

Data Structures for Large Propositional Formulas

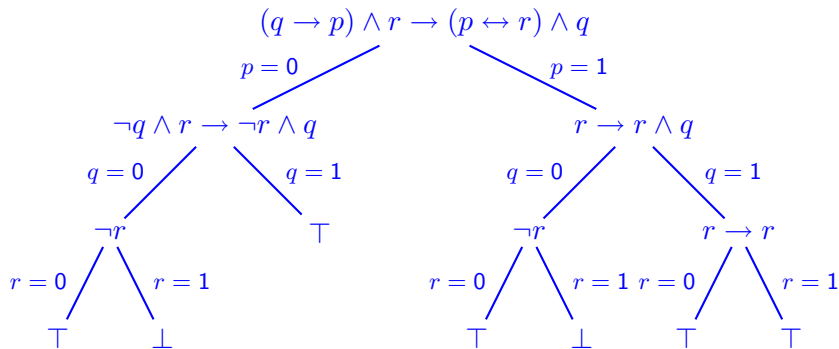
Suppose that **large propositional formulas are reused over and over again**. For example, we may

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

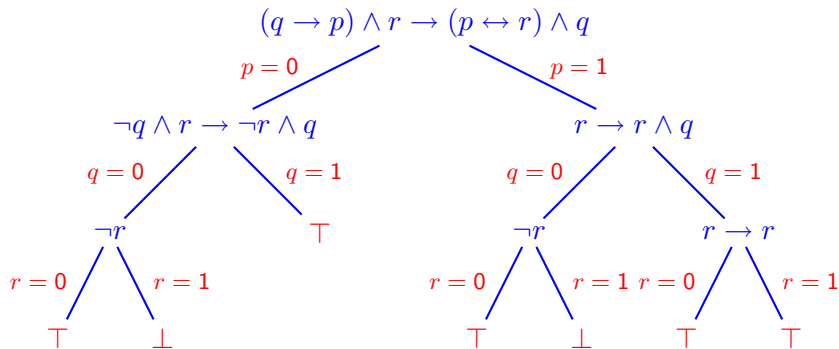
One needs **data structures** which

- 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.

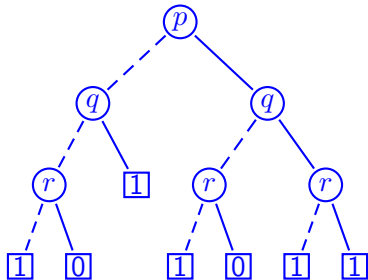
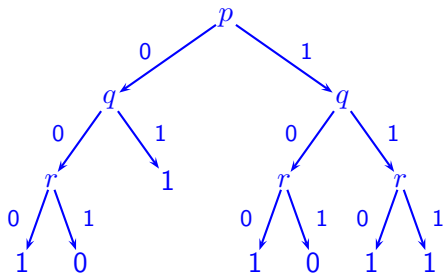
Splitting Tree



Splitting Tree



Binary Decision Tree



Algorithm for Building Binary Decision Trees

procedure *bdt*(A)

input: propositional formula A

output: a binary decision tree

parameters: function *select_atom*

begin

$A := \text{simplify}(A)$

if $A = \perp$ then return $\boxed{0}$

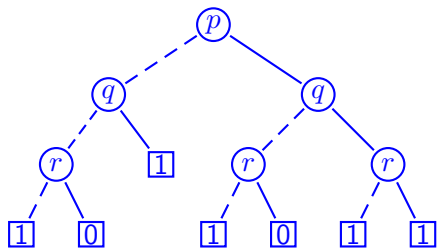
if $A = \top$ then return $\boxed{1}$

$p := \text{select_atom}(A)$

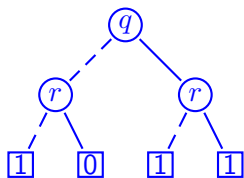
return $\text{tree}(\text{bdt}(A_p^\perp), p, \text{bdt}(A_p^\top))$

end

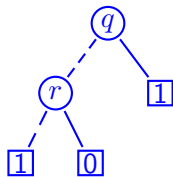
Tests correspond to “if-then-else”



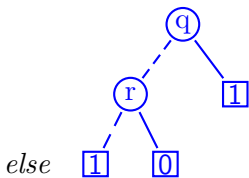
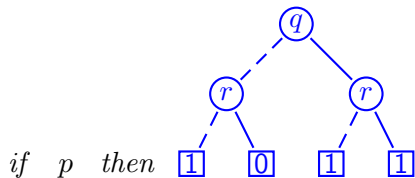
if p then



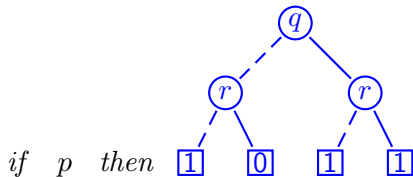
else



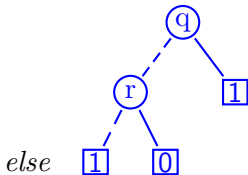
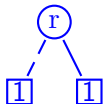
Tests correspond to “if-then-else”



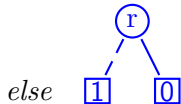
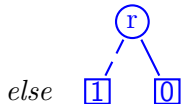
Tests correspond to “if-then-else”



if p then if q then



else if q then



If-Then-Else Normal Form

If-Then-Else Normal Form:

- if p then A else B where
 - p does not occur in A, B ,
 - A and B are in if-then-else normal form
- \perp, \top

Theorem. For any propositional formula there is an **equivalent** if-then-else normal form.

Properties of Binary Decision Trees

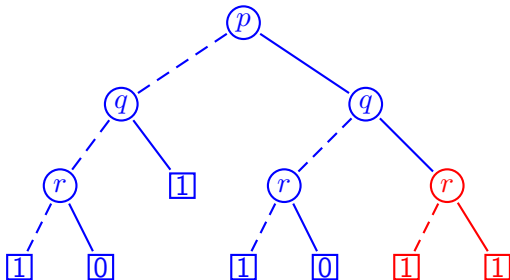
- Satisfiability checking can be done in **linear time**;
- Validity checking can be done in **linear time**;
- Equivalence checking is **very hard**.
- Some boolean operations, e.g., conjunction, are **hard to implement**.

Are binary decision trees **compact**?

Redundant Tests

Are binary decision trees **compact**?

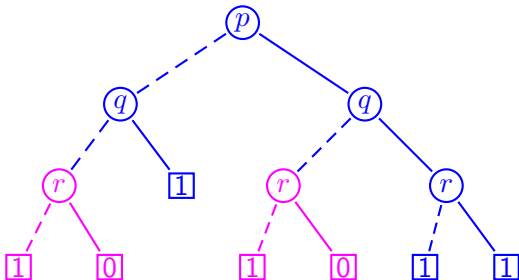
No: they may contain **redundant tests**:



Isomorphic Subtrees

Are binary decision trees **compact**?

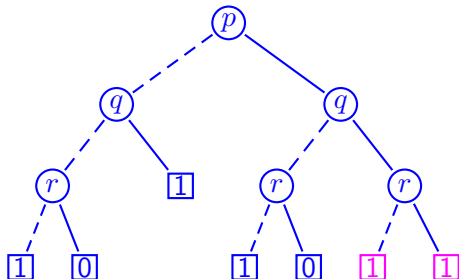
No: they may contain **isomorphic subtrees**:



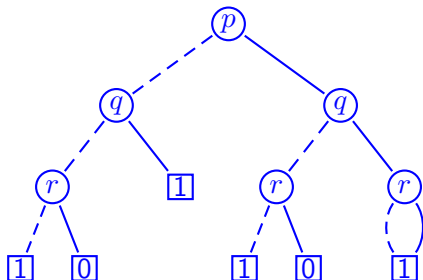
Binary Decision Diagrams

A **binary decision diagrams** or **BDD** is a dag built like a binary decision tree but containing **no redundant tests** and **no isomorphic subtrees**.

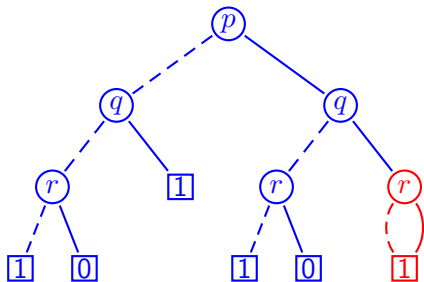
Transforming Binary Decision Tree into a BDD



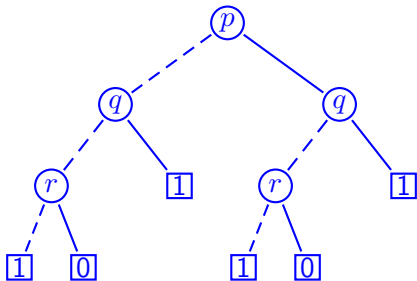
Transforming Binary Decision Tree into a BDD



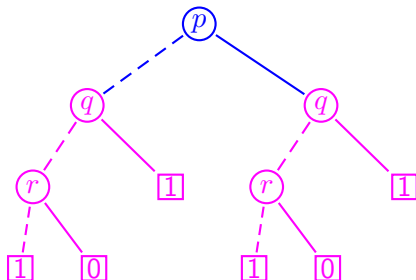
Transforming Binary Decision Tree into a BDD



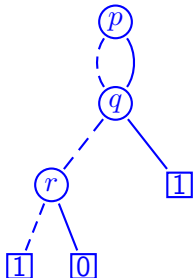
Transforming Binary Decision Tree into a BDD



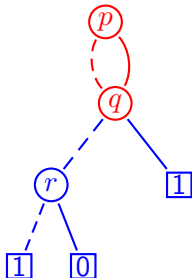
Transforming Binary Decision Tree into a BDD



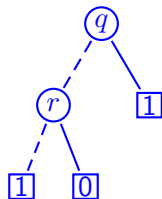
Transforming Binary Decision Tree into a BDD



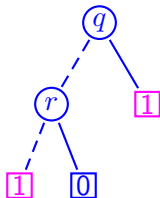
Transforming Binary Decision Tree into a BDD



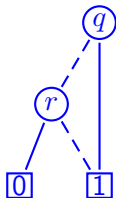
Transforming Binary Decision Tree into a BDD



Transforming Binary Decision Tree into a BDD



Transforming Binary Decision Tree into a BDD

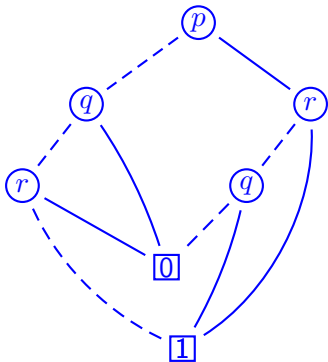


Properties

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

OBDDs

BDD but **not** an OBDD



Idea:

- introduce an **order** $>$ on atoms;
- make **tests** in this order.
- OBDD is **unique** for each boolean function (for a fixed order).

Properties

- Satisfiability checking can be done in constant time;
- Validity checking can be done in constant time;
- Equivalence checking can be done in constant time;
- Boolean operations e.g., conjunction, are easy to implement.

Algorithms on OBDDs

Suppose we have to compute an OBDD that represents a boolean function $f(b_1, \dots, b_n)$, for example $b_1 \vee \dots \vee b_n$ and we have already built OBDDs for b_1, \dots, b_n .

- We assume a **global dag** that contains all OBDDs so that all isomorphic subdags are **shared**.
- Use the property

$$\begin{aligned} f(\text{if } p \text{ then } l_1 \text{ else } r_1, \\ \dots, \\ \text{if } p \text{ then } l_n \text{ else } r_n) = \\ \text{if } p \text{ then } f(l_1, \dots, l_n) \text{ else } f(r_1, \dots, r_n). \end{aligned}$$