Spezifikation von objektorientierten Systemen mit Graphtransformationssystemen – Einführung

Jens Kosiol

22. und 24. April 2024

Überblick

- Wie lassen sich objektorientierte Systeme modellieren?
  - *Graphen modellieren Objektstrukturen*
  - *Graphtransformationen modellieren Änderungen auf Objektstrukturen*
- Wie funktioniert Graphtransformation?
  - *Getypte Graphen*
  - *Regelbasierte Graphtransformation*
  - *Graphtransformationssysteme*

Modellierung von Objektstrukturen mit Graphen

- Mit Graphen kann man verschiedenste Objektstrukturen modellieren.
- Graphknoten
  - *Objekte als Instanzen von Klassen*
  - *Verschiedene Typen von Knoten*
  - *Attribute, über elementare Datentypen getypt*
- Graphkanten
  - *Referenzen zwischen Objekten*
  - *Relationen zwischen Knoten*
  - *Verschiedene Typen von Kanten*

Java-Beispiel für zirkulären Puffer

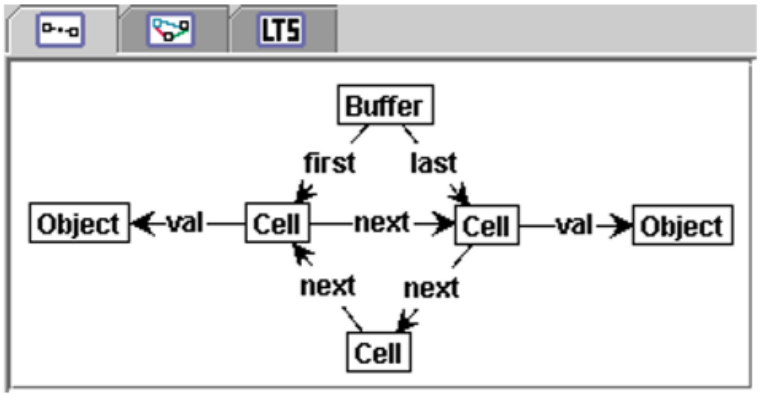
```
public class Cell {
  public Object val;
  public Cell next;
}

public class Buffer {
  private Cell first, last;

  public Buffer() {
    first = new Cell();
    first.next = new Cell();
    first.next.next = new Cell();
    last = first.next.next;
    last.next = first;
  }

  public void put(Object arg) {
    if (last.next.val == null) {
      last = last.next;
      last.val = arg;
    }
  }

  public void drop() {
    if (first.val != null) {
      first.val = null;
      first = first.next;
    }
  }
}
```

[ZR10]

Beispiel: Ein zirkulärer Puffer

Produzent-Konsument-Problem mit Puffer:

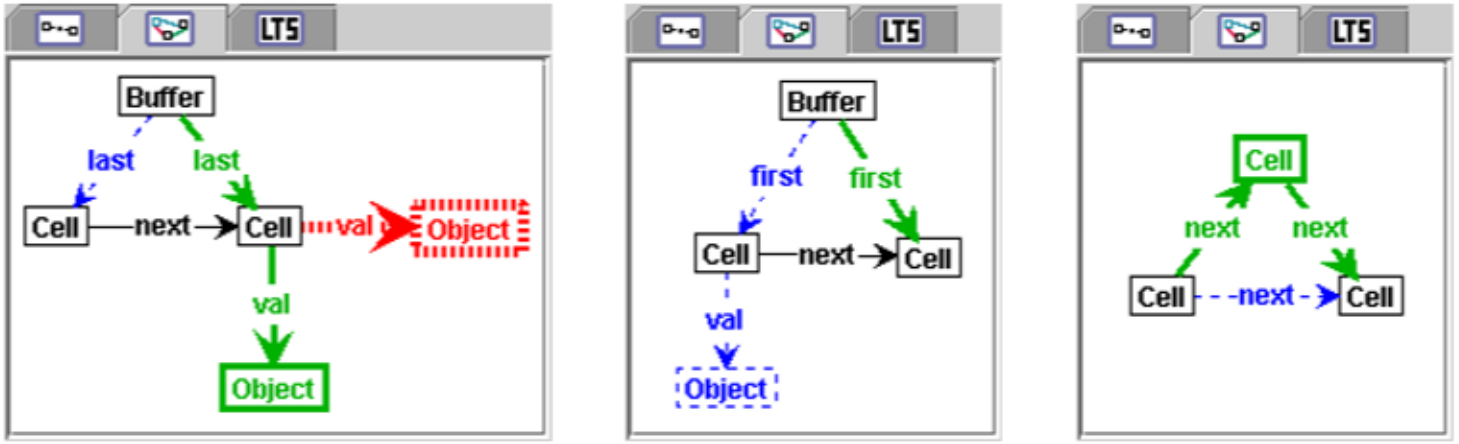
- *Der Puffer hat n Zellen, die ringförmig angeordnet sind. Diese dürfen der Reihe nach mit Werten belegt und umgekehrt geleert werden.*
- *Der Puffer darf auch um Zellen erweitert werden.*
- *Vorteil gegenüber einer Queue: Kein Shift von Elementen, Zugriffsoperationen in konstanter Zeit*



Ein Puffer mit drei Zellen, von denen zwei belegt sind. [KR06]

Beispiel: Ein zirkulärer Puffer

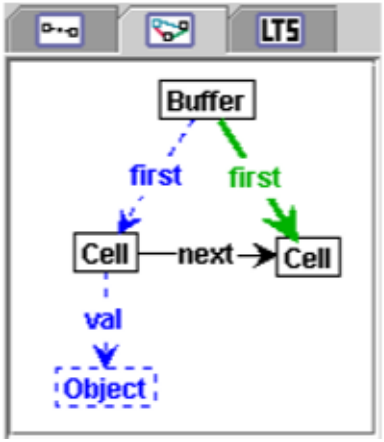
- Regelbasierte Spezifikation von Aktionen
  - *Schwarz: zu erhaltene Strukturanteile*
  - *Blau: zu löschende Strukturanteile*
  - *Rot: verbotene Strukturteile*
  - *Grün: zu erzeugende Strukturanteile*



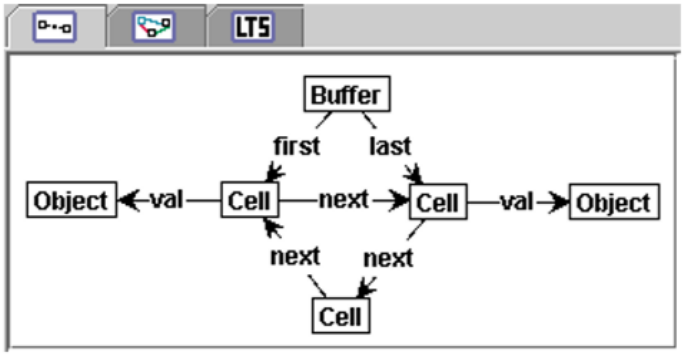
(a) put rule. (b) get rule. (c) extend rule. [KR06]

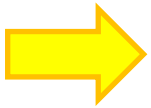
Beispiel: Regelanwendung

Regel *get*:

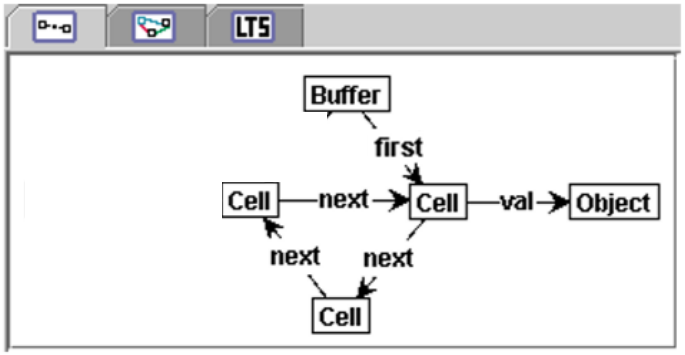


Graph G:





Graph H:



Zu formalisieren

Die folgenden Dinge müssen wir formalisieren, damit das Beispiel präzise wird:

- Graphen
  - *Es muss möglich sein, die Rollen der verschiedenen Elemente (Knoten und Kanten) irgendwie auszudrücken.*
  - *Es muss möglich sein, Datenwerte im Graphen darzustellen.*
- Graphtransformationsregeln
- Anwendungen von Graphtransformationsregeln und die Semantik, die dadurch entsteht.

Überblick

- Wie lassen sich objektorientierte Systeme modellieren?
  - *Graphen modellieren Objektstrukturen.*
  - *Graphtransformationen modellieren Änderungen auf Objektstrukturen*
- Wie funktioniert Graphtransformation?
  - *Getypte Graphen*
  - *Regelbasierte Graphtransformation*
  - *Graphtransformationssysteme*

Definition: Graph

Ein Graph G ist ein Tupel $G = (N, E, s, t)$:

- N ist eine Menge von Knoten
- E ist eine Menge von Kanten
- $s: E \to N$ ist eine Funktion, die Kanten ihre Quellknoten zuweist (Quellfunktion)
- $t: E \to N$ ist eine Funktion, die Kanten ihre Zielknoten zuweist (Zielfunktion)

Spezialfall: Ein Graph hat keine parallelen Kanten

- $E \subseteq N \times N$ mit $s(n_1, n_2) = n_1$ und $t(n_1, n_2) = n_2$

Typisierung von Graphen

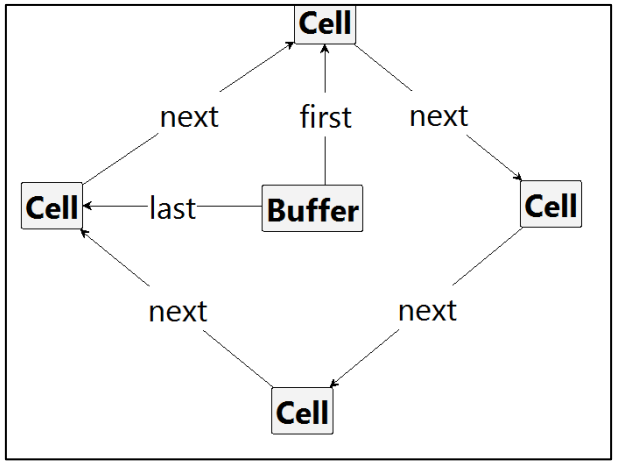
Zur Darstellung von Objektstrukturen durch Graphen

- modelliert jeder Knoten ein Objekt einer Klasse und
- jede Kante eine Referenz auf ein Objekt.

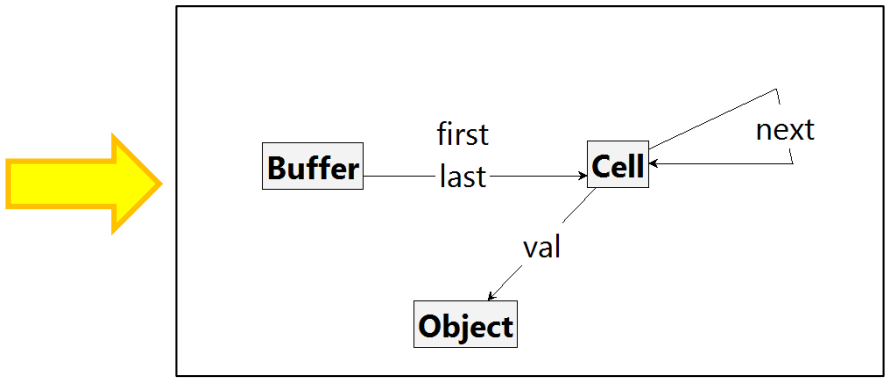
Ein getypter Graph kann diese Typisierung modellieren.

- Ein Typgraph modelliert eine Klassenstruktur (.
- Ein getypter Graph modelliert eine Objektstruktur über dieser Klassenstruktur.

Beispiel: Getypter Graph



Getypter Graph



Typgraph

Intuition:

- Ist ein Element der durch den Typgraphen definierten Sprache
- Vergleichbar mit einem Objektdiagramm

- Definiert eine Sprache von Graphen („Alphabet und Grammatik“)
- Vergleichbar mit einem Klassendiagramm

Definition: Getypter Graph

Gegeben ein Graph TG, genannt der Typgraph, besteht ein über TG getypter Graph G

- aus einem Graphen $\overline{G}$ und
- einer strukturkonformen Abbildung (Graphmorphismus) $\text{type}: \overline{G} \rightarrow TG$.

Das Tupel $G = (\overline{G}, \text{type})$ wird getypter Graph genannt und $\mathbf{Graph_{TG}}$ bezeichnet die Menge aller Graphen, die über TG getypt sind.

Definition: Graphmorphismus

Ein Graphmorphismus $f: G \to H$ zwischen Graphen $G = (N_G, E_G, s_G, t_G)$ und $H = (N_H, E_H, s_H, t_H)$ besteht aus zwei Funktionen $f_N: N_G \to \mathrm{N}_H$ und $f_E: E_G \to E_H$, sodass

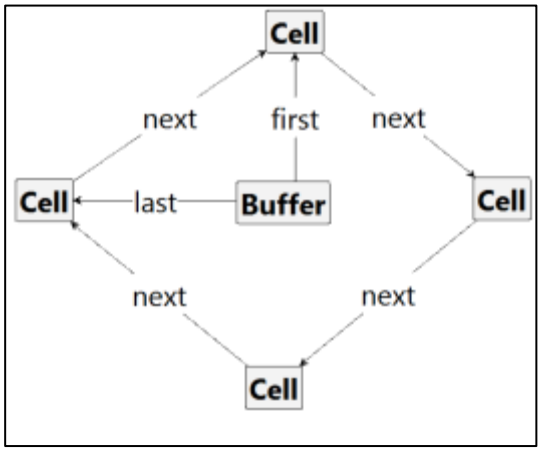
- $f_N(s_G(e)) = s_H(f_E(e))$ für alle $e \in E_G$ (Verträglichkeit mit Quellfunktion) und
- $f_N(t_G(e)) = t_H(f_E(e))$ für alle $e \in E_G$ (Verträglichkeit mit Zielfunktion).

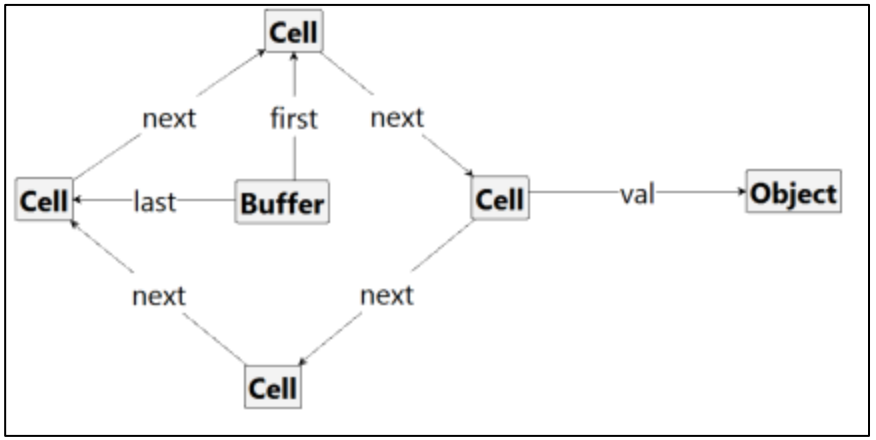
Ein Graphmorphismus f ist injektiv (surjektiv, bijektiv), falls f_N und f_E injektiv (surjektiv, bijektiv) sind. Wenn f bijektiv ist, heißen G und H isomorph (strukturgleich).

Abbildungen zwischen getypten Graphen

- Objektstrukturen dürfen nur struktur- und typkonform aufeinander abgebildet werden.
- Getypte Graphen werden durch Graphmorphismen strukturkonform abgebildet.
- Getypte Graphen werden typkonform aufeinander abgebildet, wenn
  - *ein Knoten nur auf einen Knoten desselben Typs abgebildet wird.*
  - *eine Kante nur auf eine Kante desselben Typs abgebildet wird.*

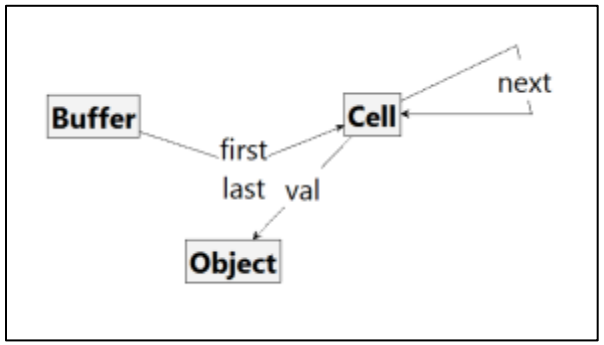
Beispiel: Getypter Graphmorphismus





Getypter Graph G_1

Getypter Graph G_2



Typgraph

Definition: Getypter Graphmorphismus

Gegeben ein Typgraph TG und zwei über TG getypte Graphen $G_1 = (\overline{G}_1, \text{type}_1)$ und $G_2 = (\overline{G}_2, \text{type}_2)$, ist ein getypter Graphmorphismus $f: G_1 \to G_2$ ein Graphmorphismus $f: \overline{G}_1 \to \overline{G}_2$, sodass $\text{type}_2 \circ f = \text{type}_1$, also:

- $type_{2N}(f_N(n)) = type_{1N}$ für alle $n \in N_{G_1}$ und
- $type_{2E}(f_E(e)) = type_{1E}$ für alle e $\in E_{G_1}$.

Zwei getypte Graphen sind isomorph, falls es einen bijektiven getypten Graphmorphismus zwischen diesen Graphen gibt.

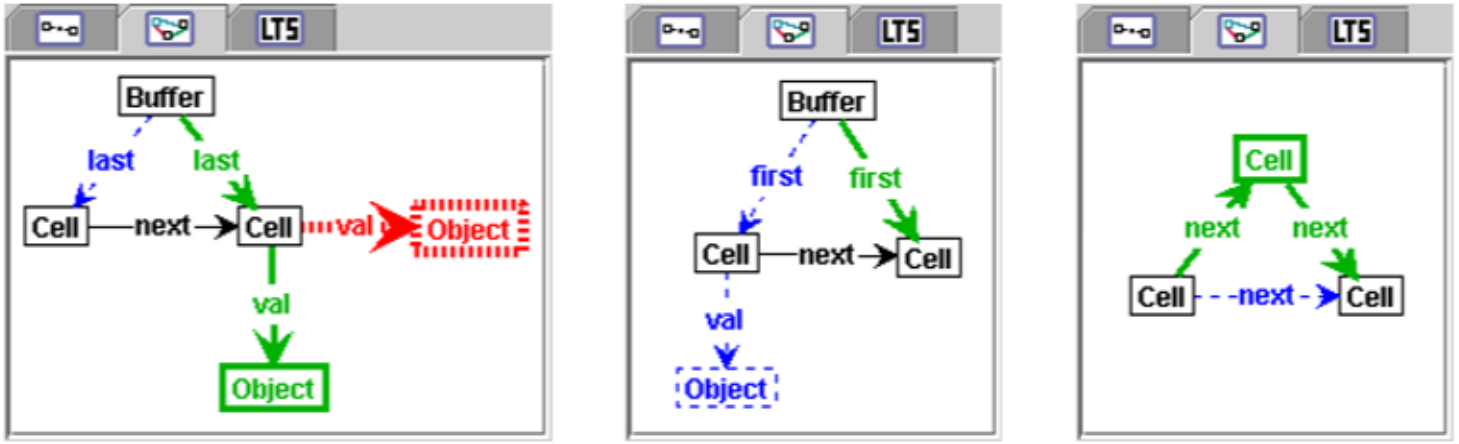
Ab jetzt: Notation eines getypten Graphen $G = (\overline{G}, \text{type})$ manchmal auch einfach als G (sowohl für den gesamten getypten als auch den unterliegenden Graphen).

Modellierung von Strukturänderungen

- Graphänderungen werden regelbasiert spezifiziert.
- Jede Regel definiert eine Aktion, die eine Änderung auf Graphen spezifiziert.
- Eine Graphregel besteht aus
  - *einer Graphstruktur, die existieren muss,*
  - *einer Menge von Änderungsaktionen und*
  - *einer Menge von negativen Anwendungsbedingungen, die die Nichtexistenz von (Teil-)graphstrukturen fordern.*
- Die Anwendung einer Regel auf einen Graphen führt die spezifizierten Aktionen auf diesem Graphen aus.

Beispiel: Ein zirkulärer Puffer

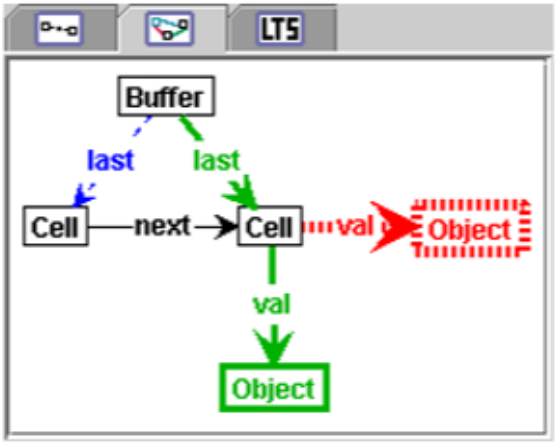
- Regelbasierte Spezifikation von Aktionen
  - *Schwarz: zu erhaltene Strukturanteile*
  - *Blau: zu löschende Strukturanteile*
  - *Rot: verbotene Strukturteile*
  - *Grün: zu erzeugende Strukturanteile*



(a) put rule. (b) get rule. (c) extend rule. [KR06]

Graphregel intuitiv

- Regelbasierte Spezifikation von Aktionen $r = (L, R, \mathrm{NAC})$
  - *Linke Regelseite L:*
    - Schwarz: zu erhaltener Graphteil
    - Blau: zu löschender Graphteil
  - *Rechte Regelseite R:*
    - Schwarz: zu erhaltener Graphteil
    - Grün: zu erzeugender Graphteil
  - *Negative Anwendungsbedingung* NAC*:*
    - Linke Regelseite
    - Rot: verbotene Strukturteile

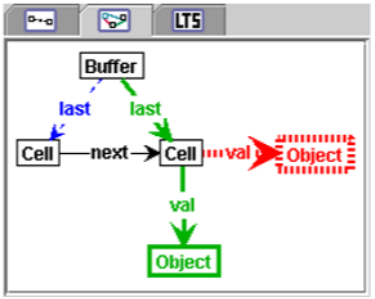


(a) put rule. [KR06]

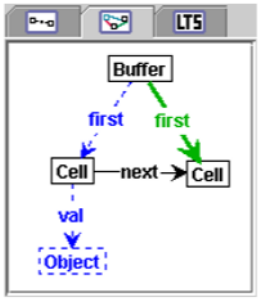
Definition: Getypte Graphregel

Gegeben ein Typgraph TG, ist eine getypte Graphregel ein Tupel $r = (L, R, \mathrm{NAC})$, wobei:

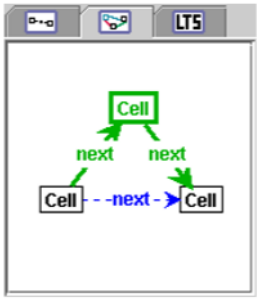
- $L = (\overline{L}, \mathrm{type}_\mathrm{L})$ und $R = (\overline{R}, \mathrm{type}_\mathrm{R})$ sind über TG getypte Graphen.
- $\overline{K} = \overline{L} \cap \overline{R}$ ist ein Graph und type_L und type_R stimmen auf $\overline{K}$ überein (sodass wir einen getypten Graphen $K = (\overline{K}, \mathrm{type}_\mathrm{K})$ erhalten).
- NAC ist eine Menge von über TG getypten Graphen, genannt negative Anwendungsbedingungen, wobei jedes $N \in \mathrm{NAC}$ ein getypter Obergraph von L ist, also gilt $\overline{L} \subseteq \overline{N}$ und $\mathrm{type}_\mathrm{L} = \mathrm{type}_\mathrm{N}$ auf $\overline{L}$.



(a) put rule.



(b) get rule.



(c) extend rule.

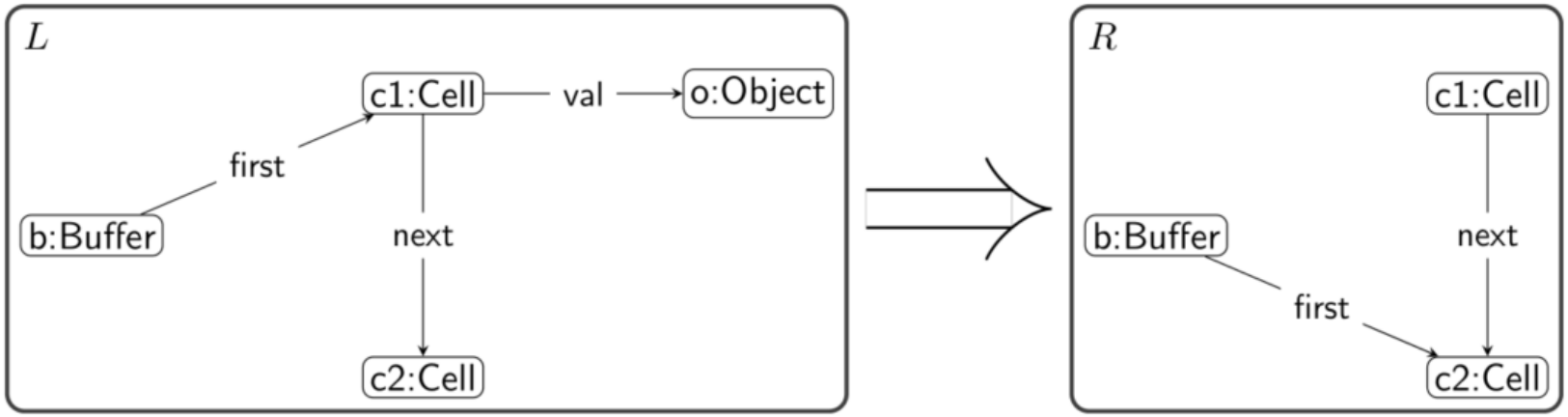
[KR06]

Notationen einer Graphregel

Graphregeln können auf verschiedene Weisen notiert werden:

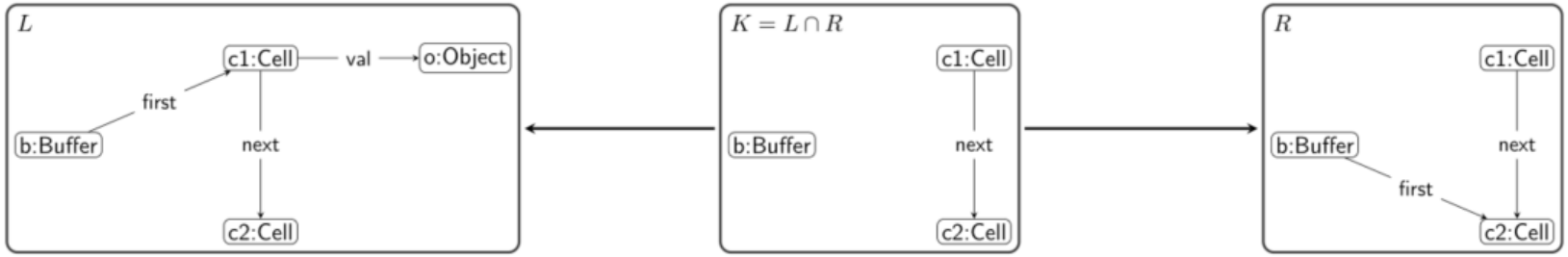
- Notation als Tupel $r = (L, R, NAC)$
  - *Die zu erhaltenden Elemente $K = L \cap R$ bleiben implizit*
- Integrierte Darstellung als ein Graph (z.B. in Groove)
  - *Die Rollen der einzelnen Elemente werden explizit durch Farbgebung und/oder Annotationen*
- Als Transformationsvorschrift $r = (L \Rightarrow R, \mathrm{NAC})$
  - *Die zu erhaltenden Elemente $K = L \cap R$ bleiben implizit*
- Notation als Span $r = (L \leftarrow K \rightarrow R, \mathrm{NAC})$

Beispiel „Transformationsvorschrift“



Darstellung der Regel *get* als Transformationsvorschrift

Beispiel „Span“



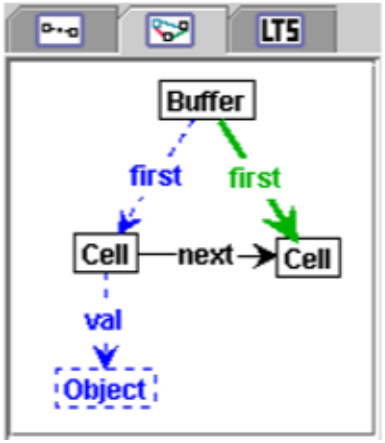
Darstellung der Regel *get* als Span

Anwendung einer Regel (Überblick)

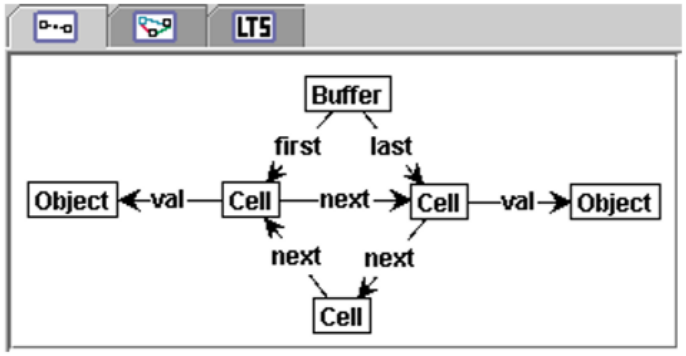
- Gegeben: ein Graph G und eine Regel $r = (L, R, \mathrm{NAC})$
- Die Regel r ist auf G anwendbar, falls
  - *L injektiv auf einen Teilgraph von G abbildbar ist,*
  - *alle Bedingungen in NAC für diese Abbildung erfüllt sind und*
  - *die Löschung der zu löschenden Graphelemente keine hängenden Kanten zurücklässt.*
- Die Anwendung von r auf G erzeugt einen Graphen H, der sich folgendermaßen ergibt:
  - *Von G werden die zu löschenden Graphelemente gelöscht und*
  - *die zu erzeugenden Graphelemente hinzugefügt.*

Beispiel: Regelanwendung

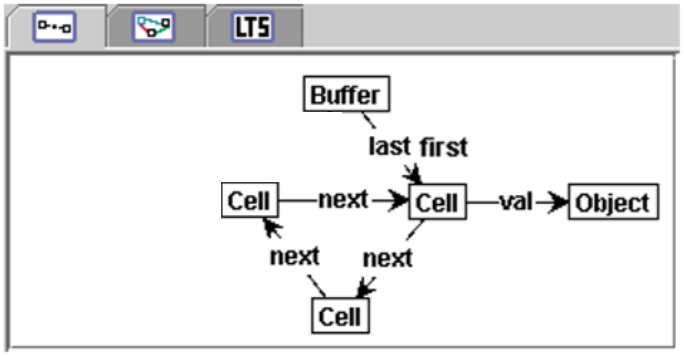
Regel *get*:



Graph G:



Graph H:



Definition: Regelanwendung (Teil 1)

Gegeben ein Typgraph TG, ein Graph G und eine Regel $r = (L, R, \mathrm{NAC})$, beide über TG getypt, ist die Regel r auf den Graphen G anwendbar (hat einen Ansatz), falls gilt:

- Es gibt einen injektiven getypten Graphmorphismus $m: L \to G$ (genannt Ansatz oder Match).
- Alle negativen Anwendungsbedingungen sind erfüllt: Für keinen Graphen $N \in \mathrm{NAC}$ existiert ein injektiver getypter Graphmorphismus $q: N \to G$ mit $q(L) = m(L)$.
- Es gibt keine hängenden Kanten:

  $$D = G \setminus m\big(L \setminus (L \cap R)\big)$$

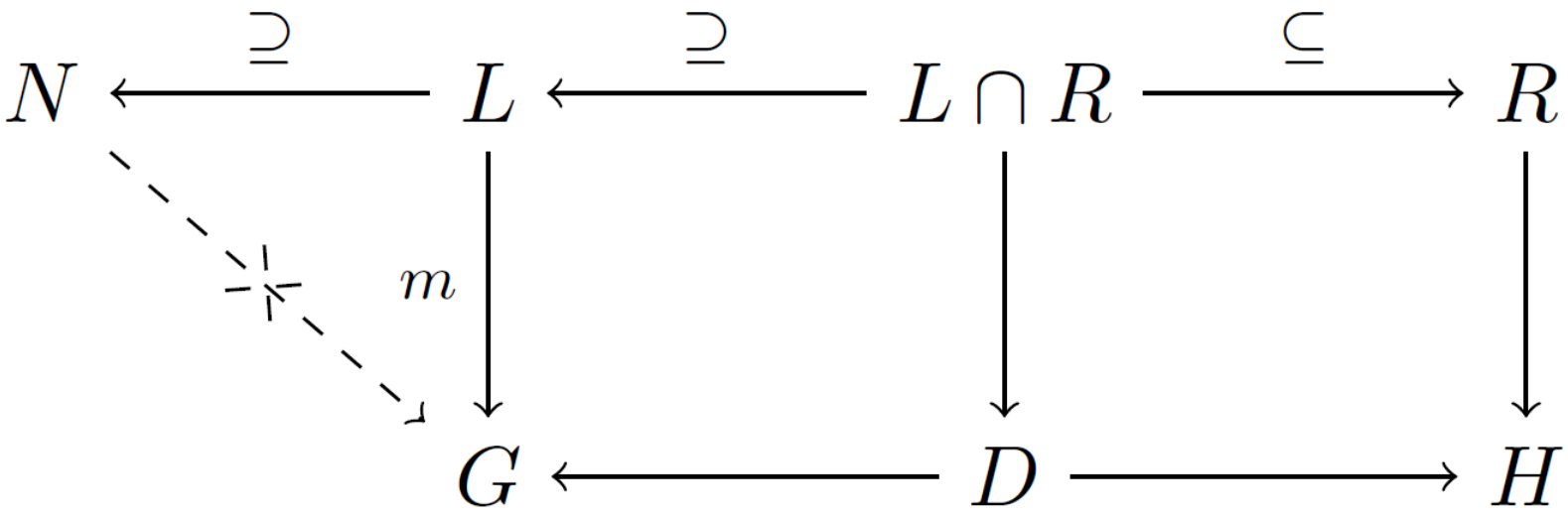
  ist ein Graph.

Definition: Regelanwendung (Teil 2)

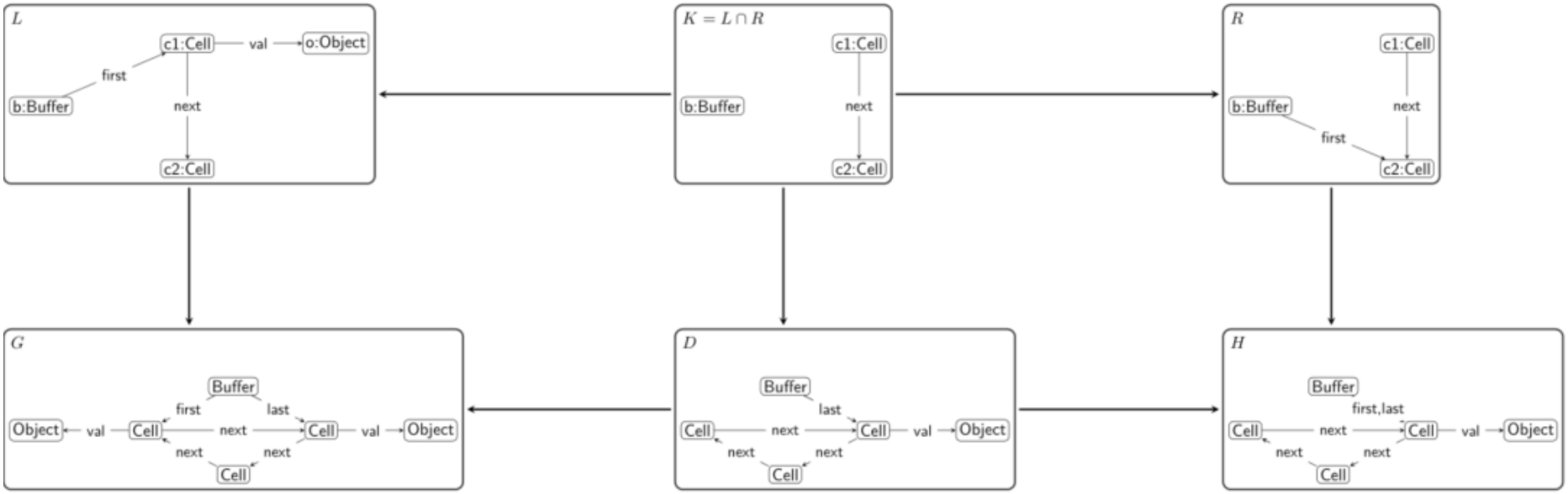
Existiert ein Ansatz m für die Regel r in G (ist die Regel also anwendbar), so berechnet sich die Anwendung an diesem Ansatz, notiert als $G \Rightarrow_{r,m} H$, wie folgt:

1. Löschen von Elementen: $D = G \setminus m\big(L \setminus (L \cap R)\big)$
   - $\mathrm{type_D}$, s_D und t_D sind die Einschränkungen der entsprechenden Funktionen von G.
2. Hinzufügen von Elementen: $H = D \uplus (R \setminus (L \cap R))$ (Achtung: disjunkte Vereinigung!)
   - $\mathrm{type_H}$ ergibt sich als Kombination von $\mathrm{type_D}$ und $\mathrm{type_R}$ (neu erzeugte Elemente erhalten den gleichen Typ wie in R).
   - Analog ergeben sich s_H und t_H als Kombinationen von s_D und s_R bzw. t_D und t_R (neu erzeugte Kanten erhalten Quelle und Ziel wie in R.)

Regelanwendung schematisch



Beispiel


L
c1:Cell
val
o:Object
first
b:Buffer
next
c2:Cell
K = L ∩ R
c1:Cell
b:Buffer
next
c2:Cell
R
c1:Cell
b:Buffer
next
first
c2:Cell
G
Buffer
first
last
Object
val
Cell
next
Cell
val
Object
next
next
Cell
D
Buffer
last
Cell
next
Cell
val
Object
next
next
Cell
H
Buffer
first,last
Cell
next
Cell
val
Object
next
next
Cell

Getyptes Graphtransformationssystem

- Ein Typgraph definiert die Struktur der beteiligten Objekt- und Datentypen.
- Der Startgraph definiert die initiale Objektstruktur.
- Eine Regel kann (eingeschränkt) mit einer Methode verglichen werden.
  - *Eine Regel kann eine einfache Methode definieren.*
  - *Für komplexe Methoden werden im Allgemeinen mehrere kontrollierte Regelanwendungen gebraucht.*

Definition: Getyptes Graphtransformationssystem

Ein getyptes Graphtransformationssystem $GTS = (TG, R, I)$ besteht aus

- einem Typgraphen TG,
- einem über TG getypten Startgraphen I und
- einer Menge R von über TG getypten Regeln.

Die Sprache

$$\mathcal{L}(GTS) := \{H \mid I \Longrightarrow_R^* H\}$$

eines Graphtransformationssystems GTS besteht aus allen getypten Graphen H, die (transitiv) per R aus I abgeleitet werden können.

Zusammenfassung

- Zur Spezifikation von objektorientierten Systemen verwenden wir getypte Graphtransformation.
  - *Typgraphen spezifizieren Klassenstrukturen.*
  - *Objektstrukturen werden durch getypte Graphen spezifiziert.*
- Regelbasierte Graphtransformationen spezifizieren erlaubte Graphänderungen.
- Ein Graphtransformationssystem spezifiziert ein objektorientiertes System.

Literatur und Links

- [KR06] Harmen Kastenberg, Arend Rensink: Model Checking Dynamic States in GROOVE. SPIN 2006: 299-305
- [ZR10] Eduardo Zambon, Arend Rensink: Using Graph Transformations and Graph Abstractions for Software Verification. ECEASST 38 (2010)
- Reiko Heckel, Gabriele Taentzer: Graph Transformation for Software Engineers, Springer, 2020, Kapitel 1 und 2
- Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, Gabriele Taentzer: Fundamentals of Algebraic Graph Transformation, Springer, 2006, Kapitel 2.1, 3.1 und 8.1
- Groove: Graphs for Object-Oriented Verification, https://groove.cs.utwente.nl/