Geschachtelte Graphbedingungen und Hoare-Style Verifikation von Graphtransformationsregeln

Jens Kosiol

10. und 17. Juni 2024

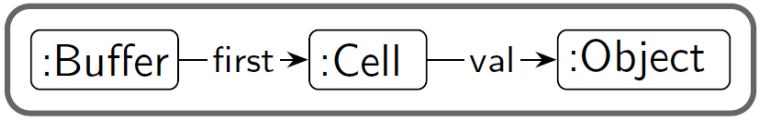
Übersicht

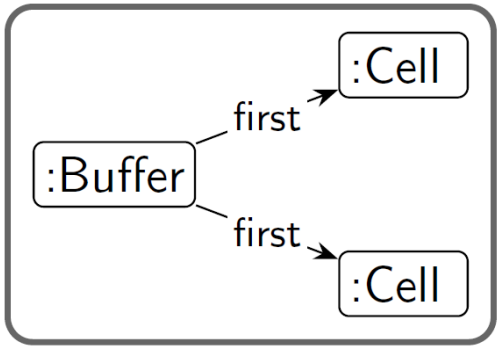
- Wie formalisieren wir gewünschte bzw. verbotene strukturelle Eigenschaften von Graphen und Graphmorphismen?
  - *Geschachtelte Graphbedingungen als (visuelle) Logik auf Graphen*
  - *Multiplizitäten als einfacher Spezialfall*
- Wie stellen wir sicher, dass das Ergebnis einer Graphtransformation gewünschte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

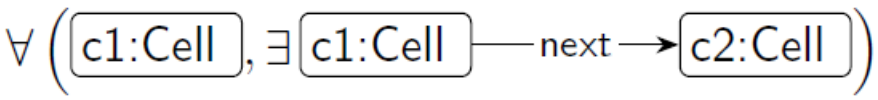
Dieser Graph matcht einen Graphen G genau dann nicht, wenn G keinen Buffer mit zwei ausgehenden first-Kanten enthält

Was fehlt?

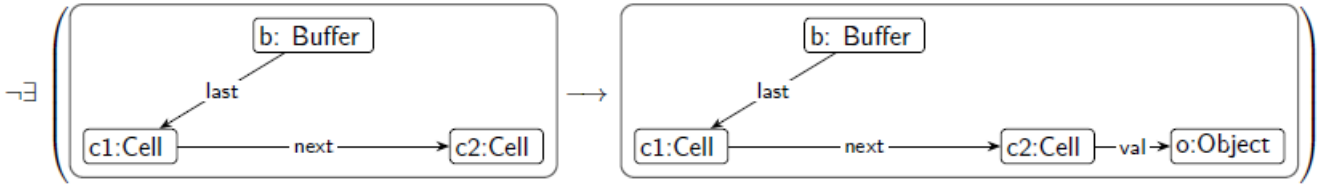
- Ausdrücken weiterer Eigenschaften über gefundene Pattern („Es gibt einen Buffer, der leer ist.“, „Für jede Zelle gilt, dass eine next-Kante von ihr ausgeht.“)
  - *Verketten von Pattern, wobei die tieferen Pattern weiter spezifizieren, welche Eigenschaften die früheren Pattern haben sollen*
  - *Formales 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

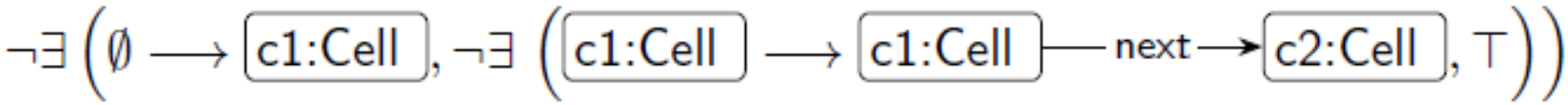
Gegeben sei ein Typgraph TG. Eine (geschachtelte) Graphbedingung über einem (über TG getypten) Graphen C_0 ist induktiv wie folgt definiert:

- 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.

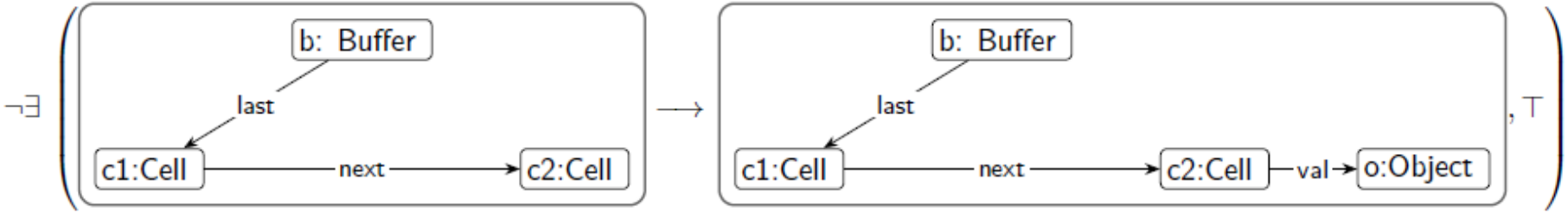
Eine (geschachtelte) Graphformel ist eine Graphbedingung über dem leeren Graphen ∅.

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.



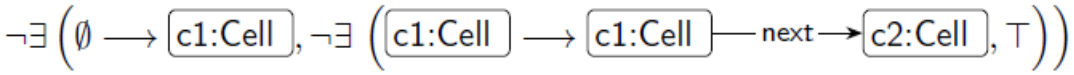
Vereinfachungen der Darstellung

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

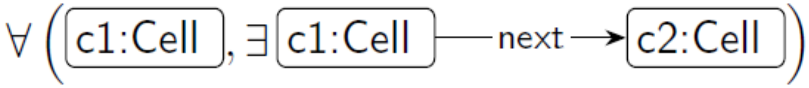
- Weglassen von $\top$ am Ende einer Graphbedingung (sofern nicht direkt davor eine Negation steht)
- Weglassen der Startgraphen der Morphismen aus denen die Graphbedingung besteht (abgesehen vom ersten Graphen falls $\neq \emptyset$)
- Kurzschreibweise $\forall(a_0, c)$ für $\neg\exists(a_0, \neg c)$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

Beispiel

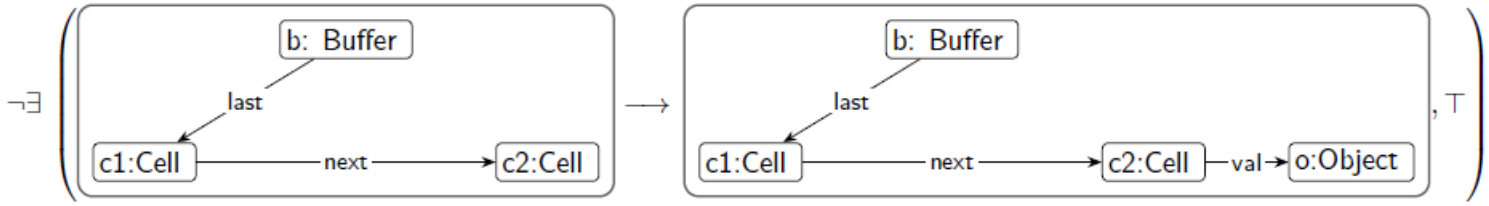
- Die Graphformel



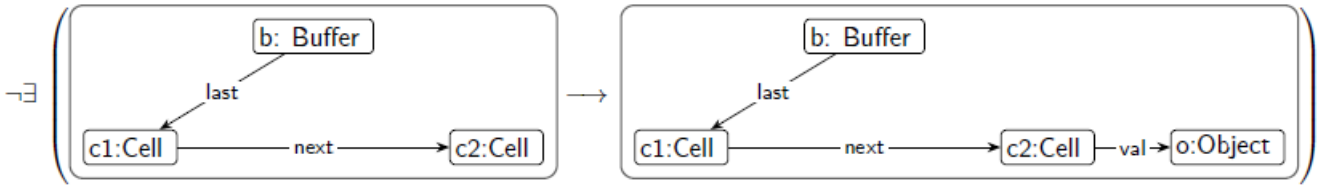
vereinfachen wir zu



- Die Graphbedingung



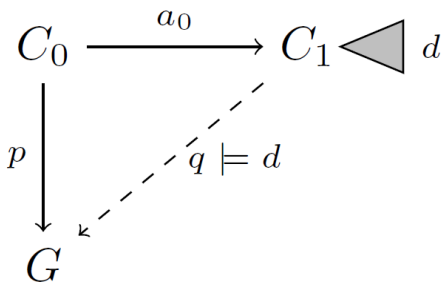
vereinfachen wir zu



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.

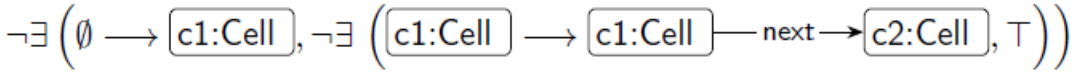
Beispiel abstrakt

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

- Ein Graph G erfüllt die Formel c, wenn er die Formel $\mathrm{d} = \exists(a_0\colon \emptyset \to C_1, \neg\exists(a_1\colon C_1 \to C_2))$ *nicht* erfüllt
- Ein Graph G erfüllt die Formel d, wenn eine Abbildung $q\colon C_1 \to G$ existiert, die die Graphbedingung $e = \neg\exists(a_1\colon C_1 \to C_2)$ 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



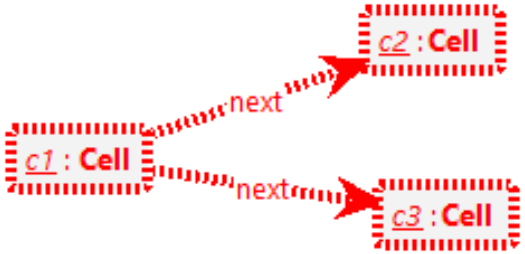
falls

- 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.

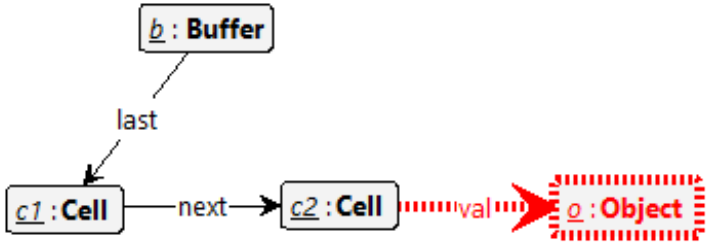
Graphbedingungen in Groove

- Integrierte Darstellung als ein Graph mit Annotationen
  - *Quantorenknoten markieren, auf welche Ebene ein Element gehört*
  - *Die Quantoren bilden einen Baum*
  - *Negation durch die Negation der einzelnen Knoten*
  - *Formalisiert in [Rensink04]*

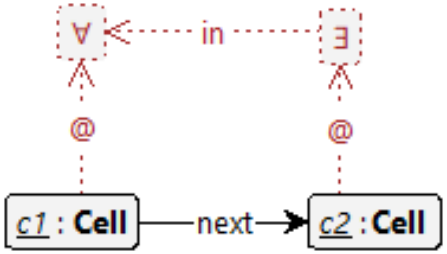
Beispiele



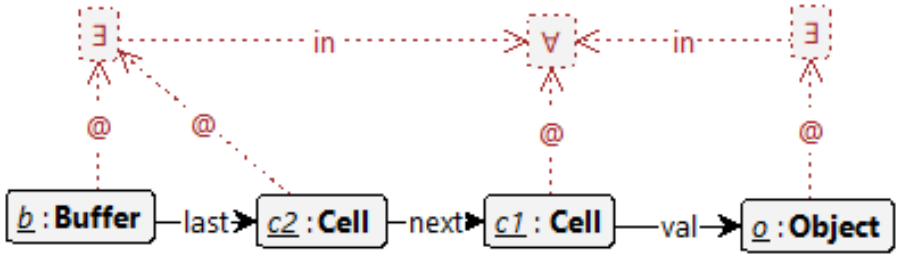
Keine verzweigenden next-Kanten



Freie nächste Zelle



Jede Zelle hat eine nächste



Alle Zellen sind belegt, außer möglicherweise der nächsten

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 $c_1 \Leftrightarrow c_2$, wenn sowohl $c_1 \Longrightarrow c_2$ als auch $c_2 \Longrightarrow c_1$ gilt.

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

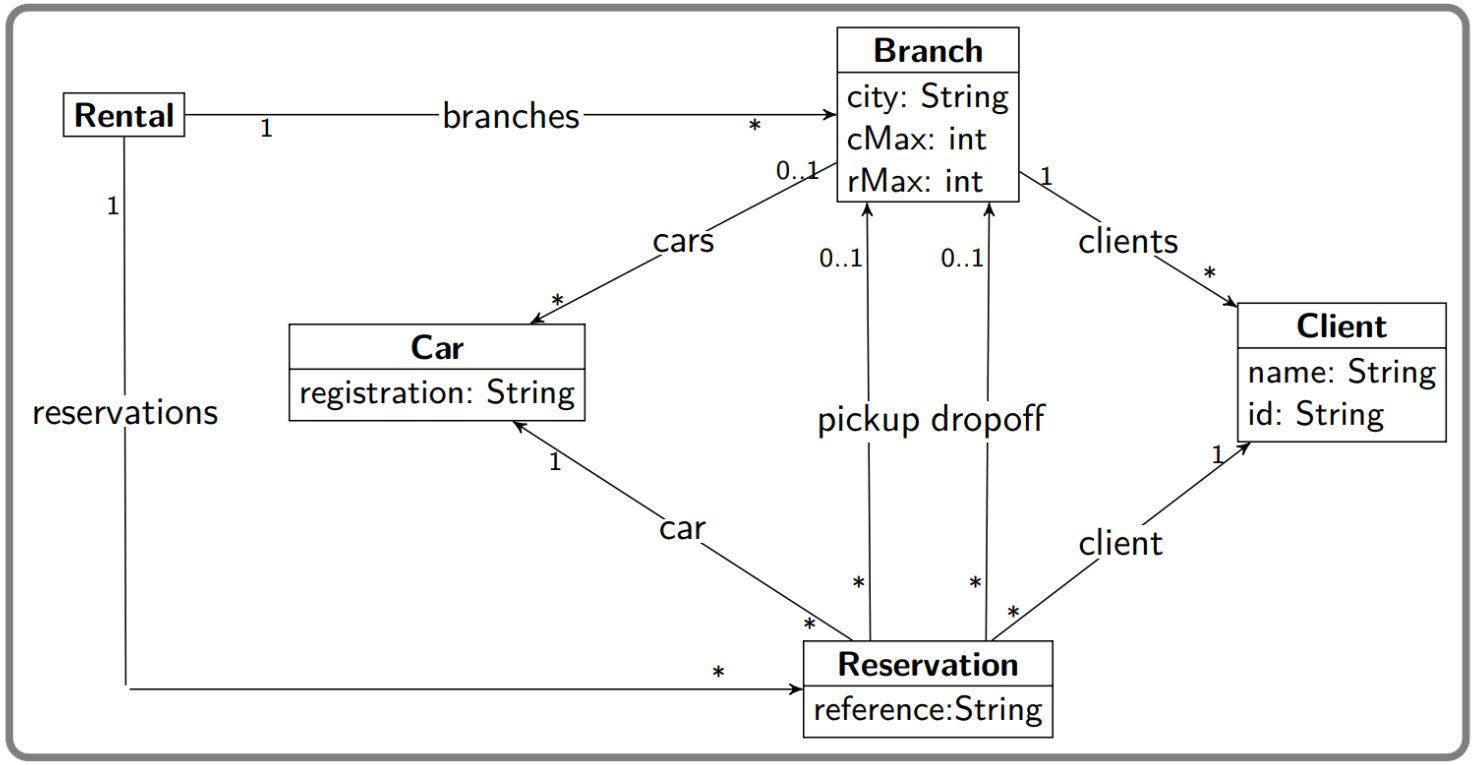
Geschachtelte Graphformeln sind gleich ausdrucksmächtig wie die Logik erster Ordnung auf Graphen [HP09, Theoreme 10 und 11; Rensink04, Theoreme 1 und 3].

- 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 unintuitiv.
  - *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 Multiplizität

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: $|\cup_{x \in \text{clan(n)}} \text{inst}_\text{N}(x)| \in m_N(n)$;
- für alle $e \in E_{TGI}$ und $p \in \cup_{x \in \text{clan}(n)} \text{inst}_\text{N}(x)$ mit $x = s_G(e)$ gilt:
$$\left|inst_E(e) \cap s_G^{-1}(p)\right| \in m_t(e);$$
- für alle $e \in E_{TGI}$ und $p \in \cup_{x \in \text{clan}(n)} \text{inst}_\text{N}(x)$ mit $x = t_G(e)$ 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 $\text{inst}_\text{N}: N_{TGI} \to \wp(N_G)$ und $\text{inst}_\text{E}: E_{TGI} \to \wp(E_G)$
(und $\wp$ ist die *Potenzmenge)*