Algebraic Approach in Cybersecurity

Dr. OLEKSANDR LETYCHEVSKYI

Glushkov Institute of cybernetics
National Academy of Sciences of Ukraine,
Kyiv, prospect Glushkova, 40,
Ukraine

Dr. Oleksandr Letychevskyi

Dr. Volodymyr Peschanenko

Prof. Alexander Letichevsky

Letichevsky Criterion

First parallel OS with distributed memory

80 Automatic Theorem Prover

90 Algebraic Programming System

Insertion Modeling System

2000 Verification of Requirements Specifications

Verification of Parallel Programs

2010 Model-Based Testing

Reverse Engineering

Cyber Security

The Defense Advanced Research Projects Agency (DARPA) launched the Cyber Grand Challenge to create defensive systems with purposes of automated, scalable, machine-speed detection of vulnerabilities and cyber infections.

MAYHEM
ForAllSecure team
from Pittsburg

XANDRA
TECH team
From Ithaka
New York

Mechanical Phish
Shellphish team
from Santa Barbara,
California,

**ALL THESE TEAMS USE SYMBOLIC (ALGEBRAIC APPROACH)
IN DETECTION OF VULNERABILITIES IN BINARY CODE**

Problems:

1. Pattern Matching.
   Redundant. Too many False Positive.
2. Simulation in isolated environment.
   Insufficient. Deep-hidden vulnerabilities cannot be detected.
3. Analysis of high-level programming languages.
   Insufficient. Security issues are on the level of third-party (non-compiled) libraries.

Our approach:

Use the algebraic matching and symbolic execution of binary code model.

1. Create algebraic model of binary code.
2. Formalize the existed vulnerabilities as algebraic patterns.
3. Provide algebraic matching of given vulnerability models with binary code model and detect the vulnerability candidates.
4. Prove the reachability of the detected vulnerability by symbolic execution.

Algebra of Behaviors

Algebra of behaviors was developed by D. Gilbert and A. Letichevsky (Senior) in 1997
It consider the operations over **actions** and **behaviors**.

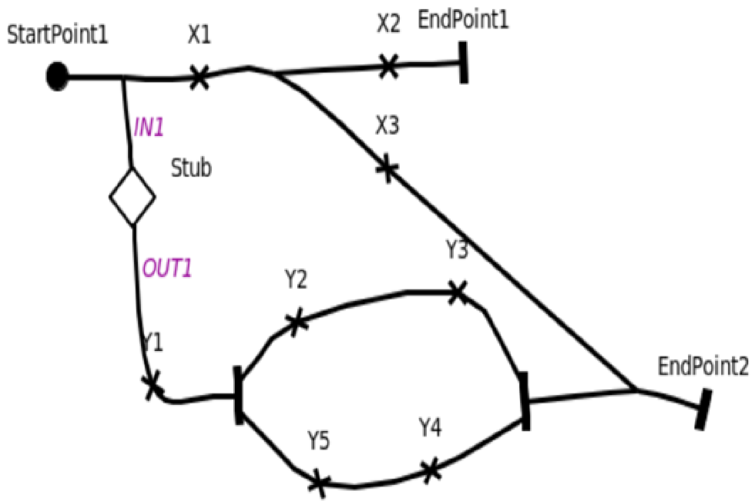
Prefixing operation *a.B* means that action *a* follows behaviour *B*. The operation of *nondeterministic choice* of behaviours *u* + *v* establishes alternative behaviours. The algebra has three terminal constants: successful termination Δ, deadlock 0, and unknown behaviour $\perp$. The parallel and sequential composition are defined on the behaviors.

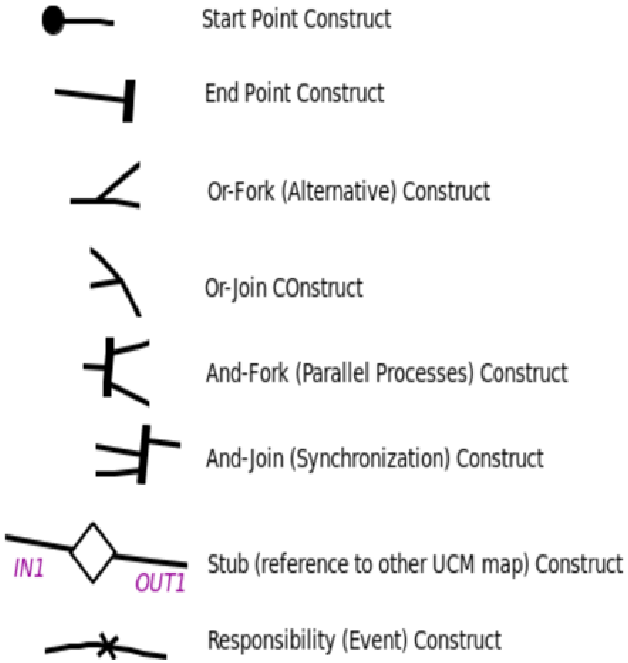
Example:

B0 = a1.a2.B1 + a3.B2,
B1 = a4.Δ,
B2 =…

The example define the order of event. The behavior B0 has two alternative - first is two actions a1 and a2 and then the rest behavior B1 or action a3 and rest behavior B2. Behavior B1 is action a4 and end of behavior etc.

Algebra Behavior (UCM diagram)





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

Algebra of Behaviors

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) -> *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.

Translation of x86 Assembler to Algebra of Behaviors

Listing after disassembling:

```
8049865:  2d e0 01 1d 08        sub eax,0x81d01e0
804986a:  c1 f8 02              sar eax,0x2
804986d:  89 c2                 mov edx,eax
8049876:  75 01                 jne 8049879
```

Model of binary code behavior:

```
B8049865 = sub(1,eax,0x81d01e0).B804986a,
B804986a = sar(1,eax,0x2).B804986d,
B804986d = mov(1,edx,eax,mov).B8049876,
B8049876 = jmp(1,jne).B8049879
```

Translation of x86 Assembler to Algebra of Behaviors

ALGEBRAIC ENVIRONMENT:

- the set of general- and special-purpose registers. Some attributes are identified as names of registers: *ax,al,bx,bl,..,eax,ebx,…,rax,rbx,…,ebp,esp,rbp,rsp,rip*
- physical memory that can be considered as the function *Memory(addr)*, where *addr* is the available memory address

Bx1 = cjne.Bz + !cjne.Bx2
Bx2 = …
cjne(n,A,B) = !(A == B) -> PI = PI + z + 3;FLAG_C = (B > A)
!cjne(n,A,B) = (A == B) -> PI = PI + 3;

Translation of x86 Assembler to Algebra of Behaviors

Set of Instructions is converted to

```
0000000000425060 <SSL_CTX_use_certificate_file>:
  425060:   41 55                   push   r13
  425062:   41 54                   push   r12
  425064:   49 89 f5                mov    r13,rsi
  425067:   55                      push   rbp
  425068:   53                      push   rbx
  425069:   49 89 fc                mov    r12,rdi
  42506c:   89 d5                   mov    ebp,edx
  42506e:   48 83 ec 08             sub    rsp,0x8
  425072:   e8 d9 24 fe ff          call   407550 <BIO_s_file@plt>
  425077:   48 89 c7                mov    rdi,rax
  42507a:   e8 a1 31 fe ff          call   408220 <BIO_new@plt>
  42507f:   48 85 c0                test   rax,rax
  425082:   0f 84 b0 00 00 00       je     425138
<SSL CTX use certificate file+0xd8>
  425088:   4c 89 e9                mov    rcx,r13
```

Set of Algebra Behavior Expressions

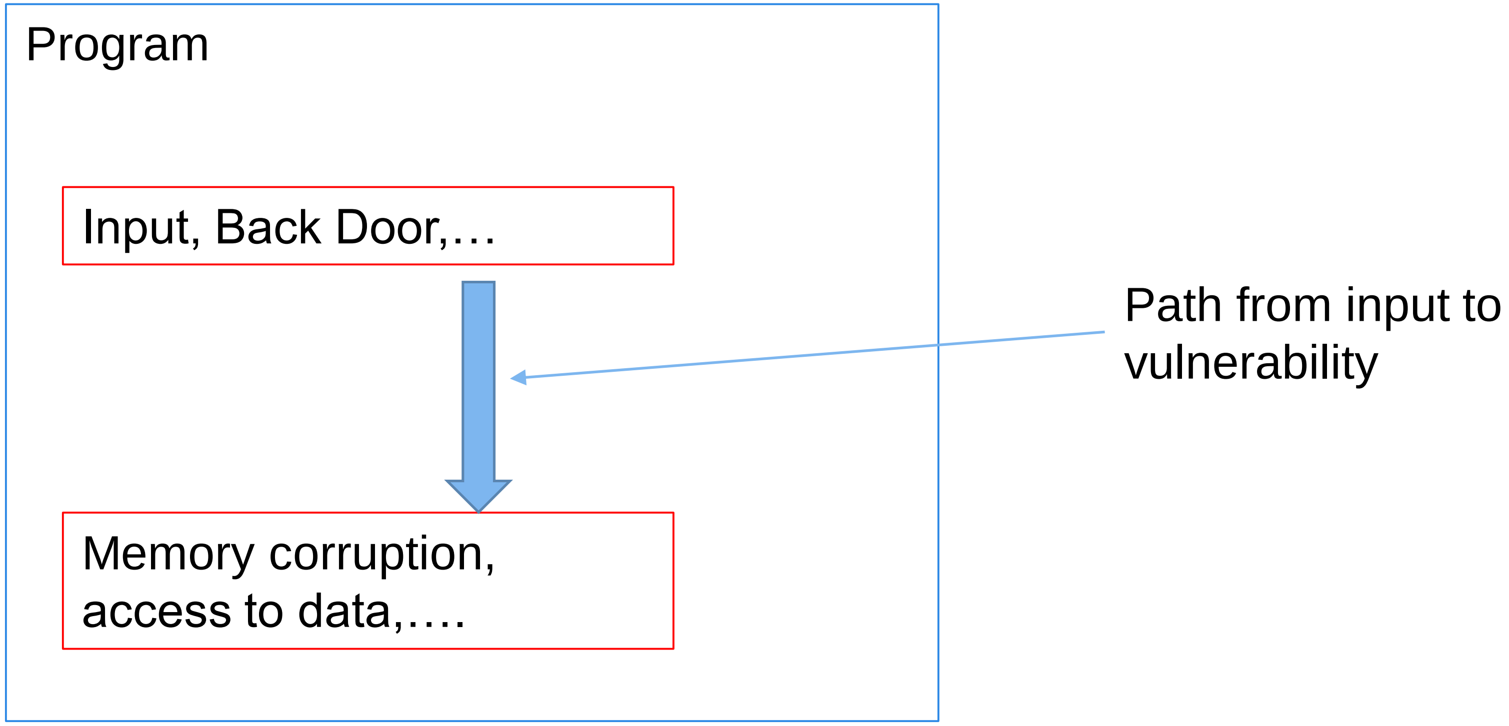
```
B425060 = a_push_33766.B425062,
B425062 = a_push_33767.B425064,
B425064 = a_mov_33768.B425067,
B425067 = a_push_33769.B425068,
B425068 = a_push_33770.B425069,
B425069 = a_mov_33771.B42506c,
B42506c = a_mov_33772.B42506e,
B42506e = a_sub_33773.B425072,
B425072 = a_call_33774.call B407550.B425077,
B425077 = a_mov_33775.B42507a,
B42507a = a_call_33776.call B408220.B42507f,
B42507f = a_test_33777.B425082,
B425082 = a_je_33778.B425138 + a_alt_je_33779.B425088,
B425088 = a_mov_33780.B42508b,
```

Set of Algebra Behavior Actions

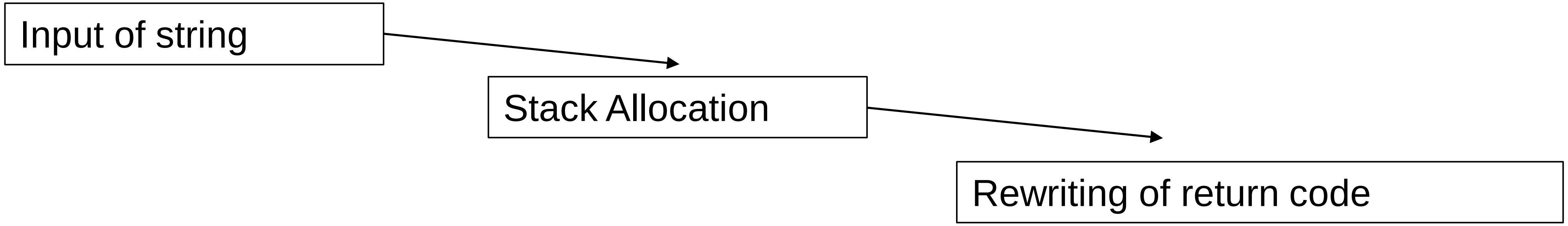
```
a_push_33766 = Operator(1  -> ("x86: action 'push 425060';")
(rip := 4345954)),
a_push_33767 = Operator(1  -> ("x86: action 'push 425062';")
(rip := 4345956)),
a_mov_33768 = Operator(1  -> ("x86: action 'mov 425064';")
(rip := 4345959; r13 := rsi)),
a_push_33769 = Operator(1  -> ("x86: action 'push 425067';")
(rip := 4345960)),
a_push_33770 = Operator(1  -> ("x86: action 'push 425068';")
(rip := 4345961)),
a_mov_33771 = Operator(1  -> ("x86: action 'mov 425069';")
(rip := 4345964; r12 := rdi)),
a_mov_33772 = Operator(1  -> ("x86: action 'mov 42506c';")
(rip := 4345966; ebp := edx)),
a_sub_33773 = Operator(1  -> ("x86: action 'sub 42506e';")
(rip := 4345970; rsp := rsp - 8; ZF := (rsp - 8 = 0); PF :=
((rsp - 8) = 0); SF := (rsp - 8) < 0))),
a_call_33774 = Operator(1  -> ("x86: action 'call 425072';")
(rip := 4345975)),
a_mov_33775 = Operator(1  -> ("x86: action 'mov 425077';")
(rip := 4345978; rdi := rax)),
a_call_33776 = Operator(1  -> ("x86: action 'call 42507a';")
(rip := 4345983)),
a_test_33777 = Operator(1  -> ("x86: action 'test 42507f';")
(rip := 4345986)),
a_je_33778 = Operator((ZF = 1)  -> ("x86: action 'je
425082';")(rip := 4345992)),
a_alt_je_33779 = Operator(( ~(ZF = 1))  -> ("x86: action 'je
425082';")(rip := 4345992)),
a_mov_33780 = Operator(1  -> ("x86: action 'mov 425088';")
(rip := 4345995; rcx := r13)),
```

Algebraic Patterns of Vulnerabilities

VulnerabilityPattern = IntruderInput; ProgramBehavior; VulnerabilityPoint



Algebraic Patterns of Vulnerabilities



BUFFER OVERFLOW VULNERABILITY

vulnerabilityBufferOverflow = input; X1; allocateStack; X2; writeStack,

input = mov(9, eax, 0x66). mov(10, ebx, 0x11). lea(11, ecx, MemoryOperand). call(12, MemoryOperand),

allocateStack = push(1,ebp).mov(2,ebp,esp).sub(3,esp,N),

writeStack = movs(8, MemoryOperand, MemoryOperand) + mov(5, MemoryOperand, regGen)

Algebraic Patterns of Vulnerabilities

ACTIONS:

call(12,MemoryOperand) = Forall(i:int, 0<=i<LengthSocket) -> Input(ecx + i) = true,
mov(2,ebp,esp)=1->StackAddr=ebp,

movs(8,MemoryOperand,MemoryOperand) = Input(RefMemorySrc) &&(StackAddr==RefMemoryDest) -> 1,
mov(5, MemoryOperand, regGen) = Input(RefMemorySrc) && (StackAddr == RefMemoryDest) -> 1,

mov(x, GenReg, MemoryOperand) = Input(RefMemorySrc) -> Input(GenReg),
mov(x, MemoryOperand, MemoryOperand) = Input(RefMemorySrc) -> Input(RefMemoryDest)

Sufficient conditions of vulnerability:

1. The bytes are written in address that is equal to the top stack pointer;
2. The bytes shall be received from the registers after system call

Behavior Matching

The first level of matching is to find behavior expressions corresponding to the algebraic pattern of a vulnerability.

Behavior matching anticipates solving behavior expressions.

The task of solving behavior equations is formulated as follows. Let *B0* be the system of behavior equations:
B0 = R(a1, a2, … , B1, B2, …,),
B1 = R(a11, a12, …, B11, B12, …), …
where *B1, B2, … B11, B12, ...* are the behaviors and *a1, a2, …* are actions. The behavior *B0* shows translated binary code.

Let behavior *X* be an unknown behavior containing a vulnerability. The task is to find *X*, that is *B0 = Y; X; Z* and *X = vulnerabilityBufferOverflow*

Model Matching

Model matching is performed by the symbolic modelling of a given behavior that was obtained by behavior matching.

During symbolic modelling, we apply the actions for which we detect the satisfiability of the expression *Env && Prec*, where *Env* is a symbolic environment of the model of the binary code and *Prec* is the precondition of the matched action.

If it is satisfiable, then we perform the postcondition operations in the pattern environment and in the environment of the binary code model.

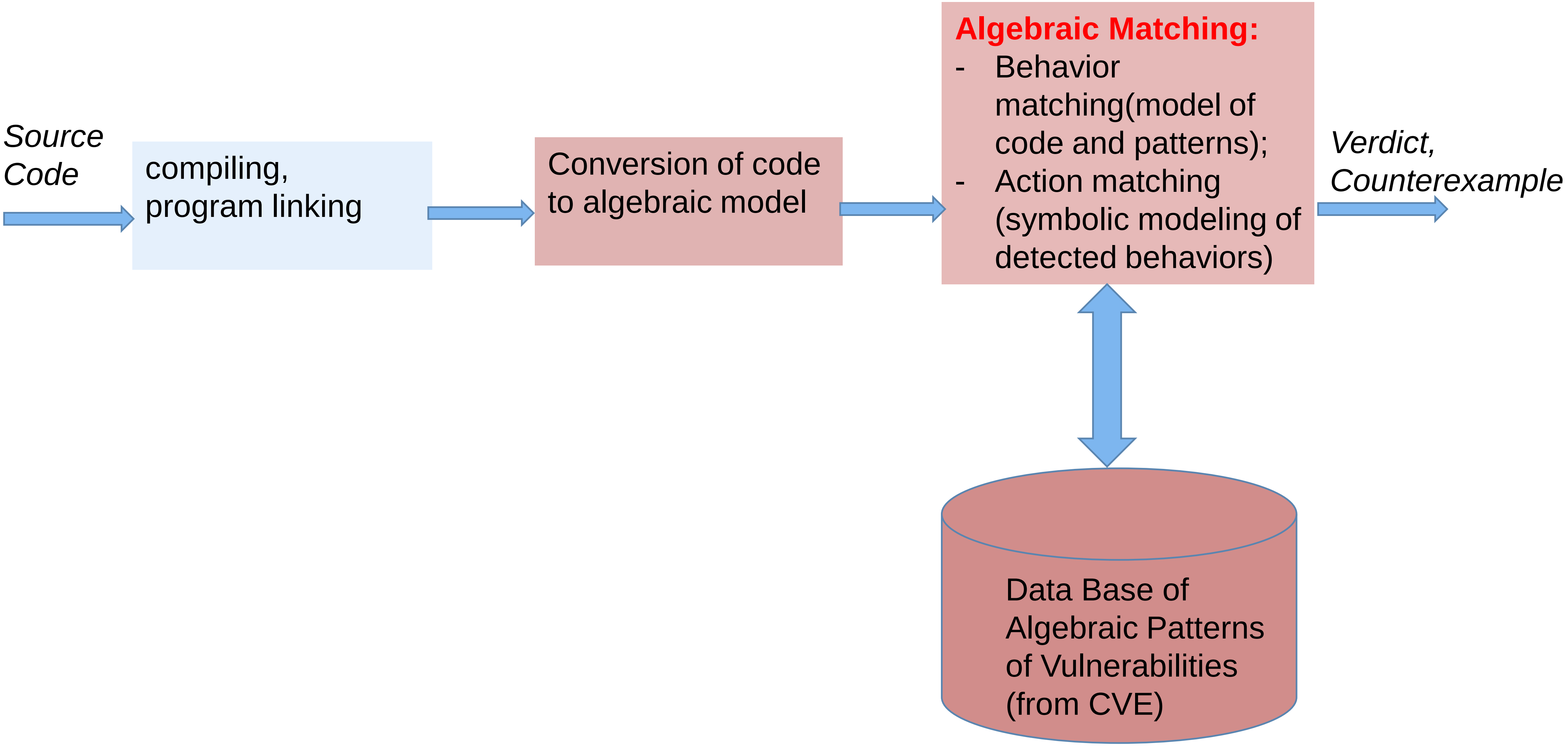
If we reach the vulnerability point in the pattern, then we have a scenario that leads from the input point.

Exploit Generation

With the symbolic environment represented by the set of formulae, it is possible to realize a concrete scenario or to define input values that cause stack buffer overflow.

Backward symbolic modeling to the input point gives the initial formula that covers the values enabling exploits to perform malicious actions.

Detection of Vulnerabilities in Binary Code



Algebraic Model of Intrusion. Toy Example

Behavior Expressions:

B_Meltdown = a_i0.X;B_i0,
B_i0 = a_i1.X;B_i1,
B_i1 = a_i2.X;B_i2,
B_i2 = a_i3.B_i0 + ~a_i3

Local Predicates:

x = Operand(a_i0) & memAuthorized(x) & SpecComp(a_i0)
type(a_i1) = rdtscp
y = Operand(a_i2) & type(a_i2) = movzx & (y = x)
type(a_i3) = rdtscp

Global Predicates:

memAuthorized(x) <=> (x > KERNEL_MEMORY_START)
& (x < KERNEL_MEMORY_END)

Algebraic Model of Intrusion. Toy Example

Behavior Expressions:

B_Meltdown = a_i0.X1;B_i0,
B_i0 = a_i1.X2;B_i1,
B_i1 = a_i2.X3;B_i2,
B_i2 = a_i3.B_i3
B_i3 = X4.B_i0 + Y

Local Predicates:

x = Operand(a_i0) & memAuthorized(x) & SpecComp(a_i0)
type(a_i1) = rdtscp
y = Operand(a_i2) & type(a_i2) = movzx & (y = x)
type(a_i3) = rdtscp

Global Predicates:

memAuthorized(x) <=> (x > KERNEL_MEMORY_START)
& (x < KERNEL_MEMORY_END)

Associated with vocabilary:

“Someone tries to perform non-authorized access to kernel memory”

“Someone measures a time in a cycle with non-authorized access to kernel memory”

Algebraic Model of Intruder's Abstract Behavior

- Example of **Spectre** exploit behavior:

```
    /* Bit twiddling to set x=training_x if j%6!=0 or malicious_x if j%6==0 */
    /* Avoid jumps in case those tip off the branch predictor */
    x = ((j % 6) - 1) & ~0xFFFF; /* Set x=FFF.FF0000 if j%6==0, else x=0 */
    x = (x | (x >> 16)); /* Set x=-1 if j&6=0, else x=0 */
    x = training_x ^ (x & (malicious_x ^ training_x));

    /* Call the victim! */
    victim_function(x);
}

/* Time reads. Order is lightly mixed up to prevent stride prediction */
for (i = 0; i < 256; i++) {
    mix_i = ((i * 167) + 13) & 255;
    addr = &array2[mix_i * 512];
    time1 = __rdtscp(&junk); /* READ TIMER */
    junk = *addr; /* MEMORY ACCESS TO TIME */
    time2 = __rdtscp(&junk) - time1; /* READ TIMER & COMPUTE ELAPSED TIME */
    if (time2 <= CACHE_HIT_THRESHOLD && mix_i != array1[tries % array1_size])
        results[mix_i]++; /* cache hit - add +1 to score for this value */
}
```

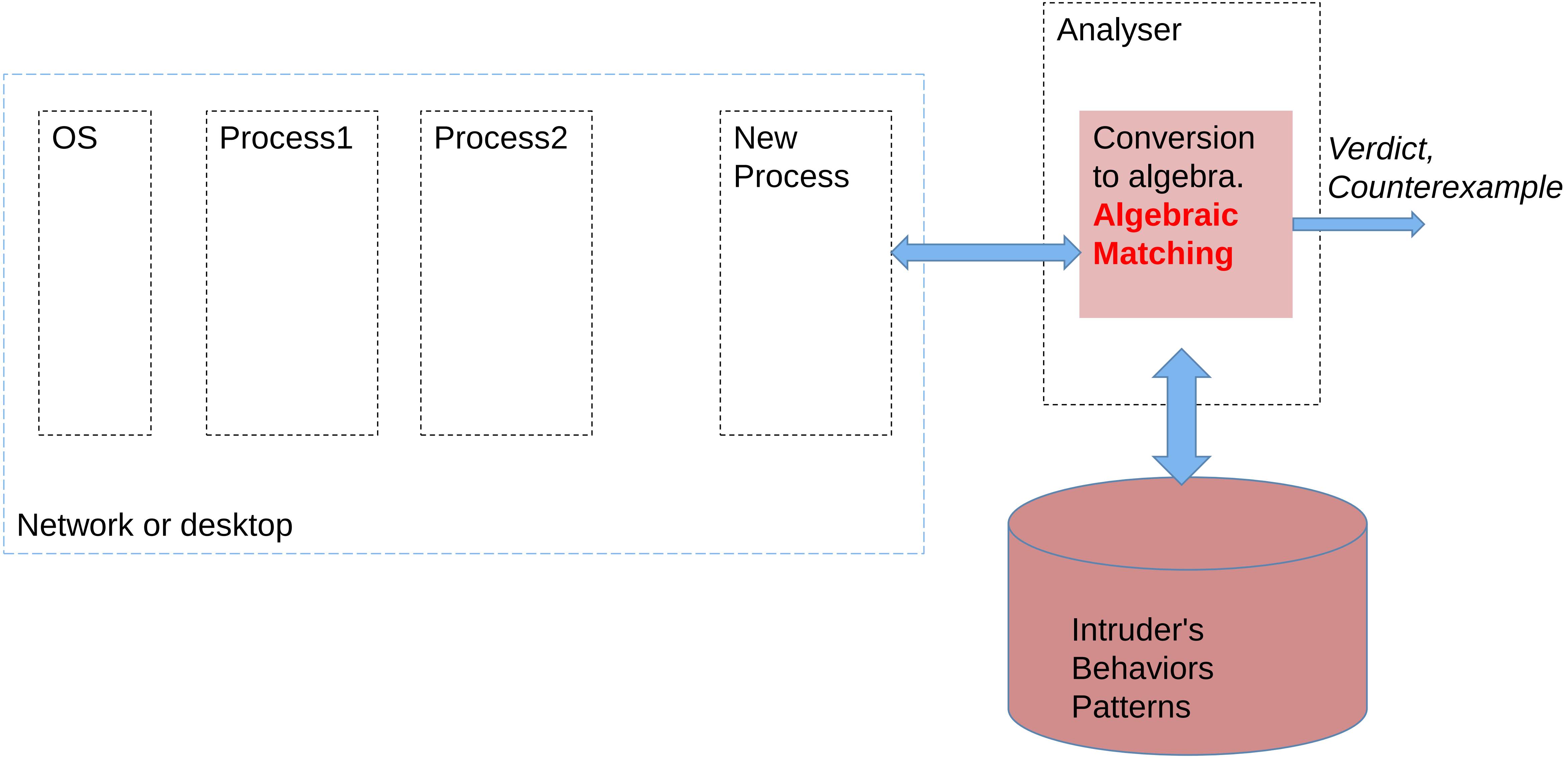
```
400801:  48 8d 45 94           lea    rax,[rbp-0x6c]
400805:  48 89 45 d8           mov    QWORD PTR [rbp-0x28],rax
400809:  0f 01 f9              rdtscp
40080c:  89 ce                 mov    esi,ecx
40080e:  48 8b 4d d8           mov    rcx,QWORD PTR [rbp-0x28]
400812:  89 31                 mov    DWORD PTR [rcx],esi
400814:  48 c1 e2 20           shl    rdx,0x20
400818:  48 09 d0              or     rax,rdx
40081b:  48 89 c3              mov    rbx,rax
40081e:  48 8b 45 d0           mov    rax,QWORD PTR [rbp-0x30]
400822:  0f b6 00              movzx  eax,BYTE PTR [rax]
400825:  0f b6 c0              movzx  eax,al
400828:  89 45 94              mov    DWORD PTR [rbp-0x6c],eax
40082b:  48 8d 45 94           lea    rax,[rbp-0x6c]
40082f:  48 89 45 c0           mov    QWORD PTR [rbp-0x40],rax
400833:  0f 01 f9              rdtscp
400836:  89 ce                 mov    esi,ecx
400838:  48 8b 4d c0           mov    rcx,QWORD PTR [rbp-0x40]
40083c:  89 31                 mov    DWORD PTR [rcx],esi
40083e:  48 c1 e2 20           shl    rdx,0x20
400842:  48 09 d0              or     rax,rdx
400845:  48 29 d8              sub    rax,rbx
400848:  48 89 c3              mov    rbx,rax
40084b:  48 83 fb 50           cmp    rbx,0x50
40084f:  77 3e                 ja     40088f <readMemoryByte+0x205>
400851:  8b 45 9c              mov    eax,DWORD PTR [rbp-0x64]
400854:  8b 0d 06 18 20 00     mov    ecx,DWORD PTR [rip+0x201806]
```

B = X1;a1(rdtscp).a2(mov,t1 = esi,ecx).X2;B1,
B1 = a3(mov,i = rax,A).X3;B2,
B2 = X4;a4(rdtscp).a5(mov,t2 = esi,ecx).X5;B3,
B3 = a6(t2 - t1 < CASH_THRESHOLD).B + a7(t2 - t1 >=CASH_THRESHOLD).B4,
B4 =

abstract behavior of intruder

AuthorizedMem(A) = true

Analysis of Suspicious Process



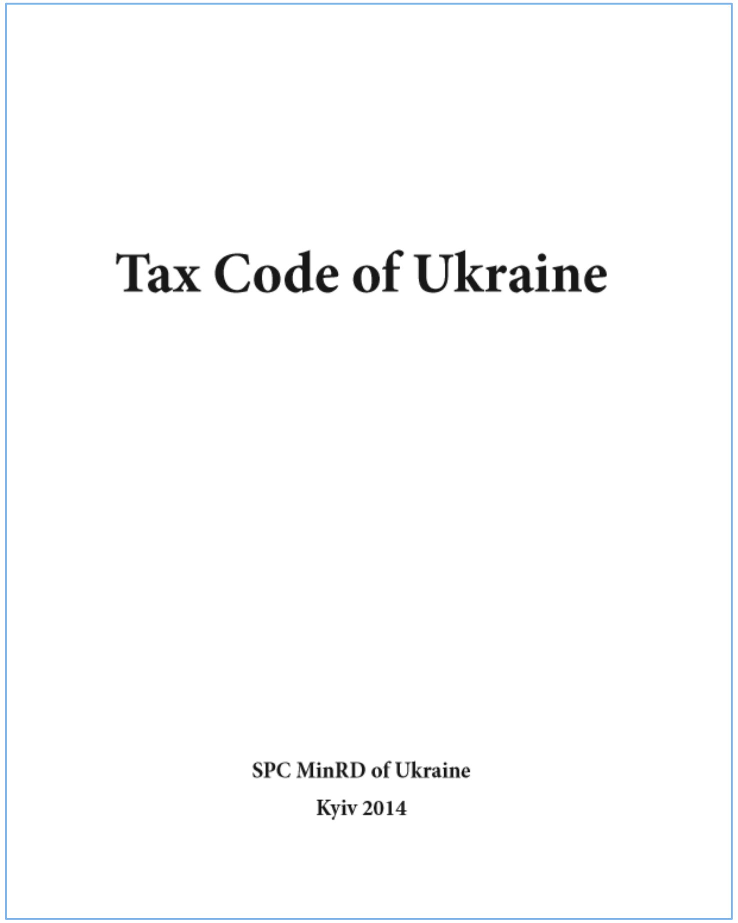
Two Main Problems

- Problem of reachability of property
- Problem of matching

Digital Justice

Checking for consistency of Tax Payer behavior with Tax Code

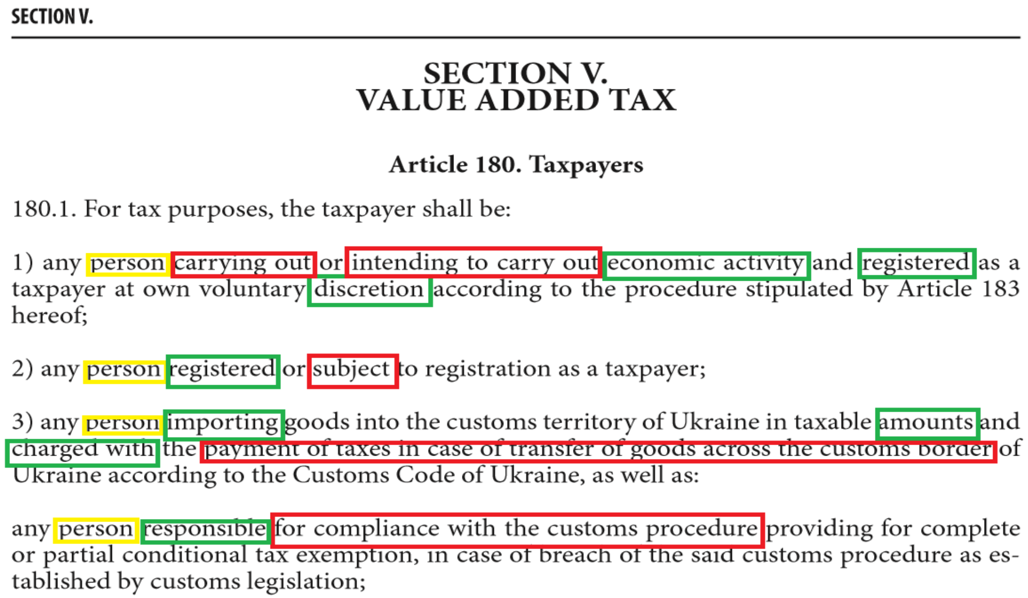
- Tax Code consists of 357 articles written on 780 pages. The articles are structured in 20 sections, with the major ones focusing on VAT and profit taxes.
- For formalization of Tax Code we used:
  - formalization technique of VRS;
  - theory of interaction of agents and environments;
  - Use Case Maps diagrams;
  - Basic protocols specifications and behavior algebra


Tax Code of Ukraine

SPC MinRD of Ukraine

Kyiv 2014

Formalization of Non-behavioral Requirements



SECTION V.

SECTION V.
VALUE ADDED TAX

Article 180. Taxpayers

180.1. For tax purposes, the taxpayer shall be:

1) any person carrying out or intending to carry out economic activity and registered as a taxpayer at own voluntary discretion according to the procedure stipulated by Article 183 hereof;

2) any person registered or subject to registration as a taxpayer;

3) any person importing goods into the customs territory of Ukraine in taxable amounts and charged with the payment of taxes in case of transfer of goods across the customs border of Ukraine according to the Customs Code of Ukraine, as well as:

any person responsible for compliance with the customs procedure providing for complete or partial conditional tax exemption, in case of breach of the said customs procedure as established by customs legislation;

Let *X is* agent of type *Person*
The statements about agent's attributes are AXIOMS in APS system
For presentation of axioms we use the logical language of APS that contains:
- first order logic operations (including quantifiers);
- enumerated types;
- boolean types;
- symbolic types;
- linear arithmetics;
-functional symbols (free terms).

X.taxPayer <=> A180_1_1 ∨ A180_1_2 ∨ A180_1_3 ...
A180_1_1 <=> ((X.commercialActivity.status = INTENT) ∨ (X.commercialActivity.status = IN_PROCESS)) ∧
(X.registration.status = DID) ∧ X.registration.discretionVoluntary
A180_1_2 <=> (X.registration.status = INTENT) ∨ (X.registration.status = DID)
Exist (i:int)
A180_1_3 <=> (x.importing(i) >= TAX_AMOUNT) & (i > 0) & (i <= salesNumber) & (x.importActivity.responsibility = TAX_PAYMENT)

Axioms from Article 180

A180_1 : (A180_1 <=> (x.taxVATPayer <=> (A180_1_1 |/ A180_1_2 |/ A180_1_3 & A180_1_3_3 |/ (A180_1_3 & (A180_1_3_1 |/ A180_1_3_2) & A180_1_3_3) |/ A180_1_4 |/ A180_1_5))),

A180_1_1 : (A180_1_1 <=>   (((x.commercialActivity.status = GONNA_BE) |/ (x.commercialActivity.status = DOES)) & (x.registrationVAT.status = DOES) & x.registrationVAT.decisionVoluntaryVAT)),

A180_1_2 : (A180_1_2 <=> (x.registrationVAT.necessity |/ (x.registrationVAT.status = DID))),

A180_1_3 : (Exist (i:int) ((A180_1_3 <=> (x.importActivity.status = DOES) & ((x.import.sales)(i)>= limitVATimport) & (i > 0) & (i <= salesNumber) & (x.importActivity.responsibility = TAX_PAYMENT))),

A180_1_3_1 : (A180_1_3_1 <=> (x.importActivity.responsibility = KEEP_TAXFREE_REQUIREMENTS) & x.importActivity.taxFreeModeViolation),

A180_1_3_2 : (A180_1_3_2 <=>x.importActivity.taxExtemption.given & ((x.importActivity.taxExtemption.usage = NOT_TARGETED) |/ (x.importActivity.taxExtemption.usage = CONDITIONS_VIOLATION))),

A180_1_3_3 : (A180_1_3_3 <=> ~(x.importActivity.taxSalesCultureValues)),

A180_1_4 : (A180_1_4 <=> (x.otherActivity = JOINT_ACTIVITY_CONTRACT)),

A180_1_5 : (A180_1_5 <=> (x.otherActivity = CONTROL_ESTATE)),

A180_1_6 : (A180_1_6 <=> (x.otherActivity = CONFISCATE))

Formalization of Behavioral Requirements

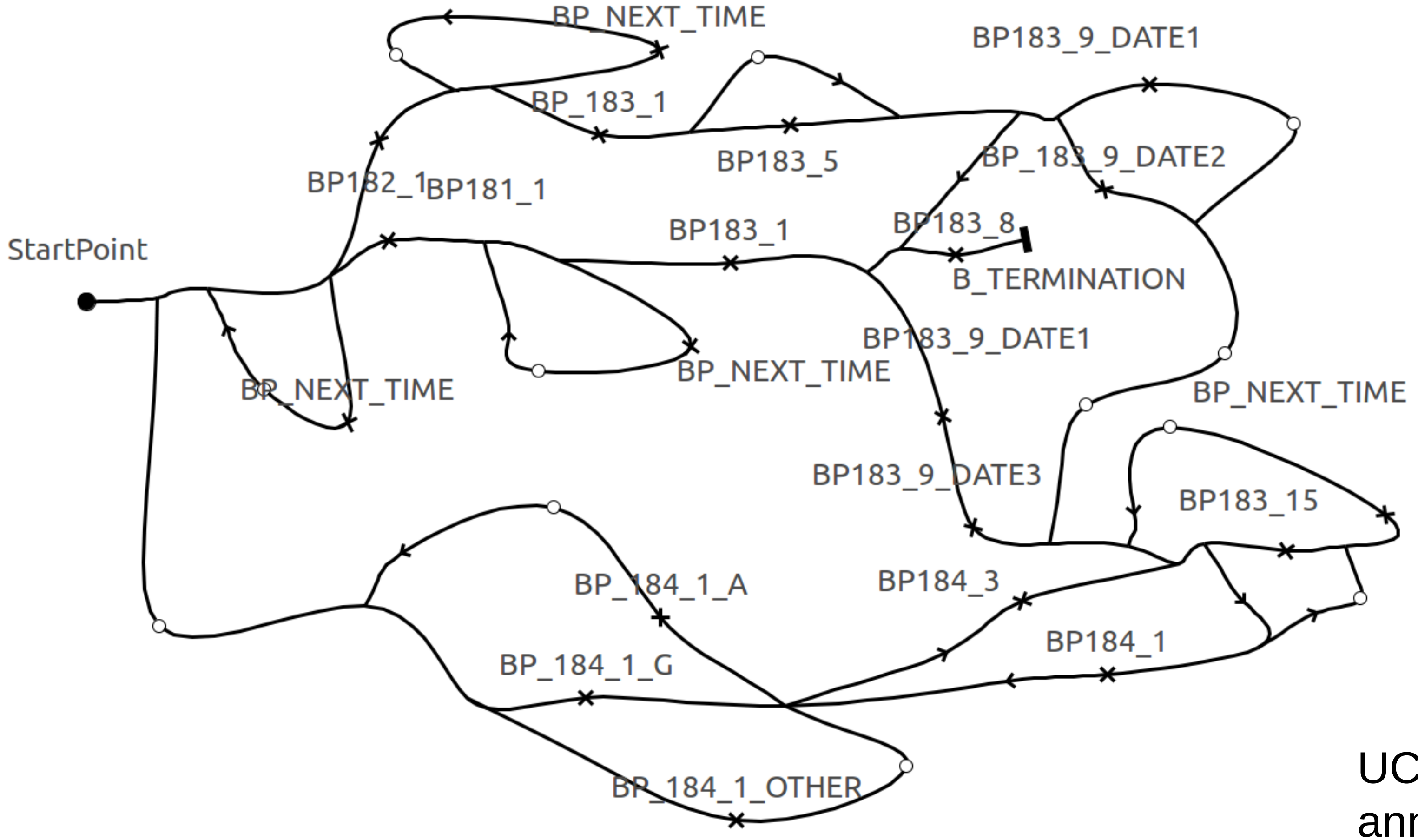
Article 184. Taxpayer registration annulment

184.1. Taxpayer registration is valid until annulment thereof by way of removal from the Register of Taxpayers and shall take place in cases as follows:

d) a person registered as a taxpayer has not filed their VAT declarations at the regulatory authority for 12 consecutive tax months and/or has filed a tax declaration (tax calculation) indicative of no supplies/purchase of goods in view of forming a tax liability or a tax credit;

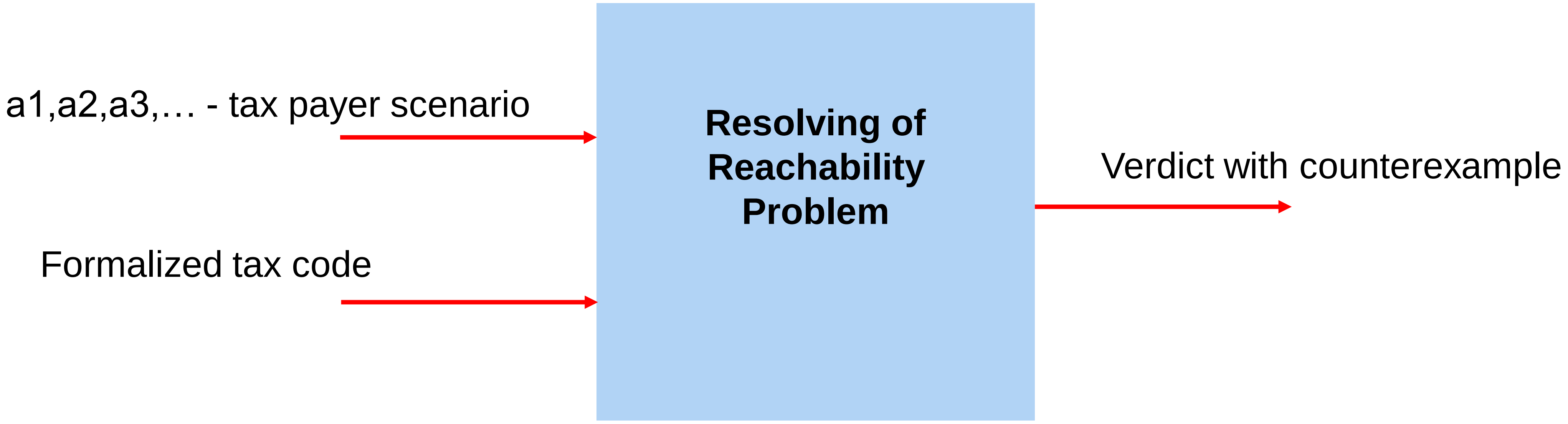
```
Forall i ((x.VAT(timeActual.month - i)=0) & (i >= 0) & (i < 12) & (timeActual.month
> 12))->
("184.1.g  VAT registrarion cancelation")
(x.registrationVAT.status := CANCELED)
```

Ordering of Behavioral Requirements (UCM diagrams)



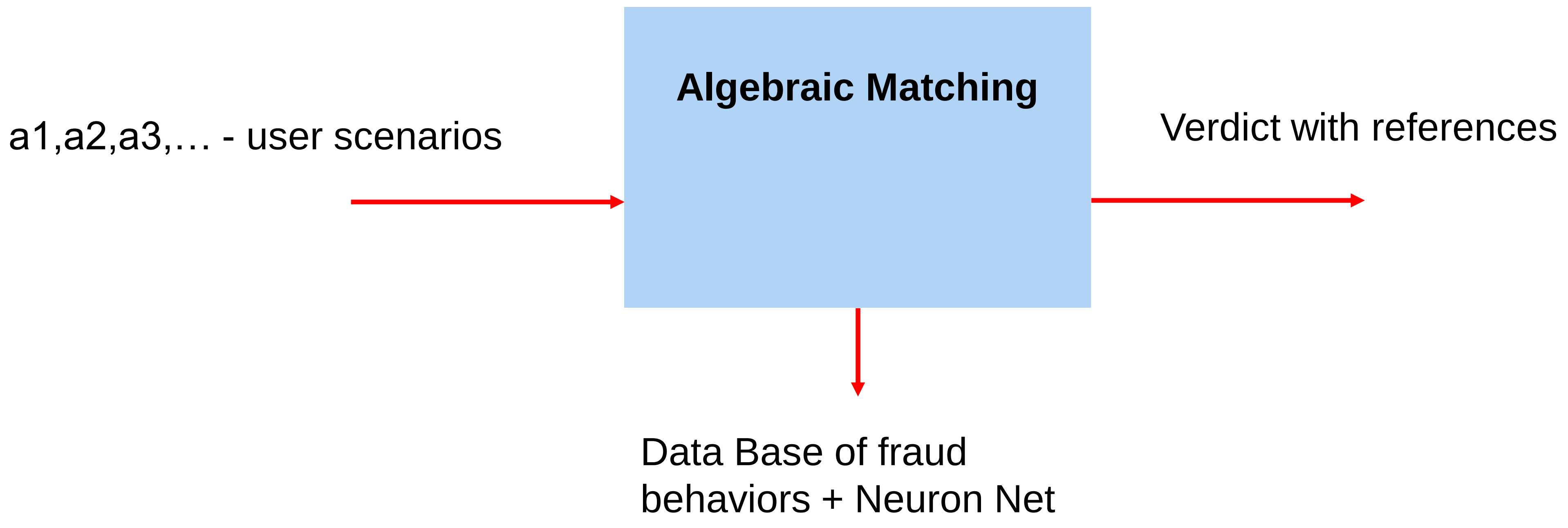
UCM with VAT payer registration and annulment procedure from Tax Code (sequential model)

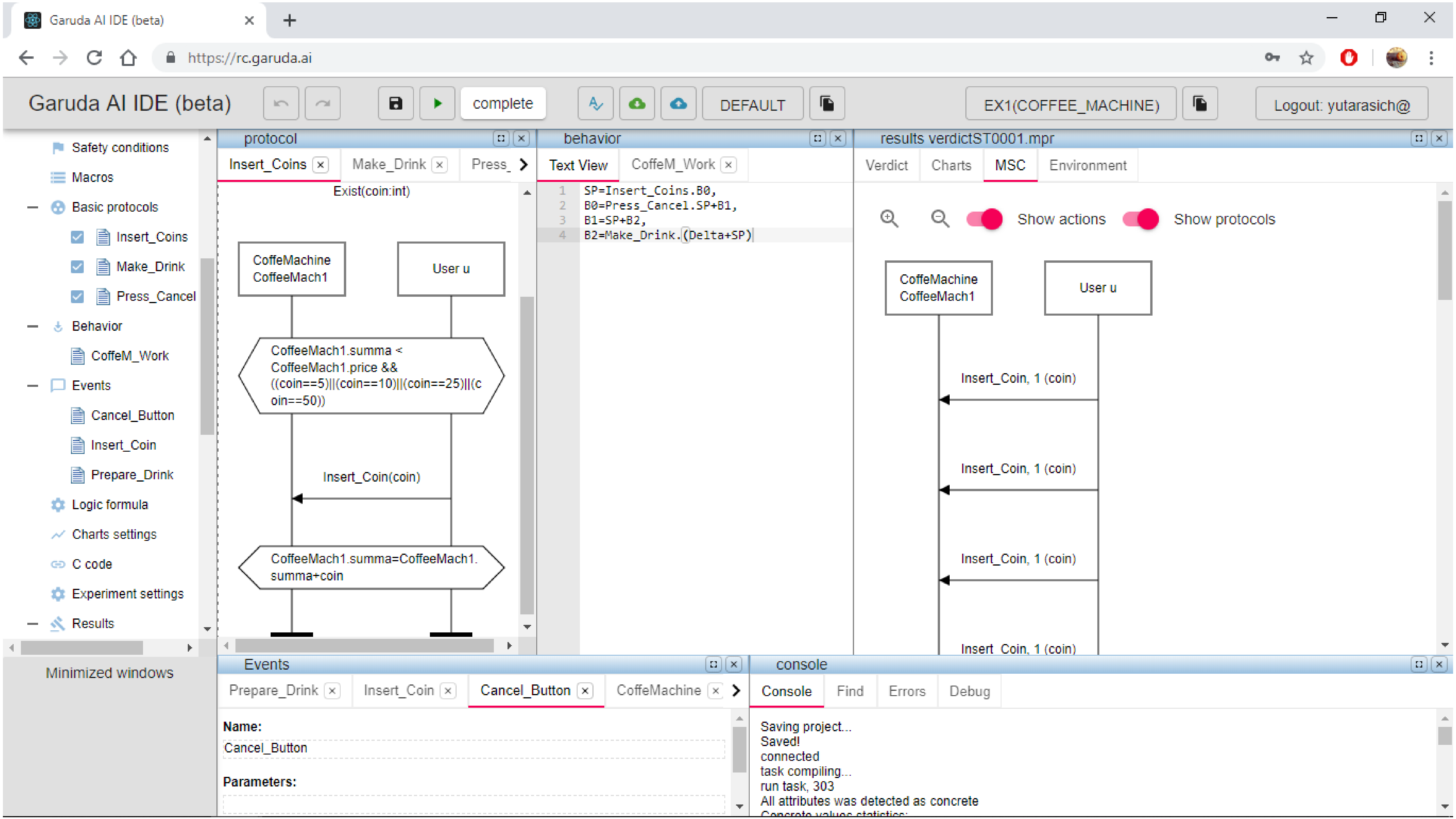
Checking of Behavior of Tax Payer for Consistency

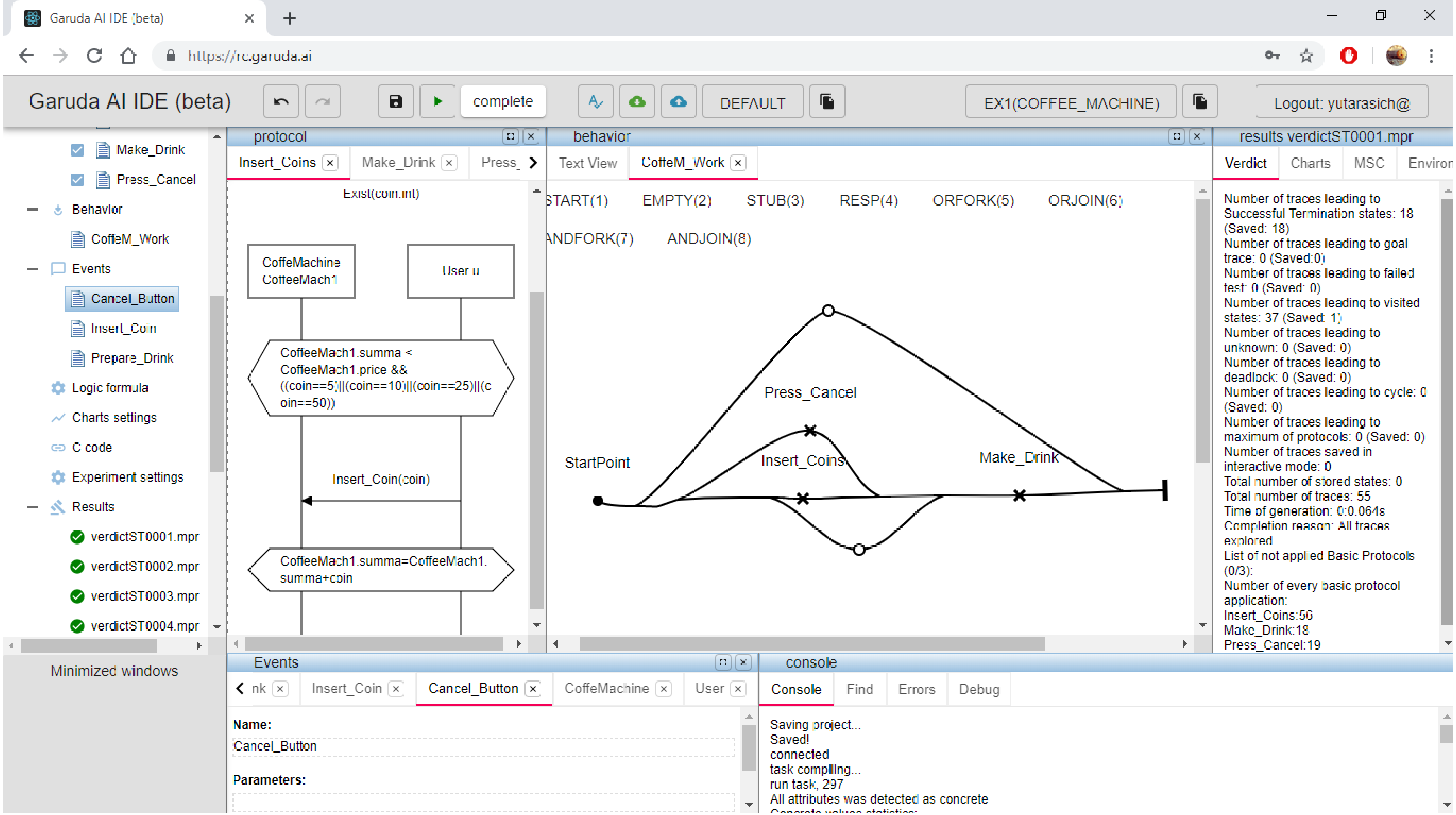


Forensic Computation

- Installation Fraud
  - Fraud behavior is described as algebra behavior expressions;
  - We observe the set of scenarios of user behavior.




Garuda AI IDE (beta)
https://rc.garuda.ai
Garuda AI IDE (beta)
complete
DEFAULT
EX1(COFFEE_MACHINE)
Logout: yutarasich@
Safety conditions
Macros
Basic protocols
Insert_Coins
Make_Drink
Press_Cancel
Behavior
CoffeM_Work
Events
Cancel_Button
Insert_Coin
Prepare_Drink
Logic formula
Charts settings
C code
Experiment settings
Results
Minimized windows
protocol
Insert_Coins
Make_Drink
Press_
Exist(coin:int)
CoffeMachine CoffeeMach1
User u
CoffeeMach1.summa < CoffeeMach1.price && ((coin==5)||(coin==10)||(coin==25)||(coin==50))
Insert_Coin(coin)
CoffeeMach1.summa=CoffeeMach1.summa+coin
behavior
Text View
CoffeM_Work
SP=Insert_Coins.B0,
B0=Press_Cancel.SP+B1,
B1=SP+B2,
B2=Make_Drink.(Delta+SP)
results verdictST0001.mpr
Verdict
Charts
MSC
Environment
Show actions
Show protocols
CoffeMachine CoffeeMach1
User u
Insert_Coin, 1 (coin)
Insert_Coin, 1 (coin)
Insert_Coin, 1 (coin)
Insert_Coin, 1 (coin)
Events
Prepare_Drink
Insert_Coin
Cancel_Button
CoffeMachine
Name:
Cancel_Button
Parameters:
console
Console
Find
Errors
Debug
Saving project...
Saved!
connected
task compiling...
run task, 303
All attributes was detected as concrete


Garuda AI IDE (beta)
https://rc.garuda.ai
Garuda AI IDE (beta)
complete
DEFAULT
EX1(COFFEE_MACHINE)
Logout: yutarasich@
Make_Drink
Press_Cancel
Behavior
CoffeM_Work
Events
Cancel_Button
Insert_Coin
Prepare_Drink
Logic formula
Charts settings
C code
Experiment settings
Results
verdictST0001.mpr
verdictST0002.mpr
verdictST0003.mpr
verdictST0004.mpr
Minimized windows
protocol
Insert_Coins
Make_Drink
Press_
Exist(coin:int)
CoffeMachine CoffeeMach1
User u
CoffeeMach1.summa < CoffeeMach1.price && ((coin==5)||(coin==10)||(coin==25)||(coin==50))
Insert_Coin(coin)
CoffeeMach1.summa=CoffeeMach1.summa+coin
behavior
Text View
CoffeM_Work
START(1)
EMPTY(2)
STUB(3)
RESP(4)
ORFORK(5)
ORJOIN(6)
ANDFORK(7)
ANDJOIN(8)
Press_Cancel
StartPoint
Insert_Coins
Make_Drink
results verdictST0001.mpr
Verdict
Charts
MSC
Environ
Number of traces leading to Successful Termination states: 18 (Saved: 18)
Number of traces leading to goal trace: 0 (Saved:0)
Number of traces leading to failed test: 0 (Saved: 0)
Number of traces leading to visited states: 37 (Saved: 1)
Number of traces leading to unknown: 0 (Saved: 0)
Number of traces leading to deadlock: 0 (Saved: 0)
Number of traces leading to cycle: 0 (Saved: 0)
Number of traces leading to maximum of protocols: 0 (Saved: 0)
Number of traces saved in interactive mode: 0
Total number of stored states: 0
Total number of traces: 55
Time of generation: 0:0.064s
Completion reason: All traces explored
List of not applied Basic Protocols (0/3):
Number of every basic protocol application:
Insert_Coins:56
Make_Drink:18
Press_Cancel:19
Events
nk
Insert_Coin
Cancel_Button
CoffeMachine
User
Name:
Cancel_Button
Parameters:
console
Console
Find
Errors
Debug
Saving project...
Saved!
connected
task compiling...
run task, 297
All attributes was detected as concrete

THANK YOU FOR ATTENTION

Glushkov Institute of cybernetics NASU