Generierung von Hyperkantenersetzungsgrammatiken zur Heapabstraktion

Christina Jansen

RWTH Aachen

14.10.2009

Wozu Heapabstraktionsgrammatiken?

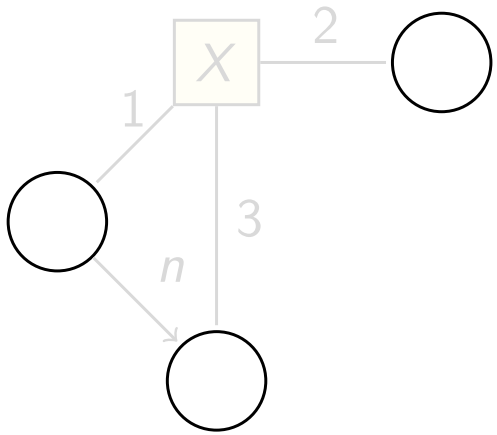
- Verifikation von auf dynamischen, heapbasierten Datenstrukturen basierenden Programmen
- Abstraktion von Heapzuständen durch Hyperkantenersetzungsgrammatiken (HRGs)
- Nutzung von Produktionsregeln der HRG zur Abstraktion & Konkretisierung
- Heapabstraktionsgrammatik = HRG + spez. Eigenschaften
- automatisches Lernen von geeigneten HRGs

Hypergraphen

Definition (Hypergraph H über Σ)

$H = (V, E, lab, att, ext)$ über Alphabet Σ:

- V Knotenmenge, E Kantenmenge
- $lab : E \to \Sigma$ Beschriftungsfunktion
- $att : E \to V^*$ Bindungsfunktion
- $ext \in V^*$ Menge externer Knoten

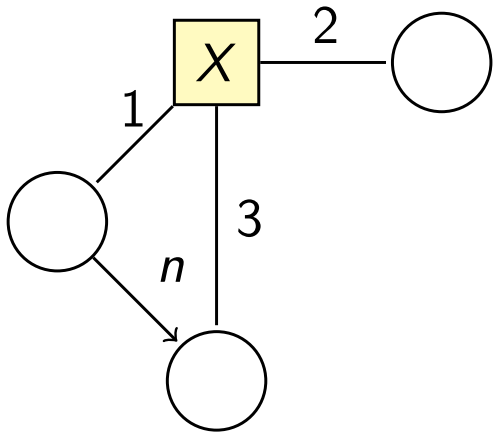


Hypergraphen

Definition (Hypergraph H über Σ)

$H = (V, E, lab, att, ext)$ über Alphabet Σ:

- V Knotenmenge, E Kantenmenge
- $lab : E \to \Sigma$ Beschriftungsfunktion
- $att : E \to V^*$ Bindungsfunktion
- $ext \in V^*$ Menge externer Knoten

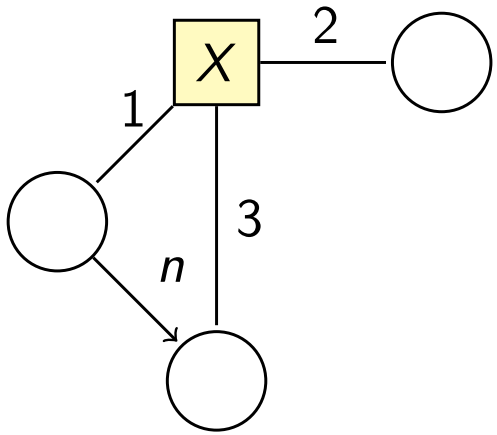


Hypergraphen

Definition (Hypergraph H über Σ)

$H = (V, E, lab, att, ext)$ über Alphabet Σ:

- V Knotenmenge, E Kantenmenge
- $lab : E \to \Sigma$ Beschriftungsfunktion
- $att : E \to V^*$ Bindungsfunktion
- $ext \in V^*$ Menge externer Knoten

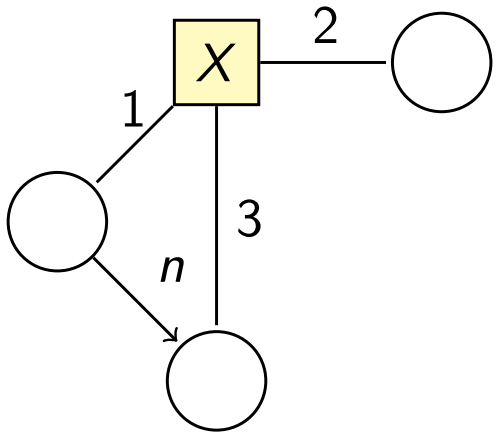


Hypergraphen

Definition (Hypergraph H über Σ)

$H = (V, E, lab, att, ext)$ über Alphabet Σ:

- V Knotenmenge, E Kantenmenge
- $lab : E \to \Sigma$ Beschriftungsfunktion
- $att : E \to V^*$ Bindungsfunktion
- $ext \in V^*$ Menge externer Knoten

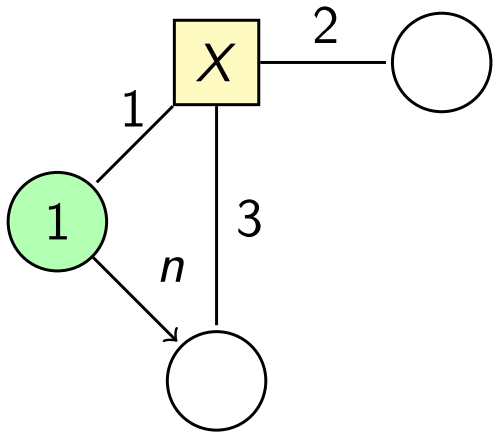


Hypergraphen

Definition (Hypergraph H über Σ)

$H = (V, E, lab, att, ext)$ über Alphabet Σ:

- V Knotenmenge, E Kantenmenge
- $lab : E \to \Sigma$ Beschriftungsfunktion
- $att : E \to V^*$ Bindungsfunktion
- $ext \in V^*$ Menge externer Knoten



Hyperkantenersetzungsgrammatik (HRG)

Definition (HRG G über Σ)

$G = (N, T, P, S)$ mit $\Sigma = N \cup T$, $N \cap T = \emptyset$:

- N Nichtterminalmenge, T Terminalmenge
- P Produktionsregelmenge, $(X, H) \in P$ mit $X \in N$ und H Hypergraph
- S Startsymbol

Beispiel: einfach verkettete Liste

$N = \{L\}$, $T = \{n\}$

Produktionsregeln:

Hyperkantenersetzungsgrammatik (HRG)

Definition (HRG G über Σ)

$G = (N, T, P, S)$ mit $\Sigma = N \cup T$, $N \cap T = \emptyset$:

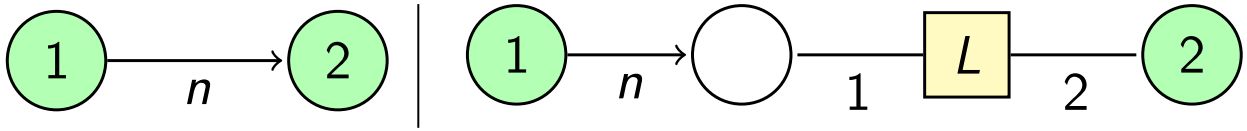
- N Nichtterminalmenge, T Terminalmenge
- P Produktionsregelmenge, $(X, H) \in P$ mit $X \in N$ und H Hypergraph
- S Startsymbol

Beispiel: einfach verkettete Liste

$N = \{L\}$, $T = \{n\}$

Produktionsregeln:

$L \rightarrow$



Hyperkantenersetzungsgrammatik (HRG)

Definition (HRG G über Σ)

$G = (N, T, P, S)$ mit $\Sigma = N \cup T$, $N \cap T = \emptyset$:

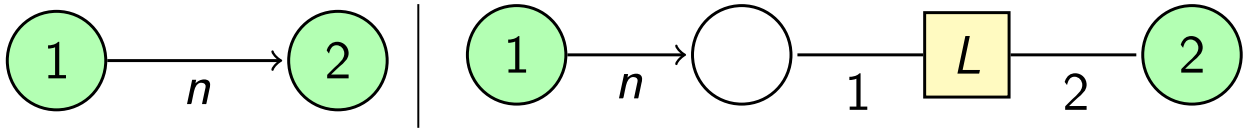
- N Nichtterminalmenge, T Terminalmenge
- P Produktionsregelmenge, $(X, H) \in P$ mit $X \in N$ und H Hypergraph
- S Startsymbol

Beispiel: einfach verkettete Liste

$N = \{L\}$, $T = \{n\}$

Produktionsregeln:

$L \rightarrow$



Übersicht: Heapabstraktion

Problem:

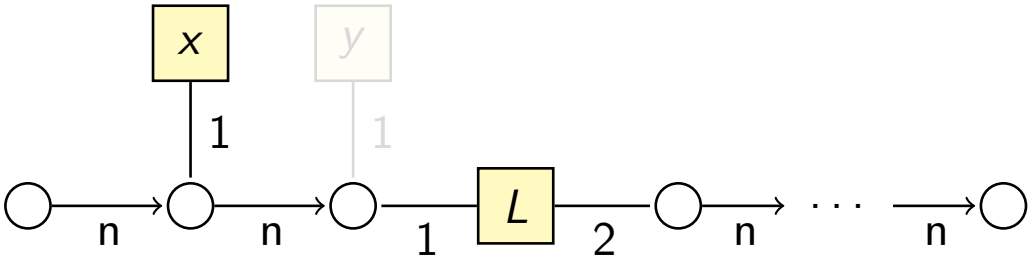
- heapbasierte, dynamische Datenstrukturen
- Konstruktion und Destruktion von Objekten zur Laufzeit
- pot. unendliche Menge an Heapzuständen

Ansatz:

- Heaprepräsentation durch Hypergraphen
- Abstraktion von Hypergraphen durch rückwärtige Produktionsregelanwendung
- Partielle Konkretisierung

Heaprepräsentation

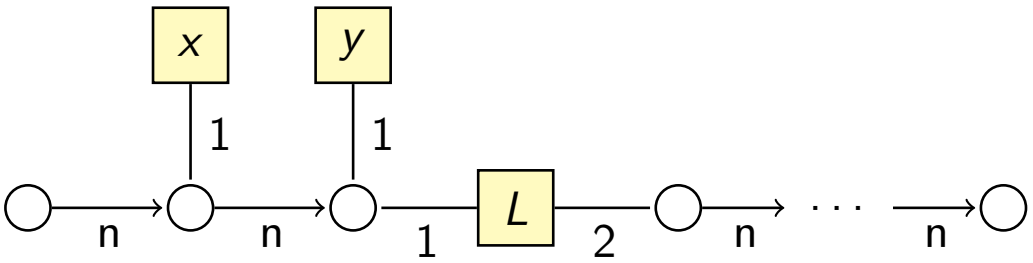
- Variablen: Terminalkanten vom Rang 1
- Zeiger: Terminalkanten vom Rang 2
- Hypergraphrepräsentation eines Heapzustandes: **Heapkonfiguration**




```
[...]
y := x.n;
x := nil;
[...]
```

Heaprepräsentation

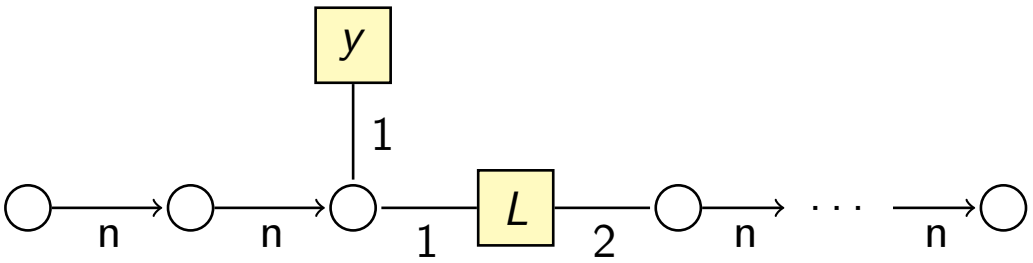
- Variablen: Terminalkanten vom Rang 1
- Zeiger: Terminalkanten vom Rang 2
- Hypergraphrepräsentation eines Heapzustandes: **Heapkonfiguration**




```
[...]
y := x.n;
x := nil;
[...]
```

Heaprepräsentation

- Variablen: Terminalkanten vom Rang 1
- Zeiger: Terminalkanten vom Rang 2
- Hypergraphrepräsentation eines Heapzustandes: **Heapkonfiguration**




```
[...]
y := x.n;
x := nil;
[...]
```

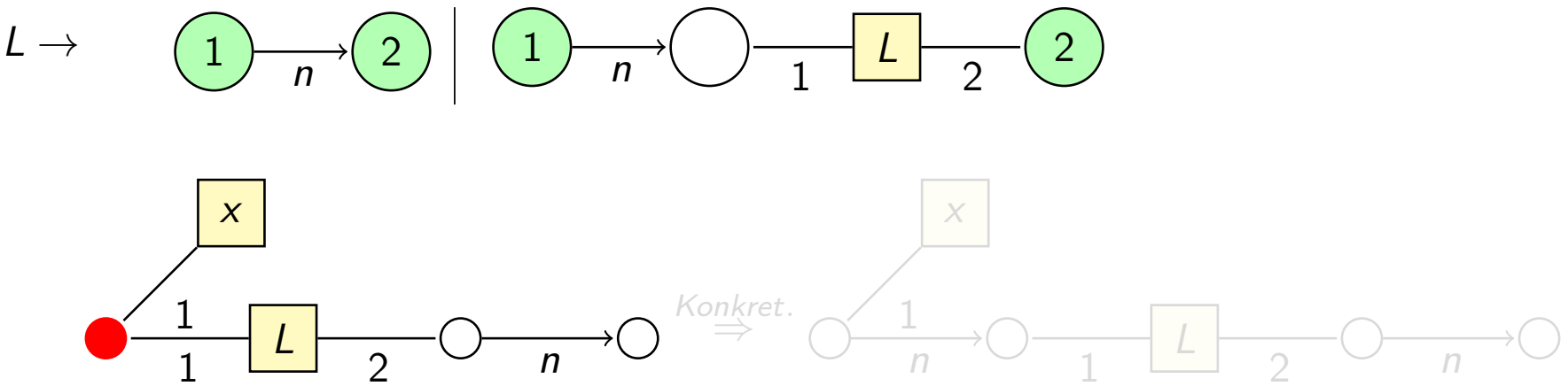
Abstraktion & Konkretisierung

- Abstraktion: soweit wie möglich
- Konkretisierung: wenn nötig (unzulässige Konfiguration)

⇒ Operationen nur auf konkreten Teilgraphen nötig

Abstraktion & Konkretisierung

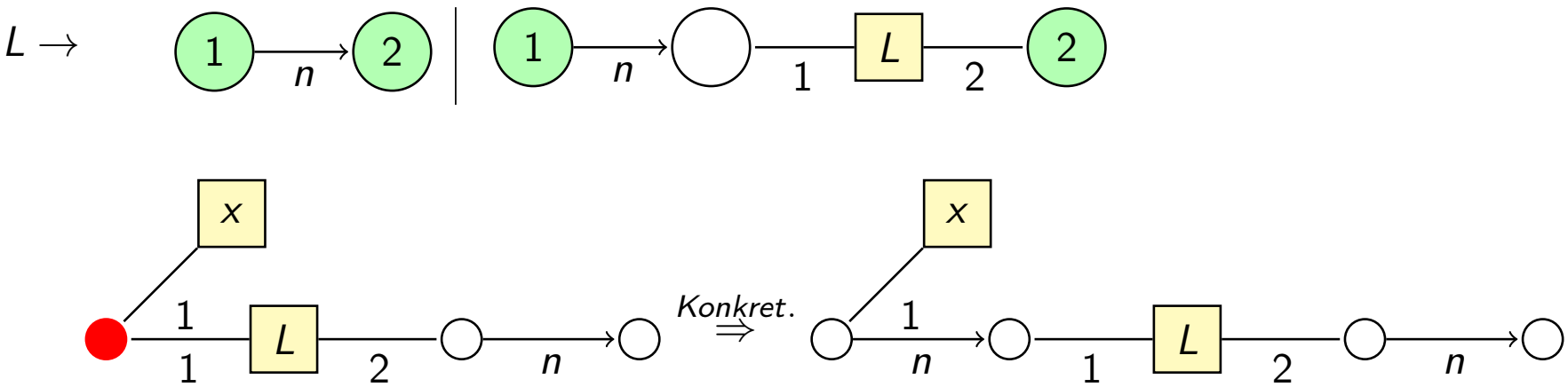
- Abstraktion: soweit wie möglich
- Konkretisierung: wenn nötig (unzulässige Konfiguration)



⇒ Operationen nur auf konkreten Teilgraphen nötig

Abstraktion & Konkretisierung

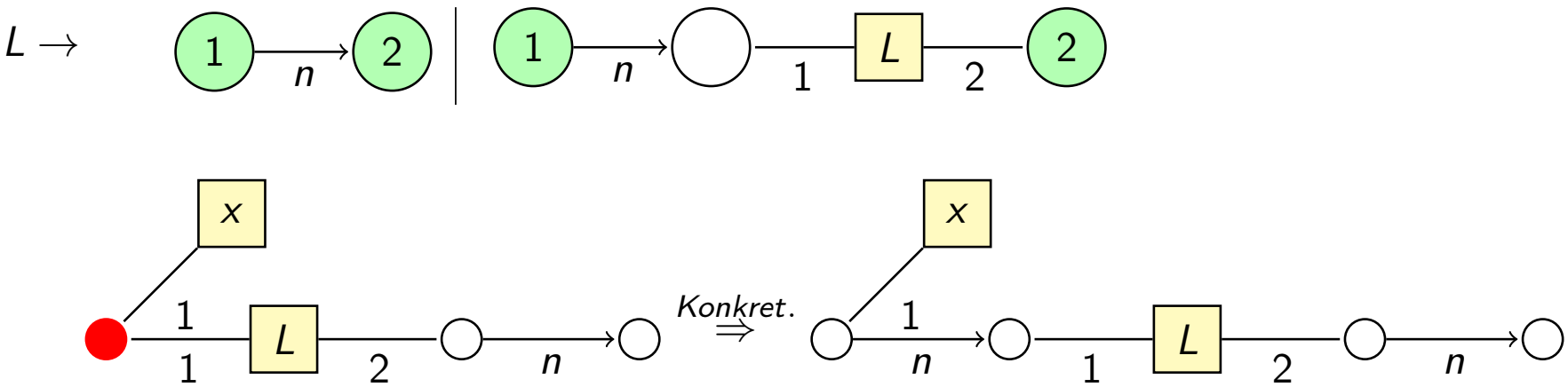
- Abstraktion: soweit wie möglich
- Konkretisierung: wenn nötig (unzulässige Konfiguration)



⇒ Operationen nur auf konkreten Teilgraphen nötig

Abstraktion & Konkretisierung

- Abstraktion: soweit wie möglich
- Konkretisierung: wenn nötig (unzulässige Konfiguration)



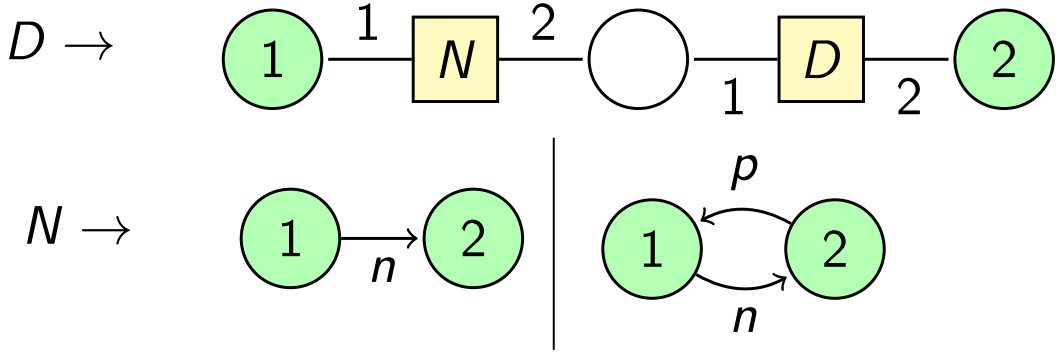
⇒ Operationen nur auf konkreten Teilgraphen nötig

HRG = Heapabstraktionsgrammatik?

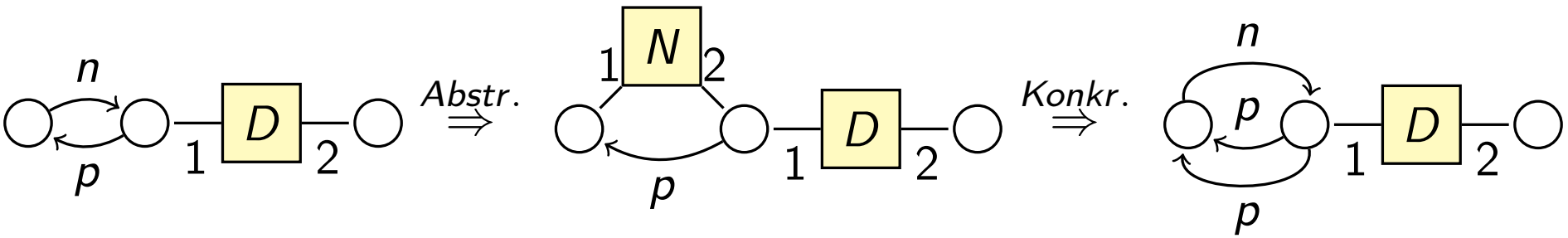
- Anpassungen für Ausführen von Produktionsregeln "rückwärts"
- Verlust von Eigenschaften wie Konfluenz, Terminierung etc.
- Heapabstraktionsgrammatik = HRG + spez. Anforderungen
  - Produktivität
  - Wachstum
  - **Typisierung**
  - Variablenfreiheit
  - **Lokale Apex-Eigenschaft**

Typisierung

Problemfall: verkettete Liste mit pot. Rückwärtszeigern

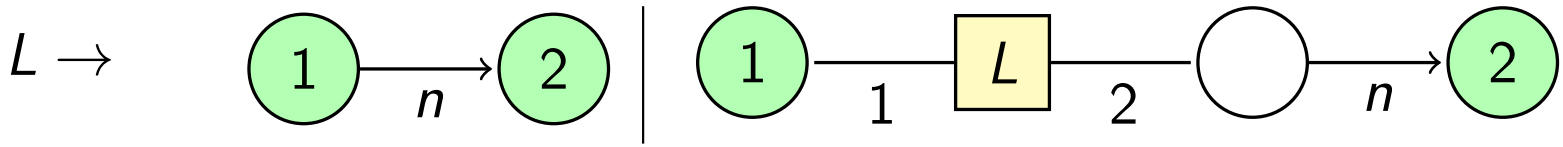


Denn z.B.:

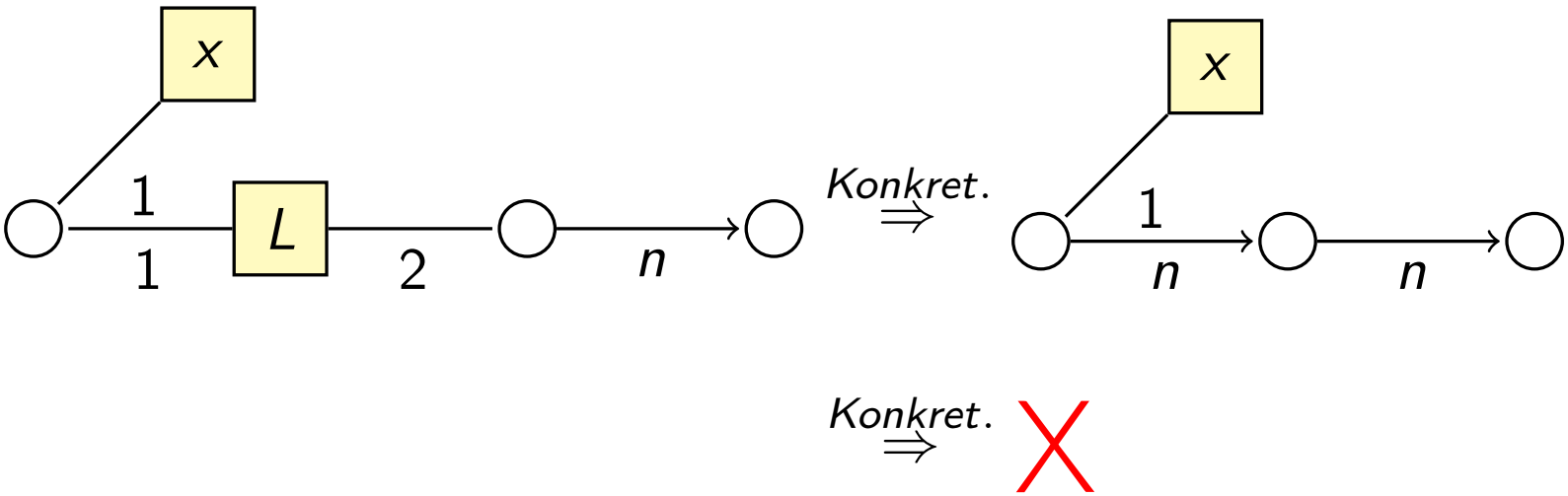


Lokale Apex-Eigenschaft

Problemfall: einfach verkettete Liste

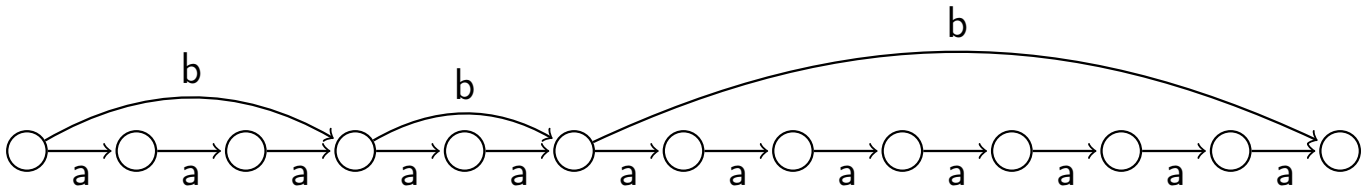


Variablenzuweisung:



Idee: Konstruktion der lokalen Apex-Eigenschaft

Datenstruktur: Grashüpfer

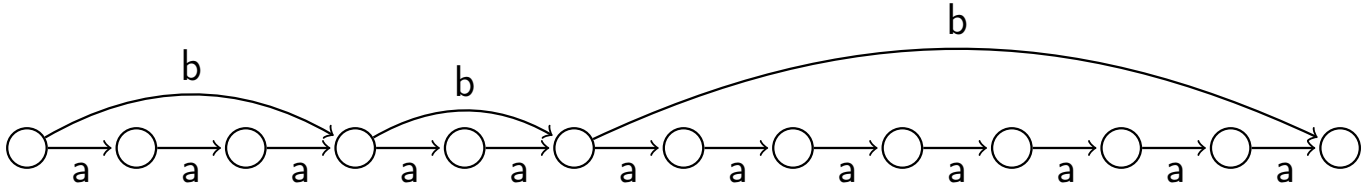


HRG (Auszug)

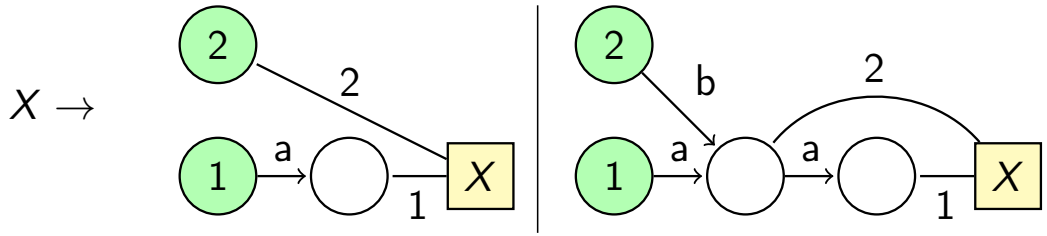
Idee: "Falten" von Teilgraphen

Idee: Konstruktion der lokalen Apex-Eigenschaft

Datenstruktur: Grashüpfer



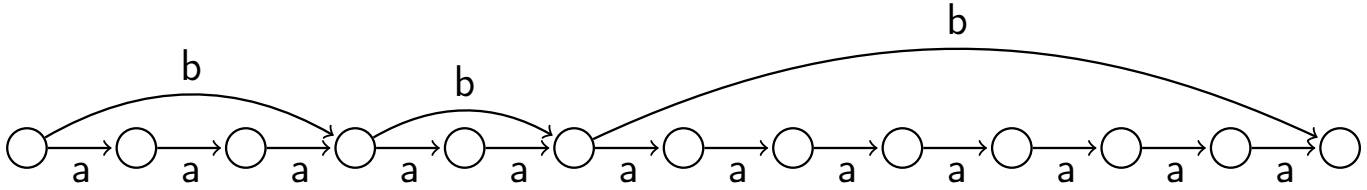
HRG (Auszug)



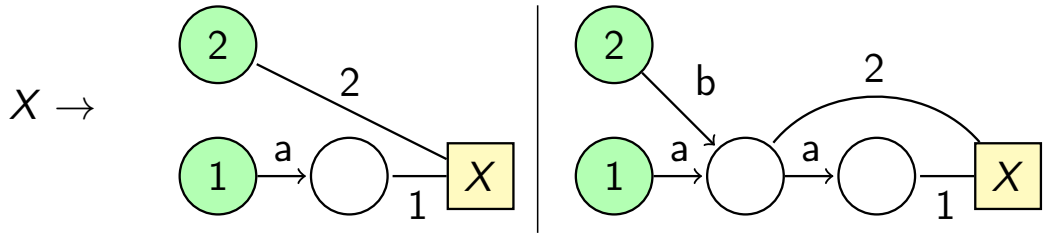
Idee: "Falten" von Teilgraphen

Idee: Konstruktion der lokalen Apex-Eigenschaft

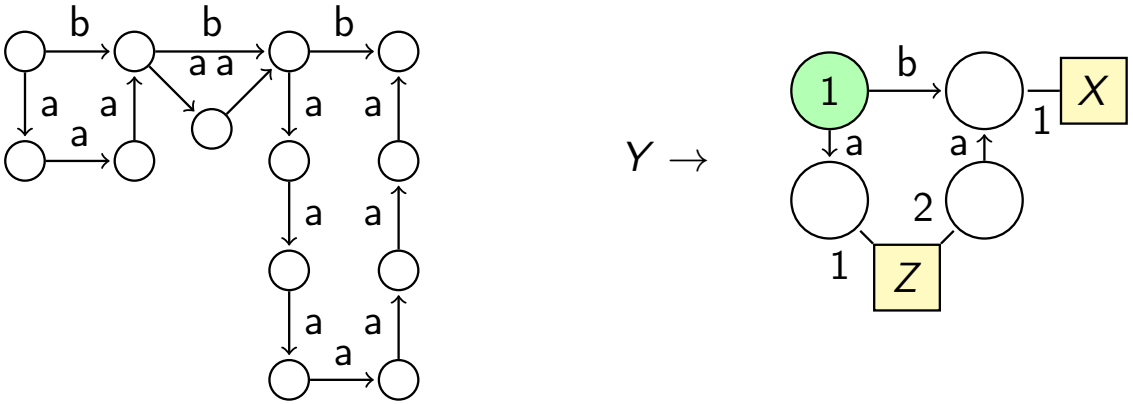
Datenstruktur: Grashüpfer



HRG (Auszug)



Idee: "Falten" von Teilgraphen



Überblick: Lernen von HRGs

- Problem: geeigente Heapabstraktionsgrammatik von Datenstruktur & Programmanweisungen abhängig
- Ziel: Automatisierte Konstruktion der Grammatik

- Vorgehensweise: Lernen von Nichtterminalen und Produktionsregeln während der Programmausführung
- Idee: Inferenz von HRGs durch Hypergraph-Beispielmengen

Überblick: Lernen von HRGs

- Problem: geeigente Heapabstraktionsgrammatik von Datenstruktur & Programmanweisungen abhängig
- Ziel: Automatisierte Konstruktion der Grammatik

- Vorgehensweise: Lernen von Nichtterminalen und Produktionsregeln während der Programmausführung
- Idee: Inferenz von HRGs durch Hypergraph-Beispielmengen

Überblick: Lernen von HRGs

- Problem: geeigente Heapabstraktionsgrammatik von Datenstruktur & Programmanweisungen abhängig
- Ziel: Automatisierte Konstruktion der Grammatik

- Vorgehensweise: Lernen von Nichtterminalen und Produktionsregeln während der Programmausführung
- Idee: Inferenz von HRGs durch Hypergraph-Beispielmengen

Grundlage: Inferenzalgorithmus

- [Jeltsch, 1990]: Inferenz von HRGs
- Grundlegende Idee: Dekomposition von Produktionsregeln
- Problem: hochgradig nichtdeterministisch
- Heuristiken zur Determinisierung:
  - Schrittweise Dekomposition
  - Zerlegung in disjunkte Teilgraphen
  - falls nötig: minimaler Schnitt
  - Zusammenfassen von Nichtterminalen mit gleichem Rang und gleicher Terminalfolge

Grundlage: Inferenzalgorithmus

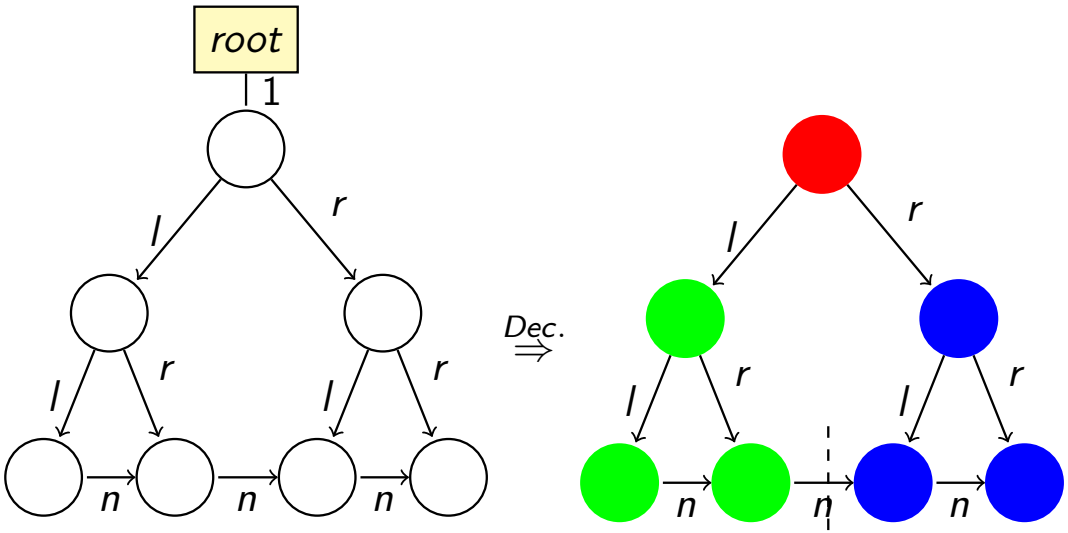
- [Jeltsch, 1990]: Inferenz von HRGs
- Grundlegende Idee: Dekomposition von Produktionsregeln
- Problem: hochgradig nichtdeterministisch
- Heuristiken zur Determinisierung:
  - Schrittweise Dekomposition
  - Zerlegung in disjunkte Teilgraphen
  - falls nötig: minimaler Schnitt
  - Zusammenfassen von Nichtterminalen mit gleichem Rang und gleicher Terminalfolge

Grundlage: Inferenzalgorithmus

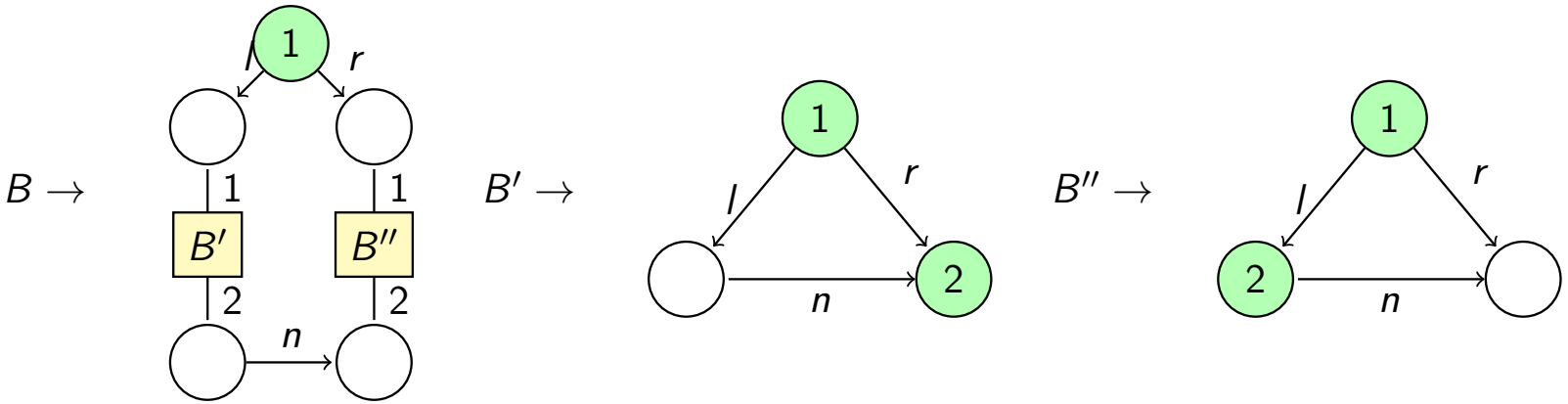
- [Jeltsch, 1990]: Inferenz von HRGs
- Grundlegende Idee: Dekomposition von Produktionsregeln
- Problem: hochgradig nichtdeterministisch
- Heuristiken zur Determinisierung:
  - Schrittweise Dekomposition
  - Zerlegung in disjunkte Teilgraphen
  - falls nötig: minimaler Schnitt
  - Zusammenfassen von Nichtterminalen mit gleichem Rang und gleicher Terminalfolge

Beispiel: Dekomposition

Hypergraph: binärer Baum mit verketteter Blattfront



Dekomposition: Produktionsregeln



Zusammenfassung & Ausblick

- Heapabstraktionsgrammatik aus gegebener HRG

- Herstellung der lokalen Apex-Eigenschaft komplex
- Grundlage: Lokale Greibach NF
- Optimierung: lokale Greibach Normalform Konstruktion

- automatische Generierung einer geeigneten HRG
- Hauptidee: Dekomposition von Beispielhypergraphen

Danke für Ihre Aufmerksamkeit!

Literatur


Stefan Rieger und Thomas Noll.
Abstracting Complex Data Structures by Hyperedge Replacement
Springer-Verlag, 2008.


Joost Engelfriet, Linda Heyker und George Leih.
Context-Free Graph Languages of Bounded Degree are Generated by Apex Graph Grammars.
Acta Inf., Vol. 31, 1994.


Eric Jeltsch and Hans-Jörg Kreowski.
Grammatical Inference Based on Hyperedge Replacement
Graph-Grammars and Their Application to Computer Science, 461-474, 1990.