Universität Frankfurt / Main
FB 20, Informatik
Wintersemester 1995 / 1996

Seminar: „Wissensbasierte Systeme“
Prof. Dr. M. Schmidt-Schauß

STRUKTURANALYSE VON ELEKTRONISCHEN SCHALTKREISEN IN EINEM DEDUKTIONSSYSTEM

Heiner Bruns

1 MOTIVATION / EINFÜHRUNG 2

2 EINE ERSTE METHODE ZUR SCHALTKREISANALYSE 3

2.1 Logische Repräsentation von Schaltkreisen 3
2.1.1 Allgemeine Syntax 3
2.1.2 Beispiel für einen einfachen Schaltkreis 3
2.1.3 Weiterführende Überlegungen und Prädikate höherer Ordnung 4

2.2 Auffinden von Strukturen in einem Schaltkreis 6
2.2.1 Reihen-, Parallel-, Stern- und Dreieckschaltungen 6
2.2.2 Schaltkreise mit rekursiven Strukturen 7

3 EIN NEUES VERFAHREN ZUR SCHALTKREISANALYSE 9

3.1 Die neue Repräsentationsmethode für Schaltkreise 9
3.1.1 Member-Prädikate 9
3.1.2 Das Sub-Circuit-Konzept 10
3.1.3 Negation von Prädikaten 11

3.2 Zwei Analyse-Verfahren für Schaltkreise 11
3.2.1 Die Bottom-Up-Methode für Impedanz-Netzwerke 11
3.2.2 Die Top-Down-Methode für elektronische Schaltkreise 16

3.3 Methoden zur Erweiterung 21

4 ZUSAMMENFASSENDE BETRACHTUNG DER METHODEN 23

5 VERWENDETE LITERATUR 23

6 ANHANG - BEISPIEL EINER BOTTOM-UP-ANALYSE 24

1 Motivation / Einführung

Eine wichtige Aufgabe im Bereich der Elektrotechnik ist die Analyse von elektronischen Schaltkreisen. Häufig kommt es vor, daß ein Ingenieur mit dem Plan eines Schaltkreises konfrontiert wird, dessen genaue Funktion ihm zunächst unbekannt oder nur teilweise bekannt ist. Einen solchen Schaltkreis gilt es nun zu analysieren, d. h. seinen Aufbau und seine Funktionsweise herauszufinden.

Normalerweise wird ein Ingenieur zunächst versuchen Strukturen, also Teilabschnitte des Schaltkreises mit ihm bekannter Funktionsweise aufzufinden. In einem zweiten Schritt versucht er dann, die einzelnen Funktionsblöcke miteinander in Beziehung zu setzen, um auf diese Weise den gesamten Schaltkreis zu verstehen. Diese Vorgehensweise beruht auf der Annahme, daß jeder elektronische Schaltkreis eine zielorientierte Verknüpfung bekannter Funktionsblöcke darstellt.

Die Analyse eines solchen Schaltkreises entspricht damit prinzipiell einem Wiederauffinden der ursprünglichen Ideen und Absichten des Schaltkreisdesigners. Aus diesem Grund kann eine gewisse Analogie zwischen elektronischen Schaltkreisen und natürlicher Sprache festgestellt werden: Beide enthalten in versteckter Form Informationen über die Absichten und Ideen des Sprechers bzw. des Designers. Darüber hinaus stellt die Zeichnung eines Schaltplans nicht nur einen physikalischen Schaltkreis als solchen, sondern auch eine Schriftsprache zur Kommunikation von Elektrotechnikern untereinander dar. Diese Analogie zwischen elektronischen Schaltkreisen und Sprachen stellt den Grundgedanken des neuen Konzepts zur Schaltkreisanalyse dar, welches ich im zweiten Teil dieser Arbeit vorstellen werde.

„Strukturanalyse von elektronischen Schaltkreisen in einem Deduktionssystem“ ist ein Forschungspapier aus dem Jahr 1986 von Takushi Tanaka am „National Language Research Institute“ in Tokio. Es beschäftigt sich mit konkreten Methoden zur Schaltkreisanalyse in Duck, einem auf der funktionalen Programmiersprache Lisp basierenden Deduktionssystem. Diese Methoden stelle ich hier vor.

Zunächst wird eine einfache Methode der logischen Repräsentation von elektronischen Schaltkreisen beschrieben, welche zum einen bereits sehr gut zur späteren Schaltkreisanalyse in Duck geeignet ist, und zum anderen recht anschaulich die Funktionsweise von Duck demonstriert. Daran schließen sich einige Überlegungen zum Auffinden von Strukturen in einem Schaltkreis an, die elementare Bedeutung haben und damit auch für die später vorgestellte Methode noch wichtig sein werden. Die Implementierung dieser Überlegungen stellt die erste vorgestellte Methode zur automatischen Schaltkreisanalyse in Duck dar, weist jedoch noch einige, zum Teil sehr unerwünschte, Schwachpunkte auf.

Um die aufgetretenen Probleme zu umgehen, wird eine neue Methode zur Schaltkreisanalyse in Duck eingeführt, die die bereits erwähnte Analogie zwischen elektronischen Schaltkreisen und Sprachen nutzt. Diese neue Methode verwendet auch eine neue logische Darstellung von Schaltkreisen, die auf den ersten Blick um einiges komplizierter als die vorangegangene erscheint, sich aber im Nachhinein als enorm leistungsfähig erweisen wird.

Mit einer zusammenfassenden Betrachtung der vorgestellten Methoden werde ich meine Arbeit schließen.

2 Eine erste Methode zur Schaltkreisanalyse

Im folgenden wird eine erste Methode zur automatischen Analyse von Schaltkreisen in Duck beschrieben. Bevor wir uns jedoch der eigentlichen Schaltkreisanalyse zuwenden können muß erst einmal definiert werden, wie Schaltkreise logisch in Duck darzustellen sind.

2.1 Logische Repräsentation von Schaltkreisen

2.1.1 Allgemeine Syntax

Um einen gegebenen Schaltkreis dem Deduktionssystem Duck mitteilen zu können, müssen wir eine logische Repräsentation für ihn finden. Wir verwenden bestimmte Prädikate für die jeweiligen elektronischen Bauteile, so z. B. das Prädikat `RESISTOR` zur Bezeichnung eines Widerstands, `DIODE` für eine Diode, `CAPACITOR` für einen Kondensator, usw. Diese Prädikate werden in symbolischen Ausdrücken verwendet, wobei ein solcher Ausdruck (`Term`) wie folgt definiert ist:

```
Term ::= (Prädikat Term)  |
         (Funktion Term)  |
         Konstante        |
         Term Term
```

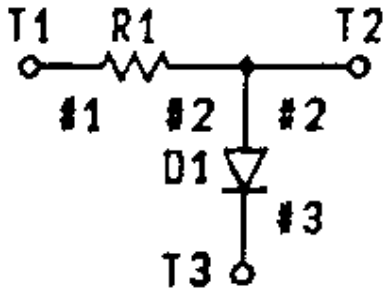
Funktionen sind dabei zum einen die bekannten logischen Funktionen wie „logisches Und“ (`AND`), „logisches Oder“ (`OR`) und „logisches Nicht“ (`NOT`), sowie zwei besonders wichtige Funktionen die die logische Implikation ausdrücken, nämlich ‘`->`‘ und ‘`<-`’. Für ein vorangegangenes Ereignis A mit Konsequenz B gelten die Implikationen:

```
(-> A B)
(<- B A)
```

Logisch gesehen sind die beiden Versionen äquivalent, vom prozeduralen Standpunkt betrachtet gibt es hier jedoch wichtige Unterschiede, die wir später noch kennenlernen werden. Solche Implikationen werden in Duck als Regeln oder Gesetze bezeichnet. Eine Regel der Form `(-> _ _)` wird als Forward-Chaining-Regel (*Forward-Chaining-Rule*) und eine Regel der Form `(<- _ _)` als Backward-Chaining-Regel (*Backward-Chaining-Rule*) bezeichnet.

2.1.2 Beispiel für einen einfachen Schaltkreis

Wie man nun mit diesen Mitteln einen Schaltkreis darstellen kann zeigt am einfachsten ein kleines Beispiel:




```
(-> (CA12) (AND (TERMINAL T1 #1)
                (TERMINAL T2 #2)
                (TERMINAL T3 #3)
                (RESISTOR R1 #1 #2)
                (DIODE D1 #2 #3)))
```

Das Prädikat `TERMINAL` steht für einen Anschlußpunkt des Schaltkreises. `(TERMINAL T1 #1)` bedeutet, daß das Schaltkreiselement mit dem Namen T1 ein `TERMINAL` (Anschlußpunkt) ist, der mit Knoten #1 verbunden ist. Analoges gilt für die beiden anderen definierten Anschlüsse. `(RESISTOR R1 #1 #2)` besagt, daß das

Schaltkreiselement mit dem Namen R1 ein Widerstand (*Resistor*) zwischen den Knoten #1 und #2 ist. In diesem Fall ist die Reihenfolge der Knoten beliebig, da ein Widerstand bekanntlich keine Polarität besitzt. `(DIODE D1 #2 #3)` kennzeichnet das Schaltkreiselement D1 als eine `DIODE` mit Anode #2 und Kathode #3. Hier ist die Knotenreihenfolge sehr wohl von Bedeutung, da eine Diode ein Element mit Polarität ist. Die Tatsache, daß die Knotenreihenfolge bei einigen Elementen von Bedeutung ist, und bei anderen nicht, ist eines der später noch gesondert zu betrachtenden Probleme.

Die einzelnen Prädikate sind in der Regel durch die Funktion `AND` (logisches Und) miteinander verknüpft und stellen die Konsequenz für den Fall dar, daß die Annahme `(CA12)` gemacht wird. `(CA12)` ist prinzipiell ein neu definiertes Prädikat ohne Argumente und kann von daher als Konstante betrachtet werden. Da wir den Schaltkreis nun durch diese Forward-Chaining-Regel Duck mitgeteilt haben, ist das System in der Lage, automatisch die gleichzeitige Gültigkeit der Annahmen „`(TERMINAL T1 #1)`“, ..., „`(DIODE D1 #2 #3)`“ zu folgern (d. h. in seine Datenbank aufzunehmen), sobald wir die Annahme „`(CA12)`“ machen, also dem Duck-System als Eingabe mitteilen.

2.1.3 Weiterführende Überlegungen und Prädikate höherer Ordnung

2.1.3.1 Polare und nicht-polare Elemente

Im vorangegangenen Beispiel wurde bereits die Problematik der Unterscheidung zwischen polaren und nicht-polaren Elementen angedeutet. Dies ist ein ernsthaftes Problem, da Duck beispielsweise `(RESISTOR R1 #1 #2)` beweisen könnte wenn `(CA12)` angenommen wird, jedoch nicht `(RESISTOR R1 #2 #1)`, da ihm diese Umkehrung schließlich nicht bekannt ist. Würde Duck hingegen zu jeder Annahme auch die entsprechende Umkehrung annehmen, so wäre dies im vorliegenden Fall eines Widerstandes zwar eine Lösung, jedoch bei einer Diode eine katastrophale und fehlerhafte Annahme. Folglich muß hier nach einer anderen, allgemeingültigen Lösung gesucht werden.

Sinnvoll wäre das Hinzufügen einer weiteren Regel mit folgender Bedeutung:

```
(FORALL (X A B) (-> (RESISTOR X A B) (RESISTOR X B A)))
```

So etwas läßt sich in Duck einfach angeben:

```
(-> (RESISTOR ?X ?A ?B) (RESISTOR ?X ?B ?A))
```

Das Vorkommen einer durch „?“ markierten Variable innerhalb einer Forward-Chaining-Regel bedeutet, daß diese Variable universell interpretierbar ist[1]. Wenn nun die Annahme `(RESISTOR R1 #1 #2)` gemacht wird (entweder direkt als Eingabe oder als von Duck geschlossene Folgerung), wird automatisch die Annahme `(RESISTOR R1 #2 #1)` gefolgert, also mit in die Datenbank aufgenommen. Fügt man solche Regeln analog für die beiden anderen nicht-polaren Elemente, `CAPACITOR` (Kondensator) und `INDUCTOR` (Spule) hinzu, so stellt die Knotenreihenfolge bei diesen Elementen keine Schwierigkeit mehr da.

[1] Innerhalb ein- und derselben Regel jedoch stets mit gleicher Bedeutung.

2.1.3.2 Prädikate höherer Ordnung

Oftmals ist es vollkommen ausreichend zu wissen, daß in einem Schaltkreis irgendwelche Impedanz-Elemente[2] vorkommen, und nicht um welche Elemente es sich konkret handelt. Aus diesem Grund ist es wünschenswert Elementklassen, also sozusagen Prädikate höherer Ordnung, bilden zu können.

In Duck können wir besagte Impedanz-Elemente ganz einfach unter dem neuen Prädikat `Z-ELM` zusammenfassen:

```
(<- (Z-ELM ?X ?A ?B) (OR (RESISTOR ?X ?A ?B)
                         (CAPACITOR ?X ?A ?B)
                         (INDUCTOR ?X ?A ?B)))
```

Bei dieser Regel wird zum ersten Mal Backward-Chaining anstelle von Forward-Chaining verwendet. Warum das sinnvoll ist werde ich gleich erläutern, zuvor sei jedoch noch darauf hingewiesen, daß man analog zu `Z-ELM` natürlich auch beliebige andere Elementklassen auf ähnliche Weise definieren kann (z. B. 2-polige Elemente, aktive Elemente, passive Elemente, ...).

`Z-ELM` hätte man allerdings auch durch drei Forward-Chaining-Regeln definieren können:

```
(-> (RESISTOR ?X ?A ?B) (Z-ELM ?X ?A ?B))
...
```

Der Nachteil bei Verwendung dieser drei Forward-Chaining-Regeln ist die Tatsache, daß Forward-Chaining-Regeln unmittelbar ausgewertet werden, sobald ihr 'Wenn-Teil' erfüllt ist. Das bedeutet, daß obige Regeln sofort ein entsprechendes `Z-ELM` zu Ducks Datenbank hinzufügen, sobald sie auf einen Widerstand, eine Spule oder einen Kondensator treffen. Dies geschieht bereits mehrfach bei der Eingabe des Schaltkreises. Bei großen Schaltkreisen, die aus sehr vielen solcher Elemente bestehen, kann dies eine dramatische Erhöhung der Annahmen und damit einen hohen Effizienzverlust zur Folge haben.

Bachward-Chaining-Regeln werden hingegen anders ausgewertet, nämlich erst dann, wenn nach ihnen „gefragt" wird. Das kann man sich folgendermaßen vorstellen: Es wird beispielsweise versucht zu beweisen, also abzuleiten, daß die Diode D1 von Knoten #2 nach Knoten #3 ein Impedanz-Element (`Z-ELM`) ist (was natürlich nicht zutrifft). Aufgrund der angegebenen Backward-Chaining-Regel durchsucht Duck seine Datenbank nun nach einem Widerstand, einem Kondensator oder einer Spule mit entsprechendem Namen und Anschlußknoten. Würde eines dieser drei Objekte gefunden, so käme Duck zu dem Ergebnis, daß D1 ein `Z-ELM` ist, und andernfalls nicht. Der Vorteil in der Verwendung von Backward-Chaining-Regeln in solchen Fällen liegt also darin, daß wir Regeln hinzufügen können, ohne die Menge der Annahmen dadurch unnötig anwachsen zu lassen.

2.1.3.3 Elementare Schaltkreis-Eigenschaften

Um auf einfache Weise herauszufinden, mit welchem Ziel man einen gegebenen Schaltkreis analysieren sollte ist es interessant, einige ganz globale Schaltkreis-Eigenschaften ermitteln zu können. Beispielsweise die Frage, ob ein Schaltkreis ein offener Schaltkreis ist, ein Impedanz-Netzwerk oder aber ein Schaltkreis mit genau zwei Anschlüssen.

[2] Der Begriff Impedanz-Element faßt Widerstände, Kondensatoren und Spulen zusammen.

Ob es sich um einen offenen Schaltkreis (also einen Schaltkreis der überhaupt irgendwelche Anschlüsse besitzt) handelt, läßt sich beispielsweise mit folgendem Prädikat einfach ermitteln:

```
(<- (OPEN-CIRCUIT) (TERMINAL ?X ?A))
```

Die `?`-Markierung einer Variable auf der rechten Seite einer Backward-Chaining-Regel wird mit der Existenz eines entsprechenden Prädikats, hier also eines Anschlusses (`TERMINAL`) gleichgesetzt. Wird nun nach der Eigenschaft `OPEN-CIRCUIT` gefragt, so sucht Duck aufgrund dieser Regel in seiner Datenbank nach der Existenz eines `TERMINAL`-Prädikats.

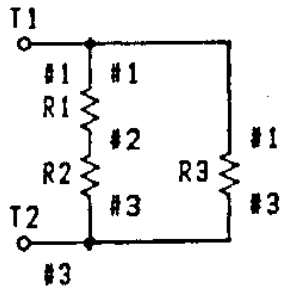
Nachdem nun beschrieben wurde, wie man Schaltkreise logisch darstellen, Klassen von Elementen zusammenfassen und elementare Schaltkreis-Eigenschaften feststellen kann, wende ich mich der eigentlichen Strukturanalyse zu.

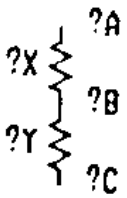
2.2 Auffinden von Strukturen in einem Schaltkreis

Zunächst soll es um die Erkennung von Eigenschaften wie „Gibt es in dem Schaltkreis Widerstände in Reihenschaltung?“ oder „Existieren parallel geschaltete Kondensatoren?“ gehen, später werden wir dann versuchen festzustellen, ob ein gegebener Schaltkreis eventuell ausschließlich aus solchen Parallel- und Reihenschaltungen aufgebaut ist.

2.2.1 Reihen-, Parallel-, Stern- und Dreieckschaltungen

Betrachten wir den folgenden Schaltkreis:





Reihenschaltung von zwei Widerständen

Angenommen, wir wollten die Existenz einer Reihenschaltung von Widerständen in obigem Schaltkreis durch ein Kriterium mit folgender Bedeutung beweisen:

```
(EXISTS (X Y A B C) (AND (RESISTOR X A B)
                         (RESISTOR Y B C)))
```

Wir würden also auf eine Reihenschaltung schließen, wenn wir zwei, an einem Ende miteinander verbundene Widerstände vorfinden würden, also unter anderem auch im Falle einer wirklichen Reihenschaltung wie in obigem Beispiel bei den Widerständen R1 und R2. Jedoch wäre nach diesem Kriterium auch ein einzelner Widerstand eine Reihenschaltung, denn es ist nirgendwo gesagt, daß X und Y verschieden sein müssen, oder besser noch, daß die Knoten A und C verschieden sein müssen. Um diese Fehlinterpretation zu vermeiden könnte man die Bedingung `(NOT (= A C))` hinzufügen. Doch auch mit dieser Erweiterung des Kriteriums würden die Widerstände

R1 und R3 oder R2 und R3 noch als Reihenschaltung interpretiert werden, was natürlich unerwünscht und falsch ist. Wir benötigen ein Kriterium welches sicherstellt, daß an dem mit B bezeichneten Knoten nur genau zwei Elemente angeschlossen sind, nämlich jeweils ein Pol der beiden in Reihe geschalteten Widerstände. Das ist mit den bisher kennengelernten Mitteln leider nicht möglich. Wir benötigen zunächst zwei neue Prädikate: `DEGREE` und `CONNECTED`. `(CONNECTED ?X ?A)` dient zur Feststellung, ob Element `?X` mit dem Knoten `?A` verbunden ist:

```
(<- (CONNECTED ?X ?A) (OR (TERMINAL ?X ?A)
                          (Z-ELM ?X ?A ?B)
                          (DIODE ?X ?A ?B)
                          (DIODE ?X ?B ?A)))
```

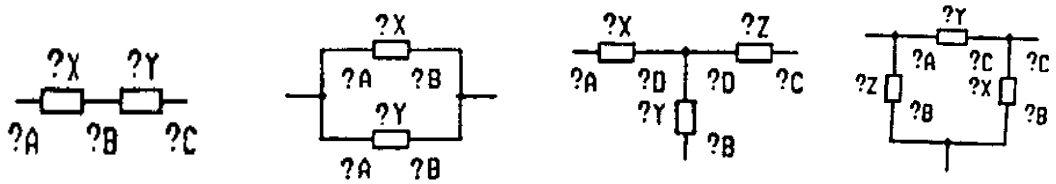
Das Prädikat `DEGREE` ermittelt den Grad[3] eines Knotens. Wir benötigen zu seiner Definition das in Duck fest eingebaute Prädikat `TOTAL`. `TOTAL` überprüft, ob sein erster Argumentterm das Produkt aus seinem zweiten Argumentterm und der Anzahl aller dem System bekannter Instanzen des dritten Argumentterms ist. Damit läßt sich `DEGREE` unter Verwendung von `CONNECTED` definieren:

```
(<- (DEGREE ?X ?DEG) (TOTAL ?DEG 1 (CONNECTED ?X ?A)))
```

Da wir nun eine Möglichkeit zur Überprüfung des Grads eines Knotens kennen, können wir das Prädikat `R-SERIES` definieren, welches gültig ist wenn zwei Widerstände in echter Reihenschaltung vorhanden sind. Um auch gleich die Reihenschaltung von Kondensatoren und Spulen zu berücksichtigen können wir mittels `Z-ELM` allgemein ein Prädikat `Z-SERIES` definieren, welches feststellt, ob Impedanz-Elemente in Reihenschaltung vorliegen:

```
(<- (Z-SERIES ?X ?Y ?A ?B ?C) (AND (Z-ELM ?X ?A ?B)
                                   (Z-ELM ?Y ?B ?C)
                                   (NOT (= ?A ?C))
                                   (DEGREE ?B 2)))
```

Auf ganz ähnliche Art und Weise können wir mit den kennengelernten Methoden natürlich auch das Vorhandensein von Parallel-, Stern- und Dreieckschaltungen testen.



Von links nach rechts: Reihen-, Parallel-, Stern- und Dreieckschaltung

2.2.2 Schaltkreise mit rekursiven Strukturen

Die bisher kennengelernten Verfahren ermöglichen es bereits, eine ganze Reihe von Strukturen und Eigenschaften in den kennengelernten Schaltkreisen aufzufinden. Es gibt jedoch gewisse Strukturen, die wir nicht erkennen bzw. die das Duck-System in eine Endlosschleife laufen lassen, wenn wir es mit den herkömmlichen Methoden versuchen. Gemeint sind rekursive Strukturen, also z. B. die Frage, ob ein gegebener Schaltkreis mit genau zwei Anschlüssen ausschließlich aus parallel oder in Reihe geschalteten Impedanz-Elementen aufgebaut ist.

[3] Der Grad eines Knotens ist die Summe der mit ihm verbundenen Elemente

Bisher können wir lediglich überprüfen, ob parallel oder in Reihe geschaltete Impedanz-Elemente vorkommen, nicht aber ob diese den kompletten Schaltkreis bilden. Das eigentliche Problem liegt darin, daß wir keine Möglichkeit haben, bestimmte Teile eines Schaltkreises als bereits abgearbeitet zu markieren. Wenn wir zwei Widerstände des Schaltkreises bereits als Reihenschaltung identifiziert haben, dann müssen wir dieses Ergebnis irgendwie festhalten, denn sonst werden wir immer wieder die beiden gleichen Widerstände als Reihenschaltung identifizieren und niemals zu anderen Schaltkreiselementen vordringen.

Die Lösung dieses Problems besteht darin, daß wir für Analysezwecke versuchen müssen, bereits erkannte Strukturen durch ein einzelnes Element zu ersetzen. Auf diesem „umgeschriebenen" Schaltkreis könnten wir dann unsere Analyse fortsetzen ohne Gefahr zu laufen, daß wir eine bereits erkannte Struktur nochmals antreffen. Besteht der gesamte Schaltkreis zum Schluß nur noch aus einem einzigen Element, so hat es sich um einen ausschließlich aus Reihen- und Parallelschaltungen von Impedanz-Elementen aufgebauten Schaltkreis gehandelt, andernfalls nicht.

Die Schwierigkeit liegt in der Modifikation des vorhandenen Schaltkreises. Den Schaltkreis umzuschreiben kommt einer Veränderung des Datenbankinhaltes des Duck-Systems gleich. Aus Gründen der Konsistenz darf eine solche Veränderung aber nur durch Regeln erfolgen. Duck stellt uns daher keine Möglichkeiten zur direkten Manipulation des Datenbankinhalts zur Verfügung. Um dies zu erreichen müssen wir uns einer Metasprache bedienen, was konkret bedeutet, daß wir das Duck-System durch externe Lisp-Funktionen steuern und manipulieren müssen.

Wir müssen zunächst aus Lisp heraus Duck anweisen, eine Reihen- oder Parallelschaltung ausfindig zu machen, und uns deren Daten mitzuteilen. Anschließend müssen wir die an der gefundenen Reihen- oder Parallelschaltung beteiligten Elemente aus der Datenbank entfernen und durch ein neues Element mit den entsprechenden Endpunkten ersetzen. Dabei müssen wir aufpassen, daß wir für das neue Element einen Namen verwenden, der noch nicht vorhanden ist. Beim Löschen von Elementen aus der Datenbank müssen wir berücksichtigen, daß wir stets auch die „Umkehrversion" löschen, die Duck bei den nicht-polaren Elementen automatisch generiert hat. Diesen ganzen Vorgang müssen wir solange wiederholen, bis Duck keine Reihen- oder Parallelschaltung mehr finden kann. Dann können wir testen, ob wirklich nur noch ein Element übriggeblieben ist und kommen schließlich zu unserem Ergebnis.

Obwohl das hier vorgestellte Repräsentations- und Analyseverfahren für elektronische Schaltkreise, oder genauer gesagt für Impedanz-Netzwerke, zunächst sehr leistungsstark aussah sind wir bei dem Versuch, rekursive Strukturen zu erkennen, an die Leistungsgrenzen des Systems gelangt. Wir konnten das Problem nicht mehr innerhalb des deduktiven Systems lösen, sondern mußten Duck von außen mittels Lisp programmieren. Außerdem können wir nach dem zuletzt durchgeführten Test keine weiteren Tests auf dem Original-Schaltkreis mehr durchführen, da dieser durch das ständige Umschreiben vollständig zerstört wurde. Diese Nachteile waren der Grund für die Suche nach anderen Analyseverfahren.

3 Ein neues Verfahren zur Schaltkreisanalyse

Aufgrund der Schwachpunkte der vorangegangenen Repräsentations- und Analysemethode ist ein neues Verfahren zur Schaltkreisanalyse wünschenswert, welches die genannten Probleme nicht aufkommen läßt, jedoch deswegen nicht komplizierter und unübersichtlicher wird. Um einen solchen neuen Ansatz geht es im folgenden.

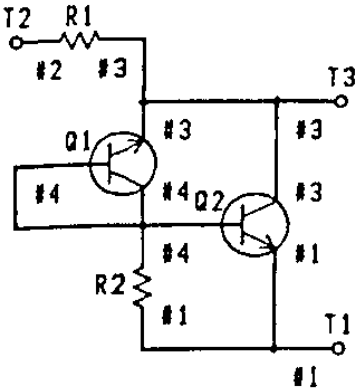
3.1 Die neue Repräsentationsmethode für Schaltkreise

In der von nun an verwendeten Repräsentationsmethode werden zur Darstellung von Schaltkreis-Elementen keine Prädikate mehr verwendet, sondern Funktionen. Die Funktion

```
(NPN-TR Q2 #4 #1 #3)
```

steht beispielsweise für einen NPN-Transistor dessen Name und seine drei Anschlüsse Basis, Emitter und Kollektor durch seine Argumente bestimmt sind. Eine Funktion der oben angegebenen Form wird dabei jedoch niemals logisch ausgewertet, sondern steht lediglich für ein ganz bestimmtes Element in einem elektronischen Schaltkreis.

Ganze Schaltkreise mit n Elementen werden in dieser neuen Repräsentationsmethode nun als ungeordnete n-Tupel von Schaltkreis-Elementen dargestellt. Ein solches Tupel wird von den Symbolen ‘!<‘ und ‘>‘ eingeschlossen. Als kleines Beispiel ein Schaltkreis und das ihn repräsentierende Tupel:




```
!< (TERMINAL T1 #1) (TERMINAL T2 #2)
   (TERMINAL T3 #3) (RESISTOR R1 #2 #3)
   (RESISTOR R2 #4 #1) (NPN-TR Q1 #4 #3 #4)
   (NPN-TR Q2 #4 #1 #3) >
```

3.1.1 Member-Prädikate

In der vorangegangenen Repräsentationsmethode war der Schaltkreis in Form von Annahmen in der Duck-Datenbank enthalten. Die Suche nach einem bestimmten Element kam damit prinzipiell dem Beweis einer - wenn auch sehr trivialen - Behauptung gleich. Wir mußten keinerlei zusätzliche Energie in diesen Bereich investieren. Das ist bei diesem neuen Ansatz leider nicht mehr so. Um festzustellen, ob ein Element im Schaltkreis vorhanden ist, müssen wir das den Schaltkreis beschreibende n-Tupel nach dem Element durchsuchen. Dazu wird ein Prädikat `MEMBER` durch Hinzufügen der folgenden Annahme zur Duck-Datenbank definiert:

```
(MEMBER ?M !< !&?L ?M !&?R >)
```

`?L`, `?M` und `?R` sind universell interpretierbare Variablen wie wir sie bereits kennen. Der Zusatz ‘`!&`’ bezeichnet eine beliebige Anzahl von Elementen eines Tupels und bindet alle diese Elemente an die entsprechende Variable. Das Element ‘`?M`’ teilt das Tupel also in einen linken Teil ‘`!&?L`’ und einen rechten Teil ‘`!&?R`’ falls es vorhanden ist.

Zusätzlich zu diesem Prädikat definieren wir analog noch ein weiteres Prädikat MEM-REST, welches es uns gestattet, den Rest des Tupels - also alles mit Ausnahme des jeweiligen Elements - direkt anzusprechen:

```
(MEM-REST ?M !< !&?L ?M !&?R > !< !&?L !&?R >)
```

Durch dieses Prädikat haben wir die Möglichkeit, auf einfache Weise einen Schaltkreis in ein bestimmtes Element und den gesamten restlichen Schaltkreis aufzuteilen, wobei wir auf beide Teile direkt zugreifen können. Diese Möglichkeit hatten wir im vorangegangenen Ansatz nicht.

3.1.2 Das Sub-Circuit-Konzept

Die Möglichkeit, auf ein Element des Schaltkreises und den gesamten Rest zugreifen zu können, ist schon sehr nützlich für unsere Zwecke, doch durch das Sub-Circuit-Konzept wird die Leistungsfähigkeit in diesem Bereich noch weiter gesteigert: Wir können so den Schaltkreis in einen beliebigen benötigten Teil und den entsprechenden Rest aufteilen. Das Sub-Circuit-Prädikat SUB-CT hat folgende Form:

```
(SUB-CT abstract-element object-circuit rest)
```

Dabei ist mit ‘abstract-element’ der uns interessierende Teil des Schaltkreises gemeint, den wir aus dem Gesamttupel extrahieren wollen. Ein kleines Beispiel scheint an dieser Stelle angebracht zu sein:

```
(<- (SUB-CT (RES ?R ?A ?B) ?CT ?REST)
    (OR (MEM-REST (RESISTOR ?R ?A ?B) ?CT ?REST)
        (MEM-REST (RESISTOR ?R ?B ?A) ?CT ?REST)))
```

RES ist eine sog. Skolem-Funktion, die die Existenz des abstrakten Elements repräsentiert. In diesem Fall handelt es sich bei dem abstrakten Element um den Widerstand ?R zwischen den Knoten ?A und ?B, falls er im Schaltkreis vorhanden ist. Durch Verwendung der Skolem-Funktion RES anstelle von RESISTOR spielt es keine Rolle mehr, in welcher „Richtung“ wir den Widerstand verwenden. Mit (RES R1 #1 #2) ist sowohl (RESISTOR R1 #1 #2), als auch (RESISTOR R1 #2 #1) gemeint. Analog zur Skolem-Funktion RES werden Skolem-Funktionen CAP und IND für Kondensatoren und Spulen definiert.

Ebenso wie im vorangegangenen Ansatz wird auch hier eine Funktion Z-ELEMENT definiert, die für ein beliebiges Impedanz-Element steht. Z-ELM ist hingegen wieder eine abstrakte Skolem-Funktion die die abstrakten Elemente RES, CAP und IND oder ein Z-ELEMENT zusammenfaßt.

Ebenfalls mit Hilfe von SUB-CT wird das abstrakte Element TRANSISTOR definiert, welches die Bezugnahme auf ein Element vom Typ NPN-TR oder PNP-TR über die zusammenfassende Bezeichnung TRANSISTOR ermöglicht. ANY-ELM schließlich ist ein abstraktes Element, welches für ein beliebiges Schaltkreiselement steht.

Das hier vorgestellte Sub-Circuit-Konzept ist der entscheidende Punkt, der die enorme Leistungsfähigkeit dieser Repräsentationsmethode für Schaltkreise ausmacht. Wir haben dieselben einfachen und leistungsfähigen Zugriffsmethoden auf einzelne Elemente oder Gruppen von Elementen des Schaltkreises wie im vorangegangenen Ansatz, nur mit dem entscheidenden Vorteil, daß wir gleichzeitig eine Partitionierung vornehmen und auf beide Teile des Schaltkreises bei Bedarf zugreifen können.

3.1.3 Negation von Prädikaten

Für die Prädikate `MEMBER` und `SUB-CT` ist es im folgenden sinnvoll, deren Negation zur Verfügung zu haben. Daher werden `NON-MEMBER` und `NON-SUBCT` nun mit Hilfe des Prädikats `THNOT` definiert. Für `(THNOT p)` gilt:

`(THNOT p)` ist genau dann wahr, wenn `p` nicht bewiesen werden kann.

Es ergibt sich damit für `NON-MEMBER` und `NON-SUBCT`:

```
  (<- (NON-MEMBER ?X ?TUP) (THNOT (MEMBER ?X ?TUP)))
(<- (NON-SUBCT ?X ?CT) (THNOT (SUB-CT ?X ?CT ?REST)))
```

Mit dieser neuen Repräsentationsmethode und vor allen Dingen dem Sub-Circuit-Konzept können wir nun alle Schaltkreisanalysen vornehmen, die wir mit dem vorangegangenen Ansatz gemacht haben, und darüber hinaus noch einiges mehr, wie wir im folgenden sehen werden.

3.2 Zwei Analyse-Verfahren für Schaltkreise

Es folgen nun zwei Analyse-Verfahren für Schaltkreise, entsprechend der Beschreibung im vorangegangenen Abschnitt. Die erste Methode, genannt Bottom-Up-Methode, führt die Analysen auf Impedanz-Netzwerken durch, die wir bereits kennengelernt haben. Dabei werden wir jedoch nicht auf die zuvor aufgetretenen Schwierigkeiten stoßen. Die zweite Methode, genannt Top-Down-Methode, ist bestens geeignet für elektronische Schaltungen die aus einzelnen Funktionsblöcken aufgebaut sind. Sie wäre in dieser Form mit dem „alten" Verfahren überhaupt nicht möglich gewesen. Beide Methoden lassen Parallelen der Schaltkreisanalyse zu Sprachen erkennen, auf die ich später noch genauer eingehen werde.

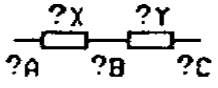
3.2.1 Die Bottom-Up-Methode für Impedanz-Netzwerke

Die Analyse-Methode, die im folgenden vorgestellt werden soll, eignet sich für Impedanz-Netzwerke, also Schaltkreise, die aus Widerständen, Kondensatoren und Spulen aufgebaut sind. Die Strukturen nach denen wir suchen werden sind daher die gleichen, wie im bereits vorgestellten Repräsentations- und Analyseansatz. Wir werden jedoch sehen, daß wir alle gestellten Anforderungen vollständig in Duck lösen können, Lisp nicht länger als externes Hilfsmittel benötigen, und von daher konsistentere Ergebnisse erhalten.

3.2.1.1 Sub-Circuits für Reihen-, Parallel-, Stern- und Dreieckschaltungen

Wie wir bereits gesehen haben, kommt es bei Impedanz-Netzwerken zunächst einmal darauf an, die wesentlichen Strukturen Reihenschaltung, Parallelschaltung, Sternschaltung und Dreieckschaltung von Impedanz-Elementen zu erkennen[4]. Im vorangegangenen Ansatz benötigten wir zur Erkennung solcher Strukturen unter anderem die komplizierten Hilfsprädikate `TOTAL`, `CONNECTED` und `DEGREE`.

Dank des Sub-Circuit-Konzepts läßt sich das jetzt eleganter lösen:




```
(<- (SUB-CT (Z-SERIES ?X ?Y ?A ?B ?C) ?CT ?REST)
    (AND (SUB-CT (Z-ELM ?X ?A ?B) ?CT ?RST1)
         (SUB-CT (Z-ELM ?Y ?B ?C) ?RST1 ?REST)
         (NON-SUBCT (ANY-ELM ?Z ?B) ?REST)))
```

[4] vgl. hierzu auch Abschnitt 2.2.1

Ein Sub-Circuit Z-SERIES des Schaltkreises ?CT, bestehend aus (?CT / ?REST)[5], existiert dann, wenn es ein Impedanz-Element ?X von Knoten ?A zu Knoten ?B gibt und ein weiteres Impedanz-Element ?Y von Knoten ?B zu Knoten ?C, welches allerdings nicht mit dem zuvor genannten identisch sein darf[6]. Weiterhin darf kein weiteres Element mit Knoten ?B verbunden sein[7].

Diese neue Variante läßt sich wesentlich besser lesen und bietet zudem noch den Vorteil, daß wir eine evtl. vorhandene Reihenschaltung von Impedanz-Elementen nicht nur erkennen und wissen, wo sie sich im Schaltkreis befindet, sondern auch noch direkten Zugriff auf den gesamten restlichen Schaltkreis haben. Dies ist, wie wir später noch sehen werden, eine sehr wichtige Voraussetzung zur Weiterverarbeitung der gewonnenen Information.

Auf ähnlich einfache Weise können wir Sub-Circuits Z-PARLLEL für Parallelschaltungen, Z-STAR für Sternschaltungen und Z-DELTA für Dreieckschaltungen von Impedanz-Elementen definieren.

3.2.1.2 Umschreiben von Schaltkreisen

Was bei der letzten Methode zu einem in Duck nicht mehr lösbaren Problem geführt hat läßt sich mit dem neuen Konzept jetzt sehr leicht realisieren. Wir wollen aus einem Schaltkreis, der eine Reihenschaltung von Impedanz-Elementen aufweist, einen äquivalenten Schaltkreis konstruieren, in dem die beiden in Reihe geschalteten Elemente durch ein einziges Element ersetzt sind. Zusätzlich fordern wir, daß der ursprüngliche Schaltkreis durch diese Transformation nicht zerstört wird und der transformierte Schaltkreis erkennen läßt, ob und an welcher Stelle eine Transformation vorgenommen wurde. Eine zweizeilige Regel ist in der Lage, diese scheinbar komplexe Aufgabe in Duck zu bewältigen:

```
(<- (SR-EQIVCT ?CT !< (Z-ELEMENT (SR ?X ?Y) ?A ?C) !&?REST >)
    (SUB-CT (Z-SERIES ?X ?Y ?A ?B ?C) ?CT ?REST))
```

`SR-EQIVCT`[8] kann also dann bewiesen werden, wenn im Schaltkreis `?CT` ein Sub-Circuit `Z-SERIES` existiert. Ist das der Fall, dann wird im zweiten Argument von `SR-EQIVCT` der transformierte Schaltkreis konstruiert. Dieser ist dargestellt durch das übliche Tupel, bestehend aus dem neu eingefügten Element und dem gesamten Rest (`?REST`) des Schaltkreises[9], also alles außer der `Z-SERIES`. Das neu eingefügte Element ist ein Impedanz-Element `(Z-ELEMENT)`, erhält den konstruierten Namen '`(SR ?X ?Y)`' und wird mit den Knoten `?A` und `?C` verbunden. In der Variable `?CT` befindet sich auch nach der Transformation noch der ursprüngliche Schaltkreis, und der Name des neuen Elements '`(SR ?X ?Y)`' gibt uns Auskunft darüber, daß es ein von `SR-EQIVCT` aus `?X` und `?Y` generiertes Element ist. Wir haben also die Transformation durchführen, und dabei die volle Transparenz erhalten können.

---

[5] Die Notation `(?CT / ?REST)` soll das Tupel bezeichnen, welches durch Entfernen aller Elemente des Tupels `?REST` aus dem Tupel `?CT` entsteht.

[6] Daß zwei unterschiedliche Elemente verwendet werden ist dadurch sichergestellt, daß zunächst Element `?X` im Tupel `?CT` gesucht wird, und erst anschließend das Element `?Y` unter den verbleibenden Elementen `(?RST1)`.

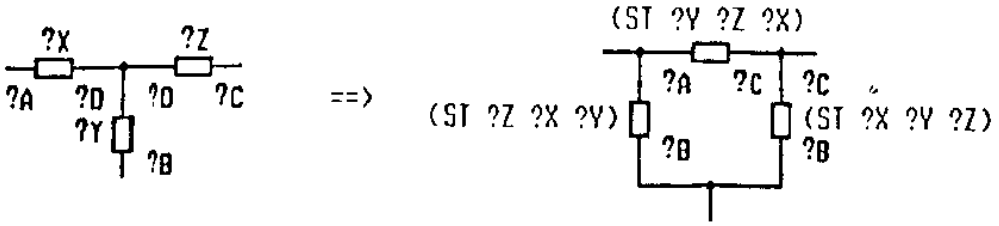
[7] Dies wird durch das `NON-SUBCT`-Prädikat sichergestellt. „Es gibt unter den verbleibenden Tupel-Elementen keinen Sub-Circuit, bestehend aus einem beliebigen Element (`ANY-ELM`), der Verbindung zum Knoten `?B` hat.

[8] steht für *series equivalent circuit*

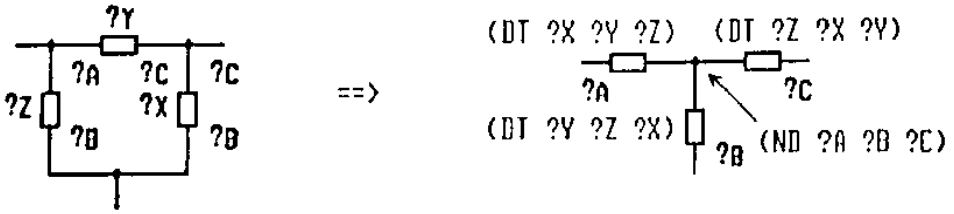
[9] der Zusatz '!&' vor der Variablen ?REST ist notwendig um Duck anzuzeigen, daß wir nicht genau wissen, wieviele Elemente an dieser Stelle in das Tupel aufgenommen werden sollen. Der Rest kann beliebig groß sein und wird von Fall zu Fall eine unterschiedliche Größe haben.

Bei parallel geschalteten Impedanz-Elementen kann eine analoge Transformation durchgeführt werden, nur daß in diesem Fall natürlich nach einem Sub-Circuit `Z-PARALLEL` gesucht werden muß. Die entsprechende Funktion heißt `PR-EQIVCT`[10] und das von ihr neu generierte Element '`(PR ?X ?Y)`'.

Stern- und Dreieckschaltungen können nicht durch ein einzelnes Element ersetzt werden, aber man kann eine Sternschaltung in eine Dreieckschaltung und umgekehrt transformieren. Die beiden folgenden Abbildungen verdeutlichen diesen Vorgang:



Transformation einer Stern- in eine Dreieckschaltung



Transformation einer Dreieck- in eine Sternschaltung

Solche Transformationen können wir natürlich ebenfalls analog zur kennengelernten Methode durchführen, nur daß wir jetzt drei Elemente in den neu zu konstruierenden Schaltkreis mit aufnehmen müssen. Das stellt jedoch keine weitere Schwierigkeit dar. Aus Gründen der Vollständigkeit möchte ich noch kurz die Namen der entsprechenden Funktionen bzw. die vergebenen Elementnamen anführen: Die Stern-Dreieck-Transformation hat den Namen ST-EQIVCT[11] und bezeichnet die von ihr eingefügten Elemente mit 'ST', gefolgt von den jeweiligen Knotennamen. Die Dreieck-Stern-Transformation hat den Namen DT-EQIVCT[12] und bezeichnet die von ihr eingefügten Elemente mit 'DT', wiederum gefolgt von den Knotennamen. Da beim Übergang von der Dreieck- zur Sternschaltung ein zusätzlicher Knoten benötigt wird (s. Abbildung) wird hierfür zusätzlich noch der dies indizierende Knotenname '(ND ?A ?B ?C)' eingeführt.

3.2.1.3 Bottom-Up-Analyse

Bottom-Up-Analyse bezeichnet das Analyseverfahren, bei dem „von unten nach oben" vorgegangen wird, in unserem Fall also von den einzelnen, zum Schaltkreis miteinander verbundenen Elementen ausgehend. Wir analysieren die Elemente des Schaltkreises, finden eine gewisse Struktur und bauen einen neuen, äquivalenten Schaltkreis. Mit diesem verfahren wir ebenso, bis wir irgendwann entweder keine bekannten Strukturen mehr finden oder das gewünschte Ziel erreicht wurde. Um das machen zu können müssen wir also lediglich die bereits kennengelernten Transformationen auf geeignete Weise iterieren. Dazu wird zunächst eine

[10] steht für *parallel equivalent circuit*
[11] steht für *star equivalent circuit*
[12] steht für *delta equvalent circuit*

Transformation SP-EQIVCT[13] definiert welche lediglich SR-EQIVCT und PR-EQIVCT kombiniert. Diese Transformation führt entweder die Reihen- oder die Paralleltransformation aus[14]:

```
(<- (SP-EQIVCT ?CT ?EQCT) (OR (SR-EQIVCT ?CT ?EQCT)
                              (PR-EQIVCT ?CT ?EQCT)))
```

Dieser Transformationsschritt muß jetzt iteriert werden. Das erreichen wir mit folgendem rekursiven Verfahren:

- Wenn im Schaltkreis eine Transformation durchgeführt werden kann, dann transformiere ihn und wende das Verfahren rekursiv auf den transformierten Schaltkreis an.
- Wenn im Schaltkreis keine Transformation durchgeführt werden kann, dann lasse den Schaltkreis unverändert.

Das folgende Prädikat M-SP-EQIVCT macht genau das oben beschriebene:

```
(<- (M-SP-EQIVCT ?CT ?EQCT)
    (OR (AND (SP-EQIVCT ?CT ?EQ1)
             (M-SP-EQIVCT ?EQ1 ?EQCT))
        (AND (THNOT (SP-EQIVCT ?CT ?EQ1))
             (= ?EQCT ?CT))))
```

Mit diesem Prädikat kann jetzt ganz einfach getestet werden, ob ein gegebener Schaltkreis ein 2-poliges Impedanz-Netzwerk ist, welches ausschließlich aus Parallel- und Reihenschaltungen besteht:

```
(<- (SERIES-PARALLEL-CIRCUIT ?CT)
    (M-SP-EQIVCT ?CT !< (Z-ELEMENT ?X ?A ?B)
                        (TERMINAL ?Y ?C)
                        (TERMINAL ?Z ?D) >))
```

Der Schaltkreis wird solange transformiert, bis er nur noch aus einem einzigen Impedanz-Element und zwei Anschlußpolen besteht. Kann dieses Ziel erreicht werden, so handelt es sich um einen Schaltkreis mit der besagten Eigenschaft, andernfalls nicht.

In Anhang A befindet sich ein Beispiel für eine solche Folge von Schaltkreistransformationen.

Bei Einführung der Schaltkreistransformationen wurde gefordert, daß es jederzeit nachvollziehbar bleiben sollte, welche Transformationen in welcher Reihenfolge ausgeführt wurden. Betrachten wir die Namens-Struktur des generierten Z-ELEMENTs bei der Beispieltransformation aus Anhang A:

```
(SR (PR (SR (PR (SR R5 R6) R4) R3) R2) R1)
```

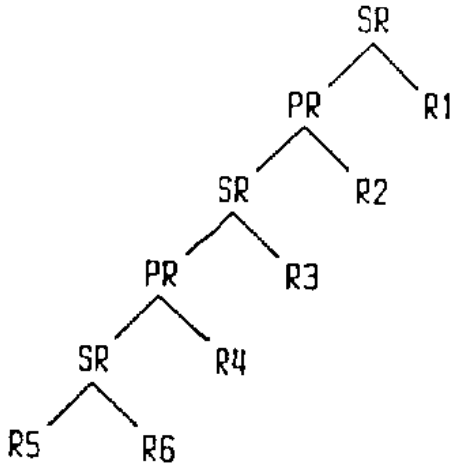
Diese ‘Namensfolge’ können wir durch einen binären Baum darstellen, dessen Blätter die realen Elemente, in unserem Fall also die Widerstände, darstellen. Die inneren Knoten lassen die Transformationsregel erkennen, die angewendet wurde, um die beiden Kindknoten zu einem einzigen Knoten zu verschmelzen:

---

[13] steht für *series-parallel equivalent circuit*

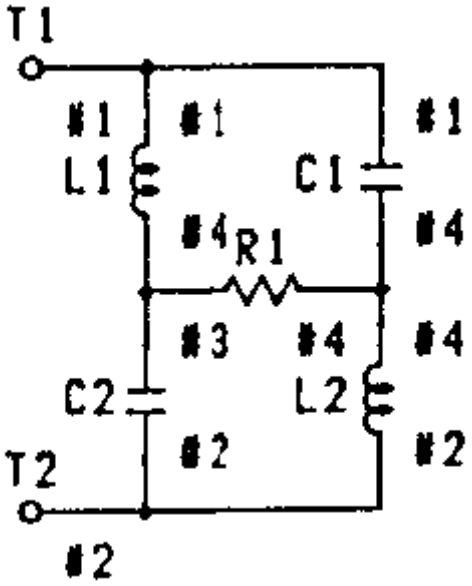
[14] SP-EQIVCT ist also möglich, wenn eine Transformation vom Typ SR-EQIVCT oder wenn eine Transformation vom Typ PR-EQIVCT durchgeführt werden kann. Die beiden Transformationen werden also mittels ‘logischem Oder’ verbunden. In diesem Fall ist es gleichgültig, ob zuerst die Reihen- oder zuerst die Paralleltransformation durchgeführt wird, nicht jedoch in später folgenden Situationen. Aus diesem Grund sei darauf hingwiesen, daß Duck die Prädikate stets in der Reihenfolge ihres Auftretens bearbeitet.

Dieser Baum veranschaulicht auch recht gut den Begriff der Bottom-Up-Analyse: die Analyse erfolgt von den Blättern ausgehend in Richtung der Wurzel. Hierin besteht eine gewisse Ähnlichkeit zum Parsen von Sätzen einer Sprache nach dem Bottom-Up-Verfahren. Hierbei wird versucht, die Wörter (Terminalsymbole; sie entsprechen hier den Schaltkreiselementen) durch Non-Terminals aus denen diese Wörter hergeleitet werden können, zu ersetzen. Die Wurzel des Parse-Baumes entspricht dann dem Startsymbol der, die Sprache beschreibenden Grammatik. Wir haben also im Prinzip ein 'Startsymbol' der Form 'Impedanz-Netzwerk mit bestimmten Eigenschaften' aus dem der vorhandene Schaltkreis hergeleitet werden kann, sofern er die geforderten Eigenschaften aufweist. Die Bottom-Up-Methode versucht, von den Elementen (Worten) ausgehend dieses Startsymbol zu erreichen.

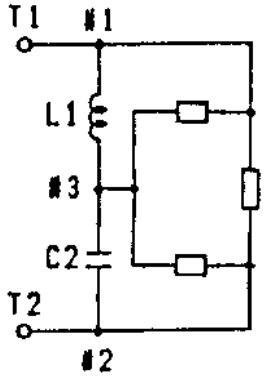


3.2.1.4 Bottom-Up-Analyse von Schaltkreisen aus Stern- und Dreieckschaltungen

Es gibt Netzwerke, die sich in ein einziges Impedanz-Element und zwei Anschlußpole transformieren lassen, aber nicht mit der vorgestellten Methode. Es handelt sich hierei um Netzwerke, die ausschließlich Stern- und Dreieckschaltungen enthalten. Die folgende Abbildung zeigt ein solches Netzwerk:

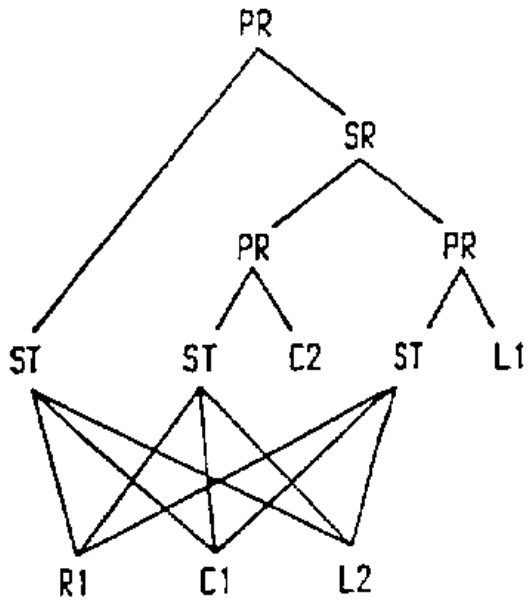


Hier kann weder eine Reihen- noch eine Paralleltransformation durchgeführt werden, da eben keine Reihen- oder Parallelschaltungen vorkommen. Wir wissen aber, daß wir Dreieck- und Sternschaltungen gegenseitig ineinander überführen können. Wenn wir in obiger Schaltung den 'Stern', bestehend aus C1, R1 und L2 in eine Dreieckschaltung transformieren erhalten wir folgendes Resultat:



Diesen Schaltkreis können wir nun wie gewohnt mit Reihen- und Paralleltransformationen in ein einzelnes Impedanzelement überführen. Der folgende 'Parse-Baum' zeigt die Vorgehensweise:

Dieser Parse-Baum läßt deutlich erkennen, daß die Stern-Dreieck-Transformation die Elementzahl nicht verringert, sondern die Elemente lediglich anders anordnet. Die obere Hälfte des Baumes stellt jedoch wieder einen schönen binären Baum dar, weil dort nur noch Parallel- und Reihentransformationen stattfinden.



Schaltkreise, die sich erst nach der Anwendung solcher 'Tricks' transformieren lassen können ebenfalls automatisch analysiert werden, wenn man `SP-EQIVCT` um die Stern-Dreieck- und die Dreieck-Stern-Transformationen erweitert (diese werden einfach mit in die Disjunktion aufgenommen). Wichtig ist dabei, daß die beiden neuen xTransformationen eine niedrigere Priorität als die Reihen- und Paralleltransformation erhalten, da es sonst zu Endlosschleifen kommen kann[15].

Mit einem auf diese Weise erweiterten Prädikat können die meisten in der Praxis eines Elektrotechnikers vorkommenden 2-poligen Impedanz-Netzwerke automatisch von Duck analysiert werden.

3.2.2 Die Top-Down-Methode für elektronische Schaltkreise

Im folgenden geht es nicht mehr um Impedanz-Netzwerke, sondern elektronische Schaltkreise, also solche die neben Impedanz-Elementen auch Halbleiterelemente wie Dioden und Transistoren enthalten. Bei der Analyse dieser Schaltkreise wird nicht mehr Bottom-Up, also von den Elementen ausgehend vorgegangen, sondern nach dem Top-Down-Verfahren. Der Grund für diese andere Strategie liegt im Designprozeß von elektronischen Schaltkreisen. Ein elektronischer Schaltkreis ist in der Regel ein aus vielen Basisschaltkreisen aufgebautes Gebilde. Es werden fertige, sich nicht verändernde Module miteinander verknüpft und in eine hierarchische Ordnung gebracht, um das gewünschte Ziel zu erreichen.

Diese Tatsache wird bei der Analyse genutzt, es wird versucht, den Schaltkreis durch Zusammensetzen solcher Module von nichts ausgehend zu erzeugen. Diese Vorgehensweise entspricht dem Ableiten von Sätzen oder Worten aus dem Startsymbol der Grammatik einer Sprache. Überträgt man dies auf Schaltkreise, so stellen die einzelnen Basismodule die Non-Terminals und die diskreten Elemente im Schaltkreis die Terminal-Symbole der Grammatik dar. Der zur entsprechenden Ableitung gehörende Syntaxbaum liefert uns dann sozusagen eine grafische Darstellung der Beziehungen zwischen den Funktionsblöcken eines Schaltkreises.

[15] Wenn die Umwandlung einer Stern- in eine Dreieckschaltung, oder umgekehrt, die Möglichkeit für die Anwendung einer den Schaltkreis verkleinernden Reihen- oder Paralleltransformation gibt, dann muß diese Reihen- bzw. Paralleltransformation zuerst ausgeführt werden, denn sonst wird die Stern- oder Dreieckschaltung wieder zurückverwandelt, ohne daß sich etwas geändert hat, und das System läuft in eine Endlosschleife. (Das passiert übrigens bei einer Schaltung, die aus einem einzigen 'Dreieck' oder einem einzigen 'Stern' besteht, zwangsläufig.)

3.2.2.1 Definition von Funktionsblöcken

In diesem Abschnitt wird dargestellt, wie man mit Hilfe des Sub-Circuit-Konzepts Module, also in Schaltkreisen immer wieder auftretende Funktionsblöcke beschreiben kann. Es braucht wohl nicht extra erwähnt werden, daß die Leistungsfähigkeit des Systems im Bereich der Analyse mit der Anzahl der ihm bekannten Module wächst. Selbstverständlich können komplexere Module wiederum aus elementareren Modulen zusammengesetzt sein, und somit lassen sich dem System quasi beliebig komplexe Funktionsblöcke 'beibringen'.

Wie man nun Funktionsblöcke in Duck definiert zeigt das folgende Beispiel. In der Elektronik werden NPN-Transistoren häufig als einfache Dioden 'mißbraucht'. Diesen Effekt erzielt man durch einfaches Kurzschließen von Basis und Kollektor des Transistors. Basis und Kollektor entsprechen dann der Anode, während der Emitter die Kathode darstellt. Betrachten wir nun die folgende Regel hierfür in Duck:

```
(<- (SUB-CT (D-TRANSISTOR (FD ?Q) ?A ?C) ?CT ?REST)
    (MEM-REST (NPN-TR ?Q ?A ?C ?A) ?CT ?REST))
```

Die Regel besagt: wenn es im Schaltkreis `?CT` einen NPN-Transistor mit Namen `?Q` gibt, dessen Basis und Kollektor mit demselben Knoten verbunden sind, dann gibt es in diesem Schaltkreis einen Sub-Circuit '`D-TRANSISTOR`', mit dem Namen '`(FD ?Q)`'. Mit dieser Regel kann man bereits in komplexen Schaltkreisen die Existenz eines solchen als Diode verwendeten Transistors feststellen. Da solche Dioden in der Praxis auch in umgekehrter Richtung, also gerade mit vertauschter Anode und Kathode verwendet werden, kann man für diesen Fall noch ein weiteres `D-TRANSISTOR`-Prädikat definieren, welches sich durch die unterschiedliche Polung und sinnvoller Weise im dafür vergebenen Namen von obigem Prädikat unterscheidet. Beispielsweise könnte man '`RD`' anstelle von '`FD`' verwenden[16].

Das folgende Beispiel zeigt zum einen, wie man bereits definierte Funktionsblöcke (hier `D-TRANSISTOR`) zur Definition neuer Module benutzen kann, und zum anderen, wie man auch Module definieren kann, deren Elementanzahl unterschiedlich sein kann. `S-DIODE` steht für beliebig viele, in Reihe geschaltete als Diode eingesetzte Transistoren (auch ein einzelner entsprechend beschalteter Transistor wird erkannt):

```
(<- (SUB-CT (S-DIODE ?NAME ?A ?C) ?CT ?REST)
    (OR (SUB-CT (D-TRANSISTOR ?NAME ?A ?C) ?CT ?REST)
        (AND (SUB-CT (D-TRANSISTOR ?Q ?A ?B) ?CT ?RST1)
             (SUB-CT (S-DIODE ?D ?B ?C) ?RST1 ?REST)
             (= ?NAME (S-DD ?Q ?D))
             (NON-SUBCT (ANY-ELM ?Z ?B) ?REST))))
```

Es wird ein `D-TRANSISTOR ?Q` zwischen Knoten `?A` und `?B` im Schaltkreis `?CT` gesucht. Anschließend wird im restlichen Schaltkreis durch rekursiven Aufruf von `S-DIODE` nach einem `D-TRANSISTOR` von Knoten `?B` zu einem Knoten `?C` gesucht, usw. Dabei wird stets sichergestellt, daß sich an diesem Knoten `?B` wirklich nur die beiden an der Reihenschaltung beteiligten Transistoren berühren. Auf diese Weise findet man mit `S-DIODE` in einem Schaltkreis einen oder mehrere in Reihe geschaltete 'Transistor-Dioden'.

[16] RD = *reverse bias diode*, FD = *forward bias diode*

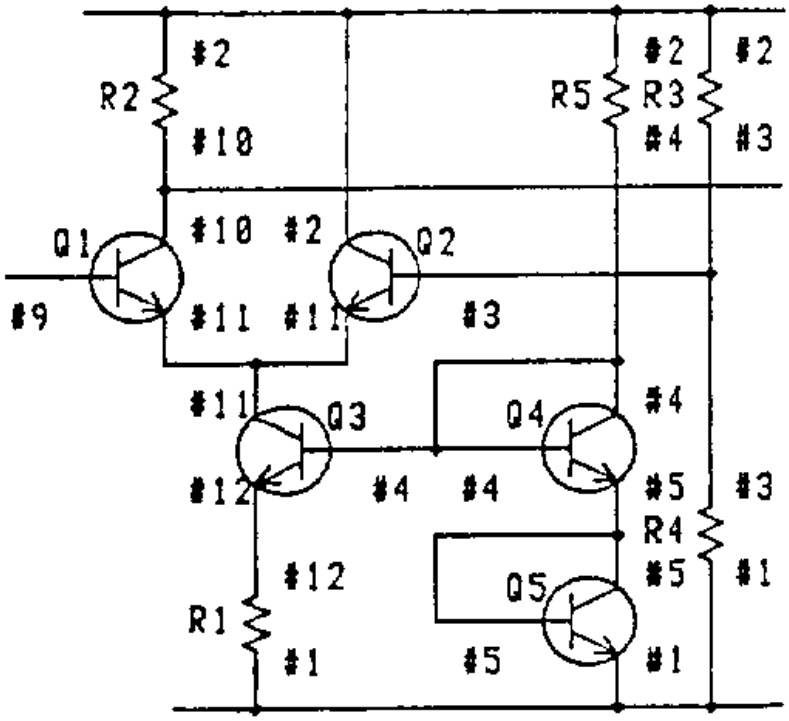
Auf dieselbe Weise, wie oben D-TRANSISTOR und S-DIODE beschrieben wurde kann man nun weitere, wesentlich kompliziertere Funktionsblöcke beschreiben. Bei der Definition eines solchen Moduls können prinzipiell alle dem System bekannten Module benutzt werden. Beispiele für elektronische Schaltungen, die auf diese Weise beschrieben werden können sind: Spannungsregulierer, Spannungsteiler, Spannungsquelle, Stromquelle, Emitter-gekoppelte Transistoren, Differenzverstärker, Current-Comparator[17], usw. Letzteren möchte ich hier als Beispiel einmal anführen:

```
(<- (SUB-CT (I-COMPARATOR (I-COMP ?AMP ?VS) ?B1 ?C1 ?P ?G) ?CT ?REST)
    (AND (SUB-CT (S-DIF-AMP ?AMP ?B1 ?B2 ?G ?C1 ?P) ?CT ?RST1)
         (SUB-CT (V-SOURCE ?VS ?B2 ?G) ?RST1 ?REST)))
```

Ein solches Objekt vom Typ I-COMPARATOR wird hier also als Kombination zweier anderer Module: S-DIF-AMP und V-SOURCE, einem Differenzverstärker und einer Spannungsquelle in entsprechender Beschaltung beschrieben. Die Regeln für den Differenzverstärker und die Spannungsquelle müssen dem System natürlich zuvor mitgeteilt worden sein.

3.2.2.2 Beispiel für eine Top-Down-Analyse

Betrachten wir den folgenden Schaltkreis:



Der Schaltkreis wird durch folgendes Tupel beschrieben:

```
kreis = !< (RESISTOR R1 #12 #1) (RESISTOR R2 #2 #10)
           (RESISTOR R3 #2 #3) (RESISTOR R4 #3 #1)
           (RESISTOR R5 #2 #4) (NPN-TR Q1 #9 #11 #10)
           (NPN-TR Q2 #3 #11 #2) (NPN-TR Q3 #4 #12 #11)
           (NPN-TR Q4 #4 #5 #4) (NPN-TR Q5 #5 #1 #5) >
```

[17] Ich verwende hier das englische Wort Current-Comparator, da mir keine passende deutsche Übersetzung eingefallen ist. 'Strom-Vergleicher' erschien mir nicht sehr geeignet.

Nehmen wir an, wir hätten nun die Vermutung, daß es sich bei diesem Schaltkreis um einen Current-Comparator, also einen Sub-Circuit vom Typ `I-COMPARATOR` handelt. Wir müssen Duck anweisen, dies zu verifizieren:

```
(SUB-CT (I-COMPARATOR ?NAME ?IN ?OUT ?POW ?GND) kreis !< >)
```

Entscheidend ist dabei das leere Tupel als Parameter für den 'Rest' des Schaltkreises, was bedeutet, daß im Prinzip keine Teile oder Elemente des Schaltkreises übrigbleiben, sondern alle verwendet werden sollen. Wir könnten dort auch Elemente angeben, die wir aus irgendeinem Grund ganz bewußt von der Analyse ausklammern möchten. Doch jetzt geht es um die vollständige Schaltung.

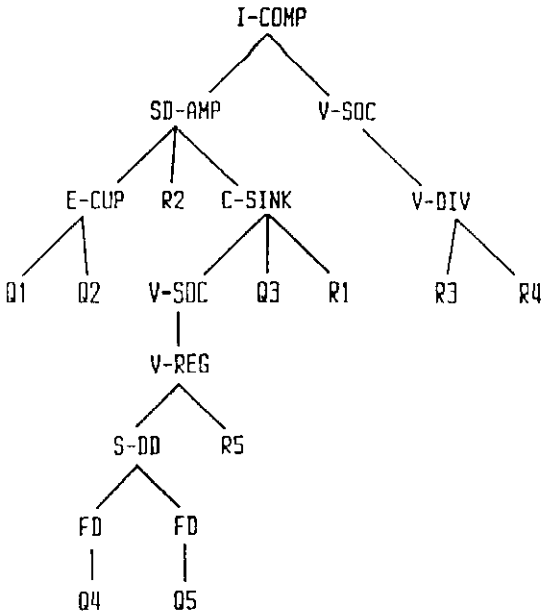
Aufgrund unserer Eingabe versucht Duck nun den Schaltkreis zu 'parsen'. Es wird dabei nach dem Top-Down-Verfahren vorgegangen, d. h. `I-COMPARATOR` ist sozusagen das Startsymbol, und es soll versucht werden, den Schaltkreis aus diesem Startsymbol herzuleiten. Der dabei entstehende Syntaxbaum wird uns dann genau die Zusammenhänge zwischen den einzelnen Elementen anzeigen.

Da `I-COMPARATOR` aus einem Differenzverstärker und einer Spannungsquelle aufgebaut ist, versucht Duck zunächst der Reihe nach, diese beiden Module im Schaltkreis zu finden. Da es sich dabei um keine elementaren Bauteile (Elemente) bzw. Terminalsymbole im übertragenen Sinne handelt, werden die auftretenden Prädikate solange durch ihre entsprechenden Regeln ersetzt, bis schließlich nur noch diskrete Elemente übrig sind. Nach diesen wird dann in der entsprechenden Konstellation im Schaltkreis gesucht. Dabei kommt es natürlich auch vor, daß Duck evtl. ein falsches Modul zum falschen Zeitpunkt anzuwenden versucht (bei Regeln die Disjunktionen enthalten). In einem solchen Fall wird das angestrebte Ziel nicht erreicht, und es muß ein anderer Weg versucht werden. Es wird also nicht genau einmal im Syntaxbaum von oben nach unten gelaufen, sondern es muß - sehr oft sogar - umgekehrt, und ein anderer Weg eingeschlagen werden (Stichwort: Backtracking). Schließlich kommt Duck jedoch zu einem Ergebnis welches im Falle unseres Beispiels positiv ausfällt. Das bedeutet, daß die Schaltung in der Tat einen Current-Comparator darstellt. Aus der Struktur des bei der Herleitung konstruierten Namens kann man sofort den dazugehörigen Syntaxbaum ableiten[18]:

```
NAME = (I-COMP (SD-AMP (E-CUP Q1 Q2)
                       R2
                       (C-SINK (V-SOC (V-REG (S-DD (FD Q4)
                                                   (FD Q5))
                                             R5))
                                       Q3
                                       R1))
               (V-SOC (V-DIV R3 R4)))
IN = #9
OUT = #10
POW = #2
GND = #1
```

[18] Da hier nicht alle definierten Funktionsblöcke beschrieben wurden folgt eine kurze Auflistung der Bedeutung der verwendeten Namen:
I-COMP = current comparator, SD-AMP = single-ended differential amplifier, E-CUP = emitter-coupled pair of NPN-transistors, C-SINK = current source, V-SOC = voltage source, V-REG = voltage regulator, S-DD = series circuit of diodes, V-DIV = voltage divider

Dieser Parse-Baum zeigt ganz genau, wie die einzelnen Schaltkreiselemente miteinander zu Funktionsblöcken verknüpft sind, und wie diese Blöcke wiederum zu komplexeren Modulen und schließlich dem kompletten Current-Comparator zusammengeschlossen werden. Die hierarchische Struktur des Schaltkreises ist hierdurch transparent geworden.



3.2.2.3 Die unterschiedlichen Anwendungsmöglichkeiten

In diesem Abschnitt soll gezeigt werden, welche unterschiedlichen Formen die Anfragen an das Duck-System haben können. Hierdurch erhält man einen sehr guten Überblick von den Möglichkeiten und der Leistungsfähigkeit der hier vorgestellten Methoden.

Anfragen der Form `(SUB-CT ?WHAT-CIRCUIT object-circuit !< >):`

Hier wird Duck angewiesen, einen gegebenen Schaltkreis komplett (also mit leerem Rest) zu untersuchen. Wenn die Lösung im Bereich der Duck bekannten Regeln zu finden ist, dann ist Duck in der Tat in der Lage, über die Variable `?WHAT-CIRCUIT` eine oder mehrere mögliche Lösungen auszugeben. Das Verfahren ist zwar äußerst ineffizient weil Duck sämtliche mögliche Produktionen an jeder Stelle ausprobieren muß, aber es funktioniert.

Anfragen der Form `(SUB-CT circuit object-circuit !< >):`

Hier wird Duck angewiesen, eine vorgegebene Struktur in einem gegeben Schaltkreis zu verifizieren. Dies ist genau das, was wir im vorangegangenen Beispiel haben durchführen lassen. Anfragen, die einem solchen Muster folgen werden sehr effizient gelöst und kommen in der praktischen Anwendung häufig vor. Sehr oft steht ein zu analysierender Schaltkreis in einem bestimmten Kontext und ein Ingenieur hat eine ungefähre Vorstellung von dem, was die Schaltung macht. Solche Vermutungen lassen sich schnell und einfach in Duck überprüfen.

Anfragen der Form `(SUB-CT ?WHAT-CIRCUIT obj.-circuit ?REST):`

Hier wird schließlich lediglich ein Schaltkreis angegeben und Duck angewiesen, beliebige Funktionsblöcke in ihm zu identifizieren, wobei dazu nicht der gesamte Schaltkreis verwendet werden muß. Obwohl dies äußerst ineffizient ist, wird Duck mit ziemlicher Sicherheit in jedem Schaltkreis irgendwelche bekannten Strukturen entdecken können. Gerade bei größeren, absolut unbekannten Schaltkreisen kann das

eine große Hilfe sein. Die einzelnen Funktionsblöcke, die Duck identifiziert können als Basis für weitere Untersuchungen dienen.

Schließlich gibt es noch eine letzte Anwendungsmöglichkeit, die quasi ein Nebenprodukt ist. Das Duck System kann auch in umgekehrter Richtung arbeiten, d. h. als Schaltkreisgenerator fungieren.

```
(SUB-CT (S-DIODE ?NAME #1 #2) ?CT !< >)
```

Diese Eingabe bewirkt die Ausgabe von allen möglichen Reihenschaltungen von Dioden, also eine unendliche Folge von Schaltkreisen. Das hier vorgestellte System ist nicht für solche Anwendungen entwickelt worden, kann aber in gewissem Umfang auch dafür benutzt werden.

Wie man sieht, gibt es eine ganze Reihe von Möglichkeiten, die einem für die Schaltkreisanalyse in diesem System zur Verfügung stehen. Doch man darf nicht vergessen, daß die Leistungskraft sehr stark mit der Menge der dem System bekannten Funktionsblöcke zusammenhängt.

3.3 Methoden zur Erweiterung

Sowohl die Bottom-Up-, als auch die Top-Down-Methode können natürlich noch optimiert, erweitert und verbessert werden. In diesem Abschnitt geht es um solche Erweiterungs- und Verbesserungsmöglichkeiten.

Bei der Bottom-Up-Analyse von Impedanz-Netzwerken haben wir Reihen- und Parallelschaltungen von Widerständen durch einen einzelnen Widerstand ersetzt. Dabei machten wir keinerlei Aussage über den Wert der neuen Elemente. Denkbar wäre eine Erweiterung um die Berechnung der resultierenden Impedanzen. Dies müßte allerdings aus Lisp heraus geschehen, da uns Duck keine Möglichkeiten zum Rechnen zur Verfügung stellt. Man könnte zu jedem Widerstand einen Impedanzwert speichern, und dann parallel zu den Umformungen jeweils die Impedanzen der neuen Widerstände berechnen.

Bei der Top-Down-Analyse der elektronischen Schaltkreise gehen die Erweiterungsmöglichkeiten allesamt in Richtung der Optimierung und der Effizienzsteigerung. Das bedeutet, es soll wenn möglich vermieden werden, daß Duck Fehlinterpretationen macht, also beispielsweise einen Spannungsteiler (ein Modul welches sehr häufig vorkommt) als eigenständiges Modul interpretiert, obwohl er in den meisten Fällen nur ein Bestandteil einer komplexeren Schaltung ist. In der Regel kann das dadurch vermieden werden, daß der Versuch, komplexere Strukturen zu erkennen, vor den weniger komplexen Strukturen durchgeführt wird. Allerdings kann man nicht in allen Fällen solche Fehlinterpretationen vermeiden. Aus diesem Grund ist es ratsam, das System weiter zu optimieren.

Eine Möglichkeit hierfür besteht in der Berücksichtigung bestimmter elektrischer Eigenschaften der einzelnen Baugruppen. Beispielsweise Informationen über Versorgungsspannungsanschlüsse oder reine Ausgänge. Derartige Zusatzinformationen schränken die Menge der anwendbaren Produktionen ein und steigern so die Effizienz. Auch eine Information darüber, daß ein bestimmter Anschluß nicht mit einem anderen verbunden werden darf kann die Lösungsmenge bereits stark einschränken und somit schneller zum Ziel führen.

Allgemein gilt: Je mehr Eigenschaften bekannt sind, und dem System mitgeteilt werden, desto effizienter wird es arbeiten und desto weniger falsche oder unsinnige Ergebnisse wird es liefern.

4 Zusammenfassende Betrachtung der Methoden

Insgesamt wurden drei verschiedene Analysemethoden für Schaltkreise vorgestellt, die auf zwei unterschiedlichen Repräsentationsmethoden beruhten.

Der erste vorgestellte Ansatz war recht intuitiv und schien zunächst sehr erfolgversprechend. Doch als es zur eigentlichen automatischen Schaltkreisanalyse kommen sollte scheiterte er in der Beziehung, als daß es nicht mehr möglich war, das angestrebte Ziel allein unter Verwendung des Deduktionssystems Duck zu erreichen. Lisp wurde zur externen Programmierung, also zur Steuerung des Systems benötigt. Dies verletzte natürlich die Konsistenz und stand in prinzipiellem Widerspruch zur Arbeitsweise eines rein auf Logik basierenden Systems. Außerdem war es sehr unschön, daß der ursprüngliche Schaltkreis durch die Transformationen zerstört wurde. Um diese Nachteile zu beseitigen wurden neue Verfahren vorgestellt, die zwar auf einer anderen Repräsentationsmethode der Schaltkreise basierten, jedoch glücklicherweise keine wesentliche Änderung der Syntax erforderten.

Das erste der beiden neuen Verfahren war die Bottom-Up-Methode zur Analyse von Impedanz-Netzwerken. Mit ihr konnten dieselben Schaltkreise wie im vorher vorgestellten Ansatz analysiert werden, nur mit dem Unterschied, daß dies nun komplett im Duck-System geschah und durch die benutzte Transformation der ursprüngliche Schaltkreis nicht zerstört wurde. Das neue Verfahren bot eine genaue Aufzeichnung der durchgeführten Schaltkreistransformationen, so daß man aus den im resultierenden Schaltkreis verwendeten Namen einen Baum konstruieren konnte, der vorbildlich die Struktur des Schaltkreises darstellte. Das Verfahren war vergleichbar mit dem Vorgang des Parsens von Sätzen oder Worten einer Sprache.

Für die Analyse von elektronischen Schaltkreisen wurde ein anderes Verfahren, die Top-Down-Methode, vorgestellt. Sie war vergleichbar mit der Produktion von Worten oder Sätzen aus dem Startsymbol einer Grammatik. Das Startsymbol war in diesem Fall eine Vermutung über die Funktion des Schaltkreises und Duck versuchte den Schaltkreis aus dem Startsymbol herzuleiten. Diese Vorgehensweise funktionierte für elektronische Schaltkreise sehr gut, da sie genau das Schema verkörpert, nach dem die meisten solchen Schaltkreise entworfen werden. Elektronische Schaltkreise werden in der Regel aus vielen einzelnen Funktionsblöcken zusammengesetzt. Solche Funktionsblöcke finden ihre grammatikalische Entsprechung in Non-Terminals und die diskreten Schaltkreiselemente entsprechen den Terminalsymbolen der Grammatik. Die verschiedenen Einsatzmöglichkeiten dieses Analyseverfahrens waren überzeugend. Es war sowohl möglich, bestimmte Vermutungen zu verifizieren, als auch Analysen an vollständig unbekannten Schaltkreisen durchzuführen.

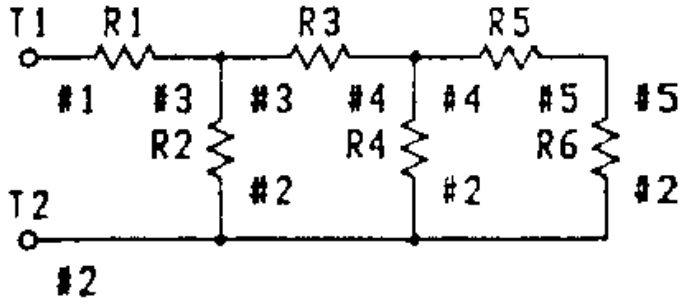
Neben der Darstellung der Analogie zwischen physikalischen Schaltkreisen und formalen Sprachen haben die vorgestellten Verfahren einen guten Einblick in die Möglichkeiten eines Deduktionssystems wie Duck geboten. Sie haben aber auch gezeigt, daß die grundsätzliche zur Modellbildung gewählte Methode ganz entscheidend für die spätere Leistungsfähigkeit und den Nutzen eines solchen Systems ist.

5 Verwendete Literatur

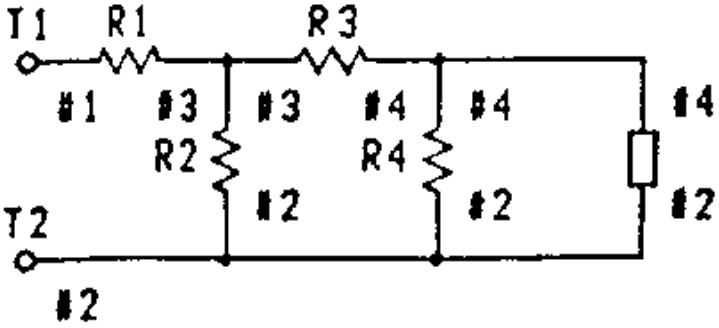
Takushi Tanaka, *„Structural Analysis of Electronic Circuits in a Deductive System“*, The National Language Research Institute, Tokio, Japan, 1986

6 Anhang - Beispiel einer Bottom-Up-Analyse

Der folgende Schaltkreis soll mit `M-SP-EQIVCT` transformiert werden:

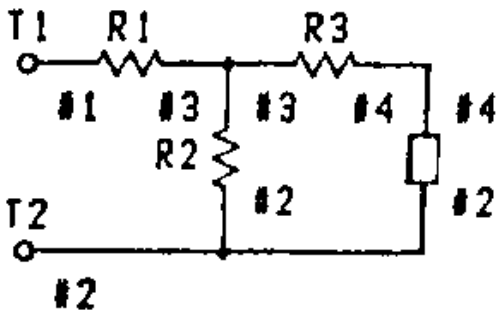


'`kreis`' stehe im folgenden für das diesen Schaltkreis beschreibende Tupel. Wir starten den Prozeß, indem wir '`(M-SP-EQIVCT kreis ?EQCT)`' eingeben. Jetzt probiert Duck zunächst '`(SP-EQIVCT kreis ?EQ1)`' auszuführen, was die Ausführung von '`(SR-EQIVCT kreis ?EQ1)`' oder '`(PR-EQIVCT kreis ?EQ1)`' nach sich zieht. Da bei dieser Disjunktion zunächst `SR-EQIVCT` ausprobiert wird und auch erfolgreich angewandt werden kann (Widerstände R5 und R6), wird zuerst eine Reihentransformation auf dem Schaltkreis durchgeführt, die folgendes Bild bzw. Tupel kreis' liefert:




```
kreis' = !< (Z-ELEMENT (SR R5 R6) #4 #2) (TERMINAL T1 #1)
            (TERMINAL T2 #2) (RESISTOR R1 #1 #3)
            (RESISTOR R2 #3 #2) (RESISTOR R3 #3 #4)
            (RESISTOR R4 #4 #2) >
```

Nun wird der zuvor beschriebene Prozeß mit dem Schaltkreis `kreis'` wiederholt. Diesmal wird eine Paralleltransformation durchgeführt, und zwar mit Widerstand R4 und dem im letzten Schritt neu konstruierten '`Z-ELEMENT (SR R5 R6)`'. Das führt uns zu folgendem Schaltkreis `kreis''`:




```
kreis'' = !< (Z-ELEMENT (PR (SR R5 R6) R4) #4 #2)
             (TERMINAL T1 #1) (TERMINAL T2 #2)
             (RESISTOR R1 #1 #3) (RESISTOR R2 #3 #2)
             (RESISTOR R3 #3 #4) >
```

Eine weitere Reihentransformation auf `kreis''` ergibt `kreis'''`:

```
kreis''' = !< (Z-ELEMENT (SR (PR (SR R5 R6) R4) R3) #3 #2)
              (TERMINAL T1 #1) (TERMINAL T2 #2)
              (RESISTOR R1 #1 #3) (RESISTOR R2 #3 #2) >
```

...und dann eine Paralleltransformation:

```
kreis'''' = !< (Z-ELEMENT (PR (SR (PR (SR R5 R6) R4) R3) R2) #3 #2)
               (TERMINAL T1 #1) (TERMINAL T2 #2)
               (RESISTOR R1 #1 #3) >
```

...und schließlich eine Reihentransformation:

```
kreis''''' = !< (Z-ELEMENT (SR (PR (SR (PR (SR R5 R6) R4) R3) R2) R1) #2 #1)
                (TERMINAL T1 #1) (TERMINAL T2 #2) >
```

Da keine weitere Transformation auf `kreis'''''` mehr möglich ist wird abgebrochen, und `kreis'''''` ist unser Ergebnis, welches schließlich in der Variable `?EQCT` vorliegt. Dieser Schaltkreis hätte beim Test mit dem Prädikat `SERIES-PARLLEL-CIRCUIT` ein positives Ergebnis geliefert, da er nach der Transformation aus genau einem `Z-ELEMENT` und zwei Anschlußpolen besteht. Zu beachten ist, daß der ursprüngliche Schaltkreis, so wie er vor den 5 Transformationen war, immer noch unverändert existiert.