

# A Formalization of the Exponential Blowup in the Transformations between CNF and DNF

Leon Raffael Schulz      Martin Desharnais-Schäfer  
Jan Johannsen

June 25, 2026

## Abstract

A well-known result about propositional logic is that transforming a formula into disjunctive or conjunctive normal form can lead to an exponential blowup of the formula size. We formalize this result in the form of two theorems. This formalization originated in Schulz's bachelor thesis [2] supervised by Johannsen with the help of Desharnais-Schäfer; the thesis formalizes the theorems as found in Johannsen's lecture notes of SAT solving [1]. The formalization was later refactored and expanded by Desharnais-Schäfer in collaboration with Schulz.

## Contents

<b>1</b>	<b>Move to HOL</b>	<b>2</b>
<b>2</b>	<b>Move to Propositional_Proof_Systems</b>	<b>2</b>
<b>3</b>	<b>Functions, Predicates, and Datatypes</b>	<b>2</b>
3.1	Formula Equivalence . . . . .	2
3.2	Conjunctive Normal Form . . . . .	3
3.3	Disjunctive Normal Form . . . . .	3
3.4	Big Conjunction . . . . .	4
3.5	Big Disjunction . . . . .	4
3.6	Formula Size . . . . .	5
3.7	Fn function . . . . .	7
3.8	Dual Function . . . . .	8
3.9	Formula Contains Atom . . . . .	9
<b>4</b>	<b>CNF to DNF</b>	<b>10</b>

## 5 DNF to CNF

**theory** *CNF-DNF-Exp-Blowup*

**imports**

*Main*

*Propositional-Proof-Systems.Formulas*

*Propositional-Proof-Systems.Sema*

*Propositional-Proof-Systems.CNF-Formulas*

**begin**

### 1 Move to HOL

**lemma** *card-Domain-le*:

**assumes** *finite A*

**shows**  $\text{card } (\text{Domain } A) \leq \text{card } A$

*<proof>*

**lemma** *card-le-card-if-mem-imp-ex-mem*:

**fixes**  $f :: 'a \Rightarrow 'b \Rightarrow 'c$  **and**  $\mathcal{X} :: 'a \text{ set}$  **and**  $\mathcal{Y} :: 'c \text{ set}$

**defines**  $XY \equiv \{(x, y). x \in \mathcal{X} \wedge f x y \in \mathcal{Y}\}$

**assumes** *finite X* **and** *finite Y* **and**

*f-inj*: *inj-on*  $(\lambda(x, y). f x y)$   $XY$  **and**

*ex-in-Y*:  $\bigwedge x. x \in \mathcal{X} \Longrightarrow \exists y. f x y \in \mathcal{Y}$

**shows**  $\text{card } \mathcal{X} \leq \text{card } \mathcal{Y}$

*<proof>*

### 2 Move to Propositional\_Proof\_Systems

**lemma** *is-disj-if-is-lit-plus*:  $\text{is-lit-plus } \varphi \Longrightarrow \text{is-disj } \varphi$

*<proof>*

**lemma** *disj-is-cnf*:  $\text{is-disj } \varphi \Longrightarrow \text{is-cnf } \varphi$

*<proof>*

**lemma** *cnf-in-nnf*:  $\text{is-cnf } \varphi \Longrightarrow \text{is-nnf } \varphi$

*<proof>*

## 3 Functions, Predicates, and Datatypes

### 3.1 Formula Equivalence

**definition** *equiv* :: *'a formula*  $\Rightarrow$  *'a formula*  $\Rightarrow$  *bool* **where**

$\text{equiv } \varphi \psi \longleftrightarrow (\forall \alpha. (\alpha \models \varphi) \longleftrightarrow (\alpha \models \psi))$

**lemma** *equiv-reflexive*:  $\bigwedge \varphi. \text{equiv } \varphi \varphi$

*<proof>*

**lemma** *equiv-symmetric[sym]*:  $\bigwedge \varphi \psi. \text{equiv } \varphi \psi \Longrightarrow \text{equiv } \psi \varphi$

*<proof>*

**lemma** *equiv-transitive[trans]*:  $\bigwedge \xi \varphi \psi. \text{equiv } \xi \varphi \implies \text{equiv } \varphi \psi \implies \text{equiv } \xi \psi$   
*<proof>*

**lemma** *equiv-Not-left-Not-right[simp]*:  $\bigwedge \varphi \psi. \text{equiv } (\text{Not } \varphi) (\text{Not } \psi) \longleftrightarrow \text{equiv } \varphi \psi$   
*<proof>*

**lemma** *equiv-Not-Not-left[simp]*:  $\bigwedge \varphi \psi. \text{equiv } (\text{Not } (\text{Not } \varphi)) \psi \longleftrightarrow \text{equiv } \varphi \psi$   
*<proof>*

**lemma** *equiv-Not-Not-right[simp]*:  $\bigwedge \varphi \psi. \text{equiv } \varphi (\text{Not } (\text{Not } \psi)) \longleftrightarrow \text{equiv } \varphi \psi$   
*<proof>*

### 3.2 Conjunctive Normal Form

**fun** *unconj* :: 'a formula  $\Rightarrow$  'a formula list **where**  
*unconj* (And  $\varphi \psi$ ) = *unconj*  $\varphi$  @ *unconj*  $\psi$  |  
*unconj*  $\varphi$  = [ $\varphi$ ]

**lemma** *unconj-neq-Nil[simp]*: *unconj*  $\varphi \neq []$   
*<proof>*

**fun** *count-And* :: 'a formula  $\Rightarrow$  nat **where**  
*count-And* (And  $\varphi \psi$ ) = *count-And*  $\varphi$  + *count-And*  $\psi$  + 1 |  
*count-And* - = 0

**lemma** *length-unconj*: *length* (*unconj*  $\varphi$ ) = *count-And*  $\varphi$  + 1  
*<proof>*

**lemma** *ball-unconj-is-disj*:  
**fixes**  $\varphi$  :: 'a formula  
**assumes** *is-cnf*  $\varphi$   
**shows**  $\bigwedge C. C \in \text{set } (\text{unconj } \varphi) \implies \text{is-disj } C$   
*<proof>*

### 3.3 Disjunctive Normal Form

**fun** *is-conj* :: 'a formula  $\Rightarrow$  bool **where**  
*is-conj* (And  $\varphi \psi$ )  $\longleftrightarrow$  (*is-lit-plus*  $\varphi$   $\wedge$  *is-conj*  $\psi$ ) |  
*is-conj*  $\varphi \longleftrightarrow$  *is-lit-plus*  $\varphi$

**fun** *is-dnf* :: 'a formula  $\Rightarrow$  bool **where**  
*is-dnf* (Or  $\varphi \psi$ )  $\longleftrightarrow$  (*is-dnf*  $\varphi$   $\wedge$  *is-dnf*  $\psi$ ) |  
*is-dnf*  $\varphi \longleftrightarrow$  *is-conj*  $\varphi$

**lemma** *conj-is-dnf*: *is-conj*  $\varphi \implies$  *is-dnf*  $\varphi$   
*<proof>*

**fun** *undisj* :: 'a formula  $\Rightarrow$  'a formula list **where**  
*undisj* (Or  $\varphi$   $\psi$ ) = *undisj*  $\varphi$  @ *undisj*  $\psi$  |  
*undisj*  $\varphi$  = [ $\varphi$ ]

**lemma** *undisj-neg-Nil[simp]*: *undisj*  $\varphi \neq []$   
 <proof>

**lemma** *ball-undisj-is-conj*:  
**fixes**  $\varphi$  :: 'a formula  
**assumes** *is-dnf*  $\varphi$   
**shows**  $\bigwedge T. T \in \text{set } (\text{undisj } \varphi) \Longrightarrow \text{is-conj } T$   
 <proof>

**fun** *count-Or* :: 'a formula  $\Rightarrow$  nat **where**  
*count-Or* (Or  $\varphi$   $\psi$ ) = *count-Or*  $\varphi$  + *count-Or*  $\psi$  + 1 |  
*count-Or* - = 0

**lemma** *length-undisj*: *length* (*undisj*  $\varphi$ ) = *count-Or*  $\varphi$  + 1  
 <proof>

### 3.4 Big Conjunction

**fun** *BigAnd'* :: 'a formula list  $\Rightarrow$  'a formula **where**  
*BigAnd'* [] = ( $\neg \perp$ ) |  
*BigAnd'* [ $F$ ] =  $F$  |  
*BigAnd'* ( $F \# Fs$ ) =  $F \wedge \text{BigAnd}' Fs$

**lemma** *atoms-BigAnd'[simp]*: *atoms* (*BigAnd'*  $Fs$ ) =  $\bigcup (\text{atoms } ' \text{set } Fs)$   
 <proof>

**lemma** *BigAnd'-semantics[simp]*:  $A \models \text{BigAnd}' Ts \longleftrightarrow (\forall f \in \text{set } Ts. A \models f)$   
 <proof>

**lemma** *is-cnf-BigAnd'*:  $(\forall C \in \text{set } Cs. \text{is-disj } C \wedge \neg(\forall \alpha. \alpha \models C)) \Longrightarrow \text{is-cnf}$   
 (*BigAnd'*  $Cs$ )  
 <proof>

**lemma** *equiv-BigAnd'-append*: *equiv* (*BigAnd'* ( $xs$  @  $ys$ )) (*And* (*BigAnd'*  $xs$ ) (*BigAnd'*  $ys$ ))  
 <proof>

### 3.5 Big Disjunction

**fun** *BigOr'* :: 'a formula list  $\Rightarrow$  'a formula **where**  
*BigOr'* Nil =  $\perp$  |  
*BigOr'* [ $F$ ] =  $F$  |  
*BigOr'* ( $F \# Fs$ ) =  $F \vee \text{BigOr}' Fs$

**lemma** *atoms-BigOr'[simp]*: *atoms* (*BigOr'*  $Fs$ ) =  $\bigcup (\text{atoms } ' \text{set } Fs)$   
 <proof>

**lemma** *BigOr'-semantics[simp]*:  $A \models \text{BigOr}' Ts \iff (\exists f \in \text{set } Ts. A \models f)$   
 ⟨proof⟩

**lemma** *is-dnf-BigOr'*:  $(\forall T \in \text{set } Ts. \text{is-conj } T \wedge (\exists \alpha. \alpha \models T)) \implies \text{is-dnf}$   
 ( $\text{BigOr}' Ts$ )  
 ⟨proof⟩

**lemma** *equiv-BigOr'-append*:  $\text{equiv } (\text{BigOr}' (xs @ ys)) (\text{Or } (\text{BigOr}' xs) (\text{BigOr}'$   
 $ys))$   
 ⟨proof⟩

**lemma** *equiv-BigOr'-undisj-if-dnf*:  
**fixes**  $\varphi :: 'a \text{ formula}$   
**shows**  $\text{equiv } (\text{BigOr}' (\text{undisj } \varphi)) \varphi$   
 ⟨proof⟩

**lemma** *equiv-BigAnd'-unconj-if-cnf*:  
**fixes**  $\varphi :: 'a \text{ formula}$   
**shows**  $\text{equiv } (\text{BigAnd}' (\text{unconj } \varphi)) \varphi$   
 ⟨proof⟩

### 3.6 Formula Size

Similar to *size*, but ignores  $\neg$  when calculating the size.

**fun** *sizef* ::  $'a \text{ formula} \Rightarrow \text{nat}$  **where**  
*sizef* Bot = 1 |  
*sizef* (Atom a) = 1 |  
*sizef* (Not  $\varphi$ ) = *sizef*  $\varphi$  |  
*sizef* (And  $\varphi \psi$ ) = *sizef*  $\varphi$  + *sizef*  $\psi$  + 1 |  
*sizef* (Or  $\varphi \psi$ ) = *sizef*  $\varphi$  + *sizef*  $\psi$  + 1 |  
*sizef* (Imp  $\varphi \psi$ ) = *sizef*  $\varphi$  + *sizef*  $\psi$  + 1

**lemma** *Suc-0-le-sizef[simp]*:  $\text{Suc } 0 \leq \text{sizef } \varphi$   
 ⟨proof⟩

**lemma** *Suc-0-le-size[simp]*:  
**fixes**  $\varphi :: 'a \text{ formula}$   
**shows**  $\text{Suc } 0 \leq \text{size } \varphi$   
 ⟨proof⟩

**lemma** *sizef-le-size*:  $\text{sizef } \varphi \leq \text{size } \varphi$   
 ⟨proof⟩

**lemma** *card-atoms-le-sizef*:  $\text{card } (\text{atoms } \varphi) \leq \text{sizef } \varphi$   
 ⟨proof⟩

**lemma** *card-atoms-le-size*:  $\text{card } (\text{atoms } \varphi) \leq \text{size } \varphi$   
 ⟨proof⟩

**lemma** *aux-exp-sizef*:  $\text{length } Ts = n \implies \forall T \in \text{set } Ts. \text{sizef } T \geq m \implies \text{sizef } (\text{BigOr}' Ts) \geq n * m$   
 ⟨proof⟩

**lemma** *aux-exp-size*:  $\text{length } Ts = n \implies \forall T \in \text{set } Ts. \text{size } T \geq m \implies \text{size } (\text{BigOr}' Ts) \geq n * m$   
 ⟨proof⟩

**lemma** *exp-sizef*:  
 assumes  $n > 0$  and  $\text{length } Ts \geq 2^n$  and  $\forall T \in \text{set } Ts. \text{sizef } T \geq m$   
 shows  $\text{sizef } (\text{BigOr}' Ts) \geq 2^n * m$   
 ⟨proof⟩

**lemma** *exp-size*:  
 assumes  $n > 0$  and  $\text{length } Ts \geq 2^n$  and  $\forall T \in \text{set } Ts. \text{size } T \geq m$   
 shows  $\text{size } (\text{BigOr}' Ts) \geq 2^n * m$   
 ⟨proof⟩

**lemma** *sizef-BigOr'*:  $xs \neq [] \implies \text{sizef } (\text{BigOr}' xs) + 1 = \text{sum-list } (\text{map sizef } xs) + \text{length } xs$   
 ⟨proof⟩

**lemma** *size-BigOr'*:  $xs \neq [] \implies \text{size } (\text{BigOr}' xs) + 1 = \text{sum-list } (\text{map size } xs) + \text{length } xs$   
 ⟨proof⟩

**lemma** *sizef-BigAnd'*:  $xs \neq [] \implies \text{sizef } (\text{BigAnd}' xs) + 1 = \text{sum-list } (\text{map sizef } xs) + \text{length } xs$   
 ⟨proof⟩

**lemma** *size-BigAnd'*:  $xs \neq [] \implies \text{size } (\text{BigAnd}' xs) + 1 = \text{sum-list } (\text{map size } xs) + \text{length } xs$   
 ⟨proof⟩

**lemma** *sizef-conv-sum-list-undisj*:  $\text{sizef } \varphi = \text{sum-list } (\text{map sizef } (\text{undisj } \varphi)) + \text{count-Or } \varphi$   
 ⟨proof⟩

**lemma** *size-conv-sum-list-undisj*:  $\text{size } \varphi = \text{sum-list } (\text{map size } (\text{undisj } \varphi)) + \text{count-Or } \varphi$   
 ⟨proof⟩

**lemma** *sizef-conv-sum-list-unconj*:  $\text{sizef } \varphi = \text{sum-list } (\text{map sizef } (\text{unconj } \varphi)) + \text{count-And } \varphi$   
 ⟨proof⟩

**lemma** *size-conv-sum-list-unconj*:  $\text{size } \varphi = \text{sum-list } (\text{map size } (\text{unconj } \varphi)) + \text{count-And } \varphi$

*<proof>*

**lemma** *sizef-BigOr'-undisj*:  
 **fixes**  $\varphi :: 'a$  formula  
 **shows**  $sizef (BigOr' (undisj \varphi)) = sizef \varphi$   
*<proof>*

**lemma** *size-BigOr'-undisj*:  
 **fixes**  $\varphi :: 'a$  formula  
 **shows**  $size (BigOr' (undisj \varphi)) = size \varphi$   
*<proof>*

**lemma** *sizef-BigAnd'-unconj*:  
 **fixes**  $\varphi :: 'a$  formula  
 **shows**  $sizef (BigAnd' (unconj \varphi)) = sizef \varphi$   
*<proof>*

**lemma** *size-BigAnd'-unconj*:  
 **fixes**  $\varphi :: 'a$  formula  
 **shows**  $size (BigAnd' (unconj \varphi)) = size \varphi$   
*<proof>*

**lemma** *sizef-BigOr'-filter-le*:  $sizef (BigOr' (filter P xs)) \leq sizef (BigOr' xs)$   
*<proof>*

**lemma** *size-BigOr'-filter-le-if*:  
 **assumes**  $\exists x \in set\ xs. P\ x$   
 **shows**  $size (BigOr' (filter P xs)) \leq size (BigOr' xs)$   
*<proof>*

**lemma** *sizef-BigAnd'-filter-le*:  $sizef (BigAnd' (filter P xs)) \leq sizef (BigAnd' xs)$   
*<proof>*

### 3.7 Fn function

**datatype** *var* = *Var* nat bool

**lemma** *inj-on-Var[simp]*: *inj-on* ( $\lambda(x, y). Var\ x\ y$ ) **for** *A*  
*<proof>*

**fun** *Fn* :: nat  $\Rightarrow$  var formula **where**  
 *Fn* 0 = ( $\neg \perp$ )|  
 *Fn* (Suc n) =  
 And  
 (And  
 (Or  
 (Atom (Var (Suc n) False))  
 (Atom (Var (Suc n) True)))  
 (Or

$(\text{Not } (\text{Atom } (\text{Var } (\text{Suc } n) \text{ False})))$   
 $(\text{Not } (\text{Atom } (\text{Var } (\text{Suc } n) \text{ True}))))$   
 $(\text{Fn } n)$

**lemma** *sizef-Fn*:  $\text{sizef } (\text{Fn } n) = 8 * n + 1$   
 $\langle \text{proof} \rangle$

**lemma** *size-Fn*:  $\text{size } (\text{Fn } n) = 10 * n + 2$   
 $\langle \text{proof} \rangle$

**lemma** *is-cnf-Fn*:  $\text{is-cnf } (\text{Fn } n)$   
 $\langle \text{proof} \rangle$

**lemma** *is-nnf-Fn*:  $\text{is-nnf } (\text{Fn } n)$   
 $\langle \text{proof} \rangle$

**lemma** *Fn-sat*:  $\exists \alpha. \alpha \models \text{Fn } n$   
 $\langle \text{proof} \rangle$

**lemma** *semantics-Fn-iff*:  $\alpha \models \text{Fn } n \longleftrightarrow (\forall i \in \{1..n\}. \alpha (\text{Var } i \text{ False}) \neq \alpha (\text{Var } i \text{ True}))$   
 $\langle \text{proof} \rangle$

### 3.8 Dual Function

**primrec** *dual* :: 'a formula  $\Rightarrow$  'a formula **where**

$\text{dual Bot} = \text{Not Bot} \mid$   
 $\text{dual } (\text{Atom } v) = \text{Not } (\text{Atom } v) \mid$   
 $\text{dual } (\text{Not } v) = v \mid$   
 $\text{dual } (\text{And } \varphi \psi) = \text{Or } (\text{dual } \varphi) (\text{dual } \psi) \mid$   
 $\text{dual } (\text{Or } \varphi \psi) = \text{And } (\text{dual } \varphi) (\text{dual } \psi) \mid$   
 $\text{dual } (\text{Imp } \varphi \psi) = \text{And } \varphi (\text{dual } \psi)$

**lemma** *sizef-dual-Fn*:  $\text{sizef } (\text{dual } (\text{Fn } n)) = 8 * n + 1$   
 $\langle \text{proof} \rangle$

**lemma** *size-dual-Fn*:  $\text{size } (\text{dual } (\text{Fn } n)) = 10 * n + 1$   
 $\langle \text{proof} \rangle$

**lemma** *is-dnf-dual-Fn*:  $\text{is-dnf } (\text{dual } (\text{Fn } n))$   
 $\langle \text{proof} \rangle$

**lemma** *sizef-dual*:  $\text{sizef } (\text{dual } \varphi) = \text{sizef } \varphi$   
 $\langle \text{proof} \rangle$

**lemma** *size-dual-le*:  $\text{size } (\text{dual } \varphi) \leq 2 * \text{size } \varphi$   
 $\langle \text{proof} \rangle$

**lemma** *equiv-dual*:  $\text{equiv } (\text{dual } \varphi) (\text{Not } \varphi)$

*<proof>*

**lemma** *is-disj-dual-if-is-conj*:  $is-conj\ \varphi \implies is-disj\ (dual\ \varphi)$   
*<proof>*

**lemma** *is-conj-dual-if-is-disj*:  $is-disj\ \varphi \implies is-conj\ (dual\ \varphi)$   
*<proof>*

**lemma** *is-dnf-dual-if-is-cnf*:  $is-cnf\ \varphi \implies is-dnf\ (dual\ \varphi)$   
*<proof>*

**lemma** *is-cnf-dual-if-is-dnf*:  $is-dnf\ \varphi \implies is-cnf\ (dual\ \varphi)$   
*<proof>*

**lemma** *dual-disj-not-taut-impl-sat*:  $is-disj\ \varphi \implies \exists \alpha. \neg \alpha \models \varphi \implies \exists \alpha. \alpha \models dual\ \varphi$   
*<proof>*

**lemma** *dual-conj-of-disjs-is-disj-of-conjs*:

**fixes**  $Cs$

**assumes**  $\forall C \in set\ Cs. is-disj\ C \wedge (\exists \alpha. \neg(\alpha \models C))$

**defines**  $Ts \equiv map\ dual\ Cs$

**shows**  $dual\ (BigAnd'\ Cs) = BigOr'\ Ts \ \forall T \in set\ Ts. is-conj\ T \wedge (\exists \alpha. \alpha \models T)$

*<proof>*

### 3.9 Formula Contains Atom

Should only be applied to a formula for which *is-nnf* holds.

**fun** *cont-pos* :: 'a formula  $\Rightarrow$  'a  $\Rightarrow$  bool **where**

*cont-pos* Bot  $l = False$  |

*cont-pos* (Atom  $v$ )  $l = (v = l)$  |

*cont-pos* (Not (Atom  $v$ ))  $l = False$  |

*cont-pos* (Not  $\varphi$ )  $l = False$  |

*cont-pos* (And  $\varphi\ \psi$ )  $l = (cont-pos\ \varphi\ l \vee cont-pos\ \psi\ l)$  |

*cont-pos* (Or  $\varphi\ \psi$ )  $l = (cont-pos\ \varphi\ l \vee cont-pos\ \psi\ l)$  |

*cont-pos* (Imp  $\varphi\ \psi$ )  $l = False$

Should only be applied to a formula for which *is-nnf* holds.

**fun** *cont-neg* :: 'a formula  $\Rightarrow$  'a  $\Rightarrow$  bool **where**

*cont-neg* Bot  $l = False$  |

*cont-neg* (Atom  $v$ )  $l = False$  |

*cont-neg* (Not (Atom  $v$ ))  $l = (v = l)$  |

*cont-neg* (Not  $\varphi$ )  $l = False$  |

*cont-neg* (And  $\varphi\ \psi$ )  $l = (cont-neg\ \varphi\ l \vee cont-neg\ \psi\ l)$  |

*cont-neg* (Or  $\varphi\ \psi$ )  $l = (cont-neg\ \varphi\ l \vee cont-neg\ \psi\ l)$  |

*cont-neg* (Imp  $\varphi\ \psi$ )  $l = False$

Should only be applied to a formula for which *is-nnf* holds.

**fun** *cont* :: 'a formula  $\Rightarrow$  'a  $\Rightarrow$  bool **where**

$cont\ Bot\ l = False$  |  
 $cont\ (Atom\ v)\ l = (v = l)$  |  
 $cont\ (Not\ (Atom\ v))\ l = (v = l)$  |  
 $cont\ (Not\ \varphi)\ l = False$  |  
 $cont\ (And\ \varphi\ \psi)\ l = (cont\ \varphi\ l \vee cont\ \psi\ l)$  |  
 $cont\ (Or\ \varphi\ \psi)\ l = (cont\ \varphi\ l \vee cont\ \psi\ l)$  |  
 $cont\ (Imp\ \varphi\ \psi)\ l = False$

**lemma** *impl-not-cont-pos*:  $\neg cont\ pos\ \varphi\ v \implies cont\ neg\ \varphi\ v \vee \neg (cont\ \varphi\ v)$   
 <proof>

**lemma** *impl-not-cont*:  $\neg cont\ \varphi\ v \implies \neg cont\ pos\ \varphi\ v \wedge \neg cont\ neg\ \varphi\ v$   
 <proof>

**lemma** *mem-atoms-if-cont-pos*:

**assumes** *cont-pos*  $T\ v$

**shows**  $v \in atoms\ T$

<proof>

**lemma**

**assumes** *is-conj*  $\varphi$

**shows**

*not-sat-conj-neg-true*:  $\exists v. cont\ neg\ \varphi\ v \wedge \alpha\ v \implies \neg(\alpha \models \varphi)$  **and**

*not-sat-conj-pos-false*:  $\exists v. cont\ pos\ \varphi\ v \wedge \neg(\alpha\ v) \implies \neg(\alpha \models \varphi)$

<proof>

**lemma** *sat-conj-val-cont-ident*:

**assumes**  $Val1 \models \varphi$  **and**  $\forall v \in \{v. cont\ \varphi\ v\}. Val1\ v = Val2\ v$  **and** *is-conj*  $\varphi$

**shows**  $Val2 \models \varphi$

<proof>

## 4 CNF to DNF

**proposition** *exp-blowup-from-Fn-to-BigOr'*:

**fixes**  $n :: nat$  **and**  $Ts :: var\ formula\ list$

**defines**  $\varphi \equiv Fn\ n$  **and**  $\psi \equiv BigOr'\ Ts$

**assumes**

*n-greater-0*:  $n > 0$  **and**

*Ts-spec*:  $(\forall T \in set\ Ts. is-conj\ T \wedge (\exists \alpha. \alpha \models T))$  **and**

*equiv*  $\varphi\ \psi$

**shows**  $size\ \psi \geq n * 2^{\wedge} n$

<proof>

**lemma** *ex-equiv-disj-list-if-is-dnf*:

**fixes**  $\varphi :: 'a\ formula$

**assumes** *dnf*: *is-dnf*  $\varphi$  **and** *sat*:  $\exists \alpha. \alpha \models \varphi$

**shows**  $\exists (Ts :: 'a\ formula\ list). equiv\ \varphi\ (BigOr'\ Ts) \wedge$

$size\ (BigOr'\ Ts) \leq size\ \varphi \wedge$

$(\forall T \in set\ Ts. is-conj\ T) \wedge$

( $\forall T \in \text{set } Ts. \exists \alpha. \alpha \models T$ )  
<proof>

**theorem** *exp-blowup-from-CNF-to-DNF*:

**fixes**  $n :: \text{nat}$   
**shows**  $\exists(\varphi :: \text{var formula}).$   
 $\text{is-cnf } \varphi \wedge$   
 $\text{size } \varphi = 10 * n + 2 \wedge$   
 $(\forall(\psi :: \text{var formula}). \text{equiv } \varphi \psi \longrightarrow \text{is-dnf } \psi \longrightarrow \text{size } \psi \geq n * 2 ^ n)$   
<proof>

## 5 DNF to CNF

**proposition** *exp-blowup-from-dual-Fn-to-BigAnd'*:

**fixes**  $n :: \text{nat}$  **and**  $Cs :: \text{var formula list}$   
**defines**  $\varphi \equiv \text{dual } (Fn\ n)$  **and**  $\psi \equiv \text{BigAnd' } Cs$   
**assumes**  
 $n\text{-greater-0}: n > 0$  **and**  
 $Cs\text{-spec}: (\forall C \in \text{set } Cs. \text{is-disj } C \wedge \neg(\forall \alpha. \alpha \models C))$  **and**  
 $\text{equiv } \varphi \psi$   
**shows**  $\text{sizef } \psi \geq n * 2 ^ n$   
<proof>

**lemma** *ex-equiv-conj-list-if-is-cnf*:

**fixes**  $\varphi :: 'a \text{ formula}$   
**assumes**  $\text{cnf}: \text{is-cnf } \varphi$   
**shows**  $\exists(Cs :: 'a \text{ formula list}). \text{equiv } \varphi (\text{BigAnd' } Cs) \wedge$   
 $\text{sizef } (\text{BigAnd' } Cs) \leq \text{sizef } \varphi \wedge$   
 $(\forall C \in \text{set } Cs. \text{is-disj } C) \wedge$   
 $(\forall C \in \text{set } Cs. \neg \models C)$   
<proof>

**theorem** *exp-blowup-from-DNF-to-CNF*:

**fixes**  $n :: \text{nat}$   
**shows**  $\exists(\varphi :: \text{var formula}).$   
 $\text{is-dnf } \varphi \wedge$   
 $\text{size } \varphi = 10 * n + 1 \wedge$   
 $(\forall(\psi :: \text{var formula}). \text{equiv } \varphi \psi \longrightarrow \text{is-cnf } \psi \longrightarrow \text{size } \psi \geq n * 2 ^ n)$   
<proof>

**end**

## References

- [1] Jan Johannsen. SAT Solving: Skript zur Vorlesung im WS 2023/24, 2023.

- [2] Leon Raffael Schulz. Formalisierung einer unteren Schranke an die Formelgröße in der Aussagenlogik mit Isabelle, 2026.