Lambek Grammars, Tree Adjoining Grammars and Hyperedge Replacement Grammars, Moot (2008)

René van Gasteren
LMNLP

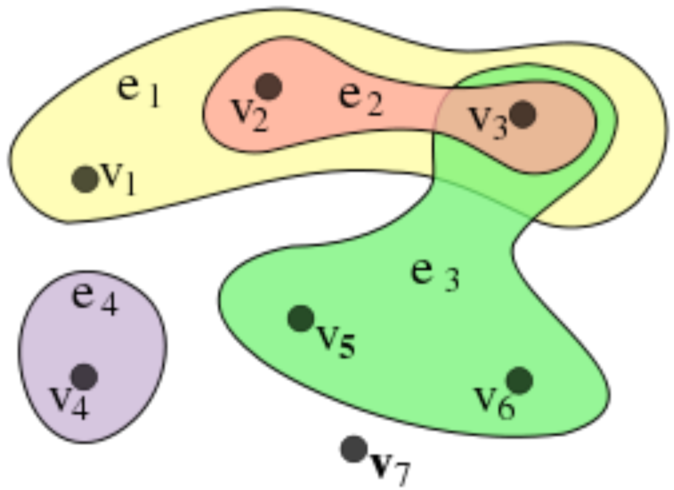
Goal

Show that both $NL\Diamond_R$ and LG generate the same class of languages as TAGs, using hyperedge replacement grammars as an intermediate step.

Hyperedge Replacement Grammars

Hypergraphs

A *hypergraph generalises the notion of graph by allowing the edges, called hyperedges, to connect not just two but any number of nodes.*



Hyperedge Replacement Grammars

Definition hypergraphs

Definition 2.1 *Let Γ be an alphabet of edge labels and let σ be an alphabet of selectors. A* hypergraph *over Γ and σ is a tuple $\langle V, E, lab, nod, ext \rangle$, where*

V is the finite set of vertices,

E is the finite set of hyperedges *disjoint with V,*

lab is the labeling function, *from E to Γ, assigning an edge label to each hyperedge,*

nod is the incidence function *that associates with each edge $e \in E$ a partial function $nod(e) : \sigma \to V$, that is, it selects a vertex for every selector σ of the edge.*

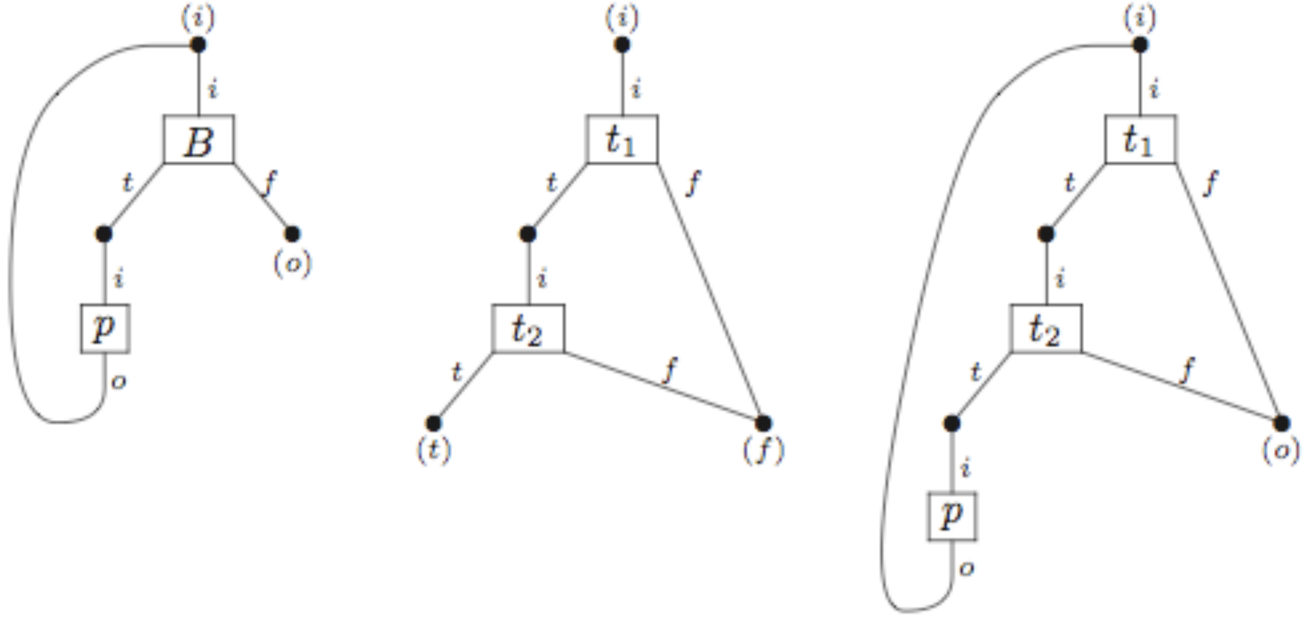
ext is the external function, *a partial function from σ to V, that is, for every selector σ of the hypergraph it select a vertex.*

Definition 2.2 *The* type *of a hypergraph H is the domain of the external function, $type(H) = dom(ext)$. The type of an edge e is the domain of the incidence fuction $type(e) = dom(nod(e))$.*

Hyperedge Replacement Grammars

Hyperedge Replacement

The operation of hyperedge replacement replaces a hyperedge by a hypergraph H of the same type



H K $H[e := K]$

Hyperedge Replacement Grammars

Definition Hyperedge Replacement

Definition 2.4 *Let H and K be two disjoint hypergraphs with the same set of edge labels Γ and the same set of selectors σ. Let e be an edge of H such that type(e) = type(K). The* hyperedge replacement *of e by G, $H[e := G] = \langle V, E, lab, nod, ext\rangle$ is defined as follows.*

$V = V_H \cup V_K$

$E = (E_H - e) \cup E_k$

$lab = lab_H \cup lab_K$ restricted to the members of E.

$nod = nod_H \cup nod_K$ restricted to the members of E.

$ext = ext_H$

For all $s \in type(e)$, $nod_H(e, s) = ext_K(s)$.

Hyperedge Replacement Grammars

Definition 2.6 *A* hyperedge replacement grammar *(or HR grammar) is a tuple* $G = \langle N, T, \sigma, P, S \rangle$ *such that.*

N *is the alphabet of nonterminal edge labels.*

T *is the disjoint alphabet of terminal edge labels.*

σ *is the alphabet of selectors.*

P *is the finite set of productions.*

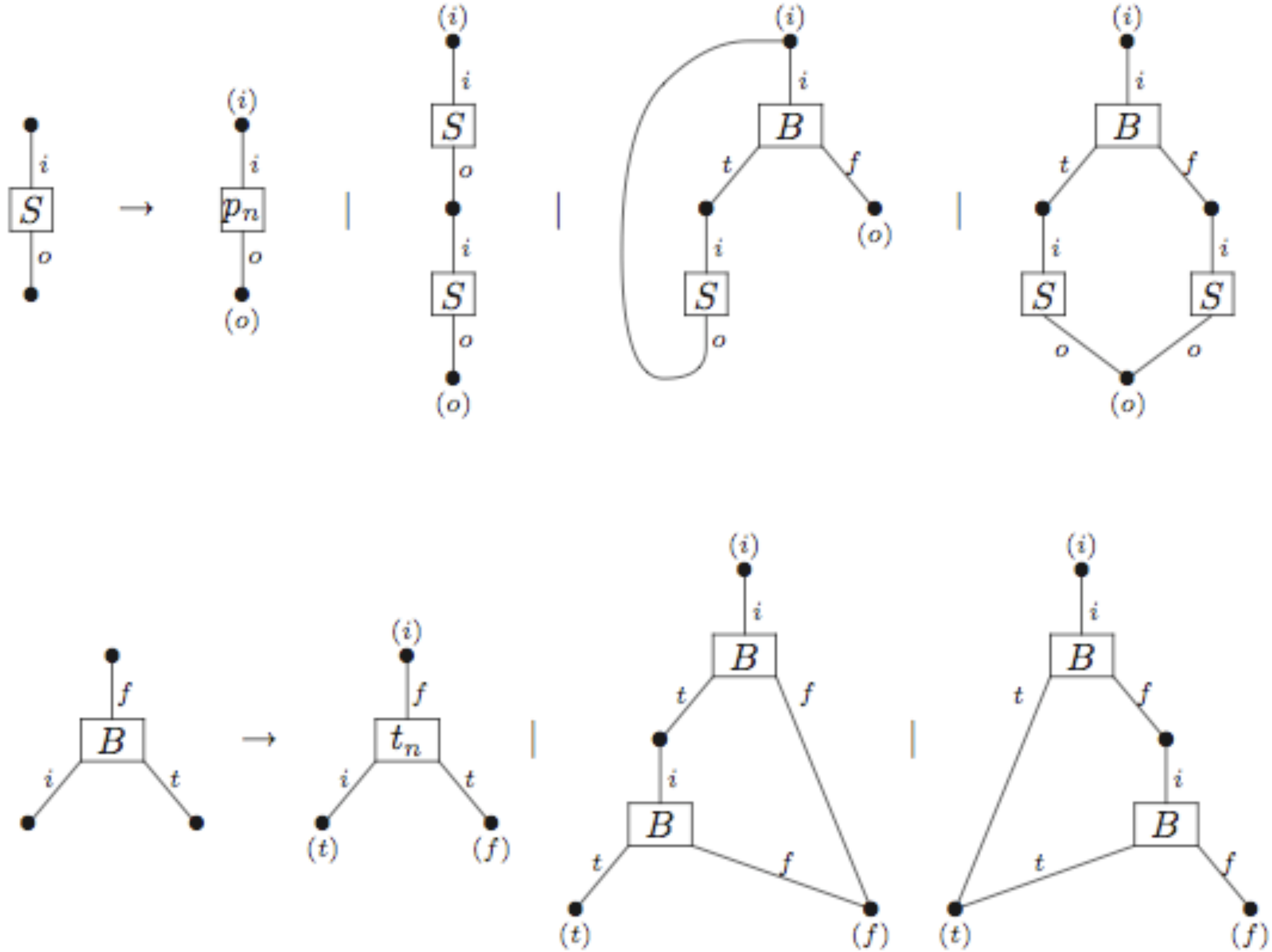
$S \in N$ *is the start nonterminal symbol.*

Definition 2.9 *Let* G *be a hyperedge replacement grammar. The* language generate by G *is the set of hypergraphs without hyperedges labeled by nonterminal edge labels derivable from* S*.*

Definition 2.10 *The* rank *of a terminal or nonterminal symbol is the number of its tentacles.*

The rank *of a hyperedge replacement grammar is the maximum rank of a nonterminal symbol in the grammar.*

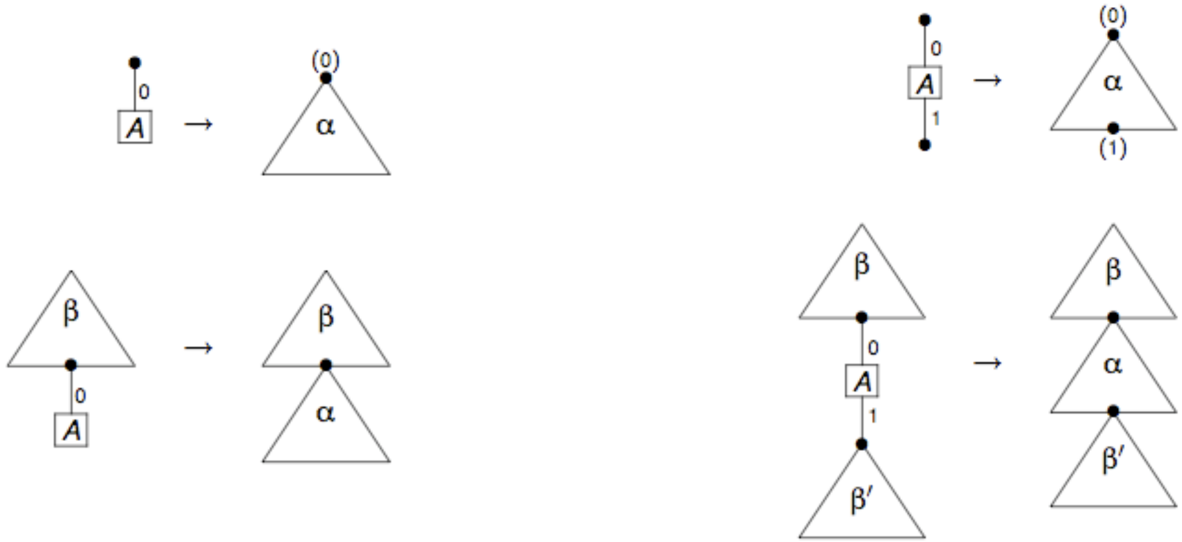
Hyperedge Replacement Grammars



Tree Adjoining Grammars as HR Grammars

Tree Adjoining Grammars can be see as a special case of hyperedge replacement grammars where:

- every non-terminal hyperedge label has at most two tentacles, that is, the rank of the grammar is (at most) two.
- every right-hand side of a HR rule is either: a tree with the root as its sole external node. a tree with a root and a leaf as its external nodes.



Tree Adjoining Grammars as HR Grammars

Moot in a presentation:

“Tree Adjoining Grammars can be seen as a special case of hyperedge replacement grammars.“

Moot in his paper*:*

“HR_2 grammars generating trees and TAG grammars are strongly equivalent.”

Question: Is this the same?

LTAG in normal form

An $LTAG_{nf}$ grammar G is an LTAG satisfying the following additional conditions:

- all internal nodes of elementary trees have exactly two daughters,
- every adjunction node either specifies the null adjunction or the obligatory adjunction con- straint without any selectional restrictions,
- every adjunction node is on the path from the lexical anchor to the root of the tree.

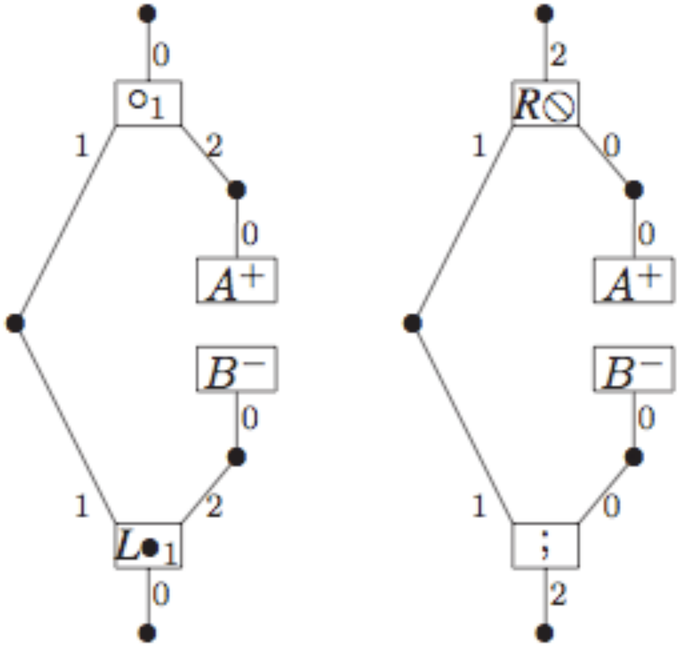
For every LTAG grammar G there is a weakly equivalent LTAG' grammar G´

LTAG$_{nf}$ as proof nets for NL$\Diamond_R$

If G is an LTAG$_{nf}$ grammar, then there exists a strongly equivalent NL$\Diamond_R$ grammar G' and a strongly equivalent LG grammar G'' tree.

Proof sketch

For each lexical tree t of G we construct a lexical tree t' in G' and a lexical tree t'' in G'' , translating every adjunction point by the left hand side of the figure for G' and by its right hand side for G''

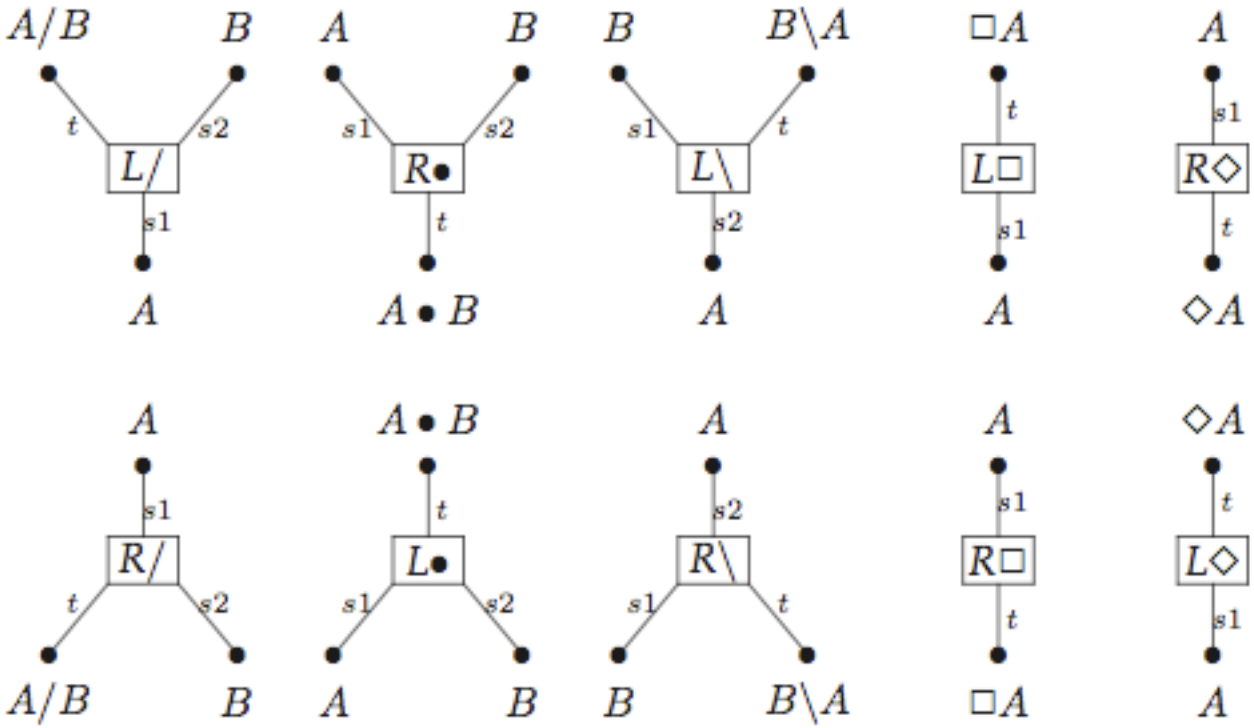


Whenever we substitute a tree….

Whenever we adjoin a tree….

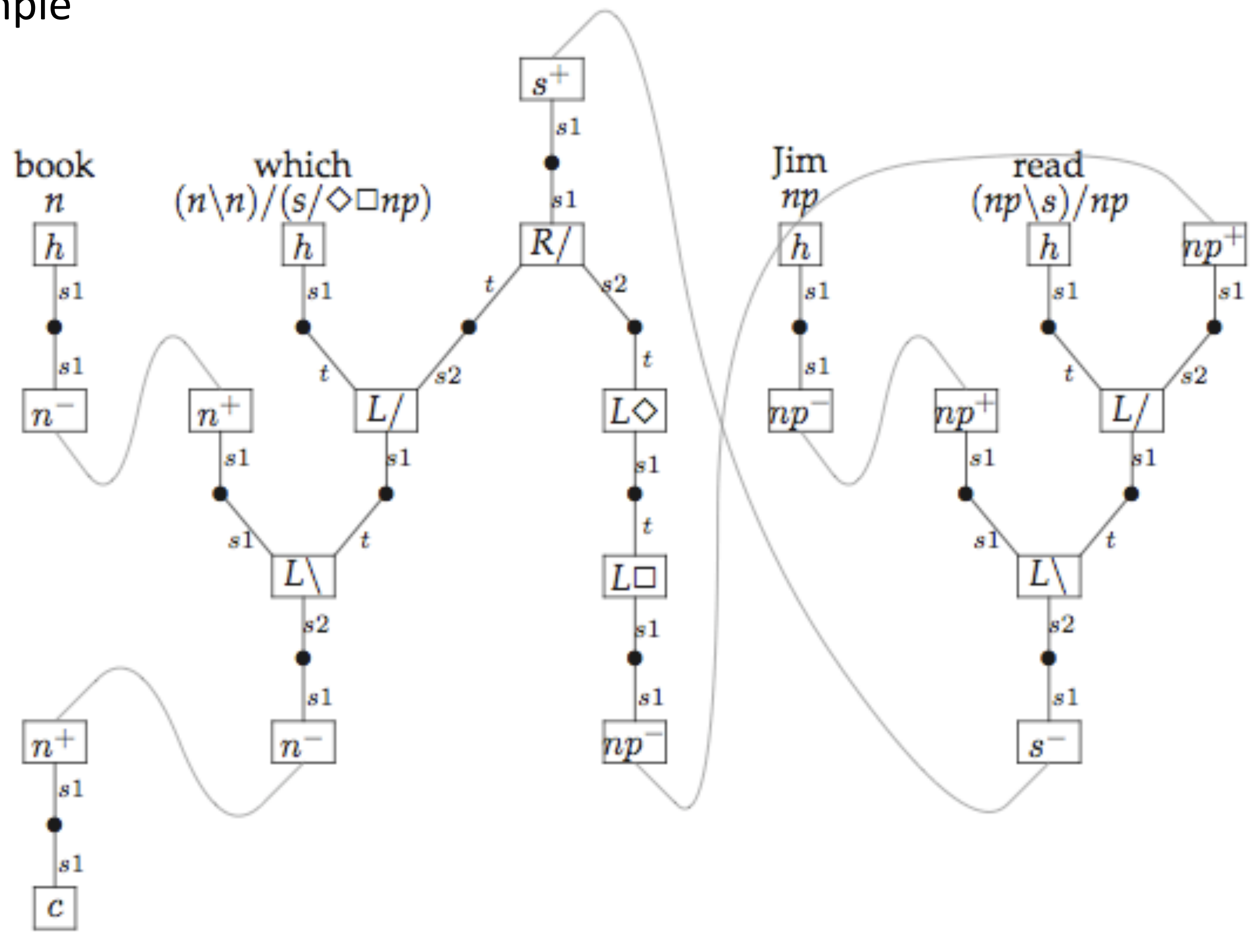
Proof nets as HRG

Links for proof structure



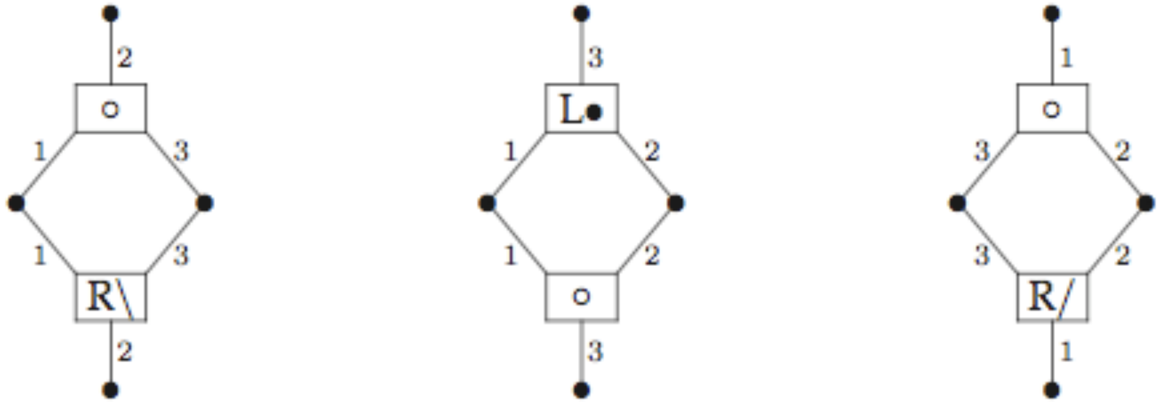
Proof nets as HRG

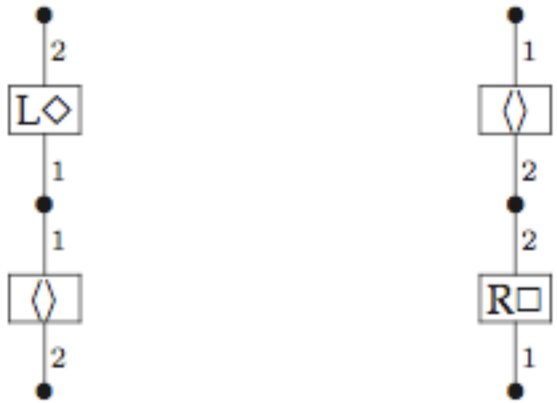
Example



Proof nets as HRG

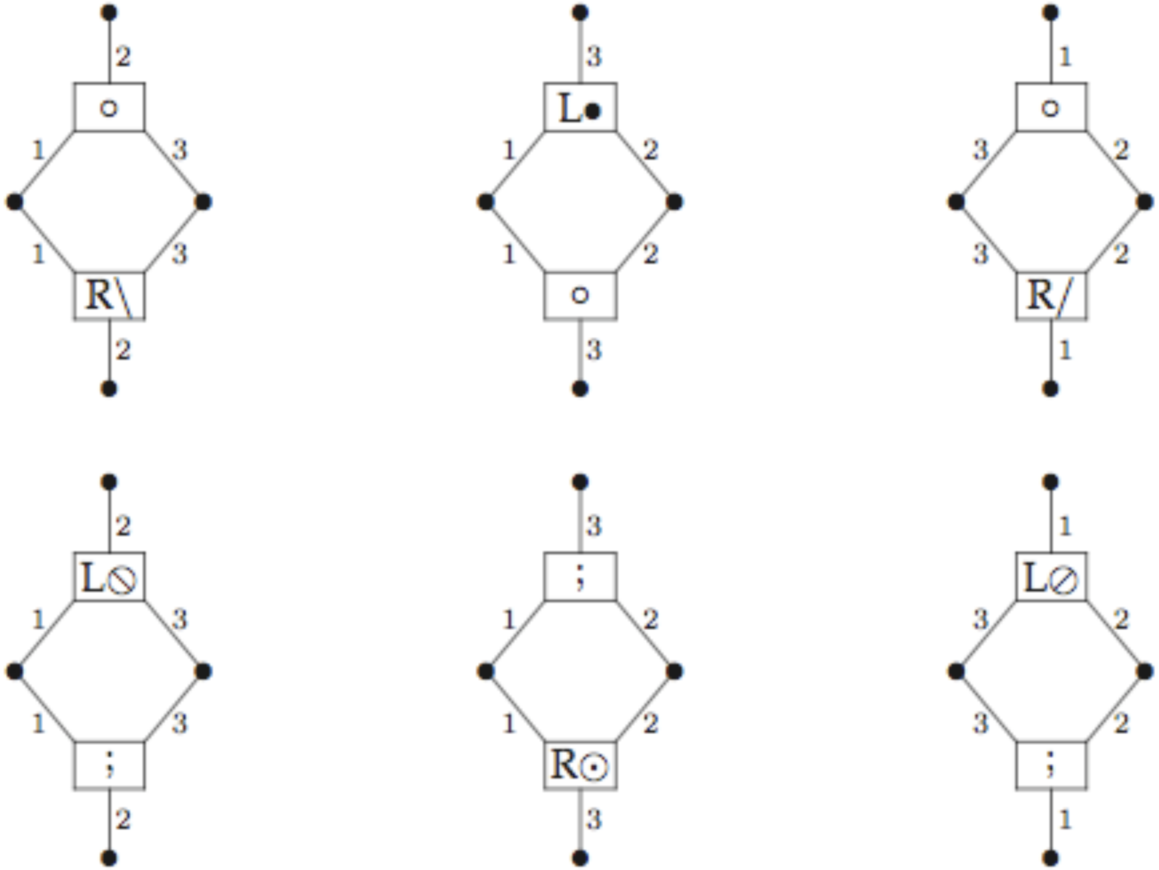
Contractions





Same for other structural rules

Proof nets as HRG



Proof nets as HRG

If G is a Lambek Grammar, then there exists a strongly equivalent HR grammar G' .

Conclusion?

NL$\Diamond_R$ and LG are mildly context-sensitive formalisms and therefore benefit from the pleasant properties this entails, such as polynomial parsability.

Conclusion?

Logic	**NL**	**L**	???	**NL**$\Diamond_{\mathcal{R}}$
Complexity	P	NP	P	PSPACE
Languages	CFL	CFL	MCSL	CSL

Melissen(2011) shows that LG recognises more than LTAG

Thanks