ESSLLI 2016 course

Foundations of Graph Transformation and Graph Grammars

Lecture 5: *Further Topics, Implementations, and Literature*

F. Drewes

19 August 2016

UMEÅ UNIVERSITET

Introduction

Yesterday & Today

Yesterday:

- We had a look at (counting) monadic second-order logic on graphs.
- Two variants, depending on whether edges can be quantified over.
- These are closely connected to context-free graph languages.
- MSO properties can be checked in fixed-parameter linear time.

Further topics touched upon today:

- Term graph rewriting
- DAG automata
- Implementations
- Books and Surveys

Term Graph Rewriting

Term Rewriting and Sharing

Term rewriting is closely related to functional programming languages.

Example: the Fibonacci function

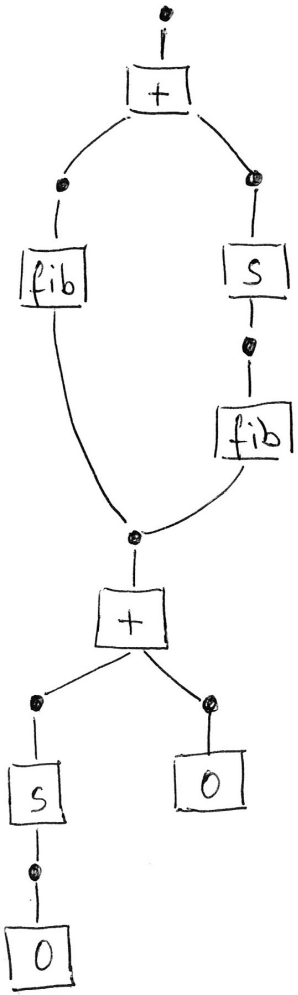
Haskell notation		Term rewrite rule	
$fib\ 0$	$= 0$	$fib(0)$	$\rightarrow 0$
$fib\ (s\ 0)$	$= s\ 0$	$fib(s(0))$	$\rightarrow s(0)$
$fib\ (s\ (s\ x))$	$= (fib\ (s\ x)) + (fib\ x)$	$fib(s(s(x)))$	$\rightarrow fib(x) + fib(s(x))$
$x + 0$	$= x$	$x + 0$	$\rightarrow x$
$x + (s\ y)$	$= s\ (x + y)$	$x + s(y)$	$\rightarrow s(x + y)$

Evaluation is straightforward but inefficient:

- Large terms with many identical subterms may occur.
- This is a waste of time and memory space.
- So, why not use sharing?

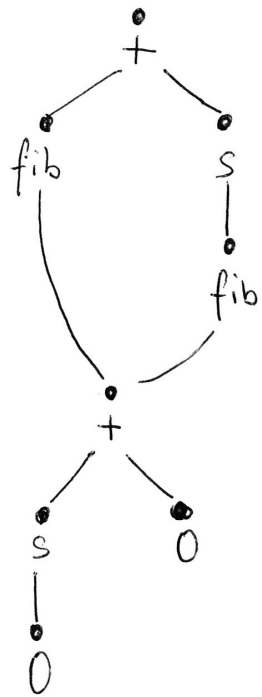
Graphs Representing Trees

To make use of sharing, we can represent terms as (hyper)graphs and implement term rewriting by graph transformation.



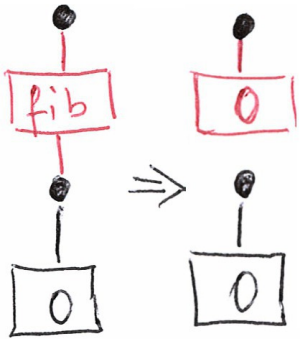
Example: $\mathit{fib}(s(0)+0)+s(\mathit{fib}(s(0),0))$

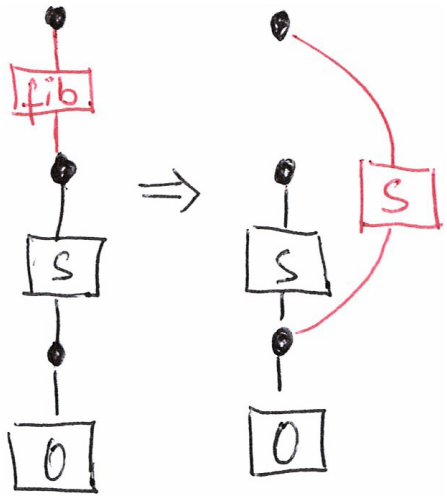
Simplified drawing:

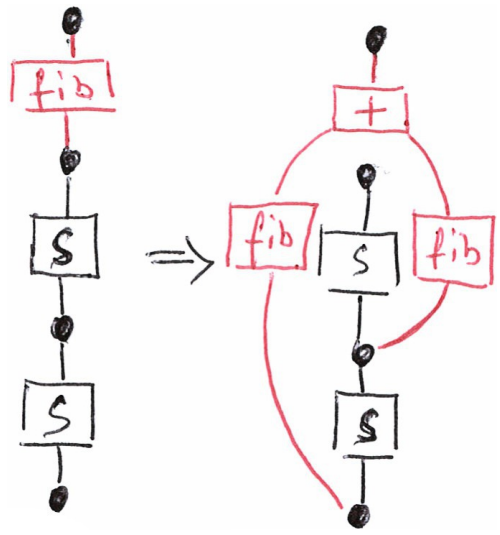


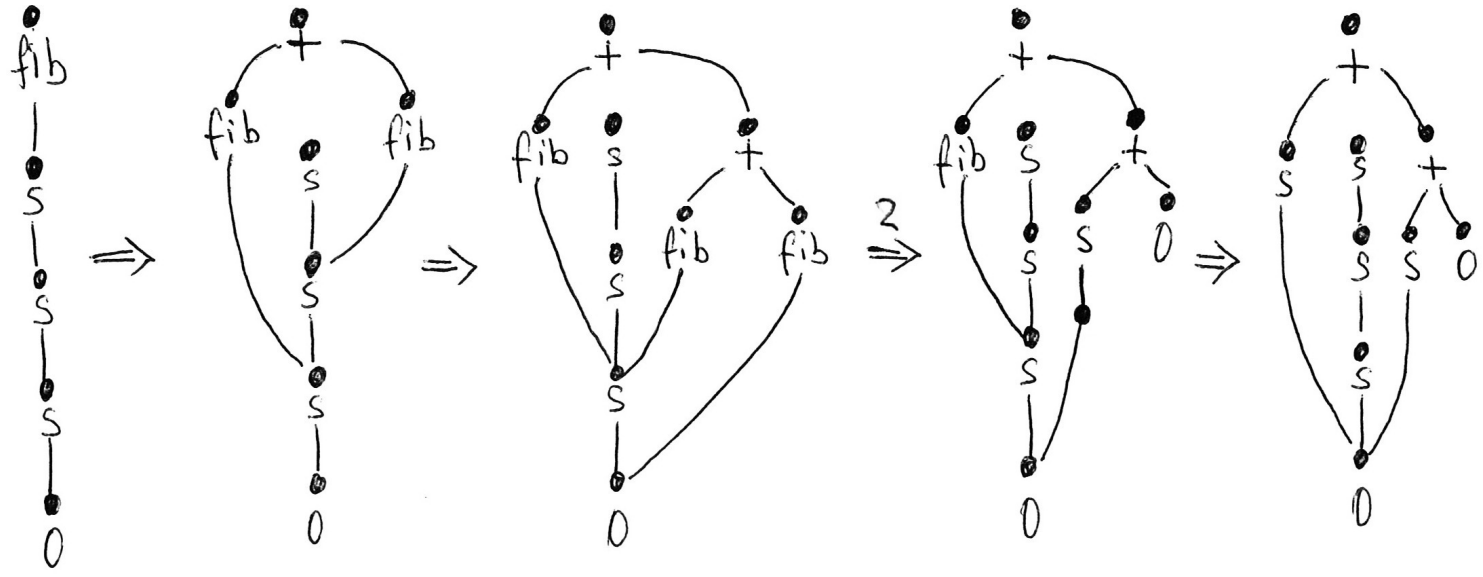
Example: Term Rewriting by Graph Transformation

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



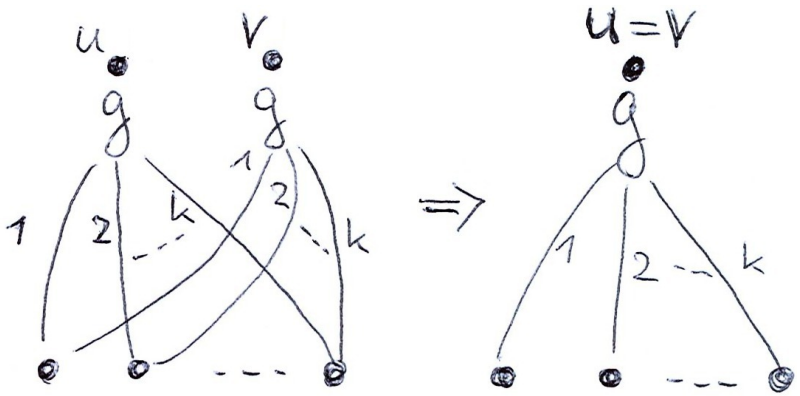




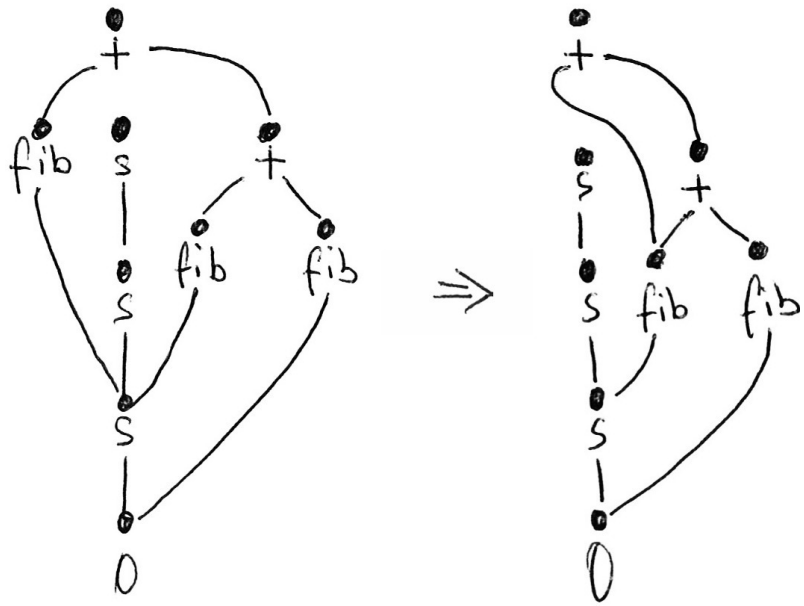


Collapsing and Garbage

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



collapsing rule



collapsing step

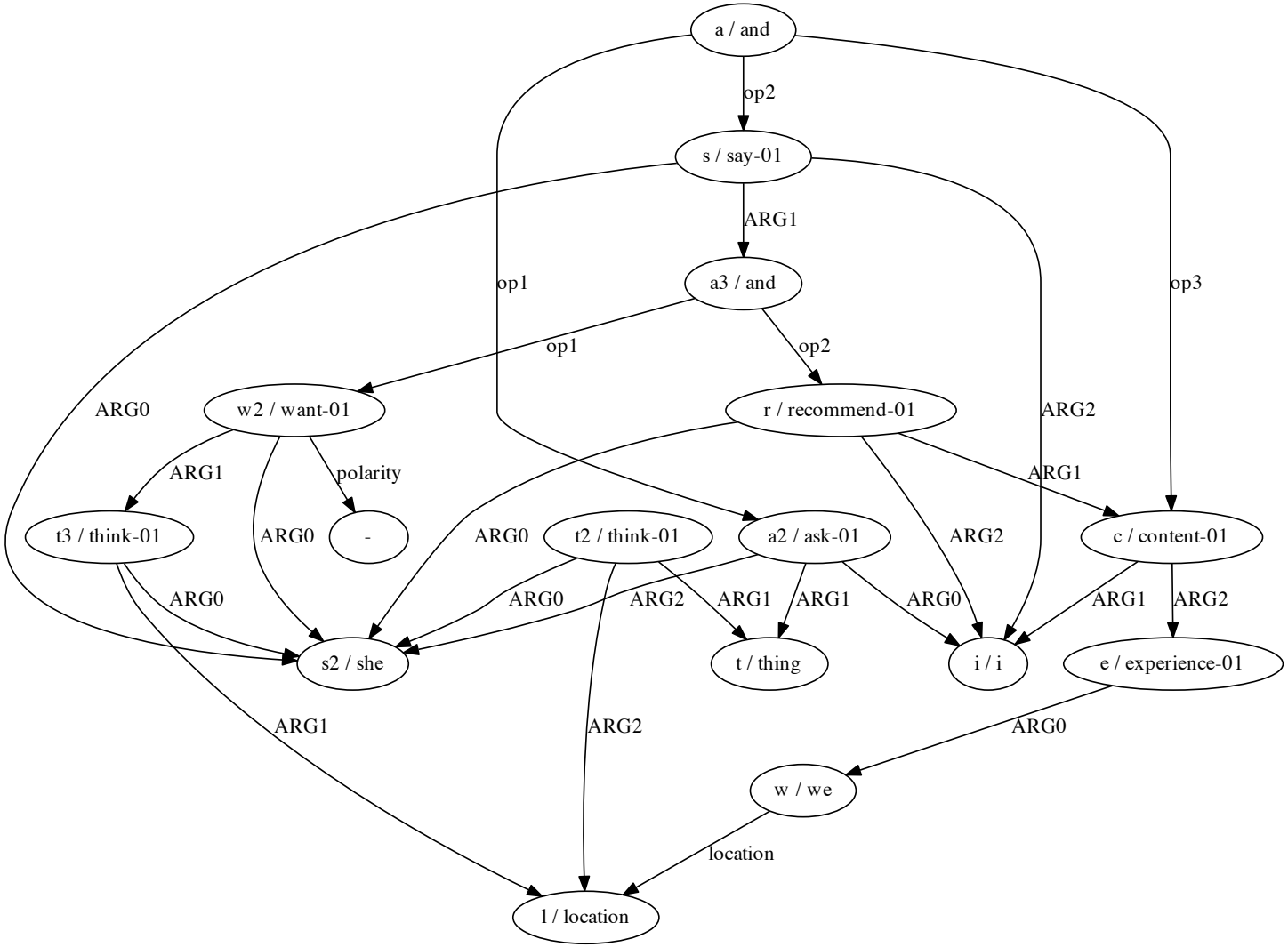
On the right we also observe the phenomenon of garbage.

DAG Automata

Automata on Directed Acyclic Graphs?

- There is no generally accepted notion of graph automata.
- The difficulty is that there is no sense of direction.
- But in many areas directed acyclic graphs (DAGs) suffice.
- Term graphs and, in NLP, meaning representations are examples.
- Various types of DAG automata have been proposed in the literature.
- For NLP, good algorithmic properties are preferable over power.
- This calls for a simple notion.
- The possibility to add weights would enhance their usefulness.
- Tree automata should be a special case.

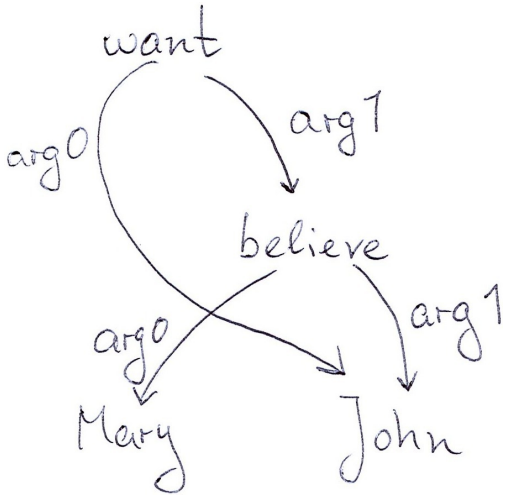
A "Real Life" DAG



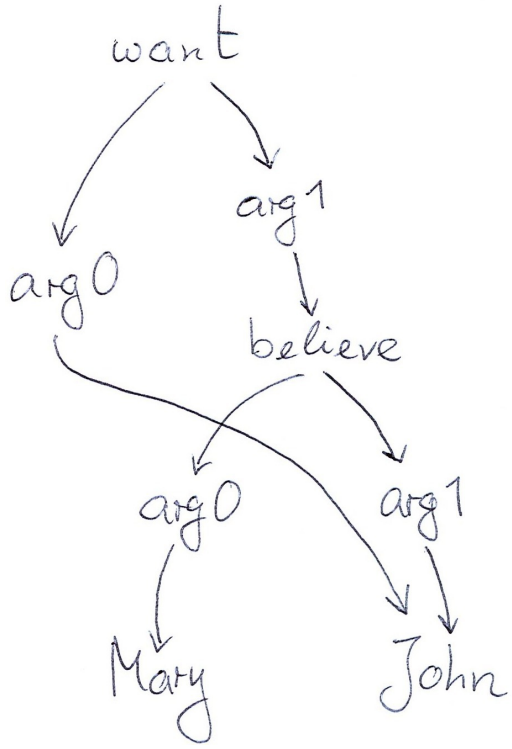
Abstract Meaning Representation (AMR) of "*I asked her what she thought about where we'd be and she said she doesn't want to think about that, and that I should be happy about the experiences we've had (which I am).*"

A Smaller AMR

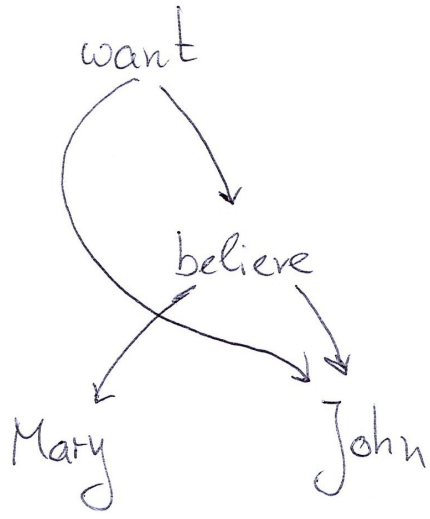
John wants Mary to believe him in slightly changing representations



use edge labels



encode using intermediate nodes

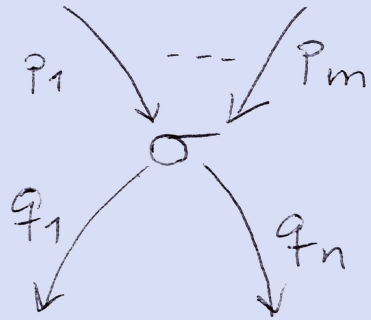


use ordered DAGs

A Simple, Ranked Type of DAG Automata

DAG automaton

1. A run is an assignment of states to edges.
2. Rules are of the form $p_1 \cdots p_m \xrightarrow{\sigma} q_1 \cdots q_n$.



3. A DAG is accepted if there is a run which is consistent with the rules at each node.

Notes

1. Tree automata are a special case.
2. No initial and final states are required.
3. DAGs may be viewed as ordered or unordered.

UMEÅ UNIVERSITET

Weighted DAG Automata

Weighted DAG automata assign a weight, say in $\mathbb{R}$, to each DAG:

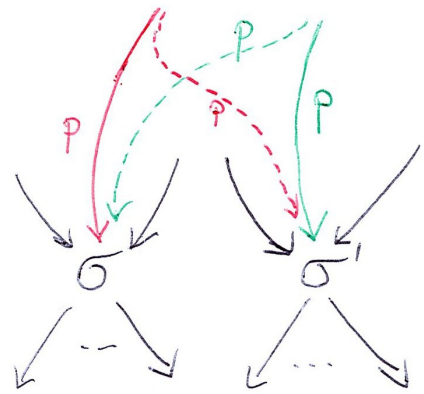
- Each rule gets a weight in $\mathbb{R}$.
- Weight 0 means that the rule "does not exist".
- The weight of a run is the product of its rules' weights.
- The weight of a DAG is the sum of the weight of all its runs.

Notes

1. $\langle \mathbb{R}, +, \cdot \rangle$ may be replaced by any commutative semiring.
2. The Boolean semiring corresponds to the unweighted case.
3. Again, weighted tree automata are a special case.

Invariance under Edge Swapping

In a run, targets of edges with the same state can be swapped:



Results whose proofs exploit edge swaps:

1. Two types of pumping lemmas.
2. The path language of a regular DAG language is regular.
3. If L is finite, it is regular iff no $D \in L$ contains an undirected cycle.
4. Unfolding turns regular DAG languages into regular tree languages.
5. Emptiness and finiteness are decidable in polynomial time.
6. Deterministic DAG automata can be minimized uniquely, and thus checked for language equivalence, in polynomial time.

UMEÅ UNIVERSITET

More Results

More results:

1. The class of HR languages is closed under intersection with regular DAG languages (by a Bar-Hillel construction).
2. Even non-uniform membership is, once again, NP-complete.
3. But bounded treewidth yields a polynomial algorithm.
4. All of this can be extended to DAG automata on DAGs with unbounded in- and out-degree. In these automata, heads and tails of rules are given by regular expressions.

Implementations and Literature

The Attributed Graph Grammar System

AGG, TU Berlin, developed since the beginning of the 1990s; latest release so far is from 2015.

- Algebraic graph transformation (single-pushout, see 1st lecture)
- Extended by positive and negative application conditions
- Nodes and edges can have attributes which are Java objects
- Comes with a graphical user interface, but you can also use the transformation engine alone for building your own system

URL: `http://www.user.tu-berlin.de/o.runge/agg/`

GRaphs for Object-Oriented VErification

Groove, University of Twente, developed since 2003 (?), latest release just a few weeks old.

- Algebraic graph transformation (single-pushout, see 1st lecture)
- Intended as a verification tool that explores the state space generated by the rules
- Three separately available subsystems: interactive graphical user interface, graph transformation engine, model checker for verifying linear temporal logic formulae

URL: `http://groove.cs.utwente.nl`

GRaphs for Object-Oriented VErification

GrGen.NET, University of Karlsruhe, developed since 2003, latest release from Feb 2016.

- Algebraic graph transformation (single-pushout, see 1st lecture) with some additions and ordinary programming constructs
- Extremely efficient rule application
- Comes with a graphical user interface for rapid prototyping, but the libraries can be used without

URL: `http://www.info.uni-karlsruhe.de/software/grgen`

The Diagram Editor Generator

DiaGen/DiaMeta, UniBw Munich, developed since a long time back, latest release from 2012(?)

- Diagram editor generator that includes a hyperedge-replacement engine and parser used for specifying/checking the logical structure of diagrams
- The HR engine/parser can be used as a backend for other purposes
- Parsing is fault tolerant to handle inputs in a meaningful way even if they are only partially correct

URL: `https://www.unibw.de/inf2/DiaGen`

Bolinas

Bolinas, USC/ISI, developed since 2012(?), latest release from 2013

- Python package for synchronous hyperedge-replacement grammars with weights
- Algorithms for rule extraction from examples, training, parsing, k-best derivation.

URL: `http://www.isi.edu/licensed-sw/bolinas`

Algebraic Language Toolkit

Alto, University of Potsdam, latest release from just a few weeks ago.

- Not a dedicated graph transformation system, but interpreted regular tree grammars
- One of the available algebras is Courcelle's HR algebra
  $\Rightarrow$ yields an implementation of (synchronous) HR grammars
- Includes parsing and other related algorithms
- Claims to be "the fastest published HRG parser in the world"

URL: `https://bitbucket.org/tclup/alto`

Some Books and Surveys

- Ehrig, Ermel, Golas, Hermann: *Graph and Model Transformation – General Framework and Applications*. Springer 2015.
- Courcelle, Engelfriet: *Graph Structure and Monadic Second-Order Logic – A Language-Theoretic Approach*. Cambridge University Press 2012.
- Ehrig, Ehrig, Prange, Taentzer: *Fundamentals of Algebraic Graph Transformation.* Springer 2006.
- *Handbook of Graph Grammars and Computing by Graph Transformation*, World Scientific. Vol. 1: Foundations (1997). Vol. 2: Applications, Languages and Tools (1999). Vol. 3: Concurrency, Parallelism, and Distribution (1999)
- Joost Engelfriet: *Context-Free Graph Grammars*. In Handbook of Formal Languages, Vol 3: Beyond Words. Springer 1997
- Annegret Habel: *Hyperedge Replacement: Grammars and Languages*. LNCS 643, Springer 1992