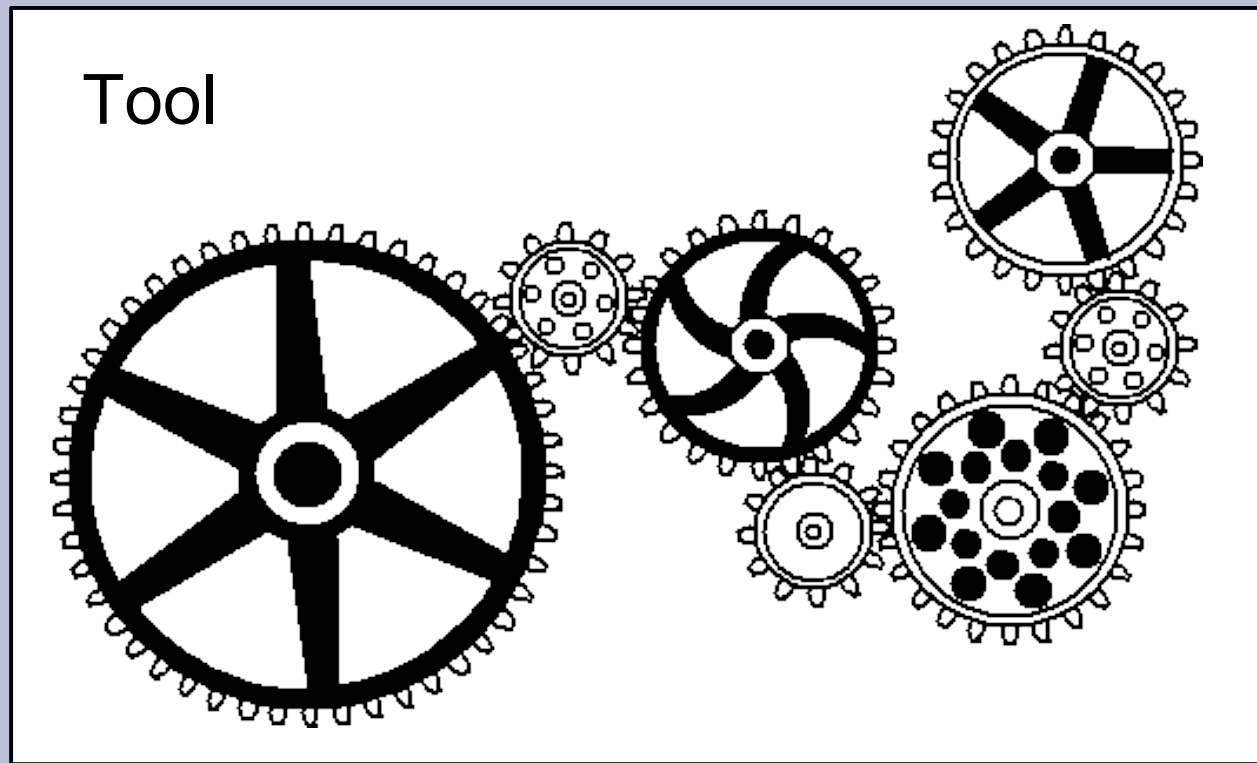
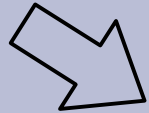


Mechanized analysis of cryptographic protocols using causality based abstraction

protocol



security
properties

Motivation



$$\{B, n_b, m\}_{k_A^+}$$



$$\{n_b, n_a\}_{k_B^+}$$



$$\{n_a\}_{k_A^+}$$



Outline

- Motivation
 - Analysis by hand
- Causality-based Abstraction
 - Graph
 - Traces
- Security Analysis
- Concrete Implementation
 - Problems
- Summary

Is this protocol secure?

Is it secure if run one hundred times in parallel?



$$\{B, n_b, m\}_{k_A^+}$$



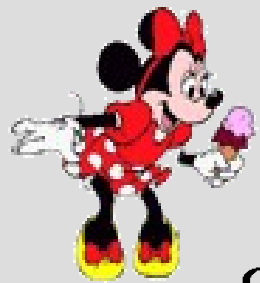
$$\{n_b, n_a\}_{k_B^+}$$



$$\{n_a\}_{k_A^+}$$



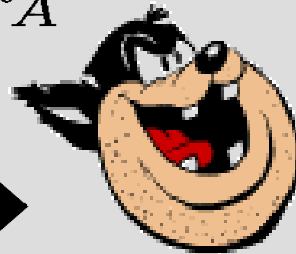
The Attack



$\{B, n_b, m\}_{k_A^+}$



$\{n_b, n_a\}_{k_B^+}$



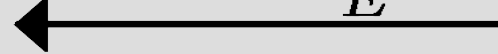
$\{B, n_b, m\}_{k_E^+}$



$\{n_b, n_a\}_{k_B^+}$



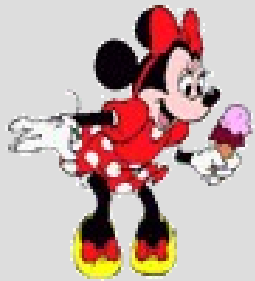
$\{n_a\}_{k_E^+}$



$\{n_a\}_{k_A^+}$



The Fix



$$\{B, n_b, m\}_{k_A^+}$$



$$\{A, n_b, n_a\}_{k_B^+}$$

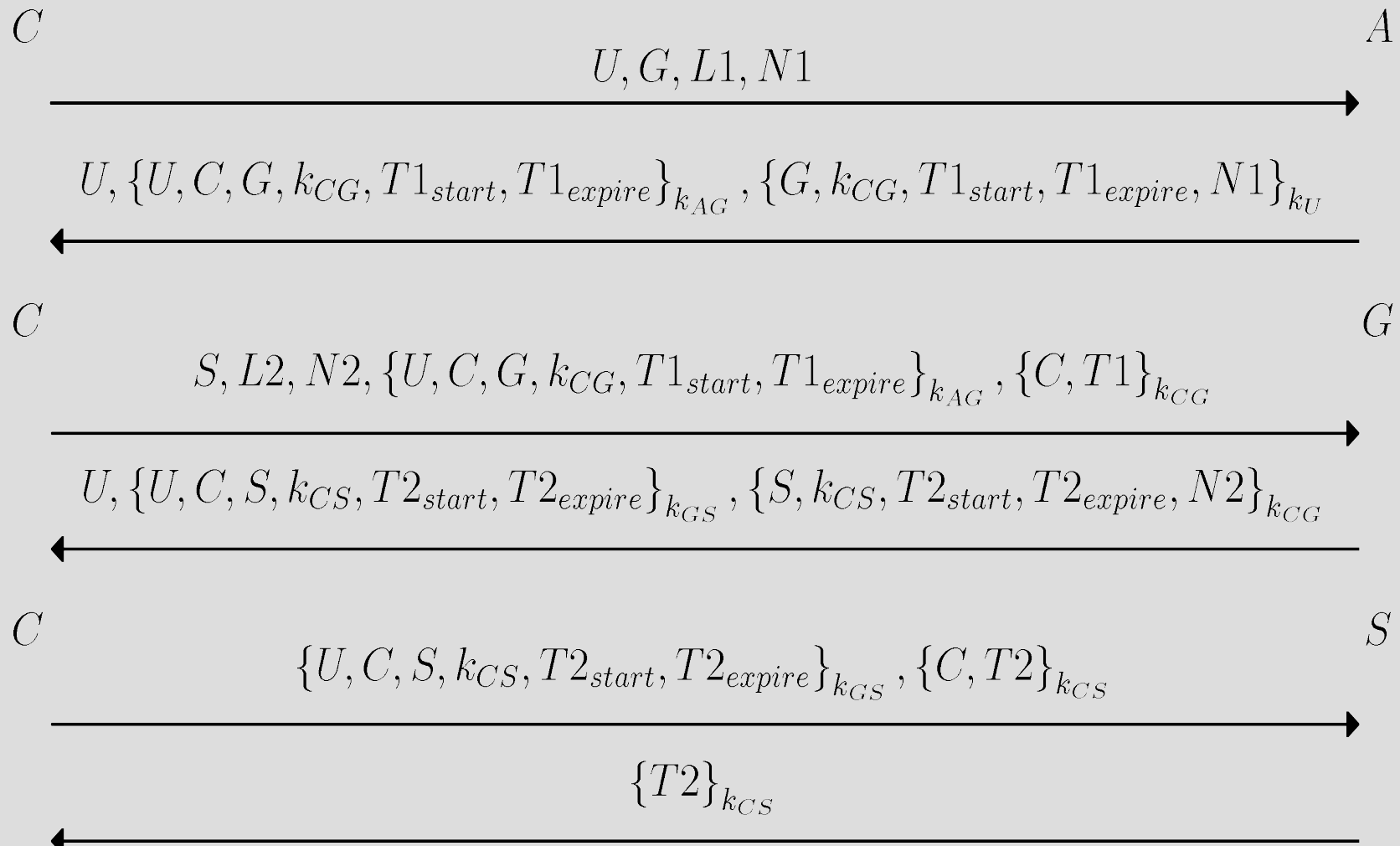


$$\{n_a\}_{k_A^+}$$



Proved secure by formal methods

Kerberos v5



Is this real-life example secure?

Problems with the Analysis

- Analysis by hand highly error prone
- Problem in general not decidable

Two approaches:

- Model Checking
- Static Analysis Techniques

Model Checking

- dynamic
- errors are really errors
- no way of proving security
- typically exponential running time

used e.g. for hardware verification

Static Analysis Techniques

- static...
- errors not necessarily errors
- may show a possible attack
- actual proofs of security
- typically poly runtime

used e.g. to verify java byte code

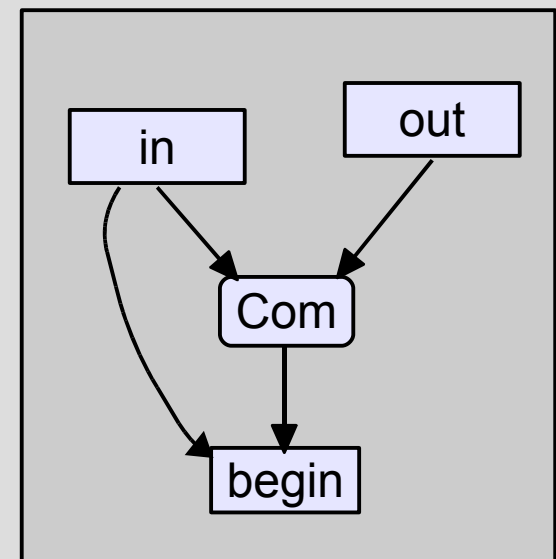
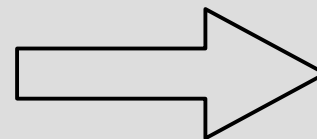
Outline

- Motivation
 - Analysis by hand
- Causality-based Abstraction
 - Graph
 - Traces
- Security Analysis
- Concrete Implementation
 - Problems
- Summary

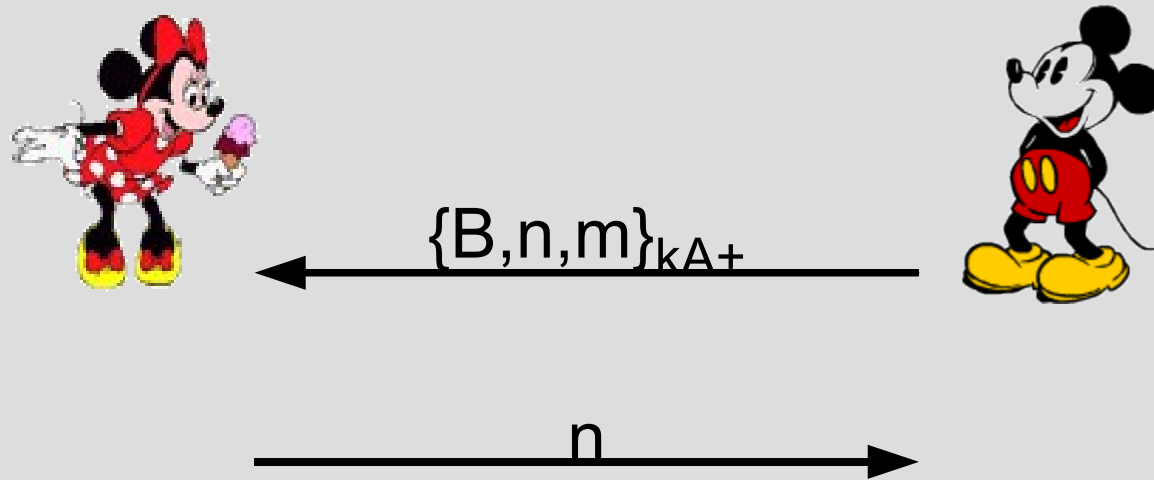
Causality-based Abstraction

- Dolev-Yao Intruder Model
- represent protocol in ρ -spi process calculus
- transform possibly infinitely many protocol sessions into a **finite** graph

$\text{out}(\{n, m\}_{k_A^+}).\text{in}(n).\text{end}(A, \dots \mid$
 $\text{in}(\{x, y\}_{k_A^-}).\text{begin}(B, A, \dots$



ρ - spi Calculus

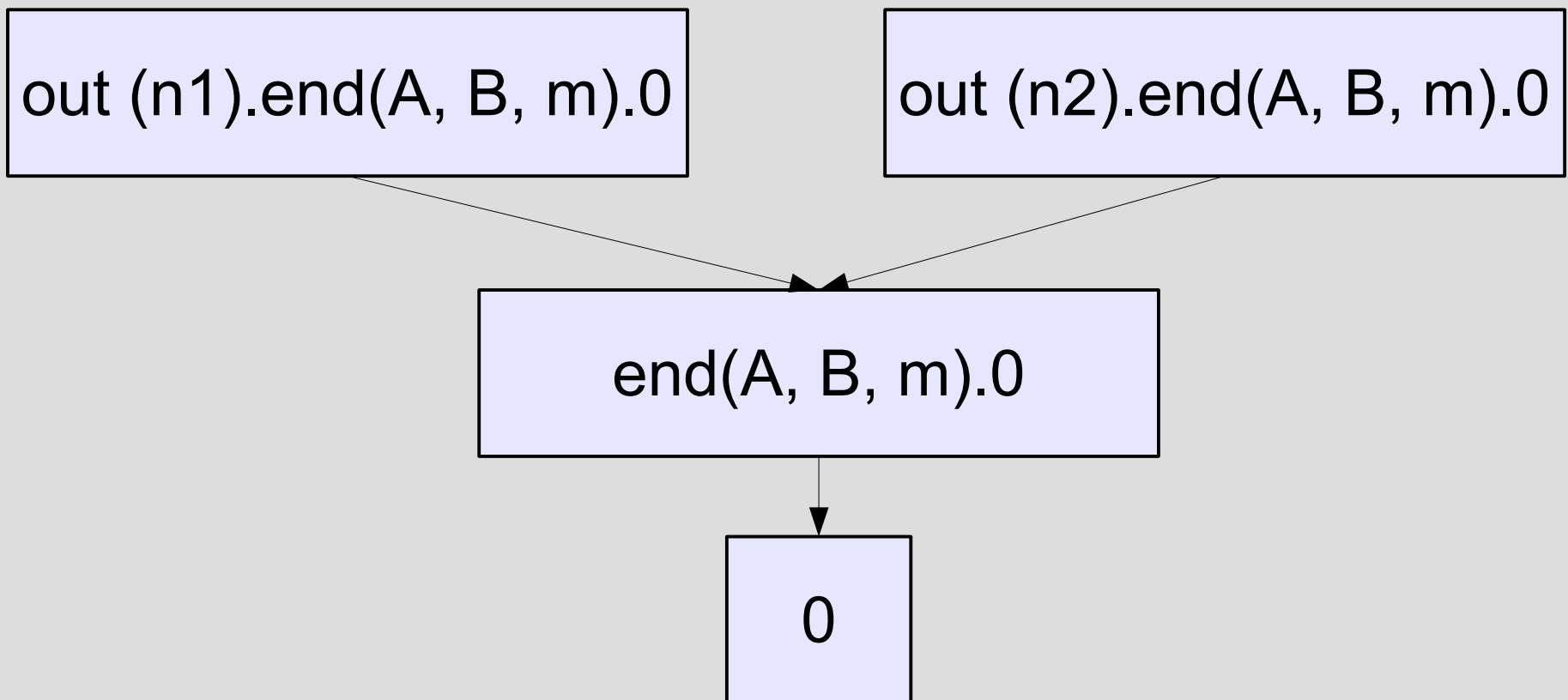


newkey(A).

(new (m).new(n).out($\{B, n, m\}_{kA^+}$).in(n).0 |
in($\{B, x, y\}_{kA^-}$).out(n).0)

Structural Reduction

- straightforward
- "*x is necessarily preceeded by y or z*"



Interprocess Reduction

- rather complicated
- "*x is necessarily preceeded by y and z*"

in (x, y).begin(B, A, x).0

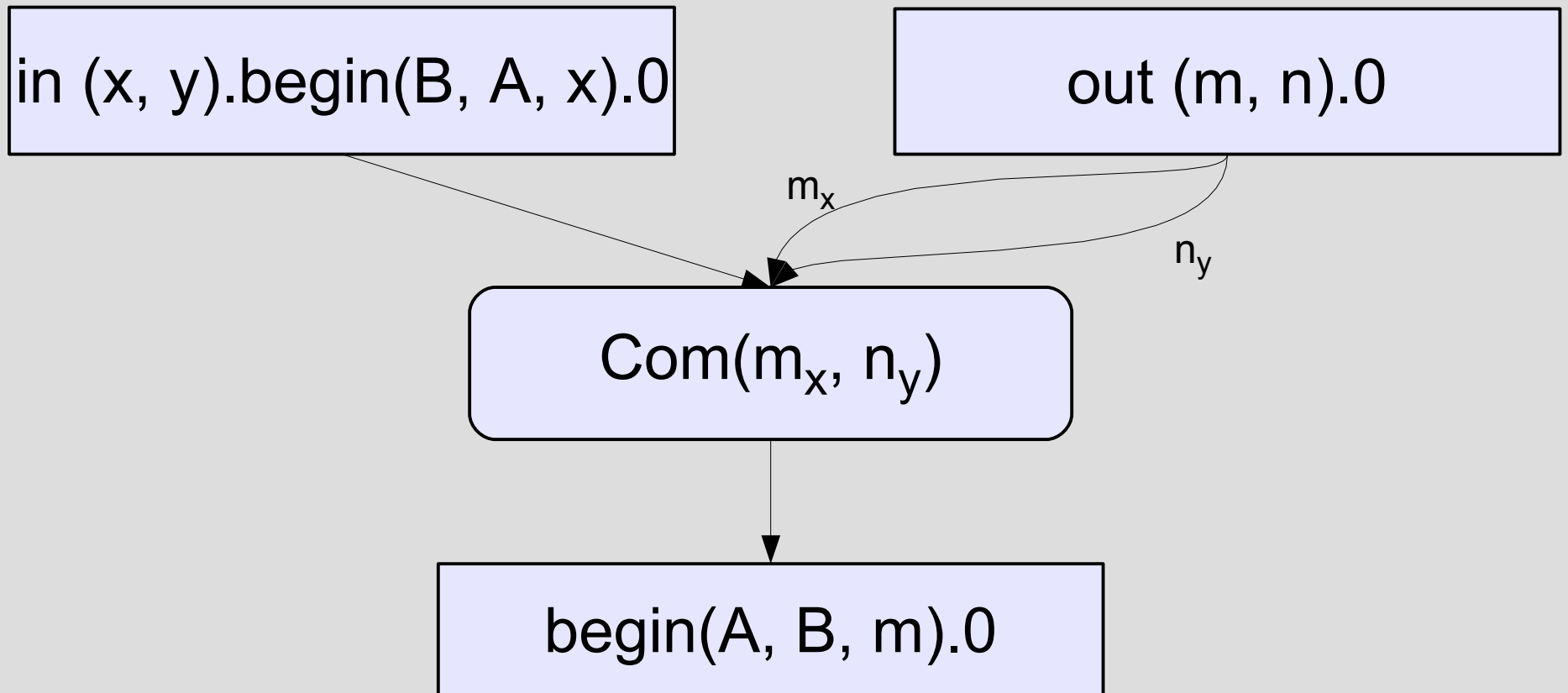
out (m, n).0

m_x

n_y

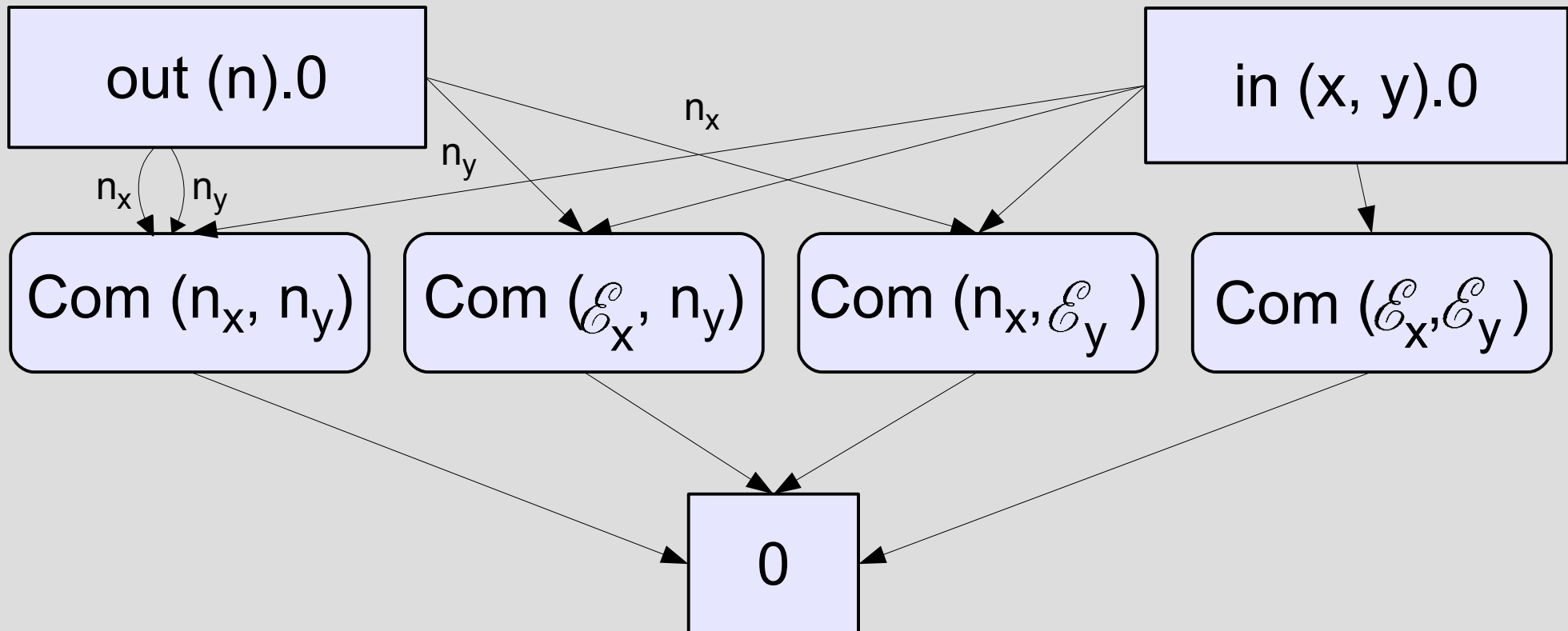
Com(m_x , n_y)

begin(A, B, m).0



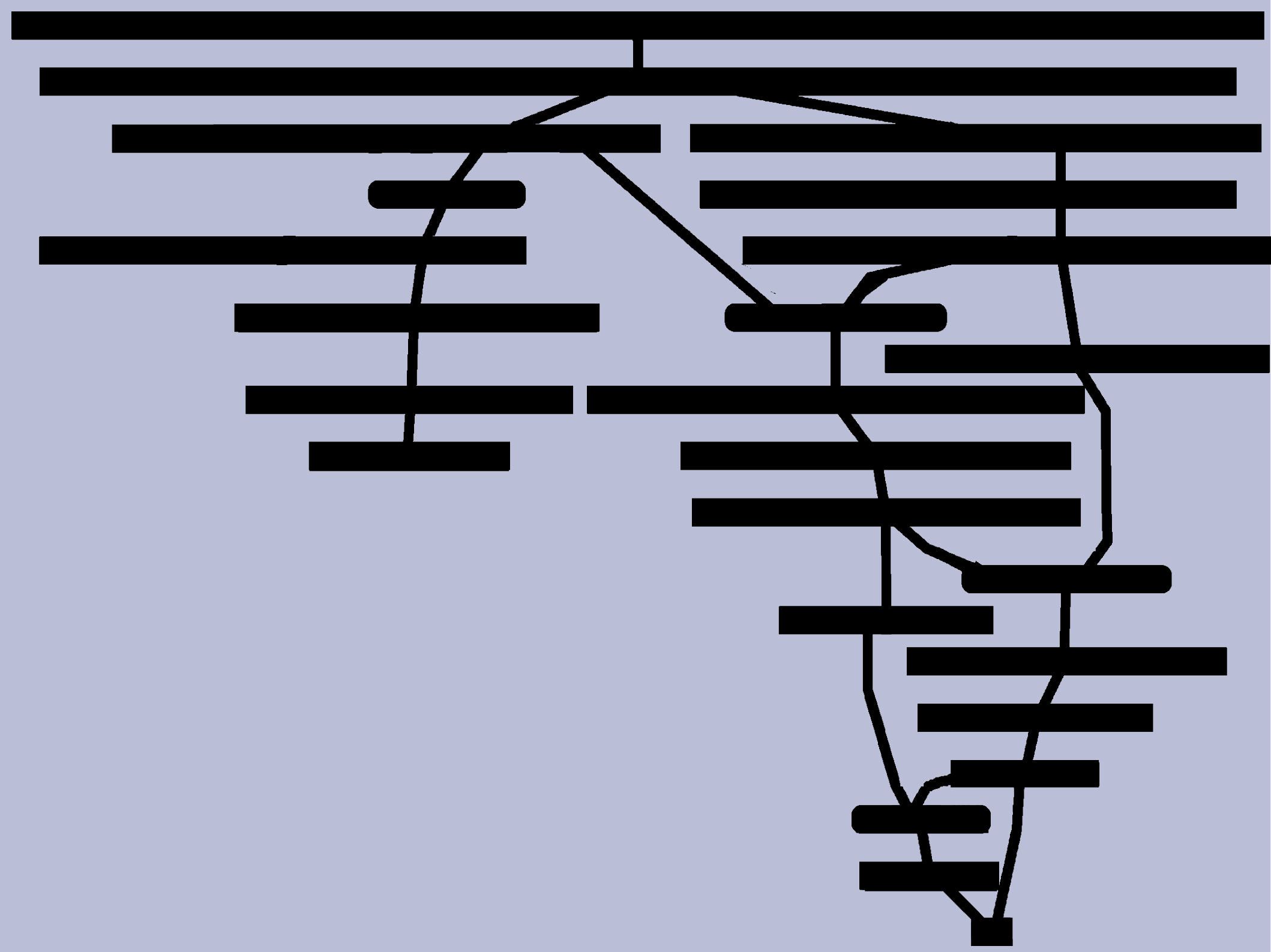
Capabilities of the Attacker \mathcal{E}

- pretty much everything (Dolev Yao)



About the Graph

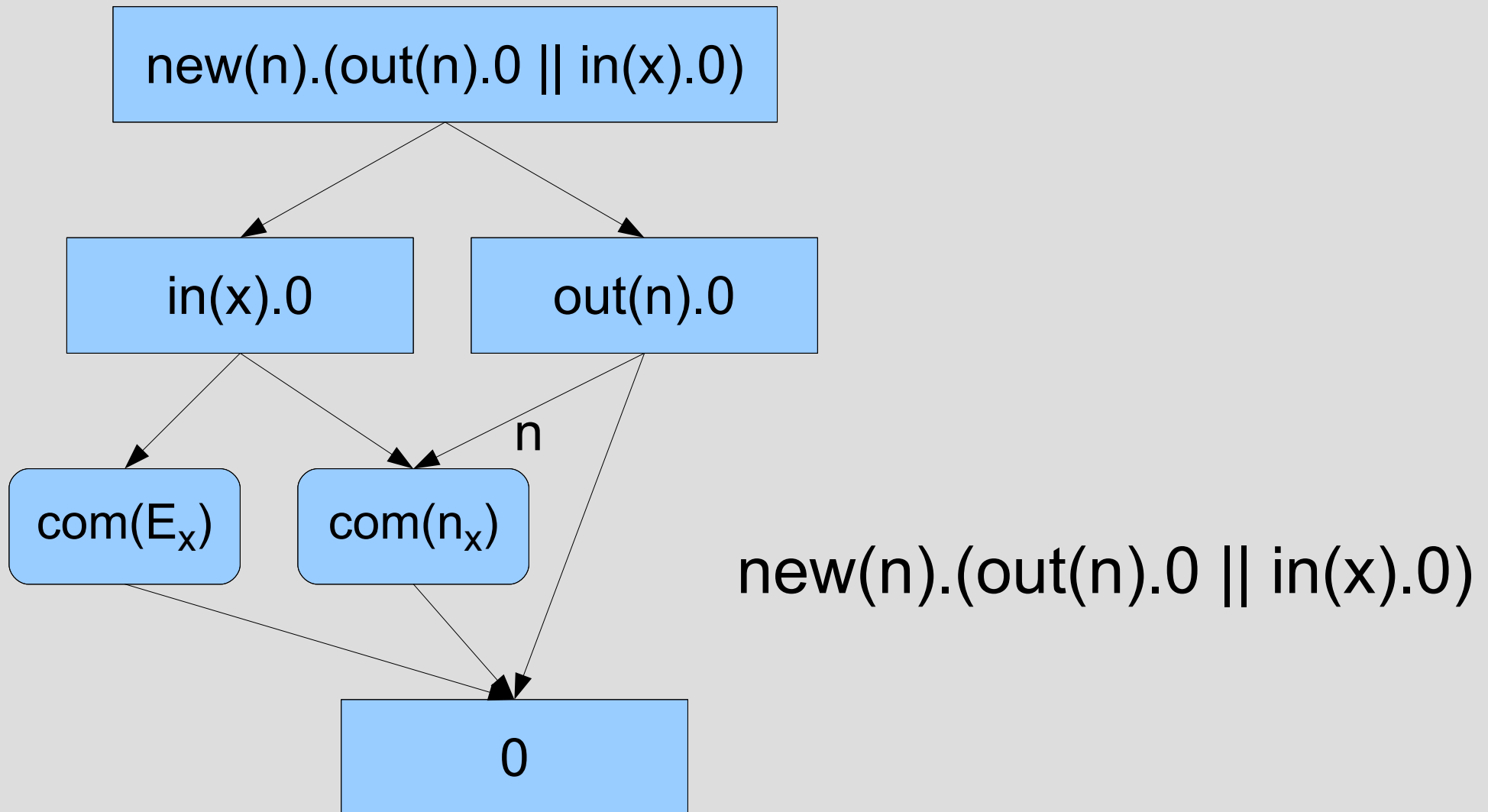
- a protocol has a unique graph
- causal net is sound abstraction
 - characterizes every possible reduction
- worst-case size: N^{x^2}



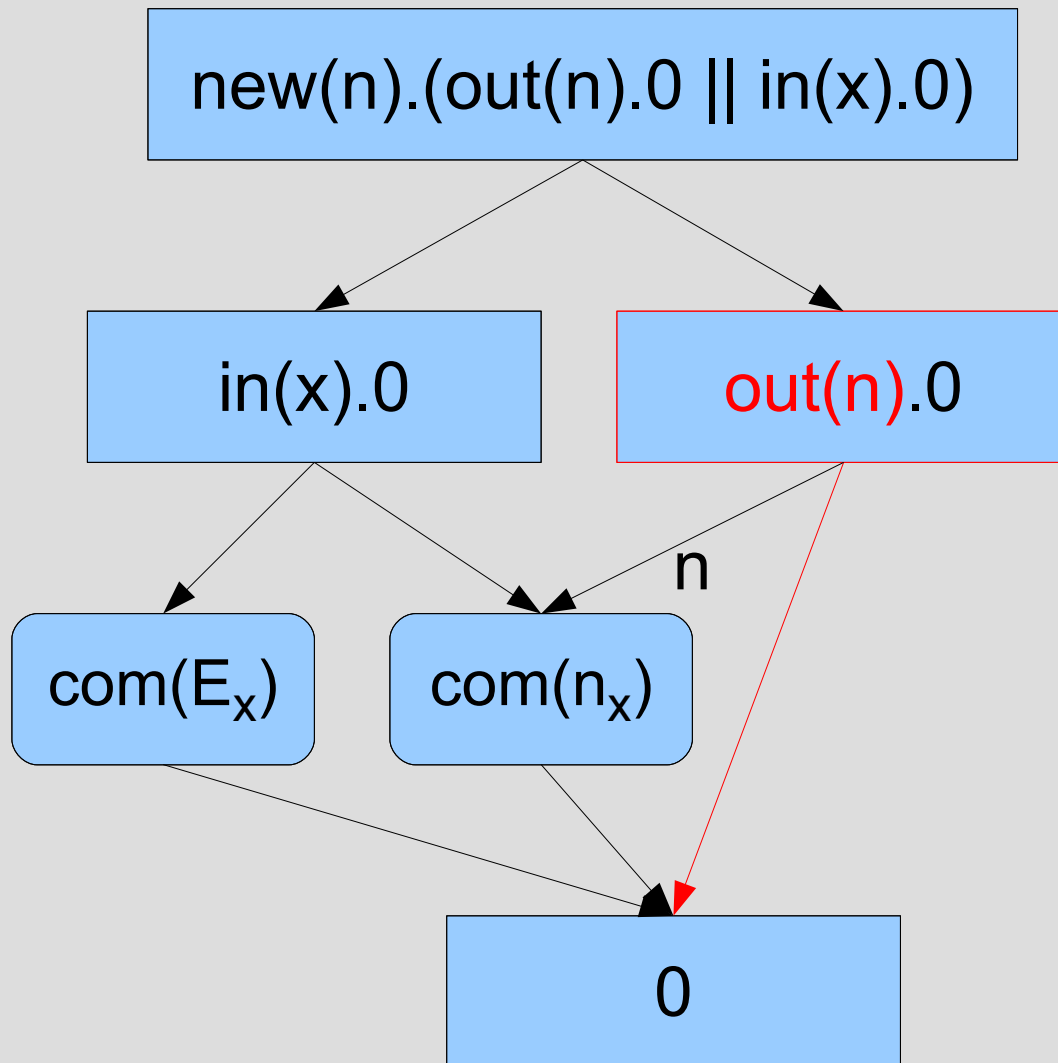
Outline

- Motivation
 - Analysis by hand
- Causality-based Abstraction
 - Graph
 - Traces
- **Security Analysis**
- Concrete Implementation
 - Problems
- Summary

from Graph to Analysis - Traces



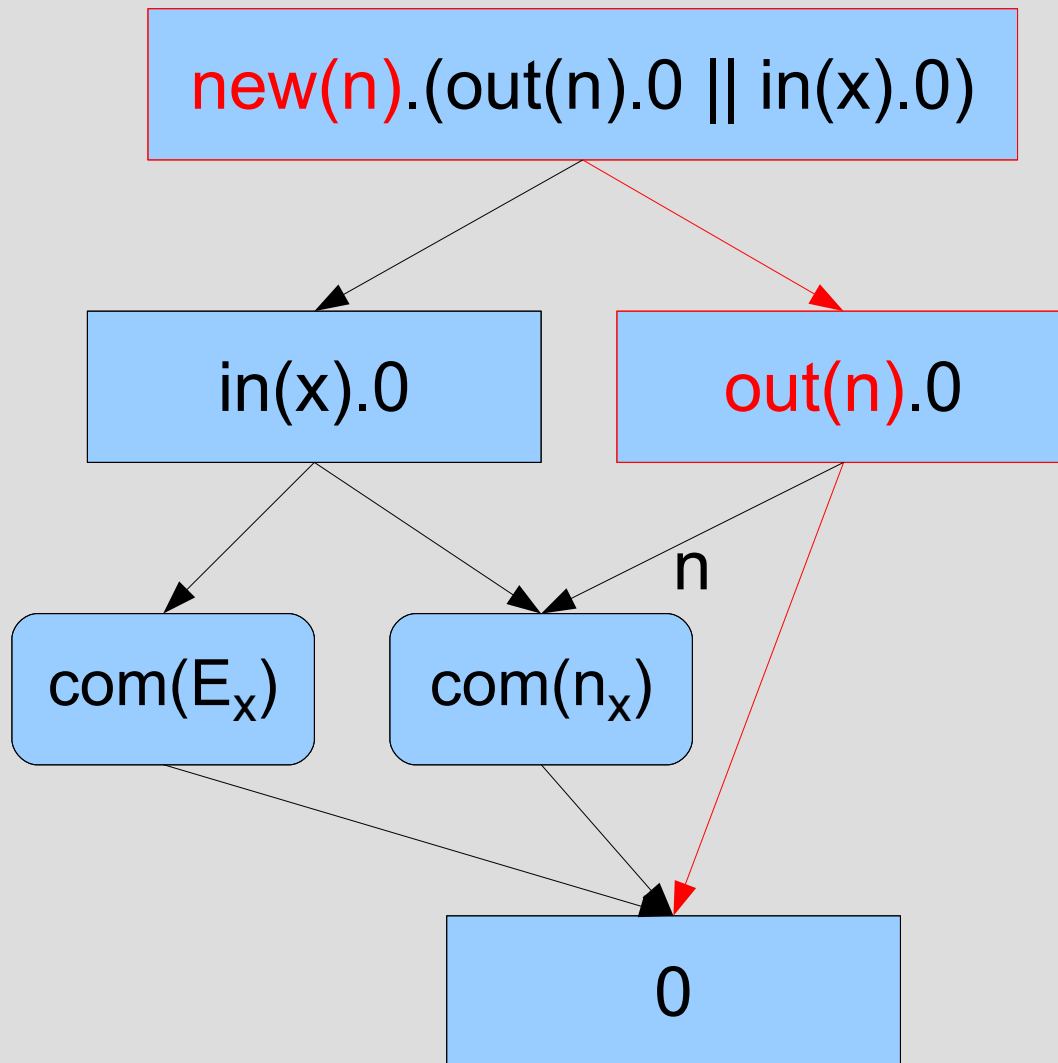
from Graph to Analysis - Traces



traces(0):

out(n)

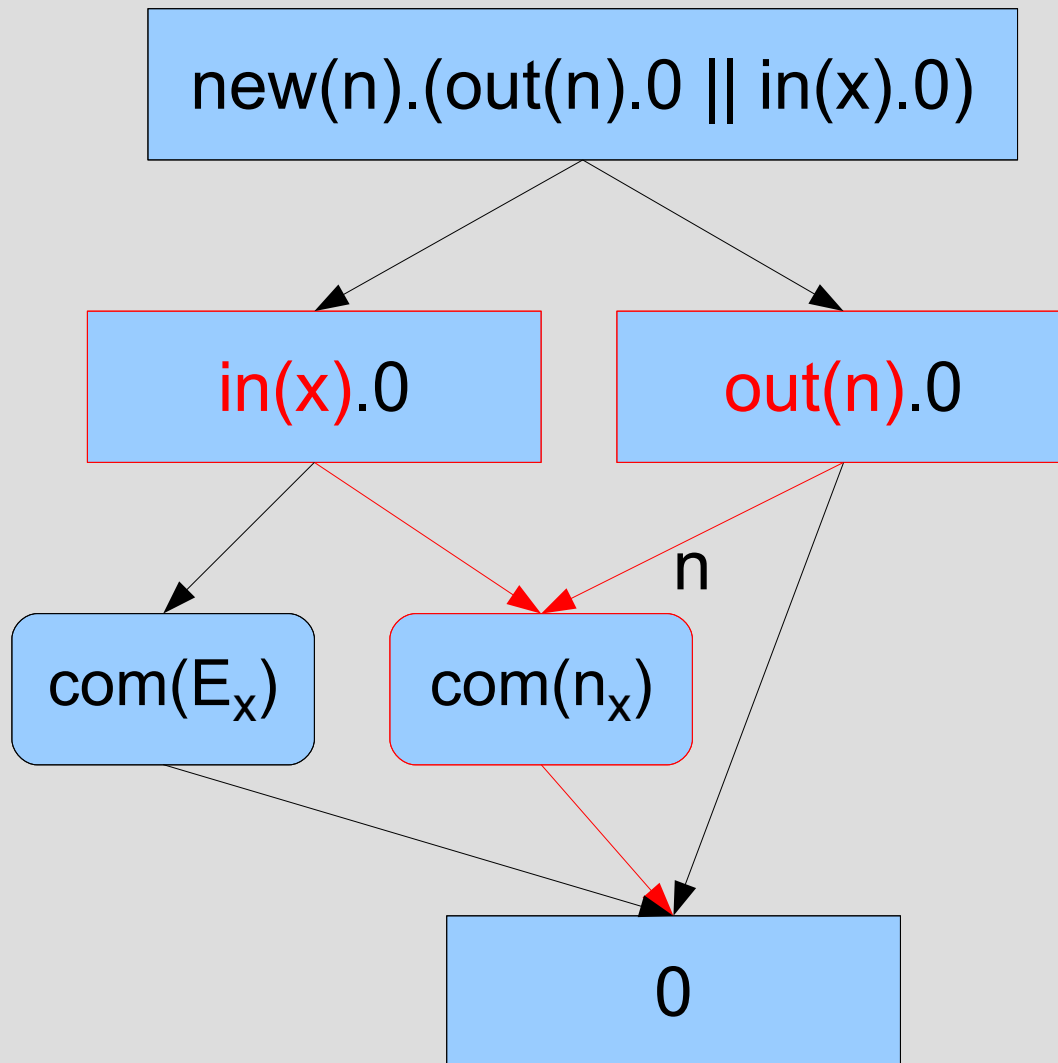
from Graph to Analysis - Traces



traces(0):

{ new(n) :: out(n)

from Graph to Analysis - Traces

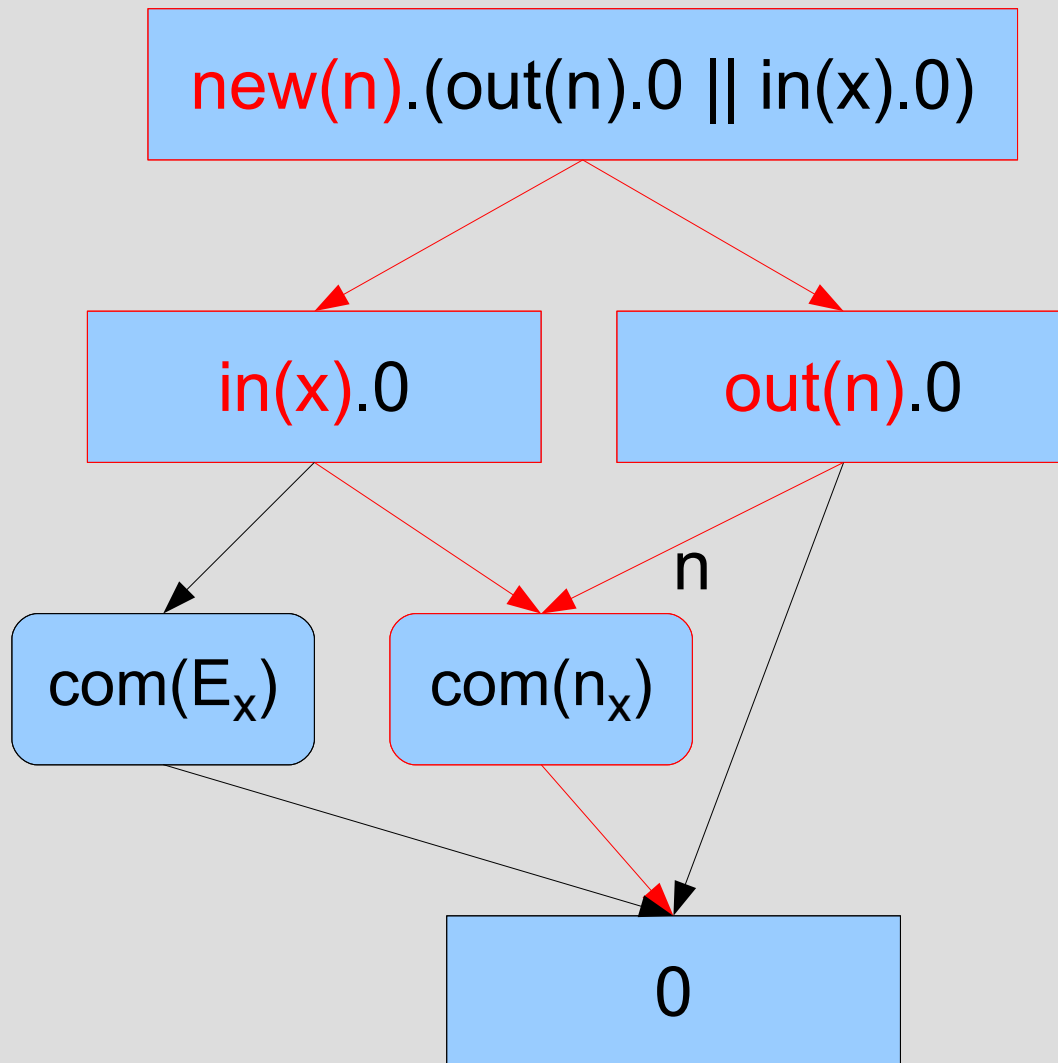


traces(0):

$\text{outcom}(n_x, n_x) :: \text{in}(n_x)$

$\text{in}(n_x)$

from Graph to Analysis - Traces

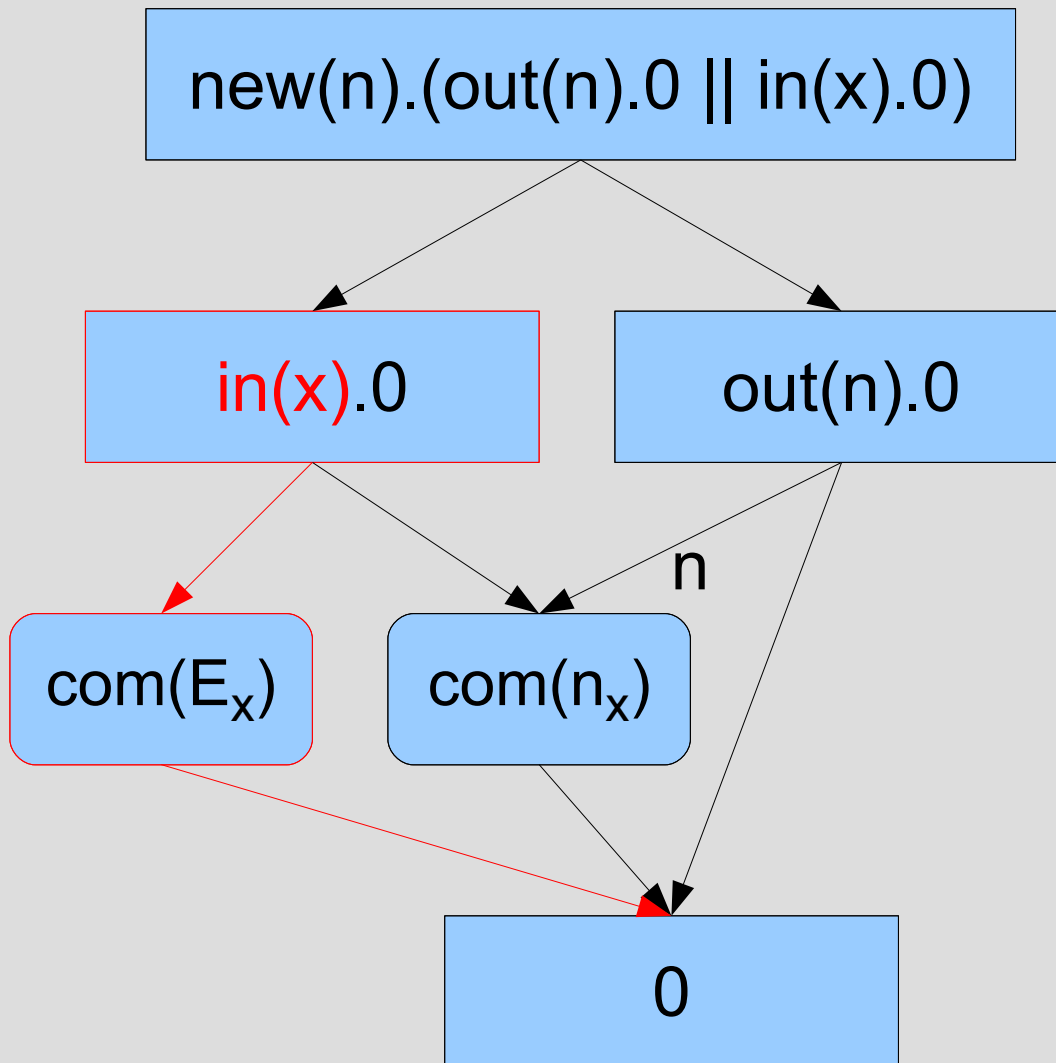


traces(0):

$\left\{ \begin{array}{l} \text{new}(n) :: \\ \text{outcom}(n_x, n_x) :: \text{in}(n_x) \end{array} \right.$

$\text{new}(n) :: \text{in}(n_x)$

from Graph to Analysis - Traces



traces(0):

$\text{in}(E_x)$

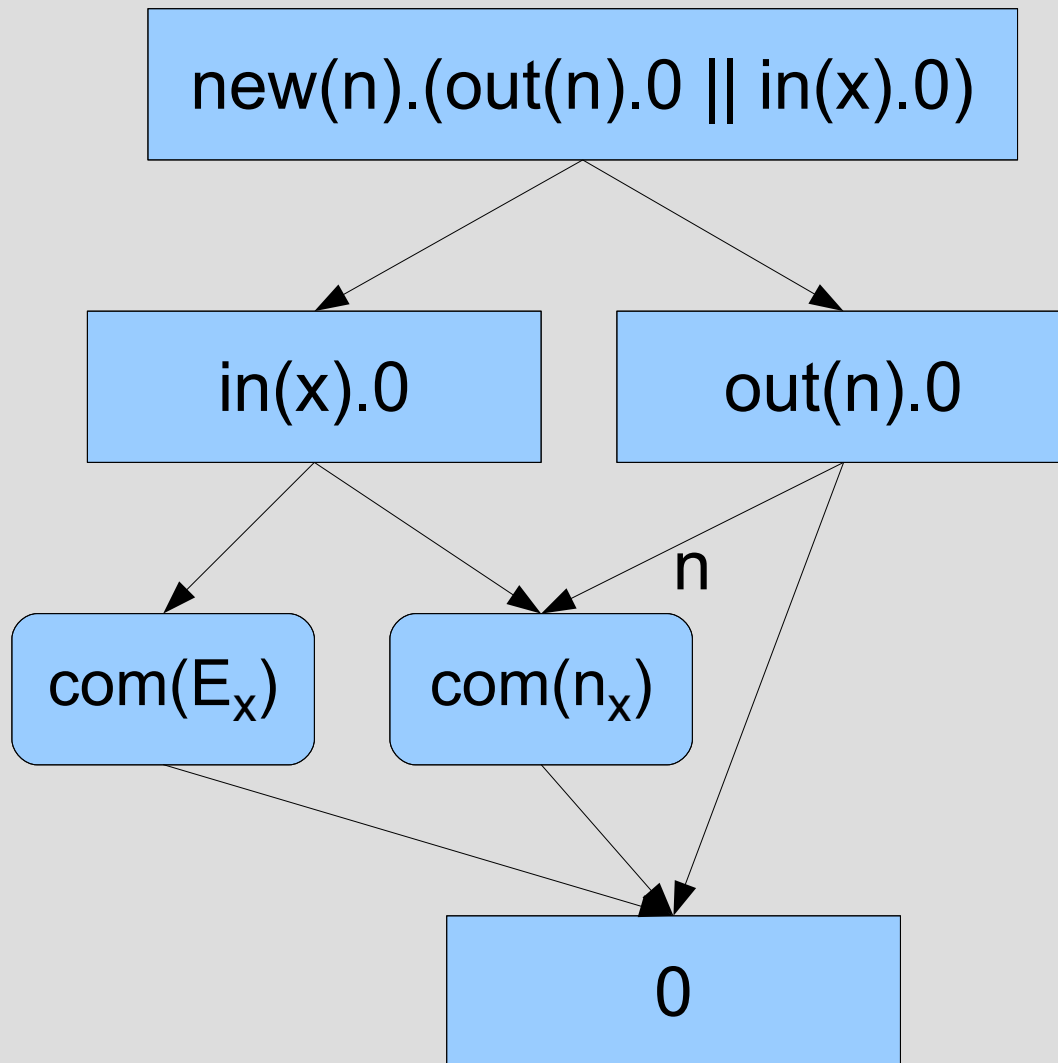
from Graph to Analysis - Traces



traces(0):

{ new(n) :: in(E_x)

from Graph to Analysis - Traces

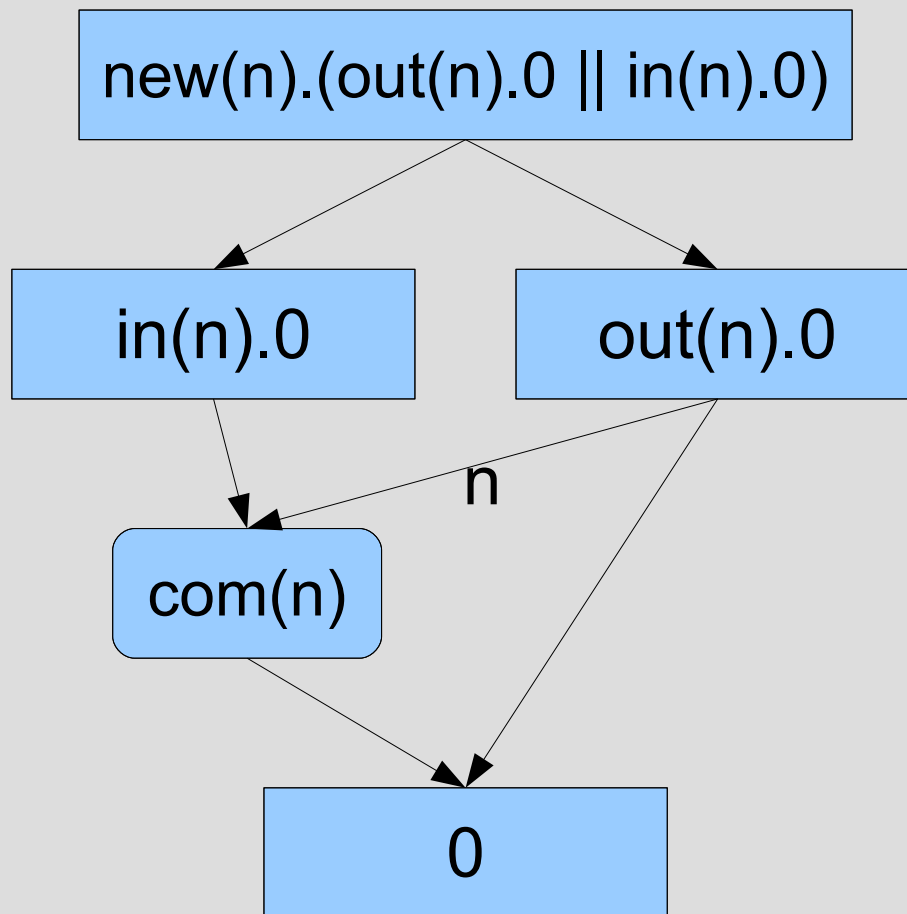


traces(0):

- $\left\{ \begin{array}{l} \text{new}(n) :: \\ \text{outcom}(n_x, n_x) :: \text{in}(n_x) \end{array} \right.$
- $\left\{ \text{new}(n) :: \text{in}(n_x) \right.$
- $\left\{ \text{new}(n) :: \text{out}(n) \right.$
- $\left\{ \text{new}(n) :: \text{in}(E_x) \right.$

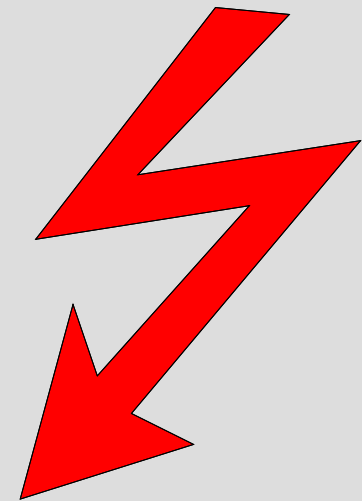
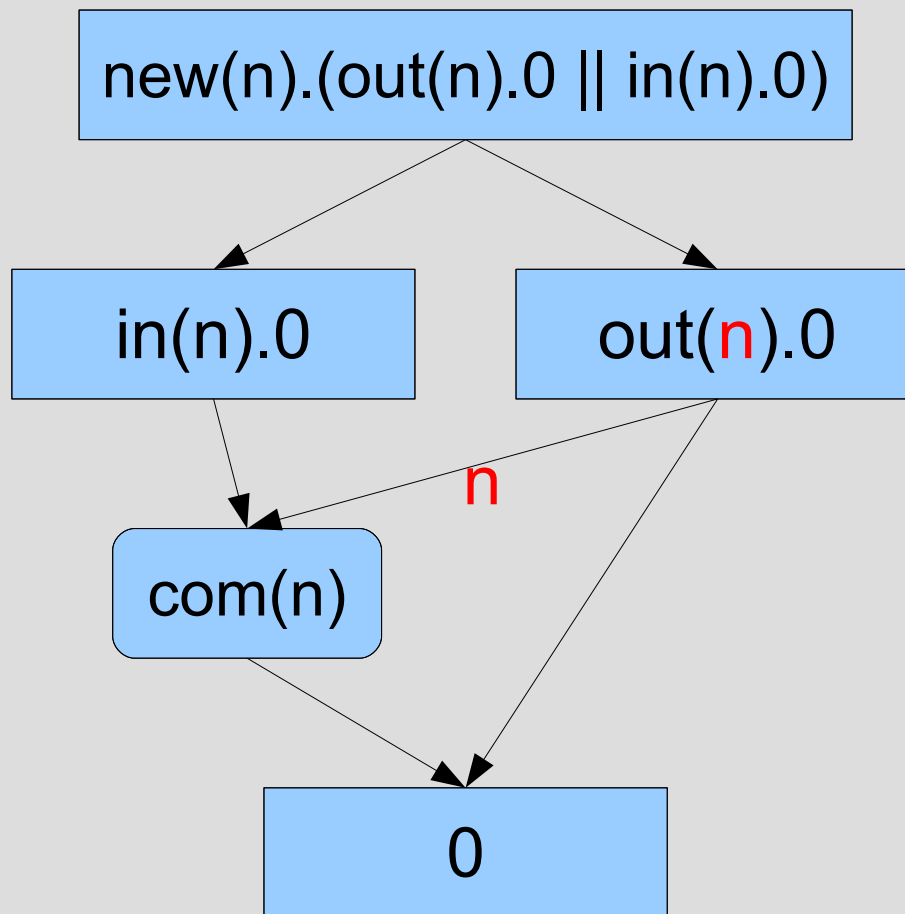
Analysis

Secrecy (of n)



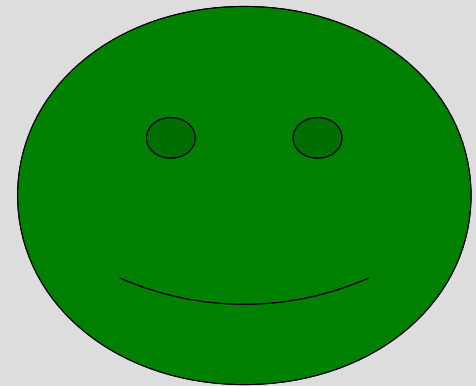
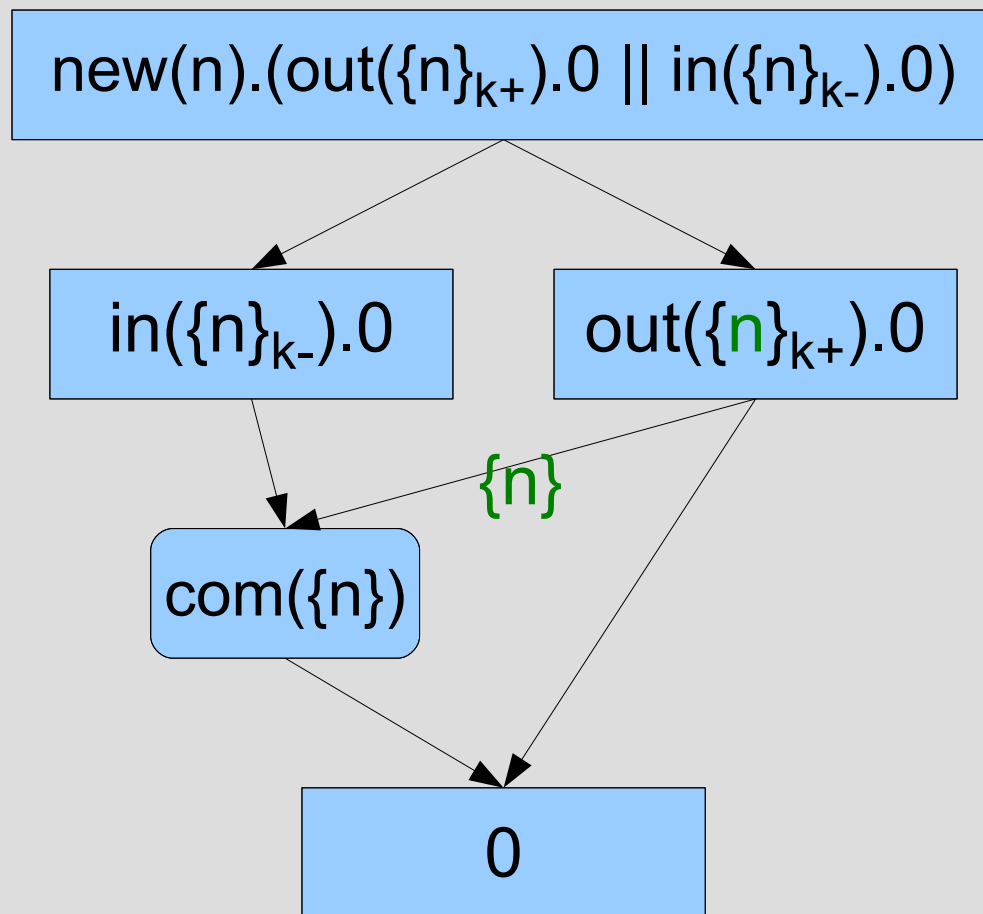
Analysis

Secrecy (of n)



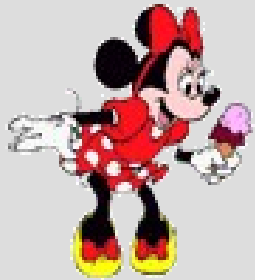
Analysis

Secrecy

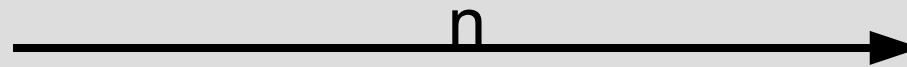


Analysis

Authenticity



$\text{begin}_n^1(A, B, m)$



$\text{end}_n^1(B, A, m)$

Analysis

Authenticity

newkey(A) :: new(m) :: new(n) ::
out($\{(B, n, m)\}_{kA+}$) :: in(n) :: $\text{end}^1_n(A, B, m)$

newkey(A) :: in($\{(B, n_x, m_z)\}_{kA-}$) ::
 $\text{begin}^1_{n_x}(B, A, m_z)$:: outcom(n, n) :: in(n) :: $\text{end}^1_n(A, B, m)$

newkey(A) :: new(m) :: new(n) ::
outcom($\{(B, n_x, m_z)\}_{kA+}, \{(B, n_x, m_z)\}_{kA+}$) ::
in($\{(B, n_x, m_z)\}_{kA-}$) :: $\text{begin}^1_n(B, A, m_z)$::
outcom(n, n) :: in(n) :: $\text{end}^1_n(A, B, m)$

Outline

- Motivation
 - Analysis by hand
- Causality-based Abstraction
 - Graph
 - Traces
- Security Analysis
- **Concrete Implementation**
 - Problems
- Summary

What did we do?

moved from stack overflow to endless loop. great result.

at some point we get now information about the place where it occurs in the parser or lexer or .. somewhere else

parser now in my opinion ugly but gives more expressive error messages than just parser error



nicer output lexer

did something

found out about precedence and went back to old version, so no more nice error messages so far

r4864: parser takes extremely long

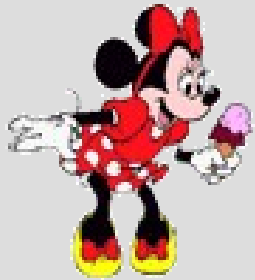
r4871: parser restored

Implementational Tricks and Features

- dynamic programming using hashtables
- static environment
- trees
- different outputs
- light-weight gui

Problems

Needham-Schroeder-Lowe



$$\{B, n_b, m\}_{k_A^+}$$



$$\{A, n_b, n_a\}_{k_B^+}$$

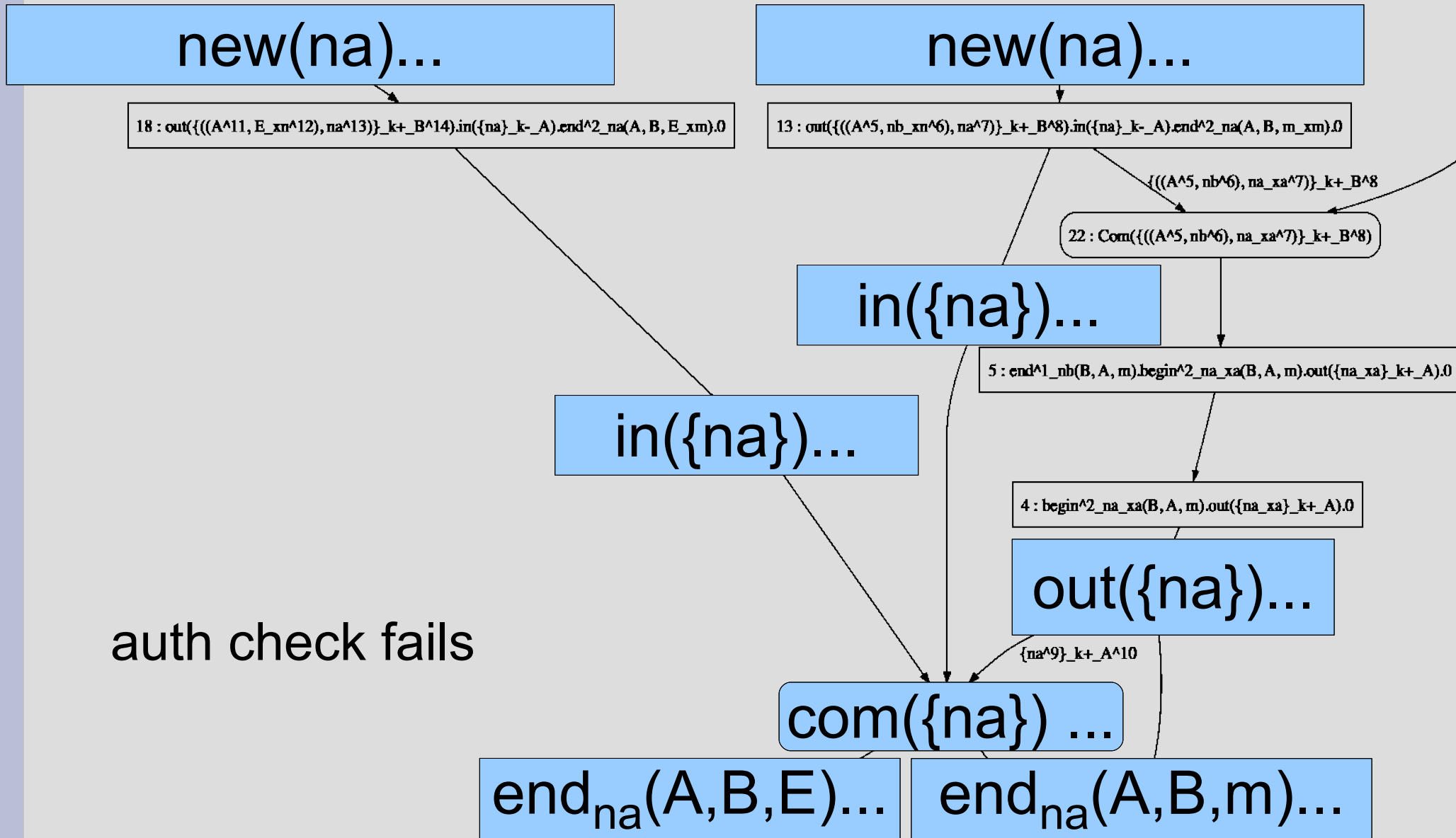


$$\{n_a\}_{k_A^+}$$



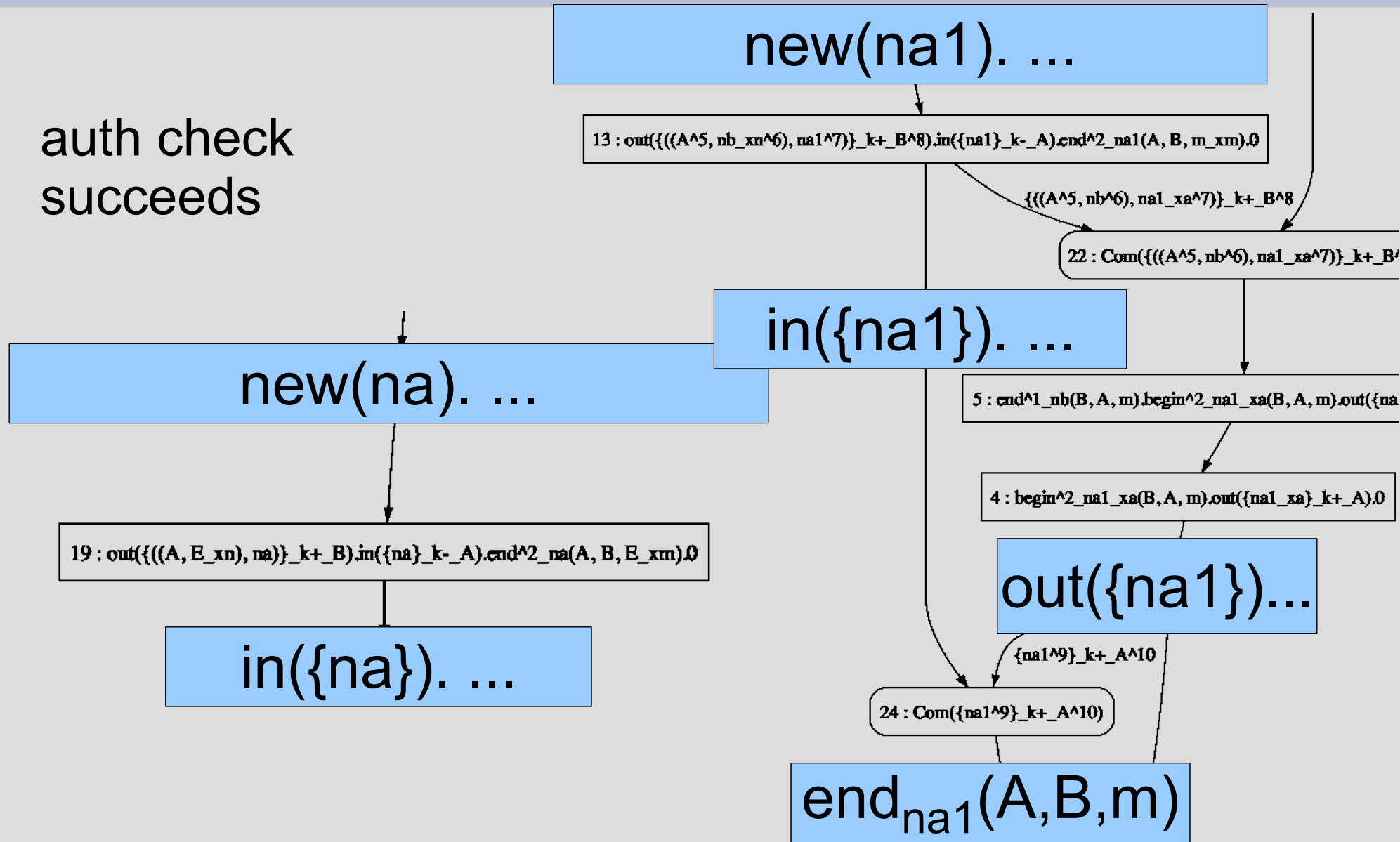
Problems

Needham-Schroeder-Lowe

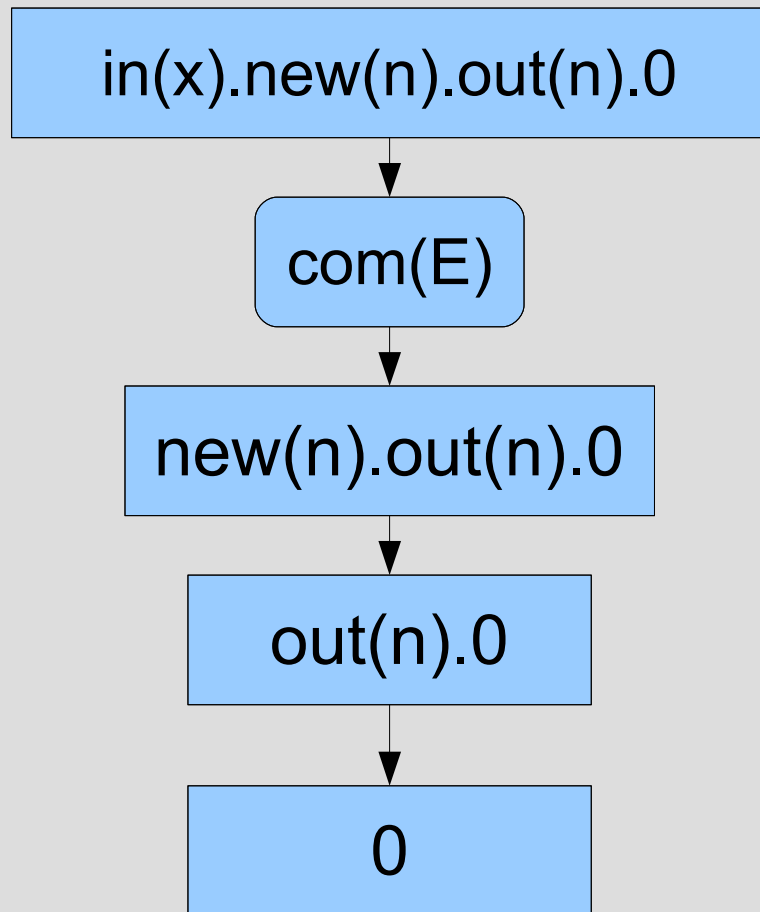


Fix - alpha-renaming

auth check
succeeds

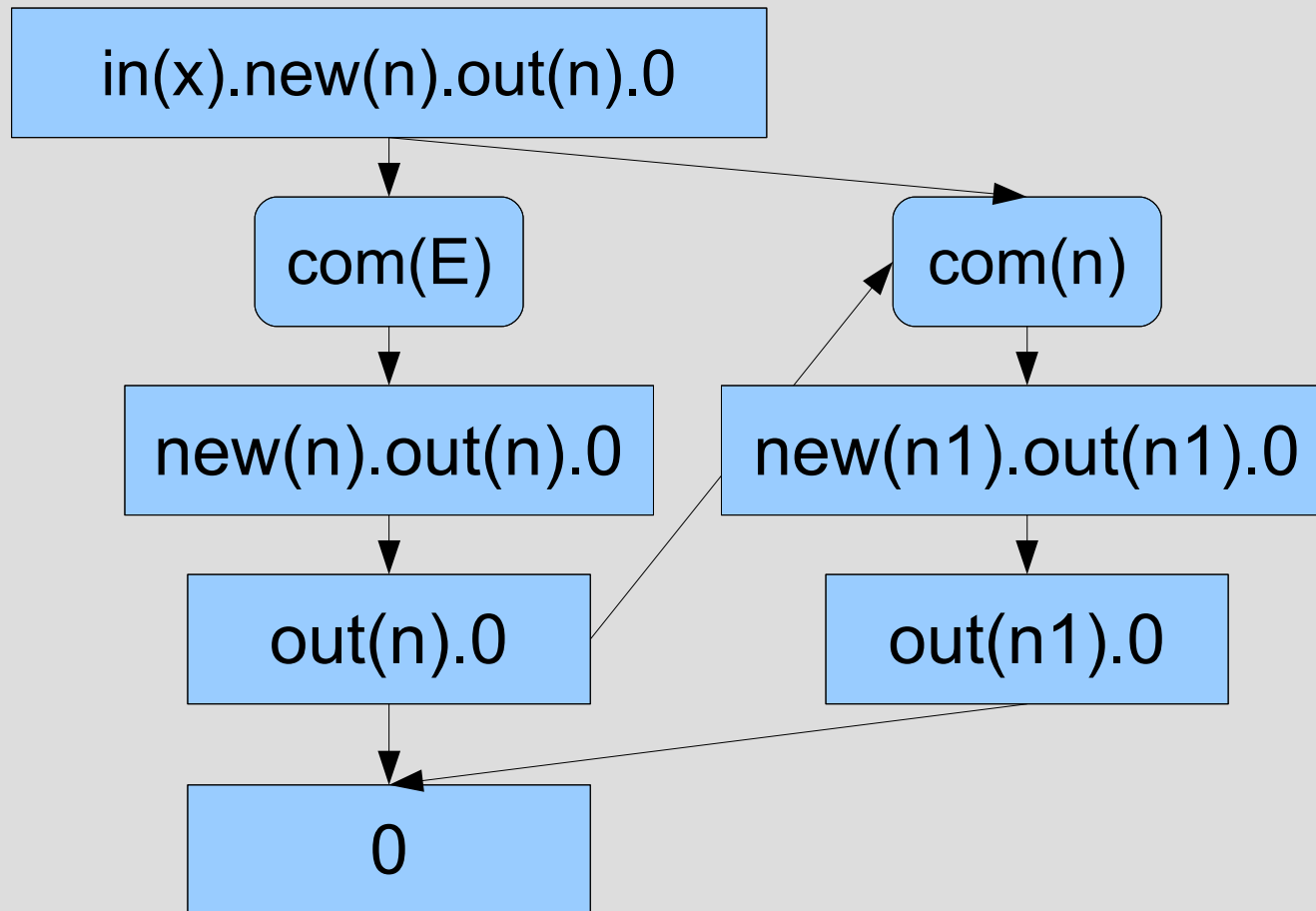


Alpha-renaming as solution?

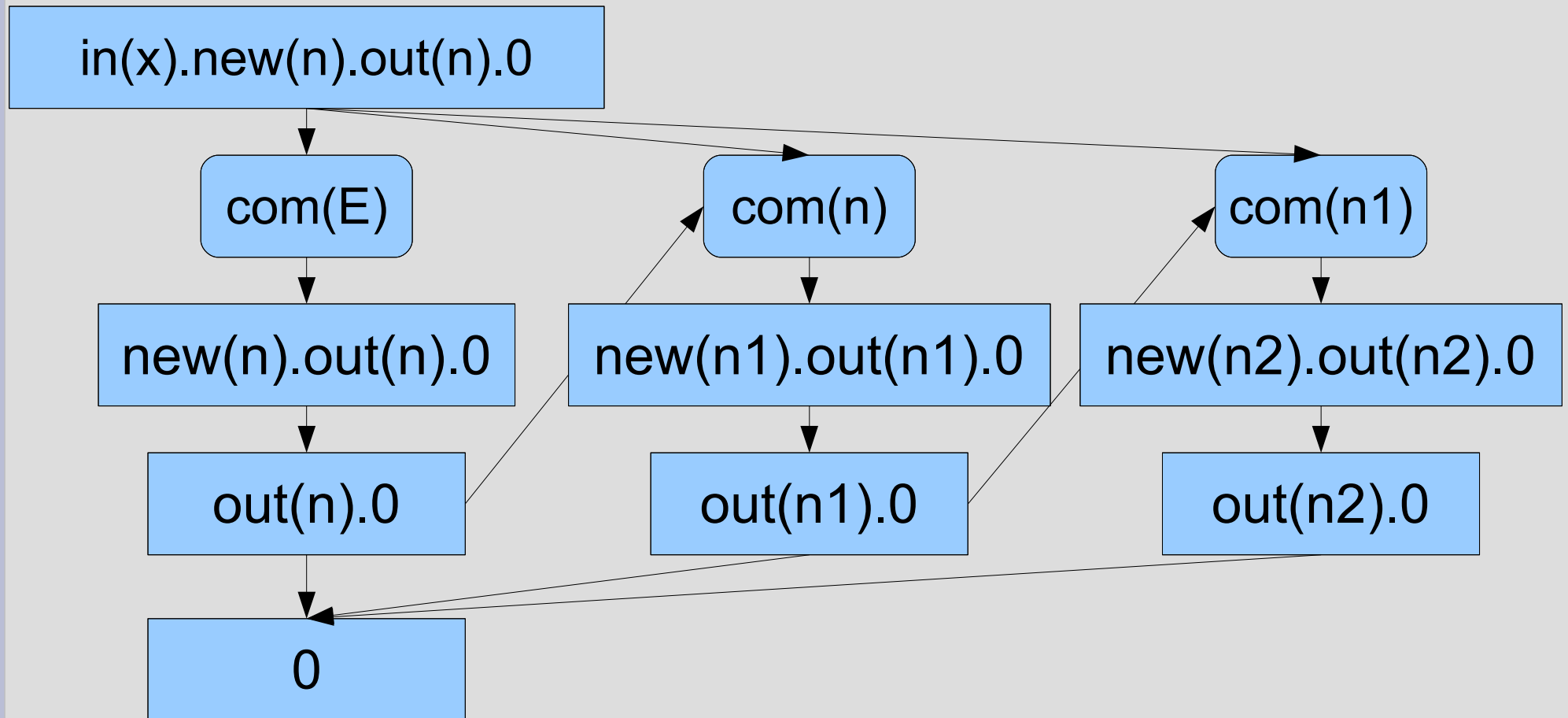


`in(x).new(n).out(n).0`

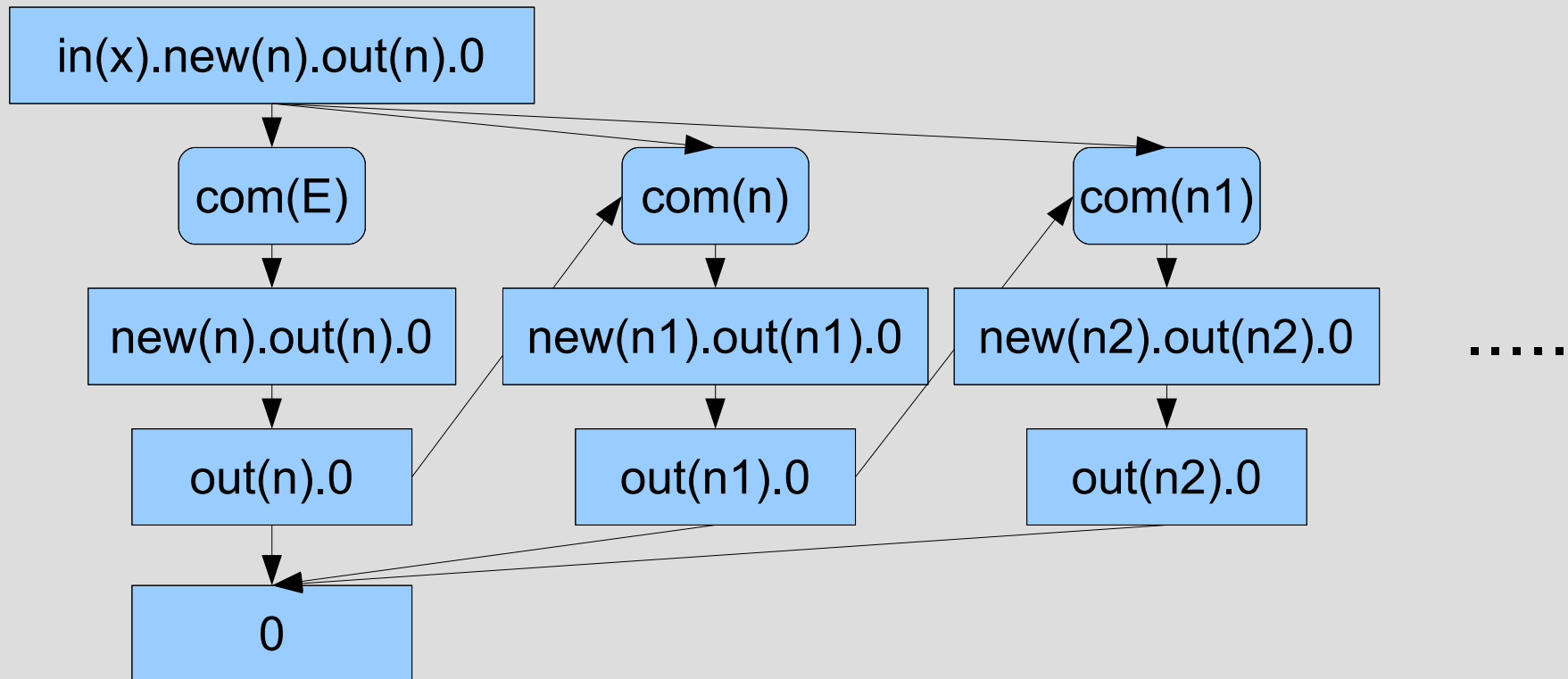
Alpha-renaming as solution?



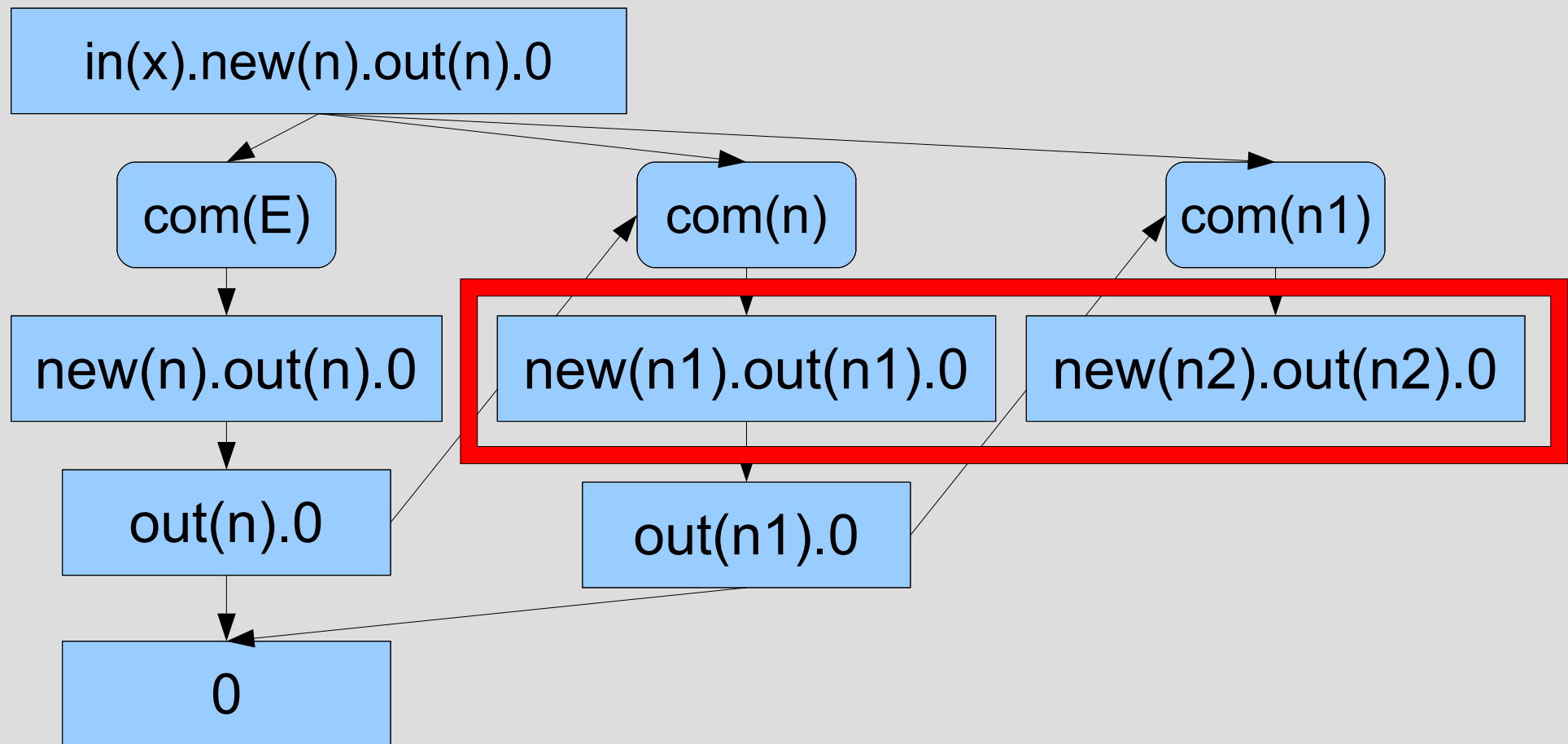
Alpha-renaming as solution?



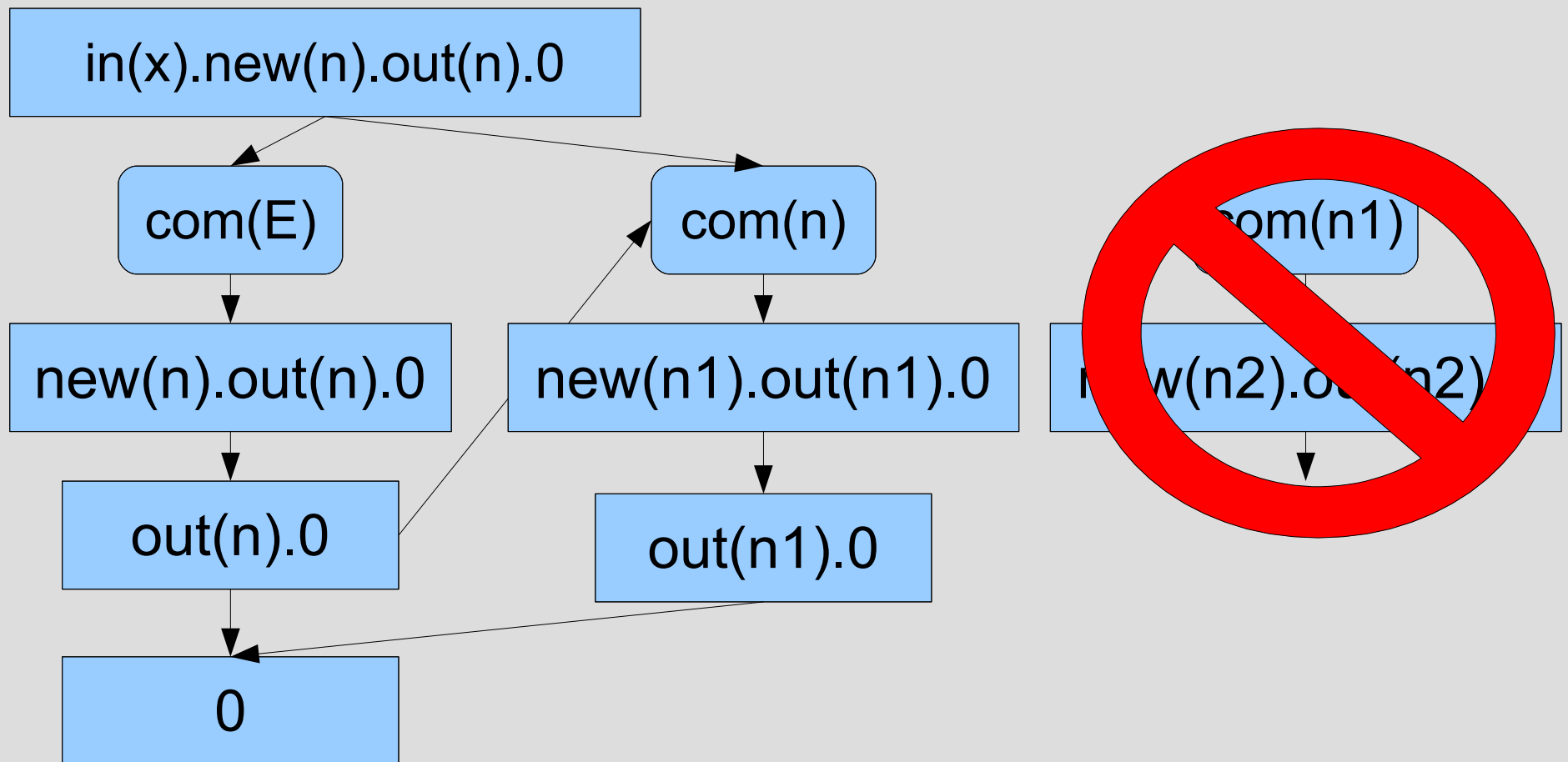
Alpha-renaming as solution?



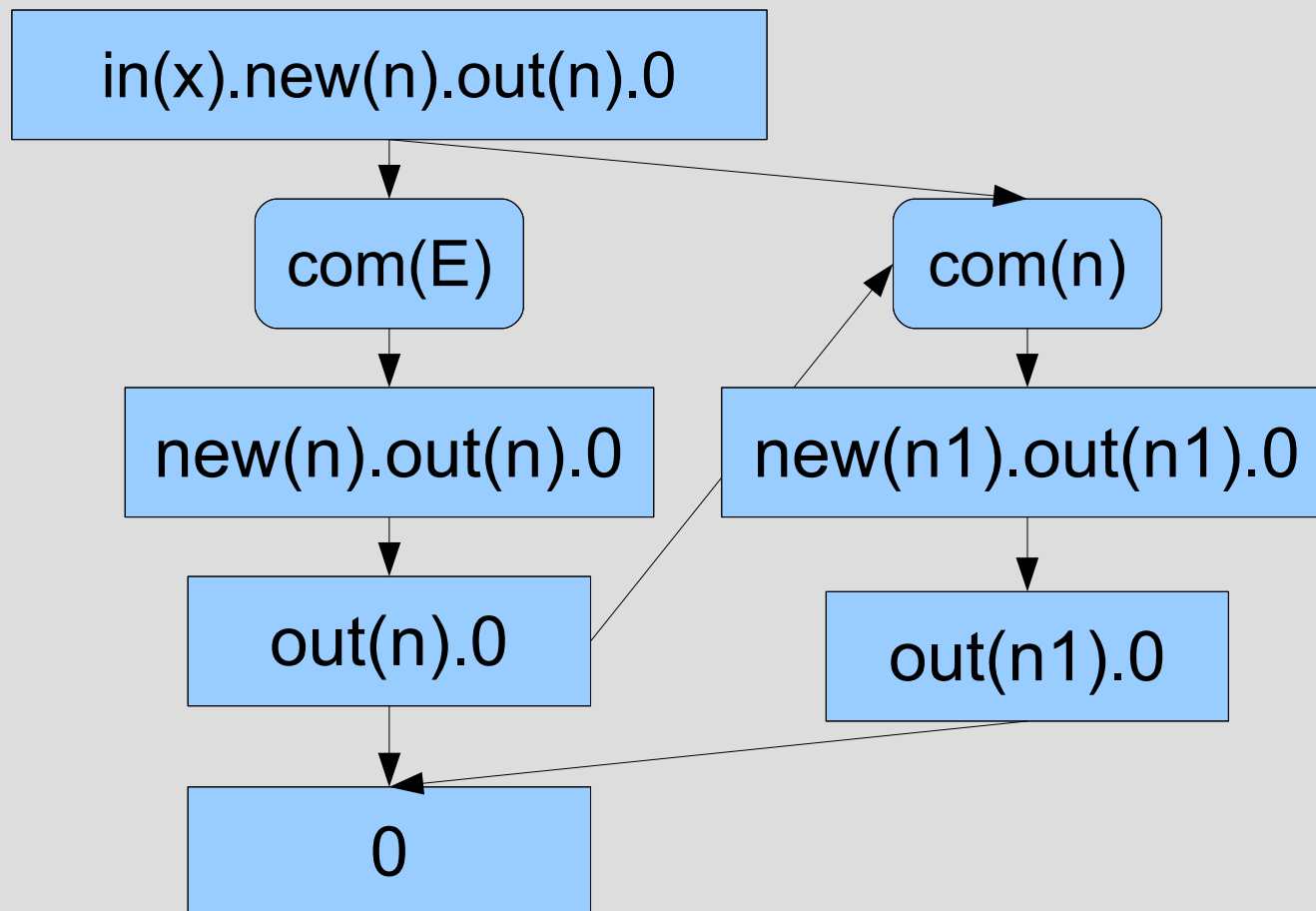
Finally the solution!



Finally the solution!



Finally the solution!



Summary

The screenshot displays a graphical user interface for a protocol checker. It features three main windows and a diagram on the right.

protocol checker gui window:

- Buttons: folder, document, magnifying glass, question mark.
- Text: "current protocol:"
- Code:

```
newkey(A).((A |> new(m).new(n).out({B,n,m | pubkey(A)}).in(n).end(1 n A B m).0)
|| (B |> in({B,x,z | seckey(A)}).begin(1 x B A z).out(x).0))
```
- Buttons: "reset", "parse"

loading file: /Users/broeni/Uni/fopraPL/protocols/prot1.prot window:

- Text:

```
loaded prot: new+-(k_A).A |> new(m).new(n).out({(B, n), m)}_k+_A).in(n).end^1_n(A, B, m).0 | B
|> in({(B, x), z)}_k-_A).begin^1_x(B, A, z).out(x).0
comments for this protocol:
# this is the protocol from the paper
#   A   B
#   <- {B,n,m}k+A --
#   begin 1 n (A,B,m)
#   ----- n ----->
#   end 1 n (B,A,m)
# provides authenticity (B is convinced he is talking to A)
# the name n is not secure
```

protocol analysis results window:

- Text:

```
The protocol is cycle invariant
The protocol does NOT guarantee secrecy for the following names:
  n leaked by nodes: 3
The protocol guarantees authenticity
```
- Button: "hide"

Diagram (test15.prot.png):

- Nodes: $out(n).out(m).0$, $4 : out(n^1).out(m).0$, 0 , $1 : out(m^2).0$, $Com((n^1, m^2))$, $0 : 0$.
- Transitions: $out(n).out(m).0 \rightarrow 4 : out(n^1).out(m).0$; $4 : out(n^1).out(m).0 \rightarrow 0$; $4 : out(n^1).out(m).0 \rightarrow 1 : out(m^2).0$ (labeled n^1); $1 : out(m^2).0 \rightarrow Com((n^1, m^2))$ (labeled m^2); $0 \rightarrow Com((n^1, m^2))$; $Com((n^1, m^2)) \rightarrow 0 : 0$.

Bottom status bar: "22 png 8bits Alpha" and "100%" zoom.

Outlook and further work

- testing the tool
- refactoring and optimization
- integration in larger project (avispa)
- support for unified description language

References

- M. Backes, M. Maffei, A. Cortesi. Causality-based Abstraction of Multiplicity in Security Protocols. To appear in proceedings of 20th IEEE Computer Security Foundations Symposium, Venice, July 2007

Thank you!