Dependency-graph-based protocol analysis

Peeter Laud

peeter.laud@ut.ee

http://www.ut.ee/~peeter_l

Tartu University & Cybernetica AS

(joint work with Ilja Tšahhirov)

Dependency graphs

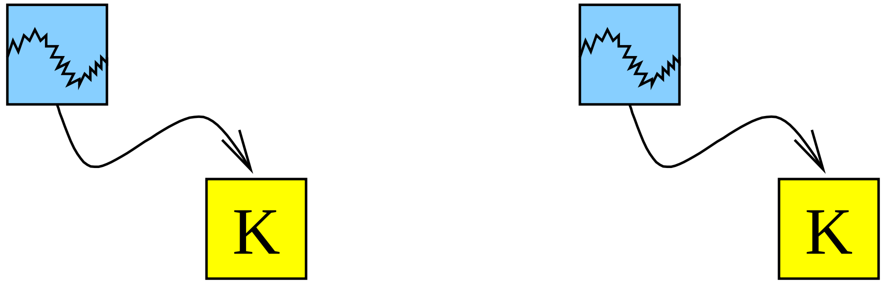
- Directed graph, nodes are labeled with operations.
  - The label of a node determines its in-degree.
  - Incoming edges are (usually) ordered.
- Nodes of a DG compute values, purely functionally.
- Edges describe where the values are used for further computations.
- Special nodes are used to bring inputs to the system.
  - . . . and transmit the outputs.

A protocol

- A wants to send the secret M to B.
- S is a trusted server.

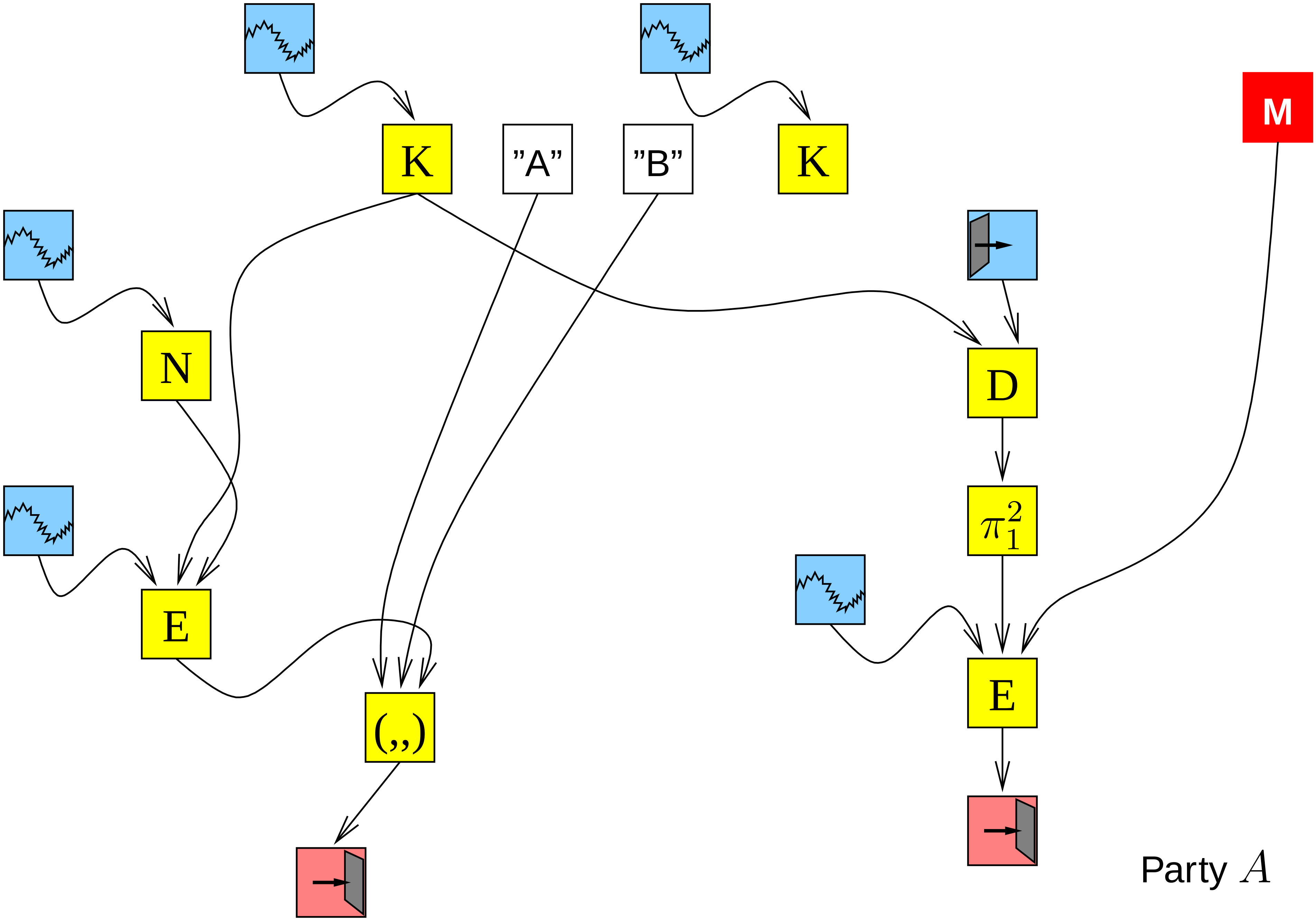
$$
\begin{aligned}
A \longrightarrow B &: \ A, B, \{N_A\}_{K_{AS}} \\
B \longrightarrow S &: \ A, B, \{N_A\}_{K_{AS}}, \{N_B\}_{K_{BS}} \\
S \longrightarrow A &: \ \{K_{AB}, N_A\}_{K_{AS}} \\
S \longrightarrow B &: \ \{K_{AB}, N_B\}_{K_{BS}} \\
A \longrightarrow B &: \ \{M\}_{K_{AB}}
\end{aligned}
$$

A protocol

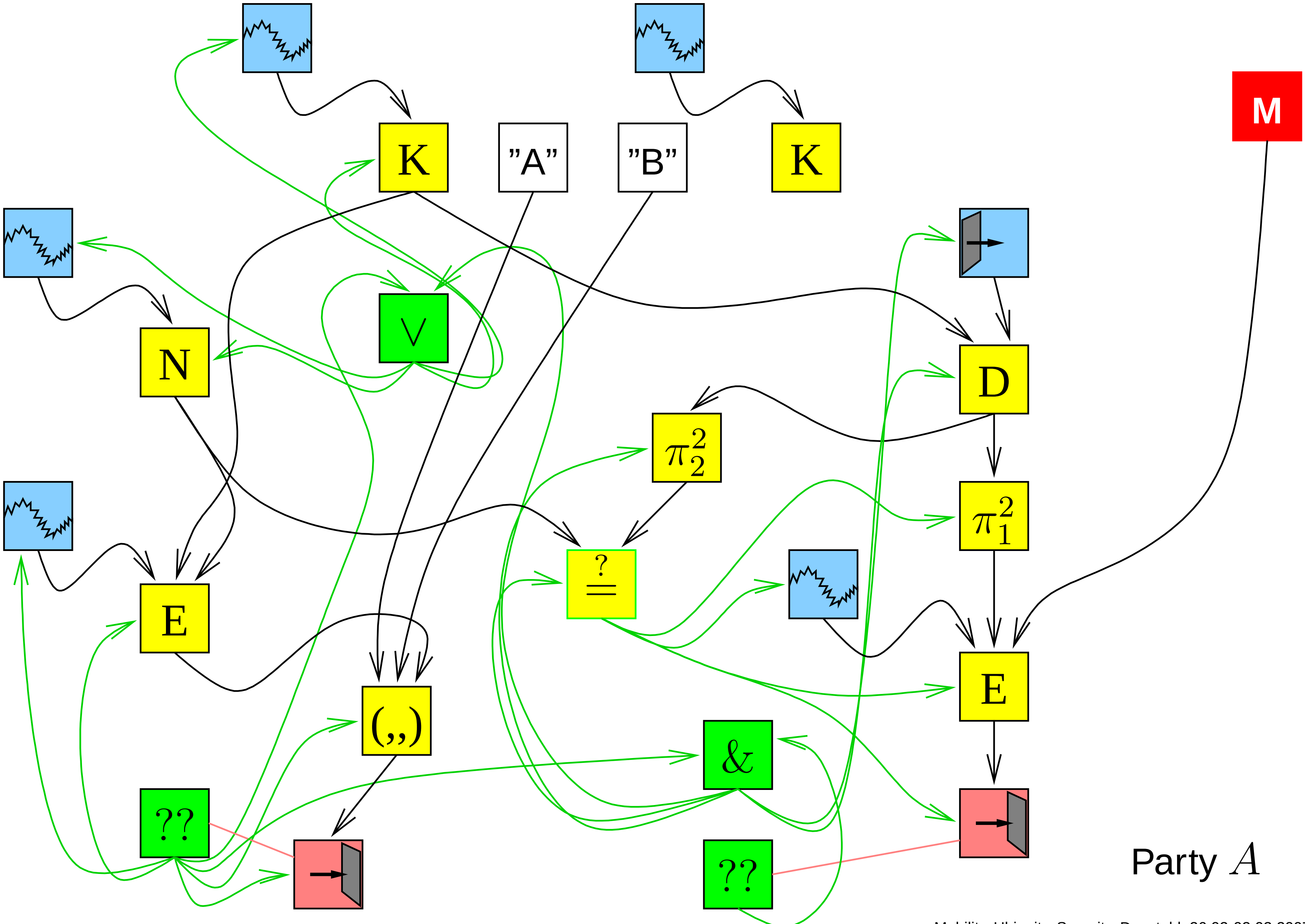


Generate keys K_{AS} and K_{BS}

A protocol

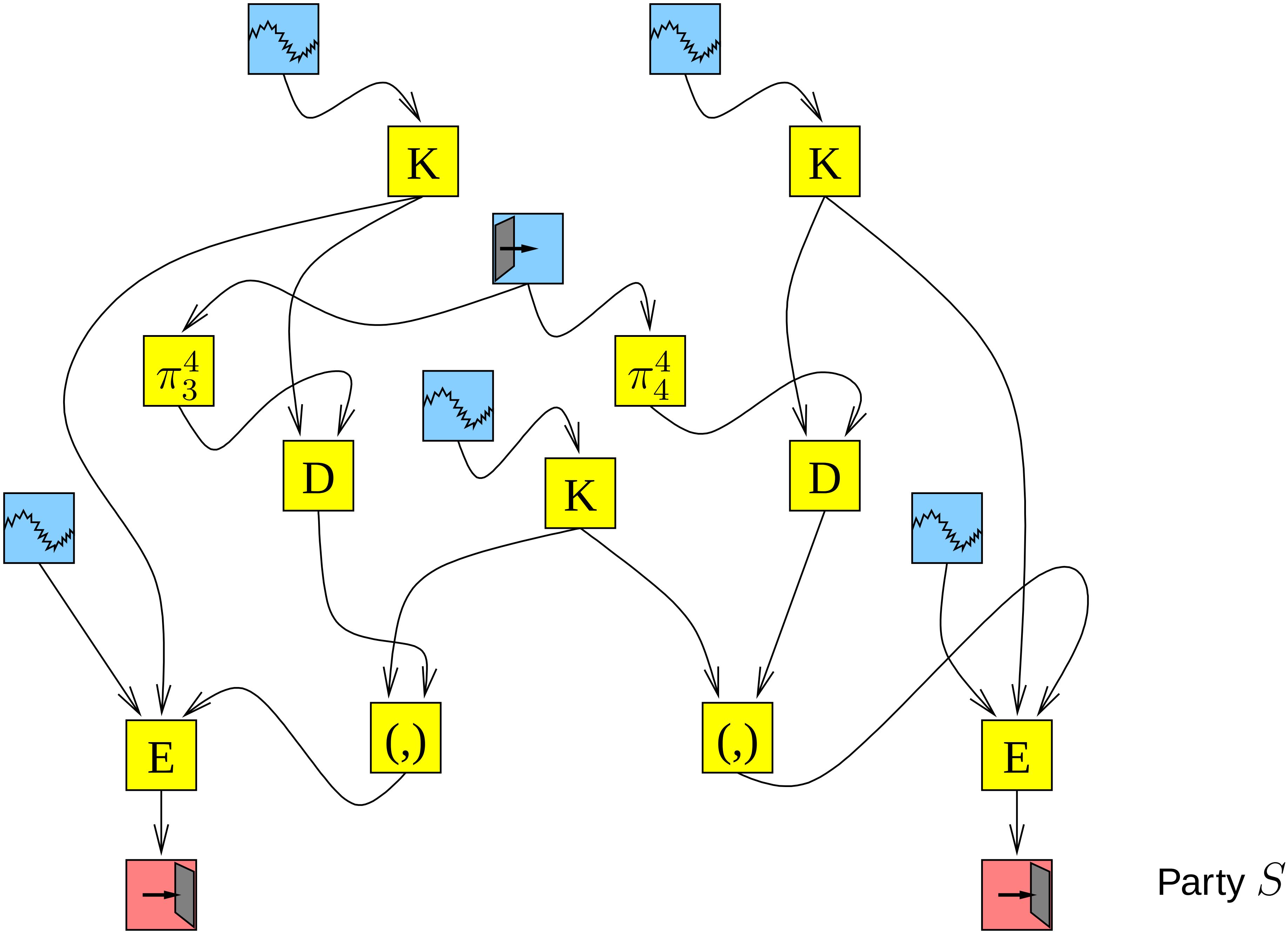


A protocol

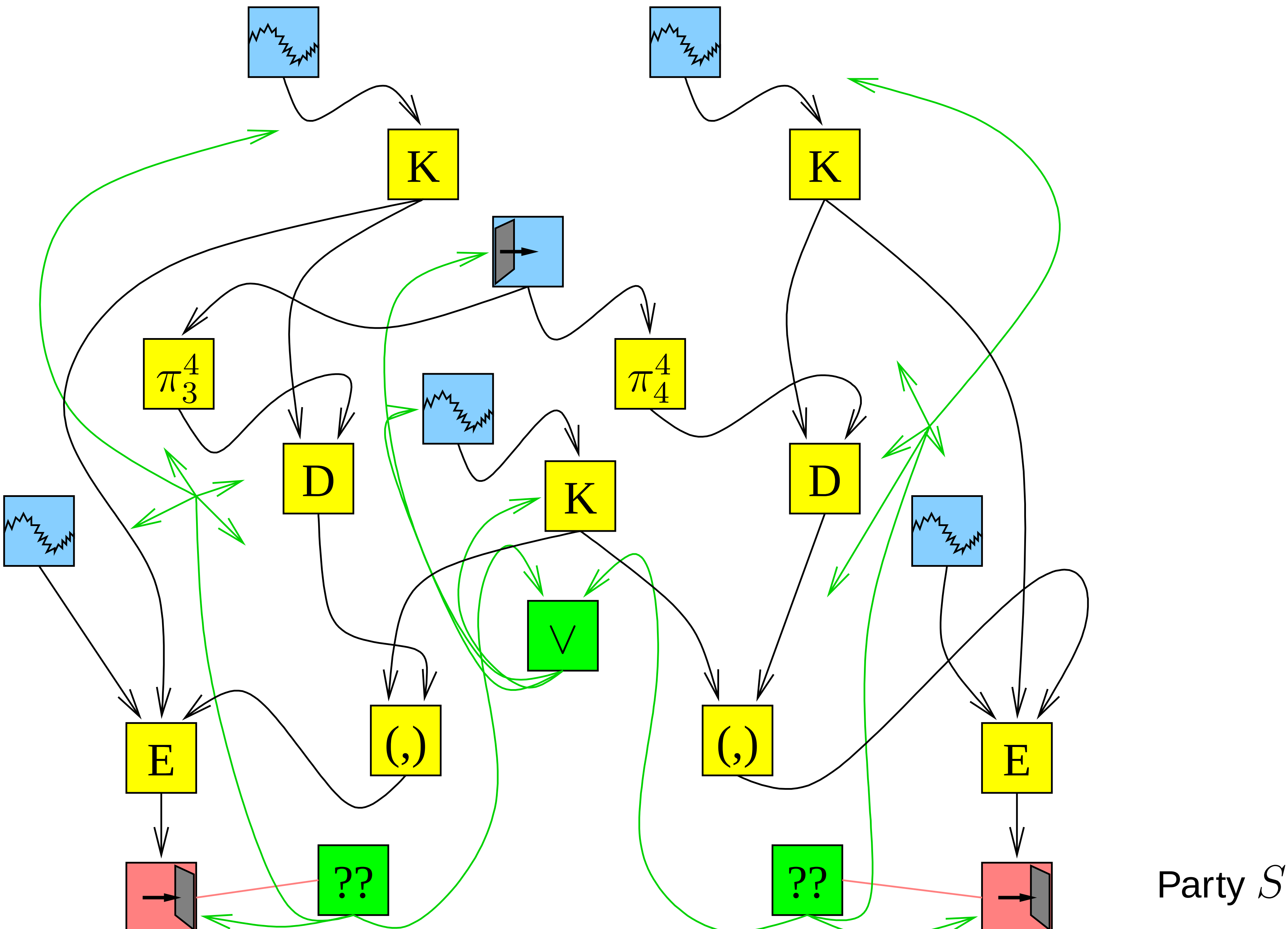


Party A

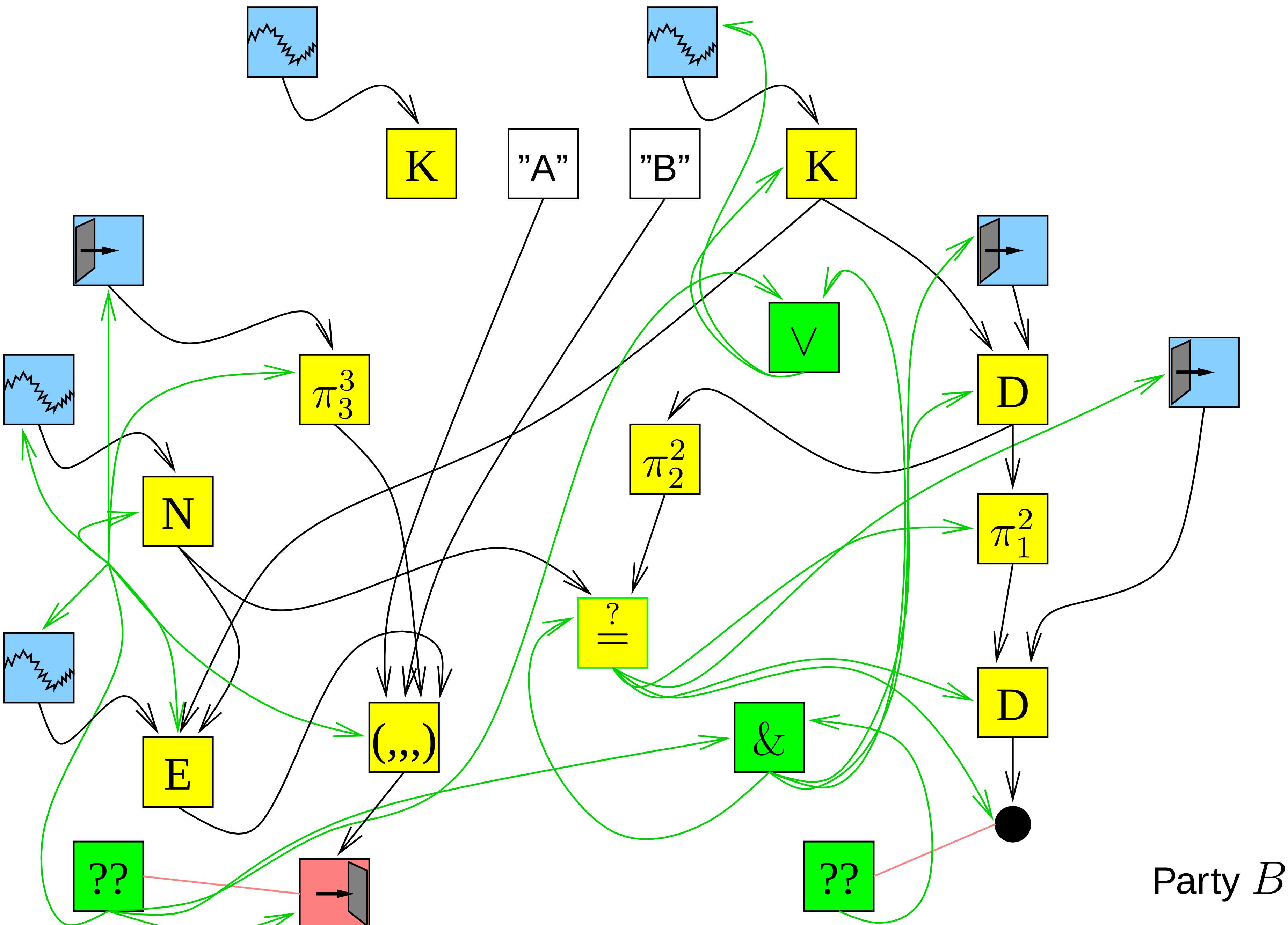
A protocol



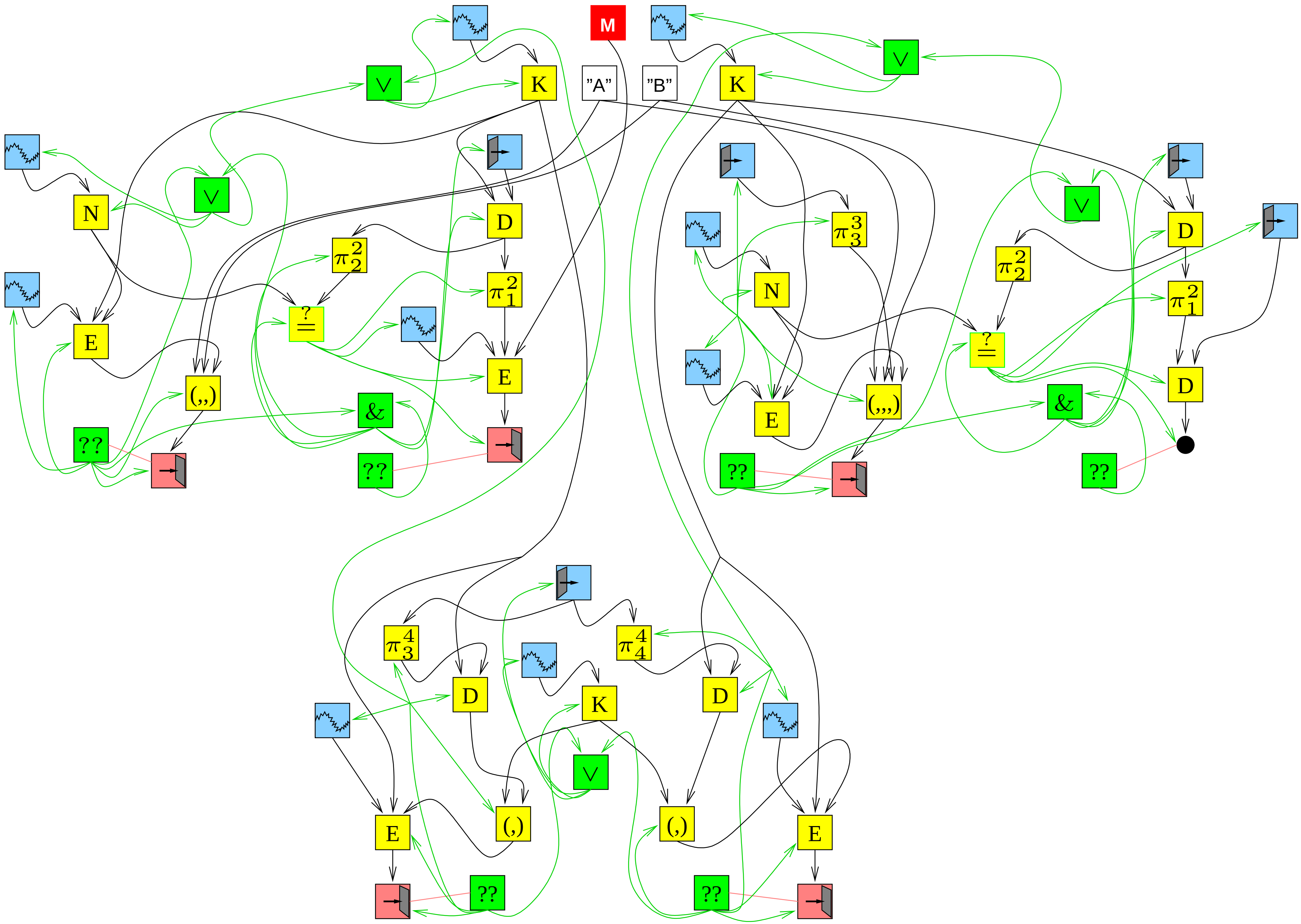
A protocol



A protocol



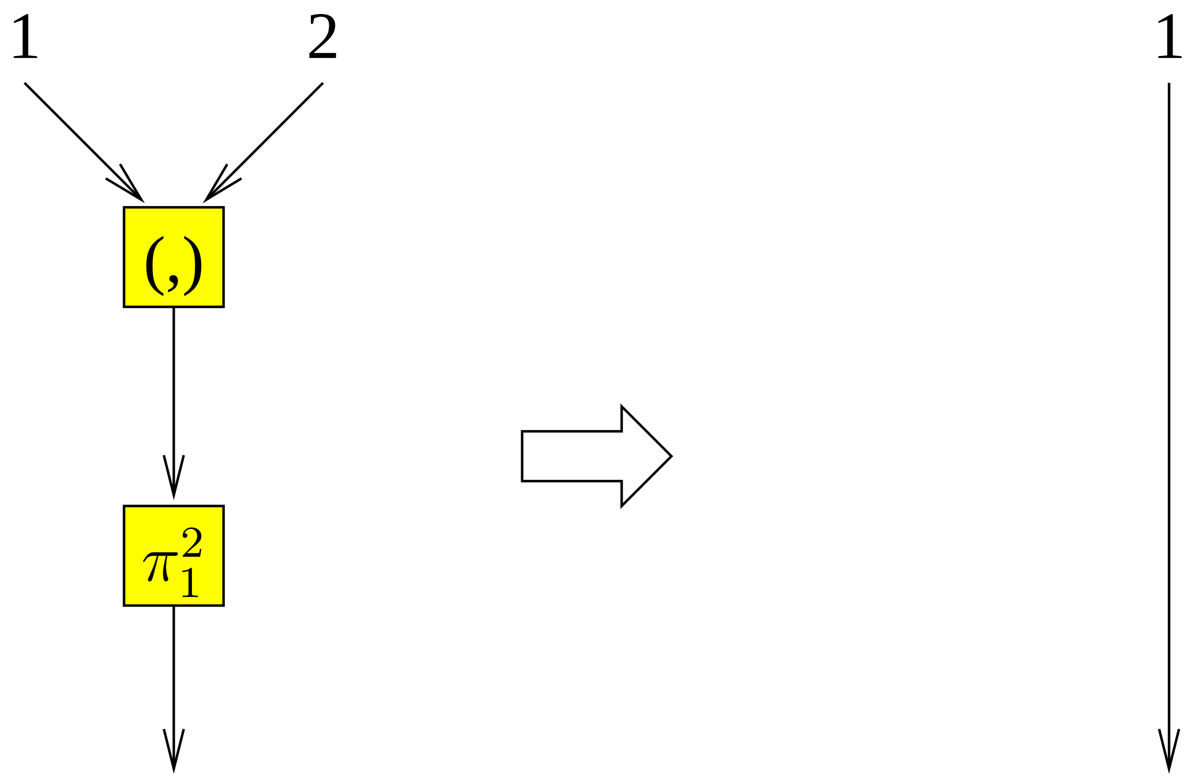
A protocol


M
"A"
"B"
K
K
$\vee$
$\vee$
$\vee$
$\vee$
$\vee$
N
N
D
D
D
D
D
E
E
E
E
E
E
π_2^2
π_1^2
π_3^3
π_2^2
π_1^2
π_3^4
π_4^4
$\stackrel{?}{=}$
$\stackrel{?}{=}$
(,,)
(,,,)
(,)
(,)
&
&
??
??
??
??
??
??

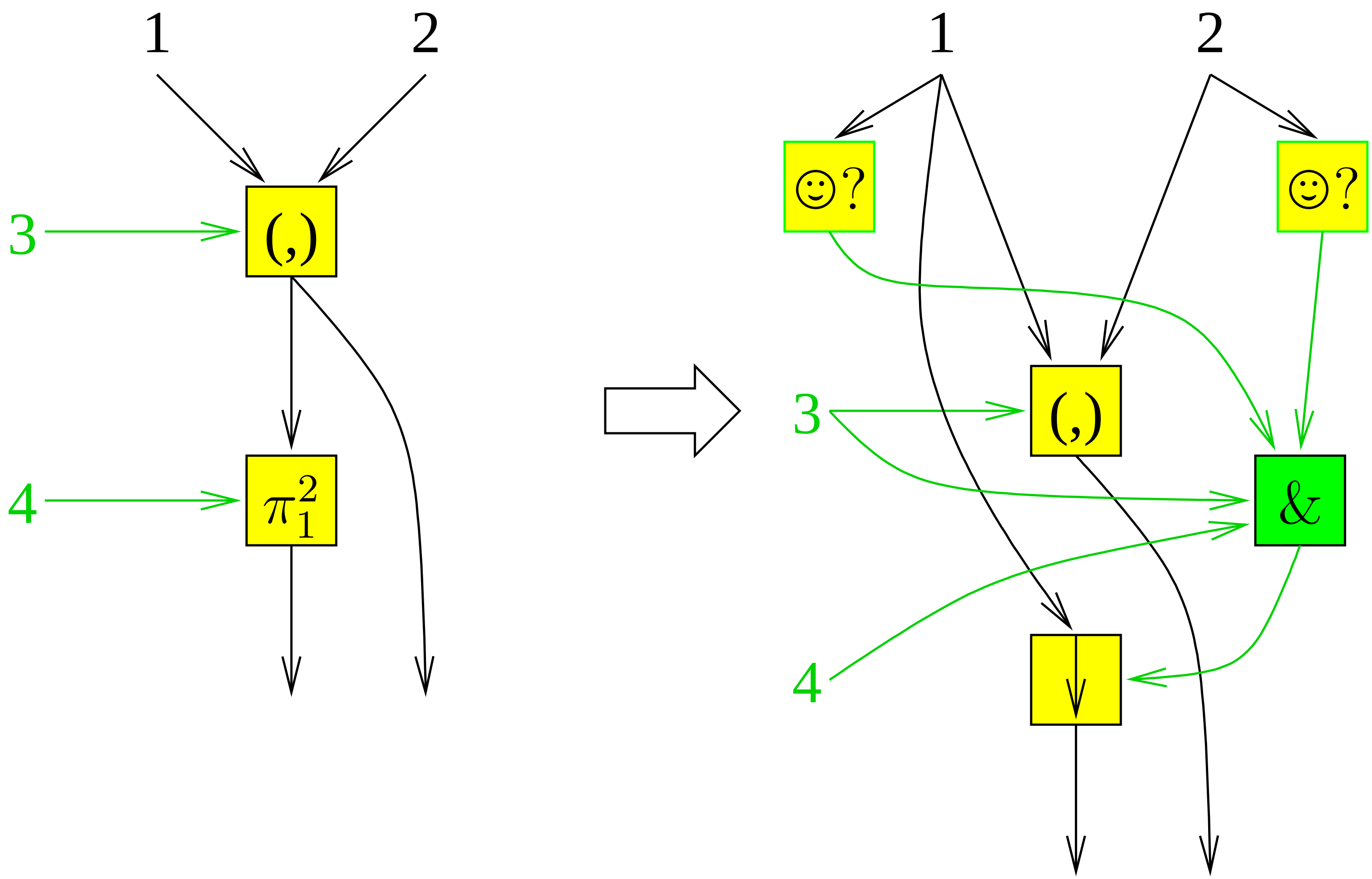
Good sides

- The structure of definitions and uses of values is explicit.
  - No copying of values.
  - No variable names at all. . .
- We immediately see what is used where.
  - . . . which greatly simplifies finding out whether some cryptographic reduction is allowed.
  - . . . and also helps doing other simplifications.

Some obvious simplifications

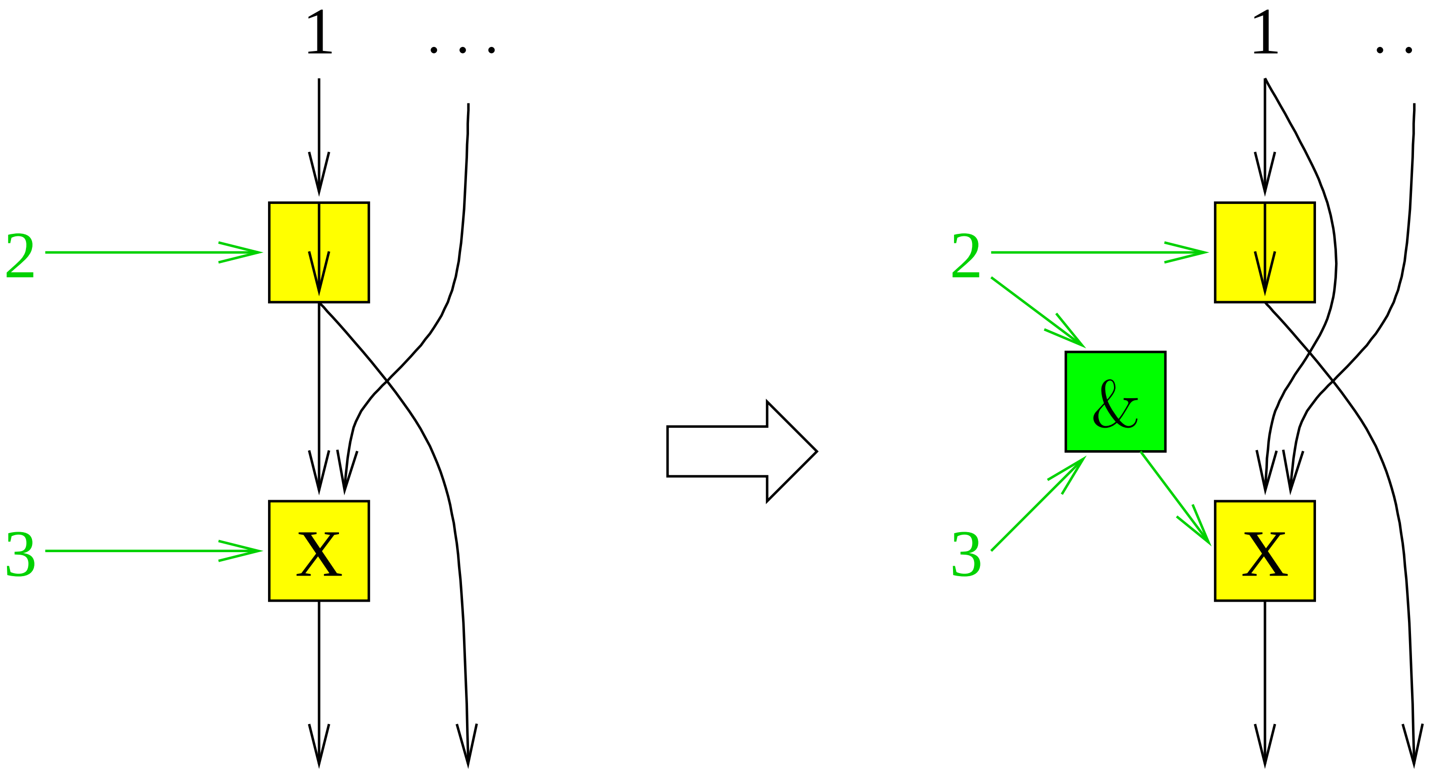


Some obvious simplifications



We can do dead code elimination afterwards.

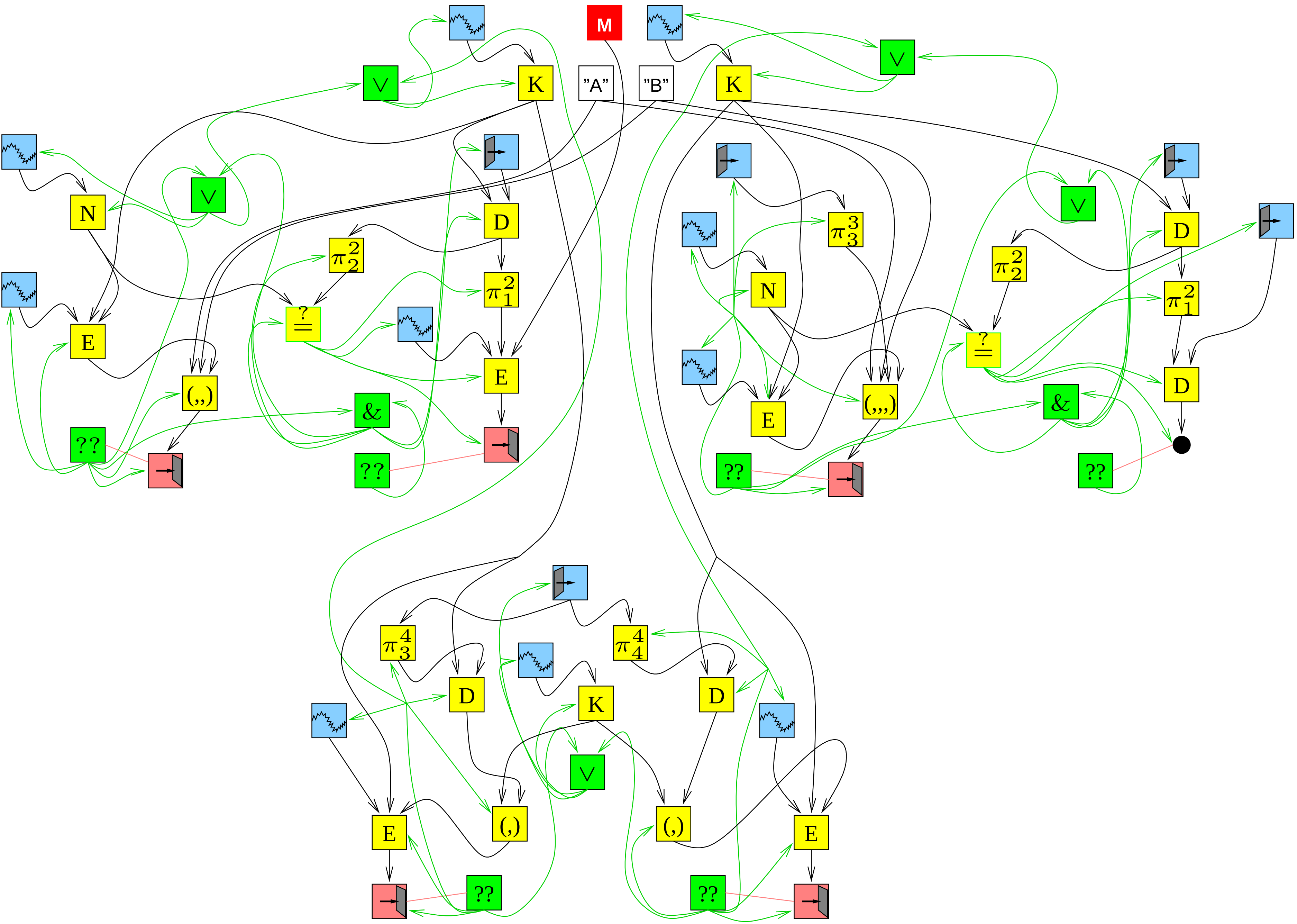
Some obvious simplifications



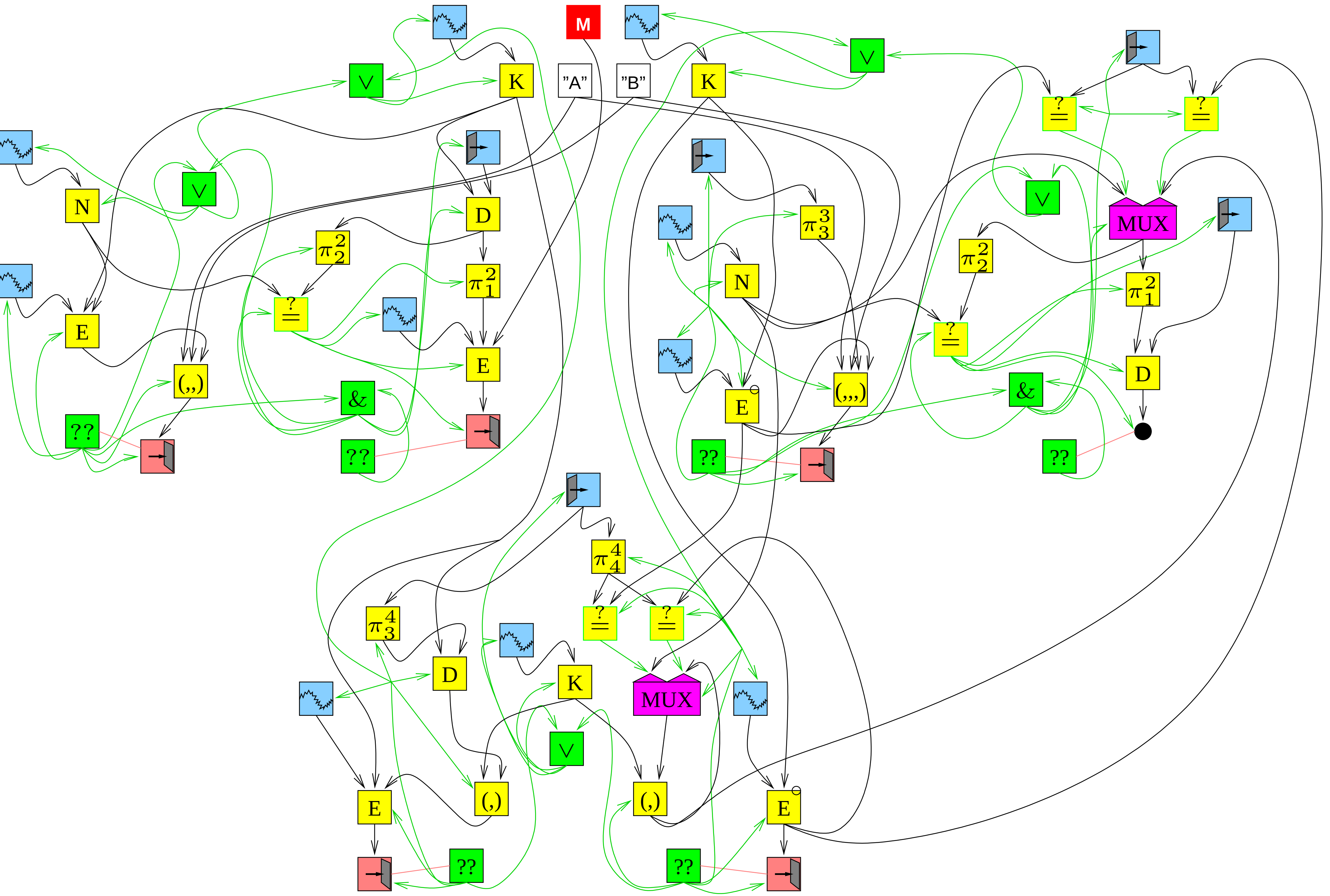
Simplifying encryption

- If the symmetric encryption is IND-CCA and INT-CTXT secure then we can replace the encryptions and decryptions as follows:
  - Encryptions — replace the plaintext with some constant $\mathbf{0}$.
  - Decryptions — replace them by
    - comparing the ciphertext with the results of all encryptions (with the same key);
    - if there is a match then take the corresponding (original) plaintext as the result;
    - if there is no match then fail.
- . . . provided that the key is used only for encrypting and decrypting.

Which keys are OK?



Replace K_{BS}



Semantics

- Let $\{0,1\}^*_\perp = \{\perp\} \cup \{0,1\}^*$ where $\perp$ is the smallest value and everything else is incomparable.
- Let $\mathbb{B} = \{\mathsf{false}, \mathsf{true}\}$ with $\mathsf{false} \leq \mathsf{true}$.
- Let V_D [resp. V_B] be the set of nodes returning bit-strings [booleans].
- The adversary may set the values of input nodes (but only moving upwards).
- The environment sets the randomness sources.
- The values of other nodes are monotonically computed from their inputs.

Semantic functions

- All yellow nodes are strict.
- Green nodes are monotone boolean operations.
- The value of blue nodes is not $\perp$ only if the incoming control dependency edge carries true.
- The MUX works as follows:
  - If the control dependency is false, or all guards are false, then the result is $\perp$.
  - Otherwise, if exactly one guard is true, then the result is the corresponding incoming value.
  - Otherwise, the result is $\top$.

Semantics

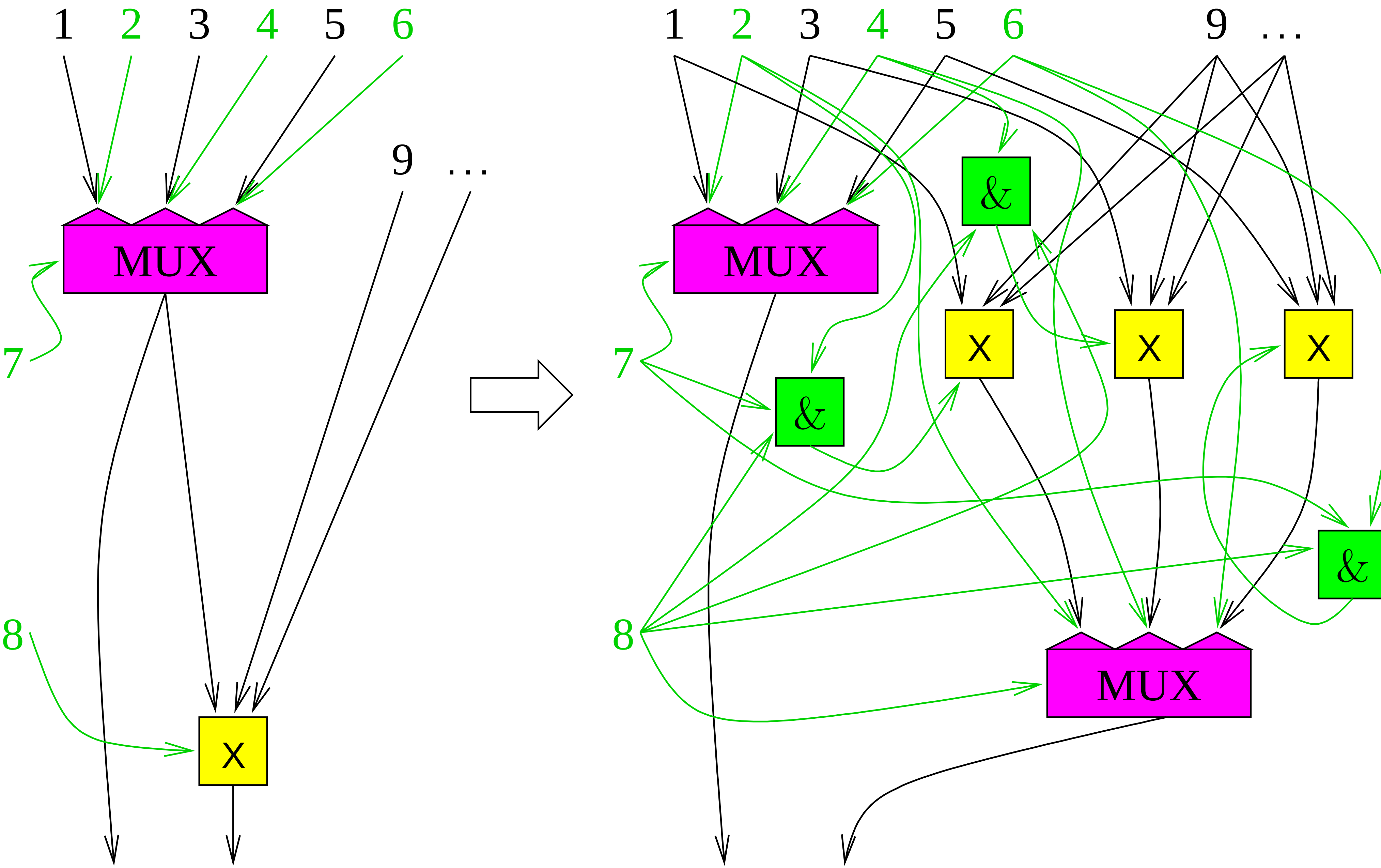
- The valuation of the entire graph has the type

$$\Big((V_D \longrightarrow \{0,1\}^*_\perp) \times (V_B \longrightarrow \mathbb{B})\Big)^\top \ .$$

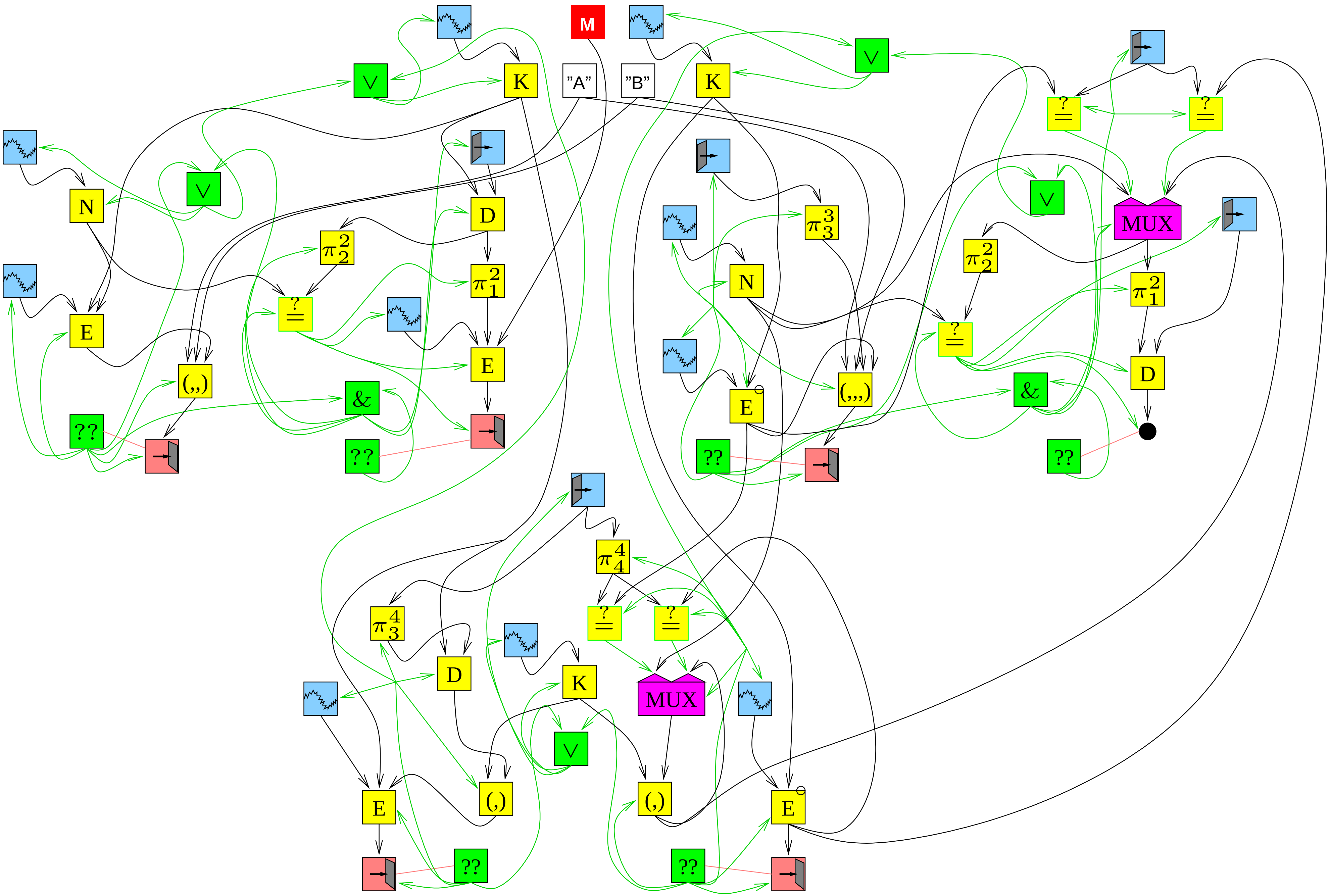
- The semantic functions of nodes define a monotone function on graph valuations.
- Its least fixed point is the semantics of the graph.

- A good thing — the order of the execution of nodes is not fixed.

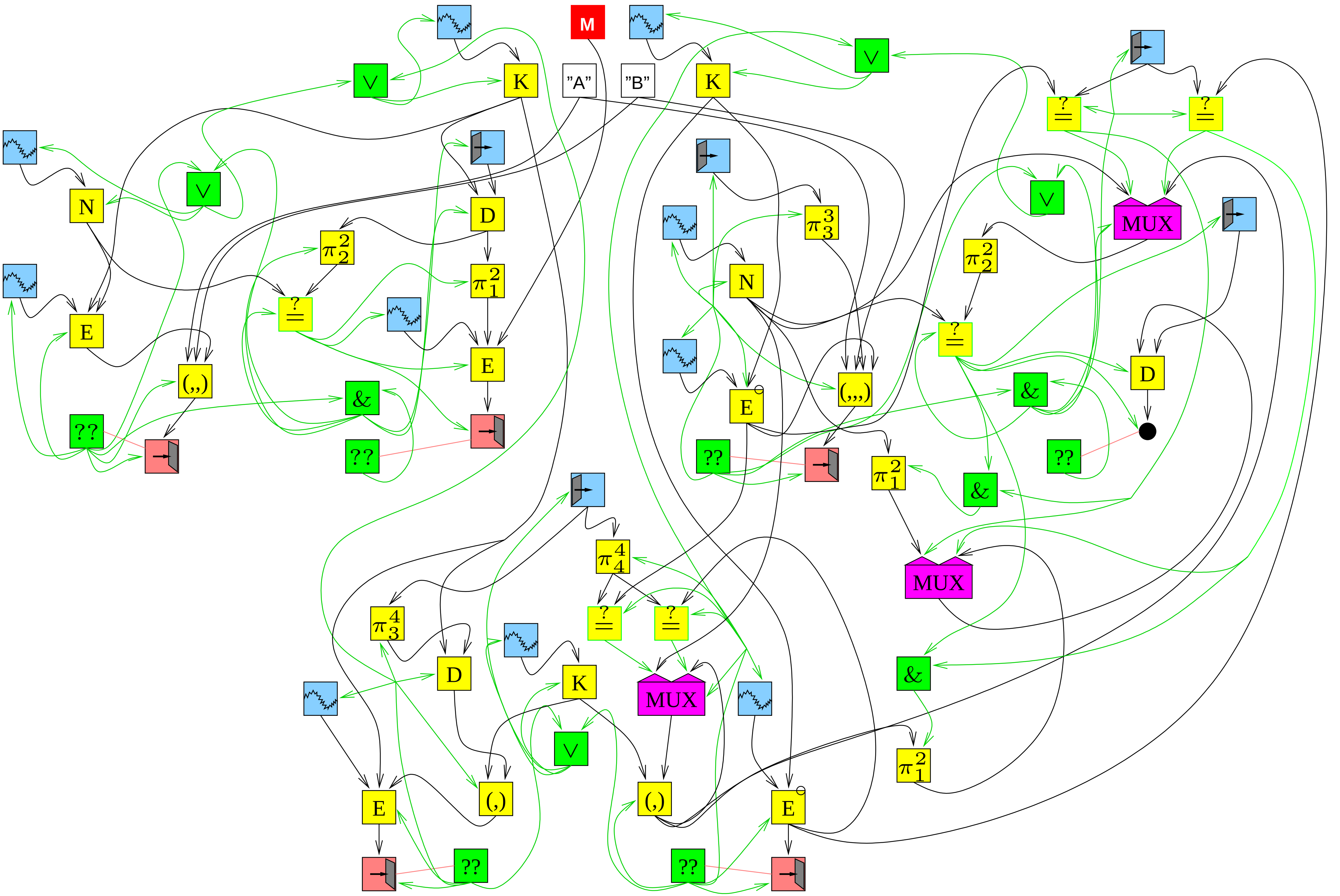
Computation $\leftrightarrow$ MUX



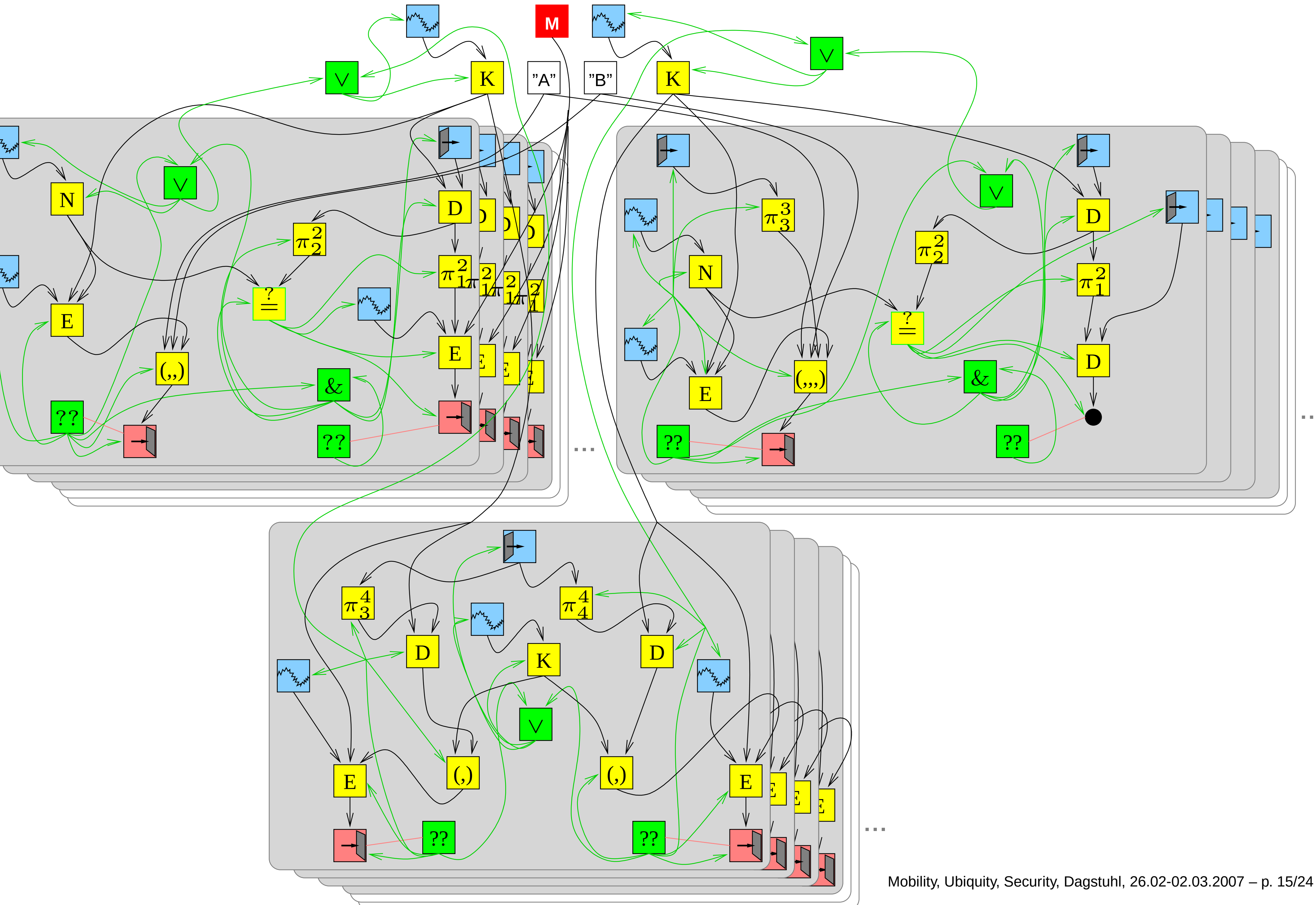
Application. . .



Application. . .



Replication



Representing infinite graphs

- Nodes in different planes, but in the same position are represented by a single node.
  - Such nodes are *one-dimensional*.
- There may be replication inside replication.
  - The corresponding nodes in the representation have more than one dimension.
- In the representation, the edges are equipped with *coordinate mappings*.
- In the representation, the edges generally cannot go from a higher-dimensional node to lower-dimensional node.
  - Exceptions: target node is an *infinite or* or MUX.
  - Then we record which dimensions are contracted.

Arguing about control

- In our experience, the hardest part of the analyser has been the simplification of control dependencies.
  - Meaning: to derive that some node is always false.
- Some simplifications can be done locally.
  - Constant propagation, copy propagation, flattening, etc.
- More interesting ones require the analysis of the whole graph.

$\Longrightarrow$

- When does $v_1 = \mathsf{true}$ imply $v_2 = \mathsf{true}$?
  - If $v_1 = \ldots \,\&\, v_2 \,\&\, \ldots$.
  - If $v_2 = \ldots \vee v_1 \vee \ldots$.
  - If $v_1 \Rightarrow v_3$ and $v_3 \Rightarrow v_2$.
  - If $v_2 = w_1 \,\&\, \cdots \,\&\, w_t$ and $v_1 \Rightarrow w_i$ for all i.
  - If $v_1 = w_1 \vee \cdots \vee w_t$ and $w_i \Rightarrow v_2$ for all i.
- On the representation, we have to record coordinate equalities, too.
- If $v_1[c_1, \ldots, c_k] = \bigvee_{j \in \mathbb{N}} v_2[c_1, \ldots, c_k, j]$ then also

$$v_1[c_1, \ldots, c_k] \Rightarrow \mathrm{OneOf}(v_2[c_1, \ldots, c_k, *]) \quad .$$

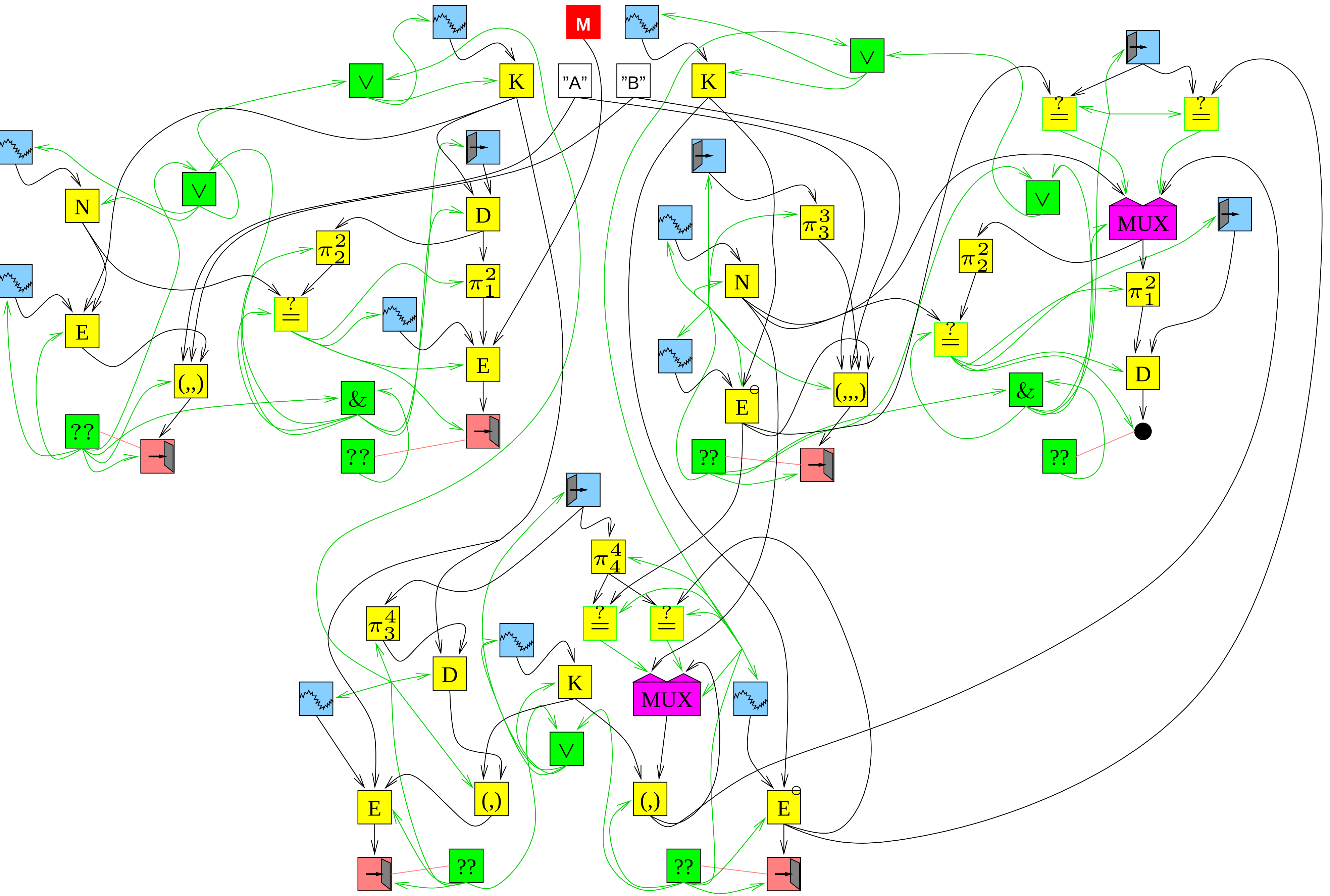
Using $\Rightarrow$

- Simplification of control dependencies.
- If the control dependency of some node u computing $X(\ldots, v, \ldots)$ implies that the node "$v \stackrel{?}{=} w$" is `true` then replace u with $X(\ldots, \mathrm{Merge}(v, w), \ldots)$.
  - In this way we record the equality of values in the graph.
- If the control dependency of some MUX implies the guard of some of its choices, then replace that MUX by that choice.

Independence and randomness

- Consider the ancestors of some node.
  - Move backwards in the dependency graph.
  - Also move from “receive”-s to “send”-s.
    - But not towards the future. (use $\Rightarrow$)
- If two nodes have non-overlapping sets of ancestors then they are *independent*.
- If at least one of them is random, then they are unequal.
- Typical application:
  - A nonce is generated but never sent out.
  - It is compared with some of the contents of some message received from the network.
  - Then the result must be false.

Our dependency graph...



NAND

- For certain two boolean nodes we can say that at most of them can be true at any moment.
- This can be propagated downwards:
  - If $v_1 \,\overline{\&}\, v_2$ and $v_3 = \ldots \& \, v_2 \, \& \ldots$ then $v_1 \,\overline{\&}\, v_3$.
  - If $v_2 = w_1 \vee \cdots \vee w_t$ and $v_1 \,\overline{\&}\, w_i$ for all i then $v_1 \,\overline{\&}\, v_2$.
- Also store coordinate equalities and exceptions to them.
- If we derive $v \,\overline{\&}\, v$ then v is false.

Integrity (correspondence) properties

- $Begin$- and End-nodes.
  - Have a control dependency and an incoming data edge.
  - Produce no output.
- Non-injective agreement — Whenever $End(x)$ is executed, $Begin(x)$ must have been executed as well.
  - ...earlier or at the same time.
  - Use "$\Rightarrow$" to show it.
- Injective agreement — each execution of $End(x)$ has a different execution of $Begin(x)$ not later than it.
  - Often $End(x)$ can happen at most once for each x.
  - Use "NAND" to show it.

In closing...

- For tracking data dependencies, our representation seems to be ideal.
- Control dependencies are also handled seemingly reasonably.
  - One can consider more or less stringent control flow structures, but the current choice looks like optimal.
- A persistent representation has to be found for data collected for $\Rightarrow$ and NAND.