CS181u Applied Logic
& Automated Reasoning

Lecture 5

Binary Decision Diagrams

BDD operations

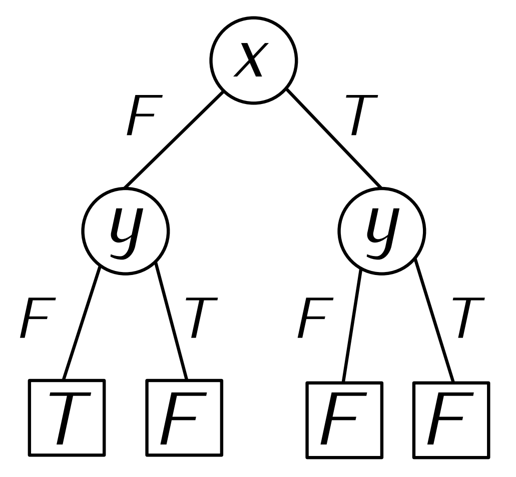
Binary Decision Diagrams

A **Binary Decision Diagram (BDD)** is a data structure for representing the truth values of formulas in propositional logic.

Example: consider the formula $\neg x \wedge \neg y$ and the truth table.

x	y	$\neg x \wedge \neg y$
F	F	T
F	T	F
T	F	F
T	T	F

The same information can be encoded in a decision tree.



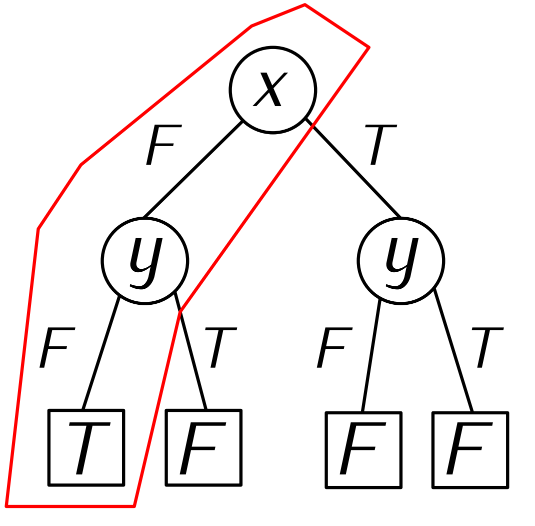
Binary Decision Diagrams

A **Binary Decision Diagram (BDD)** is a data structure for representing the truth values of formulas in propositional logic.

Example: consider the formula $\neg x \wedge \neg y$ and the truth table.

x	y	$\neg x \wedge \neg y$
F	F	T
F	T	F
T	F	F
T	T	F

The same information can be encoded in a decision tree.



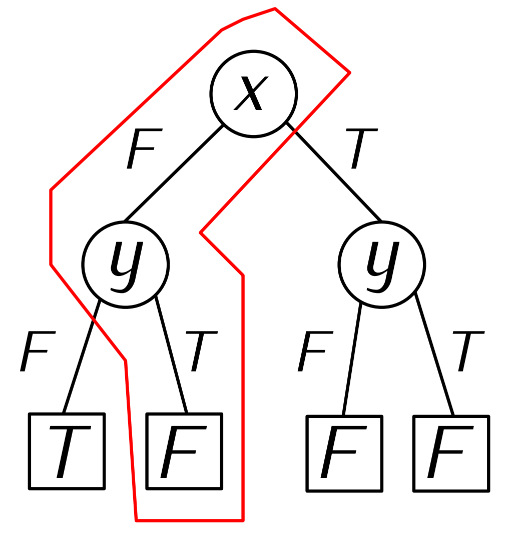
Binary Decision Diagrams

A **Binary Decision Diagram (BDD)** is a data structure for representing the truth values of formulas in propositional logic.

Example: consider the formula $\neg x \wedge \neg y$ and the truth table.

x	y	$\neg x \wedge \neg y$
F	F	T
F	T	F
T	F	F
T	T	F

The same information can be encoded in a decision tree.



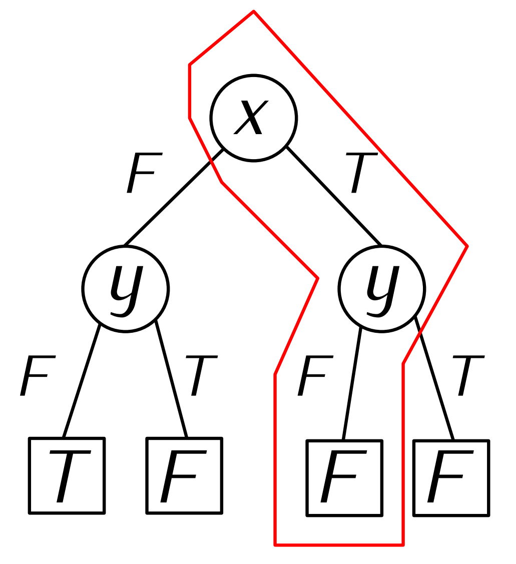
Binary Decision Diagrams

A **Binary Decision Diagram (BDD)** is a data structure for representing the truth values of formulas in propositional logic.

Example: consider the formula $\neg x \wedge \neg y$ and the truth table.

x	y	$\neg x \wedge \neg y$
F	F	T
F	T	F
T	F	F
T	T	F

The same information can be encoded in a decision tree.



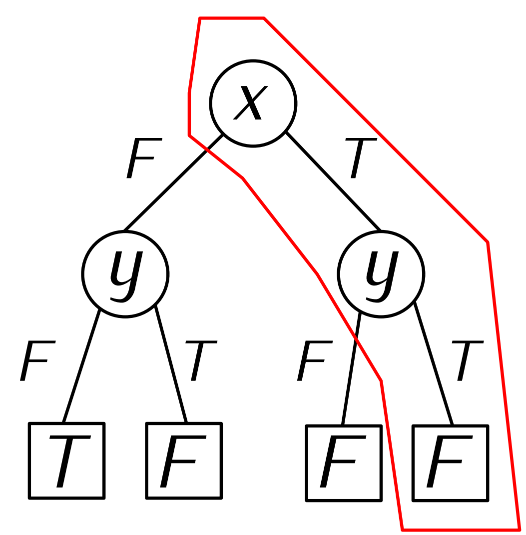
Binary Decision Diagrams

A **Binary Decision Diagram (BDD)** is a data structure for representing the truth values of formulas in propositional logic.

Example: consider the formula $\neg x \wedge \neg y$ and the truth table.

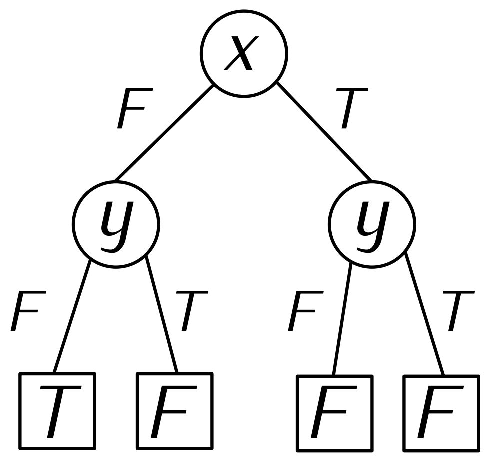
x	y	$\neg x \wedge \neg y$
F	F	T
F	T	F
T	F	F
T	T	F

The same information can be encoded in a decision tree.



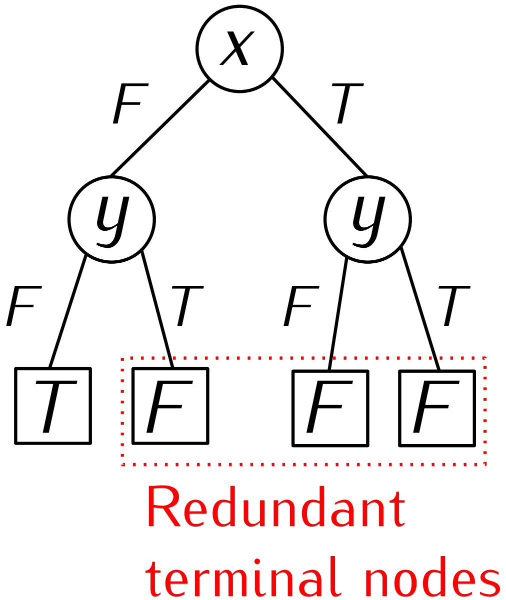
Binary Decision Diagrams

We can make the decision diagram more compact.



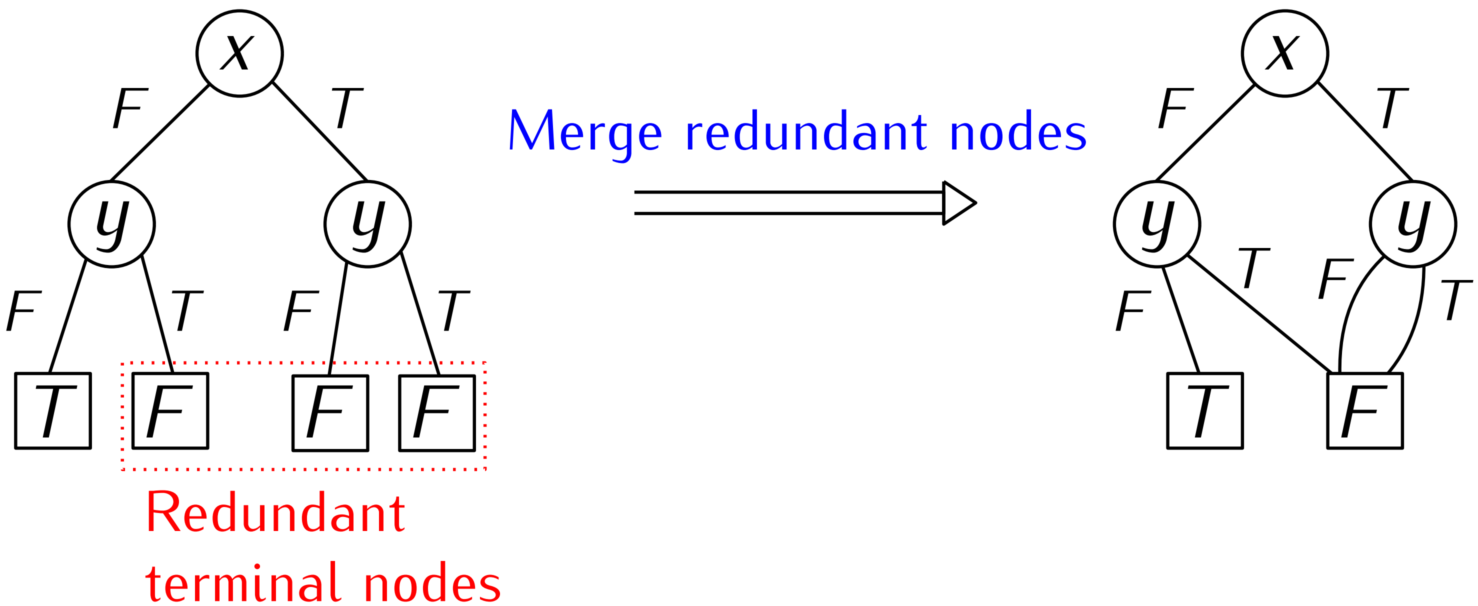
Binary Decision Diagrams

We can make the decision diagram more compact.



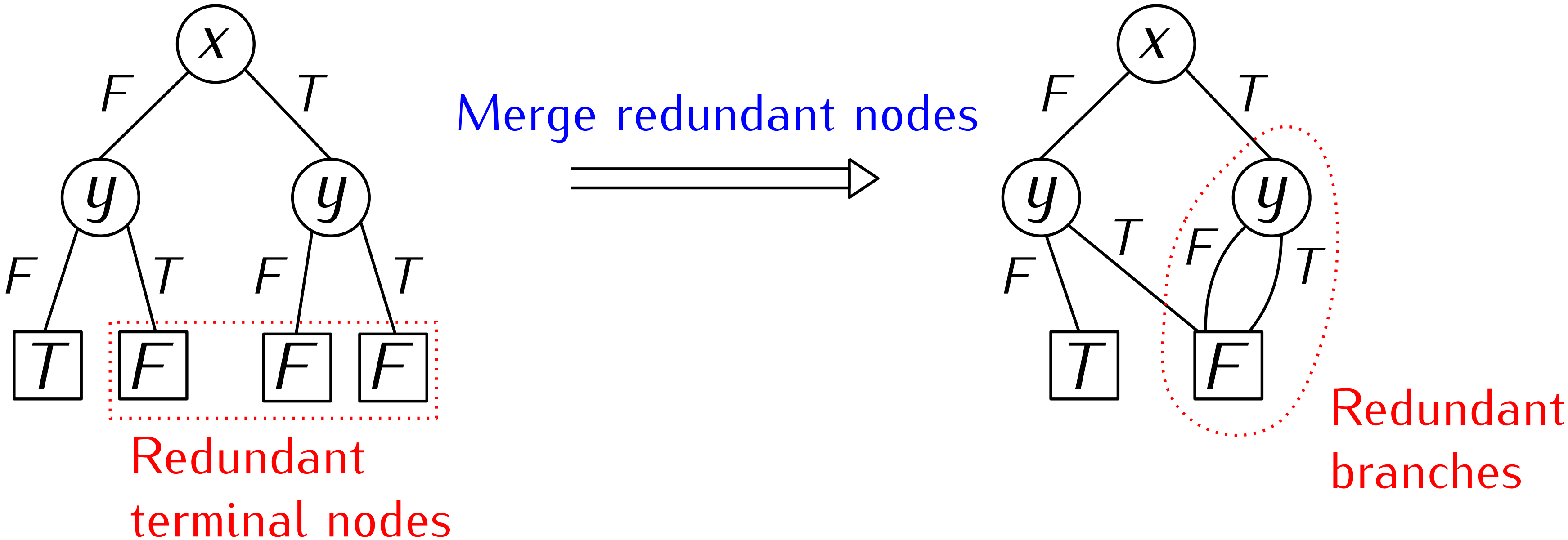
Binary Decision Diagrams

We can make the decision diagram more compact.



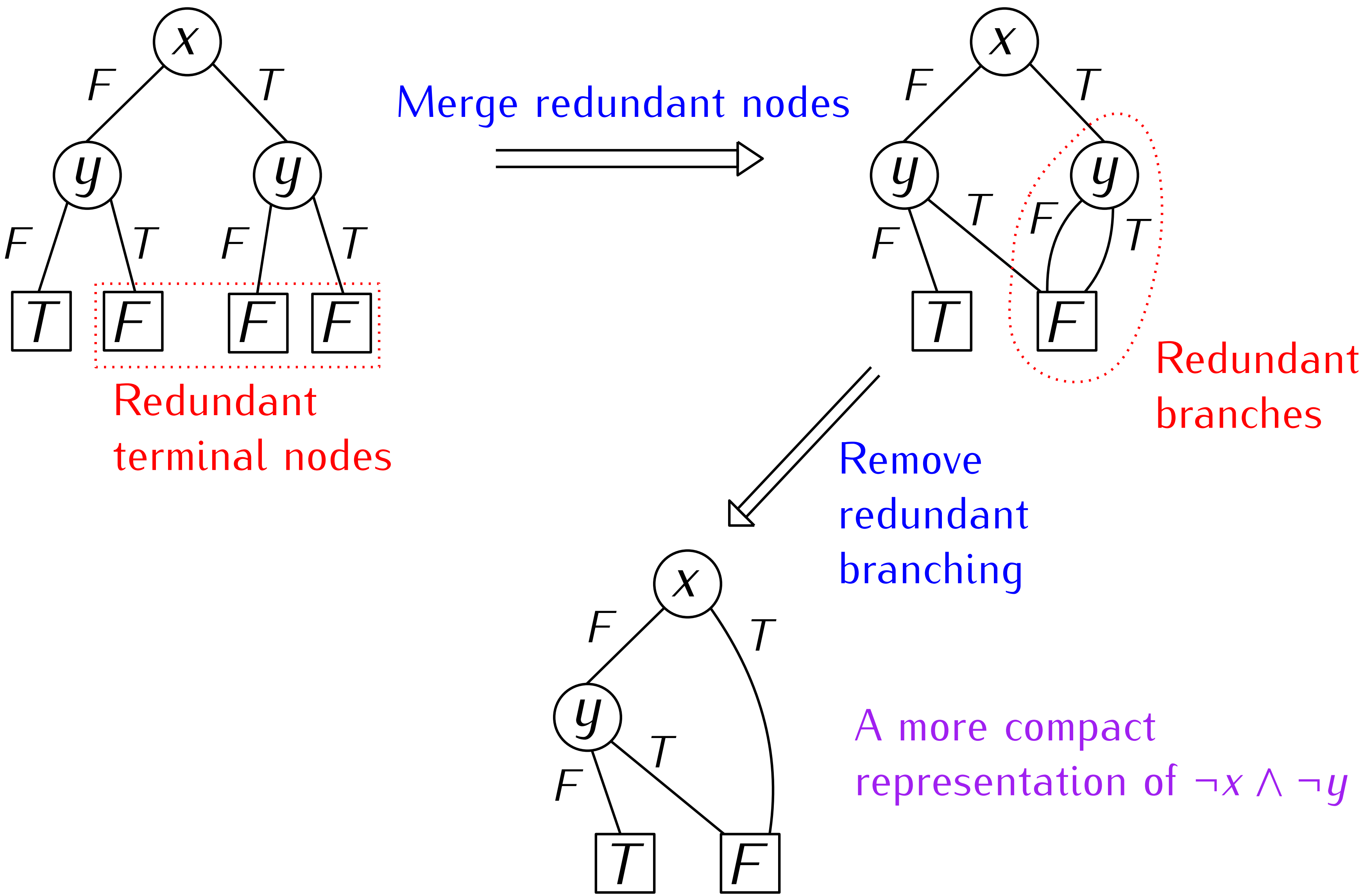
Binary Decision Diagrams

We can make the decision diagram more compact.

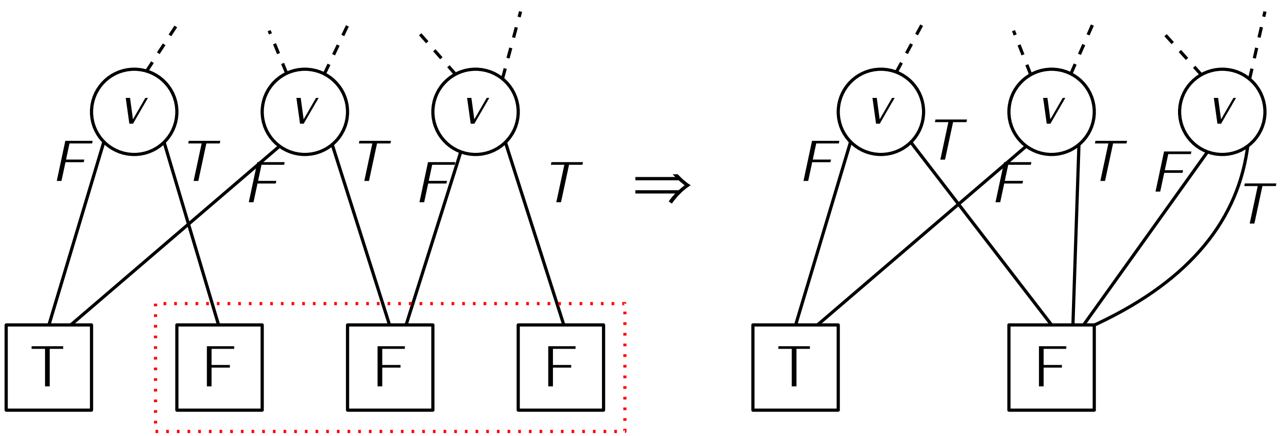


Binary Decision Diagrams

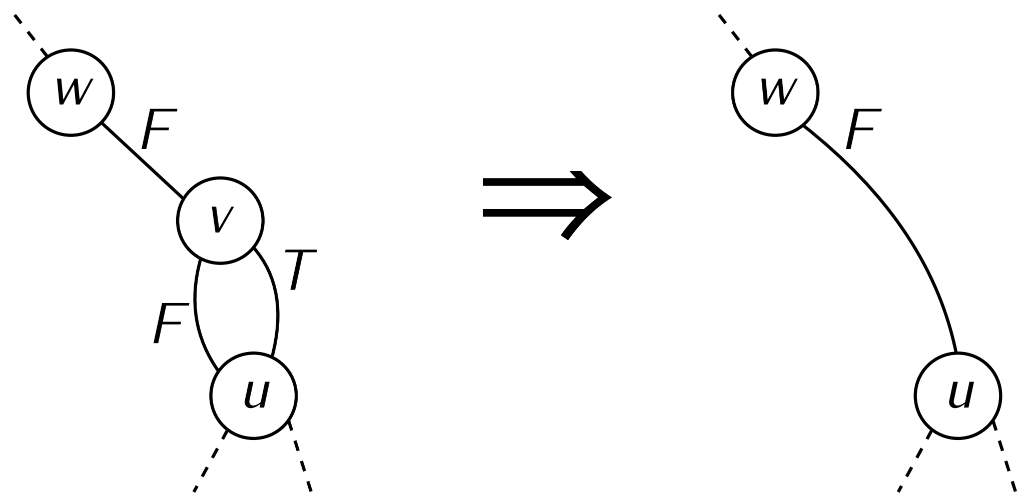
We can make the decision diagram more compact.



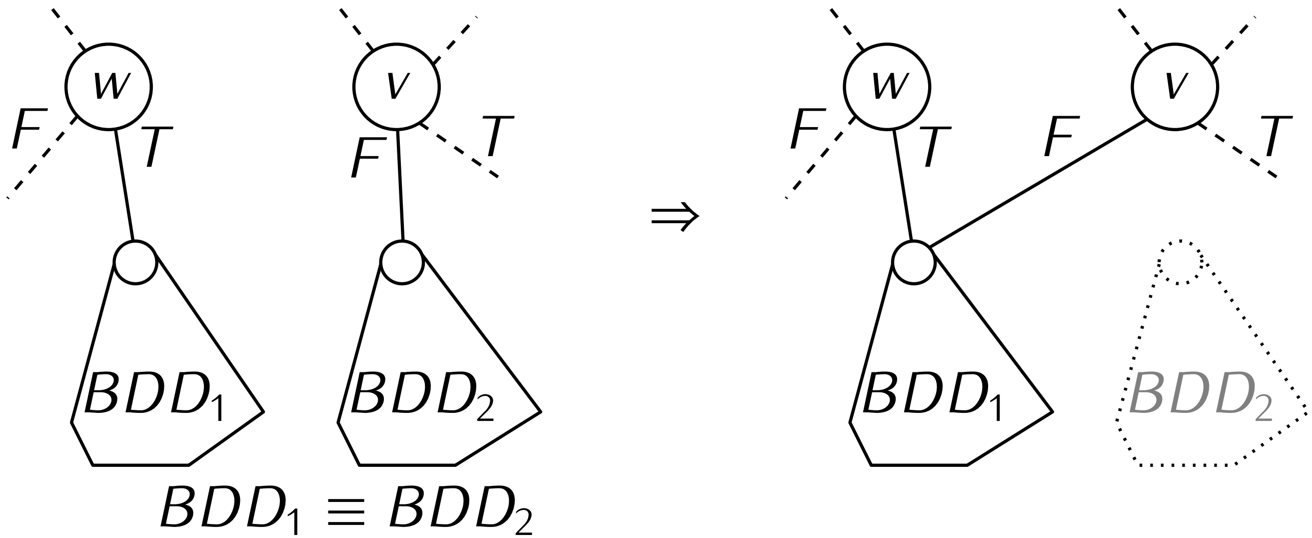
Reduction Rule 1: Merge duplicated terminal nodes.



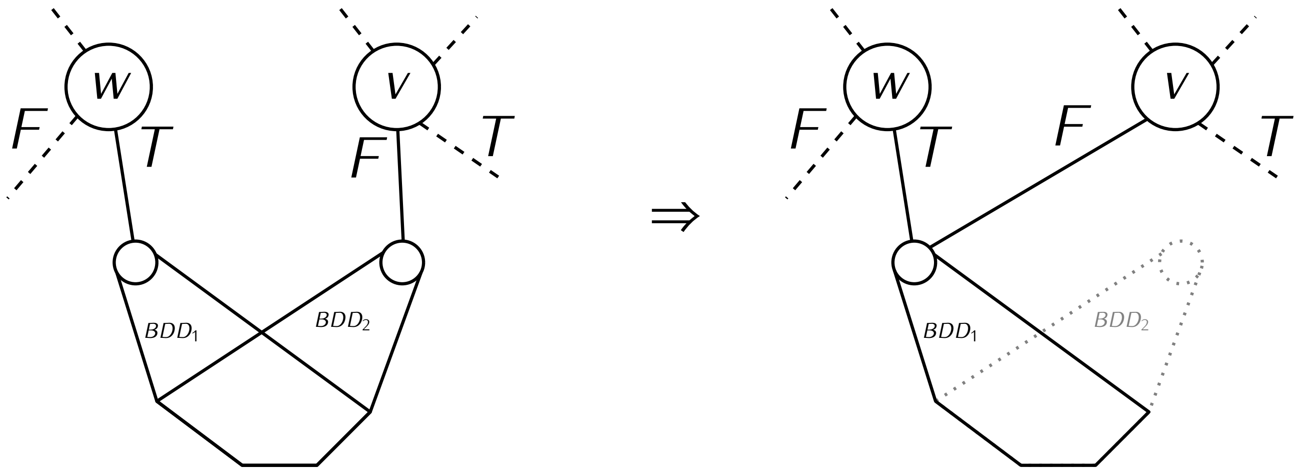
Reduction Rule 2: Remove redundant tests.



Reduction Rule 3: Remove duplicate sub-BDDs.



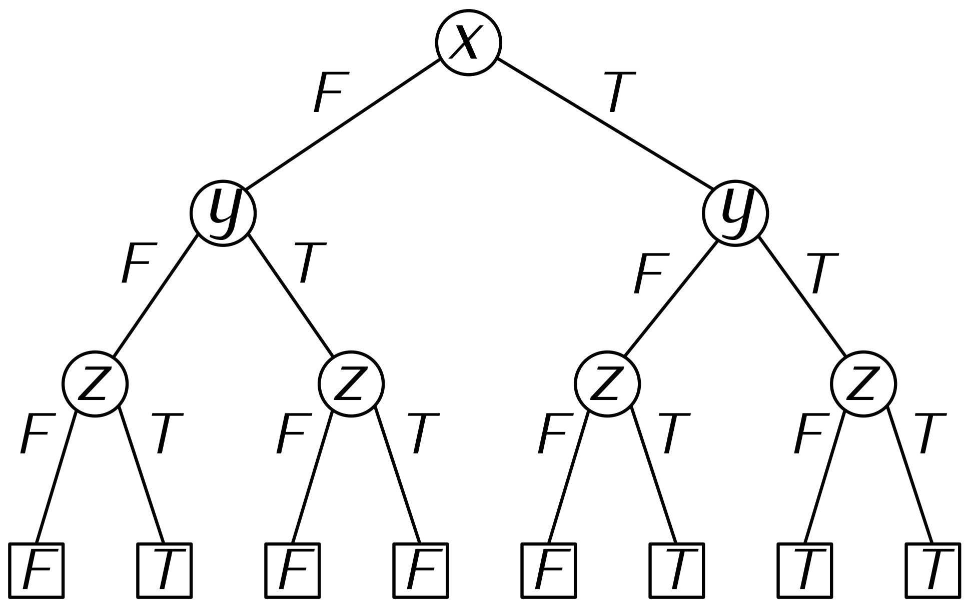
Reduction Rule 3: Remove duplicate sub-BDDs.



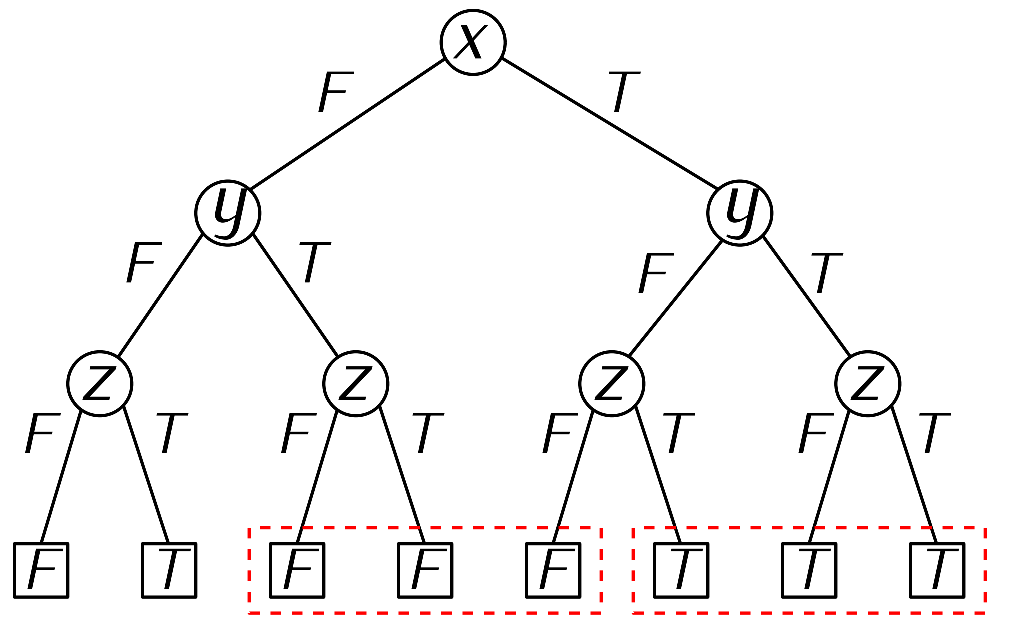
$$BDD_1 \equiv BDD_2$$

NOTE: They can be structurally identical, even if they overlap.

Example: Reduce the given BDD for the formula $(x \wedge y) \vee (\neg y \wedge z)$ to an ROBDD.

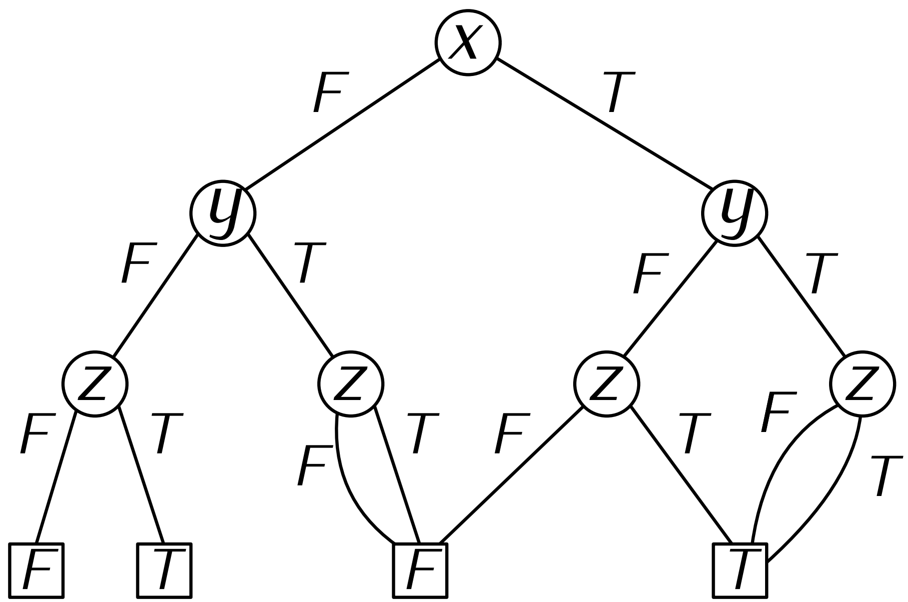


Example: Reduce the given BDD for the formula $(x \wedge y) \vee (\neg y \wedge z)$ to an ROBDD.



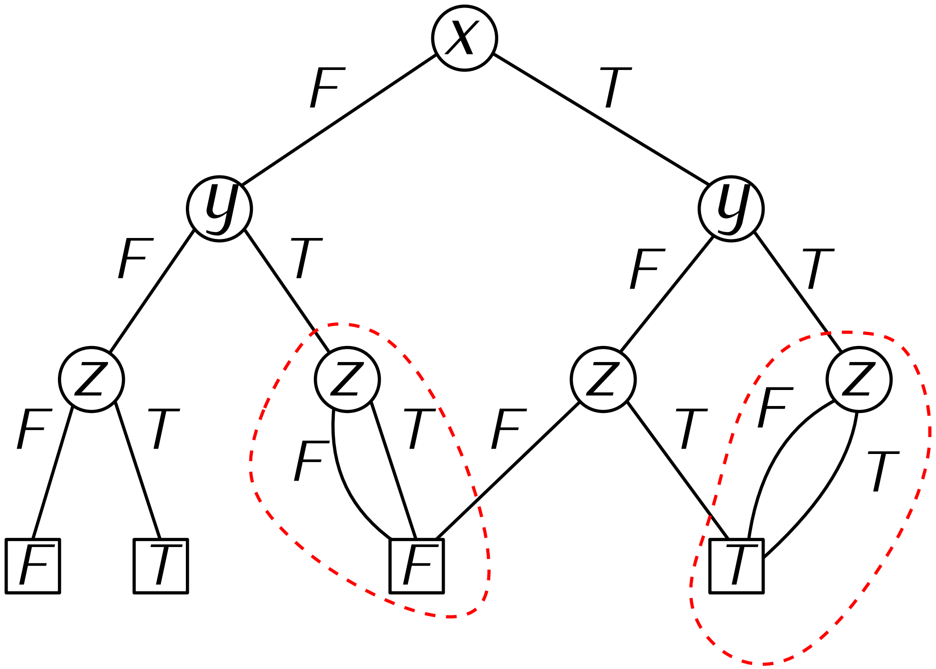
Merge duplicate terminals

Example: Reduce the given BDD for the formula $(x \wedge y) \vee (\neg y \wedge z)$ to an ROBDD.



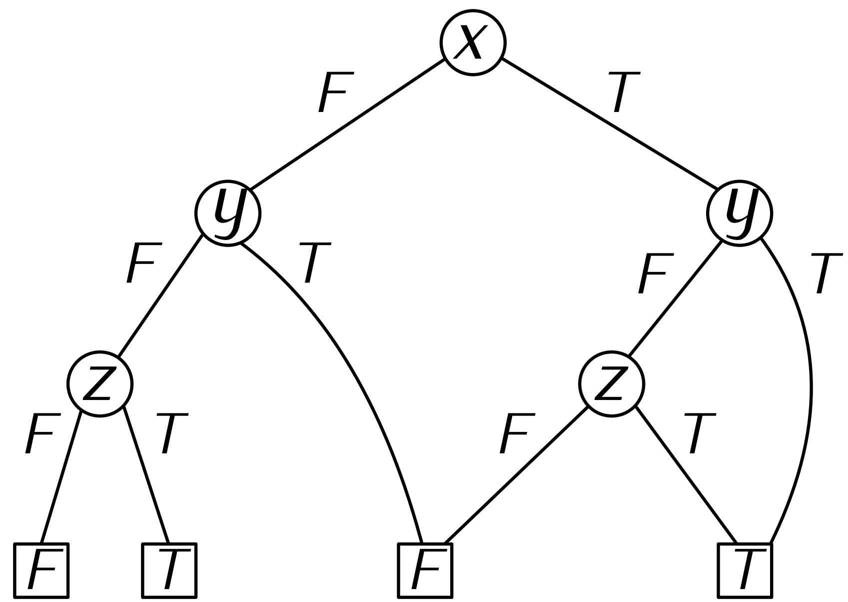
Merge duplicate terminals

Example: Reduce the given BDD for the formula $(x \wedge y) \vee (\neg y \wedge z)$ to an ROBDD.



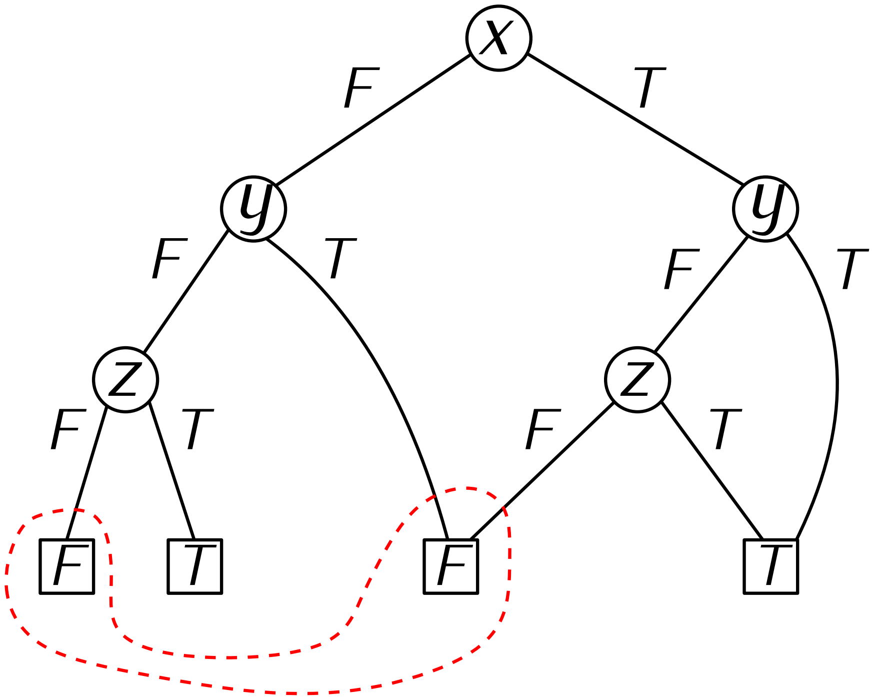
Delete redundant tests

Example: Reduce the given BDD for the formula $(x \wedge y) \vee (\neg y \wedge z)$ to an ROBDD.



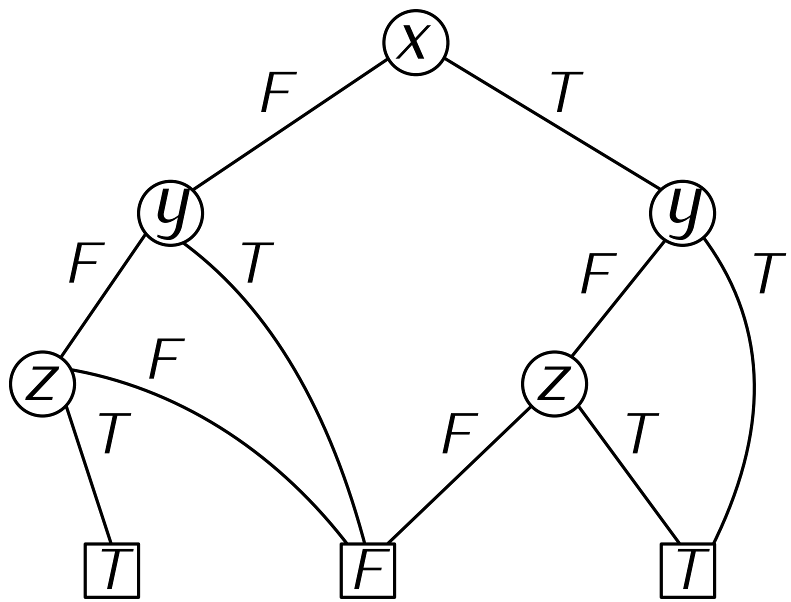
Delete redundant tests

Example: Reduce the given BDD for the formula $(x \wedge y) \vee (\neg y \wedge z)$ to an ROBDD.



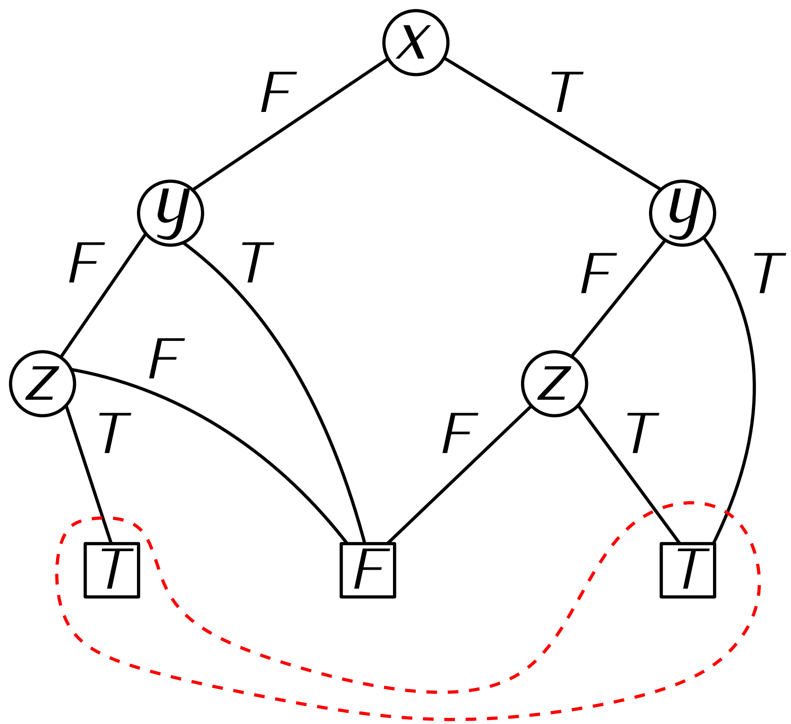
Merge more duplicate terminals

Example: Reduce the given BDD for the formula $(x \wedge y) \vee (\neg y \wedge z)$ to an ROBDD.



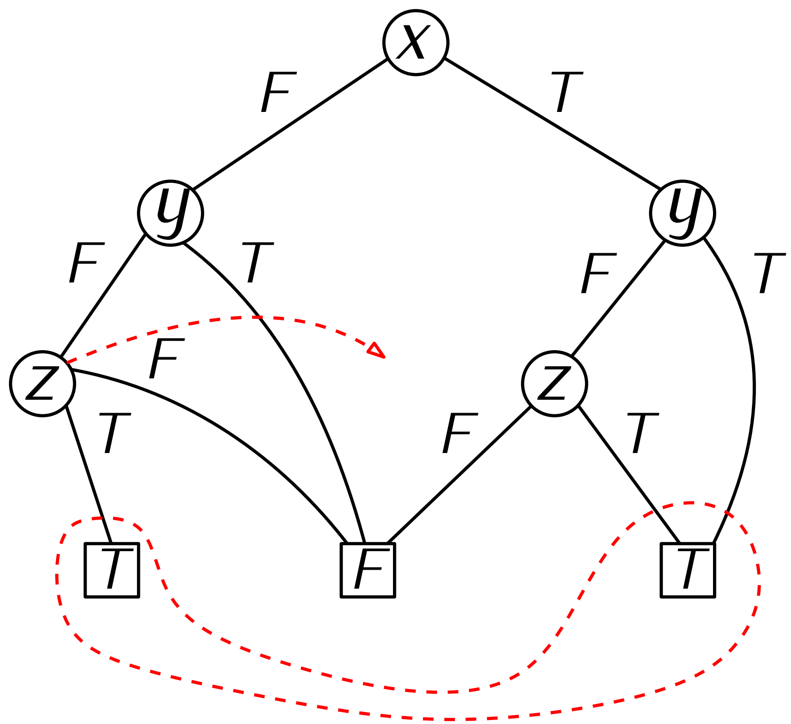
Merge more duplicate terminals

Example: Reduce the given BDD for the formula $(x \wedge y) \vee (\neg y \wedge z)$ to an ROBDD.



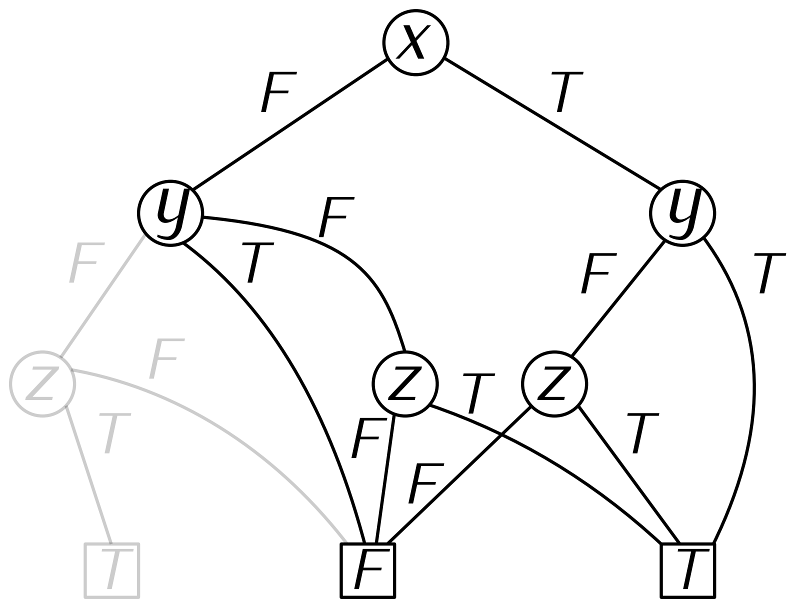
Merge more duplicate terminals

Example: Reduce the given BDD for the formula $(x \wedge y) \vee (\neg y \wedge z)$ to an ROBDD.



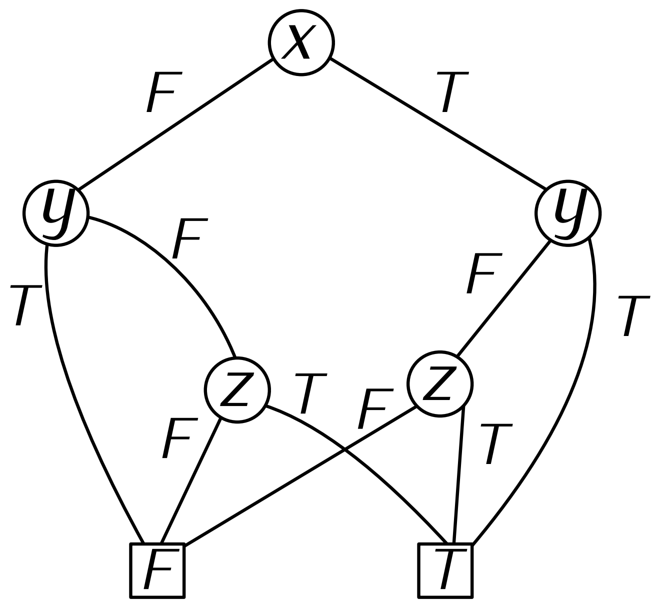
Redraw to "untangle"

Example: Reduce the given BDD for the formula $(x \land y) \lor (\neg y \land z)$ to an ROBDD.



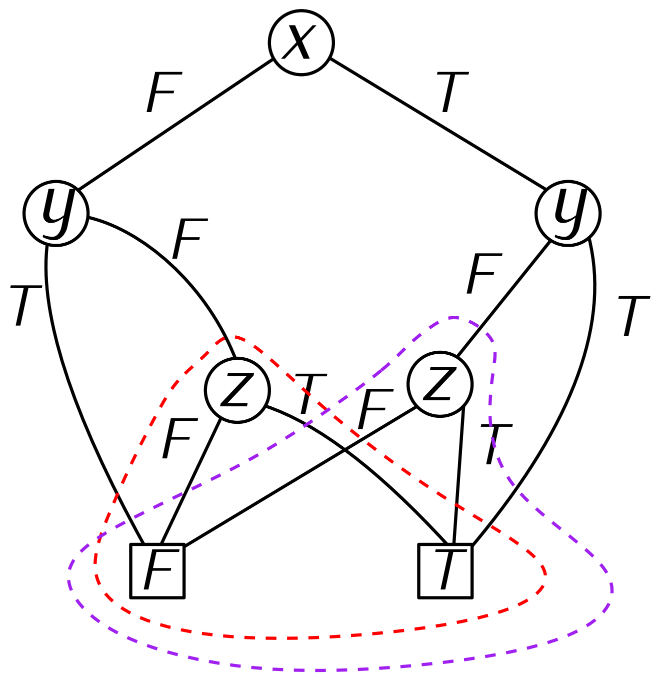
Redraw to “untangle”

Example: Reduce the given BDD for the formula $(x \wedge y) \vee (\neg y \wedge z)$ to an ROBDD.



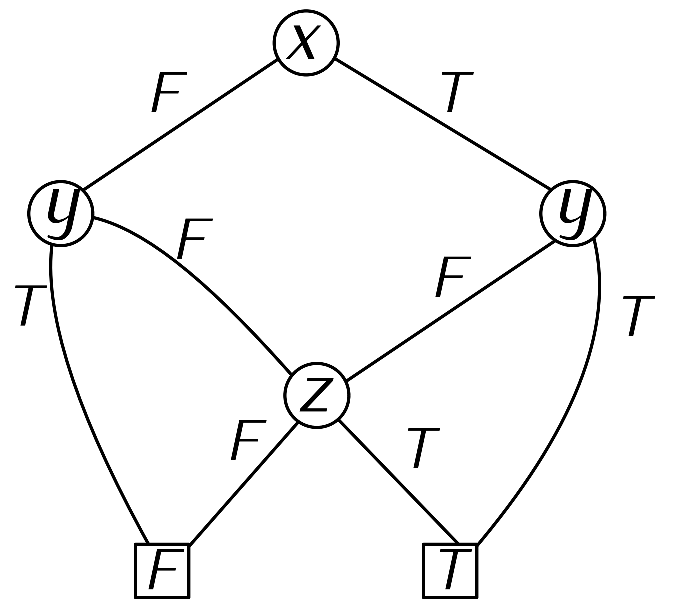
Redraw to "untangle"

Example: Reduce the given BDD for the formula $(x \land y) \lor (\neg y \land z)$ to an ROBDD.



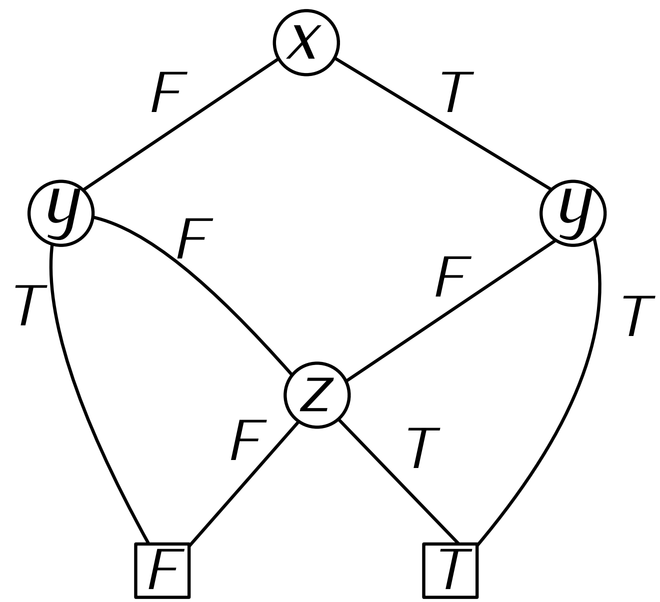
Remove duplicated sub-BDD

Example: Reduce the given BDD for the formula $(x \wedge y) \vee (\neg y \wedge z)$ to an ROBDD.



Remove duplicated sub-BDD

Example: Reduce the given BDD for the formula $(x \wedge y) \vee (\neg y \wedge z)$ to an ROBDD.



No more reduction possible

Important properties of BDDs

Ordered BDD (OBDD): variables are checked in a given order. E.g $x > y > z$.

Reduced OBDD (ROBDD): Cannot be reduced any further.

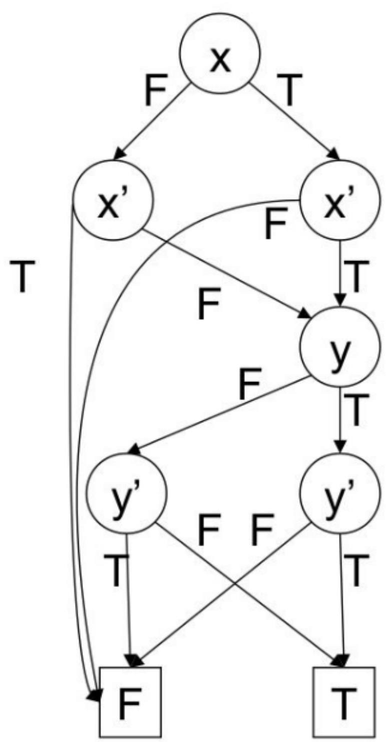
Theorem: ROBDDs are **unique** for a given ordering.

Important properties of BDDs

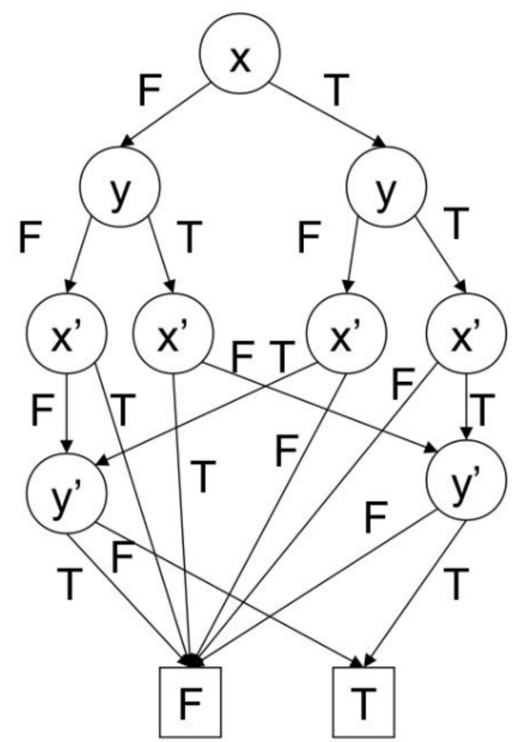
ROBDD **size** is sensitive to variable ordering!

Two different ROBDDs for the formula
$x' \Leftrightarrow x \wedge y' \Leftrightarrow y$

Variable order: x, x', y, y'



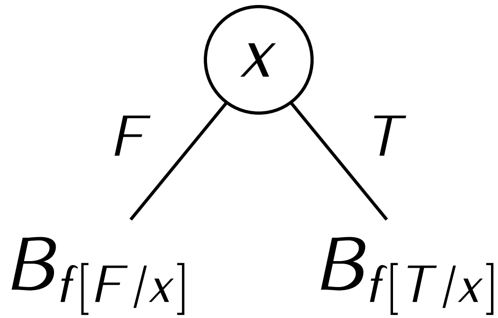
Variable order: x, y, x', y'



BDDs via Shannon Expansion

$$f \equiv (x = F) \wedge f[F/x] \;\vee\; (x = T) \wedge f[T/x]$$

For now, recursively compute $\longrightarrow$



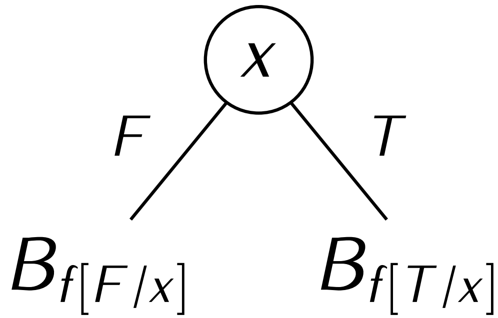
$\longrightarrow$ then reduce

In class example: $f \equiv \neg x \vee \neg y$

BDDs via Shannon Expansion

$$f \equiv (x = F) \wedge f[F/x] \;\vee\; (x = T) \wedge f[T/x]$$

For now, recursively compute $\longrightarrow$



$\longrightarrow$ then reduce

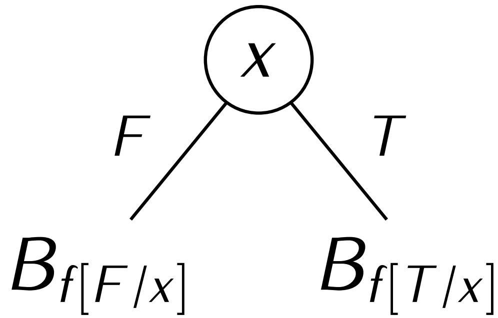
$$f \equiv (x_1 \vee x_2) \wedge x_3$$

$$g \equiv (x_1 \wedge \neg x_2)$$

BDDs via Shannon Expansion

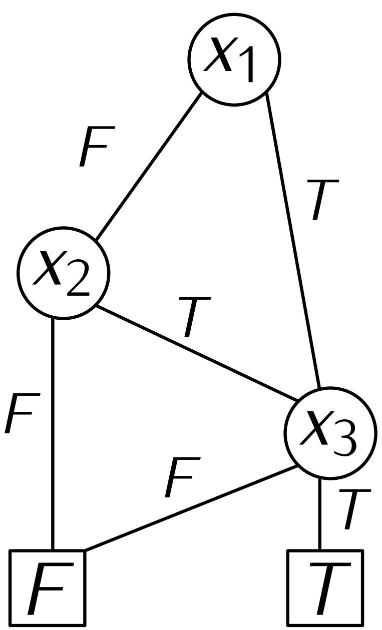
$$f \equiv (x = F) \wedge f[F/x] \;\vee\; (x = T) \wedge f[T/x]$$

For now, recursively compute $\longrightarrow$

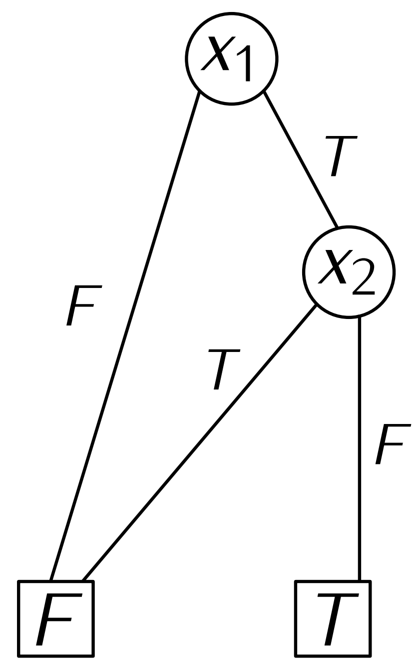


$\longrightarrow$ then reduce

$$f \equiv (x_1 \vee x_2) \wedge x_3$$



$$g \equiv (x_1 \wedge \neg x_2)$$



Symbolic Model Checking: 10^{20} States and Beyond

J. R. Burch E. M. Clarke K. L. McMillan*
School of Computer Science
Carnegie Mellon University

D. L. Dill L. J. Hwang
Stanford University

Abstract

Many different methods have been devised for automatically verifying finite state systems by examining state-graph models of system behavior. These methods all depend on decision procedures that explicitly represent the state space using a list or a table that grows in proportion to the number of states. We describe a general method that represents the state space *symbolically* instead of explicitly. The generality of our method comes from using a dialect of the Mu-Calculus as the primary specification language. We describe a *model checking* algorithm for Mu-Calculus formulas that uses Bryant's *Binary Decision Diagrams* (1986) to represent relations and formulas. We then show how our new Mu-Calculus model checking algorithm can be used to derive efficient decision procedures for CTL model checking, satisfiability of linear-time temporal logic formulas, strong and weak observational equivalence of finite transition systems, and language containment for finite ω-automata. The fixed point computations for each decision procedure are sometimes complex, but can be concisely expressed in the Mu-Calculus. We illustrate the practicality of our approach to symbolic model checking by discussing how it can be used to verify a simple synchronous pipeline circuit.

Useful things to do with BDDs

Imagine that you have two Boolean logic formulas f and g, and you also have BDDs for each of them, say B_f and B_g. How would you accomplish the following?

- Test if f is a tautology
- Test if f is a satisfiable
- Test if $f \equiv g$
- Compute the BDD for $\neg f$
- Compute the BDD for $f \wedge g$
- Compute the BDD for $f \vee g$

Binary ops on B_f and B_g

Given f and g, let's compute $B_{f \star g}$ from B_f and B_g, where $\star \in \{\wedge, \vee\}$.

Let $\mathrm{apply}(\star, B_f, B_g)$ be the function that computes $B_{f \star g}$.

We will compute $\mathrm{apply}(\star, B_f, B_g)$ recursively.

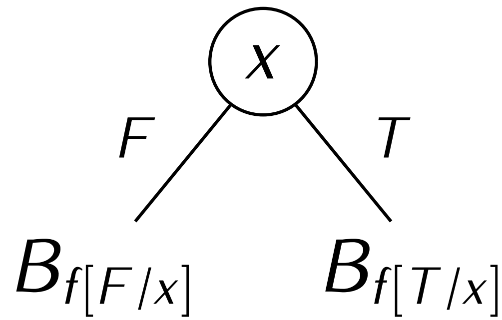
What are the base cases?

$$\mathrm{apply}(\star, \boxed{t_1}, \boxed{t_2}) = \boxed{t_1 \star t_2}$$

Binary ops on B_f and B_g: A **bunch** of recursive cases:

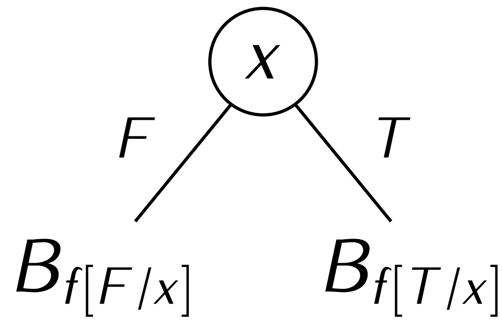
Binary ops on B_f **and** B_g**:** A **bunch** of recursive cases:

$\mathrm{apply}(\star,$

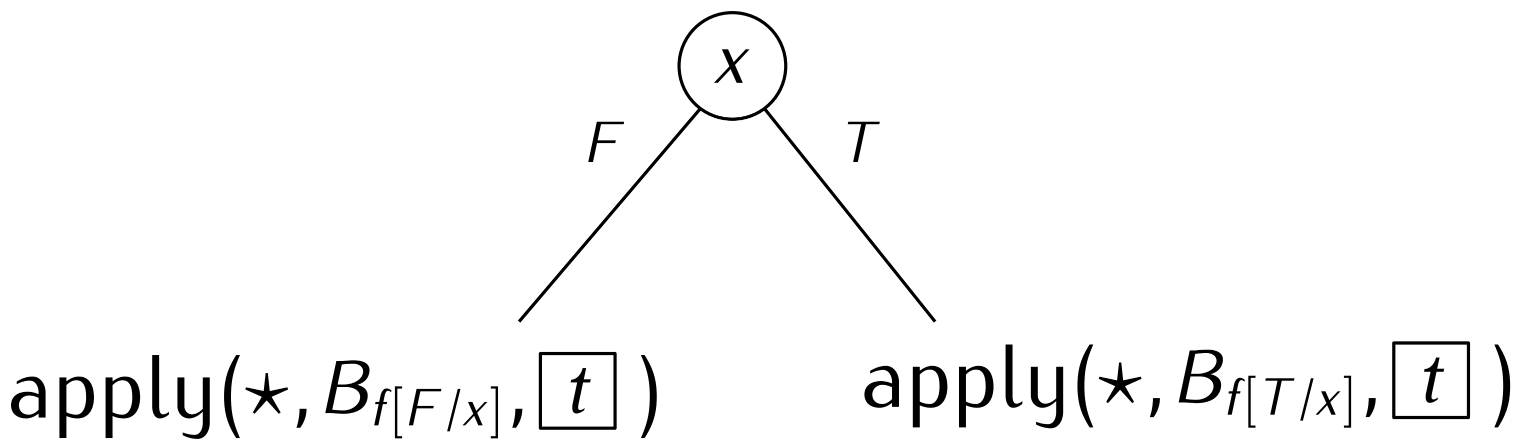


$, \boxed{t}\ ) =$

Binary ops on B_f and B_g: A **bunch** of recursive cases:

apply($\star$, 

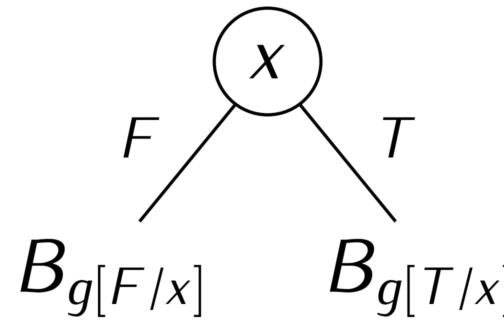
, $\boxed{t}$) =



Rule 1a

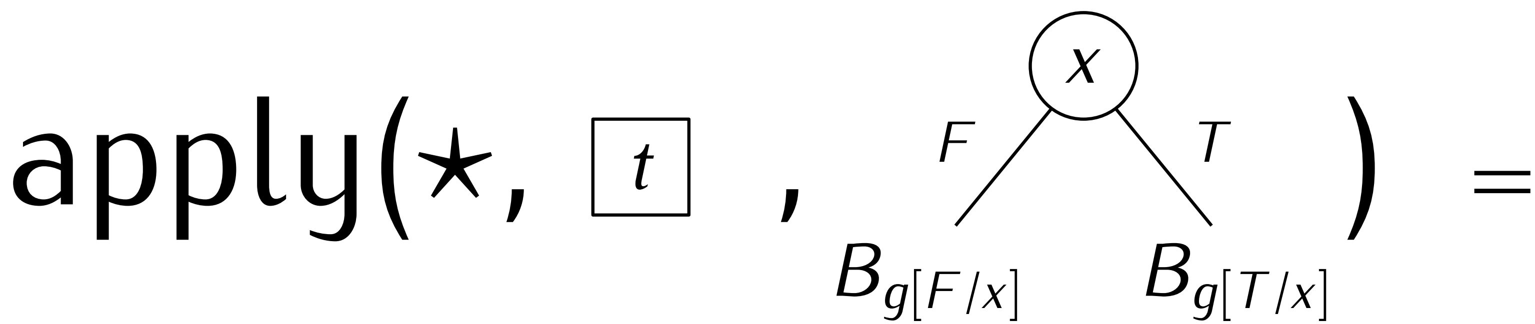
Binary ops on B_f **and** B_g**:** A **bunch** of recursive cases:

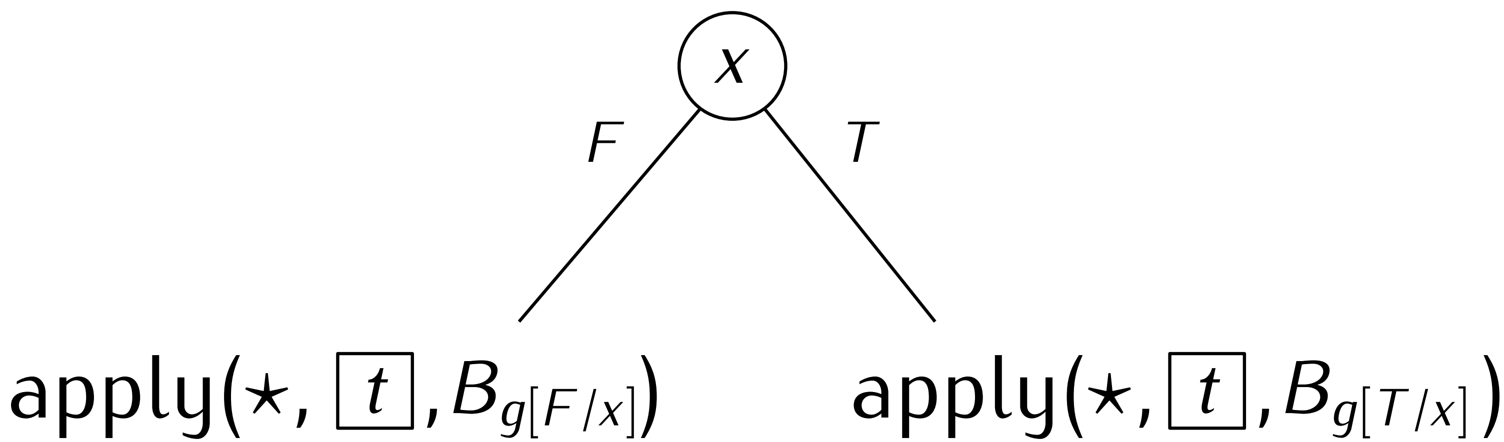
apply($\star$, $\boxed{t}$,



) =

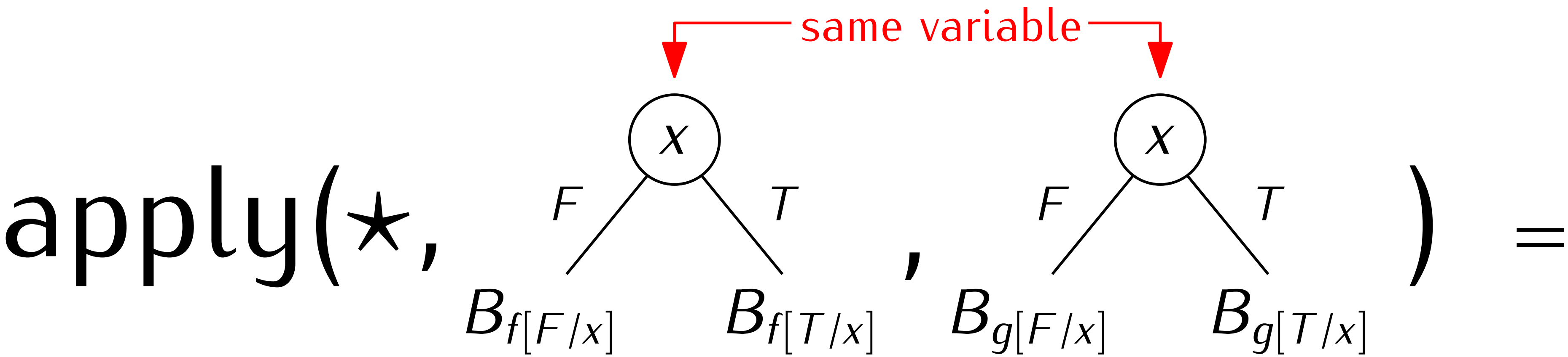
Binary ops on B_f and B_g: A **bunch** of recursive cases:



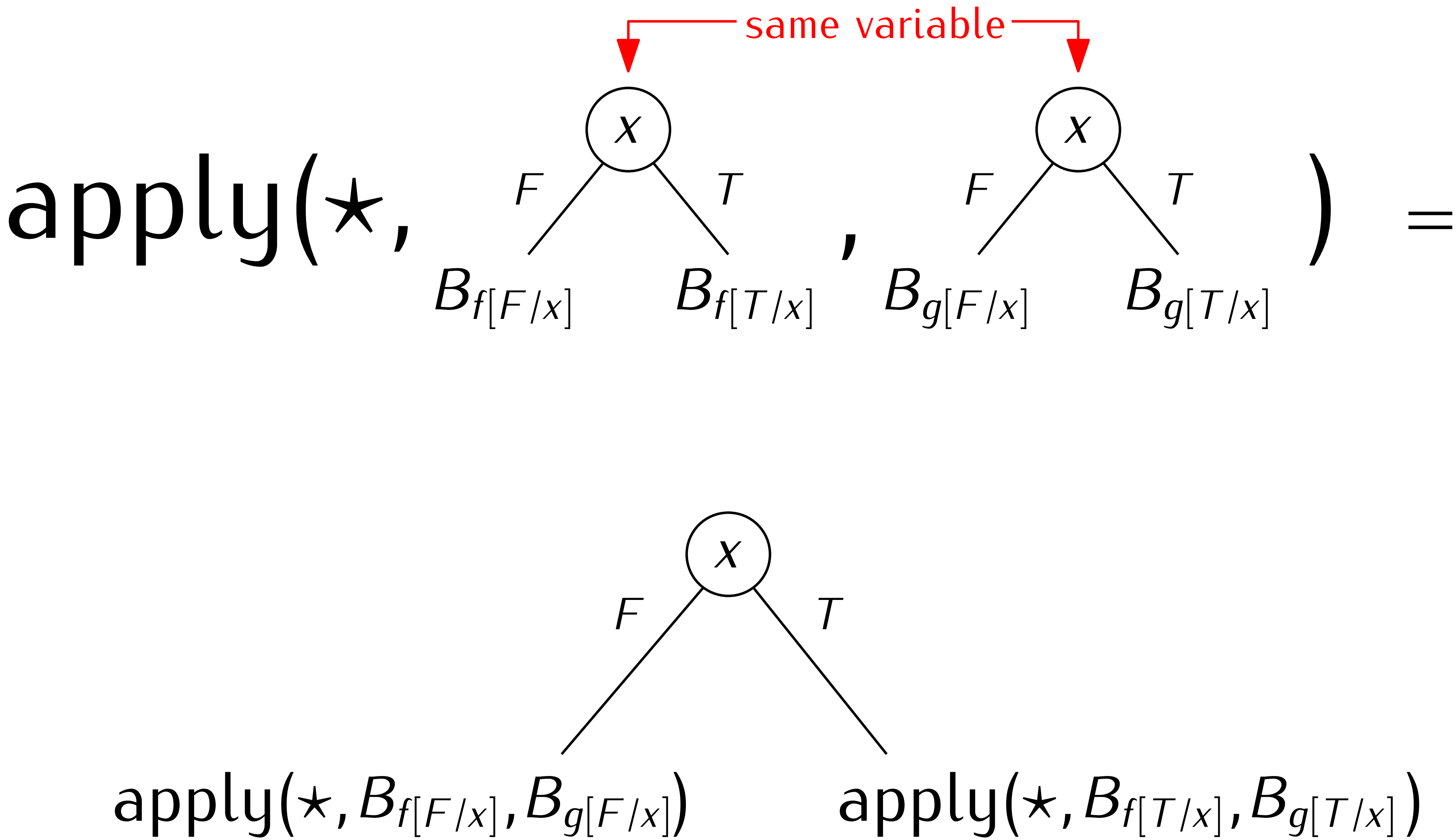


Rule 1b

Binary ops on B_f and B_g: A **bunch** of recursive cases:

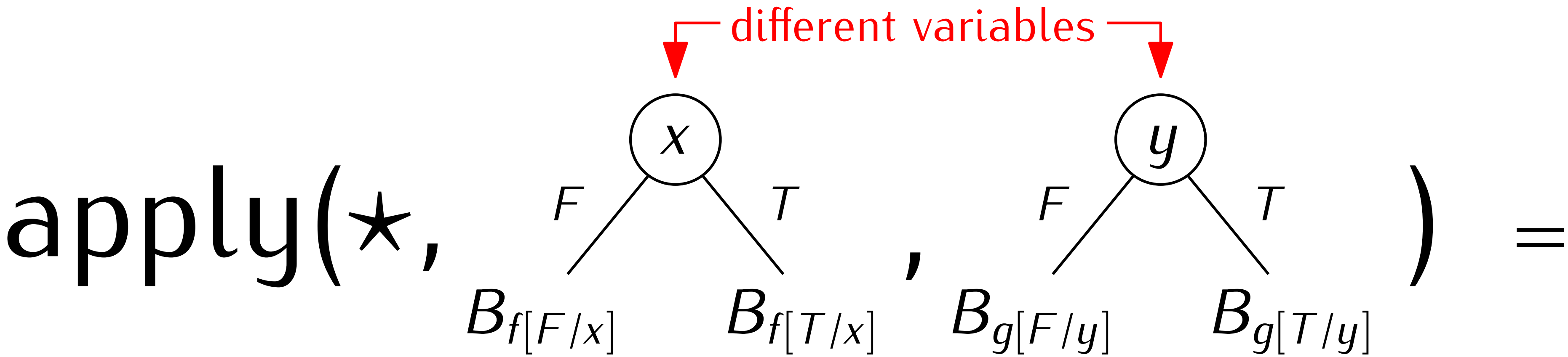


Binary ops on B_f and B_g: A **bunch** of recursive cases:

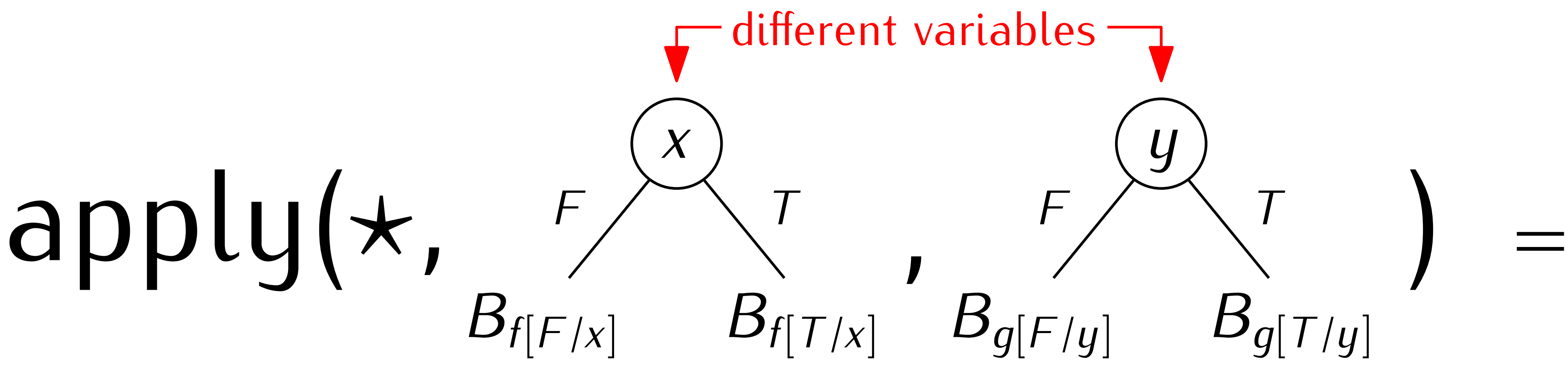


Rule 2

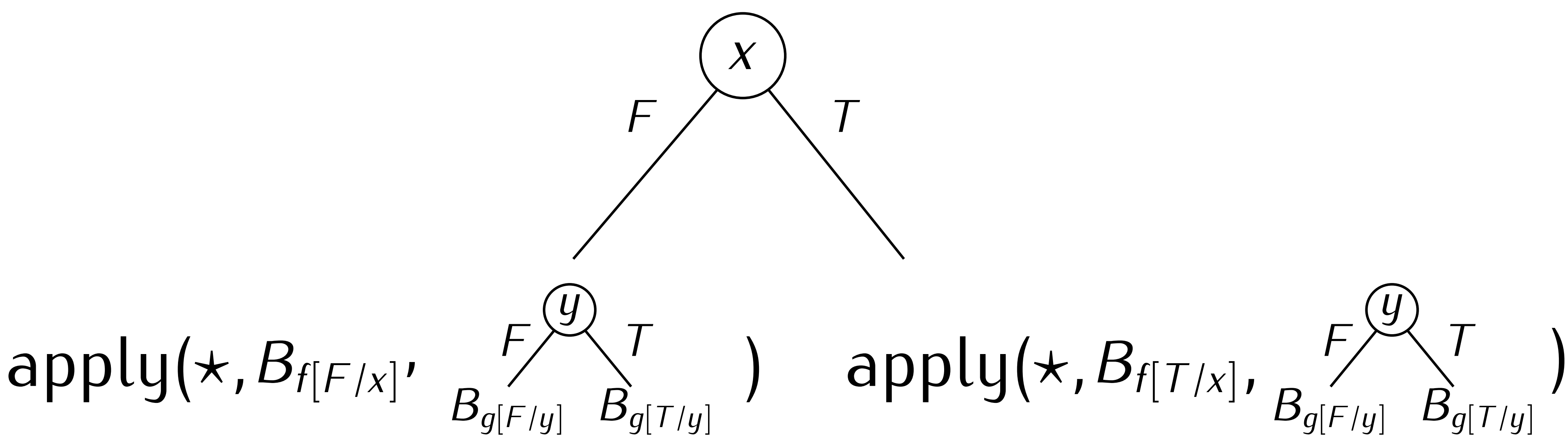
Binary ops on B_f **and** B_g**:** A **bunch** of recursive cases:



Binary ops on B_f and B_g: A **bunch** of recursive cases:



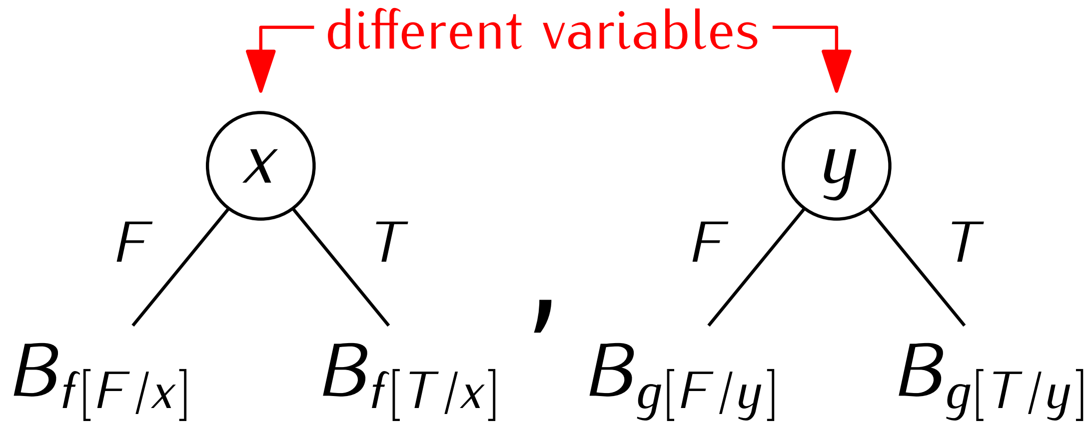
If $x > y$ in the BDD ordering:



Rule 3a

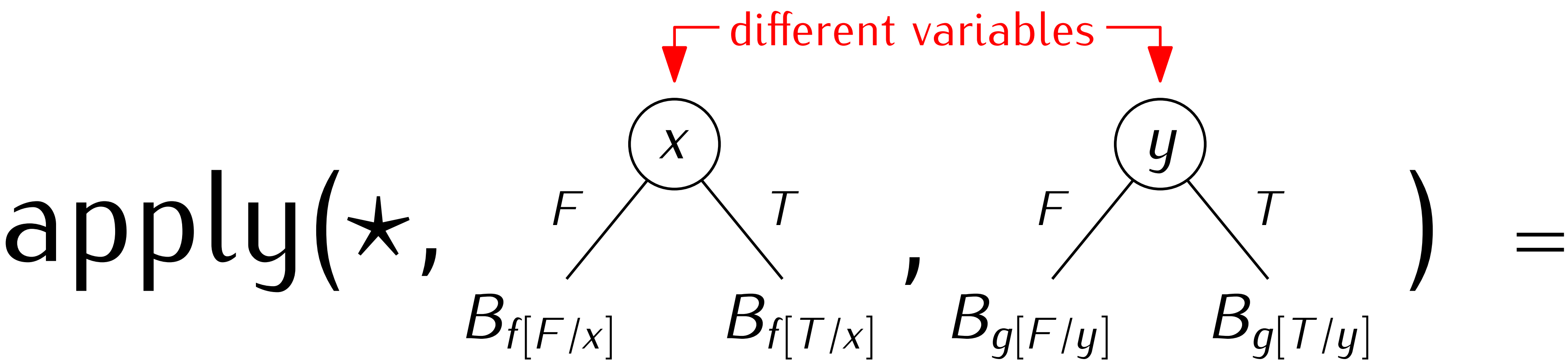
Binary ops on B_f **and** B_g**:** A **bunch** of recursive cases:

$\text{apply}(\star,$

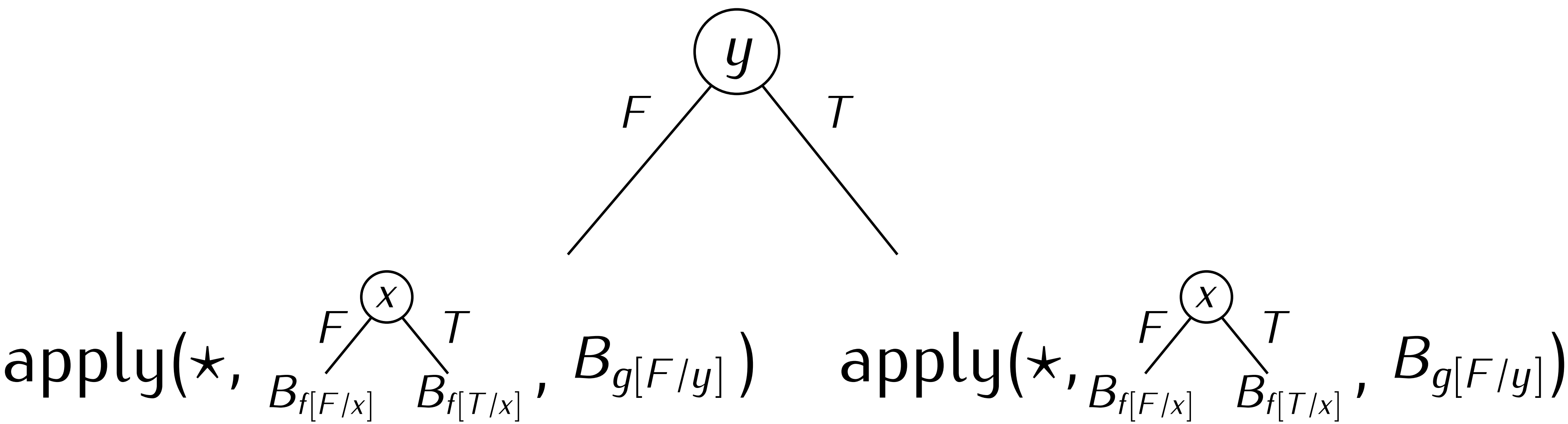


$) =$

Binary ops on B_f and B_g: A **bunch** of recursive cases:



If $y > x$ in the BDD ordering:



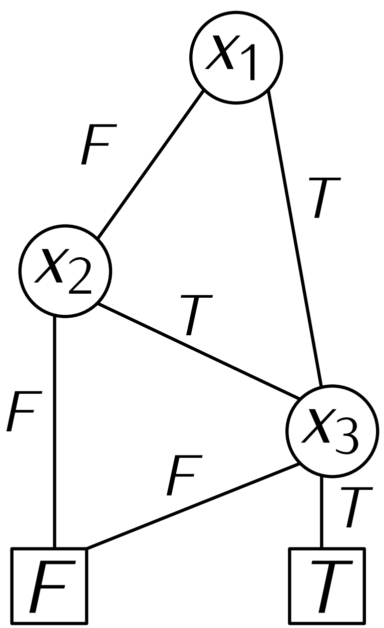
Rule 3b

Binary ops on B_f and B_g:

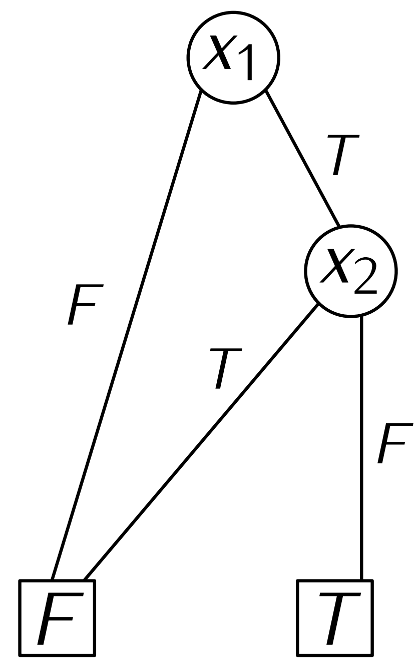
Binary ops on B_f and B_g:

Compute $B_{f \vee g}$ using B_f and B_g

$f \equiv (x_1 \vee x_2) \wedge x_3$



$g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

Binary ops on B_f and B_g:

Compute $B_{f \vee g}$ using B_f and B_g

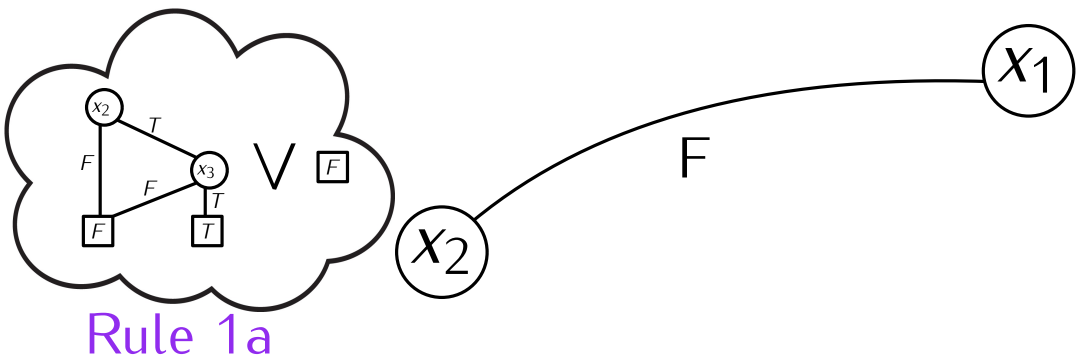
$f \equiv (x_1 \vee x_2) \wedge x_3$ $\qquad$ $g \equiv (x_1 \wedge \neg x_2)$

x_1

Binary ops on B_f and B_g:

Compute $B_{f \vee g}$ using B_f and B_g

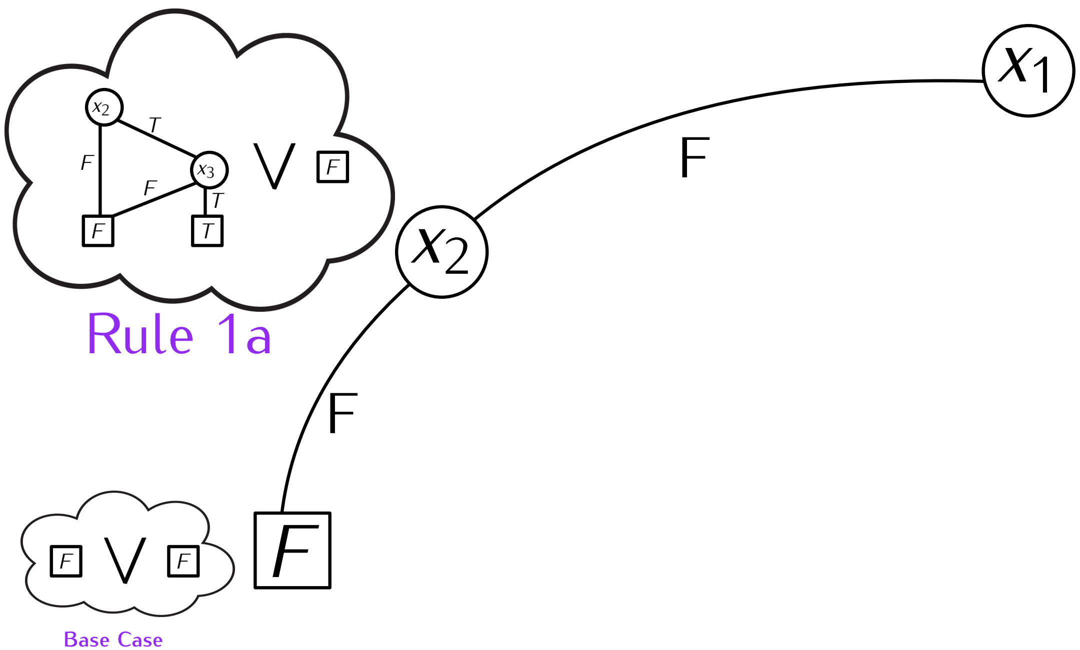
$f \equiv (x_1 \vee x_2) \wedge x_3$ $\qquad$ $g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

Compute $B_{f \vee g}$ using B_f and B_g

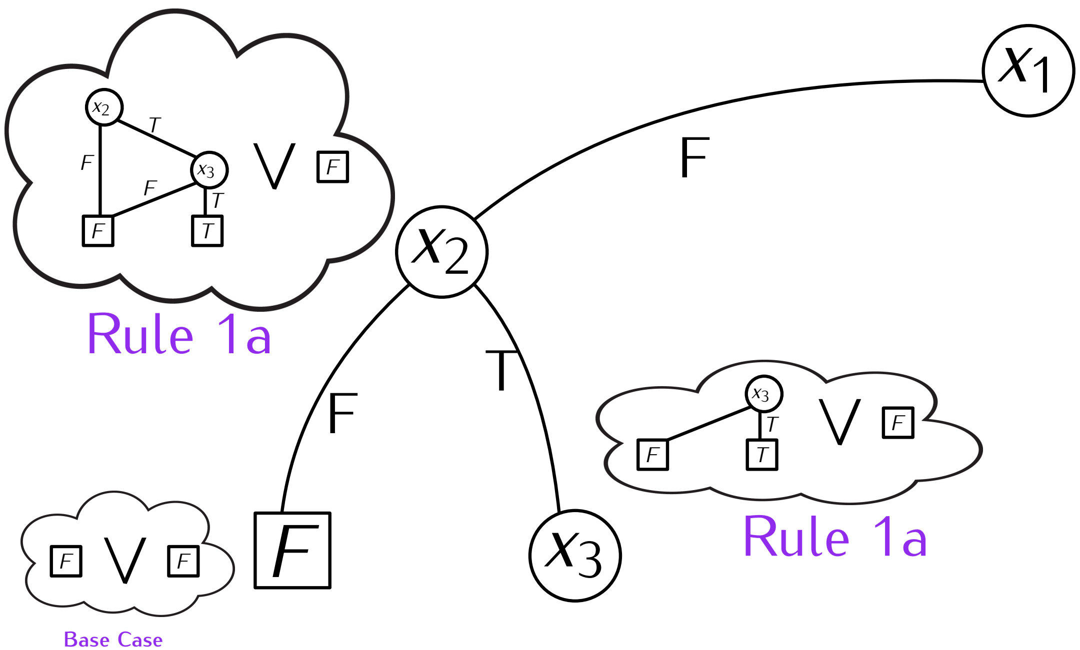
$f \equiv (x_1 \vee x_2) \wedge x_3$ $\qquad g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

Compute $B_{f \vee g}$ using B_f and B_g

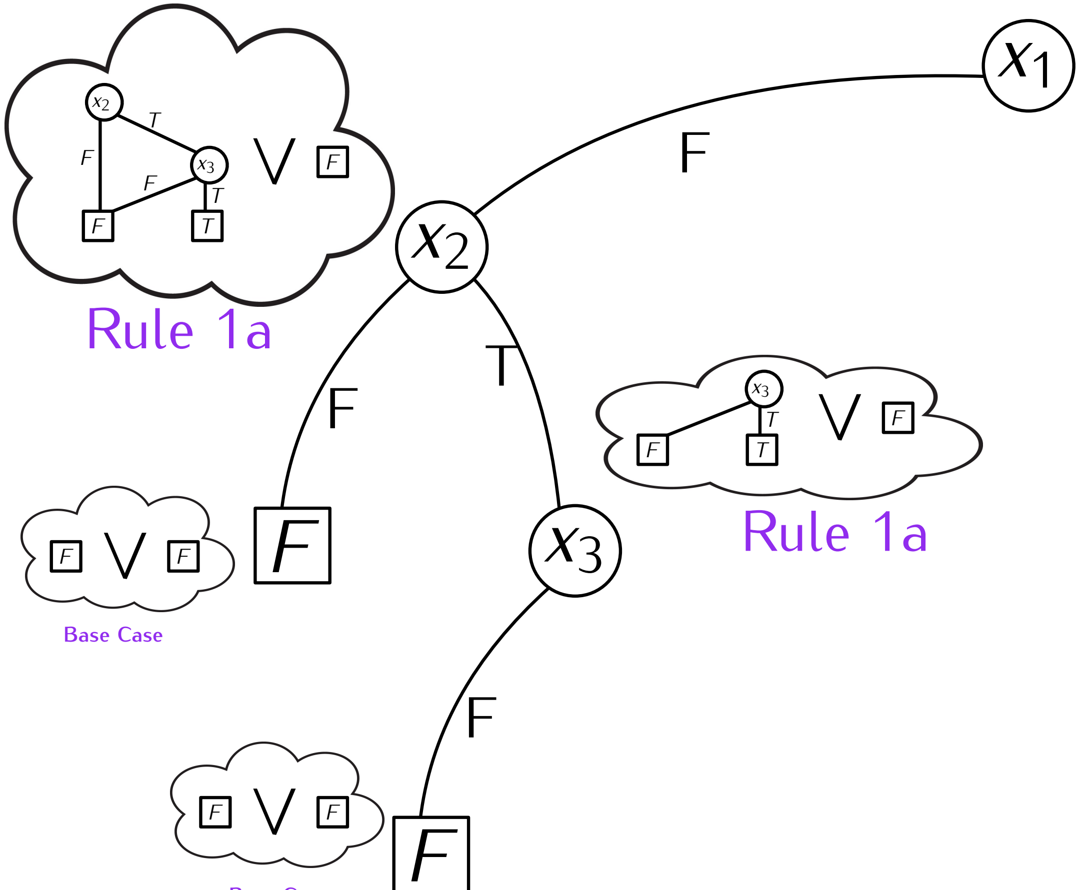
$f \equiv (x_1 \vee x_2) \wedge x_3$ $\qquad$ $g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

Compute $B_{f \vee g}$ using B_f and B_g

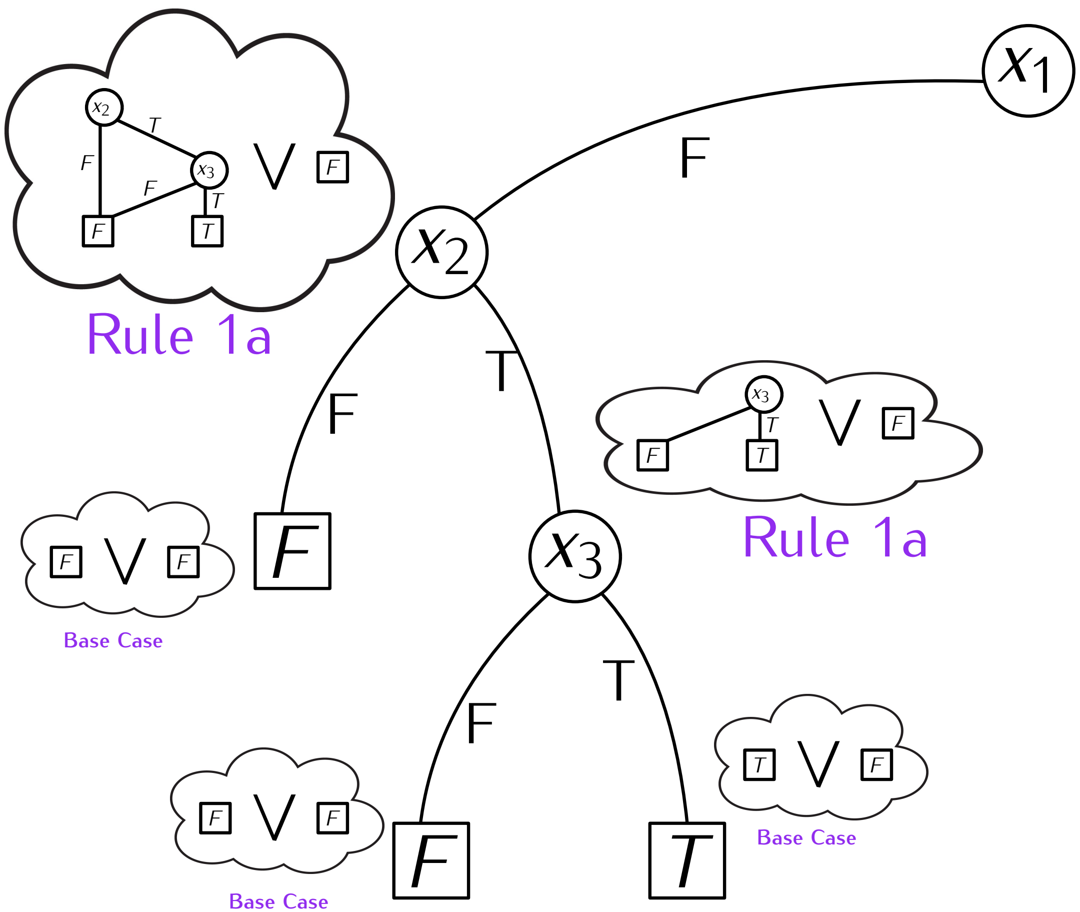
$f \equiv (x_1 \vee x_2) \wedge x_3$ $\qquad$ $g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

Compute $B_{f \vee g}$ using B_f and B_g

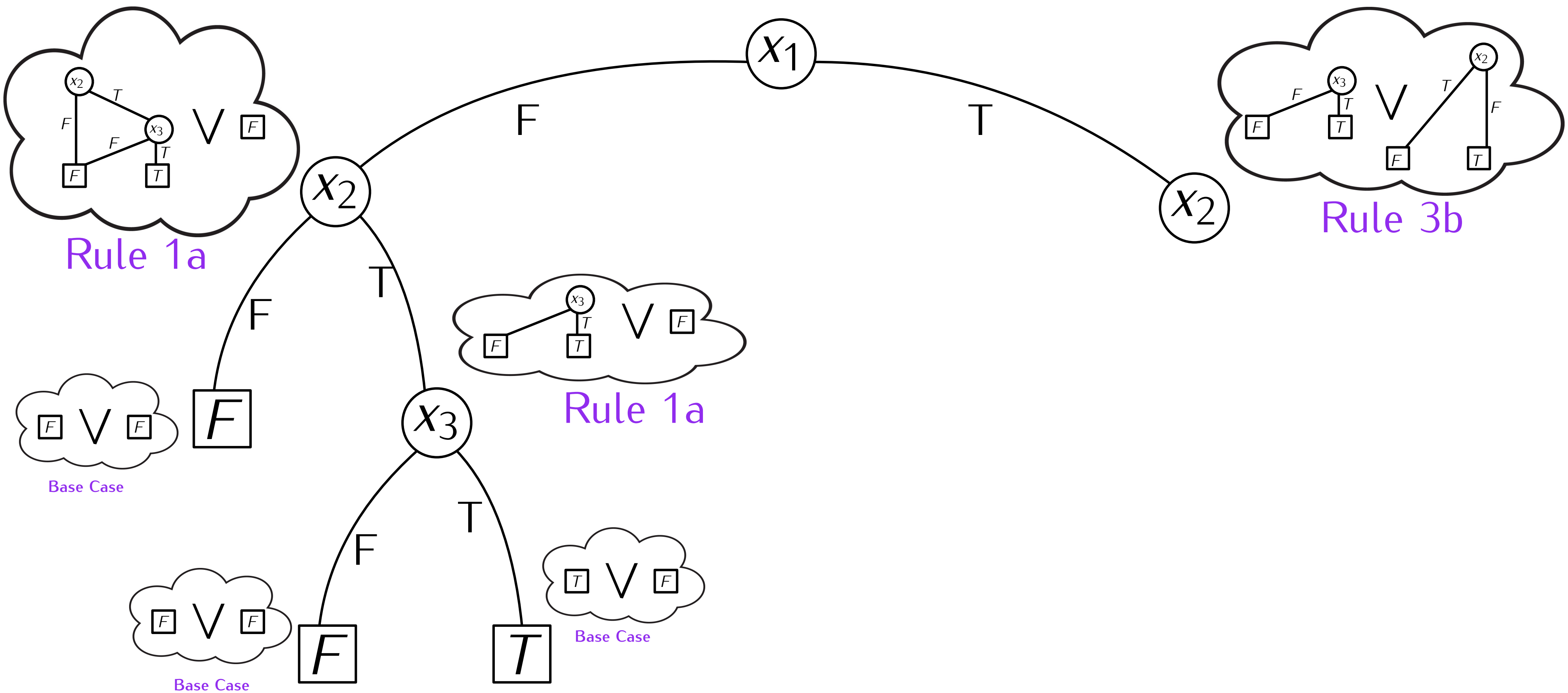
$f \equiv (x_1 \vee x_2) \wedge x_3$ $\qquad$ $g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

Compute $B_{f \vee g}$ using B_f and B_g

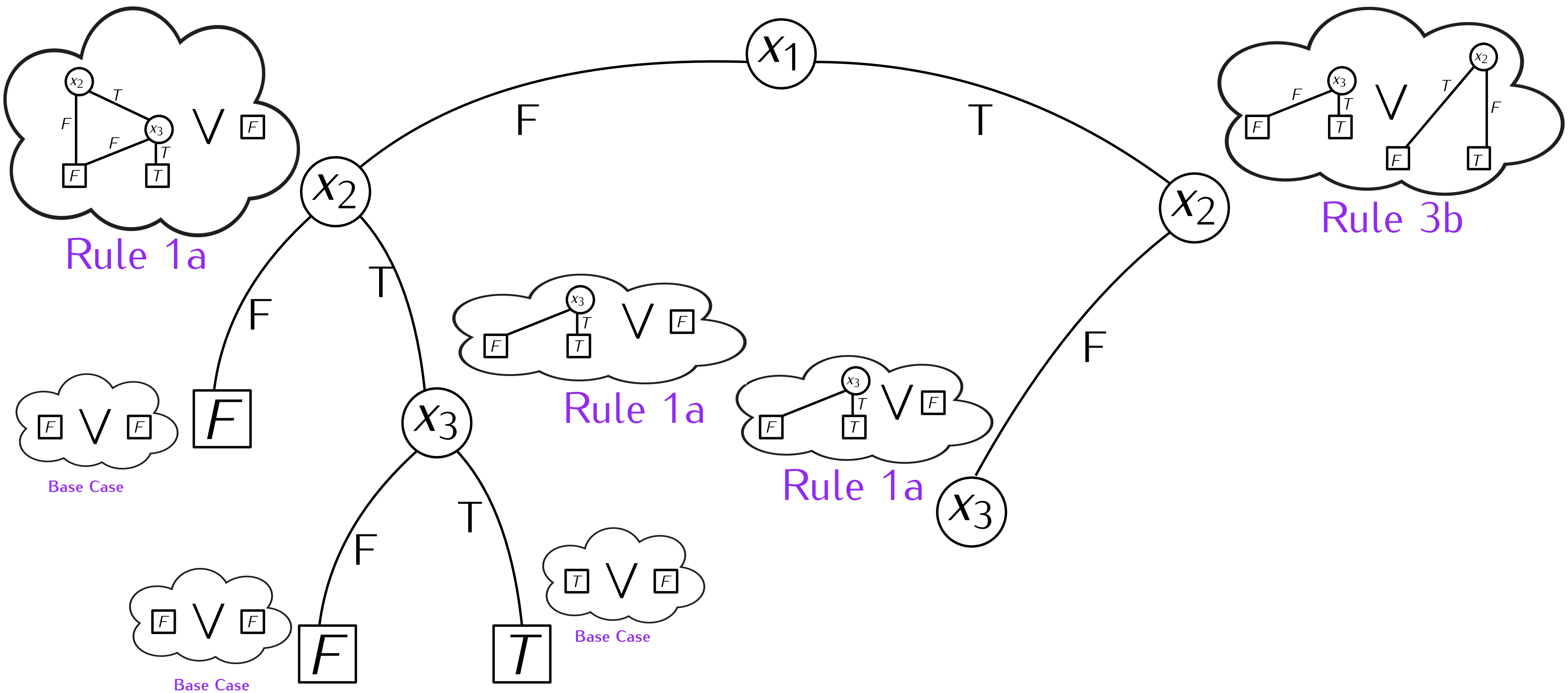
$f \equiv (x_1 \vee x_2) \wedge x_3$ $\qquad$ $g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

Compute $B_{f \vee g}$ using B_f and B_g

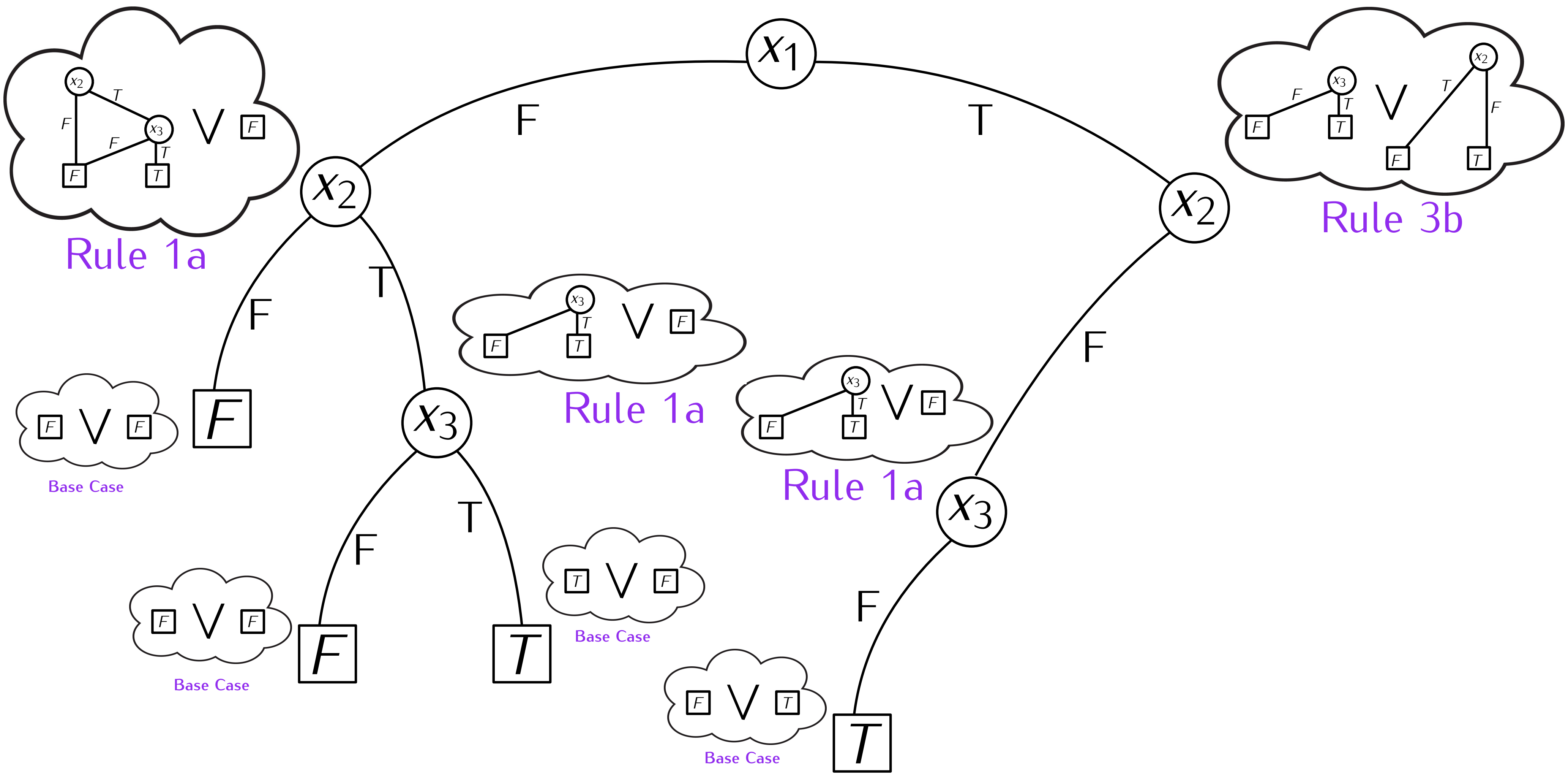
$f \equiv (x_1 \vee x_2) \wedge x_3$ $\qquad$ $g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

Compute $B_{f\vee g}$ using B_f and B_g

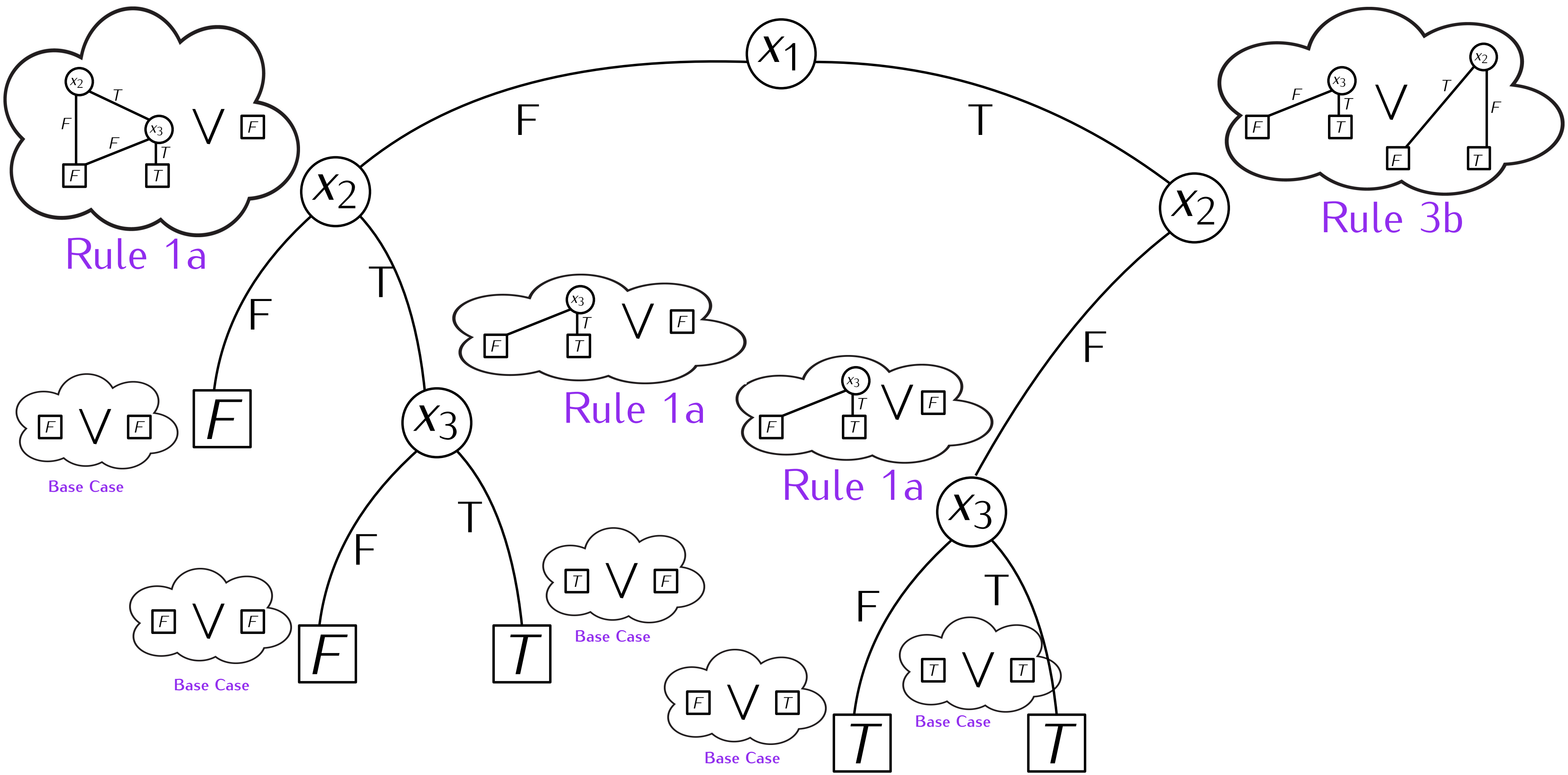
$f \equiv (x_1 \vee x_2) \wedge x_3$ $\qquad$ $g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

Compute $B_{f \vee g}$ using B_f and B_g

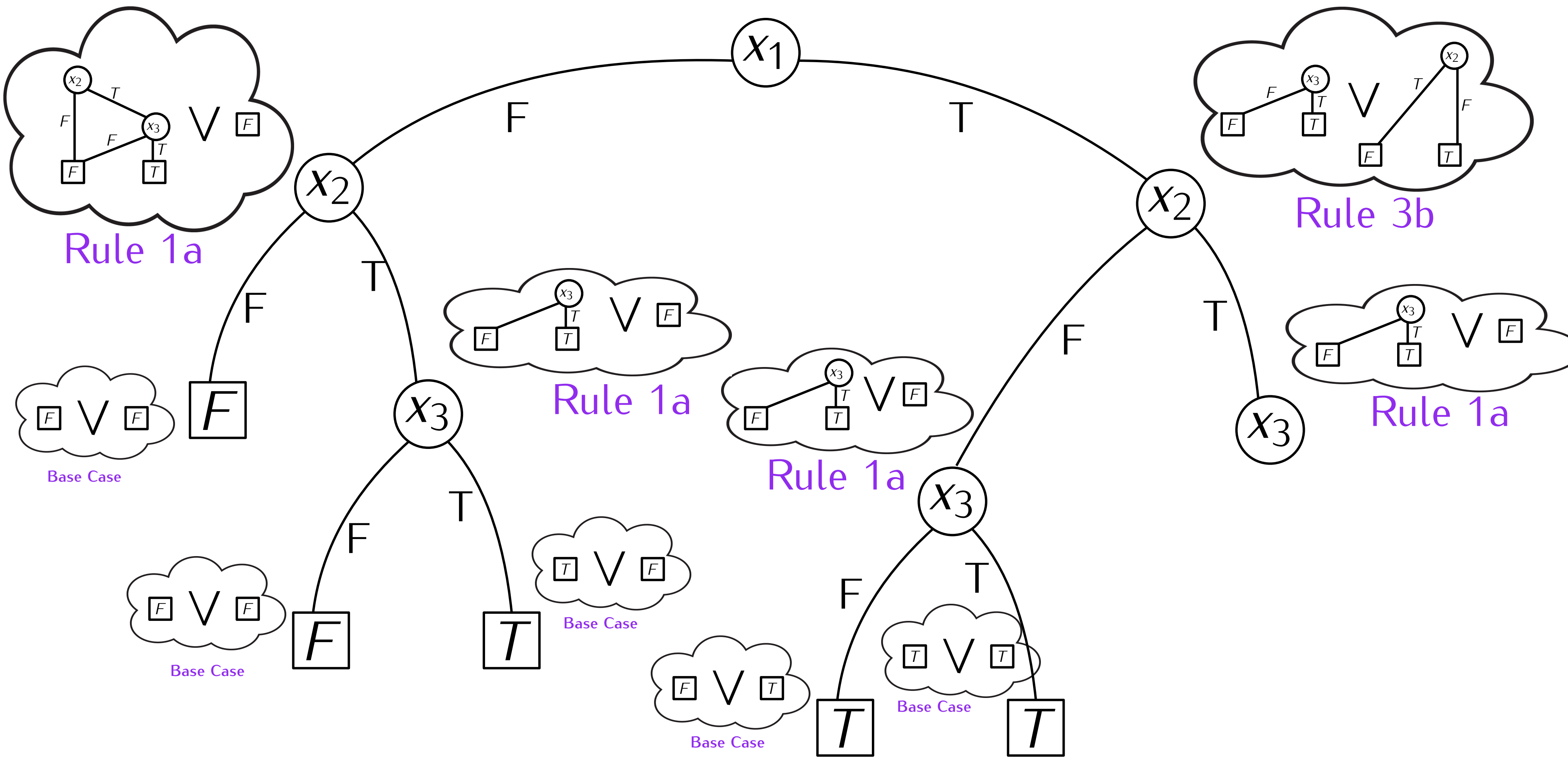
$f \equiv (x_1 \vee x_2) \wedge x_3$ $\qquad$ $g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

Compute $B_{f \vee g}$ using B_f and B_g

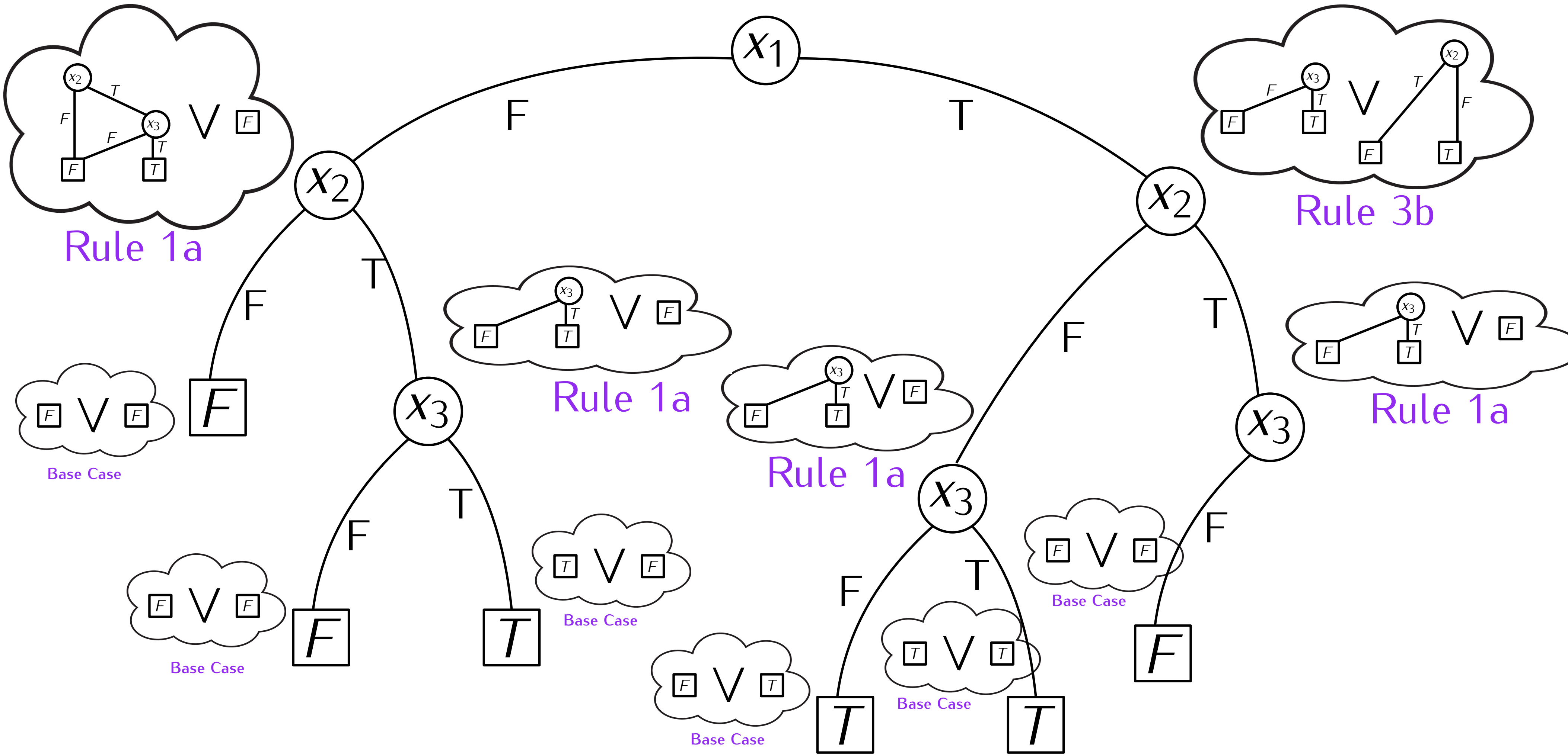
$f \equiv (x_1 \vee x_2) \wedge x_3$ $\qquad$ $g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

Compute $B_{f \vee g}$ using B_f and B_g

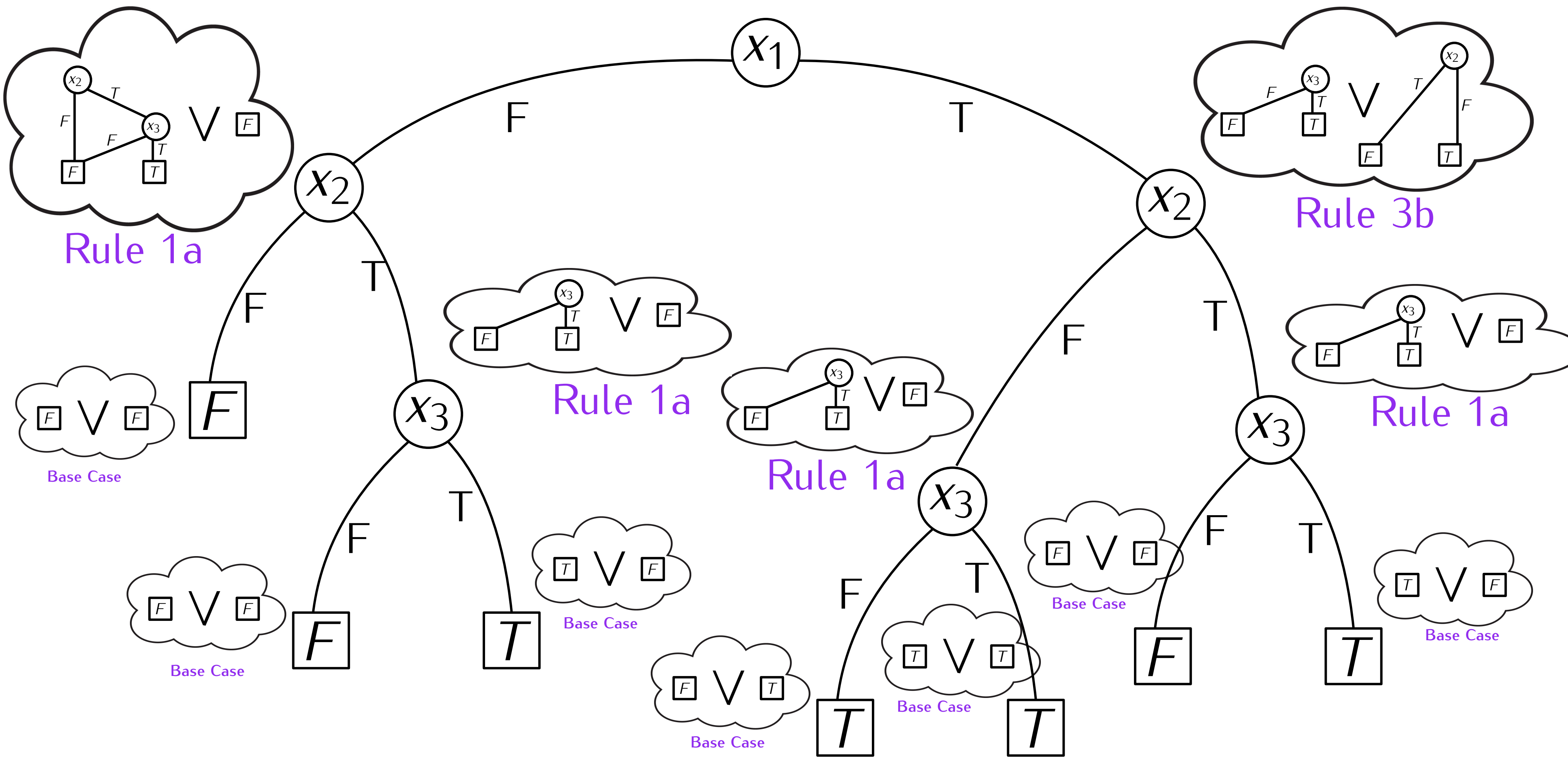
$f \equiv (x_1 \vee x_2) \wedge x_3$ $\qquad$ $g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

Compute $B_{f \vee g}$ using B_f and B_g

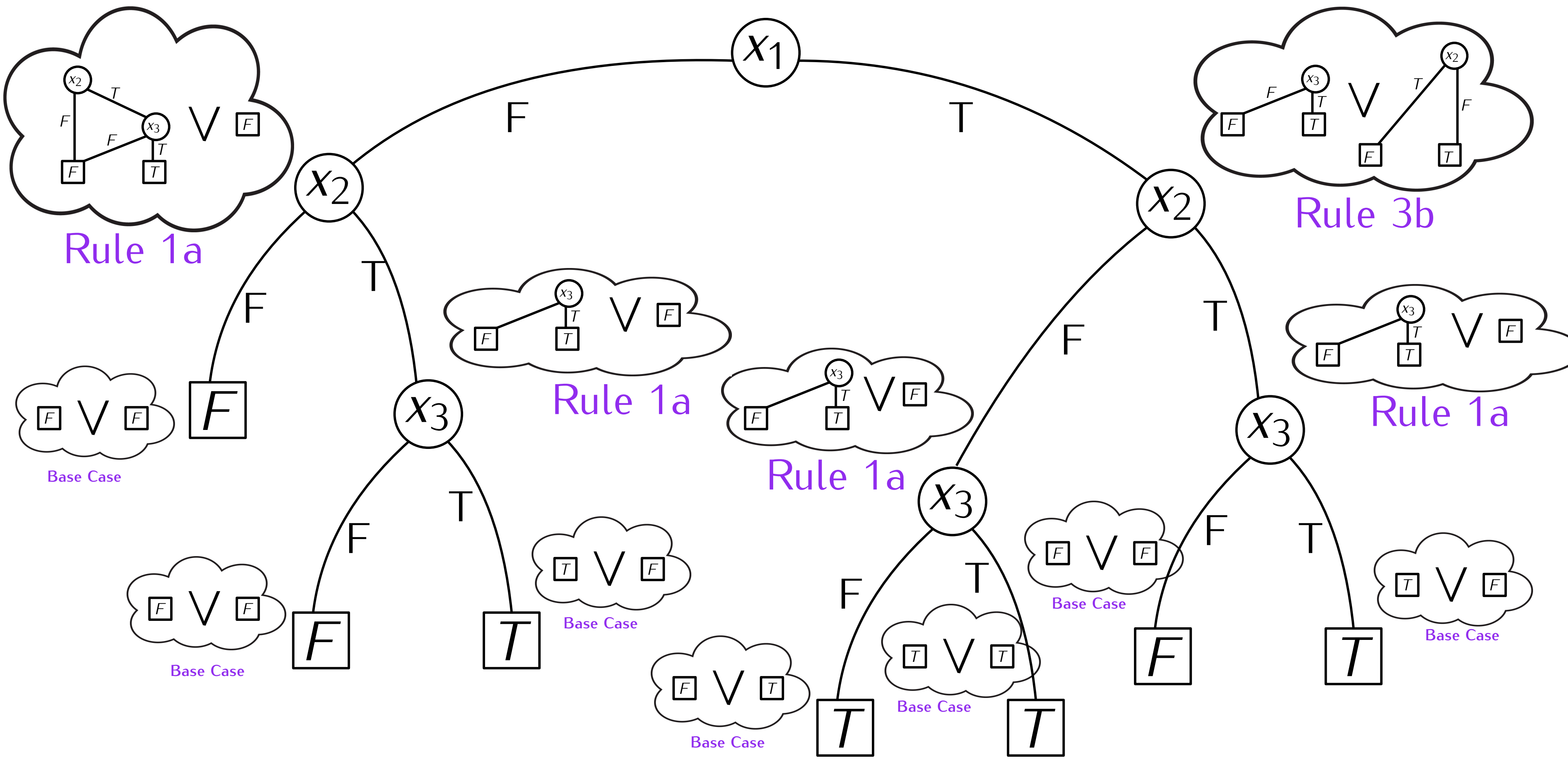
$f \equiv (x_1 \vee x_2) \wedge x_3$ $\quad$ $g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

Compute $B_{f \vee g}$ using B_f and B_g

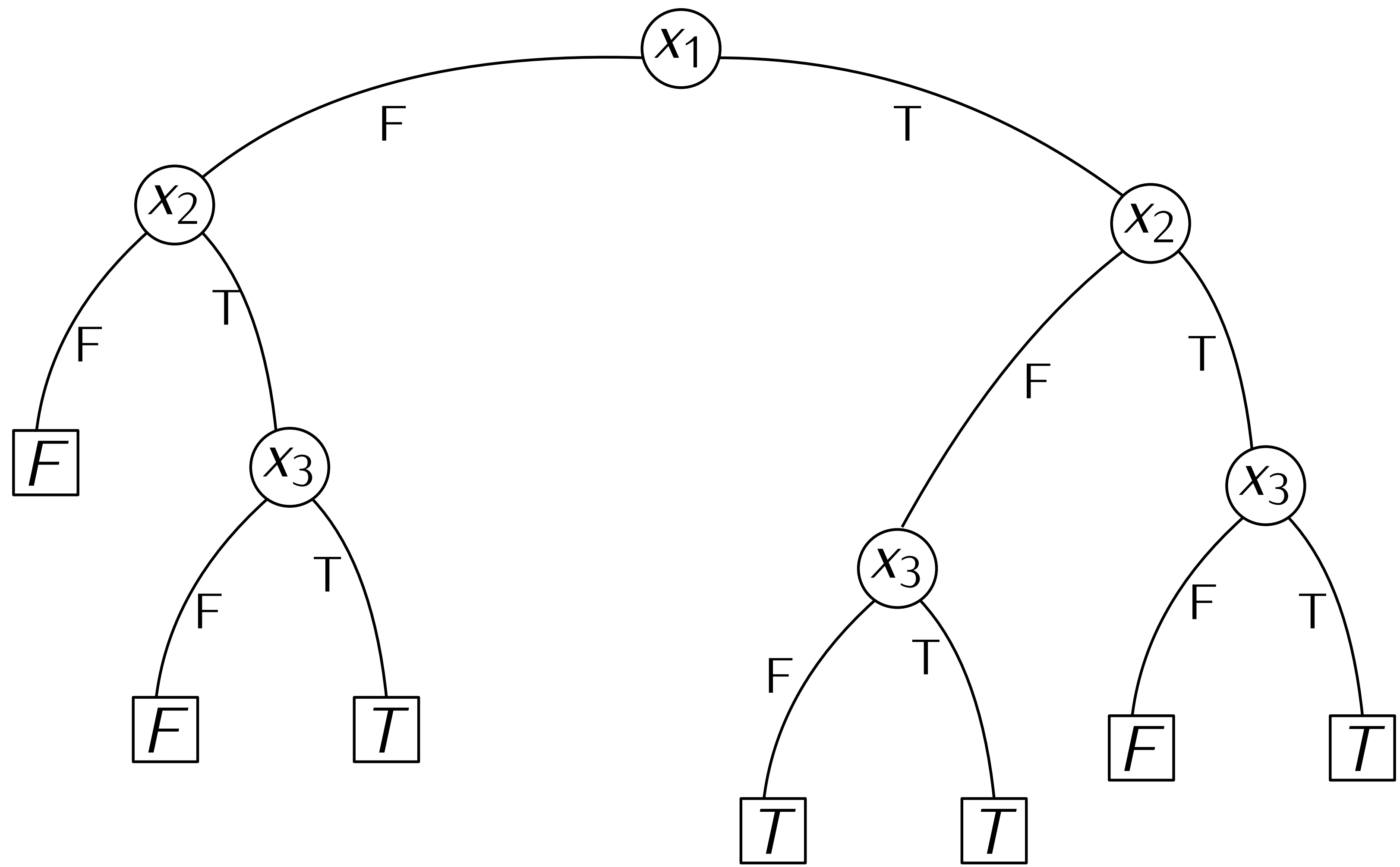
$f \equiv (x_1 \vee x_2) \wedge x_3$ $\qquad$ $g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

Compute $B_{f\vee g}$ using B_f and B_g

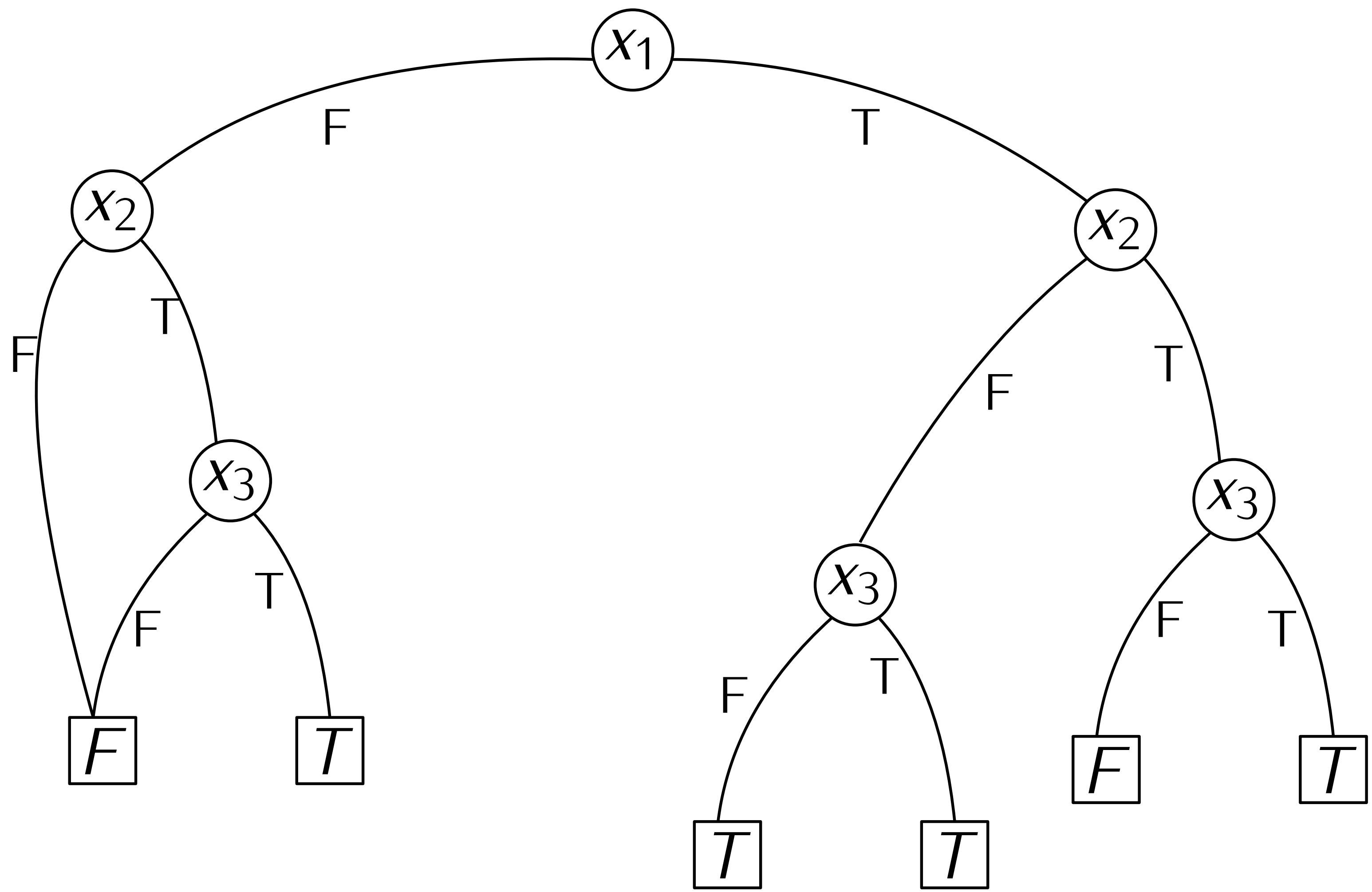
$f \equiv (x_1 \vee x_2) \wedge x_3$ $\qquad$ $g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

Compute $B_{f \vee g}$ using B_f and B_g

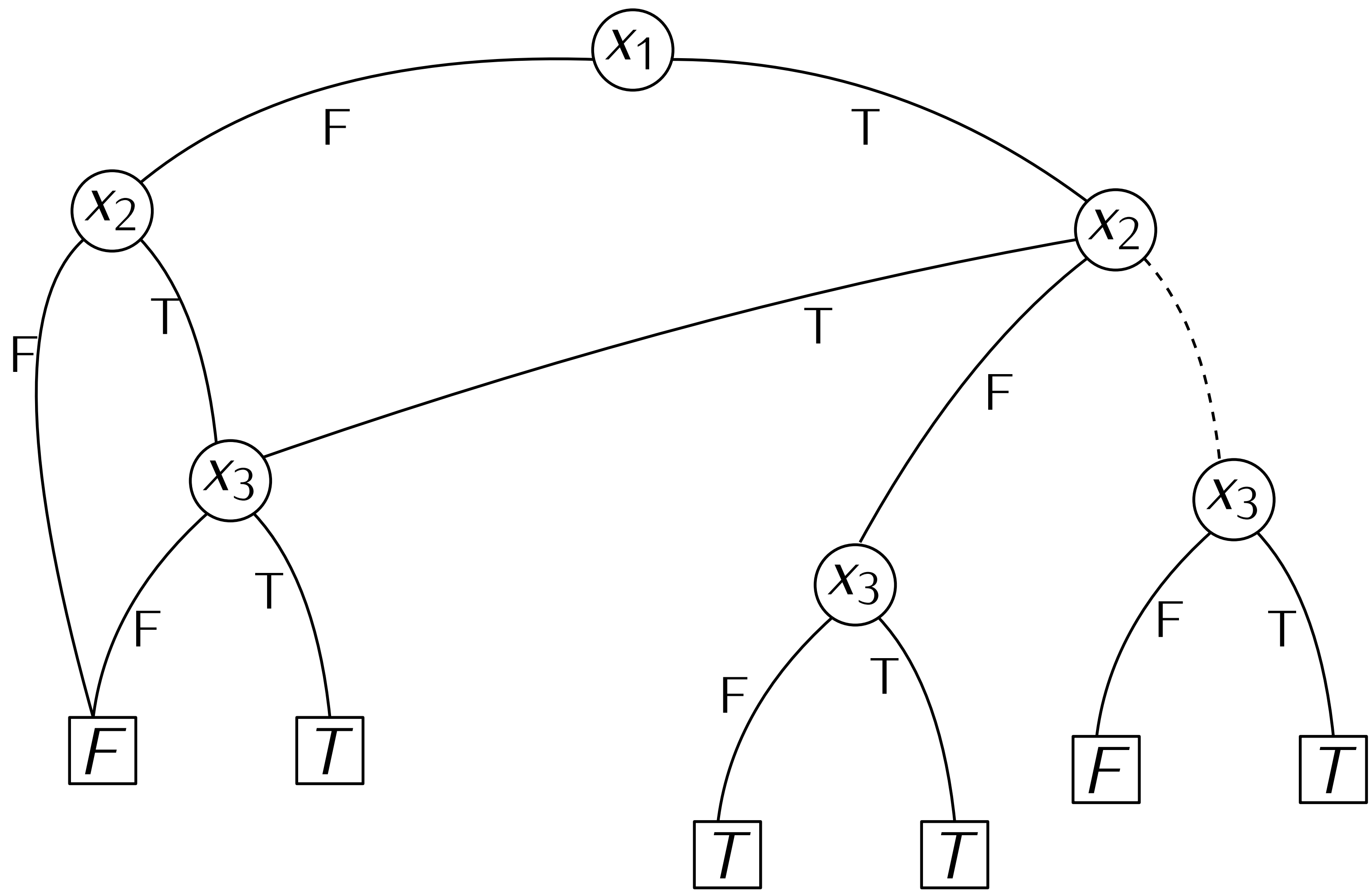
$f \equiv (x_1 \vee x_2) \wedge x_3$ $\qquad$ $g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

Compute $B_{f \vee g}$ using B_f and B_g

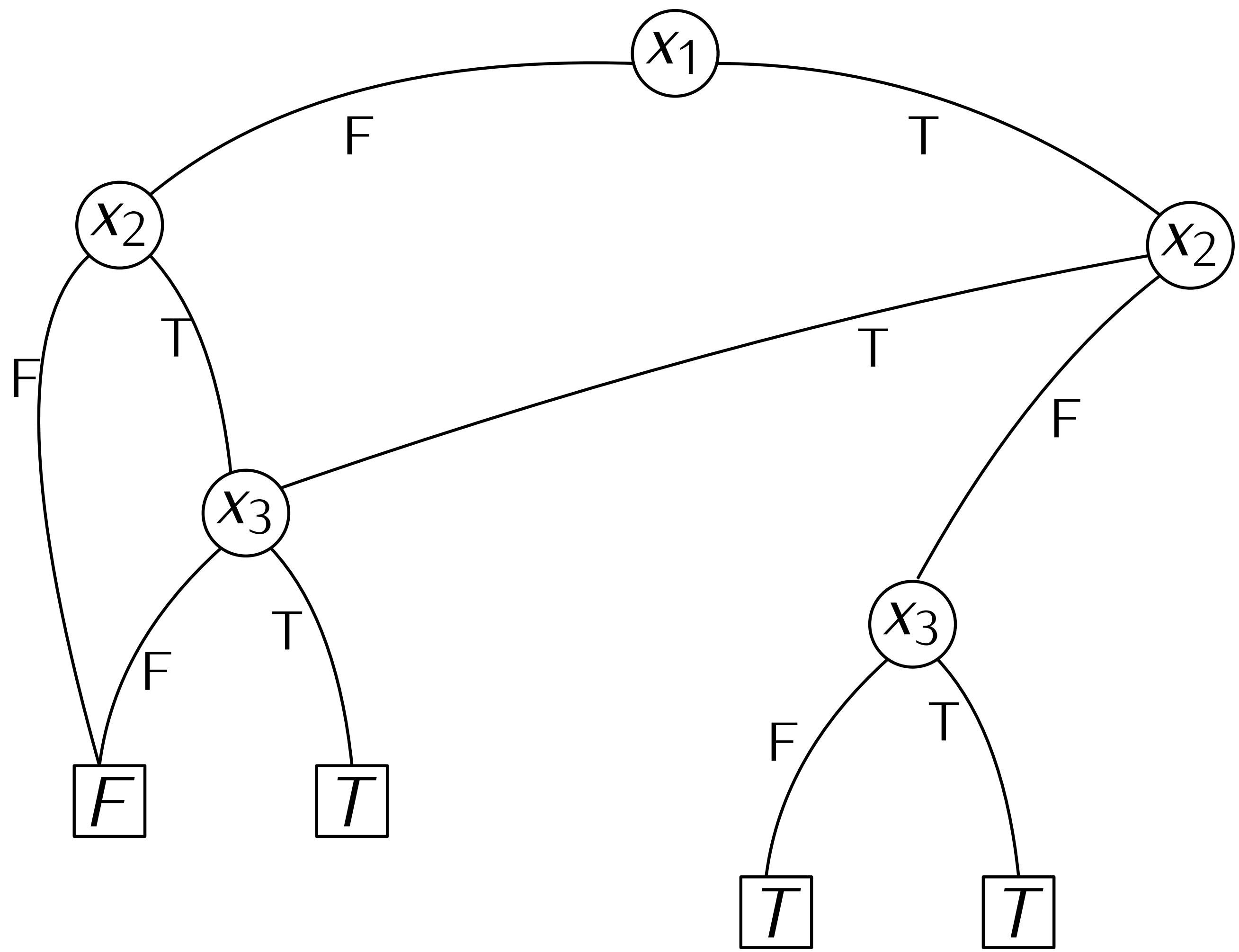
$f \equiv (x_1 \vee x_2) \wedge x_3$ $\quad$ $g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

Compute $B_{f \vee g}$ using B_f and B_g

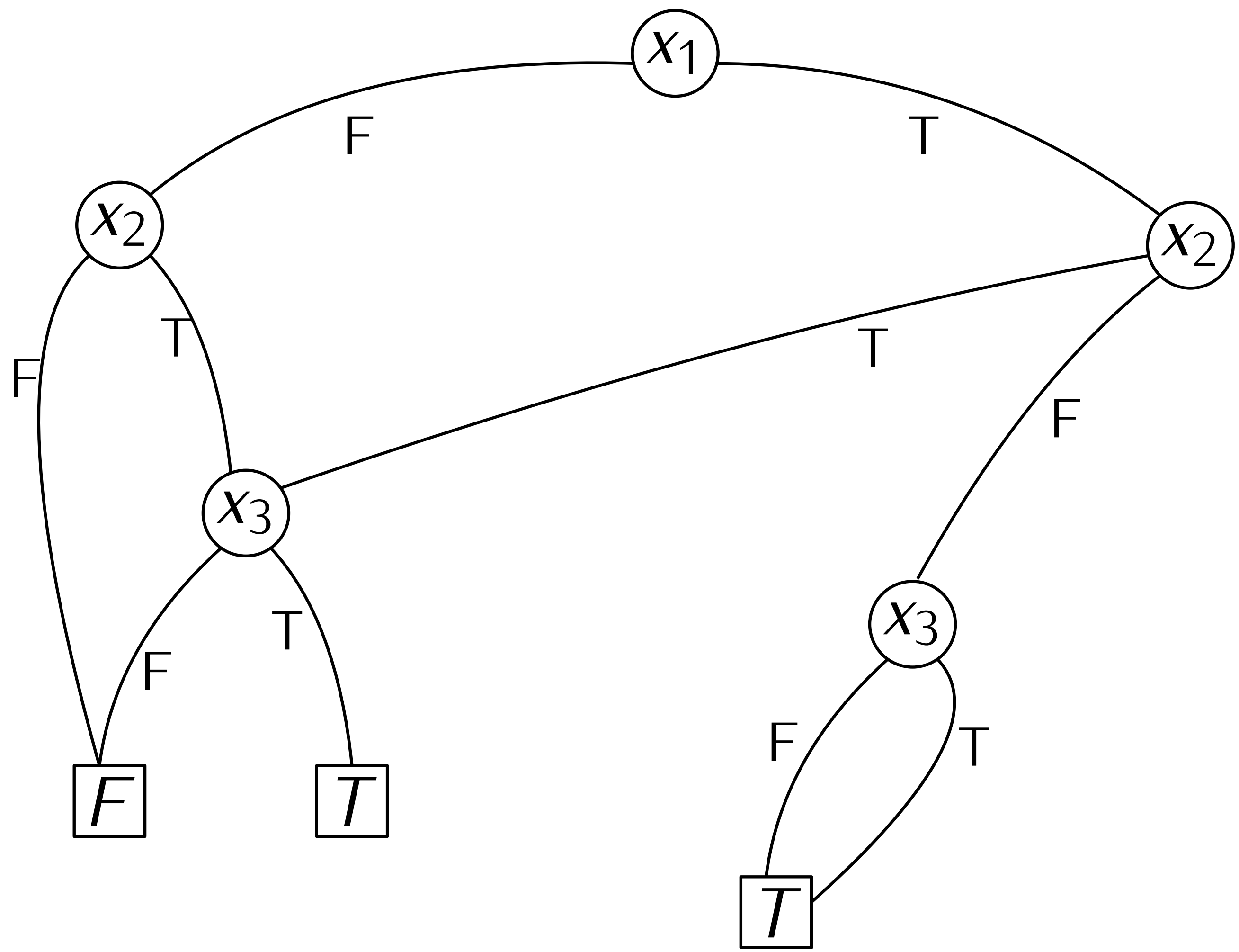
$f \equiv (x_1 \vee x_2) \wedge x_3$ $\qquad$ $g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

Compute $B_{f \vee g}$ using B_f and B_g

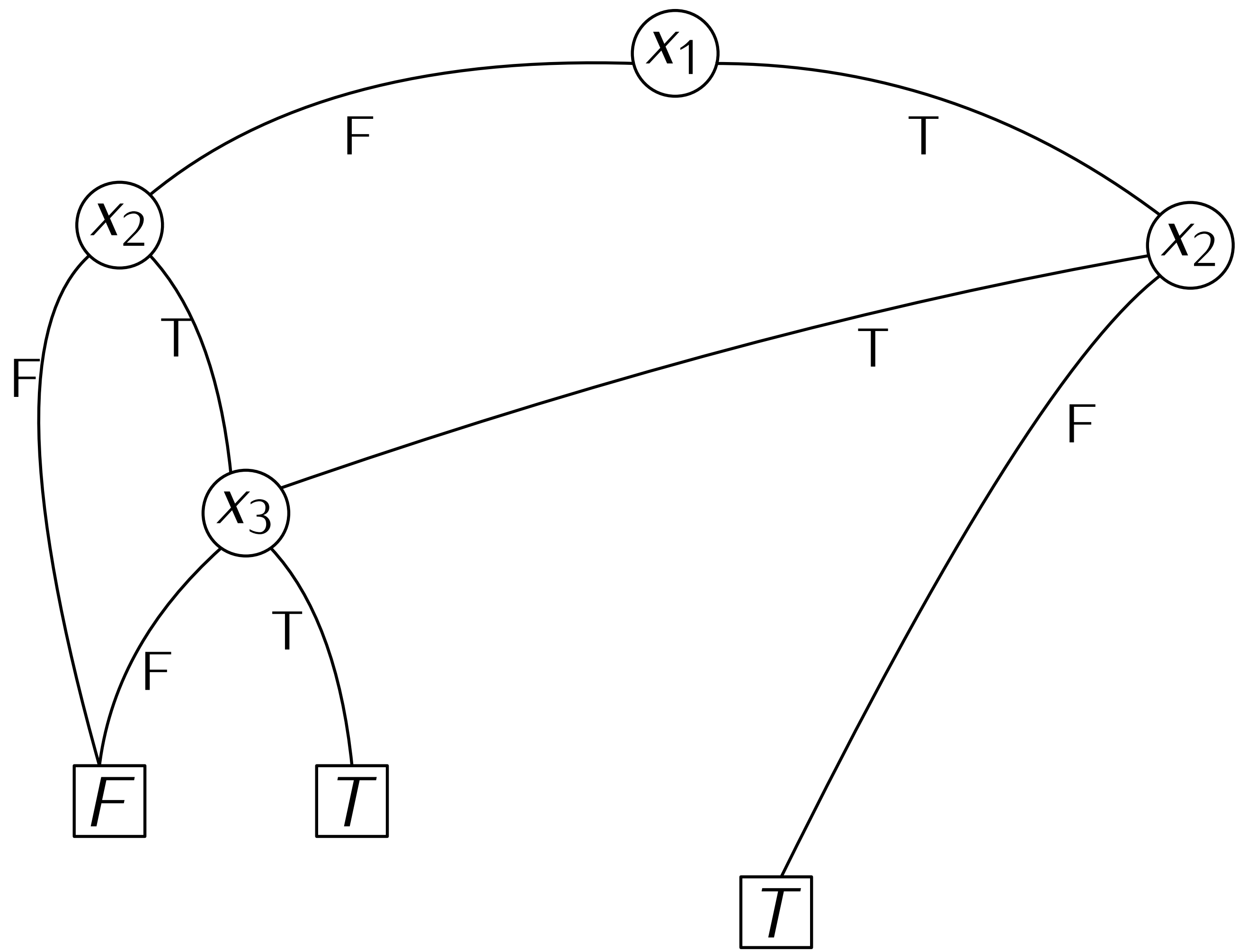
$f \equiv (x_1 \vee x_2) \wedge x_3$ $\qquad$ $g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

Compute $B_{f \vee g}$ using B_f and B_g

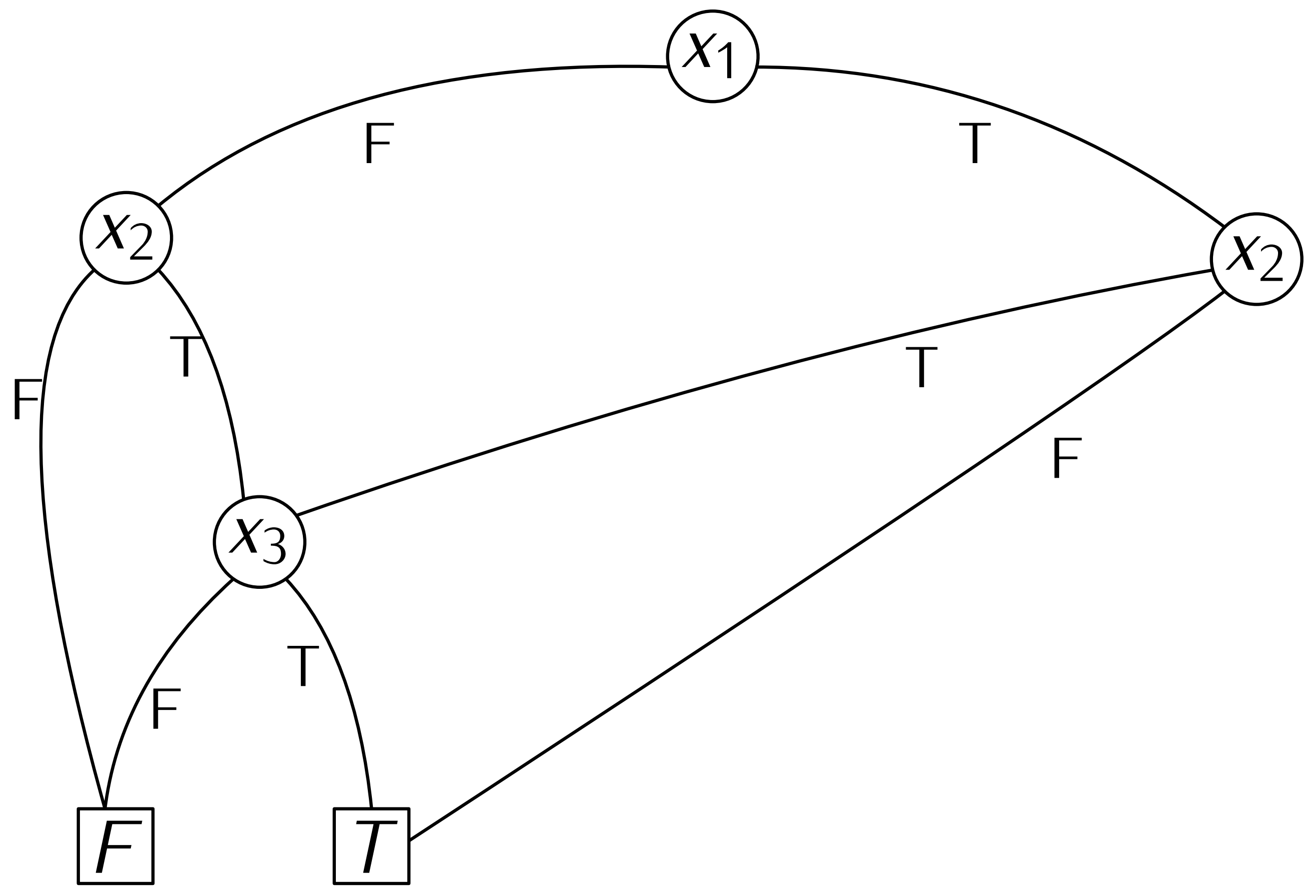
$f \equiv (x_1 \vee x_2) \wedge x_3$ $g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

Compute $B_{f \vee g}$ using B_f and B_g

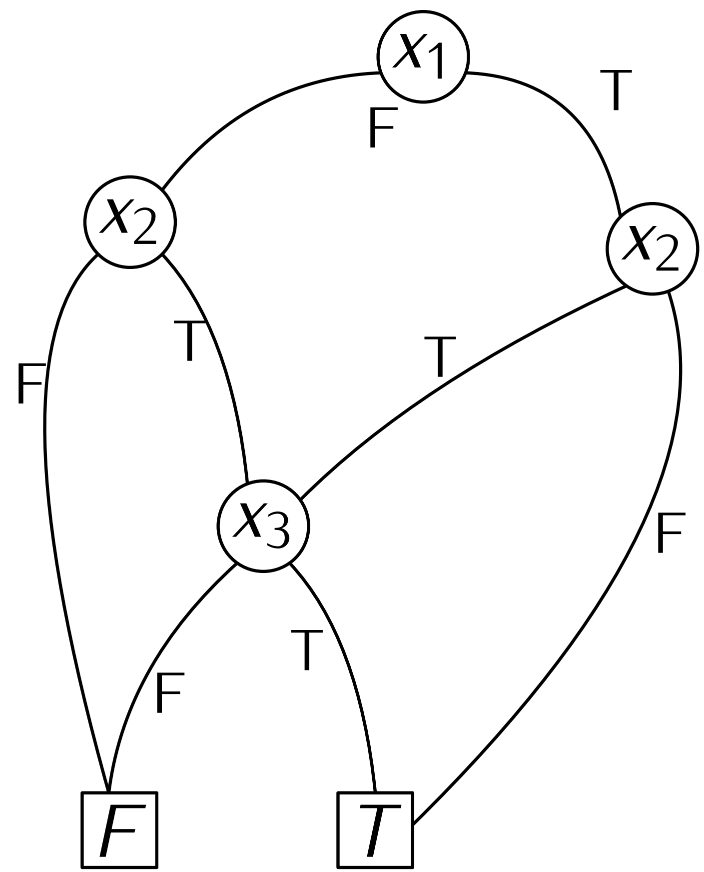
$f \equiv (x_1 \vee x_2) \wedge x_3$ $\qquad$ $g \equiv (x_1 \wedge \neg x_2)$



Binary ops on B_f and B_g:

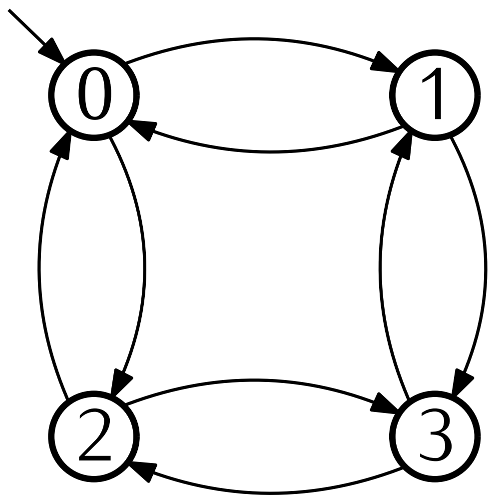
Compute $B_{f \vee g}$ using B_f and B_g

$f \equiv (x_1 \vee x_2) \wedge x_3$ $\qquad$ $g \equiv (x_1 \wedge \neg x_2)$



Transition System Representations

A transition system $\mathcal{M}$ can be specified by listing out all of the pieces.



States: $S = \{0, 1, 2, 3\}$

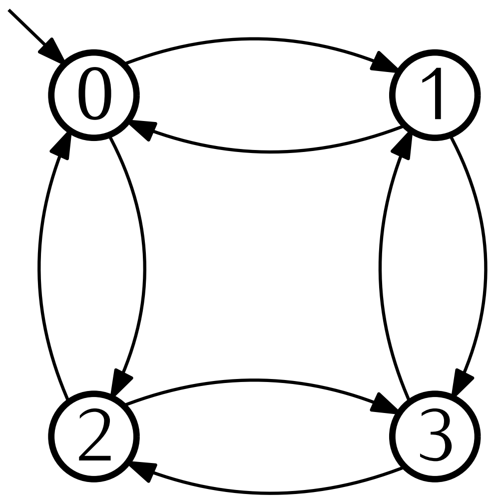
Initial States: $I = \{0\}$

Transitions:

$$R = \left\{ \begin{matrix} (0,1) & (0,2) & (1,3) & (2,3) \\ (1,0) & (2,0) & (3,1) & (3,2) \end{matrix} \right\}$$

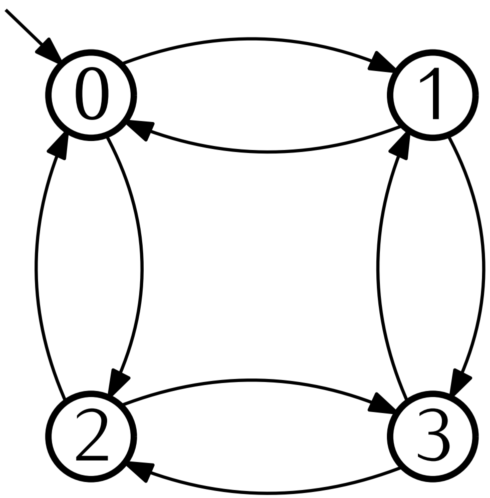
Symbolic Representation

Represent $\mathcal{M}$ using Boolean logic.



Symbolic Representation

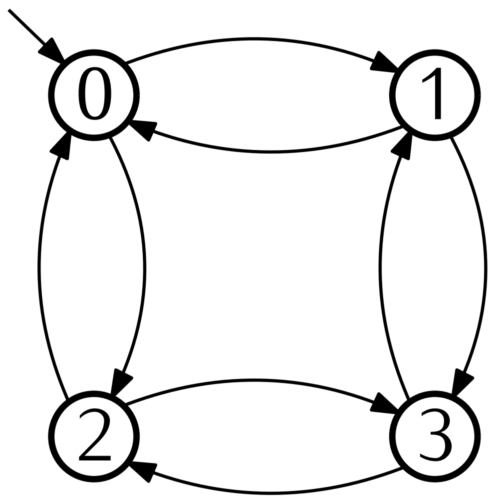
Represent $\mathcal{M}$ using Boolean logic.



States		
0		
1		
2		
3		

Symbolic Representation

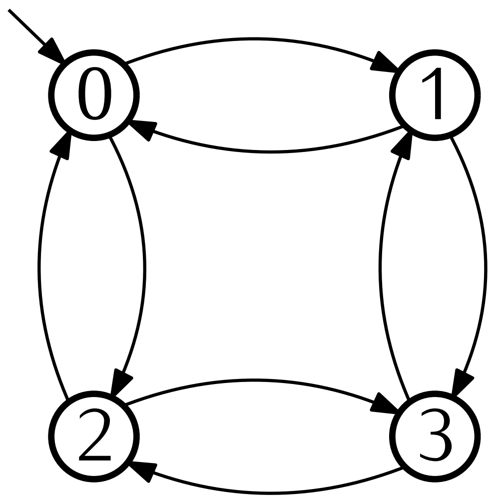
Represent $\mathcal{M}$ using Boolean logic.



States	binary	
	x	y
0	0	0
1	0	1
2	1	0
3	1	1

Symbolic Representation

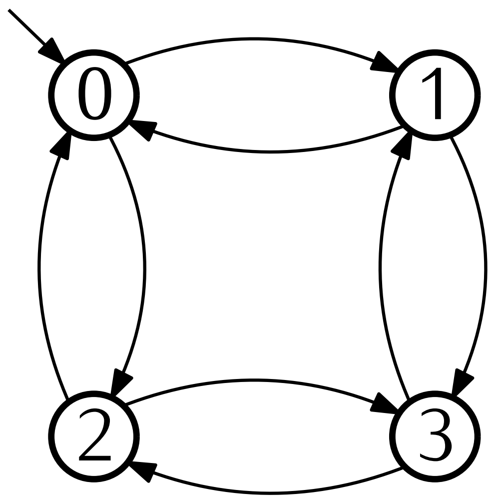
Represent $\mathcal{M}$ using Boolean logic.



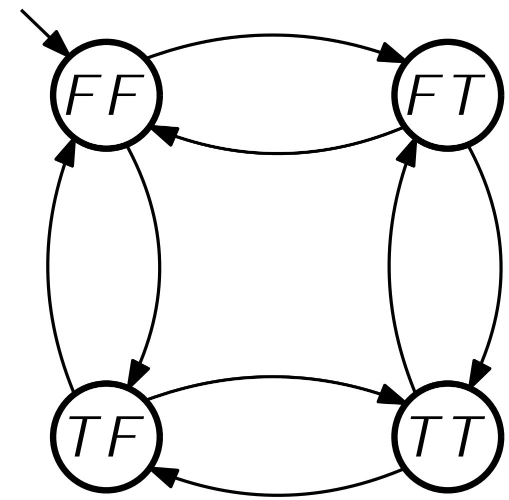
States	binary		truth values	
	x	y	x	y
0	0	0	F	F
1	0	1	F	T
2	1	0	T	F
3	1	1	T	T

Symbolic Representation

Represent $\mathcal{M}$ using Boolean logic.



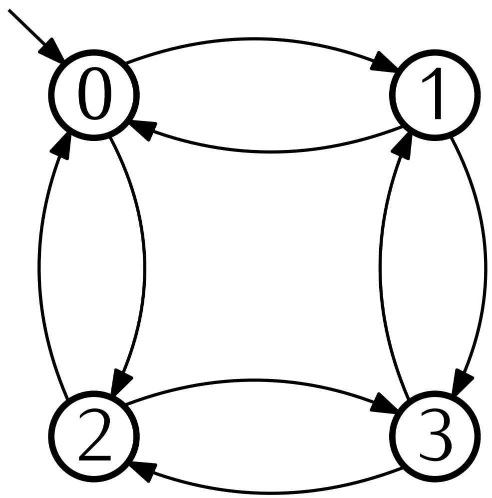
Boolean state variables
$V = \{x, y\}$



States	binary		truth values	
	x	y	x	y
0	0	0	F	F
1	0	1	F	T
2	1	0	T	F
3	1	1	T	T

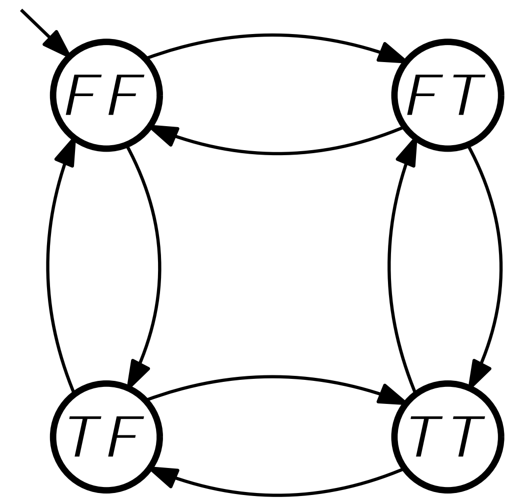
Symbolic Representation

Represent $\mathcal{M}$ using Boolean logic.



Boolean state variables

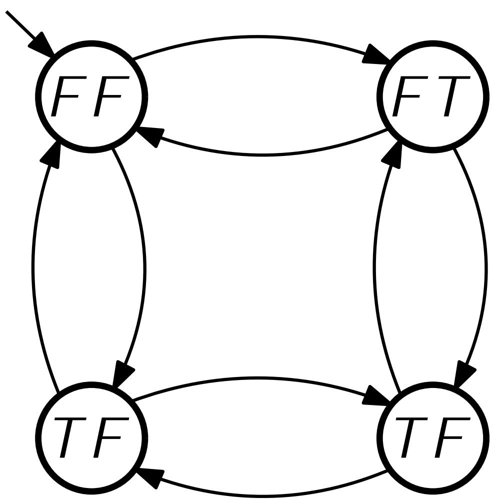
$V = \{x, y\}$



States	binary		truth values		Boolean formula
	x	y	x	y	
0	0	0	F	F	$\neg x \wedge \neg y$
1	0	1	F	T	$\neg x \wedge y$
2	1	0	T	F	$x \wedge \neg y$
3	1	1	T	T	$x \wedge y$

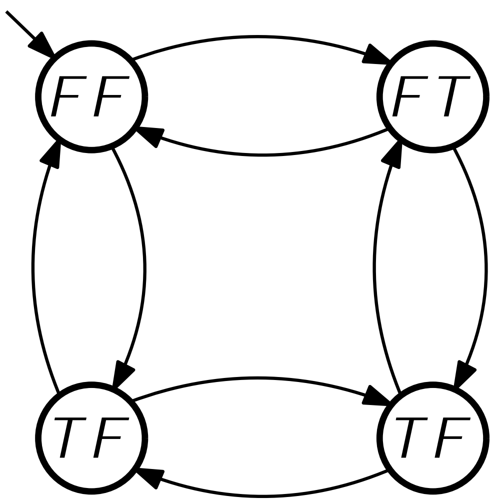
Symbolic Representation

Represent $\mathcal{M}$ using Boolean logic.



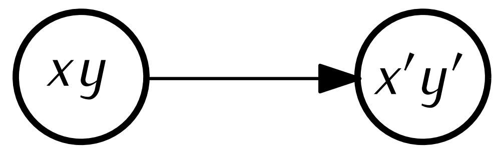
Symbolic Representation

Represent $\mathcal{M}$ using Boolean logic.



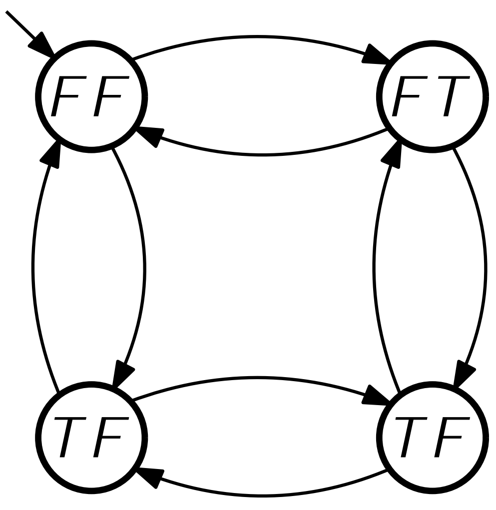
Transitions:

Let the "next" state variables be $V' = \{x', y'\}$



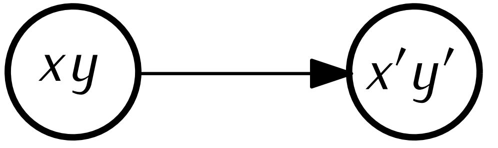
Symbolic Representation

Represent $\mathcal{M}$ using Boolean logic.



Transitions:

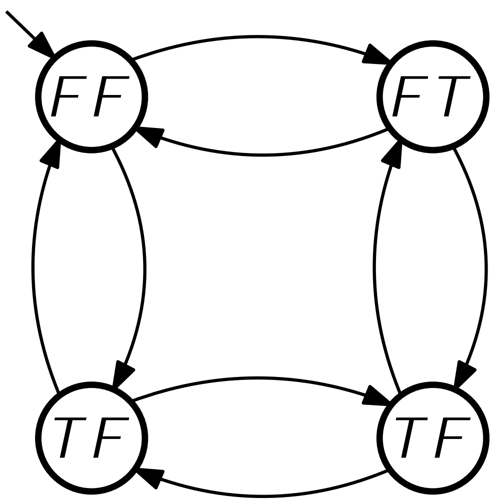
Let the "next" state variables be $V' = \{x', y'\}$



$$R \quad \equiv \quad (x' = x \wedge y' = \neg y) \vee (x' = \neg x \wedge y' = y)$$

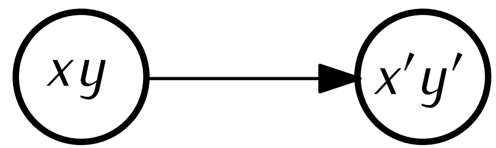
Symbolic Representation

Represent $\mathcal{M}$ using Boolean logic.



Transitions:

Let the "next" state variables be $V' = \{x', y'\}$

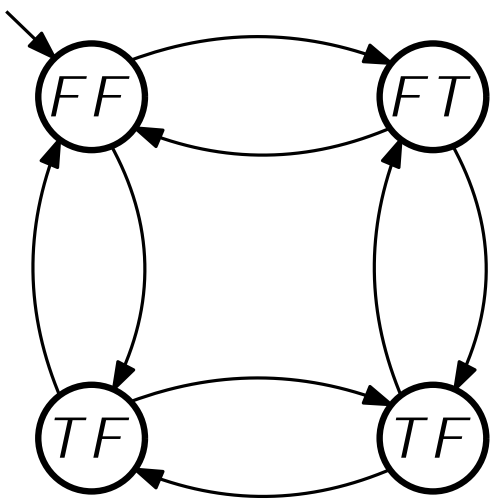


$$R \equiv (x' = x \wedge y' = \neg y) \vee (x' = \neg x \wedge y' = y)$$

"we can get from one state to the next by keeping one variable the same and negating the other"

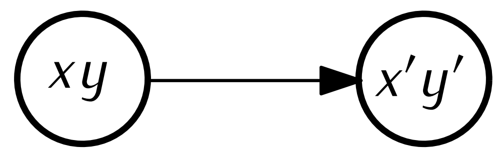
Symbolic Representation

Represent $\mathcal{M}$ using Boolean logic.



Transitions:

Let the "next" state variables be $V' = \{x', y'\}$



$$R \quad \equiv \quad (x' = x \wedge y' = \neg y) \vee (x' = \neg x \wedge y' = y)$$

<table>
<tr><td rowspan="2">Explicit transitions</td><td>$(0,1)$</td><td>$(2,3)$</td><td>$(1,3)$</td><td>$(0,2)$</td></tr>
<tr><td>$(1,0)$</td><td>$(3,2)$</td><td>$(3,1)$</td><td>$(2,0)$</td></tr>
</table>

"we can get from one state to the next by keeping one variable the same and negating the other"

Symbolic Representation

$$R \quad \equiv \quad (x' = x \wedge y' = \neg y) \vee (x' = \neg x \wedge y' = y)$$

BDD for R