A Survey of Recent Advances in Efficient Parsing for Graph Grammars

FSMNLP 2019

F. Drewes

Overview

Introduction

Context-Free Graph Grammars and Parsing

Brief facts about context-free graph grammars:

1. emerged in the 1980s
2. generalization of context-free string grammars to graphs
3. can easily generate NP-complete graph languages
   $\Rightarrow$ even non-uniform parsing is impractical
4. early polynomial solutions were merely of theoretical interest:
   - strong restrictions
   - restrictions difficult to check
   - degree of polynomial usually depends on grammar
5. renewed interest nowadays due to Abstract Meaning Representation and similar notions of semantic graphs in computational linguistics.

Different Strategies

Recent attempts use different strategies to deal with NP-completeness:

1. Do your best, but be prepared to pay the price in the worst case.
2. Generate deterministic parsers based on LL- or LR-like restrictions.
3. Make sure that the generated graphs have a unique decomposition which determine the structure of derivation trees.

exponential

$\downarrow$

polynomial

$\downarrow$

uniformly polynomial

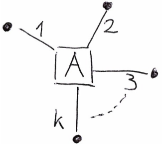
This talk will summarize those approaches.

Context-Free Graph Grammars

Here: hyperedge-replacement grammars

Hypergraphs

Graphs contain labelled hyperedges instead of edges:



The number k is the rank of A and of the hyperedge.

Rank 2 yields an ordinary edge: 

is 

.

Some nodes may be marked $1, 2, \ldots, p$ and are called ports.
The number p is the rank of the hypergraph.

From now on: "edge" means "hyperedge"
"graph" means "hypergraph"

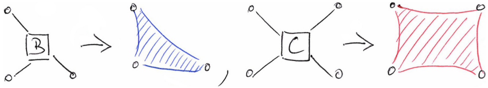
Hyperedge Replacement (HR)

Hyperedge replacement:

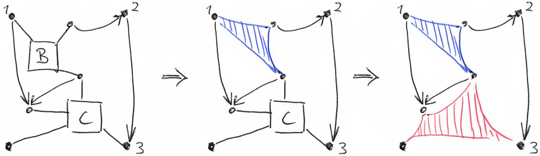
- A rule $A \to H$ consists of a label A and a graph H of equal rank.
- Rule application:
  1. remove a hyperedge e with label A,
  2. insert H by fusing its ports with the incident nodes of e.

Example

Rules:



Derivation:

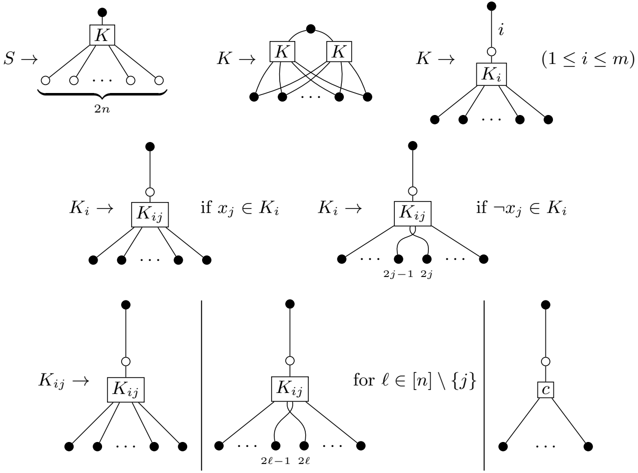


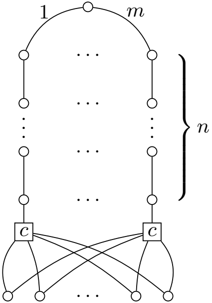
Why is Parsing Difficult?

Cocke-Kasami-Younger for HR works, but is inefficient because a graph has exponentially many subgraphs.

Even when this is not the problem, we still have too many ways to order the attached nodes of nonterminal hyperedges...

Consider a propositional formula $K_1 \wedge \cdots \wedge K_m$ over $x_1, \ldots, x_n$ in CNF.





[1]H. Björklund et al., LNCS 9618, 2016

Early Approaches to HR Grammar Parsing

- Cocke-Kasami-Younger style:
  - Conditions for polynomial running time[3]
  - DiaGen[4]
- Cubic parsing of languages of strongly connected graphs[5] [6]
- After that, the area fell more or less silent for almost 2 decades.

Then came Abstract Meaning Representation[7], and with it a renewed interest in the question.

[3] Lautemann, Acta Inf. 27, 1990

[4] Minas, Proc. IEEE Symposium on Visual Languages 1997

[5] W. Vogler, LNCS 532, 1990

[6] D., Theoretical Computer Science 109, 1993

[7] Banarescu et al., Proc. 7th Linguistic Annotation Workshop, ACL 2013

Recent General Approaches to HRG Parsing

Choosing Generality over (Guaranteed) Efficiency

Approaches that avoid restrictions (exponential worst-case behaviour):

- Lautemann's algorithm refined by efficient matching[8], implemented in Bolinas
- S-graph grammar parsing[9], using interpreted regular tree grammars as implemented in Alto
- Generalized predictive shift-reduce parsing[10], implemented in Grappa

[8] Chiang et al., ACL 2013
[9] Groschwitz et al., ACL 2015
[10] Hoffmann & Minas, LNCS 11417, 2019

The Approach by Chiang et al.

- Use dynamic programming to determine, for "every" subgraph G' of the input G, the set of nonterminals A that can derive G'.
- "Every": Consider G' that can be cut out along $rank(A)$ nodes.
- For efficient matching of rules, use tree decompositions of right-hand sides.

The algorithm runs in time $O((3^d n)^{k+1})$ where

- d is the node degree of G,
- n is the number of nodes, and
- k is the width of tree decompositions of right-hand sides.

Important: G is assumed to be connected!

- Instead of HR, use the more primitive graph construction operations by Engelfriet and Courcelle with interpreted regular tree grammars[11].
- Strategy (parsing by intersection):
  - Compute regular tree language L_G of all trees denoting G.
  - Intersect with the language of the grammar's derivation trees.
  - Trick: use a lazy approach to avoid building L_G explicitly.

The algorithm runs in time $O(n^s 3^{sep(s)})$ where

- s is the number of source names ($\sim$ number of ports)
- $sep(s)$ is Lautemann's s-separability ($\leq n$)

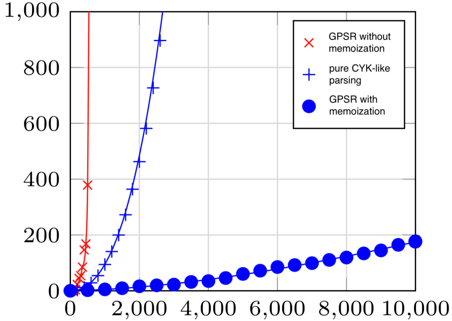
Alto is reported to be 6722 times faster than Bolinas on a set of AMRs from the "Little Prince" AMR-bank.

[11] Koller & Kuhlmann, Proc. Intl. Conf. on Parsing Technologies 2011

Generalized Predictive Shift-Reduce Parsing

- A compiler generator approach.
- Use LR parsing from compiler construction, but allow conflicts.
- Parser uses characteristic finite automaton to select actions.
- In case of conflicts, use breadth-first search implemented with graph structured stack.
- In addition, use memoization.

Grappa measurements for a grammar generating Sierpinski graphs (by M. Minas):



LL- and LR-like Restrictions to Avoid Backtracking

Predictive Parsing

Two versions of predictive parsing:

- deterministic recursive descent, generalizing SLL string parsing
  → predictive top-down[12]
- deterministic bottom-up, generalizing SLR string parsing
  → predictive shift-reduce[13]

Common modus operandi:

- View right-hand side as a list of edges to be matched step by step.
- Terminal edges are "consumed" from the input graph.
- Nonterminal edges are handled by recursive call (top-down) or reduction (bottom-up).

[12] D., Hoffmann, Minas, LNCS 10373, 2015

[13] D., Hoffmann, Minas, J. Logical and Alg. Methods in Prog. 104, 2019

Predictive Top-Down Parsing (PTD)

In PTD parsing, each nonterminal A becomes a parsing procedure:

- parser generator determines lookahead for every A-rule:
  rest graphs (lookahead sets) for alternative A-rules must be disjoint
  $\Rightarrow$ the current rest graph determines which rule to apply;
- in doing so, we have to distinguish between different profiles of A;
- alternative terminal edges require free edge choice.

Lookahead and free edge choice are approximated by Parikh sets to obtain efficiently testable conditions.

Running time of generated parser is $O(n^2)$.

Predictive Shift-Reduce Parsing (PSR)

PSR parsing reduces the input graph back to the initial nonterminal:

- parser maintains a stack representing the graph to which the input read so far has been reduced
- shift steps read the next terminal edge from the input graph (free edge choice needed here as well)
- reduce steps replace rhs on top of stack with lhs
- parser generator determines characteristic finite automaton (CFA) that guides the choice of shift and reduce steps
- CFA must be conflict free
- string parsing only faces shift-reduce and reduce-reduce conflicts; now there may also be shift-shift conflicts.

Running time of generated parser is $O(n)$.

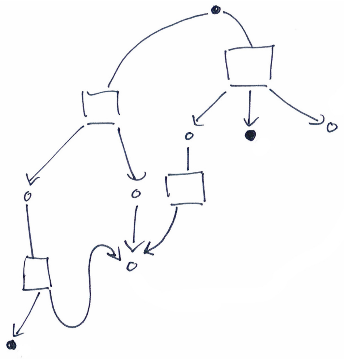
Unique Decomposability

Reentrancies

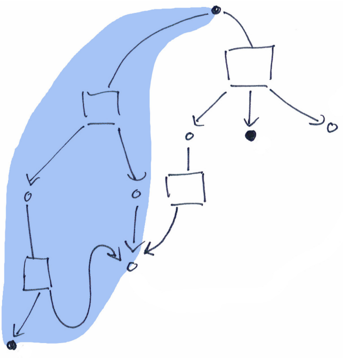
- PTD and PSR grammar analysis can be expensive for large grammars.
- In NLP, grammars may be volatile and very large
  $\Rightarrow$ uniformly polynomial parsing may be preferable.
- Restrictions take inspiration of Abstract Meaning Representation, viewing graphs as trees with reentrancies.
- Original strong assumptions[14] were later relaxed[15] and extended to weighted HR grammars[16].
- This type of HR grammar can also be learned à la Angluin[17].

[14] H. Björklund et al., LNCS 9618, 2016
[15] H. Björklund et al., 2018 (under review)
[16] H. Björklund et al., Mathematics of Language 2018
[17] J. Björklund et al., LNCS 10329, 2017

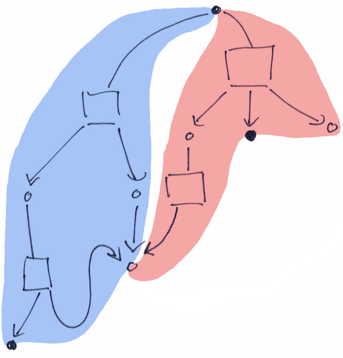
Reentrancies in a nutshell (bullets are ports)



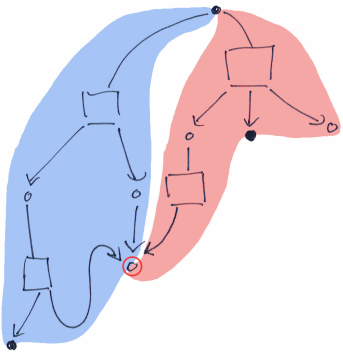
Reentrancies in a nutshell (bullets are ports)



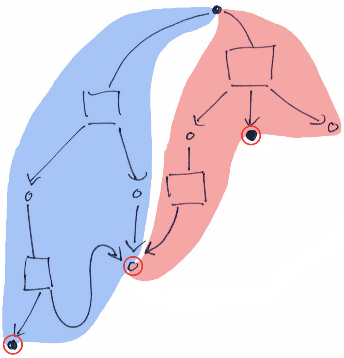
Reentrancies in a nutshell (bullets are ports)



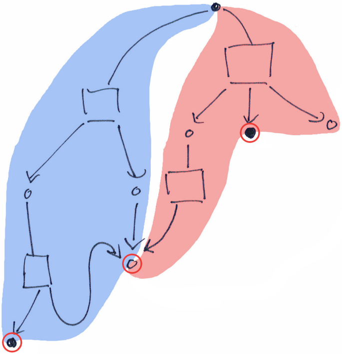
Reentrancies in a nutshell (bullets are ports)



Reentrancies in a nutshell (bullets are ports)



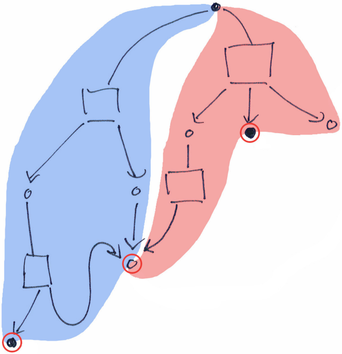
Reentrancies in a nutshell (bullets are ports)



Requirements on right-hand sides:

1. targets of every nonterminal hyperedge e are reentrant w.r.t. e
2. all nodes reachable from the root

Reentrancies in a nutshell (bullets are ports)

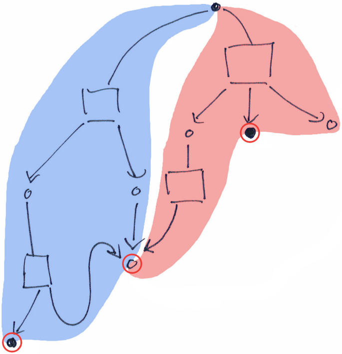


Requirements on right-hand sides:

1. targets of every nonterminal hyperedge e are reentrant w.r.t. e
2. all nodes reachable from the root

Yields a unique hierarchical decomposition revealing the structure of derivation trees.

Reentrancies in a nutshell (bullets are ports)



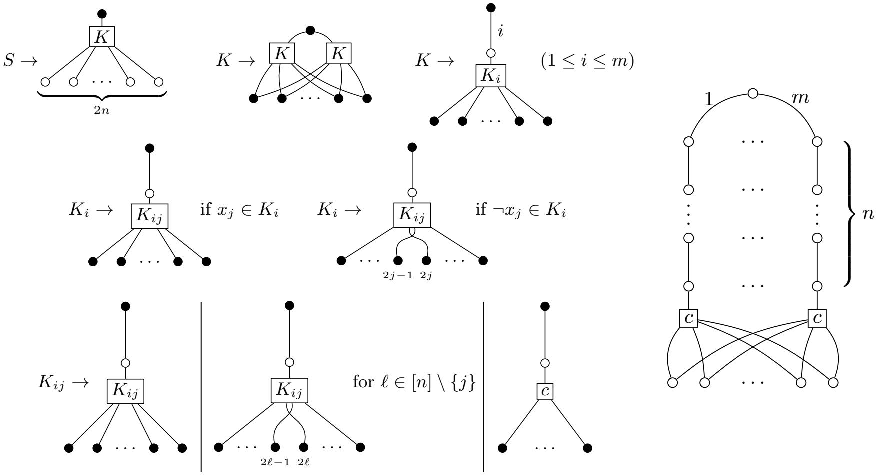
Requirements on right-hand sides:

1. targets of every nonterminal hyperedge e are reentrant w.r.t. e
2. all nodes reachable from the root

Yields a unique hierarchical decomposition revealing the structure of derivation trees.

However, there is one problem left...

Recall: Reducing SAT



Conclusion: we also need order preservation!

We must provide a binary relation on nodes that

1. is efficiently computable,
2. coincides with the order of targets of nonterminal edges, and
3. is compatible with hyperedge replacement.

Theorem

For a reentrancy and order preserving HRG $\mathcal{G}$ and a graph G as input, $G \in L(\mathcal{G})$ can be decided in time

$$O(\max(|\mathcal{G}|, |G|)^2).$$

This holds also for computing the weight of G if the rules of $\mathcal{G}$ have weights from a commutative semiring.

Systems and Tools

Bolinas[18] (USC/ISI, D. Bauer, K. Knight) implements the parser of (Chiang et al., ACL 2013).

Main features:

- weighted rules
- n-best derivations
- translation via synchronous HR grammars
- EM training from corpora

[18] http://www.isi.edu/licensed-sw/bolinas

Alto[19] (A. Koller) implements interpreted regular tree grammars.

One instantiation is the HR parser of (Koller & Kuhlmann, 2011).

Main features correspond to those of Bolinas:

- weighted rules
- n-best derivations
- translation via synchronous HR grammars
- EM training from corpora

[19] http://github.com/coli-saar/alto

Grappa[20] (M. Minas) provides parser generators for HRG grammars.

Main features:

- generators for predictive top-down (PTD), predictive shift-reduce (PSR), generalized PSR parsers
- can generate PTD and PSR parsers for contextual HR grammars[21]
- is constantly being improved and extended
- has a tasty logo

[20] http://www.unibw.de/inf2/grappa
[21] Drewes & Hoffmann, Acta Informatica 52 2015

Grappa[20] (M. Minas) provides parser generators for HRG grammars.

Main features:

- generators for predictive top-down (PTD), predictive shift-reduce (PSR), generalized PSR parsers
- can generate PTD and PSR parsers for contextual HR grammars[21]
- is constantly being improved and extended
- has a tasty logo

[20] http://www.unibw.de/inf2/grappa
[21] Drewes & Hoffmann, Acta Informatica 52 2015

Future Work?

Some Questions for Future Work

- How to make HR grammars efficiently parsable by design?
- Can HR grammars be learned from data so that they are (1) small and (2) efficiently parsable?
- What are useful and benign extensions that can be handled efficiently (like contextual HR)?
- How to handle node labels in a good way (e.g., enabling relabelling)?
- Efficient transductions that turn strings/trees into graphs?

Thank you!

Questions?