Seminar

Graph-Grammatiken

Lehrstuhl für Informatik III
RWTH Aachen
Sommersemester 2004

Graphtransformationsunits und -module

Rüdiger Heinrich

Inhaltsverzeichnis

1 Einführung **1**

2 Graphtransformationsansatz **1**
2.1 Definition des Graphtransformationsansatzes 1
2.2 Erläuterungen zum Graphtransformationsansatz 2

3 Einführung des Beispiels: Floyd-Algorithmus **4**
3.1 Beschreibung des Algorithmus 4
3.2 Der entsprechende Graphtransformationsansatz 5

4 Graphtransformationsunits **6**
4.1 Definition der Graphtransformationsunit 6
4.1.1 Verschränkte Semantik 7
4.1.2 Beispiele für Graphklassen 8
4.1.3 Beispiele für Kontrollbedingungen 9
4.2 Anwendungssequenz 10
4.3 Eigenschaften der Anwendungssequenz 10
4.4 Spezifikation des Algorithmus von Floyd mit einer GTU 11

5 Korrektheit und Laufzeit des Algorithmus **13**
5.1 Beweis der Korrektheit 13
5.2 Komplexität 15

6 Graphtransformationsmodule **16**

7 Zusammenfassung **17**

1 Einführung

Diese Arbeit behandelt das Thema Graphtransformationsunits und -module. Module, aus herkömmlichen Programmiersprachen wie *C* bekannt, fassen mehrere Funktionen zu einer Einheit zusammen. Diese Funktionen haben einen logischen Zusammenhang und können sich gegenseitig aufrufen. Module haben daher eine Import- und eine Exportschnittstelle, wo beschrieben wird, welche Funktionen sie zur Arbeit benötigen, bzw. sie der Umgebung zur Verfügung stellen. Ein Graphtransformationsmodul ähnelt einem solchen Modul einer herkömmlichen Programmiersprache. Es enthält, analog zu den oben stehenden Funktionen, Graphtransformationsunits, welche auf Graphen arbeiten. Diese Graphtransformationsunits benutzen für ihre Arbeit elementare Operationen oder auch andere Units. Durch Kontrollstrukturen wird bestimmt, in welcher Reihenfolge die Operationen ausgeführt werden.

Diese Arbeit beginnt mit der Beschreibung des Begriffs Graphtransformationsansatz, welcher für die darauf folgende Definition von Graphtransformationsunits benötigt wird. Die vorgestellten Definitionen werden anhand eines Beispiels erläutert, dem Algorithmus von Floyd. Dieser wird in Kapitel 3 erklärt und der entsprechende Graphtransformationsansatz beschrieben. Im Kapitel 4 werden Graphtransformationsunits definiert, Eigenschaften dieser erläutert und die Graphtransformationsunit für den Algorithmus von Floyd beschrieben. Die Korrektheit und Laufzeit dieses Algorithmus wird in Kapitel 5 gezeigt. Im Kapitel 6 werden Graphtransformationsmodule definiert, welche mehrere Graphtransformationsunits zu einer Einheit zusammenfassen können. Die Arbeit schließt mit einer Zusammenfassung.

2 Graphtransformationsansatz

Zur Beschreibung von Graphen gibt es verschiedene Ansätze, zum Beispiel *Node Replacement Graph Grammars* [1] oder *Hyperedge Replacement Graph Grammars* [1]. Um Graphtransformationsunits unabhängig von der benutzten Graphbeschreibung definieren zu können, wird eine abstrakte Notation für den *Graphtransformationsansatz* (kurz *GTA*, original: Graph Transformation Approach) benutzt. Auf diesem bauen Graphtransformationsunits (kurz *GTU*, original *Graph Transformation Unit*) auf.

Vor der Definition ist es notwendig, den überladenen Operator SEM einzuführen. Dieser bildet z.B. Graphklassenbeschreibungen auf die Menge der dadurch beschriebenen Graphen ab oder bildet Kontrollbedingungen auf die Menge der Graphpaare ab, die durch die Kontrollbedingung erlaubt sind (Die Definition dieser beiden Begriffe wird im Folgenden erklärt). Durch das Argument des Operators ist jeweils eindeutig bestimmt, welche Bedeutung dieser hat.

2.1 Definition des Graphtransformationsansatzes

Ein GTA ist ein Fünftupel $G = (\mathcal{G}, \mathcal{R}, \Rightarrow, \mathcal{E}, \mathcal{C})$, wobei die einzelnen Komponenten wie folgt definiert sind:

- $\mathcal{G}$ ist eine Klasse von *Graphen*, beispielsweise gerichtete Graphen, Hypergraphen, etc.
- $\mathcal{R}$ ist eine Menge von *Regeln*, welche diese Graphen verändern können. Zum Beispiel „erstelle aus einem Knoten mit der Beschriftung 1 zwei, mit einer

Kante verbundene Knoten mit der Beschriftung 2“.

- $\Rightarrow$ ist der *Regeloperator*, welcher eine binäre Relation $\Rightarrow_r \subseteq \mathcal{G} \times \mathcal{G}$ für jedes $r \in \mathcal{R}$ definiert. Er beschreibt, wie die Regel auf einen Graphen angewendet wird.
- $\mathcal{E}$ ist eine Menge von Beschreibungen von *Graphklassen*. Jede Beschreibung $e \in \mathcal{E}$ definiert über ihre Semantik eine Teilmenge $SEM(e) \subseteq \mathcal{G}$. Beispiele sind
  - Graphen, deren Knoten alle mit 1 beschriftet sind,
  - azyklische Graphen oder
  - zusammenhängende Graphen.
- $\mathcal{C}$ ist eine Menge von *elementaren Kontrollbedingungen* über einer Menge von Bezeichnern. Diese regulieren den Ableitungsprozess und schränken ihn ein. Dies kann dadurch geschehen, dass die Reihenfolge der Regeln festgelegt wird oder Abbruchbedingungen des Ableitungsprozesses angegeben werden.

2.2 Erläuterungen zum Graphtransformationsansatz

Kann ein Graph G mit der Regel r zum Graphen G' abgeleitet werden, so ist das Paar $(G, G') \in \Rightarrow_r$, wofür auch $G \Rightarrow_r G'$ geschrieben wird. Dies ist ein *direkter Ableitungsschritt*. Für eine Menge $P \subseteq \mathcal{R}$ wird die Vereinigung aller Regeln $\Rightarrow_r$ $(r \in P)$ mit $\Rightarrow_P$ bezeichnet, die reflexiv transitive Hülle mit $\Rightarrow_P^*$. Ist ein Paar von Graphen $(G, G') \in \Rightarrow_P^*$, so ist G gleich G' oder G' lässt sich aus G durch beliebig viele Anwendungen von Regeln aus P herleiten. Für $(G, G') \in \Rightarrow_P^*$ wird auch $G \Rightarrow_P^* G'$ geschrieben, dies ist eine *Ableitung* von G nach G' mit Regeln aus P.

Eine Kontrollbedingung $c \in \mathcal{C}$ legt eine Menge von möglichen Ableitungen fest. Diese werden als Paare aus $\mathcal{G} \times \mathcal{G}$ dargestellt. Eine Ableitung von G nach G' ist nur möglich, wenn das Paar (G, G') in der Semantik der Kontrollbedingung $SEM(C)$ enthalten ist. Es werden bool'sche Ausdrücke über $\mathcal{C}$ benutzt, um die elementaren Kontrollbedingungen zu *komplexen Kontrollbedingungen* zu verknüpfen. Außerdem wird die Konstante *true* eingeführt, welche alle Graphpaare $\mathcal{G} \times \mathcal{G}$ darstellt, also alle Ableitungen zulässt. Es folgen die Bedeutungen dieser komplexen Kontrollbedingungen.

- $SEM_E(true) = \mathcal{G} \times \mathcal{G}$ stellt *alle* möglichen Ableitungen dar. Es werden also keine Einschränkungen gemacht.
- $SEM_E(c_1 \vee c_2) = SEM_E(c_1) \cup SEM_E(c_2)$. Eine Ableitung wird durch $e_1 \vee e_2$ zugelassen, wenn sie in e_1 *oder* in e_2 zugelassen ist.
- $SEM_E(c_1 \wedge c_2) = SEM_E(c_1) \cap SEM_E(c_2)$. Eine Ableitung wird durch $c_1 \wedge c_2$ nur dann zugelassen, wenn sie in c_1 *und* c_2 zugelassen ist.
- $SEM_E(\overline{c}) = \mathcal{G} \times \mathcal{G} - SEM_E(c)$ lässt eine Ableitung nur zu, wenn sie *nicht* in c erlaubt ist.

$\mathcal{B}(\mathcal{C})$ ist die Menge der komplexen Kontrollbedingungen, welche aus den einfachen Kontrollbedingungen $\mathcal{C}$, die ein GTA zur Verfügung stellt, gebildet werden können.

Ein einfaches Beispiel für einen GTA:

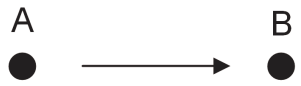

Abbildung 1: Die Regel r_1

Abbildung 2: Die Regel r_2

- $\mathcal{G}$ besteht aus gerichteten Graphen mit Knotenbeschriftungen aus dem Alphabet {A,B}.
- $\mathcal{R}$ besteht aus folgenden Regeln:
  - Regel r_1: Ändere die Beschriftung eines Knotens von A nach B (siehe Abbildung 1)
  - Regel r_2: Fasse zwei Knoten, die mit B beschriftet sind und mit einer Kante verbunden sind zu einem Knoten zusammen (siehe Abbildung 2).
- $\Rightarrow$ wird wie folgt definiert:
  - r_1 wird angewendet, indem im Graphen G ein Knoten mit der Beschriftung A gesucht wird und an diesem die Beschriftung in B geändert wird. Es entsteht der Graph G' und es gilt: $G \Rightarrow_{r_1} G'$
  - r_2 wird angewendet, indem im Graphen G zwei mit B beschriftete benachbarte Knoten gesucht werden. Einer von diesen beiden wird gelöscht und allen Kanten, die diesen Knoten als Ziel oder Quelle hatten, wird der andere Knoten als Ziel bzw. Quelle zugeordnet. Es entsteht der Graph G' und es gilt: $G \Rightarrow_{r_2} G'$ (siehe Abbildung 3).
- $\mathcal{E}$ enthält die Beschreibungen für zwei Graphklassen:
  e_1 = Die Menge der Graphen, deren Knoten alle mit A beschriftet sind,
  e_2 = Die Menge der Graphen, die genau zwei Knoten enthalten, welche mit B beschriftet sind. e_1 steht hier für die Beschreibung der Graphklasse und $SEM(e_1)$ für die Menge der Graphen, die der Beschreibung entsprechen.

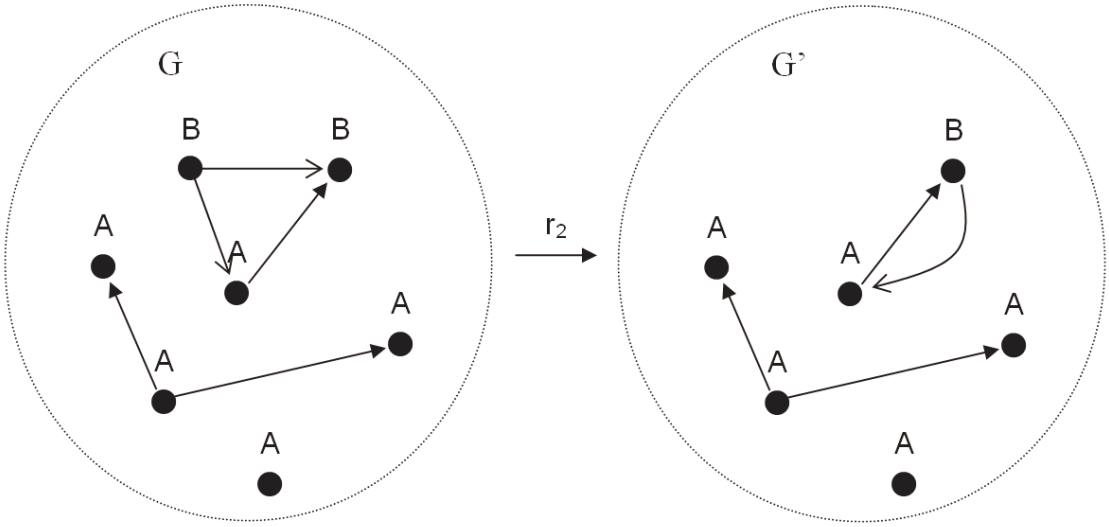

Abbildung 3: Anwendung der Regel r_2 auf einen Graphen

- $\mathcal{C}$ enthält die Kontrollbedingung c. Diese lautet: Wende auf einen Graphen G aus $SEM(e_1)$ die Regel r_1 zwei mal an. Entsteht so ein Graph G' aus $SEM(e_2)$, so wende r_2 auf diesen Graphen an. Diese Bedingung für G' kann bei zu kleinen Graphen mit nur einem Knoten nicht erfüllt werden, wie dem unteren Teilgraph in Abbildung 3 (der einzelne Knoten). Das Ergebnis dieser Ableitung ist G'' und es gilt, dass $(G, G'') \in SEM_E(c)$.

3 Einführung des Beispiels: Floyd-Algorithmus

In diesem Kapitel wird ein Beispiel eingeführt, anhand dessen die Begriffe Graphtransformationsansatz und Graphtransformationsunit erläutert werden. Dies ist der Algorithmus von Floyd.

3.1 Beschreibung des Algorithmus

Der Algorithmus von Floyd berechnet in einem gerichteten Graphen G mit Kantengewichten den kürzesten Weg zwischen allen Paaren von Punkten v und v'. Die Idee dabei ist, zu Beginn der Berechnung alle Knoten außer v und v' auszuschließen und nur direkte Wege zu suchen. Nach und nach werden immer mehr Knoten zur Berechnung zugelassen und Wege über diese Knoten gesucht. Der kürzeste Weg geht dann entweder über diesen neuen Knoten oder wird ohne Kanten, welche adjazent zu diesem Knoten sind, gebildet (siehe Abbildung 4). Kommt ein neuer Knoten $\overline{v}$ dazu, werden die Gewichte der Kanten von v nach $\overline{v}$ und von $\overline{v}$ nach v' addiert. Der Algorithmus fügt bei der Neuberechnung der Kosten eines Weges von v nach v' über $\overline{v}$ eine neue Kante in den Graphen ein, mit einem Gewicht, welches den Kosten dieses Weges entspricht. Gibt es bereits eine Kante zwischen diesen beiden Knoten, so wird die 'schwerere' der beiden entfernt. Die Kürzere repräsentiert den bisher kürzesten Weg zwischen den beiden Knoten.

Der Algorithmus terminiert, wenn alle Knoten zur Berechnung zugelassen sind, d.h. die Wege über alle möglichen Kanten berechnet wurden. Dann gibt es genau eine Kante von v nach v' mit minimalem Gewicht, oder keine Kante, falls v' von v

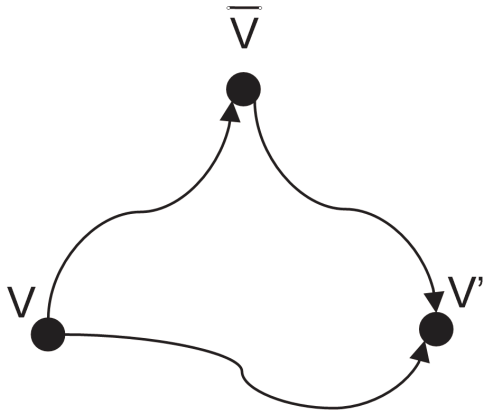

Abbildung 4: Berechnung des kürzesten Weges von v nach v' mit Berücksichtigung des Knotens $\overline{v}$

Abbildung 5: Beispiel für eine Regel mit Kontextkante

aus nicht erreichbar ist.

3.2 Der entsprechende Graphtransformationsansatz

Der Algorithmus wird in einer GTU implementiert (siehe Kapitel 4.4). Diese basiert auf folgendem GTA $G = (\mathcal{G}, \mathcal{R}, \Rightarrow, \mathcal{E}, \mathcal{C})$:

1. $\mathcal{G}$: Die hier betrachteten Graphen sind gerichtete Graphen mit Kantengewichten und einer zusätzlichen Beschriftung der Knoten durch die Funktion l: $G = (V, E, s, t, l, dist)$. V ist die Knotenmenge, E die Kantenmenge. $E(v, v')$ bezeichnet die Menge der Kanten zwischen v und v'. $s : E \to V$ ordnet jeder Kante einen Startknoten zu und $t : E \to V$ den Zielknoten. $l : V \to \{0, 1, 2\}$ ist eine Knotenbeschriftung, welche der Algorithmus benötigt, für den Graphen selbst aber ohne Belang ist. $dist : E \to \mathbb{N}$ weist den Kanten die Gewichte zu.

2. $\mathcal{R}$: Die Regelmenge besteht aus Paaren von Graphen $r = (R, L)$. Die beiden Regelseiten haben jeweils die gleiche Knotenmenge, da durch den Algorithmus weder Knoten zum Graphen hinzu kommen noch gelöscht werden. Die linke Seite der Regel L hat möglicherweise noch eine zusätzliche 'Kontextkante', welche in der Visualisierung gestrichelt angezeigt wird. Diese dient als negative Kontextbedingung, d.h. die Regel wird nur angewendet, wenn diese Kante im Graphen G nicht vorhanden ist. In Abbildung 5 ist ein Beispiel für eine Regel mit Kontextkante zu sehen. Diese Regel fügt eine Kante von einem mit 1 beschrifteten Knoten zu einem mit 2 beschrifteten Knoten in den Graphen ein, falls noch keine Kante zwischen diesen beiden Knoten existiert. Anschließend wird der erste Knoten mit 2 beschriftet. Existiert bereits eine Kante zwischen

diesen beiden Knoten, so wird diese Regel nicht angewendet, also auch keine Umbeschriftung ausgeführt.

3. $\Rightarrow_r$: Der Regeloperator zu einer Regel r kann wie folgt beschrieben werden: (1) Suche einen Teilgraph L_0 innerhalb von G, welcher zur linken Regelseite L isomorph ist. (2) Entferne die Kanten aus L_0. (3) Füge die Kanten aus der rechten Regelseite R ein. (4) Ändere die Knotenbeschriftungen, d.h. die Funktion $l(v)$, wo sie in L und R verschieden sind.

   Enthält die linke Regelseite eine Kontextkante, so dient diese als negative Kontextbedingung, d.h. die Regel kann nur angewendet werden, falls der Ausgangsgraph L_0 keine solche Kante enthält.

   Falls ein Graph G' durch diese Regelanwendung aus einem Graphen G ableitbar ist, dann ist $(G, G') \in \Rightarrow_r$, wofür auch $G \Rightarrow_r G'$ geschrieben wird.

4. $\mathcal{E}$: Es gibt zwei Graphklassen, die für den Algorithmus von Interesse sind. Diese ergeben sich aus den Knotenbeschriftungen. Die eine Klasse $SEM(Knotenbeschriftung\,0)$ enthält alle Graphen, die ausschließlich aus Knoten mit der Beschriftung 0 bestehen. $SEM(Knotenbeschriftung\,2)$ enthält alle Graphen, die ausschließlich aus Knoten mit der Beschriftung 2 bestehen. Weitere Beschreibungen von Graphklassen sind in $\mathcal{E}$ nicht enthalten, obwohl weitere Klassen aufgrund der möglichen Knotenbeschriftungen existieren.

5. $\mathcal{C}$: Die Kontrollbedingung besteht aus der Menge der regulären Ausdrücke über den Regelbezeichnungen und dem Ausdruck „so lange, wie möglich“. Diese beschreiben, in welcher Reihenfolge und wie oft die Regeln angewandt werden. Dieser Graphtransformationsansatz enthält jedoch keine konkreten Regeln und Kontrollbedingungen für den Floyd-Algorithmus. Diese werden erst in den Graphtransformationsunits in Kapitel 4.4 beschrieben. Angenommen, es gäbe die Regeln r_1, r_2 und r_3. Dann ist der Ausdruck $r_1(r_2 \vee r_3)^*$ eine Kontrollbedingung, welche beschreibt, dass zuerst die Regel r_1 genau einmal ausgeführt wird und danach beliebig oft die Regel r_2 oder r_3.

Bevor die GTU für diesen Algorithmus beschrieben wird, werden diese im folgenden Kapitel zuerst definiert.

4 Graphtransformationsunits

Im folgenden Kapitel werden Graphtransformationsunits sowie die Begriffe „Verschränkte Semantik“ und „Anwendungssequenz“ definiert. Anschließend folgt die GTU des Algorithmus von Floyd.

4.1 Definition der Graphtransformationsunit

Eine Graphtransformationsunit wird über einem Graphtransformationsansatz $\mathcal{A} = (\mathcal{G}, \mathcal{R}, \Rightarrow, \mathcal{E}, \mathcal{C})$ (Beschreibung siehe Kapitel 2) aufgebaut. Eine GTU

$$trut = (I, U, R, C, T)$$

($trut$ = GraphTRansformationsUNit) besteht aus folgenden Teilen:

1. $I \in \mathcal{E}$ ist eine Graphklasse, welche die initialen Graphen spezifiziert. Die Semantik dieser Graphklasse enthält die Graphen, die am Anfang einer Ableitung stehen dürfen.

2. U beschreibt die importierten GTUs. Jede importierte Unit u definiert über ihre Semantik $SEM(u)$ eine Menge von Graphpaaren. Der erste Graph dieser Graphpaare kann mit dieser Unit zu dem zweiten Graph abgeleitet werden. Diese Menge von Graphpaaren wird der importierenden Unit zur Verfügung gestellt.

3. $R \subseteq \mathcal{R}$ ist die Menge der vorhandenen Transformations-Regeln.

4. $C \in \mathcal{B}(\mathcal{C})$ ist die Kontrollbedingung (näheres siehe Kapitel 4.1.3).

5. $T \in \mathcal{E}$ ist eine Graphklasse, welche die Endgraphen beschreibt.

Die Elemente von *trut* werden auch wie folgt bezeichnet: U_{trut}, I_{trut}, R_{trut}, C_{trut}, T_{trut}. Die Menge aller Transformationsunits über $\mathcal{A}$ wird als $\mathcal{T}_{\mathcal{A}}$ bezeichnet.

Zur Vereinfachung der Definition, werden durch U nur bereits definierte GTUs importiert. Zu Beginn werden also Units erstellt, die keine anderen Units importieren - dies sind GTUs vom Level 0. Diese werden von Units aus Level 1 importiert, und so weiter. Insgesamt ergiebt dies eine baumartige, azyklische Import-Struktur.

Falls I nur einen einzelnen Graph definiert, U leer ist (also keine weiteren Units importiert werden) und die Kontrollbedingung C gleich der Konstanten *true* ist, definiert die GTU eine einfache Graphgrammatik als Sonderfall einer Graphtransformationsunit. Der eine initiale Graph entspricht in diesem Fall dem Startsymbol. Es wird auch nicht die Möglichkeit genutzt, Units zu importieren, oder den Ableitungsprozess durch Kontrollbedingungen einzuschränken.

4.1.1 Verschränkte Semantik

Die Semantik einer GTU besteht darin, der Umgebung eine Funktion für eine Berechnung auf Graphen zur Verfügung zu stellen. Im Allgemeinen wird aber keine eindeutige Funktion bereitgestellt, sondern eine Relation zwischen Graphen. D.h. dass einem Graphen nicht eindeutig ein anderer Graph zugeordnet wird. Im Allgemeinen kann ein Graph auf mehrere andere Graphen abgeleitet werden, weil mehrere Regeln an verschiedenen Stellen auf den Graphen angewendet werden können; der Ableitungsprozess ist nichtdeterministisch. Die Menge der Graphpaare (G, G'), wobei G ein initialer Graph für die Graphtransformationsunit ist, G' aus G abgeleitet werden kann und G' ein terminaler Graph ist, bildet eine Relation. Diese Relation wird *Verschränkte Semantik* (original: *interleaving semantics*) genannt und beschreibt alle möglichen Ableitungen der GTU. Ein Paar von Graphen (G, G') steht genau dann in dieser Relation, wenn

1. $G \in I$ und $G' \in T$, d.h. G ein initialer Graph und G' ein terminaler Graph ist,

2. G' aus G durch mehrfaches Anwenden von Regeln aus der Regelmenge R oder der importierten GTUs U ableitbar ist, d.h. es gibt Graphen $G_0, G_1, \ldots, G_n$ mit $G_0 = G$ und $G_n = G'$ und für $i = 1, \ldots, n : G_{i-1} \Rightarrow_r G_i$ für ein $r \in R$ (Ableitungsschritt mit Regel) oder $(G_{i-1}, G_i) \in SEM(t)$ für ein $t \in U$ (Ableitungsschritt mit importierter GTU),

3. diese Ableitung durch die Kontrollbedingung erlaubt ist, d.h. $(G, G') \in SEM_{E(trut)}(C)$.

Die Folge von Graphen in Punkt 2 heißt *Verschränkte Sequenz in trut von G nach G'* (original: *interleaving sequence*). Sei RIS_{trut} (*Relation of interleaving sequences*) die binäre Relation, welche durch die verschränkte Semantik gegeben ist. In RIS_{trut} stehen alle Graphpaare, die theoretisch durch Regeln aus R oder importierte GTUs aus U in beliebig vielen Schritten voneinander abgeleitet werden können. Das heißt, das

$$RIS_{trut} = (\Rightarrow_R \cup \bigcup_{t \in U} SEM(t))^*.$$

In dieser Relation stehen aber im Allgemeinen mehr Graphen, als tatsächlich in der verschränkten Semantik von *trut* enthalten sind, da in dieser Relation weder die Kontrollbedingung noch die Einschränkung auf initiale und terminale Graphen enthalten ist. $SEM(trut)$ besteht aus dem Schnitt von RIS_{trut} mit $SEM(I) \times SEM(T)$ und $SEM_{E(trut)}(C)$. Der Schnitt dieser drei Relationen kann aber sogar leer sein, was heißen würde, das keine Ableitungen möglich sind. Das ein Paar von Graphen (G, G') in der Semantik der Kontrollbedingung $SEM_{E(trut)}(C)$ enthalten ist, heißt nicht, dass es auch eine Ableitung von G nach G' gibt und umgekehrt.

Wie in Kapitel 4.1 erwähnt, werden GTUs rekursiv aufgebaut. Falls keine Units importiert werden, enthält die verschränkte Semantik nur Ableitungen durch die Regeln aus R:

$$SEM(trut) = \Rightarrow_R^* \cap (SEM(I) \times SEM(T)) \cap SEM_{E(trut)}(C).$$

Falls die Menge der initialen Graphen I nur einen einzelnen Graphen enthält, heißt die GTU laut Kapitel 15.2.3 aus [2] *sprachgenerierend*. In diesem Fall formen die durch die GTU ableitbaren Graphen, also die Graphen der zweiten Komponente der verschränkten Semantik, eine Graph-Sprache:

$$L(trut) = \{G \in SEM(T) \mid (I, G) \in SEM(trut)\}.$$

Jeder ableitbare Graph ist ein Wort dieser Graph-Sprache. Eine besondere Art der sprachgenerierenden Graphtransformationsunits sind die Graphgrammatiken. *trut* beschreibt eine Graphgrammatik, falls U leer ist (also keine Units importiert werden) und C gleich *true* ist (keine Einschränkungen durch die Kontrollbedingung). Die von der Unit erzeugte Sprache besteht aus allen terminalen Graphen:

$$L(trut) = \{G \in SEM(T) \mid I \Rightarrow_R^* G\}.$$

4.1.2 Beispiele für Graphklassen

Im Folgenden werden einige Beispiele für Graphklassen vorgestellt.

1. Ein einzelner Graph, oder eine endliche Menge von Graphen kann eine einfache Graphklasse sein. Die Semantik dieser Graphklasse ist trivialerweise die Menge der Graphen selber, d.h. $SEM(G) = \{G\}$.

2. Ein Graph ist bezüglich einer Regel oder einer Regelmenge reduziert, wenn keine weiteren Ableitungen durch diese Regel bzw. einer der Regeln aus der Menge möglich ist. Gibt es also ein $P \subseteq \mathcal{R}$, so ist die Graphklasse $SEM(P) = RED(P)$ die Menge der Graphen G, für die gilt, dass es keinen Graphen G' gibt, mit $G \Rightarrow_r G'$ für ein $r \in P$.

3. Ist Σ ein Alphabet für Knotenbeschriftungen in Graphen und $T \subseteq \Sigma$ eine Teilmenge davon. Dann ist $SEM(T)$ die Menge der Graphen, deren Knoten nur mit Elementen aus T beschriftet sind. Diese Art von Graphklassen wird auch in dem Beispielalgorithmus benötigt. Wie in Punkt 4 aus Kapitel 3.2 beschrieben, ist in diesem Fall das Alphabet $\Sigma = \{0, 1, 2\}$ und es gibt zwei Teilmengen: für die erste Graphklassenbeschreibung ist dies $T_1 = \{0\}$, für die zweite $T_2 = \{2\}$. In $SEM(T_1)$ sind also alle Graphen enthalten, deren Knoten mit 0 beschriftet sind und in den Graphen in $SEM(T_2)$ sind alle Knoten mit 2 beschriftet.

4.1.3 Beispiele für Kontrollbedingungen

Der Ableitungsprozess bietet viele Möglichkeiten der Regelanwendung an vielen verschiedenen Stellen im Graphen in verschiedenen Reihenfolgen. Dieser Nichtdeterminismus wird durch die Kontrollbedingung eingeschränkt. Ein Beispiel ist die Einschränkung auf Ableitungen, in denen die Regeln in einer bestimmten Reihenfolge angewendet werden, so dass das Hintereinanderschreiben der Regelbezeichner Wörter aus einer Kontrollsprache ergibt. Diese Kontrollsprache kann durch eine reguläre Sprache über den Regelbezeichnern gebildet werden. Eine solche wird auch im Beispiel (Kapitel 4.4) verwendet. Weitere Beispiele für Kontrollbedingungen sind:

1. Eine einfache Kontrollbedingung lautet: „wende die Regeln aus R so lange wie möglich an“.

2. Sprachkonstrukte wie Verzweigungen und Schleifen in einer Graphtransformationssprache wie PROGRESS dienen als Kontrollbedingung.

3. Prioritäten zwischen den einzelnen Regeln einer GTU dienen auch als Kontrollbedingungen.

4. Sei $E : ID \rightarrow 2^{\mathcal{G}\times\mathcal{G}}$ eine Umgebung (ID steht hier für die Menge der Bezeichner). Eine Umgebung ist die Abbildung, die einem Bezeichner eine Kontrollbedingung (eine Menge von Graphpaaren) zuordnet. E kann zur Menge der Sprachen über der Bezeichnermenge ID erweitert werden. Dies ist die Potenzmenge der Wörter über ID. $\widehat{E} : 2^{ID^*} \rightarrow 2^{\mathcal{G}\times\mathcal{G}}$ bildet eine Sprache auf die dadurch beschriebene Kontrollbedingung ab und wird für eine Sprache $L \subseteq ID^*$ definiert als $\widehat{E}(L) = \bigcup_{w\in L} \overline{E}(w)$. $\overline{E} : 2^{ID^*} \rightarrow 2^{\mathcal{G}\times\mathcal{G}}$ bildet ein Wort auf die durch das Wort beschriebene Kontrollbedingung ab. $\overline{E}$ wird rekursiv definiert als $\overline{E}(\lambda)$[1] $= \Delta\mathcal{G}$[2] und $\overline{E}(x, v) = E(x) \circ \overline{E}(v)$ für ein $x \in ID$ und ein $v \in ID^*$.

   E bildet also einen Buchstaben auf die dadurch beschriebene Konrollbedingung ab. $\overline{E}$ erhält ein Wort als Eingabe und bildet es auf die durch dieses Wort beschriebene Kontrollbedingung ab. $\widehat{E}$ erhält eine Menge von Wörtern, also eine Sprache, als Eingabe und bildet diese ab auf die Vereinigung der durch die einzelnen Wörter beschriebenen Kontrollbedingungen.

   Die Semantik einer Sprache wird also definiert über die Wörter der Sprache und deren Aufbau aus Buchstaben, welche z.B. einzelne Regeln repräsentieren

[1] λ ist das leere Wort.
[2] $\Delta\mathcal{G}$ steht für die identische Relation über $\mathcal{G}$, also alle Paare (G, G).

können. Auf diese Weise kann die Wortmenge L als Kontrollbedingung benutzt werden mit $SEM_E(L) = \widehat{E}(L)$. Diese Art von Kontrollbedingungen sind vom 'Typ Sprache'.

5. Die im letzten Punkt beschriebenen Kontrollbedingungen vom Typ Sprache können durch jede Grammatik, jeden Automat oder durch reguläre Ausdrücke beschrieben werden, die diese Sprache erzeugen/erkennen. Im Ableitungsprozess ist das Anwenden einer Regel nur dann erlaubt, wenn der Regelbezeichner durch die Grammatik/den regulären Ausdruck erlaubt ist bzw. durch den Automaten erzeugt oder erkannt wird (je nach Typ des Automaten).

4.2 Anwendungssequenz

In einer verschränkten Sequenz werden in einer bestimmten Reihenfolge Regeln aus R angewendet und importierte GTUs aufgerufen. Die Folge der Regelbezeichner wird *Anwendungssequenz* genannt. Dies bedeutet formal: Sei $G_0, \ldots, G_n$ eine verschränkte Sequenz von G nach G' ($G = G_0$ und $G' = G_n$) und für $i = 1, \ldots, n$ gilt $G_{i-1} \Rightarrow_{x_i} G_i$ und $x_i \in R$ bzw. $SEM(G_{i-1}, G_i) \in SEM(x_i)$, falls $x_i \in U$. Dann ist $x_1 \ldots x_n$ die Anwendungssequenz von (G, G'). Ist $n = 0$, so ist die Anwendungssequenz das leere Wort λ.

4.3 Eigenschaften der Anwendungssequenz

Sei 2^{ID^*} die Menge der Kontrollbedingungen vom Typ Sprache. Sei $trut = (I, U, R, L, T)$ mit der Kontrollbedingung $L \subseteq (U \cup R)^* \subseteq ID^*$. Dann gilt für alle $G, G' \in \mathcal{G}$, dass folgende Aussagen äquivalent sind.

1. $(G, G') \in SEM(trut)$
2. $(G, G') \in SEM_{E(trut)}(L) \cap (SEM(I) \times SEM(T))$
3. Es gibt eine Anwendungssequenz w von (G, G') mit $w \in L$ und $(G, G') \in (SEM(I) \times SEM(T))$

Dies bedeutet, dass folgende Aussagen das gleiche bedeuten:

1. Das Graphpaar (G, G') ist in der verschränkten Semantik der GTU.
2. (G, G') ist in der Semantik der Kontrollbedingung und G ist ein initialer Graph und G' ist ein terminaler Graph.
3. Es gibt eine Anwendungssequenz w von (G, G'), welche in der Kontrollsprache L ist und G ist ein initialer und G' ein terminaler Graph.

Beweis: Sei $(G, G') \in SEM(trut)$. Dann gibt es laut Definition eine verschränkte Sequenz in $trut$ von G nach G', also ist $(G, G') \in SEM_{E(trut)}(L)$. Außerdem muss laut Definition der verschränkten Semantik (Kapitel 4.1.1) $(G, G') \in (SEM(I) \times SEM(T))$ gelten. Also impliziert Punkt 1 den Punkt 2.

Für die weiteren Schritte wird zunächst folgende Behauptung bewiesen:

$(G, G') \in SEM_{E(trut)}(L)$ genau dann,
wenn es eine Anwendungssequenz $w \in L$ von (G, G') gibt.

Da *trut* vom Typ Sprache ist, gilt per Definition $(G, G') \in SEM_{E(trut)}(L) = \widehat{E(trut)}(L)$[3]. Dies ist genau dann der Fall, wenn $(G, G') \in \overline{E(trut)}(w)$ ist für ein Wort $w \in L$ in der Kontrollsprache. Die Ableitung von G nach G' ist also durch die Kontrollbedingung w möglich. Es wird nun per Induktion über den Aufbau von w gezeigt, dass $(G, G') \in \overline{E(trut)}(w)$ genau dann gilt, wenn w eine Anwendungssequenz von (G,G') ist.

Induktionsanfang $w = \lambda$: $(G, G') \in \overline{E(trut)}(\lambda)$ genau dann, wenn $(G, G') \in \Delta G$, das bedeutet, dass $G = G'$. Genau dann ist λ eine Anwendungssequenz von (G,G'), da keine Regel angewendet werden muss.

Induktionsschritt $w = xv$: Gelte die Behauptung für v. Sei $x \in (U \cup R)$ ein Bezeichner für eine Regel oder importierte GTU, dann ist w Anwendungssequenz von (G, G'), das heißt

$$(G, G) \in \overline{E(trut)}(xv) = E(trut)(x) \circ \overline{E(trut)}(v),$$

genau dann, wenn es einen Graphen $\overline{G} \in \mathcal{G}$ gibt, mit

$$(G, \overline{G}) \in E(trut)(x) \text{ und} \tag{1}$$

$$(\overline{G}, G') \in \overline{E(trut)}(v). \tag{2}$$

Gleichung 2 bedeutet laut Induktionsvoraussetzung, dass v Anwendungssequenz von $(\overline{G}, G')$ ist und es eine verschränkte Sequenz $G_0, \ldots, G_n$ gibt, mit $G_0 = \overline{G}$ und $G_n = G'$. Gleichung 1 bedeutet, dass $\overline{G}$ von G durch einen Ableitungsschritt mit der Regel x oder importierten GTU x erreicht werden kann, also dass $G \Rightarrow_x \overline{G}$ falls $x \in R$ bzw. $(G, \overline{G}) \in SEM(x)$, falls $x \in U$. Zusammen ergibt sich die verschränkte Sequenz $G, G_0, \ldots, G_n$ für die entspechende Anwendungssequenz $xv = w$.

In der Gegenrichtung des Beweises ist die Anwendungssequenz xv für (G, G') mit einer entsprechenden verschränkten Sequenz $G_0, \ldots, G_n$ mit $G_0 = G$ und $G_n = G'$ gegeben. Es gilt $G_0 \Rightarrow_x G_1$, falls $x \in R$ bzw. $(G_0, G_1) \in SEM(x)$, falls $x \in U$. Auf jeden Fall gilt $(G, G_1) \in E(trut)(x)$, unabhängig davon, ob die Graphen durch eine Regel der GTU oder durch eine importierte Unit abgeleitet wurden. Da v eine Anwendungssequenz von (G_1, G_n) mit der entsprechenden verschränkten Sequenz $G_1, \ldots, G_n$ ist, gilt laut Induktionsvoraussetzung, dass $(G_1, G_n) \in \overline{E(trut)}(v)$. Das Zusammensetzen dieser zwei Ableitungssequenzen ergibt, dass $(G, G') \in E(trut)(x) \circ \overline{E(trut)}(v) = \overline{E(trut)}(xv)$. Aus der bewiesenen Behauptung folgt direkt, dass Punkt 3 aus Punkt 2 folgt.

Für den Beweis, dass Punkt 1 aus Punkt 3 folgt, sei w eine Anwedungssequenz von (G, G') und (G, G') sei in $SEM(I) \times SEM(T)$. Dann gibt es laut Definition der Anwendungssequenz eine verschränkte Sequenz von G nach G'. Da die Anwendungssequenz w in der Kontrollsprache L ist, gilt $(G, G') \in SEM_{E(trut)}(L)$. Da auch $(G, G') \in (SEM(I) \times SEM(T))$ ist, gilt insgesamt, das $(G, G') \in SEM(trut)$.

□

4.4 Spezifikation des Algorithmus von Floyd mit einer GTU

Der Algorithmus von Floyd arbeitet auf gerichteten Graphen mit Kantengewichten. Zusätzlich benötigt der Algorithmus Knotenbeschriftungen aus dem Alphabet

[3]Definition von $\overline{E}$ und $\widehat{E}$ siehe Beispiel 4 in Kapitel 4.1.3

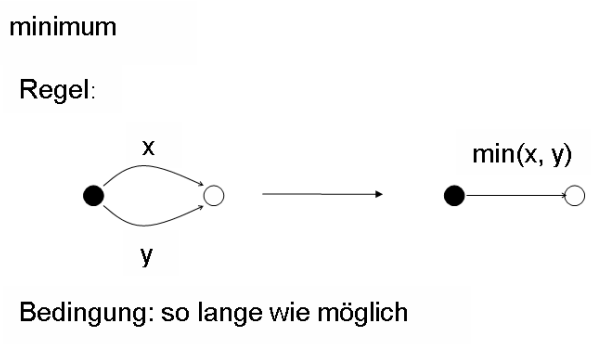

Abbildung 6: Die GTU *minimum*

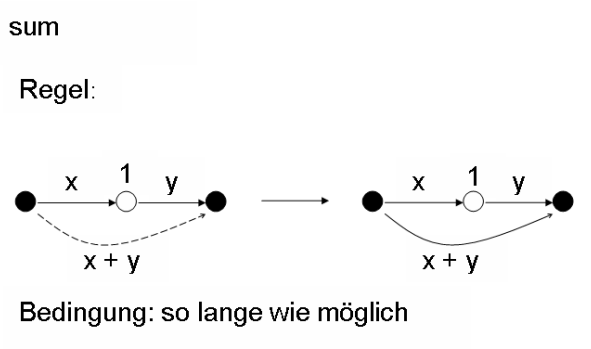

Abbildung 7: Die GTU *sum*

$\{0, 1, 2\}$. Die Menge der initialen Graphen ist $SEM(Knotenbeschriftung\,0)$, also Graphen, deren Knoten alle mit 0 beschriftet sind. Ein Knoten hat die Beschriftung 0, wenn über diesen noch keine Wege berechnet wurden. Wurde die Beschriftung eines Knotens in 1 geändert, werden Wege über diesen Knoten berechnet. Wurden *alle* Wege über diesen Knoten berechnet, d.h. die Gewichte der Wege über diesen Knoten addiert, so wird die Beschriftung dieses Knotens auf 2 gesetzt. Die Menge der terminalen Graphen ist $SEM(Knotenbeschriftung\,2)$, d.h. der Algorithmus endet erst, wenn alle Knoten mit 2 beschriftet sind.

Die Transformation Unit *shortest-path* benutzt die Units *minimum*, *sum*, *change(0,1)* und *change(1,2)*:

Die Bedeutung der einzelnen importierten Units:

minimum Diese Funktion sucht parallele Kanten und löscht diejenige mit dem größten Kantengewicht. Dies wird so lange wie möglich gemacht, also bis keine parallelen Kanten mehr im Graphen vorhanden sind, (siehe Abbildung 6).

sum Summiert die Gewichte zweier aufeinanderfolgender Kanten, falls der dazwischen liegende Knoten mit 1 beschriftet ist, und bildet eine neue Kante, deren Gewicht gleich dieser Summe ist. Bedingung dafür ist, dass diese Kante noch nicht existiert, was im Bild durch die gestrichelte Kante dargestellt wird. Dies ist eine *negative Kontextbedingung* für die Anwendung dieser GTU (siehe Abbildung 7).

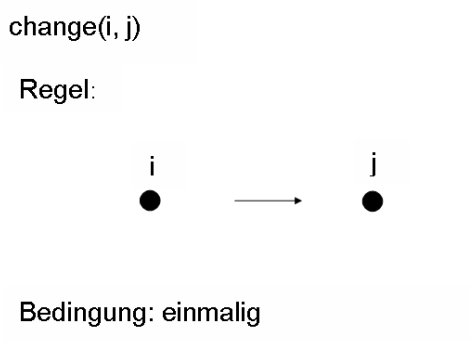

Abbildung 8: Die GTU *change(i,j)*

change(0,1) Ändert die Beschriftung eines mit 0 beschrifteten Knotens nach 1.

change(1,2) Ändert die Beschriftung eines Knotens (des einen Knotens, der mit 1 beschriftet ist) von 1 nach 2 (Bild zu diesen beiden Units siehe Abbildung 8).

Die Kontrollbedingung bestimmt die Reihenfolge, in welcher die Regeln angewendet werden. Dies geschieht durch den folgenden regulären Ausdruck über den Regelbezeichnern:

$$minimum; (change(0,1); sum; change(1,2); minimum)^*$$

Der Algorithmus entfernt also zu Beginn alle parallelen Kanten. Danach wird folgender Vorgang solange durchgeführt, bis alle Knoten mit 2 beschriftet sind: Zuerst wird ein beliebiger Knoten von 0 nach 1 umbeschriftet. Es werden dann Wege der Länge 2 über diesen Knoten gesucht und jeweils eine neue Kante vom Anfang zum Ziel eines Weges erstellt, welche das Gewicht der Summe der beiden Kantengewichte hat. Durch diesen Schritt entstehen evtl. parallele Kanten, welche unterschiedliche Gewichte haben können. Sind alle möglichen Wege bearbeitet worden, wird der Knoten mit der Beschriftung 1 nach 2 umbeschriftet. Zum Schluss werden parallele Kanten wieder entfernt und falls noch mit 0 beschriftete Knoten im Graph vorhanden sind, wird dieser Teil des Algorithmus wiederholt. Zum Schluss sind alle Knoten des Graphen mit 2 beschriftet und die kürzesten Wege für alle Knotenpaare im Graphen wurden berechnet.

5 Korrektheit und Laufzeit des Algorithmus

In diesem Kapitel werden die Korrektheit und die Laufzeit der vorgestellten GTUs gezeigt.

5.1 Beweis der Korrektheit

Die Korrektheit des Algorithmus aus Kapitel 4.4 wird über die verschränkte Semantik der einzelnen am Algorithmus beteiligten GTUs gezeigt. Ziel ist es, zu zeigen, dass die verschränkte Semantik der GTU *shortest-path* der Graph mit den kürzesten Wegen des Initialgraphen ist. Die Elemente eines gegebenen Graphen $G = (V, E, s, t, l, dist)$ werden im Folgenden auch mit $V_G, E_G, s_G, t_G, l_G, dist_G$ bezeichnet.

1. Ein Paar von Graphen (G, G') ist in der Semantik von $change(i, j)$ genau dann, wenn sich die beiden Graphen nur in der Beschriftung genau eines Knotens unterscheiden.

   $(G, G') \in SEM(change(i, j))$ genau dann, wenn $V_G = V_{G'}, E_G = E_{G'}, s_G = s_{G'}, t_G = t_{G'}, dist_G = dist_{G'}$ und es gibt genau einen Knoten v mit $l_G(v) = i$ und $l_{G'}(v) = j$. Für alle anderen Knoten $v' \neq v$ gilt: $l_G(v') = l_{G'}(v')$.

2. Die GTU *minimum* sorgt dafür, dass parallele Kanten aus dem Graphen entfernt werden. Es wird die Kante mit dem größeren Gewicht entfernt. Die Kontrollbedingung *so lange, wie möglich* sorgt dafür, dass der Graph nach der Anwendung dieser GTU keine parallelen Kanten besitzt, also ein einfacher Graph ist. Es ergibt sich folgende verschränkte Semantik für *minimum*,:

   Sei $E_G(v, v')$ die Menge der Kanten von v nach v' im Graphen G. $(G, G') \in SEM(minimum)$ genau dann, wenn $V_G = V_{G'}, l_G = l_{G'}$ und für alle Knotenpaare $(v, v') \in V_G$ mit $E_G(v, v') \neq \emptyset$ gilt, dass $|E_{G'}(v, v')| = 1$ und $dist_{G'}(e_0) = min\{dist_G(e) | e \in E_G(v, v')\}$ für alle Kanten e_0 in G'.

   Gibt es also in G mindestens eine Kante zwischen v und v', so gibt es in G' genau eine Kante zwischen diesen Knoten mit dem minimalen Gewicht aller Kanten in G zwischen diesen Knoten.

3. Wird die GTU *sum* angewendet, so wird eine neue Kante zwischen v und v' mit dem Gewicht $x + y$ erstellt, falls es nicht bereits eine solche Kante gibt und es eine Kante mit dem Gewicht x von v nach $\overline{v}$ und eine Kante mit dem Gewicht y von $\overline{v}$ nach v' gibt. Auch dies wird wegen der Kontrollbedingung *so lange wie möglich* iteriert, bis keine Summierungen mehr möglich sind. Daraus ergibt sich folgende verschränkte Semantik:

   $(G, G') \in SEM(sum)$ genau dann, wenn $V_G = V_{G'}, l_G = l_{G'}, E_G \subseteq E_{G'}, s_G = s_{G'}|_{E_G}, t_G = t_{G'}|_{E_G}, dist_G = dist_{G'}|_{E_G}$[4]. Außerdem gilt für alle Knotenpaare $v, v' \in V_G$ mit einer neu berechneten Kante $e_0 \in E_{G'}(v, v') - E_G(v, v')$, dass ihr Gewicht $dist_{G'}(e_0) = c$ genau dann, wenn es einen Knoten $\overline{v} \in V_G$ gibt. Für diesen Zwischenknoten gilt, dass $l_G(\overline{v}) = 1$ ist und es gibt die Kanten $e \in E_G(v, \overline{v})$ und $e' \in E_G(\overline{v}, v')$ über diesen Knoten mit $dist_G(e) + dist_G(e') = c$. Außerdem gibt es keine weitere Kante $\overline{e} \in E_G(v, v')$ mit $dist_G(\overline{e}) = c$.

4. Die verschränkte Semantik von *shortest-path* kann auf die bereits beschriebenen Semantiken zurückgeführt werden. Dass ein Paar von Graphen $(G, G') \in SEM(shortest - path)$ ist, bedeutet, dass es eine verschränkte Sequenz $G, G_0, \ldots, G_{4n}$ mit $n \in \mathbb{N}$ und $G_{4n} = G'$ gibt. Die Bezeichnung $4n$ bzw. $4i$ wurde gewählt, da die Schleife der Kontrollbedingung die Länge 4 hat. Diese verschränkte Sequenz hat folgende Eigenschaften (für $i = 0, \ldots, n - 1$):

   (a) $(G, G_0) \in SEM(minimum)$,

   (b) $(G_{4i}, G_{4i+1}) \in SEM(change(0, 1))$,

   (c) $(G_{4i+1}, G_{4i+2}) \in SEM(sum)$,

   (d) $(G_{4i+2}, G_{4i+3}) \in SEM(change(1, 2))$,

[4] Die Beschränkung einer Funktion $f : A \to B$ auf eine Menge $C \subseteq A$ wird mit $f|_C$ bezeichnet.

(e) $(G_{4i+3}, G_{4i+4}) \in SEM(minimum)$

Wie schon in der Beschreibung der verschränkten Semantik der anderen GTUs gesehen, bleibt die Knotenmenge V_G unverändert.

Für die Knotenbeschriftungen $l(v)$ gilt folgendes: Im initialen Graphen G ist $l_G(v) = 0$ für alle $v \in V_G$, außerdem ist $l_G = l_{G_0}$. Da die GTUs *sum* und *minimum* die Knotenbeschriftung nicht ändern, gilt für $i = 0, \ldots, n-1$ $l_{G_{4i+1}} = l_{G_{4i+2}}$ und $l_{G_{4i+3}} = l_{G_{4i+4}}$. Da ein Knoten von den *change*-GTUs neu beschriftet wird, gibt es einen Knoten $v_{i+1} \in V_{G_{4i}}$ (über welchen die Pfade gehen, deren Gewichte summiert werden) mit $l_{G_{4i}}(v_{i+1}) = 0, l_{G_{4i+1}}(v_{i+1}) = 1, l_{G_{4i+3}}(v_{i+1}) = 2$ und es gilt für alle anderen Knoten $v \neq v_{i+1}$, dass $l_{G_{4i}}(v) = l_{G_{4i+1}}(v)$ und $l_{G_{4i+2}}(v) = l_{G_{4i+3}}(v)$.

$G_{4n} = G'$ ist ein terminaler Graph, also ist $l_{G_{4n}}(v) = 2$ für alle Knoten $v \in V_{G_{4i}} = V_{G'} = V_G$. Also ist n gleich der Anzahl der Knoten, da in jedem Schleifendurchlauf nur ein einziger Knoten eine neue Beschriftung erhält.

Sei V_j die Menge der ersten j Knoten, d.h. $V_0 = \emptyset$ und $V_j = \{v_1, \ldots, v_j\}$ für $1 \leq j \leq n$. G_{4j} ist der Graph, in welchem die Wege über diese Knoten berechnet wurden. Die Knoten aus V_j sind in diesem also bereits mit 2 markiert. Dann gilt, dass in G_{4j} genau dann eine Kante von v nach v' mit dem Gewicht c existiert, wenn der kürzeste Weg von v nach v' in G über die Knoten in V_j das Gewicht c hat. D.h. es gilt für $j = 0, \ldots, n$ und alle Knotenpaare $v, v' \in V_{G_{4j}}$ mit $v \neq v'$ und ein $c \in \mathbb{N}$:

$e_0 \in E_{G_{4j}}(v, v')$ mit $dist_{G_{4j}}(e_0) = c$ genau dann, wenn der kürzeste Weg von v nach v' in G ohne die Knoten aus $V_G - V_j$ zu benutzen das Gewicht c hat.[5] Für $j = 0$ sind dies also nur die direkten Verbindungen über Kanten in G. Ist $1 \leq j < n$ enthält G_{4j} die Knoten, welche bereits mit 2 beschriftet wurden und über welche die Wege berechnet wurden. Für $j = n$ ergibt diese Behauptung genau die gewünschte Eigenschaft der verschränkten Semantik der GTU $shortest-path$, die kürzesten Wege in G zu berechnen:

$(G, G') \in SEM(shortest-path)$ genau dann, wenn $V_G = V_{G'}, l_G(v) = 0$ und $l_{G'} = 2$ für alle Knoten $v \in V_G$. Außerdem ist G' ein einfacher Graph und für alle Knoten $v, v' \in V_G, v \neq v'$ gilt: falls es eine Kante $e_0 \in E_{G'}(v, v')$ mit $dist_{G'}(e_0) = c$ gibt, so ist c das Gewicht des kürzesten Pfades von v nach v'.

5.2 Komplexität

Die Komplexität des Algorithmus ergibt sich aus der Länge der Ableitung eines Graphen. Sei im Folgenden m die größte Anzahl paralleler Kanten zwischen zwei Knoten im Graphen $G \in SEM(I)$. Diese werden zu Beginn des Algorithmus entfernt, also ist der erste Summand der Komplexität m. Sei n die Anzahl der Knoten. Dann durchläuft der Algorithmus die 'äußere Schleife' (den Teil in der Kontrollbedingung innerhalb der Klammer) n mal. Innerhalb der Schleife werden folgende Schritte durchgeführt:

1. $change(0, 1)$ wird genau einmal durchgeführt

[5] Der Beweis benutzt vollständige Induktion über die Anzahl der Knoten in V_j und steht in [2].

2. *sum* wird höchstens $(n-1)*(n-2)$ durchgeführt, da ein Knoten zu höchstens $n-1$ Knoten adjazent ist und keine Zyklen erstellt werden.

3. $change(1,2)$ wird genau einmal durchgeführt

4. *minimum* wird höchstens $(n-1)*(n-2)$ mal ausgeführt, denn so viele parallele Kanten können maximal durch *sum* erstellt werden.

Insgesamt summiert sich dies zu

$$\begin{aligned} & m + n * [1 + (n-1)(n-2) + 1 + (n-1)(n-2)] \\ & = m + 2n + 2n(n-1)(n-2) \in O(n^3 + m). \end{aligned}$$

Ist also die Anzahl der parallelen Kanten in G sehr groß im Vergleich zur Anzahl der Knoten, so bestimmt m die Laufzeit des Algorithmus. Sonst ist n^3 der Summand mit der größeren Auswirkung auf die Laufzeit. Die Laufzeit wird aber nur im schlechtesten Fall (bei einem vollständigen Graphen) so groß, da dann n mal die Gewichte der eingehenden Kanten von $n-1$ Knoten mit den Gewichten von $n-2$ ausgehenden Kanten summiert werden müssen. Die Laufzeit dieser GTU ist somit gleich der für diesen Algorithmus minimal möglichen Laufzeit.

6 Graphtransformationsmodule

Im Allgemeinen gibt es GTUs, welche von direktem Nutzen für den Anwender sind, während andere nicht direkt aufgerufen werden, sondern nur von anderen benötigt werden. Im Beispiel des Shortest-Path Algorithmus von Floyd ist die Funktion *shortest-path* für den Anwender interessant - die anderen GTUs werden nur zur Berechnung benötigt. Es liegt nahe, diese Menge von GTUs zu einer Einheit zusammenzufassen, da sie alle zusammen für einen korrekten Lauf des Algorithmus benötigt werden. Zu diesem Zweck gibt es Graphtransformationsmodule (kurz: *GTM*, original: *Graph Transformation Modules*). Diese fassen mehrere GTUs zusammen, wobei manche für den Export, d.h. für den Aufruf von außen gedacht sind und die restlichen versteckt werden. Es ist aber auch möglich, das eine GTM eine andere GTM importiert und deren Funktionen benutzt. Im Beispiel könnte die Funktion *shortest-path* z.B. von einem Routenplaner benutzt werden.

Zur Definition der GTM werden *formale Parameter Units* benötigt. Eine GTU f ist eine *formale Parameter Unit*, wenn $U_f = \emptyset$, $R_f = \emptyset$ und $C_f = true$. Da die Regelmenge leer ist und auch keine GTUs importiert werden, berechnen diese GTUs nichts, sondern stellen nur die Relation $SEM(I_f) \times SEM(T_f)$ zur Verfügung. Diese kann benutzt werden, um die Semantik einer *normalen* GTU n zu beschränken: $SEM(n) \subseteq SEM(I_f) \times SEM(T_f)$.

Ein GTM ist ein Tripel von Mengen, welche GTUs enthalten

$$MOD = (IMPORT, BODY, EXPORT),$$

1. $IMPORT$ ist eine Menge von formalen Parameter Units. Diese beschreiben die importierten GTUs, welche von den GTUs aus $BODY$ benutzt werden können.

2. $BODY$ ist die Menge der in diesem Modul implementierten GTUs. Jede dieser Units importiert nur GTUs aus $IMPORT$ und $BODY$.

3. $EXPORT$ ist eine Teilmenge der GTUs aus $IMPORT$ und $BODY$. Diese sind von außen benutzbar. Alle anderen GTUs sind von außen nicht sichtbar.

Bevor das Modul benutzt werden kann, müssen die formalen Parameter Units aus $IMPORT$ instanziiert werden. Dies geschieht mit expliziten GTUs. Deren verschränkte Semantik ist eine Teilmenge der Semantik der formalen Parameterunit, also von $(SEM(I_f) \times SEM(T_f))$. Sie dürfen kein Graphpaar in ihrer Semantik haben, welches nicht in der Semantik der formalen Parameter Unit ist. Anders ausgedrückt: für die explizite GTU muss gelten, das für jede Ableitung von G nach G' $G \in SEM(I_f)$ und $G' \in SEM(T_f)$ ist. Bei der Implementierung der GTUs in dem Body des Moduls, können die GTUs aus der Import-Schnittstelle schon benutzt werden bevor diese instanziiert sind, da eine Obermenge der verschränkten Semantik von möglichen expliziten GTUs bekannt ist. Die formalen Parameter Units können zum Beispiel mit Units instanziiert werden, welche von einem anderen Modul exportiert werden. So können mehrere Module aufeinander aufbauen.

Die Semantik des GTM ist gleich der Semantik der GTU aus $IMPORT$ und $BODY$, welche auf die Menge $EXPORT$ beschränkt wird. Dies bedeutet, dass kein Zugriff auf die Module möglich ist, die in $BODY$ aber nicht in $EXPORT$ eingetragen sind.

Der Beispielalgorithmus kann wie folgt in einem GTM zusammengefasst werden:

IMPORT $\emptyset$

BODY $shortest-path, minimum, sum, change(0,1), change(1,2)$

EXPORT $shortest-path$

Die GTUs, welche nicht in der Export-Menge stehen $(minimum, sum, change(0,1), change(1,2))$, werden vor der Umgebung „versteckt“, nur $shortest-path$ wird exportiert. Da diese GTUs ohne weitere arbeiten können, werden keine anderen GTUs importiert.

7 Zusammenfassung

In dieser Arbeit wurden Graphtransformationsunits vorgestellt, mit deren Hilfe Graphtransformationen definiert werden können. Es wurde beschrieben, wie diese auf einem Graphtransformationsansatz aufgebaut werden und in einem Graphtransformationsmodul zusammengefasst werden können. Am Beispiel des Algorithmus von Floyd wurde gezeigt, wie GTUs arbeiten. Es wurde deutlich, dass GTUs nicht eindeutig sondern nichtdeterministisch arbeiten. Obwohl der Nichtdeterminismus durch die Kontrollbedingung eingeschränkt wird, bleiben unter Umständen mehrere Möglichkeiten eine Regel auszuwählen, welche an verschiedenen Stellen im Graphen angewendet werden kann.

Interessant für die Implementierung eines Graphtransformationsmoduls sind formale Parameterunits, welche es den GTUs innerhalb des Moduls erlauben, eine andere GTU zu benutzen, bevor diese implementiert wurde. Die in dieser Arbeit vorgestellten Begriffe, Graphtransformationsansatz, -unit und -modul, können als Grundlage für konkrete Graphersetzungssysteme verwendet werden. Außerdem können die Laufzeit und Korrektheit von GTUs durch die verschränkte Semantik analysiert werden.

Literatur

[1] G. Rozenberg, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1*, 1997, World Scientific Publishing Co. Pte. Ltd., ISBN: 9810228848

[2] H. Ehrig, G. Engels, H.-J. Kreowski, G. Rozenberg, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2*, 1997, World Scientific Publishing Co. Pte. Ltd., ISBN: 9810228848