Tecniche di Verifica

Introduction to Propositional Logic

Logic

A formal logic is defined by its *syntax* and *semantics*.

Syntax

- An *alphabet* is a set of symbols.
- A finite sequence of these symbols is called an *expression*.
- A set of rules defines the *well-formed* expressions (*well-formed femulae* or *wff*'s).

Semantics

- Gives meaning to well-formed expressions
- Formal notions of induction and recursion are required to provide a rigorous semantics.

Propositional (Boolean) Logic

Propositional logic is simple but extremely important in Computer Science

1. It is the basis for day-to-day reasoning (e.g., in programming)
2. It is the theory behind digital circuits.
3. Many problems can be translated into propositional logic.
4. It is an important part of more complex logics, such as: *First-Order Logic (*also called *Predicate Logic)*, *Modal and Temporal logic*, which we will discuss later.

Propositional Logic: Syntax

Alphabet

(	Left parenthesis	Begin group
)	Right parenthesis	End group
$\neg$	Negation symbol	English: not
$\wedge$	Conjunction symbol	English: and
$\vee$	Disjunction symbol	English: or (inclusive)
$\rightarrow$	Conditional symbol	English: if, then
$\leftrightarrow$	Bi-conditional symbol	English: if and only if
$\mathbf{A_1}$	First propositional symbol	
$\mathbf{A_2}$	Second propositional symbol	
...		
$\mathbf{A_N}$	N-th propositional symbol	
...		

We are assuming a *countable* alphabet, but most of our conclusions hold equally well for an *uncountable* alphabet.

Propositional Logic: Syntax

Alphabet

Propositional connective symbols: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$

Logical symbols: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, (,).

Parameters or *nonlogical symbols*: $\mathbf{A_1}$, $\mathbf{A_2}$, $\mathbf{A_3}$, . . .

The *meaning* of *logical symbols* is always the same.

The *meaning* of *nonlogical symbols* depends on the context.

From now on, let **AP** be the set $\{\mathbf{A_1}, \mathbf{A_2}, \mathbf{A_3}, \ldots\}$, called the set of *atomic propositions*.

Propositional Logic: Syntax

A *propositional expression* is a sequence of symbols. A sequence is denoted explicitly by a comma separated list enclosed in angle brackets:

Examples

$(, A_1 , \wedge , A_3 ,)$	$(A_1 \wedge A_3)$
$(, (, \neg , A_1 ,) , \rightarrow , A_2 ,)$	$((\neg A_1) \rightarrow A_2)$
$) ,) , \leftrightarrow , A_1 , \neg , A_5$	$)) \leftrightarrow A_1 \neg A_5$

We will write these sequences as simple strings of symbols, with the understanding that the ***formal structure*** represented is a sequence containing exactly the symbols in the string.

The formal meaning becomes important when trying to prove things about expressions.

We want to restrict the kinds of expressions that will be allowed.

Propositional Logic: Syntax

Let us define the set **W** of *well-formed formulas* (*wff*'s).

(a) Every expression consisting of a single propositional symbol is in **W** (**AP** $\subseteq$ **W**);

(b) If α and β are in **W**, then so are $(\neg \alpha)$, $(\alpha \vee \beta)$, $(\alpha \wedge \beta)$, $(\alpha \rightarrow \beta)$ and $(\alpha \leftrightarrow \beta)$;

(c) No other expression is in **W**.

This definition is *inductive*: the set being defined is used as part of the definition.

How would you use this definition to prove that the expression $(\leftrightarrow A_1 \neg A_5)$ is not a *wff*?

Propositional Logic: Semantics

Intuitively, given a *wff* α and the truth value (either *true* or *false*) for each propositional symbol in α (the atomic propositions), we should be able to determine the truth value of α.

How do we make this precise?

Let υ be a function from **AP** to {0,1}, where **0** represents *false* and **1** represents *true* . Recall that in the inductive definition of *wff* 's, **AP** contains the propositional symbols.

Any function υ defined as above is called ***truth assignment***, and represent a possible ***propositional model***.

Now, we define the ***satisfaction relation*** $\models$ between υ and elements of W.

Propositional Logic: Semantics

Let υ be a function from **AP** to {0,1}, where **0** represents *false* and **1** represents *true* .

The ***satisfaction relation*** $\models$ between υ and elements of W is defined inductively as follows:

- $\upsilon \models \mathbf{A_i}$ if and only if $\upsilon(\mathbf{A_i}) = \mathbf{1}$
- $\upsilon \models (\neg\, \alpha)$ if and only if $\upsilon \not\models \alpha$
- $\upsilon \models (\alpha \wedge \beta)$ if and only if $\upsilon \models \alpha$ and $\upsilon \models \beta$
- $\upsilon \models (\alpha \vee \beta)$ if and only if $\upsilon \models \alpha$ or $\upsilon \models \beta$
- $\upsilon \models (\alpha \rightarrow \beta)$ if and only if $\upsilon \not\models \alpha$ or $\upsilon \models \beta$
- $\upsilon \models (\alpha \leftrightarrow \beta)$ if and only if $\upsilon \models \alpha$ if and only if $\upsilon \models \beta$

Truth Tables

There are other ways to present the semantics which are less formal but perhaps more intuitive.

α	$\neg\alpha$
1	0
0	1

α	β	$\alpha \wedge \beta$
1	1	1
1	0	0
0	1	0
0	0	0

α	β	$\alpha \vee \beta$
1	1	1
1	0	1
0	1	1
0	0	0

α	β	$\alpha \rightarrow \beta$
1	1	1
1	0	0
0	1	1
0	0	1

α	β	$\alpha \leftrightarrow \beta$
1	1	1
1	0	0
0	1	0
0	0	1

Truth Tables: Examples

Truth tables can be used to calculate all possible truth values for a given *wff* with respect to any possible assignment υ

There is a row for each possible truth assignment υ to the propositional atoms and connectives.

A_1	A_2	A_3	$(A_1$	$\vee$	$(A_2$	$\wedge$	$\neg A_3))$
1	1	1	1	1	1	0	0
1	1	0	1	1	1	1	1
1	0	1	1	1	0	0	0
1	0	0	1	1	0	0	1
0	1	1	0	0	1	0	0
0	1	0	0	1	1	1	1
0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	1

Satisfiability and Validity

A *wff* α is *satisfiable* if there exists some truth assignment υ which satisfies α.

Suppose Σ is a set of *wff*'s. Then Σ *tautologically implies* α, or $\Sigma \models \alpha$, if ***every truth assignment*** which satisfies each formula in Σ also satisfies α.

Particular cases:

- If $\varnothing \models \alpha$, then we say is a *tautology* or is *valid* and we write $\models \alpha$
- If Σ is *unsatisfiable*, then $\Sigma \models \alpha$ for every *wff* α
- If $\alpha \models \beta$ (shorthand for $\{\alpha\} \models \beta$) and $\beta \models \alpha$, then α and β are *tautologically equivalent*.
- $\Sigma \models \alpha$ if and only if the *wff* $\wedge \Sigma \rightarrow \alpha$ is ***valid*** ($\models \wedge\Sigma \rightarrow \alpha$).

Some Tautologies

Associative and Commutative laws for $\wedge, \vee, \leftrightarrow$

Distributive Laws

- $(A \wedge (B \vee C)) \leftrightarrow ((A \wedge B) \vee (A \wedge C))$.
- $(A \vee (B \wedge C)) \leftrightarrow ((A \vee B) \wedge (A \vee C))$.

Negation

- $\neg\neg A \leftrightarrow A$
- $\neg(A \rightarrow B) \leftrightarrow (A \wedge \neg B)$
- $\neg(A \leftrightarrow B) \leftrightarrow ((A \wedge \neg B) \vee (\neg A \wedge B))$

De Morgan's Laws

- $\neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B)$
- $\neg(A \vee B) \leftrightarrow (\neg A \wedge \neg B)$

More Tautologies

Implication

- $(A \rightarrow B) \leftrightarrow (\neg A \vee B)$

Excluded Middle

- $A \vee \neg A$

Contradiction

- $\neg(A \wedge \neg A)$

Contraposition

- $(A \rightarrow B) \leftrightarrow (\neg B \rightarrow \neg A)$

Exportation

- $((A \wedge B) \rightarrow C) \leftrightarrow (A \rightarrow (B \rightarrow C))$

Examples

- $(A \vee B) \wedge (\neg A \vee \neg B)$ is satisfiable, but not valid.
- $(A \vee B) \wedge (\neg A \vee \neg B) \wedge (A \leftrightarrow B)$ is unsatisfiable.
- $\{A, A \rightarrow B\} \models B$
- $\{A, \neg A\} \models (A \wedge \neg A)$
- $\neg(A \wedge B)$ is tautologically equivalent to $\neg A \vee \neg B$

Suppose you have an algorithm *SAT* which would take a *wff* α as input and return *true* if α is satisfiable and *false* otherwise.

How would you use this algorithm to verify each of the claims made above?

Examples

- $(A \vee B) \wedge (\neg A \vee \neg B)$ is satisfiable, but not valid.
- $(A \vee B) \wedge (\neg A \vee \neg B) \wedge (A \leftrightarrow B)$ is unsatisfiable.
- $\{A, A \rightarrow B\} \models B$ $\quad (A \wedge (A \rightarrow B) \wedge (\neg B))$
- $\{A, \neg A\} \models (A \wedge \neg A)$ $\quad (A \wedge (\neg A) \wedge \neg(A \wedge \neg A))$
- $\neg(A \wedge B)$ is tautologically equivalent to $\neg A \vee \neg B$

  $\neg(\neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B))$

Suppose you have an algorithm *SAT* which would take a *wff* α as input and return *true* if α is satisfiable and *false* otherwise.

How would you use this algorithm to verify each of the claims made above?

Examples

- $(A \vee B) \wedge (\neg A \vee \neg B)$ is satisfiable, but not valid.
- $(A \vee B) \wedge (\neg A \vee \neg B) \wedge (A \leftrightarrow B)$ is unsatisfiable.
- $\{A, A \rightarrow B\} \models B$
- $\{A, \neg A\} \models (A \wedge \neg A)$
- $\neg(A \wedge B)$ is tautologically equivalent to $\neg A \vee \neg B$

Now suppose you had an algorithm *CHECKVALID* which returns *true* when α is valid and *false* otherwise.

How would you verify the claims given this algorithm?

Satisfiability and ***validity*** are dual notions: α ***is unsatisfiable*** if and only if $\neg\alpha$ ***is valid***.

Satisfiability with Truth Tables

An Algorithm for Satisfiability

To check whether α is satisfiable, form the truth table for α. If there is a row in which **1** appears as the value for α, then α is satisfiable. Otherwise, α is unsatisfiable.

Notice that this algorithm has ***exponential complexity***, since the number of different rows in a truth table is exponential ($\mathbf{2^n}$) in the number **n** of atomic propositions occurring in α.

An Algorithm for Tautological Implication

To check whether $\{\alpha_1,\ldots,\alpha_k\} \models \beta$, check the satisfiability of the ***wff*** $(\alpha_1 \wedge \ldots \wedge \alpha_k) \wedge (\neg \beta)$. If it is unsatisfiable, then $\{\alpha_1,\ldots,\alpha_k\} \models \beta$, otherwise $\{\alpha_1,\ldots,\alpha_k\} \not\models \beta$.

Notice also that the computational ***complexity*** of the ***propositional satisfiability*** is **NP-Complete**!

Boolean Functions

- **f : Domain $\rightarrow$ Range**
- Boolean function:
  - **Domain = $\{0, 1\}^n$ = $\{0,1\} \times \ldots. \times \{0,1\}$.**
  - **Range = $\{0, 1\}$**
  - **f** is a function of **n** boolean variables.
- How many boolean functions of 3 variables are there?

Boolean Functions

- **f : Domain $\rightarrow$ Range**
- Boolean function:
  - **Domain = $\{0, 1\}^n$ = {0,1} × …. × {0,1}.**
  - **Range = {0, 1}**
  - **f** is a function of **n** boolean variables.
- How many boolean functions of 3 variables are there?
  - Answer : $2^{2^3} = 2^8$ **!**

There are 2^3 different input points and **2** possible output values for each input point. 2^{2^3} is also the number of different ***n***-ary propositional connectives

Boolean Functions & Truth Tables

x	y	z	g
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$$g : \{0, 1\} \times \{0, 1\} \times \{0, 1\} \to \{0, 1\}$$

Boolean Expressions

- Given a set of ***Boolean variables x,y,…*** and the constants **1** (true) and **0** (false):

$$t ::= x \mid 0 \mid 1 \mid \neg t \mid t \wedge t \mid t \vee t \mid t \rightarrow t \mid t \leftrightarrow t$$

- The semantics of ***Boolean Expressions*** is defined by means of ***truth tables*** as usual.
- Given an ordering of Boolean variables, ***Boolean expressions*** can be used to express ***Boolean functions***.

Boolean expressions

- Boolean functions can also be represented as boolean (propositional) expressions.
- **x $\wedge$ y** represents the function:
  - **f : {0, 1} $\times$ {0, 1} $\rightarrow$ {0, 1}**
    - **f(0, 0) =**
    - **f(0, 1) =**
    - **f(1, 0) =**
    - **f(1, 1) =**

Boolean expressions

- Boolean functions can also be represented as boolean (propositional) expressions.
- **$x \wedge y$** represents the function:
  - **$f : \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$**
    - **$f(0, 0) = 0$**
    - **$f(0, 1) = 0$**
    - **$f(1, 0) = 0$**
    - **$f(1, 1) = 1$**

Boolean functions and expressions

x	y	z	g
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$$g : \{0, 1\} \times \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$$

$$g = ((x \Leftrightarrow y) \wedge z) \vee ((x \Leftrightarrow \neg y) \wedge \neg z)$$

Boolean expressions and functions

x	y	z	g
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

$$g = (x \wedge y \wedge \neg z) \vee (x \wedge \neg y \wedge z) \vee (\neg x \wedge y)$$

Boolean expressions and functions

x	y	z	g
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

$$g = (x \wedge y \wedge \neg z) \vee (x \wedge \neg y \wedge z) \vee (\neg x \wedge y)$$

$$g : \{0, 1\} \times \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$$

Three Representations

- ***Boolean functions***
- ***Truth tables***
- ***Propositional formulas***.
- Three ***equivalent*** representations.
- We will look at a a ***fourth one*** later in the course.

Boolean Functions and Connectives

For each ***n***, there are 2^{2^n} different ***n***-place Boolean functions.

There are 2^n different input points and **2** possible output values for each input point. 2^{2^n} is also the number of different ***n***-ary propositional connectives.

0-ary connectives

There are two 0-place Boolean functions: the constants 0 and 1. We can construct corresponding 0-ary connectives ⊥ and ⊤ with the meaning that $\upsilon \not\models \bot$ and $\upsilon \models \top$ regardless of the truth assignment.

Unary connectives

There are four 1-place Boolean functions, but these include the two constant functions mentioned above and the identity function. Thus the only additional connective of interest is negation: ¬.

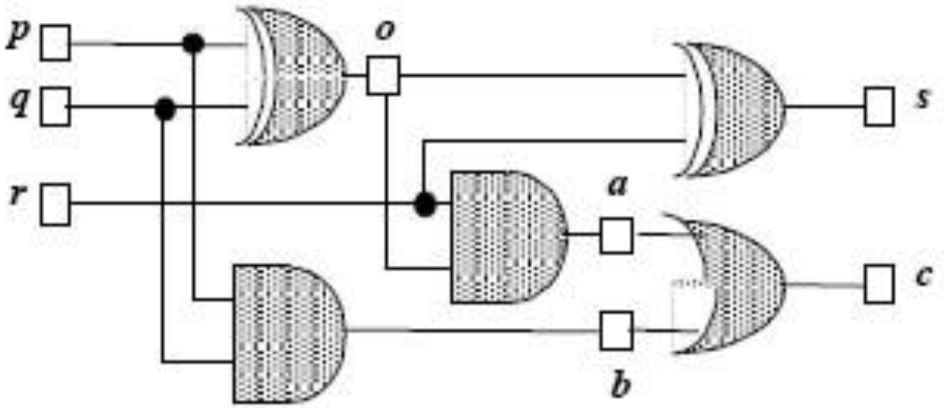
Binary connectives

There are sixteen 2-place Boolean functions. They are cataloged in the following table. Note that the first six correspond to 0-ary and unary connectives.

Binary Connectives

Symbol	Equivalent	Description
	$\bot$	constant $\mathbf{0}$
	$\top$	constant $\mathbf{1}$
	A	projection of first argument
	B	projection of second argument
	$\neg A$	negation of first argument
	$\neg B$	negation of second argument
$\wedge$	$A \wedge B$	and
$\vee$	$A \vee B$	or
$\rightarrow$	$A \rightarrow B$	conditional
$\leftrightarrow$	$A \leftrightarrow B$	bi-conditional
$\leftarrow$	$B \rightarrow A$	reverse conditional
$\oplus$	$(A \wedge \neg B) \vee (\neg A \wedge B)$	exclusive or
$\downarrow$	$\neg(A \vee B)$	nor (or Nicod stroke)
$\mid$	$\neg(A \wedge B)$	nand (or Sheffer stroke)
$<$	$\neg A \wedge B$	less than
$>$	$A \wedge \neg B$	greater than

Example: Curcuits and PL



$$o \Leftrightarrow (p \wedge \neg q) \vee (\neg p \wedge q)$$

$$a \Leftrightarrow r \wedge o$$

$$b \Leftrightarrow p \wedge q$$

$$s \Leftrightarrow (o \wedge \neg r) \vee (\neg o \wedge r)$$

$$c \Leftrightarrow a \vee b$$

Normal Forms: DNF

Normal forms in mathematics are canonical representations (i.e. all equivalent objects result in the same representation).

Definition: A formula α with $\mathbf{A_1, A_2, \ldots, A_n}$ propositional variables is in ***Disjunctive Normal Form*** (DNF) if it is has the structure:

$$(x^1_1 \wedge x^1_2 \wedge \ldots \wedge x^1_n) \vee \ldots \vee (x^m_1 \wedge x^m_2 \wedge \ldots \wedge x^m_n)$$

where $m \leq 2^n$ and for $i = 1,\ldots,n$ and $j = 1,\ldots,m$, $\mathbf{x^j_i}$ is either $\mathbf{A_i}$ or $\neg\mathbf{A_i}$ (both $\mathbf{A_i}$ and $\neg\mathbf{A_i}$ are called ***literals***).

E.g. $(\neg A_1 \wedge \neg A_2 \wedge A_3) \vee (A_1 \wedge \neg A_2 \wedge \neg A_3)$ is in DNF

$(\neg(A_1 \vee A_2) \wedge A_3)$ is not.

Each of the series of conjunctions picks out a row of the truth table where formula is true. DNF ORs together the ANDs for the true rows.

DNF

Consider the truth tables for the formulas **¬p ∧ ¬q ∧ r** and **¬p ∧ q ∧ ¬r**

	p	q	r	$\neg p \wedge \neg q \wedge r$	$\neg p \wedge q \wedge r$	$(p \wedge q \wedge \neg r)$
0	F	F	F	F	F	F
1	F	F	T	T	F	F
2	F	T	F	F	F	F
3	F	T	T	F	T	F
4	T	F	F	F	F	F
5	T	F	T	F	F	F
6	T	T	F	F	F	T
7	T	T	T	F	F	F

for **¬p ∧ ¬q ∧ r** only row 1 is true; for **¬p ∧ q ∧ ¬r** only row 3 is true; for **p ∧ q ∧ ¬ r** only row 6 is true.

DNF

Consider the truth tables for the formulas **¬p ∧ ¬q ∧ r** and **¬p ∧ q ∧ ¬r**

	p	q	r	$\neg p \wedge \neg q \wedge r$	$\neg p \wedge q \wedge r$	$(p \wedge q \wedge \neg r)$
0	F	F	F	F	F	F
1	F	F	T	T	F	F
2	F	T	F	F	F	F
3	F	T	T	F	T	F
4	T	F	F	F	F	F
5	T	F	T	F	F	F
6	T	T	F	F	F	T
7	T	T	T	F	F	F

(**¬p ∧ ¬q ∧ r**) ∨ (**¬p ∧ q ∧ ¬r**) ∨ (**p ∧ q ∧ ¬ r**) is true on rows 1, 3 and 6

DNF

Theorem: Every ***propositional formula*** that is not a contradiction is a ***logically equivalent*** to a DNF formula.

Corollary: For α, β not contradictions, $\alpha \leftrightarrow \beta$ if and only if α and β have the same DNF representation.

Proof: Two formulas are logically equivalent if and only if they have the same truth table (i.e. same true rows) and, thus, the same DNF.

DNF and Satisfiability

Theorem: Satisfiability of propositional ***formula*** in **DNF** can be checked on **Polynomial Time**.

Proof: Every formula in **DNF** is a disjunction of clauses. Therefore, the only possibility for the formula to be unsatisfiable is if every clause in isolation is unsatisfiable.

Since every clause is a conjunction of literals, for a clause of a DNF formula to be unsatisfiable, it must contain both some literal (**p**) and its complement ($\neg$**p**).

Therefore, every **DFN** ***formula*** is ***satisfiable*** unless every clause contains a pair of complementary literals.

And this can easily be checked in **Polynomial Time**.

CNF

Definition: A formula α with $A_1,A_2,\ldots,A_n$ propositional variables is in ***Conjunctive Normal Form*** (CNF) if it is has the structure:

$$(x^1_1 \vee x^1_2 \vee \ldots \vee x^1_n) \wedge \ldots \wedge (x^m_1 \vee x^m_2 \vee \ldots \vee x^m_n)$$

where $m \leq 2^n$, for $i = 1,\ldots,n$ and $j = 1,\ldots,m$, x^j_i is either A_i or $\neg A_i$.

E.g. $(\neg A_1 \vee \neg A_2 \vee A_3) \wedge (A_1 \vee \neg A_2 \vee \neg A_3)$ is in CNF

$(\neg(A_1 \vee A_2) \wedge A_3)$ is not.

Each of the series of disjunctions represents the negation of a row of the truth table where formula is false. CNF ANDs together the ORs corresponding to the negation of the flase rows.

One way to obtain the CNF form of a formula α is to write down the DNF for $\neg\alpha$, then negate it and apply De Morgan's lows as much as possilbe.

CNF and Validity

Using CNF to Check $\models \alpha$ (trivial)

$$\models (x^1_1 \vee x^1_2 \vee \ldots \vee x^1_n) \wedge \ldots \wedge (x^m_1 \vee x^m_2 \vee \ldots \vee x^m_n)$$

if and only if

$$\models (x^1_1 \vee x^1_2 \vee \ldots \vee x^1_n)$$
$$\models (x^2_1 \vee x^2_2 \vee \ldots \vee x^2_n)$$
$$\ldots$$
$$\models (x^m_1 \vee x^m_2 \vee \ldots \vee x^m_n)$$

If each x^j_i is a literal (e.g., p) or its negation (e.g., $\neg p$) then $\models (x^j_1 \vee x^j_2 \vee \ldots \vee x^j_n)$ iff there exists k and l s.t. $x^j_k = p$ and $x^j_l = \neg p$.

And this can easily be checked in **Polynomial Time**.

SAT complexity revisited

Question: So why are not ***validity*** and ***satisfiability*** polynomial problems?

Answer: Since converting a formula into an ***equivalent*** DNF or CNF can be exponential in size of the original formula.

Example:

CNF: $(\mathbf{A_1} \vee \mathbf{B_1}) \wedge (\mathbf{A_2} \vee \mathbf{B_2}) \wedge \ldots \wedge (\mathbf{A_n} \vee \mathbf{B_n})$

DNF: $(\mathbf{A_1} \wedge \mathbf{A_2} \wedge \ldots \wedge \mathbf{A_n}) \vee (\mathbf{A_1} \wedge \mathbf{A_2} \wedge \ldots \wedge \mathbf{B_n}) \vee$

$\vee (\mathbf{A_1} \wedge \mathbf{A_2} \wedge \ldots \mathbf{B_{n-1}} \wedge \mathbf{A_n}) \vee (\mathbf{A_1} \wedge \mathbf{A_2} \wedge \ldots \mathbf{B_{n-1}} \wedge \mathbf{B_n}) \vee \ldots \vee$

$\vee (\mathbf{B_1} \wedge \mathbf{B_2} \wedge \ldots \wedge \mathbf{B_n})$

In worlds, while the CNF formula contains n clauses, the DNF equivalent formula contains $\boldsymbol{2^n}$ clauses, where each clause contains, for each i, either $\mathbf{A_i}$ or $\mathbf{B_i}$.