ANHANG

A. Das Programm 'Entladung'

Das Programm wurde auf einem Apple II Mikrocomputer unter UCSD-Pascal entwickelt; die letzte Version wurde für Pascal 6000 Release 3 modifiziert und auf einer Cyber 175 implementiert - in dieser Form ist es hier wiedergegeben.

```
(*=============================================================================*)
(*                                                                             *)
(* Titel : Entladung                                                           *)
(* Autor : Ulrich Schmidt                                                      *)
(* Datum : Mai 1982                                                            *)
(* Anlass: Ueberpruefung des Beweises fuer den Vierfarbensatz                  *)
(*                                                                             *)
(*                                                                             *)
(*                                                                             *)
(* Programmbeschreibung:                                                       *)
(*                                                                             *)
(*    Dieses Programm ueberprueft das qTS(V5)-Lemma des Beweises von           *)
(* Appel und Haken: 'Every planar map is four colorable', Illinois             *)
(* Journal of Mathematics, Vol. 21, No. 3, Sept. 1977.                         *)
(*                                                                             *)
(*    Das qTS(V5)-Lemma ist einer der beiden wesentlichen Bestand-             *)
(* teile des ersten Beweisteiles, der sogenannten 'Entladungsproze-            *)
(* dur'. Es weist nach, dass nach Ausfuehrung der Entladungsprozedur           *)
(* alle Ecken vom Grad 5 entweder entladen sind (d.h. ihre Ladung              *)
(* ist 0 oder negativ) oder zu Situationen gehoeren, die eine redu-            *)
(* zible Figur (aus der unvermeidlichen Menge von reduziblen Figuren)          *)
(* als Teilgraphen enthalten.                                                  *)
(*                                                                             *)
(*    Die kombinatorischen Einzelheiten des qTS(V5)-Lemmas sind im             *)
(* Mikrofiche-Anhang der vorgenannten Veroeffentlichung enthalten.             *)
(* Die komplizierte Fallaufzaehlung, das stillschweigende Weglassen            *)
(* von zyklischen Permutationen und Reflexionen sowie einige von               *)
(* Hand gefundene Fehler waren die Hauptmotive, das qTS(V5)-Lemma              *)
(* maschinell ueberpruefen zu lassen.                                          *)
(*                                                                             *)
(*    Das nachfolgende Programm 'Entladung' erzeugt alle moeglichen            *)
(* Randkreise fuer eine Fuenferecke (Prozedur 'erzeuge'), wobei nur            *)
(* essentiell neue Randkreise betrachtet werden. Zyklische Permuta-            *)
(* tionen und Reflexionen werden durch die Funktion 'vorhanden' er-            *)
(* kannt; sie werden nicht weiter betrachtet. Neue Randkreise werden           *)
(* durch die Prozedur 'eintrage' dynamisch abgespeichert. Die Rand-            *)
(* ecken koennen vom Grad 5, 6 oder MehrAls6 sein.                             *)
(*                                                                             *)
(*    Der aus zentraler Fuenferecke und Randkreis gebildete Graph              *)
(* wird zunaechst auf Reduzierbarkeit untersucht. Ist der Graph redu-          *)
(* zibel, so wird die Identifikation der reduzierenden Figur ausge-            *)
(* druckt. Anderenfalls werden nacheinander an allen Randecken vom             *)
(* Grad MehrAls6 S-Situationen angebracht (eventuell zusaetzlich noch          *)
(* T-Situationen), bis der so veraenderte Graph entweder reduzierbar           *)
(* ist, seine Ladung nicht mehr positiv ist oder keine weiteren S-             *)
(* Situationen mehr angebracht werden koennen.                                 *)
(*                                                                             *)
(*    Im letzteren Fall wird der Graph untersucht, ob er eine L-Situ-          *)
(* ation enthaelt, welche die zentrale Fuenferecke vollstaendig ent-           *)
(* laedt. Gibt es keine solche Situation, so liegt ein 'kritischer             *)
(* Fall' vor.                                                                  *)
(*                                                                             *)
(*                                                                             *)
(*                                                                             *)
(* Dateien:                                                                    *)
(*                                                                             *)
(*    U     : unvermeidliche Menge von reduziblen Figuren (mit Aus-            *)
(*            nahme derjenigen Figuren, die Zehner- oder Elferecken            *)
(*            enthalten)                                                       *)
(*                                                                             *)
```

```
(*    Tafel: Tafel I aus dem Mikrofiche-Anhang (alle Figuren aus I     *)
(*           enthalten reduzible Figuren aus U; Tafel I dient zur      *)
(*           schnelleren Behandlung der T-Situationen)                 *)
(*                                                                     *)
(*    S    : Situationen kleiner Entladung (S-Situationen)             *)
(*                                                                     *)
(*    L    : Situationen grosser Entladung (L-Situationen)             *)
(*                                                                     *)
(*    T    : Fernentladungssituationen     (T-Situationen)             *)
(*                                                                     *)
(*    Extra: rechte Figur auf Seite 439                                *)
(*                                                                     *)
(*    Output:diese Datei enthaelt die Ergebnisse des Programms; alle   *)
(*           Randkreise mit den angebrachten S-Situationen werden      *)
(*           ausgegeben. Die Buchstaben N und R bedeuten 'normal'      *)
(*           bzw. 'reflektiert'. Bei reduzierbaren Graphen werden      *)
(*           Seite und Nummer der reduzierenden Figur mit ausgegeben.  *)
(*           'Kritische' Graphen werden durch '*****' gekennzeichnet.  *)
(*                                                                     *)
(*                                                                     *)
(*                                                                     *)
(* Prozeduren und Funktionen:                                          *)
(*                                                                     *)
(*    (Hierarchie wird durch Einruecken verdeutlicht. Einigen Proze-   *)
(*     duren und Funktionen folgt eine eingeklammerte Liste von        *)
(*     Prozedur- und Funktionsnamen; letztere werden von ersteren      *)
(*     aufgerufen, sind ihnen jedoch nicht hierarchisch unterge-       *)
(*     ordnet.)                                                        *)
(*                                                                     *)
(*    Prozedur Fehler                                                  *)
(*    Prozedur lies                                                    *)
(*    Prozedur eintrage                                                *)
(*    Prozedur initialisiere (lies, eintrage)                          *)
(*       Prozedur waehle                                               *)
(*    Prozedur entlade                                                 *)
(*       Prozedur drucke                                               *)
(*       Prozedur vervollstaendige                                     *)
(*       Prozedur positioniere                                         *)
(*       Prozedur pruefe (positioniere)                                *)
(*          Prozedur verbinde                                          *)
(*       Funktion Aehnlichkeit                                         *)
(*       Funktion Typpruefung                                          *)
(*       Prozedur reduziere                                            *)
(*          Funktion Subisomorphie (pruefe, Typpruefung)               *)
(*          Prozedur Mengenreduktion(lies,Aehnlichkeit,Subisomorphie)  *)
(*             Prozedur Ende                                           *)
(*          Prozedur Tafelreduktion (lies,Aehnlichkeit,Subisomorphie)  *)
(*       Prozedur verschmelze (pruefe)                                 *)
(*          Prozedur vorbereite                                        *)
(*          Prozedur AlterTeil (positioniere)                          *)
(*          Prozedur NeuerTeil                                         *)
(*          Prozedur vernetze                                          *)
(*             Prozedur hinzufuege                                     *)
(*             Prozedur verkleinere                                    *)
(*          Prozedur normalisiere                                      *)
(*             Prozedur schiebe                                        *)
(*          Funktion Dreieck                                           *)
(*          Funktion Fuenfeck (pruefe)                                 *)
(*       Prozedur erweitere (lies, verschmelze, entlade)               *)
```

```
(*        Prozedur viel (lies, Aehnlichkeit)                                *)
(*           Funktion LIsomorphie (pruefe, Typpruefung)                     *)
(*        Funktion zulaessig                                                *)
(*        Prozedur wenig (lies, verschmelze, entlade)                       *)
(*           Prozedur eingrenze                                             *)
(*              Prozedur setze                                              *)
(*     Prozedur umwandle                                                    *)
(*     Funktion vorhanden                                                   *)
(*     Prozedur fuelle (vorhanden, eintrage, fuelle)                        *)
(*     Prozedur erzeuge (vorhanden, eintrage)                               *)
(*                                                                          *)
(*==========================================================================*)


program Entladung(U, Tafel, S, L, T, Extra, Output);


const MehrAls4 = 1; (* Grad 5 oder hoeher *)
      MehrAls5 = 2; (* Grad 6 oder hoeher *)
      MehrAls6 = 3; (* Grad 7 oder hoeher *)
      MehrAls7 = 4; (* Grad 8 oder hoeher *)

      MinGrad = MehrAls4; (* kleinster Grad einer Ecke *)
      MaxGrad = 9;        (* groesster Grad einer Ecke *)

      MinGraph =  4; (* soviel Ecken hat der kleinste Graph *)
      MaxGraph = 63; (* soviel Ecken hat der groesste Graph *)

      MaxSeite  =  63;
      MaxNummer = 730;
      MaxLadung =  60;
      MaxT      =   7; (* Anzahl der verschiedenen T-Situationen *)
      MaxTZahl  =   6; (* Anzahl der T-Situationen pro Graph *)
      MaxTTyp   =   5;
      MaxTyp    =   3;

      Normwert = 30; (* regulaere Entladung *)
      Absatz   =  2; (* um soviel Spalten wird der Rand eingerueckt *)


type Bereich     = 0..MaxGraph;
     Teilbereich = 1..MaxGraph;

     Valenz      = 0..MaxGrad;
     Teilvalenz  = 1..MaxGrad;

     Kopfsatz    = packed record
                     Seite      : 1..MaxSeite;
                     Nummer     : 1..MaxNummer;
                     Eckenzahl  : MinGraph..MaxGraph;
                     symmetrisch: boolean;
                     Ladung     : 0..MaxLadung;
                     Vollecken  : Bereich;
                     RZahl      : Bereich;
                     TZahl      : 0..MaxTZahl;
                     Struktur   : packed array[5..MaxGrad] of Bereich
```

```
                end;

TSatz         = packed record
                   Sender     : Teilbereich;
                   Empfaenger: Teilbereich;
                   TTyp       : 1..MaxTTyp
                end;

TFeld         = packed array[1..MaxTZahl] of TSatz;
Eckenfeld     = packed array[Teilvalenz] of Bereich;

Eckensatz     = packed record
                   Grad     : Valenz;
                   Typ      : 0..MaxTyp;
                   Nachbarn: Valenz;
                   Partner  : Bereich;
                   Liste    : Eckenfeld
                end;

Graphsatz     = packed record
                   Kopf     : Kopfsatz;
                   Transfer: TFeld;
                   Ecke     : packed array[Teilbereich] of Eckensatz
                end;

Binaersatz    = packed record
                   case integer of
                      1: (Kopf     : Kopfsatz);
                      2: (Transfer: TFeld);
                      3: (Ecke     : Eckensatz)
                end;

Binaerdatei   = packed file of Binaersatz;

Kreiswert     = (fuenf, sechs, mittel, klein, regulaer);
Kreisrand     = array[2..6] of Kreiswert;

Zeiger        = ^Kreisreihe;

Kreisreihe    = record
                   Element    : Kreisrand;
                   Vorgaenger: Zeiger
                end;

Option        = (undefiniert, normal, reflektiert);

TransSatz     = packed record
                   Ebene : 1..MaxTZahl;
                   Nummer: 1..MaxT;
                   Modus : Option
                end;

SName         = packed record
                   Nummer: 1..329;
                   Modus : Option;
                   TZahl : 0..2;
                   voll  : boolean;
                   Trans : packed array[1..2] of TransSatz
                end;
```

```
     Kette        = packed array[1..16] of char;


var U, Tafel, S, L, T, Extra: Binaerdatei;
    Speicher                : ^integer;
    Reihe, Hilfsreihe       : Zeiger;
    Kreis                   : Kreisrand;
    Graph, Sonderfall       : Graphsatz;
    SFeld                   : array[2..6] of SName;
    Laenge                  : array[Valenz] of 5..MaxGrad;
    UWahl, SWahl            : array[Valenz, Valenz] of Valenz;
    Flagge, fertig          : boolean;


procedure Fehler( (* drucke *) Name: Kette (* und halte *) );

begin (* Fehler *)

   writeln;
   write('*****   Fehler in ', Name);

   halt

end   (* Fehler *);



   lies( (* von der *) var Datei: Binaerdatei;
         (* in den  *) var Graph: Graphsatz    );

var i: Teilbereich;

begin (* lies *)

   Graph.Kopf := Datei^.Kopf;
   get(Datei);

   Graph.Transfer := Datei^.Transfer;
   get(Datei);

   for i := 1 to Graph.Kopf.Eckenzahl do
      begin
         Graph.Ecke[i] := Datei^.Ecke;
         get(Datei)
      end

end   (* lies *);


procedure eintrage( (* den    *)     Kreis: Kreisrand;
                    (* in die *) var Reihe: Zeiger     );

var Neureihe: Zeiger;

begin (* eintrage *) (* speichert Randkreis dynamisch ab *)
```

```
   new(Neureihe);

   Neureihe^.Element    := Kreis;
   Neureihe^.Vorgaenger := Reihe;

   Reihe := Neureihe

end   (* eintrage *);


procedure initialisiere( (* initialisiert den *) var Kreis: Kreisrand;
                         (* und die           *) var Reihe: Zeiger    );

var i: integer;


procedure waehle;

var i, j: Valenz;

begin (* waehle *) (* initialisiert die Identifizierungsmatrix *)

   for i := MinGrad to MaxGrad do
      for j := MinGrad to MaxGrad do
         begin
            UWahl[i, j] := 0;
            SWahl[i, j] := 0
         end;

   UWahl[MehrAls4, MehrAls4] := MehrAls4;

   UWahl[MehrAls5, MehrAls5] := MehrAls5;

   UWahl[MehrAls6, MehrAls5] := MehrAls6;
   UWahl[MehrAls6, MehrAls6] := MehrAls6;

   for j := MehrAls5 to MehrAls7 do
      UWahl[MehrAls7, j] := MehrAls7;

   UWahl[5, 5] := 5;

   UWahl[6, MehrAls5] := 6;
   UWahl[6, 6       ] := 6;

   UWahl[7, MehrAls5] := 7;
   UWahl[7, MehrAls6] := 7;
   UWahl[7, 7       ] := 7;

   for j := MehrAls5 to MehrAls7 do
      UWahl[8, j] := 8;

   UWahl[8, 8] := 8;

   for j := MehrAls5 to MehrAls7 do
      UWahl[9, j] := 9;
```

```
   UWahl[9, 9] := 9;

   for j := MinGrad to MaxGrad do
      SWahl[MehrAls4, j] := j;

   SWahl[MehrAls5, MehrAls4] := MehrAls5;

   for j := MehrAls5 to MehrAls7 do
      SWahl[MehrAls5, j] := j;

   for j := 6 to MaxGrad do
      SWahl[MehrAls5, j] := j;

   for j := MehrAls4 to MehrAls6 do
      SWahl[MehrAls6, j] := MehrAls6;

   SWahl[MehrAls6, MehrAls7] := MehrAls7;

   for j := 7 to MaxGrad do
      SWahl[MehrAls6, j] := j;

   for j := MehrAls4 to MehrAls7 do
      SWahl[MehrAls7, j] := MehrAls7;

   SWahl[MehrAls7, 8] := 8;
   SWahl[MehrAls7, 9] := 9;

   SWahl[5, MehrAls4] := 5;
   SWahl[5, 5       ] := 5;

   SWahl[6, MehrAls4] := 6;
   Swahl[6, MehrAls5] := 6;
   Swahl[6, 6       ] := 6;

   for j := MehrAls4 to MehrAls6 do
      SWahl[7, j] := 7;

   SWahl[7, 7] := 7;

   for j := MehrAls4 to MehrAls7 do
      SWahl[8, j] := 8;

   SWahl[8, 8] := 8;

   for j := MehrAls4 to MehrAls7 do
      SWahl[9, j] := 9;

   SWahl[9, 9] := 9

end   (* waehle *);


begin (* initialisiere *)

   waehle;

   reset(Extra);
   lies(Extra, Sonderfall);
```

```
   for i := MinGrad to MaxGrad do (* logische Listenlaenge *)
      if i < 5 then
         Laenge[i] := MaxGrad
      else
         Laenge[i] := i;

   for i := 2 to 6 do (* erster Randkreis = 5 5 5 5 5 *)
      Kreis[i] := fuenf;

   Reihe := nil;
   eintrage(Kreis, Reihe)

end   (* initialisiere *);



procedure entlade( (* entlaedt den    *) var Graph: Graphsatz;
                   (* verwendet den   *)     Kreis: Kreisrand;
                   (* S-Situation bei *)     Index: Teilbereich;
                   (* rueckt ein um   *)     Rand : integer      );

var reduzibel, ausTafel, moeglich, erfolgreich: boolean;
    i                                         : 1..6;
    j                                         : 1..2;
    Nr                                        : integer;



procedure drucke( (* Ausgabe auf   *) var Datei : Text;
                  (* den Namen des *) var Graph : Graphsatz;
                  (* die Ecken des *)     Kreis : Kreisrand;
                  (* einrueckt den *)     Rand  : integer    );

var i: 2..6;
    j: 0..2;

begin (* drucke *)

   writeln(Datei);
   write(Datei, ' ');

   if Rand > 0 then
      write(Datei, ' ':Rand);

   for i := 2 to 6 do
      case Kreis[i] of

         fuenf   : write(Datei, '5 ');
         sechs   : write(Datei, '6 ');
         mittel  : write(Datei, 'U ');

         klein   : with SFeld[i] do
                      begin

                           if Nummer >= 100 then
                        write(Datei, 'S'  , Nummer:3)
                     else if Nummer >=  10 then
                        write(Datei, 'S0' , Nummer:2)
```

```
                        else
                           write(Datei, 'S00', Nummer:1);

                        if Modus = normal then
                           write(Datei, 'n')
                        else
                           write(Datei, 'r');

                        j := 0;

                        repeat

                           j := j + 1;

                           if Trans[j].Modus <> undefiniert then
                              begin

                                 write(Datei,'.',Trans[j].Nummer:1);

                                 if Trans[j].Modus = normal then
                                    write(Datei, 'n')
                                 else
                                    write(Datei, 'r')

                              end

                        until    (j                = TZahl)
                              or (Trans[j].Modus = undefiniert);

                        write(Datei, ' ')

                     end;

         regulaer: write(Datei, 'R ')

      end (* case *);

   write(Datei, '  z =', Graph.Kopf.Ladung:3, '   ')

end   (* drucke *);


procedure vervollstaendige( (* den *) var Graph: Graphsatz);

var i: Teilbereich;
    j: 5..MaxGrad;

begin (* vervollstaendige *)
   with Graph, Kopf do
      begin

         Vollecken := 0;

         for j := 5 to MaxGrad do
            Struktur[j] := 0;

         for i := 1 to Eckenzahl do
            with Ecke[i] do
```

```
                if Grad in [5..MaxGrad] then
                   begin
                      Vollecken      := Vollecken      + 1;
                      Struktur[Grad] := Struktur[Grad] + 1
                   end

      end
end   (* vervollstaendige *);



procedure positioniere( (* benutzt  *) var Graph    : Graphsatz;
                        (* benutzt  *) var Situation: Graphsatz;
                        (* benutzt  *)     Index    : Teilbereich;
                        (* bestimmt *) var G, S     : Teilvalenz   );

var gefunden: boolean;
    i       : Valenz;
    V       : Teilbereich;

begin (* positioniere *)
   with Graph, Ecke[Index] do
      begin

         i         := 0;
         gefunden := false;

         repeat

            i := i + 1;

            if Liste[i] <> 0 then
               gefunden := Ecke[Liste[i]].Partner <> 0

         until gefunden;

         G := i;
         S := 1;
         V := Ecke[Liste[G]].Partner;

         while Situation.Ecke[Partner].Liste[S] <> V do
            S := S + 1

      end
end   (* positioniere *);



Procedure pruefe( (* veraendert *) var Graph    : Graphsatz;
                                       X, Y, Z  : Teilbereich;
                  (* veraendert *) var Situation: Graphsatz;
                                       A, B, C  : Teilbereich;
                  (* je nach    *)     Richtung : Option;
                  (* erhaelt    *)     UFall    : boolean;
                  (* bestimmt   *) var moeglich : boolean      );

var GEcke, SEcke, Neue, i: Bereich;
    Tabelle              : array[Teilbereich] of (leer, neu, alt);
    G, S, j              : Teilvalenz;
```

```
    L                       : 5..MaxGrad;


procedure verbinde( (* die Ecken *) G, (* und *) S: Teilbereich);

begin (* verbinde *)

   with Graph.Ecke[G], Situation do
      begin

                  Partner := S;
         Ecke[S].Partner := G;

         if UFall then
            if Nachbarn < Ecke[S].Nachbarn then
               Grad := 0
            else
               Grad := UWahl[Grad, Ecke[S].Grad]
         else
            Grad := SWahl[Grad, Ecke[S].Grad]

      end;

   Tabelle[G] := neu;
   Neue       := Neue + 1

end   (* verbinde *);



begin (* pruefe *)

   Neue := 0;

   for i := 1 to Graph.Kopf.Eckenzahl do
      begin

         Tabelle[i] := leer;

         Graph.Ecke[i].Partner := 0;
         Graph.Ecke[i].Typ     := 0

      end;

   for i := 1 to Situation.Kopf.Eckenzahl do
      Situation.Ecke[i].Partner := 0;

   verbinde(X, A);
   verbinde(Y, B);
   verbinde(Z, C);

   moeglich :=     (Graph.Ecke[X].Grad <> 0)
               and (Graph.Ecke[Y].Grad <> 0)
               and (Graph.Ecke[Z].Grad <> 0);

   while moeglich and (Neue > 0) do
      begin
```

```
        i := 0;

        repeat

           i := i + 1;

           with Graph.Ecke[i], Situation do
              if (Partner <> 0) and (Tabelle[i] = neu) then
                 begin

                    positioniere(Graph, Situation, i, G, S);

                    L := Laenge[Grad];
                    j := 1;

                    repeat

                       G := G mod L + 1;

                       if Richtung = normal then
                          S := S mod L + 1
                       else
                          if S = 1 then
                             S := L
                          else
                             S := S - 1;

                       GEcke :=                Liste[G];
                       SEcke := Ecke[Partner].Liste[S];

                       if (GEcke = 0) and (SEcke <> 0) then
                          moeglich := not UFall
                       else
                          if (GEcke <> 0) and (SEcke <> 0) then
                             if Ecke[SEcke].Partner = 0 then
                                with Graph.Ecke[GEcke] do
                                   begin
                                      verbinde(GEcke, SEcke);
                                      moeglich := Grad <> 0
                                   end;

                       j := j + 1

                    until (j = L) or not moeglich;

                    Tabelle[i] := alt;
                    Neue       := Neue - 1

                 end

        until (i = Graph.Kopf.Eckenzahl) or (Neue = 0) or not moeglich

     end

end   (* pruefe *);


function Aehnlichkeit( (* zwischen *) var Graph    : Graphsatz;
```

```
                        (* und      *) var Teilgraph: Graphsatz):boolean;

var moeglich: boolean;
    i       : 4..MaxGrad;

begin (* Aehnlichkeit *)

   with Graph, Teilgraph.Kopf do
      begin

         moeglich :=     (Eckenzahl <= Kopf.Eckenzahl)
                     and (Vollecken <= Kopf.Vollecken)
                     and (TZahl     <= Kopf.TZahl    );

         i := 4;

         while moeglich and (i < MaxGrad) do
            begin
               i         := i + 1;
               moeglich := Struktur[i] <= Kopf.Struktur[i]
            end

      end;

   Aehnlichkeit := moeglich

end   (* Aehnlichkeit *);



function Typpruefung( (* von     *) var Graph    : Graphsatz;
                      (* und     *) var Teilgraph: Graphsatz;
                      (* je nach *)     Richtung : Option    ): boolean;

var i, j  : 0..MaxTZahl;
    k     : Valenz;
    S     : Teilbereich;
    Flagge: boolean;

begin (* Typpruefung *)

   i := 0;

   repeat

      i := i + 1;

      with Teilgraph, Transfer[i] do
         begin

            if Richtung = normal then
               S := Sender
            else
               with Ecke[Empfaenger] do
                  begin

                     k := 0;

                     repeat
```

```
                        k := k + 1
                     until Liste[k] = Sender;

                     if k = 1 then
                        S := Liste[Laenge[Grad]]
                     else
                        S := Liste[k - 1]

                  end;

            j := 0;

            repeat
               j      := j + 1;
               Flagge := Graph.Transfer[j].Sender = Ecke[S].Partner
            until (j = Graph.Kopf.TZahl) or Flagge;

            if Flagge then
               case Graph.Transfer[j].TTyp of
                  1: Flagge := TTyp = 1;
                  2: Flagge := TTyp in [1..4];
                  3: Flagge := TTyp in [1, 3];
                  4: Flagge := TTyp in [1, 4];
                  5: Flagge := TTyp in [1, 4, 5]
               end (* case *)

         end

   until (i = Teilgraph.Kopf.TZahl) or not Flagge;

   Typpruefung := Flagge

end   (* Typpruefung *);



procedure reduziere( (* veraendert den  *) var Graph    : Graphsatz;
                     (* zeigt an, ob    *) var reduzibel: boolean;
                     (* und ob Reduzent *) var ausTafel : boolean    );



function Subisomorphie( (* zwischen *) var Graph    :Graphsatz;
                        (* und      *) var Teilgraph:Graphsatz):boolean;

var Figur, Teilfigur: Graphsatz;
    i, Y, Z         : Bereich;
    B, C            : Teilbereich;
    j               : Valenz;
    r               : Option;
    moeglich        : boolean;
    L               : 5..MaxGrad;



begin (* Subisomorphie *)

   with Teilgraph.Ecke[1] do
      if Liste[2] = 0 then
```

```
      begin
         B := Liste[Laenge[Grad]];
         C := Liste[1]
      end
   else
      begin
         B := Liste[1];
         C := Liste[2]
      end;

i        := 0;
moeglich := false;

repeat

   i := i + 1;

   with Graph.Ecke[i], Teilgraph do
      if UWahl[Grad, Ecke[1].Grad] <> 0 then
         begin

            j := 0;
            L := Laenge[Grad];

            repeat

               j := j + 1;
               Y := Liste[j];

               if Y <> 0 then
                  if UWahl[Graph.Ecke[Y].Grad,Ecke[B].Grad] <> 0 ther
                     begin

                        r := undefiniert;

                        repeat

                           r := succ(r);

                           if r = normal then
                              Z := Liste[j mod L + 1]
                           else
                              if j = 1 then
                                 Z := Liste[L]
                              else
                                 Z := Liste[j - 1];

                           if Z <> 0 then
                            if UWahl[Graph.Ecke[Z].Grad,
                                          Ecke[C].Grad  ] <> 0
                             then
                              begin

                               Figur     := Graph;
                               Teilfigur := Teilgraph;

                               pruefe(Figur,i,Y,Z,Teilfigur,1,B,C,
                                      r, true, moeglich             );
```

```
                                        if moeglich and (Kopf.TZahl > 0) then
                                         moeglich := Typpruefung(Figur     ,
                                                                 Teilfigur,r)

                                      end

                                until (r = reflektiert) or moeglich

                             end

                    until (j = L) or moeglich

                end

      until (i = Graph.Kopf.Eckenzahl) or moeglich;

      Subisomorphie := moeglich

   end   (* Subisomorphie *);



   procedure Mengenreduktion( (* veraendert *) var Graph    : Graphsatz;
                              (* bestimmt   *) var reduzibel: boolean    );

   var Teilgraph: Graphsatz;
       Summe    : integer;
       Endseite : 1..MaxSeite;
       Endnummer: 1..MaxNummer;



   procedure Ende( (* bestimmt durch *) Seite, Nummer: integer);

   begin (* Ende *)
      Endseite  := Seite;
      Endnummer := Nummer
   end   (* Ende *);



   begin (* Mengenreduktion *)

      with Graph.Kopf do
         begin

            Summe := Struktur[7] + Struktur[8] + Struktur[9];

            if Summe > 4 then
               Ende(63, 16)
            else
               case Summe of

                  0:    Ende( 1, 32);

                  1:      if Struktur[7] = 1 then
                             Ende( 6, 35)
                          else if Struktur[8] = 1 then
                             Ende(11, 35)
```

```
                else
                   Ende(14, 29);

             2:       if Struktur[7] = 2 then
                   Ende(21, 35)
                else if (Struktur[7] = 1) and (Struktur[8] = 1) then
                   Ende(33, 21)
                else if (Struktur[7] = 1) and (Struktur[9] = 1) then
                   Ende(40, 35)
                else if Struktur[8] = 2 then
                   Ende(45,  8)
                else if (Struktur[8] = 1) and (Struktur[9] = 1) then
                   Ende(45, 28)
                else
                   Ende(45, 32);

             3:       if Struktur[7] = 3 then
                   Ende(49, 28)
                else if (Struktur[7] = 2) and (Struktur[8] = 1) then
                   Ende(57, 14)
                else if (Struktur[7] = 2) and (Struktur[9] = 1) then
                   Ende(60, 28)
                else if (Struktur[7] = 1) and (Struktur[8] = 2) then
                   Ende(61, 30)
                else
                   Ende(61, 33);

             4:       if Struktur[7] = 4 then
                   Ende(62, 24)
                else if (Struktur[7] = 3) and (Struktur[8] = 1) then
                   Ende(63, 14)
                else
                   Ende(63, 16)

          end (* case *)

   end;

with Graph, Teilgraph.Kopf do
   begin

      reset(U);
      reduzibel := false;

      repeat

         lies(U, Teilgraph);

         if Aehnlichkeit(Graph, Teilgraph) then
            begin

               reduzibel := Subisomorphie(Graph, Teilgraph);

               if reduzibel then
                  begin
                     Kopf.Seite  := Seite;
                     Kopf.Nummer := Nummer
                  end
```

```
                end

            until reduzibel or (Seite = Endseite) and (Nummer = Endnummer)

        end

end    (* Mengenreduktion *);



procedure Tafelreduktion( (* untersucht *) var Graph    : Graphsatz;
                          (* setzt      *) var reduzibel: boolean    );

var Teilgraph: Graphsatz;

begin (* Tafelreduktion *)
   with Graph, Teilgraph.Kopf do
      begin

         reset(Tafel);
         reduzibel := false;

         repeat

            lies(Tafel, Teilgraph);

            if Aehnlichkeit(Graph, Teilgraph) then
               begin

                  reduzibel := Subisomorphie(Graph, Teilgraph);

                  if reduzibel then
                     begin
                        Kopf.Seite  := Seite;
                        Kopf.Nummer := Nummer
                     end

               end

         until reduzibel or eof(Tafel)

      end
end    (* Tafelreduktion *);



begin (* reduziere *)
   if Graph.Kopf.TZahl = 0 then
      begin
         ausTafel := false;
         Mengenreduktion(Graph, reduzibel)
      end
   else
      begin

         Tafelreduktion(Graph, reduzibel);
         ausTafel := reduzibel;

         if not reduzibel then
```

```
            Mengenreduktion(Graph, reduzibel)

      end
end   (* reduziere *);


procedure verschmelze( (* verschmilzt *) var Graph    : Graphsatz;
                       (* am Dreieck  *)     X, Y, Z  : Teilbereich;
                       (* mit der     *) var Situation: Graphsatz;
                       (* je nach     *)     Richtung : Option;
                       (* bestimmt    *) var moeglich : boolean      );

var Neuzahl : Teilbereich;


procedure vorbereite( (* veraendert *) var Graph    : Graphsatz;
                      (* veraendert *) var Situation: Graphsatz;
                      (* je nach    *)     Richtung : Option;
                      (* bestimmt   *) var Neuzahl  : Teilbereich);

var i: Teilbereich;

begin (* vorbereite *)

   Neuzahl := Graph.Kopf.Eckenzahl;

   for i := 1 to Situation.Kopf.Eckenzahl do
      with Graph, Situation.Ecke[i] do
         if Partner = 0 then
            begin

               Neuzahl := Neuzahl + 1;

               Ecke[Neuzahl].Grad := Grad;

               if Richtung = normal then
                  Ecke[Neuzahl].Typ := Typ
               else
                  case Typ of
                     0, 3: Ecke[Neuzahl].Typ := Typ;
                     1   : Ecke[Neuzahl].Typ := 2;
                     2   : Ecke[Neuzahl].Typ := 1
                  end (* case *);

               Ecke[Neuzahl].Partner  := i;
               Ecke[Neuzahl].Nachbarn := 0;

               Partner := Neuzahl

            end

end   (* vorbereite *);


Procedure AlterTeil( (* veraendert *) var Graph    : Graphsatz;
                     (* benutzt    *) var Situation: Graphsatz;
```

```
                        (* je nach    *)     Richtung : Option     );

var i     : Teilbereich;
    SEcke: Bereich;
    G, S  : Teilvalenz;
    j     : 2..MaxGrad;
    L     : 5..MaxGrad;

begin (* AlterTeil *)
   for i := 1 to Graph.Kopf.Eckenzahl do
      with Graph.Ecke[i], Situation do
         if Partner <> 0 then
            begin

               positioniere(Graph, Situation, i, G, S);

               if Richtung = normal then
                  Typ := Ecke[Partner].Typ
               else
                  case Ecke[Partner].Typ of
                     0, 3: Typ := Ecke[Partner].Typ;
                     1   : Typ := 2;
                     2   : Typ := 1
                  end (* case *);

               L := Laenge[Grad];

               for j := 2 to L do
                  begin

                     G := G mod L + 1;

                     if Richtung = normal then
                        S := S mod L + 1
                     else
                        if S = 1 then
                           S := L
                        else
                           S := S - 1;

                     SEcke := Ecke[Partner].Liste[S];

                     if (Liste[G] = 0) and (SEcke <> 0) then
                        begin

                           Liste[G] := Ecke[SEcke].Partner;

                           Nachbarn := Nachbarn + 1

                        end

                  end

            end
end   (* AlterTeil *);


Procedure NeuerTeil( (* veraendert *) var Graph    : Graphsatz;
```

```
                    (* benutzt     *) var Situation: Graphsatz;
                    (* je nach     *)     Richtung : Option;
                    (* benutzt     *)     Neuzahl  : Teilbereich);

var i       : Teilbereich;
    j, G, S: Teilvalenz;
    SEcke   : Bereich;

begin (* NeuerTeil *)
   for i := Graph.Kopf.Eckenzahl+1 to Neuzahl do
      with Graph.Ecke[i], Situation do
         if Richtung = normal then
            for j := 1 to Laenge[Grad] do
               begin

                  SEcke := Ecke[Partner].Liste[j];

                  if SEcke <> 0 then
                     begin
                        Liste[j] := Ecke[SEcke].Partner;
                        Nachbarn := Nachbarn + 1
                     end
                  else
                     Liste[j] := 0

               end
         else
            begin

               S := Laenge[Grad];

               while Ecke[Partner].Liste[S] = 0 do
                  begin
                     Liste[S] := 0;
                     S        := S - 1
                  end;

               G := 1;

               for S := S downto 1 do
                  begin

                     SEcke := Ecke[Partner].Liste[S];

                     if SEcke <> 0 then
                        begin
                           Liste[G] := Ecke[SEcke].Partner;
                           Nachbarn := Nachbarn + 1
                        end
                     else
                        Liste[G] := 0;

                     G := G + 1

                  end

            end
end   (* NeuerTeil *);
```

```
procedure vernetze( (* veraendert *) var Graph   : Graphsatz;
                    (* bestimmt   *) var moeglich: boolean    );

var i, X, Y, Z, Neue: Bereich;
    Ersatz          : Teilbereich;
    j               : Valenz;
    L               : 5..MaxGrad;
    Tabelle         : array[Teilbereich] of (neu, alt);
    Nachbarecken    : array[Teilbereich] of set of Bereich;
    besetzt         : boolean;



procedure hinzufuege( (* zur        *)     Neuecke       : Teilbereich;
                      (* neben der  *)     Zentralecke   : Teilbereich;
                      (* die neue   *)     Schwesterecke: Teilbereich;
                      (* je nach    *)     Richtung      : Option;
                      (* bestimmt   *) var besetzt       : boolean;
                      (* bestimmt   *) var Ersatz        : Teilbereich  );

var G: Teilvalenz;
    L: 5..MaxGrad;

begin (* hinzufuege *)
   with Graph.Ecke[Neuecke] do
      begin

         L := Laenge[Grad];
         G := 1;

         while Liste[G] <> Zentralecke do
            G := G + 1;

         if Richtung = normal then
            G := G mod L + 1
         else
            if G = 1 then
               G := L
            else
               G := G - 1;

         if Liste[G] <> 0 then
            begin
               besetzt := true;
               Ersatz  := Liste[G]
            end
         else
            begin

               besetzt  := false;
               Liste[G] := Schwesterecke;
               Nachbarn := Nachbarn + 1;

               Nachbarecken[Neuecke] :=   Nachbarecken[Neuecke]
                                        + [Schwesterecke];

               if Tabelle[Neuecke] = alt then
```

```
                    begin
                       Tabelle[Neuecke] := neu;
                       Neue             := Neue + 1
                    end

             end

      end
end   (* hinzufuege *);



procedure verkleinere( (* veraendert *) var Graph   : Graphsatz;
                       (* ersetzt    *)     Kopie   : Teilbereich;
                       (* durch das  *)     Original: Teilbereich  );

var i: Teilbereich;
    j: Teilvalenz;
    k: 2..MaxGraph;

begin (* verkleinere *)
   with Graph, Kopf do
      begin

         with Ecke[Original] do
            Grad := SWahl[Grad, Ecke[Kopie].Grad];

         for i := 1 to Eckenzahl do
            with Ecke[i] do
               for j := 1 to Laenge[Grad] do
                  if Liste[j] = Kopie then
                     begin
                        Liste[j] := 0;
                        Nachbarn := Nachbarn - 1
                     end;

         for k := Kopie+1 to Eckenzahl do
            begin

               Ecke[k - 1] := Ecke[k];

               for i := 1 to Eckenzahl do
                  with Ecke[i] do
                     for j := 1 to Laenge[Grad] do
                        if Liste[j] = k then
                           Liste[j] := k - 1

            end;

         Eckenzahl := Eckenzahl - 1

      end
end   (* verkleinere *);



begin (* vernetze *)

   moeglich := true;
```

```
repeat

   for i := 1 to Graph.Kopf.Eckenzahl do
      with Graph.Ecke[i] do
         begin

            Tabelle[i]      := neu;
            Nachbareckeп[i] := [];

            for j := 1 to Laenge[Grad] do
               Nachbareckeп[i] := Nachbareckeп[i] + [Liste[j]]

         end;

   Neue     := Graph.Kopf.Eckenzahl;
   besetzt := false;

   repeat

      i := 0;

      repeat

         i := i + 1;

         if Tabelle[i] = neu then
            with Graph.Ecke[i] do
               begin

                  L := Laenge[Grad];
                  j := 0;

                  repeat

                     j := j + 1;
                     Y := Liste[j];
                     Z := Liste[j mod L + 1];

                     if j = 1 then
                        X := Liste[L]
                     else
                        X := Liste[j - 1];

                     if (Y <> 0) and (X <> 0) then
                      if not (Y in Nachbareckeп[X]) then
                       hinzufuege(X,i,Y,reflektiert,besetzt,Ersatz);

                     if (Y <> 0) and (Z <> 0) and not besetzt then
                      if not (Y in Nachbareckeп[Z]) then
                       hinzufuege(Z,i,Y,normal     ,besetzt,Ersatz)

                  until (j = L) or besetzt;

                  Tabelle[i] := alt;
                  Neue       := Neue - 1

               end
```

```
         until (i = Graph.Kopf.Eckenzahl) or besetzt

      until (Neue = 0) or besetzt;

      if besetzt then
         if SWahl[Graph.Ecke[Ersatz].Grad, Graph.Ecke[Y].Grad] = 0 then
            moeglich := false
         else
            if Y > Ersatz then
               verkleinere(Graph, Y, Ersatz)
            else
               verkleinere(Graph, Ersatz, Y);

   until not besetzt or not moeglich

end   (* vernetze *);



procedure normalisiere( (* veraendert *) var Graph    : Graphsatz;
                        (* benutzt    *) var Situation: Graphsatz;
                        (* benutzt    *)     X        : Teilbereich;
                        (* benutzt    *)     Richtung : Option       );

var E, S : Teilbereich;
    i    : integer;
    j    : Valenz;
    Index: 1..MaxTZahl;



procedure schiebe( (* links herum die *) var Ecke: Eckensatz);

var i: 2..MaxGrad;
    E: Bereich;

begin (* schiebe *)
   with Ecke do
      begin

         E := Liste[1];

         for i := 2 to Laenge[Grad] do
            Liste[i - 1] := Liste[i];

         Liste[Laenge[Grad]] := E

      end
end   (* schiebe *);



begin (* normalisiere *)

   with Situation, Kopf do
      for i := 1 to TZahl do
         with Transfer[i] do
            begin
```

```
            E := Empfaenger;

            if Richtung = normal then
               S := Sender
            else
               with Ecke[E] do
                  begin

                     j := 0;

                     repeat
                        j := j + 1
                     until Liste[j] = Sender;

                     if j = 1 then
                        S := Liste[Laenge[Grad]]
                     else
                        S := Liste[j - 1]

                  end;

            Index := Graph.Kopf.TZahl + i;

            Graph.Transfer[Index].Empfaenger := Ecke[E].Partner;
            Graph.Transfer[Index].Sender     := Ecke[S].Partner;
            Graph.Transfer[Index].TTyp       := TTyp;

            SFeld[X].Trans[i].Ebene := Index;
            SFeld[X].Trans[i].Modus := undefiniert

         end;

for i := 1 to Graph.Kopf.Eckenzahl do
   with Graph, Ecke[i] do
      case Grad of

         5, 6, 7, 8, 9: (* nicht schieben *);

         MehrAls4     : while    (Liste[6] <> 0) or (Liste[7] <> 0)
                              or (Liste[8] <> 0) or (Liste[9] <> 0)
                              or (Liste[1] =  0)
                        do
                           schiebe(Ecke[i]);

         MehrAls5     : while    (Liste[7] <> 0) or (Liste[8] <> 0)
                              or (Liste[9] <> 0) or (Liste[1] =  0)
                        do
                           schiebe(Ecke[i]);

         MehrAls6     : while    (Liste[8] <> 0) or (Liste[9] <> 0)
                              or (Liste[1] =  0)
                        do
                           schiebe(Ecke[i]);

         MehrAls7     : while (Liste[9] <> 0) or (Liste[1] = 0) do
                           schiebe(Ecke[i])

      end (* case *)
```

```
end    * normalisiere *)


function Dreieck( (* benutzt *) var Graph: Graphsatz): boolean;

var Kollision: boolean;
    i        : Bereich;
    j        : 0..7;
    k        : 0..5;

begin (* Dreieck *)

  i         := 0;
  Kollision := false;

  repeat

    i := i + 1;

    with Graph, Ecke[i] do
      if (Grad = 7) and (Typ = 3) then
        begin

          j := 0;

          repeat

            j := j + 1;

            if Liste[j] <> 0 then
              with Ecke[Liste[j]] do
                if (Grad = 5) and (Partner = 0) then
                  begin

                    k := 0;

                    repeat

                      k := k + 1;

                      if not (Liste[k] in [0, i]) then
                        Kollision:=Ecke[Liste[k]].Partner<>0

                    until (k = 5) or Kollision

                  end

          until (j = 7) or Kollision

        end

  until (i = Graph.Kopf.Eckenzahl) or Kollision;

  Dreieck := Kollision

end   (* Dreieck *);
```

```
function Fuenfeck(  (* benutzt *) var Graph    : Graphsatz;
                    (* benutzt *)     Y        : Teilbereich;
                    (* benutzt *)     Richtung: Option       ): boolean;

var Figur, Teilfigur: Graphsatz;
    X, Z            : Bereich;
    i               : 0..7;
    R               : Option;
    moeglich        : boolean;

begin (* Fuenfeck *)

  moeglich := false;

  with Graph, Ecke[Y] do
    if Grad = 7 then
      begin

        i := 0;

        repeat

          i := i + 1;
          X := Liste[i];

          if X <> 0 then
            with Ecke[X] do
              if (Grad = 7) and (Typ in [1, 2]) then
                begin

                  if    (Typ = 1) and (Richtung = normal)
                    or (Typ = 2) and (Richtung = reflektiert)
                    then
                      begin
                        Z := Ecke[Y].Liste[i mod 7 + 1];
                        R := reflektiert
                      end
                    else
                      begin

                        if i = 1 then
                          Z := Ecke[Y].Liste[7]
                        else
                          Z := Ecke[Y].Liste[i - 1];

                        R := normal

                      end;

                  if Z <> 0 then
                    if Ecke[Z].Grad = 5 then
                      begin

                        Figur     := Graph;
                        Teilfigur := Sonderfall;

                        pruefe(Figur,X,Y,Z,Teilfigur,1,2,3,
                               R, true, moeglich           )
```

```
                                    end

                                end

                until (i = 7) or moeglich

            end;

        Fuenfeck := moeglich

    end   (* Fuenfeck *);


    begin (* verschmelze *)

        pruefe(Graph, X,Y,Z, Situation, 1,2,3, Richtung, false, moeglich);

        if moeglich then
           begin

              vorbereite(Graph, Situation, Richtung, Neuzahl);

              AlterTeil(Graph, Situation, Richtung);
              NeuerTeil(Graph, Situation, Richtung, Neuzahl);

              Graph.Kopf.Eckenzahl := Neuzahl;

              vernetze(Graph, moeglich);

              if moeglich then
                 begin

                    normalisiere(Graph, Situation, X, Richtung);

                    if Dreieck(Graph) then
                       moeglich := false
                    else
                       moeglich := not Fuenfeck(Graph, X, Richtung)

                 end

           end

    end   (* verschmelze *);


    procedure erweitere( (* erweitert den   *) var Graph : Graphsatz;
                         (* am Rand des     *)     Kreis : Kreisrand;
                         (* rueckt ein den  *)     Rand  : integer;
                         (* zeigt an, ob    *) var Flagge: boolean    );

    var TSituation, Figur: Graphsatz;
        X, Y, Z          : Teilbereich;
        i                : 1..6;
        j                : 0..2;
        k                : Valenz;
```

```
 r                  : Option;
 moeglich           : boolean;
 Nr                 : 1..MaxT;

{- (* erweitere *)

i := 1;

repeat
   i := i + 1
until (Kreis[i] = klein) and not SFeld[i].voll;

j := 0;

repeat
   j := j + 1
until SFeld[i].Trans[j].Modus = undefiniert;

SFeld[i].voll := j = SFeld[i].TZahl;

with Graph, Kopf, Transfer[SFeld[i].Trans[j].Ebene] do
   begin

      TZahl := TZahl - 1;

      X := Empfaenger;
      Z := Sender;

      with Ecke[X] do
         begin

            k := 0;

            repeat
               k := k + 1
            until Liste[k] = Z;

            if k = 1 then
               Y := Liste[Laenge[Grad]]
            else
               Y := Liste[k - 1]

         end

   end;

Flagge := false;

with TSituation.Kopf, Graph.Transfer[SFeld[i].Trans[j].Ebene] do
   for r := normal to reflektiert do
      begin

         reset(T);

         repeat

            lies(T, TSituation);

            if     ((r = normal) or not symmetrisch)
```

```
                    and (    (TTyp = 1)
                         or (TTyp = 2) and (Nummer in [5..7])
                         or (TTyp = 3) and (Nummer in [1, 5..7])
                         or (TTyp = 4) and (Nummer <> 1)
                         or (TTyp = 5) and (Nummer in [2..4, 6, 7]))
                    then
                       begin

                          Figur := Graph;

                          if r = normal then
                             verschmelze(Figur, X, Y, Z, TSituation,
                                         normal, moeglich           )
                          else
                             verschmelze(Figur, X, Z, Y, TSituation,
                                         reflektiert, moeglich      );

                          if moeglich then
                             begin

                                Flagge := true;

                                SFeld[i].Trans[j].Nummer := Nummer;
                                SFeld[i].Trans[j].Modus  := r;

                                entlade(Figur, Kreis, i, Rand + Absatz);

                                reset(T);
                                Nr := Nummer;

                                repeat
                                   lies(T, TSituation)
                                until Nummer = Nr

                             end

                       end

                until eof(T)

             end;

          SFeld[i].Trans[j].Modus := undefiniert;
          SFeld[i].voll           := false

       end   (* erweitere *);



       procedure viel( (* entlaedt *)     Graph: Graphsatz;
                       (* benutzt  *)     Kreis: Kreisrand;
                       (* bestimmt *) var Nr   : integer    );

       var LSituation: Graphsatz;
           gefunden  : boolean;
           i         : 1..6;
```

```
function LIsomorphie( (* zwischen *) var Graph    :Graphsatz;
                      (* und      *) var Teilgraph:Graphsatz;
                      (* benutzt  *)     X        :Teilbereich):boolean;

var Figur, Teilfigur: Graphsatz;
    moeglich        : boolean;
    j               : Valenz;
    L               : 5..MaxGrad;
    r               : Option;
    Z               : Bereich;

begin (* LIsomorphie *)

   moeglich := false;

   with Graph.Ecke[X], Teilgraph do
      if UWahl[Grad, Ecke[1].Grad] <> 0 then
         begin

            j := 0;

            repeat
               j := j + 1
            until Liste[j] = 1;

            L := Laenge[Grad];
            r := undefiniert;

            repeat

               r := succ(r);

               if r = normal then
                  Z := Liste[j mod L + 1]
               else
                  if j = 1 then
                     Z := Liste[L]
                  else
                     Z := Liste[j - 1];

               if Z <> 0 then
                  if UWahl[Graph.Ecke[Z].Grad, Ecke[3].Grad] <> 0 then
                     begin

                        Figur     := Graph;
                        Teilfigur := Teilgraph;

                        pruefe(Figur, X, 1, Z, Teilfigur, 1, 2, 3,
                               r, true, moeglich                    );

                        if moeglich and (Kopf.TZahl > 0) then
                           moeglich := Typpruefung(Figur, Teilfigur, r)

                     end

            until (r = reflektiert) or moeglich

         end;
```

```
   LIsomorphie := moeglich

end   (* LIsomorphie *);



begin (* viel *)

   i := 1;

   repeat
      i         := i + 1;
      gefunden := Kreis[i] in [mittel, regulaer]
   until gefunden or (i = 6);

   Nr := 0;

   if gefunden then
      with Graph, LSituation.Kopf do
         begin

            if Kreis[i] = regulaer then
               Kopf.Ladung := Kopf.Ladung + Normwert;

            reset(L);

            repeat

               lies(L, LSituation);

               if Ladung >= Kopf.Ladung then
                  if Aehnlichkeit(Graph, LSituation) then
                     if LIsomorphie(Graph, LSituation, i) then
                        Nr := Nummer

            until (Nr <> 0) or eof(L)

         end

end   (* viel *);



function zulaessig( (* ist der       *) Kreis: Kreisrand;
                    (* enthalten in *) Reihe: Zeiger     ): boolean;

var gefunden: boolean;
    i       : 2..6;

begin (* zulaessig *)

   for i := 2 to 6 do
      if Kreis[i] = regulaer then
         Kreis[i] := mittel;

   gefunden := false;

   while not gefunden and (Reihe <> nil) do
      begin
```

```
        gefunden := Reihe^.Element = Kreis;
        Reihe    := Reihe^.Vorgaenger
      end;

   zulaessig := gefunden

end   (* zulaessig *);


procedure wenig( (* entlaedt den    *) var Graph : Graphsatz;
                 (* verwendet den   *)     Kreis : Kreisrand;
                 (* indiziert durch *)     X     : Teilbereich;
                 (* verwendet den   *)     Rand  : integer;
                 (* zeigt an, ob    *) var Flagge: boolean      );

var Figur, SSituation: Graphsatz;
    links, rechts    : Teilvalenz;
    Anfang, Ende     : integer;
    Richtung         : Option;
    Z                : 2..6;
    moeglich         : boolean;
    i                : 1..2;


procedure eingrenze( (* von      *)     links : Teilvalenz;
                     (* nach     *)     rechts: Teilvalenz;
                     (* bestimmt *) var Anfang: integer;
                     (* bestimmt *) var Ende  : integer     );



procedure setze( (* je nach *) A, E: integer);

begin (* setze *)
   Anfang := A;
   Ende   := E
end   (* setze *);


begin (* eingrenze *)
   case links of

      MehrAls4  : setze(1, 329);

      MehrAls5  : if rechts in [5, 6] then
                     setze(  0,   0)
                  else
                     setze(151, 329);

      MehrAls6,
      MehrAls7,
      7, 8, 9   : if rechts in [5, 6] then
                     setze(  0,   0)
                  else
                     setze(231, 329);
```

```
      5          : case rechts of

                      MehrAls4  : setze( 1, 150);

                      MehrAls5  : setze( 2, 150);

                      MehrAls6,
                      MehrAls7,
                      7, 8, 9   : setze(11, 150);

                      5         : setze( 1,   1);

                      6         : setze( 2,   8)

                   end (* case *);

      6          : if rechts in [5, 6] then
                      setze(  0,   0)
                   else
                      setze(151, 240)

   end (* case *)
end   (* eingrenze *);



begin (* wenig *)

   links  := Graph.Ecke[(X - 1) mod 5 + 2].Grad;
   rechts := Graph.Ecke[(X + 2) mod 5 + 2].Grad;

   Flagge := false;

   for Richtung := normal to reflektiert do
      begin

         if Richtung = normal then
            begin
               eingrenze(links, rechts, Nr, Ende);
               Z := (X + 2) mod 5 + 2
            end
         else
            begin
               eingrenze(rechts, links, Nr, Ende);
               Z := (X - 1) mod 5 + 2
            end;

         if Ende > 0 then
            with SSituation.Kopf, Figur do
               begin

                  reset(S);

                  repeat

                     lies(S, SSituation);

                     if     (Graph.Kopf.Ladung > Ladung)
```

```
                                end ((Richtung = normal) or not symmetrisch)
                                then
                                   begin

                                      Figur := Graph;

                                      verschmelze(Figur, X, 1, Z, SSituation,
                                                  Richtung, moeglich          );

                                      if moeglich then
                                         begin

                                            Flagge := true;

                                            Kopf.Ladung := Kopf.Ladung - Ladung;
                                            Kopf.TZahl  := Kopf.TZahl  + TZahl;

                                            SFeld[X].Nummer := Nummer;
                                            SFeld[X].TZahl  := TZahl;
                                            SFeld[X].Modus  := Richtur ;
                                            SFeld[X].voll   := TZahl = 0;

                                            for i := 1 to 2 do
                                               with SFeld[X].Trans[i] do
                                                  Modus := undefiniert;

                                            entlade(Figur,Kreis,X,Rand+Absatz);

                                            reset(S);

                                            repeat
                                               lies(S, Figur)
                                            until Figur.Kopf.Nummer = Nummer;

                                            SSituation := Figur

                                         end

                                   end

                             until Nummer = Ende

                          end

                end

          end   (* wenig *);


begin (* entlade *)

   drucke(Output, Graph, Kreis, Rand);
   vervollstaendige(Graph);
   reduziere(Graph, reduzibel, ausTafel);

   if reduzibel then
      with Graph.Kopf do
         if ausTafel then
```

```
            write('I', Seite:1, '-', Nummer:2)
         else
            write(     Seite:2, '-', Nummer:2)
   else
      if Graph.Kopf.Ladung - Graph.Kopf.RZahl * Normwert > 0 then
         begin

            viel(Graph, Kreis, Nr);

            if Nr = 0 then
               if Graph.Kopf.TZahl > 0 then
                  begin

                     erweitere(Graph, Kreis, Rand, moeglich);

                     if not moeglich then
                        write('keine T-Situation passt')

                  end
               else
                  write('***** ist kritisch')
            else
               write(' L', Nr:3)

         end
      else
         begin

            erfolgreich := false;
            i           := Index - 1;

            repeat

               i := i + 1;

               if Kreis[i] = mittel then
                  begin

                     Kreis[i] := klein;

                     if zulaessig(Kreis, Hilfsreihe) then
                        begin
                           Graph.Kopf.RZahl := Graph.Kopf.RZahl - 1;
                           wenig(Graph, Kreis, i, Rand, moeglich);
                           erfolgreich := erfolgreich or moeglich
                        end;

                     Kreis[i] := regulaer;

                     with Graph.Kopf do
                        if Ladung >= Normwert then
                           Ladung := Ladung - Normwert
                        else
                           Ladung := 0

                  end

            until (i = 6) or (Graph.Kopf.Ladung = 0);
```

```
            if not erfolgreich then
               write('entlaedt regulaer')

         end

end   (* entlade *);



procedure umwandle( (* den      *)     Kreis: Kreisrand;
                    (* in einen *) var Graph: Graphsatz  );

var i: 1..6;
    j: Teilvalenz;
    Q: 0..22222;

begin (* umwandle *)
   with Graph, Kopf do
      begin

         Seite       := 1;
         Nummer      := 1;
         Eckenzahl   := 6;
         symmetrisch := false;

         Q :=   ord(Kreis[2]) * 10000
              + ord(Kreis[3]) *  1000
              + ord(Kreis[4]) *   100
              + ord(Kreis[5]) *    10
              + ord(Kreis[6])        ;

               if (Q = 1111) or (Q = 11111) then
            Ladung := 10
         else if Q = 111 then
            Ladung := 20
         else if (Q = 1112) or (Q = 11112) then
            Ladung := 30
         else if (Q =   11) or (Q =  112) or (Q =  1011) or (Q = 1102)
              or (Q = 1121) or (Q = 1122) or (Q = 11122) then
            Ladung := 40
         else
            Ladung := 60;

         RZahl := 0;
         TZahl := 0;

         for i := 1 to 6 do
            with Ecke[i] do
               if i = 1 then
                  begin

                     Grad     := 5;
                     Typ      := 0;
                     Nachbarn := 5;

                     for j := 1 to 5 do
                        Liste[j] := j + 1

                  end
```

```
                else
                   begin

                      case Kreis[i] of

                         fuenf          : Grad := 5;
                         sechs          : Grad := 6;

                         mittel         : begin
                                             Grad  := MehrAls6;
                                             RZahl := RZahl + 1
                                          end;

                         klein, regulaer: Fehler('umwandle        ')

                      end (* case *);

                      Typ      := 0;
                      Nachbarn := 3;

                      Liste[1] := (i - 1) mod 5 + 2;
                      Liste[2] := 1;
                      Liste[3] := (i + 2) mod 5 + 2;

                      for j := 4 to Laenge[Grad] do
                         Liste[j] := 0

                   end

      end
end   (* umwandle *);



function vorhanden( (* ist der *) Kreis: Kreisrand;
                    (* in der  *) Reihe: Zeiger     ): boolean;

var Variante: array[1..,10] of Kreisrand;
    i       : 0..10;
    j       : 2.. 6;
    gefunden: boolean;

begin (* vorhanden *)

   for i := 1 to 5 do
      for j := 2 to 6 do
         Variante[i, (i + j - 3) mod 5 + 2] := Kreis[j];

   for i := 6 to 10 do
      for j := 6 downto 2 do
         Variante[i, (i - j + 5) mod 5 + 2] := Kreis[j];

   gefunden := false;

   while not gefunden and (Reihe <> nil) do
      begin

         i := 0;
```

```
         repeat
            i         := i + 1;
            gefunden := Reihe^.Element = Variante[i]
         until (i = 10) or gefunden;

         Reihe := Reihe^.Vorgaenger

      end;

   vorhanden := gefunden

end   (* vorhanden *);



procedure fuelle( (* den      *)     Kreis     : Kreisrand;
                  (* in die   *) var Hilfsreihe: Zeiger;
                  (* bestimmt *) var moeglich  : boolean    );

var i     : 2..6;
    RZahl : 0..5;
    Flagge: boolean;

begin (* fuelle *)

   moeglich := false;

   if not vorhanden(Kreis, Hilfsreihe) then
      begin

         RZahl := 0;

         for i := 2 to 6 do
            if Kreis[i] = mittel then
               begin

                  RZahl    := RZahl + 1;
                  Kreis[i] := klein;

                  fuelle(Kreis, Hilfsreihe, Flagge);

                  if Flagge then
                     begin
                        moeglich := true;
                        eintrage(Kreis, Hilfsreihe)
                     end;

                  Kreis[i] := mittel

               end;

         moeglich := moeglich or (RZahl <= 1)

      end

end   (* fuelle *);
```

```
procedure erzeuge(  (* erzeugt neuen        *) var Kreis : Kreisrand;
                    (* und addiert ihn zur *) var Reihe : Zeiger;
                    (* und zeigt an, falls *) var fertig: boolean    );

var Uebertrag, neu: boolean;
    i             : 2..6;

begin (* erzeuge *)

   repeat

      Uebertrag := true;

      for i := 6 downto 2 do
         if Uebertrag then
            case Kreis[i] of

               fuenf, sechs   : begin
                                   Kreis[i]  := succ(Kreis[i]);
                                   Uebertrag := false
                                end;

               mittel         : Kreis[i] := fuenf;

               klein, regulaer: Fehler('''erzeuge''       ')

            end (* case *);

      fertig := Uebertrag;

      if not fertig then
         neu := not vorhanden(Kreis, Reihe)

   until fertig or neu;

   if neu then
      eintrage(Kreis, Reihe)

end   (* erzeuge *);



begin (* Entladung *)

   initialisiere(Kreis, Reihe);

   repeat
      umwandle(Kreis, Graph);
      new(Speicher);
      Hilfsreihe := nil;
      fuelle(Kreis, Hilfsreihe, Flagge);
      entlade(Graph, Kreis, 2, 0);
      release(Speicher);
      erzeuge(Kreis, Reihe, fertig)
   until fertig

end   (* Entladung *).
```

8. Output

Die folgenden drei Seiten sind ein typischer Ausschnitt aus dem vom Programm 'Entladung' generierten Output.

Pro Zeile wird jeweils zuerst der Randkreis gelistet. Die Symbole '5', '6' und 'U' zeigen den betreffenden Grad der Randecke an. Ist an einer 5-U-Kante eine S-Situation angebracht, so wird an Stelle von 'U' die Nummer der S-Situation ausgegeben (mit vorangestelltem 'S'); darauf folgt entweder ein 'n' (für normale Anbringung) oder ein 'r' (für reflektierte Anbringung).

Auf den Randkreis folgt die Ladung z der zentralen Fünferecke. Dabei wird die Entladung zu den Ecken 'U' nicht in die Rechnung mit einbezogen, da deren Entladewert zunächst unbestimmt ist (er kann entweder durch eine S-Situation bestimmt werden, oder explizit durch eine reguläre Entladung, doch wird dann das 'U' durch ein 'R' ersetzt).

Schließlich folgt das Ergebnis:
1) Seite und Nummer der reduziblen Teilfigur, oder
2) 'entlaedt regulaer', oder
3) '***** ist kritisch'.
Dies gilt nur, falls der Fall keine Unterfälle (eingerückt) hat.

```
  S231n S237r S261r S244n U    z = 30    16- 5
  S231n S237r S261r S245n U    z = 30    I3-21
  S231n S237r S261r S252n U    z = 30    22- 1
  S231n S237r S261r S253n U    z = 25    22- 1
  S231n S237r S261r S261n U    z = 20    16- 1
  S231n S237r S261r S262n U    z = 20    I3-21
  S231n S237r S261r S263n U    z = 20    I3-29
  S231n S237r S261r S264n U    z = 20    I3-21
  S231n S237r S261r S266n U    z = 20
    S231n S237r S261r S266n S231r    z = 20    16- 1
    S231n S237r S261r S266n S241r    z = 10     1- 1
    S231n S237r S261r S266n S242r    z = 10     1- 1
    S231n S237r S261r S266n S244r    z = 10    16- 5
    S231n S237r S261r S266n S245r    z = 10    I3-21
    S231n S237r S261r S266n S252r    z = 10    22- 1
  S231n S237r S261r S268n U    z = 20    I3-29
  S231n S237r S261r S270n U    z = 15    18- 4
  S231n S237r S261r S271n U    z = 20    18- 6
  S231n S237r S261r S274n U    z = 20
    S231n S237r S261r S274n S253n    z =  5    22-25
    S231n S237r S261r S274n S253r    z =  5    22- 1
  S231n S237r S261r S301n U    z = 20     1- 1
  S231n S237r S261r S302n U    z = 20    22- 1
  S231n S237r S261r S303n U    z = 20    22- 1
  S231n S237r S261r S304n U    z = 20     2- 1
  S231n S237r S261r S305n U    z = 20     1- 1
  S231n S237r S261r S306n U    z = 20     1- 1
  S231n S237r S261r S308n U    z = 20    22- 5
  S231n S237r S261r S309n U    z = 20    52-22
  S231n S237r S261r S310n U    z = 20     1- 1
  S231n S237r S261r S311n U    z = 20     1- 1
  S231n S237r S261r S312n U    z = 15    52-20
  S231n S237r S261r S319n U    z = 20    22- 4
  S231n S237r S261r S320n U    z = 20    22- 4
  S231n S237r S261r S321n U    z = 20    22- 4
  S231n S237r S261r S322n U    z = 20    63- 8
  S231n S237r S261r S323n U    z = 20    22-34
  S231n S237r S261r S324n U    z = 20
    S231n S237r S261r S324n S231r    z = 20    16- 1
    S231n S237r S261r S324n S241r    z = 10     1- 1
    S231n S237r S261r S324n S242r    z = 10     1- 1
    S231n S237r S261r S324n S244r    z = 10    16- 5
    S231n S237r S261r S324n S245r    z = 10    I3-21
    S231n S237r S261r S324n S252r    z = 10    22- 1
  S231n S237r S261r S325n U    z = 20
    S231n S237r S261r S325n S231r    z = 20    16- 1
    S231n S237r S261r S325n S241r    z = 10     1- 1
    S231n S237r S261r S325n S242r    z = 10     1- 1
    S231n S237r S261r S325n S244r    z = 10    16- 5
    S231n S237r S261r S325n S245r    z = 10    I3-21
    S231n S237r S261r S325n S252r    z = 10    22- 1
  S231n S237r S261r S253r U    z = 25    22-25
  S231n S237r S261r S317r U    z = 20     2- 2
  S231n S237r S261r S318r U    z = 20     2- 9
S231n S237r S262r U U    z = 40    16-15
S231n S237r S263r U U    z = 40    I3-11
S231n S237r S266r U U    z = 40    16-12
S231n S237r S268r U U    z = 40    I3- 8
S231n S237r S271r U U    z = 40    16-16
S231n S237r S275r U U    z = 40    16-13
```

```
  S231n S237r S276r U U    z = 40    16-13
  S231n S237r S277r U U    z = 40    16-13
  S231n S237r S278r U U    z = 40    16-13
  S231n S237r S279r U U    z = 40    20-16
  S231n S237r S280r U U    z = 40    20-16
  S231n S237r S281r U U    z = 40    20-16
  S231n S237r S282r U U    z = 40    46-26
  S231n S237r S283r U U    z = 40     1- 2
  S231n S237r S284r U U    z = 40    16-12
  S231n S237r S285r U U    z = 40    I1-33
  S231n S237r S286r U U    z = 40    16-13
  S231n S237r S287r U U    z = 40    16-13
  S231n S237r S288r U U    z = 40    16-13
  S231n S237r S289r U U    z = 40    16-13
  S231n S237r S290r U U    z = 40    16-13
  S231n S237r S301r U U    z = 40     8-32
  S231n S237r S302r U U    z = 40     8-32
  S231n S237r S303r U U    z = 40     8-32
  S231n S237r S304r U U    z = 40     8-32
  S231n S237r S305r U U    z = 40     8-31
  S231n S237r S306r U U    z = 40     8-31
  S231n S237r S307r U U    z = 40     8-31
  S231n S237r S310r U U    z = 40    23-26
  S231n S237r S311r U U    z = 40    23-26
  S231n S237r S312r U U    z = 35    25-22
  S231n S237r S319r U U    z = 40    23- 3
  S231n S237r S320r U U    z = 40    23- 3
  S231n S237r S321r U U    z = 40     1- 3
  S231n S237r S323r U U    z = 40     1- 3
  S231n S237r S324r U U    z = 40     2- 2
  S231n S237r S325r U U    z = 40     2- 2
  S231n S237r S326r U U    z = 40    23- 2
  S231n S237r S327r U U    z = 40    23- 2
  S231n S237r S328r U U    z = 40    23- 2
  S231n S237r S329r U U    z = 40    23- 2
S231n S241r U U U    z = 50
  S231n S241r S231n U U    z = 50     1- 1
  S231n S241r S241n U U    z = 40     1- 1
  S231n S241r S244n U U    z = 40    16- 2
  S231n S241r S245n U U    z = 40    I3-21
  S231n S241r S252n U U    z = 40     1- 1
  S231n S241r S253n U U    z = 35     1- 1
  S231n S241r S261n U U    z = 30     1- 1
  S231n S241r S262n U U    z = 30    I3-21
  S231n S241r S263n U U    z = 30    I1- 8
  S231n S241r S264n U U    z = 30    I3-21
  S231n S241r S266n U U    z = 30
    S231n S241r S266n S232n U    z = 30    16-12
    S231n S241r S266n S233n U    z = 30    16-12
    S231n S241r S266n S234n U    z = 30    16-12
    S231n S241r S266n S235n U    z = 30    16-12
    S231n S241r S266n S236n U    z = 30    16-12
    S231n S241r S266n S237n U    z = 30    16-12
    S231n S241r S266n S249n U    z = 20    16-12
    S231n S241r S266n S250n U    z = 20    I3- 8
    S231n S241r S266n S275n U    z = 10    16-12
    S231n S241r S266n S276n U    z = 10    16-12
    S231n S241r S266n S277n U    z = 10    16-12
    S231n S241r S266n S278n U    z = 10    16-12
```

```
S231n S241r S266n S279n U    z = 10    16-12
S231n S241r S266n S280n U    z = 10    16-12
S231n S241r S266n S281n U    z = 10     1- 3
S231n S241r S266n S282n U    z = 10    16-12
S231n S241r S266n S283n U    z = 10    16-12
S231n S241r S266n S284n U    z = 10    16-13
S231n S241r S266n S285n U    z = 10    I3- 8
S231n S241r S266n S326n U    z = 10    23- 2
S231n S241r S266n S327n U    z = 10    23- 2
S231n S241r S266n S328n U    z = 10    23- 2
S231n S241r S266n S329n U    z = 10    23- 2
S231n S241r S266n S231r U    z = 30    16-12
S231n S241r S266n S234r U    z = 30    16-12
S231n S241r S266n S235r U    z = 30    16-12
S231n S241r S266n S237r U    z = 30    16-13
S231n S241r S266n S241r U    z = 20    16-13
S231n S241r S266n S242r U    z = 20    16-13
S231n S241r S266n S243r U    z = 20    16-13
S231n S241r S266n S244r U    z = 20    16-12
S231n S241r S266n S245r U    z = 20    16-15
S231n S241r S266n S248r U    z = 20    16-12
S231n S241r S266n S250r U    z = 20    16-12
S231n S241r S266n S252r U    z = 20    23- 2
S231n S241r S266n S261r U    z = 10    entlaedt regulaer
S231n S241r S266n S262r U    z = 10    16-15
S231n S241r S266n S263r U    z = 10    16-15
S231n S241r S266n S266r U    z = 10    16-12
S231n S241r S266n S268r U    z = 10    I3- 8
S231n S241r S266n S270r U    z =  5    16-14
S231n S241r S266n S271r U    z = 10    16-16
S231n S241r S266n S275r U    z = 10    16-13
S231n S241r S266n S276r U    z = 10    16-13
S231n S241r S266n S277r U    z = 10    16-13
S231n S241r S266n S278r U    z = 10    16-13
S231n S241r S266n S279r U    z = 10    46-26
S231n S241r S266n S280r U    z = 10    46-27
S231n S241r S266n S281r U    z = 10    46-27
S231n S241r S266n S282r U    z = 10    46-26
S231n S241r S266n S283r U    z = 10    30-15
S231n S241r S266n S284r U    z = 10    16-12
S231n S241r S266n S285r U    z = 10    I3-25
S231n S241r S266n S286r U    z = 10    16-13
S231n S241r S266n S287r U    z = 10    16-13
S231n S241r S266n S288r U    z = 10    16-13
S231n S241r S266n S289r U    z = 10    16-13
S231n S241r S266n S290r U    z = 10    16-13
S231n S241r S266n S301r U    z = 10    23- 6
S231n S241r S266n S302r U    z = 10    23- 6
S231n S241r S266n S303r U    z = 10    23- 6
S231n S241r S266n S304r U    z = 10    23- 6
S231n S241r S266n S305r U    z = 10    23- 4
S231n S241r S266n S306r U    z = 10    23- 4
S231n S241r S266n S307r U    z = 10    23- 4
S231n S241r S266n S310r U    z = 10    23-26
S231n S241r S266n S311r U    z = 10    23-26
S231n S241r S266n S312r U    z =  5    53- 6
S231n S241r S266n S319r U    z = 10    23- 3
S231n S241r S266n S320r U    z = 10    23- 3
S231n S241r S266n S321r U    z = 10     7-31
```