UNIKASSEL VERSITÄT | ELEKTROTECHNIK INFORMATIK

Jens Kosiol

Ausgabe: 24. April 2023
Abgabe: 8. Mai 2023

Graph- and Model-Driven Engineering
Übungsblatt 1

Allgemeine Hinweise

- Übungsblätter können in Gruppen von zwei Personen bearbeitet werden.
- Die Abgabe erfolgt per E-Mail an jens.kosiol@uni-kassel.de *vor Beginn der Vorlesung* am Montag.
- Aufgaben, bei denen Graphtransformationssysteme zu entwerfen sind, sollen in Groove bearbeitet werden. Es ist immer die entworfene Grammatik als gps-Datei mit abzugeben. Zur Erklärung ihrer Lösung können Sie entweder sog. Kommentarknoten in Groove verwenden oder Ihre Graphen und Regeln als Bilder exportieren und in eine Textdatei einbinden. Geben Sie Ihre Lösungen als *eine* zip-Datei ab, die die entworfenen Grammatiken und *maximal eine* pdf-Datei mit Erklärungen enthält. Verwenden Sie Ihre(n) Nachnamen im Dateinamen Ihrer Abgabe.
- Bei Fragen darf gerne der fragen-Kanal auf dem Discord-Server zur Vorlesung verwendet werden. Hier können Sie sich auch gegenseitig Hilfestellungen geben. Bloßes Abschreiben vollständiger Lösungen führt allerdings zur Bewertung der Aufgabe mit 0 Punkten.
- Bei Erreichen von mindestens 70 % der Übungspunkte im Laufe des Semesters gibt es einen Bonus von einer Notenstufe auf die (wahrscheinlich mündliche) Abschlussprüfung. Das heißt, statt beispielsweise einer 2,0 erhalten Sie in diesem Fall eine 1,7 auf die Prüfung.

1 Graphtransformationssystem (GTS) aus Code (10 Punkte)

Gegeben sei die Java-Implementierung eines Stack aus Listing 1.[1] Entwerfen Sie, analog zum Beispiel für zirkuläre Buffer aus der Vorlesung, ein getyptes Graphtransformationssystem, das diese Datenstruktur modelliert:

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

[1] Quelle für den Code: https://de.wikipedia.org/wiki/Stapelspeicher

Jens Kosiol

Ausgabe: 24. April 2023
Abgabe: 8. Mai 2023

Graph- and Model-Driven Engineering
Übungsblatt 1

b) Erstellen Sie einen sinnvollen Startgraphen für das Graphtransformationssystem. (Achtung: Der Startgraph ergibt sich nicht aus dem Code.)

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

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

```
class Stack<T> {
  private class Item {
    T value; // Das zu verwaltende Objekt
    Item next; // Referenz auf den naechsten Knoten

    Item(T value, Item next) {
      this.value = value;
      this.next = next;
    }
  }

  private Item peek;

  // Prueft, ob der Stapel leer ist
  public boolean empty() {
    return this.peek == null;
  }

  // Speichert ein neues Objekt
  public void push(T value) {
    this.peek = new Item(value, this.peek);
  }

  // Gibt das oberste Objekt wieder und entfernt es aus dem Stapel
  public T pop() {
    T temp = this.peek.value;
```

Jens Kosiol

Ausgabe: 24. April 2023
Abgabe: 8. Mai 2023

Graph- and Model-Driven Engineering
Übungsblatt 1

```
    if (!empty())
      this.peek = this.peek.next;

    return temp;
  }
}
```

Listing 1: Java-Implementierung eines Stack.

2 Ein adaptiertes Rucksackproblem (12 Punkte)

In dem *adaptierten Rucksackproblem* soll der Wert von in einen Rucksack gepackten Gegenständen maximiert werden unter den Nebenbedingungen, dass das Volumen des Rucksack nicht überschritten werden darf und von jeder Art von Gegenstand maximal eine bestimmte Anzahl gewählt werden darf. Zusätzlich existieren Abhängigkeiten zwischen den einzelnen Gegenständen.

Formal: Sei $W \geq 0$ eine Zahl (Volumen des Rucksacks). Gibt es n verschiedene Arten von Gegenständen und bezeichnet v_i jeweils den Wert des i-ten Gegenstandes, w_i dessen Gewicht und c_i die Häufigkeit, wie oft dieser Gegenstand verwendet werden darf, dann besteht das (adaptierte) Rucksackproblem darin, den Wert

$$\sum_{i=1}^{n} v_i x_i$$

zu maximieren unter den Nebenbedingungen

$$\sum_{i=1}^{n} w_i x_i \leq W \text{ und } 0 \leq x_i \leq c_i \ ,$$

wobei x_i zählt, wie oft der i-te Gegenstand gewählt wurde. Zusätzlich können (Un)gleichungen zwischen den x_i gegeben seien, die durch die Lösung erfüllt sein müssen, wie etwa

$$x_k \leq x_l \text{ für } l \neq k \in \{1, \ldots, n\}$$

oder

$$(x_k > 0) \implies (x_l = 0) \text{ für } l \neq k \in \{1, \ldots, n\} \ .$$

Jens Kosiol

Ausgabe: 24. April 2023
Abgabe: 8. Mai 2023

Graph- and Model-Driven Engineering
Übungsblatt 1

Entwerfen Sie einen Typgraphen und eine Menge von Graphtransformationsregeln in Groove, die das Wählen und Entfernen von Gegenständen aus dem Rucksack modellieren. Die folgenden Bedingungen sollen hierbei erfüllt sein:

- Das Volumen des Rucksacks sei $W = 40$. Es gibt 4 verschiedene Arten von Gegenständen mit den Eigenschaften (Wert/Gewicht/Häufigkeit): [(5,10,3),(4,7,3),(3,4,2),(4,7,2)]. Außerdem gelten folgende Abhängigkeiten:
  - Es sind immer gleichviele Gegenstände der ersten und der dritten Art im Rucksack.
  - Gibt es einen Gegenstand der zweiten Art im Rucksack, dann auch einen der dritten.
  - Gibt einen Gegenstand der vierten Art, dann keinen der zweiten.
- Sie sollen ein Regelsystem entwerfen, das es erlaubt, jeden Gegenstand dem Rucksack hinzuzufügen oder jede Art von Gegenstand wieder aus dem Rucksack zu entfernen. Dabei soll das Hinzufügen oder Entfernen eines Gegenstandes zu/aus dem Rucksack nur möglich sein, falls dadurch weder das Volumen des Rucksacks noch die erlaubte Anzahl von Gegenständen dieser Art überschritten wird noch die Abhängigkeiten zwischen den Gegenständen verletzt.

Wichtig: Entwerfen Sie einen Typgraphen und aktivieren Sie diesen. Der Startgraph für Ihre Grammatik darf einfach leer sein. Beachten Sie, dass Sie keinen Algorithmus zur Lösung des Rucksackproblems entwerfen müssen, sondern lediglich Regeln zum Hinzufügen und Entfernen von Gegenständen in den Rucksack, also Regeln, die innerhalb eines solchen Algorithmus eingesetzt werden könnten.

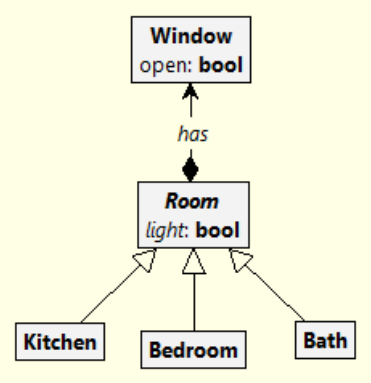
3 Graphtransformationssysteme mit Vererbung (6 Punkte)

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

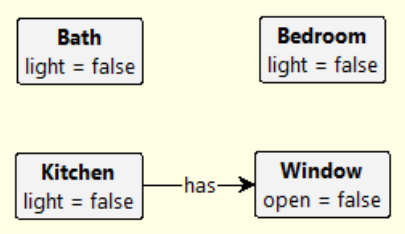
a) Spezifizieren Sie *formal* das getypte Graphtransformationssystem, welches aus dem Typgraph *TG*, dem Startgraph *SG* und der Regel *FensterAuf* besteht. Geben Sie

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.).

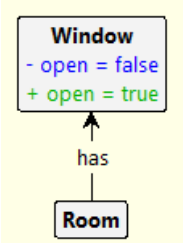
b) Klopfen Sie den Typgraphen in Bezug auf die darin enthaltene Vererbungshierarchie flach. Entfernen Sie also jegliche Vererbung und passen Sie Objekte und Kanten so an, dass keine Informationen verloren gehen. Überlegen und begründen Sie zudem, wieviele Regeln benötigt werden, um das Verhalten der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



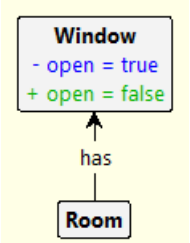
(a) TG



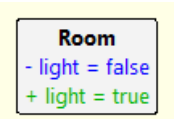
(b) SG



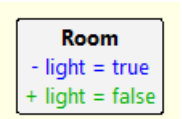
(c) FensterAuf



(d) FensterZu



(e) LichtAn



(f) LichtAus

Abbildung 1: Typgraph, Startgraph und Regeln eines GTS.