Bevezetés a matematikába

Járai Antal

Ezek a programok csak szemléltetésre szolgálnak

1. Halmazok

```
> restart;
```

1.1. Logikai alapok

1.1.1. Az axiomatikus módszer.

1.1.2. Logikai jelek, predikátumok, formulák.

Logikai jelek:

```
> "¬∧∨⇒⇔⊕|||";
```

"¬∧∨⇒⇔⊕|||" (1.1.2.1)

Kvantorok:

```
> "∃∀";
```

"∃∀" (1.1.2.2)

Formulák generálása:

```
> ppnot := "¬" : ppand := "∧" : ppor := "∨" : ppimply := "⇒" : ppiff := "⇔" :
      ppexist := "∃" : ppforall := "∀" :
> stringseq:=proc() local i,s;
  if nargs=0 then return "" fi; s:=convert(args[1],string);
  for i from 2 to nargs do s:=cat(s,",",args[i]) od; s; end;
```

```
stringseq := proc()
    local i, s;
    if nargs = 0 then
        return ""
    end if;
    s := convert(args[1], string);
    for i from 2 to nargs do
```

(1.1.2.3)

```
        s := cat(s,
        ",", args[i])
    end do;
    s
end proc
```

```
> with(combinat):
  genform:=proc(L::list(list(string)),n::nonnegint)
  local V,VV,i,j,c,a,C,S,SL,SR,sr,sl;
  if nops(L)<=1 then return {} fi; S:={}; V:=L[1];
  if n=0 then VV:=[]; for i from 2 to nops(L) do
    C:=choose(VV,i-2); for c in C do for a in L[i] do
      S:=S union {cat(a,"(",stringseq(op(c)),")")};
    od; od; VV:=[op(VV),op(V)]; od;
  else SL:=genform(L,n-1); C:=choose(V,1);
  for sl in SL do S:=S union {cat(ppnot,sl)}; for c in C do
    S:=S union {cat("(",ppexist,op(c)," ",sl,")"),
    cat("(",ppforall,op(c)," ",sl,")")}
  od; od;
  for j to n do SL:=genform(L,j-1); SR:=genform(L,n-j);
    for sl in SL do for sr in SR do
      S:=S union {cat("(",sl,ppand,sr,")"),cat("(",sl,ppor,
  sr,")"),
      cat("(",sl,ppimply,sr,")"),cat("(",sl,ppiff,sr,")")}
    od; od;
  od; fi; S; end;
```

```
genform := proc(L::(list(list(string))), n::nonnegint)                    (1.1.2.4)
    local V, VV, i, j,
    c, a, C, S, SL, SR, sr, sl;
    if nops(L) <= 1 then
        return {}
    end if;
    S := {};
    V := L[1];
    if n = 0 then
        VV := [];
        for i from 2 to nops(L) do
            C := combinat:-choose(VV, i - 2);
            for c in C do
                for a in L[i] do
                    S := union(S, {cat(a, "(", stringseq(op(c)), ")")})
                end do
```

```
            end do;
            VV := [op(VV), op(V)]
        end do
    else
        SL := genform(L, n − 1);
        C := combinat:-choose(V, 1);
        for sl in SL do
            S := union(S, {cat(ppnot, sl)});
            for c in C do
                S := union(S, {cat("(", ppexist, op(c), " ", sl, ")"), cat("(",
                ppforall, op(c), " ", sl, ")")})
            end do
        end do;
        for j to n do
            SL := genform(L, j − 1);
            SR := genform(L, n − j);
            for sl in SL do
                for sr in SR do
                    S := union(S, {cat("(", sl,
                    ppand, sr, ")"), cat("(", sl, ppor, sr, ")"), cat("(", sl,
                    ppimply, sr, ")"), cat("(", sl, ppiff, sr, ")")})
                end do
            end do
        end do
    end if;
    S
end proc
```

```
> genform([["x","y"],["A","B"],["C","D"],["E","F"]],0);
```

{"A()", "B()", "C(x)", "D(x)", "C(y)", "D(y)", "E(x,x)", "F(x,x)", "E(x,y)", "F(x,y)", "E(y,y)", "F(y,y)"} (1.1.2.5)

```
> genform([["x","y"],[],["P","E"],["I"]],1);
```

{"(P(x)∨P(y))", "(E(y)∧P(x))", "(P(y)∨E(x))", "(E(y)⇔P(x))", "(∃x P(x))", "(P(x)⇔E(y))", "(E(y)⇔P(y))", "(P(x)⇔P(y))", "(P(y)∧P(y))", "(P(x)⇔E(x))", "(P(y)∧P(x))", "(E(x)⇔E(x))", "(P(x)∨P(x))", "(P(y)∨E(y))", "(E(y)∨P(x))", "(P(y)⇔E(y))", "(E(x)∨E(x))", "(P(x)∧P(x))", "(E(y)∧E(y))", "(E(x)⇔P(x))", "(P(y)∨P(x))", (1.1.2.6)

"(E(x)∧P(y))", "(∃x P(y))", "(E(y)∨E(x))", "(P(x)∨E(y))", "(E(y)⇔E(y))",
"(P(x)⇔P(x))", "(E(y)∨E(y))", "(P(y)⇔E(x))", "(P(y)⇔P(y))", "(∃x E(x))",
"(P(x)∧E(y))", "(∃x E(y))", "(E(x)⇔E(y))", "(E(x)∨E(y))", "(P(y)∨P(y))",
"(E(x)⇔P(y))", "(P(x)∧P(y))", "(E(x)∨P(y))", "(E(x)∨P(x))",
"(P(y)∧E(y))", "(∃y P(y))", "(E(y)∧P(y))", "(P(x)∨E(x))", "(∃y P(x))",
"(E(x)∧E(y))", "(E(y)⇔E(x))", "(E(x)∧E(x))", "(E(y)∧E(x))",
"(P(x)⇒I(x,x))", "(P(x)⇒I(x,y))", "(P(x)⇒I(y,y))", "(E(x)⇒I(x,x))",
"(E(x)⇒I(x,y))", "(E(x)⇒I(y,y))", "(P(y)⇒I(x,x))", "(P(y)⇒I(x,y))",
"(P(y)⇒I(y,y))", "(E(y)⇒I(x,x))", "(E(y)⇒I(x,y))", "(E(y)⇒I(y,y))",
"(I(x,x)⇒P(x))", "(I(x,x)⇒E(x))", "(I(x,x)⇒P(y))", "(I(x,x)⇒E(y))",
"(I(x,x)⇒I(x,x))", "(I(x,x)⇒I(x,y))", "(I(x,x)⇒I(y,y))", "(I(x,y)⇒P(x))",
"(I(x,y)⇒E(x))", "(I(x,y)⇒P(y))", "(I(x,y)⇒E(y))", "(I(x,y)⇒I(x,x))",
"(I(x,y)⇒I(x,y))", "(I(x,y)⇒I(y,y))", "(I(y,y)⇒P(x))", "(P(y)⇔P(x))",
"(∃y E(x))", "(E(y)∨P(y))", "(E(x)∧P(x))", "(P(y)∧E(x))", "(P(x)∧E(x))",
"(∃y E(y))", "(∀x P(x))", "(∀y P(x))", "(∀x E(x))", "(∀y E(x))",
"(∀x P(y))", "(∀y P(y))", "(∀x E(y))", "(∀y E(y))", "(∃x I(x,x))",
"(∀x I(x,x))", "(∃y I(x,x))", "(∀y I(x,x))", "(∃x I(x,y))", "(∀x I(x,y))",
"(∃y I(x,y))", "(∀y I(x,y))", "(∃x I(y,y))", "(∀x I(y,y))", "(∃y I(y,y))",
"(∀y I(y,y))", "(P(x)⇒P(x))", "(P(x)⇒E(x))", "(P(x)⇒P(y))",
"(P(x)⇒E(y))", "(P(x)∧I(x,x))", "(P(x)∨I(x,x))", "(P(x)⇔I(x,x))",
"(P(x)∧I(x,y))", "(P(x)∨I(x,y))", "(P(x)⇔I(x,y))", "(P(x)∧I(y,y))",
"(P(x)∨I(y,y))", "¬P(x)", "¬E(x)", "¬P(y)", "¬E(y)", "¬I(x,x)", "¬I(x,y)",
"¬I(y,y)", "(P(x)⇔I(y,y))", "(E(x)⇒P(x))", "(E(x)⇒E(x))", "(E(x)⇒P(y))",
"(E(x)⇒E(y))", "(E(x)∧I(x,x))", "(E(x)∨I(x,x))", "(E(x)⇔I(x,x))",
"(E(x)∧I(x,y))", "(E(x)∨I(x,y))", "(E(x)⇔I(x,y))", "(E(x)∧I(y,y))",
"(E(x)∨I(y,y))", "(E(x)⇔I(y,y))", "(P(y)⇒P(x))", "(P(y)⇒E(x))",
"(P(y)⇒P(y))", "(P(y)⇒E(y))", "(P(y)∧I(x,x))", "(P(y)∨I(x,x))",
"(P(y)⇔I(x,x))", "(P(y)∧I(x,y))", "(P(y)∨I(x,y))", "(P(y)⇔I(x,y))",
"(P(y)∧I(y,y))", "(P(y)∨I(y,y))", "(P(y)⇔I(y,y))", "(E(y)⇒P(x))",
"(E(y)⇒E(x))", "(E(y)⇒P(y))", "(E(y)⇒E(y))", "(E(y)∧I(x,x))",
"(E(y)∨I(x,x))", "(E(y)⇔I(x,x))", "(E(y)∧I(x,y))", "(E(y)∨I(x,y))",
"(E(y)⇔I(x,y))", "(E(y)∧I(y,y))", "(E(y)∨I(y,y))", "(E(y)⇔I(y,y))",
"(I(x,x)∧P(x))", "(I(x,x)∨P(x))", "(I(x,x)⇔P(x))", "(I(x,x)∧E(x))",
"(I(x,x)∨E(x))", "(I(x,x)⇔E(x))", "(I(x,x)∧P(y))", "(I(x,x)∨P(y))",
"(I(x,x)⇔P(y))", "(I(x,x)∧E(y))", "(I(x,x)∨E(y))", "(I(x,x)⇔E(y))",
"(I(x,x)∧I(x,x))", "(I(x,x)∨I(x,x))", "(I(x,x)⇔I(x,x))", "(I(x,x)∧I(x,y))",
"(I(x,x)∨I(x,y))", "(I(x,x)⇔I(x,y))", "(I(x,x)∧I(y,y))", "(I(x,x)∨I(y,y))",
"(I(x,x)⇔I(y,y))", "(I(x,y)∧P(x))", "(I(x,y)∨P(x))", "(I(x,y)⇔P(x))",

"(I(x,y)∧E(x))", "(I(x,y)∨E(x))", "(I(x,y)⇔E(x))", "(I(x,y)∧P(y))",
"(I(x,y)∨P(y))", "(I(x,y)⇔P(y))", "(I(x,y)∧E(y))", "(I(x,y)∨E(y))",
"(I(x,y)⇔E(y))", "(I(x,y)∧I(x,x))", "(I(x,y)∨I(x,x))", "(I(x,y)⇔I(x,x))",
"(I(x,y)∧I(x,y))", "(I(x,y)∨I(x,y))", "(I(x,y)⇔I(x,y))", "(I(x,y)∧I(y,y))",
"(I(x,y)∨I(y,y))", "(I(x,y)⇔I(y,y))", "(I(y,y)∧P(x))", "(I(y,y)∨P(x))",
"(I(y,y)⇔P(x))", "(I(y,y)∧E(x))", "(I(y,y)∨E(x))", "(I(y,y)⇔E(x))",
"(I(y,y)∧P(y))", "(I(y,y)∨P(y))", "(I(y,y)⇔P(y))", "(I(y,y)∧E(y))",
"(I(y,y)∨E(y))", "(I(y,y)⇔E(y))", "(I(y,y)∧I(x,x))", "(I(y,y)∨I(x,x))",
"(I(y,y)⇔I(x,x))", "(I(y,y)∧I(x,y))", "(I(y,y)∨I(x,y))", "(I(y,y)⇔I(x,y))",
"(I(y,y)∧I(y,y))", "(I(y,y)∨I(y,y))", "(I(y,y)⇔I(y,y))", "(I(y,y)⇒E(x))",
"(I(y,y)⇒P(y))", "(I(y,y)⇒E(y))", "(I(y,y)⇒I(x,x))", "(I(y,y)⇒I(x,y))",
"(I(y,y)⇒I(y,y))"}

```
> genform([["x","y"],[],["P"],["I"]],2):
```

Az alábbi parser program egy sztring elemzését végzi, hogy az érvényes formula–e? Ha igen, akkor a true értéket és párok egy listáját, és a maradék sztringet adja vissza. A párok második koordinátája a megtalált szintaktikai alapegység. Az első koordináta az alapegység tipusa, az alábbiak szerint:

k	kvantor
l	logikai jel
p	zárójel
c	vessző
f	szabad változó
b	kötött változó
0,1,...	predikátum, adott számú változóval
?	predikátum, még ismeretlen számú változóval

Hiba esetén false értéket kapunk, a lista és a maradék sztring pedig utal a hiba helyére.

A változók és a predikátumok neve betűvel kell kezdődjön és betűket és számjegyeket tartalmazhat. Elválasztó jelként tetszőleges "whitespace" karakterekből álló sorozat használható.

Az egyszerűbb eljárásokkal kezdjük:

A parseparentheses eljárás egy kezdő zárójellel kezdődő sztringben megkeresi az ehhez tartozó záró zárójelet, és ennek indexét adja vissza. Ha sikertelen, akkor nulla az eredmény.

A parsename eljárás egy sztringet szétvág egy névre és egy maradékra.

A parsevarsec eljárás vesszőkel elválasztott változók sorozatát ismeri fel.

A parsepredicate eljárás egy predikátumot ismer fel.

A főprogram a parsesentence eljárás.

```
> parseparentheses:=proc(s::string) local n,j;
  if s="" then return 0 fi; n:=0;
  for j to length(s) do
    if s[j]="(" then n:=n+1 fi;
    if s[j]=")" then n:=n-1; fi;
    if n=0 then return j fi;
  od; 0; end;
```

```
parseparentheses := proc(s::string)                                   (1.1.2.7)
    local n, j;
    if s = "" then
        return 0
    end if;
    n := 0;
    for j to length(s) do
        if s[j] = "(" then
            n := n + 1
        end if;
        if s[j] = ")" then
            n := n - 1
        end if;
        if n = 0 then
            return j
        end if
    end do;
    0
end proc
```

```
> with(StringTools):
```

```
> parsename:=proc(s::string) local ls,rs;
  rs:=TrimLeft(s);
  if length(rs)=0 then return "","" fi;
  if IsAlpha(rs[1]) then ls:=rs[1]; rs:=Drop(rs,1) else
```

```
return "",rs fi;
while length(rs)>0 do
  if IsAlphaNumeric(rs[1]) then
    ls:=cat(ls,rs[1]); rs:=Drop(rs,1);
  else return ls,rs fi;
od; ls,rs end;
```

parsename := **proc**(*s*::*string*) (1.1.2.8)

local *ls*, *rs*;
rs := *StringTools*:-*TrimLeft*(*s*);
if *length*(*rs*) = 0 **then**
return "", ""
end if;
if *StringTools*:-*IsAlpha*(*rs*[1]) **then**
ls := *rs*[1];
rs := *StringTools*:-*Drop*(*rs*, 1)
else
return "", *rs*
end if;
while 0 < *length*(*rs*) **do**
if *StringTools*:-*IsAlphaNumeric*(*rs*[1]) **then**
ls := *cat*(*ls*, *rs*[1]);
rs := *StringTools*:-*Drop*(*rs*, 1)
else
return *ls*, *rs*
end if
end do;
ls, *rs*
end proc

```
> parsevarseq:=proc(s::string) local L,rs,x;
rs:=TrimLeft(s); L:=[]; if rs="" then return true,L,"" fi;
while true do
  x:=parsename(rs);
  if x[1]="" then return false,L,x[2] fi;
  L:=[op(L),["f",x[1]]];
  rs:=TrimLeft(x[2]); if rs="" then return true,L,"" fi;
  if rs[1]<>"," then return false,L,rs fi;
  L:=[op(L),["c",","]];
  rs:=TrimLeft(Drop(rs,1));
  if rs="" then return false,L,"" fi;
od; end;
```

(1.1.2.9)

```
parsevarseq := proc(s::string)                                    (1.1.2.9)
    local L, rs, x;
    rs := StringTools:-TrimLeft(s);
    L := [];
    if rs = "" then
        return true,
       L, ""
    end if;
    do
        x := parsename(rs);
        if x[1] = "" then
            return false, L, x[2]
        end if;
        L := [op(L), ["f", x[1]]];
        rs := StringTools:-TrimLeft(x[2]);
        if rs = "" then
            return true,
           L, ""
        end if;
        if rs[1] <> "," then
            return false, L, rs
        end if;
        L := [op(L), ["c", ","]];
        rs := StringTools:-TrimLeft(StringTools:-Drop(rs, 1));
        if rs = "" then
            return false, L, ""
        end if
    end do
end proc
```

```
> parsepredicate:=proc(s::string) local L,x,ls,rs,j;
  x:=parsename(s);ls:=x[1];
  if ls="" then return false,[],x[2] fi; rs:=x[2];
  if rs="" then return false,["?",ls],"" fi;
  if rs[1]<>"(" then return false,["?",ls],rs fi;
  j:=parseparentheses(rs);
  if j=0 then return false,["?",ls],rs fi;
  x:=parsevarseq(rs[2..j-1]);
  if x[1] then
    if x[2]=[] then
      true,[[0,ls],["p","("],["p",")"]],Drop(rs,j)
```

```
      else
        true,[[(nops(x[2])+1)/2,ls],["p","("],op(x[2]),["p",")
  "]],
          Drop(rs,j)
      fi;
    else x[1],x[2],cat(x[3],Drop(rs,j-1)) fi; end;
```

$$parsepredicate := \textbf{proc}(s::string) \tag{1.1.2.10}$$

local *L*, *x*, *ls*, *rs*, *j*;
$x := parsename(s)$;
$ls := x[1]$;
if $ls = ""$ **then**
  return *false*, [ ], $x[2]$
end if;
$rs := x[2]$;
if $rs = ""$ **then**
  return *false*, ["?", *ls*], ""
end if;
if $rs[1] <> "("$ **then**
  return *false*, ["?", *ls*], *rs*
end if;
$j := parseparentheses(rs)$;
if $j = 0$ **then**
  return *false*, ["?", *ls*], *rs*
end if;
$x := parsevarseq(rs[2 .. j - 1])$;
if $x[1]$ **then**
  if $x[2] = [\,]$ **then**
    true, [[0, *ls*], ["p", "("], ["p", ")"]],
    StringTools:-Drop(*rs*, *j*)
  else
    true,
    $[[1/2 * nops(x[2]) + 1/2, ls]$, ["p", "("], $op(x[2])$, ["p",
    ")"]], *StringTools:-Drop*(*rs*, *j*)
  end if
else
  $x[1]$, $x[2]$,
  cat($x[3]$, *StringTools:-Drop*(*rs*, $j - 1$))
end if
end proc

```
> parsesentence:=proc(s::string) local ls,rs,L,x,j,n;
    global ppnot,ppand,ppor,ppimply,ppiff,ppexist,ppforall;
  rs:=TrimLeft(s); if rs="" then return false,[],"" fi;
  if IsPrefix(ppnot,rs) then
    rs:=Drop(rs,length(ppnot));
    x:=parsesentence(rs);
    x[1],[["l",ppnot],op(x[2])],x[3]
  elif rs[1]<>"(" then
    parsepredicate(rs)
  else
    L:=[["p","("]]; j:=parseparentheses(rs);
    if j=0 then return false,L,rs[2..-1] fi;
    ls:=TrimLeft(rs[2..j-1]); rs:=Drop(rs,j-1);
    if IsPrefix(ppexist,ls) or IsPrefix(ppforall,ls) then
      if IsPrefix(ppexist,ls) then
        L:=[op(L),["k",ppexist]]; ls:=Drop(ls,length(ppexist)
  );
      else
        L:=[op(L),["k",ppforall]]; ls:=Drop(ls,length
  (ppforall));
      fi;
      x:=parsename(ls); if x[1]="" then return false,L,cat(x
  [2],rs) fi;
      n:=x[1];L:=[op(L),["b",n]];
      x:=parsesentence(x[2]);
      if not x[1] or x[3]<>"" then
        return false,[op(L),op(x[2])],cat(x[3],rs) fi;
      for j in x[2] do
        if j[1]="f" and j[2]=n then
          L:=[op(L),["b",n]] else
          L:=[op(L),j] fi; od;
      L:=[op(L),["p",")"]]; rs:=TrimLeft(Drop(rs,1));
      true,L,rs
    else
      x:=parsesentence(ls);
      if not x[1] then return false,x[2],cat(x[3],rs) fi;
      L:=[op(L),op(x[2])]; ls:=TrimLeft(x[3]);
      if IsPrefix(ppand,ls) or IsPrefix(ppor,ls)
          or IsPrefix(ppimply,ls) or IsPrefix(ppiff,ls) then
        if IsPrefix(ppand,ls) then
          L:=[op(L),["l",ppand]];ls:=Drop(ls,length(ppand))
        elif IsPrefix(ppor,ls) then
          L:=[op(L),["l",ppor]];ls:=Drop(ls,length(ppor))
        elif IsPrefix(ppor,ls) then
          L:=[op(L),["l",ppor]];ls:=Drop(ls,length(ppor))
        else
          L:=[op(L),["l",ppor]];ls:=Drop(ls,length(ppor))
        fi;
```

```
        x:=parsesentence(ls);L:=[op(L),op(x[2])];
        if not x[1] or x[3]<>"" then
          return false,L,cat(x[3],rs) fi;
        L:=[op(L),["p",")"]]; rs:=TrimLeft(Drop(rs,1));
        true,L,rs
      else
        false,L,cat(ls,rs)
      fi;
    fi;
  fi; end;
```

parsesentence := **proc**(*s*::*string*) (1.1.2.11)

```
    local ls, rs, L, x, j, n;
    global ppnot,
    ppand, ppor, ppimply, ppiff, ppexist, ppforall;
    rs := StringTools:-TrimLeft(s);
    if rs = "" then
        return false, [ ], ""
    end if;
    if StringTools:-IsPrefix(ppnot, rs) then
        rs := StringTools:-Drop(rs, length(ppnot));
        x := parsesentence(rs);
        x[1], [["l", ppnot], op(x[2])], x[3]
    elif rs[1] <> "(" then
        parsepredicate(rs)
    else
        L := [["p", "("]];
        j := parseparentheses(rs);
        if j = 0 then
            return false, L,
            rs[2 .. -1]
        end if;
        ls := StringTools:-TrimLeft(rs[2 .. j - 1]);
        rs := StringTools:-Drop(rs, j - 1);
        if StringTools:-IsPrefix(ppexist,
        ls) or StringTools:-IsPrefix(ppforall, ls) then
            if StringTools:-IsPrefix(ppexist, ls) then
                L := [op(L), ["k",
                ppexist]];
                ls := StringTools:-Drop(ls, length(ppexist))
            else
```

```
            L := [op(L), ["k", ppforall]];
            ls := StringTools:-Drop(ls, length(ppforall))
        end if;
        x := parsename(ls);
        if x[1] = "" then
            return false, L,
            cat(x[2], rs)
        end if;
        n := x[1];
        L := [op(L), ["b", n]];
        x := parsesentence(x[2]);
        if not x[1] or x[3] <> "" then
            return false, [op(L), op(x[2])], cat(x[3], rs)
        end if;
        for j in x[2] do
            if j[1] = "f" and j[2] = n then
                L := [op(L),
                ["b", n]]
            else
                L := [op(L), j]
            end if
        end do;
        L := [op(L), ["p", ")"]];
        rs := StringTools:-TrimLeft(StringTools:-Drop(rs, 1));
        true, L, rs
    else
        x := parsesentence(ls);
        if not x[1] then
            return false, x[2], cat(x[3], rs)
        end if;
        L := [op(L),
        op(x[2])];
        ls := StringTools:-TrimLeft(x[3]);
        if StringTools:-IsPrefix(ppand,
        ls) or StringTools:-IsPrefix(ppor,
        ls) or StringTools:-IsPrefix(ppimply,
        ls) or StringTools:-IsPrefix(ppiff, ls) then
            if StringTools:-IsPrefix(ppand, ls) then
```

```
                L := [op(L), ["l",
               ppand]];
                ls := StringTools:-Drop(ls,
               length(ppand))
            elif StringTools:-IsPrefix(ppor,
           ls) then
                L := [op(L), ["l", ppor]];
                ls := StringTools:-Drop(ls, length(ppor))
            elif StringTools:-IsPrefix(ppor, ls) then
                L := [op(L), ["l", ppor]];
                ls := StringTools:-Drop(ls,
               length(ppor))
            else
                L := [op(L), ["l", ppor]];
                ls := StringTools:-Drop(ls, length(ppor))
            end if;
            x := parsesentence(ls);
            L := [op(L), op(x[2])];
            if not x[1] or x[3] <> "" then
                return false, L,
               cat(x[3], rs)
            end if;
            L := [op(L), ["p", ")"]];
            rs := StringTools:-TrimLeft(StringTools:-Drop(rs, 1));
            true, L, rs
        else
            false, L, cat(ls, rs)
        end if
    end if
  end if
end proc
```

```
> parsesentence("I(x,y)");
  parsesentence("∀x I(x,x)");
  parsesentence("(∀x I(x,x))");
  parsesentence("(∀x(∀y(∃z(I(x,z)∧I(y,z)))))");
  parsesentence("((E(x)∧E(y))∧¬I(x,y))");
```

true, [[2, "I"], ["p", "("], ["f", "x"], ["c", ","], ["f", "y"], ["p", ")"]], ""

false, [], "∀x I(x,x)"

true, [["p", "("], ["k", "∀"], ["b", "x"], [2, "I"], ["p", "("], ["b", "x"], ["c",

",", ["b", "x"], ["p", ")"], ["p", ")"]], ""
true, [["p", "("], ["k", "∀"], ["b", "x"], ["p", "("], ["k", "∀"], ["b", "y"], ["p", "("], ["k", "∃"], ["b", "z"], ["p", "("], [2, "I"], ["p", "("], ["b", "x"], ["c", ","], ["b", "z"], ["p", ")"], ["l", "∧"], [2, "I"], ["p", "("], ["b", "y"], ["c", ","], ["b", "z"], ["p", ")"], ["p", ")"], ["p", ")"], ["p", ")"], ["p", ")"]], ""
true, [["p", "("], ["p", "("], [1, "E"], ["p", "("], ["f", "x"], ["p", ")"], ["l", "∧"], [1, "E"], ["p", "("], ["f", "y"], ["p", ")"], ["p", ")"], ["l", "∧"], ["l", "¬"], [2, "I"], ["p", "("], ["f", "x"], ["c", ","], ["f", "y"], ["p", ")"], ["p", ")"]], "" (1.1.2.12)

▶ ***1.1.3. Megjegyzés.***

▶ ***1.1.4. Példa.***

▶ ***1.1.5. Példa.***

▶ ***1.1.6. Példa.***

▶ ***1.1.7. Matematikai elméletek.***

▶ ***1.1.8. Matematikai logika.***

▶ ***1.1.9. Egyenlőség.***

▶ ***1.1.10. Nyelvek műveletekkel.***

▶ ***1.1.11. Az axiómák jelentése.***

▶ ****1.1.12. Az axiómák kiválasztása.***

▶ ***–>1.1.13. Feladat.***

▶ ***–>1.1.14. Feladat.***

▶ ***–>1.1.15. Feladat.***

▶ ***–>1.1.16. Feladat.***

▶ ***1.1.17. Feladat.***

▶ ***1.1.18. Feladat.***

▶ ***–>1.1.19. Feladat.***

▶ ***1.1.20. Feladat.***

▶ ***–>1.1.21. Feladat.***

▶ ***–>1.1.22. Feladat.***

▶ ***1.1.23. További feladatok.***

▼ 1.2. Halmazelméleti alapfogalmak

```
> restart;
```

1.2.1. Halmazelmélet.

```
> A:={a,b,c}; member(b,A); member(d,A); b in A; evalb(%); d
  in A; evalb(%);
```

$$A:=\{b, c, a\}$$
$$true$$
$$false$$
$$b\in(\{b, c, a\})$$
$$true$$
$$d\in(\{b, c, a\})$$
$$false \qquad (1.2.1.1)$$

```
> whattype(A); whattype(a); whattype(2); whattype
  (krikszkraksz); whattype("krikszkraksz");
```

$$set$$
$$symbol$$
$$integer$$
$$symbol$$
$$string \qquad (1.2.1.2)$$

1.2.2. Meghatározottság.

```
> {a,a,b,b,a}; {b,a};
```

$$\{b, a\}$$
$$\{b, a\} \qquad (1.2.2.1)$$

* 1.2.3. Meghatározottsági axióma.

1.2.4. Részhalmazok.

```
> {a,b} subset {a,b,d};
```

$$true \qquad (1.2.4.1)$$

```
> {a,c} subset {a,b,d};
```

$$false \qquad (1.2.4.2)$$

```
> {a,b} subset {a,b};
```

$$true \qquad (1.2.4.3)$$

```
> X subset X; X subset Y;
```

$$true$$
$$X\subseteq Y \qquad (1.2.4.4)$$

```
> A:={1,2,3,4,5,6,7}; select(x->isprime(x),A);
```

$$A := \{1, 2, 3, 4, 5, 6, 7\}$$
$$\{2, 3, 5, 7\} \quad (1.2.4.5)$$

* 1.2.5. Részhalmaz-axióma.

* 1.2.6. Tétel.

* 1.2.7. Megjegyzés.

1.2.8. Néhány egyszerű halmaz.

```
> {}; {a,b}; {a,a}; {a,b,c};
```

$$\{\ \}$$
$$\{b, a\}$$
$$\{a\}$$
$$\{b, c, a\} \quad (1.2.8.1)$$

```
> a:=b; {a,b};
```

$$a := b$$
$$\{b\} \quad (1.2.8.2)$$

```
> a:='a'; {a,b};
```

$$a := a$$
$$\{b, a\} \quad (1.2.8.3)$$

```
> {} subset X;
```

$$true \quad (1.2.8.4)$$

1.2.9. Az üres halmaz axiómája.

1.2.10. Páraxióma.

1.2.11. Unió.

> $\{a, b\} \bigcup \{b, c\}$; $\{a, b\}$ *union* $\{b, c\}$;

$$\{b, c, a\}$$
$$\{b, c, a\} \quad (1.2.11.1)$$

> $\mathcal{A} := \{\{a\}, \{b, c\}, \{1, 2, b\}, \{\}, \{a, b\}\}$; $op(\mathcal{A})$; `` `union`(op(\mathcal{A})) ``;
> `` `union`(\{a\}, \{b, c\}, \{1, 2, b\}, \{\}, \{a, b\}) ``;

$$\mathcal{A} := \{\{\}, \{b, a\}, \{a\}, \{1, 2, b\}, \{b, c\}\}$$
$$\{\}, \{b, a\}, \{a\}, \{1, 2, b\}, \{b, c\}$$
$$\{1, 2, b, c, a\}$$
$$\{1, 2, b, c, a\} \quad (1.2.11.2)$$

```
> {a,b} union {b,c}; `union`({a},{b,c},{1,2,b},{},{a,b});
```

$$\{b, c, a\}$$

(1.2.11.3)

$$\{1, 2, b, c, a\} \quad (1.2.11.3)$$

```
> A:={a,b}; B:={b,c}; `union`(A,B);
```

$$A := \{b, a\}$$
$$B := \{b, c\}$$
$$\{b, c, a\} \quad (1.2.11.4)$$

```
> `union`();
```

$$\{\ \} \quad (1.2.11.5)$$

* 1.2.12. *Unióaxióma.*

1.2.13. Állítás: az unió tulajdonságai.

```
> X union {};
```

$$X \quad (1.2.13.1)$$

```
> X union Y; Y union X;
```

$$X \cup Y$$
$$X \cup Y \quad (1.2.13.2)$$

```
> (X union Y) union Z; X union (Y union Z);
```

$$union(X, Y, Z)$$
$$union(X, Y, Z) \quad (1.2.13.3)$$

```
> X union X;
```

$$X \quad (1.2.13.4)$$

1.2.14. Metszet.

```
> {a,b,1} intersect {a,c,2,1};
```

$$\{1, a\} \quad (1.2.14.1)$$

```
> `intersect`({a,b,c,d},{a,b,c,1},{a,b,1,2});
```

$$\{b, a\} \quad (1.2.14.2)$$

```
> A:={a,b}; B:={b,c}; C:={c,a}; A intersect B; B intersect C;
  C intersect A; `intersect`(A,B); `intersect`(A,B,C);
```

$$A := \{b, a\}$$
$$B := \{b, c\}$$
$$C := \{c, a\}$$
$$\{b\}$$
$$\{c\}$$
$$\{a\}$$
$$\{b\}$$
$$\{\ \} \quad (1.2.14.3)$$

1.2.15. Állítás: a metszet tulajdonságai.

Az első négy tulajdonságot a Maple is ismeri:

```
> X intersect {};
```

$$\{\ \}\tag{1.2.15.1}$$

```
> X intersect Y; Y intersect X;
```

$$X \cap Y$$

$$X \cap Y\tag{1.2.15.2}$$

```
> X intersect (Y intersect Z); (X intersect Y) intersect Z;
```

$$intersect(X, Y, Z)$$

$$intersect(X, Y, Z)\tag{1.2.15.3}$$

```
> X intersect X;
```

$$X\tag{1.2.15.4}$$

->1.2.16. Feladat.

->1.2.17. Feladat.

->1.2.18. Feladat.

->1.2.19. Feladat.

1.1.20. Feladat.

1.2.21. Feladat.

1.2.22. Állítás: disztributivitási szabályok.

```
> X intersect (Y union Z); expand(%);
```

$$(Y \cup Z) \cap X$$

$$X \cap Y \cup X \cap Z\tag{1.2.22.1}$$

```
> X union (Y intersect Z); A:={a,b,c}; B:={b,c,d}; C:={c,d,e}
; A union (B intersect C); (A union B) intersect (A union
C);
```

$$X \cup Y \cap Z$$

$$A := \{b, c, a\}$$

$$B := \{b, c, d\}$$

$$C := \{c, d, e\}$$

$$\{b, c, a, d\}$$

$$\{b, c, a, d\}\tag{1.2.22.2}$$

->1.2.23. Feladat.

1.2.24. *Különbség és komplementer.*

```
> A:={a,b}; B:={b,c}; C:={a,b,c,d}; A minus B;
  symmdiff(A,B); symmdiff(A,B,C);
```

$$A := \{b, a\}$$
$$B := \{b, c\}$$
$$C := \{b, c, a, d\}$$
$$\{a\}$$
$$\{c, a\}$$
$$\{b, d\} \quad (1.2.24.1)$$

1.2.25. *Állítás.*

```
> A; C minus (C minus A);
```

$$\{b, a\}$$
$$\{b, a\} \quad (1.2.25.1)$$

```
> C; C minus {};
```

$$\{b, c, a, d\}$$
$$\{b, c, a, d\} \quad (1.2.25.2)$$

```
> {}, C minus C;
```

$$\{\}, \{\} \quad (1.2.25.3)$$

```
> {}; A intersect (C minus A);
```

$$\{\ \}$$
$$\{\ \} \quad (1.2.25.4)$$

```
> C; A union (C minus A);
```

$$\{b, c, a, d\}$$
$$\{b, c, a, d\} \quad (1.2.25.5)$$

```
> B:={a,b,d}; A; C minus B; C minus A;
```

$$B := \{b, a, d\}$$
$$\{b, a\}$$
$$\{c\}$$
$$\{c, d\} \quad (1.2.25.6)$$

```
> A; B:={b,c}; C minus (A union B); (C minus A) intersect (C
  minus B);
```

$$\{b, a\}$$
$$B := \{b, c\}$$
$$\{d\}$$

(1.2.25.7)

$\{d\}$ (1.2.25.7)

```
> C minus (A intersect B); (C minus A) union (C minus B);
```

$\{c, a, d\}$
$\{c, a, d\}$ (1.2.25.8)

->1.2.26. Feladat.
->1.2.27. Feladat.
->1.2.28. Feladat.
->1.2.29. Feladat.
->1.2.30. Feladat.
1.2.31. Feladat.
1.2.32. Feladat.
1.2.33. Feladat.
->1.2.34. Feladat.
->1.2.35. Feladat.
1.2.36. Hatványhalmaz.

```
> with(combinat,powerset): powerset({a,b,c}); powerset({a,b})
  ;
  powerset({a}); powerset({});
```

$\{\{\}, \{b, c, a\}, \{b, a\}, \{c, a\}, \{b\}, \{c\}, \{a\}, \{b, c\}\}$
$\{\{\}, \{b, a\}, \{b\}, \{a\}\}$
$\{\{\}, \{a\}\}$
$\{\{\ \}\}$ (1.2.36.1)

* 1.2.37. Hatványhalmaz-axióma.
->1.2.38. Feladat.
->1.2.39. Feladat.
1.2.40. Feladat.
->1.2.41. Feladat.
* 1.2.42. Végtelenségi axióma.

1.2.43. Megjegyzés.
1.2.44. További feladatok részletes megoldással.
1.2.45. További feladatok.

1.3. Relációk

1.3.1. Rendezett pár.

A rendezett pár a Maple-ben [x,y].

```
> evalb({{x},{x,y}}={{y},{x,y}});evalb([x,y]=[y,x]);
```

$$false$$
$$false \quad (1.3.1.1)$$

```
> p:=[x,y]; p[1]; p[2];
```

$$p := [x, y]$$
$$x$$
$$y \quad (1.3.1.2)$$

```
> `type/ordpair`:=proc(x) type(x,list) and nops(x)=2 end;

  type([a,b],ordpair); type([1,2,3],ordpair);
```

$$type/ordpair := \mathbf{proc}(x)\ type(x, list)\ \mathbf{and}\ nops(x) = 2\ \mathbf{end\ proc}$$
$$true$$
$$false \quad (1.3.1.3)$$

->1.3.2. Feladat.

1.3.3. Descartes-szorzat.

```
> bincartprod:=proc(X::set,Y::set) local x,y,Z; Z:={};
  for x in X do for y in Y do Z:=Z union {[x,y]}; od; od; Z;
  end;
```

(1.3.3.1)

```
bincartprod := proc(X::set, Y::set)
    local x, y, Z;
    Z := {};
    for x in X do
        for y in Y do
            Z := union(Z, {[x, y]})
        end do
```

```
    end do;
    Z
end proc
```

```
> bincartprod({1,2,3},{a,b});
```

$$\{[1, b], [1, a], [2, b], [2, a], [3, b], [3, a]\} \quad (1.3.3.2)$$

▶ *->1.3.4. Feladat.*

▶ *->1.3.5. Feladat.*

▶ *1.3.6. Feladat.*

▶ *1.3.7. Feladat.*

▼ *1.3.8. Binér relációk.*

```
> binrel:=proc(R::set(ordpair),X::set,Y::set) local r;
  if nargs<1 or nargs>3 then error ": needs 1..3 arguments"
  fi;
  for r in R do
    if nargs=2 and not (member(r[1],X) and member(r[2],X))
  then
      return false;
    elif nargs=3 and not (member(r[1],X) and member(r[2],Y))
  then
      return false
    fi;
  od; true; end;
```

(1.3.8.1)

```
binrel := proc(R::(set(ordpair)), X::set, Y::set)
    local r;
    if nargs < 1 or 3 < nargs then
        error ": needs 1..3 arguments"
    end if;
    for r in R do
        if nargs = 2 and not (member(r[1],
        X) and member(r[2], X)) then
            return false
        elif nargs = 3 and not (member(r[1],
        X) and member(r[2], Y)) then
            return false
        end if
    end do;
    true
```

end proc

```
> binrel({});
  R:=bincartprod({1,2,3},{a,b});binrel(R);binrel(R,{0,1,2,3},
  {a,b,c});
  binrel(R,{a,b},{1,2,3});binrel(R,{1,2,3,a,b});
```

$$true$$
$$R := \{[1, b], [1, a], [2, b], [2, a], [3, b], [3, a]\}$$
$$true$$
$$true$$
$$false$$
$$true \quad (1.3.8.2)$$

```
> id:=proc(X) local x; map(x->[x,x],X); end; id({1,3,a});
```

id := **proc**(*X*)
  local *x*;
  map(**proc**(*x*)
    option *operator*, *arrow*;
    [*x*, *x*]
  end proc, *X*)
end proc

$$\{[1, 1], [3, 3], [a, a]\} \quad (1.3.8.3)$$

```
> F:={{1},{2},{1,2}}; FF:=bincartprod(F,F); select(x->x[1]
  subset x[2],FF);select(x->x[1] subset x[2] and not x[1]=x
  [2],FF);
```

$$F := \{\{1\}, \{2\}, \{1, 2\}\}$$
$$FF := \{[\{2\}, \{1\}], [\{2\}, \{2\}], [\{2\}, \{1, 2\}], [\{1, 2\}, \{1\}], [\{1, 2\}, \{2\}], [\{1, 2\}, \{1, 2\}], [\{1\}, \{1\}], [\{1\}, \{2\}], [\{1\}, \{1, 2\}]\}$$
$$\{[\{2\}, \{2\}], [\{2\}, \{1, 2\}], [\{1, 2\}, \{1, 2\}], [\{1\}, \{1\}], [\{1\}, \{1, 2\}]\}$$
$$\{[\{2\}, \{1, 2\}], [\{1\}, \{1, 2\}]\} \quad (1.3.8.4)$$

```
> irem(13,5); X:={1,2,3,4,5,6};XX:=bincartprod(X,X):
  R:=select(x->irem(x[2],x[1])=0,XX);
```

$$3$$
$$X := \{1, 2, 3, 4, 5, 6\}$$
$$R := \{[1, 2], [1, 1], [3, 3], [5, 5], [6, 6], [1, 3], [1, 4], [1, 5], [1, 6], [2, 2], [2, 4], [2, 6], [3, 6], [4, 4]\} \quad (1.3.8.5)$$

1.3.9. Feladat.

1.3.10. Feladat.

1.3.11. Feladat.

1.3.12. Feladat.

1.3.13. Feladat.

1.3.14. Relációk gráfja.

1.3.15. Értelmezési tartomány, értékkészlet.

```
> R:={[1,a],[1,b],[2,b],[3,d],[2,d],[4,e]};
  dmn:=proc(R::set(ordpair)) map(x->x[1],R); end; dmn(R);
  rng:=proc(R::set(ordpair)) map(x->x[2],R); end; rng(R);
```

$$R := \{[1, b], [1, a], [2, b], [3, d], [2, d], [4, e]\}$$

dmn := **proc**(*R*::(*set*(*ordpair*)))
    map(**proc**(*x*)
        option *operator*,
        arrow;
        x[1]
    end proc, *R*)
end proc

$$\{1, 2, 3, 4\}$$

rng := **proc**(*R*::(*set*(*ordpair*)))
    map(**proc**(*x*)
        option *operator*,
        arrow;
        x[2]
    end proc, *R*)
end proc

$$\{b, a, d, e\} \qquad (1.3.15.1)$$

->1.3.16. Feladat.

1.3.17. Kiterjesztés, leszűkítés.

```
> R; select(x->(x[1]>1 and x[2]<>b),R);
```

```
restrict:=proc(R::set(ordpair),X) select(x->(x[1] in X),R);
end;
X:={2,3}; restrict(R,X);
```

$$\{[1, b], [1, a], [2, b], [3, d], [2, d], [4, e]\}$$
$$\{[3, d], [2, d], [4, e]\}$$

restrict := **proc**(*R*::(*set*(*ordpair*)), *X*)
  select(**proc**(*x*)
    option *operator*, *arrow*;
    in(*x*[1], *X*)
  end proc, *R*)
end proc

$$X := \{2, 3\}$$
$$\{[2, b], [3, d], [2, d]\} \qquad (1.3.17.1)$$

1.3.18. Inverz.

```
> relinv:=proc(R::set(ordpair)) map(x->[x[2],x[1]],R); end;
  R; dmn(R); rng(R); S:=relinv(R); dmn(S); rng(S); relinv(S);
```

relinv := **proc**(*R*::(*set*(*ordpair*)))
  map(**proc**(*x*)
    option *operator*,
    arrow;
    [*x*[2], *x*[1]]
  end proc, *R*)
end proc

$$\{[1, b], [1, a], [2, b], [3, d], [2, d], [4, e]\}$$
$$\{1, 2, 3, 4\}$$
$$\{b, a, d, e\}$$
$$S := \{[b, 2], [d, 3], [d, 2], [e, 4], [b, 1], [a, 1]\}$$
$$\{b, a, d, e\}$$
$$\{1, 2, 3, 4\}$$
$$\{[1, b], [1, a], [2, b], [3, d], [2, d], [4, e]\} \qquad (1.3.18.1)$$

1.3.19. Halmaz képe és inverz képe.

```
> mapset:=proc(R::set(ordpair),A::set) rng(restrict(R,A))
  end;
  invmapset:=proc(R::set(ordpair),A::set) rng(restrict(relinv
```

```
(R),A)) end;
R; S:=relinv(R); A:={1,4}; B:={b,e}; mapset(R,A); mapset(R,
B);
invmapset(R,B); mapset(S,B);
```

mapset := **proc**(*R*::(*set*(*ordpair*)), *A*::*set*)
  rng(*restrict*(*R*, *A*))
end proc

invmapset := **proc**(*R*::(*set*(*ordpair*)), *A*::*set*)
  rng(*restrict*(*relinv*(*R*),
  A))
end proc

$$\{[1, b], [1, a], [2, b], [3, d], [2, d], [4, e]\}$$
$$S := \{[b, 2], [d, 3], [d, 2], [e, 4], [b, 1], [a, 1]\}$$
$$A := \{1, 4\}$$
$$B := \{b, e\}$$
$$\{b, a, e\}$$
$$\{\ \}$$
$$\{1, 2, 4\}$$
$$\{1, 2, 4\} \qquad (1.3.19.1)$$

->1.3.20. Feladat.

->1.3.21. Feladat.

->1.3.22. Feladat.

->1.3.23. Feladat.

1.3.24. Kompozíció.

```
> relcomp:=proc(R::set(ordpair),S::set(ordpair)) local r,s,T;
  T:={};
  for r in R do for s in S do
    if s[2]=r[1] then T:=T union {[s[1],r[2]]}; fi;
  od; od; T; end;

  R; S:={[aa,1],[bb,3],[cc,5]}; relcomp(R,S);
```

relcomp := **proc**(*R*::(*set*(*ordpair*)), *S*::(*set*(*ordpair*)))
  local *r*, *s*, *T*;
  T := { };
  for *r* **in** *R* **do**
    for *s* **in** *S* **do**

```
            if s[2] = r[1] then
                T := union(T,
                {[s[1], r[2]]})
            end if
        end do
    end do;
    T
end proc
```

$$\{[1, b], [1, a], [2, b], [3, d], [2, d], [4, e]\}$$
$$S := \{[aa, 1], [bb, 3], [cc, 5]\}$$
$$\{[aa, b], [aa, a], [bb, d]\} \quad (1.3.24.1)$$

1.3.25. Feladat.

->1.3.26. Feladat.

1.3.27. Állítás.

```
> R; S:={[aa,1],[aa,2],[bb,3],[cc,4],[dd,5]}; relcomp(R,S);
  rng(R); rng(%%);
```

$$\{[1, b], [1, a], [2, b], [3, d], [2, d], [4, e]\}$$
$$S := \{[aa, 1], [bb, 3], [aa, 2], [cc, 4], [dd, 5]\}$$
$$\{[aa, b], [aa, a], [bb, d], [aa, d], [cc, e]\}$$
$$\{b, a, d, e\}$$
$$\{b, a, d, e\} \quad (1.3.27.1)$$

```
> T:={[xx,aa],[xx,cc]}; relcomp(R,relcomp(S,T)); relcomp
  (relcomp(R,S),T);
```

$$T := \{[xx, aa], [xx, cc]\}$$
$$\{[xx, b], [xx, a], [xx, d], [xx, e]\}$$
$$\{[xx, b], [xx, a], [xx, d], [xx, e]\} \quad (1.3.27.2)$$

```
> relinv(relcomp(R,S)); relcomp(relinv(S),relinv(R));
```

$$\{[b, aa], [a, aa], [d, bb], [d, aa], [e, cc]\}$$
$$\{[b, aa], [a, aa], [d, bb], [d, aa], [e, cc]\} \quad (1.3.27.3)$$

1.3.28. Állítás.

```
> R; IX:=id({1,2,3,4,5}); relcomp(R,IX); IY:=id({a,b,c,d,e});
  relcomp(IY,R);
```

$$\{[1, b], [1, a], [2, b], [3, d], [2, d], [4, e]\}$$
$$IX := \{[1, 1], [3, 3], [5, 5], [2, 2], [4, 4]\}$$
$$\{[1, b], [1, a], [2, b], [3, d], [2, d], [4, e]\}$$
$$IY := \{[a, a], [b, b], [c, c], [d, d], [e, e]\}$$
$$\{[1, b], [1, a], [2, b], [3, d], [2, d], [4, e]\} \quad (1.3.28.1)$$

1.3.29. Definíció.

```
> istransitive:=proc(R::set(ordpair)) local r,s;
  for r in R do for s in R do
    if r[2]=s[1] and not [r[1],s[2]] in R then
      return false fi;
  od; od; true; end;

  X:={1,2,3,4,5,6}; XX:=bincartprod(X,X):
  R:=select(x->irem(x[2],x[1])=0,XX); istransitive(R);
```

istransitive := **proc**(*R*::(*set*(*ordpair*)))
  local *r*, *s*;
  for *r* **in** *R* **do**
    for *s* **in** *R* **do**
      if *r*[2] = *s*[1] **and not** *in*([*r*[1], *s*[2]], *R*) **then**
        return *false*
      end if
    end do
  end do;
  true
end proc

$$X := \{1, 2, 3, 4, 5, 6\}$$

$$R := \{[1, 2], [1, 1], [3, 3], [5, 5], [6, 6], [1, 3], [1, 4], [1, 5], [1, 6], [2, 2], [2, 4], [2, 6], [3, 6], [4, 4]\}$$

$$true \quad (1.3.29.1)$$

```
> issymmetric:=proc(R::set(ordpair)) local r,s;
  for r in R do
    if not [r[2],r[1]] in R then return false fi;
  od; true; end;

  issymmetric(R);
```

issymmetric := **proc**(*R*::(*set*(*ordpair*)))
  local *r*, *s*;

```
    for r in R do
        if not in([r[2], r[1]], R) then
            return false
        end if
    end do;
    true
end proc
```

false (1.3.29.2)

```
> isantisymmetric:=proc(R::set(ordpair)) local r;
  for r in R do
    if [r[2],r[1]] in R then if r[1]<>r[2] then return false
  fi; fi;
  od; true; end;

  isantisymmetric(R);
```

```
isantisymmetric := proc(R::(set(ordpair)))
    local r;
    for r in R do
        if in([r[2], r[1]], R) then
            if r[1] <> r[2] then
                return false
            end if
        end if
    end do;
    true
end proc
```

true (1.3.29.3)

```
> isstriclyantisymmetric:=proc(R::set(ordpair)) local r;
  for r in R do
    if [r[2],r[1]] in R then return false fi;
  od; true; end;

  isstriclyantisymmetric(R);
```

```
isstriclyantisymmetric := proc(R::(set(ordpair)))
    local r;
    for r in R do
        if in([r[2], r[1]], R) then
            return false
        end if
```

end do;
true
end proc

false (1.3.29.4)

```
> isreflexive:=proc(X::set,R::set(ordpair)) local x;
  if not binrel(R,X) then return false fi;
  for x in X do if not [x,x] in R then return false fi; od;
  true; end;

  isreflexive(X,R);
```

isreflexive := **proc**(*X*::*set*, *R*::(*set*(*ordpair*)))
local *x*;
if not *binrel*(*R*,
X) **then**
return *false*
end if;
for *x* **in** *X* **do**
if not *in*([*x*, *x*], *R*) **then**
return *false*
end if
end do;
true
end proc

true (1.3.29.5)

```
> isirreflexive:=proc(X::set,R::set(ordpair)) local x;
  if not binrel(R,X) then return false fi;
  for x in X do if [x,x] in R then return false fi; od; true;
  end;

  isirreflexive(X,R);
```

isirreflexive := **proc**(*X*::*set*, *R*::(*set*(*ordpair*)))
local *x*;
if not *binrel*(*R*, *X*) **then**
return *false*
end if;
for *x* **in** *X* **do**
if *in*([*x*, *x*], *R*) **then**
return *false*
end if

```
    end do;
    true
end proc
```

false (1.3.29.6)

```
> istrichotom:=proc(X::set,R::set(ordpair)) local x,y;
  if not binrel(R,X) then return false fi;
  for x in X do for y in X do
    if x<>y then if ([x,y] in R and [y,x] in R) or
      ((not [x,y] in R) and (not [y,x] in R)) then return
  false
    fi; fi;
  od; od; true; end;

  istrichotom(X,R);
```

```
istrichotom := proc(X::set, R::(set(ordpair)))
    local x, y;
    if not binrel(R, X) then
        return false
    end if;
    for x in X do
        for y in X do
            if x <> y then
                if in([x, y], R) and in([y, x],
                R) or not (in([x, y], R) or in([y, x], R)) then
                    return false
                end if
            end if
        end do
    end do;
    true
end proc
```

false (1.3.29.7)

```
> isdichotom:=proc(X::set,R::set(ordpair)) local x,y;
  if not binrel(R,X) then return false fi;
  for x in X do for y in X do
    if not([x,y] in R or [y,x] in R) then return false fi;
  od; od; true; end;

  isdichotom(X,R);
```

```
isdichotom := proc(X::set, R::(set(ordpair)))
```

```
    local x, y;
    if not binrel(R, X) then
        return false
    end if;
    for x in X do
        for y in X do
            if not (in([x, y], R) or in([y, x], R)) then
                return false
            end if
        end do
    end do;
    true
end proc
```

false (1.3.29.8)

->1.3.30. Feladat.

->1.3.31. Feladat.

1.3.32. Feladat.

->1.3.33. Feladat.

1.3.34. Feladat.

->1.3.35. Feladat.

1.3.36. Feladat.

1.3.37. Reflexív, szimmetrikus illetve tranzitív relációk gráfjának egyszerűsítése.

1.3.38. Ekvivalenciareláció, osztályozás.

```
> isequivalence:=proc(X::set,R::set(ordpair))
  istransitive(R) and issymmetric(R) and isreflexive(X,R);
  end;

  isequivalence(X,R);
```

```
isequivalence := proc(X::set, R::(set(ordpair)))
    istransitive(R) and issymmetric(R) and isreflexive(X, R)
end proc
```

false (1.3.38.1)

```
> E:=select(x->irem(x[1],3)=irem(x[2],3),XX); isequivalence
  (X,E);
```

$E := \{[1, 1], [3, 3], [5, 5], [6, 3], [6, 6], [1, 4], [2, 2], [2, 5], [3, 6], [4, 1], [4, 4], [5, 2]\}$

$$true \tag{1.3.38.2}$$

```
> ispartition:=proc(X::set,c0::set(set)) local Y,Z;
  for Y in c0 do if Y={} then return false  fi;
    for Z in c0 do if Y<>Z and Y intersect Z<>{} then return
  false; fi; od; od; if `union`(op(c0))<>X then false else
  true fi; end;
```

ispartition := **proc**(X::set, c0::(set(set))) (1.3.38.3)

```
    local Y, Z;
    for Y in c0 do
        if Y = {} then
            return false
        end if;
        for Z in c0 do
            if Y <> Z and intersect(Y, Z) <> {} then
                return false
            end if
        end do
    end do;
    if union(op(c0)) <> X then
        false
    else
        true
    end if
end proc
```

```
> X; c0:={{1,4},{2,5},{3,6}}; ispartition(X,c0);
  c0:={{1},{2,3,4}}; ispartition(X,c0);
  c0:={{1,2,3},{4,5,6,7}}; ispartition(X,c0);
  c0:={{1,2,3,4},{4,5,6}}; ispartition(X,c0);
```

$$\{1, 2, 3, 4, 5, 6\}$$

$$c0 := \{\{1, 4\}, \{2, 5\}, \{3, 6\}\}$$

$$true$$

$$c0 := \{\{1\}, \{2, 3, 4\}\}$$

$$false$$

$$c0 := \{\{4, 5, 6, 7\}, \{1, 2, 3\}\}$$

$$false$$

$$cO := \{\{4, 5, 6\}, \{1, 2, 3, 4\}\}$$

$$false \quad (1.3.38.4)$$

1.3.39. Tétel.

```
> equi2part:=proc(X::set,E::set(ordpair)) local c0,x,y,tx;
  c0:={};
  for x in X do tx:={};
    for y in X do if [x,y] in E then tx:=tx union {y} fi; od;
    c0:=c0 union {tx}
  od; c0; end;

  X; E; c0:=equi2part(X,E);
```

```
equi2part := proc(X::set, E::(set(ordpair)))
    local cO, x, y, tx;
    cO := {};
    for x in X do
        tx := {};
        for y in X do
            if in([x, y], E) then
                tx := union(tx, {y})
            end if
        end do;
        cO := union(cO, {tx})
    end do;
    cO
end proc
```

$$\{1, 2, 3, 4, 5, 6\}$$

$\{[1, 1], [3, 3], [5, 5], [6, 3], [6, 6], [1, 4], [2, 2], [2, 5], [3, 6], [4, 1], [4, 4], [5, 2]\}$

$$cO := \{\{1, 4\}, \{2, 5\}, \{3, 6\}\} \quad (1.3.39.1)$$

```
> part2equi:=proc(X::set,c0::set(set)) local E,Y,x,y; E:={};
  for Y in c0 do for x in Y do for y in Y do
    E:=E union {[x,y]}
  od; od; od; E; end;

  part2equi(X,c0);
```

```
part2equi := proc(X::set, cO::(set(set)))
    local E, Y, x, y;
```

```
    E:= {};
    for Y in cO do
        for x in Y do
            for y in Y do
                E:= union(E, {[x, y]})
            end do
        end do
    end do;
    E
end proc
```

{[1, 1], [3, 3], [5, 5], [6, 3], [6, 6], [1, 4], [2, 2], [2, 5], [3, 6], [4, 1], [4, 4], [5, 2]} (1.3.39.2)

```
> c0; equi2part(X,part2equi(X,c0)); E; part2equi(X,equi2part
  (X,E));
```

{{1, 4}, {2, 5}, {3, 6}}

{{1, 4}, {2, 5}, {3, 6}}

{[1, 1], [3, 3], [5, 5], [6, 3], [6, 6], [1, 4], [2, 2], [2, 5], [3, 6], [4, 1], [4, 4], [5, 2]}

{[1, 1], [3, 3], [5, 5], [6, 3], [6, 6], [1, 4], [2, 2], [2, 5], [3, 6], [4, 1], [4, 4], [5, 2]} (1.3.39.3)

1.3.40. Példa.

->1.3.41. Feladat.

->1.3.42. Feladat.

->1.3.43. Feladat.

->1.3.44. Feladat.

1.3.45. Részbenrendezés, rendezés.

```
> ispartialordering:=proc(X::set,R::set(ordpair))
  istransitive(R) and isantisymmetric(R) and isreflexive(X,R)
  ; end;

  X; R; ispartialordering(X,R);
```

```
ispartialordering := proc(X::set, R::(set(ordpair)))
    istransitive(R) and isantisymmetric(R) and isreflexive(X, R)
end proc
```

$$\{1, 2, 3, 4, 5, 6\}$$

$\{[1, 2], [1, 1], [3, 3], [5, 5], [6, 6], [1, 3], [1, 4], [1, 5], [1, 6], [2, 2], [2, 4], [2, 6], [3, 6], [4, 4]\}$

$$true \tag{1.3.45.1}$$

```
> iscomparable:=proc(x,y,R::set(ordpair))
  evalb([x,y] in R or [y,x] in R); end;

  iscomparable(2,6,R); iscomparable(2,3,R);
```

iscomparable := **proc**(*x*, *y*, *R*::(*set*(*ordpair*)))
evalb(*in*([*x*, *y*], *R*) **or** *in*([*y*, *x*], *R*))
end proc

$$true$$

$$false \tag{1.3.45.2}$$

```
> isordering:=proc(X::set,R::set(ordpair)) local x;
  ispartialordering(X,R) and isdichotom(X,R) end;

  isordering(X,R); S:=select(x->x[1]<=x[2],XX); isordering(X,
  S);
```

isordering := **proc**(*X*::*set*, *R*::(*set*(*ordpair*)))
local *x*;
ispartialordering(*X*, *R*) **and** *isdichotom*(*X*, *R*)
end proc

$$false$$

$S := \{[1, 2], [1, 1], [3, 3], [5, 5], [5, 6], [6, 6], [1, 3], [1, 4], [1, 5], [1, 6], [2, 2], [2, 3], [2, 4], [2, 5], [2, 6], [3, 4], [3, 5], [3, 6], [4, 4], [4, 5], [4, 6]\}$

$$true \tag{1.3.45.3}$$

```
> ischain:=proc(X::set,R::set(ordpair)) local S;
  S:=R intersect bincartprod(X,X);
  isordering(X,S); end;

  ischain({1,2,4},R); ischain({1,2,3},R);
```

ischain := **proc**(*X*::*set*, *R*::(*set*(*ordpair*)))
local *S*;
S := *intersect*(*R*, *bincartprod*(*X*, *X*));
isordering(*X*, *S*)

end proc

true

false (1.3.45.4)

1.3.46. Példa.

1.3.47. Szigorú és gyenge reláció.

1.3.48. Szigorú és gyenge rendezés.

```
> strictrel:=proc(X::set,R::set(ordpair)) R minus id(X); end;
  X; R; S:=strictrel(X,R);
```

strictrel := **proc**(*X*::*set*, *R*::(*set*(*ordpair*))) *minus*(*R*, *id*(*X*)) **end proc**

$\{1, 2, 3, 4, 5, 6\}$

$\{[1, 2], [1, 1], [3, 3], [5, 5], [6, 6], [1, 3], [1, 4], [1, 5], [1, 6], [2, 2], [2, 4], [2, 6], [3, 6], [4, 4]\}$

$S := \{[1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [2, 4], [2, 6], [3, 6]\}$ (1.3.48.1)

```
> weakrel:=proc(X::set,R::set(ordpair)) R union id(X); end;
  weakrel(X,R);
```

weakrel := **proc**(*X*::*set*, *R*::(*set*(*ordpair*))) *union*(*R*, *id*(*X*)) **end proc**

$\{[1, 2], [1, 1], [3, 3], [5, 5], [6, 6], [1, 3], [1, 4], [1, 5], [1, 6], [2, 2], [2, 4], [2, 6], [3, 6], [4, 4]\}$ (1.3.48.2)

```
> istransitive(S); isirreflexive(X,S); isstriclyantisymmetric
  (S);
  istrichotom(X,S);
```

true

true

true

false (1.3.48.3)

```
> R:=select(x->x[1]<=x[2],XX); S:=strictrel(X,R); istrichotom
  (X,S);
```

$R := \{[1, 2], [1, 1], [3, 3], [5, 5], [5, 6], [6, 6], [1, 3], [1, 4], [1, 5], [1, 6], [2, 2], [2, 3], [2, 4], [2, 5], [2, 6], [3, 4], [3, 5], [3, 6], [4, 4], [4, 5], [4, 6]\}$

$S := \{[1, 2], [5, 6], [1, 3], [1, 4], [1, 5], [1, 6], [2, 3], [2, 4], [2, 5], [2, 6], [3, 4], [3, 5], [3, 6], [4, 5], [4, 6]\}$

true (1.3.48.4)

▶ *->1.3.49. Feladat.*

▶ *->1.3.50. Feladat.*

▼ ***1.3.51. Intervallumok.***

```
> int_o_o:=proc(X::set,R::set(ordpair),x,y) local S,z; S:={};
  for z in X do
    if [x,z] in R and x<>z and [z,y] in R and z<>y then S:=S
  union {z}; fi;
  od; S; end;

  int_o_c:=proc(X::set,R::set(ordpair),x,y) local S,z; S:={};
  for z in X do
    if [x,z] in R and x<>z and [z,y] in R then S:=S union {z}
  ; fi;
  od; S; end;

  int_i_o:=proc(X::set,R::set(ordpair),x) local S,y; S:={};
  for y in X do
    if [y,x] in R and x<>y then S:=S union {y}; fi;
  od; S; end;
```

```
int_o_o := proc(X::set, R::(set(ordpair)), x, y)
    local S, z;
    S := {};
    for z in X do
        if in([x, z], R) and x <> z and in([z, y],
        R) and z <> y then
            S := union(S, {z})
        end if
    end do;
    S
end proc
int_o_c := proc(X::set, R::(set(ordpair)), x, y)
    local S, z;
    S := {};
    for z in X do
        if in([x, z], R) and x <> z and in([z, y], R) then
            S := union(S, {z})
        end if
    end do;
```

S

end proc

$int\_i\_o :=$ **proc**$(X::set, R::(set(ordpair)), x)$ (1.3.51.1)

local $S, y;$

$S := \{\};$

for y **in** X **do**

if $in([y, x], R)$ **and** $x <> y$ **then**

$S := union(S, \{y\})$

end if

end do;

S

end proc

```
> X:={1,2,3,4,5,6}; XX:=bincartprod(X,X):
  R:=select(x->irem(x[2],x[1])=0,XX);
  int_o_o(X,R,1,6); int_o_c(X,R,1,6); int_i_o(X,R,6);
```

$$X := \{1, 2, 3, 4, 5, 6\}$$

$$R := \{[1, 2], [1, 1], [3, 3], [5, 5], [6, 6], [1, 3], [1, 4], [1, 5], [1, 6], [2, 2], [2, 4], [2, 6], [3, 6], [4, 4]\}$$

$$\{2, 3\}$$

$$\{2, 3, 6\}$$

$$\{1, 2, 3\} \quad (1.3.51.2)$$

```
> $ 2..6;$ 1..9;
```

$$2, 3, 4, 5, 6$$

$$1, 2, 3, 4, 5, 6, 7, 8, 9 \quad (1.3.51.3)$$

1.3.52. Részbenrendezések Hasse-diagrammja.

1.3.53. Legkisebb, legnagyobb, minimális és maximális elem.

```
> least:=proc(X::set,R::set(ordpair)) local x,y,f;
  for x in X do f:=true;
    for y in X do if not [x,y] in R then f:=false; break; fi;
  od;
    if f then return x fi;
  od; NULL; end;

  greatest:=proc(X::set,R::set(ordpair)) local x,y,f;
  for x in X do f:=true;
    for y in X do if not [y,x] in R then f:=false; break; fi;
  od;
    if f then return x fi;
```

```
od; NULL; end;
```

```
least := proc(X::set, R::(set(ordpair)))
    local x, y, f;
    for x in X do
        f := true;
        for y in X do
            if not in([x, y], R) then
                f := false;
                break
            end if
        end do;
        if f then
            return x
        end if
    end do;
    NULL
end proc
greatest := proc(X::set, R::(set(ordpair)))                    (1.3.53.1)
    local x, y, f;
    for x in X do
        f := true;
        for y in X do
            if not in([y, x], R) then
                f := false;
                break
            end if
        end do;
        if f then
            return x
        end if
    end do;
    NULL
end proc
```

```
> X; R; least(X,R); greatest(X,R);
```

$$\{1, 2, 3, 4, 5, 6\}$$

$$\{[1, 2], [1, 1], [3, 3], [5, 5], [6, 6], [1, 3], [1, 4], [1, 5], [1, 6], [2, 2], [2, 4], [2, 6], [3, 6], [4, 4]\}$$

(1.3.53.2)

$$1 \qquad (1.3.53.2)$$

```
> mins:=proc(X::set,R::set(ordpair)) local x,y,f,S; S:={};
  for x in X do f:=true;
    for y in X do if [y,x] in R and x<>y then f:=false;
  break; fi; od;
    if f then S:=S union {x}; fi;
  od; S; end;

  maxs:=proc(X::set,R::set(ordpair)) local x,y,f,S; S:={};
  for x in X do f:=true;
    for y in X do if [x,y] in R and x<>y then f:=false;
  break; fi; od;
    if f then S:=S union {x}; fi;
  od; S; end;
```

mins := **proc**(*X*::*set*, *R*::(*set*(*ordpair*)))
  local *x*, *y*, *f*, *S*;
  S := {};
  for *x* **in** *X* **do**
    f := *true*;
    for *y* **in** *X* **do**
      if *in*([*y*, *x*], *R*) **and** *x* <> *y* **then**
        f := *false*;
        break
      end if
    end do;
    if *f* **then**
      S := *union*(*S*,
      {*x*})
    end if
  end do;
  S
end proc

maxs := **proc**(*X*::*set*, *R*::(*set*(*ordpair*))) (1.3.53.3)
  local *x*, *y*, *f*, *S*;
  S := {};
  for *x* **in** *X* **do**
    f := *true*;
    for *y* **in** *X* **do**
      if *in*([*x*, *y*], *R*) **and** *x* <> *y* **then**
        f := *false*;

```
break
end if
end do;
if f then
S := union(S,
{x})
end if
end do;
S
end proc
```

```
> X; R; mins(X,R); maxs(X,R);
```

$\{1, 2, 3, 4, 5, 6\}$

$\{[1, 2], [1, 1], [3, 3], [5, 5], [6, 6], [1, 3], [1, 4], [1, 5], [1, 6], [2, 2], [2, 4], [2, 6], [3, 6], [4, 4]\}$

$\{1\}$

$\{4, 5, 6\}$ (1.3.53.4)

1.3.54. Példa.

1.3.55. Korlátok.

```
> islowerbound:=proc(X::set,R::set(ordpair),Y::set,x) local
y;
for y in Y do if not [x,y] in R then return false fi; od;
true; end;

isupperbound:=proc(X::set,R::set(ordpair),Y::set,x) local
y;
for y in Y do if not [y,x] in R then return false fi; od;
true; end;
```

```
islowerbound := proc(X::set, R::(set(ordpair)), Y::set, x)
local y;
for y in Y do
if not in([x, y], R) then
return false
end if
end do;
true
end proc
```

(1.3.55.1)

isupperbound := **proc**(*X*::*set*, *R*::(*set*(*ordpair*)), *Y*::*set*, *x*) (1.3.55.1)

```
    local y;
    for y in Y do
        if not in([y, x], R) then
            return false
        end if
    end do;
    true
end proc
```

```
> X; R; islowerbound(X,R,{2,3,5},1); isupperbound(X,R,{2,3,5}
  ,6);
```

$$\{1, 2, 3, 4, 5, 6\}$$

{[1, 2], [1, 1], [3, 3], [5, 5], [6, 6], [1, 3], [1, 4], [1, 5], [1, 6], [2, 2], [2, 4], [2, 6], [3, 6], [4, 4]}

true

false (1.3.55.2)

```
> lowerbounds:=proc(X::set,R::set(ordpair),Y::set) local S,x;
  S:={};
  for x in X do if islowerbound(X,R,Y,x) then S:=S  union {x}
  fi; od;
  S; end;
```

lowerbounds := **proc**(*X*::*set*, *R*::(*set*(*ordpair*)), *Y*::*set*) (1.3.55.3)

```
    local S, x;
    S := {};
    for x in X do
        if islowerbound(X, R, Y, x) then
            S := union(S, {x})
        end if
    end do;
    S
end proc
```

```
> upperbounds:=proc(X::set,R::set(ordpair),Y::set) local S,x;
  S:={};
  for x in X do if isupperbound(X,R,Y,x) then S:=S  union {x}
  fi; od;
  S; end;
```

upperbounds := **proc**(*X*::*set*, *R*::(*set*(*ordpair*)), *Y*::*set*) (1.3.55.4)

```
    local S, x;
```

```
    S:= {};
    for x in X do
        if isupperbound(X, R, Y, x) then
            S:= union(S, {x})
        end if
    end do;
    S
end proc
```

```
> X:={1,2,3,4,5,6}; XX:=bincartprod(X,X):
  R:=select(x->irem(x[2],x[1])=0,XX);
  lowerbounds(X,R,{2,4}); upperbounds(X,R,{2,4});
```

$$X := \{1, 2, 3, 4, 5, 6\}$$

$$R := \{[1, 2], [1, 1], [3, 3], [5, 5], [6, 6], [1, 3], [1, 4], [1, 5], [1, 6], [2, 2], [2, 4], [2, 6], [3, 6], [4, 4]\}$$

$$\{1, 2\}$$

$$\{4\} \quad (1.3.55.5)$$

```
> inf:=proc(X::set,R::set(ordpair),Y::set)
  greatest(lowerbounds(X,R,Y),R); end;

  sup:=proc(X::set,R::set(ordpair),Y::set)
  least(upperbounds(X,R,Y),R); end;

  inf(X,R,{2,4}); sup(X,R,{2,4});
```

```
inf := proc(X::set, R::(set(ordpair)), Y::set)
    greatest(lowerbounds(X,
   R, Y), R)
end proc
sup := proc(X::set, R::(set(ordpair)), Y::set)
    least(upperbounds(X,
   R, Y), R)
end proc
```

$$2$$

$$4 \quad (1.3.55.6)$$

▶ ***->1.3.56. Feladat.***

▶ ***->1.3.57. Feladat.***

▼ ***->1.3.58. Feladat.***

->1.3.59. Feladat.

->1.3.60. Feladat.

->1.3.61. Feladat.

1.3.62. Feladat.

->1.3.63. Feladat.

->1.3.64. Feladat: program futási sebességének optimalizálása részbenrendezés kiterjesztéseinek segítségével.

1.3.65. Jólrendezés.

```
> iswellordering:=proc(X::set,R::set(ordpair)) local f,Y,P;
  f:=isordering(X,R); if not f then return f fi;
  P:=powerset(X);
  for Y in P do
    if Y<>{} then f:=least(Y,R); if f=NULL then return f; fi;
  fi;
  od; true; end;
```

iswellordering := **proc**(X::set, R::(set(ordpair))) (1.3.65.1)

```
    local f, Y, P;
    f := isordering(X, R);
    if not f then
        return f
    end if;
    P := combinat:-powerset(X);
    for Y in P do
        if Y <> {} then
            f := least(Y, R);
            if f = NULL then
                return f
            end if
        end if
    end do;
    true
end proc
```

```
> X; R; iswellordering(X,R);

  R:=select(x->x[1]<=x[2],XX);iswellordering(X,R);
```

$$\{1, 2, 3, 4, 5, 6\}$$

{[1, 2], [1, 1], [3, 3], [5, 5], [6, 6], [1, 3], [1, 4], [1, 5], [1, 6], [2, 2], [2, 4], [2, 6], [3, 6], [4, 4]}

false

R := {[1, 2], [1, 1], [3, 3], [5, 5], [5, 6], [6, 6], [1, 3], [1, 4], [1, 5], [1, 6], [2, 2], [2, 3], [2, 4], [2, 5], [2, 6], [3, 4], [3, 5], [3, 6], [4, 4], [4, 5], [4, 6]}

true (1.3.65.2)

1.3.66. Példa.

->1.3.67. Feladat.

->1.3.68. Feladat.

->1.3.69. Feladat.

->1.3.70. Feladat.

1.3.71. Példák.

```
> orderingprod:=proc(X::set,R::set(ordpair),Y::set,S::set
(ordpair))
local x,y,xp,yp,RS; RS:={};
for x in X do for y in Y do for xp in X do for yp in Y do
  if [x,xp] in R and [y,yp] in S then RS:=RS union {[[x,y],
[xp,yp]]}; fi;
od; od; od; od; RS; end;
```

orderingprod := **proc**(*X*::*set*, *R*::(*set*(*ordpair*)), *Y*::*set*, *S*::(*set*(*ordpair*))) (1.3.71.1)

```
local x, y, xp, yp, RS;
RS := {};
for x in X do
    for y in Y do
        for xp in X do
            for yp in Y do
                if in([x, xp], R) and in([y, yp], S) then
                    RS := union(RS, {[[x, y], [xp, yp]]})
                end if
            end do
        end do
```

```
        end do
    end do;
    RS
end proc
```

```
> X:={1,2}; R:={[1,1],[1,2],[2,2]}; isordering(X,R);
  Y:={a,b}; S:={[a,a],[a,b],[b,b]}; isordering(Y,S);
  XY:=bincartprod(X,Y); RS:=orderingprod(X,R,Y,S);
  isordering(XY,RS);
```

$$X := \{1, 2\}$$
$$R := \{[1, 2], [1, 1], [2, 2]\}$$
$$true$$
$$Y := \{b, a\}$$
$$S := \{[a, a], [b, b], [a, b]\}$$
$$true$$
$$XY := \{[1, b], [1, a], [2, b], [2, a]\}$$
$$RS := \{[[1, b], [1, b]], [[1, b], [2, b]], [[1, a], [1, b]], [[1, a], [1, a]], [[1, a], [2, b]], [[1, a], [2, a]], [[2, b], [2, b]], [[2, a], [2, b]], [[2, a], [2, a]]\}$$
$$false \qquad (1.3.71.2)$$

```
> strictorderingprod:=proc(X::set,R::set(ordpair),Y::set,
  S::set(ordpair)) local x,y,xp,yp,RS; RS:={};
  for x in X do for y in Y do for xp in X do for yp in Y do
    if [x,xp] in R and x<>xp and [y,yp] in S and y<>yp then
      RS:=RS union {[[x,y],[xp,yp]]}; fi;
  od; od; od; od; RS; end;
```

(1.3.71.3)

```
strictorderingprod := proc(X::set, R::(set(ordpair)), Y::set,
    S::(set(ordpair)))
    local x, y, xp, yp, RS;
    RS := {};
    for x in X do
        for y in Y do
            for xp in X do
                for yp in Y do
                    if in([x, xp],
                    R) and x <> xp and in([y, yp],
                    S) and y <> yp then
                        RS := union(RS, {[[x, y], [xp,
```

```
                        yp]]})
                    end if
                end do
            end do
        end do
    end do;
    RS
end proc
```

```
> lexorderingprod:=proc(X::set,R::set(ordpair),Y::set,S::set
  (ordpair))
  local x,y,xp,yp,RS; RS:={};
  for x in X do for y in Y do for xp in X do for yp in Y do
    if ([x,xp] in R and x<>xp) or (x=xp and [y,yp] in S) then

      RS:=RS union {[[x,y],[xp,yp]]}; fi;
  od; od; od; od; RS; end;
```

```
lexorderingprod := proc(X::set, R::(set(ordpair)), Y::set,        (1.3.71.4)
    S::(set(ordpair)))
    local x, y, xp, yp, RS;
    RS := {};
    for x in X do
        for y in Y do
            for xp in X do
                for yp in Y do
                    if in([x, xp],
                    R) and x <> xp or x = xp and in([y, yp], S) then
                        RS := union(RS, {[[x, y], [xp, yp]]})
                    end if
                end do
            end do
        end do
    end do;
    RS
end proc
```

```
> strictrel(XY,RS); strictorderingprod(X,R,Y,S);
  strictrel(XY,lexorderingprod(X,R,Y,S));
```

{[[1, b], [2, b]], [[1, a], [1, b]], [[1, a], [2, b]], [[1, a], [2, a]], [[2,

$a], [2, b]]\}$

$$\{[[1, a], [2, b]]\}$$

$$\{[[1, b], [2, b]], [[1, a], [1, b]], [[1, a], [2, b]], [[1, a], [2, a]], [[2, a], [2, b]], [[1, b], [2, a]]\} \quad (1.3.71.5)$$

```
> sort([cc,ca,cb,bb,aa,ab,ba],lexorder);
```

$$[aa, ab, ba, bb, ca, cb, cc] \quad (1.3.71.6)$$

->1.3.72. Feladat.
1.3.73. Feladat.
->1.3.74. Feladat.
1.3.75. Feladat.
1.3.76. Feladat.
1.2.77. További feladatok részletes megoldással.
1.2.78. További feladatok.

1.4. Függvények

1.4.1. Függvény.

```
> isfunction:=proc(f::set(ordpair),X::set,Y::set) local x,y,
  S;
  if not binrel(args) then return false fi;
  for x in dmn(f) do S:={}; for y in rng(f) do
    if [x,y] in f then S:=S union {y} fi;
    od; if nops(S)>1 then return false fi;
  od; true; end;
```

isfunction := **proc**(f::(set(ordpair)), X::set, Y::set) (1.4.1.1)

```
    local x, y, S;
    if not binrel(args) then
        return false
    end if;
    for x in dmn(f) do
        S := {};
        for y in rng(f) do
            if in([x, y], f) then
                S := union(S, {y})
            end if
        end do;
```

```
        if 1 < nops(S) then
            return false
        end if
    end do;
    true
end proc
> f:=id({a,b}); isfunction(f); isfunction(f,{a,b,1});
  isfunction(f,{a,b},{1,2});f:={[a,1],[b,2],[a,2]};
  isfunction(f);
```

$$f := \{[a, a], [b, b]\}$$

true

true

false

$$f := \{[a, 2], [b, 2], [a, 1]\}$$

false (1.4.1.2)

```
> isinjective:=proc(f::set(ordpair))
  isfunction(f) and isfunction(relinv(f)); end;

  isinjective({[a,1],[b,2]}); isinjective({[a,1],[b,1]});
```

$isinjective := \textbf{proc}(f{::}(set(ordpair)))$

    $isfunction(f)$ **and** $isfunction(relinv(f))$

end proc

true

false (1.4.1.3)

```
> issurjective:=proc(f::set(ordpair),Y::set)
  isfunction(f) and rng(f)=Y; end;

  issurjective({[a,1],[b,1]},{1,2});

  issurjective({[a,1],[b,2]},{1,2});
```

$issurjective := \textbf{proc}(f{::}(set(ordpair)), Y{::}set)$

    $isfunction(f)$ **and** $rng(f) = Y$

end proc

false

true (1.4.1.4)

```
> isbijective:=proc(f::set(ordpair),Y::set)
  isinjective(f) and issurjective(f,Y); end;

  isbijective(id({1,2,3}),{1,2,3});
```

```
isbijective({[a,1],[b,2]},{1,2,3});
```

isbijective := **proc**(*f*::(*set*(*ordpair*)), *Y*::*set*)
isinjective(*f*) **and** *issurjective*(*f*, *Y*)
end proc

$$true$$

$$false \tag{1.4.1.5}$$

```
> f:=x->x^2; f(1);f(2);f(3); eval(f); type(f,procedure);
```

$$f := x \to x^2$$

$$1$$

$$4$$

$$9$$

$$x \to x^2$$

$$true \tag{1.4.1.6}$$

```
> f:='f'; eval(f); whattype(f);
  f(1):=1; eval(f);
  f(2):=4; f(3):=8;
  f(1);f(2);f(3);f(4);
```

$$f := f$$

$$f$$

$$symbol$$

$$f(1) := 1$$

proc() **option** *remember*; '*procname*(*args*)' **end proc**

$$f(2) := 4$$

$$f(3) := 8$$

$$1$$

$$4$$

$$8$$

$$f(4) \tag{1.4.1.7}$$

```
> isarrowfromto:=proc(f::procedure,X::set,Y::set) local x;
  for x in X do if not f(x) in Y then return false fi; od;
  true; end;

  isarrowfromto(f,{1,2,3},{1,4,8,10});
  isarrowfromto(f,{1,2},{1,8});
```

isarrowfromto := **proc**(*f*::*procedure*, *X*::*set*, *Y*::*set*)
local *x*;
for *x* **in** *X* **do**

```
        if not in(f(x), Y) then
            return false
        end if
    end do;
    true
end proc
```

true

false (1.4.1.8)

```
> makefunction:=proc(R::set(ordpair)) local x,y,f;
  if not isfunction(R) then return NULL fi;
  for x in dmn(R) do for y in rng(R) do if [x,y] in R then f
  (x):=y fi;
  od; od; eval(f); end;

  f:='f'; R:={[1,1],[2,4],[3,9]}; f(1);f(2);f(3);f(4);

  f:=makefunction(R);f(1);f(2);f(3);f(4);
```

```
makefunction := proc(R::(set(ordpair)))
    local x, y, f;
    if not isfunction(R) then
        return NULL
    end if;
    for x in dmn(R) do
        for y in rng(R) do
            if in([x, y], R) then
                f(x) := y
            end if
        end do
    end do;
    eval(f)
end proc
```

$$f := f$$

$$R := \{[1, 1], [2, 4], [3, 9]\}$$

$$f(1)$$

$$f(2)$$

$$f(3)$$

$$f(4)$$

f := **proc**( ) **option** *remember*; '*procname*(*args*)' **end proc**

1

$$4$$
$$9$$
$$f(4) \qquad (1.4.1.9)$$

->1.4.2. Feladat.

->1.4.3. Feladat.

->1.4.4. Feladat.

->1.4.5. Feladat.

1.4.6. Feladat.

->1.4.7. Feladat.

1.4.8. Feladat.

1.4.9. Feladat.

1.4.10. Feladat.

1.4.11. Állítás.

1.4.12. Kanonikus leképezés.

```
> makecanonical:=proc(X::set,R::set(ordpair)) local x,rx,y,f;
  if not isequivalence(X,R) then return FAIL fi;
  for x in X do rx:={}; for y in X do
    if [x,y] in R then rx:=rx union {y} fi; od; f(x):=rx
  od; eval(f) end;

  f:=makecanonical({1,2,3},{[1,1],[1,2],[2,1],[2,2],[3,3]});

  f(1);f(2);f(3);
```

```
makecanonical := proc(X::set, R::(set(ordpair)))
    local x, rx, y, f;
    if not isequivalence(X, R) then
        return FAIL
    end if;
    for x in X do
        rx := {};
        for y in X do
            if in([x, y], R) then
                rx := union(rx, {y})
            end if
```

end do;
$f(x) := rx$
end do;
$eval(f)$
end proc
$f := \mathbf{proc}()$ **option** *remember*, *'procname(args)'* **end proc**

$$\{1, 2\}$$

$$\{1, 2\}$$

$$\{3\} \quad (1.4.12.1)$$

->1.4.13. Feladat.

1.4.14. Feladat.

1.4.15. Feladat.

1.4.16. Feladat.

*1.4.17. Feladat.

1.4.18. Monoton függvények.

Az alábbi két függvényben R rendezés X-en, S pedig rendezés Y-on.

```
> isincreasing:=proc(f::procedure,X::set,R::set(ordpair),
  Y::set,S::set(ordpair)) local x,y;
  if not isarrowfromto(f,X,Y) then return FAIL fi;
  if not ispartialordering(X,R) or not ispartialordering(Y,S)
  then return
    FAIL fi;
  for x in X do for y in X do
    if [x,y] in R and not [f(x),f(y)] in S then return false
  fi;
  od; od; true end;

  X:={1,2,3}; XX:=bincartprod(X,X): R:=select(x->irem(x[2],x
  [1])=0,XX);
  Y:=X; S:=select(x->x[1]<=x[2],XX); f:=x->x; isincreasing(f,
  X,R,Y,S);isincreasing(f,Y,S,X,R);
```

isincreasing := **proc**(*f*::*procedure*, *X*::*set*, *R*::(*set*(*ordpair*)), *Y*::*set*,
S::(*set*(*ordpair*)))
local *x*, *y*;
if not *isarrowfromto*(*f*, *X*, *Y*) **then**
return *FAIL*
end if;

if not (*ispartialordering*(*X*, *R*) and *ispartialordering*(*Y*, *S*)) then
return *FAIL*
end if;
for *x* in *X* do
for *y* in *X* do
if *in*([*x*, *y*], *R*) and not *in*([*f*(*x*), *f*(*y*)], *S*) then
return *false*
end if
end do
end do;
true
end proc

$$X := \{1, 2, 3\}$$
$$R := \{[1, 2], [1, 1], [3, 3], [1, 3], [2, 2]\}$$
$$Y := \{1, 2, 3\}$$
$$S := \{[1, 2], [1, 1], [3, 3], [1, 3], [2, 2], [2, 3]\}$$
$$f := x \rightarrow x$$
true
false (1.4.18.1)

->1.4.19. Feladat.

->1.4.20. Feladat.

1.4.21. Indexelt családok.

```
> issetfamily:=proc(Iset::set,f::procedure) local i;
  for i in Iset do if not type(f(i),set) then return false
  fi;
  od; true; end;

  f:='f'; f(1):={a,b};f(2):={b,c,d}; issetfamily({1,2},f);
  issetfamily({1,2,3},f);
```

issetfamily := proc(*Iset*::*set*, *f*::*procedure*)
local *i*;
for *i* in *Iset* do
if not *type*(*f*(*i*), *set*) then
return *false*

```
        end if
    end do;
    true
end proc
```

$$f := f$$
$$f(1) := \{b, a\}$$
$$f(2) := \{b, c, d\}$$
$$true$$
$$false \quad (1.4.21.1)$$

1.4.22. De Morgan-szabályok.

1.4.23. Megjegyzés.

```
> `union`();
```

$$\{\ \} \quad (1.4.23.1)$$

```
> `intersect`();
Error, invalid input: `intersect` expects 1 or more arguments, but received 0
```

1.4.24. Tétel.

1.4.25. Feladat.

->1.4.26. Feladat.

1.4.27. Feladat.

1.4.28. Feladat.

1.4.29. Reláció és Descartes-szorzat általános esetben.

```
> s:=x,y; s[1]; s[2]; t:=y,x; evalb(s=t);
```

$$s := x, y$$
$$x$$
$$y$$
$$t := y, x$$
$$false \quad (1.4.29.1)$$

```
> descartesprod:=proc(L::list(set)) local x,y,i,S,SS;
  if nops(L)=0 then return {} fi; S:=map(x->[x],L[1]);
  for i from 2 to nops(L) do SS:={}; for x in S do for y in L
  [i] do
    SS:=SS union {[op(x),y]}
  od; od; S:=SS od; S; end;
```

```
descartesprod([]);
descartesprod([{1,2}]);
descartesprod([{1,2},{a,b}]);
descartesprod([{1,2},{}]);
descartesprod([{1,2},{a},{x,y}]);
descartesprod([{1,2},{a,b,c},{x,y}]);
```

```
descartesprod := proc(L::(list(set)))
    local x, y, i, S, SS;
    if nops(L) = 0 then
        return {}
    end if;
    S := map(proc(x)
        option operator, arrow;
        [x]
    end proc, L[1]);
    for i from 2 to nops(L) do
        SS := {};
        for x in S do
            for y in L[i] do
                SS := union(SS, {[op(x), y]})
            end do
        end do;
        S := SS
    end do;
    S
end proc
```

$$\{\ \}$$

$$\{[1], [2]\}$$

$$\{[1, b], [1, a], [2, b], [2, a]\}$$

$$\{\ \}$$

$$\{[1, a, x], [1, a, y], [2, a, x], [2, a, y]\}$$

$$\{[1, a, x], [1, a, y], [2, a, x], [2, a, y], [1, b, x], [1, b, y], [2, b, x], [2, b, y], [1, c, x], [1, c, y], [2, c, x], [2, c, y]\} \quad (1.4.29.2)$$

```
> isselection:=proc(Iset::set,f::procedure,x::procedure)
  local i;
  if not issetfamily(Iset,f) then return FAIL fi;
  for i in Iset do if not x(i) in f(i) then return false fi;
```

```
od; true end;

Iset:={a,b,c}; f:='f'; f(a):={1,2};f(b):={1,3};f(c):={2,3};
x(a):=1;x(b):=3;x(c):=3; isselection(Iset,f,x);

x(b):=2; isselection(Iset,f,x);
```

isselection := **proc**(*Iset*::*set*, *f*::*procedure*, *x*::*procedure*)
    local *i*;
    if not *issetfamily*(*Iset*, *f*) **then**
        return *FAIL*
    end if;
    for *i* **in** *Iset* **do**
        if not *in*(*x*(*i*), *f*(*i*)) **then**
            return *false*
        end if
    end do;
    true
end proc

$$Iset := \{b, c, a\}$$
$$f := f$$
$$f(a) := \{1, 2\}$$
$$f(b) := \{1, 3\}$$
$$f(c) := \{2, 3\}$$
$$x(a) := 1$$
$$x(b) := 3$$
$$x(c) := 3$$
$$true$$
$$x(b) := 2$$
$$false \qquad (1.4.29.3)$$

```
>
> agent:={[D209,"Peti"],[KISZ1,"Fleto"],[Puf3,"Gyula"]};

event:={[KISZ1,"Balaton",19930706],[Puf3,"Nyugati",
19561108],[KISZ1,"Motim",19961231],[D209,"Paks",20000103],
[KISZ1,"Fittelina",19980320],[D209,"Gresham",20010908],
[KISZ1,"Nomentana",19951122]};

descartesprod([agent,event]):select(x->x[1][1]=x[2][1],%)
:map(x->[x[1][2],x[2][2],x[2][3],x[2][1]],%):active:=select
(x->x[3]>19891023,%);
```

$agent := \{[D209, \text{"Peti"}], [KISZ1, \text{"Fleto"}], [Puf3, \text{"Gyula"}]\}$

$event := \{[KISZ1, \text{"Balaton"}, 19930706], [Puf3, \text{"Nyugati"}, 19561108], [KISZ1, \text{"Motim"}, 19961231], [D209, \text{"Paks"}, 20000103], [KISZ1, \text{"Fittelina"}, 19980320], [D209, \text{"Gresham"}, 20010908], [KISZ1, \text{"Nomentana"}, 19951122]\}$

$active := \{[\text{"Peti"}, \text{"Paks"}, 20000103, D209], [\text{"Peti"}, \text{"Gresham"}, 20010908, D209], [\text{"Fleto"}, \text{"Balaton"}, 19930706, KISZ1], [\text{"Fleto"}, \text{"Motim"}, 19961231, KISZ1], [\text{"Fleto"}, \text{"Fittelina"}, 19980320, KISZ1], [\text{"Fleto"}, \text{"Nomentana"}, 19951122, KISZ1]\}$ (1.4.29.4)

```
> isselection:=proc(Iset::set,f::procedure,x::procedure)
  local i;
  if not issetfamily(Iset,f) then return FAIL fi;
  for i in Iset do if not x(i) in f(i) then return false fi;
  od; true end;

  Iset:={a,b,c}; f:='f'; f(a):={1,2};f(b):={1,3};f(c):={2,3};
  x(a):=1;x(b):=3;x(c):=3; isselection(Iset,f,x);

  x(b):=2; isselection(Iset,f,x);
```

isselection := **proc**(*Iset*::*set*, *f*::*procedure*, *x*::*procedure*)
    local *i*;
    if not *issetfamily*(*Iset*, *f*) **then**
        return *FAIL*
    end if;
    for *i* **in** *Iset* **do**
        if not *in*(*x*(*i*), *f*(*i*)) **then**
            return *false*
        end if
    end do;
    true
end proc

$$Iset := \{b, c, a\}$$

$$f := f$$

$$f(a) := \{1, 2\}$$

$$f(b) := \{1, 3\}$$

$$f(c) := \{2, 3\}$$

$$x(a) := 1$$

$$x(b) := 3$$

$$x(c) := 3$$

$$true$$

$$x(b) := 2$$
$$false \qquad (1.4.29.5)$$

->1.4.30. Feladat.

->1.4.31. Feladat.

1.4.32. Feladat.

->1.4.33. Feladat.

->1.4.34. Feladat.

->1.4.35. Feladat.

1.4.36. Műveletek.

```
> isbinop:=proc(X::set,f::procedure) local x,y;
  for x in X do for y in X do
    if not f(x,y) in X then return false fi;
  od; od; true end;

  f:='f'; f(0,0):=0;f(0,1):=1;f(1,0):=1;f(1,1):=0; isbinop(
  {0,1},f);

  isbinop({0,1,2},f);
```

```
isbinop := proc(X::set, f::procedure)
    local x, y;
    for x in X do
        for y in X do
            if not in(f(x, y), X) then
                return false
            end if
        end do
    end do;
    true
end proc
```

$$f := f$$
$$f(0, 0) := 0$$
$$f(0, 1) := 1$$
$$f(1, 0) := 1$$
$$f(1, 1) := 0$$

true

false (1.4.36.1)

```
> &+(0,0):=0;&+(0,1):=1;&+(1,0):=1;&+(1,1):=0;
  0&+1;1&+1;&+(0,1);
  isbinop({0,1},(x,y)->x &+ y);
```

$0 \mathbin{\&+} 0 := 0$

$0 \mathbin{\&+} 1 := 1$

$1 \mathbin{\&+} 0 := 1$

$1 \mathbin{\&+} 1 := 0$

1

0

1

true (1.4.36.2)

```
> isunop:=proc(X::set,f::procedure) local x;
  for x in X do if not f(x) in X then return false fi; od;
  true end;

  f:='f'; f(0):=1;f(1):=0; isunop({0,1},f); isunop({0,1,2},f)
  ;
```

isunop := **proc**(*X*::*set*, *f*::*procedure*)
  local *x*;
  for *x* **in** *X* **do**
    if not *in*(*f*(*x*), *X*) **then**
      return *false*
    end if
  end do;
  true
end proc

$f := f$

$f(0) := 1$

$f(1) := 0$

true

false (1.4.36.3)

```
> &inc(0):=1;&inc(1):=0; &inc 0; &inc 1; isunop({0,1},x->&inc
  x);
```

$\&inc(0) := 1$

$\&inc(1) := 0$

1

$$0$$
$$true \quad (1.4.36.4)$$

```
> 9();9(x);9(x,y);
```

$$9$$
$$9$$
$$9 \quad (1.4.36.5)$$

```
> isnullop:=proc(X::set,f::procedure) evalb(f() in X) end;

  f:='f'; f():=1; eval(f); isnullop({0,1},f); isnullop({0,2},
  f);
```

$$isnullop := \textbf{proc}(X::set, f::procedure)\ evalb(in(f(), X))\ \textbf{end proc}$$
$$f := f$$
$$f() := 1$$
$$\textbf{proc}()\ \textbf{option}\ remember, \ 'procname(args)'\ \textbf{end proc}$$
$$true$$
$$false \quad (1.4.36.6)$$

1.4.37. Példa.

1.4.38. Példák.

```
> X:={1,2,3}; P:=combinat[powerset](X); isbinop(P,(x,y)->x
  union y);
```

$$X := \{1, 2, 3\}$$
$$P := \{\{\}, \{1\}, \{2\}, \{1, 2\}, \{3\}, \{1, 3\}, \{1, 2, 3\}, \{2, 3\}\}$$
$$true \quad (1.4.38.1)$$

1.4.39. Példák: művelet megadása táblázattal.

```
> true and true; true and false; false and true; false and
  false;

  T:=table();
  T[true,true]:=true;T[true,false]:=false;
  T[false,true]:=true;T[false,false]:=false;
  print(T);
```

$$true$$
$$false$$
$$false$$
$$false$$

$$T := table([\,])$$
$$T_{true,\, true} := true$$
$$T_{true,\, false} := false$$
$$T_{false,\, true} := true$$
$$T_{false,\, false} := false$$
$$table([(false, false) = false, (true, false) = false, (true, true) = true, (false, true) = true]) \quad (1.4.39.1)$$

->1.4.40. Feladat.

->1.4.41. Feladat.

1.4.42. Feladat.

1.4.43. Műveletek függvényekkel.

```
> f:=x->x^2; g:=x->x^3; (f*g)(2); (f*g)(3); (f/g)(2); (f/g)
  (0); (g/f)(0);
```

$$f := x \to x^2$$
$$g := x \to x^3$$
$$32$$
$$243$$
$$\frac{1}{2}$$

```
Error, numeric exception: division by zero
Error, numeric exception: division by zero
```

1.4.44. Példa.

1.4.45. Példák.

```
> f:=[true,true,false,false];
  g:=[true,false,true,false];
  zip((x,y)->x and y,f,g);
```

$$f := [true, true, false, false]$$
$$g := [true, false, true, false]$$
$$[true, false, false, false] \quad (1.4.45.1)$$

1.4.46. Műveletttartó leképezések.

```
> ishom:=proc(phi::procedure,X::set,f::procedure,Y::set,
  g::procedure)
  local x,y;
  if not isarrowfromto(phi,X,Y) then return FAIL fi;
  if not isbinop(X,f) then return FAIL fi;
  if not isbinop(Y,g) then return FAIL fi;
  for x in X do for y in X do
    if phi(f(x,y))<>g(phi(x),phi(y)) then return false fi;
  od; od; true end;

  X:={true,false}; Y:=X; ishom(x->x,X,(x,y)->x and y,Y,(x,y)-
  >x or y);

  ishom(x-> not x,X,(x,y)->x and y,Y,(x,y)->x or y);
```

ishom := **proc**(phi::*procedure*, *X*::*set*, *f*::*procedure*, *Y*::*set*,
g::*procedure*)
  local *x*, *y*;
  if not *isarrowfromto*(phi, *X*, *Y*) **then**
    return *FAIL*
  end if;
  if not *isbinop*(*X*, *f*) **then**
    return *FAIL*
  end if;
  if not *isbinop*(*Y*, *g*) **then**
    return *FAIL*
  end if;
  for *x* **in** *X* **do**
    for *y* **in** *X* **do**
      if phi(*f*(*x*, *y*)) <> *g*(phi(*x*), phi(*y*)) **then**
        return *false*
      end if
    end do
  end do;
  true
end proc

$$X := \{false, true\}$$

$$Y := \{false, true\}$$

$$false$$

(1.4.46.1)

$$true \quad (1.4.46.1)$$

1.4.47. Példa.

```
> a:='a';  a^(x+y); expand(%);
```

$$a := a$$
$$a^{x+y}$$
$$a^x a^y \quad (1.4.47.1)$$

1.4.48. Feladat.

1.4.49. További feladatok.

2. Természetes számok

3. A számfogalom bővítése

4. Véges halmazok

5. Végtelen halmazok

6. Számelmélet

7. Gráfelmélet

8. Algebra

9. Kódolás

10. Algoritmusok

```
>
>
```