(\* PEOJECT DESCRIPTION:

\* This program computes the conjunctive normal form "cnf" of a sentential logic.

\* It implement a datatype called "sentence" and logical connectives.

\* In order to get get cnf of a sentence, the following functions are implemented:

\* - removeArrows: transforms arrows(-->, <-> ) into their equivalent in "v" and "&"

\* - bringInNegation: eliminates 2 consecutive negations and distributes negation

in such a way that negation of "v" is "&" and vice-versa.

\* - distributeDisjInConj: distribute disjunction in conjunction.

\* - prints out results. \*)

(\*================================================================================\*)

Control.Print.printDepth := 200; (\* set the depth of an object (list) to print \*)

Control.Print.printLength := 200; (\* set the length of an object (list) to print \*)

(\*================================================================================\*)

(\* DEFINE THE LOGICAL CONNECTIVES - INFIX (as in the usual arithmetic operators \*)

infix -->; (\* implication operator: reads p implies q \*)

infix v; (\* disjunction operator: reads p or q\*)

infix &; (\* conjunction operator: reads p and q \*)

infix <->; (\* equivalence operator: reads p equivalent to q\*)

(\* DATA TYPE FOR A SENTENCE \*)

datatype sentence = P | Q | R | S | T (\* allowable sent. vars \*)

| ~ of sentence (\* negation: ~P \*)

| v of sentence \* sentence (\* disjunction: P v Q \*)

| & of sentence \* sentence (\* conjunction: P & Q \*)

| --> of sentence \* sentence (\* conditional: P --> Q \*)

| <-> of sentence \* sentence; (\* biconditional: P <-> Q \*)

(\*===============================================================================\*)

(\*REMOVE ARROWS: "-->" and "<->" from formulae\*)

fun removeArrows(~f) = ~(removeArrows (f))

|removeArrows(f & g) = (removeArrows (f) & removeArrows g)

|removeArrows(f v g) = (removeArrows (f) v removeArrows g)

|removeArrows(f --> g) = (~(removeArrows f) v (removeArrows g))

|removeArrows(f <-> g) = (((removeArrows f) & (removeArrows g)) v

(~(removeArrows f) & ~(removeArrows g)))

|removeArrows (f) = f;

(\*===============================================================================\*)

(\*BRING IN NEGATION: eliminates 2 consecutive negations and distribute negation in such a way that negation of "v" is "&" and vice-versa \*)

fun bringInNegation (~(~f)) = (bringInNegation f)

|bringInNegation (f & g) = (bringInNegation f) & (bringInNegation g)

|bringInNegation (f v g) = (bringInNegation f) v (bringInNegation g)

|bringInNegation (~(f v g)) = (bringInNegation (~f)) & (bringInNegation (~g))

|bringInNegation (~(f & g)) = (bringInNegation (~f)) v (bringInNegation (~g))

|bringInNegation (~f) = ~(bringInNegation f)

|bringInNegation (f) = f;

(\*DISTRIBUTE DISJUNCTION IN CONJUNCTION: distribute disjunction in conjunction\*)

fun distributeDisjInConj (f v (g & h)) = (distributeDisjInConj(f v g) &

distributeDisjInConj(f v h))

|distributeDisjInConj ((g & h) v f) = (distributeDisjInConj(g v f) &

distributeDisjInConj(h v f))

|distributeDisjInConj (f & g) = (distributeDisjInConj (f) &

distributeDisjInConj (g))

|distributeDisjInConj (f v g) = (distributeDisjInConj (f) v

distributeDisjInConj (g))

|distributeDisjInConj (f) = f;

(\*===============================================================================\*)

(\* TRANSFORM A SENTENCE INTO ITS EQUIVALENCE IN CONJUNCTIVE NORMAL FORM \*)

fun cnf (f) = cnf\_1(bringInNegation(removeArrows f))

and cnf\_1 (f v g) = distributeDisjInConj((cnf\_1 f) v (cnf\_1 g))

|cnf\_1 (f & g) = (cnf\_1 f) & (cnf\_1 g)

|cnf\_1 (f) = f;

(\*================================================================================\*)

(\*PRINTING SENTENCES\*)

fun show2 (P) = (print"P")

|show2 (Q) = (print"Q")

|show2 (R) = (print"R")

|show2 (S) = (print"S")

|show2 (T) = (print"T")

|show2 (~(f & g)) = (print"(-"; show2 (f & g); print")")

|show2 (~(f v g)) = (print"(-"; show2 (f v g); print")")

|show2 (~(f --> g)) = (print"(-"; show2 (f --> g); print")")

|show2 (~(f <-> g)) = (print"(-"; show2 (f <-> g); print")")

|show2 (~f) = (print"-" ; show2 f)

|show2 (f as \_) = (print"(" ; show f; print")")

and show (f v g) = (show2 f ; print" v " ; show2 g)

|show (f & g) = (show2 f ; print" & " ; show2 g)

|show (f --> g) = (show2 f ; print" -> " ; show2 g)

|show (f <-> g) = (show2 f ; print" <-> "; show2 g)

|show (~f) = (print"-"; show2 f)

|show (f) = (show2 f);

(\*===============================================================================\*)

(\* TOP LEVEL FUNCTIONS \*)

(\*Runs on a sentence and prints it out as well as its CNF\*)

fun run sentence = (print "\nSentence is: ";

show sentence;

print "\n\nIts CNF is : ";

show(cnf sentence);

print "\n\n");

(\*Prints out from 0 to N strings/character strings\*)

fun printNStr(str, 0) = ()

|printNStr(str, n) = (print str; printNStr(str,n-1));

(\*Runs on all sentence on a list of sentences\*)

fun go1(\_,\_,nil) = print "\n"

|go1(i,n,s::ss) = if i>n

then ()

else (print "\n";

if i>=10 then printNStr(" ",69) else printNStr(" ",70);

print "Formula #";

print(Int.toString i);

run s;

printNStr("=", 80);

go1(i+1,n,ss));

(\*===============================================================================\*)

(\* TOP LEVEL DRIVING FUNCTION \*)

fun go setenceList = let

val count = length setenceList

in

(printNStr("=",80);

go1(1,count,setenceList) )

end;

(\*===============================================================================\*)

(\*For debugging and verification only\*)

(\*get conjuncts\*)

fun getConjuncts (c1 & c2) = (getConjuncts(c1); getConjuncts(c2))

|getConjuncts (c) = (print("\* ("); show c; print")\n");

(\* Verify if cnf of all sentences are indeed in conjunctive normal form by

\* printing out their respective conjuncts\*)

fun verifyCNFs ([], i) = ()

|verifyCNFs (p::xp, i) = (printNStr("=", 25);

print("\nSENTENCE #"^(Int.toString(i))^": ");

print("\nCONJUNCTS: \n");

getConjuncts(cnf p);

verifyCNFs(xp, i+1))

(\*================================================================================\*)

(\*This is just for convenience for program testing\*)

fun exec () = (use "project4.sml");

(\*================================================================================\*)