

Beispielmakro: Dicomplement.omf

Makro zur Berechnung von Dikomplementen in Verbänden

Allgemeine Bemerkungen:

Ein Dikomplement auf einem Verband (L, \leq) ist ein Paar (Δ, ∇) von einstellig Operationen auf L , so dass folgende Bedingungen erfüllt sind:

$$\begin{array}{ll} x^{\Delta\Delta} \leq x & x^{\nabla\nabla} \geq x \\ x \leq y \Rightarrow x^{\Delta} \geq y^{\Delta} & x \leq y \Rightarrow x^{\nabla} \geq y^{\nabla} \\ (x \wedge y) \vee (x \wedge y^{\Delta}) = x & (x \vee z) \wedge (x \vee y^{\nabla}) = x \end{array}$$

Für weitere Angaben zum Thema Dikomplemente vergleiche:

Leonard Kwuida. *Dicomplemented Lattices - A Contextual Generalization of Boolean Algebras*. Aachen: Shaker Verlag, 2004.

Das Makro in [Dicomplement.omf](#) berechnet auf einem Verband, den der Anwender frei wählen kann, die Menge aller möglichen Definitionen für ein Dikomplement, genauer aller möglichen Operationen, die gemäss obiger Definition als erste Komponente eines Dikomplements zulässig sind. Zu diesem Zweck werden alle möglichen einstelligen Operationen als Liste ihrer Werte erzeugt, d.h. eine einstellige Operation f auf (L, \leq) ist gegeben durch die Liste $(f(x_1), \dots, f(x_n))$ ihrer Werte auf $L = \{x_1, \dots, x_n\}$. Um den Algorithmus zu beschleunigen, werden nicht die vollständigen Listen auf Erfüllung der Bedingungen getestet, sondern vielmehr bereits *partielle* Listen auf das *potentielle* Erfüllen der Bedingungen, da im Falle des Nicht-Erfüllens sämtliche Erweiterungen der Liste gar nicht mehr untersucht werden müssen.

Die Implementation (Download: unformatierter Quelltext in [Dicomplement.omf](#)):

```
% isdicomplement prüft, ob die Liste argumentlist bis zum Index i die Bedingungen eines
% Dikomplements erfüllen. %
isdicomplement:=lambda(i)
  % Die drei Bedingungen werden mit "AND" logisch verknüpft. %
  and(
    % Für alle x, y muss gelten  $(x \wedge y) \vee (x \wedge y^{\Delta}) = x$ : %
    forall(x in L, forall(y in L,
      % argumentlist wird nur bis Index i getestet, also darf der Index von y in L höchstens i sein:
      %
      if <=(indexof(L, y), i) then
        % Die folgenden Zwischenschritte könnten auch ausgelassen werden und durch "if equal
        (sup(inf(x,y),inf(x,item(argumentlist,indexof(L,y))))),x) then true else false endif" ersetzt werden: %
```

```

% yc: =  $y^\Delta$  : %
yc:=item(argumentlist,indexof(L,y));
% infxy :=  $(x \wedge y)$  : %
infxy:=inf(x,y);
% infxyc :=  $(x \wedge y^\Delta)$  : %
infxyc:=inf(x,yc);
% s:=  $(x \wedge y) \vee (x \wedge y^\Delta)$  : %
s:=sup(infxy,infxyc);
if equal(s,x) then true else false endif
else true endif),
% Für alle x muss gelten  $x^{\Delta\Delta} \leq x$  : %
forall(x in L,
% argumentlist wird nur bis Index i getestet, also darf der Index von x in L höchstens i sein:
%
if <=(indexof(L, x), i) then
xc:=item(argumentlist,indexof(L,x));
% argumentlist wird nur bis Index i getestet, also darf der Index von xc in L auch
höchstens i sein: %
if <=(indexof(L, xc), i) then
xcc:=item(argumentlist,indexof(L,xc));
if <=(xcc,x) then true else false endif
else true endif
else true endif),
% Für alle x muss gelten  $x \leq y \Rightarrow x^\Delta \geq y^\Delta$  %
forall(x in L, forall(y in L,
% argumentlist wird nur bis Index i getestet, also dürfen die Indices von x und y in L
höchstens i sein: %
if and(<=(indexof(L, x), i),<=(indexof(L, y), i),<=(x,y)) then
xc:=item(argumentlist,indexof(L,x));
yc:=item(argumentlist,indexof(L,y));
if >=(xc,yc) then true else false endif
else true endif)))
endlambda;

```

% findnextlist erhöht argumentlist, d.h. die nächste Liste von Werten für die einstellige Operation wird bestimmt und gegebenenfalls ausgegeben. %
% Der Parameter i ist die Stelle in argumentlist, die aktuell erhöht werden muss. Wenn i=maxindex, dann ist die letzte Stelle in argumentlist dran, d.h. die Operation ist vollständig und kann deshalb ausgegeben werden. Ist i<maxindex, dann ist die Operation nur bis zur Stelle i geprüft, und für jede gültige partielle Definition (d.h. falls isdicomplement(i) true ergibt) muss findnextlist(i+1) durchgeführt werden. %

```
findnextlist:=lambda(i)
```

```
for k in L do
```

```
% Die i-te Stelle von argumentlist durchläuft ganz L: %
```

```
setitem(argumentlist, i, k);
```

```
% Prüfen, ob argumentlist für eine gültige (partielle) Operation steht: %
```

```
if isdicomplement(i) then
```

```
if equal(i,maxindex) then
```

```

    % Die gefundene gültige Operation ist vollständig, also Ausgabe : %
    numfound:=+(numfound,i);
    writeln("");
    write("No. "); writeln(numfound);
    for k1 in L do write(k1); write(" -> "); writeln(item(argumentlist,indexof(L,k1))) endfor;
    true
  else % Die gefundene gültige Operation ist noch nicht vollständig: % findnextlist(+i,i)
endif
  else false endif
endfor
endlambda;

% "Hauptprogramm": %
% L auswählen: %
L:=listprompt("Select a structure", structures);
% Operations- und Relationstabellen für Ordnung vorbereiten: %
prepare(L);
% Der höchste Index von argumentlist ist "Anzahl Elemente in L"-1, da Nummerierung bei 0
beginnt: %
maxindex:=difference(card(L),1);
argumentlist:=emptyset;
numfound:=0;
% argumentlist mit dem Element von L mit kleinstem Index füllen: %
for e in L do additem(argumentlist, item(L,0)) endfor;
% ... und los geht's: %
findnextlist(0);

```