Geschachtelte Graphbedingungen und Hoare-Style Verifikation von Graphtransformationsregeln

Jens Kosiol

10., 17. und 19. Juni 2024

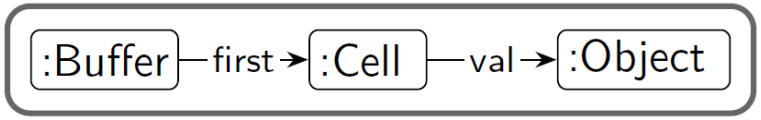
Übersicht

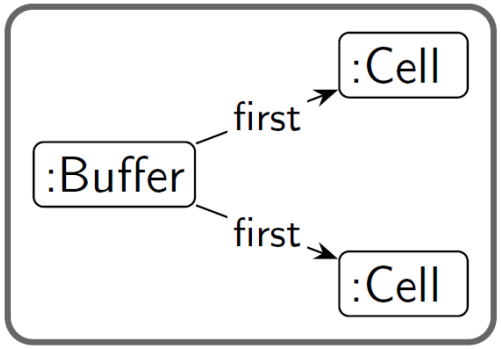
- Wie formalisieren wir gewünschte bzw. verbotene strukturelle Eigenschaften von Graphen und Graphmorphismen?
  - *Geschachtelte Graphbedingungen als (visuelle) Logik auf Graphen*
  - *Multiplizitäten als einfacher Spezialfall*
- Wie stellen wir sicher, dass das Ergebnis einer Graphtransformation gewünschte Eigenschaften hat?
  - *Automatische Berechnung von Anwendungsbedingungen aus Graphbedingung und Regel*

Ausgangsbeobachtung

- Ein Graph P ist schon eine einfache Formel.
  - *Ein Graph G erfüllt P, falls es einen injektiven Graphmorphismus $P \rightarrow G$ gibt (das Pattern P ist in G vorhanden).*
  - *Falls es keinen injektiven Graphmorphismus $P \rightarrow G$ gibt, erfüllt G die Eigenschaft P nicht.*
- Das Zulassen von Negationen ermöglicht das Verbieten von einfachen Strukturen.

Beispiel





Dieser Graph matcht einen Graphen G genau dann, wenn G einen nicht-leeren Buffer enthält.

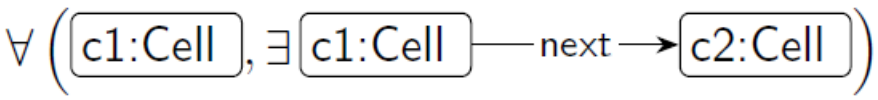
Dieser Graph matcht einen Graphen G genau dann nicht, wenn G keinen Buffer mit zwei ausgehenden first-Kanten enthält

Was fehlt?

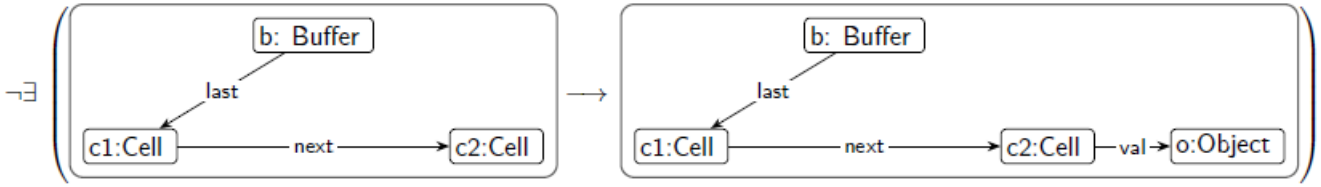
- Ausdrücken weiterer Eigenschaften über gefundene Pattern („Es gibt einen Buffer, der leer ist.“, „Für jede Zelle gilt, dass eine next-Kante von ihr ausgeht.“)
  - *Verketten von Pattern, wobei die tieferen Pattern weiter spezifizieren, welche Eigenschaften die früheren Pattern haben sollen*
  - *Formales Mittel: Graphmorphismen (bzw. Einbettungen)*
- Quantoren und Junktoren, um komplexere Formeln aus einfachen zu bauen
- Ausdrücken von Eigenschaften von Morphismen statt von Graphen (analog zu NACs)
  - *Semantik wird für Morphismen definiert*

Beispiel

- Für jede Zelle gilt, dass eine next-Kante von ihr ausgeht.



- Ein Morphismus aus dem ersten Graphen (LHS der put-Regel) soll Zelle c2 so abbilden, dass diese kein Object besitzt.



Definition: Geschachtelte Graphbedingung

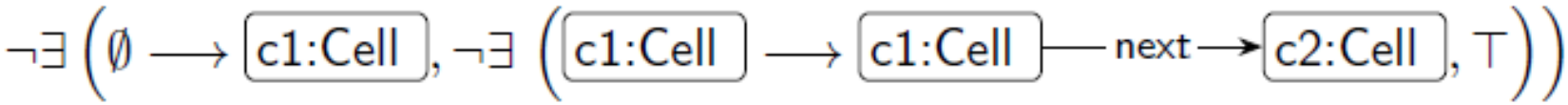
Gegeben sei ein Typgraph TG. Eine (geschachtelte) Graphbedingung über einem (über TG getypten) Graphen C_0 ist induktiv wie folgt definiert:

- true (⊤) ist eine Graphbedingung über C_0.
- Ist c eine Graphbedingung über C_1 und $a_0: C_0 \to C_1$ ein injektiver Morphismus (zwischen über TG getypten Graphen), dann ist $\exists(a_0, c)$ eine Graphbedingung über C_0.
- Sind c, c_1, c_2 Graphbedingungen über C_0, so sind $\neg c$ und $c_1 \vee c_2$ Graphbedingungen über C_0.

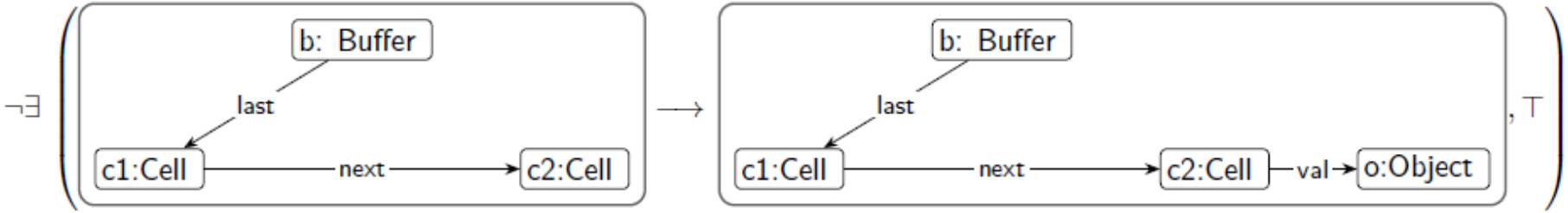
Eine (geschachtelte) Graphformel ist eine Graphbedingung über dem leeren Graphen ∅.

Beispiel

- Für jede Zelle gilt, dass eine next-Kante von ihr ausgeht.



- Ein Morphismus aus dem ersten Graphen (LHS der put-Regel) soll Zelle c2 so abbilden, dass diese kein Object besitzt.



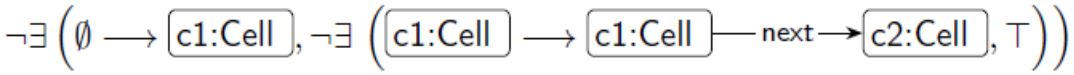
Vereinfachungen der Darstellung

Die folgenden Konventionen vereinfachen die Syntax von geschachtelten Graphbedingungen und –formeln:

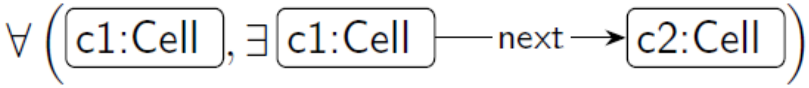
- Weglassen von $\top$ am Ende einer Graphbedingung (sofern nicht direkt davor eine Negation steht)
- Weglassen der Startgraphen der Morphismen aus denen die Graphbedingung besteht (abgesehen vom ersten Graphen falls $\neq \emptyset$)
- Kurzschreibweise $\forall(a_0, c)$ für $\neg\exists(a_0, \neg c)$
- Verwendung von $\wedge, \rightarrow, \ldots$ wie für Boolesche Operatoren bekannt

Beispiel

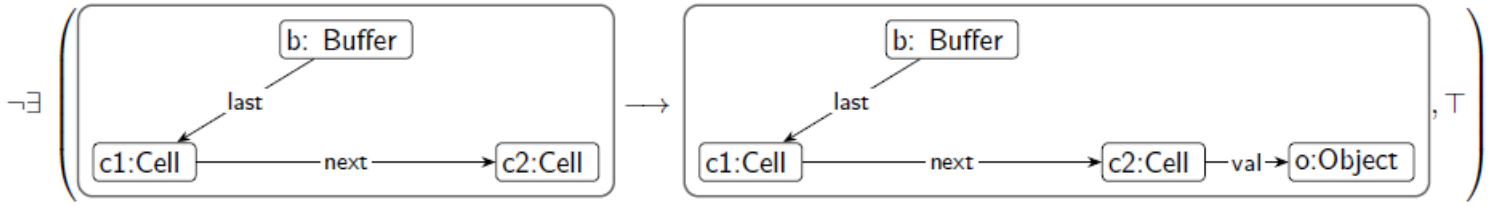
- Die Graphformel



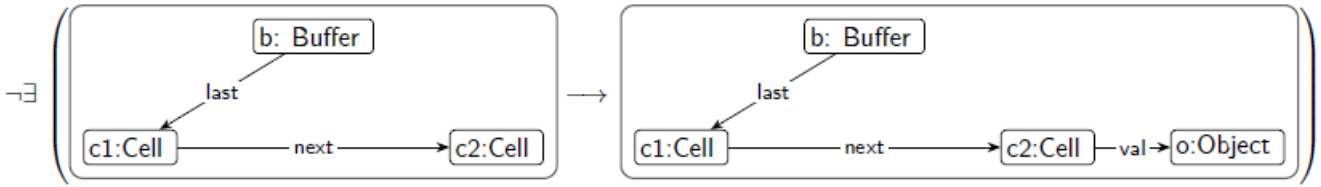
vereinfachen wir zu



- Die Graphbedingung



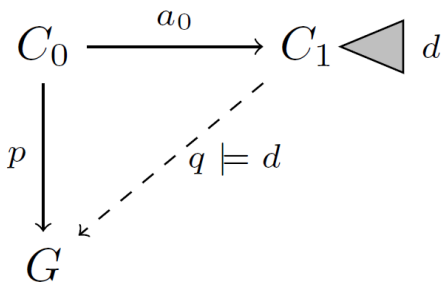
vereinfachen wir zu



Definition: Semantik von Graphbedingungen

Dass ein Morphismus $p: C_0 \to G$ eine Graphbedingung c über C_0 erfüllt (notiert als $p \vDash c$), ist induktiv wie folgt definiert:

- Jeder Morphismus p erfüllt $c = \top$.
- Der Morphismus p erfüllt $c = \exists(a_0: C_0 \to C_1, d)$, wenn ein injektiver Morphismus $q: C_1 \to G$ existiert, sodass $p = q \circ a_0$ und $q \vDash d$.



- Der Morphismus p erfüllt $c = \neg d$, wenn p die Bedingung d nicht erfüllt, und $c = d_1 \vee d_2$, wenn p die Bedingung d_1 oder die Bedingung d_2 erfüllt.

Ein Graph G erfüllt eine Graphformel c, wenn der leere Morphismus $\emptyset \to G$ die Formel c erfüllt.

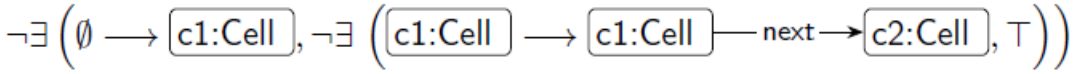
Beispiel abstrakt

Semantik von $\mathrm{c} = \neg\exists(a_0 \colon \emptyset \to C_1, \neg\exists(a_1 \colon C_1 \to C_2, \top))$ schrittweise:

- Ein Graph G erfüllt die Formel c, wenn er die Formel $\mathrm{d} = \exists(a_0 \colon \emptyset \to C_1, \neg\exists(a_1 \colon C_1 \to C_2, \top))$ nicht erfüllt.
- Ein Graph G erfüllt die Formel d, wenn eine Abbildung $q \colon C_1 \to G$ existiert, die die Graphbedingung $e = \neg\exists(a_1 \colon C_1 \to C_2, \top)$ erfüllt.
- Also erfüllt G die Formel d nicht, falls
  - *keine Abbildung $q \colon C_1 \to G$ existiert oder*
  - *jede solche Abbildung e nicht erfüllt.*
- Eine Abbildung $q \colon C_1 \to G$ erfüllt e, falls keine Abbildung $q' \colon C_2 \to G$ mit $q' \circ a_1 = q$ existiert
- Insgesamt erfüllt also G die Formel c, falls
  - *keine Abbildung $q \colon C_1 \to G$ existiert oder*
  - *für jede solche Abbildung q eine Abbildung $q' \colon C_2 \to G$ mit $q' \circ a_1 = q$ existiert.*

Beispiel konkret

Ein Graph G erfüllt die Formel



falls

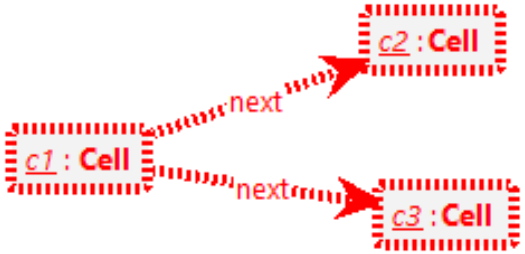
- G keinen Knoten vom Typ „Cell“ enthält oder
- für jeden solchen Knoten eine ausgehende Kante vom Typ „next“ zu einem weiteren Knoten vom Typ „Cell“ führt.

Graphbedingungen in Groove

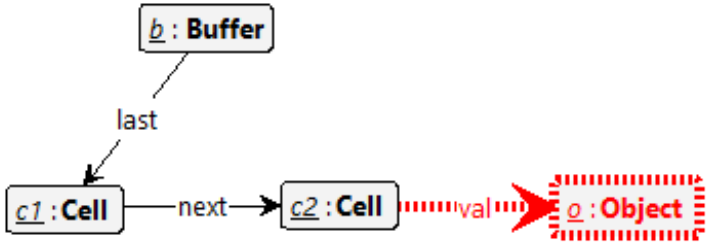
In Groove gibt es eine integrierte Darstellung als ein Graph mit Annotationen von Graphbedingungen.

- Quantorenknoten markieren, auf welche Ebene ein Element gehört
- Die Quantoren bilden einen Baum
- Negation durch die Negation der einzelnen Knoten
- Formalisiert in [Rensink04]
- Ausgehende Zweige aus einem Allquantor-Knoten werden als Disjunktion interpretiert, ausgehende Zweige aus einem Existenzquantor-Knoten als Konjunktion

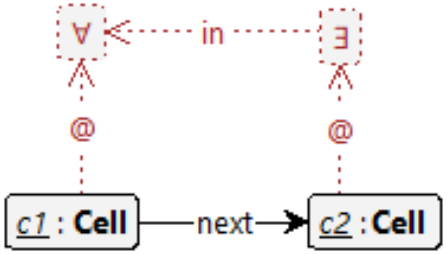
Beispiele



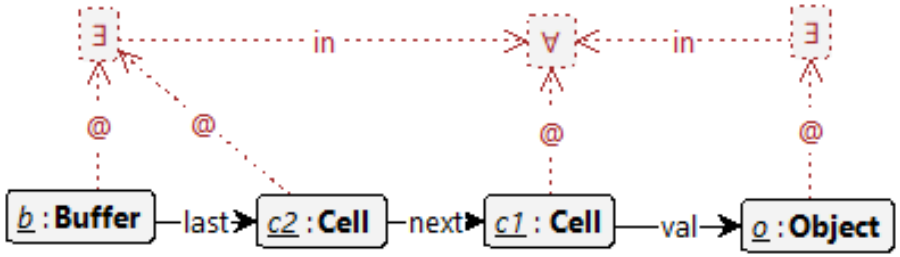
Keine verzweigenden next-Kanten



Freie nächste Zelle



Jede Zelle hat eine nächste



Alle Zellen sind belegt, außer möglicherweise der nächsten

Definition: Implikation und Äquivalenz

Eine Graphbedingung c_1 impliziert eine Graphbedingung c_2, notiert als $c_1 \Longrightarrow c_2$, wenn für jeden Morphismus p mit $p \vDash c_1$ auch $p \vDash c_2$ gilt.

Zwei Graphbedingungen c_1 und c_2 sind äquivalent, notiert als $c_1 \equiv c_2$ oder auch als $c_1 \Longleftrightarrow c_2$, wenn sowohl $c_1 \Longrightarrow c_2$ als auch $c_2 \Longrightarrow c_1$ gilt.

Beispiele Äquivalenzen

Es seien a, b Morphismen, ι ein Isomorphismus, c, c_j Graphbedingungen (über den jeweils passenden Graphen), und $C_1 \subseteq C_2$ Graphen. Dann gelten die folgenden Äquivalenzen [Pennemann09]:

$$\exists\left(a, \bigvee_{j \in J} c_j\right) \equiv \bigvee_{j \in J} \exists\left(a, c_j\right)$$
$$\exists(\iota, c) \equiv c$$
$$\exists(a, \text{false}) \equiv \text{false}$$
$$\exists\left(a, \exists(b, c)\right) \equiv \exists(b \circ a, c)$$
$$\exists C_1 \vee \exists C_2 \equiv \exists C_1$$

Ausdrucksmächtigkeit

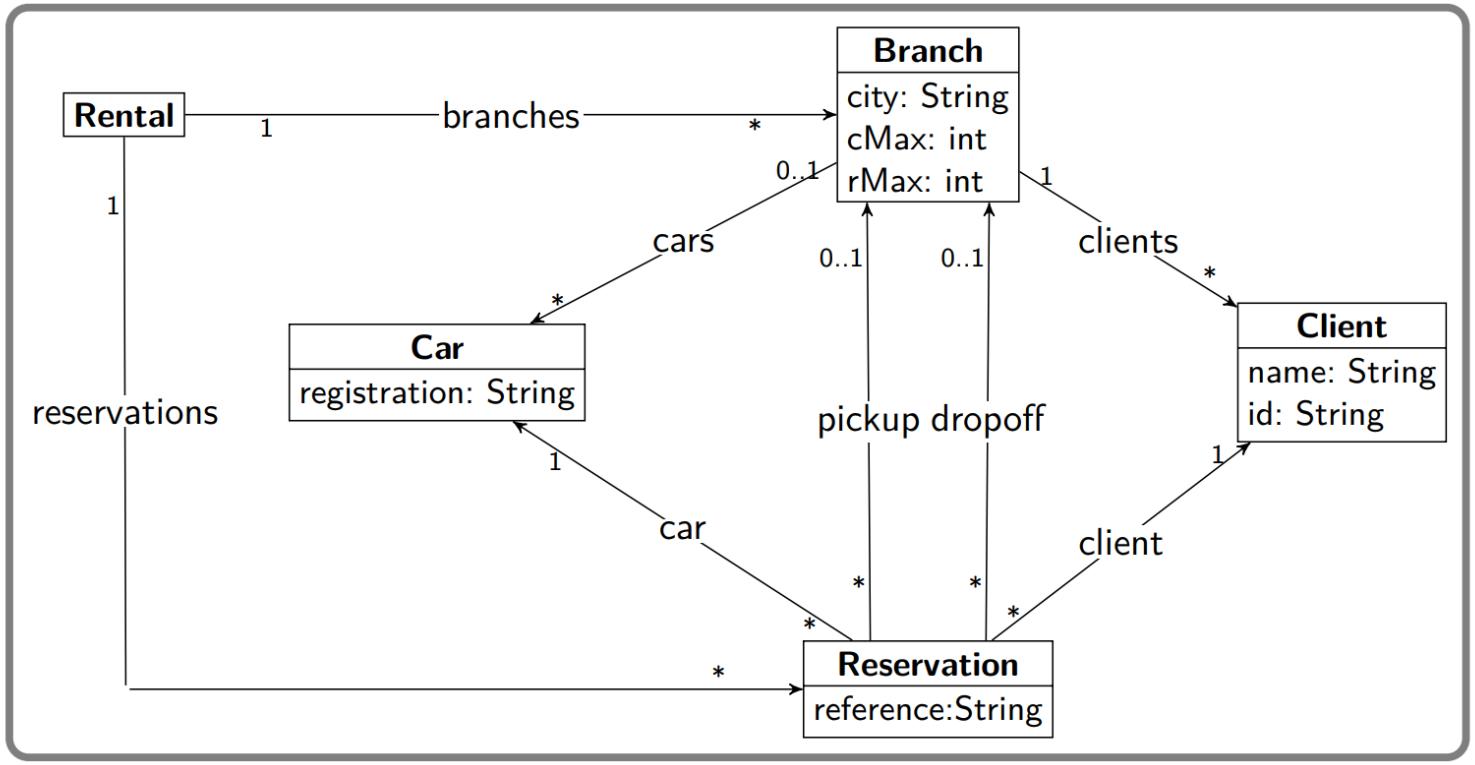
Geschachtelte Graphformeln sind gleich ausdrucksmächtig wie die Logik erster Ordnung auf Graphen [HP09, Theoreme 10 und 11; Rensink04, Theoreme 1 und 3].

- Erfüllbarkeitsproblem, Tautologieproblem und Implikationsproblem auf geschachtelten Graphformeln sind unentscheidbar.
- Eigenschaften wie Zusammenhang oder Kreisfreiheit sind nicht ausdrückbar.

Vererbung und Attribute

- Vererbung kann gut mit geschachtelten Graphbedingungen kombiniert werden.
  - *Die Morphismen in den Bedingungen dürfen dann „heruntertypen“*
  - *Die Auswertung der Gültigkeit von Graphbedingungen erfolgt über Morphismen, die „heruntertypen“ dürfen*
- Bedingungen über Attributwerte können oft als geschachtelte Graphbedingungen dargestellt werden, aber das ist aufwändig und nicht intuitiv.
  - *Ausdruck von geforderten Eigenschaften auf Attributwerten durch spezieller (auf Prädikatenlogik basierender) Sprache*
  - *Annotation von Regeln durch prädikatenlogische Formeln, die Bedingungen auf Attributwerten ausdrücken*

Typgraph mit Multiplizitäten



Definition: Typgraph mit Multiplizitäten

Ein Typgraph mit Multiplizitäten ist ein Tupel $TGM = (TGI, m_n, m_s, m_t)$ mit

- einem Typgraph TGI mit Vererbung,
- Funktion $m_n: N_{TGI} \to \mathrm{Mult}$ für Knotenmultiplizitäten und
- Funktionen $m_s, m_t: E_{TGI} \to \mathrm{Mult}$ für Kantenmultiplizitäten.

Eine Multiplizität ist ein Paar $[i, j] \in \mathbb{N} \times (\mathbb{N} \cup \{*\})$ mit $i \leq j$ oder $j = *$.

- Die Menge aller Multiplizitäten ist Mult.
- * ist ein spezieller Wert für eine nicht vorhandene obere Schranke.
- Für eine Menge X gilt: $X \in [i, j]$, falls $i \leq |X|$ und $|X| \leq j$ oder $j = *$.

Definition: Semantik eines Typgraphs mit Multiplizitäten

Ein Graph $G = (N_G, E_G, s_G, t_G)$ erfüllt einen Typgraph $TGM = (TGI, m_n, m_s, m_t)$ mit Multiplizitäten, falls es einen Clan-Morphismus $\text{type}: G \to TGI$ gibt, sodass

- für alle $n \in N_{TGI}$ gilt: $|\cup_{x \in \text{clan}(n)} \text{inst}_N(x)| \in m_N(n)$;
- für alle $e \in E_{TGI}$ und $p \in \cup_{x \in \text{clan}(n)} \text{inst}_N(x)$ mit $n = s_G(e)$ gilt:
$$\left|inst_E(e) \cap s_G^{-1}(p)\right| \in m_t(e);$$
- für alle $e \in E_{TGI}$ und $p \in \cup_{x \in \text{clan}(n)} \text{inst}_N(x)$ mit $n = t_G(e)$ gilt:
$$\left|inst_E(e) \cap t_G^{-1}(p)\right| \in m_s(e);$$

Hierbei sind inst die zum Clan-Morphismus type inversen Funktionen $\text{inst}_N: N_{TGI} \to \wp(N_G)$ und $\text{inst}_E: E_{TGI} \to \wp(E_G)$
(und $\wp$ ist die *Potenzmenge).*

Übersicht

- Wie formalisieren wir gewünschte bzw. verbotene strukturelle Eigenschaften von Graphen und Graphmorphismen?
  - *Geschachtelte Graphbedingungen als (visuelle) Logik auf Graphen*
  - *Multiplizitäten als einfacher Spezialfall*
- Wie stellen wir sicher, dass das Ergebnis einer Graphtransformation gewünschte Eigenschaften hat?
  - *Automatische Berechnung von Anwendungsbedingungen aus Graphbedingung und Regel*

Inspiration: Hoare Logik

- Ein Hoare Tripel hat die Form $\{P\}C\{Q\}$. Hierbei sind P und Q logische Aussagen (assertions) und C ein Befehl (command).
- Wenn die Vorbedingung P erfüllt ist, ist garantiert, dass nach der Ausführung von C die Nachbedingung Q gilt.
- Mit Hilfe von Inferenzregeln kann man (rückwärts) für die Korrektheit von Programmen argumentieren.

$$\overline{\{P[E/x]\}x := E\{P\}}$$

(Zuweisungsaxiom)

$$\frac{\{P\}S\{Q\} \quad , \quad \{Q\}T\{R\}}{\{P\}S;T\{R\}}$$

(Sequenzregel)

Garantierende Anwendungsbedingungen

Ziel: Konstruktion einer Anwendungsbedingung, sodass garantiert ist, dass die Regel nur anwendbar ist, wenn eine gegebene Graphformel nach der Anwendung erfüllt ist.

Konstruktionsidee:

1. Überlappe jeden Graphen einer Graphformel auf jede mögliche Weise mit der rechten Seite einer Regel. Dies liefert einen Überblick über die möglichen Interaktionen zwischen der Regelanwendung und der Formel.
2. Wende die Regel rückwärts auf die Ergebnisse an. Dies liefert dann den Überblick, welche Vorbedingungen herrschen müssen, damit nach Regelanwendung die Formel erfüllt ist.
3. Interpretiere dieses Ergebnis als Anwendungsbedingung für die Regel.

Definition: Graphregel mit allgemeiner Anwendungsbedingung

Gegeben sei ein Typgraph TG. Eine getypte Graphregel (mit allgemeiner Anwendungsbedingung) ist ein Tupel $r = (L, R, AC)$, wobei:

- *L ist ein getypter Graph ($\underline{L}$, $type_L$)*
- *R ist ein getypter Graph ($\underline{R}$, $type_R$)*
- *$K = L \cap R$ ist ein getypter Graph, $type_L \supseteq type_K \subseteq type_R$*
- *AC ist eine geschachtelte Graphbedingung über L (Anwendungsbedingung)*

Die Anwendung ist genau definiert wie bisher, nur dass der Ansatz m nicht mehr die Menge der NACs sondern die Anwendungsbedingung AC erfüllen muss.

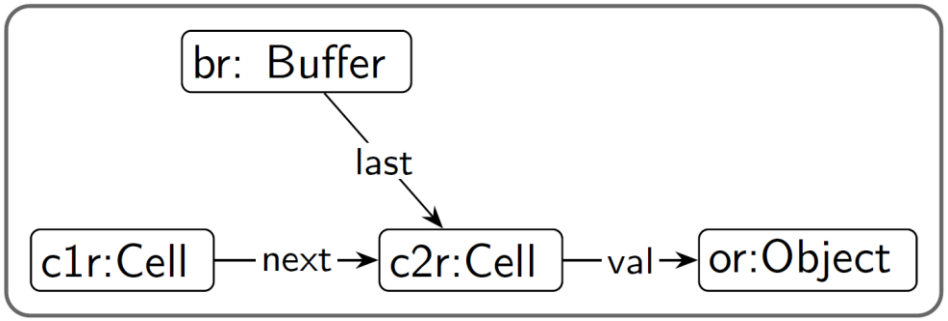
Definition: Überlappung von Graphen

Gegeben sei ein Typgraph TG (ohne Vererbung) und zwei über TG getypte Graphen R und C, sodass C keine Attribute enthält. Eine Überlappung von R und C ist ein Graph O mit zwei injektiven Morphismen $i_1: R \to O$ und $i_2: C \to O$, die gemeinsam surjektiv sind.

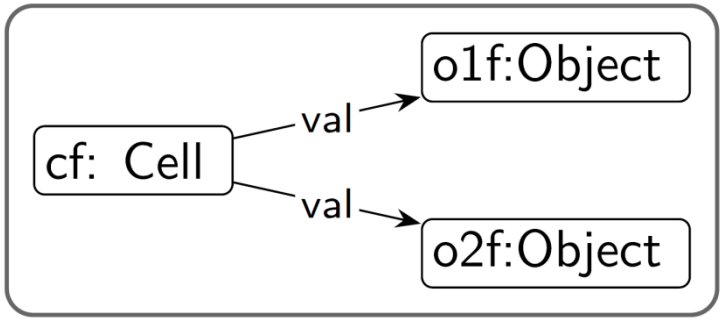
Gegeben zwei injektive Morphismen $a_1: X \to C$ und $a_2: X \to R$, so ist eine Überlappung von R und C über X ein Graph O mit zwei injektiven Morphismen $i_1: R \to O$ und $i_2: C \to O$, die gemeinsam surjektiv sind und für die $i_1 \circ a_2 = i_2 \circ a_1$ gilt.

Gemeinsam surjektiv: Jedes Element aus O hat ein Urbild in R oder in C unter i_1, i_2.

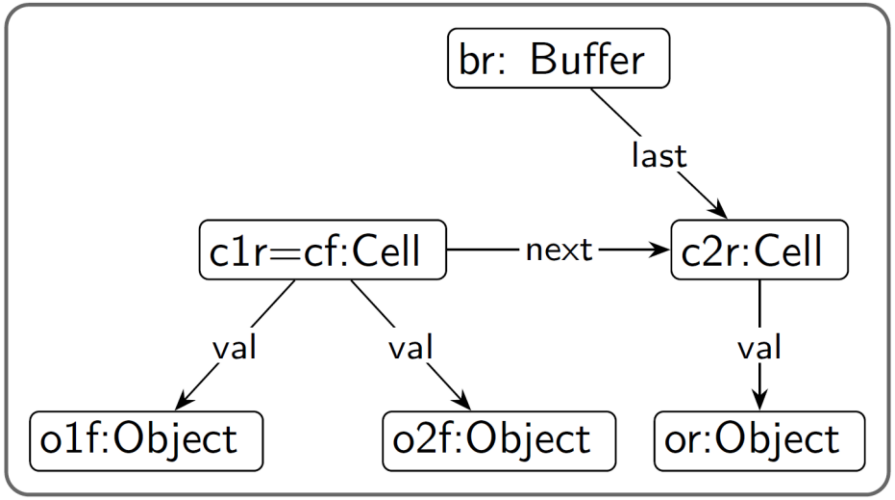
Beispiel



RHS put-Regel

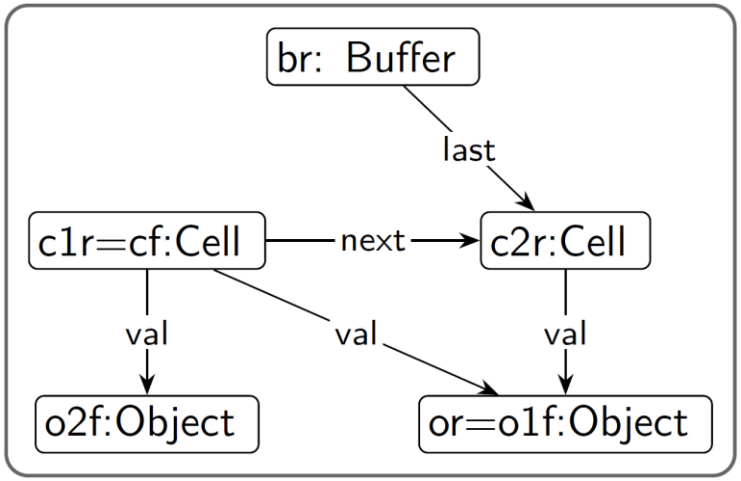


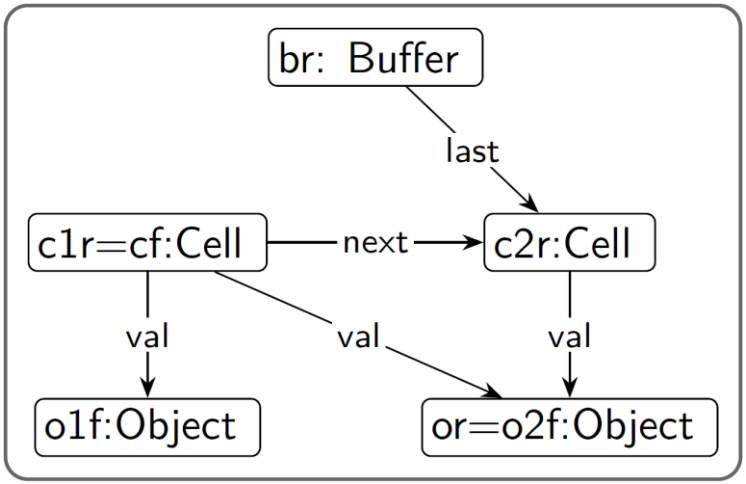
Graph „Keine zwei Objekte“

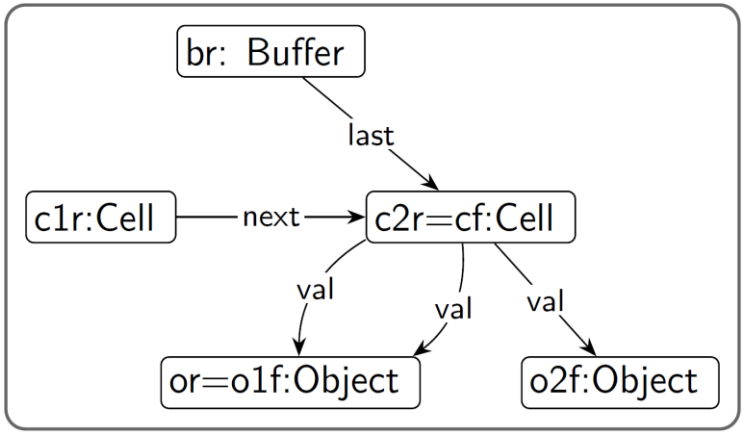


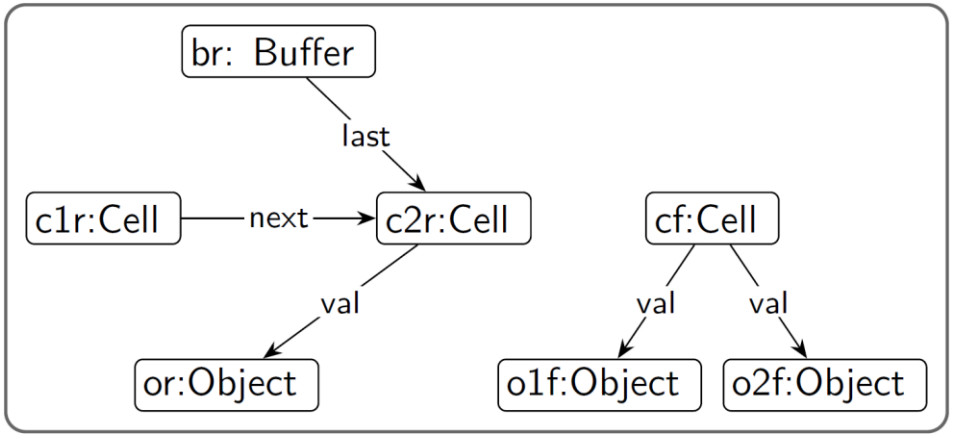
Eine Überlappung

Weitere Überlappungen





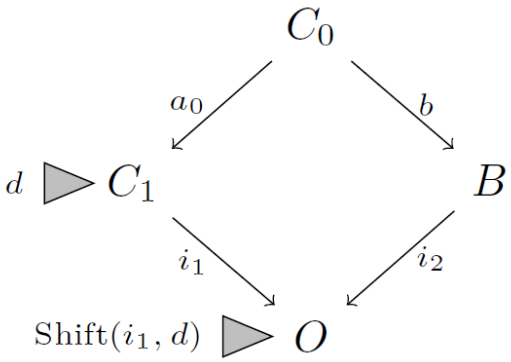




Definition: Verschieben von Graphbedingungen entlang von Morphismen

Gegeben einen injektiven Morphismus $b: C_0 \to B$ und eine Graphbedingung c über C_0, dann ist die entlang b verschobene Graphbedingung, notiert als *Shift(b, c)*, über *B* wie folgt definiert:

- Gilt $c = \top$, so gilt auch Shift$(b, c) = \top$.
- Gilt $c = \exists(a_0: C_0 \to C_1, d)$, so gilt Shift$(b, c) = \bigvee_{(i_1, i_2) \in F} \exists(i_2, \text{Shift}(i_1, d))$, wobei F die Menge aller Überlappungen von C_1 und B über C_0 ist.



- Gilt $c = \neg d$, so gilt Shift$(b, c) = \neg$Shift(b, d)
- Gilt $c = c_1 \vee c_2$, so gilt $\text{Shift}(b, c) = \text{Shift}(b, c_1) \vee \text{Shift}(b, c_2)$

Korrektheit von Shift

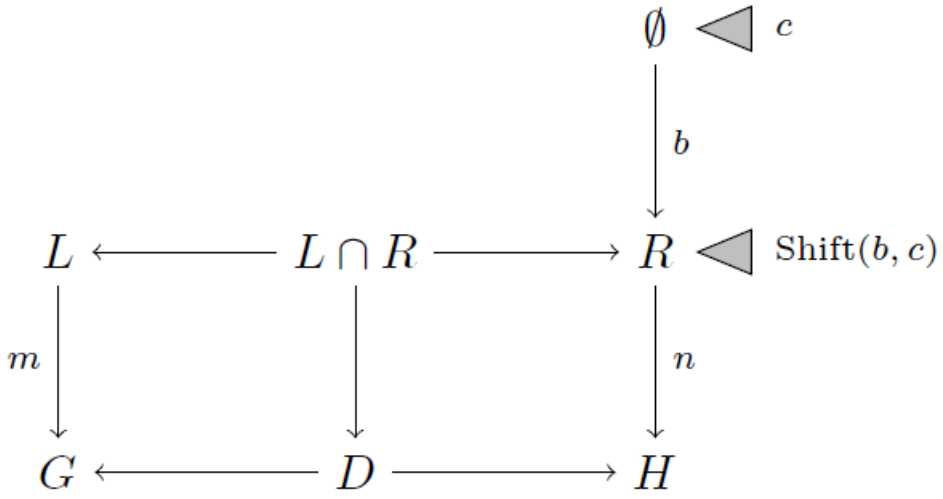
Für jede Graphbedingung c über einem Graphen C_0, jeden injektiven Morphismus $b: C_0 \to B$ und jeden injektiven Morphismus $g: B \to G$ gilt:

$$g \vDash \text{Shift}(b, c) \Leftrightarrow g \circ b \vDash c.$$

[EGHLO14, Lemma 3.11]

Korollar zur Korrektheit von Shift

Gegeben eine Regel $r = (L, R)$ und eine Graphformel c, dann erfüllt der Ko-Ansatz $n: R \to H$ nach einer Regelanwendung die Graphbedingung $\mathrm{Shift}(\emptyset \to R, c)$ genau dann, wenn der Graph H die Formel c erfüllt.



Definition: Verschieben von Graphbedingungen entlang von Regeln (Teil 1)

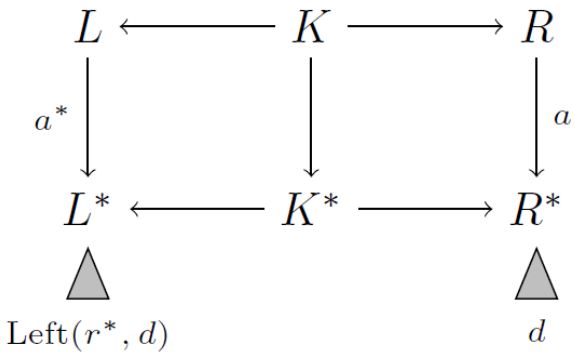
Gegeben eine Regel $r = (L, R)$ und eine Graphbedingung c über R, dann ist die entlang r verschobene Graphbedingung, notiert als $\mathrm{Left}(r, c)$, über L wie folgt definiert:

- Gilt $c = \top$, so gilt $\mathrm{Left}(r, c) = \top$.
- Gilt $c = \neg d$, so gilt $\mathrm{Left}(r, c) = \neg \mathrm{Left}(r, d)$.
- Gilt $c = c_1 \vee c_2$, so gilt $\mathrm{Left}(r, c) = \mathrm{Left}(r, c_1) \vee \mathrm{Left}(r, c_2)$.

Definition: Verschieben von Graphbedingungen entlang von Regeln (Teil 2)

Gegeben eine Regel $r = (L, R)$ und eine Graphbedingung c über R, dann ist die entlang r verschobene Graphbedingung, notiert als $\text{Left}(r, c)$, über L wie folgt definiert:

- Gilt $c = \exists(a, d)$, so gilt
$$\text{Left}(r, c) = \exists(a^*, \text{Left}(r^*, d))$$
falls die inverse Regel $r^{-1} = (R, L)$ am Ansatz a auf den Graphen R^* anwendbar ist (hierbei ist $r^* = (L^*, R^*)$). Sonst gilt $\text{Left}(r, c) = \text{false}$.

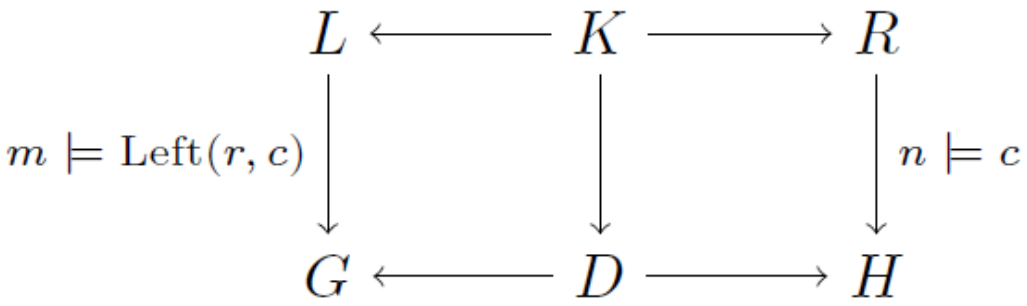


Korrektheit von Left

Für jede Regel $r = (L, R)$, Graphbedingung c über R und Anwendung der Regel r mit Ansatz m und Ko-Ansatz n gilt:

$$m \models \text{Left}(r, c) \Leftrightarrow n \models c.$$

[EGHLO14, Lemma 3.14]



Definition: Garantierende Anwendungsbedingung

Gegeben eine Regel $r = (L, R)$ und eine Graphformel c, dann ist die garantierende Anwendungsbedingung (für r bezüglich c) die Bedingung

$$\mathrm{ac_g} = \mathrm{Left}\big(r, \mathrm{Shift}(\emptyset \to R, c)\big)$$

über L.

Für die Regel $r_\mathrm{g} = (L, R, \mathrm{ac_g})$ gilt dann: Existiert eine Transformation $G \Longrightarrow_{r,m} H$, so existiert eine Transformation $G \Longrightarrow_{r_\mathrm{g},m} H$ genau dann, wenn $H \vDash c$ gilt. [HP09, Corallary 5]

Zusammenfassung

- Geschachtelte Graphbedingungen und -formeln bilden eine visuelle Logik mit präziser Semantik auf Graphen.
  - *Bedingungen drücken Eigenschaften von Morphismen aus, Formeln Eigenschaften von Graphen.*
  - *Die Logik ist eine Logik erster Ordnung auf Graphen.*
  - *Multiplizitäten sind praktisch wichtiger Spezialfall mit eigener Definition und Notation.*
- Graphformeln können als Anwendungsbedingungen in Regeln integriert werden.
  - *Die resultierenden Regeln sind nur noch in Situationen anwendbar, in denen das Ergebnis der Regelanwendung die Formel erfüllt.*
  - *Die entstehenden Anwendungsbedingungen sind oft unnötig komplex.*

Ausblick

- Es gibt auch eine Konstruktion, die bewahrende Anwendungsbedingungen berechnet, also Anwendungsbedingungen, die unter der Voraussetzung, dass der Input eine Graphformel erfüllt, garantieren, dass der berechnete Output die Bedingung auch erfüllt. [HP09]
- Für Spezialfälle gibt es Optimierungen, die eine logisch äquivalente, aber strukturell einfachere garantierende Anwendungsbedingung erzeugen. [HW95,BGH20,NKAT20]
- Es ist auch möglich, eine mehrwertige Logik auf Graphen einzuführen und dann zu untersuchen, wann Regelanwendungen zumindest keine neuen Verletzungen von Formeln einführen. [KSTZ22]

Literatur

- [BGH20] Nicolas Behr, Maryam Ghaffari Saadat, Reiko Heckel: Commutators for Stochastic Rewriting Systems: Theory and Implementation in Z3. GCM@STAF 2020: 126–144
- [EGHLO14] Hartmut Ehrig, Ulrike Golas, Annegret Habel, Leen Lambers, Fernando Orejas: $\mathcal{M}$-adhesive transformation systems with nested application conditions. Part 1: parallelism, concurrency and amalgamation. Math. Struct. Comput. Sci. 24(4) (2014)
- [HP09] Annegret Habel, Karl-Heinz Pennemann: Correctness of high-level transformation systems relative to nested conditions. Math. Struct. Comput. Sci. 19(2): 245–296 (2009)
- [HW95] Reiko Heckel, Annika Wagner: Ensuring consistency of conditional graph rewriting – a constructive approach. SEGRAGRA 1995: 118–126
- [KSTZ22] Jens Kosiol, Daniel Strüber, Gabriele Taentzer, Steffen Zschaler: Sustaining and improving graduated graph consistency: A static analysis of graph transformations. Sci. Comput. Program. 214 (2022)
- [NKAT20] Nebras Nassar, Jens Kosiol, Thorsten Arendt, Gabriele Taentzer: Constructing optimized constraint-preserving application conditions for model transformation rules. J. Log. Algebraic Methods Program. 114 (2020)
- [Pennemann09] Karl-Heinz Pennemann: Development of correct graph transformation systems. Dissertation. University of Oldenburg, Germany, 2009
- [Rensink04] Arend Rensink: Representing First-Order Logic Using Graphs. ICGT 2004: 319–335
- [TR05] Gabriele Taentzer, Arend Rensink: Ensuring Structural Constraints in Graph-Based Models with Type Inheritance. FASE 2005: 64–79