fun foo x y = x+y

real real real

(real → real → real)

curry form → functions can take one argument

fun foo x*y = x+y

(real * real) → real

uncurry form

'a list → 'a } most general type

λ-lambda calculus → log language

Introducing λ-calculus : to capture any type of functions.

Defn:

λ-expr → { λ variable . expr (abstraction)
expr. expr (application)
variable

functions using abstraction

(i) $\lambda x . \lambda y . xx$

(ii) $\lambda x . \lambda y . (x+y)$

function application

$(\lambda x . \lambda y . xy)(\lambda x . x)$

argument

$\lambda y . (\lambda x . x)\, y$

~~Beta Reduction~~ Notations & Conventions

$f\ g\ h \Rightarrow (f\ g)\ h$

↑ left associative

$f\ g\ h\ i\ j \Rightarrow (((f\ g)\ h)\ i)\ j)$

} convention of function application

$\lambda x.f\ g\ x. \Rightarrow \lambda x.(f\ g)\ x$

} application is given more priority

Free & Bound Variables

$\lambda x.x$

↑ Bound Variable

Variable is bound under the abstraction.

$\lambda x.x\ y$

↑ free Variable

$(\lambda x.x)\ (\lambda y.y\ x) \Leftrightarrow (\lambda m.m)\ (\lambda y.y\ x)$

$$FV(A) = \begin{cases} FV(x) = \{x\} \\ FV(\lambda x.B) = FV(B) \setminus \{x\} \\ FV(BC) = FV(B) \cup FV(C) \end{cases}$$

β-Reduction (Function Application)

$(\lambda x.e)\, f \Rightarrow_\beta [f/x]\, e$

FV(e)

In expr. e, replace all free occurrences of x by f

$(\lambda x.x)\, y \Rightarrow_\beta [y/x]\, x = y$

$(\lambda x.x\, z)\, y \Rightarrow_\beta [y/x]\, x z x = y z y$

$(\lambda x.z)\, y \Rightarrow_\beta [y/x]\, z = z$

$(\lambda x.xx)(\lambda y.y) \Rightarrow_\beta [\lambda y.y/x]\, xx = (\lambda y.y)(\lambda y.y)$

$\Downarrow_\beta$

$[\lambda y.y/y]$

$\downarrow$

$\lambda y.y$

α-Reduction → renaming the bound variables

$\lambda m.m \leftrightarrow \lambda t.t \leftrightarrow \lambda x.x$

$[y(\lambda x.x)/x]\, \lambda y.(\lambda x.x)\, y x \Rightarrow [y(\lambda x.x)/x]\, \lambda y.(\lambda x.x)\, y x$

$\Downarrow$

$\lambda y.(\lambda x.x)\, y\, y\, (\lambda x.x)$ — got bound here

$\Rightarrow_\alpha \left[y(\lambda t.t) / x \right] (\lambda w.(\lambda m.m) w\ x)$

$\Downarrow_\beta$ free should be free

$\lambda w.(\lambda m.m)\ w\ y(\lambda t.t)$

Recursive Defⁿ of β-defⁿ

(i) $[e/x]\ x \Rightarrow_\beta e$

(ii) $[e/x]\ y \Rightarrow_\beta y$

(iii) $[e/x]\ (f\ g) \Rightarrow_\beta [e/x] f\ [e/x] g$

(iv) $[e/x]\ \lambda x.f \Rightarrow_\beta \lambda x.f$

(v) $[e/x]\ \lambda y.f \Rightarrow_\beta \lambda y.\ [e/x] f$

y shouldn't be free here

Q. $S = \lambda x.\lambda y.\lambda z.(x\ z\ (y\ z))$

$K = \lambda x.\lambda y.x$

$S\ K\ K$

$[(\lambda x.\lambda y.\lambda z.(xy(yz)))(\lambda x.\lambda y.x)](\lambda x.\lambda y.x)$

SK

$\Rightarrow$

$\Rightarrow [\lambda m.\lambda z.((\lambda x.\lambda y.x)\, m\,(m\,z))]\ (\lambda x.\lambda y.x)$

$\Rightarrow$ ~~$[\lambda z.((\lambda x.\lambda y.x)(\lambda x.\lambda y.x))](\lambda x.\lambda y.x)$~~

$\lambda z.((\lambda x.\lambda y.x)(\lambda x.\lambda y.x))$

$\Rightarrow$

$[\lambda x.\lambda y.\lambda z.(x\,z\,(y\,z))]\ K\ K$

$\Rightarrow [\lambda y.\lambda z.(K\,z\,(y\,z))]\ K$

$\Rightarrow \lambda z.(K\,z\,(K\,z))$

$\Rightarrow \lambda z.((\lambda x.\lambda y.x)\,z\,(K\,z))$

$\Rightarrow \lambda z.((\lambda y.z)(K\,z))$

$\Rightarrow \lambda z.z$

~~$\lambda x.\lambda y.\lambda z.(x\,z\,(y\,z))$~~

$\Rightarrow \lambda z.((\lambda x.\lambda y.x)\,z\,(K\,z))$

$\lambda z.((\lambda y.z)(K\,z))$

$\lambda z.z$

~~$\lambda z.(z\,z)$~~

$\Rightarrow$ ~~$\lambda z.z\,z$~~

$$e = \begin{cases} x \\ \lambda x.e \\ f\,e \end{cases}$$

Type

$$\boxed{\tau = bool \mid \tau_1 \to \tau_2}$$

, bool
$(bool \to bool) \to bool$

$e = x$
$= f\,e$
$= \lambda(x:\tau).e$
$= true : bool$
$= false : bool$
$= and \quad (bool \to bool \to bool)$
$= not \quad (bool \to bool)$

Type-checking

Given an expression 'e', check whether it is well-typed.
↳ Yes or No

~~Idea~~

Type for 'e' needs knowledge about the types of free variables of e.

$\Gamma : \{ x_1 : \tau_1 , x_2 : \tau_2 , \ldots\ldots\ldots , \ldots \}$

↓ assumptions of type of free variables

(i) $\Gamma \vdash e : \tau$

$\{ x : bool \} \vdash x : bool$

$\lambda x . x \qquad \{\} \vdash (bool \to bool)$

Type Inference Rules

(i) VAR

$$\frac{\text{no precondition}}{\Gamma \cup \{ x : \tau \} \vdash \{ x : \tau \}}$$

(ii) APP $(f\,e)$

$$\frac{\Gamma \vdash f : \tau_1 \to \tau_2 \quad \Gamma \vdash e : \tau_1}{\Gamma \vdash f\,e : \tau_2}$$

(iii) ABS $(\lambda x . e)$

$$\frac{\Gamma \cup \{ x : \tau_1 \} \vdash e : \tau_2}{\Gamma \vdash (\lambda x . e) : (\tau_1 \to \tau_2)}$$

(i) $\lambda x.x$

$\{x:bool\} \vdash x:bool$ VAR

$\{\} \vdash \lambda(x:bool).x : bool \to bool$ ABS

(ii) $\lambda x.\lambda y.x$

0.) $\Gamma = \{\}$

1.) $\Gamma \cup \{x:\tau_1\} \vdash x:\tau_1$ (VAR)

2.) $\Gamma \cup \{y:\tau_2\} \cup \{x:\tau_1\} \vdash x:\tau_1$

3.) $\Gamma \vdash (\lambda y.x) : \tau_2 \to \tau_1$ (ABS)

4.) $\Gamma \cup \{x:\tau_1\} \vdash (\lambda y.x) : \tau_2 \to \tau_1$

5.) $\Gamma \vdash (\lambda x.\lambda y.x) : \tau_1 \to \tau_2 \to \tau_1$ (ABS)

(iii) $\lambda x.\lambda y.\lambda z.\ x\ z\ (y\ z)$

(iv) $\lambda x.\lambda y.\lambda z.\ x\ y\ z$

0.) $\Gamma = \{\}$

1.) $\Gamma \cup \{x:\tau_1, y:\tau_2, z:\tau_3\} \vdash x:\tau_1, y:\tau_2, z:\tau_3$ (VAR)

2.) $\Gamma \vdash y:\tau_3 \to \tau_3, \Gamma \vdash z:\tau_3$

$\Gamma \vdash yz : \tau_3$

3.) $\Gamma \vdash x:\tau_2 \to \tau_3, \Gamma \vdash yz:\tau_2$

$\Gamma \vdash xyz : \tau_3$

0.) $\Gamma = \{\}$

1.) $\Gamma \cup \{z:\tau_1\} \vdash z:\tau_1$

2.) $\Gamma \vdash y:\tau_1 \to \tau_2, \Gamma \vdash z:\tau_1$

$\Gamma \vdash yz : \tau_2$

4.) $\Gamma \cup \{z:\tau_1\} \vdash (xyz):\tau_3$

$\Gamma \vdash (\lambda z.xyz) : (\tau_1 \to \tau_3)$

(5.) $\dfrac{\Gamma \cup \{y : \tau_1 \to \tau_2\} \vdash (\lambda z.\, x\, y\, z) : (\tau_1 \to \tau_3)}{\Gamma \vdash (\lambda y.\, \lambda z.\, x\, y\, z) : (\tau_1 \to \tau_2) \to (\tau_1 \to \tau_3)}$

(6.) $\dfrac{\Gamma \cup \{x : \tau_2 \to \tau_3\} \vdash (\lambda y.\, \lambda z.\, x\, y\, z) : (\tau_1 \to \tau_2) \to (\tau_1 \to \tau_3)}{\Gamma \vdash (\lambda x.\, \lambda y.\, \lambda z.\, x\, y\, z) : (\tau_2 \to \tau_3) \to (\tau_1 \to \tau_2) \to (\tau_1 \to \tau_3)}$

(ii) $\lambda x.\, \lambda y.\, \lambda z.\, x\, z\, (y\, z)$

0.) $\Gamma = \{\}$

1.) $\Gamma \cup \{z : \tau_1\} \vdash z : \tau_1$ (VAR)

2.) $\dfrac{\Gamma \cup \{y : \tau_1 \to \tau_2\}, \Gamma \vdash z : \tau_1}{\Gamma \vdash (y\, z) : \tau_2}$

3.) $\dfrac{\Gamma \vdash \{x : \tau_3 \to \tau_4\}, \Gamma \vdash z\,(y\,z) : \tau_3}{\Gamma \vdash x\, z\, (y\, z) : \tau_4}$ (APP)

4.) $\dfrac{\Gamma \cup \{z : \tau_1\} \vdash (x\, z\, (y\, z)) : \tau_4}{\Gamma \vdash \lambda z.\, (x\, z\, (y\, z)) : \tau_1 \to \tau_4}$ (ABS)

5.) $\dfrac{\Gamma \cup \{y : \tau_1 \to \tau_2\}, \Gamma \vdash (\lambda z.\, (x\, z\, (y\, z)) : \tau_1 \to \tau_4}{\Gamma \vdash \lambda y.\, \lambda z.\, (x\, z\, (y\, z)) : (\tau_1 \to \tau_2) \to (\tau_1 \to \tau_4)}$

6.) $\dfrac{\Gamma \cup \{x : \tau_3 \to \tau_4\} \vdash (\lambda y.\, \lambda z.\, (x\, z\, (y\, z)) : (\tau_1 \to \tau_2) \to (\tau_1 \to \tau_4)}{\Gamma \vdash (\tau_3 \to \tau_4) \to (\tau_1 \to \tau_2) \to (\tau_1 \to \tau_4)}$

Monotypes

$\tau = bool$
$\quad | \ \tau_1 \to \tau_2$
$\quad | \ \alpha$ → type variable

Polytype

$\sigma = \tau \mid \forall\alpha, \sigma_1 \mid \forall\alpha_1 \forall\alpha_2 \ldots\ldots \forall\alpha_m . \tau$

* $\forall\alpha \ (\alpha \to \beta)$ } σ_1
* ~~$\forall\beta, \forall\alpha$~~ $(\alpha \to \beta)$ — σ_1 — σ_2

~~GEN~~ SPECIALIZATION

$\sigma = \forall\alpha, \alpha \to \alpha$
$\sigma' = \forall\beta, \forall\gamma, \ (\beta \to \gamma \to \beta) \to (\beta \to \gamma \to \beta)$
$\sigma' \leq \sigma$

$$\frac{\Gamma \vdash e : \sigma}{\Gamma \vdash e : \sigma'} \quad \sigma' \leq \sigma$$

$\sigma = \forall\alpha_1\ \forall\alpha_2 \ldots\ldots \forall\alpha_n . \tau$

Fresh Variable

$\sigma_0 = \tau\,[\tau_1/\alpha_1]\,[\tau_2/\alpha_2] \ldots\ldots [\tau_n/\alpha_n]$

$\sigma' = \forall\beta_1\forall\beta_2\forall\beta_3 \ldots\ldots \forall\beta_n\,(\tau[\tau_1/\alpha_1] \ldots\ldots [\tau_n/\alpha_n])$

$\sigma' \leq \sigma$

map: $\forall'a, \forall'b$ $\overset{\tau}{[\,('a \to 'b) \to 'a\ list \to 'b\ list\,]}$

specialisation ↓

$\sigma_0 = \tau\,[\ldots/\ldots]$

$\sigma' = \forall'a, \forall'\gamma, \forall'c$ $[\,'a \to ('\gamma \to 'c) \to 'a\ list \to ('\gamma \to 'c)\ list$

GENERALISATION

$$\frac{\Gamma \vdash e : \sigma}{\Gamma \vdash e : \forall\alpha.\sigma}$$

α is not free in Γ

eg. ① $\lambda x.x$

$\{x:\alpha\} \vdash x:\alpha$

$\{\} \vdash \lambda x.x : \alpha \to \alpha$

$\{\} \vdash \lambda x.x : \forall\alpha, \alpha \to \alpha$

$\{x:\alpha\} \vdash x:\alpha$

$\{x:\alpha\} \vdash x:\forall\alpha\,\alpha$

$\{x:\alpha\} \vdash x:\forall\beta\,\beta$

$\{\} \vdash \lambda x.x\ \forall\alpha\,\forall\beta\,(\alpha \to \beta)$

wrong because α is free in Γ

Hindley-Milner

Input: e, Γ → set of assumptions (expr)

Output: $\sigma = \forall\alpha_1, \forall\alpha_2 \ldots\ldots, \forall\alpha_n . \tau$

$\Gamma \vdash e : \tau \xrightarrow{\text{closure}} \Gamma \vdash e : \sigma$

$\sigma \equiv \forall\alpha_1, \forall\alpha_2 \ldots\ldots \forall\alpha_m \tau$

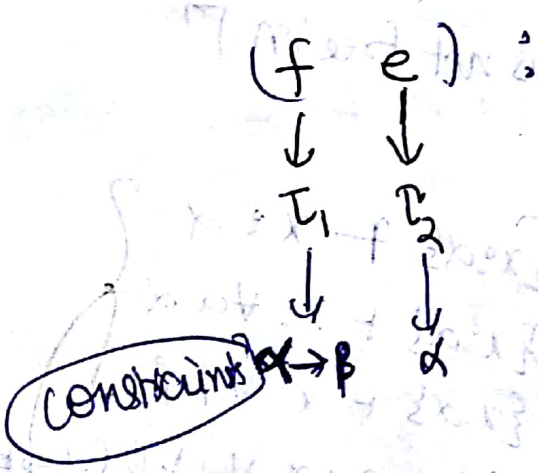
$\{\alpha_1, \alpha_2 \ldots\ldots \alpha_m\}$
$= FV(\tau) \setminus FV(\Gamma)$

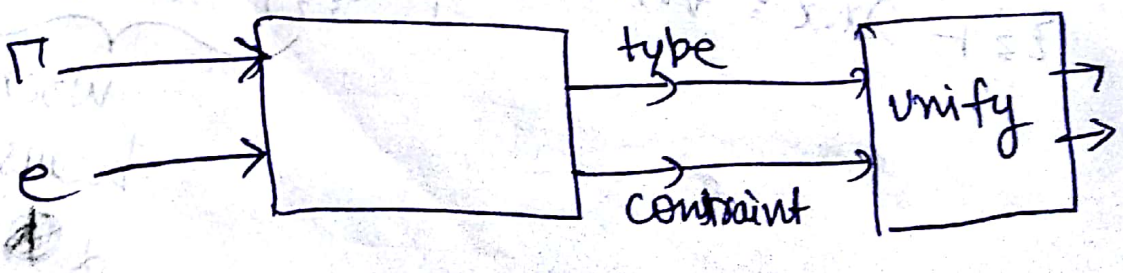
* No quantification in assumption

map: $('a \to 'b) \to ('a\ list \to 'b\ list)$

fun: $(int \to int)$

(mapfun): $(int\ list \to int\ list)$





Inference Algo.

$$infer(x, \Gamma \equiv \{x : \tau\}) \Rightarrow (\{x : \tau\}, \emptyset)$$

type — constraint

$$infer(fe, \Gamma \equiv \{\dots\}) \Rightarrow$$

- $infer(f, \Gamma) \Rightarrow (\tau_1, c_1)$
- $infer(e, \Gamma) \Rightarrow (\tau_2, c_2)$
- $(\beta, c_1 \cup c_2 \cup \{\tau_1 \equiv \alpha \to \beta, \tau_2 \equiv \alpha\})$

$$infer(\lambda x.e, \Gamma)$$

- ~~$infer(e, \Gamma) \Rightarrow$~~
- $infer(e, \Gamma \cup \{x : \tau_0\}) \Rightarrow (\tau_1, c_1)$
- $infer(\lambda x.e, \Gamma) \Rightarrow (\tau \to \tau_1, c_1)$

map: $('a \to 'b) \to ('a\ list \to 'b\ list)$

inc : $(int \to int)$

map inc ?

$\Gamma \vdash (map\ inc, \emptyset) \Rightarrow$

$\Gamma \vdash (map, \emptyset) \Rightarrow (\tau_1, C_1)$

$\Gamma \vdash (inc, \emptyset) \Rightarrow (\tau_2, C_2)$

$\beta,\ C_1 \cup C_2 \cup \{('a \to 'b) \to 'a\ list \to 'b\ list\} \equiv \alpha \to \beta$

$int \to int \equiv \alpha$

$('a \to 'b) \to 'a\ list \to 'b\ list$

$int \to int \equiv \alpha$

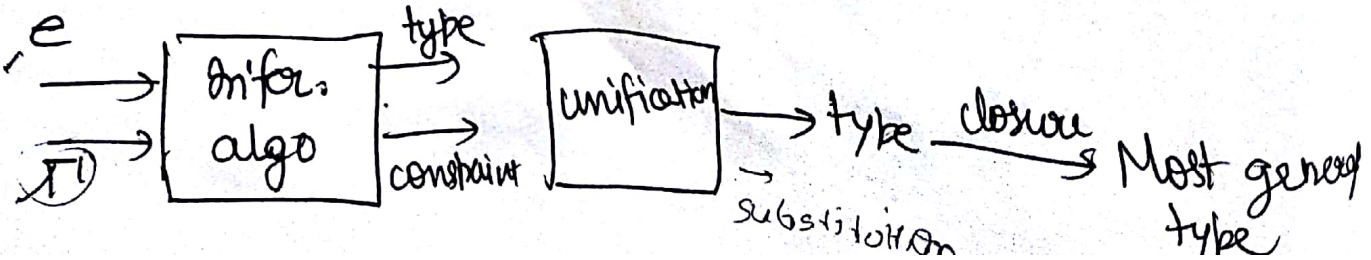
$\alpha \equiv 'a \to 'b$

$\beta \equiv 'a\ list \to 'b\ list$

$'a \equiv int$

$'b \equiv int$

$\beta \equiv int\ list \to int\ list$



Unification (final Algo)

$$C = (\tau_{11} \equiv \tau_{12}, \tau_{21} \equiv \tau_{22}, \tau_{31} \equiv \tau_{32}, \ldots\ldots,)$$

Structure $(\mathcal{S}) = \{f_0, f_1, f_2, \ldots\ldots, f_m\}$

- f_0: set of functions
- f_1: one arg.
- f_m: m-arguments

terms $(\mathcal{S}) \Rightarrow f_i^0 \quad i = 0$

$\quad\quad | \; f_j^0(t_1, t_2, \ldots\ldots, t_j^0)$

finite & countably infinite

$V \equiv$ set of variables.

Terms $(\mathcal{S}, V)$

$t = V$

$f_i(t_1, t_2, \ldots\ldots, t_{n_i})$ where t_i's are terms

$\mathcal{S} = \{\rightarrow$ (2 arg), bool (0 arg)$\}$

terms$(\mathcal{S}) \Rightarrow \tau_1 \rightarrow \tau_2 \Rightarrow$ bool

$\mathcal{S} = \{0, S\} \rightarrow$ terms$(\mathcal{S}) = 0 \mid S(t)\}$ NN

eg. $V = \{\alpha, \beta\}$

$t = \text{Bool} \mid t_1 \rightarrow t_2 \mid \alpha.$

Telescoping

telescopic property : all substitutions must satisfy.

$M \equiv \{\alpha \rightarrow \alpha\beta \;,\; \beta \rightarrow \beta\alpha\}$ } doesn't satisfy

for eg. $\alpha \rightarrow \alpha$

$\hookrightarrow \alpha\beta \rightarrow \alpha\beta$

$\hookrightarrow \alpha\alpha\beta \rightarrow \alpha\alpha\beta$

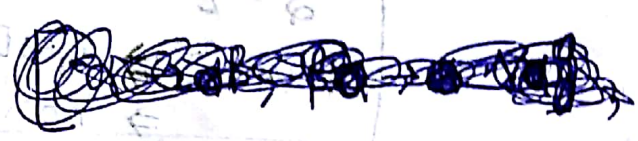
set of substitutions

$$\left[t_1/x_1 , t_2/x_2 , \ldots\ldots , t_k/x_k \right]$$

t_i shouldn't contain $x_1, x_2, \ldots x_i$

substitution is valid only if this property is valid.

it is the property of the sequence not the set.



Telescoping is the property of a sequence. If you rearrange the sequence, it is not necessary that new sequence, satisfy this prop.

Constraint: $t_1 \equiv t_2$

$$t_1[M] \equiv t_2[M]$$

(M: set of subs.)

$t_2[M]$ → substitution of constraints

$t \equiv t$

t → is unifier for t_1 & t_2

unifier:

$t \leq t_1$

$t \leq t_2$

We need most general unifier.

$\alpha \to \beta \equiv \gamma \to \beta$

eg: $\beta \to int$, $\alpha \to int$, $\gamma \to int$ } not most general

$\alpha \to \gamma$ → most general

what is most general unifier?

$t_1[M] \equiv t_2[M]$

$\forall t,\ t \leq t'$, where t' is most general unifier and t is unifier for t_1 & t_2

$t_1 \equiv t_2$

$\downarrow$

$t_{11} = t_{12}$, $t_{21} \equiv t_{22}$, $t_{31} \equiv t_{32}$, ----- , ----- $t_{n1} \equiv t_{n2}$

$('a \to 'b) \to 'a\ list \to 'b\ list \equiv \alpha \to \beta$

↓ breaking the constraint

$\alpha \equiv ('a \to 'b)$

$\beta \equiv ('a\ list \to 'b\ list)$

cond^n for breaking :-

$t_1 = f_n(t_{11}, t_{21}, t_{31}, \dots, t_{n1})$ ← no. of terms → arity

$t_2 = f_n(t_{12}, t_{22}, t_{32}, \dots, t_{n2})$

modified constraints

unify2 (t_1 , t_2 , M) → M'

↑ terms ↑ set of constraints

→ set of substitutions

unifyList (x M) (foldr unify2)

↑ set of constraints

$\{t_{11} \equiv t_{21}, t_{21} \equiv t_{22}, \ldots\}$

return modified constraints

folds from left

fun unifyList = foldr unify2;

unifyList (C, M)

C : set of constraints

M : set of substitutions

unify2 (t_1 , t_2 , S)

case 1

$t_1 \equiv f_i(t_{11}, t_{12}, \ldots, t_{1n_i})$

$t_2 \equiv f_j(t_{21}, t_{22}, \ldots t_{2n_j})$

if $i \neq j$ → fail.

o/w. unifyList $[(t_{11} \equiv t_{21}), (t_{12} \equiv t_{22}), \ldots, (t_{1n_i} \equiv t_{2n_j})]$

Case-2

If $t_1 = V$ (let t_1 is variable) (without loss of generality)

(i) $t_2 = V$, then $\in S$

(ii) $V \in Var(t_2)$ then fail

(iii) $[t/V] \in S \Rightarrow$ then unify2(t, t_2, S)

(iv) S append $\rightarrow \left[\frac{t_2[S]}{V}\right]$

↓

(v) after performing all substitutions

Resolution

$C_1 \wedge C_2 \wedge \dots \wedge C_n$

↓

Clauses → sequence of literals.

$C_i \Rightarrow l_{i1} \vee l_{i2} \vee \dots \vee l_{im_i}$

$l \Rightarrow x \mid \bar{x}$

→ CNF

$C_1 \vee C_2 \vee \dots \vee C_n$

↓

clause $\Rightarrow l_{i1} \wedge l_{i2} \wedge \dots \wedge l_{im_i}$

$l \Rightarrow x \mid \bar{x}$

→ DNF