Weitere Konzepte für Graphtransformationssysteme – Attributierung und Vererbung

Jens Kosiol

6. und 8. Mai 2024

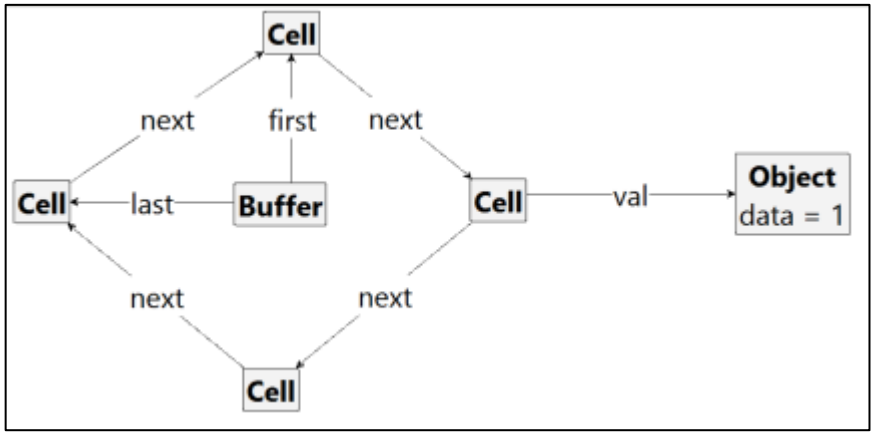
Überblick

- Wie werden Objekte mit Attributen modelliert?
  - *Attributierte Graphen und Graphtransformation*
- Wie lässt sich Vererbung formal fassen?
  - *Abstrakte Klassen*
  - *Vererbungshierarchie zwischen Knoten*

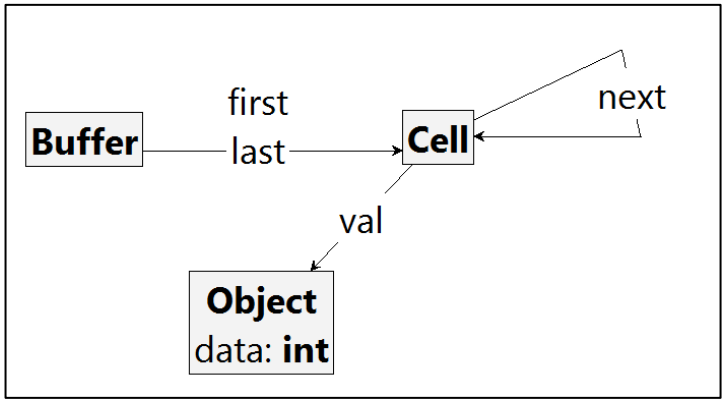
Attributierte getypte Graphtransformation

- Eine Klasse spezifiziert nicht nur Objektreferenzen, sondern auch Attribute.
- Ein Typgraph definiert alle beteiligten Objekt- und Datentypen sowie Referenzen und Attribute.
- Ein attributierter getypter Graph hat Objektknoten und Datenknoten.
  - *Datenknoten stellen Datenwerte dar.*
  - *Kanten von Objektknoten zu Datenknoten attributieren Objekte.*
- Regeln dürfen Datenvariablen benutzen und können Rechnungen auf Attributwerten spezifizieren.
- Regeln können Input- und Output-Parameter haben.

Beispiel: Attributierter getypter Graph



Attributierter Graph



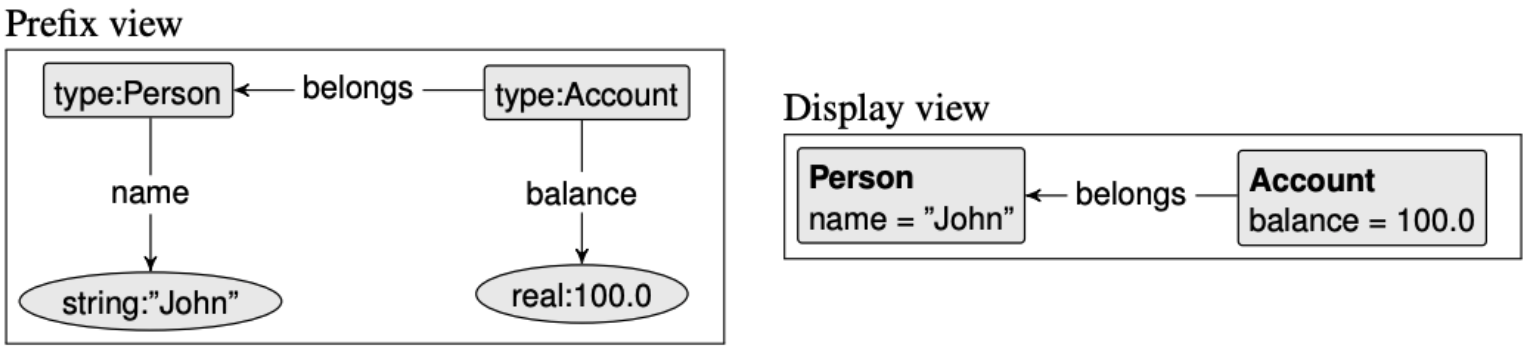
Attributierter Typgraph

Definition: Erweiterter Graph

Ein erweiterter Graph (E-Graph) G ist ein Tupel $G = (N, E, D, A, s_E, t_E, s_A, t_A)$:

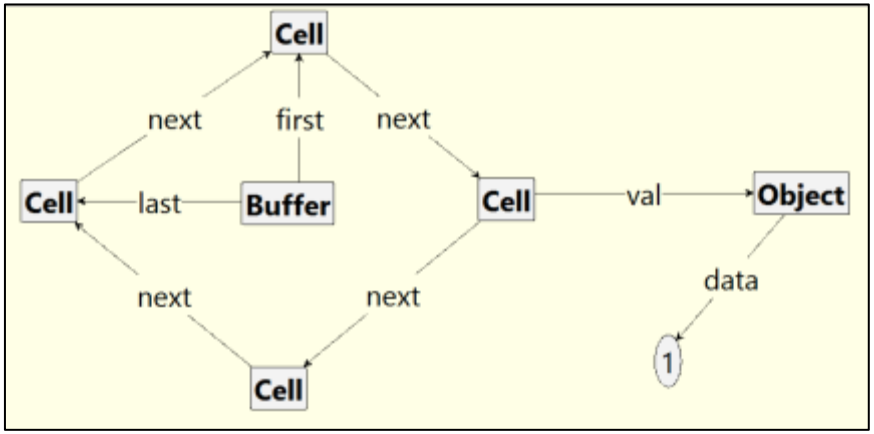
- N ist eine Menge von Objektknoten.
- E ist eine Menge von Objektkanten.
- D ist eine Menge von Datenknoten.
- A ist eine Menge von Attributierungskanten.
- Die Funktionen $s_E: E \to N$ und $t_E: E \to N$ weisen Objektkanten ihre Quell- bzw. Zielknoten zu.
- Die Funktion $s_A: A \to N$ weist Attributierungskanten ihre Objektknoten zu.
- Die Funktion $t_A: A \to D$ weist Attributierungskanten ihre Datenwerte zu.

Beispiel: Attributierter getypter Graph in zwei Sichten



[Groove Manual]

Beispiel: Attributierter getypter Graph

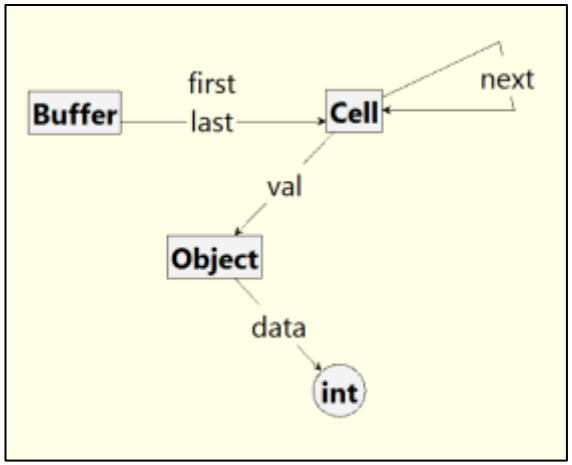


Repräsentation im Groove-Editor entspricht der formalen Spezifikation

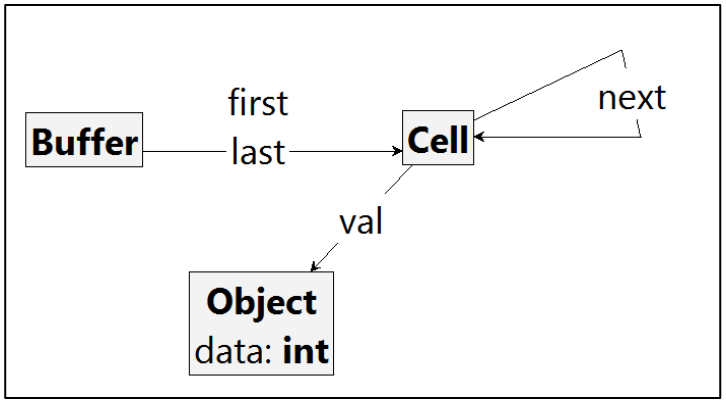
Sig =
sorts int
opns 0: → int
+: int int → int

Alg(Sig):
D_{int} ist die Menge aller ganzen Zahlen.

Beispiel: Attributierter Typgraph



Repräsentation im Groove-Editor entspricht der formalen Spezifikation



Endansicht des Typgraphen

Definition: Attributierter getypter Graph

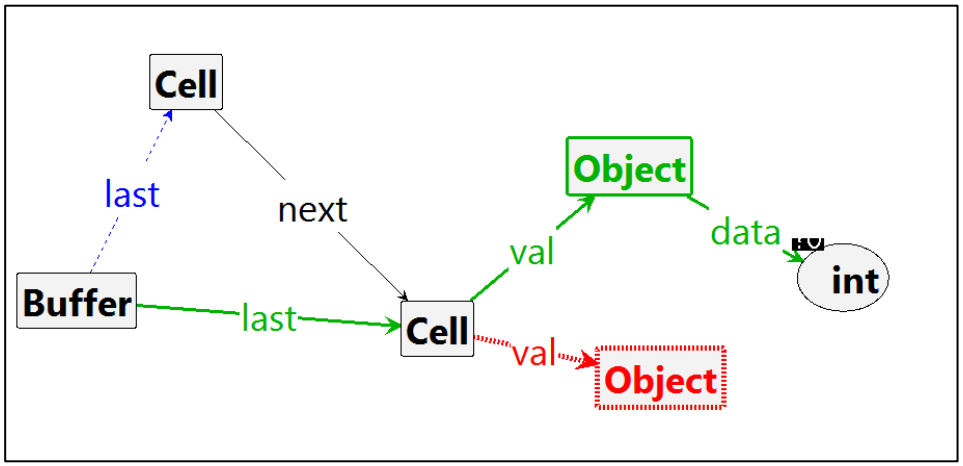
Gegenben sei eine Signatur Σ, die die gewünschten Datentypen (String, Int, Bool, …) und deren Operationen festlegt.

Ein attributierter Typgraph TG ist ein E-Graph, in dem ein einziger Datenknoten für jeden benötigten Datentyp existiert.
(Formal: Die gewählte Datenalgebra ist die finale Σ-Algebra.)

Ein attributierter getypter Graph über TG ist ein Tupel $G = (\overline{G}, \mathrm{type_G})$, wobei $\overline{G}$ ein E-Graph ist und $\mathrm{type_G}: \overline{G} \to TG$ ein E-Graph-Morphismus.

- Die Datenknoten in $\overline{G}$ bilden eine Σ-Algebra.
- Ein E-Graph-Morphismus ist definiert wie ein Graphmorphismus und bildet zusätzlich die Datenknoten homomorph ab.

Attributierte Graphtransformation

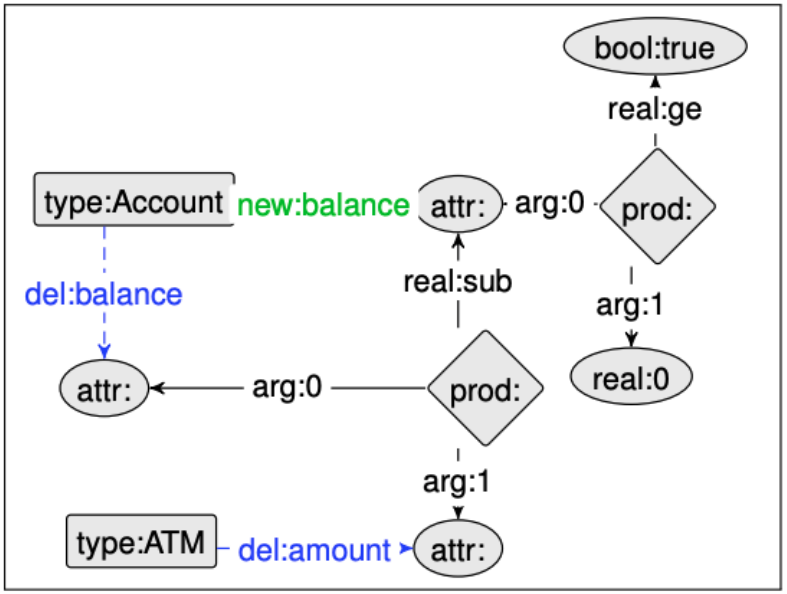


Beispiel: put(int)

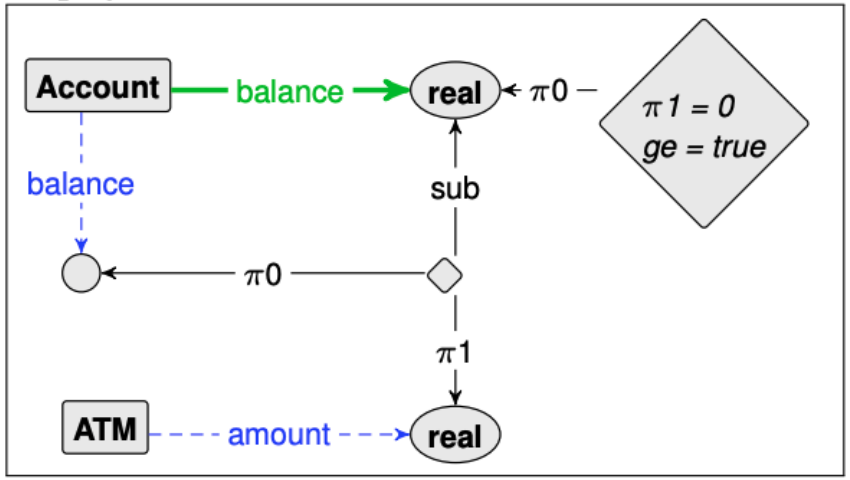
- Graphtransformation auf attributierten Graphen funktioniert weitgehend analog zur Transformation gewöhnlicher Graphen.
- Regeln löschen oder erzeugen keine Datenwerte, sondern nur Graphstruktur und Attributierungskanten.
- Mit Regeln lassen sich auch Berechnungen auf Attributwerten ausdrücken:
  - Die Datenalgebra in den Graphen der Regel kann eine Termalgebra über einer Variablenmenge V sein. (Ein Term ist ein Ausdruck über den Konstanten und Operationen der Signatur und der Menge V).
  - Ein Regelansatz belegt dann alle Variablen aus V mit konkreten Werten.
  - Eine Regelanwendung wertet alle Terme der Regel mit der aktuellen Variablenbelegung zu konkreten Werten aus.

Beispiel: Attributierte Graphregel in zwei Sichten

Prefix view



Display view



[Groove Manual]

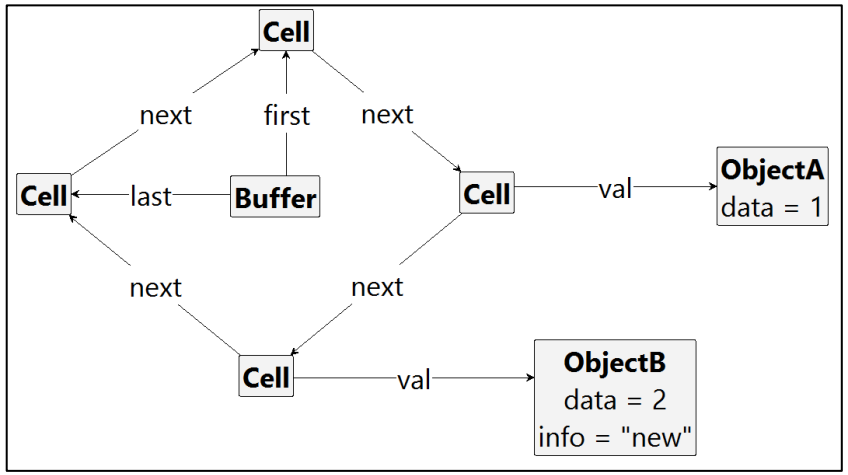
Überblick

- Wie werden Objekte mit Attributen modelliert?
  - *Attributierte Graphen und Graphtransformation*
- Wie lässt sich Vererbung formal fassen?
  - *Abstrakte Klassen*
  - *Vererbungshierarchie zwischen Knoten*

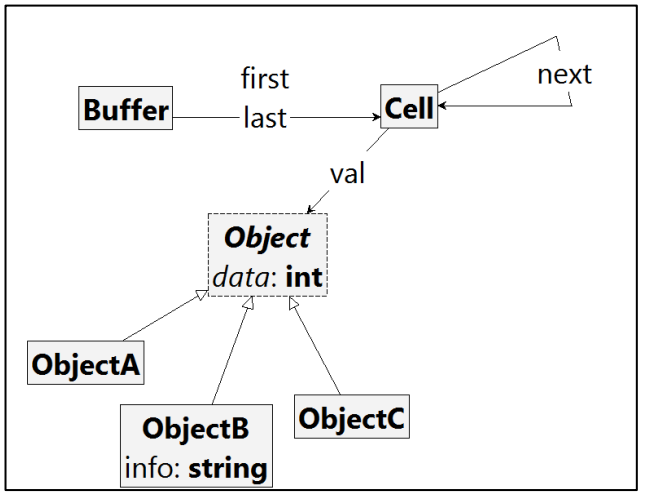
Typgraphen mit Vererbung

- Abstrakte Objekttypen
  - *haben keine Instanzen*
  - *dürfen in Regeln verwendet werden.*
- Vererbungsbeziehung zwischen Objekttypen
  - *Verallgemeinerung von Graphmorphismen: Typen müssen nicht mehr gleich sein, sondern einer Verfeinerungs-beziehung folgen.*

Beispiel: Typgraph mit Vererbung



Getypter Graph



Typgraph mit Vererbung und abstraktem Typ *Object*

Definition: Typgraph mit Vererbung

Ein Typgraph mit Vererbung ist ein Tupel $TGI = (T, I, A)$, wobei:

- $T = (N, E, s, t)$ ist ein Graph (oder ein E-Graph, attributiert über der finalen Σ-Algebra);
- $I \subseteq N \times N$ ist eine partielle Ordnung, die Vererbungshierarchie;
- Der Clan eines Knotens ist die Menge der kleineren Knoten: für $n \in N$ gilt $\mathrm{clan}_I(n) = \{m \in N \mid (m, n) \in I\}$; wir schreiben auch $m \leq n$ für $(m, n) \in I$;
- $A \subseteq N$ ist eine Menge von abstrakten Typen.

Definition: Clan-Morphismus

Gegenben einen Typgraph mit Vererbung TGI, so ist ein Graph (oder E-Graph) G über TGI getypt, falls es ein Paar von Funktionen $\text{type}_{\text{GN}}: N_G \to N_T$ und $\text{type}_{\text{GE}}: E_G \to E_T$ gibt mit:

- $\text{type}_{\text{GN}} \circ s_G(e) \leq s_T \circ \text{type}_{\text{GE}}(e)$ und
- $\text{type}_{\text{GN}} \circ t_G(e) \leq t_T \circ \text{type}_{\text{GE}}(e)$

für alle $e \in E_G$.

Das Paar von Funktionen $\text{type}_{\text{G}} = (\text{type}_{\text{GN}}, \text{type}_{\text{GE}})$ heißt Clan-Morphismus.

Der Graph G ist konkret getypt, falls kein $n \in N_G$ existiert mit $\text{type}_{\text{GN}}(n) \in A$.

Im Falle eines E-Graphen werden formal auch Datenwerte und Attributierungskanten über den Clan-Morphismus abgebildet.

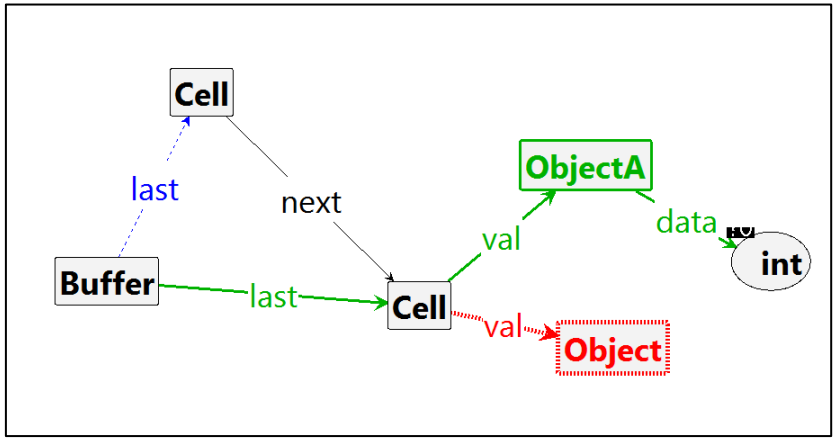
Definition: Getypter Graphmorphismus

Gegeben sei ein Typgraph mit Vererbung $TGI = (T, I, A)$. Ein getypter Graphmorphismus $f: G \to H$ zwischen über TGI getypten Graphen G und H ist definiert durch zwei Funktionen $f_N: N_G \to N_H$ und $f_E: E_G \to E_H$, sodass gilt:

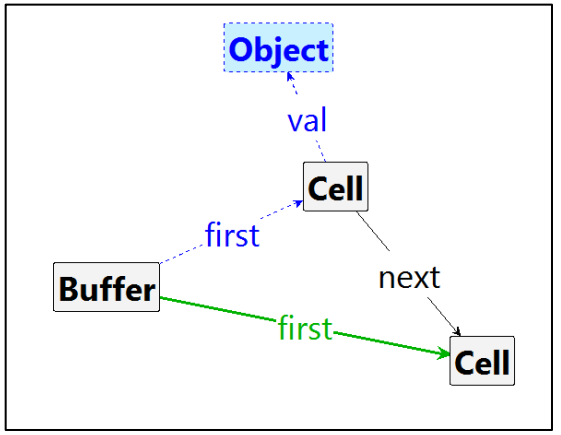
- $f_N \circ s_G = s_H \circ f_E$ (strukturverträglich)
- $f_N \circ t_G = t_H \circ f_E$ (strukturverträglich)
- $\text{type}_{\text{HN}} \circ f_N(n) \leq \text{type}_{\text{GN}}(n)$ für alle $n \in N_G$ (typverträglich)
- $\text{type}_{\text{HE}} \circ f_E(e) = \text{type}_{\text{GE}}(e)$ für alle $e \in E_G$ (typverträglich)

Slogan: Ein Bildknoten ist gleich oder konkreter getypt (bzgl. der Vererbungshierarchie) als sein(e) Urbild(er).

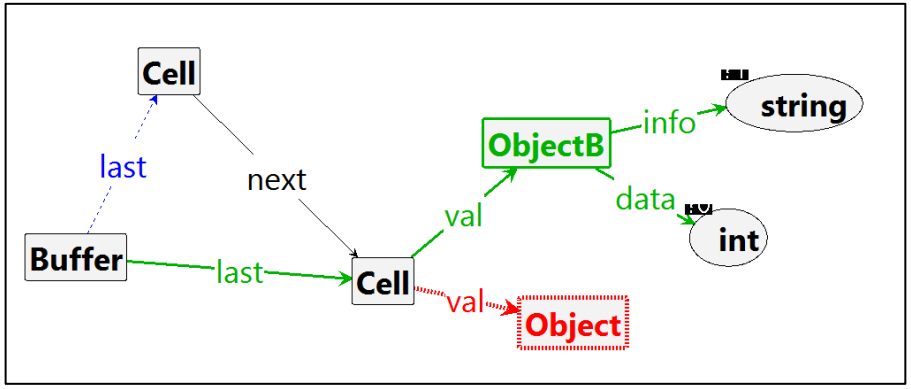
Beispiel: Abstrakte Graphregeln



putObjectA(int)



get



putObjectB(int,string)

Definition: Graphregel mit Vererbung

Gegeben sei ein Typgraph mit Vererbung TGI. Eine über TGI getypte Graphregel ist ein Tupel $r = (L, R, \mathrm{NAC})$, wobei gilt:

- L und R sind über TGI getypte Graphen.
- $K = L \cap R$ ist ein getypter Graph (der zu erhaltende Graphteil).
- Es findet keine Umtypisierung statt: $\mathrm{type_L} = \mathrm{type_K} = \mathrm{type_R}$ (auf K).
- Erzeugte Knoten sind konkret typisiert:
$$\mathrm{type_{RN}}(R \setminus K) \cap A = \emptyset.$$
- NAC ist eine Menge von über TGI getypten Graphen, die jeweils L erweitern, die negativen Anwendungsbedingungen. Eine NAC $N \in \mathrm{NAC}$ darf feiner getypt sein als L: für alle $\mathrm{N} \in \mathrm{NAC}$, für alle $n \in N_L$ und alle $e \in E_L$ gilt: $\mathrm{type_{NN}}(n) \leq \mathrm{type_{LN}}(n)$ und $\mathrm{type_{NE}}(e) = \mathrm{type_{LE}}(e)$.

Definition: Anwendung einer Regel mit Vererbung

Gegeben sei ein Typgraph mit Vererbung TGI, ein konkret getypter Graph G und eine Regel $r = (L, R, \mathrm{NAC})$, die über TGI getypt ist. Die Regel r ist auf G anwendbar (hat einen Ansatz), falls:

- Es gibt einen injektiven getypten Graphmorphismus $m\colon L \to G$ (mit $\mathrm{type}_{\mathrm{GN}} \circ m \leq \mathrm{type}_{\mathrm{LN}}$; d. h., G darf konkreter getypt sein).
- Für alle $N \in \mathrm{NAC}$: Es gibt keinen injektiven getypten Graph-morphismus $q\colon N \to G$ mit $q(L) = m(L)$ (und $\mathrm{type}_{\mathrm{GN}} \circ q \leq \mathrm{type}_{\mathrm{NN}}$).
- Keine hängenden Kanten: $D = G \setminus m(L \setminus (L \cap \mathrm{R}))$ ist ein Graph.

Anwendung von Regel r auf Graph G an Ansatz m (notiert als $G \Rightarrow_{r,m} H$): Analog zum Fall mit Vererbung

Theorem: Äquivalenzresultat

Gegeben seien ein Typgraph mit Vererbung TGI, ein konkret getypter Graph G, eine Regel $r = (L, R, \mathrm{NAC})$ über TGI getypt und ein Ansatz $m: L \to G$. Dann sind die folgenden Aussagen äquivalent:

- **Abstrakte Transformation:**
  Es gibt die Transformation $G \Rightarrow_{r,m} H$.
- **Konkrete Transformation:**
  Es gibt eine konkret getypte Regel $r' = (L', R', \mathrm{NAC}')$ über dem Typgraph TG, dem flachgeklopften Äquivalent von TGI, einen Ansatz $m': L' \to G$ für die Regel r', der dem Ansatz m entspricht und eine Transformation $G \Rightarrow_{r',m'} H$.

Beweis in [LBE+07]

Zusammenfassung

- Graphen und Graphtransformationsregeln können um Attributierung erweitert werden.
  - *Datentypen werden durch Signaturen und Algebren spezifiziert.*
  - *Attributierung erfolgt durch spezielle Kanten.*
  - *Attributänderungen werden durch Löschen und Erzeugen von Kanten umgesetzt.*
  - *Variablen und Parameter werden unterstützt.*
- Vererbung ermöglicht die kompaktere Spezifikation von Graphtransformationssystemen
  - *Abstrakte Typen werden nicht instanziiert.*

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, groove.cs.utwente.nl
- [LBE+07] Juan de Lara, Roswitha Bardohl, Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, Gabriele Taentzer: Attributed graph transformation with node type inheritance. Theor. Comput. Sci. 376(3): 139-163 (2007)