Automated Theorem Proving

Lecture 3: Propositional Logic Continued

Prof. Dr. Jasmin Blanchette

based on slides by Dr. Uwe Waldmann

Winter Semester 2026/27

2.4 Normal Forms

Many theorem proving calculi do not operate on arbitrary formulas, but only on some restricted class of formulas.

Normal Forms

We define conjunctions of formulas as follows:

$$\bigwedge_{i=1}^{0} F_i = \top.$$

$$\bigwedge_{i=1}^{1} F_i = F_1.$$

$$\bigwedge_{i=1}^{n+1} F_i = \bigwedge_{i=1}^{n} F_i \wedge F_{n+1} \quad \text{for } n \geq 1.$$

And analogously disjunctions:

$$\bigvee_{i=1}^{0} F_i = \bot.$$

$$\bigvee_{i=1}^{1} F_i = F_1.$$

$$\bigvee_{i=1}^{n+1} F_i = \bigvee_{i=1}^{n} F_i \vee F_{n+1} \quad \text{for } n \geq 1.$$

Literals and Clauses

A literal is either a propositional variable P or a negated propositional variable $\neg P$.

A clause is a (possibly empty) disjunction of literals.

CNF and DNF

A formula is in conjunctive normal form (CNF, also clausal normal form) if it is a conjunction of disjunctions of literals (or in other words, a conjunction of clauses).

A formula is in disjunctive normal form (DNF) if it is a disjunction of conjunctions of literals.

Warning: definitions in the literature differ:

- are complementary literals (e.g., P and $\neg P$) permitted?
- are duplicated literals permitted?
- are empty disjunctions/conjunctions permitted?

CNF and DNF

Checking the validity of CNF formulas or the unsatisfiability of DNF formulas is easy:

> A formula in CNF is valid if and only if each of its disjunctions contains a pair of complementary literals P and $\neg P$.
>
> Conversely, a formula in DNF is unsatisfiable if and only if each of its conjunctions contains a pair of complementary literals P and $\neg P$.

On the other hand, checking the unsatisfiability of CNF formulas or the validity of DNF formulas is coNP-complete.

Conversion to CNF/DNF

Proposition 2.4.1:
For every formula there is an equivalent formula in CNF (and also an equivalent formula in DNF).

Proof:
We describe a (naive) algorithm to convert a formula to CNF.

Apply the following rules as long as possible (modulo commutativity of $\wedge$ and $\vee$):

Step 1: Eliminate equivalences:

$$H[F \leftrightarrow G]_p \;\Rightarrow_{\mathsf{CNF}}\; H[(F \rightarrow G) \wedge (G \rightarrow F)]_p$$

Conversion to CNF/DNF

Step 2: Eliminate implications:

$$H[F \to G]_p \;\Rightarrow_{\mathsf{CNF}}\; H[\neg F \vee G]_p$$

Step 3: Push negations inward:

$$H[\neg(F \vee G)]_p \;\Rightarrow_{\mathsf{CNF}}\; H[\neg F \wedge \neg G]_p$$
$$H[\neg(F \wedge G)]_p \;\Rightarrow_{\mathsf{CNF}}\; H[\neg F \vee \neg G]_p$$

Step 4: Eliminate multiple negations:

$$H[\neg\neg F]_p \;\Rightarrow_{\mathsf{CNF}}\; H[F]_p$$

Conversion to CNF/DNF

Step 5: Push disjunctions inward:

$$H[(F \wedge F') \vee G]_p \Rightarrow_{\mathsf{CNF}} H[(F \vee G) \wedge (F' \vee G)]_p$$

Step 6: Eliminate $\top$ and $\bot$:

$$\begin{aligned}
H[F \wedge \top]_p &\Rightarrow_{\mathsf{CNF}} H[F]_p \\
H[F \wedge \bot]_p &\Rightarrow_{\mathsf{CNF}} H[\bot]_p \\
H[F \vee \top]_p &\Rightarrow_{\mathsf{CNF}} H[\top]_p \\
H[F \vee \bot]_p &\Rightarrow_{\mathsf{CNF}} H[F]_p \\
H[\neg\bot]_p &\Rightarrow_{\mathsf{CNF}} H[\top]_p \\
H[\neg\top]_p &\Rightarrow_{\mathsf{CNF}} H[\bot]_p
\end{aligned}$$

Conversion to CNF/DNF

Proving termination is easy for steps 2, 4, and 6; steps 1, 3, and 5 are slightly more complicated.

The resulting formula is equivalent to the original one and in CNF.

The conversion of a formula to DNF works in the same way, except that it is conjunctions that must be pushed inward in step 5. □

Conversion to CNF/DNF

Example 2.4.2:

The following steps convert the formula $(P \vee Q) \leftrightarrow R$ to CNF:

$$
\begin{aligned}
& (P \vee Q) \leftrightarrow R \\
\Rightarrow_{CNF(1)} \; & ((P \vee Q) \rightarrow R) \wedge (R \rightarrow (P \vee Q)) \\
\Rightarrow^{2}_{CNF(2)} \; & (\neg(P \vee Q) \vee R) \wedge (\neg R \vee (P \vee Q)) \\
\Rightarrow_{CNF(3)} \; & ((\neg P \wedge \neg Q) \vee R) \wedge (\neg R \vee (P \vee Q)) \\
\Rightarrow_{CNF(5)} \; & ((\neg P \vee R) \wedge (\neg Q \vee R)) \wedge (\neg R \vee (P \vee Q))
\end{aligned}
$$

Exploiting the associativity of $\wedge$ and $\vee$, we obtain

$$(\neg P \vee R) \wedge (\neg Q \vee R) \wedge (\neg R \vee P \vee Q)$$

Negation Normal Form (NNF)

The formula after application of step 4 is said to be in Negation Normal Form, i.e., it contains neither $\rightarrow$ nor $\leftrightarrow$ and negation symbols occur only in front of propositional variables (atoms).

Complexity

Conversion to CNF (or DNF) may produce a formula whose size is exponential in the size of the original one.

Example:

$$
\begin{aligned}
& (P \leftrightarrow Q) \leftrightarrow (R \leftrightarrow S) \\
\Rightarrow^{+}_{CNF}\ & (\neg P \vee \neg Q \vee \neg R \vee S) \wedge (\neg P \vee \neg Q \vee R \vee \neg S) \wedge \\
& (\neg P \vee Q \vee \neg R \vee \neg S) \wedge (\neg P \vee Q \vee R \vee S) \wedge \\
& (P \vee \neg Q \vee \neg R \vee \neg S) \wedge (P \vee \neg Q \vee R \vee S) \wedge \\
& (P \vee Q \vee \neg R \vee S) \wedge (P \vee Q \vee R \vee \neg S)
\end{aligned}
$$

2.5 Improving the CNF Transformation

The goal

> "Given a formula F,
>
> find an *equivalent* formula G in CNF"

is unpractical.

But if we relax the requirement to

> "Given a formula F,
>
> find an *equisatisfiable* formula G in CNF"

we can get an efficient transformation.

Improving the CNF Transformation

Literature:

Andreas Nonnengart and Christoph Weidenbach: Computing small clause normal forms, in *Handbook of Automated Reasoning*, pages 335-367. Elsevier, 2001.

Christoph Weidenbach: Automated Reasoning (Chapter 2). Textbook draft, 2021.

Tseitin Transformation

Proposition 2.5.1:

A formula $H[F]_p$ is satisfiable if and only if $H[Q]_p \wedge (Q \leftrightarrow F)$ is satisfiable, where Q is a new propositional variable that works as an abbreviation for F.

Tseitin Transformation

Satisfiability-preserving CNF transformation (Tseitin 1970):

Apply Prop. 2.5.1 recursively bottom up to all subformulas F in the original formula except $\bot$, $\top$, and literals.
This introduces a linear number of new propositional variables Q and definitions $Q \leftrightarrow F$.

Convert the resulting conjunction to CNF.
This increases the size only by an additional factor, since each formula $Q \leftrightarrow F$ yields at most four clauses in the CNF.

Tseitin Transformation

Example 2.5.2:

We convert the formula $(P \vee Q) \leftrightarrow R$ to CNF using the Tseitin transformation. First, we name the subformulas:

$$\underbrace{\underbrace{(P \vee Q)}_{Q_1} \leftrightarrow R}_{Q_2}.$$

Next, we compute the following equisatisfiable formula:

$$Q_2 \wedge (Q_2 \leftrightarrow (Q_1 \leftrightarrow R)) \wedge (Q_1 \leftrightarrow P \vee Q).$$

Finally, we apply the CNF transformation.

2.6 The DPLL Procedure

Goal:

Given a propositional formula in CNF (or alternatively, a finite set N of clauses), check whether it is satisfiable (and optionally: output *one* solution if it is satisfiable).

Preliminaries

Recall:

$\mathcal{A} \models C$ if and only if $\mathcal{A} \models L$ for some literal $L \in C$.

$\mathcal{A} \models N$ if and only if $\mathcal{A} \models C$ for all clauses C in N.

Preliminaries

Assumptions:

Clauses contain neither duplicated literals nor complementary literals.

The order of literals in a clause is irrelevant.

$\Rightarrow$ Clauses behave like *sets* of literals.

Notation:

We use the notation $C \vee L$ to denote a clause with some literal L and a clause rest C. Here L need *not* be the last literal of the clause and C may be empty.

$\overline{L}$ is the complementary literal of L, i.e., $\overline{P} = \neg P$ and $\overline{\neg P} = P$.

Partial Valuations

Since we will construct satisfying valuations incrementally,
we consider partial valuations (that is, partial mappings $\mathcal{A} : \Pi \to \{0, 1\}$).

Every partial valuation $\mathcal{A}$ corresponds to a set M of literals that does not contain complementary literals, and vice versa:

- $\mathcal{A}(L)$ is true if $L \in M$.
- $\mathcal{A}(L)$ is false if $\overline{L} \in M$.
- $\mathcal{A}(L)$ is undefined if neither $L \in M$ nor $\overline{L} \in M$.

We will use $\mathcal{A}$ and M interchangeably.

Partial Valuations

A clause is true in a partial valuation $\mathcal{A}$ (or in a set M of literals)
if one of its literals is true;
it is false (or "conflicting") if all its literals are false;
otherwise it is undefined (or "unresolved").

Unit Clauses

Observation:
Let $\mathcal{A}$ be a partial valuation. If the set N contains a clause C such that all literals in C but one are false in $\mathcal{A}$, then the following properties are equivalent:

- there is a valuation that is a model of N and extends $\mathcal{A}$.
- there is a valuation that is a model of N and extends $\mathcal{A}$ and makes the remaining literal L of C true.

C is called a unit clause; L is called a unit literal.

Pure Literals

One more observation:
Let $\mathcal{A}$ be a partial valuation and P a variable that is undefined in $\mathcal{A}$. If P occurs only positively (or only negatively) in the unresolved clauses in N, then the following properties are equivalent:

- there is a valuation that is a model of N and extends $\mathcal{A}$.
- there is a valuation that is a model of N and extends $\mathcal{A}$ and assigns 1 (0) to P.

P is called a pure literal.

The Davis-Putnam-Logemann-Loveland Procedure

```
boolean DPLL(literal set M, clause set N) {
    if (all clauses in N are true in M) return true;
    elsif (some clause in N is false in M) return false;
    elsif (N contains unit literal L) return DPLL(M ∪ {L}, N);
    elsif (N contains pure literal L) return DPLL(M ∪ {L}, N);
    else {
        let P be some undefined variable in N;
        if (DPLL(M ∪ {¬P}, N)) return true;
        else return DPLL(M ∪ {P}, N);
    }
}
```

The Davis-Putnam-Logemann-Loveland Procedure

Initially, DPLL is called with an empty literal set and the clause set N.

The Davis-Putnam-Logemann-Loveland Procedure

Example 2.6.1:

We run the DPLL procedure on the clause set

$$N = \{\neg P \vee R, \neg Q \vee R, \neg R \vee P \vee Q\}.$$

We start with $M = \emptyset$.

Since there are no unit or pure literals, we arbitrarily set R to false: $M := \{\neg R\}$.

Then $\neg P \vee R$ contains the unit literal $\neg P$, so $M := \{\neg R, \neg P\}$.

Moreover, $\neg Q \vee R$ contains the unit literal $\neg Q$, so $M := \{\neg R, \neg P, \neg Q\}$.

At this point, all clauses in N are true in M, so the procedure stops with the model M.

2.7 From DPLL to CDCL

The DPLL procedure can be improved significantly:

The pure literal check is done only while preprocessing
(otherwise it is too expensive).

If a conflict is detected,
information is reused by conflict analysis and learning.

The algorithm is implemented iteratively
$\Rightarrow$ the backtrack stack is managed explicitly
(it may be possible and useful to backtrack more than one level).

The branching variable is not chosen randomly.

Under certain circumstances, the procedure is restarted.

From DPLL to CDCL

The improved procedure is called CDCL: conflict-driven clause learning.

From DPLL to CDCL

Literature:

Lintao Zhang and Sharad Malik: The Quest for Efficient Boolean Satisfiability Solvers, Proc. CADE-18, LNAI 2392, pp. 295–312, Springer, 2002.

Robert Nieuwenhuis, Albert Oliveras, Cesare Tinelli: Solving SAT and SAT Modulo Theories—From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T), pp. 937–977, *Journal of the ACM*, 53(6), 2006.

From DPLL to CDCL

Literature:

Armin Biere, Marijn Heule, Hans van Maaren, Toby Walsh (eds.):
Handbook of Satisfiability, IOS Press, 2009

Daniel Le Berre's slides at VTSA '09:
`http://www.mpi-inf.mpg.de/vtsa09/`.

See also Johannsen's SAT Solving practical.

2.8 OBDDs

Goal:

Efficient manipulation of (equivalence classes of) propositional formulas.

Method: Minimized graph representation of decision trees,
based on a fixed ordering on propositional variables.

$\Rightarrow$ Canonical representation of formulas.

$\Rightarrow$ Satisfiability checking as a side effect.

OBDDs

Literature:

Randal E. Bryant: Graph-Based Algorithms for Boolean Function Manipulation, IEEE Transactions on Computers, 35(8):677-691, 1986.

Randal E. Bryant: Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams, ACM Computing Surveys, 24(3), September 1992, pp. 293-318.

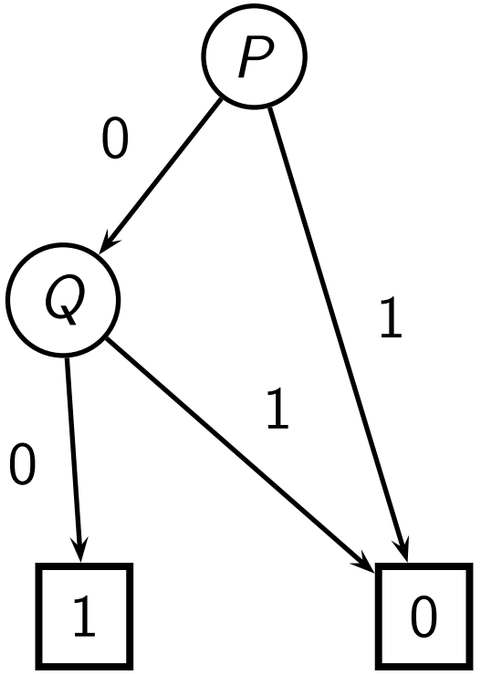
Michael Huth and Mark Ryan: *Logic in Computer Science: Modelling and Reasoning about Systems*, Chapter 6.1/6.2; Cambridge Univ. Press, 2000.

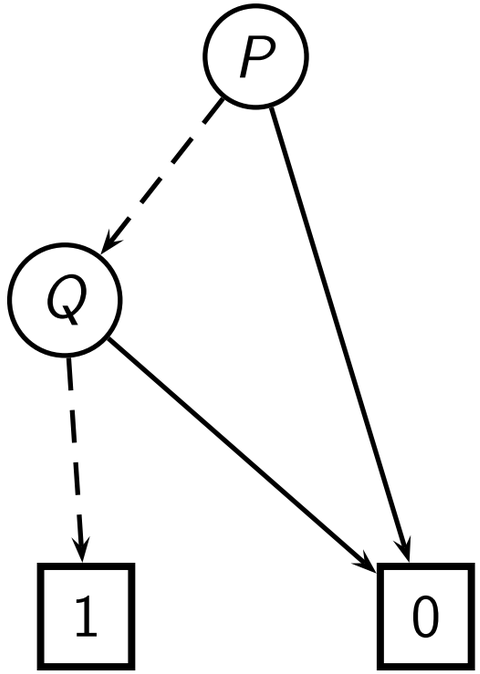
BDDs

BDD (Binary decision diagram):

- Labeled DAG (directed acyclic graph).
- Leaf nodes:
  - labeled with a truth value (0 or 1).
- Nonleaf nodes (inner nodes):
  - labeled with a propositional variable,
  - exactly two outgoing edges,
    labeled with 0 ($\dashrightarrow$) and 1 ($\longrightarrow$).

BDDs





BDDs

Every BDD node can be interpreted as a mapping from valuations to truth values:
Traverse the BDD from the given node to a leaf node;
for any node labeled with P, take the 0 edge or 1 edge
depending on whether $\mathcal{A}(P)$ is 0 or 1.

$\Rightarrow$ Compact representation of truth tables.

OBDDs

OBDD (Ordered BDD):

Let $<$ be a total ordering of the propositional variables.

An OBDD w.r.t. $<$ is a BDD where every edge from a nonleaf node leads either to a leaf node or to a nonleaf node with a strictly larger label w.r.t. $<$.

OBDDs

OBDDs can be converted to formulas:

A leaf node $\boxed{0}$ represents $\bot$ (or any unsatisfiable formula).

A leaf node $\boxed{1}$ represents $\top$ (or any valid formula).

If a nonleaf node v has the label P,
and its 0 edge leads to a node representing the formula F_0,
and its 1 edge leads to a node representing the formula F_1,
then v represents the formula

$$\begin{aligned} F &\models\models \text{ if } P \text{ then } F_1 \text{ else } F_0 \\ &\models\models (P \wedge F_1) \vee (\neg P \wedge F_0) \\ &\models\models (P \rightarrow F_1) \wedge (\neg P \rightarrow F_0) \end{aligned}$$

OBDDs

Conversely, formulas can be converted to OBDDs:

Define $F\{P \mapsto H\}$ as the formula obtained from F by replacing every occurrence of P in F by H.

For every formula F and propositional variable P:

$$F \models\!\models (P \wedge F\{P \mapsto \top\}) \vee (\neg P \wedge F\{P \mapsto \bot\})$$

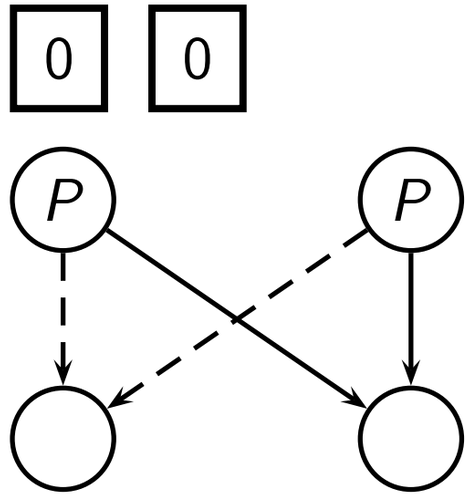
This is the *Shannon expansion* of F, originally due to Boole.
Using the Shannon expansion recursively, we can easily construct an OBDD.

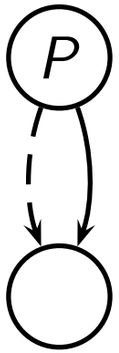
Consequence: Every formula F can be represented by an OBDD.

Reduced OBDDs

An OBDD is called *reduced* if it has

- no duplicated leaf nodes
- no duplicated nonleaf nodes
- no redundant tests



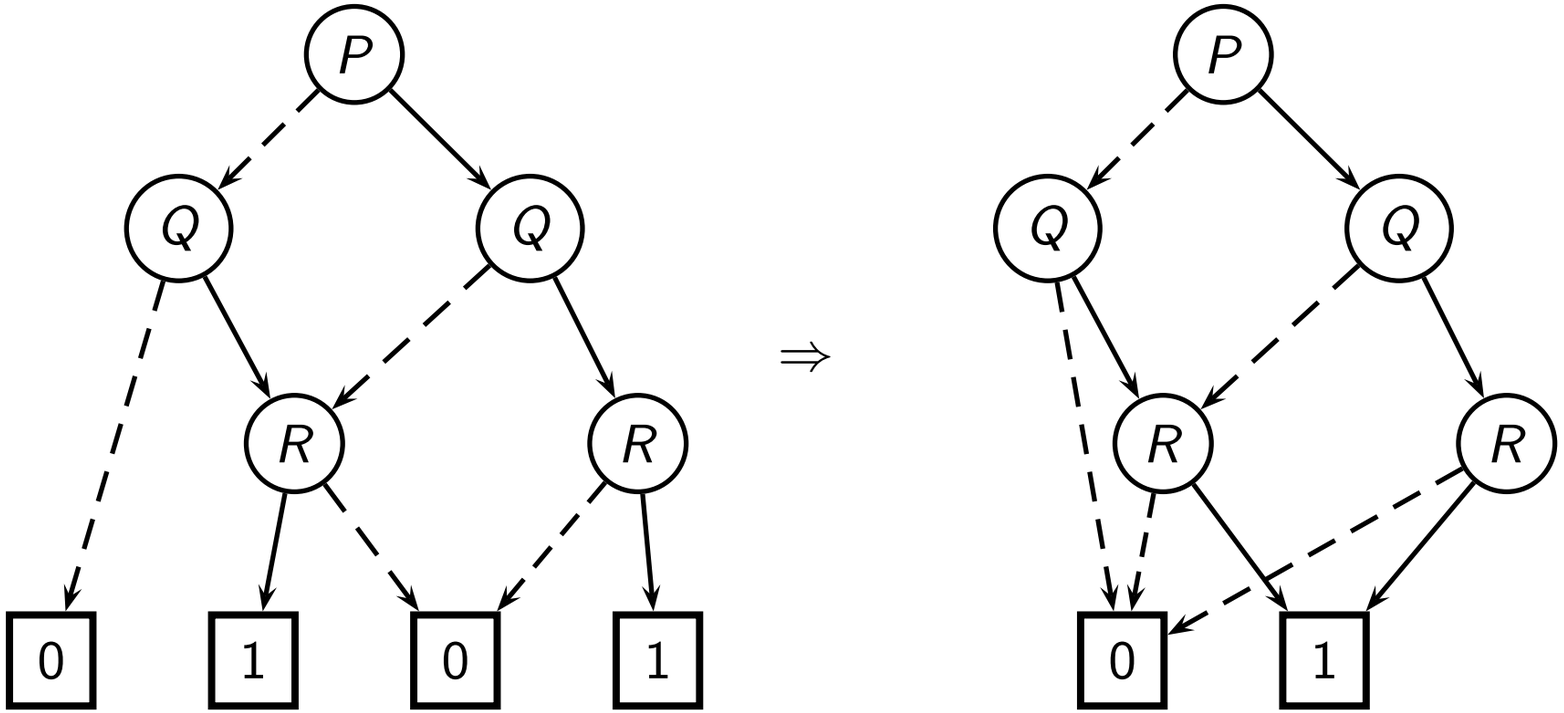


Reduced OBDDs

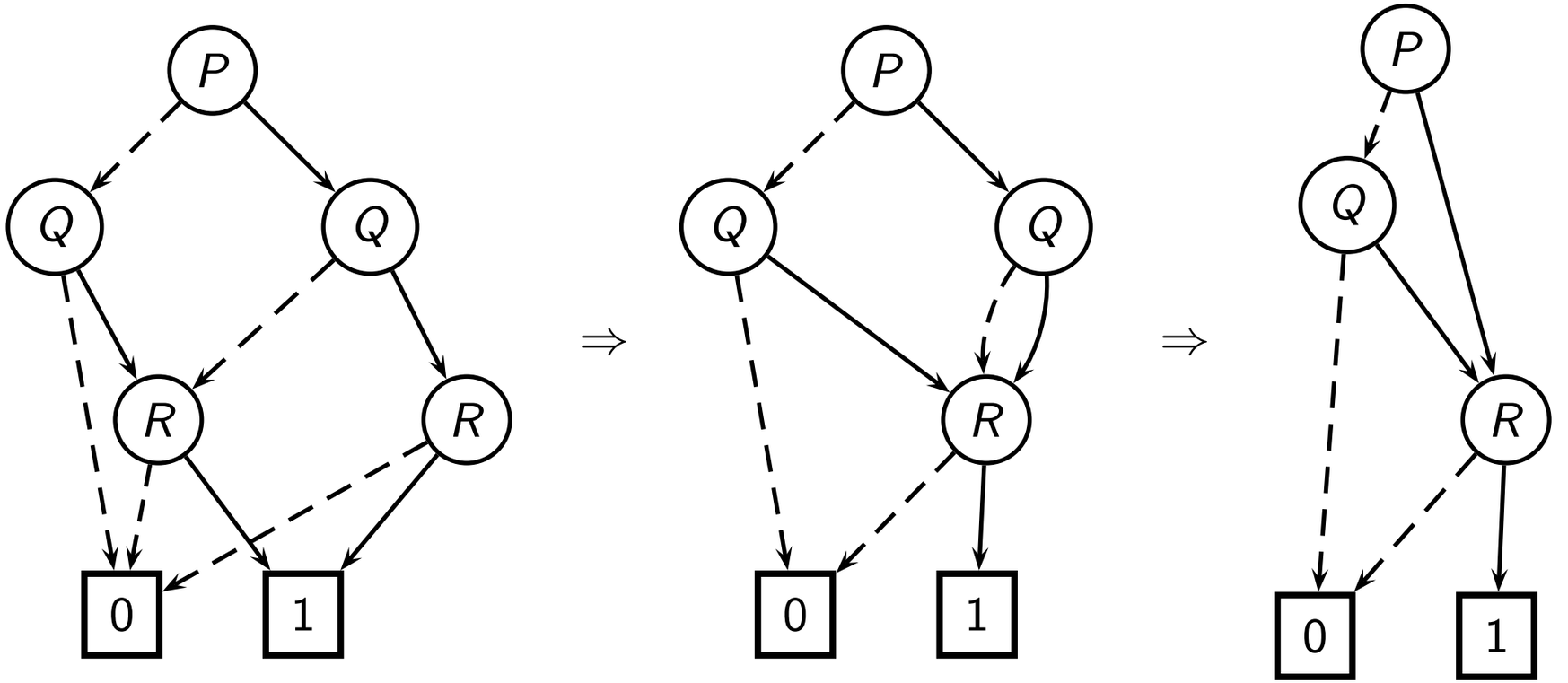
Theorem 2.8.1 (Bryant 1986):

Every OBDD can be converted into an equivalent reduced OBDD.

Example: Reducing an OBDD



Example: Reducing an OBDD



Reduced OBDDs

Assumptions from now on:

One fixed ordering $>$.

We consider only reduced OBDDs.

Reduced OBDDs

Theorem 2.8.2 (Bryant 1986):
If v and v' are two different nodes in a reduced OBDD, then they represent nonequivalent formulas.

Reduced OBDDs

Corollary 2.8.3:

F is valid if and only if it is represented by $\boxed{1}$.

F is unsatisfiable if and only if it is represented by $\boxed{0}$.

Variable Ordering in OBDDs

The size of an OBDD for a given formula depends crucially on the chosen ordering of the propositional variables:

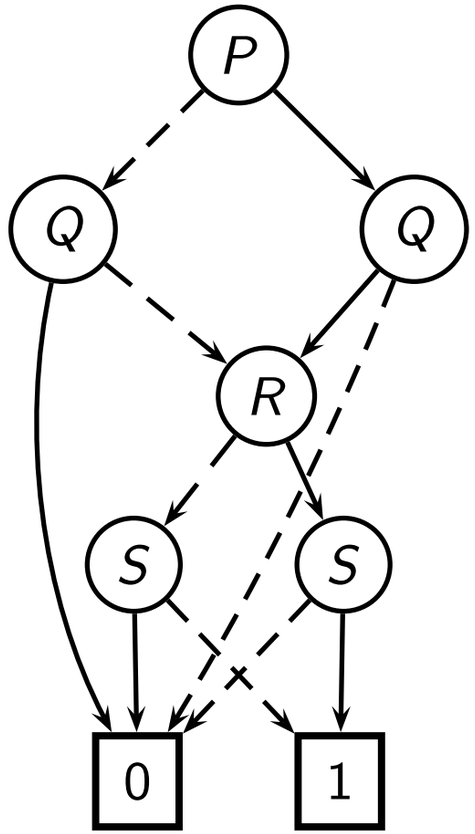
Let $F = (P_1 \wedge P_2) \vee (P_3 \wedge P_4) \vee \cdots \vee (P_{2n-1} \wedge P_{2n})$.

$P_1 < P_2 < P_3 < P_4 < \cdots < P_{2n-1} < P_{2n}$: $2n + 2$ nodes.

$P_1 < P_3 < \cdots < P_{2n-1} < P_2 < P_4 < \cdots < P_{2n}$: 2^{n+1} nodes.

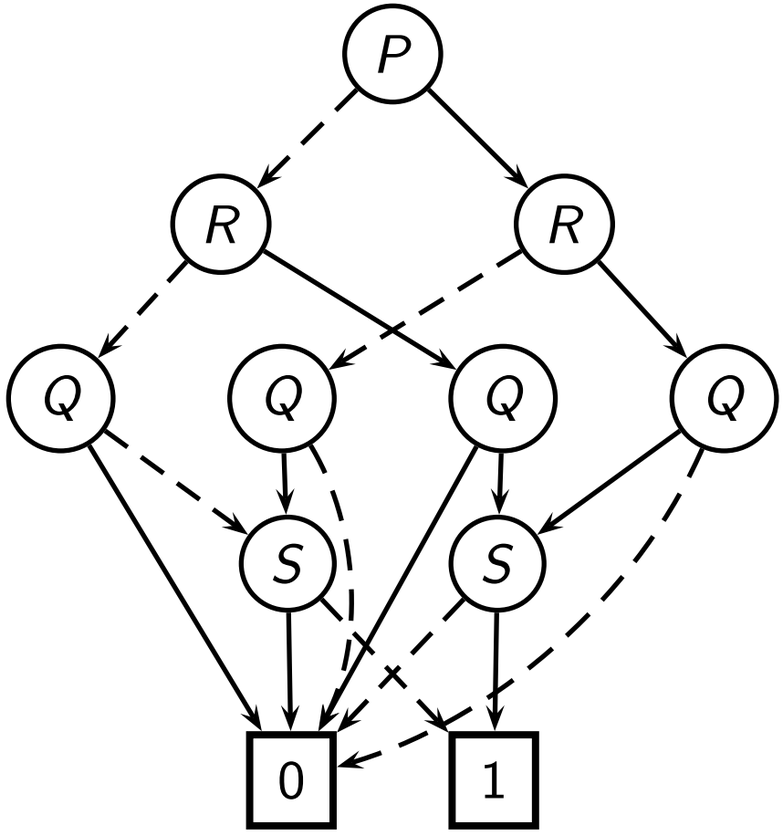
Example: Variable Ordering in OBDDs

The reduced OBDD for $(P \leftrightarrow Q) \wedge (R \leftrightarrow S)$ with variable ordering $P < Q < R < S$.



Example: Variable Ordering in OBDDs

The reduced OBDD for $(P \leftrightarrow Q) \wedge (R \leftrightarrow S)$ with variable ordering $P < R < Q < S$.



Operations on OBDDs

Even worse: There are (practically relevant) formulas for which the OBDD has exponential size *for every ordering* of the propositional variables.

2.9 Other Calculi

FRAIGs (functionally reduced and-inverter graphs)

Ordered resolution

Tableau calculus

Hilbert calculus

Sequent calculus

Natural deduction