An Introduction to Graph Transformation

Dagstuhl Workshop *Formal Models of Graph Transformation in Natural Language Processing*

F. Drewes

16 March 2015

Introduction

Graph transformation...

- started around 1970 in the form of graph grammars,
- studies rewrite systems that act on graphs,
- ranges from Turing complete models of computation to context-free graph grammars,
- does not provide very successful automata models for graphs (in the sense of FSA) though there do exist some attempts,
- has established strong connections between context-free graph languages and monadic second-order logic.

Guiding idea behind most of it

Use rules that replace local substructures.
Apply them iteratively.

About this Introduction

Here, I attempt to given an overview of some of the most important concepts and facts.

- Certainly heavily biased
- Trys to focus on what I expect to be potentially interesting for CL/NLP
- Subjective choice, will certainly include & omit the wrong things

Since we are here to learn from each other, please interrupt, ask, comment, correct, add, jump in, etc.

Structure of the Presentation

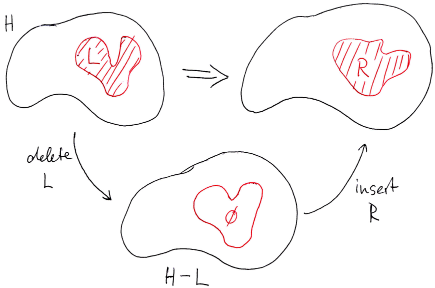
1. Introduction
2. General Graph Transformation Systems
3. Context-Free Graph Grammars
4. Parsing HR Languages
5. Monadic Second-Order Logic
6. Term Graphs
7. Concluding Remarks

General Graph Transformation Systems

General idea of rule application

Applying a rule $L \Rightarrow R$ to a host graph H

1. locates (a copy of) the left-hand side (lhs) L in H,
2. deletes L from H, and
3. inserts the right-hand side (rhs) R.



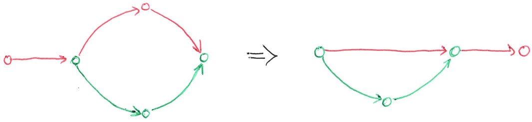
Obvious question: What does locate/delete/insert mean?

In the Zoo

The locate/delete/insert question can be answered in several ways
$\Rightarrow$ many possible approaches to graph transformation

- Basically all of them are Turing complete.
- Some add control structures (like programmed graph grammars).
- Here: focus on the "algebraic approach".
- Comes in two flavors: double-pushout and single-pushout approach.
- There is also a pullback approach, but I won't talk about that one.

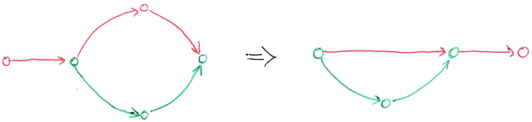
Note: Pushouts and pullbacks are notions from category theory that we do not need to care about.



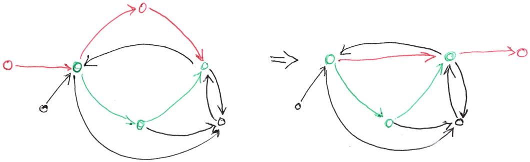
- Left-hand side (lhs) and right-hand side (rhs) intersect in the green items that form the gluing graph.
- The red part of the lhs (rhs) is to be deleted (inserted, resp.)
- The purpose of the gluing graph is to establish the connection between old and new parts.

Note: In general, nodes and edges can be labeled.

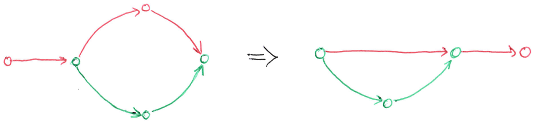
Getting More Concrete: Applying the Rule (1)



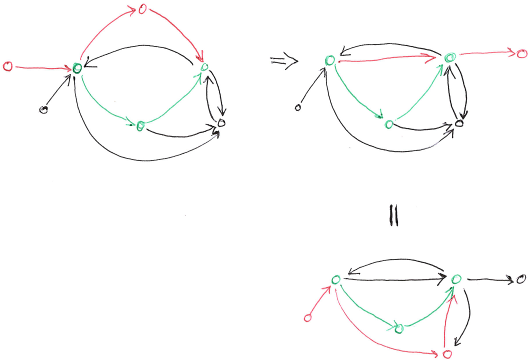
yields



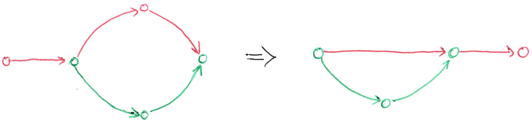
Getting More Concrete: Applying the Rule (1)



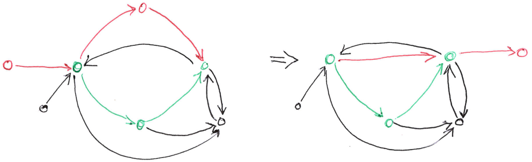
yields

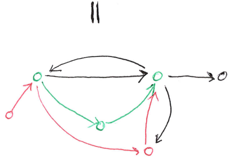


Getting More Concrete: Applying the Rule (1)



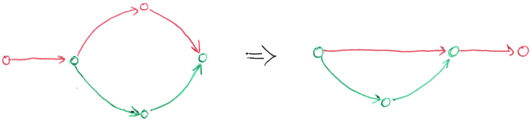
yields



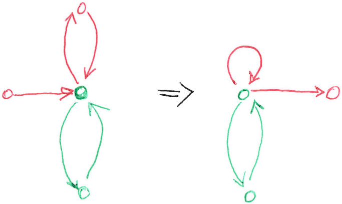


Dangling condition: All edges that are incident with deleted nodes must be deleted. (Deletion of the red part creates no dangling edges.)

Getting More Concrete: Applying the Rule (2)



yields



Identification condition: Identify no deleted (red) items with other items.

Alternative: Generally require injective occurrences (forbid identification).

Remark: Formalization as Double vs. Single Pushout

Application of $L \Rightarrow R$ to obtain a derivation step $G \Rightarrow H$:

$$\begin{array}{lccccc} \text{Rule:} & L & \supseteq & K & \subseteq & R \\ & \downarrow & & \downarrow & & \downarrow \\ \text{Step:} & G & \supseteq & D & \subseteq & H \end{array}$$

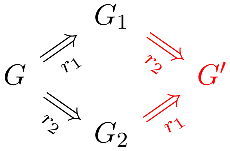
This diagram exists and is unique if the dangling and identification conditions are satisfied. (Formally, the squares are pushouts.)

In the single pushout approach a rule is a partial mapping $L \to R$ and only one square is constructed:

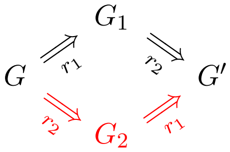
$$\begin{array}{lccc} \text{Rule:} & L & \longrightarrow & R \\ & \downarrow & & \downarrow \\ \text{Step:} & G & \longrightarrow & H \end{array}$$

This imposes no dangling or identification condition. Instead, deletion gets priority over preservation.

Rule applications can be made in parallel if they are independent.
Formulated for the double pushout case:



Parallel independence: The red part exists iff the two applications overlap in gluing items only.



Sequential independence: The red part exists iff all items that are both in the rhs of r_1 and the lhs of r_2 are gluing items of both.

- These two are equivalent.
- The parallel rule $r = r_1 \uplus r_2$ is applicable iff the two individual applications are parallel independent, and then $G \underset{r}{\Rightarrow} G'$.

Context-Free Graph Grammars

Idea: A rule should replace an atomic item with a nonterminal label.

Derivation: Start from an axiom (e.g., a single nonterminal item). Apply rules until no nonterminal is left.

Have your choice

An atomic item can be a node or an edge.

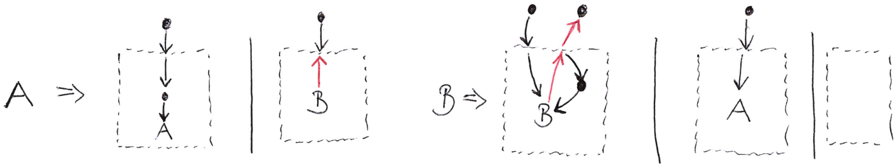
⇒ two different types of grammars based on

- node replacement and
- edge replacement, resp.

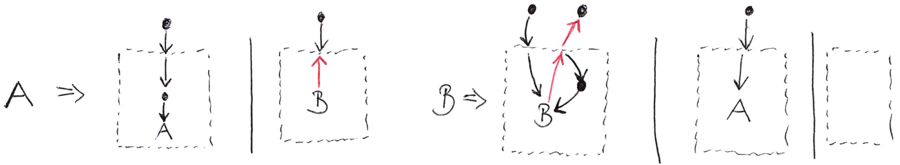
Node/Vertex Replacement (VR)

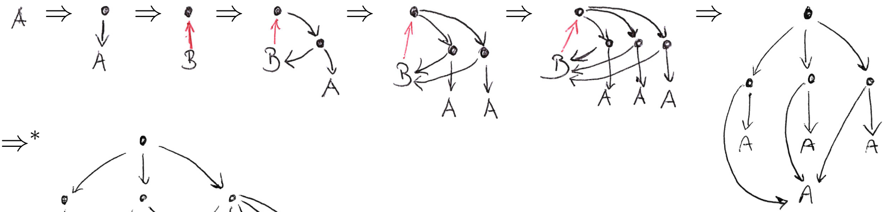
- Rules replace nodes labeled with nonterminals
  $\Rightarrow$ the left-hand side of a rule is a nonterminal label.
- Problem: we need to specify how the right-hand side shall be connected to the host graph.
- Replacement steps:
  1. remove the lhs node with its incident edges,
  2. add the rhs disjointly, and
  3. use the connection instructions to connect it to former neighbors of the replaced node.

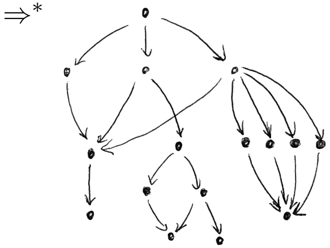
VR by Example



VR by Example





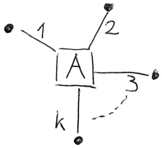


Remarks

- Context-freeness requires confluence.
- Confluence may be violated if there are adjacent nonterminals.
- The boundary condition guarantees confluence but is stronger.
- Non-confluence gives PSPACE-complete languages.
- Confluence ensures containment of the languages in NP.

Hyperedge Replacement (HR)

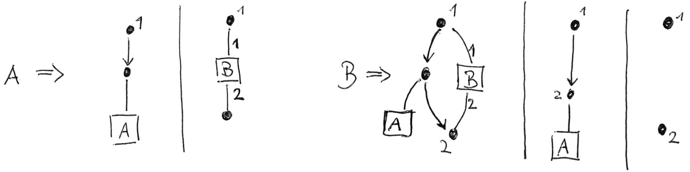
- Rules replace directed hyperedges labeled with nonterminals $\Rightarrow$ the left-hand side of a rule is a nonterminal label.
- A hyperedge of rank k connects a sequence of k nodes:



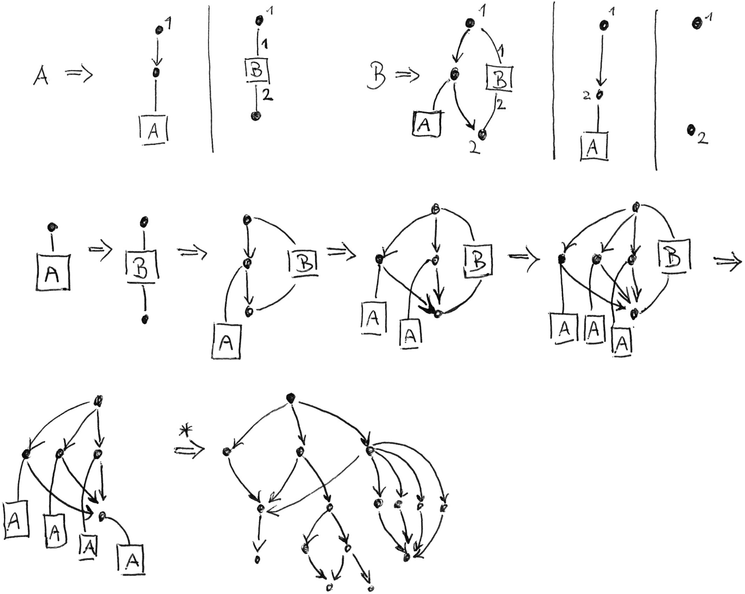
- In the right-hand side a sequence of k nodes called sources is distinguished.
- Replacement steps:
  1. remove a hyperedge e whose label is that of the lhs,
  2. add the rhs disjointly, and
  3. fuse the ith incident node of e with the ith source.

Note: We use hyperedges instead of edges in order to be able to "control" more than two nodes.

HR by Example

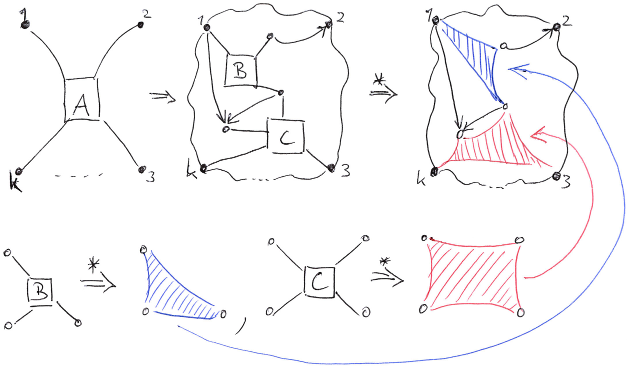


HR by Example



UMEÅ UNIVERSITET

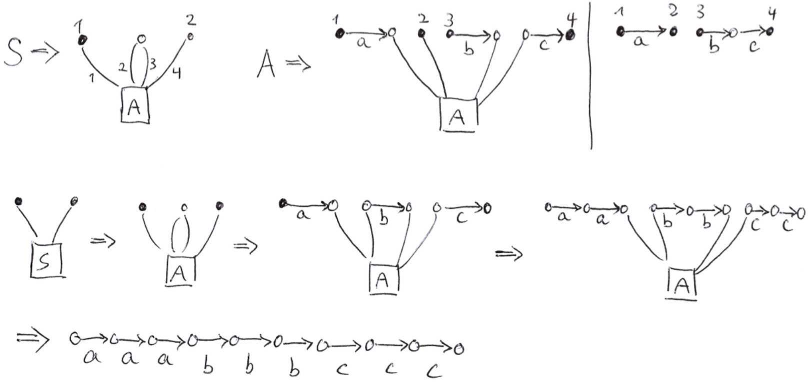
Context-Freeness of HR



$(G \Rightarrow^n H$ iff $H = G[G_1/e_1, \ldots, G_n/e_l]$ with $G(e_i) \Rightarrow^{n_i} G_i$, $n = \sum_i n_i)$

- Paths connecting “inside” and “outside” must pass attached nodes.
- Treewidth of language is bounded.
- Special case of node replacement (more or less).
- Chomsky normal-form puts languages into NP.

Look at this:



- Same power as DTWT, MCFG, etc.
- Work on Early-style parsing algorithms for string-generating HR grammars was done by Fischer et al.

HR and VR Graph Operations

Old idea by Mezei, Wright (1967)

Context-free generation = regular tree grammar + evaluation of trees in some algebra (i.e., view a symbol of rank k as a k-ary operation).

HR	VR
Objects: graphs with partial injective source label mapping $src\colon V \to LAB$	Objects: graphs with partial port label mapping $port\colon V \to LAB$
Operations (many variants possible)	
binary composition $/\!/$ (take disjoint union & fuse sources with same label)	binary disjoint union $\oplus$ (put two graphs next to each other)
	edge creation $add_{a\to b}$ (add all edges from a-ports to b-ports)
unary relabeling rel_ρ (relabel all sources according to partial injective $\rho\colon LAB \to LAB$)	unary relabeling rel_ρ (relabel all ports according to $\rho\colon LAB \to LAB$)

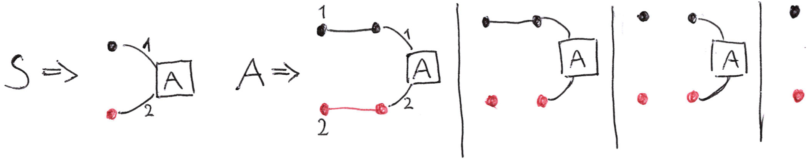
Parsing HR Languages

NP-Completeness of linear VR and HR Languages

Two similar independent "historical" proofs. Flexible because of simplicity.

Language 1 (Aalbersberg, Ehrenfeucht, Rozenberg 1986):

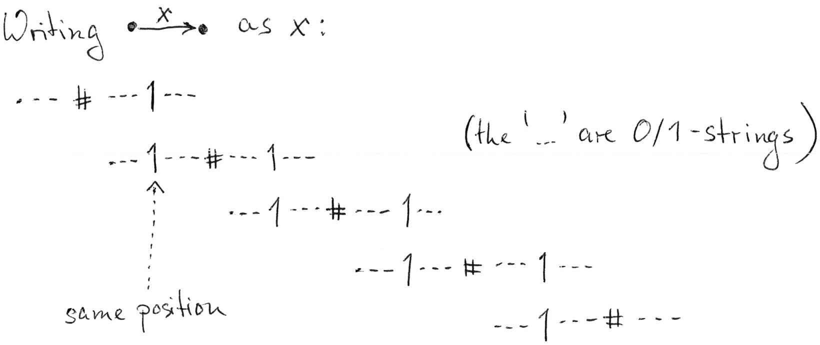
Grammar:



NP-complete by reduction of 3-PARTITION

NP-Completeness of VR and HR Languages

Language 2 (Lange, Welzl 1987):

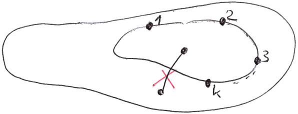


NP-complete by reduction of HAMILTONIAN PATH.

Component $u\#u$ is interpreted as a (mirrored) incidence list of a node v. ('1' in position i means ith edge is incident with v.)

Polynomial Parsing of HR Languages

- CFG-parsing by dynamic programming (CKY) is polynomial because a string has $O(n^2)$ substrings. But a graph has $O(2^n)$ subgraphs.
- Sometimes it helps that we only need to consider



  $\Rightarrow$ Connected + bounded degree yields P (Rozenberg, Welzl 1986).
- More general: Lautemann (1990) requires logarithmic k-separability.
- Chiang et al. (2013) show the bound $O((3^d n)^{w+1})$, where w is the treewidth of the rules (requires connectedness!)

Note 1: Chiang et al. (2013): replace d by k-separability?

Note 2: Forgotten concept by Lautemann: componentwise derivations.

Note 3: Polynomial algorithms are non-uniform (fixed grammar).

Monadic Second-Order Logic

MSO on Graphs

Viewing a graph G as a logical structure:

- Nodes (and edges?) are elements of the domain $dom(G)$.
- If only nodes are in the universe, graphs are simple.
- Predicates for source labels etc ($src_a(x) = true$ if x is a-source)
- Predicates for incidence or adjacency ($edg_f(x, y) = true$ if (x, y) is an edge with label f, or $s(e, x) = true$ if x is the source of e and $t(e, y) = true$ if y is the target of e).

Formulas are built as usual, including quantification $\forall x$, $\exists x$, $\forall X$, $\exists X$ over singletons $x \in dom(G)$ and sets $X \subseteq dom(G)$.

Note: "monadic" means that there is no quantification over relations.

Counting MSO is a useful generalization containing cardinality predicates $card_p^q(X) \equiv |X| = p \pmod q$.

Connections between MSO and Context-Freeness

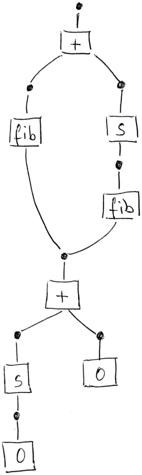
- We have to use the "right" relational structures (e.g., HR needs quantification over edges whereas VR uses simple graphs).
- A (counting) MSO sentence ϕ defines the graph language $\{G \mid G \models \phi\}$.
- Context-freeness is not equivalent to definability (counterexample: $a^n b^n$ viewed as string graphs). However, the following hold:
  - $\{G \in L(\mathcal{G}) \mid G \models \phi\}$ is effectively context-free.
  - Consequently, it is decidable whether all/infinitely many/finitely many/no graphs of a context-free graph language satisfy ϕ.
- Generalization: The image of a context-free graph language under a CMSO transduction is effectively context-free.

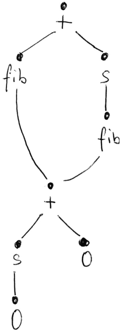
Most MSO-based constructions/algorithms are inefficient, but they provide a good starting point.

Term Graphs

Graphs Representing Trees

(Hyper)graphs can represent trees with shared subtrees
$\Rightarrow$ we can implement term rewriting by graph transformation.

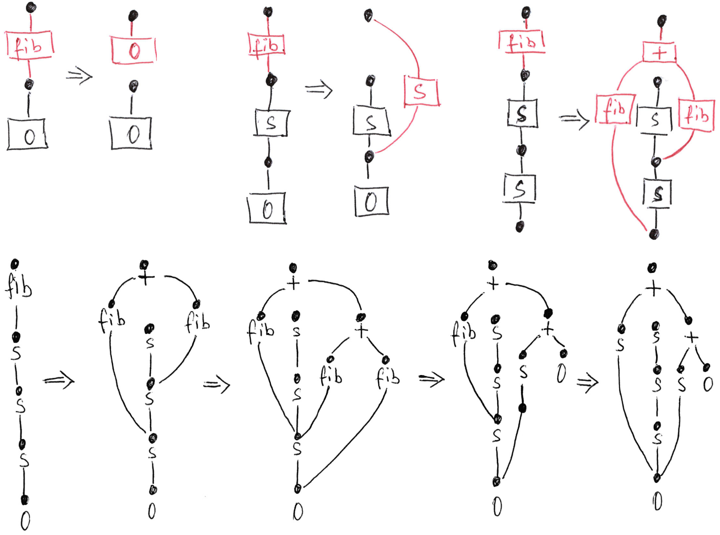




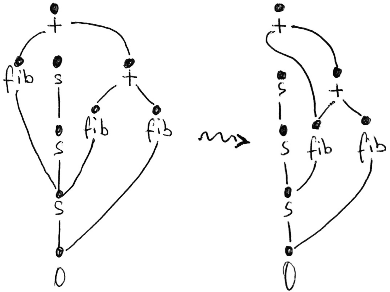
- Example:
  symbols $+,\ \mathit{fib},\ s,\ 0$ (arities 2, 1, 1, 0)
  term $\mathit{fib}(s(0)+0) + s(\mathit{fib}(s(0),0))$
- Unfolding removes sharing by copying shared subtrees.
- Conversely, collapsing equal subtrees creates a compact representation.

Example: Term Rewriting by Graph Transformation

$$fib(0) \to 0, \quad fib(s(0)) \to s(0), \quad fib(s(s(x))) \to fib(x) + fib(s(x))$$



Collapsing nodes that represent identical subtrees increases efficiency (but removes degrees of freedom):



Another phenomenon observed in this term graph is garbage.

Concluding Remarks

Many Things have been Left Out

Among the many things left out are:

- rules with application conditions
- structuring principles such as transformation units
- graph programs
- graphs with attributes
- generalizations of graph transformation
- ...

Systems Implementing Graph Transformation

- AGG: transform graphs with attributes, based on single-pushout approach, TU Berlin (G. Taentzer). Development stopped?
- GrGen.NET: fast implementation using single-pushout approach, Univ. Karlsruhe (R. Geiß et al.). Latest release from 2014.
- GP: implements graph programs, University of York (D. Plump et al.). Ongoing development (I think).
- GMTE: implements several approaches to graph transformation, LAAS-CNRS (Houda Khlif et al.). Ongoing development.
- GROOVE: system for model transformation based on gra tra, intended for verification, University of Twente (A. Rensink et al.).
- Bolinas: graph processing package implementing (synchronous) HR grammars, USC/ISI (D. Bauer et al.). Ongoing.