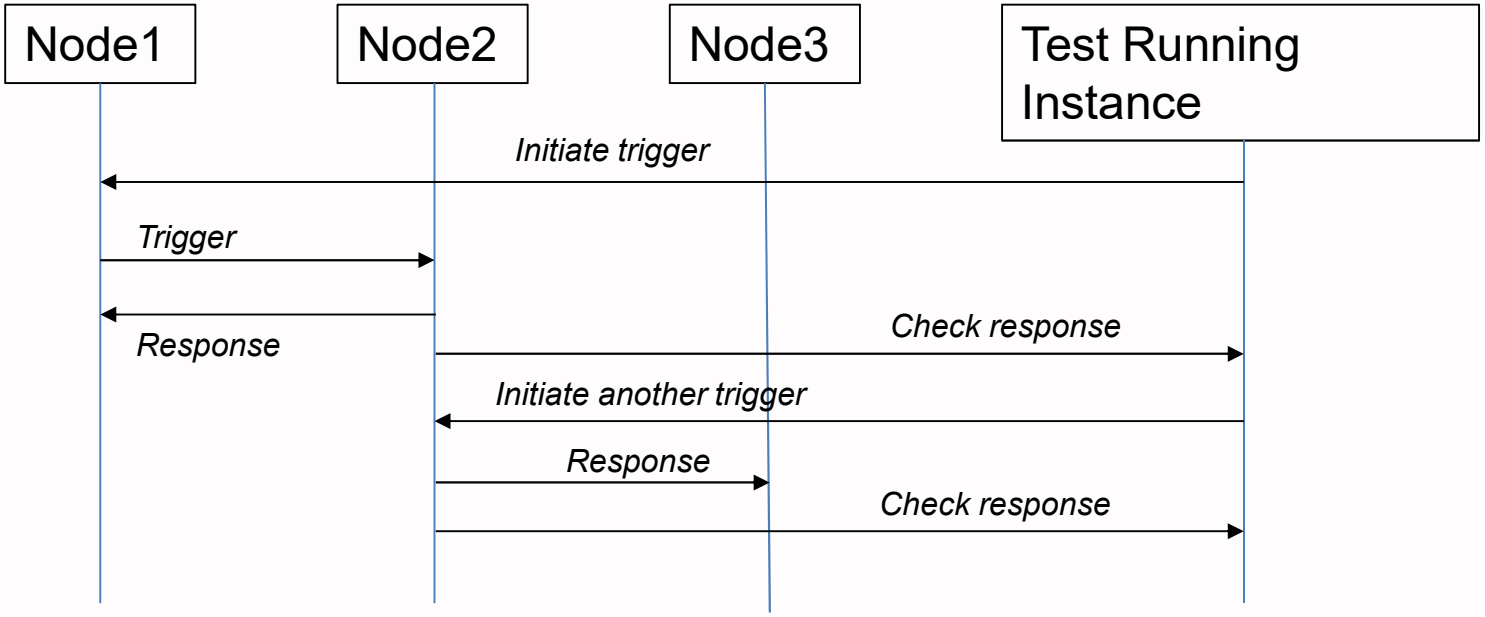
$$PT(Si, Pai, Qai) = Si+1$$

- By applying the predicate transformer function to different agent states, we can obtain the sequence S0, S1,… of formulas that express the agent states changing from the initial state. We present the **trace** by the sequence of actions a1, a2,…

# MB testing



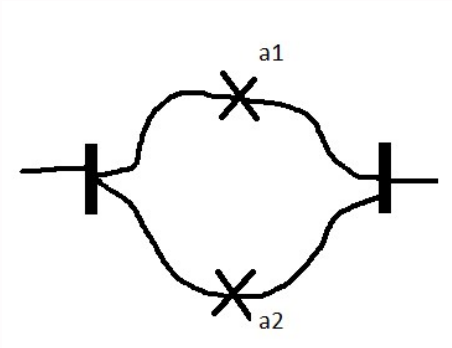
# MB testing




Every Node has triggering (user) actions that initiate transaction. Special library is used for test harness especially for checking of the environment conformance and initiating triggering events.

# Permutability.

- What is permutability?



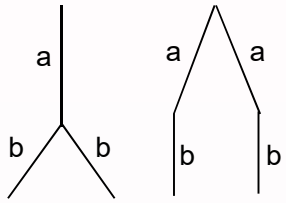

*B = a1 || a2 = a1.a2 + a2.a1*
*a1 = ($\alpha$, $a$), a2 = ($\beta$,$b$)*
Two actions are ***statically permutable*** if

$$pt(\alpha \wedge pt(\alpha \wedge \beta, b), a) = pt(\beta \wedge pt(\alpha \wedge \beta, a), b)$$
$$pt(\alpha \wedge pt(\neg\alpha \wedge \beta, b), a) = 0$$
$$pt(\beta \wedge pt(\alpha \wedge \neg\beta, a), b) = 0$$

*B = a1||a2 = a1.a2*

***It is not necessary to consider behavior a2.a1 in test generation***

# Behavior Equivalence




***Trace equivalence.***
*Two behaviors B1 and B2 are* ***trace equivalence*** *if T(B1) ⇔ T(B2) where T(B) is the set of linear behaviors b1.b2. …*
***Partial trace equivalence*** *of behaviors B1 and B2 if T(B1) ⊑ T(B2)*

***Context bi-similar equivalence*** *of behaviors B1 and B2 if*
*T(B1) ⇔ T(B2)*
*Preinv(B1) ⇔ Preinv(B2)*
*Postinv(B1) ⇔ Postinv(B2)*

***Equivalence is proving by rewriting technique with symbolic modeling***

***Generation of tests and coverage can be defined up to equivalence***

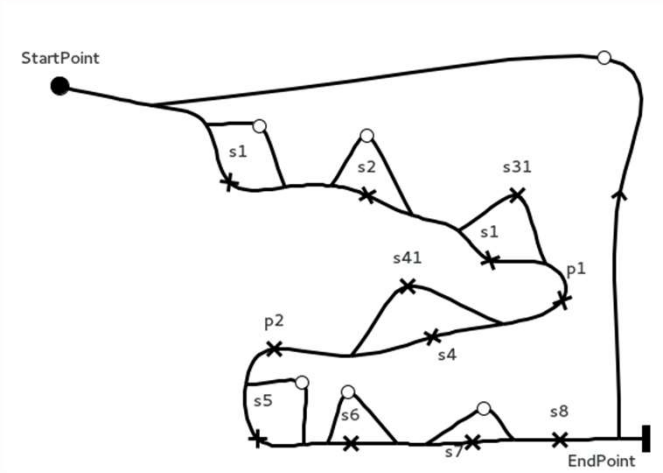
# Other properties

- Monotonicity of preinvariants (reducing of cycles);
- Encapsulation of behavior into action with definition of pre- and postinvariant;
- Transformation of behavior (reducing of parallel composition, elimination of multiple transition, exclusion of transit vertex, reducing of permutable paths)
- User defined equivalences and permutabilities

# Symbolic dynamic testing

- Symbolic dynamic (online) testing means modelling of ***controlled*** parallel composition of testing and tested model in behavior algebra environment;
- For this purpose we should translate code to behavior algebra expressions;
- Controlled parallel composition means implementation of trigger initialization algorithm with strategies

```
main() {
        int A=0,B=0,D, A_OUT = 0,B_OUT = 0;
        for (i=0;;i++) {
                if (A==0) A = load();
                        if (B==0) B = load();
                        if (A>=15) {A_OUT = A_OUT + 15;A = A – 15;}
        else {A_OUT = A_OUT + A;A = 0;}
        PERCENT_A(&A_OUT);
        if (B>=9) {B_OUT = B_OUT  + 9;B = B – 9;}
                        else {B_OUT = B_OUT  + B;B=0;}
        PERCENT_B(&B_OUT);
        if ((A_OUT <= B_OUT) && (A_OUT < 10)) {D = A_OUT; A_OUT = 0; B_OUT = B_OUT - A_OUT;}
        if ((A_OUT > B_OUT) && (B_OUT < 10)) {D = B_OUT; B_OUT = 0; A_OUT = A_OUT - B_OUT;}
        if((A_OUT<=B_OUT)&&(A_OUT>=10)|/(A_OUT>B_OUT)&&(B_OUT>=10))
                        {A_OUT=A_OUT -10; B_OUT=B_OUT -10; D=10;                }
        }
        produce(D);
}
```



# Symbolic dynamic testing

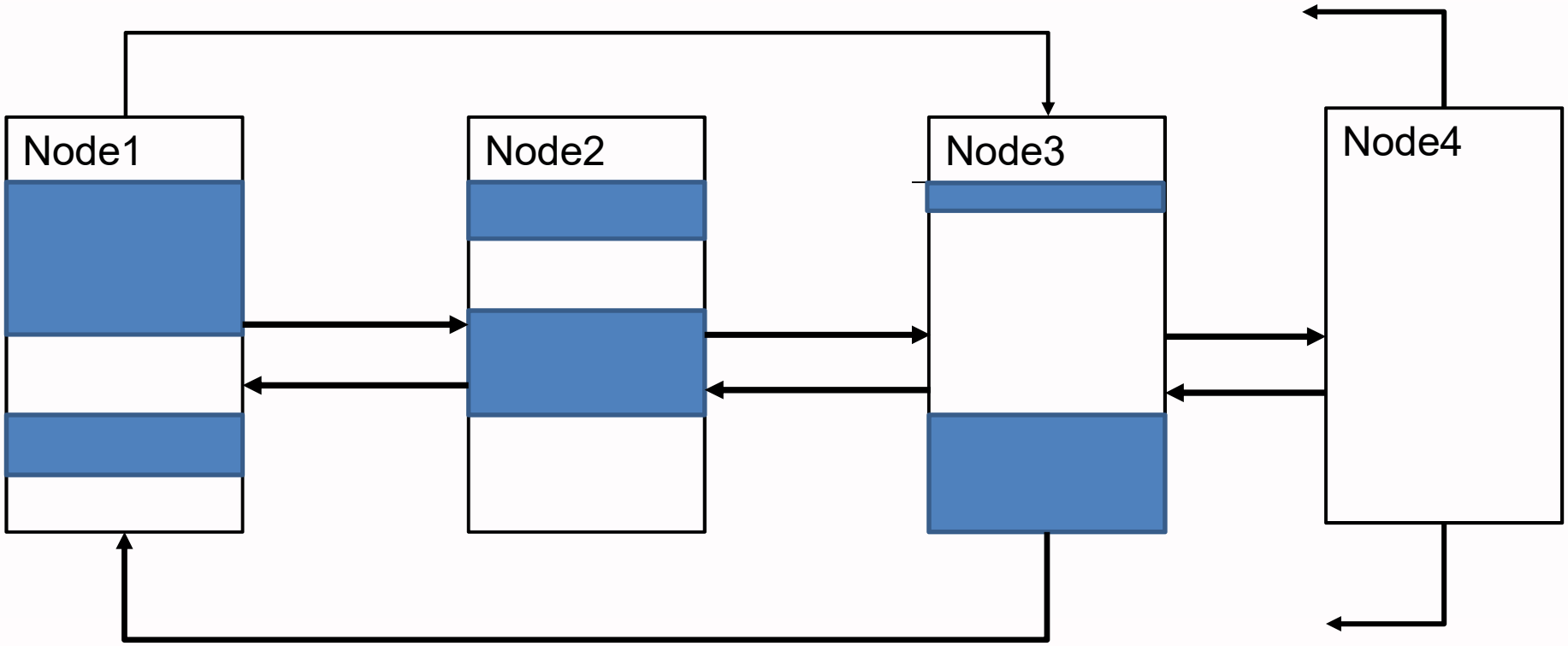
**Checking for conformance**

- Y – expected, X – current environment
- Y & ~X – unsatisfiable - **test OK**
- Y & ~X – satisfiable – **test can continue** but there exist states that are unexpected (Z = Y & ~X)
- X & Y – unsatisfiable – **test failure**

Benefits:

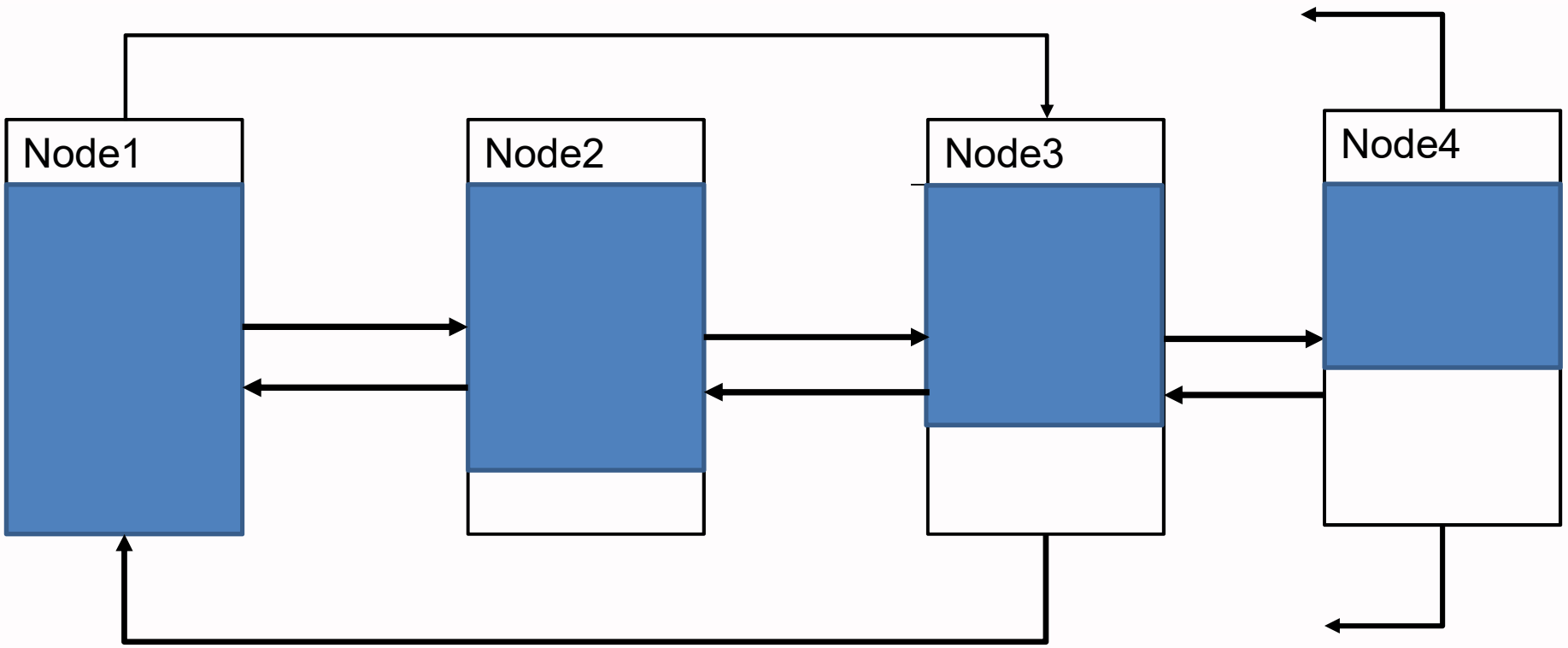
- We don't need test generation;
- Possible more deep checking;
- Coverage of states.

# Experiment 1. (symbolic dynamic testing)




Covered code lines up to trace equivalence

# Experiment 2. (symbolic dynamic testing)




Covered code lines up to context bisimilar equivalence

# Problem of symbolic testing for arbitrary number of nodes

This problem shall be decided in the scope of proving of equivalency of state without test generation.

# Verification and Case Generation

Checking of inconsistency and incompleteness in the behavior algebra expressions

$$\bigcap a_i = 0$$

$$\bigcup a_i = 1$$

Checking of safety properties

Generation of counterexample as test

# Conclusions and Future Usage of Algebraic Approach

Algebraic approach showed its efficiency in test generation and reducing of test number. Anyway some questions are still open: optimal coverage, optimal number of entities. New properties in the scope of behavior algebra shall be researched.

**Future research:**

Fuzzing techniques

Modeling of attacks

# Thank you!