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

Leon Raffael Schulz Martin Desharnais-Schäfer
Jan Johannsen

June 16, 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 Schulzs 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

**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** *card* (*Domain A*) ≤ *card A*
⟨*proof*⟩

**lemma** *card-le-card-if-mem-imp-ex-mem*:
**fixes** $f :: 'a \Rightarrow 'b \Rightarrow 'c$ **and** $\mathcal{X} :: 'a\ set$ **and** $\mathcal{Y} :: 'c\ set$
**defines** $XY \equiv \{(x,\ y).\ x \in \mathcal{X} \wedge f\ x\ y \in \mathcal{Y}\}$
**assumes** *finite* $\mathcal{X}$ **and** *finite* $\mathcal{Y}$ **and**
*f-inj*: *inj-on* $(\lambda(x,\ y).\ f\ x\ y)\ XY$ **and**
*ex-in-*$\mathcal{Y}$: $\bigwedge x.\ x \in \mathcal{X} \Longrightarrow \exists\, y.\ f\ x\ y \in \mathcal{Y}$
**shows** *card* $\mathcal{X} \leq$ *card* $\mathcal{Y}$
⟨*proof*⟩

# 2 Move to Propositional_Proof_Systems

**lemma** *is-disj-if-is-lit-plus*: *is-lit-plus* $\varphi \Longrightarrow$ *is-disj* $\varphi$
⟨*proof*⟩

**lemma** *disj-is-cnf*: *is-disj* $\varphi \Longrightarrow$ *is-cnf* $\varphi$
⟨*proof*⟩

**lemma** *cnf-in-nnf*: *is-cnf* $\varphi \Longrightarrow$ *is-nnf* $\varphi$
⟨*proof*⟩

# 3 Functions, Predicates, and Datatypes

## 3.1 Formula Equivalence

**definition** *equiv* :: *'a formula* ⇒ *'a formula* ⇒ *bool* **where**
*equiv* $\varphi\ \psi \longleftrightarrow (\forall\, \alpha.\ (\alpha \models \varphi) \longleftrightarrow (\alpha \models \psi))$

**lemma** *equiv-reflexive*: $\bigwedge \varphi.$ *equiv* $\varphi\ \varphi$
⟨*proof*⟩

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

⟨*proof*⟩

**lemma** *equiv-transitive*[*trans*]: $\bigwedge\xi\ \varphi\ \psi.\ equiv\ \xi\ \varphi \Longrightarrow equiv\ \varphi\ \psi \Longrightarrow equiv\ \xi\ \psi$
⟨*proof*⟩

**lemma** *equiv-Not-left-Not-right*[*simp*]: $\bigwedge\varphi\ \psi.\ equiv\ (Not\ \varphi)\ (Not\ \psi) \longleftrightarrow equiv\ \varphi\ \psi$
⟨*proof*⟩

**lemma** *equiv-Not-Not-left*[*simp*]: $\bigwedge\varphi\ \psi.\ equiv\ (Not\ (Not\ \varphi))\ \psi \longleftrightarrow equiv\ \varphi\ \psi$
⟨*proof*⟩

**lemma** *equiv-Not-Not-right*[*simp*]: $\bigwedge\varphi\ \psi.\ equiv\ \varphi\ (Not\ (Not\ \psi)) \longleftrightarrow equiv\ \varphi\ \psi$
⟨*proof*⟩

## 3.2 Conjunctive Normal Form

**fun** *unconj* :: *'a formula* ⇒ *'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* ⇒ *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** $\bigwedge C.\ C \in set\ (unconj\ \varphi) \Longrightarrow is\text{-}disj\ C$
⟨*proof*⟩

## 3.3 Disjunctive Normal Form

**fun** *is-conj* :: *'a formula* ⇒ *bool* **where**
*is-conj* (*And* $\varphi\ \psi$) ⟷ (*is-lit-plus* $\varphi$ ∧ *is-conj* $\psi$) |
*is-conj* $\varphi$ ⟷ *is-lit-plus* $\varphi$

**fun** *is-dnf* :: *'a formula* ⇒ *bool* **where**
*is-dnf* (*Or* $\varphi\ \psi$) ⟷ (*is-dnf* $\varphi$ ∧ *is-dnf* $\psi$) |
*is-dnf* $\varphi$ ⟷ *is-conj* $\varphi$

**lemma** *conj-is-dnf*: *is-conj* $\varphi \Longrightarrow$ *is-dnf* $\varphi$
⟨*proof*⟩

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

**lemma** *undisj-neq-Nil*[*simp*]: *undisj* $\varphi \neq$ []
⟨*proof*⟩

**lemma** *ball-undisj-is-conj*:
**fixes** $\varphi$ :: *'a formula*
**assumes** *is-dnf* $\varphi$
**shows** $\bigwedge T$. $T \in$ *set* (*undisj* $\varphi$) ⟹ *is-conj* $T$
⟨*proof*⟩

**fun** *count-Or* :: *'a formula* ⇒ *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* ⇒ *'a formula* **where**
*BigAnd'* [] = (¬⊥) |
*BigAnd'* [*F*] = *F* |
*BigAnd'* (*F* # *Fs*) = *F* ∧ *BigAnd' Fs*

**lemma** *atoms-BigAnd'*[*simp*]: *atoms* (*BigAnd' Fs*) = ⋃(*atoms* ‘ *set Fs*)
⟨*proof*⟩

**lemma** *BigAnd'-semantics*[*simp*]: $A \models$ *BigAnd' Ts* ⟷ (∀ $f \in$ *set Ts*. $A \models f$)
⟨*proof*⟩

**lemma** *is-cnf-BigAnd'*: (∀ $C \in$ *set Cs*. *is-disj* $C$ ∧ ¬(∀ $\alpha$. $\alpha \models C$)) ⟹ *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* ⇒ *'a formula* **where**
*BigOr' Nil* = ⊥ |
*BigOr'* [*F*] = *F* |
*BigOr'* (*F*#*Fs*) = *F* ∨ *BigOr' Fs*

**lemma** *atoms-BigOr'*[*simp*]: *atoms* (*BigOr' Fs*) = ⋃(*atoms* ‘ *set Fs*)
⟨*proof*⟩

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

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

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

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

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

## 3.6 Formula Size

Similar to *size*, but ignores ¬ when calculating the size.

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

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

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

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

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

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

**lemma** *aux-exp-sizef*: *length Ts* = *n* ⟹ ∀ *T* ∈ *set Ts*. *sizef T* ≥ *m* ⟹ *sizef* (*BigOr′ Ts*) ≥ *n* ∗ *m*
⟨*proof*⟩

**lemma** *aux-exp-size*: *length Ts* = *n* ⟹ ∀ *T* ∈ *set Ts*. *size T* ≥ *m* ⟹ *size* (*BigOr′ Ts*) ≥ *n* ∗ *m*
⟨*proof*⟩

**lemma** *exp-sizef*:
**assumes** *n* > *0* **and** *length Ts* ≥ *2^n* **and** ∀ *T* ∈ *set Ts*. *sizef T* ≥ *m*
**shows** *sizef* (*BigOr′ Ts*) ≥ *2^n* ∗ *m*
⟨*proof*⟩

**lemma** *exp-size*:
**assumes** *n* > *0* **and** *length Ts* ≥ *2^n* **and** ∀ *T* ∈ *set Ts*. *size T* ≥ *m*
**shows** *size* (*BigOr′ Ts*) ≥ *2^n* ∗ *m*
⟨*proof*⟩

**lemma** *sizef-BigOr′*: *xs* ≠ [] ⟹ *sizef* (*BigOr′ xs*) + *1* = *sum-list* (*map sizef xs*) + *length xs*
⟨*proof*⟩

**lemma** *size-BigOr′*: *xs* ≠ [] ⟹ *size* (*BigOr′ xs*) + *1* = *sum-list* (*map size xs*) + *length xs*
⟨*proof*⟩

**lemma** *sizef-BigAnd′*: *xs* ≠ [] ⟹ *sizef* (*BigAnd′ xs*) + *1* = *sum-list* (*map sizef xs*) + *length xs*
⟨*proof*⟩

**lemma** *size-BigAnd′*: *xs* ≠ [] ⟹ *size* (*BigAnd′ xs*) + *1* = *sum-list* (*map size xs*) + *length xs*
⟨*proof*⟩

**lemma** *sizef-conv-sum-list-undisj*: *sizef* φ = *sum-list* (*map sizef* (*undisj* φ)) + *count-Or* φ
⟨*proof*⟩

**lemma** *size-conv-sum-list-undisj*: *size* φ = *sum-list* (*map size* (*undisj* φ)) + *count-Or* φ
⟨*proof*⟩

**lemma** *sizef-conv-sum-list-unconj*: *sizef* φ = *sum-list* (*map sizef* (*unconj* φ)) + *count-And* φ
⟨*proof*⟩

**lemma** *size-conv-sum-list-unconj*: *size* φ = *sum-list* (*map size* (*unconj* φ)) + *count-And* φ

⟨*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*)) ≤ *sizef* (*BigOr' xs*)
⟨*proof*⟩

**lemma** *size-BigOr'-filter-le-if*:
**assumes** ∃ *x* ∈ *set xs*. *P x*
**shows** *size* (*BigOr'* (*filter P xs*)) ≤ *size* (*BigOr' xs*)
⟨*proof*⟩

**lemma** *sizef-BigAnd'-filter-le*: *sizef* (*BigAnd'* (*filter P xs*)) ≤ *sizef* (*BigAnd' xs*)
⟨*proof*⟩

## 3.7 Fn function

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

**lemma** *inj-on-Var*[*simp*]: *inj-on* (λ(*x*, *y*). *Var x y*) *A* **for** *A*
⟨*proof*⟩

**fun** *Fn* :: *nat* ⇒ *var formula* **where**
*Fn 0* = (¬⊥)|
*Fn* (*Suc n*) =
*And*
(*And*
(*Or*
(*Atom* (*Var* (*Suc n*) *False*))
(*Atom* (*Var* (*Suc n*) *True*)))
(*Or*

(*Not* (*Atom* (*Var* (*Suc n*) *False*)))
(*Not* (*Atom* (*Var* (*Suc n*) *True*)))))
(*Fn n*)

**lemma** *sizef-Fn*: *sizef* (*Fn n*) = *8* ∗ *n* + *1*
⟨*proof*⟩

**lemma** *size-Fn*: *size* (*Fn n*) = *10* ∗ *n* + *2*
⟨*proof*⟩

**lemma** *is-cnf-Fn*: *is-cnf* (*Fn n*)
⟨*proof*⟩

**lemma** *is-nnf-Fn*: *is-nnf* (*Fn n*)
⟨*proof*⟩

**lemma** *Fn-sat*: ∃ α. α ⊨ *Fn n*
⟨*proof*⟩

**lemma** *semantics-Fn-iff*: α ⊨ *Fn n* ⟷ (∀ *i* ∈ {*1*..*n*}. α (*Var i False*) ≠ α (*Var i True*))
⟨*proof*⟩

## 3.8 Dual Function

**primrec** *dual* :: *'a formula* ⇒ *'a formula* **where**
*dual Bot* = *Not Bot* |
*dual* (*Atom v*) = *Not* (*Atom v*) |
*dual* (*Not v*) = *v* |
*dual* (*And* φ ψ) = *Or* (*dual* φ) (*dual* ψ) |
*dual* (*Or* φ ψ) = *And* (*dual* φ) (*dual* ψ) |
*dual* (*Imp* φ ψ) = *And* φ (*dual* ψ)

**lemma** *sizef-dual-Fn*: *sizef* (*dual* (*Fn n*)) = *8* ∗ *n* + *1*
⟨*proof*⟩

**lemma** *size-dual-Fn*: *size* (*dual* (*Fn n*)) = *10* ∗ *n* + *1*
⟨*proof*⟩

**lemma** *is-dnf-dual-Fn*: *is-dnf* (*dual* (*Fn n*))
⟨*proof*⟩

**lemma** *sizef-dual*: *sizef* (*dual* φ) = *sizef* φ
⟨*proof*⟩

**lemma** *size-dual-le*: *size* (*dual* φ) ≤ *2* ∗ *size* φ
⟨*proof*⟩

**lemma** *equiv-dual*: *equiv* (*dual* φ) (*Not* φ)

⟨*proof*⟩

**lemma** *is-disj-dual-if-is-conj*: *is-conj* $\varphi \Longrightarrow$ *is-disj* (*dual* $\varphi$)
⟨*proof*⟩

**lemma** *is-conj-dual-if-is-disj*: *is-disj* $\varphi \Longrightarrow$ *is-conj* (*dual* $\varphi$)
⟨*proof*⟩

**lemma** *is-dnf-dual-if-is-cnf*: *is-cnf* $\varphi \Longrightarrow$ *is-dnf* (*dual* $\varphi$)
⟨*proof*⟩

**lemma** *is-cnf-dual-if-is-dnf*: *is-dnf* $\varphi \Longrightarrow$ *is-cnf* (*dual* $\varphi$)
⟨*proof*⟩

**lemma** *dual-disj-not-taut-impl-sat*: *is-disj* $\varphi \Longrightarrow \exists\, \alpha.\ \neg\ \alpha \models \varphi \Longrightarrow \exists\, \alpha.\ \alpha \models$ *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* ⇒ *′a* ⇒ *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* ∨ *cont-pos* $\psi$ *l*) |
*cont-pos* (*Or* $\varphi$ $\psi$) *l* = (*cont-pos* $\varphi$ *l* ∨ *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* ⇒ *′a* ⇒ *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* ∨ *cont-neg* $\psi$ *l*) |
*cont-neg* (*Or* $\varphi$ $\psi$) *l* = (*cont-neg* $\varphi$ *l* ∨ *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* ⇒ *′a* ⇒ *bool* **where**

*cont Bot l = False* |
*cont (Atom v) l = (v = l)* |
*cont (Not (Atom v)) l = (v = l)* |
*cont (Not φ) l = False* |
*cont (And φ ψ) l = (cont φ l ∨ cont ψ l)* |
*cont (Or φ ψ) l = (cont φ l ∨ cont ψ l)* |
*cont (Imp φ ψ) l = False*

**lemma** *impl-not-cont-pos*: ¬ *cont-pos φ v* ⟹ *cont-neg φ v* ∨ ¬ (*cont φ v*)
⟨*proof*⟩

**lemma** *impl-not-cont*: ¬ *cont φ v* ⟹ ¬ *cont-pos φ v* ∧ ¬ *cont-neg φ v*
⟨*proof*⟩

**lemma** *mem-atoms-if-cont-pos*:
**assumes** *cont-pos T v*
**shows** *v* ∈ *atoms T*
⟨*proof*⟩

**lemma**
**assumes** *is-conj φ*
**shows**
*not-sat-conj-neg-true*: ∃ *v*. *cont-neg φ v* ∧ *α v* ⟹ ¬(*α* ⊨ *φ*) **and**
*not-sat-conj-pos-false*: ∃ *v*. *cont-pos φ v* ∧ ¬(*α v*) ⟹ ¬(*α* ⊨ *φ*)
⟨*proof*⟩

**lemma** *sat-conj-val-cont-ident*:
**assumes** *Val1* ⊨ *φ* **and** ∀ *v* ∈ {*v*. *cont φ v*}. *Val1 v* = *Val2 v* **and** *is-conj φ*
**shows** *Val2* ⊨ *φ*
⟨*proof*⟩

# 4 CNF to DNF

**proposition** *exp-blowup-from-Fn-to-BigOr′*:
**fixes** *n* :: *nat* **and** *Ts* :: *var formula list*
**defines** *φ* ≡ *Fn n* **and** *ψ* ≡ *BigOr′ Ts*
**assumes**
*n-greater-0*: *n* > *0* **and**
*Ts-spec*: (∀ *T*∈*set Ts*. *is-conj T* ∧ (∃ *α*. *α* ⊨ *T*)) **and**
*equiv φ ψ*
**shows** *sizef ψ* ≥ *n*∗*2*^*n*
⟨*proof*⟩

**lemma** *ex-equiv-disj-list-if-is-dnf*:
**fixes** *φ* :: *'a formula*
**assumes** *dnf*: *is-dnf φ* **and** *sat*: ∃ *α*. *α* ⊨ *φ*
**shows** ∃ (*Ts* :: *'a formula list*). *equiv φ* (*BigOr′ Ts*) ∧
*size* (*BigOr′ Ts*) ≤ *size φ* ∧
(∀ *T* ∈ *set Ts*. *is-conj T*) ∧

$(\forall\, T \in \mathit{set}\ \mathit{Ts}.\ \exists\, \alpha.\ \alpha \models T)$
⟨*proof*⟩

**theorem** *exp-blowup-from-CNF-to-DNF*:
**fixes** $n :: \mathit{nat}$
**shows** $\exists\,(\varphi :: \mathit{var\ formula}).$
$\mathit{is\text{-}cnf}\ \varphi \wedge$
$\mathit{size}\ \varphi = 10 * n + 2 \wedge$
$(\forall\,(\psi :: \mathit{var\ formula}).\ \mathit{equiv}\ \varphi\ \psi \longrightarrow \mathit{is\text{-}dnf}\ \psi \longrightarrow \mathit{size}\ \psi \geq n * 2\ \hat{}\ n)$
⟨*proof*⟩

# 5 DNF to CNF

**proposition** *exp-blowup-from-dual-Fn-to-BigAnd′*:
**fixes** $n :: \mathit{nat}$ **and** $\mathit{Cs} :: \mathit{var\ formula\ list}$
**defines** $\varphi \equiv \mathit{dual}\ (\mathit{Fn}\ n)$ **and** $\psi \equiv \mathit{BigAnd'}\ \mathit{Cs}$
**assumes**
*n-greater-0*: $n > 0$ **and**
*Cs-spec*: $(\forall\ C \in \mathit{set}\ \mathit{Cs}.\ \mathit{is\text{-}disj}\ C \wedge \neg(\forall\ \alpha.\ \alpha \models C))$ **and**
$\mathit{equiv}\ \varphi\ \psi$
**shows** $\mathit{sizef}\ \psi \geq n*2\hat{}n$
⟨*proof*⟩

**lemma** *ex-equiv-conj-list-if-is-cnf*:
**fixes** $\varphi :: 'a\ \mathit{formula}$
**assumes** *cnf*: $\mathit{is\text{-}cnf}\ \varphi$
**shows** $\exists\,(\mathit{Cs} :: 'a\ \mathit{formula\ list}).\ \mathit{equiv}\ \varphi\ (\mathit{BigAnd'}\ \mathit{Cs}) \wedge$
$\mathit{sizef}\ (\mathit{BigAnd'}\ \mathit{Cs}) \leq \mathit{sizef}\ \varphi \wedge$
$(\forall\, C \in \mathit{set}\ \mathit{Cs}.\ \mathit{is\text{-}disj}\ C) \wedge$
$(\forall\, C \in \mathit{set}\ \mathit{Cs}.\ \neg \models C)$
⟨*proof*⟩

**theorem** *exp-blowup-from-DNF-to-CNF*:
**fixes** $n :: \mathit{nat}$
**shows** $\exists\,(\varphi :: \mathit{var\ formula}).$
$\mathit{is\text{-}dnf}\ \varphi \wedge$
$\mathit{size}\ \varphi = 10 * n + 1 \wedge$
$(\forall\,(\psi :: \mathit{var\ formula}).\ \mathit{equiv}\ \varphi\ \psi \longrightarrow \mathit{is\text{-}cnf}\ \psi \longrightarrow \mathit{size}\ \psi \geq n * 2\ \hat{}\ n)$
⟨*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.