Some Generalised Reductions of Ordered Binary Decision Diagramm (GroBdd)

Joan Thibault

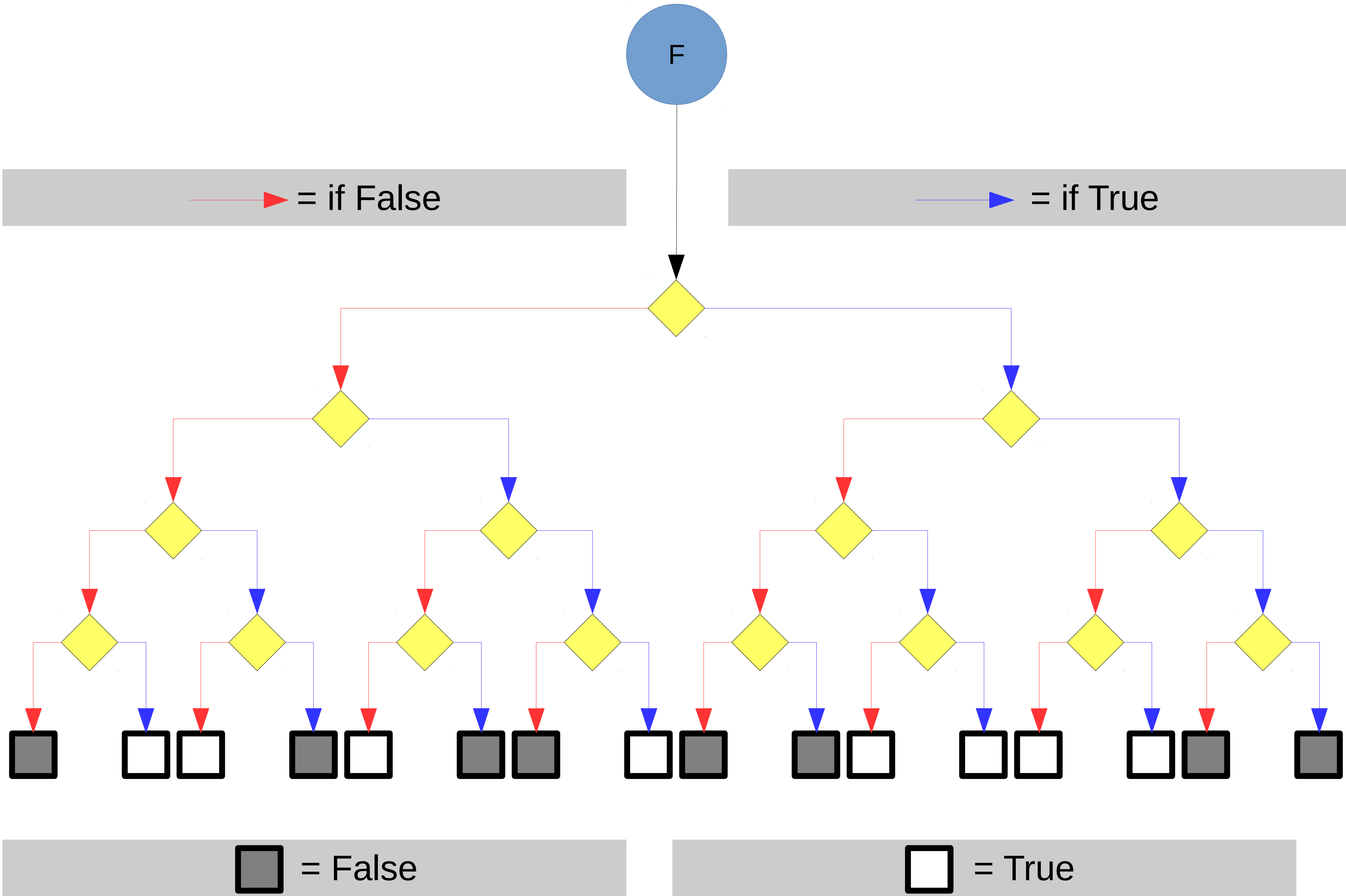
Boolean Functions

- Why ?
  - Computer Aided Design (e.g. digital circuit synthesis)
  - Knowledge Representation (e.g. Artificial Intelligence)
  - Combinatorial Problems (e.g. N-Queens problem)
- What ?
  - Compact representation
  - Operations (e.g. composing, concatening, evaluation)
  - Operators (e.g. AND, XOR, ITE, NOT)
  - Reductions (e.g. quantification, partial evaluation, SAT)

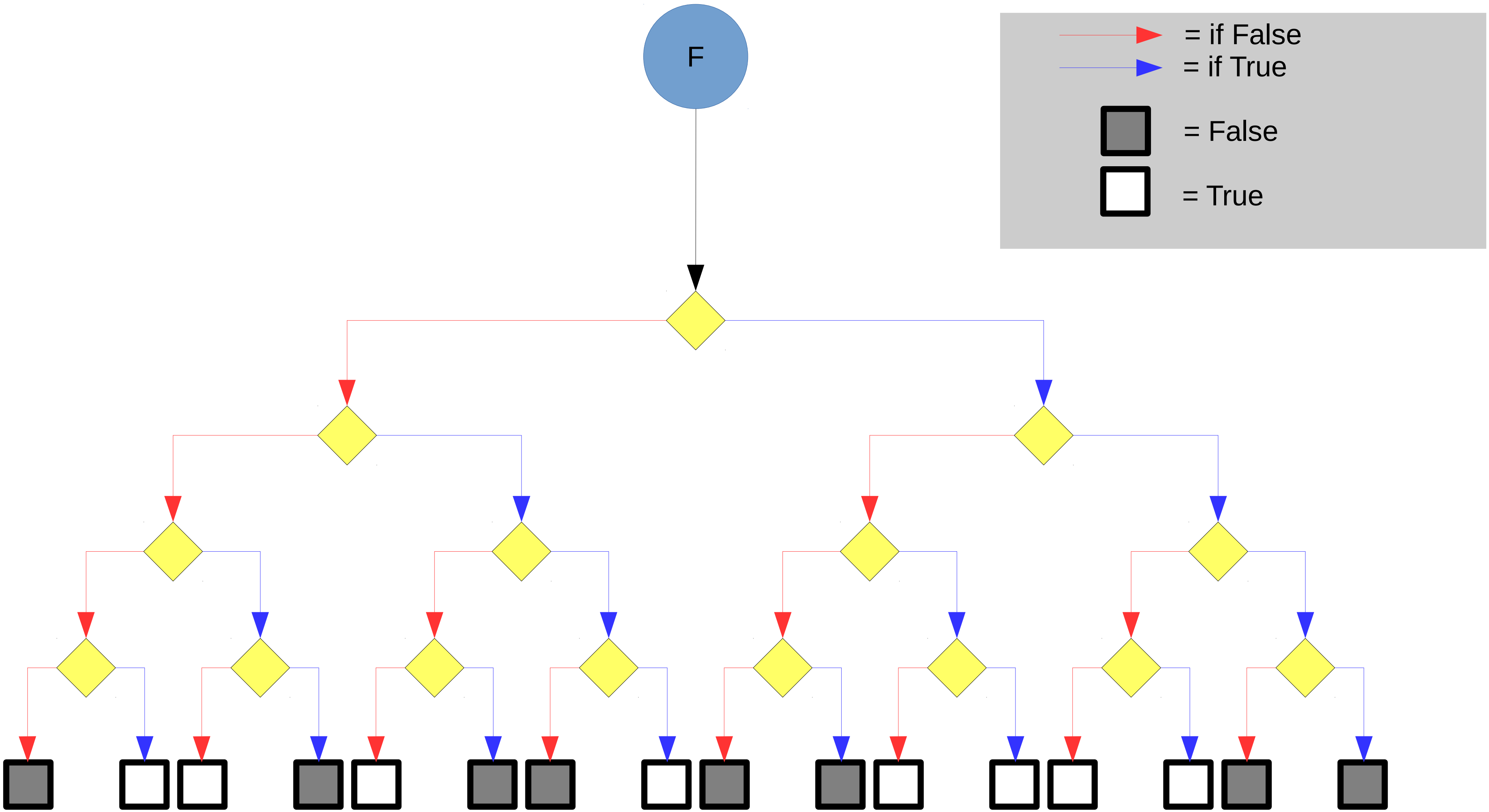
Boolean Functions

- Various representations
  - Truth Table
  - Conjonctive / Disjonctive Normal Form
  - And Inverter Graph
  - Binary Decision Diagramm
    - Reduced Ordered BDD
    - Zero supressed BDD
    - Xor based BDD

Section 1
What is a ROBDD ?

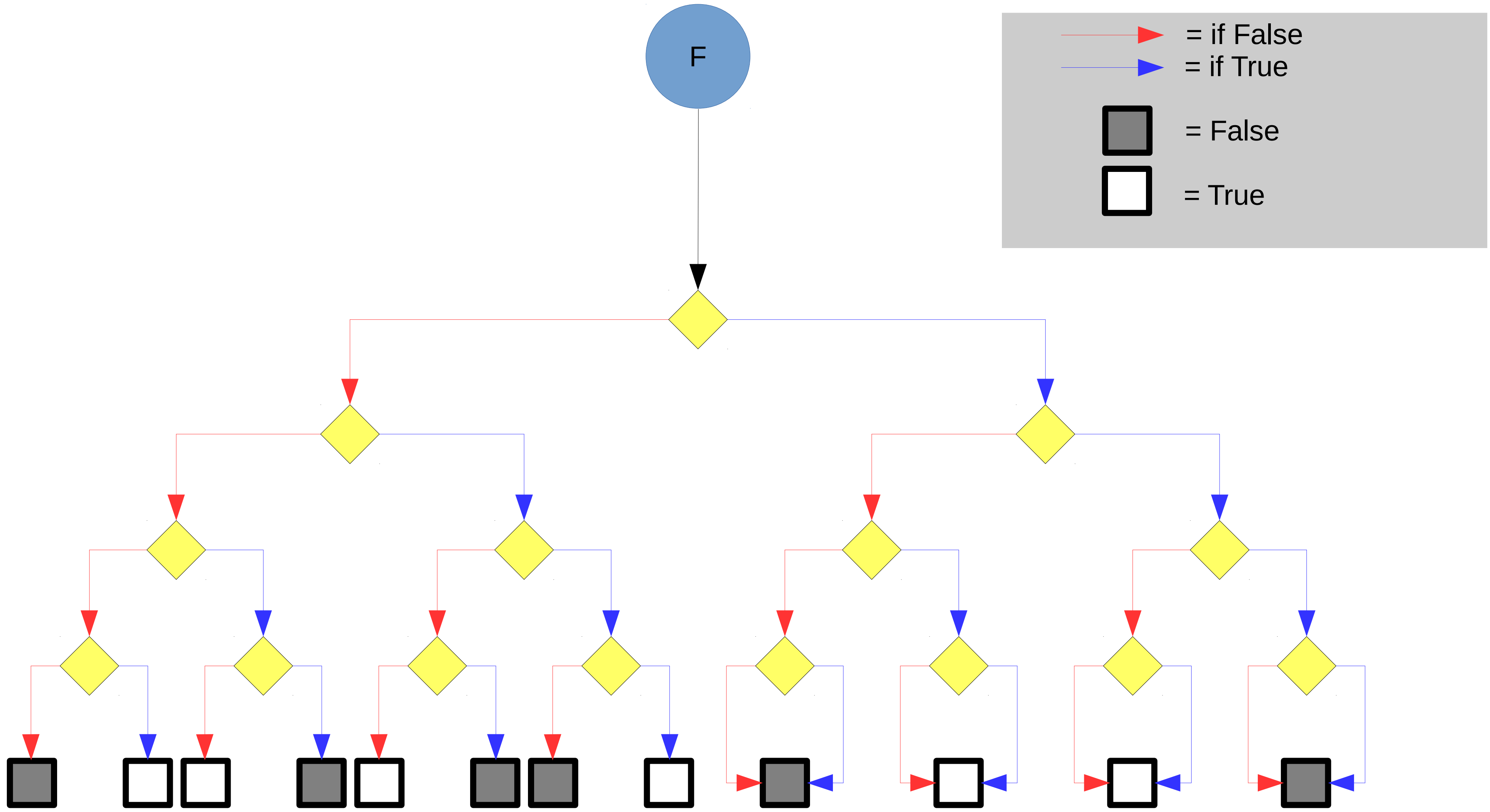

F
= if False
= if True
= False
= True

(Bryant) Step 1: we merge isomorphic sub-graphs



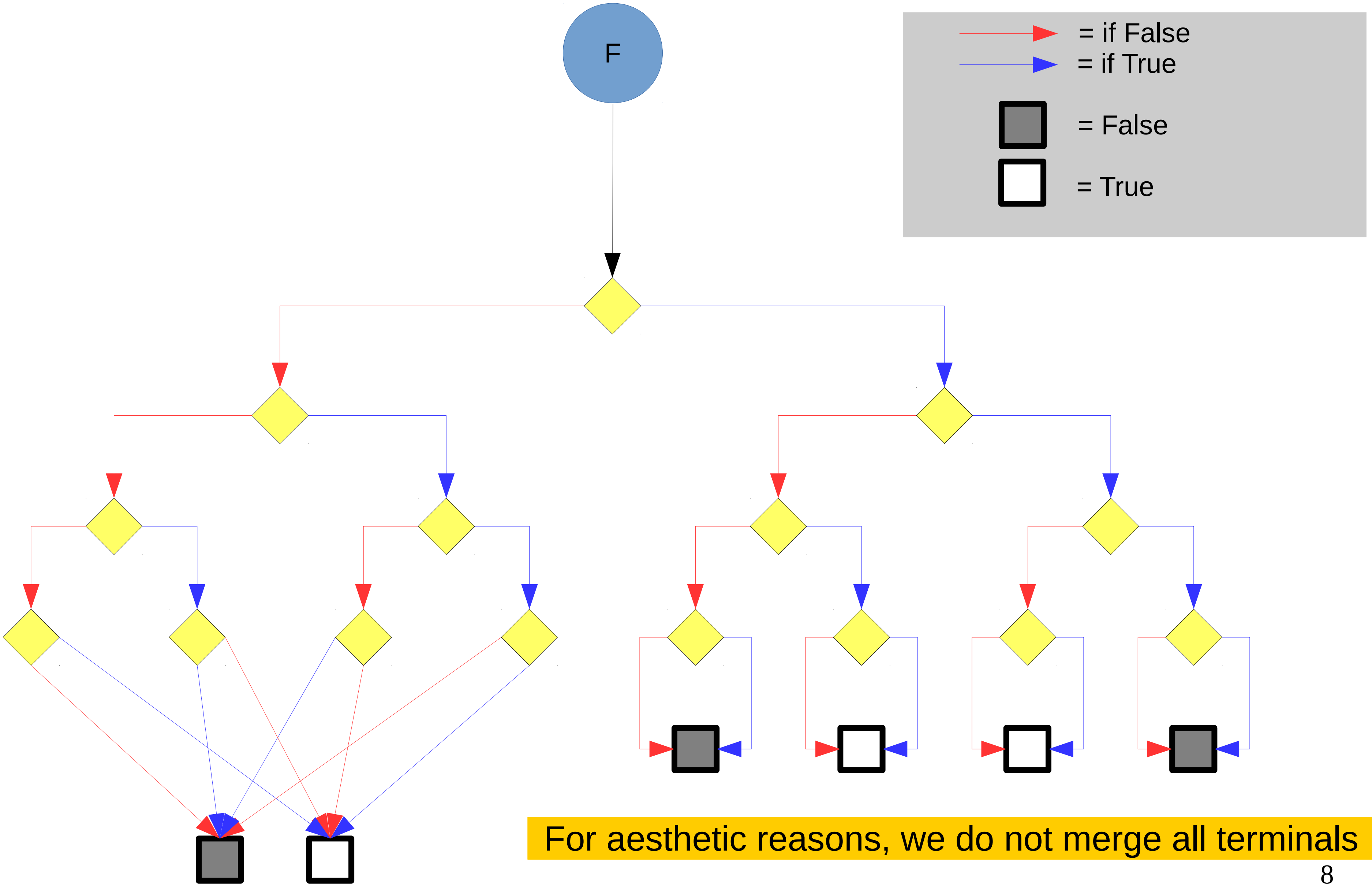
For aesthetic reasons, we do not merge all terminals

(Bryant) Step 1: we merge isomorphic sub-graphs



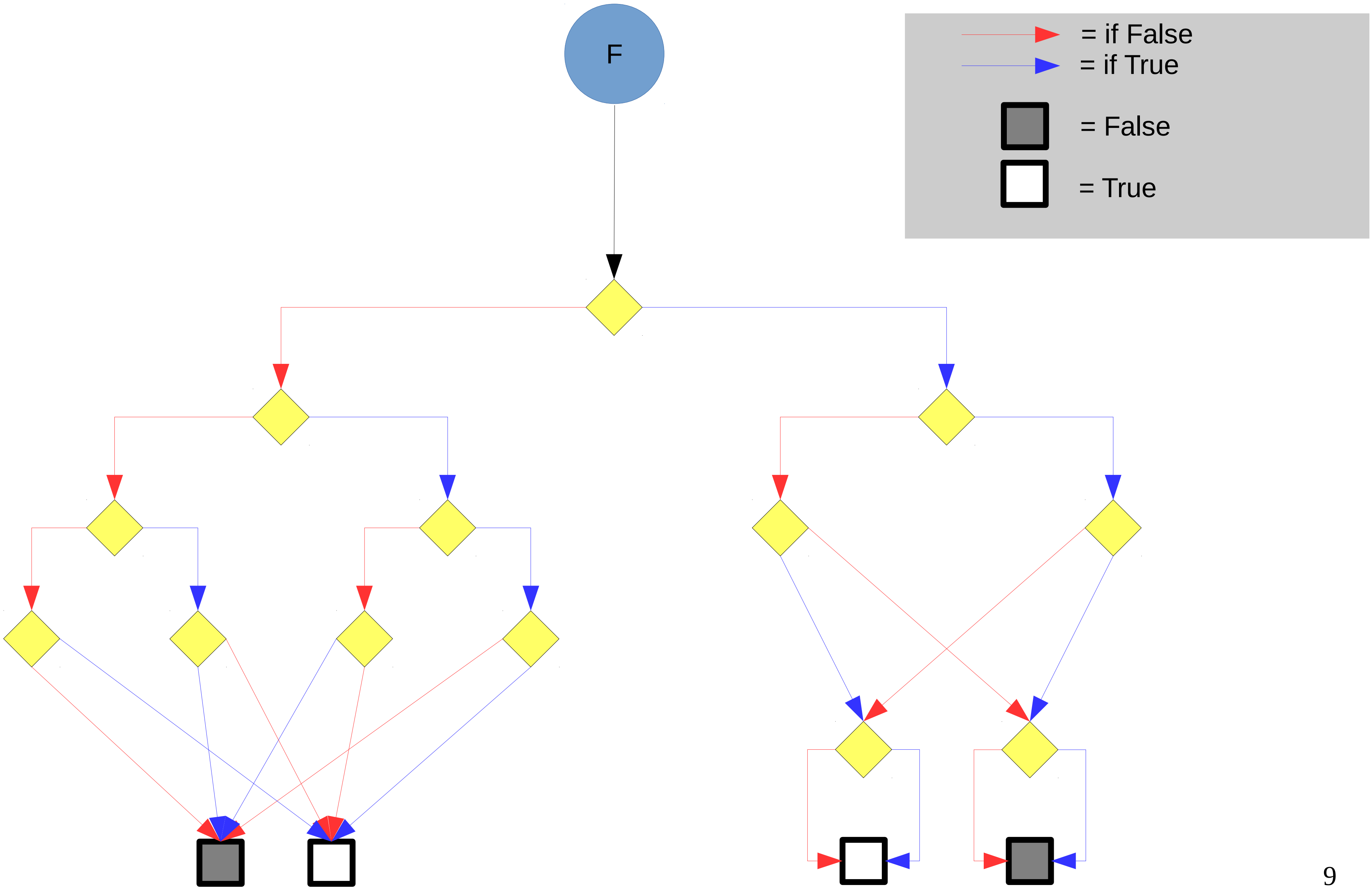
For aesthetic reasons, we do not merge all terminals

(Bryant) Step 1: we merge isomorphic sub-graphs

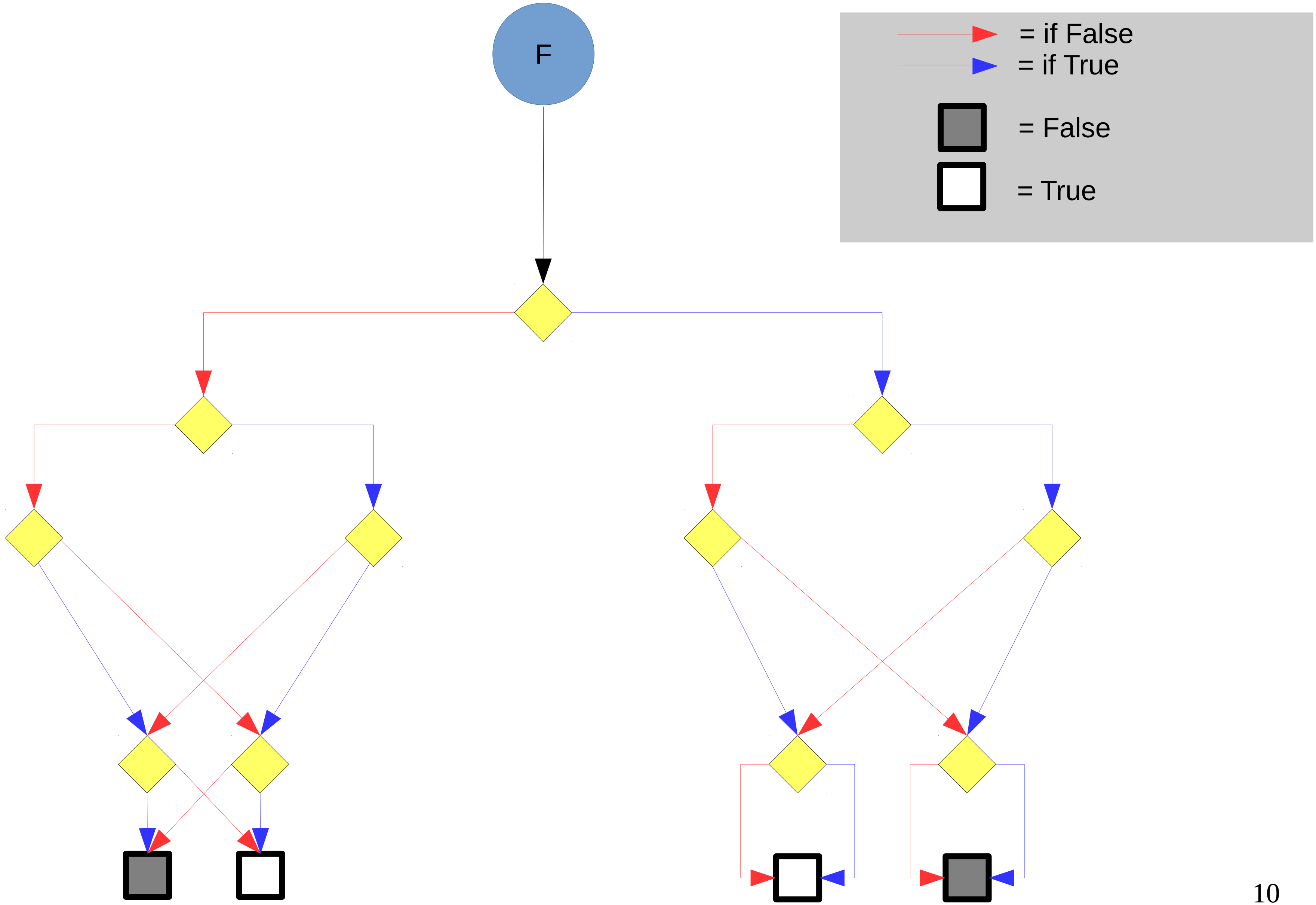


For aesthetic reasons, we do not merge all terminals

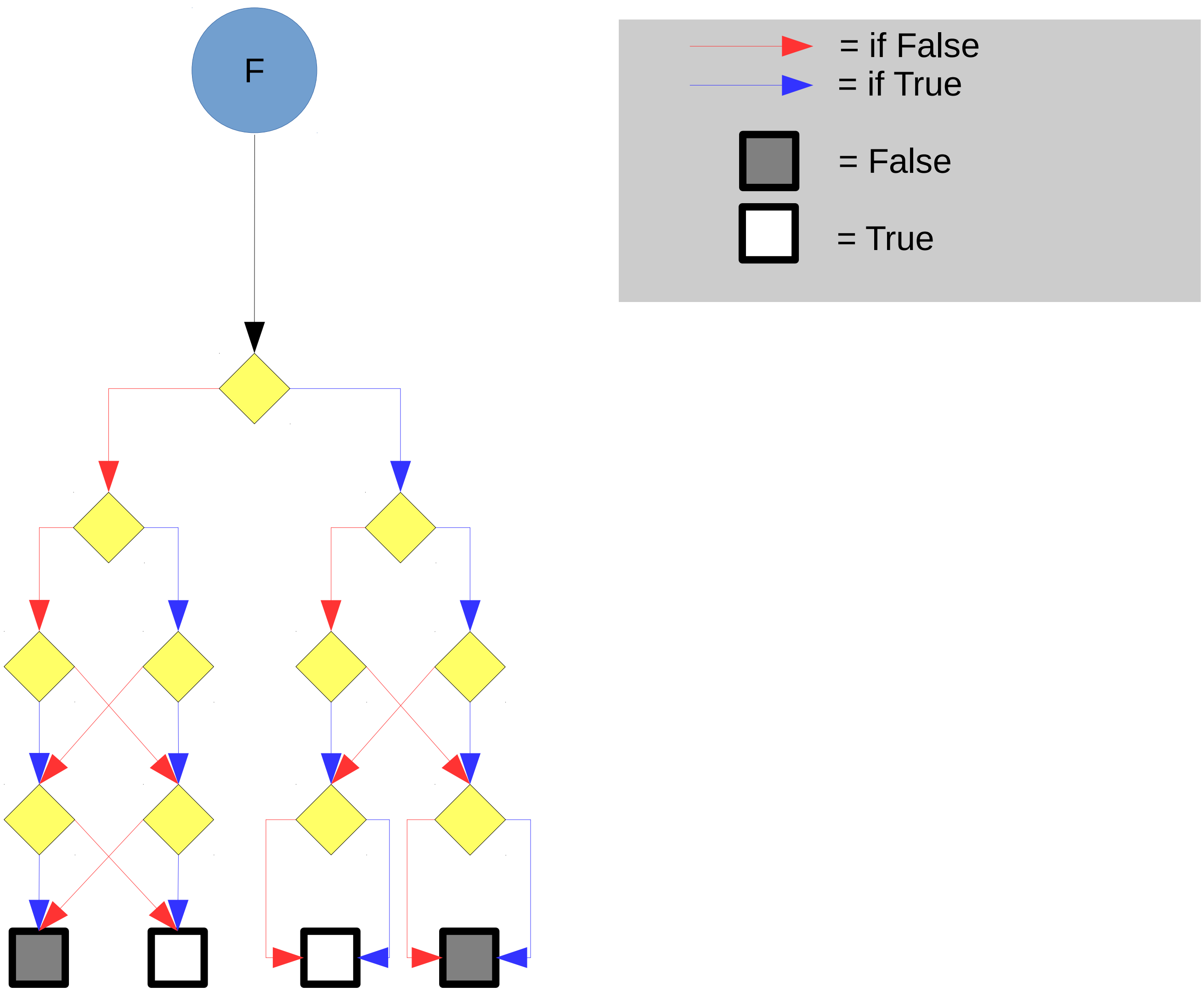
(Bryant) Step 1: we merge isomorphic sub-graphs



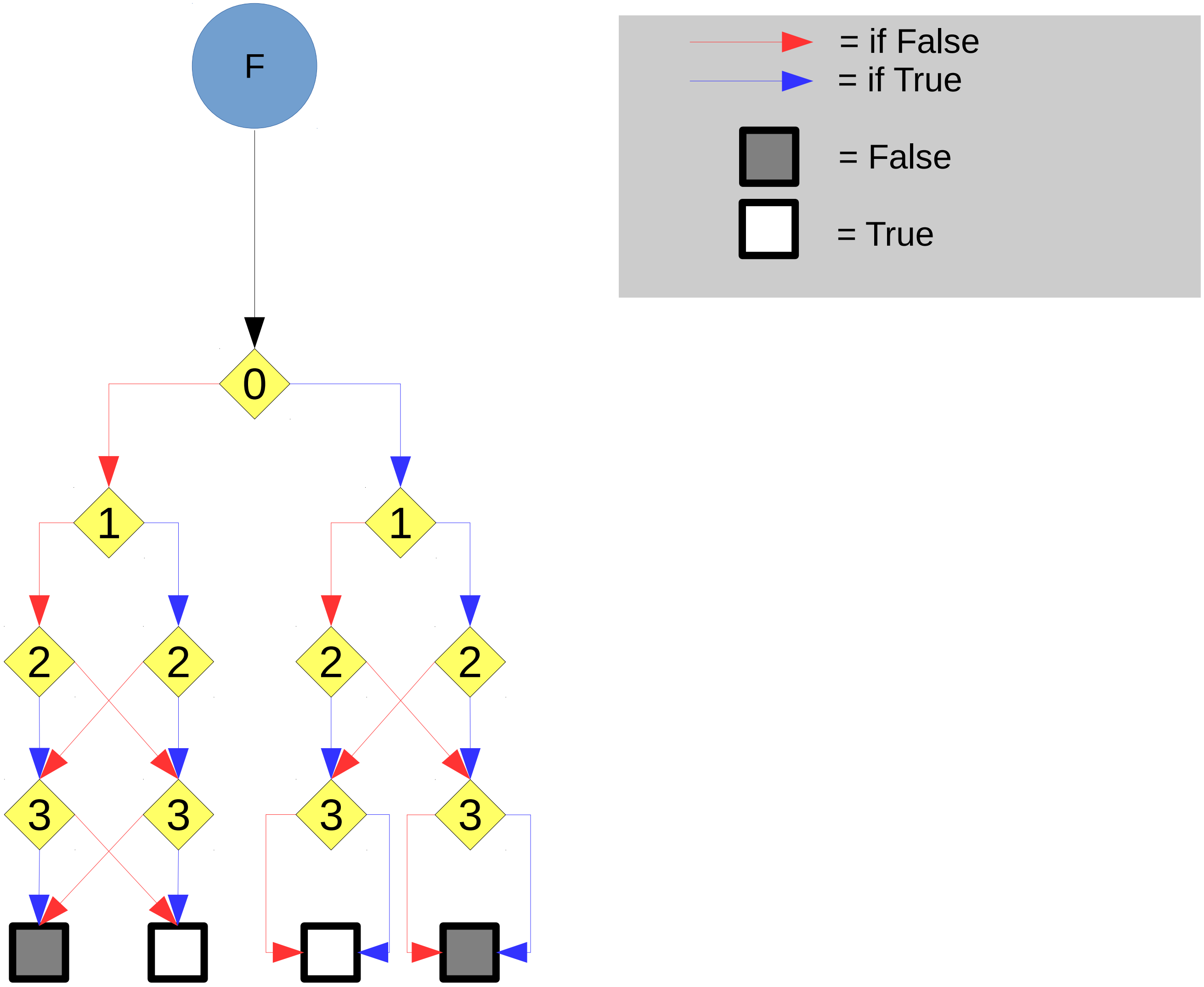
(Bryant) Step 1: we merge isomorphic sub-graphs



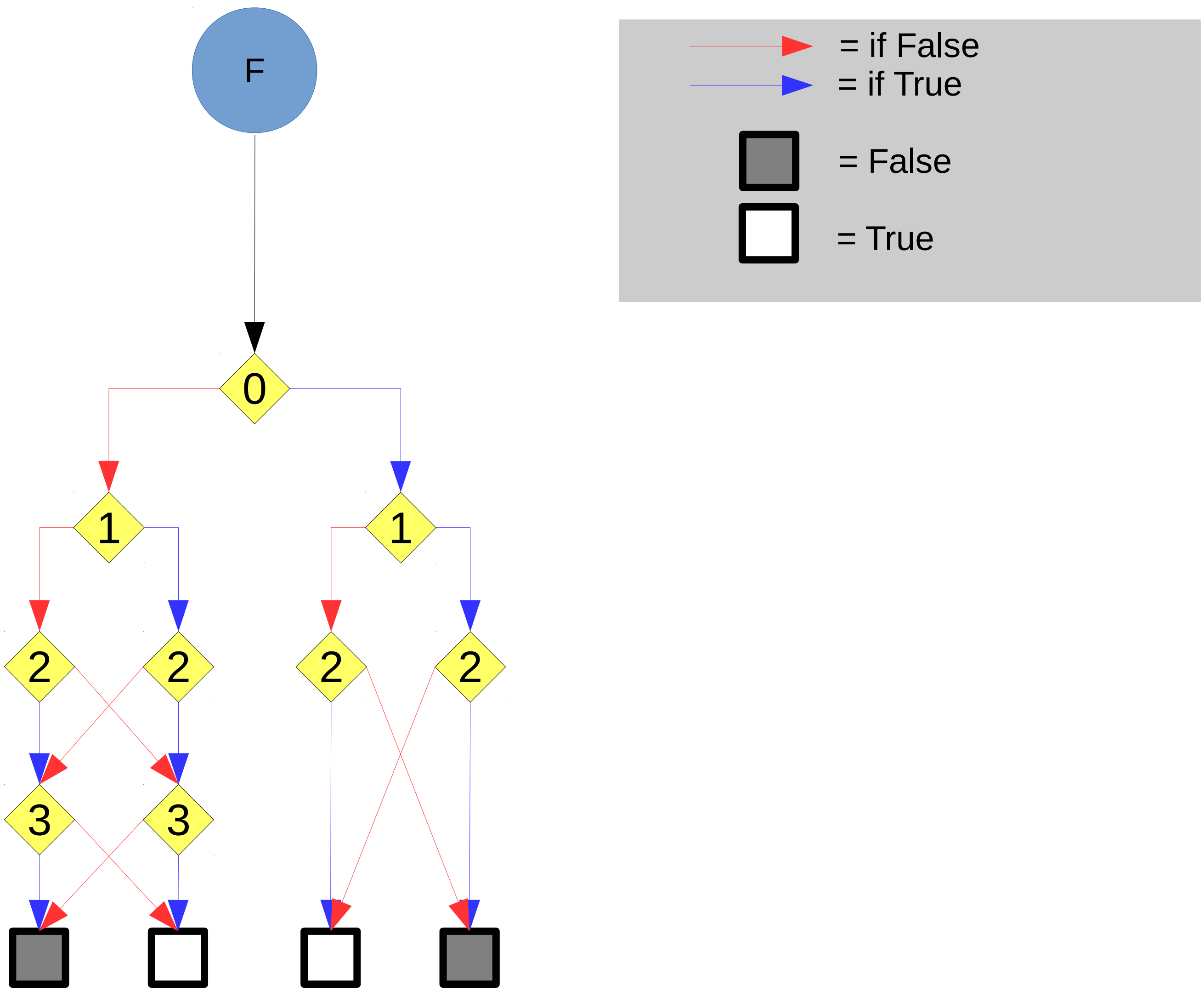
(Bryant) Step 1: we merge isomorphic sub-graphs



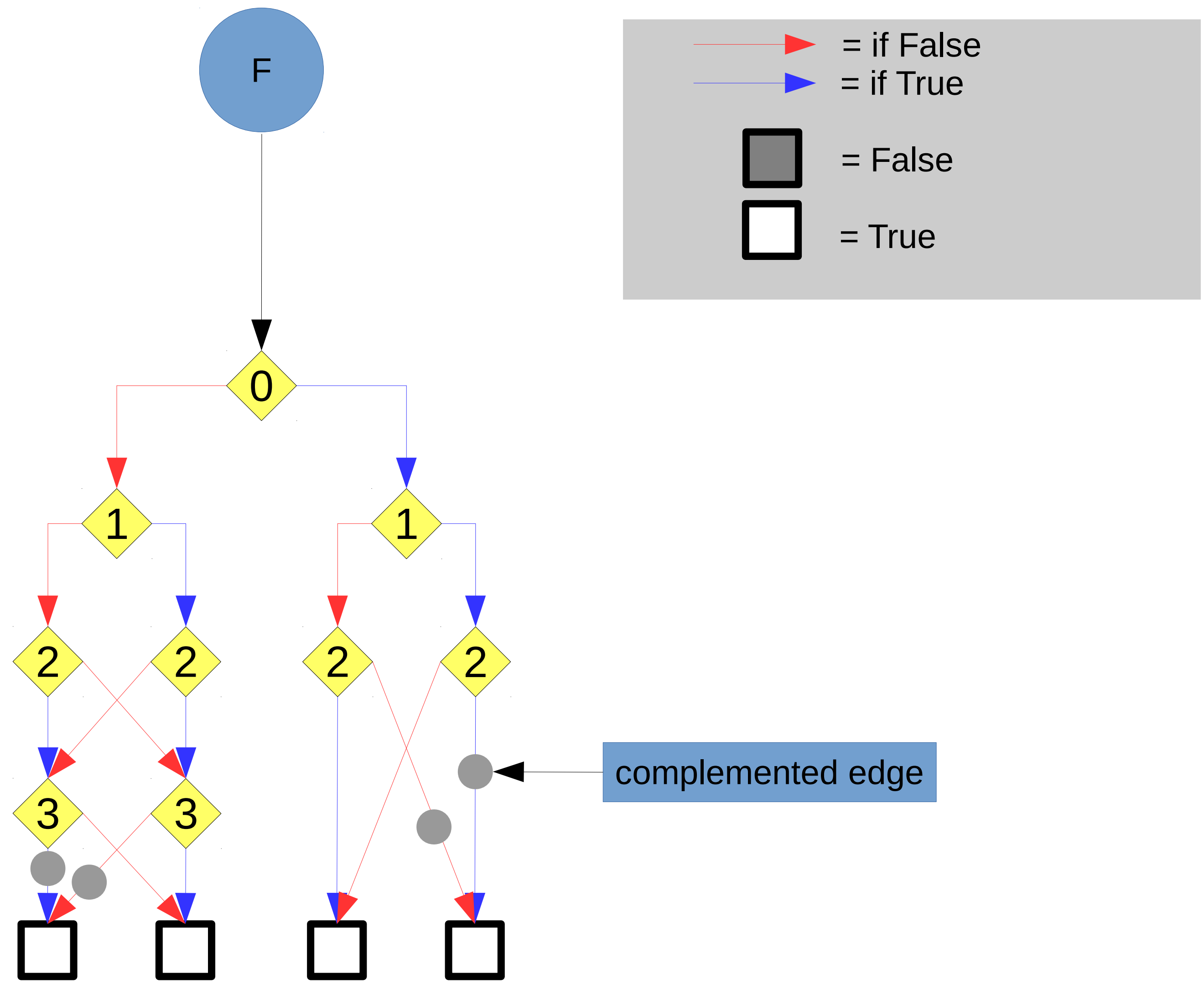
(Bryant) Step 2: we specify for each node: on which variable the decision is made



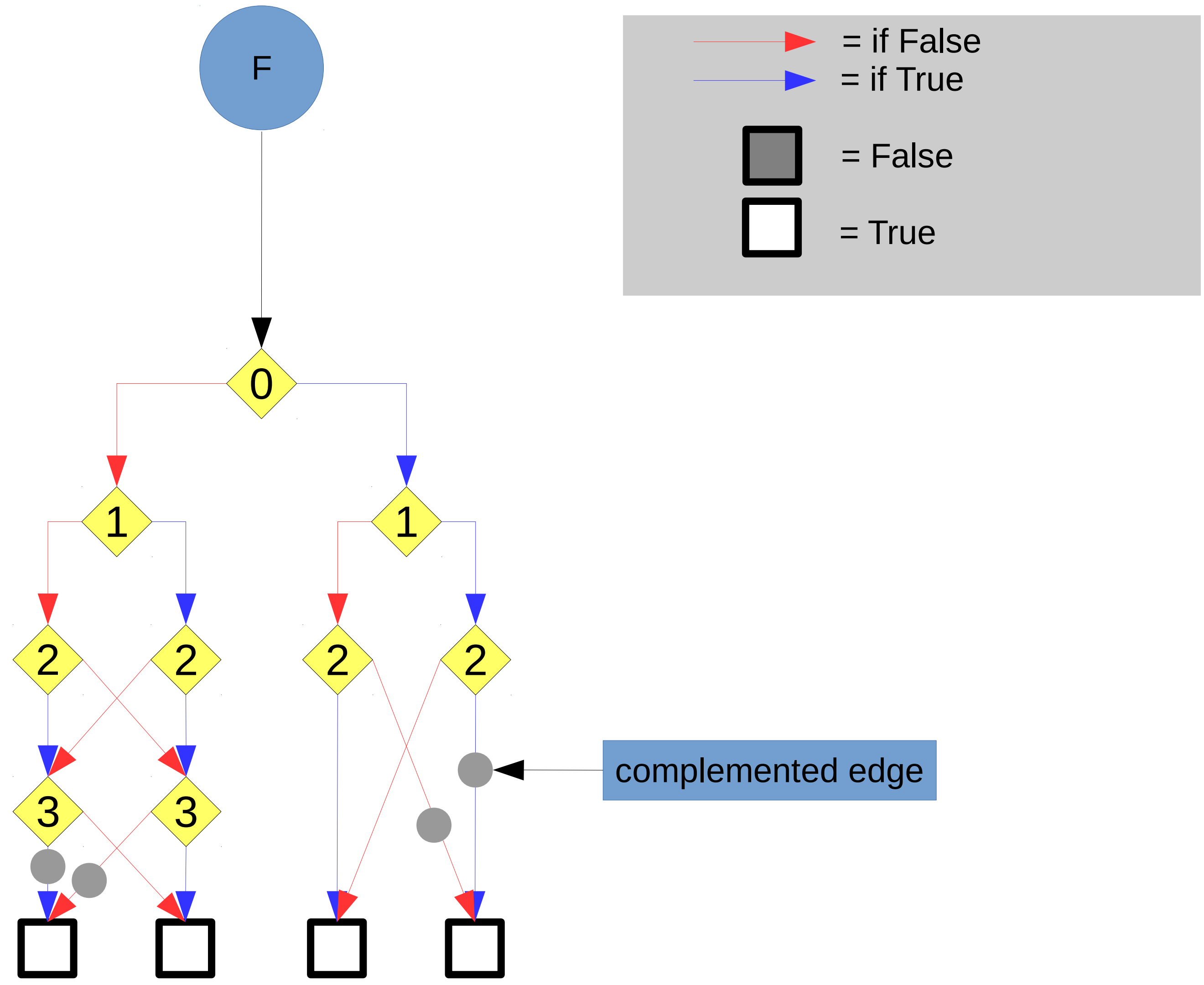
(Bryant) Step 3: we remove useless decisions



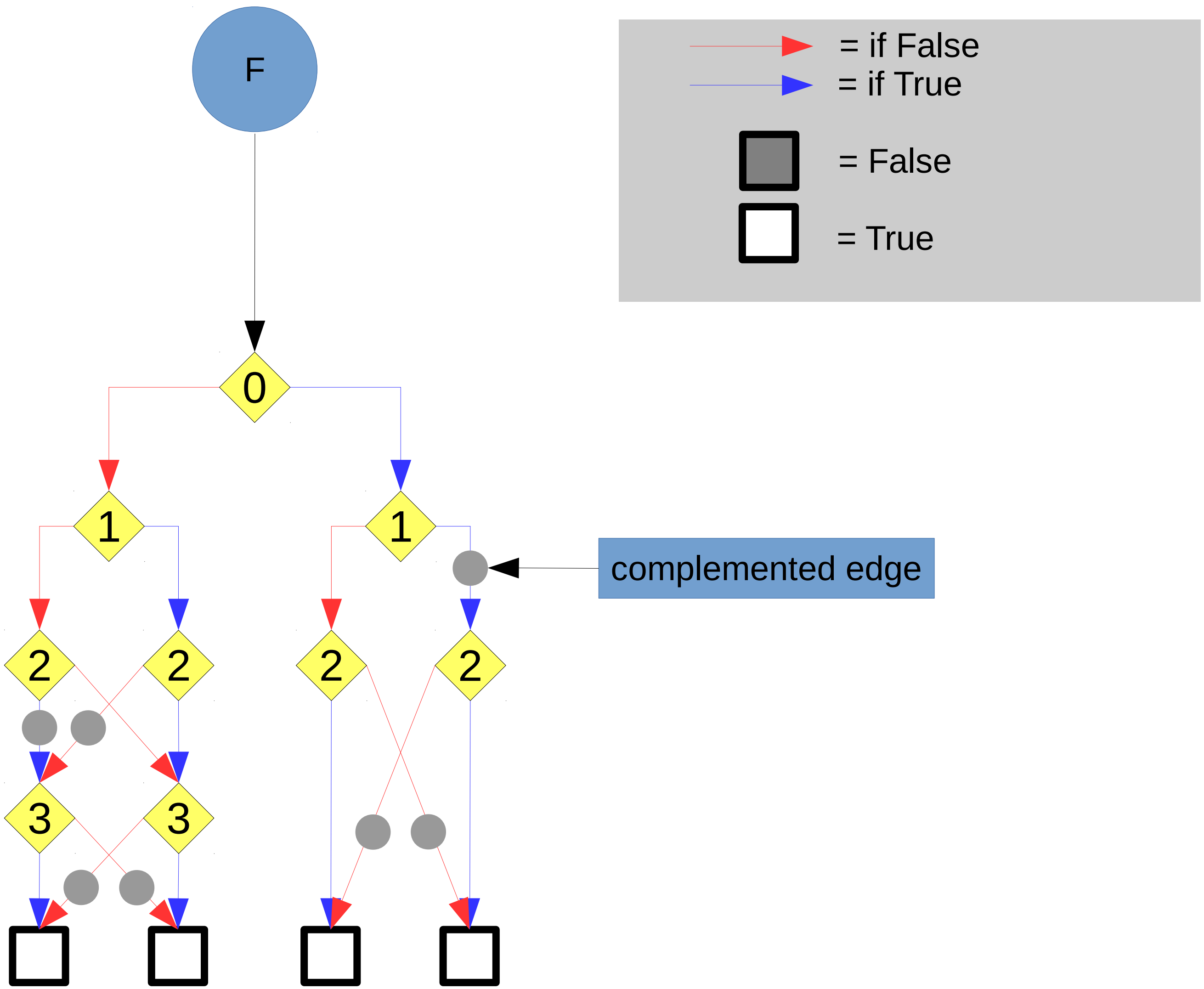
(Complemented Edges) Step 1: we replace the False node by a complemented edge to True



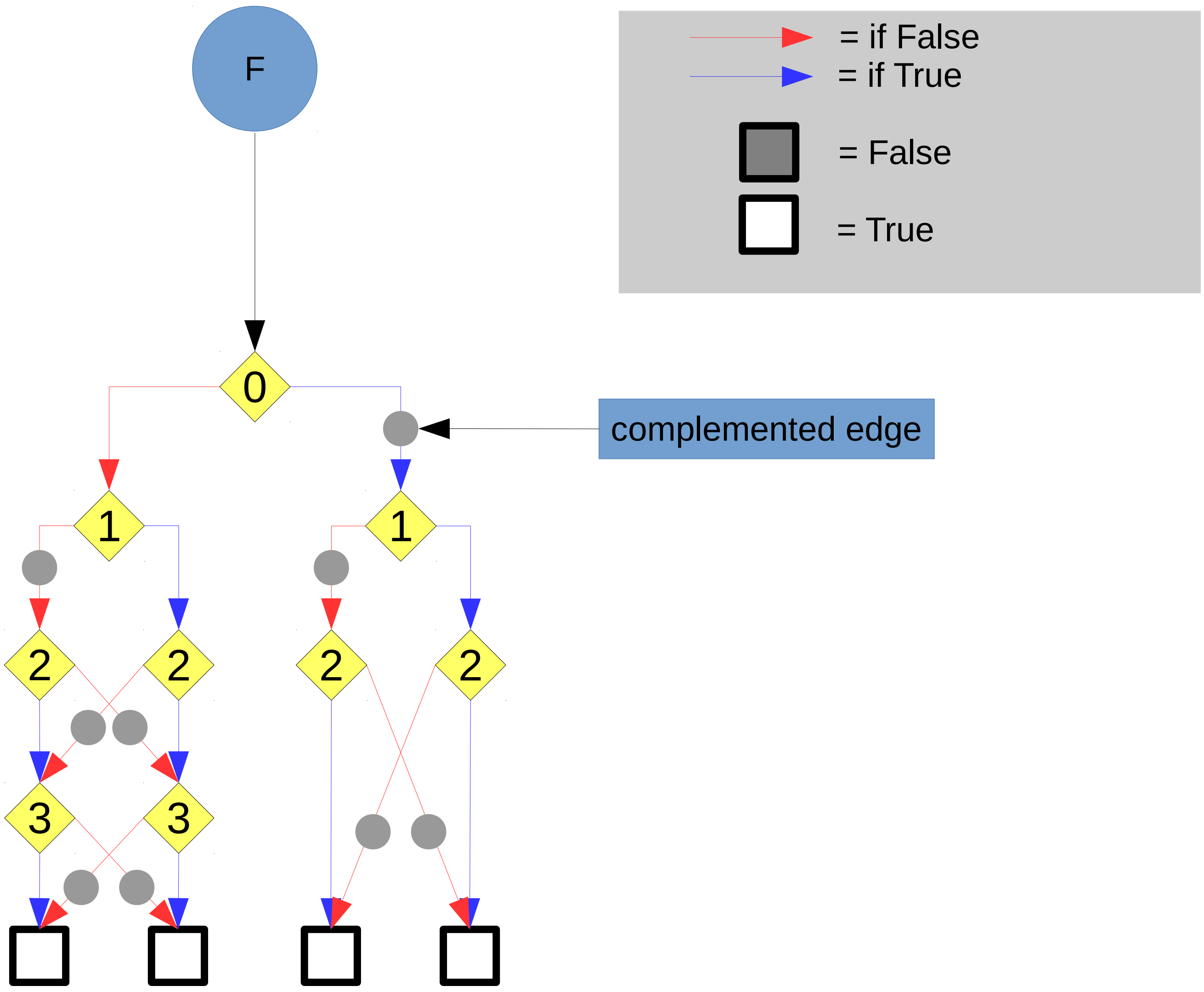
(Complemented Edges) Step 2 : we propagate inverted edges upward, ensuring that no “if True” edge is complemented



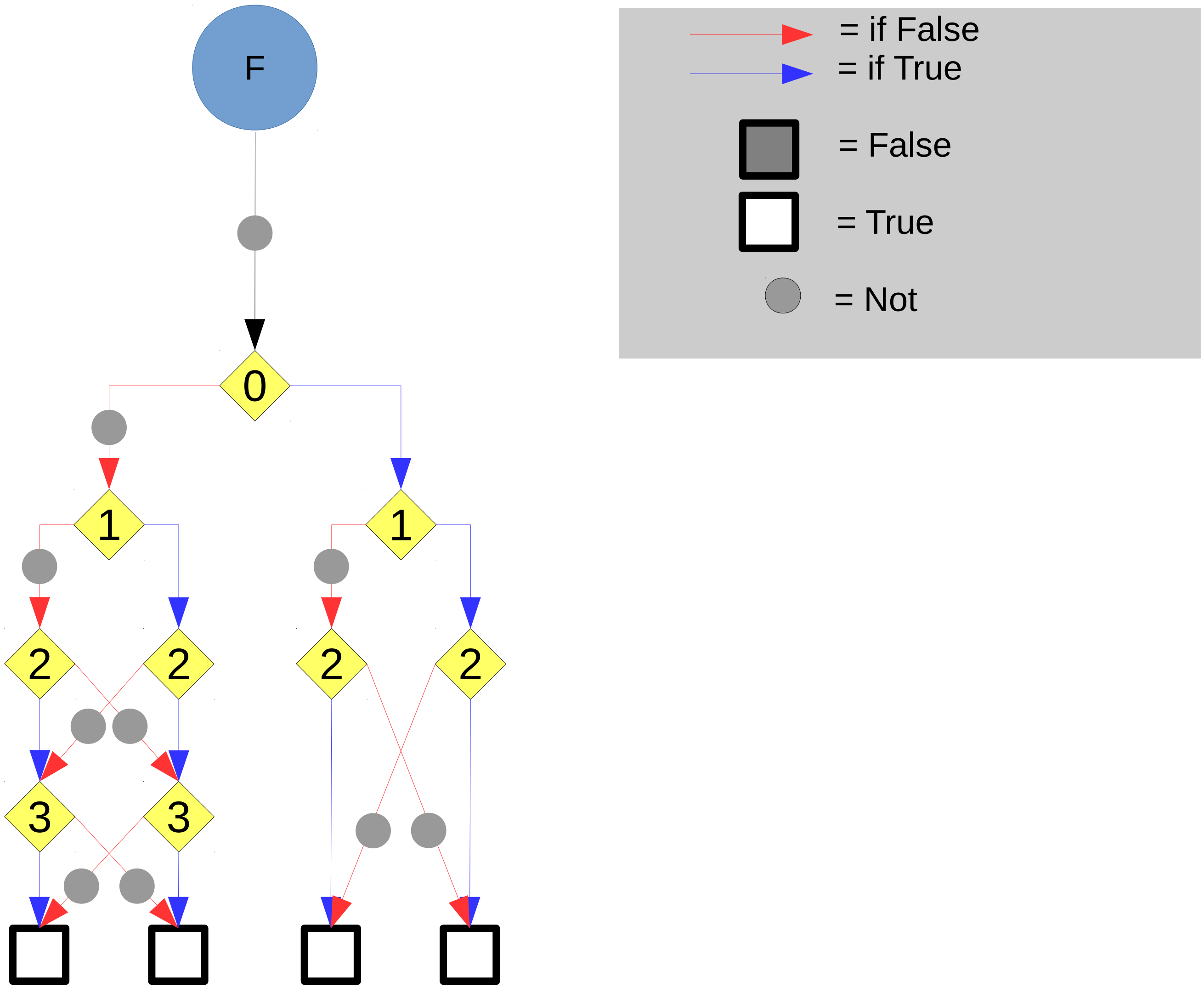
(Complemented Edges) Step 2 : we propagate inverted edges upward, ensuring that no “if True” edge is complemented



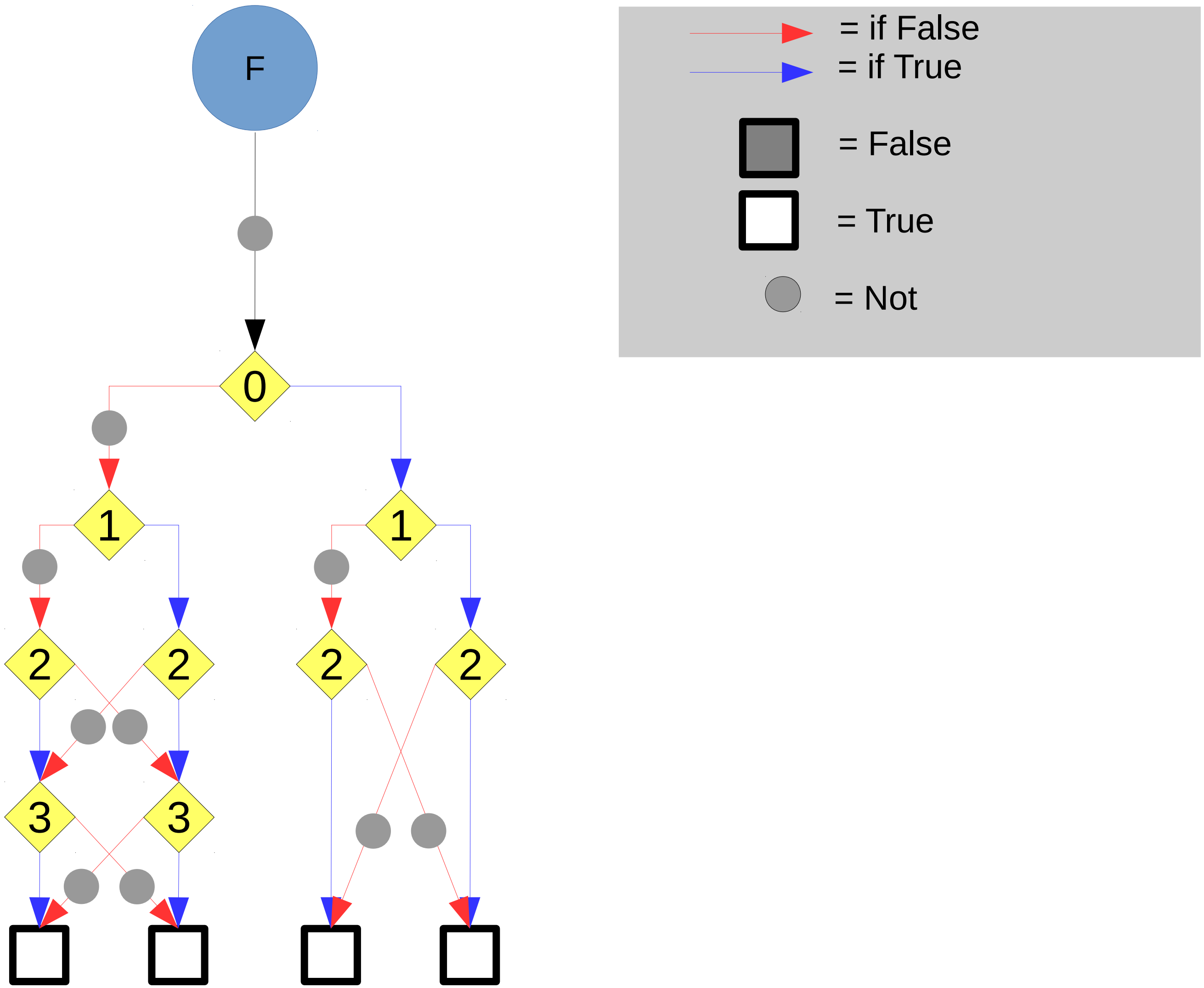
(Complemented Edges) Step 2 : we propagate inverted edges upward, ensuring that no “if True” edge is complemented



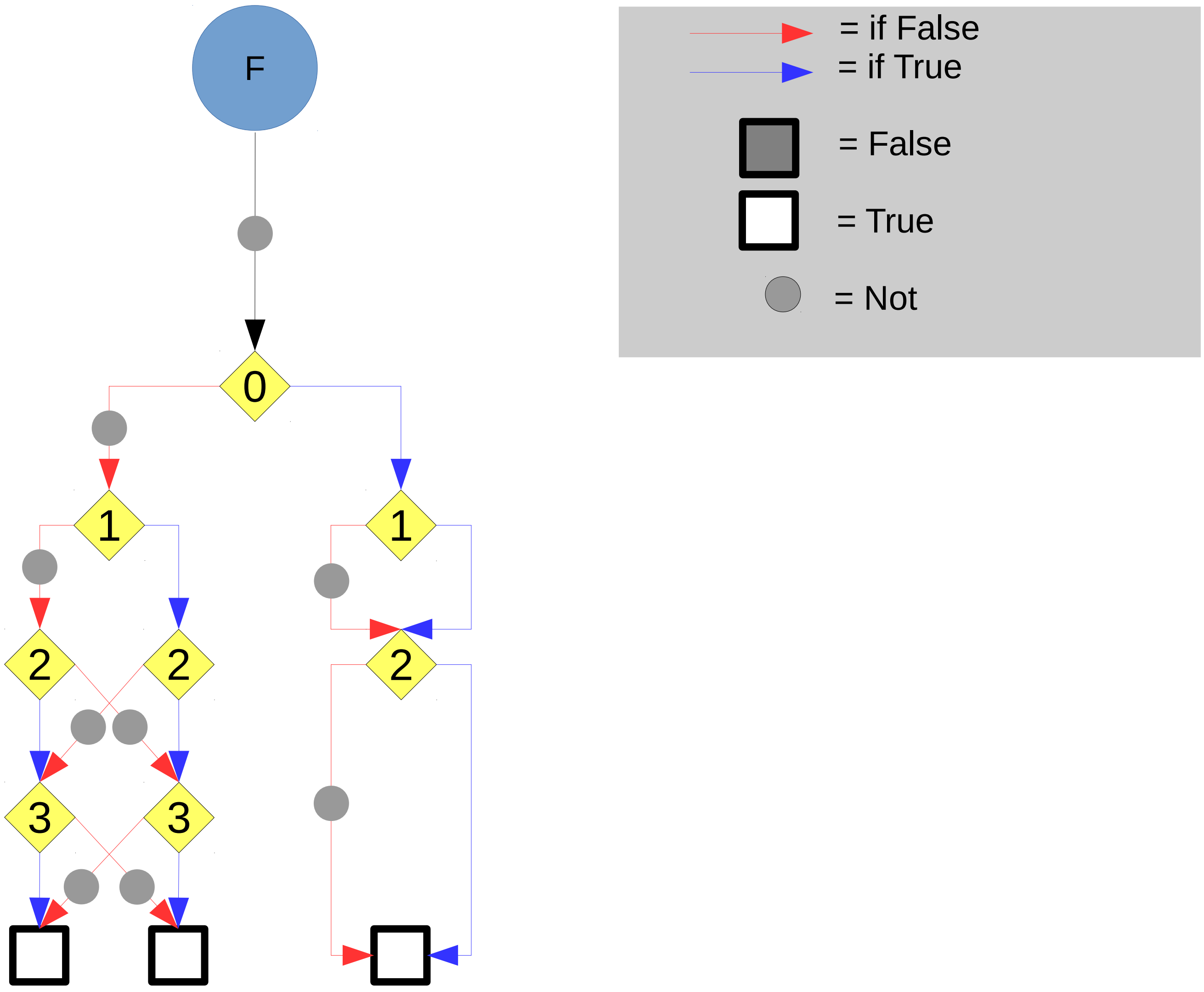
(Complemented Edges) Step 2 : we propagate inverted edges upward, ensuring that no “if True” edge is complemented



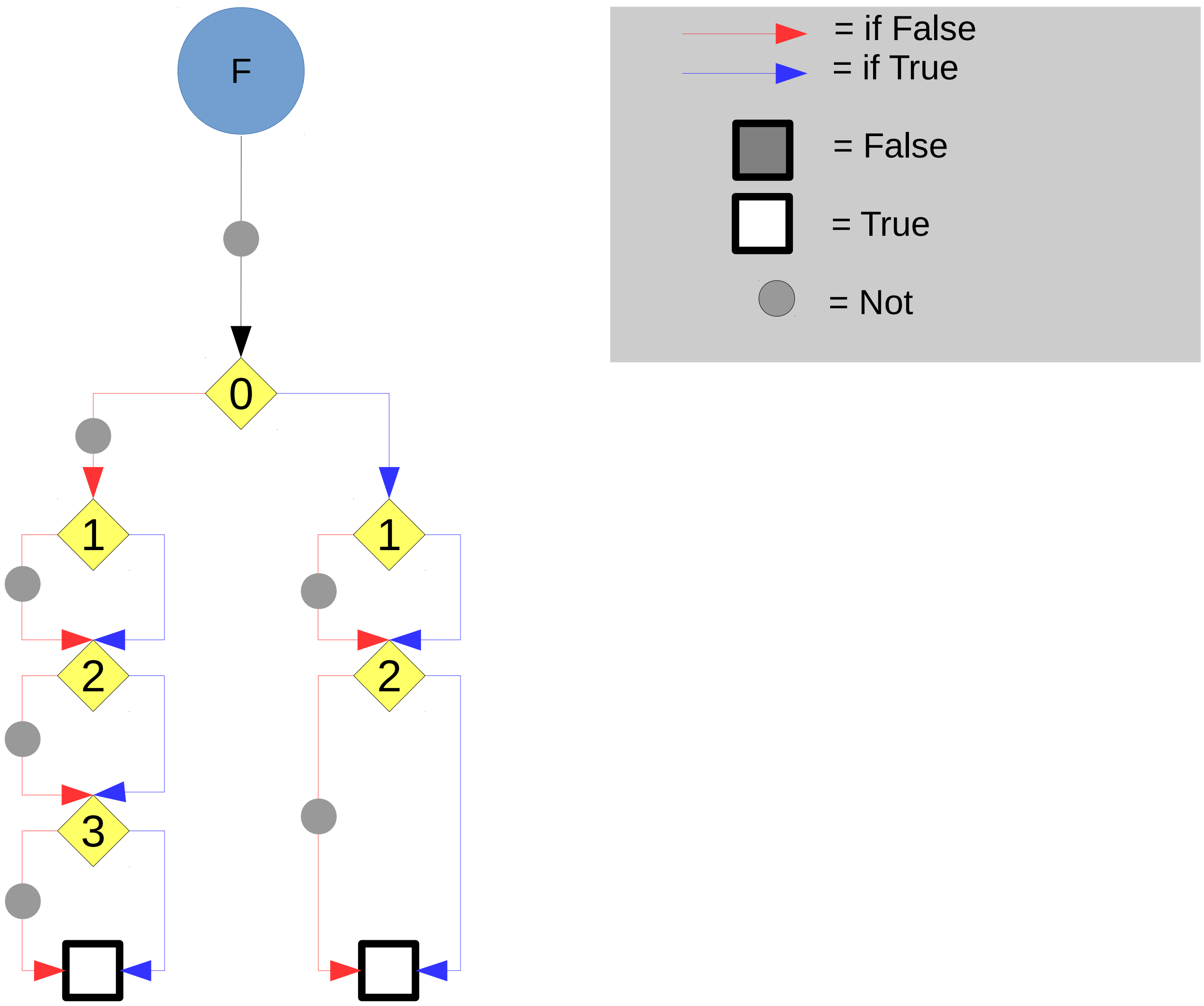
we merge isomorphic sub-graphs



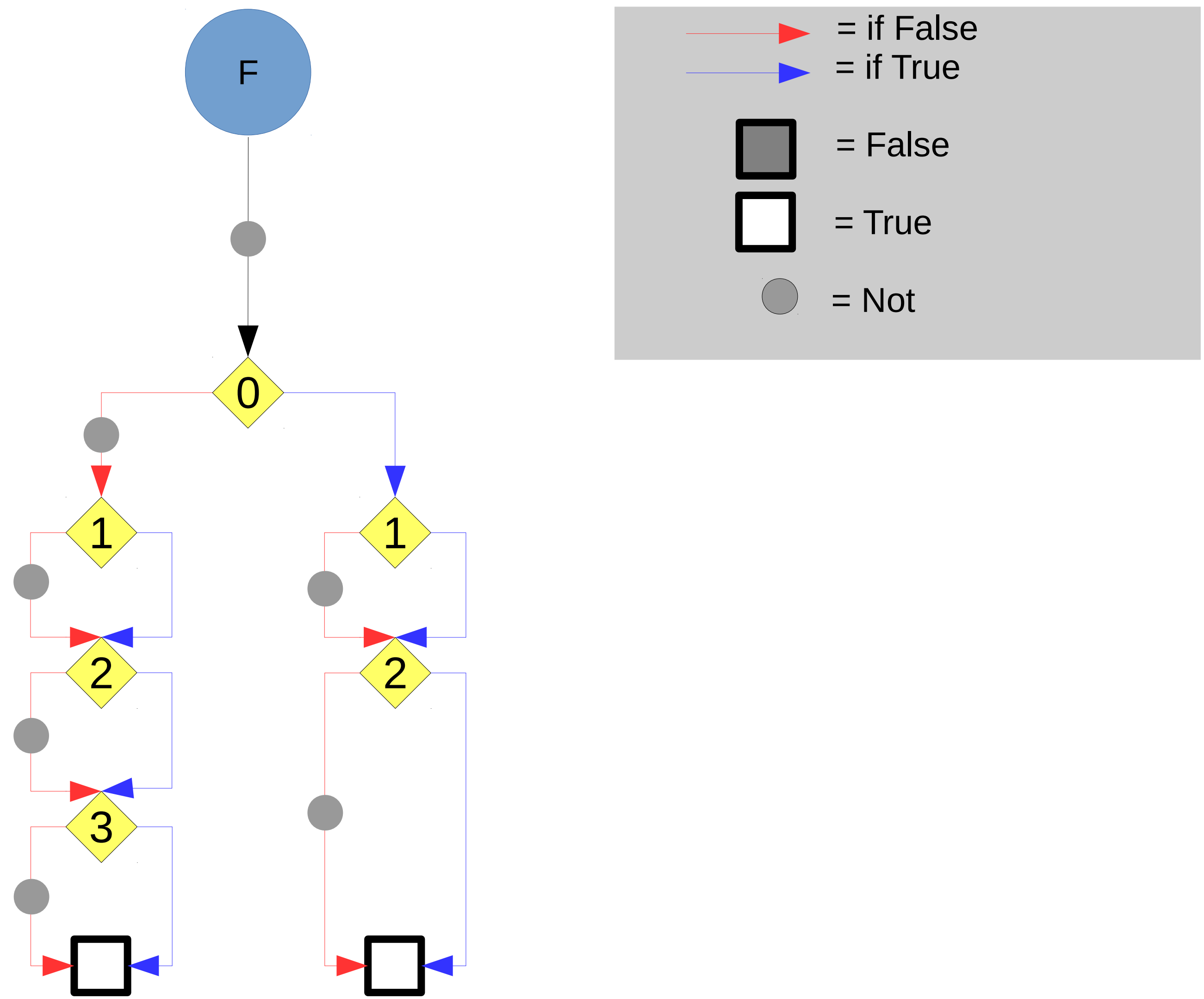
we merge isomorphic sub-graphs



we merge isomorphic sub-graphs

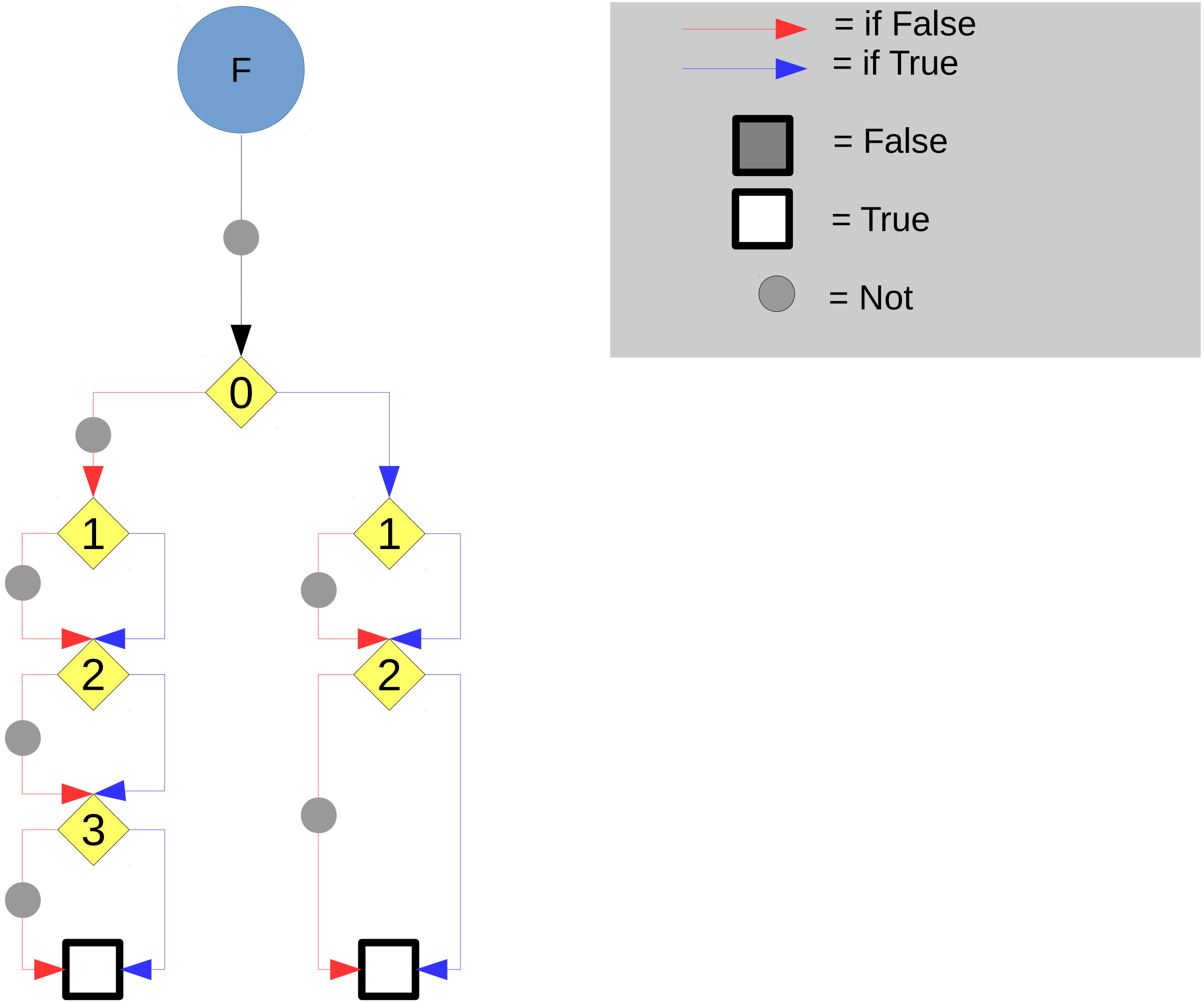


we merge isomorphic sub-graphs



Augmenting edges with
negation can be
performed in linear time in #node

State Of The Art since 2000s

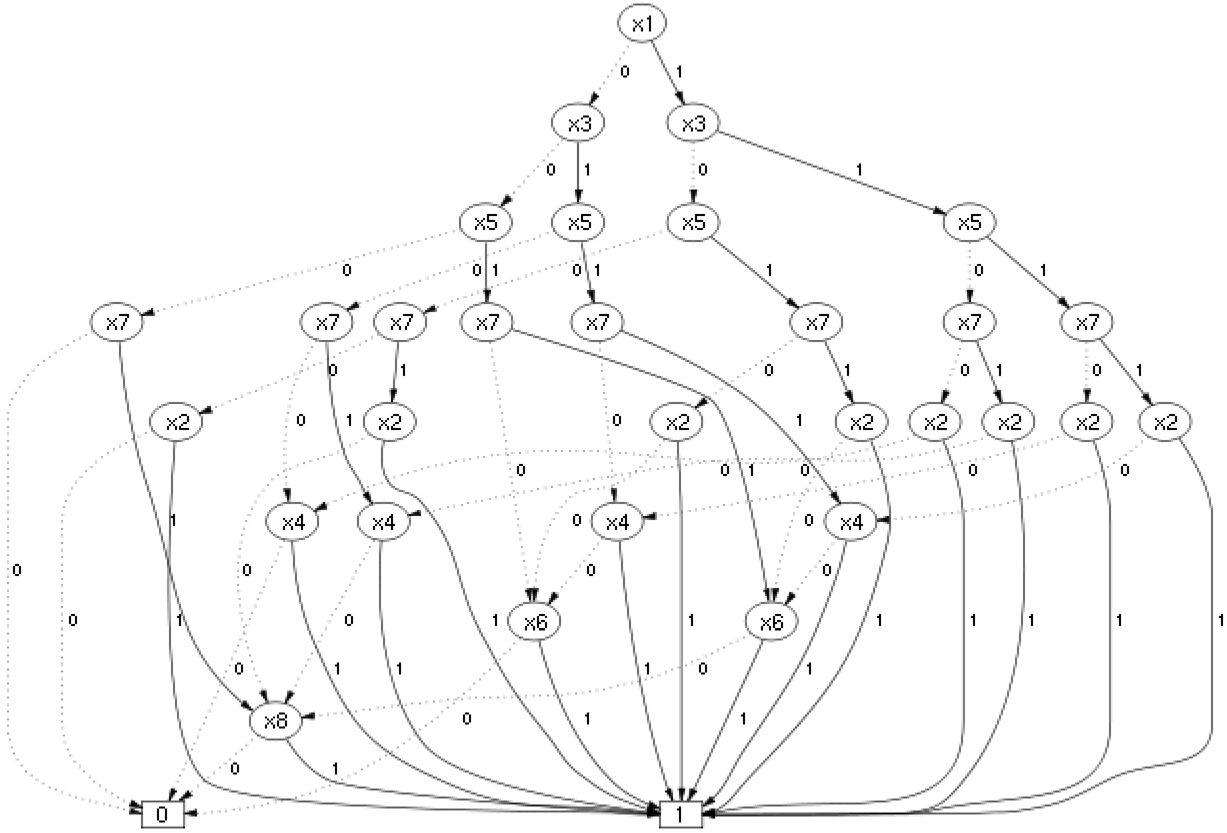


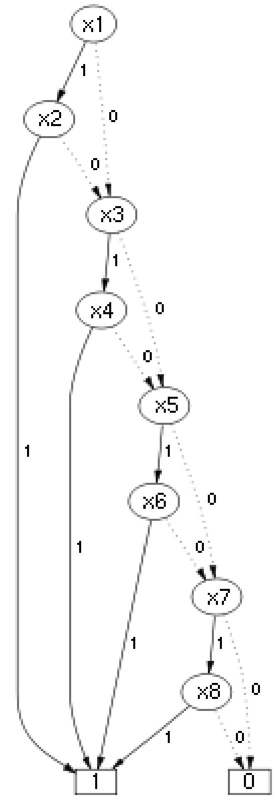
Reduced Ordered BDD

- =) :
  - SAT : constant time
  - Any/Max/Min SAT : linear time (#variable)
    number of
  - #SAT : linear time (#node)
  - NOT : constant time
- =(:
  - AND, XOR : quadratic time/space (#node)
  - #node is order dependent

#node is order dependent

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee (x_5 \wedge x_6) \vee (x_7 \wedge x_8)$$





Objective

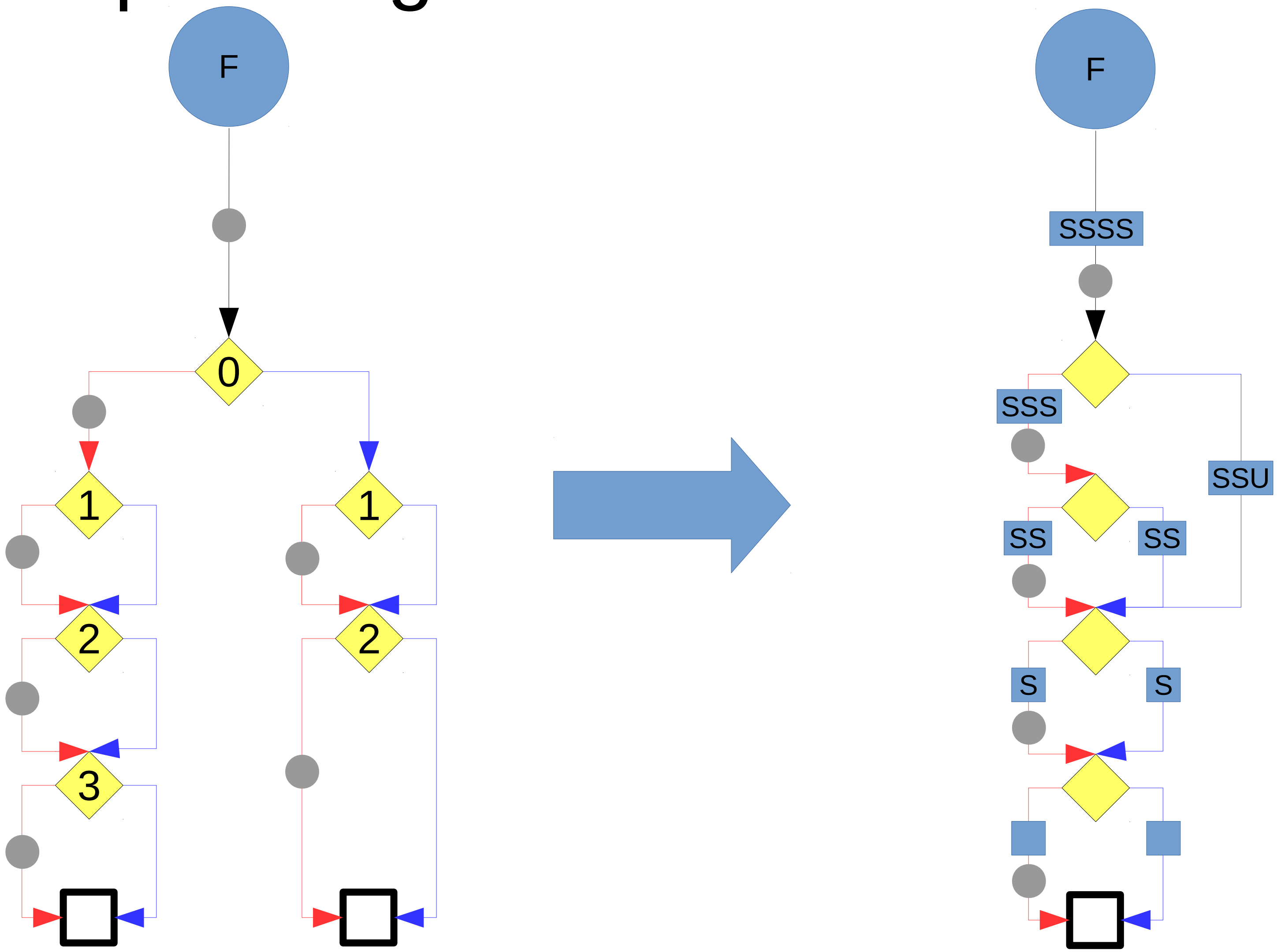
Reduce #node

Objective

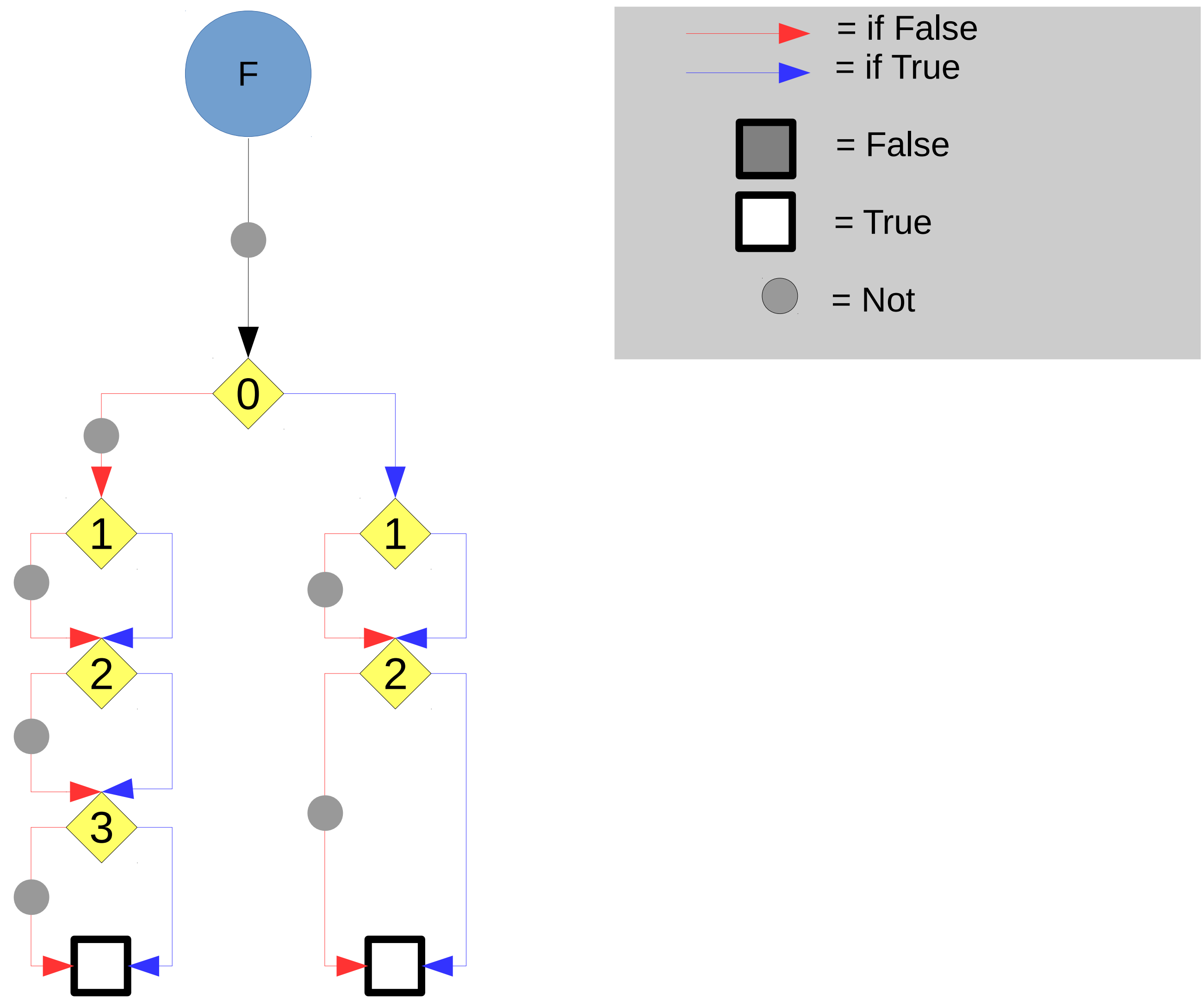
Reduce #node

Capture information on the edges => less but bigger nodes

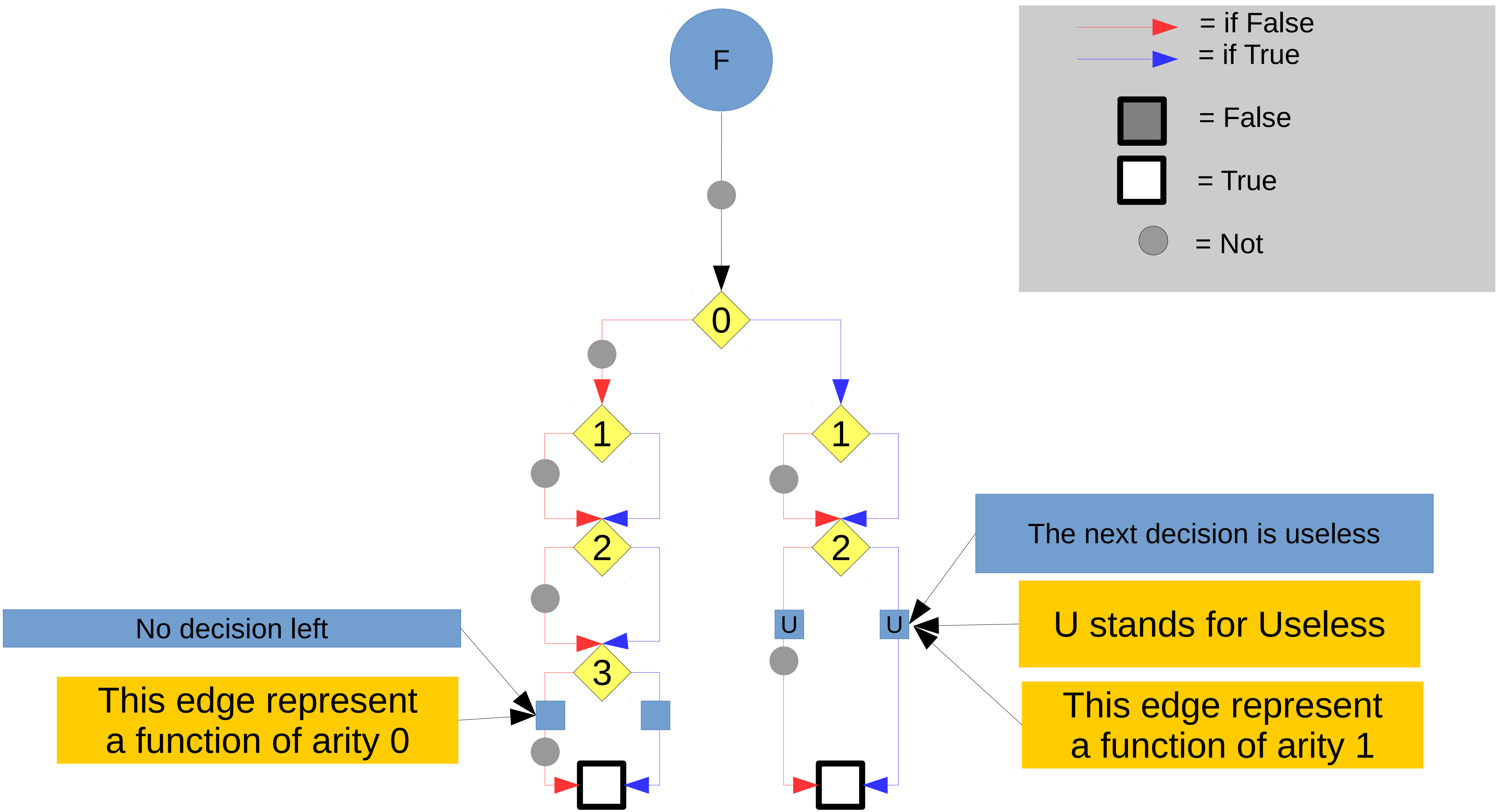
Section 2
Compressing a ROBDD into a GroBdd



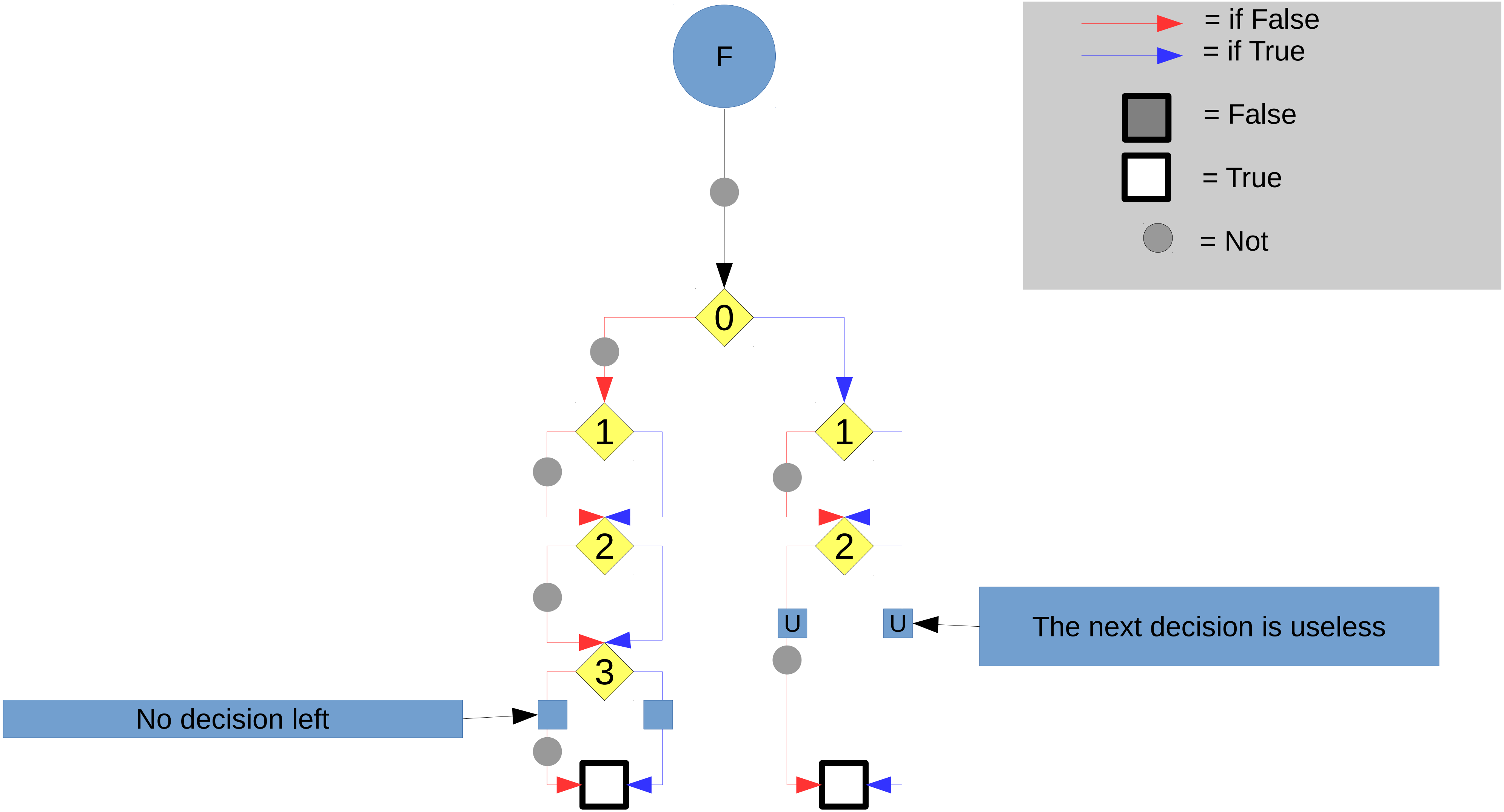
(Model NU) Step 1: for terminal leading edges, we unary represent the number of useless decisions



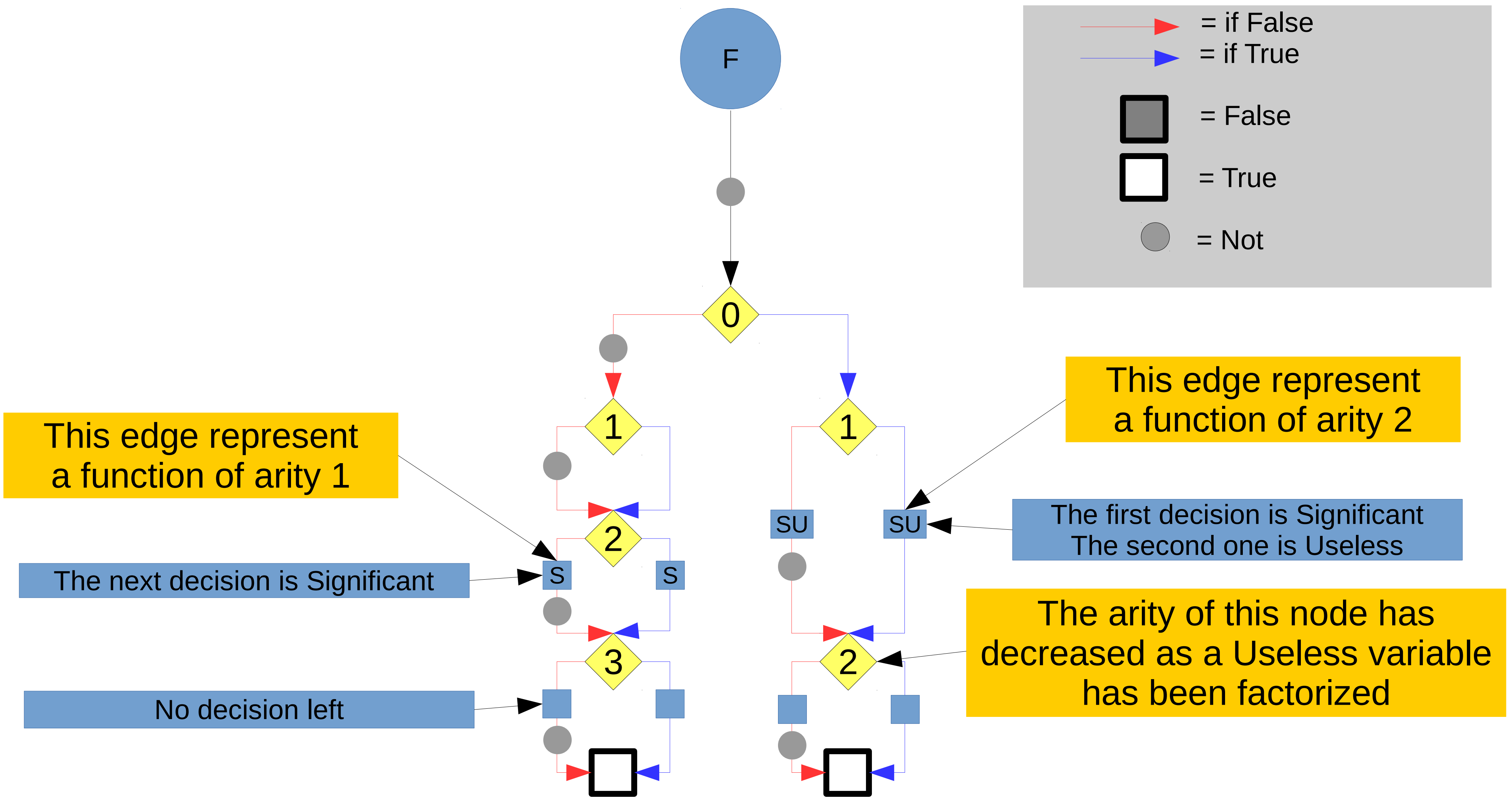
(Model NU) Step 1: for terminal leading edges, we unary represent the number of useless decisions



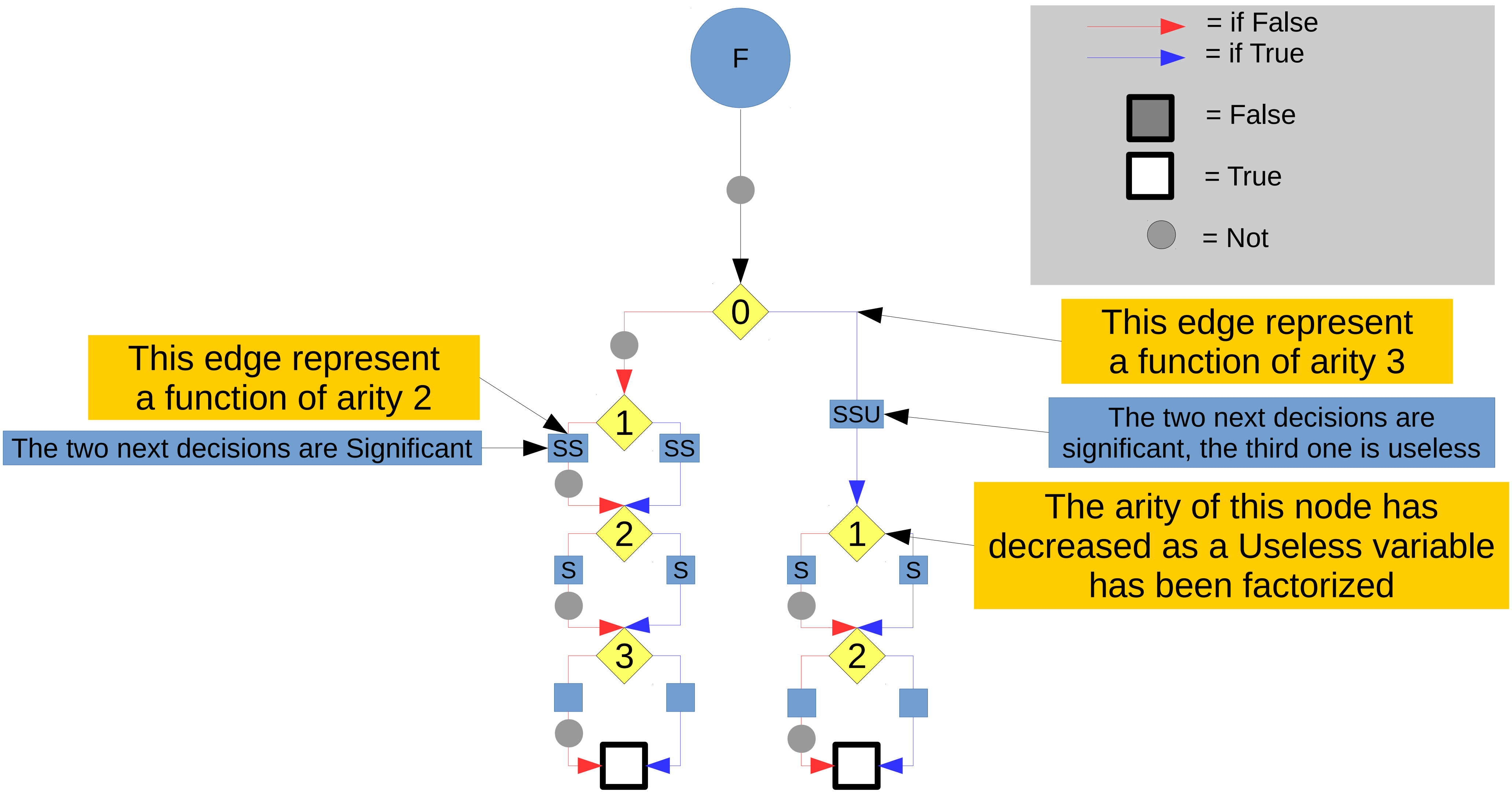
(Model NU) Step 2: we factorize useless variables



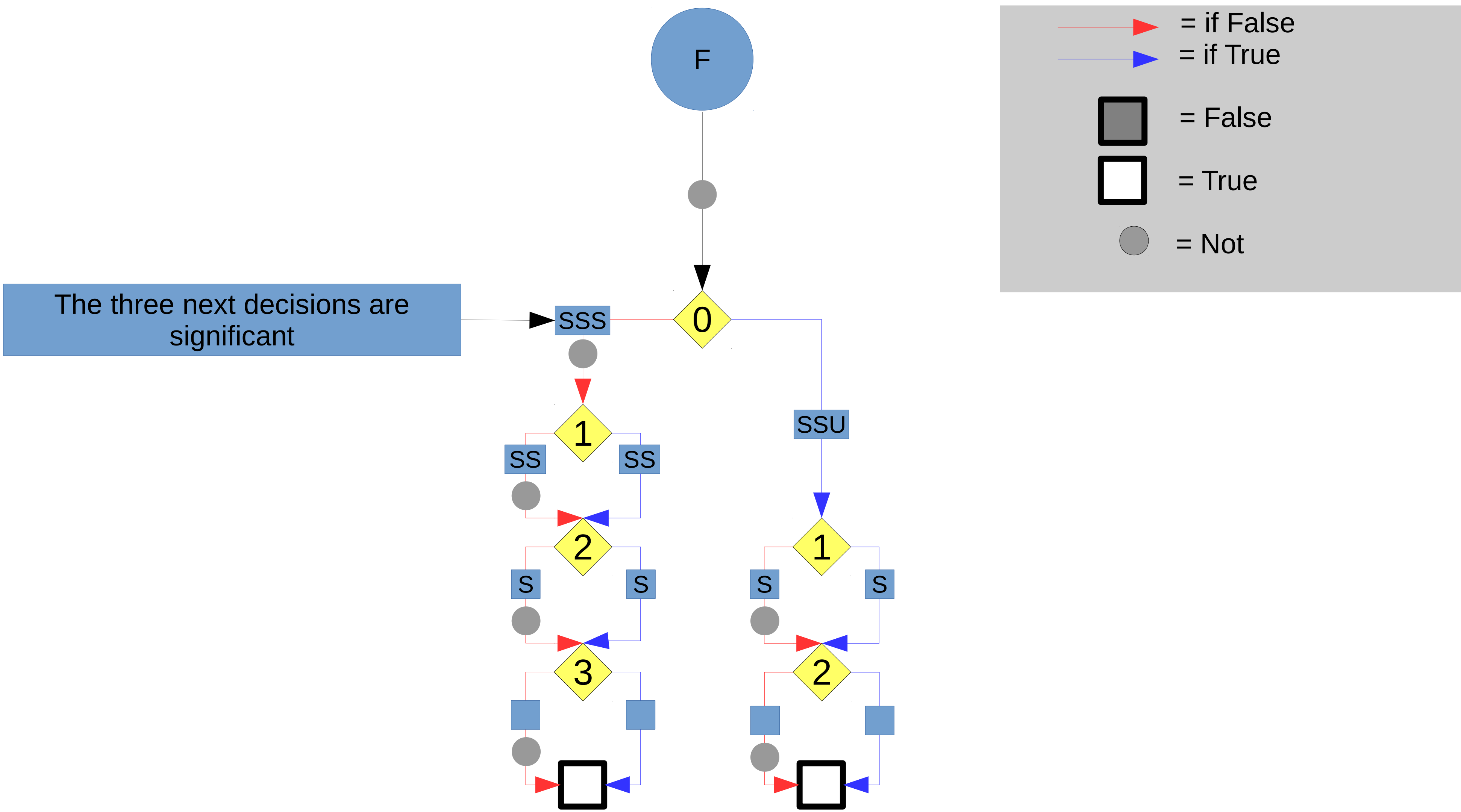
(Model NU) Step 2: we factorize useless variables



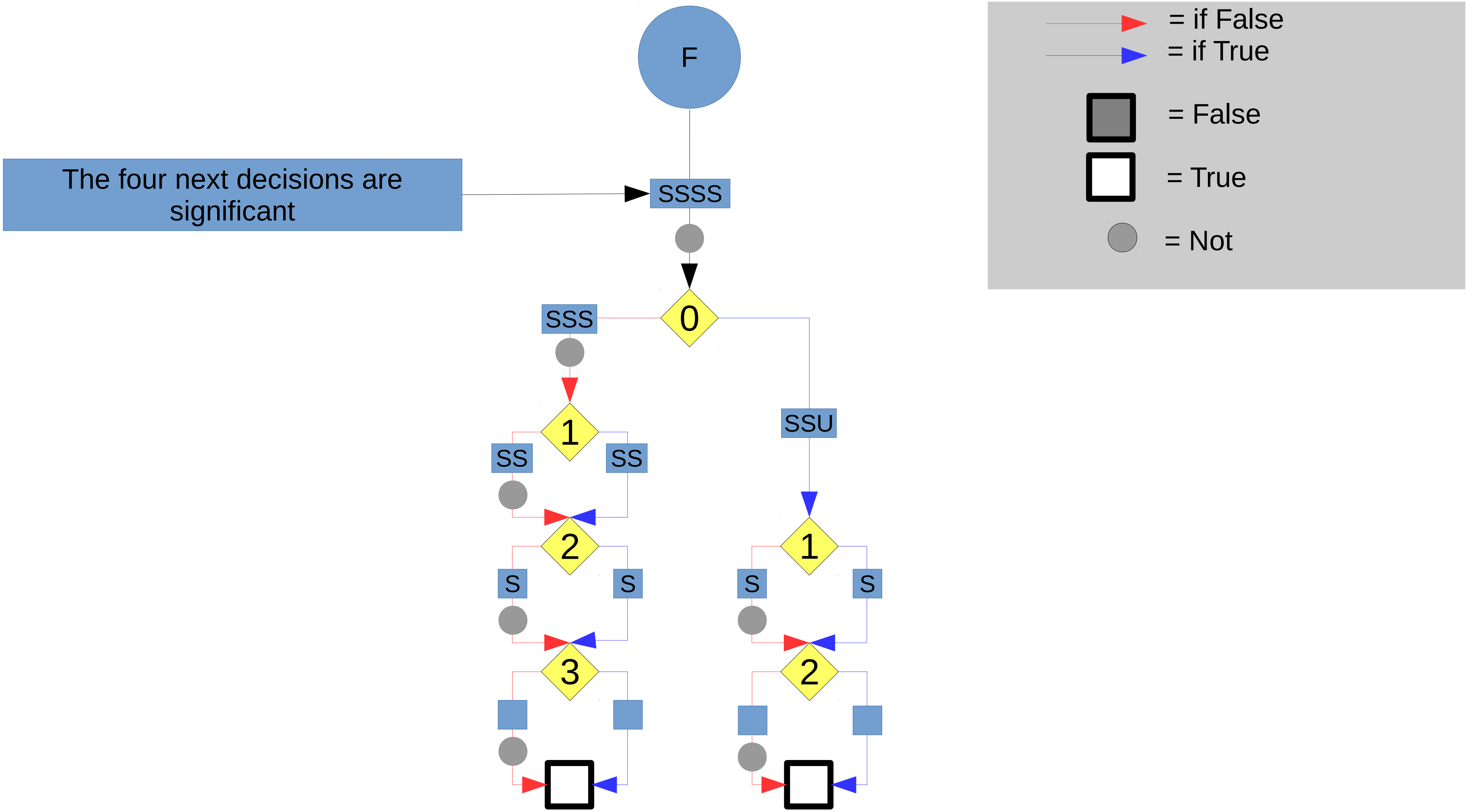
(Model NU) Step 2: we factorize useless variables



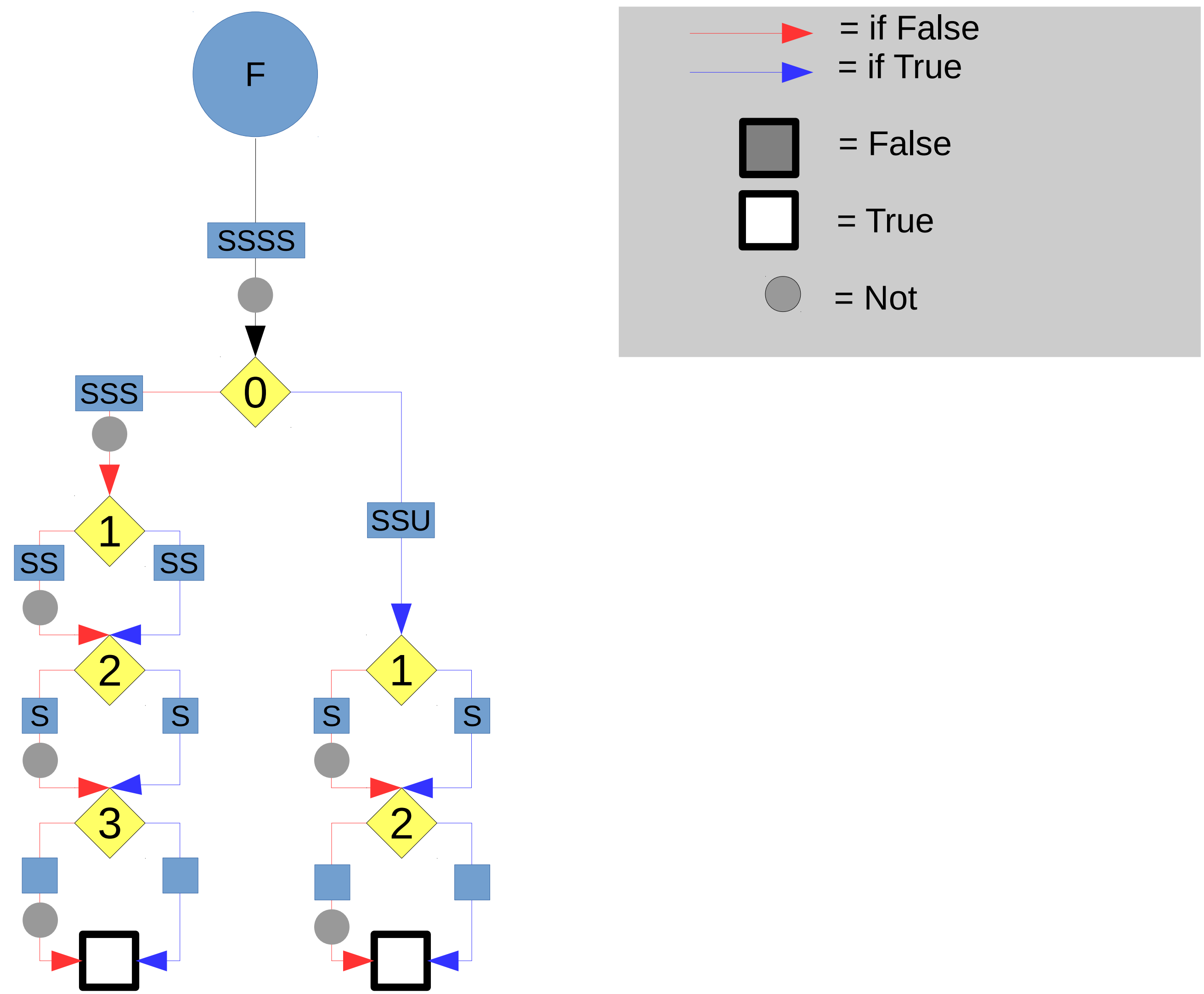
(Model NU) Step 2: we factorize useless variables



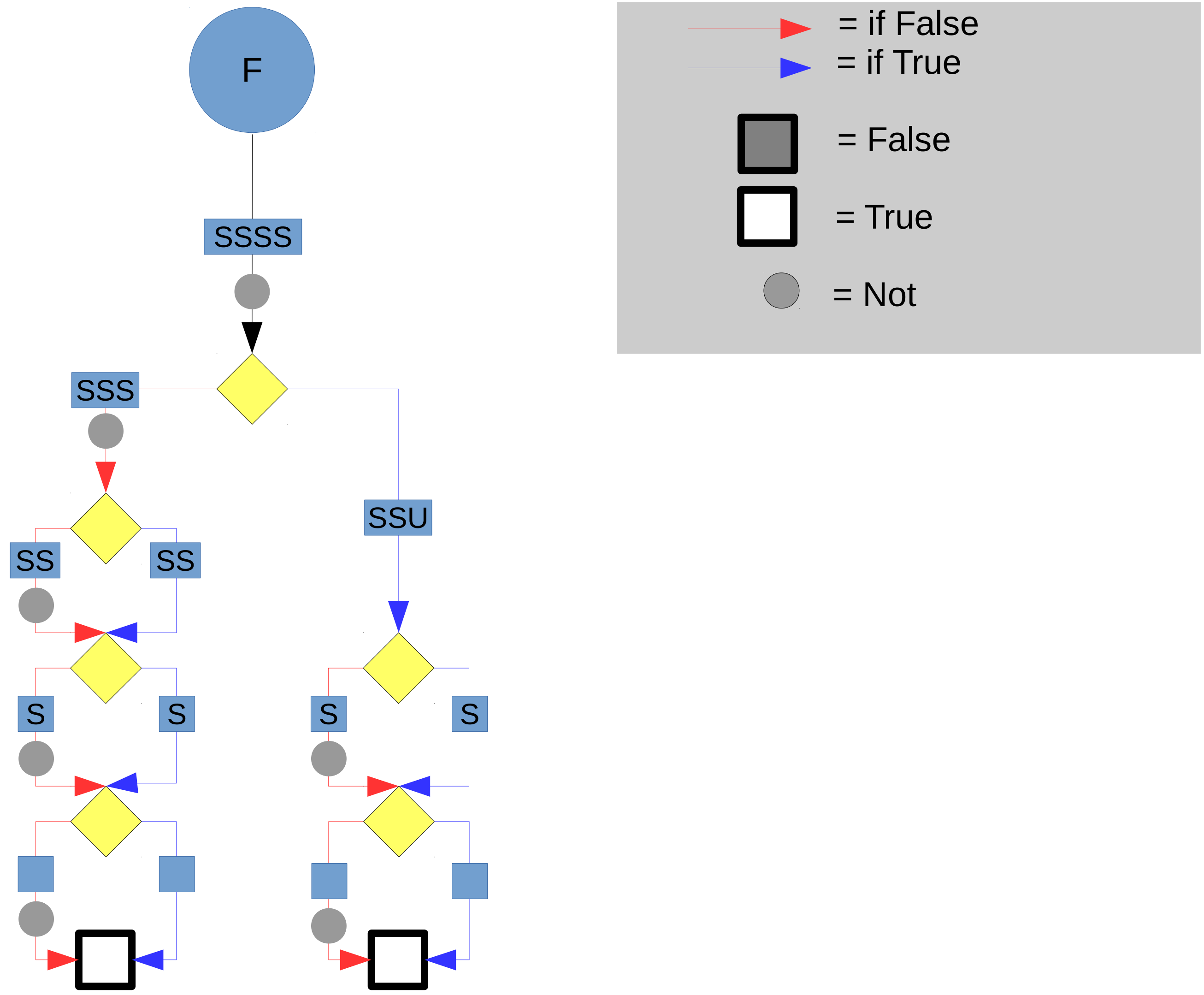
(Model NU) Step 2: we factorize useless variables



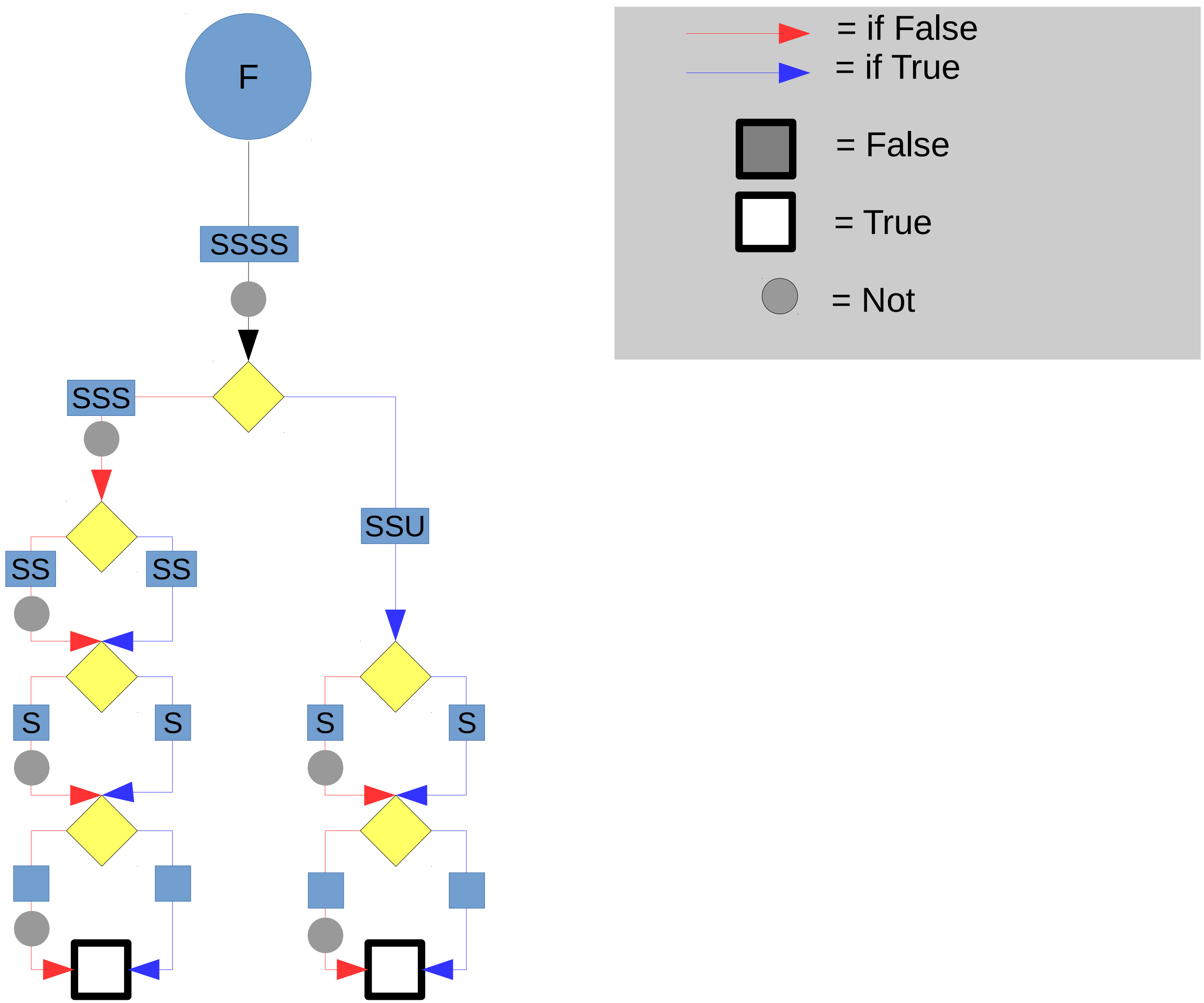
(Model NU) Step 3: we forget every node's depth



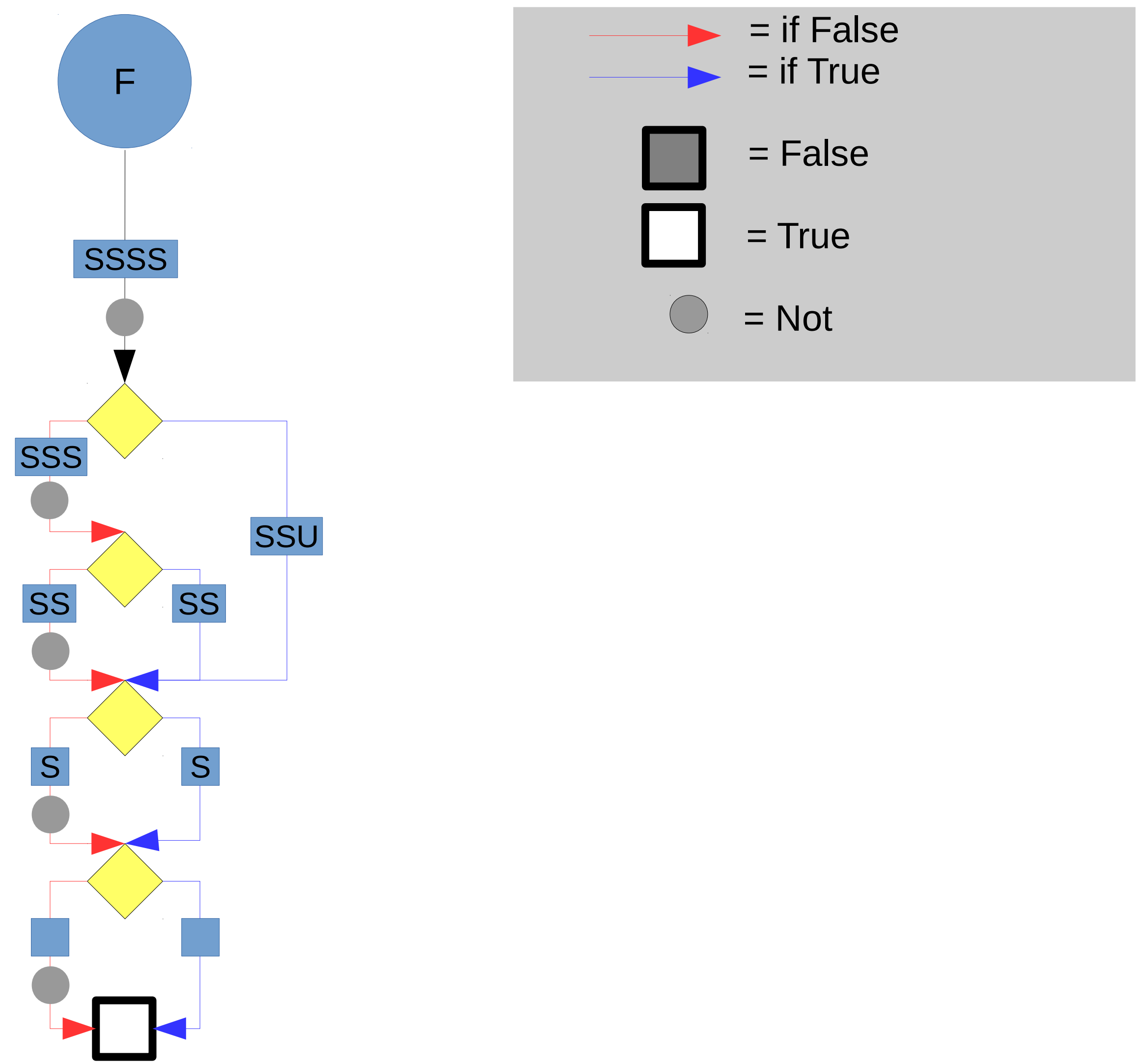
(Model NU) Step 3: we forget every node’s decision variable



we merge isomorphic sub-graphs



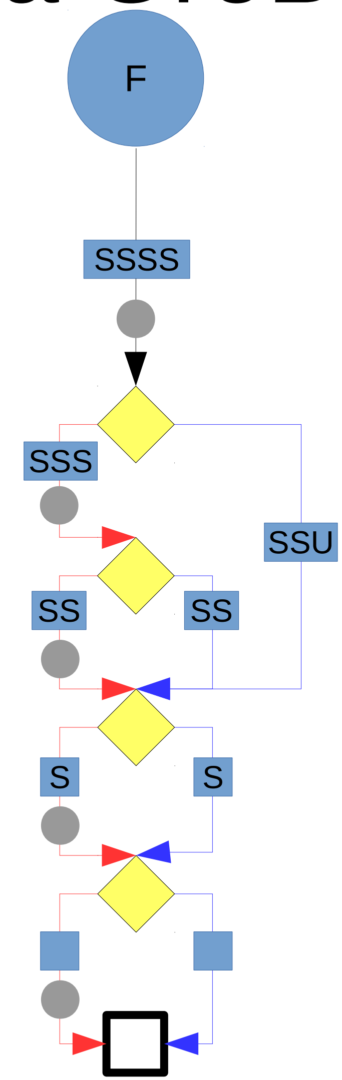
we merge isomorphic sub-graphs



Section 3
Compiling a formula into a GroBdd

$$f\left(x_0^3\right)=x_1\oplus x_2\oplus\left(\neg x_0\wedge x_3\right)$$

Represents a vector of four elements:
$x_0^3=\left(x_0, x_1, x_2, x_3\right)$

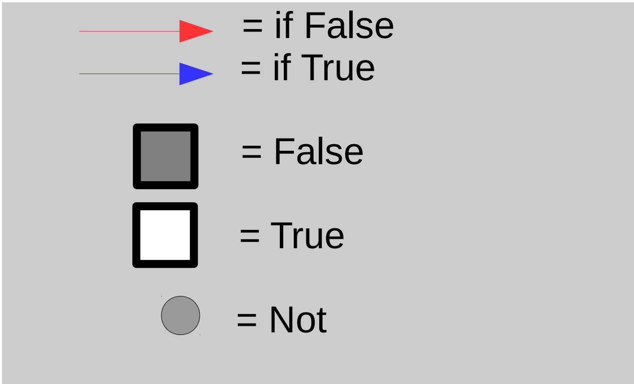


How to compile a formula into a GroBdd

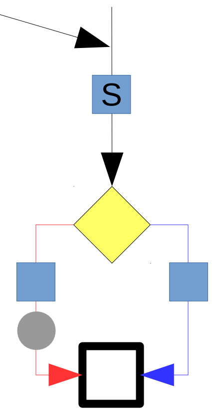
$$f\left(x_0^3\right) = x_1 \oplus x_2 \oplus \left(\neg x_0 \wedge x_3\right)$$

How to compile a formula into a GroBdd

$$f\left(x_0^3\right)=x_1\oplus x_2\oplus\left(\neg x_0\land x_3\right)$$

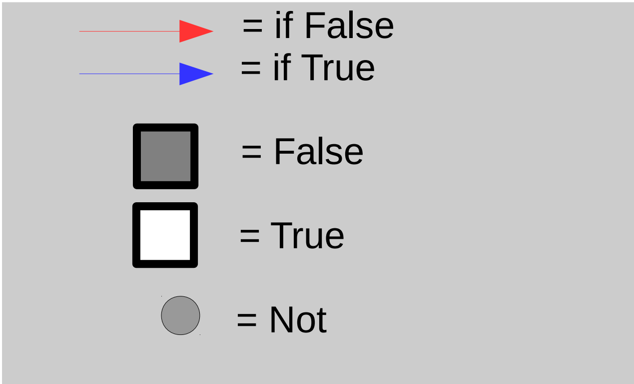


Step 1: we build
the identity function



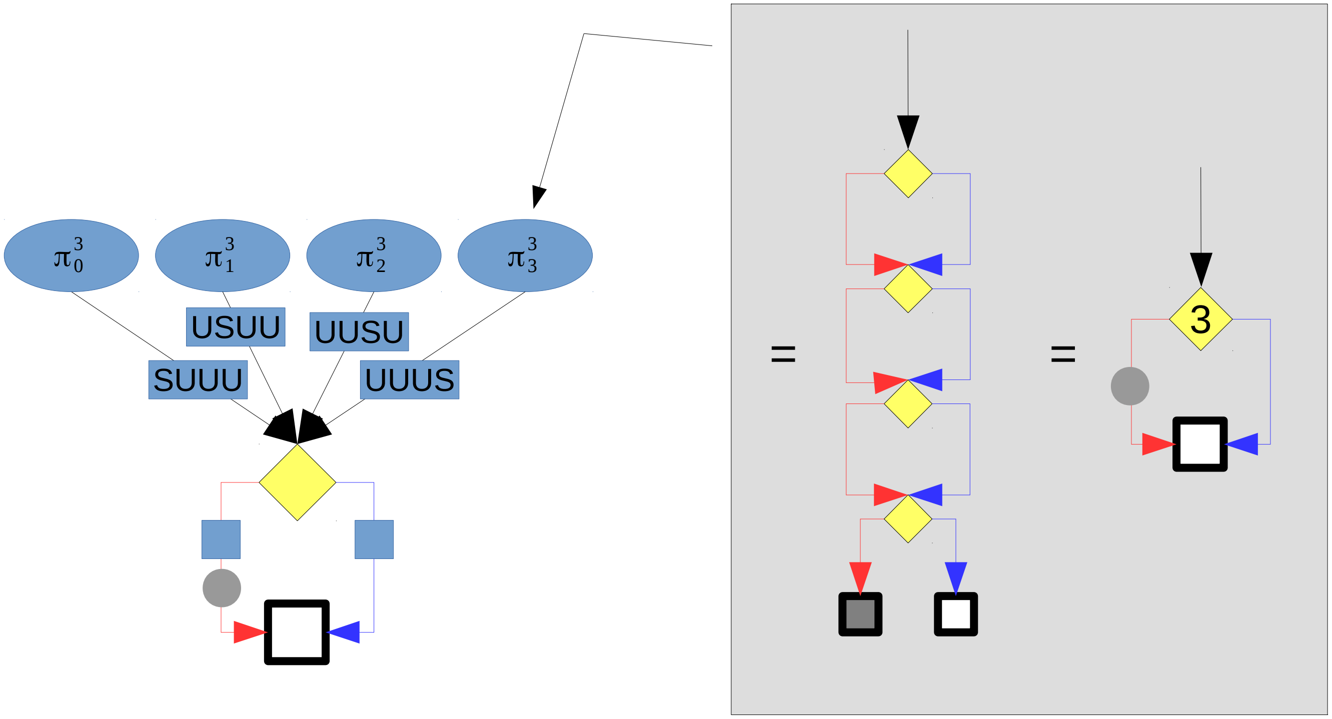
How to compile a formula into a GroBdd

$$f(x_0^3) = x_1 \oplus x_2 \oplus (\neg x_0 \wedge x_3)$$



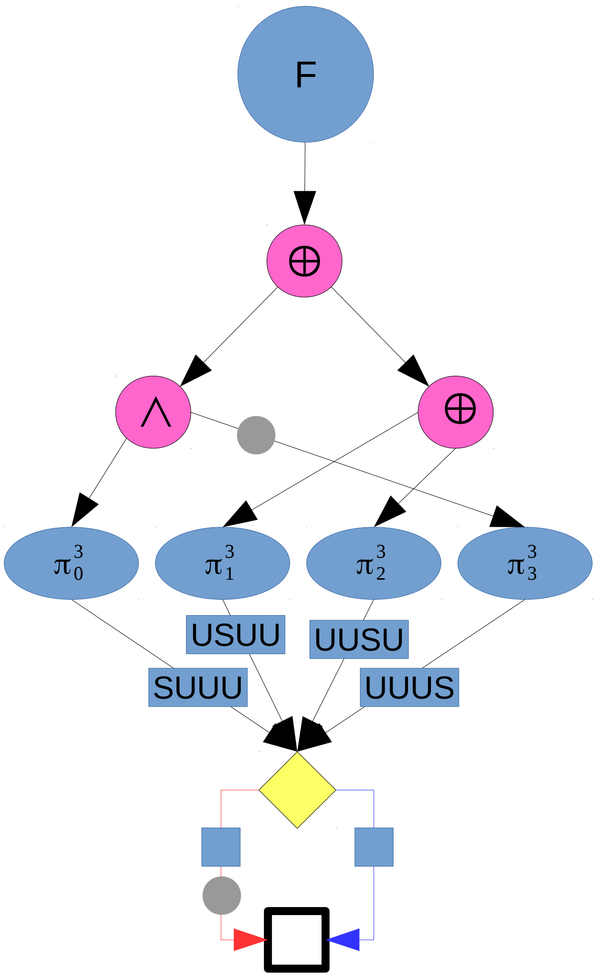
Step 2: we build one projection per variable

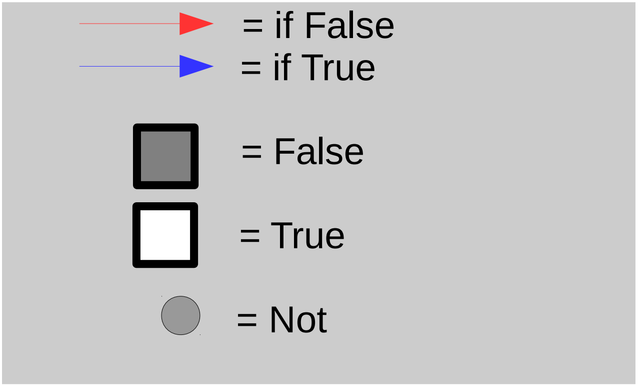
$\pi_k^n(x_0^n) = x_k$

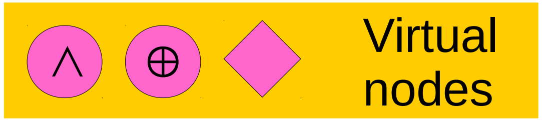


How to compile a formula into a GroBdd

$$f(x_0^3) = x_1 \oplus x_2 \oplus (\neg x_0 \wedge x_3)$$



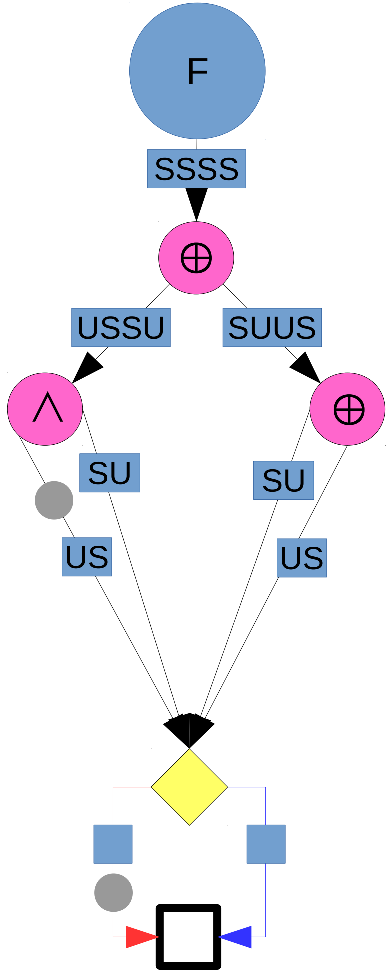


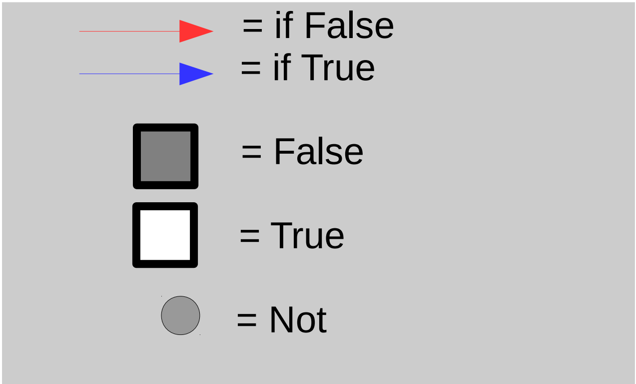


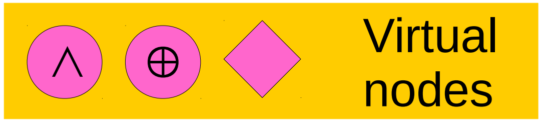
Step 3: we build the formula

How to compile a formula into a GroBdd

$$f\left(x_0^3\right)=x_1 \oplus x_2 \oplus \left(\neg x_0 \wedge x_3\right)$$



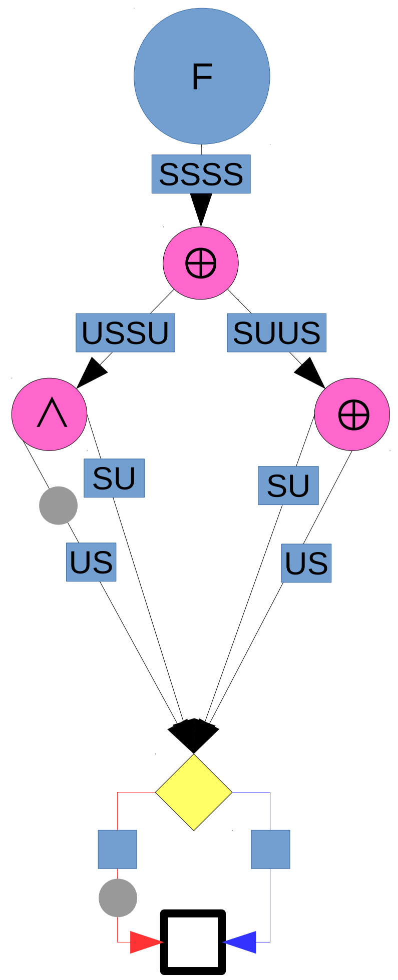


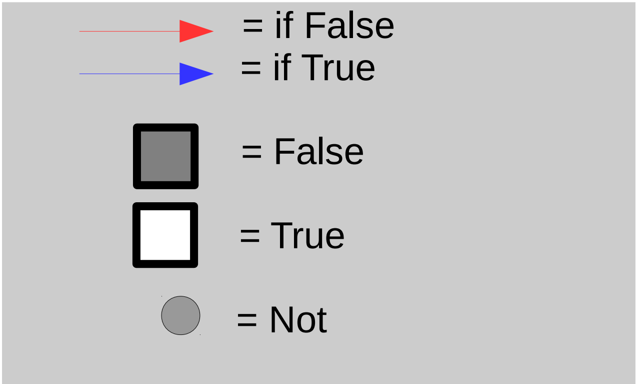


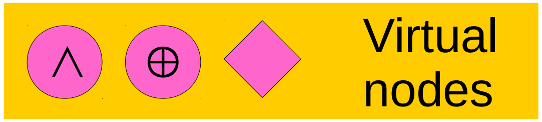
Step 4: we factorize useless variables

How to compile a formula into a GroBdd

$$f\left(x_0^3\right)=x_1\oplus x_2\oplus\left(\neg x_0\wedge x_3\right)$$



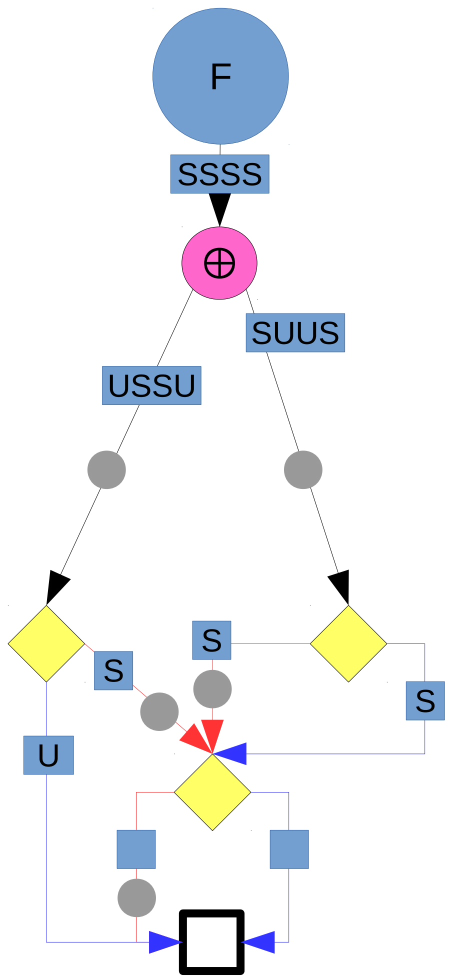


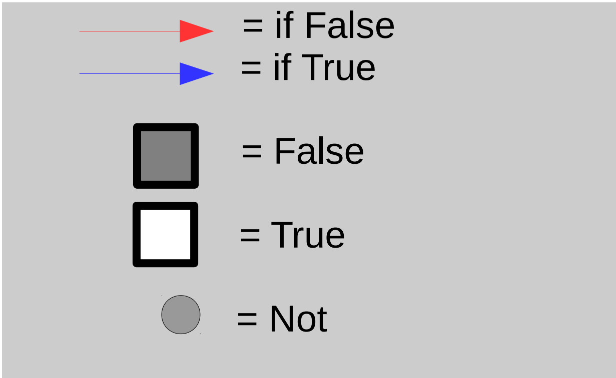


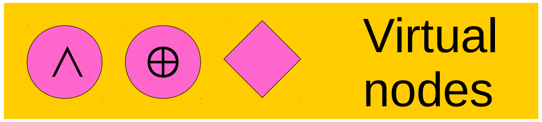
Step 5: we compute operator nodes

How to compile a formula into a GroBdd

$$f(x_0^3) = x_1 \oplus x_2 \oplus (\neg x_0 \wedge x_3)$$



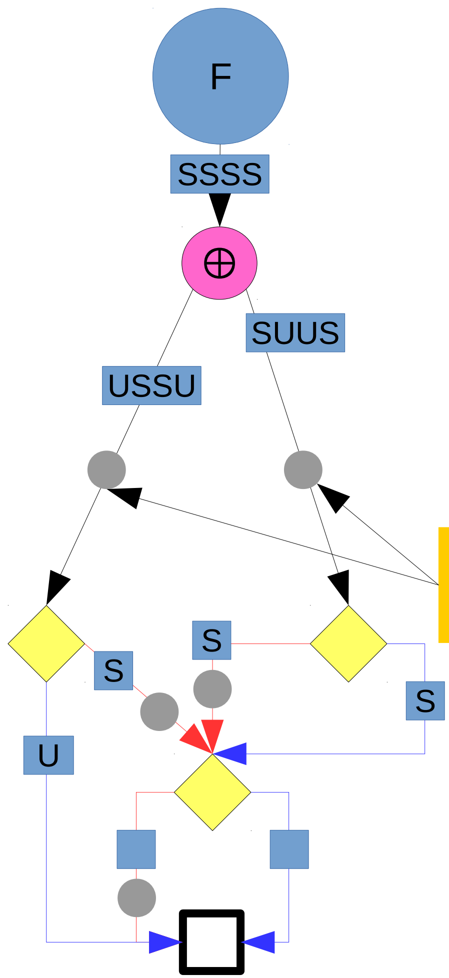




Step 5: we compute operator nodes

How to compile a formula into a GroBdd

$$f\left(x_0^3\right)=x_1 \oplus x_2 \oplus \left(\neg x_0 \wedge x_3\right)$$



= if False
= if True
= False
= True
= Not

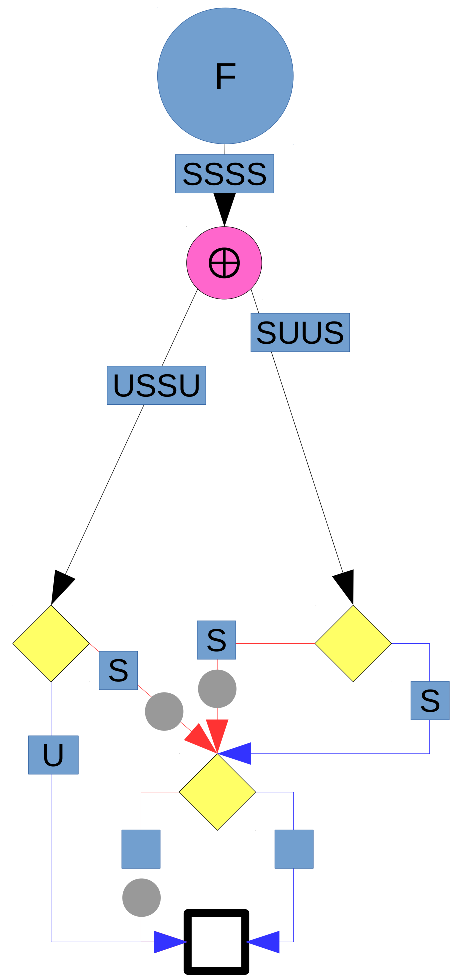
Virtual nodes

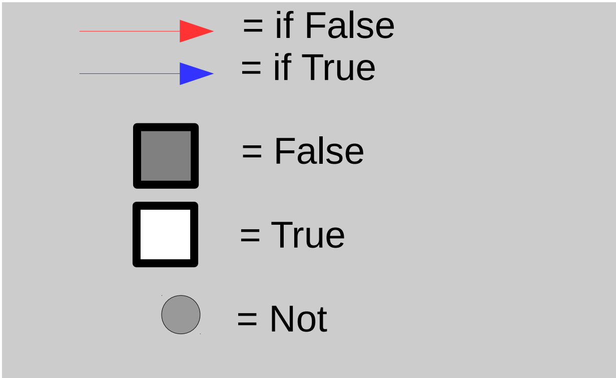
$$\neg f \oplus \neg g = f \oplus g$$

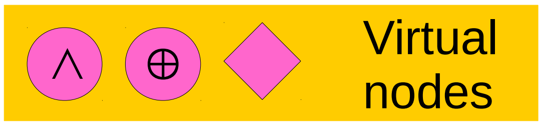
Step 5: we compute operator nodes

How to compile a formula into a GroBdd

$$f\left(x_0^3\right)=x_1\oplus x_2\oplus\left(\neg x_0\wedge x_3\right)$$



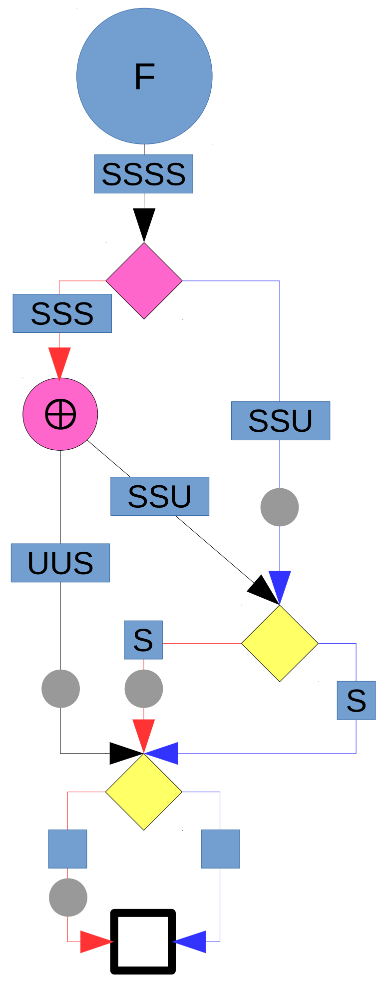


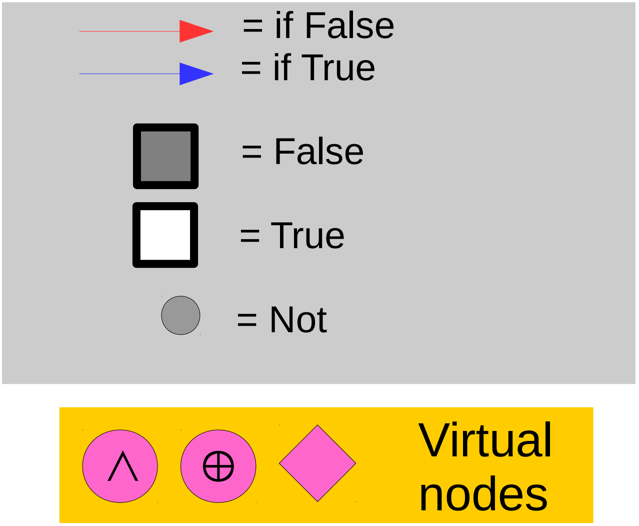


Step 5: we compute operator nodes

How to compile a formula into a GroBdd

$$f\left(x_0^3\right)=x_1 \oplus x_2 \oplus \left(\neg x_0 \wedge x_3\right)$$

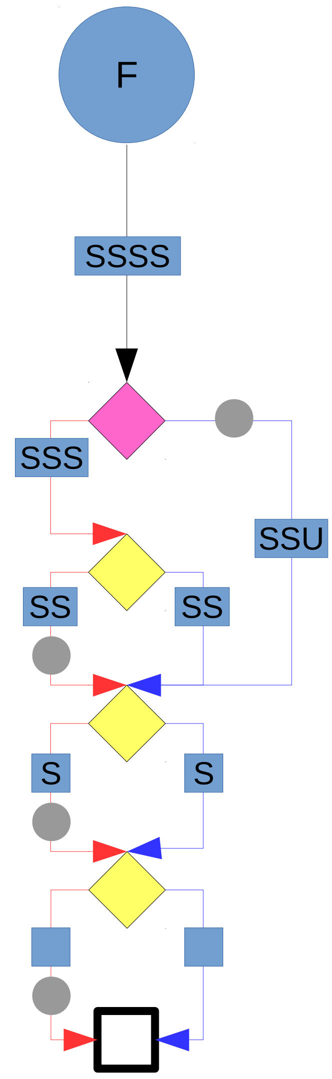


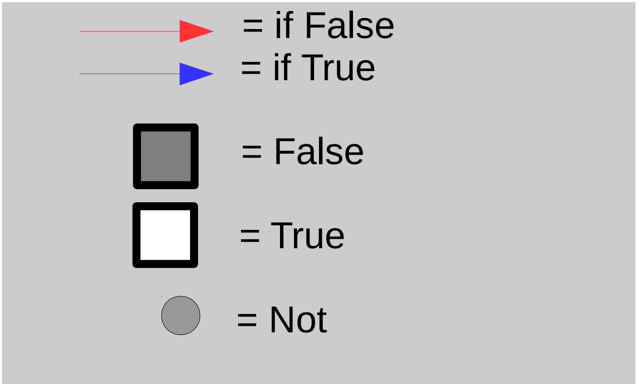


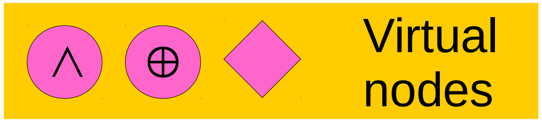
Step 5: we compute operator nodes

How to compile a formula into a GroBdd

$$f(x_0^3) = x_1 \oplus x_2 \oplus (\neg x_0 \wedge x_3)$$



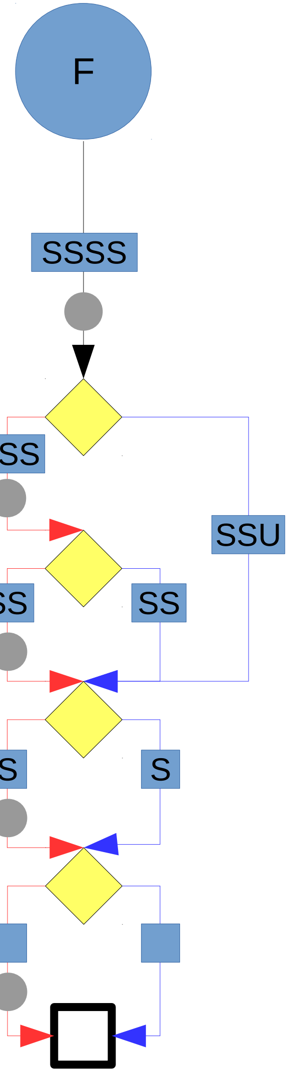


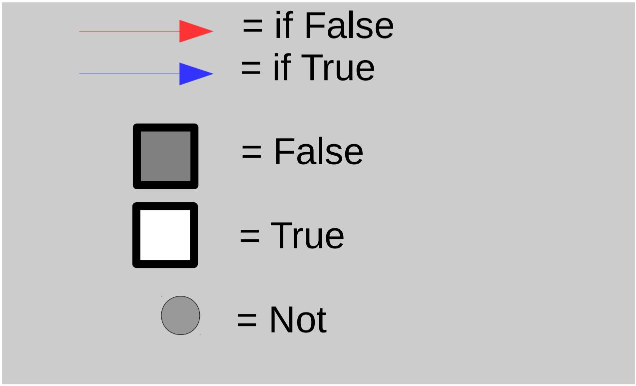


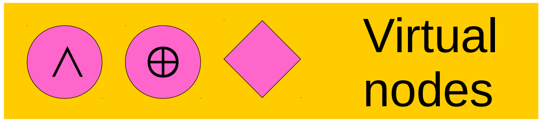
Step 5: we compute operator nodes

How to compile a formula into a GroBdd

$$f\left(x_0^3\right)=x_1\oplus x_2\oplus\left(\neg x_0\wedge x_3\right)$$







Step 5: we compute operator nodes

Section 4
Results

Average reduction of the number of nodes / estimated memory cost on three benchmarks

	#node	memory[1]
lgsynth91	-26%	-32%
iscas99	-25%	-32%
satlib/uf20-91	-3%	-3%

[1]: memory cost estimated using (a fix 16 bytes (= 2 x 64 bits pointer) + a variable length encoding of model's extra information) per node

lgsynth91:
- Downloaded from https://ddd.fit.cvut.cz/prj/Benchmarks/LGSynth91.7z
- Compiled from Verilog to Verilog using ABC (https://people.eecs.berkeley.edu/~alanmi/abc/) (DAGaml supports only a subset of Verilog)
- Compiled from Verilog to GroBdd using DAGaml (our software : https://github.com/JoanThibault/DAGaml/tree/grobdd-dev)

iscas99 :
- Downloaded from http://www.pld.ttu.ee/~maksim/benchmarks/iscas99/vhdl/
- Compiled from bench to pla using ABC
- Compiled from pla to GroBdd using DAGaml

satlib/uf20-91 (CNF formulas : 20 variables, 91 clauses)
- Dowloaded from http://www.cs.ubc.ca/~hoos/SATLIB/Benchmarks/SAT/RND3SAT/uf20-91.tar.gz
- Compiled from DIMACS (CNF) to GroBdd using DAGaml

Conclusion

- Software implemented in OCaml:
  - https://github.com/JoanThibault/DAGaml/tree/grobdd-dev
  - ~ 12 000 lines of OCaml
- Fewer nodes & Less memory
- Future Work
  - Quantify the dependency between variables' order and #node
  - Solve & Implement NUA-X and NNI-X versions
- TO DO
  - Parallelism & hardware acceleration
  - Quantification Operators
  - Variable Reordering
- Other Applications
  - Apply similar strategies to compress other DAG
    - DAG / Graph isomorphism
    - Unification