Disjunktive und Konjunktive Normalform

DNF via Wahrheitstabellen

```
(p \/ q /\ r) /\ (~p \/ ~r)
```

```
# print_truthtable fm;;
p     q     r     | formula
---------------------------
false false false | false
false false true  | false
false true  false | false
false true  true  | true
true  false false | true
true  false true  | false
true  true  false | true
true  true  true  | false
---------------------------
```

```
~p /\ q /\ r \/ p /\ ~q /\ ~r \/ p /\ q /\ ~r
```

DNF via Umformung

```
let rec distrib fm =
  match fm with
    And(p,(Or(q,r))) -> Or(distrib(And(p,q)),distrib(And(p,r)))
  | And(Or(p,q),r) -> Or(distrib(And(p,r)),distrib(And(q,r)))
  | _ -> fm;;
```

```
let rec rawdnf fm =
  match fm with
    And(p,q) -> distrib(And(rawdnf p,rawdnf q))
  | Or(p,q) -> Or(rawdnf p,rawdnf q)
  | _ -> fm;;
```

```
# rawdnf <<(p \/ q /\ r) /\ (~p \/ ~r)>>;;
- : prop formula =
<<(p /\ ~p \/ (q /\ r) /\ ~p) \/ p /\ ~r \/ (q /\ r) /\ ~r>>
```

Mengenbasierte Darstellung

```
let distrib s1 s2 = setify(allpairs union s1 s2);;

let rec purednf fm =
  match fm with
    And(p,q) -> distrib (purednf p) (purednf q)
  | Or(p,q) -> union (purednf p) (purednf q)
  | _ -> [[fm]];;
```

```
# purednf <<(p \/ q /\ r) /\ (~p \/ ~r)>>;;
- : prop formula list list =
[[<<p>>; <<~p>>]; [<<p>>; <<~r>>]; [<<q>>; <<r>>; <<~p>>];
 [<<q>>; <<r>>; <<~r>>]]
```

```
let trivial lits =
  let pos,neg = partition positive lits in
  intersect pos (image negate neg) <> [];;
```

```
# filter (non trivial) (purednf <<(p \/ q /\ r) /\ (~p \/ ~r)>>);;
- : prop formula list list = [[<<p>>; <<~r>>]; [<<q>>; <<r>>; <<~p>>]]
```

Mengenbasierte Darstellung

```
let simpdnf fm =
  if fm = False then [] else if fm = True then [[]] else
  let djs = filter (non trivial) (purednf(nnf fm)) in
  filter (fun d -> not(exists (fun d' -> psubset d' d) djs)) djs;;
```

```
let dnf fm = list_disj(map list_conj (simpdnf fm));;
```

```
# let fm = <<(p \/ q /\ r) /\ (~p \/ ~r)>>;;
val fm : prop formula = <<(p \/ q /\ r) /\ (~p \/ ~r)>>
# dnf fm;;
- : prop formula = <<p /\ ~r \/ q /\ r /\ ~p>>
# tautology(Iff(fm,dnf fm));;
- : bool = true
```

Konjunktive Normalform

Nach den DeMorganschen Gesetzen gilt, dass wenn $\neg p \Leftrightarrow \bigvee_{i=1}^{m} \bigwedge_{j=1}^{n} p_{ij}$

dann $p \Leftrightarrow \bigwedge_{i=1}^{m} \bigvee_{j=1}^{n} \neg p_{ij}.$

```
let purecnf fm = image (image negate) (purednf(nnf(Not fm)));;
```

```
# let fm = <<(p \/ q /\ r) /\ (~p \/ ~r)>>;;
val fm : prop formula = <<(p \/ q /\ r) /\ (~p \/ ~r)>>
# cnf fm;;
- : prop formula = <<(p \/ q) /\ (p \/ r) /\ (~p \/ ~r)>>
# tautology(Iff(fm,cnf fm));;
- : bool = true
```

Definitorische KNF

$(p \lor (q \land \neg r)) \land s$

$(p1 \Leftrightarrow q \land \neg r) \land (p \lor p1) \land s$

$(p1 \Leftrightarrow q \land \neg r) \land (p2 \Leftrightarrow p \lor p1) \land p2 \land s$

$(p1 \Leftrightarrow q \land \neg r) \land (p2 \Leftrightarrow p \lor p1) \land (p3 \Leftrightarrow p2 \land s) \land p3$

$(\neg p1 \lor q) \land (\neg p1 \lor \neg r) \land (p1 \lor \neg q \lor r) \land$
$(\neg p2 \lor p \lor p1) \land (p2 \lor \neg p) \land (p2 \lor \neg p1) \land$
$(\neg p3 \lor p2) \land (\neg p3 \lor s) \land (p3 \lor \neg p2 \lor \neg s) \land$
$p3$

Definitorische KNF

```
let mkprop n = Atom(P("p_"^(string_of_num n))),n +/ Int 1;;
```

```
let rec maincnf (fm,defs,n as trip) =
  match fm with
    And(p,q) -> defstep mk_and (p,q) trip
  | Or(p,q) -> defstep mk_or (p,q) trip
  | Iff(p,q) -> defstep mk_iff (p,q) trip
  | _ -> trip
```

```
and defstep op (p,q) (fm,defs,n) =
  let fm1,defs1,n1 = maincnf (p,defs,n) in
  let fm2,defs2,n2 = maincnf (q,defs1,n1) in
  let fm' = op fm1 fm2 in
  try (fst(apply defs2 fm'),defs2,n2) with Failure _ ->
  let v,n3 = mkprop n2 in (v,(fm'|->(v,Iff(v,fm'))) defs2,n3);;
```

Definitorische KNF

```
let max_varindex pfx =
  let m = String.length pfx in
  fun s n ->
    let l = String.length s in
    if l <= m or String.sub s 0 m <> pfx then n else
    let s' = String.sub s m (l - m) in
    if forall numeric (explode s') then max_num n (num_of_string s')
    else n;;
```

```
let mk_defcnf fn fm =
  let fm' = nenf fm in
  let n = Int 1 +/ overatoms (max_varindex "p_" ** pname) fm' (Int 0) in
  let (fm'',defs,_) = fn (fm',undefined,n) in
  let deflist = map (snd ** snd) (graph defs) in
  unions(simpcnf fm'' :: map simpcnf deflist);;
```

```
let defcnf fm = list_conj(map list_disj(mk_defcnf maincnf fm));;
```

Definitorische KNF

```
# defcnf <<(p \/ (q /\ ~r)) /\ s>>;;
- : prop formula =
<<(p \/ p_1 \/ ~p_2) /\
  (p_1 \/ r \/ ~q) /\
  (p_2 \/ ~p) /\
  (p_2 \/ ~p_1) /\
  (p_2 \/ ~p_3) /\
  p_3 /\
  (p_3 \/ ~p_2 \/ ~s) /\ (q \/ ~p_1) /\ (s \/ ~p_3) /\ (~p_1 \/ ~r)>>
```