# Lecture 4
# Module I: Model Checking
# Topic: CTL Symbolic Model Checking

J.Vain

03.03.2022

# Our Roadmap [based on McMillan et al. LICS 90]

- Recall that
  1. CTL temporal operators can be expressed using **base operators** $\mathrm{EX}$, $\mathrm{EG}$ and $\mathrm{EU}$;
  2. the base operators can be expressed as **fixpoints** and can be computed iteratively;

  3. explicit state notation can be transformed to symbolic notation by representing sets of states $S$ and the transition relation $R$ as **Boolean logic formulas**

- Then, fixpoint computation becomes formula manipulation, that includes:
  1. **pre-image** (EX) computation and **existentially bound variable elimination**;
  2. conjunction (intersection), disjunction (union), negation (set difference), and equivalence checks;
  3. Using **Binary Decision Diagrams** (BDDs) as efficient data structure for computing truth values of boolean logic formulas.

# Example: Mutual Exclusion Protocol (revisited)

Two concurrently executing processes are trying to enter their critical section without violating mutual exclusion condition

```
Process 1:
while (true) {
   out:  a := true; turn := true;
   wait: await (b = false or turn = false);
   cs:   a := false;
}
||
Process 2:
while (true) {
   out:  b := true; turn := false;
   wait: await (a = false or turn=true);
   cs:   b := false;
}
```

# Encoding State Space *S*

- Encode the state space using only boolean variables

- We have two variables for program counters: `pc1, pc2`
  with domains `{out, wait, cs}`
- We need two boolean variables per program counter to encode their 3 values:
  for `pc1`: $pc1_0$ and $pc1_1$ for `pc2`: $pc2_0$ and $pc2_1$
  - Encoding:

`pc1 = out` $\longrightarrow$ $\neg pc1_0 \land \neg pc1_1$

`pc1 = wait` $\longrightarrow$ $\neg pc1_0 \land pc1_1$

`pc1 = cs` $\longrightarrow$ $pc1_0 \land pc1_1$

- The other three variables `turn,a,b` are already booleans.

# Encoding State Space *S*

- Each state can be written as a tuple of boolean variables:

$$(pc1_0, pc1_1, pc2_0, pc2_1, turn, a, b)$$

  - So, after encoding:
    `(○,○,F,F,F)` becomes (F,F,F,F,F,F,F)
    `(○,c,F,T,F)` becomes (F,F,T,T,F,T,F)

- We map boolean state vector to logic formula on variables $pc1_0$, $pc1_1$, $pc2_0$, $pc2_1$, turn, a, b to represent state vector symbolically:

  $\{(F,F,F,F,F,F,F)\} \longmapsto \neg pc1_0 \land \neg pc1_1 \land \neg pc2_0 \land \neg pc2_1 \land \neg turn \land \neg a \land \neg b$

  $\{(F,F,T,T,F,F,T)\} \longmapsto \neg pc1_0 \land \neg pc1_1 \land pc2_0 \land pc2_1 \land \neg turn \land \neg a \land b$

  and represent the **set of states** by disjoining individual state formulas:

  $\{(F,F,F,F,F,F,F), (F,F,T,T,F,F,T)\} \longmapsto$

  $\neg pc1_0 \land \neg pc1_1 \land \neg pc2_0 \land \neg pc2_1 \land \neg turn \land \neg a \land \neg b$
  $\lor \neg pc1_0 \land \neg pc1_1 \land pc2_0 \land pc2_1 \land \neg turn \land \neg a \land b$
  $\equiv \neg pc1_0 \land \neg pc1_1 \land \neg turn \land \neg b \land (pc2_0 \land pc2_1 \leftrightarrow b)$

# Encoding Initial States

- We can also write the initial states as a boolean logic formuli
  - recall that, initially: `pc1=o and pc2=o`
  - but other variables may have any value in their domain

In set notation:

*I* ≡ { `(o,o,F,F,F), (o,o,F,F,T), (o,o,F,T,F),`
`(o,o,F,T,T), (o,o,T,F,F), (o,o,T,F,T),`
`(o,o,T,T,F), (o,o,T,T,T)` }

mapping it to logic notation:

$\longmapsto \quad \neg \mathrm{pc1}_0 \wedge \neg\, \mathrm{pc1}_1 \wedge \neg \mathrm{pc2}_0 \wedge \neg\, \mathrm{pc2}_1$

This logic formula tells that programm counters pc1 and pc2 are set to *false* and other variables may have arbitrary boolean values (they do not influence on the truth value of the formula)

# Encoding the Transition Relation

- We use boolean logic formulas and *primed variables* to encode the transition relation *R.*
- So we use two sets of variables:
  - Current state variables: $pc1_0$, $pc1_1$, $pc2_0$, $pc2_1$, turn, a, b
  - Next state variables: $pc1_0'$, $pc1_1'$, $pc2_0'$, $pc2_1'$, turn', a', b'
- For example, we can write a boolean logic formula for the command of process 1:

```
cs:   a := false;
```

Formula below describes the effect of executing command symbolically:

$$\overbrace{pc1_0 \wedge pc1_1 \wedge \neg pc1_0' \wedge \neg pc1_1'}^{\text{Pgm. counter variables that change}} \wedge \overbrace{\neg a'}^{\text{Data variable that changes}} \wedge$$

$$\underbrace{(pc2_0' \leftrightarrow pc2_0) \wedge (pc2_1' \leftrightarrow pc2_1) \wedge (turn' \leftrightarrow turn) \wedge (b' \leftrightarrow b)}_{\text{Other data variables that do not change}}$$

Let's denote this formula with symbol $R_{1c}$

# Encoding the Transition Relation

- Similarly we can write a formula $R_{ij}$ for each command in the program

- Then the overall transition relation is is disjunction

  $R \equiv R_{1\mathrm{o}} \vee R_{1\mathrm{w}} \vee R_{1\mathrm{c}} \vee R_{2\mathrm{o}} \vee R_{2\mathrm{w}} \vee R_{2\mathrm{c}}$

- Having the model *M* in symbolic form, we also need to know for symbolic model checking of CTL formula $\varphi$ how to interprete the temporal operators of $\varphi$ on this symbolic representation of *M.*

# Symbolic Pre-Image Computation

IMPORTANT

- Recall the pre-image is a functional

  $\mathrm{EX} : 2^S \to 2^S$

  which is defined (in set notation) as:

  $\mathrm{EX}(\varphi) = \{\, s \mid (s, s') \in [\![ R ]\!] \text{ and } s' \in [\![ \varphi ]\!] \,\}$

- We can represent *pre-image* symbolically as usual 1st order logic formula

$$\boxed{\mathrm{EX}(\varphi) \equiv \exists V' \,(R \wedge \varphi\,[V' / V])}$$

  where

  - $V$ : values of Boolean state variables in the current-state
  - $V'$ : values of Boolean state variables in the next-state
  - $\varphi\,[V' / V]$ : renaming variables in $\varphi$ by replacing current-state variables with the corresponding next-state variables
  - $\exists V' f$: means existentially quantifying variables $V'$ in $f$
  - $R$ denotes the symbolic formula of transition relation

# Renaming (or substitution)

Example:

- Assume that we have two variables $x, y$
- and sets $V = \{x, y\}$ and $V' = \{x', y'\}$

- Renaming example:

  Given formula $\varphi \equiv x \wedge y$ ,

  we apply variable substitution $[V' / V]$ to variables in formula $\varphi$ :

  $\varphi[V' / V] \equiv (x \wedge y)\,[V' / V] \equiv x' \wedge y'$

Note: for correct substitution the order of variables must be fixed in $V'$ and $V$

# Existential Quantifier Elimination

IMPORTANT

- Given a boolean formula $f$ and variable $v$ we can rewrite quantified formula as

$$\exists v\, f \equiv f\,[true/v] \vee f\,[false/v] \qquad (*)$$

Here, we eliminate the existential quantifier by doing following:
- first, substitute the existentially bound variable $v$ with *true* in the formula $f$
- then substitute $v$ with *false* in $f$ and
- then take the disjunction of two results.

- Example: Let the transition relation conjoined with $\varphi$ be $f \equiv \neg x \wedge y \wedge x' \wedge y'$

The pre-image of $f$ according to (*) is

$\exists V' f \;\equiv \exists x' (\, \exists y' (\neg x \wedge y \wedge x' \wedge y')\,)$ % after applying (*) to $\exists y'$ we get

$\equiv \exists x' ((\neg x \wedge y \wedge x' \wedge y' )[true/y'] \vee (\neg x \wedge y \wedge x' \wedge y' )[false/y'])$

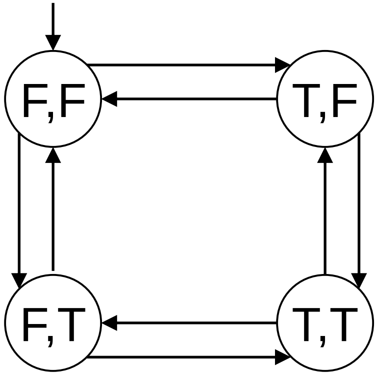
$\equiv \exists x' (\neg x \wedge y \wedge x' \cancel{\wedge true} \vee \cancel{\neg x \wedge y \wedge x' \wedge false} ) \equiv \exists x'( \neg x \wedge y \wedge x' )$

$\equiv (\neg x \wedge y \wedge x')[true/x'] \vee (\neg x \wedge y \wedge x')[false/x'])$

$\equiv \neg x \wedge y \cancel{\wedge true} \vee \cancel{\neg x \wedge y \wedge false}$

$\equiv \neg x \wedge y$

# An Extremely Simple Example



Variables: $x$, $y$: boolean

Set of explicit states:
$S$ = {(F,F), (F,T), (T,F), (T,T)}

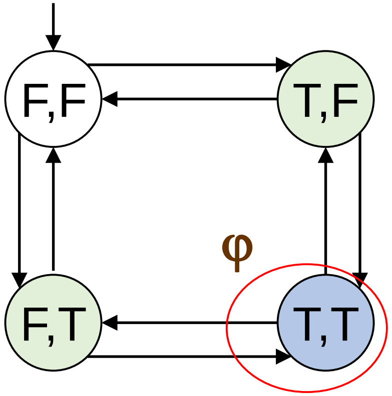
Set of states symbolically:
$S \equiv true$

Initial state condition:
$I \equiv \neg x \wedge \neg y$

Transition relation (after simplification):
$R \equiv x' = \neg x \wedge y' = y \vee x' = x \wedge y' = \neg y$

("≡" means "by definition")

# An Extremely Simple Example – EX $\boldsymbol{\varphi}$



- Given $\varphi \equiv x \wedge y$ and $R \equiv x' = \neg x \wedge y' = y \vee x' = x \wedge y' = \neg y$
- Compute $\mathrm{EX}(\varphi)$

$\mathrm{EX}(\varphi) \equiv \exists V' R \wedge \varphi[V' / V]$

$\equiv \exists V' R \wedge x' \wedge y'$ | by substit

$\equiv \exists V' (x' = \neg x \wedge y' = y \vee x' = x \wedge y' = \neg y) \wedge x' \wedge y'$ | by substit

$\equiv \exists V' (x' = \neg x \wedge y' = y) \wedge x' \wedge y' \vee (x' = x \wedge y' = \neg y) \wedge x' \wedge y'$ | by distrib. law

$\equiv \exists V' \neg x \wedge y \wedge x' \wedge y' \vee x \wedge \neg y \wedge x' \wedge y'$ | by $\leftrightarrow$

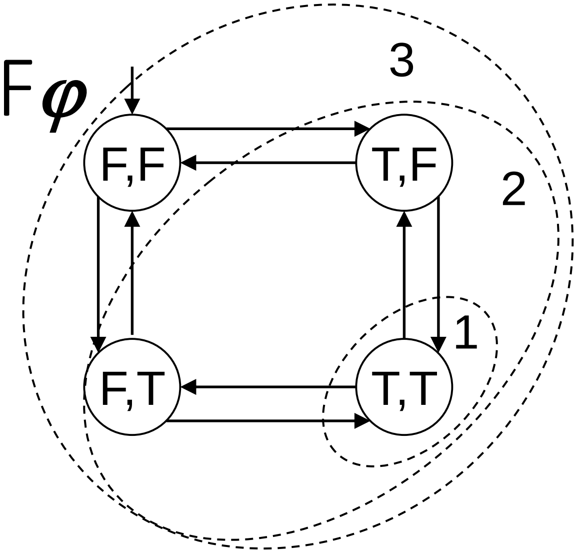
$\equiv \neg x \wedge y \vee x \wedge \neg y$ | by $\exists$ -elimination

The states in pre-image

$\mathrm{EX}(x \wedge y) \equiv \neg x \wedge y \vee x \wedge \neg y$, are denoted with purple in KS diagram.

In terms of explicit states $\mathrm{EX}(\{(T,T)\}) \equiv \{(F,T), (T,F)\}$

# An Extremely Simple Example -EF$\boldsymbol{\varphi}$



Let's compute $\mathrm{EF}(x \wedge y)$ on model $M$ by applying fixpoint algorithm (see Lecture 4).

The fixpoint computation sequence provides symbolic values:
$\mathit{false},\quad x{\wedge}y\ ,\quad x{\wedge}y \vee \mathrm{EX}(x{\wedge}y)\ ,\quad x{\wedge}y \vee \mathrm{EX}\ (x{\wedge}y \vee \mathrm{EX}(x{\wedge}y)\ )\ , \ldots$

If we do the $\mathrm{EX}$ computation iteratively, we get a sequence of symbolic states:
Result: $\underbrace{\mathit{false},}_{0}\quad \underbrace{x \wedge y}_{1},\quad \underbrace{x \wedge y \vee \neg x \wedge y \vee x \wedge \neg y,}_{2}\quad \underbrace{\mathit{true}}_{3}$
Step no:

$\mathrm{EF}(x \wedge y) \equiv \mathit{true}$ (means full state space)
In terms of explicit states $\mathrm{EF}(\{(\mathrm{T,T})\}) \equiv \{(\mathrm{F,F}),(\mathrm{F,T}), (\mathrm{T,F}),(\mathrm{T,T})\}$

# An Extremely Simple Example

- Based on our results, shown on example transition system $T = (S, I, R)$ we saw that
  - If inital states $I$ satisfy $\mathrm{EF}(x \wedge \mathrm{y})$ , i.e.

    $$I \subseteq \mathrm{EF}(x \wedge y) \quad (\subseteq \text{ corresponds to implication})$$

    then:

    $$T \vDash \mathrm{EF}(x \wedge y)$$

  i.e., there exists a path from the initial state s.t. eventually $x$ and $y$ become true in the same state
- In the first example, since

  $$I \nsubseteq \mathrm{EX}(x \wedge y)$$

  then:

  $$T \nvDash \mathrm{EX}(x \wedge \mathrm{y})$$

  *Property is not satisfied in* $T$

  i.e., there is not a path from the initial state such that in the next state of path both $x$ and $y$ become true.

# An Extremely Simple Example – AF$\boldsymbol{\varphi}$

- Let’s try one more property AF$(x \wedge y)$

- To check this property we first convert it to a formula which uses only temporal operators in our basis:

  $$\mathrm{AF}(x \wedge y) \equiv \neg\ \mathrm{EG}(\neg(x \wedge y))$$

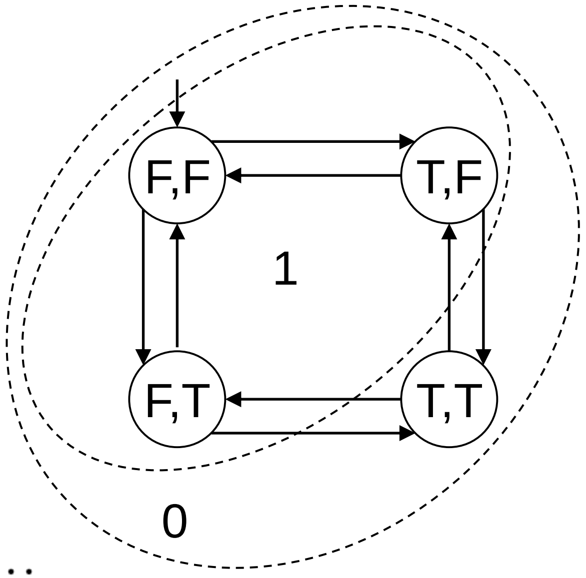
  i.e.,

  if we can find such a initial state which satisfies $\mathrm{EG}(\neg(x \wedge y))$,

  then we know that the transition system $T$ does not satisfy property

  $$\mathrm{AF}(x \wedge y)$$

# An Extremely Simple Example



Let’s compute $\mathrm{EG}(\neg(x \wedge y))$

The fixpoint computation sequence is:

$$\mathit{true}, \quad \neg x \vee \neg y, \quad (\neg x \vee \neg y) \wedge \mathrm{EX}(\neg x \vee \neg y)\ , \ldots$$

If we do the $\mathrm{EX}$ computations, we get:

$$\underbrace{\mathit{true},}_{0.} \quad \underbrace{\neg x \vee \neg y,}_{1.} \quad \boxed{\underbrace{\neg x \vee \neg y,}_{2.}}$$

This is fixpoint

i.e.

$$\mathrm{EG}(\neg(x \wedge y)) \equiv \neg x \vee \neg y$$

Since $\quad I \cap \mathrm{EG}(\neg(x \wedge y)) \neq \varnothing$

we conclude that $\quad T \not\models \ \mathrm{AF}(x \wedge y)$

# Symbolic CTL Model Checking Algorithm (in general)

- Translate the formula to a formula which uses the CTL basis
  - $\mathrm{EX}\varphi$, $\mathrm{EG}\varphi$, $\varphi\,\mathrm{EU}\,\psi$

- Atomic propositions can be interpreted in states by inspecting whether the formula is in the set AP of given state labels.

- For $\mathrm{EX}\varphi$ compute the pre-image using existential variable elimination

- For $\mathrm{EG}\varphi$ and $\mathrm{EU}\varphi$ compute the fixpoints iteratively

# Symbolic Model Checking Algorithm (1)

Check (*f* : CTL formula) :

(here we use logic encoding of sets of states)

```
case: $f \in AP$                 return $f$;
case: $f \equiv \neg \varphi$              return $\neg Check(\varphi)$;
case: $f \equiv \varphi \wedge \psi$            return $Check(\varphi) \wedge Check(\psi)$;
case: $f \equiv \varphi \vee \psi$            return $Check(\varphi) \vee Check(\psi)$;
case: $f \equiv EX\ \varphi$  return $\boxed{\exists V'.\ R \wedge Check(\varphi)[V'/V]}$;
```

# Symbolic Model Checking Algorithm (2)

Check(*f*)

```
…
case: f ≡ EG $\varphi$
    Y  := true;                  // initializing Y  (includes all states)
    P  := Check($\varphi$);             // P – set of states where $\varphi$ is true
    Y' := P ∧ Check(EX(Y));
    while (Y ≠ Y')               // fixpoint condition
    {
        Y   := Y';               // save previous step result
        Y'  := P ∧ Check(EX(Y)); // find pre-image
    }
    return Y;                    //Y – set of states where EG $\varphi$ is true
```

# Symbolic Model Checking Algorithm (3)

Check(*f*)

```
…
case: f ≡ $\varphi$ EU $\psi$
    Y := false;                    // (empty set)
    P := Check($\varphi$);               // P–set of states where $\varphi$ is true
    Q := Check($\psi$);               // Q–set of states where $\psi$ is true
    Y' := Q ∨ [P ∧ Check(EX(Y))];      // here Y' = Q
    while (Y ≠ Y')
    {
        Y := Y';
        Y' := Q ∨ [$\overbrace{\text{P} \wedge \text{Check(EX(Y))}}^{\text{P–states from which states of Y are 1 step reachable}}$];
    }
    return Y;
```

# Binary Decision Diagrams (BDDs)

- Binary Decision Diagrams (BDDs)
  - An efficient data structure for boolean formula manipulation.
  - There are BDD packages available, e.g. https://github.com/johnyf/tool_lists/blob/master/bdd.md

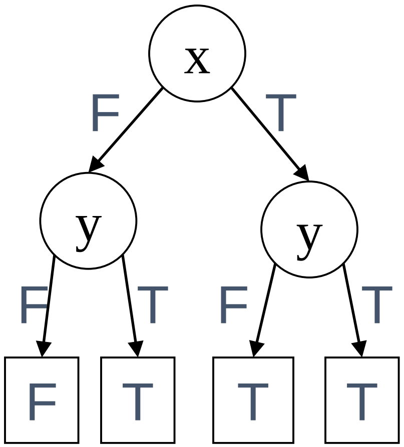
- BDD data structure can be used to implement symbolic model checking algorithms discussed above because predicate transformers include boolean connectives.

- BDDs are *canonical representation* for boolean logic formulas, i.e.
  - given formulas $F$ and $G$, they are $F \Leftrightarrow G$ if their BDD representations are identical.

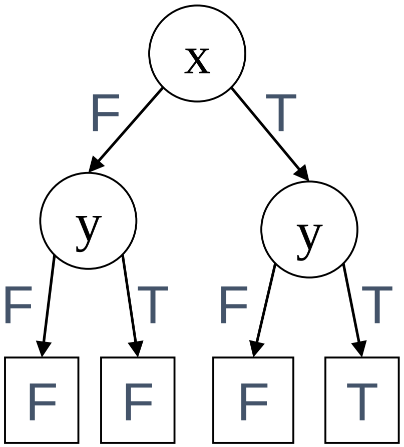
# Binary Decision Trees (BDT)

- Fix the order of variables in the boolean formula,
- Build a tree where in each branch of the same level the node is labeled with same variable and
- Outgoing edges from node are labeled with possible values of this variable
- Examples of BDT-s for boolean formulas of two variables:

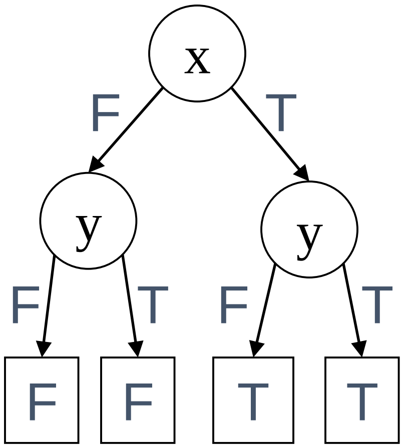
  Variable order: $x$, $y$



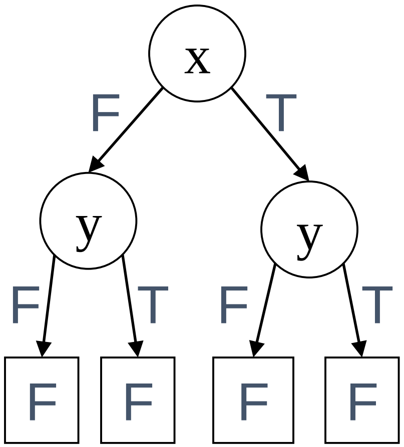
$x \vee y$



$x \wedge y$



$x$

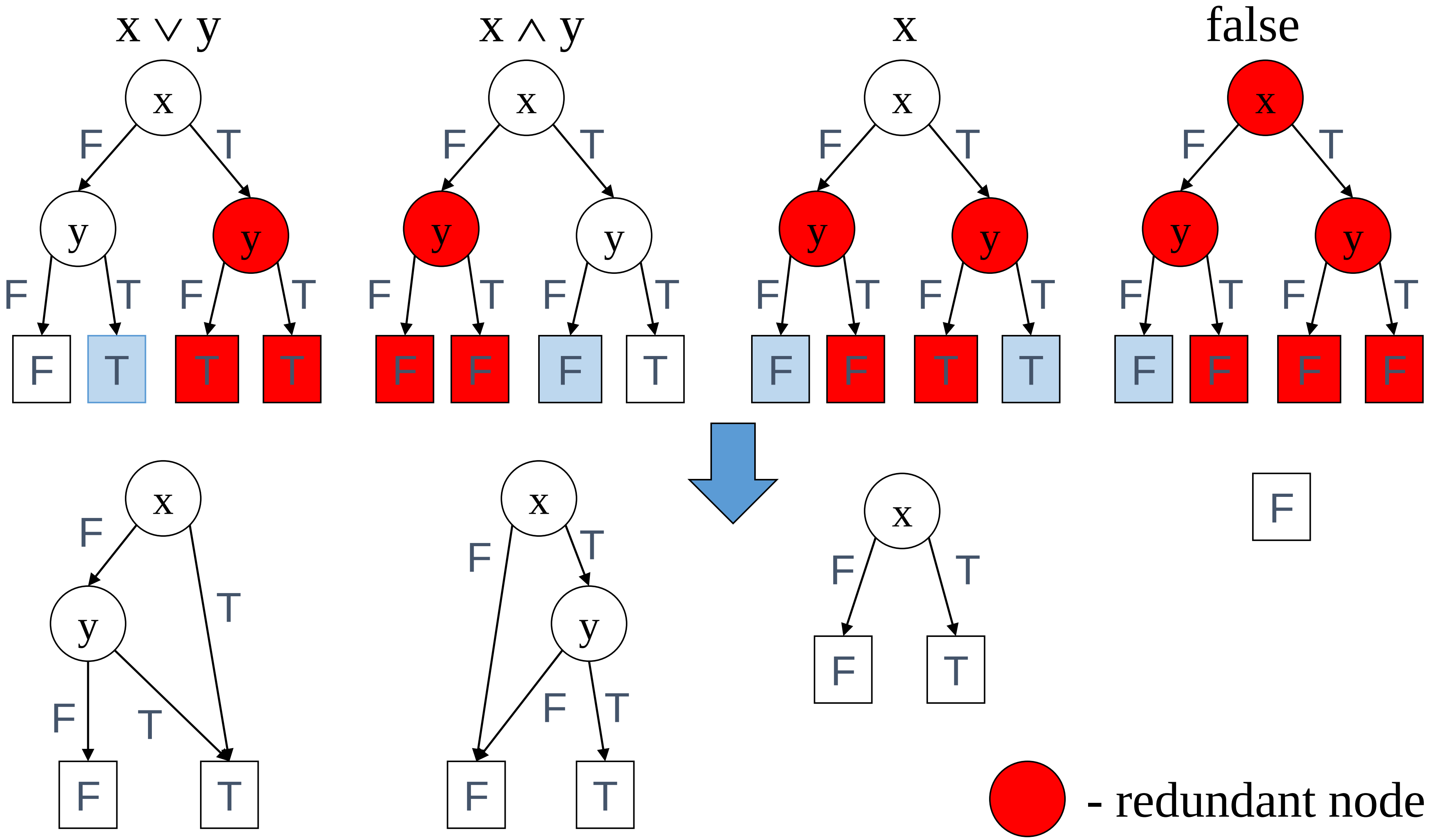


false

# Transforming BDT to BDD

- BDT has a lot of overhead and can be optimized to more compact form of **directed acyclic graph** – binary decision diagram (BDD).
- Method:
  - Repeatedly apply the following transformations to a BDT:
    - Remove duplicate terminals
      - redraw connections to remaining terminal nodes that have same label as deleted ones
    - Remove duplicate non-terminals
      - redraw connections to remaining non-terminal nodes that have same label as deleted ones
    - Remove redundant tests

# Mapping Binary Decision Trees to BDDs



# Good News About BDDs

- Given BDDs for two boolean logic formulas $\varphi$ and $\psi$,
  - the BDDs for $\varphi \wedge \psi$ and $\varphi \vee \psi$ are of size $|\varphi| \times |\psi|$ (and can be computed in that time)
  - the BDD for $\neg\ \varphi$ is of size $|\varphi|$ (and can be computed in that time)
  - Equivalence $\varphi \overset{?}{\Leftrightarrow} \psi$ can be checked in constant time
  - Satisfiability of $\varphi$ can be checked in constant time

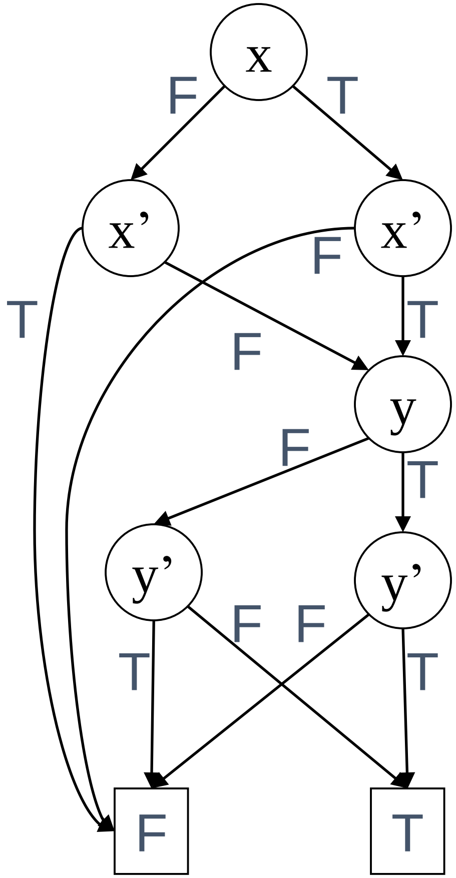
# Bad News About BDDs

- The size of a BDD can be exponential in the number of boolean variables
- The sizes of the BDDs are very sensitive to the ordering of variables. Bad variable ordering can cause exponential increase in the size of the BDD
- There are functions which have BDDs that are exponential for any variable ordering (for example binary multiplication)
- Pre-image computation requires existential variable elimination
  - Existential variable elimination can cause an exponential blow-up in the size of the BDD

# BDDs are Sensitive to Variables Order

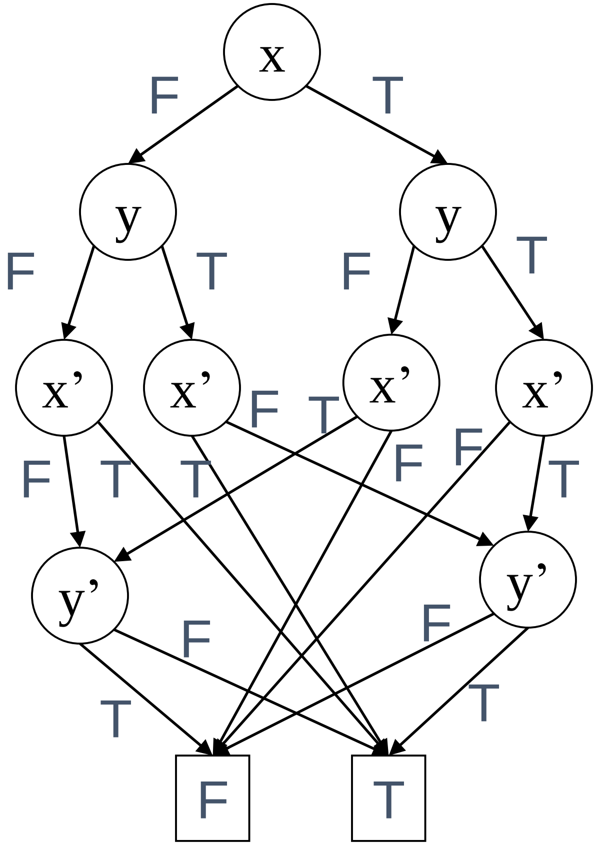
Identity relation for two variables: $(x' \leftrightarrow x) \wedge (y' \leftrightarrow y)$

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



For n variables we have $3n+2$ nodes

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



For n variables we have $3\times 2^n - 1$ nodes

# LTL and CTL* Model Checking complexity?

- The complexity of the model checking problem for LTL and CTL* is:
  - $(|S|+|R|) \times 2^{O(|f|)}$

  where $| f |$ is the number of logic connectives in $f$.

- Typically the size of the formula is much smaller than the size of the transition system
- So the exponential complexity in the size of the formula is not very critical in practice, the property specifications typically involve few variables and logic operators.