

# Outline

## Binary Decision Diagrams

Binary Decision Trees

If-then-else Normal Form

# Data Structures for Large Propositional Formulas

Suppose that **large propositional formulas** are reused over and over **again**. For example, we may

- ▶ Build a **conjunction** of several formulas;
- ▶ **Negate** a formula;
- ▶ Check if two formulas are **equivalent** . . .

One needs **data structures** which

- ▶ give **compact representation** of formulas, or the boolean functions represented by the formulas;
- ▶ facilitate **boolean operations** on this formulas, for example, taking conjunction of several formulas;
- ▶ facilitate **checking properties of formulas**, such as satisfiability or equivalence checking.

# Data Structures for Large Propositional Formulas

Suppose that **large propositional formulas** are reused over and over **again**. For example, we may

- ▶ Build a **conjunction** of several formulas;
- ▶ **Negate** a formula;
- ▶ Check if two formulas are **equivalent** . . .

One needs **data structures** which

- ▶ give **compact representation** of formulas, or the boolean functions represented by the formulas;
- ▶ facilitate **boolean operations** on this formulas, for example, taking conjunction of several formulas;
- ▶ facilitate **checking properties of formulas**, such as satisfiability or equivalence checking.

# Data Structures for Large Propositional Formulas

Suppose that **large propositional formulas** are reused over and over **again**. For example, we may

- ▶ Build a **conjunction** of several formulas;
- ▶ **Negate** a formula;
- ▶ Check if two formulas are **equivalent** . . .

One needs **data structures** which

- ▶ give **compact representation** of formulas, or the boolean functions represented by the formulas;
- ▶ facilitate **boolean operations** on this formulas, for example, taking conjunction of several formulas;
- ▶ facilitate **checking properties of formulas**, such as satisfiability or equivalence checking.

# Data Structures for Large Propositional Formulas

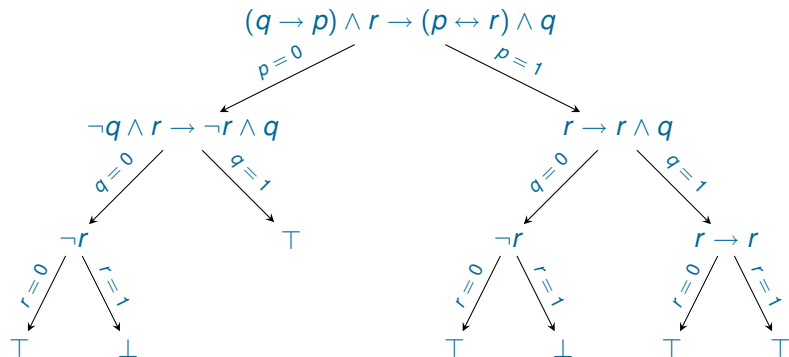
Suppose that **large propositional formulas are reused over and over again**. For example, we may

- ▶ Build a **conjunction** of several formulas;
- ▶ **Negate** a formula;
- ▶ Check if two formulas are **equivalent** . . .

One needs **data structures** which

- ▶ give **compact representation** of formulas, or the boolean functions represented by the formulas;
- ▶ facilitate **boolean operations** on this formulas, for example, taking conjunction of several formulas;
- ▶ facilitate **checking properties of formulas**, such as satisfiability or equivalence checking.

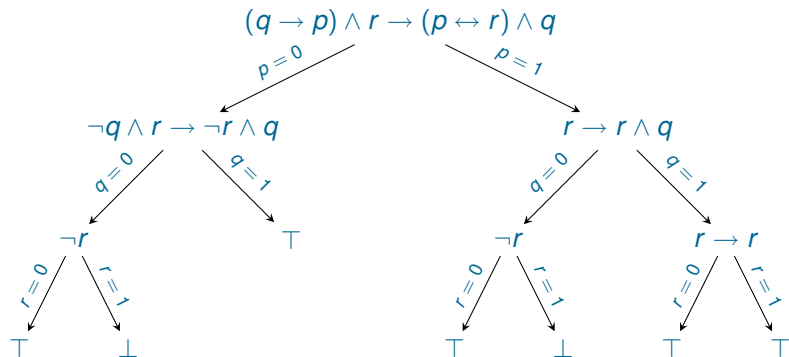
# Splitting Tree



Let us “forget” about formulas in the tree

We do not see the **syntax** of the formula but the **semantics is preserved**: the tree encodes all models of the formula. Any formula having the same tree with “forgotten” formulas has the same models.

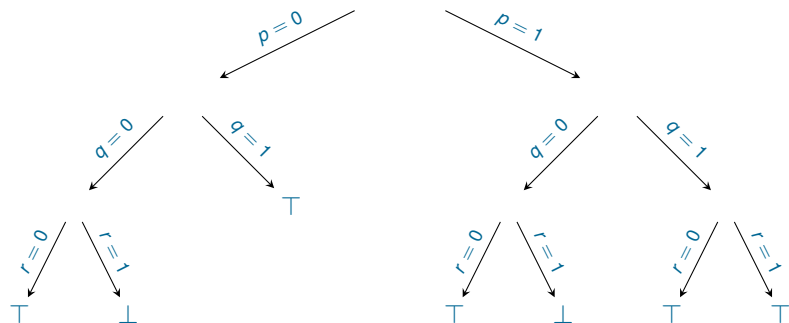
# Splitting Tree



Let us “forget” about formulas in the tree

We do not see the **syntax** of the formula but the **semantics is preserved**: the tree encodes all models of the formula. Any formula having the same tree with “forgotten” formulas has the same models.

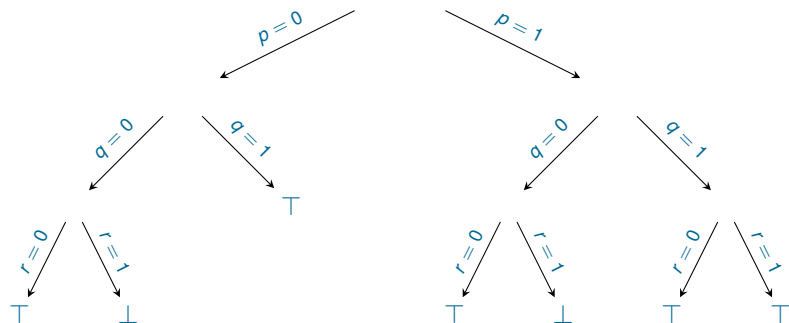
# Splitting Tree



Let us “forget” about formulas in the tree

We do not see the **syntax** of the formula but the **semantics is preserved**: the tree encodes all models of the formula. Any formula having the same tree with “forgotten” formulas has the same models.

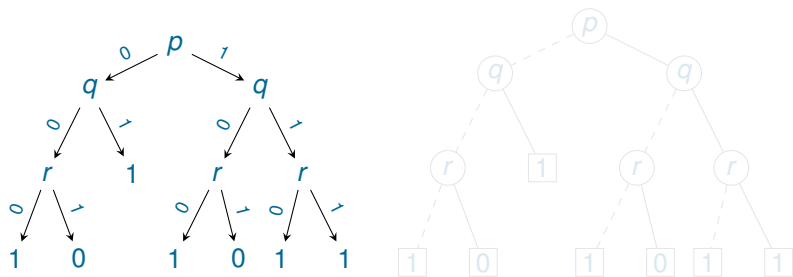
# Splitting Tree



Let us “forget” about formulas in the tree

We do not see the **syntax** of the formula but the **semantics is preserved**: the tree encodes all models of the formula. Any formula having the same tree with “forgotten” formulas has the same models.

# Binary Decision Tree

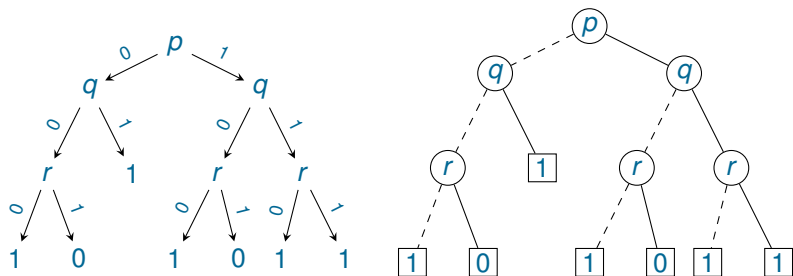


A circled node, such as  $\textcircled{p}$ , denote the choices on the variable of this node.

Terminal nodes are squared, for example  $\boxed{1}$ .

Solid lines correspond to choices when the variable is true, dashed lines to the choices when the variable is false.

# Binary Decision Tree

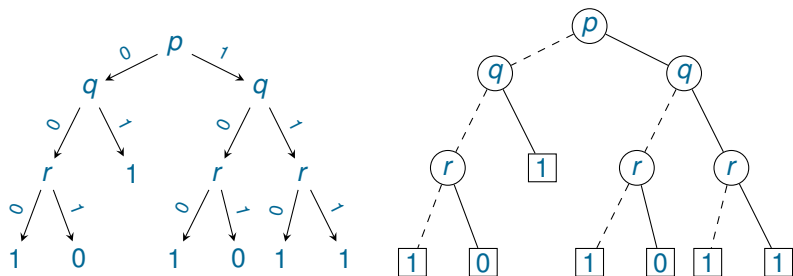


A circled node, such as  $\textcircled{p}$ , denote the choices on the variable of this node.

Terminal nodes are squared, for example  $\boxed{1}$ .

Solid lines correspond to choices when the variable is true, dashed lines to the choices when the variable is false.

# Binary Decision Tree

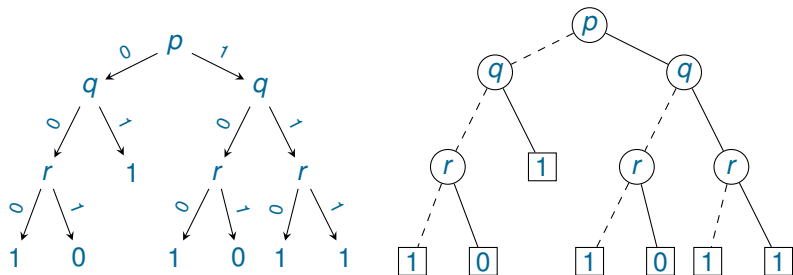


A circled node, such as  $\textcircled{p}$ , denote the choices on the variable of this node.

Terminal nodes are squared, for example  $\boxed{1}$ .

Solid lines correspond to choices when the variable is true, dashed lines to the choices when the variable is false.

# Binary Decision Tree

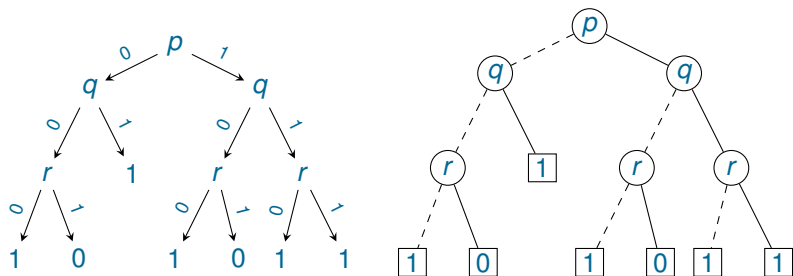


A circled node, such as  $\textcircled{p}$ , denote the choices on the variable of this node.

Terminal nodes are squared, for example  $\boxed{1}$ .

Solid lines correspond to choices when the variable is true, dashed lines to the choices when the variable is false.

# Binary Decision Tree

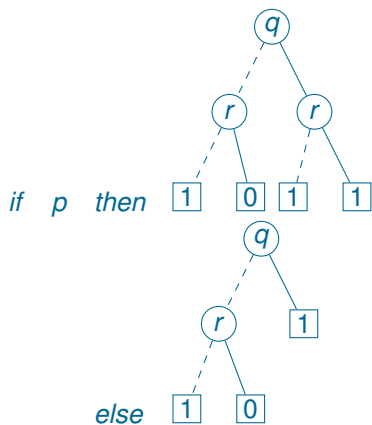
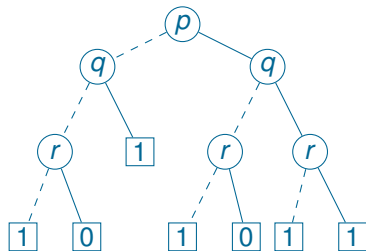


A circled node, such as  $\textcircled{p}$ , denote the choices on the variable of this node.

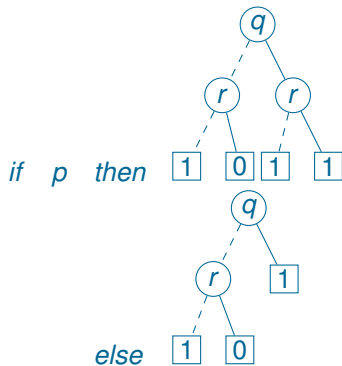
Terminal nodes are squared, for example  $\boxed{1}$ .

Solid lines correspond to choices when the variable is true, dashed lines to the choices when the variable is false.

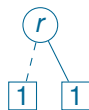
# Tests correspond to “if-then-else”



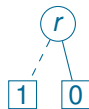
# Tests correspond to “if-then-else”



*if p then if q then*



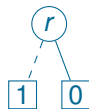
*else*



*else if q then*



*else*



## Tests correspond to “if-then-else”

<i>if</i>	<i>p</i>	<i>then</i>	<i>if</i>	<i>q</i>	<i>then</i>	<i>if</i>	<i>r</i>	<i>then</i>	⊤
								<i>else</i>	⊤
						<i>else</i>	<i>if</i>	<i>r</i>	⊤
								<i>else</i>	⊥
		<i>else</i>	<i>if</i>	<i>q</i>	<i>then</i>	⊤			
					<i>else</i>	<i>if</i>	<i>r</i>	<i>then</i>	⊤
								<i>else</i>	⊥

*if A then B else C*  $\equiv (A \rightarrow B) \wedge (\neg A \rightarrow C)$ .

## Tests correspond to “if-then-else”

<i>if</i>	<i>p</i>	<i>then</i>	<i>if</i>	<i>q</i>	<i>then</i>	<i>if</i>	<i>r</i>	<i>then</i>	⊤
								<i>else</i>	⊤
						<i>else</i>	<i>if</i>	<i>r</i>	⊤
								<i>else</i>	⊥
		<i>else</i>	<i>if</i>	<i>q</i>	<i>then</i>	⊤			
					<i>else</i>	<i>if</i>	<i>r</i>	<i>then</i>	⊤
								<i>else</i>	⊥

*if A then B else C*  $\equiv (A \rightarrow B) \wedge (\neg A \rightarrow C)$ .

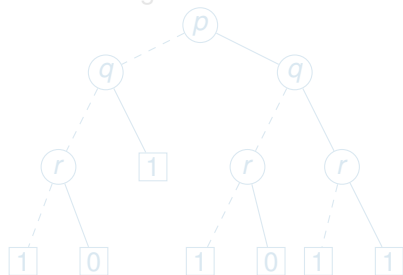
# If-Then-Else Normal Form

- ▶ The only connectives are *if-then-else*,  $\top$  and  $\perp$ ;
- ▶ The if-formula  $A$  in *if A then B else C* is atomic.

# Evaluating the Formula

We can evaluate a formula on an interpretation  $I$  if we know the binary decision tree of this formula.

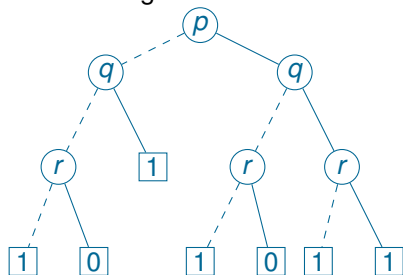
Consider, for example the interpretation  $\{p \mapsto 0, q \mapsto 0, r \mapsto 1\}$  and the following tree.



# Evaluating the Formula

We can evaluate a formula on an interpretation  $I$  if we know the binary decision tree of this formula.

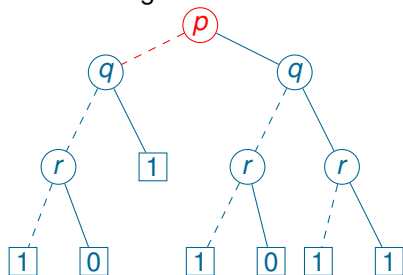
Consider, for example the interpretation  $\{p \mapsto 0, q \mapsto 0, r \mapsto 1\}$  and the following tree.



# Evaluating the Formula

We can evaluate a formula on an interpretation  $I$  if we know the binary decision tree of this formula.

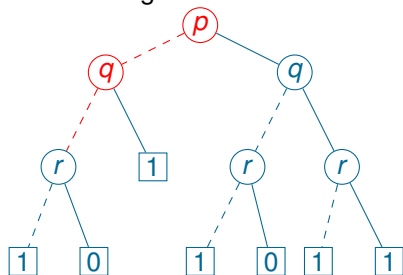
Consider, for example the interpretation  $\{p \mapsto 0, q \mapsto 0, r \mapsto 1\}$  and the following tree.



# Evaluating the Formula

We can evaluate a formula on an interpretation  $I$  if we know the binary decision tree of this formula.

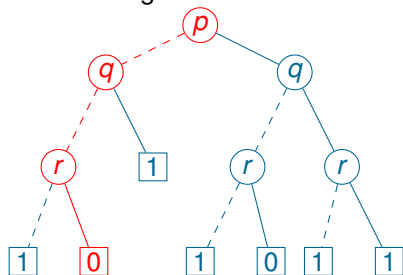
Consider, for example the interpretation  $\{p \mapsto 0, q \mapsto 0, r \mapsto 1\}$  and the following tree.



# Evaluating the Formula

We can evaluate a formula on an interpretation  $I$  if we know the binary decision tree of this formula.

Consider, for example the interpretation  $\{p \mapsto 0, q \mapsto 0, r \mapsto 1\}$  and the following tree.



# Properties

One needs **data structures** that

- ▶ facilitate **checking properties of formulas**, such as satisfiability or equivalence checking;
- ▶ facilitate **boolean operations** on this formulas, for example, taking conjunction of several formulas;
- ▶ give **compact representation** of formulas, or the boolean functions represented by the formulas.

Properties of binary decision trees:

- ▶ **Satisfiability checking** can be done in linear time;
- ▶ **Validity checking** can be done in linear time;
- ▶ **Equivalence checking** is **very hard**.
- ▶ Some boolean operations, e.g., conjunction, are **hard to implement**.

Are binary decision trees **compact**?

# Properties

One needs **data structures** that

- ▶ facilitate **checking properties of formulas**, such as satisfiability or equivalence checking;
- ▶ facilitate **boolean operations** on this formulas, for example, taking conjunction of several formulas;
- ▶ give **compact representation** of formulas, or the boolean functions represented by the formulas.

Properties of binary decision trees:

- ▶ **Satisfiability checking** can be done in **linear time**;
- ▶ **Validity checking** can be done in **linear time**;
- ▶ **Equivalence checking** is **very hard**.
- ▶ Some boolean operations, e.g., conjunction, are **hard to implement**.

Are binary decision trees **compact**?

# Properties

One needs **data structures** that

- ▶ facilitate **checking properties of formulas**, such as satisfiability or equivalence checking;
- ▶ facilitate **boolean operations** on this formulas, for example, taking conjunction of several formulas;
- ▶ give **compact representation** of formulas, or the boolean functions represented by the formulas.

Properties of binary decision trees:

- ▶ **Satisfiability checking** can be done in **linear time**;
- ▶ **Validity checking** can be done in **linear time**;
- ▶ **Equivalence checking** is **very hard**.
- ▶ **Some boolean operations**, e.g., conjunction, are **hard to implement**.

Are binary decision trees **compact**?

# Properties

One needs **data structures** that

- ▶ facilitate **checking properties of formulas**, such as satisfiability or equivalence checking;
- ▶ facilitate **boolean operations** on this formulas, for example, taking conjunction of several formulas;
- ▶ give **compact representation** of formulas, or the boolean functions represented by the formulas.

Properties of binary decision trees:

- ▶ **Satisfiability checking** can be done in **linear time**;
- ▶ **Validity checking** can be done in **linear time**;
- ▶ **Equivalence checking** is **very hard**.
- ▶ **Some boolean operations**, e.g., conjunction, are **hard to implement**.

Are binary decision trees **compact**?

# Properties

One needs **data structures** that

- ▶ facilitate **checking properties of formulas**, such as satisfiability or equivalence checking;
- ▶ facilitate **boolean operations** on this formulas, for example, taking conjunction of several formulas;
- ▶ give **compact representation** of formulas, or the boolean functions represented by the formulas.

Properties of binary decision trees:

- ▶ **Satisfiability checking** can be done in **linear time**;
- ▶ **Validity checking** can be done in **linear time**;
- ▶ **Equivalence checking** is **very hard**.
- ▶ **Some boolean operations**, e.g., conjunction, are **hard to implement**.

Are binary decision trees **compact**?

# Properties

One needs **data structures** that

- ▶ facilitate **checking properties of formulas**, such as satisfiability or equivalence checking;
- ▶ facilitate **boolean operations** on this formulas, for example, taking conjunction of several formulas;
- ▶ give **compact representation** of formulas, or the boolean functions represented by the formulas.

Properties of binary decision trees:

- ▶ **Satisfiability checking** can be done in **linear time**;
- ▶ **Validity checking** can be done in **linear time**;
- ▶ **Equivalence checking** is **very hard**.
- ▶ **Some boolean operations**, e.g., conjunction, are **hard to implement**.

Are binary decision trees **compact**?

# Algorithm for Building Binary Decision Trees

```
procedure bdt(A)  
input: propositional formula A  
output: a binary decision tree  
parameters: function select_atom  
begin  
  A := simplify(A)  
  if A =  $\perp$  then return 0  
  if A =  $\top$  then return 1  
  p := select_atom(A)  
  return tree(bdt( $A_p^\perp$ ), p, bdt( $A_p^\top$ ))  
end
```

Here *tree*( $T_1, p, T_2$ ) builds the tree



# Example

