KAPITEL 6

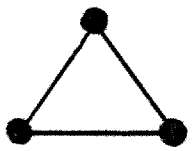
ÜBERPRÜFUNG DES BEWEISES

6.1 Vorgehensweise

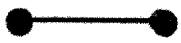
Der Beweis des Vierfarbensatzes von Appel und Haken ist von vergleichsweise einfacher Logik, jedoch von großer kombinatorische Komplexität. Schuld daran ist vor allem die große Zahl von 485 verschiedenen S- und L-Situationen; in der logisch einfachen Entladungsprozedur muß eine Vielzahl von Fällen unterschieden werden. Die sich daraus ergebende große Anzahl von Figuren in der unvermeidlichen Menge U erfordert eine gleich große Anzahl von Reduzibilitätsnachweisen. Auch Reduzibilitätsnachweise sind logisch ziemlich einfach, aber sie erfordern die Behandlung sehr vieler verschiedener Randkreisfärbungen, und für viele dieser Randkreisfärbungen wird eine große Zahl von Kempekettenargumenten benötigt.

Die Reduktionsberechnungen von Hand nachzuvollziehen ist menschenunmöglich. Dies ist leicht einzusehen, wenn man bedenkt, daß zur Durchführung der Reduktionen 1200 Stunden CPU-Zeit moderner Großrechner benötigt wurden. Die Richtigkeit der Reduktionsergebnisse kann praktisch als gesichert gelten. Viele Mathematiker haben mit verschiedenen Programmen und verschiedenen Rechnern dieselben Ergebnisse erzielt. Insbesondere haben Allaire und Swart (3) systematisch alle reduziblen Figuren gelistet, deren Randlänge kleiner als 11 ist. Eine ähnlich erschöpfende Berechnung für Randlängen von 11 und 12 steht zu erwarten.

Von Fortschritten in der Reduktionstheorie ließe sich noch am ehesten eine Vereinfachung des Beweises für den Vierfarbensatz erwarten. Swart versucht zur Zeit, das 5-5-5-Dreieck zu reduzieren (Abb. 6.1). Dieses Dreieck ist in 942 Figuren der unvermeidlichen Menge enthalten. Gelänge es, seine Reduzibilität nachzuweisen, dann könnten diese 942 Figuren aus U entfernt werden. Eine noch dramatischere Vereinfachung wäre möglich, wenn sich die Reduzibilität des 5-5-Paares nachweisen ließe (Abb. 6.1).



5-5-5-Dreieck



5-5-Paar

Abb. 6.1

Die Entladungsprozedur von Appel und Haken läßt sich im Gegensatz zu den Reduktionen von Hand nachprüfen, nicht zuletzt deshalb, weil ihre letzte Version von Hand ausgeführt wurde. Der Großteil der kombinatorischen Details ist in einem ca. 600 Seiten starken Microfiche-Anhang enthalten. Der Autor schätzt, daß eine komplette Überprüfung von Hand mindestens sechs Monate erfordern würde (entsprechende Erfahrungswerte liegen vor).

Bei der Überprüfung eines solchen Beweises kann man sich verschiedene Vorgehensweisen zu eigen machen. Man könnte z.B. die Haltung der "Ungläubigen" einnehmen und versuchen, den Vierfarbensatz durch ein Gegenbeispiel zu widerlegen. Dies ist nicht so abwegig, wie es im ersten Moment klingen mag. Bis zum Erscheinen des Beweises von Appel und Haken haben

mehrere ernst zu nehmende Mathematiker den Vierfarbensatz für falsch gehalten. Edgar Moore z.B. konstruierte Landkarten, die zur Zeit ihrer Entstehung keine der damals bekannten reduziblen Figuren enthielten. Die Färbung einer solchen Karte ist keineswegs trivial. 1977 produzierte Moore eine Karte, die keine reduzible Figur enthält, deren Randlänge kleiner als 12 ist (sie enthält allerdings einen reduziblen 12er). Diese interessante Tatsache zeigt, daß es keine unvermeidliche Menge reduzibler Figuren geben kann, deren maximale Randlänge kleiner als 12 ist. Andererseits haben die Figuren aus der Appel/Hakenschen unvermeidlichen Menge eine maximale Randlänge von 14. Es bleibt also offen, ob dieser Wert noch auf 13 (möglich) oder gar 12 (unwahrscheinlich) erniedrigt werden kann.

Erfolgversprechender wird es wohl sein, sich mit dem Beweis selbst zu befassen, und dort vor allem mit der Entladungsprozedur. Der Beweis zeigt nicht, wie man die unvermeidliche Menge U konstruiert, er weist vielmehr nach, daß U bei der gegebenen Entladungsprozedur P(T, S, L) tatsächlich unvermeidlich ist.

<u>Entladungssatz für P(T, S, L), U:</u> Falls G' U vermeidet, dann entlädt P(T, S, L) G' vollständig.

Da jedes Element von U reduzibel ist, folgt daraus die Nichtexistenz des minimalen Gegenbeispiels G'.

Eine Ecke des Grades i wollen wir im folgenden mit v_i bezeichnen. Ihre Anfangsladung (<u>vor</u> Ausführung der Entladungsprozedur)

sei $z_o(v_i)$, ihre Endladung (nach Ausführung der Entladungsprozedur) sei $z(v_i)$.

Da die Sechserecke zu Beginn der Entladungsprozedur die Ladung $z_o(v_6) = 0$ hat und sie weder Ladung aufnimmt noch abgibt, folgt unmittelbar $z(v_6) = 0$. Es bleibt also noch zu zeigen, daß

(A) $z(v_5) \leq 0$ für jede Fünferecke v_5 aus G' und daß
(B) $z(v_i) \leq 0$ für jede Ecke höheren Grades v_i aus G' ($i \geq 7$).

In dieser Arbeit wird nur Fall (A) behandelt. Es bleibt also noch einiges zu tun, zumal der Fall der v_8 laut Haken der Fall mit der größten kombinatorischen Komplexität ist. Bei der v_5-Untersuchung haben wir schon im letzten Kapitel nach der Zahl μ der Nachbarn höheren Grades unterschieden. Für alle Fälle mit $\mu = 0$ konnten wir zeigen, daß die zentrale v_5 einer reduziblen Figur angehört. Reduzible Figuren fanden sich auch in fünf der zehn Fälle mit $\mu = 1$. Zwei weitere Fälle konnten wir nach Einführung der T-Entladungen vollständig entladen; dies kann sich jedoch infolge der S-Situationen geändert haben.

Von der größten kombinatorischen Komplexität ist der Fall $\mu = 3$; hier treten auch die meisten Fehler auf. Der Autor hat die umfangreichen Fallunterscheidungen bis einschließlich $\mu = 4$ von Hand nachgeprüft und ist dabei auf drei Arten von Fehlern gestoßen: es sind Fälle behandelt worden, die gar nicht auftreten können (ein harmloser Fehler), es sind Fälle falsch behandelt worden, und es sind Fälle nicht behandelt

worden, obwohl sie auftreten können - genau wie bei Kempe.

Allerdings sind die Konsequenzen nicht so bitter wie bei Kempe, denn in den meisten Fällen war eine Reparatur ohne weiteres möglich. Einige Fälle allerdings erwiesen sich als nicht reparierbar; um diese Fehler zu entschärfen, hat Haken seine Entladungsprozedur an einigen Stellen geändert. Es hat sich allerdings weder an der Zahl der T- noch an der Zahl der S- oder L-Situationen etwas geändert. Es wurden lediglich die Definitionen einzelner S- und L-Situationen geändert (11).

Menschen können ermüden; dies gilt sowohl für Appel und Haken bei der Niederschrift der vielen tausend Fallaufzählungen als auch für jeden, der diese Fallaufzählungen von Hand nachprüfen möchte. Computer ermüden nicht, und machen fast niemals Fehler bei der Ausführung eines logisch korrekten Algorithmus (vorausgesetzt, dieser Algorithmus ist richtig implementiert). Es lag deshalb nahe, den Computer nicht nur zur Überprüfung der Reduktionen, sondern auch zur Überprüfung der Entladungsprozedur einzusetzen.

Das im folgenden vorgestellte Programm leistet eine solche Überprüfung. Zwar ist es in der jetzigen Form nur auf die v_5-Entladung anwendbar, doch kann es, so glaubt der Autor, ohne größere Schwierigkeiten (und unter Ausnutzung der bestehenden Datenbestände) für die Überprüfung von Fall (8) modifiziert werden.

6.2 Datenstrukturen

In diesem und den folgenden Abschnitten werden die Datenstrukturen und Algorithmen vorgestellt, die im Programm ENTLADUNG (s. Anhang) Verwendung finden. Dabei handelt es sich um einen Überblick, nicht um eine erschöpfende Darstellung aller programmtechnischen Einzelheiten. Letzteres würde beim Leser eine gründliche Kenntnis der verwendeten Datenbasis voraussetzen, also der unvermeidlichen Menge reduzibler Figuren, der T-, S- und L-Situationen sowie einiger weiterer Tabellen, die im Microfiche-Anhang zum Beweis enthalten sind - insgesamt etwa 2500 individuelle Zeichnungen. Dabei müßte eine Reihe bisher nicht diskutierter Sonderfälle und Ausnahmesitutionen behandelt werden (siehe z.B. (6), S. 439), die zwar für die Logik der Überprüfung irrelevant sind, in einer konkreten Implementierung jedoch berücksichtigt werden müssen.

Es muß erwähnt werden, daß einige der Sonderregeln nicht implementiert worden sind, da dies einen unverhältnismäßig hohen Aufwand erfordert hätte. Ein Beispiel dafür sind die Aufspaltungsregeln bei T-Situationen (siehe Abb. 5.8), die schwierig zu implementieren sind, weil sie einen nicht-lokalen Aspekt in das Problem einbringen. Im allgemeinen ist das Programm ENTLADUNG "eckenbezogen": Ecken werden entfernt, hinzugefügt, miteinander identifiziert, entladen und aufgeladen. Die Kante kommt als Datenstruktur überhaupt nicht vor, das Gebiet (im Falle einer Triangulation immer ein Dreieck) ebenfalls nicht. Das muß nicht so sein, doch hat die Ecke im Vergleich zur Kante oder zum Gebiet die einfachste Datenstruktur.

Bei den wenigen nicht implementierten Bestandteilen des Originalproblems wurde jedoch sorgfältig darauf geachtet, daß sich das Programm stets "auf der sicheren Seite" befand. Dies bedeutet, daß es unter Umständen zu einer Behandlung unmöglicher Fälle kommt, jedoch niemals zu einer Nichtbehandlung möglicher Fälle. In der Praxis heißt das, daß die vom Programm als "kritisch" bezeichneten Fälle noch einmal von Hand nachuntersucht werden müssen, was bei der verschwindenden Anzahl dieser Fälle keine allzu große Belastung darstellt.

Das Programm ist in Pascal geschrieben, einer Programmiersprache, die innerhalb eines kleinen, effizienten Sprachumfangs fortgeschrittene Möglichkeiten zur Strukturierung von Daten und zur Modularisierung logisch getrennter Programmteile bietet. Die Syntax hält sich an den Jensen/Wirth-Standard, wie er in "Pascal - User Manual and Report" definiert ist; damit dürfte die Portabilität gesichert sein. Die folgenden Programmbeispiele verwenden Pascal-Syntax bzw. eine Art informelles Pseudo-Pascal, wenn dadurch algorithmische Strukturen besser verdeutlicht werden.

Bei der Auswahl einer geeigneten Datenstruktur für Graphen muß man von der Information ausgehen, die man darstellen möchte. Die beliebte Adjazenzmatrix z.B., in der das Element (i, j) gleich 1 ist, falls die Ecke i der Ecke j benachbart ist, ansonsten 0, ist zwar einfach zu implementieren, enthält jedoch keinerlei Information über die Grade der Ecken oder über die Anordnung der Nachbarn für eine gegebene Ecke. Der Autor hat sich für die in Abb. 6.2 gezeigten Datenstrukturen entschieden.

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

      MaxGrad  =  9; (* höchster Grad einer Ecke *)
      MaxGraph = 63; (* soviel Ecken hat der größte Graph *)



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

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

     Kopfsatz    = record
                     Seite       : integer;
                     Nummer      : integer;
                     Eckenzahl   : Bereich;
                     symmetrisch: boolean;
                     Ladung      : integer;
                     Vollecken   : Bereich;
                     RZahl       : 0..5;
                     Struktur    : array[5..MaxGrad] of Bereich
                   end;

     Eckenfeld   = array[Teilvalenz] of Bereich;

     Eckensatz   = record
                     Grad    : Valenz;
                     Nachbarn: Valenz;
                     Partner : Bereich;
                     Liste   : Eckenfeld
                   end;

     Graphsatz   = record
                     Kopf : Kopfsatz;
                     Ecke : array[Teilbereich] of Eckensatz
                   end;
```

Abb. 6.2 Datenstrukturen für Graphen

In den S-, L- und T-Situationen kommen keine Ecken der Grade 10 oder 11 vor; deshalb können wir den maximalen Grad mit 9 ansetzen. Dies ergibt fünf vollständig spezifizierte Grade: 5, 6, 7, 8 und 9. Die teilweise spezifizierten Grade werden der Einfachheit halber mit den Zahlen 1 bis 4 gleichgesetzt.

'Kopfsatz' enthält einige notwendige und nützliche Globalinformationen über den Graphen. 'Seite' und 'Nummer' ermöglichen eine eindeutige Identifizierung; sie entsprechen der Tabellenorganisation im Beweis von Appel und Haken. Die 'Eckenzahl' gibt an, aus wieviel Ecken der Graph besteht. 'symmetrisch' wird true, falls die 5-U-Kante, entlang der Ladung transferiert wird, Teil einer Symmetrieachse durch den Graphen ist, ansonsten false. Die 'Ladung' ist die der zentralen Fünferecke des Graphen. 'Vollecken' ist gleich der Zahl der Ecken mit vollständig spezifizierten Graden. 'RZahl' ist gleich der Zahl der Kanten, entlang denen die zentrale Fünferecke regulär entlädt. Das Feld 'Struktur' gibt an, wieviel Fünfer-, Sechser-, Siebener-, Achter- und Neunerecken der Graph enthält.

'Eckensatz' enthält notwendige Lokalinformation über eine Ecke. Neben dem 'Grad' ist die Anzahl der 'Nachbarn' von Interesse. Die Variable 'Partner' wird benötigt, wenn die Ecke mit einer anderen Ecke identifiziert werden soll. Das Feld 'Liste' schließlich enthält die Namen der Nachbarecken (wobei der Name einfach die Ordnungszahl der Eckenbeschreibung innerhalb der Graphbeschreibung ist); die Reihenfolge der Nachbarecken muß einheitlich im ganzen Graphen entweder im oder gegen den Uhrzeigersinn gewählt sein.

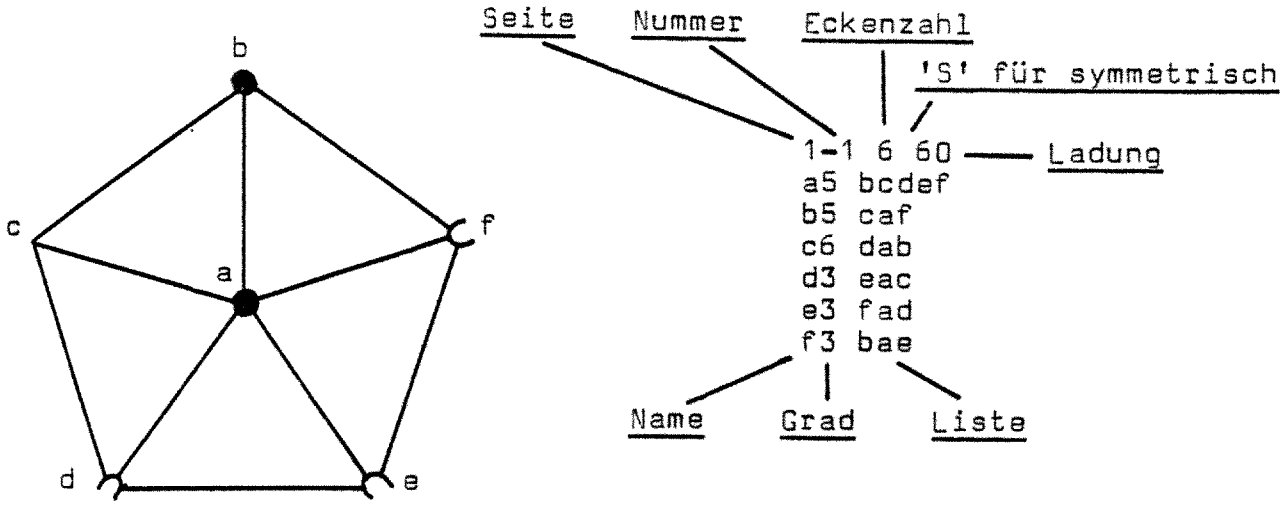


Abb. 6.3 Repräsentation eines Graphen

In Abb. 6.3 ist ein einfacher Graph in geometrischer und analytischer Form dargestellt. Letztere konstituiert in etwas anderer Form auch den Input für das Programm 'Entladung'. Die Benennung der Ecken ist willkürlich, sie muß lediglich mit 'a' beginnen und darf keine Lücken aufweisen (ein fehlendes 'b' etwa). Man beachte, daß die 'Listen' einheitlich gegen den Uhrzeigersinn angelegt sind; die Gegenrichtung wäre ebenfalls möglich gewesen. Richtung und Gegenrichtung korrespondieren zu der Aussage, ob ein Graph in einem anderen Graphen normal oder reflektiert enthalten ist.

Physikalisch hat eine 'Liste' neun Elemente, logisch hat sie soviel Elemente, wie der Grad der Ecke angibt, zu der sie gehört. Die logische Länge der 'Listen' der Ecken a und b ist also 5, die der Ecke c ist 6. Die logische Länge der 'Listen' der teilweise spezifizierten Ecken ist immer gleich der maximalen Länge, also 9. Hat eine Ecke weniger Nachbarn als ihre logische Länge angibt, so werden die restlichen Listenelemente mit Leerstellen (in der Implementierung mit dem Zahlenwert 0) besetzt. Diese Leerstellen entsprechen der Tatsache, daß der Grad der Randkreisecken unbekannt ist.

Die restlichen Daten (wie z.B. 'Vollecken' oder 'Struktur') sind eine Funktion der schon vorgestellten Eingangsdaten. Sie werden deshalb innerhalb des Programms berechnet.

Der Autor hat sich der Mühsal unterzogen, mehrere tausend solcher Darstellungen dem Computer einzugeben. Dabei konnten Eingabefehler natürlich nicht ausbleiben. Daher wurde ein Programm entwickelt, welches die Eingabedaten auf Konsistenz überprüft (dieses Programm ist hier nicht wiedergegeben). Eine solche Überprüfung ist möglich, da in der gewählten Beschreibungsform einige Redundanz vorhanden ist. Die Tatsache beispielsweise, daß die Ecke a fünf Nachbarn hat, kann man entweder direkt der Eckenbeschreibung von a entnehmen oder indirekt, indem man zählt, wie oft a in den Listen der anderen Ecken vorkommt. Beide Werte müssen natürlich übereinstimmen.

6.3 Randkreise

Im folgenden wollen wir unter einem Randkreis stets den Randkreis der zu entladenden Fünferecke verstehen. Die Randkreisecken können Fünferecken, Sechserecken oder Ecken höheren Grades sein; die Ecken höheren Grades brauchen wir nicht weiter zu differenzieren, da der Entladungswert unabhängig vom Grad einer solchen Ecke ist. Randkreise werden wir künftig außer durch Diagramme auch durch Angabe der Grade der Randkreisecken charakterisieren, wobei wir den Grad 5 durch das Symbol '5', den Grad 6 durch das Symbol '6' und höhere Grade durch das Symbol 'U' darstellen werden. Der Randkreis des in Abb. 6.3 dargestellten Graphen läßt sich beispielsweise durch den Ausdruck

5 6 U U U .

charakterisieren.

Wenn wir den Fall (A) des Entladungssatzes überprüfen wollen, dann müssen wir alle möglichen Randkreise untersuchen; entweder entlädt die zentrale Fünferecke vollständig, oder der Graph, zu dem der Randkreis gehört, muß eine reduzible Figur der unvermeidlichen Menge enthalten. Mit dieser Aufgabenstellung können wir bereits das erste 'Programm' formulieren:

erzeuge einen Graphen, bestehend aus v_5 und Randkreis 5 5 5 5 5;

repeat

untersuche den Graphen auf vollständige Entladung oder reduziblen Teilgraphen;

erzeuge einen Graphen, bestehend aus v_5 und dem lexikographisch nächsten Randkreis

until alle Randkreise erzeugt

Die lexikographische Ordnung sei dabei wie folgt definiert:

5 5 5 5 5
5 5 5 5 6
5 5 5 5 U
5 5 5 6 5
5 5 5 6 6
...

Bei fünf "Stellen" und drei möglichen "Werten" pro Stelle erhalten wir so $3^5 = 243$ Randkreise. Die meisten dieser Randkreise sind allerdings zyklische Permutationen und Reflexionen anderer Randkreise. So entsteht z.B. der Randkreis 6 5 U U U aus dem Randkreis 5 6 U U U durch Reflexion und anschließende Rotation um eine Ecke gegen den Uhrzeigersinn.

Es ist klar, daß wir nur die wesentlich verschiedenen Randkreise zu betrachten brauchen. Davon gibt es genau 39:

5 5 5 5 5
5 5 5 5 6
5 5 5 5 U
5 5 5 6 6
5 5 5 6 U
5 5 5 U U
5 5 6 5 6
5 5 6 5 U
5 5 6 6 6
5 5 6 6 U
5 5 6 U 6
5 5 6 U U
5 5 U 5 U
5 5 U 6 U
5 5 U U U
5 6 5 6 6
5 6 5 6 U
5 6 5 U U
5 6 6 5 U
5 6 6 6 6
5 6 6 6 U
5 6 6 U 6
5 6 6 U U
5 6 U 5 U
5 6 U 6 U
5 6 U U 6
5 6 U U U
5 U 5 U U
5 U 6 6 U
5 U 6 U U
5 U U U U

6 6 6 6 6
6 6 6 6 U
6 6 6 U U
6 6 U 6 U
6 6 U U U
6 U 6 U U
6 U U U U

U U U U U

Diese 39 Randkreise werden sukzessive dynamisch abgespeichert. Der Pointer 'Reihe' zeigt dabei immer auf den zuletzt abgespeicherten Randkreis; er wird in der Prozedur 'initialisiere' zu nil gesetzt. Die Prozedur 'initialisiere' erzeugt auch den ersten Randkreis (5 5 5 5 5). In der Prozedur 'erzeuge' wird solange der lexikographisch nächste Randkreis erzeugt, bis dieser sich wesentlich von allen in der 'Reihe' abgespeicherten Randkreisen unterscheidet; ist dies nicht möglich, so setzt die Prozedur die boolesche Variable 'allesprobiert' zu true, andernfalls zu false.

Die Variable 'Kreis' enthält den jeweils zu untersuchenden Randkreis. Um Speicherplatz zu sparen, haben Randkreise einen eigenen Datentyp:

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

var Kreis: Kreisrand;
    Graph: Graphsatz;
```

Der Randkreis wird mit 2 bis 6 indiziert; die 1 bleibt der zentralen Fünferecke vorbehalten. Das Symbol 'U' entspricht dem Kreiswert 'mittel'; die restlichen Kreiswerte werden später behandelt. Der zusätzliche Datentyp macht es erforderlich, in einer Prozedur 'umwandle' den Randkreis in einen Graphen mit zentraler Fünferecke sowie entsprechenden Nachbarecken umzuwandeln. In derselben Prozedur wird auch die Anfangsladung der Zentralecke bestimmt; diese wird in den meisten Fällen 60 sein, wenn jedoch die Lemmata (5-6-6) oder (6-6-6) zur Anwendung kommen, kann sich dieser Wert auch erniedrigen.

Die eigentliche Arbeit geschieht in der Prozedur 'entlade'. Diese versucht, die Zentralecke vollständig zu entladen bzw. einen reduziblen Teilgraphen zu finden; das Ergebnis wird auf Drucker bzw. Monitor ausgegeben.

Wir sind nunmehr in der Lage, den anfänglichen Programmentwurf zu formalisieren:

```
begin (* Entladung *)
   initialisiere(Kreis, Reihe);
   repeat
      umwandle(Kreis, Graph);
      entlade(Graph);
      erzeuge(Kreis, Reihe, allesprobiert)
   until allesprobiert
end   (* Entladung *).
```

6.4 Der Verschmelzungsalgorithmus

Abb. 6.4 zeigt eine S- und eine L-Situation.

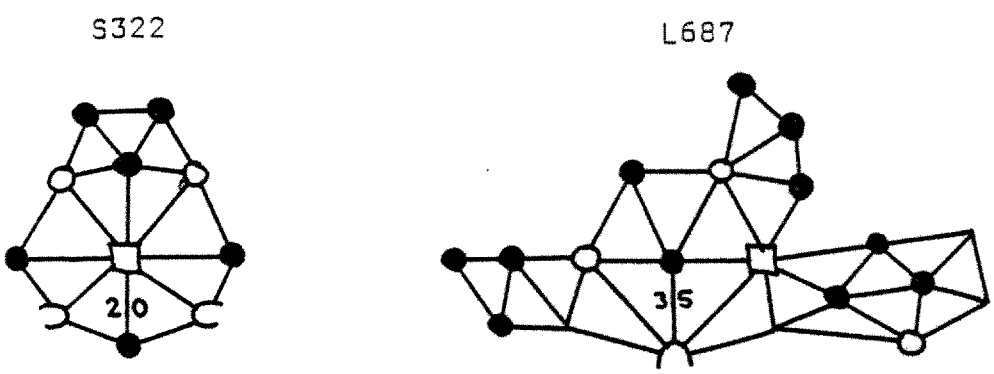


Abb. 6.4 S- und L-Situation

In den Diagrammen, welche die S- und L-Situationen definieren, ist jeweils eine 5-U-Kante x durch Angabe eines Entladungswertes hervorgehoben. In Abb. 6.4 ist dieser Wert gleich 20 bei der S-Situation Nr. 322 und gleich 35 bei der L-Situation Nr. 687. S322 ist übrigens ein Beispiel für eine symmetrische Figur.

Sei G eine beliebige Triangulation, und y eine Kante in G. Dann sagen wir, daß eine S-Situation C an y angebracht (attached) ist, falls C so in G enthalten ist, daß ihre hervorgehobene Kante x mit y identisch ist (diese Definition geht für L-Situationen analog). Intuitiv kann man sich das so vorstellen, daß zwei Graphen so übereinander gelegt werden, daß ihre beiden Kanten x und y aufeinander zu liegen kommen. Dies erzwingt meistens die paarweise Identifizierung weiterer Ecken der beiden Graphen. Wir sagen, daß eine S-Situation C an y angebracht werden kann, falls die miteinander identifizierten Ecken aus C und G denselben Grad haben. Sind die

beiden Graphen derart "verschmolzen", so haben wir einen neuen, nicht notwendigerweise größeren Graph erhalten. Aufgabe des in diesem Abschnitt vorgestellten "Verschmelzungsalgorithmus" ist es festzustellen, ob eine gegebene S- oder L-Situation an einer gegebenen Kante angebracht werden kann, und bejahendenfalls diese Verschmelzung durchzuführen. Abb. 6.5 zeigt ein Beispiel für eine solche Verschmelzung.

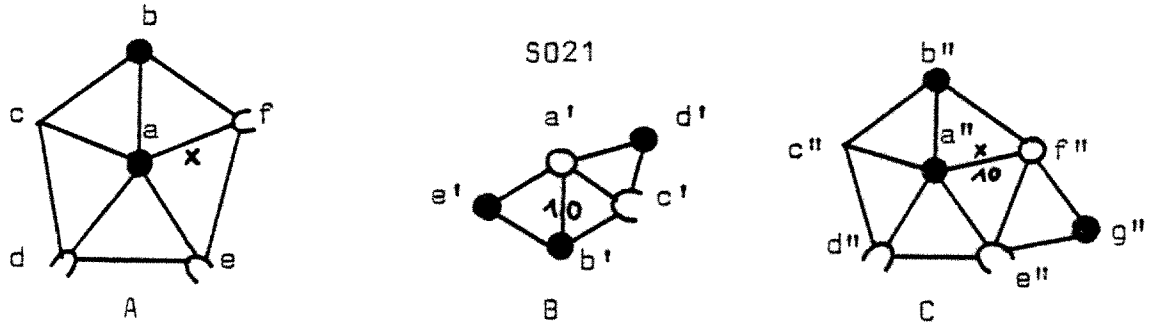
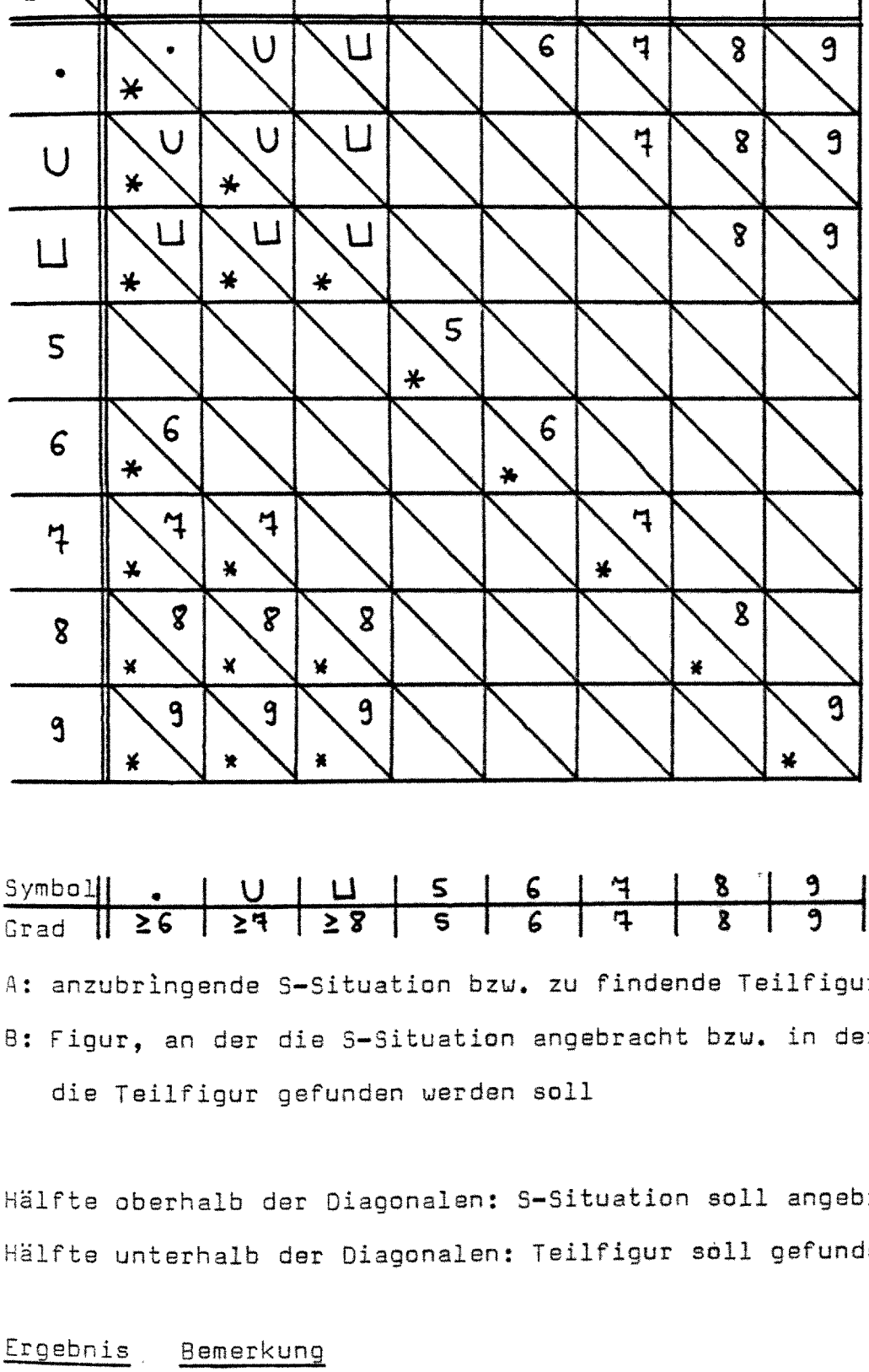


Abb. 6.5 Anbringen einer S-Situation

In Abb. 6.5 wird die S-Situation Nr. 21 an der Kante x der Figur A angebracht. Die Ecke a wird mit der Ecke b' identifiziert, b mit e', e mit c' und f mit a'. Diese Identifikationen sind zulässig, da die jeweils betroffenen Ecken denselben Grad haben bzw. - wie bei f und a' - der Gradbereich der einen Ecke den der anderen einschließt. Eine genaue Darstellung, wann zwei Ecken miteinander identifiziert werden können, findet sich in Abb. 6.6; dabei interessiert uns vorerst nur die Hälfte oberhalb der Diagonalen - diese enthält den Grad, der in der aus der Verschmelzung hervorgegangenen Figur (in Abb. 6.5 ist dies C) verwendet wird.

In Abb. 6.5 ist die S-Situation <u>normal</u> angebracht worden. Wäre b eine Ecke höheren Grades und e eine Fünferecke gewesen, dann wären b mit c' und e mit e' identifiziert worden - die

Abb. 6.6 Identifizierungsmatrix



Symbol	·	U	⊔	5	6	7	8	9
Grad	≥6	≥7	≥8	5	6	7	8	9

A: anzubringende S-Situation bzw. zu findende Teilfigur

B: Figur, an der die S-Situation angebracht bzw. in der die Teilfigur gefunden werden soll

Hälfte oberhalb der Diagonalen: S-Situation soll angebracht werden

Hälfte unterhalb der Diagonalen: Teilfigur soll gefunden werden

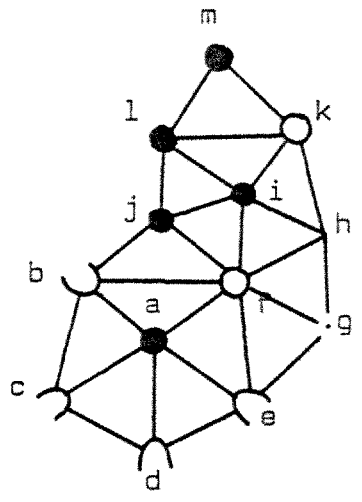
Ergebnis	Bemerkung
leer	die beiden Ecken können nicht miteinander identifiziert werden
*	die beiden Ecken können miteinander identifiziert werden
Symbol	resultierender Grad siehe Symbol/Grad-Tabelle

S-Situation wäre reflektiert angebracht worden.

Betrachten wir nun ein komplizierteres Beispiel, um uns die Mechanismen des Verschmelzungsalgorithmus zu verdeutlichen. Die in Abb. 6.7 abgebildete S-Situation soll an der a-b-Kante der ebenfalls dort abgebildeten Figur angebracht werden. Die Unterstreichungen der Listen entsprechen in ihrer Länge der jeweiligen logischen Listenlä ge (wir erinnern uns: diese war gleich dem Grad der betreffenden Ecke, mit Ausnahme der teilweise spezifizierten Ecken, wo sie gleich 9 war).

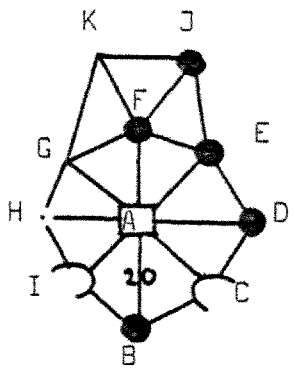
Miteinander identifizierte Ecken wollen wir auch als gepaarte Ecken bezeichnen. Aus der Aufgabenstellung ergeben sich bereits die Paarungen (a,8) und (b,A). Da wir außerdem die S-Situation normal anbringen wollen, erhalten wir die Paarung (f,C). Damit haben wir ein Gebiet der Figur mit einem Gebiet der S-Situation identifiziert, und daraus ergeben sich wegen des Zusammenhangs sowohl der Figur als auch der S-Situation alle anderen Paarungen zwangsläufig.

Es ist die Aufgabe der Prozedur 'pruefe', solche Paarungen herzustellen, indem die Namen der gepaarten Ecken in das jeweils gegnerische 'Partner'-Feld eingetragen werden. Ein leeres 'Partner'-Feld zeigt an, daß die betreffende Ecke mit keiner anderen Ecke gepaart wurde oder werden konnte. Die eigentliche Paarung geschieht in der Prozedur 'verbinde'. Hier werden zuerst die Partner eingetragen, dann wird die Identifizierungsmatrix (Abb. 6.6) ausgewertet. Sind die Grade der zu paarenden Ecken inkompatibel, so wird der Grad der Figurecke gleich Null gesetzt. Dies wird von 'pruefe' erkannt,



Figur

Ecke	Partner	Liste
a	B	bcdef
b	A	cafj
c	I	dab
d		eac
e		gfad
f	C	aeghijb
g		hfe
h		kifg
i		fhklj
j	D	bfil
k		mlih
l		jikm
m		lk



S-Situation

Ecke	Partner	Liste
A	b	BCDEFGHI
B	a	CAI
C	f	DAB
D	j	EAC
E		JFAD
F		AEJKG
G		HAFK
H		IAG
I	c	BAH
J		KFE
K		GFJ

Abb. 6.7 Ergebnis der Prozedur 'pruefe'

und der Verschmelzungsalgorithmus terminiert. Sind die Grade jedoch kompatibel, so wird der in der Matrix gefundene Wert zum neuen Grad der Figurecke.

So hat z.B. der Prozeduraufruf 'verbinde(b, A)' zur Folge, daß A in das Partnerfeld von b und b in das Partnerfeld von A eingetragen werden. Der Grad von b war vor der Paarung nur teilweise spezifiziert; nach der Paarung hat er den spezifischen Wert 8.

Der Prozedur 'pruefe' wird von außen die Kante vorgegeben, an der die S-Situation angebracht werden soll (Ecken a und b). Der Parameter 'Richtung' bestimmt die dritte Ecke (f, falls Richtung = normal, und c, falls Richtung = reflektiert). Damit ist ein Dreieck der Figur mit einem (ebenfalls von außen übergebenen) Dreieck der S-Situation identifiziert. Die Prozedur 'pruefe' gewinnt neue Paarungen, indem sie die Randkreise bereits gepaarter Ecken zur Deckung bringt. Dazu müssen die beiden Randkreise eine gemeinsame, d.h. bereits gepaarte, Ecke aufweisen. Diese existiert immer, weil wegen der triangularen Struktur sowohl der Figur als auch der S-Situation jede gepaarte Ecke zwei Randecken hat, die ebenfalls gepaart sind und mit denen sie ein Dreieck bildet.

Die Prozedur 'pruefe' untersucht die Randkreise aller bereits gepaarten Ecken der Figur. a ist die erste derartige Ecke. Ihr Partner ist B. Es muß jetzt also versucht werden, in den Randkreisen von a und B (implementiert als 'Listen') eine gemeinsame Ecke zu finden. Hierzu ruft 'pruefe' die Prozedur 'positioniere' auf. 'positioniere' sieht, daß bereits die

erste Randecke von a, nämlich b, einen Partner hat, nämlich A. Jetzt braucht A nur noch im Randkreis von B gefunden zu werden, und die beiden Randkreise sind korrekt positioniert:

Ecke	Partner	Liste
a	B	bcdef
B	a	AI C

'positioniere' gibt nun die Kontrolle an 'pruefe' zurück. 'pruefe' paart durch Aufruf von 'verbinde' die in derselben Spalte befindlichen Listenelemente, sofern diese nicht bereits gepaart oder leer sind. Dadurch gewinnen wir die neue Paarung (c,I), deren Richtigkeit der Leser leicht an Hand der Diagramme von Abb. 6.7 nachprüfen kann.

Die obige Darstellung gilt für den Fall, daß Richtung = normal ist. Wäre Richtung = reflektiert gewesen, dann wären die beiden Randkreise wie folgt positioniert worden:

Ecke	Partner	Liste
a	B	bcdef
B	a	AC I

Die neue Paarung wäre dann (f,I) gewesen.

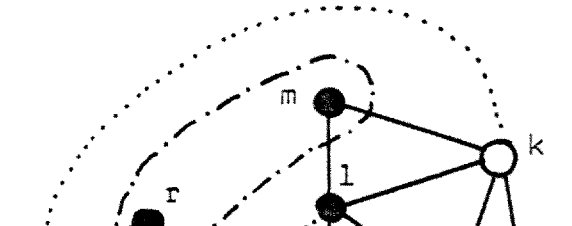
'pruefe' macht so viele Durchgänge durch die Ecken der Figur, bis in einem Durchgang keine neuen Paarungen mehr aufgetreten sind. Bei unserem Beispiel treten zwei neue Paarungen auf - (c,I) und (j,D).

Da der Verschmelzungsalgorithmus die Figur mit der S-Situation zu einer neuen Figur verschmelzen soll, vergrößern wir jetzt die Figur um diejenigen Ecken der S-Situation, die ohne Partner geblieben sind. Dies sind die Ecken E, F, G, H, J und K; in der vergrößerten Figur tragen sie die Namen n, o, p, q, r und s. Diese Vergrößerung wird von der Prozedur 'vorbereite' durchgeführt.

Danach transferieren die Prozeduren 'AlterTeil' und 'NeuerTeil' die Randkreise der Ecken der S-Situation zu den Partnerecken der Figur. Die Prozedur 'AlterTeil' befaßt sich mit den 'alten' Ecken a bis m; für jede der gepaarten Ecken (in unserem Beispiel a, b, c, f und j) wird der Randkreis über den Randkreis des jeweiligen Partners positioniert - genau so, wie wir es bereits durchgeführt haben. Steht nun in derselben Spalte ein nicht-leeres Listenelement aus der S-Situation über einem leeren Listenelement aus der Figur, so wird das Listenelement aus der S-Situation in die Figur übernommen. Abb. 6.8 zeigt die derart vergrößerte Figur.

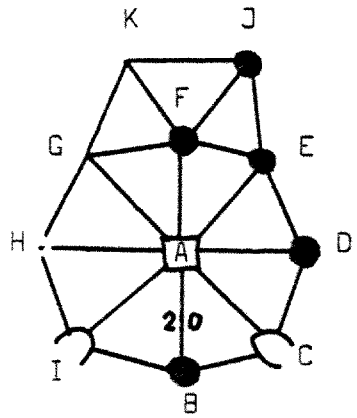
Die Diagramme machen deutlich, was die Prozedur 'pruefe' nicht zu leisten vermochte: die mit der Verschmelzung entstandenen neuen Nachbarschaftsrelationen sind noch nicht erkannt worden. Diese neuen Relationen sind im Diagramm durch gepunktete Linien angedeutet; sie ergeben sich aus der Forderung, daß die Figur trianguliert sein muß.

Da j eine Fünferecke ist, muß l zu n benachbart sein, denn sonst läge keine Triangulation vor. Daraus wiederum folgt, daß n und m benachbart sind, denn auch l ist eine Fünferecke. Dasselbe Argument führt zur Nachbarschaft von l und r.



Figur

Ecke	Partner	Liste
a	B	bcdef
b	A	cafjnopq
c	I	dabq
d		eac
e		gfad
f	C	aeghijb
g		hfe
h		kifg
i		fhklj
j	D	bfiln
k		mlih
l		jikm
m		lk
n	E	robj
o	F	bnrsp
p	G	qbos
q	H	cbp
r	J	son
s	K	por



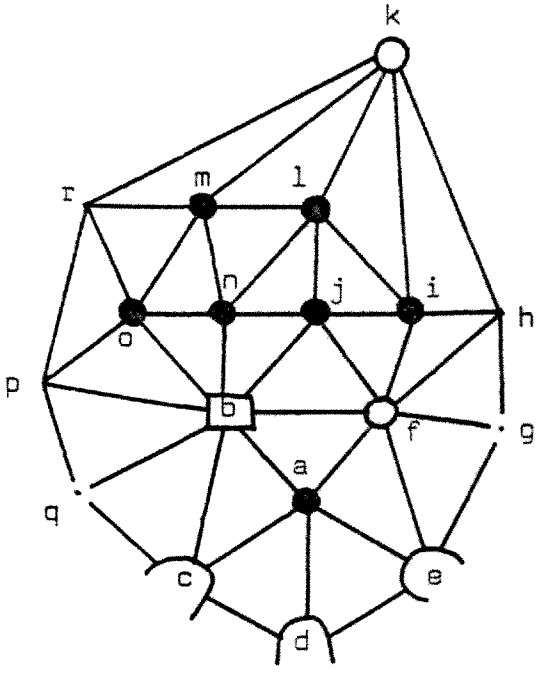
S-Situation

Abb. 6.8 Vergrößerte Figur

Die gleichzeitige Nachbarschaft von n und m bzw. von l und r kann aber nur dann die Bedingung der Ebenheit erfüllen, wenn die Ecken m und r identisch sind. Die strichpunktierte Hülle soll diese Identität symbolisieren. Wären m und r verschiedenen Grades, so würde der Verschmelzungsalgorithmus an dieser Stelle terminieren. In unserem Beispiel können jedoch m und r gepaart werden; in der Figurbeschreibung wird die 'neue' Ecke r

entfernt, und alle ihr folgenden Ecken "rücken eins auf". Die Bezugnahmen auf r in den Listen der anderen Ecken werden durch Bezugnahmen auf die mit r identische 'alte' Ecke, also m, ersetzt. Zum Schluß folgt noch aus der Identität von m und r, daß s und k benachbart sind.

Die soeben erläuterte Triangulierung der vergrößerten Figur wird von der Prozedur 'vernetze' wahrgenommen. 'vernetze' ruft dabei die Prozeduren 'hinzufuege' und 'verkleinere' auf. 'hinzufuege' trägt die sich aus der Triangulierung ergebenden neuen Nachbarschaftsrelationen in die Randkreise (Listen) der betroffenen Ecken ein, während 'verkleinere' überzählige Ecken (d.h. wegen Identität doppelt vorhandene Eckenbeschreibungen) aus der Figur entfernt und die dadurch entstandene "Lücke" durch "Aufrücken" der lexikographisch folgenden Ecken schließt. Somit sieht unsere endgültige Figur wie in Abb. 6.9 dargestellt aus.



Ecke	Liste	Ecke	Liste
a	bcdef	j	bfiln
b	cafjnopq	k	mlih r
c	dabq	l	jikmn
d	eac	m	lkron
e	gfad	n	mobjl
f	aeghijb	o	bnmrp
g	hfe	p	qbor
h	kifg	q	cbp
i	fhklj	r	pomk

Abb. 6.9 Nach der Verschmelzung

Das hier dargestellte Beispiel ist bewußt ausgewählt worden, um eine nachträgliche Triangulierung zu zeigen. In der Praxis treten solche Fälle selten auf.

Wir sind nun in der Lage, den Verschmelzungsalgorithmus in seinen Hauptbestandteilen zu skizzieren:

```
begin (* verschmelze *)

   'pruefe', ob Figur und S-Situation miteinander verschmol-
   zen werden können (falls ja, moeglich := true, andernfalls
   moeglich := false);

   if moeglich then
      begin

         vergrößere die Figur um die Zahl der partnerlos
         gebliebenen Ecken der S-Situation ('vorbereite');

         ergänze die Randkreise der alten Figurecken durch
         die Randkreise ihrer Partner ('AlterTeil');

         übernehme als Randkreise der neuen Figurecken die
         Randkreise ihrer Partner ('NeuerTeil');

         trianguliere die vergrößerte Figur ('vernetze') -
         falls möglich, moeglich := true, andernfalls
         moeglich := false

      end

end   (* verschmelze *);
```

Die Prozedur 'verschmelze' setzt die Flagge 'moeglich' (Ausgabeparameter), je nachdem ob die Verschmelzung gelungen ist oder nicht. Ist sie gelungen, so enthält der Ein-/Ausgabeparameter 'Graph' die aus der Verschmelzung von Figur und S-Situation entstandene neue Figur.

6.5 Auffinden reduzibler Teilfiguren

Für die Aufgabe, in einer Figur eine Teilfigur zu finden, können wir Teile des in 6.4 vorgestellten Verschmelzungsalgorithmus benutzen. Wir hatten gesehen, daß die Prozedur 'pruefe' die Ecken zweier Figuren paart, soweit dies ohne zusätzliche Triangulierung möglich ist.

An Stelle einer Figur und einer S-Situation übergeben wir nun der Prozedur 'pruefe' eine Figur und eine zu findende Teilfigur als Parameter. Sind nach Beendigung von 'pruefe' alle Ecken der Teilfigur gepaart, so wissen wir, daß die Teilfigur vollständig in der Figur enthalten ist. Ist die Teilfigur reduzibel, dann ist auch die sie enthaltende Figur reduzibel.

Nach diesem Muster arbeitet die Prozedur 'reduziere'. Sie 'prueft' für eine gegebene Figur, ob diese ein Element der unvermeidlichen Menge reduzibler Figuren enthält. Ist dies der Fall, so wird die Flagge 'reduzibel' zu true gesetzt und 'Seite' und 'Nummer' der reduzierenden Figur werden ausgegeben; andernfalls wird 'reduzibel' zu false gesetzt.

Bei der Suche nach einer Teilfigur wird übrigens im Gegensatz zum Verschmelzungsalgorithmus nicht von der Hälfte oberhalb der Diagonalen, sondern von der Hälfte unterhalb der Diagonalen der Identifizierungsmatrix (Abb. 6.6) Gebrauch gemacht.

6.6 Die Prozedur 'entlade'

Die Prozedur 'entlade' vereinigt die bisher vorgestellten Algorithmen zur Verschmelzung und zum Auffinden von Teilfiguren:

```
begin (* entlade *)

   drucke den Randkreis (siehe z.B. Output im Anhang);
   reduziere den Graphen;

   if reduzibel then
      drucke Seite und Nummer der reduziblen Teilfigur (z.B. 59-15)
   else
      if Zentralecke nicht entladbar durch reguläre Entladg. then
         begin

            suche nach L-Situation (Prozedur 'viel');

            if erfolglos then
               drucke '***** ist kritisch'
            else
               drucke Nr. der L-Situation (z.B. L411)

         end
      else
         begin

            i := 1;

            repeat

               i := i + 1;

               if Ecke i gehört zu Kante regulärer Entladg. then
                  begin
                     bringe alle möglichen S-Situationen an
                     dieser Kante an und 'entlade' den neuen
                     Graphen (rekursiver Aufruf!);
                     mache Kante wieder zu regulärer Kante
                  end

            until (i = 6) or Graph entladen;

            if keine S-Situation konnte angebracht werden then
               drucke('entlaedt regulaer')

         end

end   (* entlade *);
```

6.7 Ergebnisse

Wegen des hohen Rechenaufwandes konnte das Programm in der zur Verfügung stehenden Zeit nicht komplett durchgerechnet werden. Trotz einer Vielzahl implementierter und hier nicht im einzelnen wiedergegebener Effizienzkriterien, die die Berechnung essentiell gleicher Situationen vermeiden sollen, dürfte das Programm immer noch einige zehn Stunden CPU-Zeit in Anspruch nehmen.

Mit Hilfe zusätzlicher Symmetrieüberlegungen und unter Ausnutzung bestimmter Spezialregeln könnte die Laufzeit sicher noch verringert werden, jedoch nur unter Inkaufnahme einer nicht durch die Problemstellung bedingten Komplizierung des Programms. Kompliziertheit kann höhere Effizienz bedeuten, höhere Sicherheit bringt sie nicht. Im Gegenteil - in dem Bemühen, möglichst effizient zu arbeiten, könnten wir nur allzuleicht Fälle übersehen. Daher wurde beim Programmentwurf der Sicherheit stets mehr Gewicht eingeräumt als der Effizienz. Nicht zuletzt darum ist auch Pascal als Programmiersprache gewählt worden, und nicht etwa Assembler (die Reduktionsprogramme von Appel und Haken sind in IBM-Assembler geschrieben).

In dem bisher produzierten voluminösen Output sind nur sehr wenige 'kritische' Fälle enthalten. Darunter sind auch die bereits von Hand gefundenen Fehler; die anderen 'kritischen' Fälle sind alle auf die Nichtimplementierung gewisser Spezialregeln zurückzuführen.

Es sind also (bis jetzt) keine neuen Fehler gefunden worden. Dies stellt insofern keine Überraschung da, als vier Fünftel des Falles (A) des Entladungssatzes vom Autor schon vor Beginn dieser Arbeit von Hand überprüft worden waren. Der Autor wird, sobald Fall (A) vollständig gerechnet worden ist, das Programm auf den bisher nicht überprüften Fall (B) (Ecken höheren Grades) ausdehnen. Hier ist besonders der Fall der Achterecke interessant, da dieser nach Hakens eigenen Worten der kombinatorisch schwierigste der ganzen Entladungsprozedur ist. Ist auch Fall (B) durchgerechnet, so ist die Entladungsprozedur von Appel und Haken verifiziert (oder falsifiziert, je nach Ergebnis).