Universität Paderborn
Fachbereich Mathematik und Informatik

Diplomarbeit

Layout-Graphgrammatiken für die Darstellung von hierarchisch strukturierten Graphen am Beispiel von Wellendigitalstrukturen

vorgelegt bei
Prof. Dr. Hans Kleine Büning

von

Frank Benteler

August, 2002

Gesetzt mit pdf- / LaTeX am 14. August 2002.

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbständig und ohne unerlaubte fremde Hilfe angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen und Hilfsmittel wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Ort,	*Datum,*	*Unterschrift*

Inhaltsverzeichnis

1. Einführung/Problemstellung **9**

I. Wellendigitalstrukturen **11**

2. Einführung in Wellendigitalstrukturen **13**
2.1. Grundelemente 14
2.2. Adaptoren 16
2.3. Beispiel 17

3. Abbildung passiver elektrischer Schaltungen auf Wellendigitalstrukturen **19**
3.1. SPC-Baum 20
3.2. Algorithmus ADAPTORS() 21

II. Layout-Graphgrammatiken **25**

4. Einführung in Graphgrammatiken **27**
4.1. Begriffserklärungen / Definitionen 27
4.2. Beispiel 31
4.3. Eigenschaften / Komplexitätsbetrachtungen 33

5. Einführung in Layout-Graphgrammatiken **35**
5.1. Berechnung der Positionen 36
5.2. Beispiel 38
5.3. Layout-Suche 43

6. Layout-Graphgrammatik für WDS **45**
6.1. Graphgrammatik für WDS 50
6.2. Ableitungstabelle / Graph-Parser 59
6.3. Beispiel 62
6.4. Suchstrategie 67

7. Zusammenfassung und Ausblick **69**

Abbildungsverzeichnis

1.1. Die prinzipiellen Schritte der Transformation einer elektrischen Schaltung in eine WDS . . . 9

2.1. Torbedingung $i_1 = i_2$ . . . 14
2.2. Elektrische Bauteile und ihre Wellenflussgraphen . . . 16
2.3. Serienverbindung und Serienadaptoren . . . 17
2.4. Parallelverbindung und Paralleladaptoren . . . 17
2.5. Wellendigitalstruktur einer elektrischen Schaltung . . . 18

3.1. Dreifach verbundene elektrische Netzwerke . . . 19
3.2. Transformation einer elektrischen Schaltung in einen SPC-Baum . . . 20
3.3. Elektrische Schaltung S eines Cauer-Filters . . . 22
3.4. Erstellung des Graphen G der Schaltung S . . . 22
3.5. Anreichern des Graphen G_1 mit virtuellen Kanten . . . 22
3.6. Abbilden des Graphen G_1 in den SPC-Baum T_1 . . . 23
3.7. SPC-Baum T des Graphen G und normalisierte Form . . . 23
3.8. Vom SPC-Baum zu einer WDS . . . 24

4.1. Aufbau einer Produktion . . . 29
4.2. Beispiel für Abeitungschritt $G \Rightarrow_p G'$ . . . 29
4.3. Produktionen der Graphgrammatik . . . 31
4.4. Produktionen p_i der Ableitung δ . . . 31
4.5. Graphen der terminalen Ableitung aus Beispiel 4.2 . . . 32
4.6. Ableitungsbaum zur Ableitung δ . . . 33

5.1. Nicht berechenbare bzw. ungültige Positionierungen . . . 36
5.2. Weitere nicht berechenbare bzw. ungültige Positionierungen . . . 37
5.3. Layout-Produktion (P_1, $c_{1,1}$) und (P_2, $c_{2,1}$) . . . 38
5.4. Graphen G_0 bis G_5 der Layout-Ableitung . . . 39
5.5. Bedingungen für x- und y-Position der Knoten des Graphen G_2 . . . 39
5.6. Ungültige Positionierung . . . 41
5.7. Neue Menge von Positionsbedingungen $c_{2,2}$ . . . 42
5.8. Die entstehenden Graphen G'_1 bis G'_3 . . . 42

6.1. Produktionen von GG_{Try} . . . 46
6.2. Layoutproduktionen (P_0, $c_{0,1}$) und (P_2, $c_{2,1}$) . . . 46

6.3. Layoutproduktionen (P_1, $c_{1,1}$) bis (P_1, $c_{1,6}$) 47
6.4. Gegebener Graph 47
6.5. Ableitungsbaum zum Graphen in Abbildung 6.4 48
6.6. Layout-Ableitung, die zu einer ungültigen Positionierung führt 48
6.7. Verbleibende Positionierungen 49
6.8. Teil 1 der Produktionen von GG_{WDS} 50
6.9. Teil 2 der Produktionen von GG_{WDS} 51
6.10. Ableitungsbäume zu Graph von Seite 47. 51
6.11. Layoutproduktionen (P_0, $c_{0,1}$) und (P_{13}, $c_{13,1}$) bis (P_{16}, $c_{16,1}$) 52
6.12. Layoutproduktionen (P_1, $c_{1,1}$) und (P_1, $c_{1,2}$) 52
6.13. Layoutproduktionen (P_2, $c_{2,1}$) und (P_2, $c_{2,2}$) 52
6.14. Layoutproduktionen (P_3, $c_{3,1}$) und (P_3, $c_{3,2}$) 53
6.15. Layoutproduktionen (P_4, $c_{4,1}$) und (P_4, $c_{4,2}$) 53
6.16. Layoutproduktionen (P_6, $c_{6,1}$) und (P_6, $c_{6,2}$) 53
6.17. Layoutproduktionen (P_9, $c_{9,1}$) und (P_9, $c_{9,2}$) 53
6.18. Graphen G_0 bis G_4 der Layout-Ableitung δ 55
6.19. Temporäre Positionierungen des Graphen G_4 56
6.20. Ableitungen vom Startgraph zum Graph G 59
6.21. Darstellung der Ableitungstabelle als Graph 63
6.22. Suchraum der LGG_{WDS} für Beispiel 6.3 64
6.23. Suchraum der LGG_{WDS} für Beispiel 6.3 65
6.24. Positionierungen eines SP-Baums 66

1. Einführung/Problemstellung

Graphgrammatiken sind eine Verallgemeinerung von Wortgrammatiken auf Graphen. Sie bestehen aus endlich vielen Graphersetzungsregeln (oder Produktionen) und erlauben eine endliche Beschreibung einer unendlichen Klasse von knoten- und kantenmarkierten Graphen.

Layout-Graphgrammatiken (LGG) sind Graphgrammatiken, die neben der Erstellung eines Graphen auch Positionsbedingungen mitgenerieren. Diese Bedingungen bestimmen, wo die Knoten und Kanten des Graphen zu positionieren sind.

Ziel dieser Diplomarbeit ist es, zu untersuchen, ob für einen gegebenen Baum mit Layout-Graphgrammatiken eine angemessene Positionierung mit vertretbarem Aufwand berechnet werden kann. Die berechnete Positionierung sollte Eigenschaften besitzen wie gut lesbar, verständlich und kompakt, wobei der Berechnungsaufwand nicht zu groß werden darf, da die Berechnung von flächenminimalen Positionierungen NP-hart[1] ist. Es werden hier daher Heuristiken benutzt.

Als Beispiel soll hier ein Positionierungsproblem aus der Elektrotechnik dienen, und zwar die Berechnung einer Positionierung für *Wellendigitalstrukturen*, bei denen es sich um digitale Nachbildungen elektrischer Schaltungen handelt.

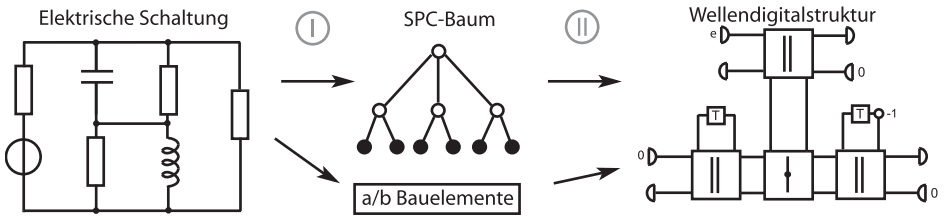

Abbildung 1.1.: Die prinzipiellen Schritte der Transformation einer elektrischen Schaltung in eine WDS

[1]Eine Einführung in die Theorie der NP-Vollständigkeit befindet sich in [Garey, Johnson 1979].

Der erste Schritt der Nachbildung besteht darin, die elektrische Schaltung in einen so genannten SPC-Baum abzubilden. Ein SPC-Baum kann als eine Datenstruktur angesehen werden, die alle strukturellen Informationen der WDS beinhaltet.
Im zweiten Schritt wird der SPC-Baum in eine WDS abgebildet. Bei dieser Abbildung entsteht das eben beschriebene Positionierungsproblem. Es muss entschieden werden, wo die einzelnen Komponenten in der Fläche zu zeichnen sind. Dieses Problem wird mit Layout-Graphgrammatiken gelöst.

Der Rest dieser Arbeit ist wie folgt aufgeteilt:

- Im Teil 1 werden Wellendigitalstrukturen erläutert,
  - Kapitel 1 ist diese Einführung,
  - Kapitel 2 behandelt die mathematischen Grundlagen der Wellendigitalstrukturen,
  - Kapitel 3 stellt einen Algorithmus vor, der die Abbildung einer passiven elektrischen Schaltung in eine Wellendigitalstruktur automatisiert.
- Teil 2 befasst sich mit Layout-Graphgrammatiken.
  - Kapitel 4 gibt eine Übersicht über grundlegende Eigenschaften von Graphgrammatiken,
  - Kapitel 5 liefert eine Beschreibung von den in dieser Arbeit verwendeten Layout-Graphgrammatiken,
  - Kapitel 6 beschreibt eine Layout-Graphgramatik für Wellendigitalstrukturen,
  - Kapitel 7 beinhaltet die Zusammenfassung und den Ausblick.

Teil I.

Wellendigitalstrukturen

2. Einführung in Wellendigitalstrukturen

In diesem Kapitel wird kurz gezeigt, was Wellendigitalstrukturen sind. Die hier vorgestellten Grundlagen sind bereits zahlreich in der Literatur erläutert und stellen lediglich eine Zusammenfassung dar, in der auf die Eigenschaften hingewiesen wird, die zum Verständnis der Problemstellung erforderlich sind. Für eine allgemeine Einführung in das Konzept der Wellendigitalstrukturen siehe z.B. [K.Ochs, Stein 2001].

Wellendigitalstrukturen (kurz WDS) können zur digitalen Nachbildung passiver analoger elektrischer Schaltungen verwendet werden und sind eine spezielle Art von Wellenflussgraphen. Sie verallgemeinern das von Fettweis [Fettweis 1986] vorgeschlagene Verfahren zur Erzeugung von digitalen Filtern, das zur digitalen Nachbildung zeitkontinuierlicher Filter verwendet wird.

Als Signalgrößen werden nicht unmittelbar Spannungen u und Ströme i verwendet, sondern die aus der Streuparametertheorie bekannten *Wellengrößen* a und b. Werden zwei Klemmen eines Netzwerkes zu einem Tor zusammengefasst, so ist zusätzlich die sogenannte Torbedingung zu erfüllen, d.h. dass der in die eine Klemme des Tores hineinfliessende und der aus der anderen Klemme desselben Tores herausfliessende Strom identisch sein muss (Abbildung 2.1). Der Begriff der Wellengröße kennzeichnet eine Linearkombination aus Torstrom und -spannung. Dabei werden die Größen a und b über

$$a = u + Ri, \qquad b = u - Ri \tag{2.1}$$

definiert, wobei R eine positive Konstante ist, die als *Torwiderstand* bezeichnet wird. In diesem Zusammenhang heißt a die *einfallende* Welle und b die *ausfallende* oder *reflektierte* Welle. Durch einfache Umformung erhält man die Beziehung für die Rücktransformation in die Größen Spannung und Strom:

$$\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 1 & R \\ 1 & -R \end{pmatrix} \begin{pmatrix} u \\ i \end{pmatrix} \qquad \Rightarrow \qquad \begin{pmatrix} u \\ i \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ \frac{1}{R} & -\frac{1}{R} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}. \tag{2.2}$$

Die digitalen Nachbildungen von elektrischen Bauelementen werden in dieser Arbeit *Grundelemente* genannt und die Nachbildungen der Verbindungen zwischen den Bauelementen bezeichnet man mit *Adaptoren*.

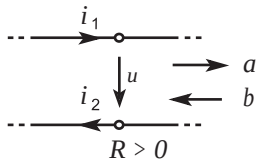

Abbildung 2.1.: Torbedingung $i_1 = i_2$

2.1. Grundelemente

Um eine elektrische Schaltung in eine entsprechende Wellendigitalstruktur umzusetzen, müssen zunächst die Kirchhoffschen Gleichungen und die entsprechenden Bauelementegleichungen aufgestellt werden. Diese Gleichungen werden dann in den Wellendigitalbereich transformtiert, wo sie ein System linearer Differenzengleichungen mit konstanten Koeffizienten bilden.
Die hier vorgestellten Transformationen ausgehend von der Strom- / Spannungsbeziehung an den elektrischen Bauteilen in die Gleichungen des Wellendigitalmodells, werden in der Abbildung 2.2 auf Seite 16 graphisch dargestellt.

Widerstand Im Fall des Widerstandes besteht zwischen Spannung und Strom der Zusammenhang:

$$u = R_1 i.$$

Mit der Definition der Wellengröße $b = u - Ri$ (Gleichung 2.1) berechnet sich die reflektierte Welle dementsprechend zu :

$$b = (R_1 - R)i.$$

Wählt man nun den Torwiderstand zu $R = R_1$ ergibt sich

$$b = 0. \tag{2.3}$$

Resistive Spannungsquelle Die WDS einer resistiven Spannungsquelle erhält man ähnlich, wie beim zuvor betrachtete Widerstand. Wird die Bauelementegleichung

$$u = e - R_0 i$$

in die Definition 2.1 eingesetzt, so berechnet sich die einfallende Welle mit $R = R_0$ zu

$$a = e - R_0 i + Ri,$$

$$a = e. \tag{2.4}$$

Kapazität Die Beziehung zwischen Strom und Spannung an einer Kapazität C ist gegeben durch die Differentialgleichung

$$i(t) = C\frac{d}{dt}u(t) \quad \Leftrightarrow \quad u(t) = u(t_0) + \frac{1}{C}\int_{t_0}^{t} i(\tau)d\tau.$$

Im folgenden sei $u_k = u(t_k)$ und $i_k = i(t_k)$.
Durch den Übergang von dem zeitkontinuierlichen in den zeitdiskreten Bereich, geht die Gleichung über in

$$u_k = u_{k-1} + \frac{1}{C}\int_{t_{k-1}}^{t_k} i(\tau)d\tau, \qquad t_k = t_0 + kT, k \in I\!N.$$

Wird zur Diskretisierung des Integrals die Trapezregel angesetzt, so folgt

$$u_k = u_{k-1} + \frac{1}{C}\frac{T}{2}(i_k + i_{k-1}).$$

Durch Festlegung des Torwiderstandes zu $R = \frac{T}{2C}$ wird die Gleichung zu

$$u_k - Ri_k = u_{k-1} + Ri_{k-1}$$

und nach einsetzen der Definition 2.1 ergibt sich für die Kapazität

$$b_k = a_{k-1}. \tag{2.5}$$

Induktivität Die Bauelementegleichung einer Induktivität L ist durch die Differentialgleichung

$$u(t) = L\frac{d}{dt}i(t) \quad \Leftrightarrow \quad i(t) = i(t_0) + \frac{1}{L}\int_{t_0}^{t} u(\tau)d\tau$$

gegeben. Wird wieder der Übergang von dem zeitkontinuierlichen in den zeitdiskreten Bereich durchgeführt und das Integral durch die Trapezregel angenähert, so geht die Gleichung über in

$$i_k = i_{k-1} + \frac{T}{2L}(u_k + u_{k-1}) \quad \Leftrightarrow \quad u_k - \frac{2L}{T}i_k = -(u_{k-1} + \frac{2L}{T}i_{k-1}).$$

Mit Wahl des Torwiderstandes zu $R = \frac{2L}{T}$ und nach einsetzen der Definition 2.1 lautet die Wellengleichung für eine Induktivität

$$b_k = -a_{k-1}. \tag{2.6}$$

Die Wellenflussgraphen der Gleichungen 2.3 bis 2.6 sind in der Abbildung 2.2 zu sehen.

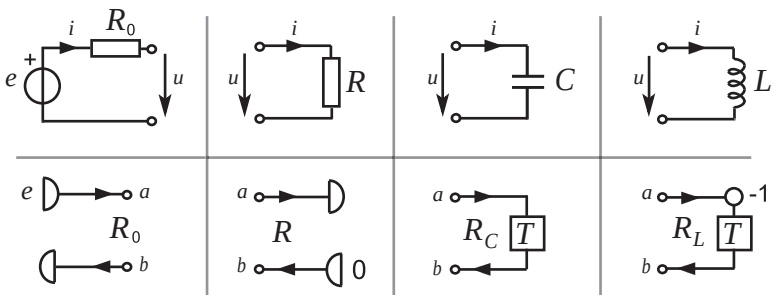

Abbildung 2.2.: Elektrische Bauteile und ihre Wellenflussgraphen

2.2. Adaptoren

Um aus den bisher betrachteten Bauelementen eine Wellendigitalstruktur aufzubauen, muss noch die bei der Torweisen Zerlegung der elektrischen Schaltung gefundene Topologie digital nachgebildet werden. Dieses geschieht mit Hilfe spezieller Mehrtorbausteine, die als *Adaptoren* bezeichnet werden.

Intern werden die Adaptoren durch Wellenflussgraphen beschrieben und lassen sich in *Serienadaptoren* (Abbildung 2.3 b und c), *Paralleladaptoren* (Abbildung 2.4 b und c) und weitere Adaptortypen unterteilen, wobei auf die letzteren hier nicht näher eingegangen wird.
Ein n-Tor-Paralleladaptor bildet eine Parallelverbindung von n Toren, und ein n-Tor-Serienadaptor eine Serienverbindung von n Toren.

Genau ein Tor eines Adaptors kann *reflexionsfrei* abgeschlossen werden, d.h die ausfallende Welle eines Tores ist dann unabhängig von der einfallenden Welle desselben Tores. Die Abbildungen 2.3 c und 2.4 c zeigen jeweils ein Adaptor mit reflexionsfrei abgeschlossenem Tor 3.

Jedes Tor eines Adaptors besitzt einen Torwiderstand. Dieser ist jedoch abhängig von dem angeschlossenen Bauteil. Zwei Tore lassen sich nur miteinander verbinden, wenn die Torwiderstände gleich sind. Das bedeutet, wenn an einem Tor eines Adaptors ein Bauteil angeschlossen ist, das den Torwiderstand R besitzt, muss auch das Tor des Adaptors den Torwiderstand R annehmen (siehe Abbildung 2.1). Des weiteren werden jedem Adaptor Adaptorkoeffizienten zugewiesen, auf deren Berechnung hier nicht weiter eingegangen wird, da sie nichts mit der späteren Positionierung der Adaptoren zu tun haben werden.

Jeder n-Tor-Parallel-Adaptor ($n \geq 3$) lässt sich durch eine Zusammenschaltung von 3-Tor-Parallel-Adaptoren ersetzen, daher wird im folgenden immer nur abkürzend Paralleladaptor verwendet (analog für Serienadaptoren).

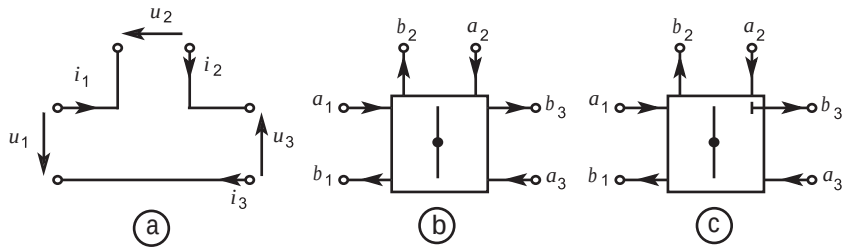

Abbildung 2.3.: Serienverbindung und Serienadaptoren

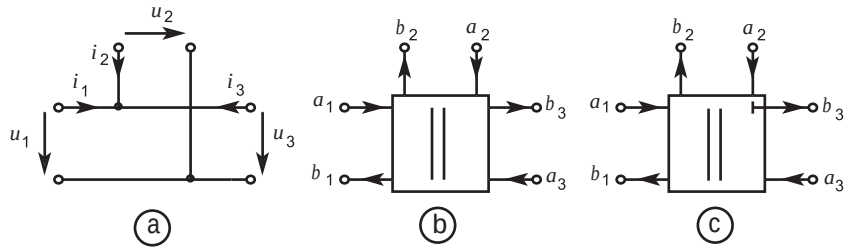

Abbildung 2.4.: Parallelverbindung und Paralleladaptoren

2.3. Beispiel

Zum Abschluss dieses Kapitels noch ein kleines Beispiel, in dem eine elektrische Schaltung in eine Wellendigitalstruktur überführt wird. Eine Veranschaulichung des Beispiels befindet sich in Abbildung 2.5 auf Seite 18.

Ausgehend von einem Verbindungsnetzwerk (Abb. 2.5 a) bestehend aus einer resistiven Spannungsquelle (e mit R_0), einem Widerstand (R) und einer Induktivität (L) wird eine Wellendigitalstruktur generiert. Zunächst werden die Stellen im Netzwerk eingezeichnet, bei denen die Torbedingung erfüllt ist (Abbildung 2.5 b).

Im zweiten Schritt werden die einzelnen Komponenten mit Hilfe der Tabelle in Abb. 2.2 in ihre Wellenflussgraphen überführt und deren Torwiderstände berechnet. Für den Widerstand R ist der Torwiderstand der Widerstand R selber, für die Spannungsquelle ist der Torwiderstand gleich R_0 und für die Induktivität berechnet sich der Torwiderstand zu $R_L = \frac{2L}{T}$ (Abb. 2.5 c).

Danach muss noch die Verbindungsstruktur der elektrischen Schaltung durch Adaptoren nachgebildet werden. Da es sich hier um eine Serienverbindung handelt, wird ein Serienadaptor verwendet (Abb.2.5 d). Die Torwiderstände des Adaptors sind dabei gleich den angeschlossenen Bauteilen zu wählen.

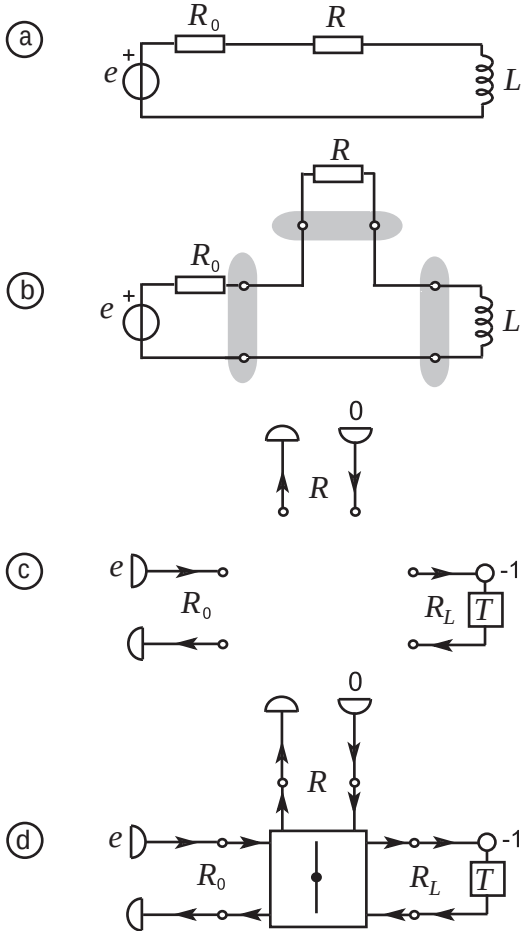

Abbildung 2.5.: Wellendigitalstruktur einer elektrischen Schaltung

3. Abbildung passiver elektrischer Schaltungen auf Wellendigitalstrukturen

Ausgehend von einer elektrischen Schaltung werden folgende Schritte bei der Abbildung einer elektrischen Schaltung in eine Wellendigitalstruktur durchgeführt:

1. Torweise Zerlegung der elektrischen Schaltung
2. Abbildung der elektrischen Bauteile in den Wellendigitalbereich
3. Erzeugung einer geeigneten Adaptorstruktur

Die Abbildung der Komponenten kann mit Hilfe einer Tabelle erfolgen, in der alle Bauelemente mit deren Wellenflussgraphen aufgelistet sind (siehe Abbildung 2.2 auf Seite 16).

Die Schwierigkeit in dem Vorgehen liegt darin, eine *geeignete* Adaptorstruktur zu finden. Es gibt elektrische Schaltungen, die nicht nur aus Serien- und Parallelverbindungen bestehen. In der Abbildung 3.1 befinden sich Beispiele für solche Schaltungen.

Diese Strukturen werden als dreifach zusammenhängende Komponenten oder C-Komponenten bezeichnet und müssen in einer gegebenen Schaltung erkannt und gesondert behandelt werden.

Um nun für jede Schaltung eine geeignete Adaptorstruktur zu finden und die in der Schaltung möglicherweise auftretenden C-Komponenten zu entdecken, wird hier

Abbildung 3.1.: Dreifach verbundene elektrische Netzwerke

ein Algorithmus von [K.Ochs, Stein 2001] benutzt, um eine WDS zu generieren. Dieser Algorithmus bildet die elektrische Schaltung in einen SPC-Baum ab.
Die Transformation wird aufgespalten in eine globale und eine lokale Transformation. Global bezieht sich hier darauf, dass der gesamte Graph als Ganzes für die Herleitung der Adaptorstruktur betrachtet werden muss. Die lokale Transformation bezeichnet hier, dass jedes Bauteil unabhängig von den anderen Bauteilen in den entsprechenden Wellenflussgraphen abgebildet werden kann.

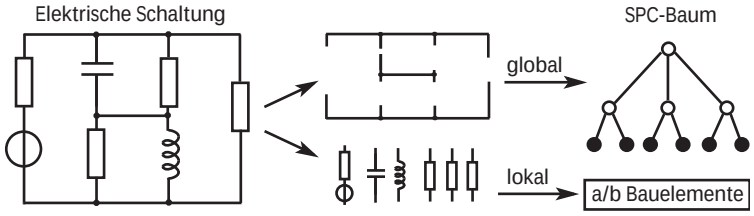

Abbildung 3.2.: Transformation einer elektrischen Schaltung in einen SPC-Baum

Der durch den Algorithmus entstehende SPC-Baum wird dann in einem zweiten Schritt in eine WDS umgewandelt.

3.1. SPC-Baum

Ein SPC-Baum kann als eine Datenstruktur angesehen werden, die alle strukturellen Informationen der WDS beinhaltet.

Definition SPC-Baum Ein SPC-Baum ist ein Baum mit Wurzel-Knoten, dessen Knoten vom Typ S-Knoten, P-Knoten, C-Knoten oder Blatt-Knoten sind. Die Nachfolger eines S-Knoten sind geordnet.
Ein S-Knoten (bzw. P-Knoten) steht für die Serienschaltung (bzw. Parallelschaltung) seiner Nachfolger-Knoten.
Ein C-Knoten steht für eine dreifach zusammenhängende Komponente und folgt keiner bestimmten Gesetzmäßigkeit der Verbindung seiner Nachfolger.
Die Blätter eines SPC-Baums stehen eins zu eins für die Bauteile in der elektrischen Schaltung.

Bei einem C-Knoten geht die Information verloren, wie die Nachfolger miteinander verbunden sind.

3.2. Algorithmus ADAPTORS()

Der Algorithmus ADAPTORS() stammt aus [K.Ochs, Stein 2001] und bestimmt für eine elektrische Schaltung S einen entsprechenden SPC-Baum.

Input: eine elektrische Schaltung S
Output: Ein SPC-Baum

1. Generiere den entsprechenden Graph G von S
2. Partitioniere G entsprechend den zweifach zusammenhängenden Komponenten $G_1, \ldots, G_m$. Für jedes G_i sind die folgenden Schritte erforderlich:
   a) Überprüfe auf unzulässige Aufteilung
   b) Erkenne dreifach zusammenhängende Komponenten
   c) Erzeuge einen SPC-Baum
3. Erzeuge den SPC-Baum T des gesamten Graphen G
4. Normalisiere T
5. Berechne die Torwiderstände und Adaptorkoeffizienten

Der Algorithmus wird nun an einem Beispiel erläutert.

Beispiel
Hier wird ein Cauer-Filter 7-ter Ordnung mit resistivem Abschluss in einen SPC-Baum transformiert. Das Beispiel ist so gewählt, dass keine C-Komponenten vorkommen, und da die Schaltung zweifachzusammenhängend ist, auch keine unzulässige Aufteilung existiert.

Die elektrische Schaltung S des Filters ist in der Abbildung 3.3 zu sehen.

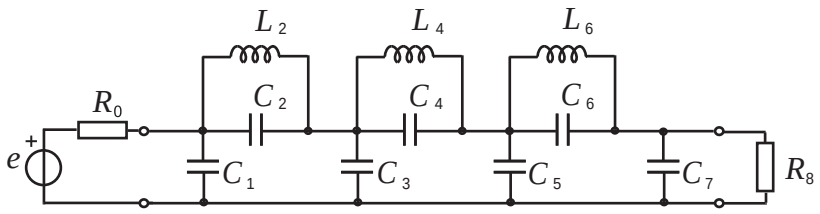

Abbildung 3.3.: Elektrische Schaltung S eines Cauer-Filters

Im ersten Schritt wird der Graph G der Schaltung S erzeugt. Dieses geschieht dadurch, dass alle Bauteile durch Kanten ersetzt, und Verbindungspunkte in der Schaltung zu Knoten des Graphen G werden. Kurzschlüsse zwischen Verbindungspunkten werden zu einem Knoten zusammengefasst. Der dabei entstehende Graph G ist in Abbildung 3.4 rechts gezeichnet.

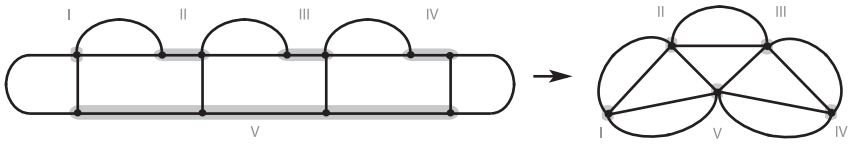

Abbildung 3.4.: Erstellung des Graphen G der Schaltung S

Die Partitionierung des Graphen G entsprechend den zweifach zusammenhängenden Komponenten geschieht durch auftrennen jeweils eines Knotens des Graphen G, sodass dadurch zwei Graphen entstehen, die nicht miteinander verbunden sind. Die so resultierenden Subgraphen sind die Graphen G_1 bis G_m. In diesem Beispiel gibt es keine solche Zerlegung, es gibt nur den einen Subgraphen $G_1 = G$.
Das Erkennen von dreifach zusammenhängenden Komponenten beginnt mit dem Einfügen von sog. *virtuellen Kanten* in den Graphen G_1. Virtuelle Kanten treten immer paarweise auf und trennen den Graphen G in jeweils zwei nicht miteinander verbundene Graphen (graue Kanten in Abbildung 3.5 links).

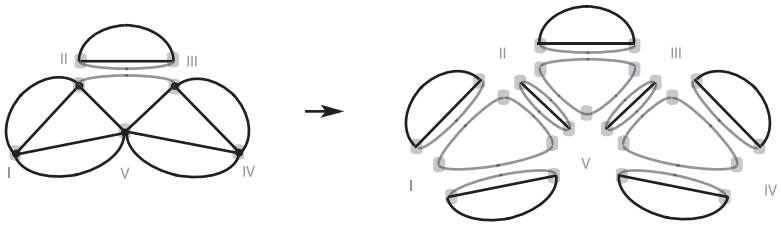

Abbildung 3.5.: Anreichern des Graphen G_1 mit virtuellen Kanten

Dabei entsteht eine Aufteilung des Graphen in Serien-, Parallel- und C-Komponenten. Die vollständige Zerlegung des Graphen G_1 ist in Abb. 3.5 rechts dargestellt.
Eine Parallel- (bzw. Serien-) Komponente mit n Kanten wird bei der Erstellung des SPC-Baums in einen P-Knoten (bzw. S-Knoten) mit n Nachbarn überführt (siehe Abbildung 3.6 links und in der Mitte).
Durch das Zusammenfügen von diesen (Teil-)Bäumen (vgl. Abbildung 3.6 rechts) wird nach und nach der SPC-Baum T_1 des Subgraphen G_1 erzeugt.

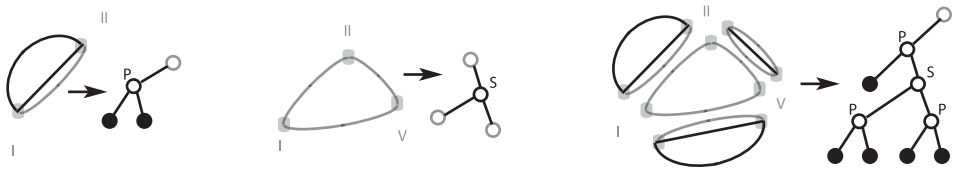

Abbildung 3.6.: Abbilden des Graphen G_1 in den SPC-Baum T_1

Die Erzeugung des SPC-Baums T des gesamten Graphen G ist nun einfach; da nur ein Subgraph G_1 existiert, ist somit der SPC-Baum T gleich dem SPC-Baum des Subgraphen G_1 : $T = T_1$ (dargestellt in Abb. 3.7 links).
Das Normalisieren des SPC-Baums ist in zwei Schritten durchzuführen. Zuerst werden alle S- bzw. P-Knoten mit mehr als drei Nachbarn durch einen vollständigen, binären S- bzw. P-Baum ersetzt, dessen innere Knoten nur jeweils drei Nachbarn haben. In diesem Beispiel kommt so etwas nicht vor (siehe hierzu [K.Ochs, Stein 2001]).
Im nächsten Schritt wird durch eine Zentrumsuche im SPC-Baum T ein Knoten ausgewählt, der zu allen Blättern minimale Pfadlänge hat. Dieser Knoten wird zur Wurzel des SPC-Baumes.

Das Ergebnis des Normalisierens und zugleich das Ergebnis dieses Beispiels ist in Abbildung 3.7 rechts dargestellt.

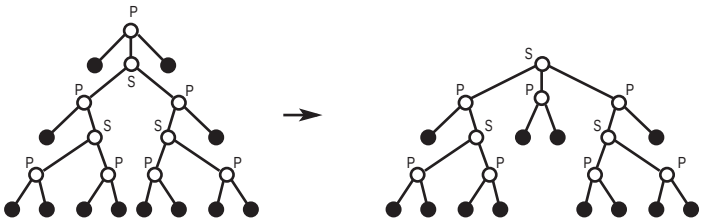

Abbildung 3.7.: SPC-Baum T des Graphen G und normalisierte Form

Der SPC-Baum enthält bis auf die Verbindungsstruktur der C-Komponenten alle Informationen der ursprünglichen Schaltung S.

In den nun folgenden Kapiteln wird darauf eingegangen, wie mit den Informationen des SPC-Baums eine WDS positioniert werden kann.

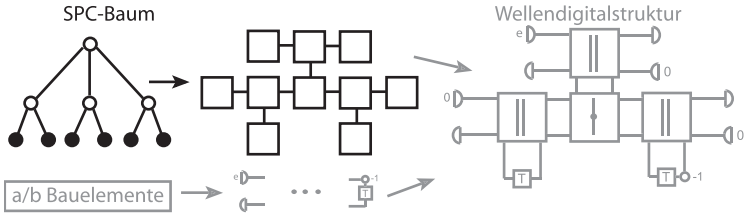

Abbildung 3.8.: Vom SPC-Baum zu einer WDS

In dieser Arbeit werden nur Parallel- bzw. Seriellgraphen berücksichtigt. Es werden also SPC-Bäume betrachtet, in denen keine C-Komponenten vorkommen, daher wird im folgenden immer nur der Begriff SP-Baum benutzt.

Teil II.

Layout-Graphgrammatiken

4. Einführung in Graphgrammatiken

Es wird vorausgesetzt, dass der Leser mit Grundzügen der Graphentheorie vertraut ist. Um ein einheitliches Vokabular zu garantieren, werden hier kurze Erklärungen der Begriffe gegeben. Die betrachteten Graphen bestehen aus gerichteten, markierten Kanten und markierten Knoten. Knotenmarkierungen sind aus einem Alphabet Σ und Kantenmarkierungen sind aus einem Alphabet Δ[1].

Definition Graph Ein *Graph* G=(V, E, m) über (Σ, Δ) besteht aus einer endlichen *Menge von Knoten* V, einer *Knotenmarkierungsfunktion* m:V$\rightarrow \Sigma$ und einer *Menge von gerichteten, markierten Kanten* $E \subseteq \{(v, x, w) \mid v, w \in V \text{ und } x \in \Delta\}$. Sei e=(v, x, w) eine Kante von G. Dann heißt v *Quell-* und w *Zielknoten* von e. Die Knoten v und w sind *adjazent* und die Kante e ist *inzident* mit Knoten v und w. Eine ungerichtete Kante u besteht aus zwei einander entgegengesetzten, gerichteten Kanten mit der gleichen Markierung u={(v, x, w), (w, x, v)}.
V(G) bezeichne die Menge aller Knoten, E(G) die Menge aller Kanten und m_G die Knotenmarkierungsfunktion von G.

Ein Graph ist *zusammenhängend*, genau dann, wenn man den Graphen nicht in zwei unabhängige Teilgraphen aufspalten kann, ohne eine Kante zu durchschneiden. Der *Grad eines Knotens* ist die Anzahl seiner inzidenten Kanten, der *Grad eines Graphen* ist das Maximum über alle Knotengrade.

Die hier verwendeten Graphgrammatiken sind Knotenersetzungssysteme und eine Verallgemeinerung von kontextfreien Wortgrammatiken. Genau wie bei Wortgrammatiken wird zwischen terminalen und nichtterminalen Knotenmarkierungen unterschieden.

4.1. Begriffserklärungen / Definitionen

Der Kern einer Graphgrammatik ist eine Menge von Graphersetzungsregeln, im weiteren auch Produktion genannt, und eine spezielle nichtterminale Knotenmarkierung, das Axiom[2] der Graphgrammatik. Produktionen, deren linke Seite aus ge-

[1] Σ und Δ können auch zusammengefasst werden ($\Theta=\Delta \cup \Sigma$).

[2] Das Axiom ist wie das Startsymbol einer Grammatik: es markiert den ersten Knoten der Graphgrammatik, auf den dann die erste Produktion angewendet werden kann.

nau einem Knoten bestehen, werden *kontextfrei* genannt. Eine Produktion (L, R, C) besteht aus einer linken Seite L, wobei L ein nichtterminal markierter, einknotiger Graph ist, einer rechten Seite R, wobei R ein nicht-leerer Graph ist, und einer zusätzlichen dritten Komponente C, den Einbettungsregeln.

Eine Produktion ist auf einen Graphen G anwendbar, falls der Knoten v $\in$ L in G enthalten ist. Die Anwendung der Produktion, der so genannte Ableitungsschritt, transformiert G in einen neuen Graphen G', der wie folgt entsteht. Ersetze L durch R in G, d.h. entferne zunächst den Knoten v der linken Seite und all seine inzidenten Kanten in G, füge R in G' ein, und erzeuge nun, gemäß den Einbettungsregeln C, Kanten in G' zwischen Knoten von R und den Nachbarn N von v in G. Die Einbettungsregeln werden später formal erläutert.

Definition Graphgrammatik Eine *Graphgrammatik* GG ist ein Tupel (N, T $\cup$ Δ, P, S). Dabei bezeichnen N und T die *Alphabete der nichtterminalen und terminalen Knotenmarkierungen* und Δ das *Alphabet der Kantenmarkierungen.* S $\in$ N ist das *Axiom* und P eine endliche *Menge von Produktionen.*
Eine *Produktion* p $\in$ P ist ein Tupel der Form (L, R, C). Die *linke Seite* L von p ist ein einknotiger, nichtterminal markierter Graph. Die *rechte Seite* R von p ist ein Graph mit |V(R)| $\geq$1, dessen nichtterminale Knoten keine Schleifen[3] aufweisen.
Die *Einbettungsregeln* C von p sind eine Menge von Elementen der Form (x, d, u), wobei x $\in$ Δ, d $\in$ {in, out} und u $\in$ V(R) gilt.
Sei c=(x, d, u) eine Einbettungsregel. Ist d=in, so heißt c *Einbettungsregel für einlaufende Kanten.* Ist d=out, so heißt c *Einbettungsregel für auslaufende Kanten.*

Aufbau einer Produktion Eine Produktion p=(L, R, C) hat eine einfache graphische Repräsentation (vgl. Abbildung 4.1):

1. Der Knoten {v}=V(L) wird als umschließendes Rechteck mit Label in der linken oberen Ecke gezeichnet.

2. Der Graph R der rechten Seite wird in Form von Knoten und Kanten innerhalb des umschließenden Rechtecks gezeichnet.

3. Die Einbettungsregeln C der Produktion sind als Kanten, die das umschließende Rechteck schneiden, dargestellt. Wo oder wie die Kanten das Rechteck schneiden, ist nicht von Bedeutung.

An dieser Stelle sei bereits angemerkt, dass eine Produktion p=(L, R, C) für alle Produktionen steht, die für L und R isomorphe[4] Graphen besitzen. Eine Graphgrammatikproduktion steht also für alle isomorphen Kopien p' von p.

[3]Schleifen sind Kanten der Form (v, x, v).
[4]Zwei Graphen G, G' sind isomorph, gdw. es einen Isomorphismus π: V(G)$\rightarrow$V(G') gibt, so dass $\forall$ v $\in$ V(G) m_G(v) = $m_{G'}(\pi$(v)) und $\forall$ (v, x, w) $\in$ E(G), (π(v), w, π(w)) $\in$ E(G') gilt.

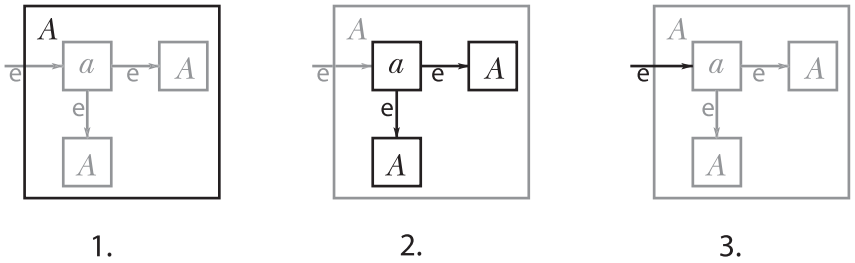

Abbildung 4.1.: Aufbau einer Produktion

Angewendet werden die Einbettungsregeln wie folgt. Seien N die Nachbarn von v in G. Es gibt genau dann eine Kante in G' mit Markierung x von einem Knoten w von N zu einem Knoten u aus R, wenn es eine Kante in G mit Markierung x von w nach v und eine Einbettungsregel für einlaufende Kanten (x, in, u) $\in$ C gibt. Analoges gilt für eine Einbettungsregel für auslaufende Kanten.

Definition Ableitungsschritt Sei GG eine Graphgrammatik, G ein Graph, mit $\{v\} \in V(G)$, p=(L, R, C) eine Produktion von GG mit $V(L)=\{v\}$ und $V(R) \cap V(G)=\emptyset$. Ein *Ableitungsschritt* $G \Rightarrow_p G'$ in GG ist wie folgt definiert: Ersetze L durch R in G, und erzeuge, gemäß den Einbettungsregeln C, die Kanten von R in G′.

- (w, x, u) ist eine einlaufende Kante von R in G′, gdw. es eine Kante (w, x, v) $\in$ E(G) und eine Einbettungsregel (x, in, u)$\in$ C gibt.
- (u, x, w) ist eine auslaufende Kante von R in G′, gdw. es eine Kante (v, x, w) $\in$ E(G) und eine Einbettungsregel (x, out, u)$\in$ C gibt.

Der Knoten $\{v\}=V(L)$ der linken Seite einer Produktion (L, R, C) legt also eindeutig den Knoten im Graphen G fest, der durch anwenden der Produktion ersetzt wird.

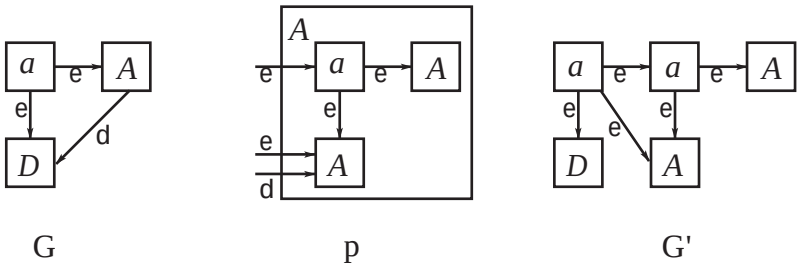

Abbildung 4.2.: Beispiel für Abeitungschritt $G \Rightarrow_p G'$

Die Abbildung 4.2 macht deutlich, dass dabei sowohl Kanten gelöscht, als auch Kanten verdoppelt werden können.
Die Kante von Knoten A zu Knoten D mit Markierung d in Graph G wird durch anwenden der Produktion p gelöscht. Da es in den Einbettungsreglen der Produktion p keine Regel für auslaufende Kanten mit Markierung d existiert, gibt es auch keine solche Kante in G′.
Die zwei Einbettungsregeln für einlaufende Kanten mit Markierung e in Produktion p, sind dafür verantwortlich, dass aus der einen Kante von Knoten a zu Knoten A mit Markierung e in G, zwei Kanten in G' werden.

Folgen von Ableitungsschritten bilden Ableitungen.

Definition Ableitung Sei GG eine Graphgrammatik und seien p_i=(L_i, R_i, C_i) Produktionen in GG $\forall\ i \in \{0, \ \ldots, \ \text{n-1}\}$. Eine *Ableitung* δ in GG ist eine Folge (G_0, p_0, G_1, p_1, ..., G_{n-1}, p_{n-1}, G_n), gdw.

- $G_i \Rightarrow_{p_i} G_{i+1}$ ist ein Ableitungsschritt in GG $\forall\ i \in \{0, \ \ldots, \ \text{n-1}\}$,
- die Knotenmengen $V(R_i)$ $\forall\ i \in \{0, \ \ldots, \ \text{n-1}\}$ sind paarweise disjunkt.

G_n wird als das *Ergebnis* von δ bezeichnet. Ist $|V(G_0)|$=1 und der Knoten von G_0 mit dem Axiom der Grammatik markiert, so heißt δ *initial*. Ist δ initial, und sind alle Knoten von G_n terminal markiert, so heißt δ *terminal*.

Jeder Knoten der linken Seite einer Produktion in einer Ableitung, mit Ausnahme des Knotens der linken Seite der ersten Produktion, ist ein Knoten der rechten Seite von genau einer anderen Produktion. Die Darstellung dieser Beziehung in einem Graphen führt zu Ableitungsbäumen.

Definition Ableitungsbaum Sei GG eine Graphgrammatik und δ=(G_0, p_0, ..., G_n) eine Ableitung in GG. Der *Ableitungsbaum* t_δ von δ ist ein Baum, dessen Knotenmenge die Menge aller Produktionen von δ ist. Es gibt eine Kante in t_δ von der Produktion p_i=(L_i, R_i, C_i) zu der Produktion p_j=(L_j, R_j, C_j) in t_δ, gdw. sich der Knoten {v}=$V(L_j)$ der linken Seite von p_j in der rechten Seite von p_i befindet {v} $\in V(R_i)$. Knoten- und Kantenmarkierungen finden keine Verwendung.

Im Kapitel 4.2 wird ein graphisches Beispiel für einen Ableitungsbaum gegeben. Wie gewohnt bilden die Ergebnisse aller terminalen Ableitungen in einer Graphgrammatik die Sprache der Graphgrammatik.

Definition Sprache einer Graphgrammatik Sei GG eine Graphgrammatik. Die von GG erzeugte *Sprache* L(GG) ist die Menge der Ergebnisse aller terminalen Ableitungen in GG.

Klassen von Graphen, die man beispielsweise mit Graphgrammatiken erzeugen kann, sind Bäume, Serien-Parallel-Graphen, Programmfluss-Graphen, bipatite Graphen, maximal außenplanare Graphen und vollständige Graphen. Man kann aber z.B. nicht Gitter-Graphen (grids), planare Graphen oder gerichtete azyklische Graphen mit Graphgrammatiken erzeugen [Brandenburg 1994].

4.2. Beispiel

Nun folgt ein Beispiel einer Ableitung δ in der Graphgrammatik GG.
GG=(N, T $\cup$ Δ, P, S)

- N={A} sei die Menge nichtterminale Knotenmarkierungen
- T={a} die Menge teminaler Knotenmarkierungen
- Δ={e} die Menge teminaler Kantenmarkierungen
- P={P_1, P_2} ist die Menge von Produktionen (siehe Abbildung 4.3)
- S=A $\in$ N sei das Axiom der GG

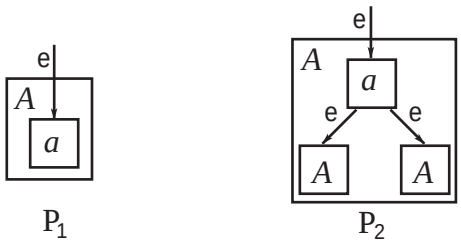

Abbildung 4.3.: Produktionen der Graphgrammatik

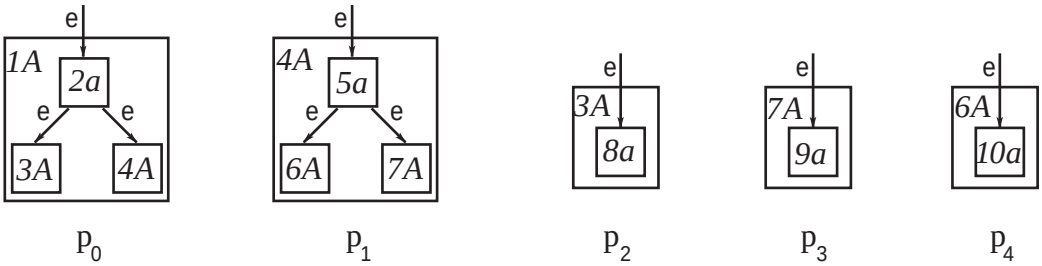

Abbildung 4.4.: Produktionen p_i der Ableitung δ

Die Ableitung δ sei gegeben zu (G_0, p_0, G_1, p_1, G_2, p_2, G_3, p_3, G_4, p_4, G_5), wobei p_i isomorphe Kopien der Produktionen P_1 bzw. P_2 sind. Der Knoten $\{v_i\}=V(L_i)$ der linken Seite einer Produktion $p_i=(L_i, R_i, C_i)$ ist jeweils in dem Graphen enthalten, auf den die Produktion angewendet wird $\{v_i\}\in V(G_i)$.

Die Produktionen $p_0, \ldots, p_5$ sind in Abbildung 4.4 auf Seite 31 und die Graphen G_i der Ableitung δ sind in Abbildung 4.5 graphisch dargestellt. Dabei besteht die Beschriftung der Knoten aus zwei Teilen: zum einen ist der Buchstabe die Markierung des Knotens, zum anderen ist die Zahl die Nummer des Knotens. Die Nummer dient hier nur zu Erklärungszwecken, und zwar um Knoten mit gleicher Markierung unterscheiden zu können.

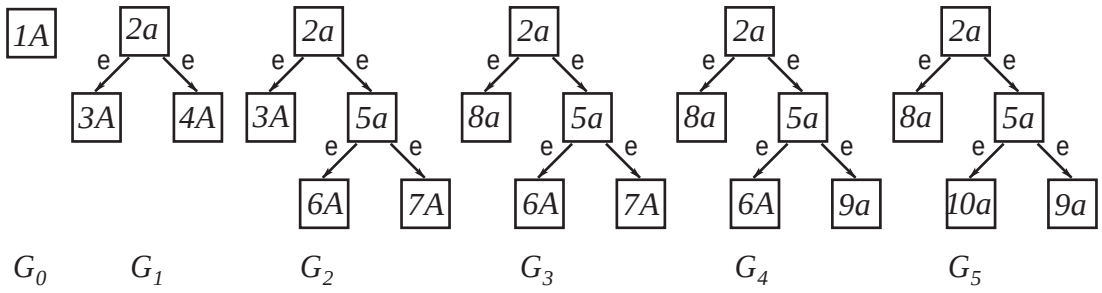

Abbildung 4.5.: Graphen der terminalen Ableitung aus Beispiel 4.2

Der erste Graph G_0 der Ableitung besteht nur aus dem Knoten 1A, der mit dem Axiom der Graphgrammatik markiert ist. Durch anwenden der Produktion p_0 entsteht der Graph G_1, indem der Knoten 1A gelöscht und die Knoten 2a, 3A und 4A mit entsprechenden Kanten hinzugefügt werden.
Beim Anwenden der Produktion p_1 auf den Grahen G_1 wird 4A durch die Knoten 5a, 6A und 7A ersetzt. Da sowohl eine Kante von 2a nach 4A mit Markierung e in G_1, als auch eine Einbettungsregel für einlaufende Kanten mit Markierung e in p_1 existiert, gibt es eine Kante von 2a nach 5a mit Markierung e in G_2.
Im dritten Ableitungsschritt, der zu G_3 führt, wird der Knoten 3A von der Produktion p_2 durch den Knoten 8a ersetzt. Die inzidente Kante bleibt dabei wie eben erhalten.
Die beiden verbleibenden Ableitungsschritte verlaufen analog.

Diese Ableitung ist terminal, da alle Knoten im Graphen G_5 terminale Markierungen haben und der einzige Knoten des ersten Graphen G_0 mit dem Axiom der Graphgrammatik markiert ist.
Der Ableitungsbaum t_δ dieser Ableitung δ ist in Abbildung 4.6 zu sehen. Der Knoten 4A der linken Seite von p_1 ist in den Knoten des Graphen der rechten Seite von p_0, daher gibt es eine Kante von p_0 nach p_1 in t_δ, analog für die anderen Kanten.

Abbildung 4.6.: Ableitungsbaum zur Ableitung δ

4.3. Eigenschaften / Komplexitätsbetrachtungen

Das *Wortproblem* für eine Graphgrammatik ist die Frage, ob ein Graph zur erzeugten Sprache gehört. Ein Algorithmus, der zusätzlich eine Folge der Ersetzungsschritte bis auf Isomorphie konstruiert, heißt *Graph-Parser*.
Mit der Komplexität einer Graphgrammatik ist hier die Komplexität ihres Wortproblems gemeint.

Es gibt Graphgrammatiken, für die das Wortproblem PSPACE-vollständig ist. In dieser Arbeit werden nur Graphgrammatiken betrachtet, für die das Wortproblem in polynomialer Zeit gelöst werden kann. Dazu ist es nötig, dass die Graphgrammatiken einige Eigenschaften besiten, die im folgenden erläutert werden.

Definition Konfluenz Eine Graphgrammatik GG ist *konfluent*, genau dann, wenn für jeden ableitbaren Graphen X und Knoten u und w, u$\neq$w von V(X) und Produktionen p=(L_p, R_p, C_p), q=(L_q, R_q, C_q) von GG mit L_p={u}und L_q={w} gilt:

$$X \Rightarrow_p U_1 \Rightarrow_q Z_1 \quad \text{und} \quad X \Rightarrow_q U_2 \Rightarrow_p Z_2 \qquad \text{impliziert} \qquad Z_1 = Z_2.$$

Diese Eigenschaft wird in der Literatur häufig auch mit endliche Church Rosser Eigenschaft bezeichnet.

Eine Graphgrammatik ist von beschränktem Grad, genau dann, wenn es eine Konstante $c < \infty$ gibt, für die der Grad aller herleitbaren Graphen $\leq$ c ist.

Nach [Brandenburg 1988/1] gilt dann für alle Graphgrammatiken, die konfluent, zusammenhängend und von beschränktem Grad sind, dass die Lösung des Wortproblems in polynomialer Zeit möglich ist.

Eine Graphgrammatik GG heißt *polynomial*, wenn es einen Algorithmus mit polynomialer Laufzeit gibt, der für jeden Graphen G $\in$ L(GG) eine Ableitung konstruiert. Dies bedeutet, dass für eine polynomiale Graphgrammatik das Wortproblem in polynomialer Zeit gelöst werden kann.

5. Einführung in Layout-Graphgrammatiken

Ist ein Graph zu zeichnen, so muss entschieden werden, *wo* und *wie* die Knoten und Kanten gezeichnet werden sollen. Das 'wie' beschreibt das Aussehen der Knoten und Kanten, z.B. dass Knoten als Rechtecke und Kanten als verbindende Linien zwischen den Rechtecken dargestellt werden. 'Wo' beschreibt die Position der Knoten und Kanten auf dem verfügbaren Zeichenraum, z.B. dass der erste Knoten in der Mitte des Zeichenraums zu platzieren ist. Eine *Positionierung* ist eine solche Festlegung von Positionen.

Die *Position eines Knotens* v wird durch kartesische Koordinaten $(x, y) \in \mathbb{N}_0 \times \mathbb{N}_0$ angegeben. Dabei bezeichnet posX(v) die horizontale Entfernung zur y-Achse, bzw. posY(v) die vertikale Entfernung zur x-Achse.
Eine Positionierung eines Graphen wird als *gültig* bezeichnet, wenn allen Knoten des Graphen unterschiedliche Positionen zugewiesen wird, jede Kante keine anderen Kanten schneidet und jede Kante nur ihren Quell- und Zielknoten berührt.

Layout-Graphgrammatiken produzieren Graphen, die Positionsbedingungen beinhalten, die bei der Berechnung der Position der Knoten des Graphen zu berücksichtigen sind. Diese Bedingungen sind zusätzliche Kanten und zusätzliche Einbettungsregeln, die den Produktionen hinzugefügt werden. Die Menge der Kantenmarkierungen der Positionsbedingungen besteht aus vier Elementen: $\{x, y, v, h\} \notin \Delta$, wobei Δ die Menge der Kantenmarkierungen der zugehörigen Graphgrammatik ist.
Für eine Kante e von Knoten u nach Knoten w gilt folgende Positionsbedingung:

- Ist die Markierung der Kante e gleich x, dann ist u mindestens eine Position weiter links von w zu platzieren.
- Ist die Markierung der Kante e gleich y, dann ist u mindestens eine Position unterhalb von w zu platzieren.
- Ist die Markierung der Kante e gleich v bzw. h, dann ist die Kante ungerichtet[1] und hat die Bedingung, dass beide Knoten in gleicher vertikaler bzw. horizontaler Position zu platzieren sind.

[1] d.h. es existiert auch eine Kante von Knoten w nach Knoten u mit gleicher Markierung

Die Positionsbedingungen werden als Layout-Komponente einer Graphgrammatik bezeichnet.

Definition Layout-Graphgrammatik Eine *Layout-Graphgrammatik* LGG=(GG, LK) ist eine Graphgrammatik GG mit einer *Layout-Komponente* LK, wobei LK jeder Produktion p_i=(L_i, R_i, C_i) von GG $k(i)$ Mengen von endlichen, nicht leeren Mengen von Positionsbedingungen $c_{i,1}, \ldots, c_{i,k(i)}$ mit $k(i) \geq 1$ und $|c_{i,j}| \geq 1$ $\forall j \in \{1, \ldots, k(i)\}$ zuordnet.
Eine Menge von Positionsbedingungen besteht aus Kanten e=(u, l, w) und/oder Einbettungsregeln (l, d, u) mit u, w $\in$ V(R_i), l $\in$ {x, y, v, h} und d $\in$ {in, out}. Ein Tupel (p_i, $c_{i,j}$) steht für die Erweiterung der Produktion p_i um eine Menge von Positionsbedingungen $c_{i,j}$ und heißt *Layoutproduktion*.
Sind allen Produktionen p_i einer Ableitung δ=(G_0, p_0, G_1, p_1, ... ,G_n) jeweils eine Menge an Positionsbedingungen $c_{i,j}$ zugeordnet, so bezeichnet
(G_0, (p_0, $c_{0,j'}$), G_1, (p_1, $c_{1,j''}$), ..., G_n) eine *Layout-Ableitung*.

Eine Layoutproduktion ist formal wieder eine Produktion
(p_i, $c_{i,j}$) = (L_i, $R_i \cup \{(u, l, w) | (u, l, w) \in c_{i,j}\}$, $C_i \cup \{(l, d, u) | (l, d, u) \in c_{i,j}\}$). Die Anwendung einer Layoutproduktion verläuft analog zu einem Ableitungschritt in einer Graphgrammatik. Ebenso kann eine Layout-Graphgrammatik wieder als eine Graphgrammatik mit erweiterter Menge von Produtionen interpretiert werden.

5.1. Berechnung der Positionen

Eine Layout-Ableitung hat als Ergebnis einen Graphen, der Positionsbedingungen für die Knoten beinhaltet. Dabei kann es vorkommen, dass eine Positionierung, die alle Bedingungen einhält, nicht berechnet werden kann. Dies ist genau dann der Fall, wenn sich Positionsbedingungen wiedersprechen (vgl. Abbildung 5.1 1. und 2.).
Ist die Layout-Kompontente so gewählt, dass die Berechnung einer Positionierung bei jedem erzeugbaren Graphen von GG möglich ist, dann sind immer noch Positionierungen möglich, die ungültig sind (vgl. Abbildung 5.1 3.).

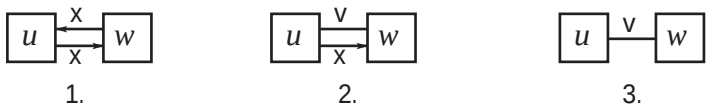

Abbildung 5.1.: Nicht berechenbare bzw. ungültige Positionierungen

1. Der Knoten u kann zwar links von Knoten w positioniert werden, jedoch kann nicht gleichzeitig der Knoten w links von Konten u platziert werden.

2. Beide Knoten sollen in gleicher vertikaler Position, und zugleich soll der Knoten u links von Knoten w platziert werden.

3. Schon die Initialisierung der Positionen der Knoten mit (0, 0) hält die Bedingung ein, dass der Knoten v in gleicher vertikaler Position wie der Knoten w zu platzieren ist. Diese Positionierung ist aber ungültig, da die beiden Knoten die gleiche Position zugewiesen bekommen haben.

Als Konsequenz aus den obigen Beispielen werden nun notwendige Bedingungen an die Layout-Komponente formuliert.

1. Die Mengen von Positionsbedingungen müssen so gewählt werden, dass in jedem erzeugten Graph der x-Graph[2] und y-Graph jeweils gerichtet und azyklisch sind.

2. Zwischen zwei Knoten müssen sich v- und x-Pfade[3] gegenseitig ausschliessen, d.h. wenn zwischen zwei Knoten ein v-Pfad existiert, darf kein x-Pfad zwischen den Knoten existieren und umgekehrt (analog für h- und y-Pfade).

3. Für jede v-Kante muss eine y-Kante zwischen den adjazenten Knoten existieren, damit den Knoten nicht die gleiche Position zugewiesen werden kann (analog für h- und x-Kanten).

Diese Bedingungen sind jedoch nicht hinreichend, da auch für die Graphen in Abbildung 5.2 keine Positionierung berechnet werden kann, bzw. die Positionierung ungültig ist, obwohl die oben genannten Bedingungen eingehalten werden.

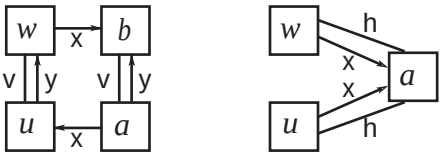

Abbildung 5.2.: Weitere nicht berechenbare bzw. ungültige Positionierungen

Fordert man zusätzlich zu den oben genannten Bedingungen, dass zwischen jeweils zwei beliebigen Knoten des Graphen mindestens ein x- oder y-Pfad existiert, so lässt sich zeigen, dass für jeden erzeugten Graphen einer LGG eine Positionierung berechenbar und ist den Knoten des Graphen paarweise verschiedene Positionen zugewiesen werden (Beweis siehe [Brandenburg 1994]). Über den Verlauf der Kanten des Graphen wird dabei nichts ausgesagt.

[2]Der x-Graph G_x eines Graphen G=(V, E, m) ist G_x=(V, E_x, m) mit E_x={(v, x, w)|(v, x, w)∈E}.
[3]Ein x-Pfad von Knoten v_1 zu Knoten w_n besteht aus gerichteten Kanten e_i=(v_i, x, w_i), i={1, . . . , n} für die gilt v_j=w_{j+1} für alle j={1, . . . , n-1}.

Algorithmus für Positionsberechnung Ein konkreter Algorithmus, der die Positionen der Knoten für alle möglichen Layout-Komponenten mit optimalem Zeitaufwand berechnet, kann nicht angegeben werden, jedoch werden immer die folgenden Schritte abgearbeitet:

1. Für alle Knoten v_i von G setzte $posX(v_i)=posY(v_i)=0$.
2. Korrigiere $posX(v_i)$ für alle $v_i \in V(G)$, sodass alle Positionsbedingungen von x- und v-Kanten eingehalten werden.
3. Korrigiere $posY(v_i)$ für alle $v_i \in V(G)$, sodass alle Positionsbedingungen von y- und h-Kanten eingehalten werden.
4. (optional, je nach LK) Überprüfe, ob Überschneidungen auftreten.

Die Komplexität des Algorithmus kann daher immer nur für eine konkrete Layout-Graphgrammatik angegeben werden.

5.2. Beispiel

Die Graphgrammatik GG der LGG=(GG, LK) sei wie in Beispiel 4.2 auf Seite 31, ebenso wird hier die gleiche Ableitung zu einer Layout-Ableitung erweitert.
$GG=(N, T \cup \Delta, P, S)$

- $N=\{A\}$ sei die Menge nichtterminaler Knotenmarkierungen
- $T=\{a\}$ die Menge teminaler Knotenmarkierungen
- $\Delta=\{e\}$ die Menge der Kantenmarkierungen
- $P=\{P_1, P_2\}$ ist die Menge von Produktionen (siehe Abbildung 4.3 auf Seite 31)
- $S=A \in N$ sei das Axiom der GG

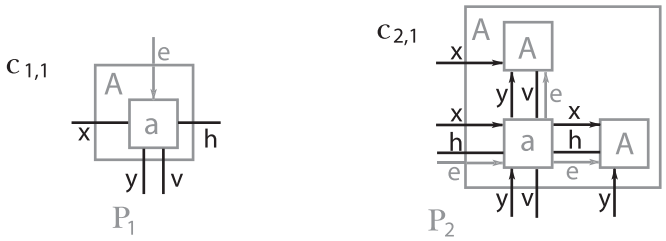

Abbildung 5.3.: Layout-Produktion (P_1, $c_{1,1}$) und (P_2, $c_{2,1}$)

Die Layout-Komponente LK sei wie folgt aufgebaut: Für jede Produktion p_i gibt es genau *eine* Menge an Positionsbedingungen $c_{i,1}$. Eine graphische Darstellung der Layoutproduktionen ist in Abbildung 5.3 zu sehen.

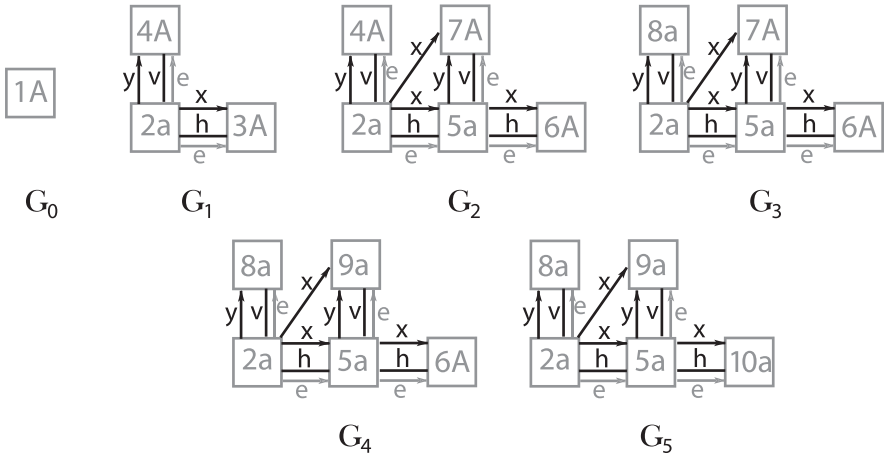

Abbildung 5.4.: Graphen G_0 bis G_5 der Layout-Ableitung

Die Layout-Ableitung der Ableitung δ=(G'$_0$, p_0, G'$_1$, p_1, G'$_2$, p_2, ..., G'$_5$) sei (G_0, (p_0, $c_{2,1}$), G_1, (p_1, $c_{2,1}$), G_2, (p_2, $c_{1,1}$), G_3, (p_3, $c_{1,1}$), G_4, (p_4, $c_{1,1}$), G_5), wobei die Graphen G_0, ..., G_5 der Layout-Ableitung in Abbildung 5.4 dargestellt sind.

Die Ableitung der Graphen wurde bereits in Kapitel 4.2 erläutert, in diesem Beispiel geht es darum, eine Positionierung der Knoten eines Graphen zu berechnen.

Die Berechnung wird an dem Graphen G_2 gezeigt; die Berechnung der Positionen in den anderen Graphen verläuft analog. Zur einfacheren Erklärung sind die Bedingungen für die x-Positionen getrennt von den Bedingungen für die y-Position in Abbildung 5.5 dargestellt.

Abbildung 5.5.: Bedingungen für x- und y-Position der Knoten des Graphen G_2

Hier sind nun die vier prinzipiellen Schritte des Algorithmus der Positionsberechnung ausgeführt:

1. Jedem Knoten des Graphen G_2 wird die Position (0, 0) zugewiesen

$$posX(v) = posY(v) = 0 \quad \forall v \in V(G).$$

2. Da eine x-Kante von Knoten 2a zu Knoten 5a existiert, muss der Knoten 2a mindestens eine Position weiter links als der Knoten 5a platziert werden. Dies wird erfüllt, indem der Knoten 5a eine Position weiter rechts als der Knoten 2a positioniert wird

$$\begin{aligned} posX(5a) &= posX(2a) + 1 \\ &= 0 + 1 \quad = 1. \end{aligned}$$

Die gleiche Beziehung gilt für die x-Kante von Knoten 2a nach Knoten 7A und für die x-Kante von Knoten 5a nach Knoten 6A. Der Knoten 7A (bzw. 6A) wird eine Position weiter rechts als der Knoten 2a (bzw. 5a) platziert

$$posX(7A) = posX(2a) + 1 = 0 + 1 = 1$$

$$posX(6A) = posX(5a) + 1 = 1 + 1 = 2.$$

Für die beiden v-Kanten gilt, dass die inzidenten Knoten die gleiche vertikale Position haben müssen. Dies ist bei beiden Kanten der Fall

$$posX(2a) == posX(4A) \quad \text{und} \quad posX(5a) == posX(7A).$$

Es werden nun alle Bedingungen von x- und v-Kanten eingehalten.

3. Die Berechnung der y-Positionen verläuft analog. Die y-Kante von Knoten 2a nach Knoten 4A steht für die Bedingung, dass der Knoten 2a unterhalb von Knoten 4A zu platzieren ist. Diese Bedingung wird erfüllt, indem der Knoten 4A eine Position weiter oberhalb von Knoten 2a positioniert wird.

$$\begin{aligned} poxY(4A) &= posY(2a) + 1 \\ &= 0 + 1 \quad = 1. \end{aligned}$$

Analog dazu wird die Position des Knoten 7A durch die y-Kante von Knoten 5a zu Knoten 7A bestimmt. Der Knoten 7A wird eine Position weiter oberhalb von Knoten 5a positioniert

$$posY(7A) = posY(5a) + 1 = 0 + 1 = 1.$$

Die Positionsbedingung einer h-Kante ist, dass die inzidenten Knoten die gleiche horizontale Position haben müssen. Die beiden vorhandenen h-Kanten haben die inzidenten Knoten 2a, 5a und 6A. Die Knoten liegen alle auf y-Position 0, womit die Bedingung erfüllt ist

$$posY(2a) == posY(5a) \quad \text{und} \quad posY(5a) == posY(6A).$$

Jetzt werden alle Bedingungen von x-, y-, v- und h-Kanten eingehalten.

4. In diesem Schritt findet die Überprüfung statt, ob Überschneidungen auftreten. Dazu wird zunächst kontrolliert, ob verschiedenen Knoten die gleiche Position zugewiesen wurde. Danach wird getestet, ob die Kanten des Graphen (hier die Kanten mit Markierung e), andere als die inzidenten Knoten berühren und ob sich Kanten untereinander berühren.

   Bei dieser Positionierung treten keine Überschneidungen auf und die Positionierung ist somit gültig.

Verändert man das Beispiel etwas, wird eine Schwäche des Verfahrens deutlich.

1. Änderung Die Layout-Ableitung sei nun als $(G_0, (p_0, c_{2,1}), G_1, (p_1, c_{2,1}), G_2, (p'_2, c_{2,1}), \hat{G}_3)$ gegeben. Auf den Graphen G_2 wird nun nicht die Layoutproduktion $(p_2, c_{1,1})$ angewendet, sondern die Layoutproduktion $(p'_2, c_{2,1})$, wobei p'_2 eine isomorphe Kopie der Produktion P_2 und die linke Seite der Produktion p'_2 der Knoten 4A ist. Der durch diese Änderung entstehende Graph $\hat{G}_3$ ist in Abbildung 5.6 dargestellt, für den eine ungültige Positionierung berechnet wird.

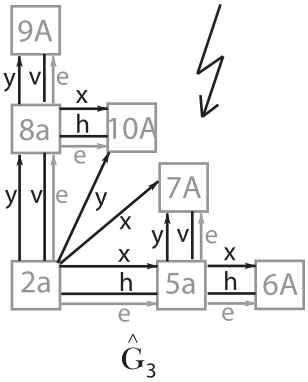

Abbildung 5.6.: Ungültige Positionierung

Der Positionierungsalgorithmus berechnet für die Knoten 7A und 10A

$$posX(7A) = posX(10A) = 1 \qquad \text{und} \qquad posY(7A) = posY(10A) = 1.$$

Die Knoten liegen also beide auf der Position (1, 1). Erst im Schritt 4 wird die Überschneidung erkannt und der Algorithmus bricht ab. Allgemeiner: es ist nicht möglich, mit diesem Algorithmus für diese Layout-Graphgrammatik eine Positionierung für einen Binärbaum der Tiefe 2 zu berechnen (Die Wurzel des Baumes in Abbildung 5.6 ist der Knoten 2a).

Durch eine weitere Veränderung wird dieses Verhalten verbessert.

2. Änderung Die Menge von Positionsbedingungen $c_{2,1}$ der Produktion P_2 wird nun um zwei Kanten und zwei Eingbettungsregeln erweitert. Diese neue Menge $c_{2,2}$ ersetzt in der Layout-Ableitung alle Vorkommen von $c_{2,1}$ und ist in Abbildung 5.7 dargestellt.

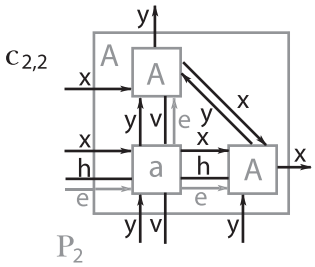

Abbildung 5.7.: Neue Menge von Positionsbedingungen $c_{2,2}$

Die Layout-Ableitung wird zu $(G_0, (p_0, c_{2,2}), G'_1, (p_1, c_{2,2}), G'_2, (p'_2, c_{2,2}), G'_3)$, dessen Graphen in Abbildung 5.8 zu sehen sind.

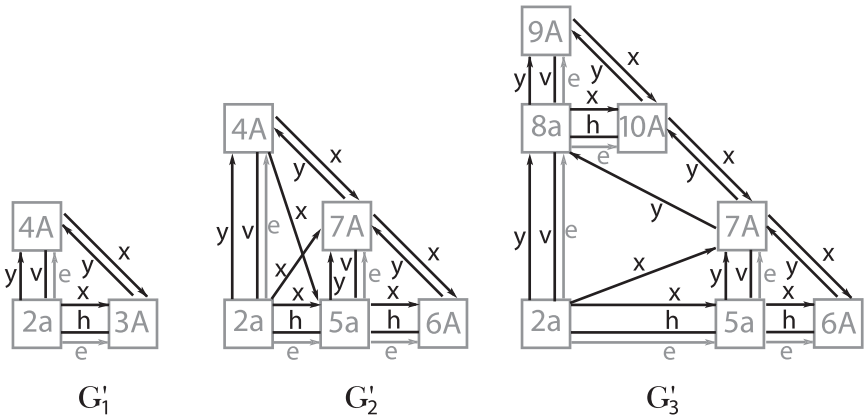

Abbildung 5.8.: Die entstehenden Graphen G'_1 bis G'_3

Benutzt man nur die Positionsbedingungen $c_{2,2}$ für isomorphe Kopien der Produktion P_2 in einer Ableitung, so besitzt jeder Graph der Ableitung eine gültige Positionierung. Mit dieser Layout-Graphgrammatik ist man in der Lage, beliebige binäre Bäume zu positionieren.
Der Nachteil bei dieser Vorgehensweise ist, dass der Platzbedarf nicht optimal ist, wie man an der benutzten Fläche der Graphen G'_2 und G'_3 sehen kann:

Der Knoten 4A des Graphen G'_2 könnte eine Position weiter unten stehen, womit die Positionsbedingung der y-Kante von Knoten 7A zu Knoten 4A nicht erfüllt wäre.

Ebenso könnten die Knoten 5a, 6A und 7A des Graphen G'_3 eine Position weiter links positioniert werden, allerdings würde dadurch die Positionsbedingung der x-Kante von Knoten 10A zu Knoten 7A verletzt.

Trotzdem würde in beiden Fällen eine gültige Positionierung der Graphen mit geringerer benutzter Fläche erreicht.

5.3. Layout-Suche

Beim Entwurf der Layout-Komponente für eine Graphgrammatik ist darauf zu achten, dass die Mengen von Positionsbedingungen pro Produktion der GG so angeben werden, dass es eine Möglichkeit gibt, eine gültige Positionierung zu berechnen. Für jede Ableitung muss eine Layout-Ableitung existieren, mit deren Ergebnis es möglich ist, eine gültige Positionierung zu berechnen.

Gibt es für eine Ableitung mehrere gültige Positionierungen der Ergebnisse der zugehörigen Layout-Ableitungen, so wird jeder dieser Graphen mit Hilfe einer Kostenfunktion[4] bewertet und eine Positionierung mit einer Suche ausgewählt.
Eine Suche besteht aus (vgl. [Vorlesung Suche])

1. **Datenbasis** Eine Codierung (Zustände), die zur Repräsentation der Objekte im Kandidatenraum S dient.

2. **Operatoren** Einen Mechanismus, um ein Objekt aus S in ein anderes zu transformieren.

3. **Steuerungsstrategie** Ein effektives Verfahren, um die Reihenfolge möglicher Transformationen festzulegen.

   Ziel: Das gesuchte Objekt so schnell wie möglich finden.

Ein Zustand ist in diesem Fall ein Graph, der von der LGG erzeugt wurde und die Operatoren sind die Layoutproduktionen der LGG. Sie überführen einen Graphen G in einen Graphen G′.

Top-Down-Optimierung: Gegeben ist eine *terminale Ableitung*. Man sucht nach einer Layout-Ableitung, sodass die daraus resultierende Positionierung kostenoptimal ist.
Bottom-Up-Optimierung: Gegeben ist ein *Graph* in der Sprache der Graphgrammatik. Hier muss zunächst die Menge aller terminalen Ableitungen gefunden werden, die auf den Graphen führen. Man sucht nach einer Layout-Ableitung aus der Menge aller Ableitungen, die eine kostenoptimale Positionierung liefert.

[4] Verschiedene Kostenfunktionen werden in Kapitel 6 betrachtet.

Die Layout-Graphgrammatik bestimmt in beiden Optimierungen die Zustände und Operatoren der Layoutsuche.

Die Suche in der Bottom-Up-Optimierung ist umfangreicher als in der Top-Down-Optimierung; in ersterer ist bereits die Ableitung gegeben, während man bei der Top-Down-Optimierung erst die Menge aller Ableitungen bestimmen muss.

6. Layout-Graphgrammatik für WDS

In diesem Kapitel wird eine Layout-Graphgrammatik vorgestellt, deren Graphgrammatik die Klasse der SP-Bäume erzeugen kann. Ein SP-Baum ist ein gerichteter Baum, dessen innere Knoten die Markierung S oder P besitzen und genau drei Nachbarn[1] haben. Der Aufbau wurde bereits in Kapitel 3.1 vorgestellt. Die inneren Knoten des Baumes werden hier nun vereinfacht als Knoten mit Markierung a (für Adaptor) und die Blätter als Knoten mit Markierung b (für Bauteil) interpretiert. Die Kantenmarkierungen im SP-Baum seien alle gleich e und werden der Übersichtlichkeit halber an vielen Stellen weggelassen.

Zunächst wird eine Graphgrammatik GG vorgestellt, die die Klasse der SP-Bäume erzeugen kann. Anschliessend erweitert man die Graphgrammatik um eine Layout-Komponente LK, die jeder Produktion p_i der Graphgrammatik Mengen von Positionsbedingungen $c_{i,k}$ zuordnet.

Was dann noch fehlt, ist die Information, welche Produktionen in welcher Reihenfolge anzuwenden sind, damit der gegebene Graph entsteht. Dafür wird ein Graph-Parser angegeben, der für den jeweils gegebenen Graphen und an Hand der Graphgrammatik die Ableitungstabelle[2] generiert.
Der hier vorgestellte Graph-Parser ist eine spezielle Art von Wort-Parsern, und ist auf SP-Bäume angepasst.

Dann kann die eigentliche Layoutsuche beginnen. Aus der Ableitungstabelle entnimmt man die möglichen Produktionen, die mit deren jeweiligen Positionsbedingungen zu Layoutproduktionen erweitert werden. Die durch Anwenden von Layoutproduktionen entstandenen Graphen werden mit einer Kostenfunktion bewertet, und mit dem besten[3] Graphen wird weitergesucht. Welcher Graph dann das gefundene Ziel ist, hängt von der Art der Suche und der Kostenfunktion ab.

[1] Die Summe aus Eingangs- und Ausgangsgrad eines Knotens ist gleich drei.
[2] Hierin werden alle gefundenen Ableitungen gespeichert.
[3] Welcher Graph der beste ist, hängt von der Kostenfunktion ab.

Erster Versuch für eine Layout-Graphgrammatik
Die Graphgrammatik GG_{Try}=(N, T ∪ Δ, P, S) ist wie folgt aufgebaut

- N = {A, S} sei die Menge nichtterminaler Knotenmarkierungen
- T = {a, b} sei die Menge terminaler Knotenmarkierungen
- Δ = {e} sei die Menge der Kantenmarkierungen
- P = {P_0, P_1, P_2} sei die Menge der Produktionen (siehe Abbildung 6.1)
- S = S ∈ N sei das Axiom der GG_{Try}

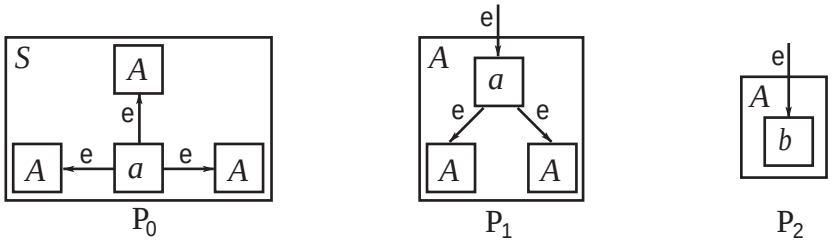

Abbildung 6.1.: Produktionen von GG_{Try}

Die erste Produktion einer Ableitung ist immer die Produktion P_0, da nur diese Produktion das Axiom S der GG_{Try} als Markierung des Knotens der linken Seite besitzt. Sonst kommt diese Markierung nicht wieder vor, das bedeutet, dass die Produktion nur einmal in einer Ableitung vorkommen kann. Die Produktion P_1 erzeugt die eigentliche Baum-Struktur, in der alle inneren Knoten drei Nachbarn haben und die Produktion P_2 ersetzt die noch nichtterminal markierten Knoten durch Knoten mit Markierung b. Die Sprache der GG_{Try} besteht offensichtlich aus SP-Bäumen.

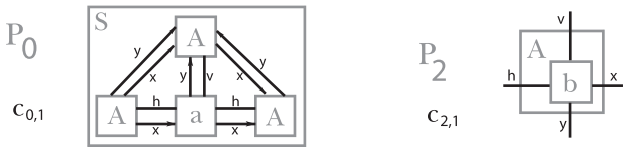

Abbildung 6.2.: Layoutproduktionen (P_0, $c_{0,1}$) und (P_2, $c_{2,1}$)

Die Layout-Komponente sei wie folgt aufgebaut: Den Produktionen P_0 und P_2 wird jeweils genau eine Menge an Positionsbedingungen zugewiesen: $c_{0,1}$ bzw. $c_{2,1}$ (siehe Abbildung 6.2), und die Produktion P_1 erhält sechs Mengen von Positionsbedingungen $c_{1,1}$ bis $c_{1,6}$ (dargestellt in Abbildung 6.3).

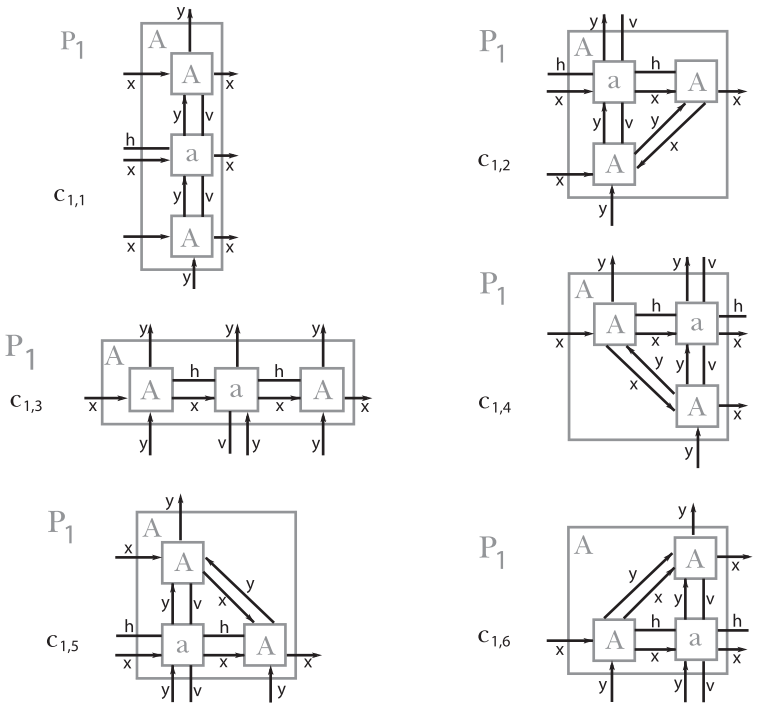

Abbildung 6.3.: Layoutproduktionen (P_1, $c_{1,1}$) bis (P_1, $c_{1,6}$)

Für einen beliebigen SP-Baum gibt es genau einen möglichen Ableitungsbaum[4]. Dies hat zur Folge, dass die Bottom-UP-Optimierung in diesem Fall zu einer Top-Down-Optimierung würde, was ein großer Vorteil der GG_{Try} wäre.

Es wird sich allerdings im folgenden ein Nachteil dieser LGG zeigen, der durch eine Erweiterung (nächstes Kapitel) zu beheben ist.

Beispiel Es sei der folgende SP-Baum gegeben (siehe Abbildung 6.4), für den ein Layout gefunden werden soll.

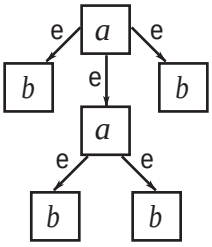

Abbildung 6.4.: Gegebener Graph

[4]Diese Eigenschaft wird in dem nachfolgenden Beispiel erklärt.

Um diesen Graphen mit einer Ableitung der GG_{Try} herzuleiten, ist es notwending, nach Anwenden der Produktion P_0 die Produktion P_1 auf einen nichtterminal markierten Knoten anzuwenden. Die dann existierenden nichtterminal markierten Knoten müssen mit (isomorphen Kopien) der Produktion P_2 ersetzt werden. Für diese Ableitungen gibt es genau einen Ableitungsbaum (siehe Abbildung 6.5), und genau da liegt der Nachteil in diesem Vorgehen: für die Graphgrammatik die drei Knoten der Produktion P_0 alle die Markierung A. Es ist egal, welchen Knoten die Produktion P_1 ersetzt. Für die Layout-Graphgrammatik gilt dies nicht. Die Knoten haben verschiedene Positionsbedingungen, sie sind also nicht gleichwertig. Daher müssen alle drei verschiedene Möglichkeiten in der Layout-Graphgrammatik betrachtet werden (drei verschiedene Knoten und jeweils sechs verschiedene Mengen von Positionsbedingungen).

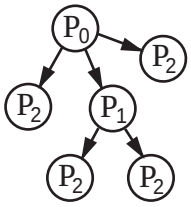

Abbildung 6.5.: Ableitungsbaum zum Graphen in Abbildung 6.4

Ein weiterer Nachteil ist, dass einige der Mengen von Positionsbedingungen der Produktion P_1 bei Anwenden in einer Layout-Ableitung zu einer ungültigen Positionierung führt. Als ein solches Beispiel ist die Ableitung (G_0, (p_0, $c_{0,1}$), G_1, (p, $c_{1,1}$), G_2), die in Abbildung 6.6 dargestellt ist. Die Positionierung ist ungültig, weil die e-Kante zwischen Knoten 1a und Knoten 5a durch den Knoten 7A verläuft.

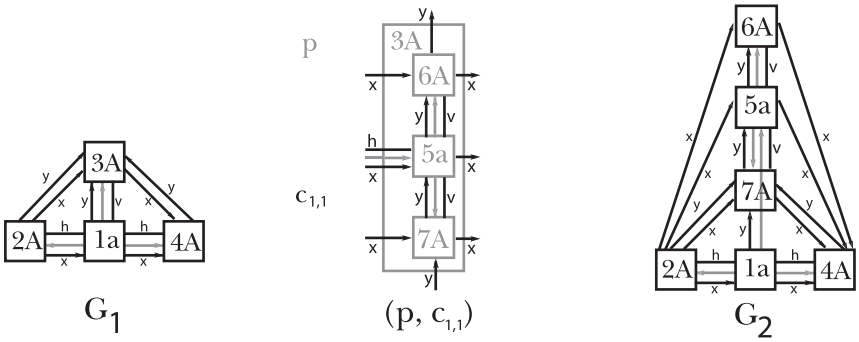

Abbildung 6.6.: Layout-Ableitung, die zu einer ungültigen Positionierung führt

Ebenso würden Layout-Ableitungen, die anstatt $c_{1,1}$ die Menge $c_{1,2}$ oder $c_{1,4}$ benuzten, zu ungültigen Positionierungen führen (vgl. Abbildung 6.7 oben). Lediglich die unteren drei Graphen in Abb. 6.7 ergeben eine gültige Positionierung.

Der Knoten 3A von G_1 liegt oberhalb seines adjazenten Knoten 1a. Genau die Mengen von Positionsbedingungen, die eine Erweiterung des Knotens nach unten haben, führen zu ungültigen Positionierungen.

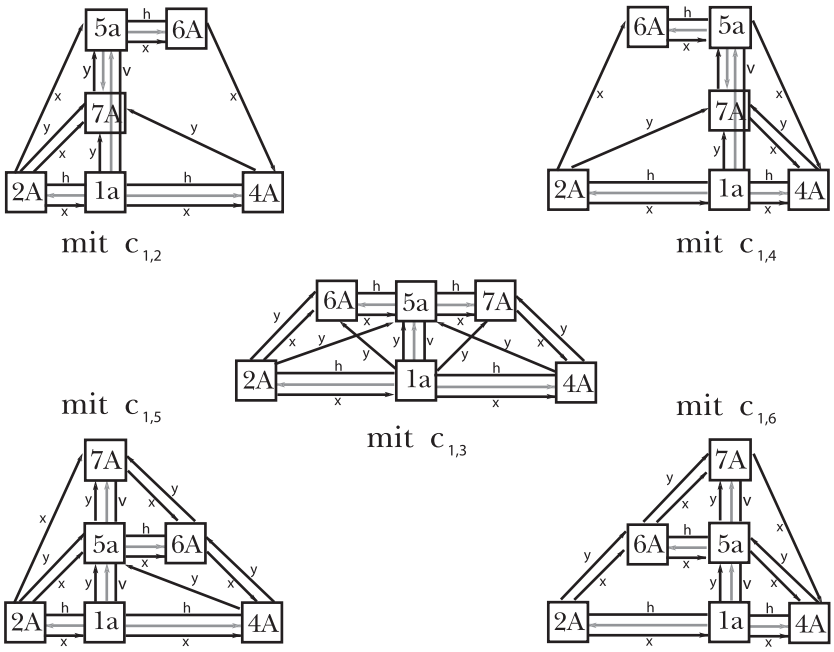

Abbildung 6.7.: Verbleibende Positionierungen

Die Graphgrammatik muss also um die folgenden Punkte erweitert werden.

1. Die nichtterminal markierten Knoten der rechten Seite der Produktionen müssen durch ihre Markierung unterschieden werden.

2. Die Richtung der einlaufenden e-Kante eines Knotens muss in dem Knoten abgespeichert werden.

6.1. Graphgrammatik für WDS

Die Graphgrammatik GG_{WDS}=(N, T ∪ Δ, P, S) ist wie folgt aufgebaut

- N = {L, O, R, U, S} sei die Menge nichtterminaler Knotenmarkierungen
- T = {a, b} die Menge terminaler Knotenmarkierungen
- Δ = {e} die Menge terminaler Kantenmarkierungen
- P = {P_0, ..., P_{16}} ist die Menge der Produktionen (siehe Abbildungen 6.8 und 6.9)
- S = S ∈ N ist das Axiom der GG_{WDS}

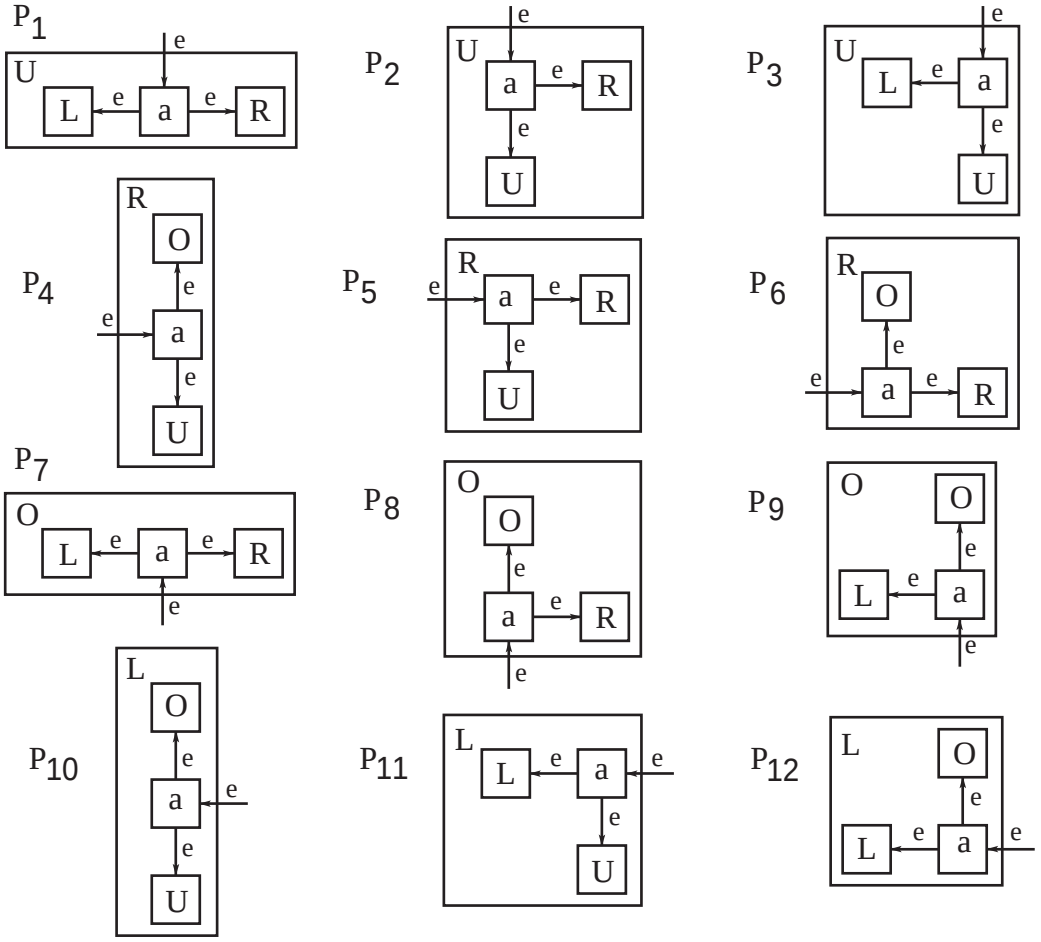

Abbildung 6.8.: Teil 1 der Produktionen von GG_{WDS}

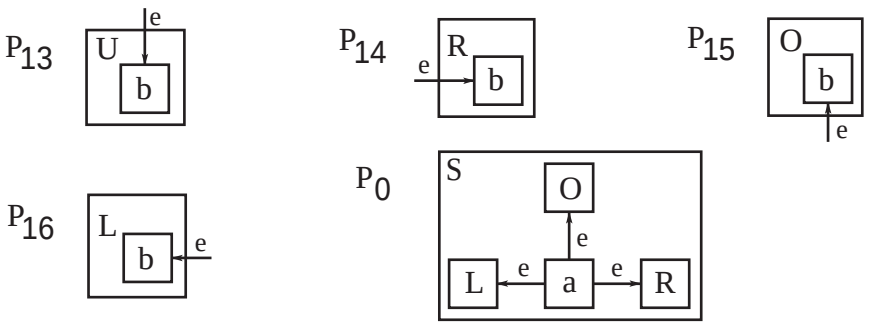

Abbildung 6.9.: Teil 2 der Produktionen von GG_{WDS}

Dabei steht die Markierung eines Knotens für die Richtung der eingehenden Kante in der späteren Positionierung:
U steht für *u*nten, O für *o*ben, L für *l*inks und R für *r*echts.

Die Produktionen P_1 und P_7 unterscheiden sich nur durch die Markierung des Knotens der linken Seite[5]. Das gleiche gilt für die Produktionen P_2 und P_5, P_3 und P_{11}, P_4 und P_{10}, P_6 und P_8 bzw. P_9 und P_{12}.

Mit jedem Ableitungsschritt wird dem Graphen genau ein terminaler Knoten hinzugefügt. Hat ein gegebener SP-Baum n Knoten, so hat eine initiale Ableitung, die den SP-Baum herleitet, genau n Ableitungsschritte.

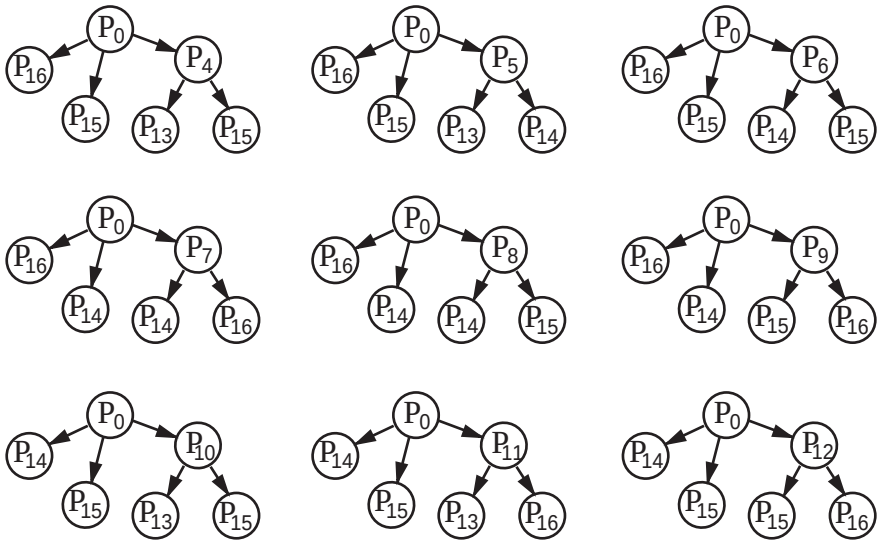

Abbildung 6.10.: Ableitungsbäume zu Graph von Seite 47.

[5]Produktion P_1 ersetzt einen Knoten mit Markierung U und P_7 einen Knoten mit Markierung O

Layout-Komponente Die Layout-Komponente LK_{WDS} ordnet den Produktionen P_0 und P_{13}, ..., P_{16} wird jeweils nur eine Menge an Positionsbedingungen $c_{j,1}$ zu, und den Produktionen P_1 bis P_{12} werden zwei Mengen an Positionsbedingungen zugewiesen $c_{i,1}$ bzw. $c_{i,2}$.

Angegeben sind die Layoutproduktionen (P_1, $c_{1,1}$) und (P_1, $c_{1,2}$) in Abbildung 6.12, die Layoutproduktionen (P_2, $c_{2,1}$) und (P_2, $c_{2,2}$) in Abbildung 6.13, (P_3, $c_{3,1}$) und (P_3, $c_{3,2}$) in Abbildung 6.14, (P_4, $c_{4,1}$) und (P_4, $c_{4,2}$) in Abbildung 6.15, (P_6, $c_{6,1}$) und (P_6, $c_{6,2}$) in Abbildung 6.16, (P_9, $c_{9,1}$) und (P_9, $c_{9,2}$) in Abbildung 6.17 und die Layoutproduktionen (P_0, $c_{0,1}$) und (P_{13}, $c_{13,1}$) bis (P_{16}, $c_{16,1}$) in Abbildung 6.11. Die verbleibenden Mengen von Positionsbedingungen $c_{5,i}$, $c_{7,i}$, $c_{8,i}$, $c_{10,i}$, $c_{11,i}$ und $c_{12,i}$ sind analog zu $c_{2,i}$, $c_{1,i}$, $c_{6,i}$, $c_{4,i}$, $c_{3,i}$ und $c_{9,i}$ aufgebaut.

Die Kanten der Produktionen der Graphgrammatik wurden wegen der Übersichtlichkeit weggelassen.

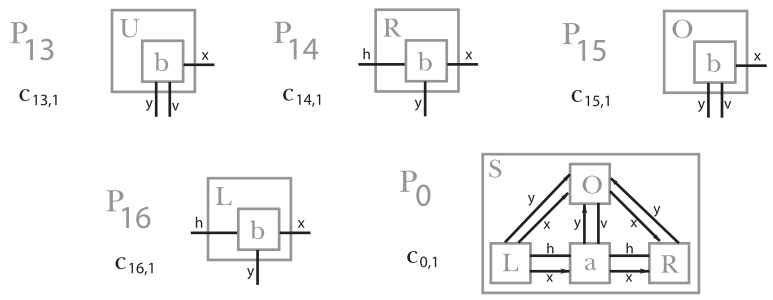

Abbildung 6.11.: Layoutproduktionen (P_0, $c_{0,1}$) und (P_{13}, $c_{13,1}$) bis (P_{16}, $c_{16,1}$)

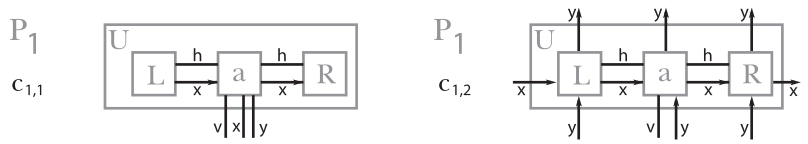

Abbildung 6.12.: Layoutproduktionen (P_1, $c_{1,1}$) und (P_1, $c_{1,2}$)

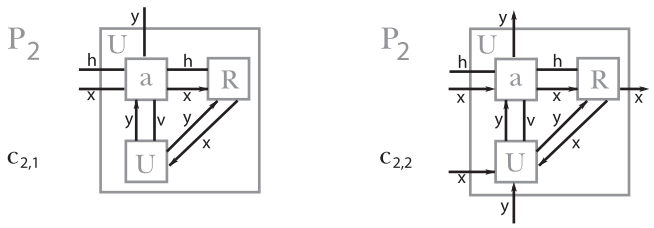

Abbildung 6.13.: Layoutproduktionen (P_2, $c_{2,1}$) und (P_2, $c_{2,2}$)

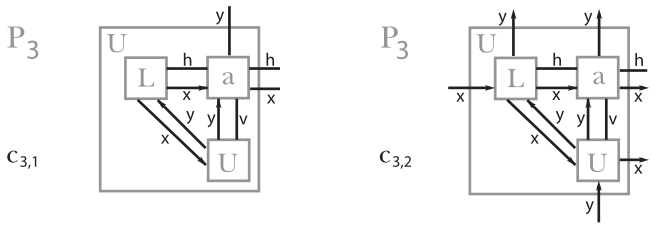

Abbildung 6.14.: Layoutproduktionen (P_3, $c_{3,1}$) und (P_3, $c_{3,2}$)

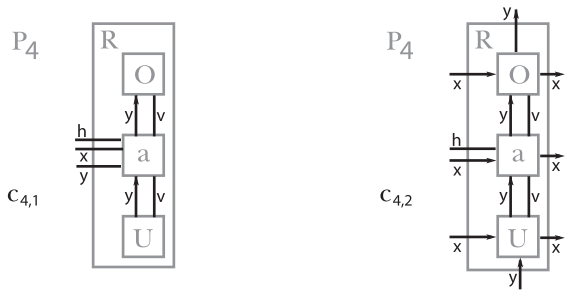

Abbildung 6.15.: Layoutproduktionen (P_4, $c_{4,1}$) und (P_4, $c_{4,2}$)

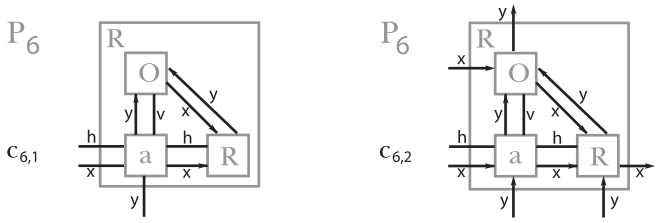

Abbildung 6.16.: Layoutproduktionen (P_6, $c_{6,1}$) und (P_6, $c_{6,2}$)

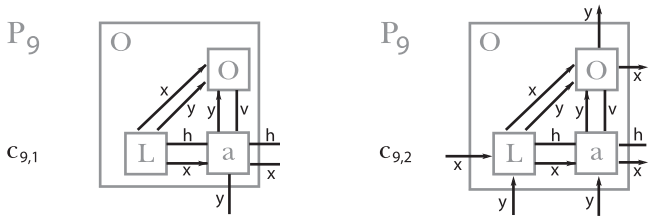

Abbildung 6.17.: Layoutproduktionen (P_9, $c_{9,1}$) und (P_9, $c_{9,2}$)

Die Layout-Graphgrammatik für die Wellendigitalstruktur ergibt sich dann zu LGG_{WDS}=(GG_{WDS}, LK_{WDS}).

Die Bestimmung der Positionen der Knoten eines Graphen, der von der LGG_{WDS} erzeugt wurde, geschieht nun wie folgt.

Algorithmus computePositions()
input: Graph G, generiert von LGG_{WDS}
output: posX und posY für jeden Knoten von G

1. Setze bei allen Knoten von G posX und posY auf 0.
2. Speichere Topologische Sortierung über x-Kanten in einer Liste.
3. Berechne für alle Knoten v_i aus dieser Liste:
   für alle eingehenden x-Kanten $e_{i,j}$ von v_i:
   posX(v_i) = max(posX(v_i, posX(Quellknoten($e_{i,j}$))+1)
4. Überprüfe für alle v-Kanten e_i, ob die inzidenten Knoten gleiche posX besitzen, falls nicht:
   a) Korrigiere posX = max(posX(Quellknoten(e_i)), posX(Zielknoten(e_i))).
   b) Weiter mit Schritt 3.
5. Überprüfe ob es noch mindestens einen Knoten mit posX=0 gibt.
   falls nicht: Beende Algorithmus mit „Positionierung ist nicht gültig.“
6. Überprüfe für jeden Knoten v_i, der mindestens eine ausgehende x-Kante hat, ob für mindestens eine dieser ausgehenden x-Kanten $e_{i,j}$ gilt:
   posX(Zielknoten($e_{i,j}$))-posX(Quellknoten($e_{i,j}$)) = 1,
   falls nicht:
   a) Korrigiere posX(v_i) sodass Bedingung erfüllt ist.
   b) Weiter mit Schritt 3.
7. wiederhole Schritte 3 bis 6 für posY mit y- und h-Kanten.
8. Überprüfe,
   a) ob sich Knoten überschneiden,
   b) ob sich Knoten mit Kanten schneiden,
   c) ob sich Kanten schneiden,

   falls eine Bedingung wahr ist: Beende Algorithmus mit „Positionierung ist nicht gültig.“

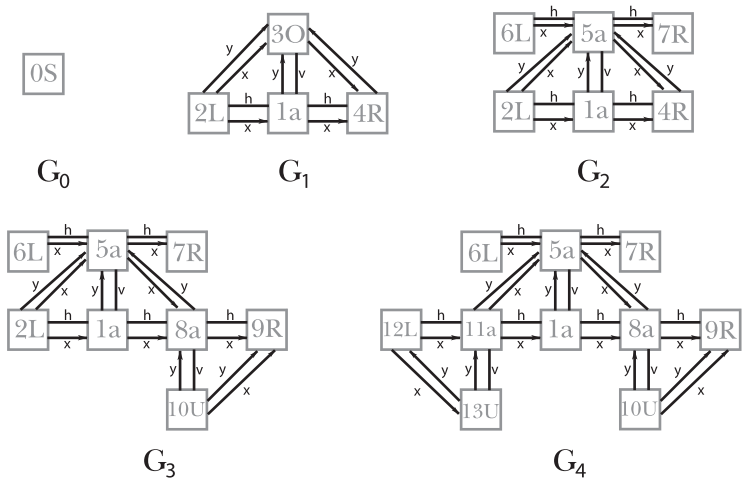

Abbildung 6.18.: Graphen G_0 bis G_4 der Layout-Ableitung δ

Beispiel zur Positionsberechnung eines Graphen Gegeben sei die Layout-Ableitung $\delta = (G_0, (P_0, c_{0,1}), G_1, (P_7, c_{7,1}), G_2, (P_{11}, c_{11,1}), G_3, (P_5, c_{5,1}), G_4)$, wobei die Graphen G_0 bis G_4 in Abbildung 6.18 dargestellt sind.
Betrachtet wird nun der Graphen G_4, um den Algorithmus computePositions() einmal Schritt für Schritt zu durchlaufen.

1. Setze bei allen Knoten von G_4 posX und posY auf 0.

2. Topologische Sortierung über x-Kanten
   z.B.: {12L, 11a, 6L, 10U, 5a, 7R, 1a, 13U, 8a, 9R}

3. Der Knoten 12L hat keine eingehenden x-Kanten, behält also die Position posX(12L)=0. Der Knoten 11a hat eine eingehende x-Kante, also wird ihm zugewiesen
$$\begin{aligned} posX(11a) &= max(posX(11a, posX(12L) + 1) \\ &= max(0, \quad 0 + 1) \qquad = 1. \end{aligned}$$
   Die Knoten 6L und 10U besitzen wie Knoten 12L keine eingehenden x-Kanten
$$posX(6L) = 0 \qquad posX(10U) = 0.$$
   Der Knoten 5a besitzt hingegen 2 eingehende x-Kanten, für die
$$posX(5a) = max(posX(5a, posX(11a) + 1) \quad \text{und}$$
$$posX(5a) = max(posX(5a, posX(6L) + 1) \qquad = 2$$
   berechnet wird. Für die restlichen Konten ergibt sich $posX(7R) = 3$, $posX(1a) = 2$, $posX(13U) = 1$, $posX(8a) = 3$ und $posX(9R) = 4$.

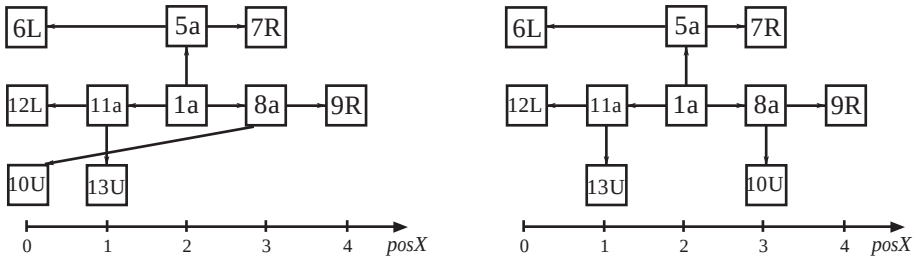

Abbildung 6.19.: Temporäre Positionierungen des Graphen G_4

Die momentane x-Positionierung der Knoten ist in Abbildung 6.19 links dargestellt. Die y-Position der Knoten ist eigentlich zur Zeit überall gleich null, zur besseren Übersicht sind die y-Positionen jedoch dem tatsächlichen Ergebnis der Positionierung entnommen.

4. Es gibt drei v-Kanten in G_4: {(13U, v, 11a), (5a, v, 1a), (10U, v, 8a)}.
   Für die ersten beiden Kanten gilt, dass die inzidenten Knoten die gleiche x-Position haben (vgl. Abbildung 6.19 links)

   $$posX(13U) == posX(11a) \qquad \text{und} \qquad posX(5a) == posX(1a).$$

   Die dritte v-Kante erfüllt diese Bedingung (noch) nicht, daher wird angepasst

   $$posX(10U) = max(posX(10U), posX(8a))$$

   $$posX(8a) = max(posX(10U), posX(8a)).$$

   Dieser Anpassungsschritt sollte sofort für alle Knoten, die mit den inzidenten Knoten über v-Kanten verbundenen sind, durchgeführt werden. Die jetzige Positionierung ist in Abbildung 6.19 rechts dargestellt.

   Da eine Änderung vorgenommen wurde muss zu Schritt 3 zurück gesprungen werden. Beim erneuten durchlaufen dieses Schrittes ergeben sich allerdings keine weiteren Änderungen an den x-Positionen.
   Der Schritt 4 wird dann ebenfalls nochmal durchlaufen, wobei jetzt alle Bedingungen der v-Kanten eingehalten werden, und somit zum nächsten Schritt gewechselt wird.

5. Durch die Überprüfung ob es noch mindestens einen Knoten mit posX=0 gibt, wird sichergestellt, dass der nächste Schritt nicht eine Endlosschleife auslöst. Die Knoten 6L und 12L haben beide die x-Position posX=0.

6. Die Positionierung in Abbildung 6.19 rechts erfüllt alle Positionsbedingungen von x- und v-Kanten. Für eine kompaktere Positionierung (wie in Ab-

bildung 6.18 G_4) muss wird für jeden Knoten überprüft, ob es eine ausgehende x-Kante gibt, die eine Länge[6] von eins hat. Ist der Wert größer, kann der Quellknoten näher zum Zielknoten platziert werden, ohne dass eine x-Positionsbedingung[7] verletzt wird. Für einen solchen Knoten wird die x-Position aus dem Maximum[8] über alle x-Positionen von Zielknoten ausgehender x-Kanten bestimmt. Knoten ohne ausgehende x-Kanten werden dabei nicht betrachtet.

In diesem Beispiel erfüllen alle Knoten ausser dem Knoten 6L diese Bedingung. Der Knoten 6L hat nur eine ausgehende x-Kante, ihm wird also zugewiesen:

$$posX(6L) = posX(5a) - 1 = 1.$$

Durch diese Änderung muss zu Schritt 3 zurück gesprungen werden, um eine erneute Überprüfung der Positionsbedingungen von x- und v-Kante durchzuführen. Dabei werden in diesem Beipiel keine weiteren x-Positionen verändert.

7. Die Berechnung der y-Positionen verläuft analog zur Berechnung der x-Positionen. Die Schritte 3 bis 6 sind daher nochmals für posY mit y- und h-Kanten durchzuführen.
   Die Positionierung der Knoten ist in Abbildung 6.19 G_4 dargestellt.

8. Im diesem letzten Schritt des Algorithmus wird getestet, ob die gerade berechnete Positionierung gültig ist. Das alle Positiosbedingungen eingehalten werden, wurde bereits in den Schritten zuvor sichergestellt. Hier wird daher nur getestet, ob verschiedenen Knoten die gleiche Position zugewiesen wurde, ob Kanten weitere Knoten ausser ihren inzidenten Knoten berühren und ob sich Kanten schneiden.
   Solche Überschneidungen treten in diesem Beispiel nicht auf, es ist daher eine gültige Positionierung berechnet worden.

[6] Die Länge bezeichnet hier den Abstand zwischen der Position von Quell- und Zielknoten der x-Kante.
[7] Positionsbedingungen von v-Kanten können dabei allerdings verletzt werden.
[8] Maximum minus eins $posX = max(\ldots) - 1$.

Die nun folgenden Definitionen haben zum Ziel, einen Graph-Parser anzugeben, der die Ableitungstabelle erzeugt. Dazu zunächst die Darstellung eines Baumes als ein String.
Die Idee dabei ist, den Baum in Pre-Order zu durchlaufen und dabei die Markierung der Knoten in einem String zu speichern. Damit die Reihenfolge der Nachfolger eindeutig ist, wird der größere Unterbaum zuerst in die String-Repräsentation aufgenommen.

Definition String-Repräsentation Sei G=(V, E, m) ein Baum, erzeugt von der GG_{WDS}, mit Wurzelknoten r und es sei v∈V. Ist v ein Blatt, so ist die String-Repräsentation str(v)=m(v), wobei m(v) die Knotenmarkierungsfunktion ist.
Sind v_1 bis v_k Nachfolger von v, mit lex(str(v_i), str(v_j))≤ 0, 1≤i<j≤k, so ist str(v)=m(v) + '(' + str(v_1) + ...+ str(v_k) + ')'. Dabei ist der +Operator die Verkettung der Strings und lex(): (String , String) → $I\!N$ ist ein Vergleich zweier Strings, bei dem folgendes gilt:

1. lex(x, x)=0 für alle Strings x,
2. lex(x, y)= -lex(y, x) für alle Strings x, y,
3. lex(a, b)<0, lex(b, L)<0, lex(L, O)<0, lex(O, R)<0, lex(R, U)<0,
4. lex(x, '(')<0 und lex('(', ')')<0.

Die String-Repräsentation des Baumes G entspricht der String-Repräsention der Wurzel des Baumes, es wird im folgenden abkürzend str(G) anstatt str(Wurzelkonten(G)) benutzt.
Eine String-Repräsentation steht für die Menge von Bäumen, die diese String-Repräsentation haben.

Beispiel Die Graphen aus Abbildung 6.18 auf Seite 55 haben die String-Repräsentation str(G_0)=S, str(G_1)=a(LOR), str(G_2)=a(a(LR)LR), str(G_3)=a(a(LR)a(RU)L) und str(G_4)=a(a(LR)a(LU)a(RU)).

Auch die Produktionen p_i=(L_i, R_i, C_i) der Graphgrammatik GG_{WDS} aus Abbildung 6.8 und 6.9 lassen sich so vereinfacht durch einen String darstellen. Die Einbettungsregeln C_i haben dabei keine Verwendung.

```
 0. S -> a(LOR)
 1. U -> a(LR)          2. U -> a(RU)          3. U -> a(LU)
 4. R -> a(OU)          5. R -> a(RU)          6. R -> a(OR)
 7. O -> a(LR)          8. O -> a(OR)          9. O -> a(LO)
10. L -> a(OU)         11. L -> a(LU)         12. L -> a(LO)
13. U -> b      14. R -> b      15. O -> b      16. L -> b
```

Anhand der String-Repräsentation eines Graphen kann mit Hilfe einer Indexfunktion auf die Knoten des Graphen zugegriffen werden.

Definition Indexfunktion auf String-Repräsentation Sei G ein SP-Baum, erzeugt von der GG_{WDS}, mit Wurzelknoten r, str=str(r)=str(G).
Dann liefert nodeInStringAt(str, i) den Knoten, dessen Markierung in der String-Repräsentation an der i-ten Stelle ist. Befindet sich an der betreffenden Stelle eine Klammer, wird eine null zurückgeliefert.

Beispiel Gegeben sei der SP-Baum G_2 aus Abbildung 6.18 auf Seite 55. Die String-Repräsentation des Graphen wurde schon im vorherigen Beispiel mit str=str(G_2)=a(a(LR)LR) angegeben. Dann liefert
nodeInStringAt(str, 0) den Knoten 1a, nodeInStringAt(str, 1) null,
nodeInStringAt(str, 2) den Knoten 5a, nodeInStringAt(str, 3) null,
nodeInStringAt(str, 4) den Knoten 6L, nodeInStringAt(str, 5) den Knoten 7R,
nodeInStringAt(str, 6) null, nodeInStringAt(str, 7) den Knoten 2L,
nodeInStringAt(str, 8) den Knoten 4R, nodeInStringAt(str, 9) null.

6.2. Ableitungstabelle / Graph-Parser

Die Anzahl der Elemente in der Sprache einer Graphgrammatik ist im allgemeinen nicht beschränkt. Dennoch kann es sein, dass ein gegebener Graph nicht in der Sprache der GG enthalten ist, der Graph G also nicht mit der GG herleitbar ist.

Bei einem Test, ob $G \in L(GG)$ gilt, ist es deshalb nicht sinnvoll, bei dem Axiom der GG zu starten und solange die Elemente der Sprache aufzählen, bis der gesuchte Graph G gefunden wurde: gilt $G \notin L(GG)$, kann das Aufzählen nicht terminieren.

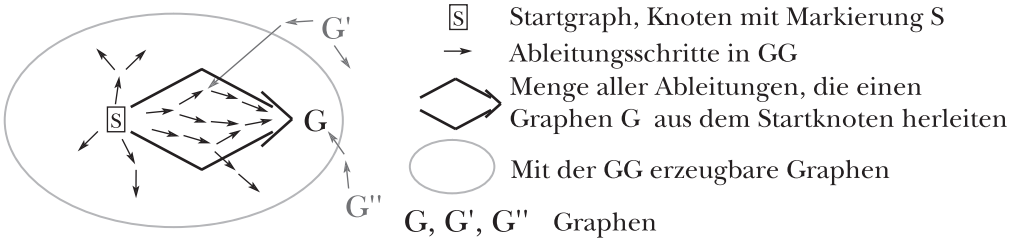

Abbildung 6.20.: Ableitungen vom Startgraph zum Graph G

Man muss also anders vorgehen, und zwar beginnend mit dem Graphen G versuchen, durch Rückwärtsanwenden der Produktionen das Axiom der GG herzuleiten. Dabei kann man allerdings auch Ableitungen finden, die nicht mit dem Axiom der GG beginnen (vgl. Abbildung 6.20 G′ und G″).

Sollen zusätzlich zur Entscheidung, ob $G \in L(GG)$ gilt, noch alle möglichen Ableitungen von GG gefunden werden, die auf den Graphen G führen, so müssen alle Wege, die von dem Startknoten der GG auf den Graphen G führen, abgespeichert werden. Dafür werden Ableitungstabellen benutzt.

Der allgemeine Aufbau einer solchen Ableitungstabelle ist in [Brandenburg 1988/1] zu finden. Im folgenden wird eine spezielle Variante vorgestellt, die für die Graphgrammatik GG_{WDS}, deren Sprache SP-Bäume enthält, optimiert ist.

Das Rückwärtsanwenden einer Produktion geschieht nun dadurch, dass in der String-Repräsentation des Graphen die String-Repräsentation der rechten Seite einer Produktion durch die entsprechende String-Repräsentation der linken Seite ersetzt wird.
Entsteht bei der Ersetzung eine String-Repräsentation, die noch nicht in der Tabelle aufgeführt ist, wird diese in die Tabelle eingefügt.
Für jede Ersetzung wird in einem Parsingschritt gespeichert, welche Produktion auf welchen Knoten angewendet werden muss, um von dieser neuen String-Repräsentation durch anwenden der Produktion die bereits vorhandene zu erreichen.

Sind mehrere Ersetzungen bei einer String-Repräsentation möglich, werden alle nacheinander ausgeführt und in die Tabelle aufgenommen.

Die Größe der Tabelle ist im allgemeinen nicht beschränkt. Da wir uns aber auf eine besonders einfache Graphgrammatik beschränken, reicht es aus, die Menge aller Ableitungsbäume zu bestimmen, um alle möglichen Ableitungen darzustellen. Sucht man also nur nach verschiedenen Ableitungsbäumen, die den Graphen herleiten, muss man die anzuwendenden Produktionen etwas sorgfältiger auswählen (anstatt alle möglichen anzuwenden). Dadurch kann die Grösse der Tabelle bedeutend reduziert werden.

Definition Parsingschritt Seien G_1, G_2 Graphen, beide von GG_{WDS} erzeugt, p=(L, R, C) eine Produktion von GG_{WDS},
nodeInStringAt(str(G_1), k)=nodeInStringAt(str(L), 0), sodass gilt $G_1 \Rightarrow_p G_2$ ist ein Ableitungsschritt in GG_{WDS}.
Dann ist σ=(p, k, str(G_2)) ein *Parsingschritt für str(G_1).*

Die Ableitungstabelle besteht aus Einträgen in der Form von String-Repräsentationen, denen eine Menge von Parsingschritten zugewiesen wird.

Definition Ableitungstabelle Sei G ein SP-Baum.
Die *Ableitungstabelle von G* in GG_{WDS} ist τ=(Γ, ϕ), mit

- Γ ist Menge von String-Repräsentationen str und
- ϕ=ϕ(Γ): str $\mapsto$ Menge von Parsingschritten σ.
- Γ und ϕ(Γ) sind wie folgt rekursiv definiert.
  - str(G) ist ein Element von Γ.
  - Seien G_1, G_2 Bäume, G_1, G_2 von GG_{WDS} erzeugt, str(G_2) $\in \Gamma$, p=(L, R, C) eine Produktion von GG_{WDS}, sodass gilt $G_1 \Rightarrow_p G_2$ ist ein Ableitungsschritt in GG_{WDS}.
    Dann ist str(G_1) $\in \Gamma$ und (p, k, str(G_2)) ein Element von ϕ(str(G_1)).
  - Γ und ϕ(Γ) enthalten nur Elemente, die wie oben konstruiert wurden.

Um die Tabelle möglichst klein zu halten, werden String-Repräsentationen mit doppeltem Vorkommen *einer* nichtterminalen Knotenmarkierung innerhalb einer Klammer (z.B.: a(LL)) nicht in die Tabelle aufgenommen. Kein Baum, der eine solche String-Repräsentation hat, kann von GG_{WDS} generiert werden.

Da nicht alle möglichen Ableitungen benötigt werden, sondern es ausreicht, mindestens eine Ableitung von jedem möglichen Ableitungsbaum zu finden, wird die Menge der Rückwärtsanzuwendenen Produktionen eingeschränkt. Es wird in der String-Repräsentation nicht nach allen Vorkommen der rechten Seite einer Produktion gesucht, sondern es wird immer nur die letzte innere Klammer der String-Repräsentation betrachtet und nach Ersetzungen dieses Ausdrucks gesucht.

Bildlich gesprochen ist die letzte innere Klammer der kleinste Unterbaum des kleinsten Unterbaums des Graphen. Wird dieser Teil durch das Rückwärtsanwenden einer Produktion noch kleiner, wird die String-Repräsentation nur an dieser Stelle verändert.

Auch bei dieser Vorgehensweise ist es möglich, dass String-Repräsentationen gefunden werden, die nicht mit der GG erzeugt werden können.
Zum Beispiel kann auf die String-Repräsentation a(a(LR)bb) die Produktion P_1 rückwärtsangewendet werden, wobei sich dann die String-Repräsentationen a(bbU) ergibt. a(bbU) kann nicht von GG_{WDS} erzeugt werden, da die einzige Produktion mit vier Knoten in dem Graphen der rechten Seite (P_0: S $-$ > a(LOR)) keinen Knoten mit der Markierung U besitzt.

6.3. Beispiel

Gegeben sei die String-Repräsentation eines SP-Baums G mit 2 inneren Knoten und 4 Blättern (wie in Abbildung 6.4 auf Seite 47) mit String-Repräsentation str(G)=a(a(bb)bb). Die zugehörige Ableitungstabelle τ=(Γ, ϕ) ist:

```
0, a(a(bb)bb), {}
1, a(a(bO)bb), {[15, 5, 0]}
2, a(a(bR)bb), {[14, 5, 0]}
3, a(a(bU)bb), {[13, 5, 0]}
4, a(a(LO)bb), {[16, 4, 1]}
5, a(a(LR)bb), {[16, 4, 2]}
6, a(a(OR)bb), {[15, 4, 2]}
7, a(a(LU)bb), {[16, 4, 3]}
8, a(a(OU)bb), {[15, 4, 3]}
9, a(a(RU)bb), {[14, 4, 3]}
10, a(bbO), {[9, 4, 4],  [7, 4, 5],  [8, 4, 6]}
11, a(bbL), {[12, 4, 4], [11, 4, 7], [10, 4, 8]}
12, a(bbU), {[1, 4, 5],  [3, 4, 7],  [2, 4, 9]}
13, a(bbR), {[6, 4, 6],  [4, 4, 8],  [5, 4, 9]}
14, a(bLO), {[16, 3, 10], [15, 4, 11]}
15, a(bOR), {[14, 4, 10], [15, 3, 13]}
16, a(bLR), {[14, 4, 11], [16, 3, 13]}
17, a(LOR), {[14, 4, 14], [16, 2, 15], [15, 3, 16]}
18, S, {[0, 0, 17]}
```

Pro Zeile ist ein Element von Γ angegeben und die Zeilen sind durchnumeriert. ϕ=ϕ(Γ) steht als Menge von Parsingschritten σ_i hinter der entsprechenden String-Repräsentation.

Der erste Eintrag in der Tabelle (Zeile 0) ist die String-Repräsentation des SP-Baumes G selbst str(G)=a(a(bb)bb).
Da der SP-Baum nur terminal markierte Knoten hat, gibt es keine Parsingschritte für diese String-Repräsentation.
Zeile 18 beinhaltet die String-Repräsentation S $\in \Gamma$ und ϕ(S)={[0, 0, 17]} besteht aus genau einem Parsingschritt: Produktion 0 (P_0) auf Knoten 0 (nodeInStringAt(S, 0)) anwenden, dann ist die String-Repräsentation des produzierten Graphen in Zeile 17. In der Zeile 17 wiederum steht die String-Repräsentation a(LOR) und eine Menge von drei Parsingschritten: ϕ(a(LOR))={[14, 4, 14], [16, 2, 15], [15, 3, 16]}. Dies bedeutet, es gibt drei mögliche Produktionen, die in diesem Zustand anwendbar sind: P_{14}, P_{15} und P_{16}. Die Produktion P_{14} ersetzt den Knoten nodeInStringAt(a(LOR), 4), also den Knoten mit Markierung R, die Produktion P_{15} ersetzt den Knoten nodeInStringAt(a(LOR), 3), also den Knoten mit Markierung O und P_{16} ersetzt den Knoten nodeInStringAt(a(LOR), 2), also der Knoten mit Markierung L.

Der Graph ist mit der GG_{WDS} herleitbar, da das Axiom der Graphgrammatik der letzte Eintrag in der Tabelle ist.

In Abbildung 6.21 ist die Ableitungstabelle als ein Graphen dargestellt. Bei der Berechnung der Ableitungstabelle wird bei Zeile 0 angefangen und der letzte Eintrag ist in Zeile 18. Auf den Graphen übertragen bedeutet dies, dass der Graph von links nach rechts aufgebaut wird. Es sei daran erinnert, dass die String-Repräsentation für die Menge von Bäumen steht, die diese String-Repräsentation besitzen.

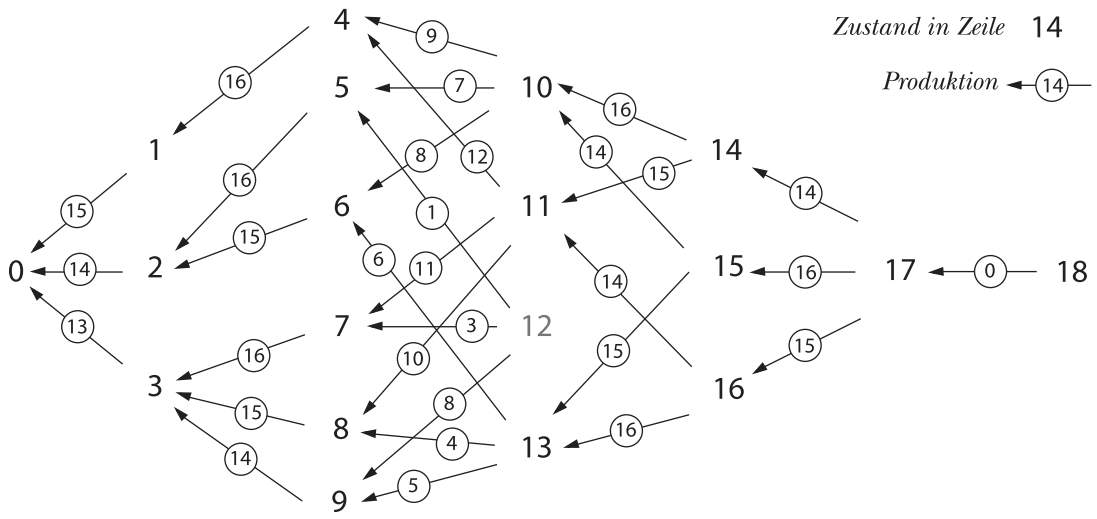

Abbildung 6.21.: Darstellung der Ableitungstabelle als Graph

Es gibt keinen Parsingschritt der aus dem Zustand in Zeile 12 führt. Dies bedeutet, dass es gibt keine initiale Ableitung in GG, sodass das Ergebnis der Ableitung die String-Repräsentation aus Zeile 12 hat.

Der Suchraum der LGG_{WDS} für dieses Beispiel ist in den Abbildungen 6.22 und 6.23 visualisiert.

In jeder Produktion der GG_{WDS} wird genau ein nichtterminaler Knoten durch einen terminalen Knoten ersetzt. Für einen SP-Baum mit n Knoten benötigt man eine Ableitung mit n Ableitungsschritten. Daher liegen alle Ziele in gleicher Tiefe im Suchraum.

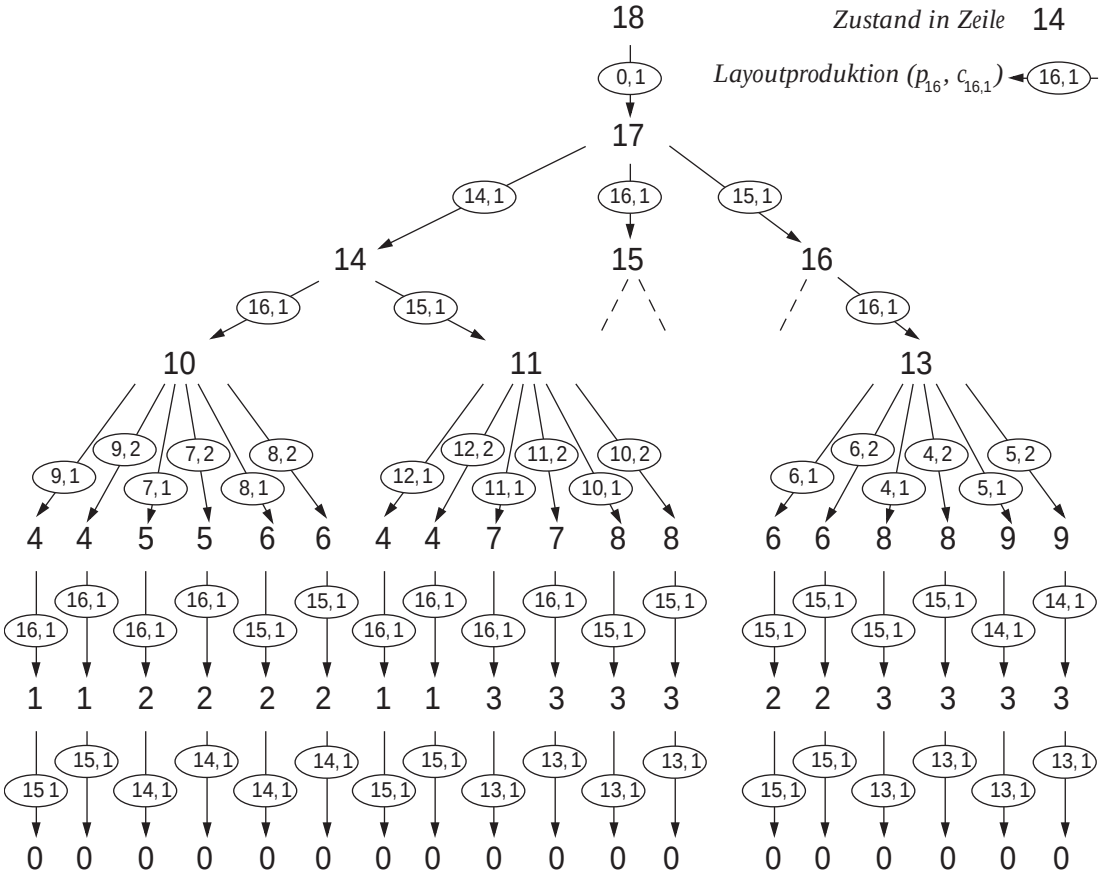

Abbildung 6.22.: Suchraum der LGG_{WDS} für Beispiel 6.3

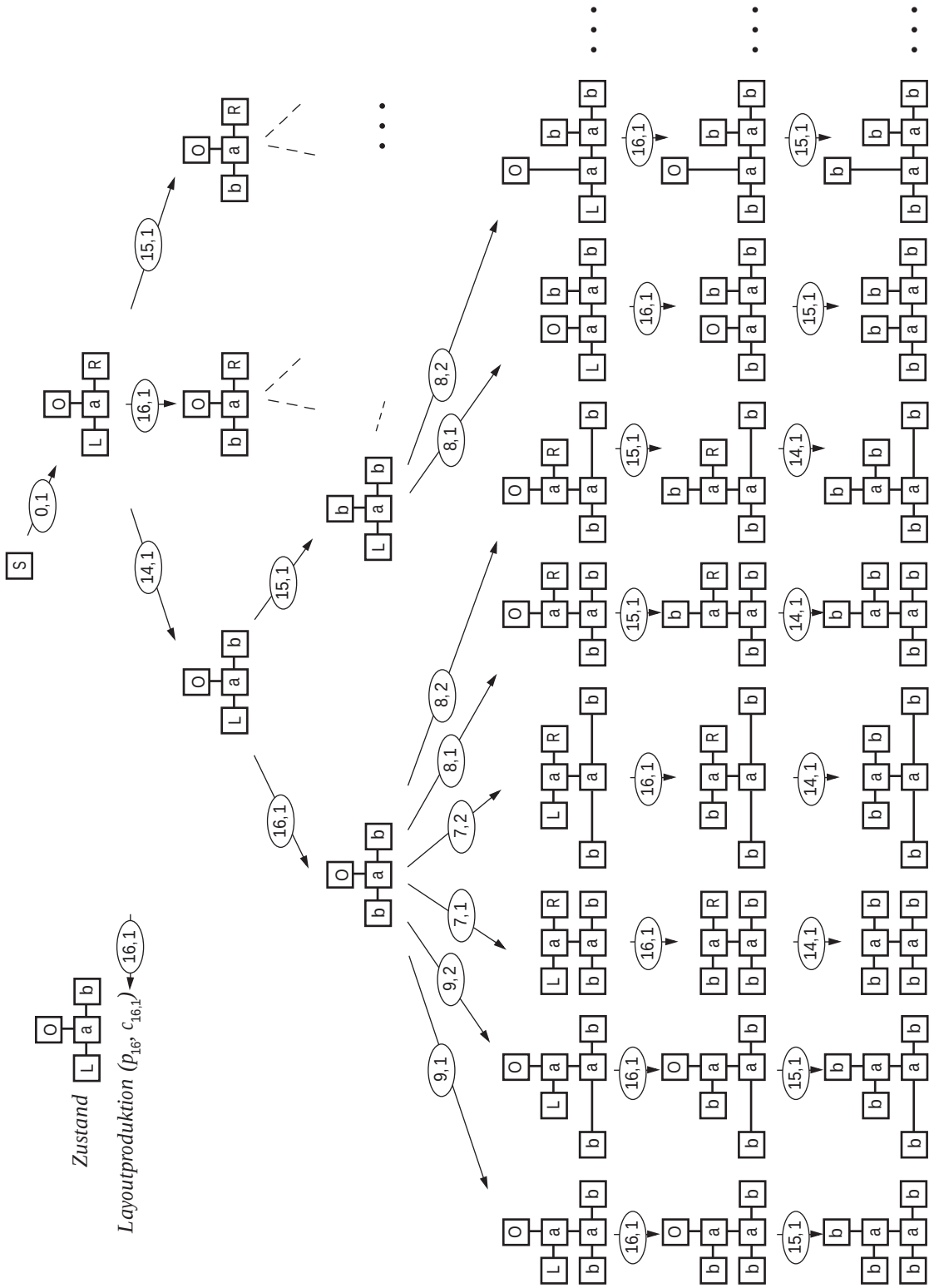

Abbildung 6.23.: Suchraum der LGG_{WDS} für Beispiel 6.3

Kostenfunktionen Eine Kostenfunktion bewertet eine Positionierung eines Graphen (den SP-Baum). Ein Graph soll möglichst "schön" sein. Doch was bedeutet "schön" ? Die übliche Lösung ist eine mathematische Formulierung, etwa

1. Minimiere die Anzahl der Kreuzungen.

2. Minimiere die Anzahl der Knicke.

3. Minimiere die benötigte Fläche.

4. Minimiere die Länge der Kanten.

In Abbildung 6.24 sind vier verschiedene Positionierungen für einen SP-Baum der Tiefe 2 dargestellt, an denen verschiedene Kostenfunktionen verglichen werden.

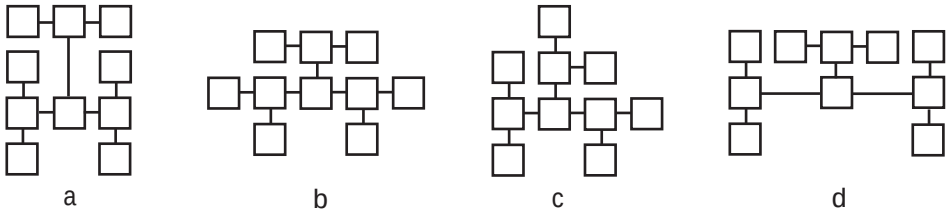

Abbildung 6.24.: Positionierungen eines SP-Baums

Die Positionierung b entspricht der Darstellung die ein Ingenieur wählen würde, wenn er eine solche WDS zeichnet. Daher sollte auch die Kostenfunktion ein Minimum für dieses Beispiel liefern.

Eine Wellendigitalstruktur ist immer knickfrei und kreuzungsfrei zu zeichen, deshalb ist die Berechnung der Anzahl vorhandener Knicke und Kreuzungen unnötig.

Die umschliessende Rechteckfläche zu minimieren ist nicht ausreichend, wie das obige Beispiel zeigt: die Positionierung a belegt eine Rechteckfläche von $4 * 3 = 12$, Positionierungen b und d belegen jeweils 15 und c sogar 16.

Die Summe der Kantenlängen zu minimieren ist hier besser geeignet: für die Positionierung a ist die Summe der Kantenlängen gleich 10, für b und c ist die Summe gleich 9 und in der Positionierung d ergibt die Summierung den Wert 11.

Eine Kombination aus der Summe der Kantenlängen und der umschliessenden Rechteckfläche ist nach Ansicht des Autors dieser Arbeit eine geeignete Kostenfunktion, die beschreibt, wie „schön“ ein Graph gezeichnet ist.

6.4. Suchstrategie

Die Berechnung eines flächenminimalen Layouts ist schon für binäre Bäume NP-hart (siehe [Brandenburg 1988/2]). Daher beschränken wir uns auf eine bestimmte Graphklasse und benutzen eine Heuristik, um in dieser Graphklasse zu suchen. Die Graphklasse ist durch die Layout-Graphgrammatik vorgegeben und die Heuristik besteht im Wesentlichen darin, nicht das flächenminimale Layout zu finden, sondern nur durch Tiefensuche ein gutes Ergebnis zu erhalten, das eventuell suboptimal ist.

Der angegebene Algorithmus basiert auf einer BF*-Suche aus der [Vorlesung Suche]. Sei T ein SP-Baum, für den eine Positionierung berechnet werden soll.

1. Berechne die String-Repräsentation und die Indexfunktion für T.
2. Erstelle die Ableitungstabelle τ=(Γ, ϕ) von T in GG_{WDS}.
3. Generiere den Startgraphen S bestehend aus dem Startknoten der GG_{WDS}.
4. Füge den Startgraphen S in die OPEN-Liste ein.
5. Wenn die OPEN-Liste leer ist, breche ab mit „Failure“
6. Entferne aus der OPEN-Liste einen Graphen G, für den f minimal ist.
7. Wenn G ein Ziel ist, dann weiter mit Schritt 11.
8. Erzeuge die Nachfolger von G mit Hilfe der Parsingschritte $\phi(\mathrm{str}(G))$.
9. Für jeden Graphen G' aus der Menge der Nachfolger durchlaufe die Schritte:
   - Berechne die Positionen der Knoten von G'.
   - Berechne Kostenfunktion $f(G')$. Weise $f(G')$ G' zu.
   - Wenn G' eine gültige Positionierung besitzt, dann füge G' in die OPEN-Liste ein.
10. Weiter mit Schritt 5.
11. Übertrage Position der Knoten von G auf die Knoten des Baumes T mit Hilfe der Indexfunktion.

Die Kostenfunktion $f(G'))$ setzt sich aus zwei Teilen zusammen $f(G')=g(G')+h(G')$, wobei $g(G')$ die tatsächlichen Kosten des Graphen G' beschreibt, wie im vorherigen Kapitel und $h(G')$ die geschätzten Kosten bis zum nächsten Ziel. Ein Graph ist hier ein Ziel, wenn der Graph ohne Positionsbedingungen dem Graphen entspricht, für den eine Positionierung gesucht wird.
Dabei ist zu beachten, dass die geschätzen Kosten $h(G')$ immer kleiner oder gleich den tatsächlichen Kosten bis zum nächsten Ziel sind.

Der Algorithmus ist nur ein Beispiel, wie in dem Suchraum, den die LGG_{WDS} aufspannnt, gesucht werden kann. Je nach Bedürfnis des Benutzers kann der Algorithmus angepasst werden. Soll schnell ein Ergebnis gefunden werden, sollte ein Tiefensuche-Algorithmus verwendet werden, falls ein sehr kompaktes Layout benötigt wird, ist eher in der Breite, bzw. Best-First zu suchen.

7. Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Verfahren vorgestellt, dass mit Hilfe von Graphgrammatiken eine Positionierung der Knoten eines SP-Baumes berechnet.

Ziel dieser Diplomarbeit war es, zu untersuchen, ob für einen gegebenen SP-Baum mit Layout-Graphgrammatiken eine angemessene Positionierung mit vertretbarem Aufwand berechnet werden kann.
Dazu wurde eine Layout-Graphgrammatik (LGG_{WDS}) entwickelt und ein entsprechendes Verfahren angegeben, um mit dieser LGG eine Positionierung der Knoten zu berechnen. Durch die Beschränkung auf eine spezielle Klasse von Positionierungen des SP-Baumes und durch die Verwendung von Heuristiken wurde dieses Ziel teilweise erreicht.

Der größte Teil des Berechnugsaufwandes besteht darin die Menge von Ableitungsbämen zu finden, die den SP-Baum herleiten. Im Falle der LGG_{WDS} ist dieser Aufwand so groß, dass in der Praxis nur SP-Bäume betrachtet werden sollten, die bis zu 100 Knoten (ca. 50 Bauelemente) besitzen. Bei größeren Graphen ist der Zeitaufwand für die Berechnung zu groß.
Vielleicht ist es möglich die Menge von Ableitungsbäumen in einer effizienteren Art und Weise zu bestimmen.

Ist dies nicht möglich, dann sind Layout-Graphgrammatiken, nach Ansicht des Autors, dort besser geeignet, wo die Ableitung bereits gegeben ist: in der Top-Down-Optimierung. Hier ist es nur ein geringer zusätzlicher Aufwand, die Positionsbedingungen der Ableitung hinzuzufügen, und dann nach einer Layout-Abeitung zu suchen, deren Ergebnis eine optimale Positionierung liefert.

[Wanke 1989] beschreibt die Möglichkeiten, einen hierarchischen Graphen zu vereinfachen, indem Subgraphen als einzelne Knoten beschrieben werden. Dieses kann die Lesbarkeit der Darstellung erheblich verbessern. C-Komponenten eines SPC-Baumes können so dargestellt werden.

Da in dieser Arbeit nur Parallel- und Seriellgraphen betrachtet wurden, könnte in einer Erweiterung die Verwendung von Layout-Graphgrammatiken für Wellendigitalstrukturen, bei denen auch C-Komponenten vorkommen, untersucht werden.

Literaturverzeichnis

[Brandenburg 1990] F.J. Brandenburg: *Layout Graph Grammars: the Placement Approach*, Proc. 4.Intern. Workshop on Graph Grammars, In Lecture Notes in Computer Science 532, 144-156, Bremen, Germany, 1990

[Brandenburg 1994] F.J. Brandenburg: *Designing Graph Drawings by Layout Graph Grammars*, In Proc. of the DIMACS Int. Workshop on Graph Drawing 416-427, Princeton, New Jersey, 1994

[Brandenburg 1988/1] F.J. Brandenburg: *On Polynomial Time Graph Grammars*, Proc. 5. Annual Symposium on Theoretical Aspects of Computer Science, STACS 88, In Lecture Notes in Computer Science 294, 227-236, Bordeaux, France, 1988

[Brandenburg 1988/2] F.J. Brandenburg: *Nice Layouts of Graphs and Trees are computationally hard* MIP Bericht, Universität Passau, 1988

[Hickl 1995] Timo Hickl: *Rechtwinkliges Layout von hierarchisch strukturierten Graphen*, Dissertation Universität Passau, 1995

[Wanke 1989] Egon Wanke: *Algorithmen und Komplexitätsanalyse für die Verarbeitung hierarchisch definierter Graphen und hierarchisch definierter Graphfamilien*, Dissertation Universität Paderborn, 1989

[Kaul 1983] M. Kaul: *Parsing of graphs in linear time* In H. Ehrig, M. Nagl, and G. Rozenberg, editors, Graph Grammars and Their Application to Computer Science, Lecture Notes in Computer Science 153, pages 206–218, 1983.

[Kaul 1985] M. Kaul: *Syntaxanalyse von Graphen bei Präzedenz-Graphgrammatiken* Dissertation, Universität Passau, Germany, 1985

[Graphlet] Graphlet: A toolkit for graph editors and graph algorithms, Faculty of Mathematics and Computer Science, University of Passau, http://www.infosun.fmi.uni-passau.de/Graphlet/

[Löwe, Müller, Taentzer 1995] Michael Löwe und Jürgen Müller: *Einführung in die Theoretische Informatik, Graphersetzung* Eine Einführung in typische Anwendungen, Probleme und Techniken, überarbeitet von Gabriele Taentzer, 1995

[Steinmetz 2001] Rita Steinmetz: *Analysis of Design Graph Grammar*, Studienarbeit, Universtiät Paderborn, 2001

[Lok, Feiner 2001] Simon Lok and Steven Feiner: *A Survey of Automated Layout Techniques for Information Presentations*, Dept. of Computer Science, Columbia University, Ney York, USA, 2001

[Battista, Eades, Tamassia, Tollis] G. Di Battista, P. Eades, R. Tamassia and I. G. Tollis: *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice-Hall Inc., Upper Saddle River, New Jersey, 1999

[Vorlesung Suche] Benno Stein: *Vorlesung Suche*, Universität Paderborn, SS 2000

[Perl 1984] Judea Pearl: *Heuristics* Intelligent Search strategies for Computer Problem Solving, Addison-Wesley, 1984

[Cormen, Leiserson, Rivest 1990] Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest: *Introduction to Algorithms* MIT Press / McGraw-Hill, Cambridge, Massachusetts, 1990

[Garey, Johnson 1979] Michael R. Garey, David S. Johnson: *Computers and Intractability* A Guide to the Theory of NP-Completeness, Bell Laboratories, Murray Hill, New Jeresy, 1979

[Stein 2001] Benno Stein: *Modell Construction in Analysis and Synthesis Tasks*, Professorial dissertation (to appear), Universität Paderborn, 2001

[K.Ochs, Stein 2001] Karlheinz Ochs and Benno Stein: *On the Design and Use of Wave Digital Structures*, Series Computer Science, Universität Paderborn, 2001

[J.Ochs 2000] Jörg Ochs: *Untersuchungen zur digitalen Nachbildung nichtlinearer reaktiver Bauelemente*, Diplomarbeit, Universität Paderborn, 2000

[Fettweis 1981] Fettweis, Alfred: *Principles of Complex Wave Digital Filters* Circuit Theory and Applications 9, S. 119-134, 1981.

[Fettweis 1986] Alfred Fettweis: *Wave digital filters: Theory and practice*, Proceedings of the IEEE Vol. 74, Feb. 1986, Seiten 270-327.

[Fettweis 1990] Fettweis, Alfred: *Elemente nachrichtentechnischer Systeme* Teubner Studienbücher, Stuttgart 1990

...und hier endet es.