UNIKASSEL VERSITÄT | ELEKTROTECHNIK INFORMATIK

Jens Kosiol

Graph- and Model-Driven Engineering
Übungsblatt 1 – Lösungsskizze

1 Graphtransformationssystem (GTS) aus Code (10 Punkte)

Gegeben sei die Java-Implementierung einer Liste (Chain) aus Listing 1.[1] Entwerfen Sie, analog zum Beispiel für zirkuläre Puffer aus der Vorlesung und eines Stack aus der Übung, ein getyptes Graphtransformationssystem, das diese Datenstruktur modelliert:

a) Geben Sie einen passenden Typgraphen und Regeln, die den Methoden insert und deleteNext entsprechen, an.

b) Erstellen Sie einen sinnvollen Startgraphen für das Graphtransformationssystem, der sich aus dem Konstruktor für Chain ergibt.

c) Erstellen Sie eine Regel, die der Methode isEmpty entspricht, und erläutern Sie, wie die Anwendung der Regel als Anwendung der Methode isEmpty zu interpretieren ist.

Geben Sie außerdem einen Graphen an, der mit Hilfe Ihrer Regeln aus Ihrem Startgraph *nicht* abgeleitet werden kann.

```
public class Chain<T> {
    Cell<T> anchor;

    Chain() { this.anchor = new Cell<T>(); }

    public Boolean isEmpty() {
        return this.anchor.next == null;
    }
}

public class Cell<T> {
    T content;
    Cell<T> next;

    Cell(){};
```

[1] Inspirationsquelle für den Code: Heinz-Peter Gumm und Manfred Sommer: *Einführung in die Informatik*. Bd 1: *Programmierung, Algorithmen und Datenstrukturen* (De Gruyter Oldenbourg 2016).

Graph- and Model-Driven Engineering
Übungsblatt 1 – Lösungsskizze

```
    Cell(T content, Cell<T> next) {
        this.content = content;
        this.next = next;
    }

    public void insert(T content) {
        this.next = new Cell<T>(content, this.next);
    }

    public void deleteNext() {
        if (this.next != null) {
            this.next = this.next.next;
        }
    }
}
```

Listing 1: Java-Implementierung einer Chain.

Lösung

Lösungsvorschlag in der beigefügten gps-Datei Ueb01_Chain.

2 Regelsatz für ein adaptiertes Behälterproblem (12 Punkte)

In dem Behälterproblem geht es grundsätzlich darum, zu entscheiden, ob es möglich ist, eine gegebene Anzahl von Objekten bestimmter Größen so auf k Behälter zu verteilen, dass keiner der Behälter überläuft; wir betrachten hier eine Variante, wo jeder Behälter seine eigene Größe hat. Formal: Gegeben sind k Behälter jeweils der Größe $b_j \in \mathbb{N}$ für $j = 1, \dots, k$ und n Objekte der Größen $a_i \in \mathbb{N}$ für $i = 1, \dots, n$. Gesucht ist eine Zuordnung $f\colon \{1, \dots, n\} \to \{1, \dots, k\}$, sodass für jedes $j \in \{1, \dots, k\}$ gilt:

$$\sum_{f(i)=j} a_i \leq b_j.$$

Jens Kosiol

Graph- and Model-Driven Engineering
Übungsblatt 1 – Lösungsskizze

Entwerfen Sie in Groove einen Typgraphen, einen Startgraphen und eine Menge von Graphtransformationsregeln, die das Einfügen von Gegenständen in Behälter, das Entfernen von Gegenständen aus Behältern und das Verschieben von Gegenständen zwischen Behältern modellieren.

Die folgenden Bedingungen sollen hierbei erfüllt sein:

- Der Startgraph soll aus drei Behältern mit den Volumen 50 (2×) und 75 bestehen. Des weiteren sind 10 Objekte mit den Gewichten 8 (2×), 12 (2×), 17 (2×), 22, 25 und 27 (2×) gegeben.
- Die Regeln sollen so gestaltet sein, dass ein Gegenstand nie zwei Behältern gleichzeitig zugeordnet sein kann und Behälter nicht überladen werden können.

Wichtig: Entwerfen Sie einen Typgraphen und aktivieren Sie diesen. Beachten Sie, dass Sie keinen Algorithmus zur Lösung des Behälterproblems entwerfen müssen, sondern lediglich Regeln zum Hinzufügen, Entfernen und Verschieben von Gegenständen, also Regeln, die innerhalb eines solchen Algorithmus eingesetzt werden könnten.

Lösung

Lösungsvorschlag in der beigefügten gps-Datei Ueb01_Behaelterproblem.

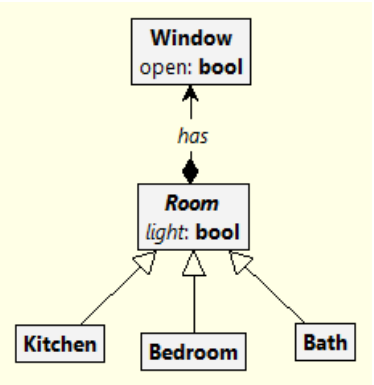
3 Graphtransformationssysteme mit Vererbung (8 Punkte)

Abbildung 1 zeigt den Typgraphen, Startgraphen und Transformationsregeln eines getypten Graphtransformationssystems mit Vererbung.

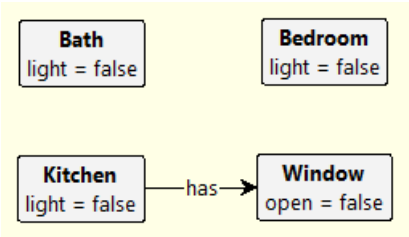
a) Spezifizieren Sie *formal* das getypte Graphtransformationssystem, welches aus dem Typgraph *TG*, dem Startgraph *SG* und der Regel *FensterZu* besteht. Geben Sie also textuell alle Tupel, Mengen und Morphismen an, welche das Graphtransformationssystem definieren. Nutzen Sie ggfs. Indizes, um gleichnamige Elemente unterschiedlicher Graphen zu unterscheiden (bspw. $Kitchen_{TG}$, $Kitchen_{SG}$, $Kitchen_{R}$ usw.).

Graph- and Model-Driven Engineering
Übungsblatt 1 – Lösungsskizze

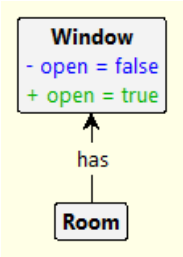
b) Klopfen Sie den Typgraphen in Bezug auf die darin enthaltene Vererbungshierarchie flach. Entfernen Sie also jegliche Vererbung und passen Sie Objekte und Kanten so an, dass keine Informationen verloren gehen. Überlegen und begründen Sie zudem, wieviele Regeln benötigt werden, um das Verhalten der in Abbildung 1 dargestellten Regeln für den angepassten Typgraphen zu realisieren. Legen Sie Ihrer Lösung den angepassten Typgraphen als Zeichnung oder Grafik bei.



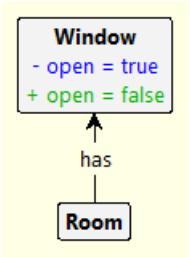
(a) TG



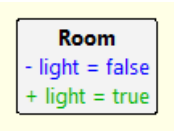
(b) SG



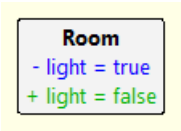
(c) FensterAuf



(d) FensterZu



(e) LichtAn



(f) LichtAus

Abbildung 1: Typgraph, Startgraph und Regeln eines GTS.

Lösung

a) Der Typgraph mit Vererbung $TG = (T, I, A)$ besteht aus einem erweiterten Graphen $T = (N_T, E_T, D_T, A_T, s_{E_T}, t_{E_T}, s_{A_T}, t_{A_T})$, einer partiellen Ordnung $I \subseteq N_T \times$

Graph- and Model-Driven Engineering
Übungsblatt 1 – Lösungsskizze

N_T und einer Menge von abstrakten Knoten $A \subseteq N_T$. Für diese Mengen und Funktionen gilt konkret:

$$\begin{aligned}
N_T &= \{\text{Window}, \text{Room}, \text{Kitchen}, \text{Bedroom}, \text{Bath}\} \\
E_T &= \{\text{has}\} \\
D_T &= \{\mathsf{bool}\} \\
A_T &= \{\text{open}, \text{light}\} \\
s_{E_T}(\text{has}) &= \text{Room} \\
t_{E_T}(\text{has}) &= \text{Window} \\
s_{A_T} &= \{\text{open} \mapsto \text{Window}, \text{light} \mapsto \text{Room}\} \\
t_{A_T} &= \{\text{open} \mapsto \mathsf{bool}, \text{light} \mapsto \mathsf{bool}\} \\
I &= \{(\text{Kitchen}, \text{Room}), (\text{Bedroom}, \text{Room}), (\text{Bath}, \text{Room})\} \cup \\
&\quad \{(X, X) \mid X \in N_T\} \\
A &= \{\text{Room}\}
\end{aligned}$$

(Hierbei ist D_T die einelementige Boolesche Algebra, wo $\neg\mathsf{bool} = \mathsf{bool}$, $\mathsf{bool} \wedge \mathsf{bool} = \mathsf{bool}$ usw.)

Der Startgraph SG ist ein erweiterter getypter Graph $SG = ((N_{SG}, E_{SG}, D_{SG}, A_{SG}, s_{E_{SG}}, t_{E_{SG}}, s_{A_{SG}}, t_{A_{SG}}), type_{SG})$, wobei:

$$\begin{aligned}
N_{SG} &= \{\text{Window}_{SG}, \text{Kitchen}_{SG}, \text{Bedroom}_{SG}, \text{Bath}_{SG}\} \\
E_{SG} &= \{\text{has}_{SG}\} \\
D_{SG} &= \{\mathsf{true}, \mathsf{false}\} \\
A_{SG} &= \{\text{open}_{SG}, \text{light}_1, \text{light}_2, \text{light}_3\} \\
s_{E_{SG}}(\text{has}_{SG}) &= \text{Kitchen}_{SG} \\
t_{E_{SG}}(\text{has}_{SG}) &= \text{Window}_{SG} \\
s_{A_{SG}} &= \{\text{open} \mapsto \text{Window}_{SG}, \text{light}_1 \mapsto \text{Kitchen}_{SG}, \text{light}_2 \mapsto \text{Bath}_{SG}, \\
&\quad \text{light}_3 \mapsto \text{Bathroom}_{SG}\} \\
t_{A_{SG}} &= \{\text{open} \mapsto \mathsf{false}, \text{light}_i \mapsto \mathsf{false} \mid i = 1, 2, 3\}
\end{aligned}$$

(Hierbei ist D_{SG} die gewöhnliche Boolesche Algebra mit Elementen true und false und dem normalen Verhalten der Operatoren $\neg, \wedge, \ldots$. Die Namen, die wir den

Graph- and Model-Driven Engineering
Übungsblatt 1 – Lösungsskizze

Elementen in obigen Mengen gegeben haben, sind nicht weiter wichtig. Wir hätten genauso $x, y, z, \ldots$ verwenden können. Ihre Bedeutung erhalten die Elemente durch die folgende Typisierungsfunktion.)

Die Typisierungsfunktion $type_{SG}$ bildet ab, wie bereits durch die Wahl der Namen angedeutet:

$$type^N_{SG} \coloneqq \begin{cases} N_{SG} \to N_T \\ \text{Window}_{SG} \mapsto \text{Window}, \text{Kitchen}_{SG} \mapsto \text{Kitchen}, \text{Bedroom}_{SG} \mapsto \text{Bedroom}, \\ \text{Bath}_{SG} \mapsto \text{Bath} \end{cases}$$

$$type^E_{SG} \coloneqq \begin{cases} E_{SG} \to E_T \\ \text{has}_{SG} \mapsto \text{has} \end{cases}$$

$$type^D_{SG} \coloneqq \begin{cases} D_{SG} \to D_T \\ \mathsf{true} \mapsto \mathsf{bool}, \mathsf{false} \mapsto \mathsf{bool} \end{cases}$$

$$type^A_{SG} \coloneqq \begin{cases} A_{SG} \to A_T \\ \text{open}_{SG} \mapsto \text{open}, \text{light}_i \mapsto \text{light} \ (i = 1, 2, 3) \end{cases}$$

Die Regel *FensterZu* besteht aus zwei typisierten erweiterten Graphen L und R (der linken und der rechten Regelseite); die Menge der NACs ist für diese Regel leer. Die Graphen L und R und ihre Typisierungsfunktionen werden aufgeschrieben wie der Graph SG oben. Wichtig zu beachten ist hierbei das Folgende:

- Elemente, die in beiden Regelseiten auftauchen, also nicht gelöscht oder erzeugt werden, müssen in beiden Graphen gleich heißen (und natürlich gleich typisiert werden); sonst liegen sie nicht im Schnitt der beiden Graphen. Es gilt also etwa $E_L = E_R = \{\text{has}_{LR}\}$. Allgemein sind die beiden Graphen also beinahe identisch.
- Elemente, die gelöscht bzw. erzeugt werden, müssen individuelle Namen haben (damit sie nicht im Schnitt von L und R liegen). Dies betrifft in der Regel *FensterZu* nur A_L und A_R, wo z. B. gilt $A_L = \{\text{open}_L\}$ und $A_R = \{\text{open}_R\}$ (beide über open getypt). Es gilt $s_{A_L}(\text{open}_L) = s_{A_R}(\text{open}_R) = \text{Window}_{LR}$, aber $t_{A_L}(\text{open}_L) = \mathsf{true}$ und $t_{A_R}(\text{open}_R) = \mathsf{false}$.

Jens Kosiol

Graph- and Model-Driven Engineering
Übungsblatt 1 – Lösungsskizze

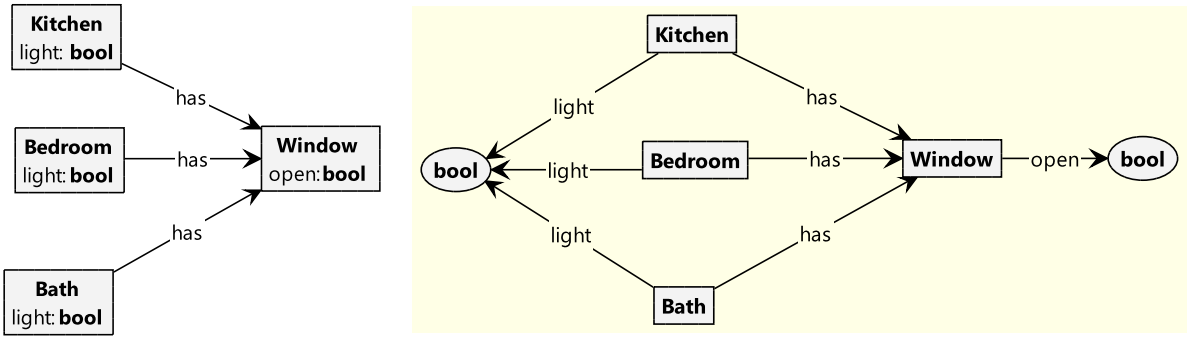


Abbildung 2: Flachgeklopfter Typgraph, einmal in der kompakten (links) und einmal in der Bearbeitungsansicht (rechts) von Groove

- Beide Graphen einer Regel enthalten immer (!) die gleichen Datenknoten (die gleiche Algebra). Es gilt also $D_L = D_R$ und die Menge muss mindestens true und false enthalten (naheliegendste Wahl ist also wieder die zweielementige Boolesche Algebra, die aus true und false besteht).

b) Die Abbildung 2 zeigt den entsprechenden Typgraphen. Ohne Vererbung werden $4 \cdot 3 = 12$ Regeln benötigt: für jede der vier Regeln drei Varianten für die konkreten Räume.