Automata and Grammars for Weighted Graph Languages

(with emphasis on parsing and weight computation)

Frank Drewes

Department of Computing Science
Umeå University

23 June 2021

Contents

Outline

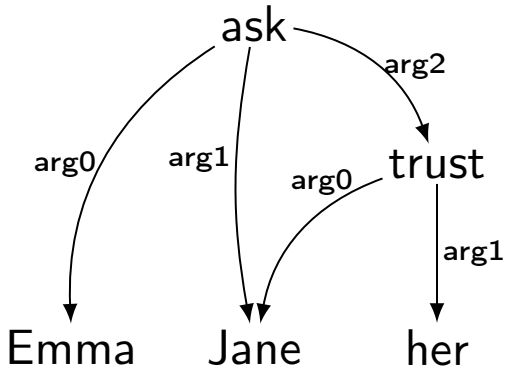
Graphs

Graphs occur everywhere in computer science.

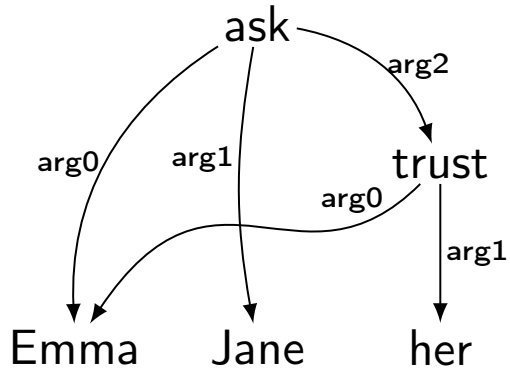
Usually, there have to adhere to some well-formedness requirement.

Examples: knowledge bases, program analysis, chemical structures, natural language semantics, ...

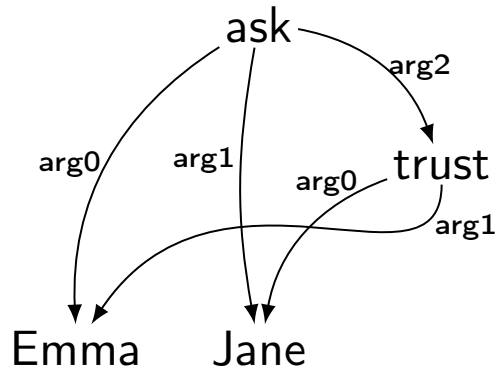
Emma asks Jane to trust her.



(correct)



(incorrect)



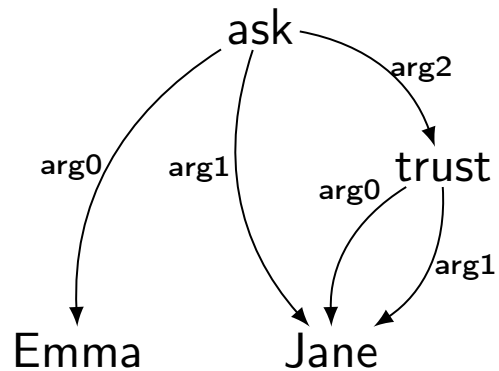
(more likely?)

Graph Languages

As usual, collecting well-formed graphs yields a graph language.

(Finite) formalisms to describe such languages are automata, grammars, logic, . . .

Weights (e.g., probabilities) become useful if some are more likely than others or only somewhat correct.



Emma asks Jane to trust her(self?).

Lost in Graphs

Major difficulty when considering graphs instead of strings/trees:

Graphs do not provide a natural processing order.

No self-evident unique starts and ends.

Associative-commutative substructures.

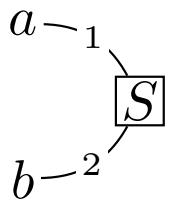
Disconnectedness.

Automorphisms.

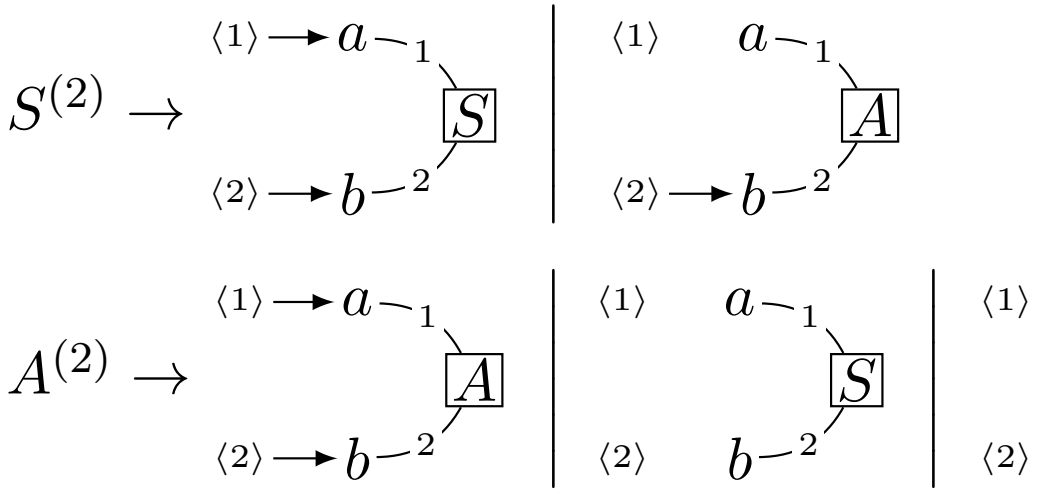
Outline

An Old NP-Completeness Result

[Aalbersberg, Rozenberg, Ehrenfeucht '86] (also [Lange, Welzl '87])

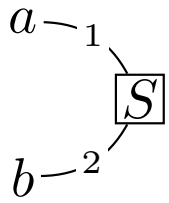
Start: 

Rules:

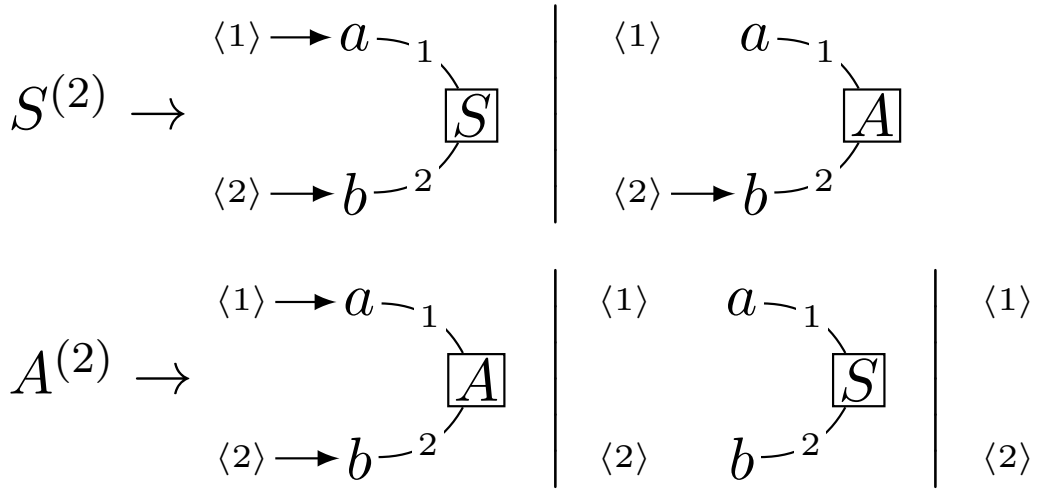


An Old NP-Completeness Result

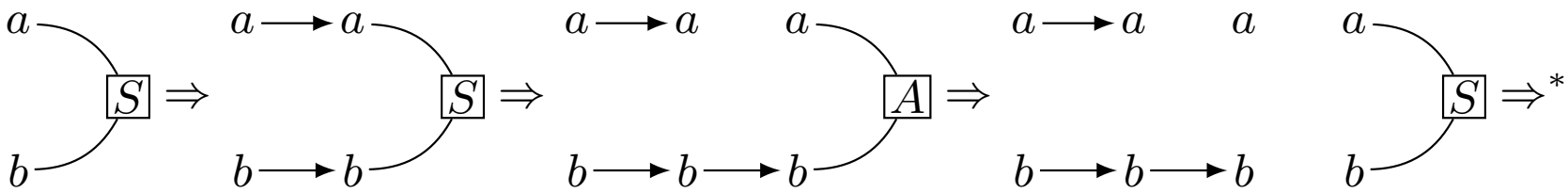
[Aalbersberg, Rozenberg, Ehrenfeucht '86] (also [Lange, Welzl '87])

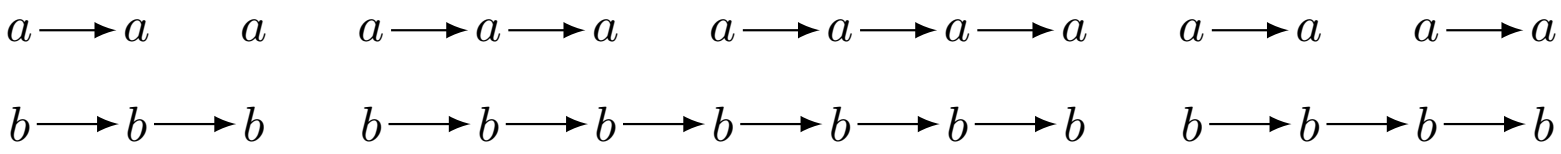
Start: 

Rules:



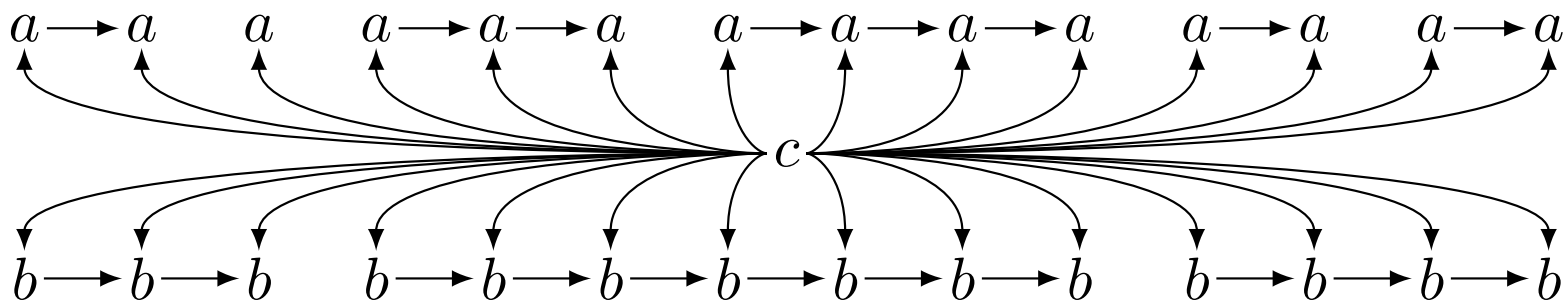
Derivation:





For Those Who Dislike Disconnected Graphs

To generate connected graphs, add a node with edges to all nodes:



(Together, connectedness and bounded node degree do help.)

Outline

A Simple Notion of DAG Automata

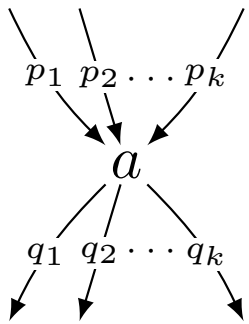
[Chiang, FD, Gildea, Lopez, Satta '18 & Blum, FD '19])

Consider directed acyclic graphs (labeled nodes, unlabeled edges).

Runs assign states to edges.
Rules are of the form

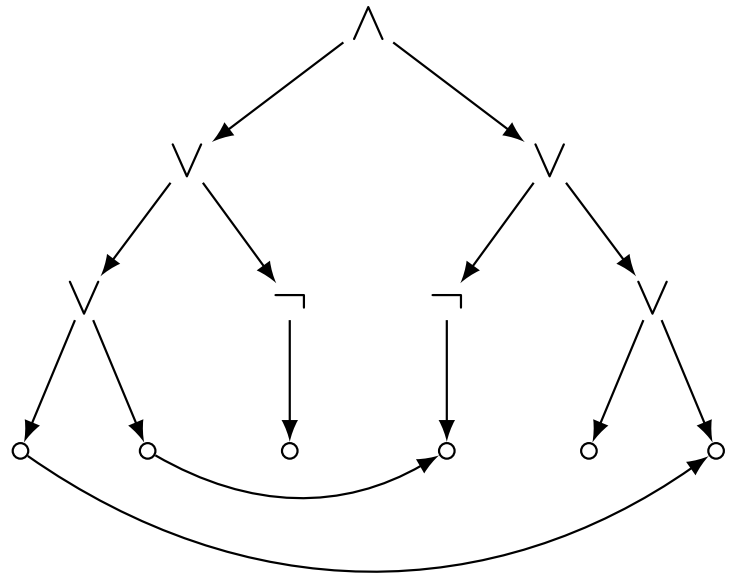
$$\langle p_1, \ldots, p_k \rangle \xrightarrow{a} \langle q_1, \ldots, q_\ell \rangle$$

Graphically:



Simple formalism, nice theory, but again NP-complete.

Reduction from SAT:



A Weighted Version

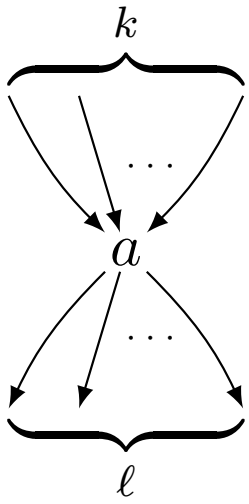
Weights from a semiring $\mathbb{S}$ can be added to rules as usual.

Expectedly,

- rule weights of a run are multiplied
- weights of alternative runs are summed up.

We can compute the weight of a DAG by edge contraction.

Weight Computation

For every star 

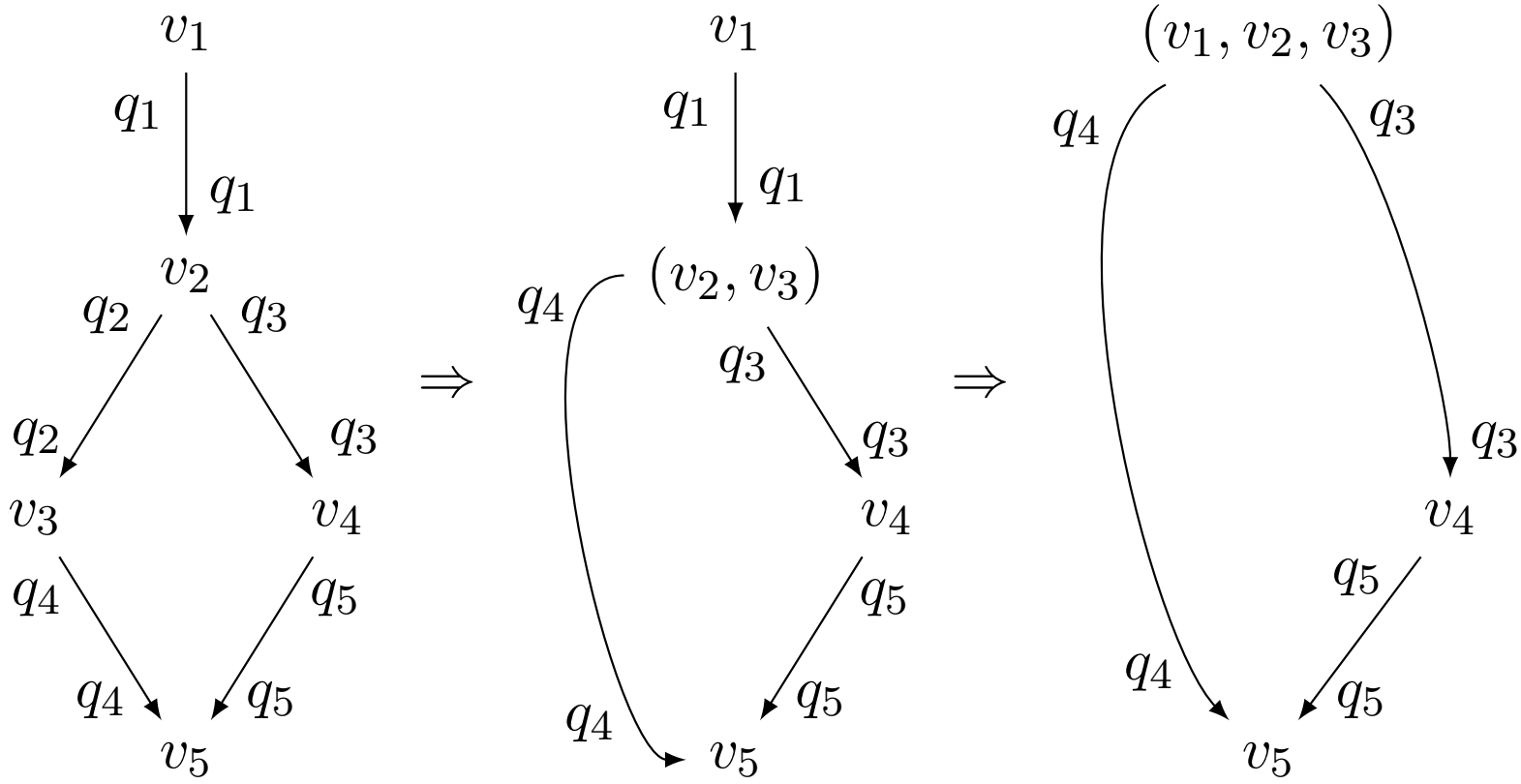
maintain a partial mapping of weight assignments $\alpha\colon Q^{k+\ell} \rightharpoonup \mathbb{S}$.

Initially, weights are given by the rules $\langle p_1, \ldots, p_k \rangle \xrightarrow{a/w} \langle q_1, \ldots, q_\ell \rangle$.

In each step, an edge is contracted, joining two stars.

When only one node is left, $\alpha() \in \mathbb{S}$ is the computed weight.

Weight Computation Algorithm – Example



$\Rightarrow^2 (v_1, v_2, v_3, v_4, v_5)$

Weight Computation Algorithm – Efficiency

Running time depends on the how big the produced stars become.

$\Rightarrow$ order of edge contractions is important.

An optimal tree decomposition of the DAG's line graph $\mathcal{LG}(D)$ yields the most economical order.

Running time then becomes $O(|E| \cdot |Q|^{treewidth(\mathcal{LG}(D))+1})$.

In some cases this can be improved by binarizing DAGs and rules.

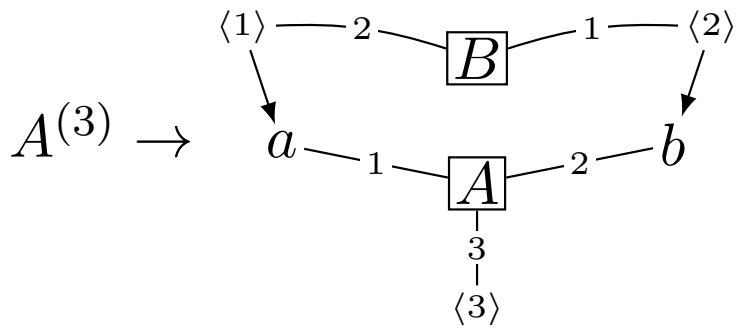
Outline

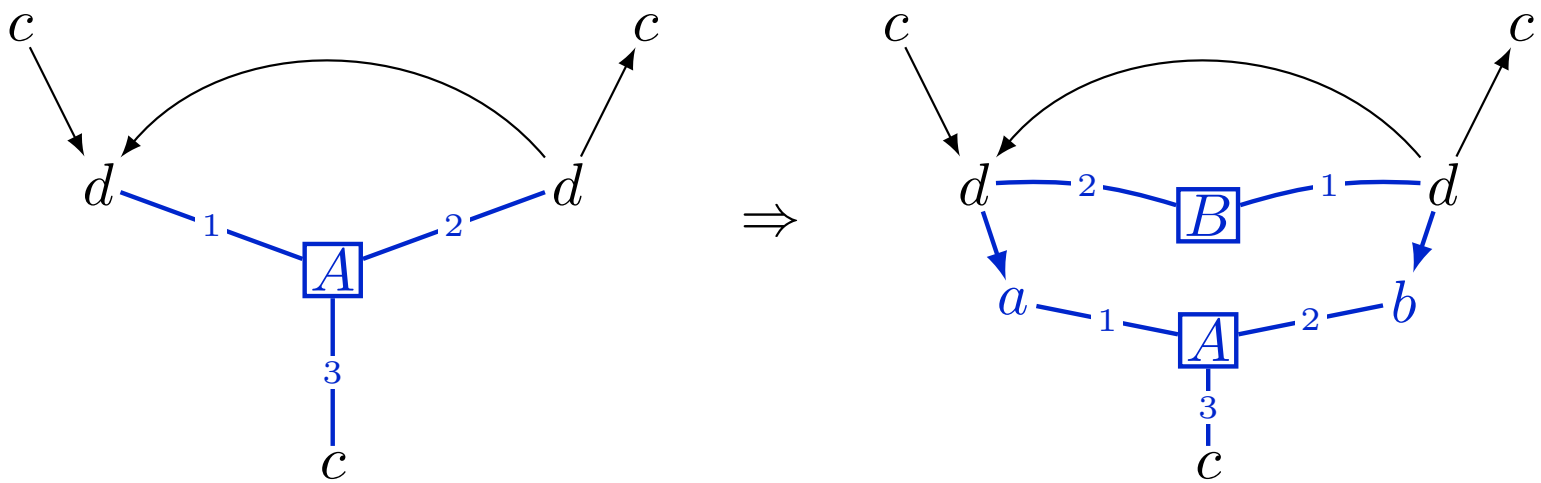
Hyperedge Replacement (HR)

HR grammars (HRGs) use rules that replace hyperedges.

Example:

(rule)





(application)

Parsing Weighted HRGs

Now, let's add weights . . .

Weight computation is difficult (by NP-completeness).

However, in general, not even the definition seems obvious (to me).

Consider rules $S^{(0)} \xrightarrow{1} \boxed{A}^{h} \; \boxed{A}^{h'}$, $A^{(0)} \xrightarrow{1} \textcircled{a}$ generating $\textcircled{a}^{v} \; \textcircled{a}^{v'}$.

We have two (concrete) derivation trees: h generates either v or v'.

But the grammar is deterministic.
We probably do not want to sum up over both.

If not, then restrictions for polynomial parsing to make CYK efficient may yield wrong results if used to compute weights.

Outline

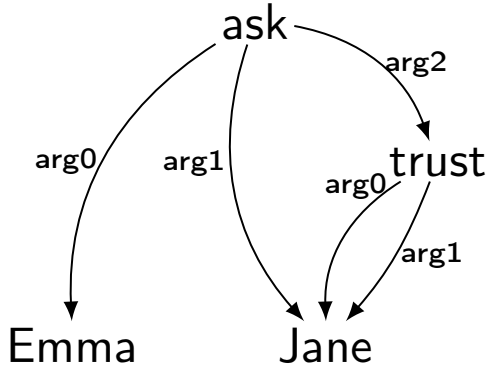
Reentrancy Preservation

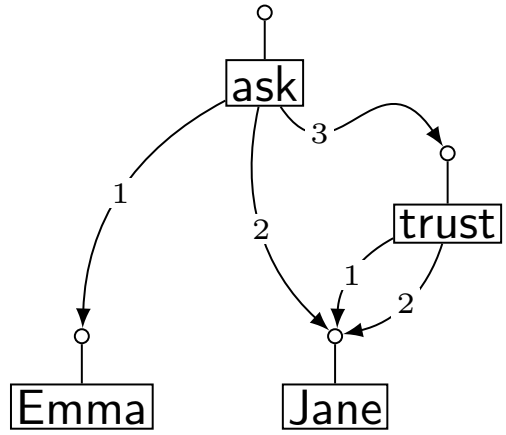
[Björklund, FD, Ericson, Starke '21, H. Björklund, FD, Ericson '19]

Possible solution: restrictions that ensure unique decompositions.

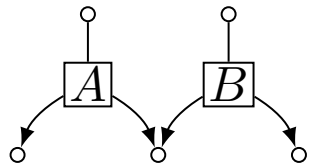
One way to achieve this: reentrancy preservation.

1. Useful because reentrancies are important in NLP.

Recall 

or 

2. Idea: decisions about reentrancies must be made immediately.



Reentrancy-Preserving HRGs

A rule is reentrancy preserving if, in its right-hand side

1. all targets of nonterminal hyperedges are reentrant,
2. all nodes are reachable from the "root" $\langle 0 \rangle$,
3. the out-degree of every node is at most 1.[1]

$\Rightarrow$ reentrancies are preserved during the course of a derivation

$\Rightarrow$ the subgraph rooted at node v that is potentially generated by nonterminal A is uniquely determined by cutting off at the reentrant nodes.

$\Rightarrow$ uniformly quadratic bottom-up parsing algorithm

Since the problems discussed earlier are avoided, this algorithm carries over to the weighted case.

[1] . . . except in so-called duplication rules.

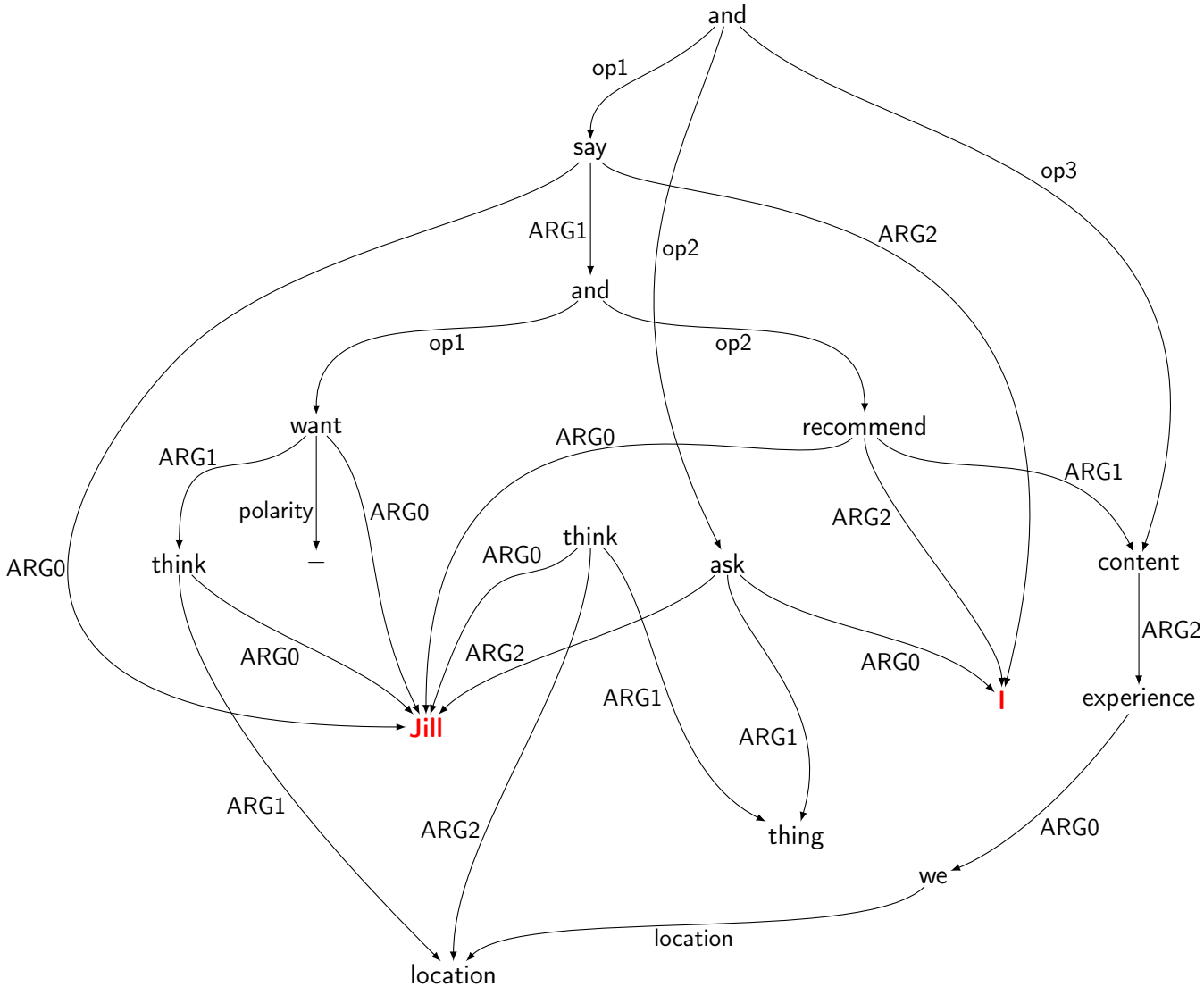
Outline

NP-Complete, yet too Weak?

In NLP, we have both

- structural reentrancies (caused by, e.g., control structures) and
- non-structural ones (caused by, e.g., pronouns).

HRGs are good at the former, but quite bad at the latter.



I asked Jill 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).

What Do We Need?

We need a way for rules to access nodes not incident with the replaced hyperedge.

Should not allow much control (to limit power).

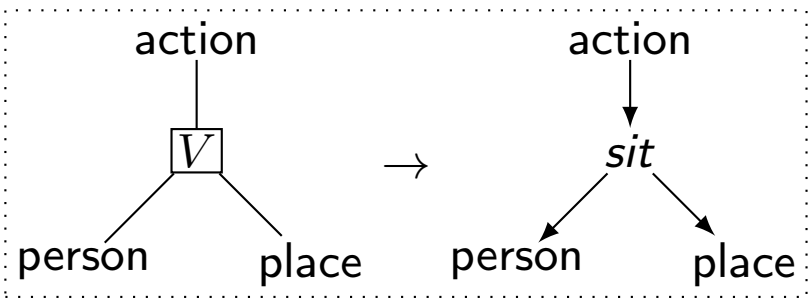
This is the idea of contextual hyperedge replacement.

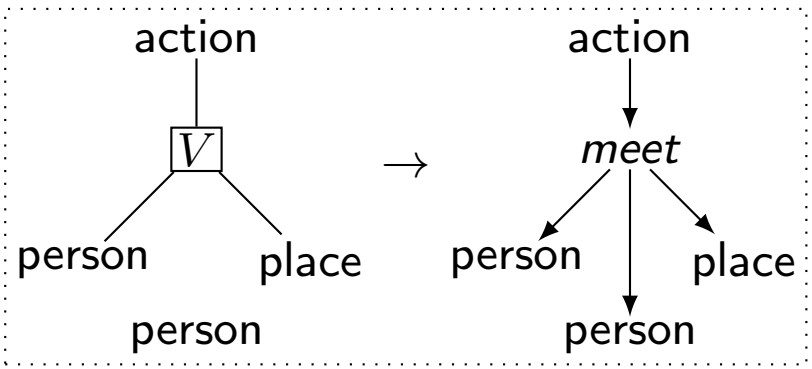
Contextual Hyperedge Replacement

[Joint work with Hoffmann & Minas]

Left-hand sides can now contain additional isolated nodes.

Such a node can refer to any (equally labeled) node in the graph
$\Rightarrow$ non-structural connections can be made.





Disadvantage: Parsing has to take cyclic dependencies into account
$\Rightarrow$ yet another detriment to efficient parsing (though still in NP)

A "tamed" version: graph extension grammars.

Detour: HRG à la Mezei & Wright

View a hypergraph H with n nonterminals $h_1, \ldots, h_n$ as an operation:

$$H(G_1, \ldots, G_n) := H[h_1 \leftarrow G_1, \ldots, h_n \leftarrow G_n]$$

Then an HRG can be understood as

- a regular tree grammar generating trees over such operations
- followed by an evaluation of the generated trees.

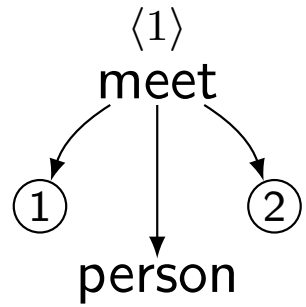
By the context-freeness of HR, this yields an equivalent view of HRGs.

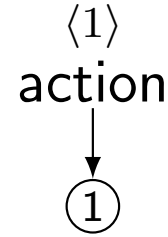
Graph Extension Grammars
[J. Björklund, FD, Jonsson '21]

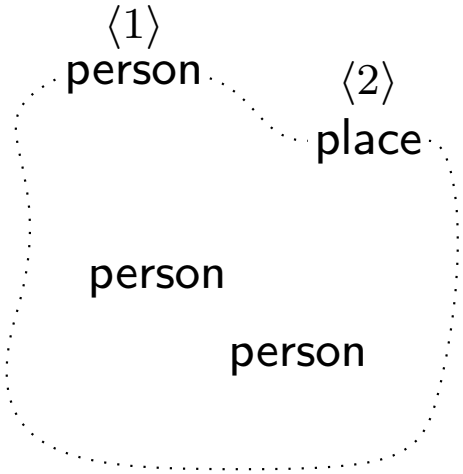
Idea of graph extension grammars:

- Use the tree-based formulation of HRGs.
- Extend operations by contextual nodes, so that they can "reach out" to nodes in the argument graph.
- Restrict operations structurally to enable polynomial parsing.

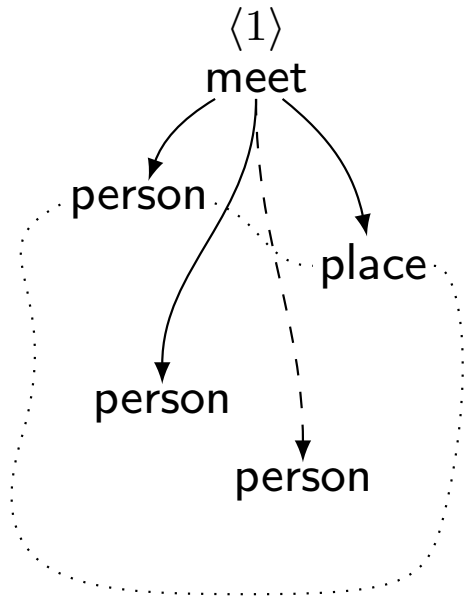
Graph Extension Operation by Example

Operations: $H_1 =$ 

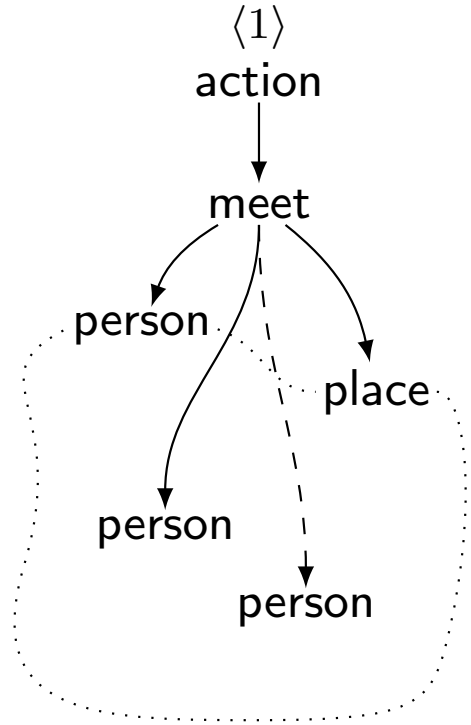
$H_2 =$ 



graph G



$H_1(G)$



$H_2(H_1(G))$

Graph Extension Grammars

A graph extension grammar consists of

1. a set of graph extension and graph union operations and
2. a regular tree grammar generating trees over those operations.

Evaluating the generated trees yields the generated graph language.

The restrictions allow for polynomial parsing.

We do not yet have a weighted version, but I would love to find out whether it's possible while preserving polynomial parsing.

If anyone is interested to join, please let us know.

Thank you very much for listening!